www.mientayvn.com

Khi đọc qua tài liệu này, nếu phát hiện sai sót hoặc nội dung kém chất lượng xin hãy thông báo để chúng tôi sửa chữa hoặc thay thế bằng một tài liệu cùng chủ đề của tác giả khác. Tài liệu này bao gồm nhiều tài liệu nhỏ có cùng chủ đề bên trong nó. Phần nội dung bạn cần có thể nằm ở giữa hoặc ở cuối tài liệu này, hãy sử dụng chức năng Search để tìm chúng.

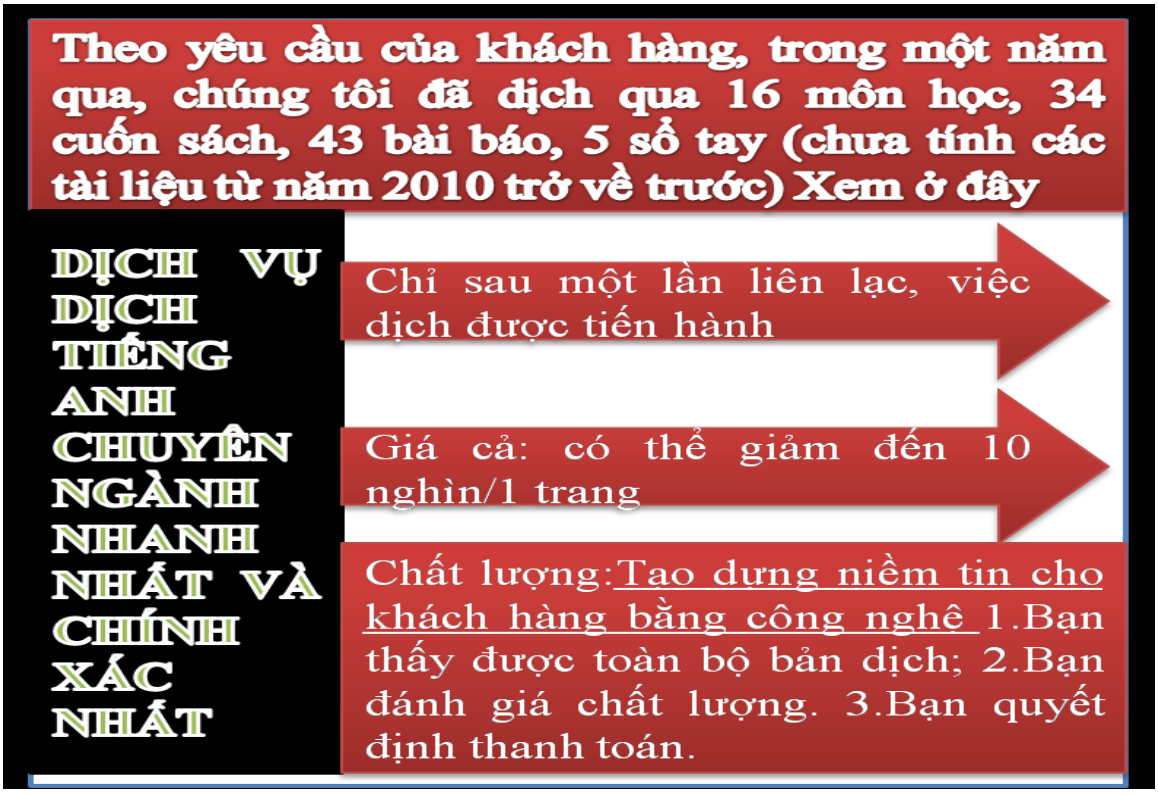
Bạn có thể tham khảo nguồn tài liệu được dịch từ tiếng Anh tại đây:

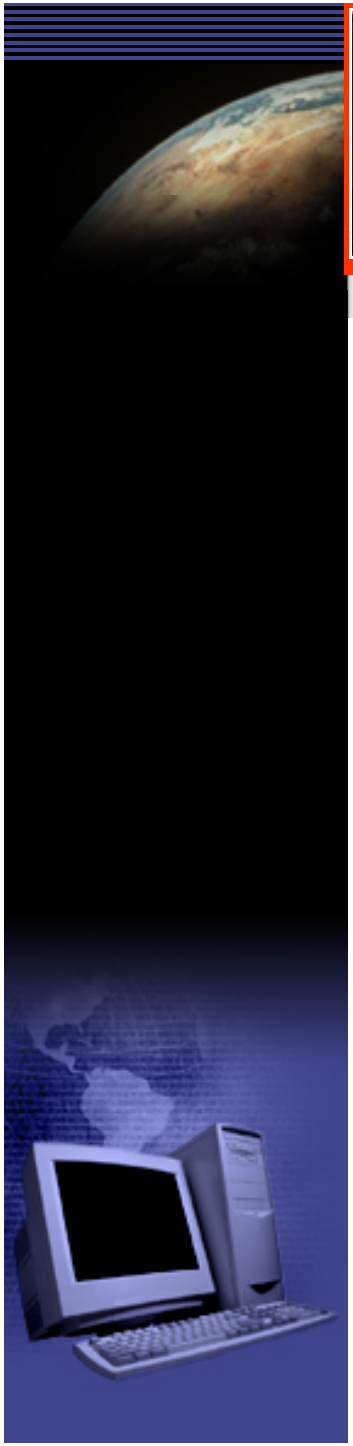
http://mientayvn.com/Tai_lieu_da_dich.html

Thông tin liên hệ:

Yahoo mail: thanhlam1910_2006@yahoo.com

Gmail: frbwrthes@gmail.com



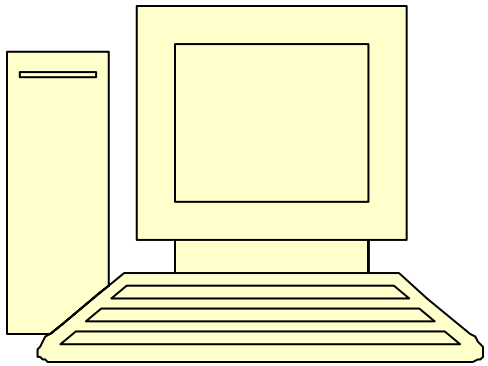


ĐẠI HỌC ĐÀ NẴNG

TRƯỜNG ĐẠI HỌC BÁCH KHOA

KHOA CÔNG NGHỆ THÔNG TIN

NGUYÊN LÝ HỆ ĐIỀU HÀNH



Giới thiệu

Nội dung giáo trình

CHƯƠNG 1. MỞ ĐẦU

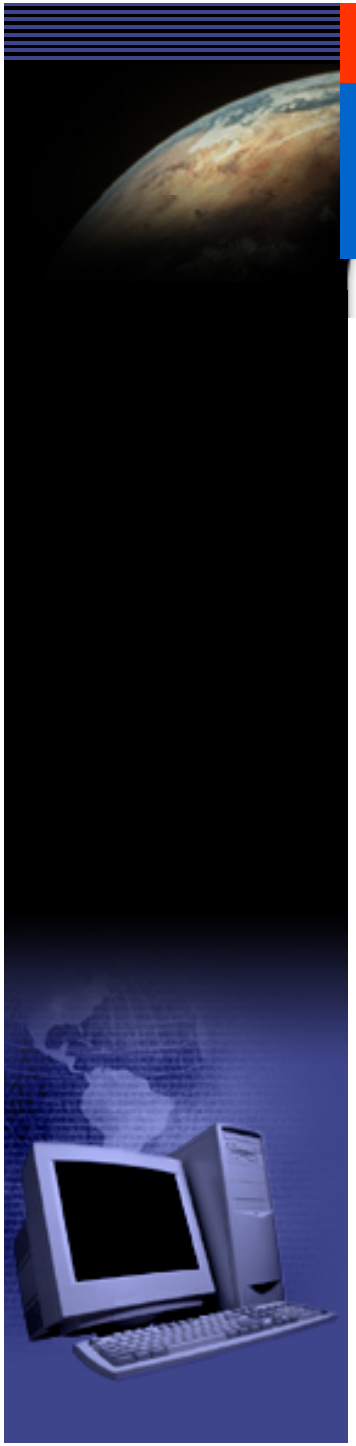
CHƯƠNG 2. TIẾN TRÌNH

CHƯƠNG 3. VÀO/RA

CHƯƠNG 4. QUẢN LÝ BỘ NHỚ

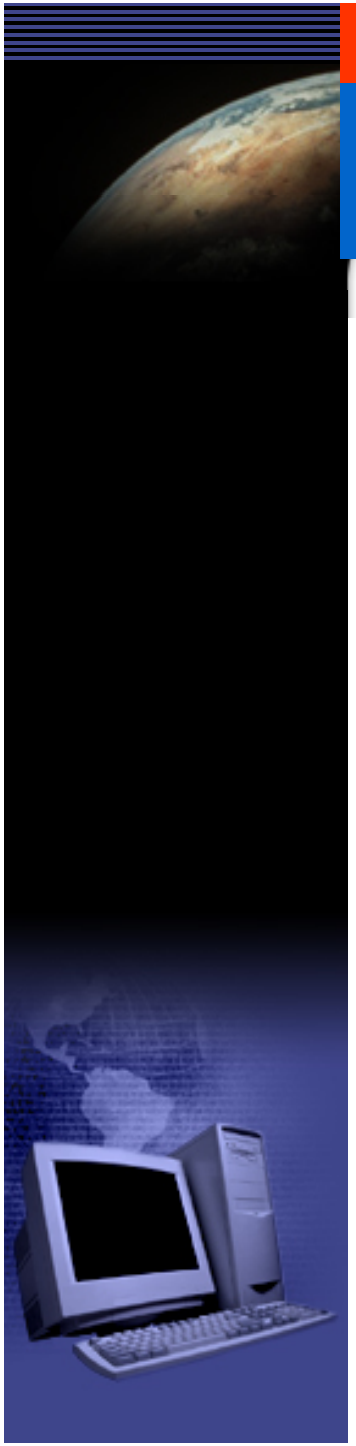
CHƯƠNG 5. HỆ THỐNG FILE

CHƯƠNG 1. MỞ ĐẦU



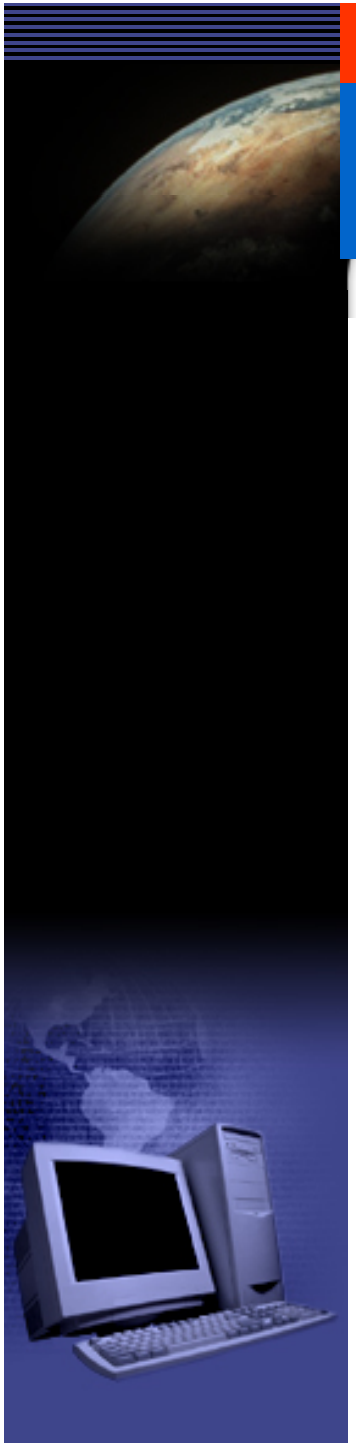
Các vấn đề

1. Khái niệm hệ điều hành
2. Chức năng của hệ điều hành
3. Vị trí của hệ điều hành
4. Các thành phần của hệ điều hành
5. Cấu trúc của hệ điều hành



Khái niệm hệ điều hành

Hệ điều hành (HĐH) là phần gắn bó trực tiếp với phần cứng và là môi trường cho các chương trình ứng dụng chạy trên nó.



Chức năng của hệ điều hành

- Quản lý và phân phối tài nguyên 1 cách hợp lý
- Giả lập một máy tính mở rộng và tạo giao diên tiện lợi với người sử dụng

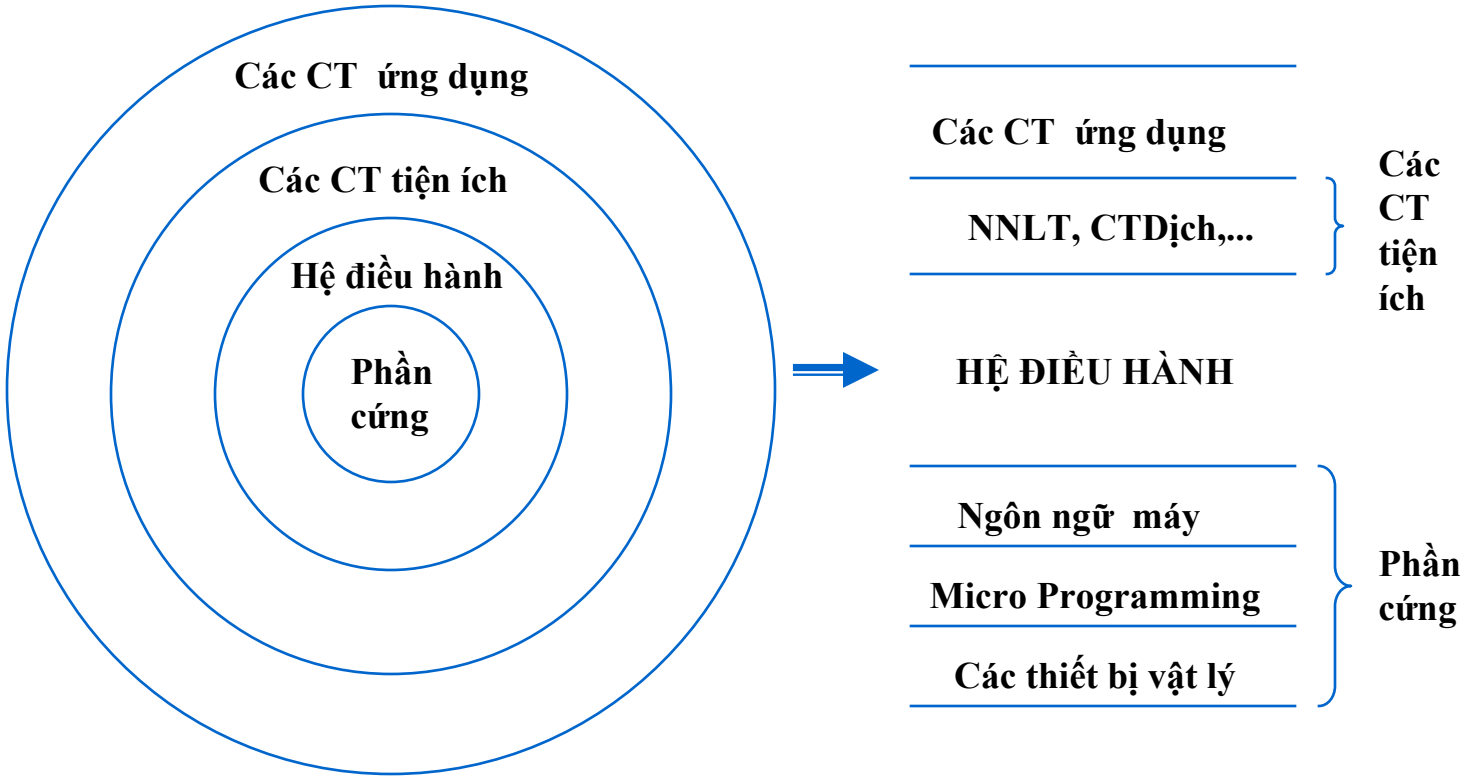
Tài nguyên

- **Tài nguyên phần cứng**
  - **Bộ xử lý**
  - **Bộ nhớ**
  - **Các thiết bị nhập xuất**
- **Tài nguyên phần mềm**

Các file, chương trình dùng chung,...

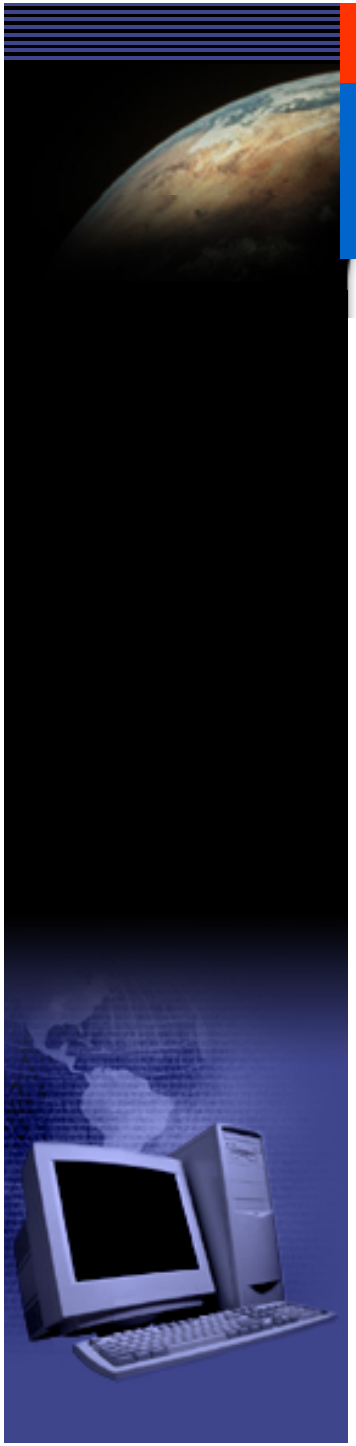
CHƯƠNG 1. MỞ ĐẦU

Vị trí của hệ điều hành



Các thành phần của hệ điều hành

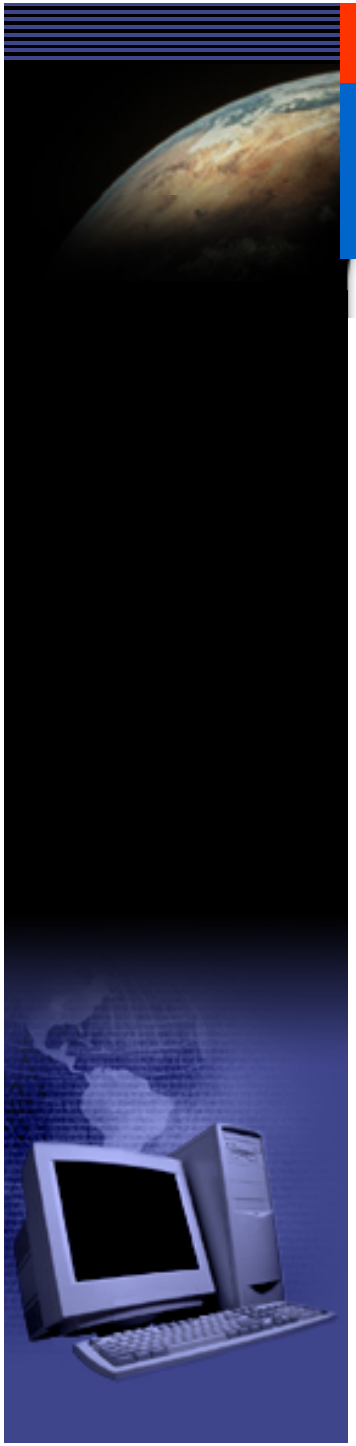
- Quản lý tiến trình
- Quản lý bộ nhớ
- Quản lý nhập xuất
- Quản lý tập tin
- Hệ thống bảo vệ
- Hệ thông dịch lệnh (Shell)
- Quản lý mạng



Các thành phần của hệ điều hành

Quản lý tiến trình

- Tạo lập, huỷ bỏ một tiến trình
- Tạm dừng, tái kích hoạt một tiến trình
- Cung cấp các cơ chế trao đổi thông tin giữa các tiến trình
- Cung cấp cơ chế đồng bộ hoá các tiến trình



Các thành phần của hệ điều hành

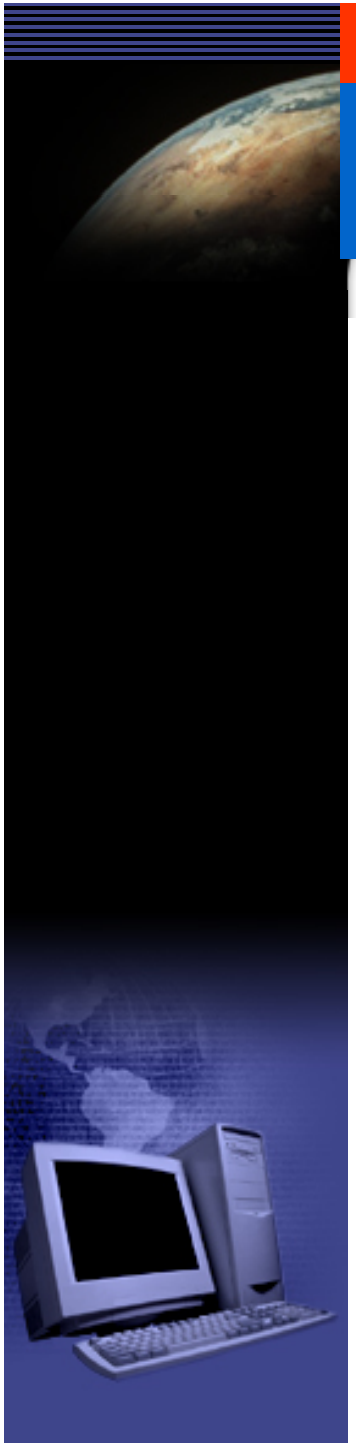
Quản lý bộ nhớ

- Cấp phát và thu hồi vùng nhớ cho tiến trình khi cần thiết
- Ghi nhận tình trạng bộ nhớ chính: vùng đã cấp phát, vùng còn có thể sử dụng...
- Quyết định tiến trình nào được nạp vào bộ nhớ chính khi có một vùng nhớ trống.

Các thành phần của hệ điều hành

Quản lý nhập xuất

- Gửi các lệnh điều khiển đến các thiết bị
- Tiếp nhận các ngắt
- Xử lý lỗi



Các thành phần của hệ điều hành

Quản lý tập tin

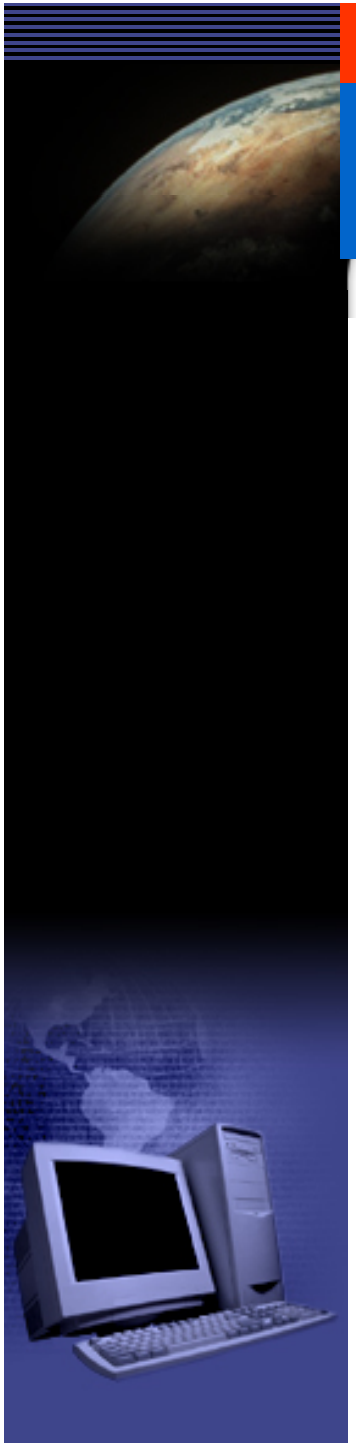
- Tạo lập, huỷ bỏ một tập tin.
- Tạo lập và huỷ bỏ một thư mục.
- Cung cấp các thao tác xử lý tập tin và thư mục.
- Tạo lập quan hệ tương ứng giữa tập tin và bộ nhớ phụ chứa nó.

Các thành phần của hệ điều hành

Hệ thống bảo vệ

- **Xây dựng cơ chế bảo vệ thích hợp.**

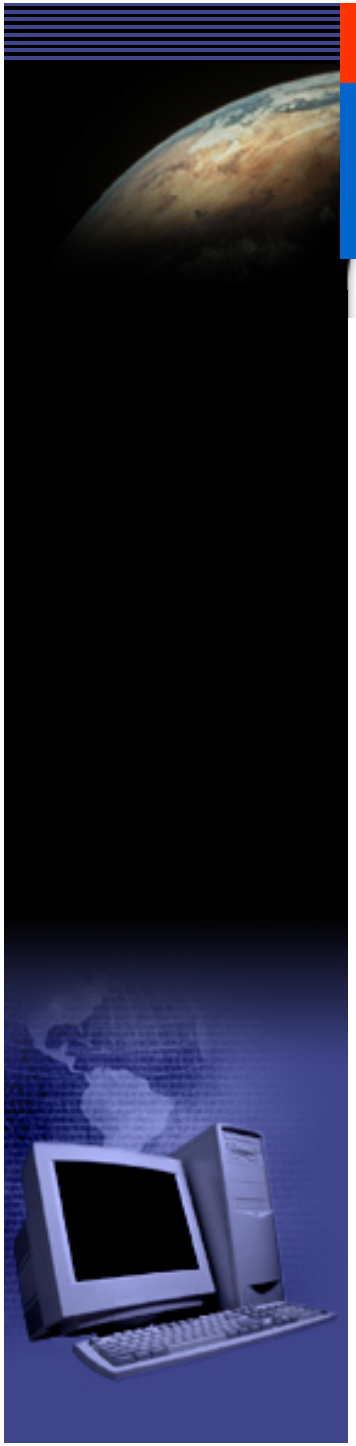
Trong trường hợp nhiều người cùng sử dụng đồng thời các tiến trình.



Các thành phần của hệ điều hành

Hệ thông dịch lệnh (Shell)

- Đóng vai trò giao diện giữa NSD và HĐH
- Các lệnh được chuyển đến HĐH dưới dạng chỉ thị điều khiển.
- Shell nhận lệnh và thông dịch lệnh để HĐH có xử lý tương ứng



Các thành phần của hệ điều hành

Quản lý mạng

- Một hệ thống phân bố nhiều bộ xử lý với các bộ nhớ độc lập.
- Các tiến trình trong hệ thống có thể kết nối với nhau qua mạng truyền thông.
- Việc truy xuất đến tài nguyên mạng thông qua các trình điều khiển giao tiếp mạng.

Cấu trúc của hệ điều hành

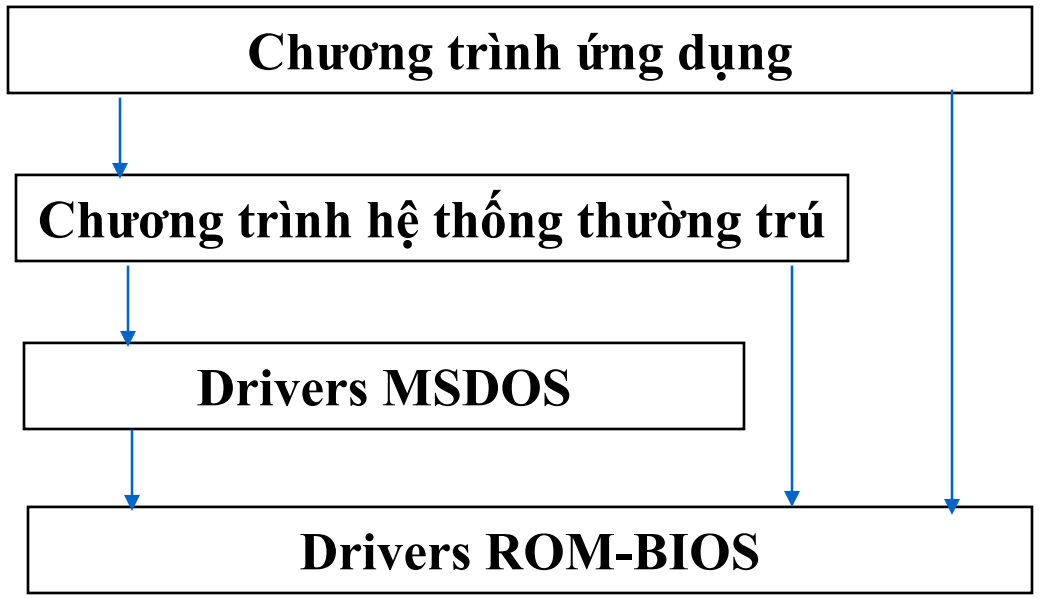
- Hệ thống nguyên khối (Monolithic System)
- Hệ thống phân lớp (Layer System)
- Máy ảo (Virtual Machine)
- Mô hình Client-Server (Client-Server Model)

Hệ thống nguyên khối

- Cấu trúc HĐH được xem là ko cấu trúc
- HĐH được xây dựng dựa trên tập hợp các thủ tục riêng lẻ.
- Mỗi thủ tục có thể gọi lẫn nhau khi cần
- CT ứng dụng có thể truy xuất đến thủ tục cấp thấp, phần cứng. Do vậy HĐH khó kiểm soát và bảo vệ hệ thống
- Khi xây dựng thủ tục phải định nghĩa rõ tham số đầu vào, tham số đầu ra
- HĐH thiếu tính chủ động trong việc quản lý môi trường. (tính chất tĩnh, chỉ được kích hoạt khi cần)

Hệ thống nguyên khối

- Ví dụ: Cấu trúc MSDOS



Hệ thống nguyên khối

- **Hoạt động của bộ xử lý được chia làm 2 chế độ**

- Chế độ Kernel: chạy thực hiện các thủ tục của HĐH (lời gọi hệ thống)

- Chế độ User: chạy thực hiện các CT của NSD

Hệ thống nguyên khối

- Khi HĐH khởi động tất cả các lời gọi hệ thống đều được nạp và định vị vào RAM.
- HĐH tạo bảng Dispatch gồm các Slot, mỗi Slot là một con trỏ trỏ đến Đ/C đầu của một CT phục vụ

Hệ thống phân lớp

- Hệ thống được xây dựng bởi nhiều lớp.
- Mỗi lớp được xây dựng dựa trên các lớp bên trong
- Lớp trong cùng (lớp 0): phần cứng
- Lớp ngoài cùng (lớp N): giao diện với NSD
- Mỗi lớp là một đối tượng trừu tượng (dữ liệu+thao tác xử lý dữ liệu).
- Mỗi lớp có thể gọi các thủ tục của các lớp bên trong

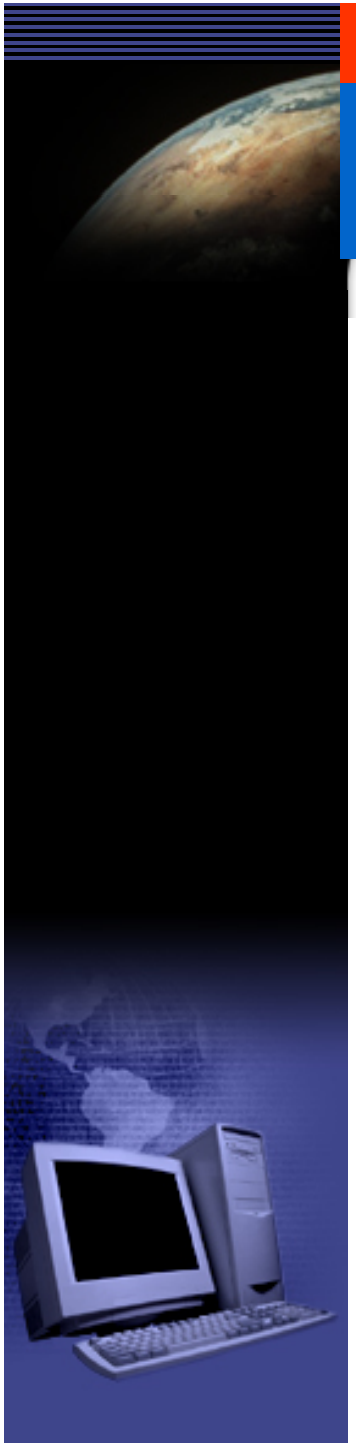
Hệ thống phân lớp

❖ **Ví dụ: hệ thống THE (Technische Hogeschool Eidhoven) thiết kế năm 1968**

Lớp 5: Chương trình ứng dụng
Lớp 4: Quản lý bộ đệm cho thiết bị nhập/xuất
Lớp 3: Trình điều khiển thao tác console
Lớp 2: Quản lý bộ nhớ
Lớp 1: Điều phối CPU
Lớp 0: Phần cứng

CHƯƠNG 1. MỞ ĐẦU

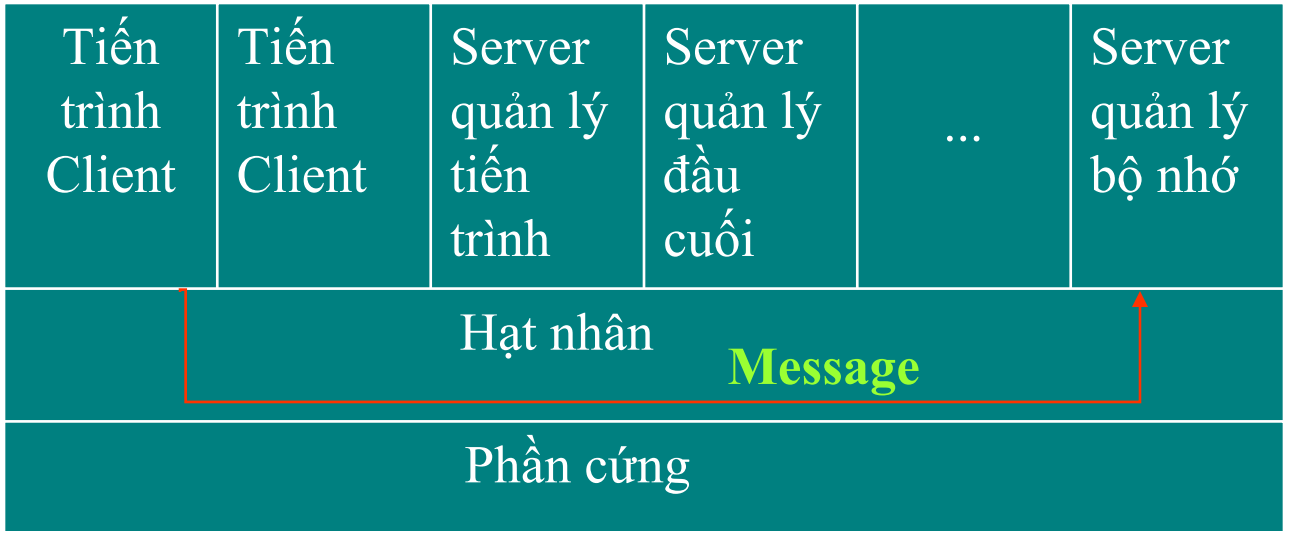
Máy ảo



Mô hình Client-Server

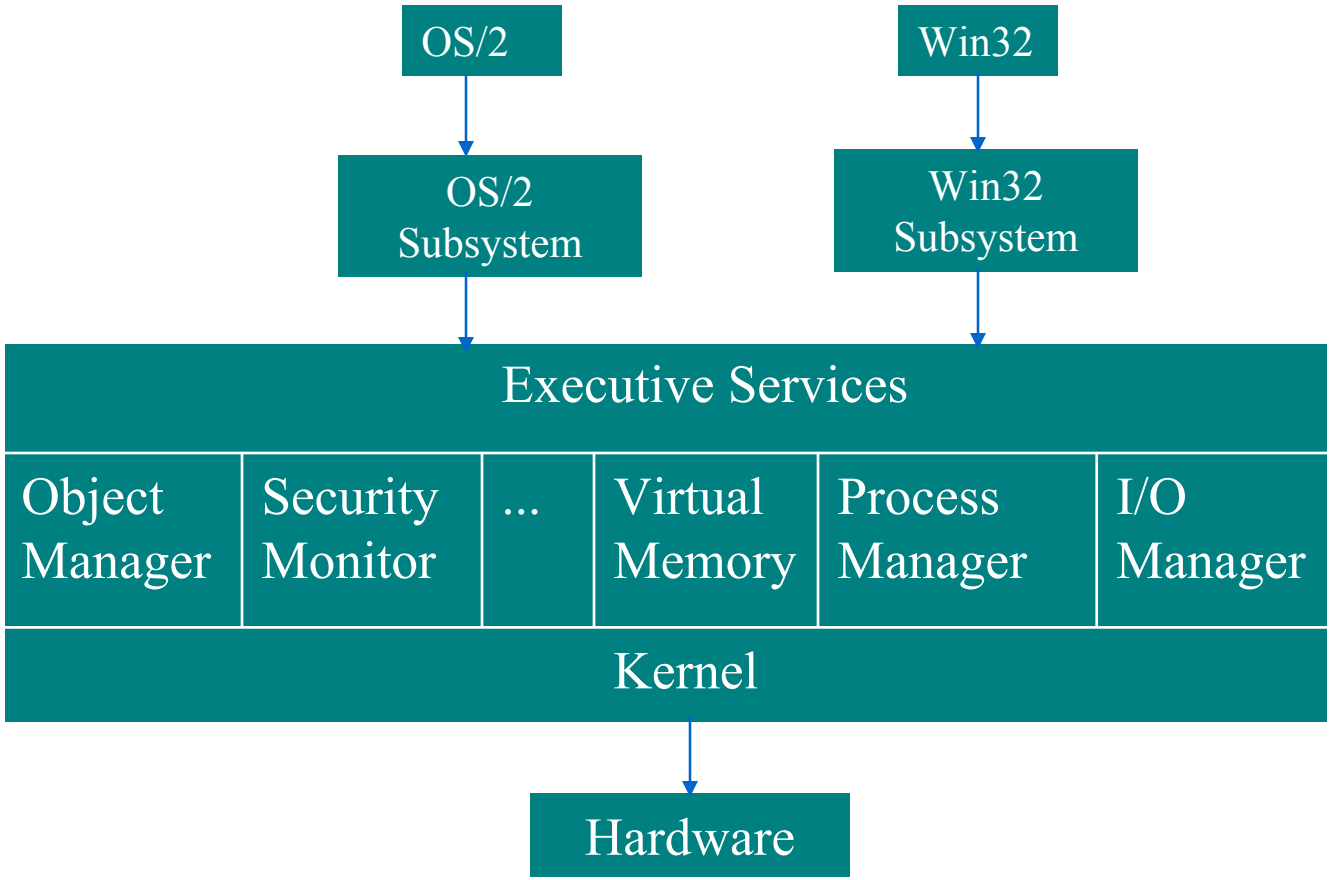
- **HĐH bao gồm nhiều tiến trình đóng vai trò Server với các chức năng chuyên biệt.**
- **Phần hạt nhân HĐH đóng vai trò giao tiếp giữa tiến trình Client và tiến trình Server.**
- **Chỉ có phần hạt nhân cực nhỏ phụ thuộc vào phần cứng.**

Mô hình Client-Server



Mô hình Client-Server

Ví dụ: Cấu trúc Windows NT



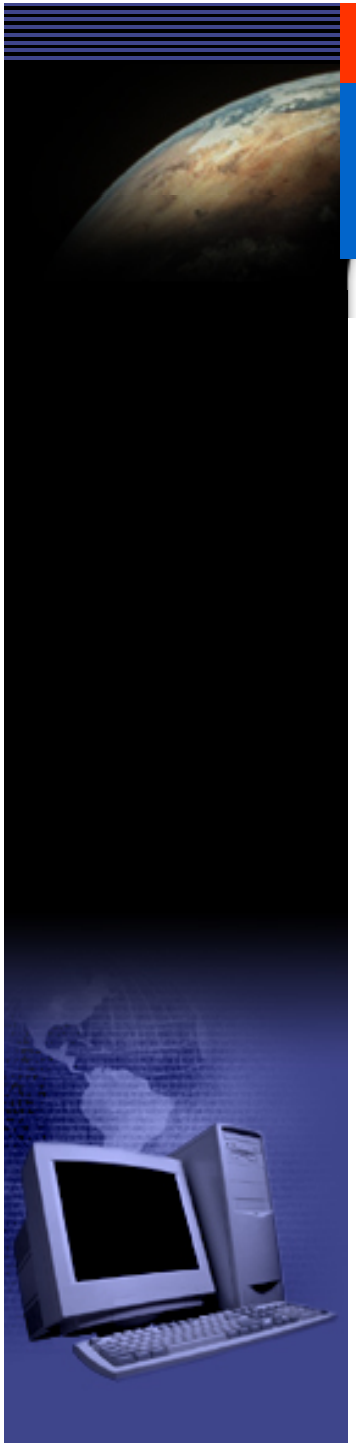
CHƯƠNG 2. TIẾN TRÌNH

Các vấn đề

1. Các khái niệm
2. Mô hình trạng thái
3. Thao tác trên tiến trình
4. Điều phối tiến trình
5. Đồng bộ hoá tiến trình

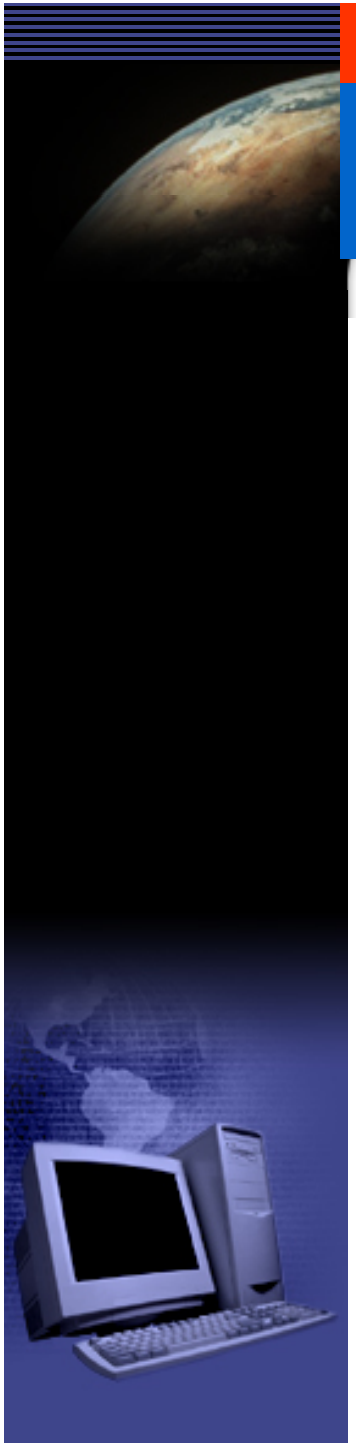
Các khái niệm

- Tiến trình (Process): chương trình đang thực hiện
- Mỗi tiến trình có một tập tài nguyên và môi trường riêng (con trỏ lệnh, Stack, thanh ghi, không gian địa chỉ)
- Các tiến trình hoàn toàn độc lập với nhau, có thể liên lạc thông qua các cơ chế truyền tin giữa các tiến trình.



Các khái niệm

- **Tiến trình hệ thống: được sinh ra khi thực hiện các lời gọi hệ thống**
- **Tiến trình của người sử dụng: được sinh ra khi thực thi CT của NSD**



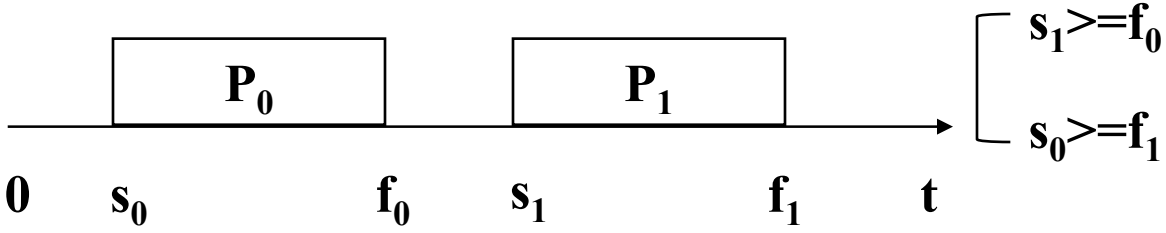
Các khái niệm

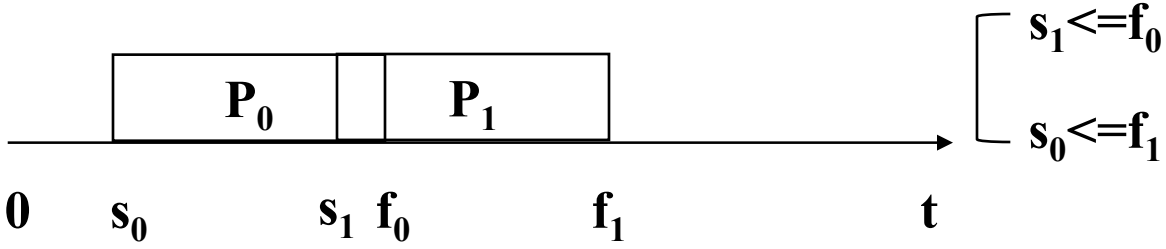
- **Có 2 loại tiến trình:**

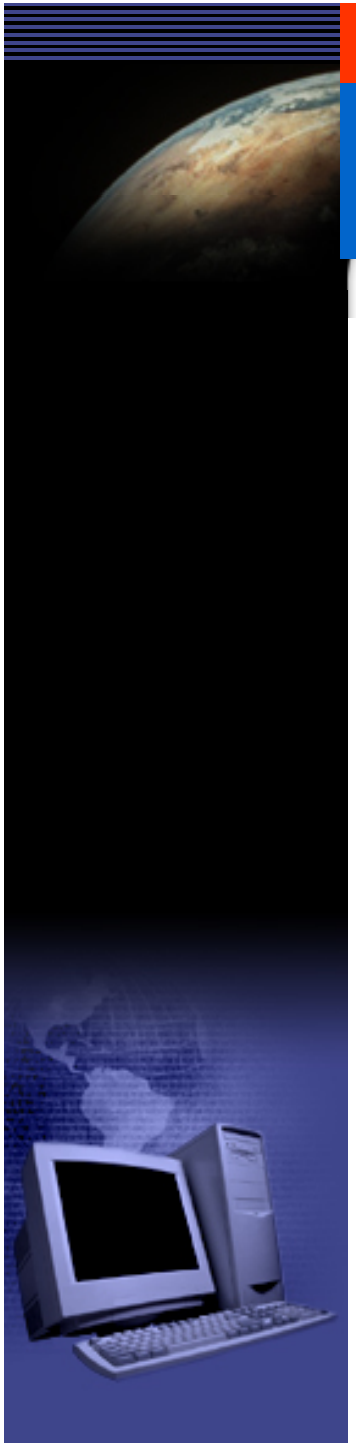
- Tiến trình kế tiếp: thời điểm bắt đầu của tiến trình này nằm sau thời điểm kết thúc của tiến trình kia

- Tiến trình song song: thời điểm bắt đầu của tiến trình này nằm trước thời điểm kết thúc của tiến trình kia

Các khái niệm

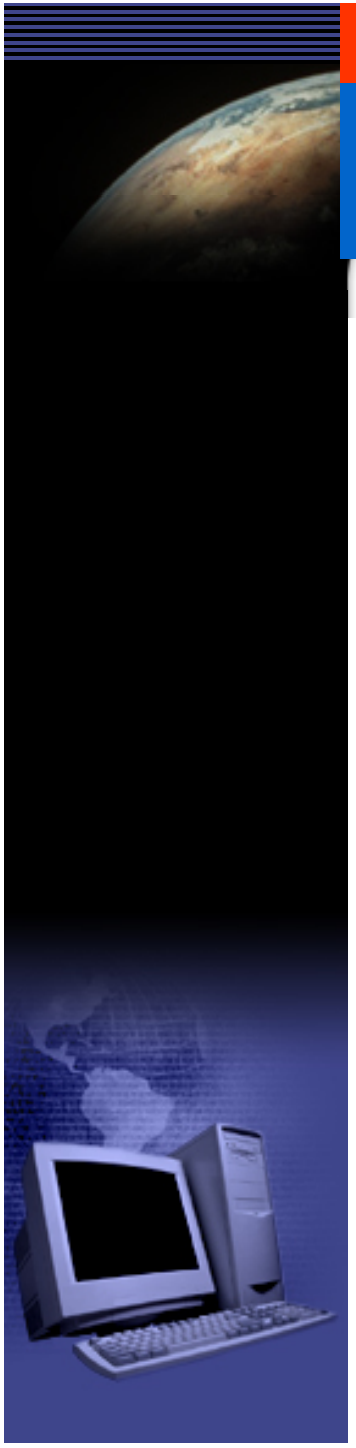






Các khái niệm

- **HĐH quản lý tiến trình thông qua khối quản lý tiến trình (Process Control Block:PCB)**
- **PCB: vùng nhớ lưu trữ các thông tin mô tả cho tiến trình như:**
  - **Định danh của tiến trình: phân biệt giữa các tiến trình.**
  - **Trạng thái tiến trình: hoạt động hiện hành của tiến trình.**



Các khái niệm

- **Ngữ cảnh của tiến trình:**

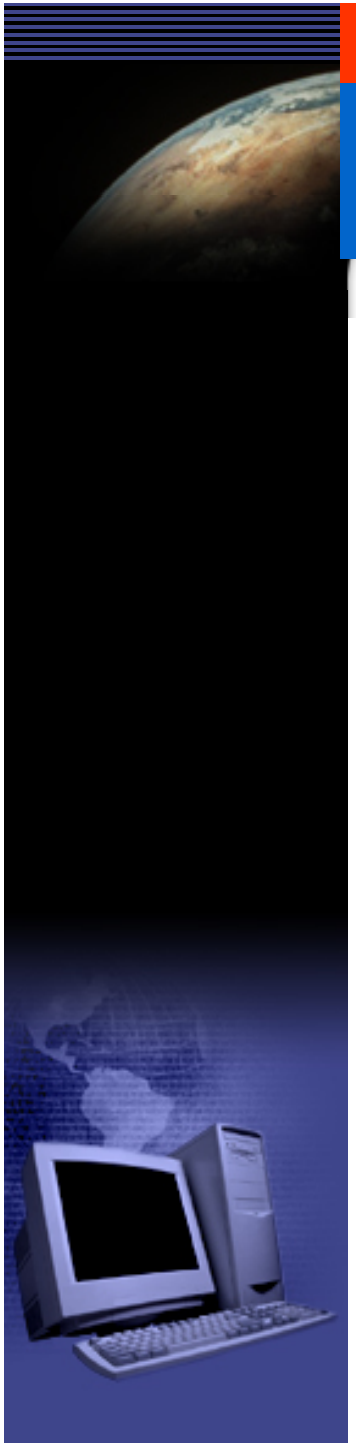
- Trạng thái CPU: nội dung các thanh ghi (IP). Lưu trữ nội dung thanh ghi khi xảy ra ngắt.

- Bộ xử lý: xác định số hiệu CPU mà tiến trình đang sử dụng (máy có cấu hình nhiều CPU).

- Bộ nhớ chính: danh sách các vùng nhớ được cấp cho tiến trình.

- Tài nguyên sử dụng: danh sách các tài nguyên hệ thống mà tiến trình đang sử dụng.

- Tài nguyên tạo lập: danh sách các tài nguyên được tiến trình tạo lập.



Các khái niệm

- **Thông tin giao tiếp:**

- Tiến trình cha: tiến trình tạo lập tiến trình này

- Tiến trình con: các tiến trình do tiến trình này tạo ra

- Độ ưu tiên: thông tin giúp bộ điều phối lựa chọn tiến trình được cấp CPU

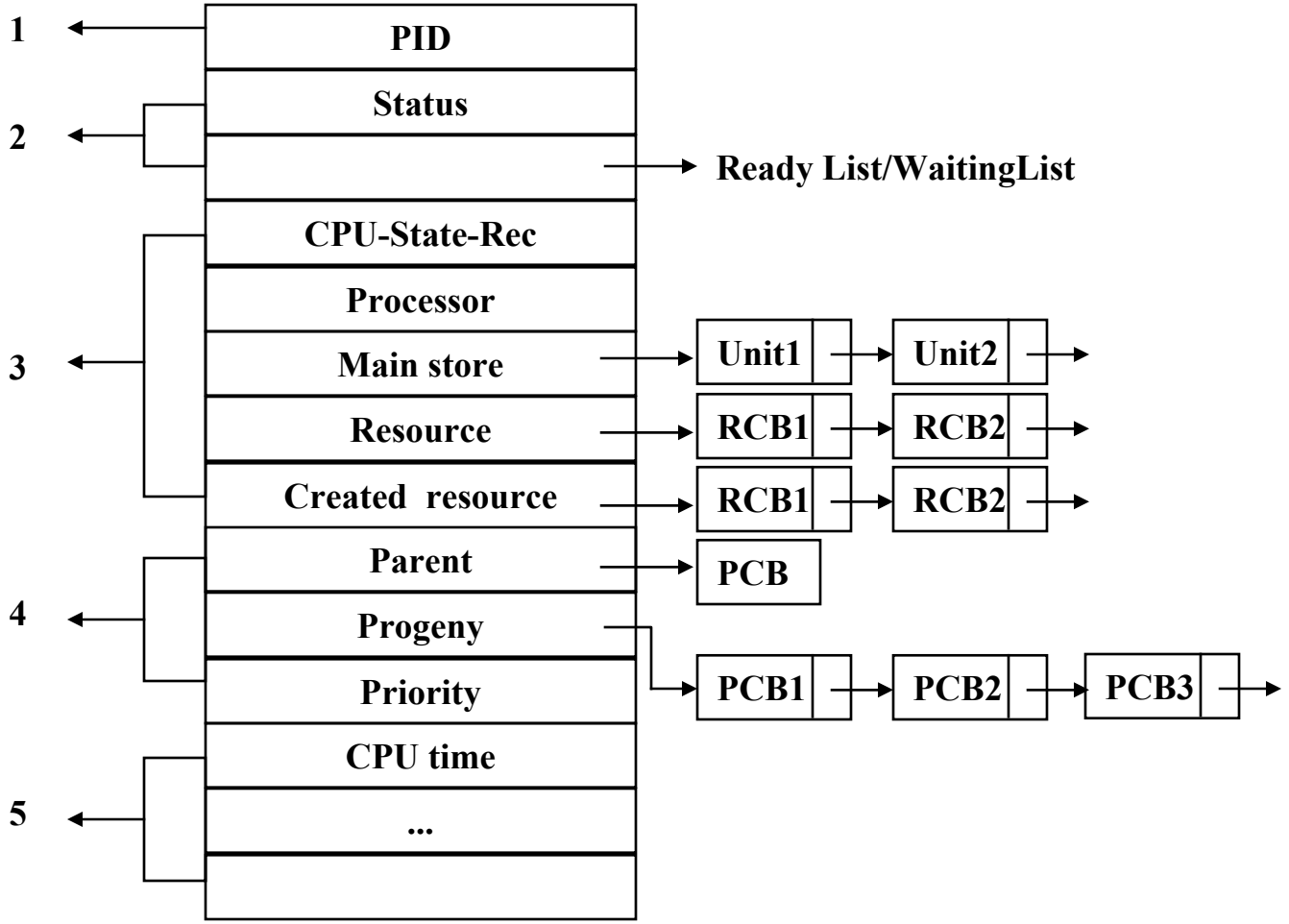
- **Thông tin thống kê về hoạt động của tiến trình:**

- Thời gian sử dụng CPU

- Thời gian chờ

Các khái niệm

Khối quản lý tiến trình



Các khái niệm

- Tiểu trình (Threads): một đơn vị xử lý cơ bản của hệ thống.
- Một tiểu trình cũng có thể tạo lập các tiến trình con
- Một tiến trình có thể sở hữu nhiều tiểu trình
- Các tiểu trình trong cùng một tiến trình có thể:
  - Chia sẻ một không gian địa chỉ.
  - Truy xuất đến các Stack của nhau

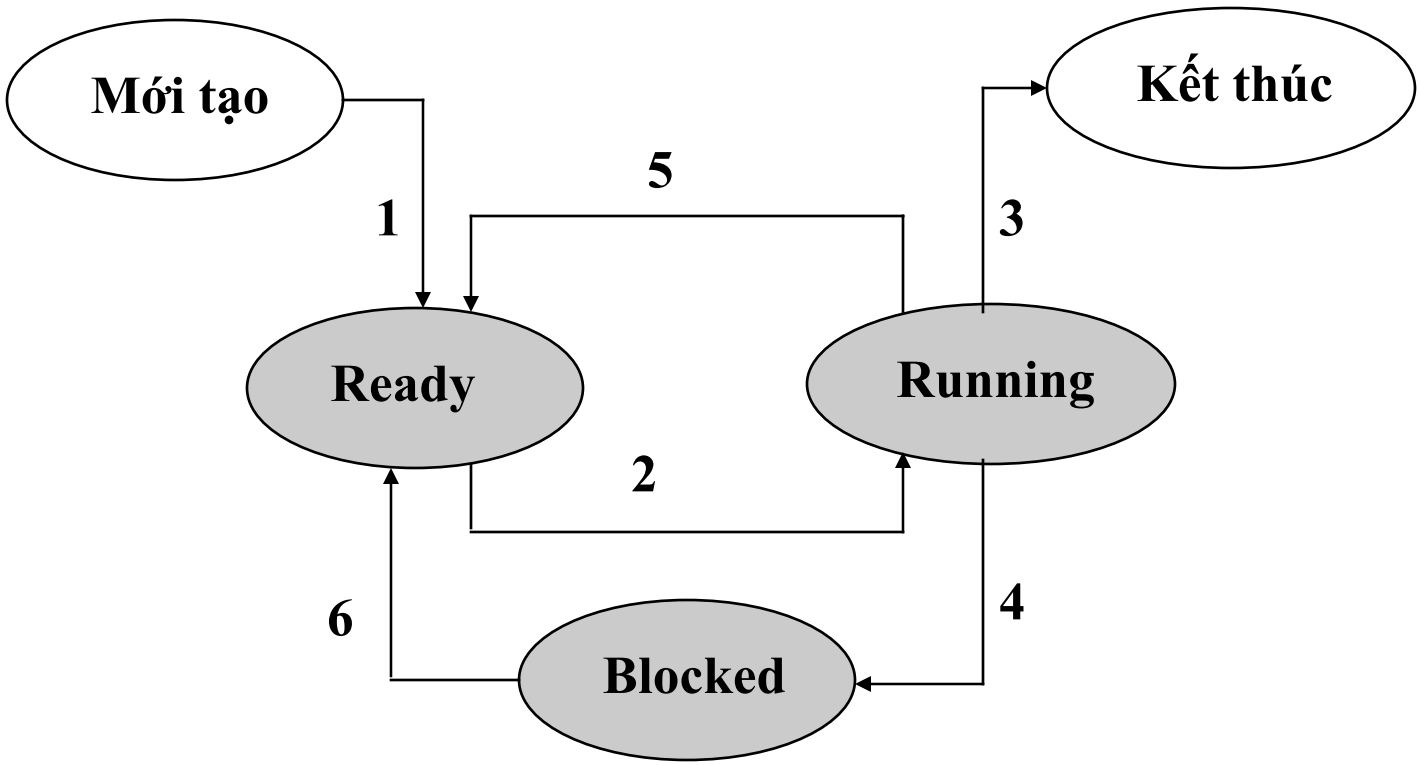
Mô hình trạng thái

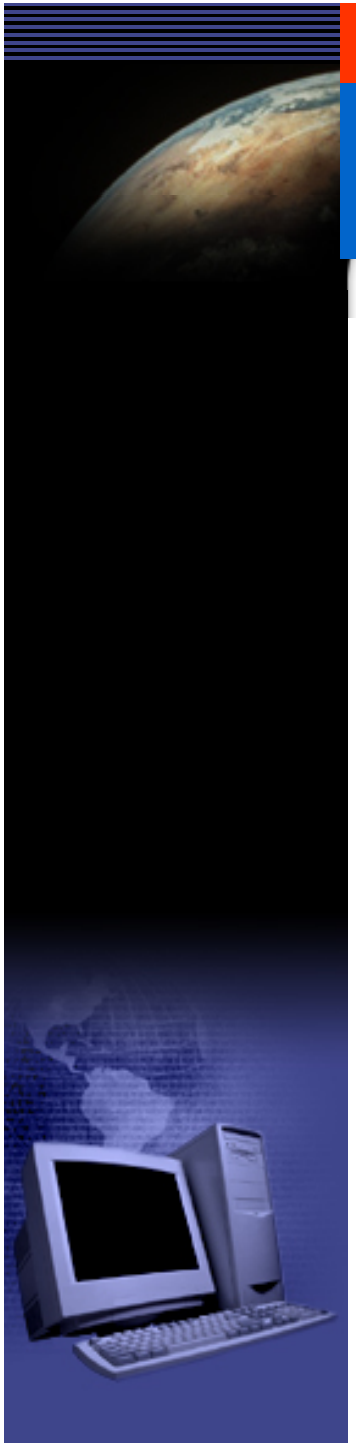
❖ Các trạng thái của tiến trình

- **Mới tạo: tiến trình đang được tạo lập.**
- **Running: tiến trình đang được xử lý.**
- **Ready: tiến trình đang sẵn sàng, chờ cấp CPU để xử lý**
- **Blocked: tiến trình bị chặn, không thể tiếp tục.**
- **Kết thúc: tiến trình hoàn tất xử lý.**

Mô hình trạng thái

❖ Sơ đồ chuyển trạng thái của tiến trình





Mô hình trạng thái

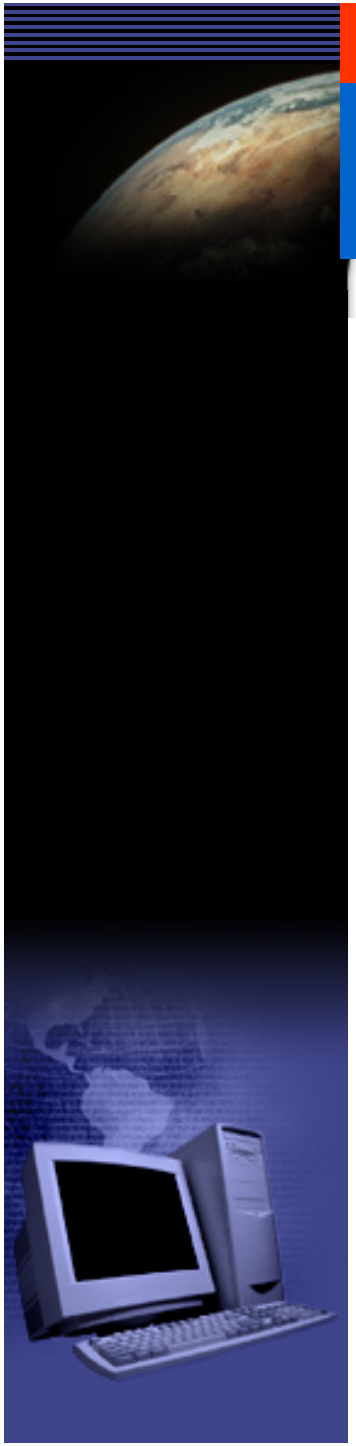
❖ **Sơ đồ chuyển trạng thái của tiến trình**

(1) Tiến trình mới tạo lập được đưa vào hệ thống.

(2) Bộ điều phối cấp phát cho tiến trình một khoảng thời gian sử dụng CPU.

(3) Tiến trình kết thúc, bộ điều phối thu lại CPU.

(4) Tiến trình yêu cầu tài nguyên nhưng chưa được đáp ứng, hoặc phải chờ một sự kiện hay thao tác nhập/xuất.

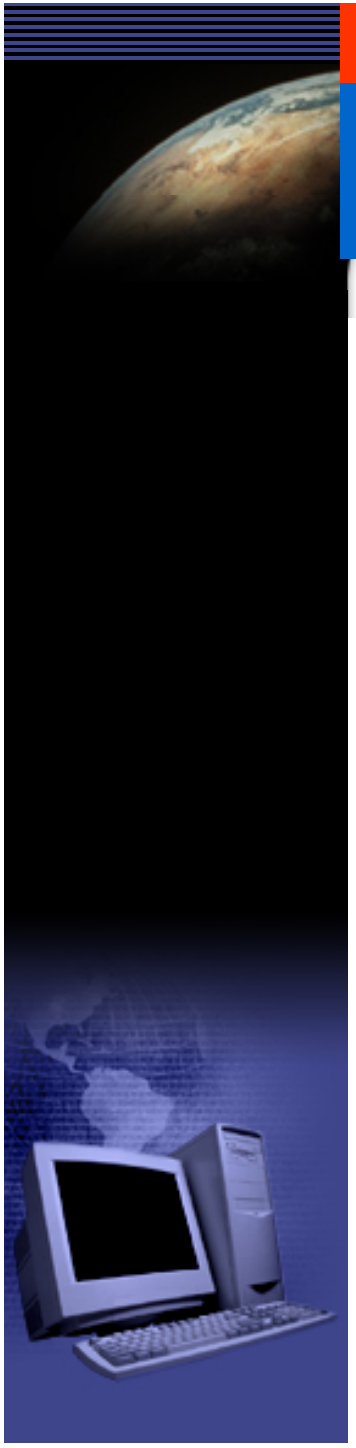


Mô hình trạng thái

❖ Sơ đồ chuyển trạng thái của tiến trình

(5) Bộ điều phối chọn một tiến trình khác để xử lý.

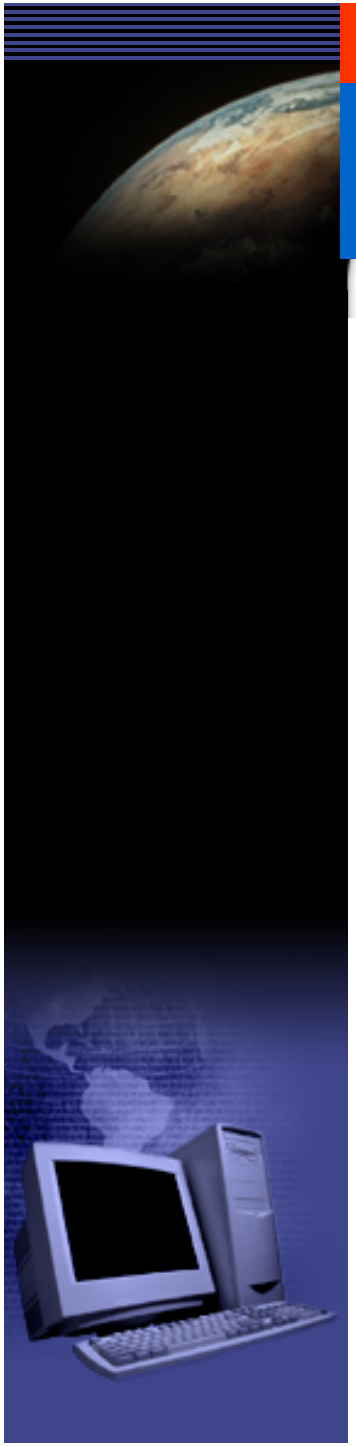
(6) Tài nguyên mà tiến trình yêu cầu đã sẵn sàng để cấp phát, hay sự kiện, thao tác nhập/xuất tiến trình đang đợi hoàn tất.



Thao tác trên tiến trình

❖ Tạo lập tiến trình

- Một tiến trình có thể tạo lập nhiều tiến trình mới
- Tiến trình tạo ra tiến trình mới gọi là tiến trình cha
- Tiến trình mới được tạo ra gọi là tiến trình con
- Tiến trình con đến lượt lại tạo ra một loạt các tiến trình con của nó,... Quá trình này tiếp tục sẽ tạo thành cây tiến trình



Thao tác trên tiến trình

❖ Tạo lập tiến trình

- Khi tạo lập tiến trình,HĐH cần thực hiện:

✓ Định danh cho tiến trình (PID)

✓ Đưa tiến trình vào danh sách quản lý của hệ thống

✓ Xác định độ ưu tiên của tiến trình

✓ Tạo khối quản lý tiến trình (PCB)

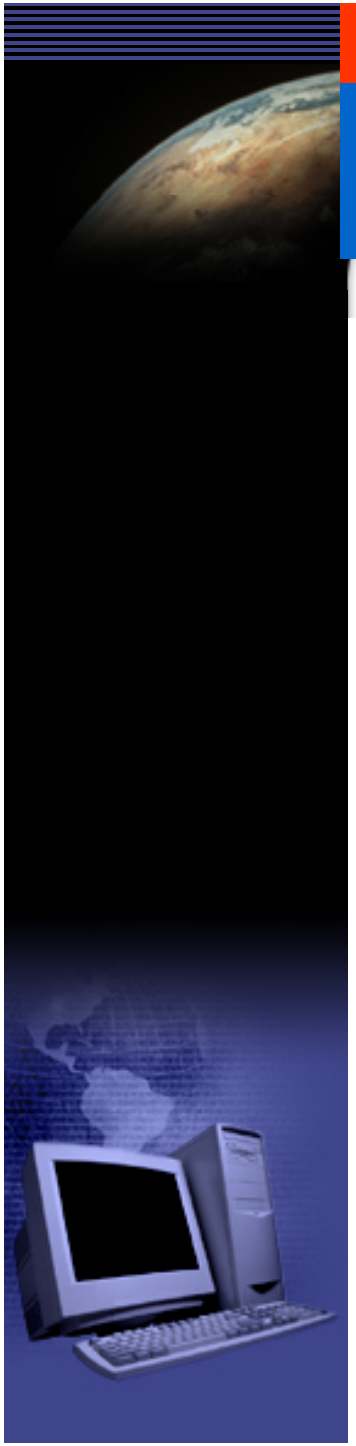
✓ Cấp phát tài nguyên ban đầu cho tiến trình

Thao tác trên tiến trình

❖ Kết thúc tiến trình

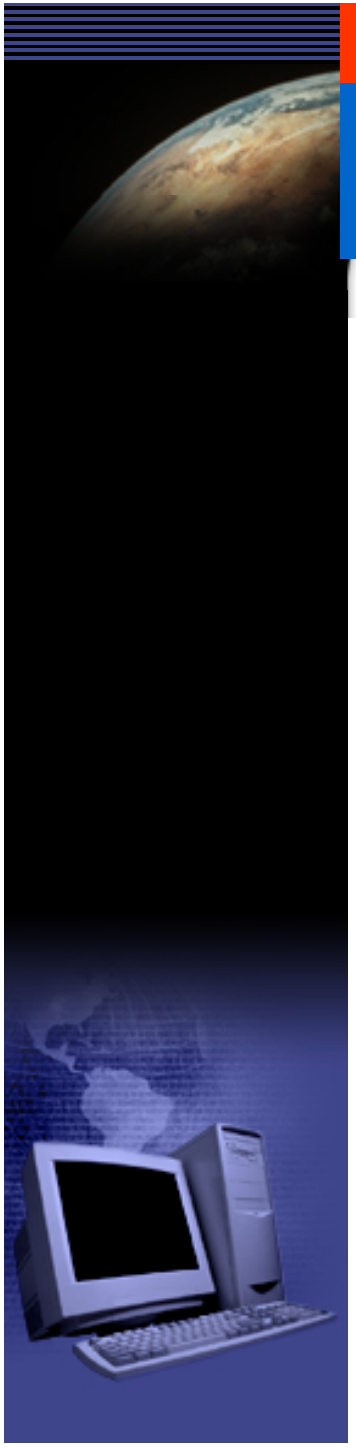
Khi tiến trình kết thúc, HĐH thực hiện:

- **Thu hồi các tài nguyên của hệ thống đã cấp phát cho tiến trình**
- **Huỷ tiến trình khỏi tất cả các danh sách quản lý của hệ thống**
- **Huỷ bỏ PCB của tiến trình**



Điều phối tiến trình

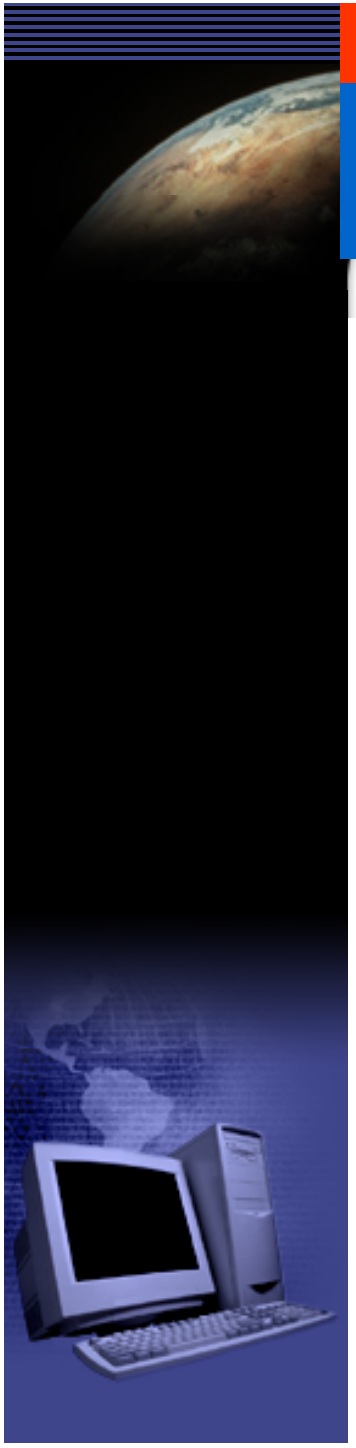
- Mục tiêu điều phối
- Tiêu chuẩn điều phối
- Điều phối không độc quyền, điều phối độc quyền
- Đồng hồ ngắt giờ
- Độ ưu tiên của tiến trình
- Tổ chức điều phối
- Các chiến lược điều phối



Điều phối tiến trình

❖ Mục tiêu điều phối

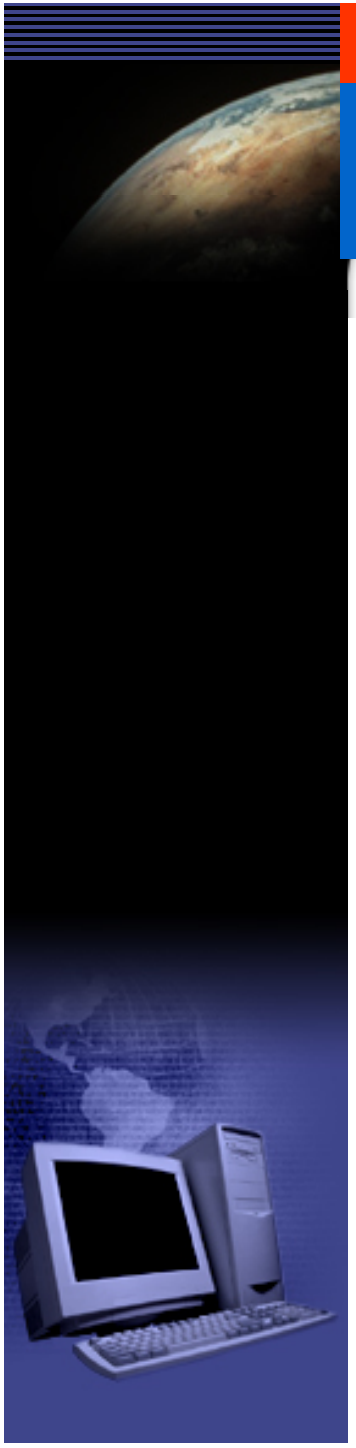
- Sự công bằng giữa các tiến trình
- Tính hiệu quả (tận dụng 100% thời gian sử dụng CPU)
- Cực tiểu hoá thời gian lưu lại trong hệ thống
- Thời gian đáp ứng hợp lý (cực tiểu hoá thời gian hồi đáp cho các tương tác của NSD)
- Thông lượng tối đa (cực đại hoá số công việc được xử lý trong một thời gian cố định)



Điều phối tiến trình

❖ **Tiêu chuẩn điều phối (đặc điểm của tiến trình)**

- Tính hướng xuất/nhập của tiến trình
- Tính hướng xử lý của tiến trình
- Tiến trình tương tác hay xử lý theo lô
- Độ ưu tiên của tiến trình
- Thời gian đã sử dụng CPU của tiến trình
- Thời gian còn lại tiến trình cần để hoàn tất



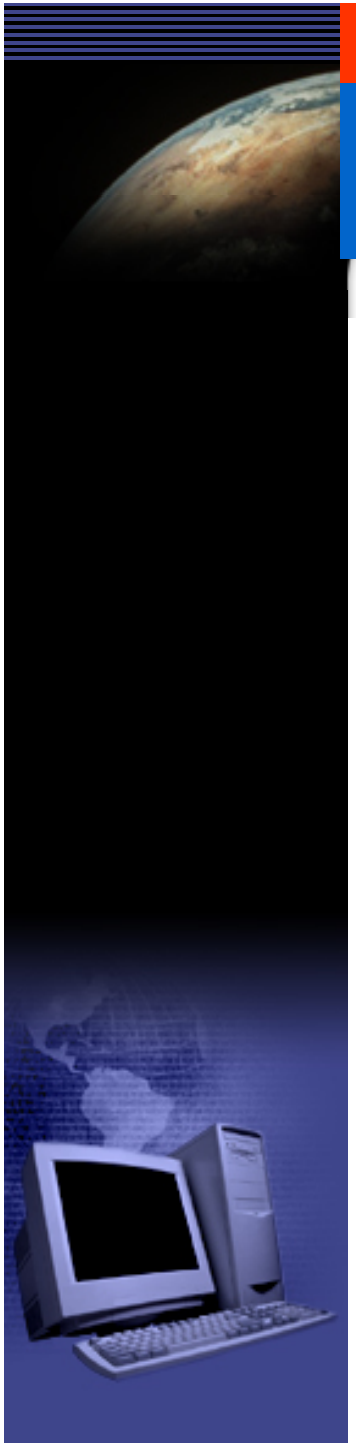
Điều phối tiến trình

❖ Điều phối độc quyền

- Tiến trình khi nhận được CPU thì có độc quyền sử dụng cho đến khi tiến trình hoàn tất hay tự nguyện giải phóng CPU
- Quyết định điều phối CPU xảy ra khi:

  + Tiến trình chuyển từ trạng thái Running sang Blocked

  + Tiến trình kết thúc
- Giải thuật đơn giản, dễ cài đặt nhưng ngăn cản các tiến trình còn lại trong hệ thống có cơ hội để xử lý



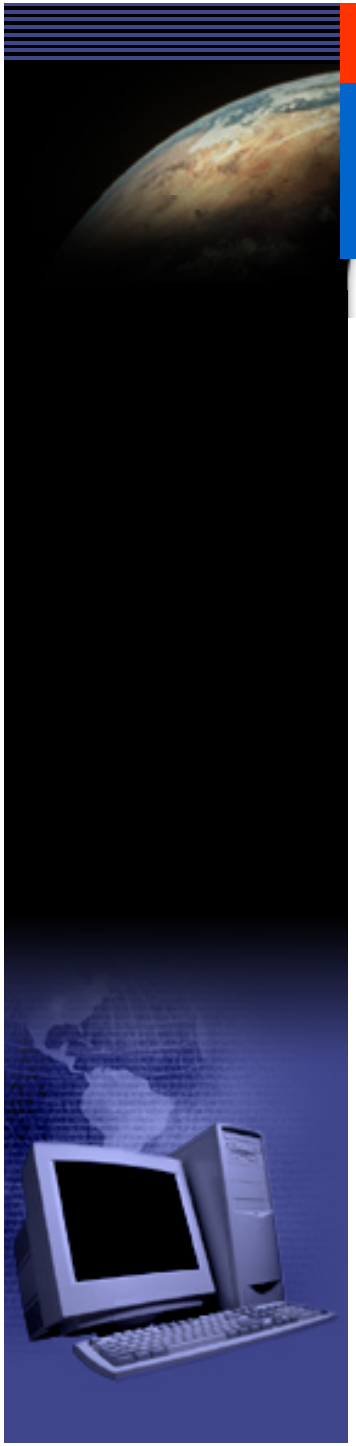
Điều phối tiến trình

❖ Điều phối không độc quyền

- Tiến trình có thể bị tạm dừng hoạt động bất cứ lúc nào mà không được báo trước, để tiến trình khác xử lý. (khi có một tiến trình khác có độ ưu tiên cao hơn về quyền dành sử dụng CPU)
- Quyết định điều phối CPU xảy ra khi:

  + Tiến trình chuyển từ trạng thái Running sang Blocked

  + Tiến trình chuyển từ trạng thái Running sang Ready



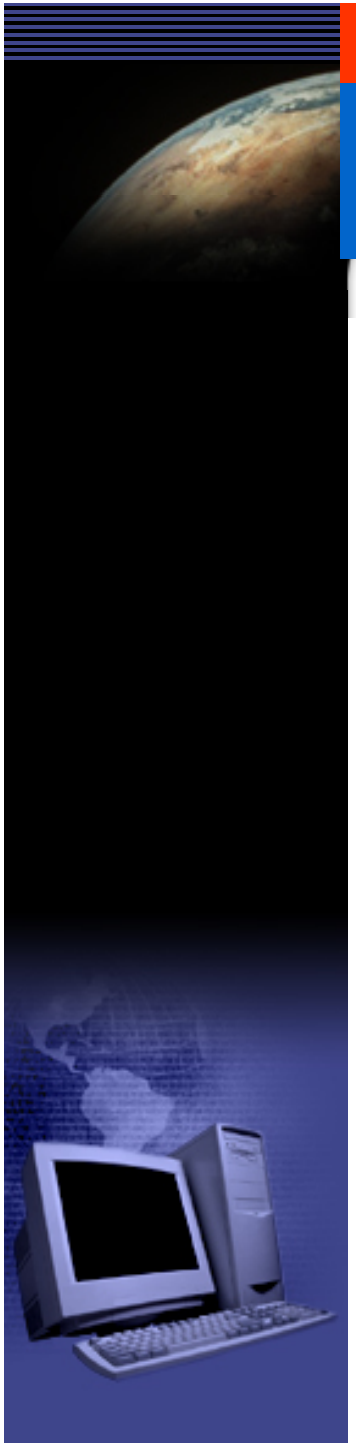
Điều phối tiến trình

❖ **Điều phối không độc quyền**

+ Tiến trình chuyển từ trạng thái blocked sang Ready

+ Tiến trình kết thúc

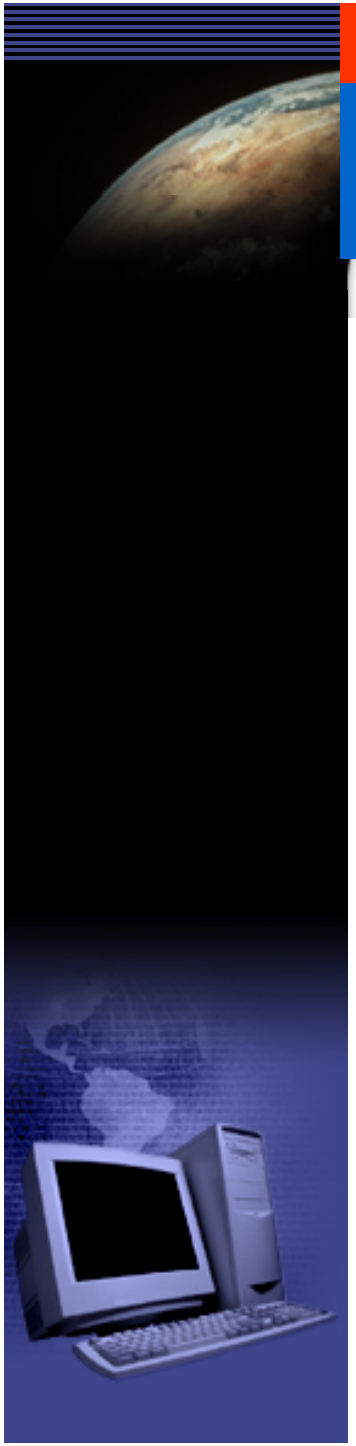
- **Ngăn cản được tình trạng các tiến trình độc chiếm CPU, nhưng việc tam dừng một tiến trình dẫn đến các mâu thuẫn trong truy xuất. Đòi hỏi phương pháp đồng bộ hoá thích hợp**



Điều phối tiến trình

❖ Đồng hồ ngắt thời gian

- Bộ đếm thời gian qui định một thông số thời gian t thích hợp ứng với một lượt cấp CPU cho một tiến trình
- Sau một khoảng thời gian t sẽ xảy ra một ngắt báo hiệu hết thời gian sử dụng CPU của tiến trình hiện hành. HĐH sẽ thu hồi CPU và bộ điều phối sẽ quyết định tiến trình nào sẽ được cấp phát.



Điều phối tiến trình

❖ Độ ưu tiên của tiến trình

- Độ ưu tiên của tiến trình: giá trị giúp phân định tầm quan trọng của các tiến trình
- Độ ưu tiên tĩnh:

  + Được gán sẵn cho tiến trình khi mới được ta ra

  + Không thay đổi
- Độ ưu tiên động: thay đổi theo thời gian và môi trường xử lý của tiến trình

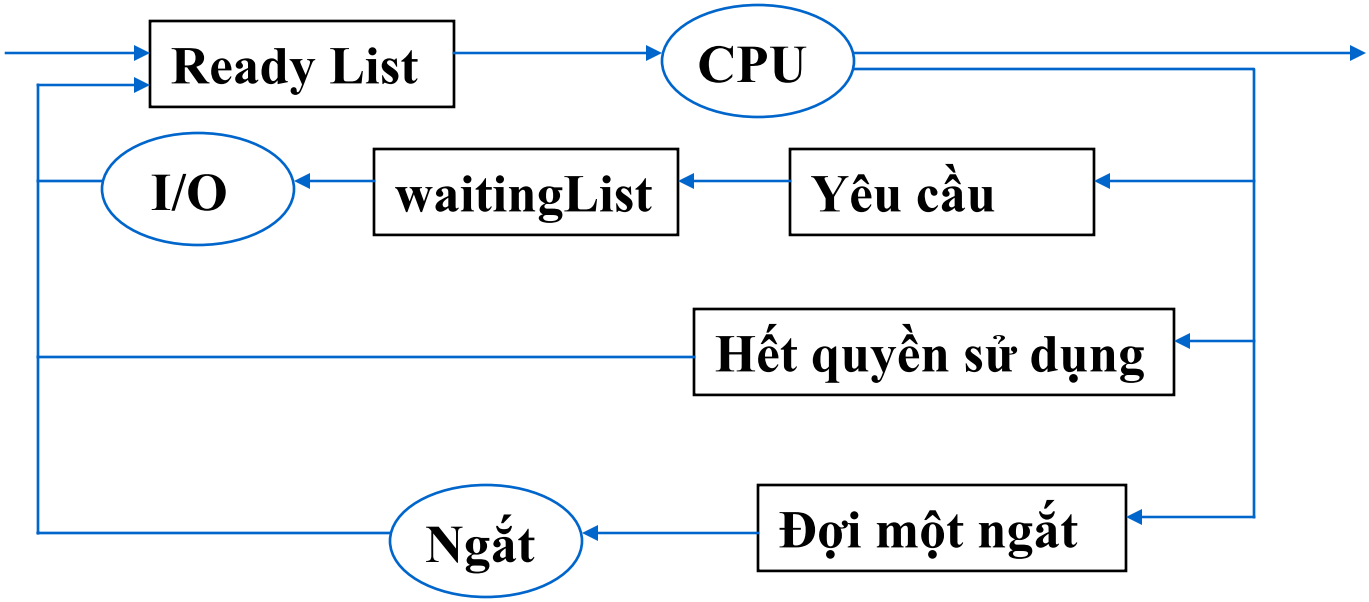
Điều phối tiến trình

❖ Tổ chức điều phối

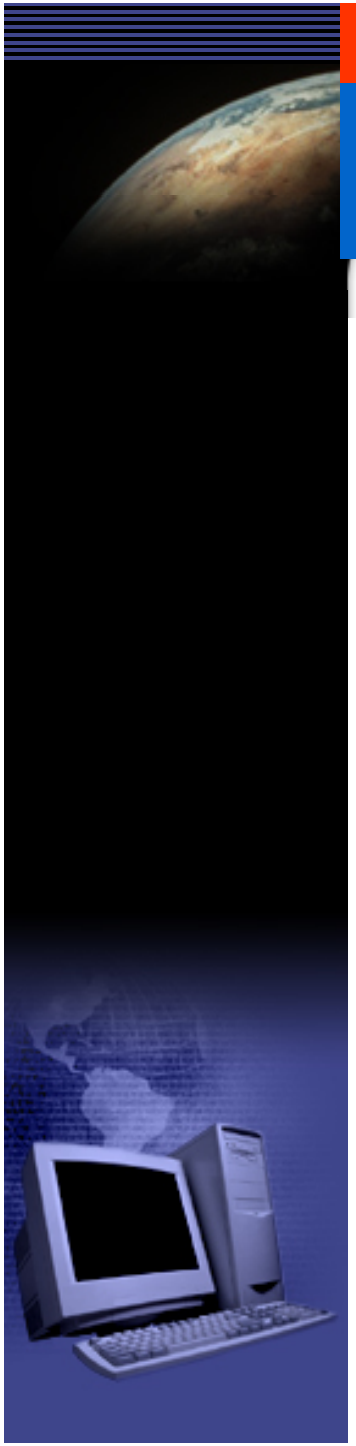
- Danh sách sẵn sàng (Ready List)
- Danh sách chờ đợi (Waiting List)
- Các danh sách chờ đợi riêng cho từng tài nguyên (thiết bị ngoại vi)

Điều phối tiến trình

❖ Tổ chức điều phối



Sơ đồ chuyển đổi giữa các danh sách điều phối



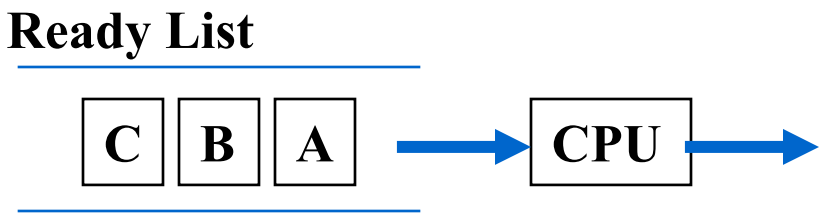
Điều phối tiến trình

❖ Chiến lược điều phối

- Thuật toán FIFO
- Thuật toán Round Robin (xoay vòng)
- Thuật toán SJF (Shortest-Job-First)
- Thuật toán sử dụng độ ưu tiên

Điều phối tiến trình

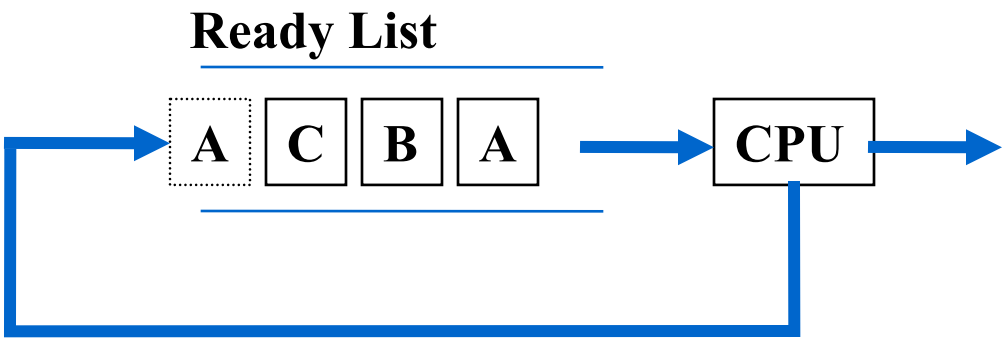
❖ Chiến lược điều phối



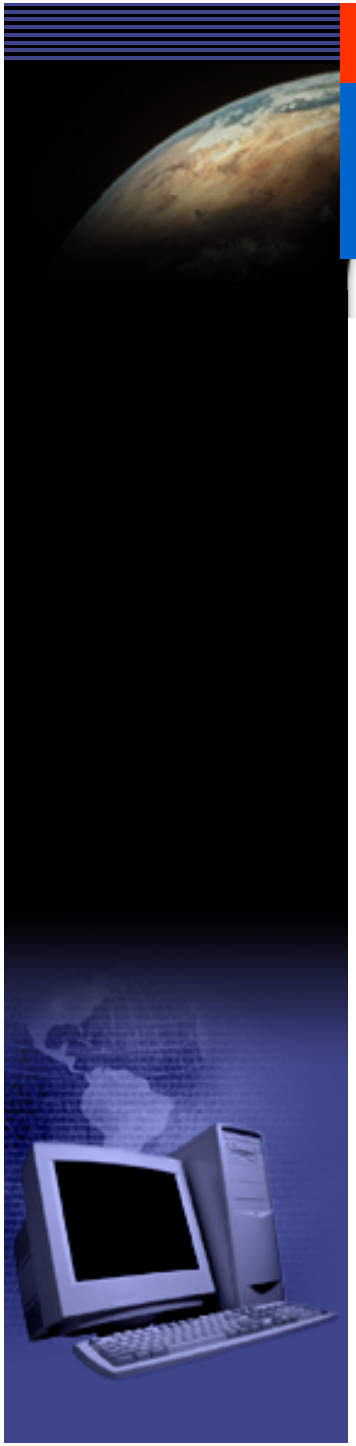
Điều phối FIFO

Điều phối tiến trình

❖ Chiến lược điều phối



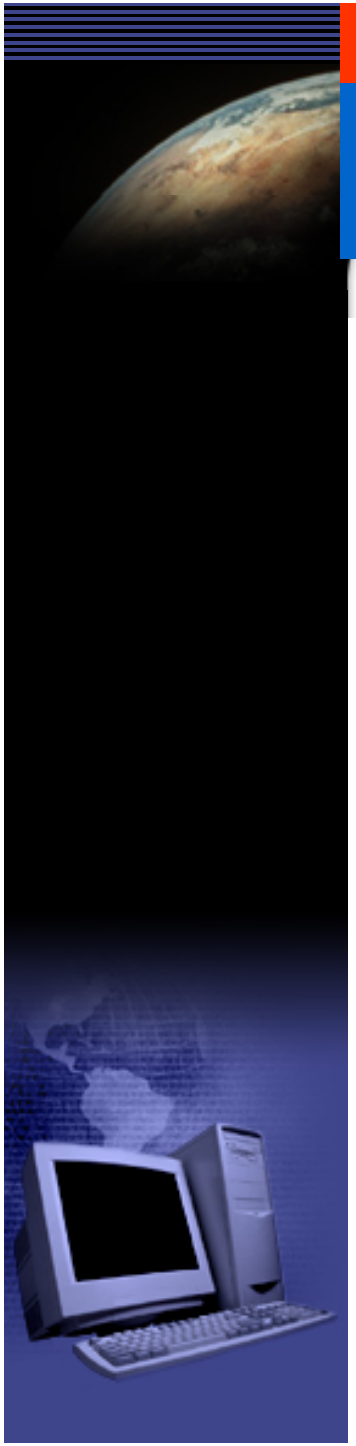
Điều phối Round Robin



Điều phối tiến trình

❖ Tổ chức điều phối

- Danh sách sẵn sàng (Ready List)
- Danh sách chờ (Waiting List)
- Các danh sách chờ riêng cho từng tài nguyên (thiết bị ngoại vi)



Đồng bộ hoá tiến trình

❖ Nhu cầu đồng bộ hoá

- Yêu cầu truy xuất độc quyền
- Yêu cầu phối hợp

Đồng bộ hoá tiến trình

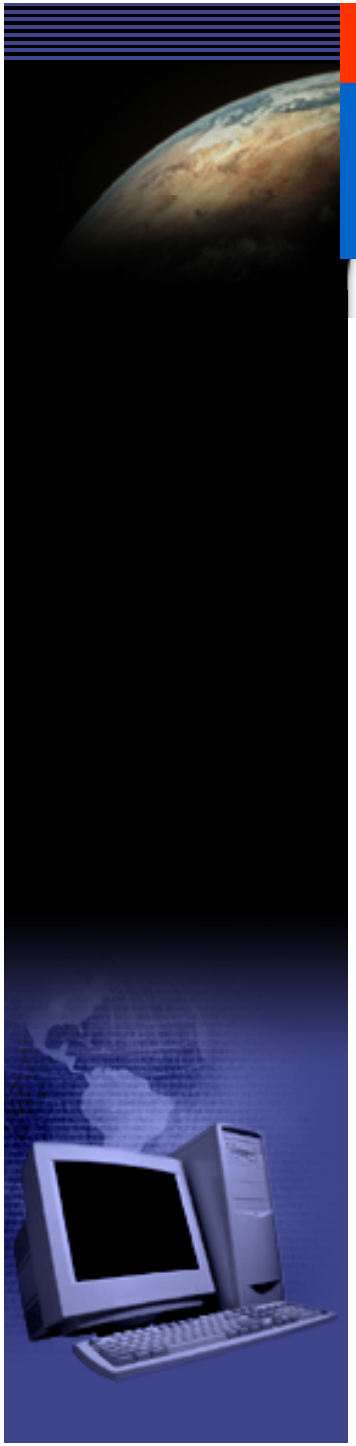
❖ Miền găng (Critical Section)

- **Vấn đề tranh đoạt điều khiển**

```
if (taikhoan-tienrut)>=0
    taikhoan=taikhoan-tienrut;
else
    error (<<khong the rut tien!>>);
```

- **Khái niệm miền găng:**

  Đoạn chương trình có khả năng xảy ra các mâu thuẫn truy xuất trên tài nguyên chung

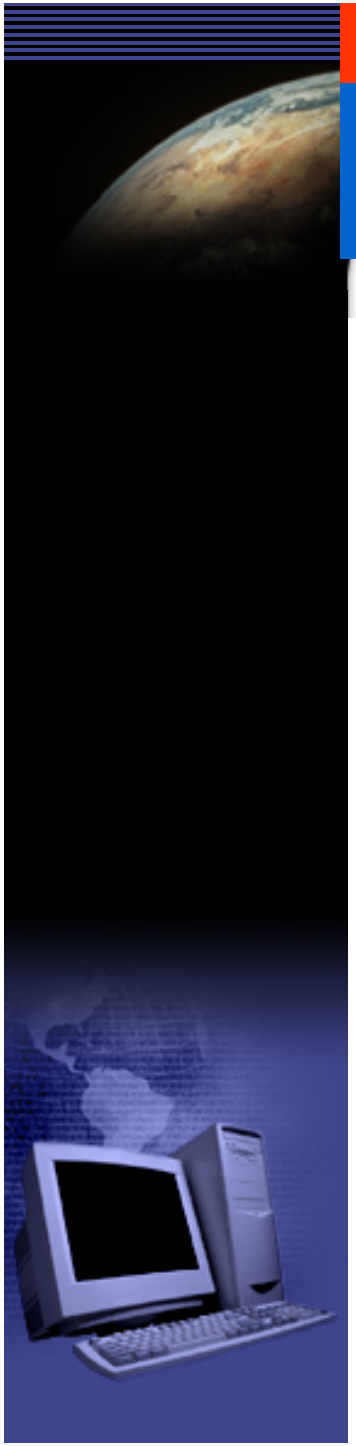


Đồng bộ hoá tiến trình

❖ Miền găng (Critical Section)

- **Điều kiện giải quyết tốt bài toán miền găng:**

- **Không có 2 tiến trình cùng ở trong miền găng**
- **Không phụ thuộc vào tốc độ của tiến trình**
- **Một tiến trình tạm dừng bên ngoài miền găng không được ngăn cản các tiến trình khác vào miền găng**
- **Không có tiến trình nào phải chờ vô hạn để được vào miền găng.**



Đồng bộ hoá tiến trình

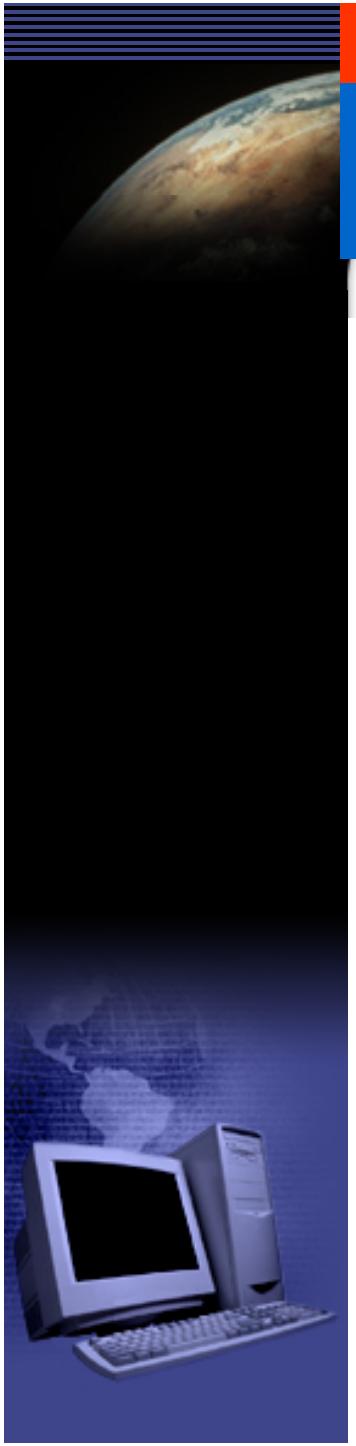
❖ Giải pháp

➢ Sử dụng biến khoá

- Dùng biến lock chung cho các tiến trình

- Nếu lock==1 thì khoá, không cho tiến trình vào miền găng. Chờ cho đến khi lock==0

- Nếu lock==0 thì cho tiến trình vào miền găng, đặt lock==1 để khoá không cho các tiến trình khác vào miền găng



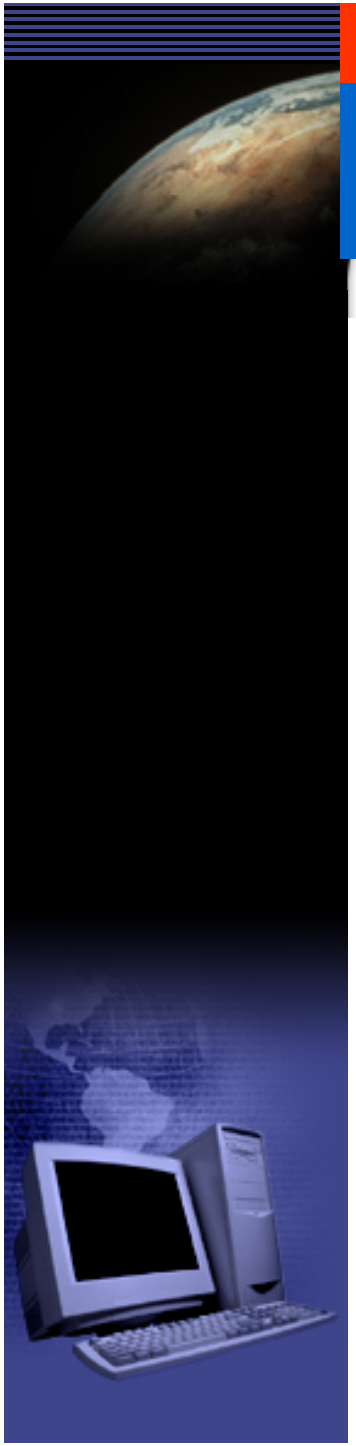
Đồng bộ hoá tiến trình

❖ Giải pháp

➢ Sử dụng biến khoá

- Giải thuật sử dụng biến khoá để đồng bộ

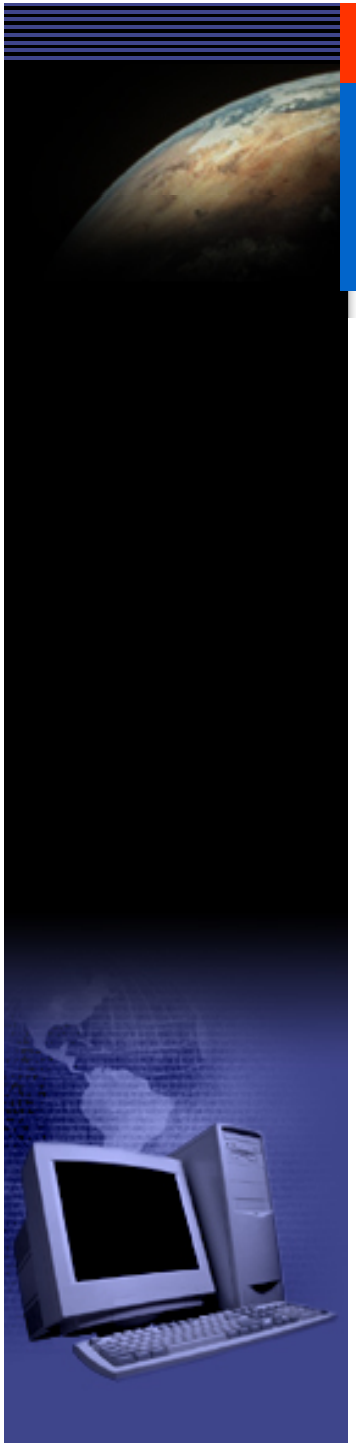
```
while (1) {
    while (lock==1);// wait
    lock=1;
    critical_section();
    lock=0;
    Noncritical_section();
}
```



Đồng bộ hoá tiến trình

- Giải thuật sử dụng biến khoá để đồng bộ

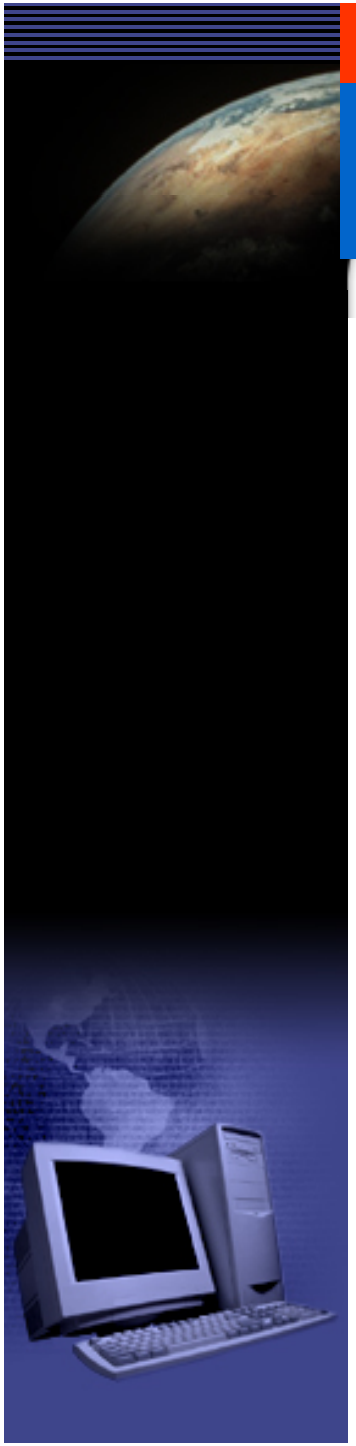
```
while (1) {
    while (lock==1);// wait
    lock=1;
    critical_section();
    lock=0;
    non_critical_section();
    }
```



Đồng bộ hoá tiến trình

- Ví dụ: Áp dụng giải thuật sử dụng biến khoá để đồng bộ

```
while (1) {
    t=t*2;
    while (lock==1);// wait
    lock=1;
    for  (s=0,i=0;i<=t;i++)  s+=i;
    printf("s=%i",s);
    lock=0;
    break;
    }
```



Đồng bộ hoá tiến trình

❖ Giải pháp

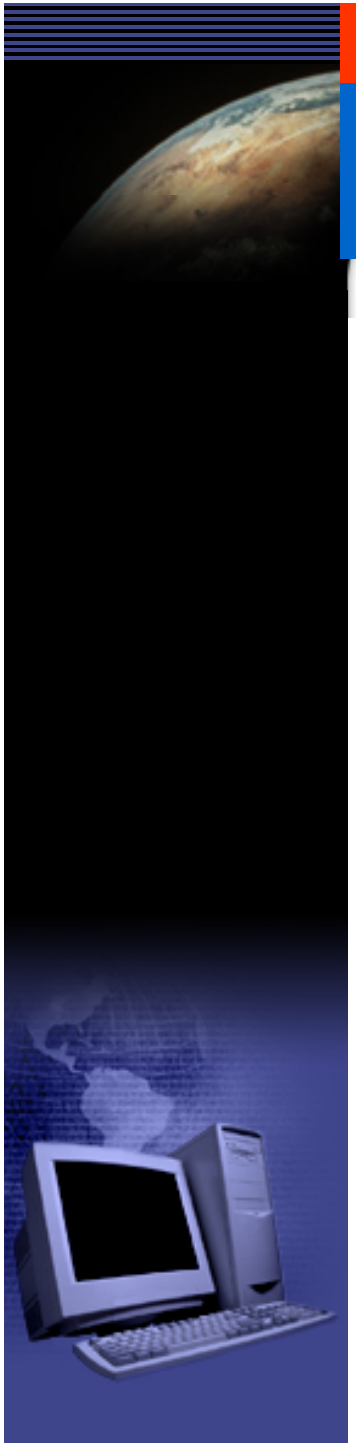
➢Kiểm tra luân phiên

- Các tiến trình muốn đi vào miền găng thì được gắn nhãn 0|1

- Sử dụng biến turn để chỉ thứ tự luân phiên.

- Nếu turn==0: tiến trình có nhãn 0 được vào miền găng

- Nếu turn==1: tiến trình có nhãn 1 được vào miền găng



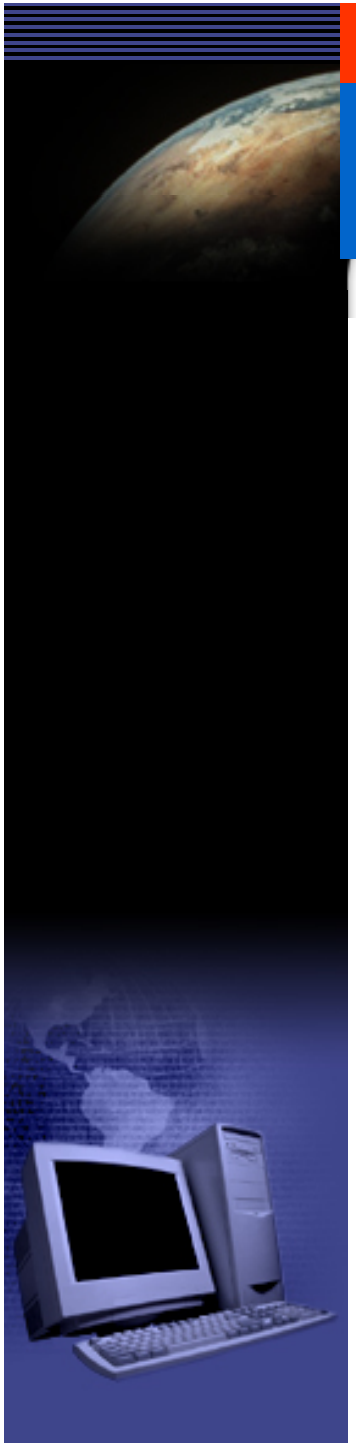
Đồng bộ hoá tiến trình

❖ Giải pháp

➢Kiểm tra luân phiên

- Giải thuật của tiến trình có nhãn 0

```
while (1) {
    while (turn != 0);// wait
    critical_section();
    turn=1;
    non_critical_section();
}
```



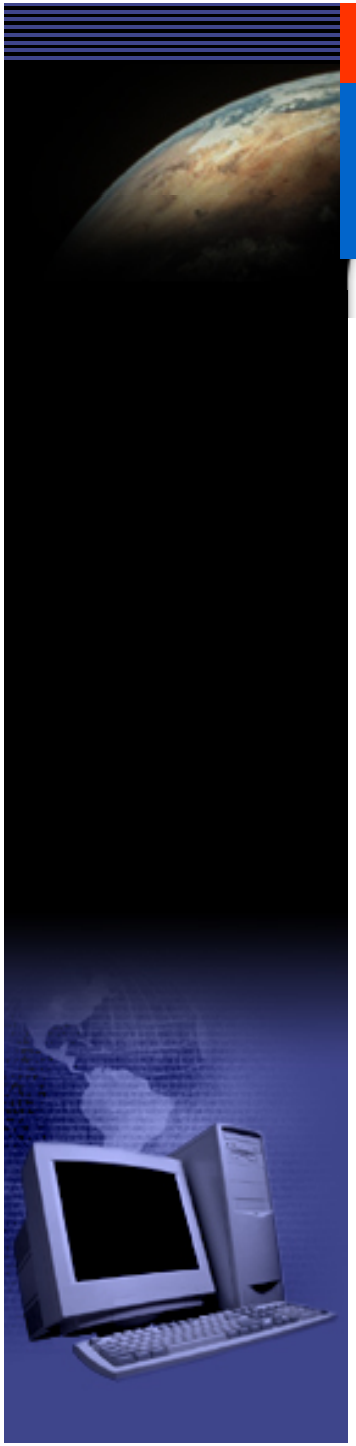
Đồng bộ hoá tiến trình

❖ Giải pháp

➢Kiểm tra luân phiên

- Giải thuật của tiến trình có nhãn 1

```
while (1) {
    while (turn != 1);// wait
    critical_section();
    turn=0;
    non_critical_section();
}
```



Đồng bộ hoá tiến trình

❖ Giải pháp

➢ Giải pháp Peterson

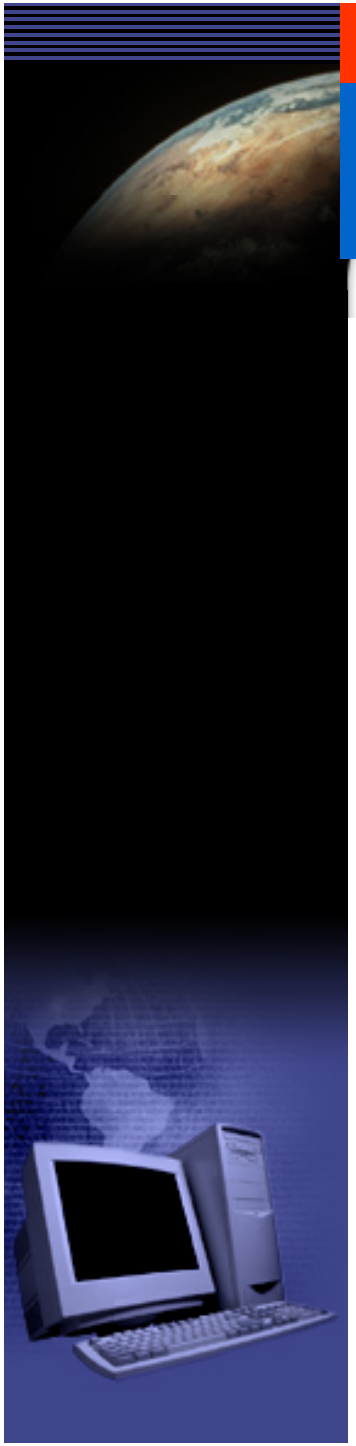
```
#define N  2  // Chỉ 2 tiến trình
int turn=0, interested[N]={0,0};
void enter_region(int process) // Vào ĐG
{  int other=1-process;//other là tiến trình đối của process
   interested[prcess]=1;
   turn=process;
   while ((turn==process)&&interested[other]==1);//chờ
}
```

Đồng bộ hoá tiến trình

❖ Giải pháp

➢ Giải pháp Peterson

```
void leave_region(int process) // Ra khỏi ĐG
{
  interested[prcess]=0;
}
```



Đồng bộ hoá tiến trình

❖ Giải pháp

➢ Giải pháp Sleep and Wakeup

- Sử dụng 2 thủ tục: sleep và wakeup

- Khi tiến trình chưa đủ điều kiện để vào miền găng, nó goi sleep để tự khoá đến khi một tiến trình khác gọi wakeup để đánh thức nó.

- Tiến trình khi ra khỏi miền găng sẽ gọi wakeup để đánh thức tiến trình khác.

- int busy;// 1: nếu miền găng đang bận, 0:không bận

- int blocked;//đếm số lượng tiến trình đang bị khoá

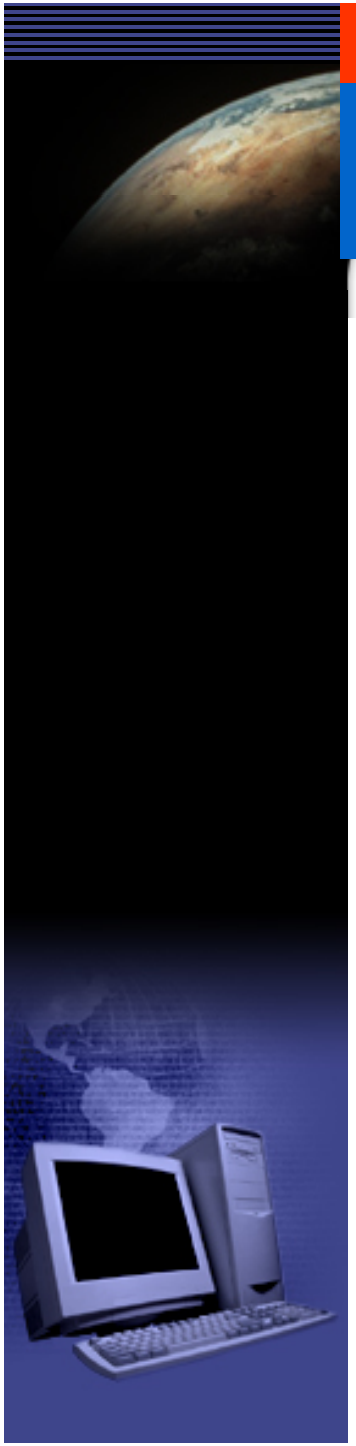
Đồng bộ hoá tiến trình

❖ Giải pháp

➢ Giải pháp Sleep and Wakeup

Giải thuật:

```
while (1) {
    if (busy) {
        blocked=blocked+1;
        sleep();
    }
    else busy=1;
    critical_section();
    busy=0;
    if (blocked) {
        wakeup(process);
        blocked=blocked-1;
    }
    noncritical_section();
}
```



Xác định trạng thái an toàn

❖ **Thuật toán:**

Sử dụng các cấu trúc dữ liệu sau:

int allocation[numprocs,numresources];

//allocation[p,r] số lượng tài nguyên r thực sự cấp phát cho p

int max[numprocs,numresources];

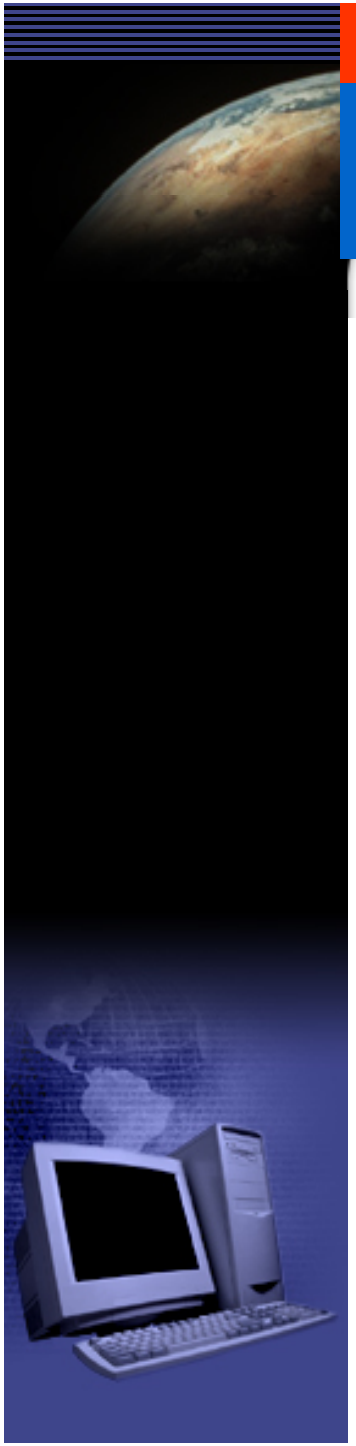
// max[p,r] nhu cầu tối đa của tiến trình p về tài nguyên r

int need[numprocs,numresources];

//need[p,r]=max[p,r]-allocation[p,r]

int available[numresources]

//available[r] số lượng tài nguyên r còn tự do



Xác định trạng thái an toàn

❖ **Thuật toán:**

int word[numresouces]=available;

int finish[numproces]=false;

1. **Tìm i sao cho**

   a. finish[i]==false;

   b. need[i,j]<=word[j]; với mọi tài nguyên j

   nếu không có i như thế, đến bước 3

2. **Word[j]=word[j]+allocation[i,j];**

   finish[i]=true;

   đến bước 1;

3. **Nếu finish[i]==true với mọi i thì hệ thống ở trạng thái an toàn. Ngược lại hệ thống bị tắc nghẽn**

Xác định trạng thái an toàn

❖ **ví dụ: giả sử tình trạng hiện hành của hệ thống được mô tả ở bảng dưới. Nếu tiến trình P2 yêu cầu cấp 4 R1, 1 R3. Hãy cho biết yêu cầu này có thể đáp ứng mà không xảy ra tình trạng tắt nghẽn**

	Max			Allocation			Available		
	R1	R2	R3	R1	R2	R3	R1	R2	R3
P1	3	2	2	1	0	0	4	1	2
P2	6	1	3	2	1	1			
P3	3	1	4	2	1	1			
P4	4	2	2	0	0	2			

Xác định trạng thái an toàn

❖ **ví dụ:**

Available[1]=4, Available[3]=2 đủ để thoả mãn yêu cầu của P2, ta có

	Need			Allocation			Available		
	R1	R2	R3	R1	R2	R3	R1	R2	R3
P1	2	2	2	1	0	0	0	1	1
P2	0	0	1	6	1	2			
P3	1	0	3	2	1	1			
P4	4	2	0	0	0	2			

Xác định trạng thái an toàn

❖ **ví dụ:**

	Need			Allocation			Available		
	R1	R2	R3	R1	R2	R3	R1	R2	R3
P1	2	2	2	1	0	0	6	2	3
P2	0	0	0	0	0	0			
P3	1	0	3	2	1	1			
P4	4	2	0	0	0	2			

Xác định trạng thái an toàn

❖ **ví dụ:**

	Need			Allocation			Available		
	R1	R2	R3	R1	R2	R3	R1	R2	R3
P1	0	0	0	0	0	0	7	2	3
P2	0	0	0	0	0	0			
P3	1	0	3	2	1	1			
P4	4	2	0	0	0	2			

Xác định trạng thái an toàn

❖ **Ví dụ:**

	Need			Allocation			Available		
	R1	R2	R3	R1	R2	R3	R1	R2	R3
P1	0	0	0	0	0	0	9	3	4
P2	0	0	0	0	0	0			
P3	0	0	0	0	0	0			
P4	4	2	0	0	0	2			

Xác định trạng thái an toàn

❖ **Ví dụ:**

	Need			Allocation			Available		
	R1	R2	R3	R1	R2	R3	R1	R2	R3
P1	0	0	0	0	0	0	9	3	6
P2	0	0	0	0	0	0			
P3	0	0	0	0	0	0			
P4	0	0	0	0	0	0			

Trạng thái kết quả là an toàn, có thể cấp phát.

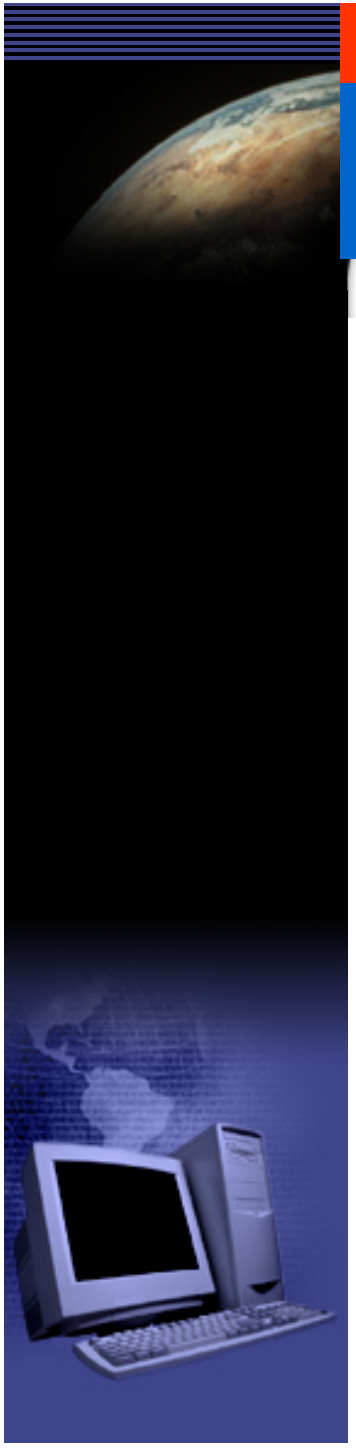
Xác định trạng thái an toàn

❖ **Bài tập:**

	Max			Allocation			Available		
	R1	R2	R3	R1	R2	R3	R1	R2	R3
P1	3	2	2	1	0	0	4	1	2
P2	6	1	3	2	1	1			
P3	3	1	4	2	1	1			
P4	4	2	2	0	0	2			

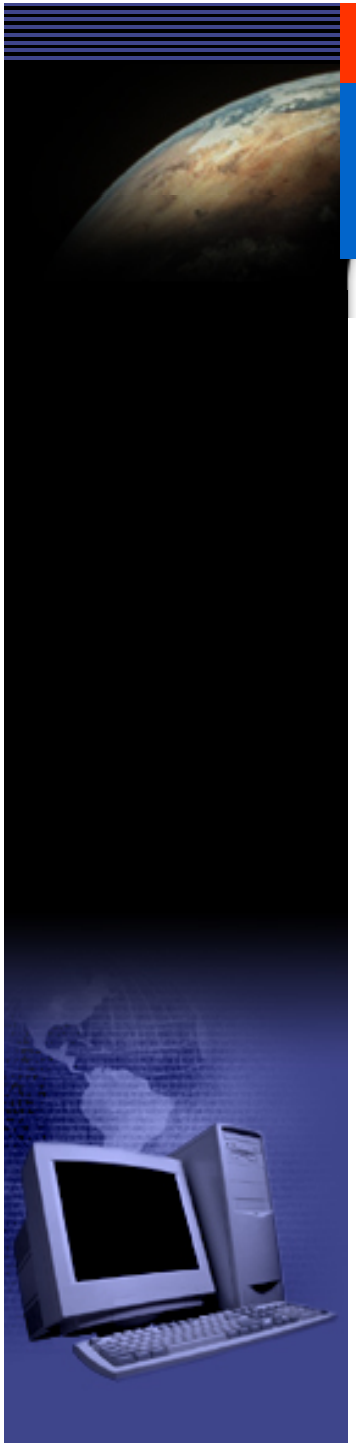
Tiến trình P2 yêu cầu 4 R1, 1 R3. Hãy cho biết yêu cầu này có thể đáp ứng mà đảm bảo không xảy ra tình trạng tắt nghẽn hay không?

CHƯƠNG 4. QUẢN LÝ BỘ NHỚ



Các vấn đề

1. Khái niệm
2. Không gian địa chỉ và không gian vật lý
3. Cấp phát liên tục
4. Cấp phát không liên tục
5. Bộ nhớ ảo

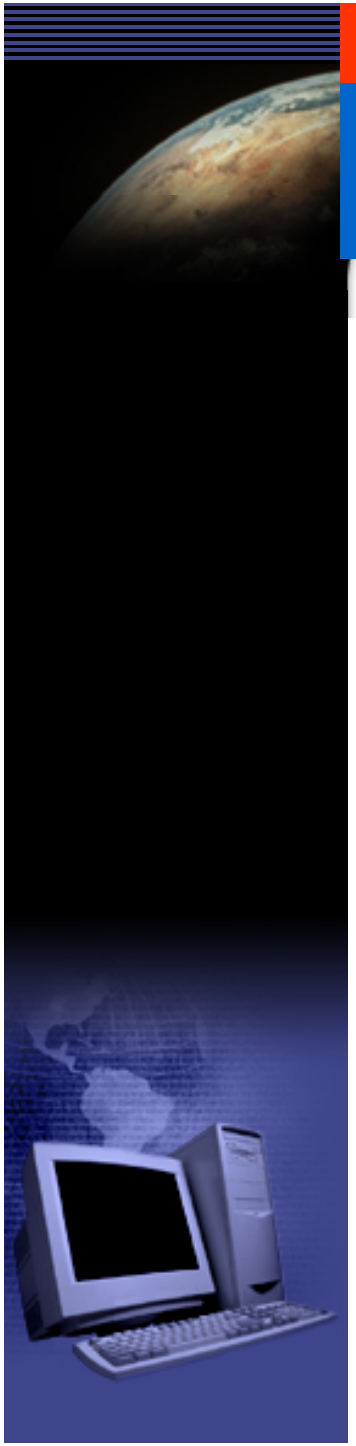


Khái niệm

- **Bộ nhớ là thiết bị lưu trữ duy nhất thông qua đó CPU có thể trao đổi thông tin với môi trường ngoài.**
- **Bộ nhớ chính được tổ chức như một mảng một chiều các từ nhớ (word), mỗi từ nhớ có một địa chỉ.**
- **Việc trao đổi với môi trường ngoài thông qua thao tác đọc, ghi dữ liệu vào một địa chỉ cụ thể trong bộ nhớ**

Khái niệm

- **Hệ điều hành thực hiện:**
  - **Sự tương ứng giữa địa chỉ logic và địa chỉ vật lý**
  - **Quản lý bộ nhớ vật lý**
  - **Chia sẻ thông tin**
  - **Bảo vệ**

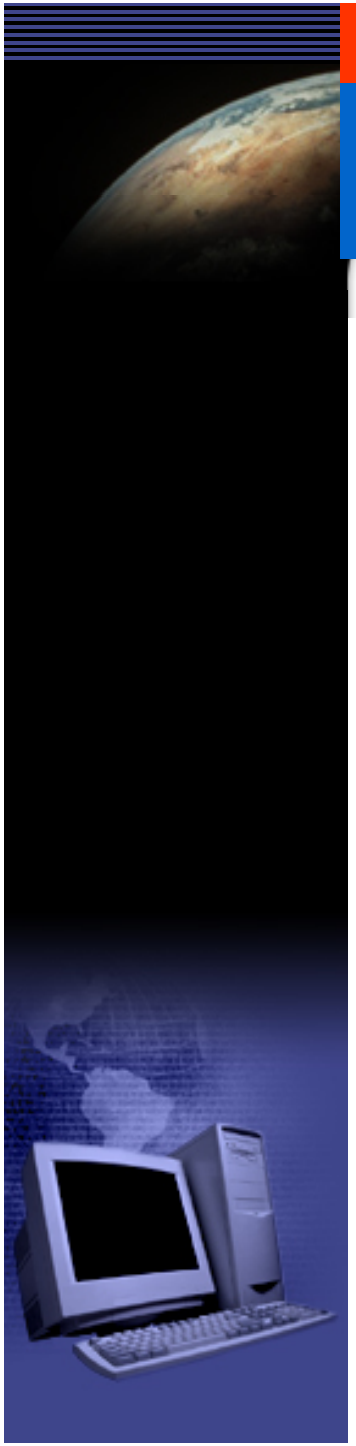


Không gian địa chỉ và không gian vật lý

- **Địa chỉ logic (địa chỉ ảo): các địa chỉ do bộ xử lý tạo ra.**
- **Địa chỉ vật lý: địa chỉ thực tế mà trình quản lý bộ nhớ nhìn thấy và thao tác.**
- **Không gian địa chỉ: tập hợp tất cả các địa chỉ ảo phát sinh bởi một chương trình.**

Không gian địa chỉ và không gian vật lý

- **Không gian vật lý: tập hợp tất cả các địa chỉ vật lý tương ứng với các địa chỉ ảo.**
- **MMU (Memory Management Unit): một cơ chế phần cứng chuyển đổi địa chỉ ảo thành địa chỉ vật lý.**
- **Chương trình của NSD chỉ thao tác trên địa chỉ ảo.**

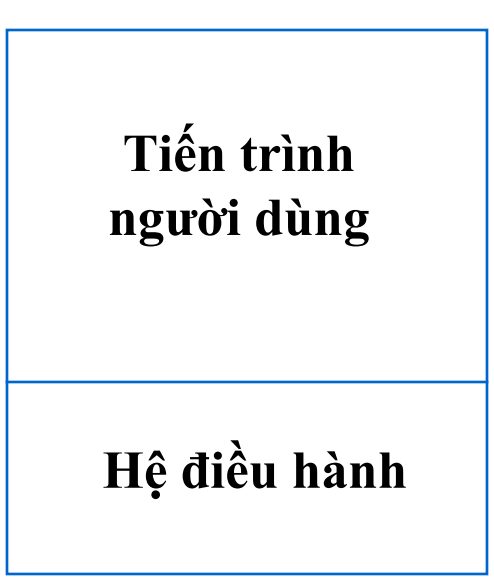


Cấp phát liên tục

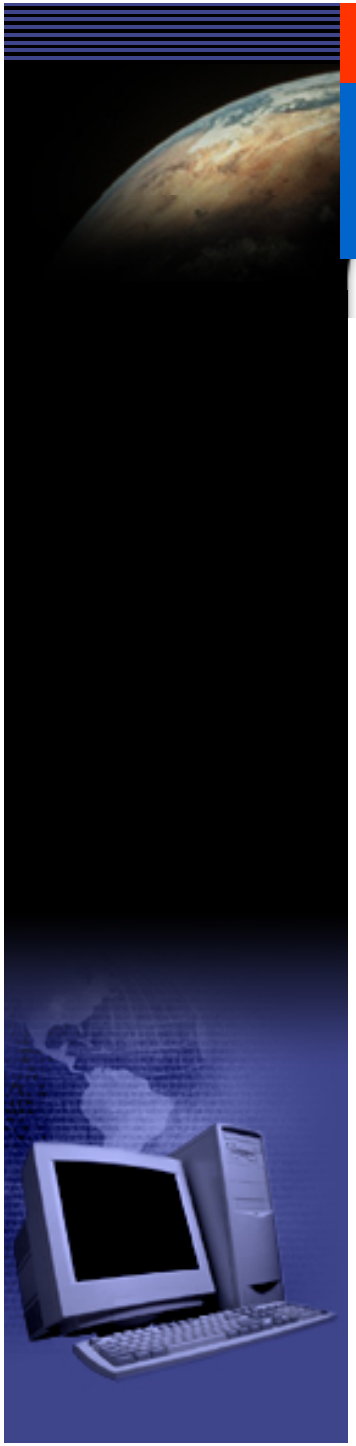
- Các hệ đơn chương
- Các hệ thống đa chương với phân vùng cố định
- Các hệ thống đa chương với phân vùng động
- Các hệ thống đa chương với kỹ thuật “Swapping”

Cấp phát liên tục

- Các hệ đơn chương



Tổ chức bộ nhớ trong hệ thống đơn chương



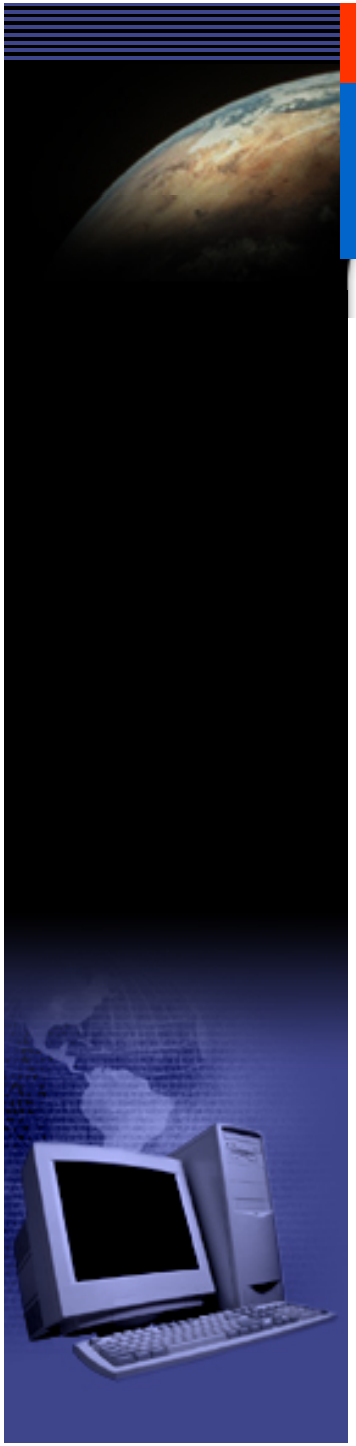
Cấp phát liên tục

❖ **Các hệ thống đơn chương**

- **Sử dụng thanh ghi giới hạn: địa chỉ cao nhất của vùng nhớ được cấp cho HĐH**
- **Tất cả các địa chỉ được tiến trình NSD truy xuất đến sẽ được so sánh với nội dung thanh ghi giới hạn.**

  + Nếu lớn hơn: hợp lý.

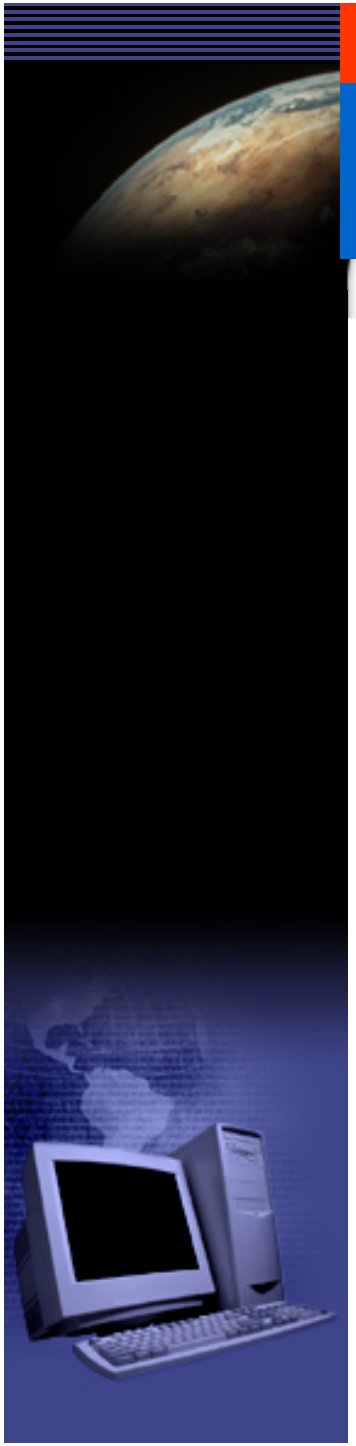
  + Ngược lại : một ngắt sẽ được phát sinh báo sự truy xuất bất hợp lý.
- **Tại một thời điểm chỉ có một chương trình được xử lý.**



Cấp phát liên tục

- **Các hệ thống đơn chương**

Ví dụ: Trong HĐH MSDOS, một lúc chỉ thực thi được một lệnh. Khi NSD gõ lệnh lập tức lệnh đó được thực hiện và sau khi hoàn tất, con trỏ xuất hiện sau dấu nhắc đợi lệnh chờ NSD gõ lệnh tiếp theo.

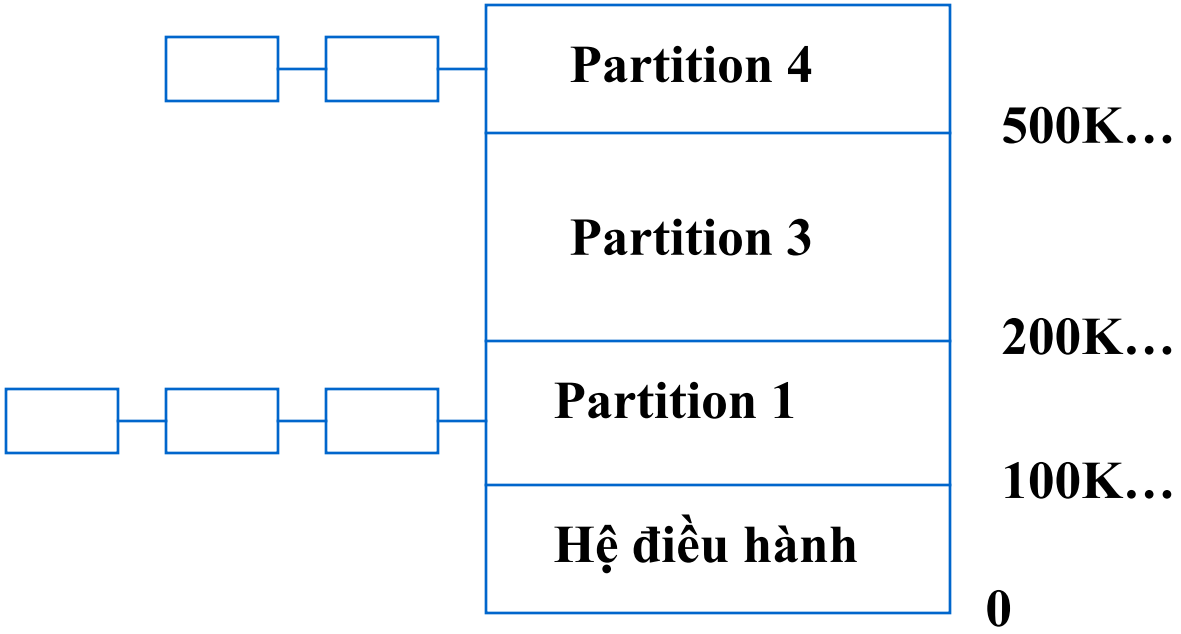


Cấp phát liên tục

- **Các hệ thống đa chương với phân vùng cố định**
  - **Bộ nhớ được chia thành các phân vùng (kích thước khác hay bằng nhau)**
  - **Các tiến trình có nhu cầu bộ nhớ sẽ được lưu trữ vào hàng đợi.**
  - **Sử dụng nhiều hàng đợi**
  - **Sử dụng một hàng đợi**

Cấp phát liên tục

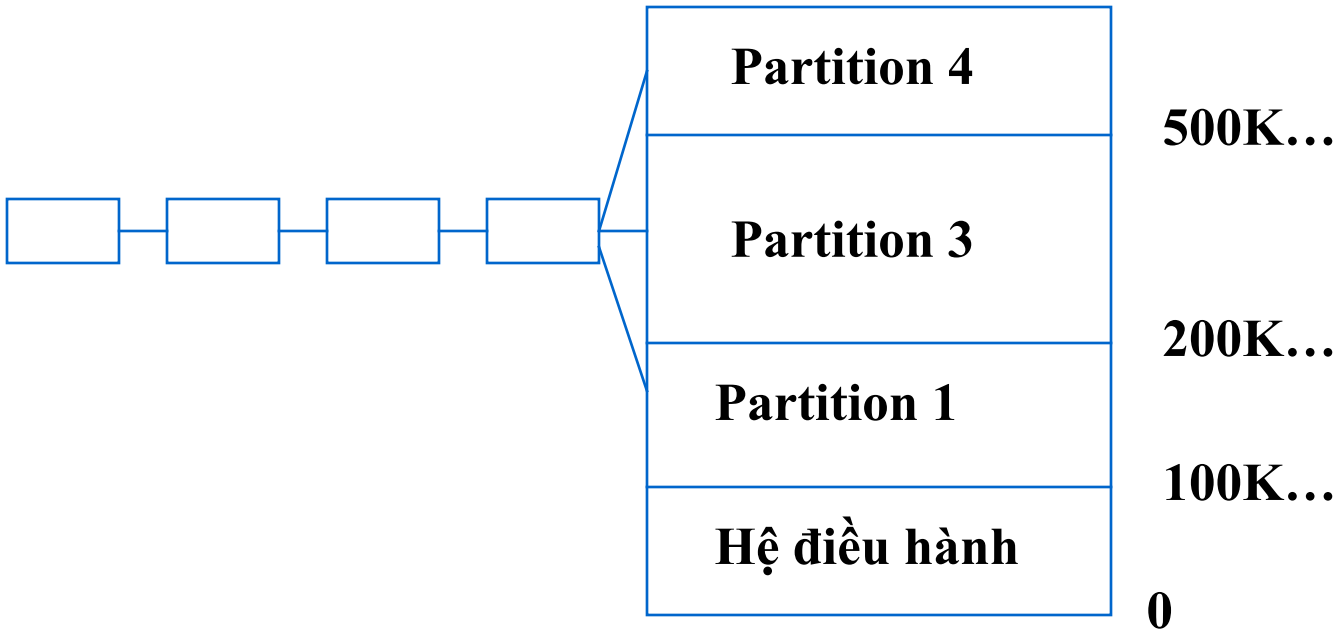
- Các hệ thống đa chương với phân vùng cố định



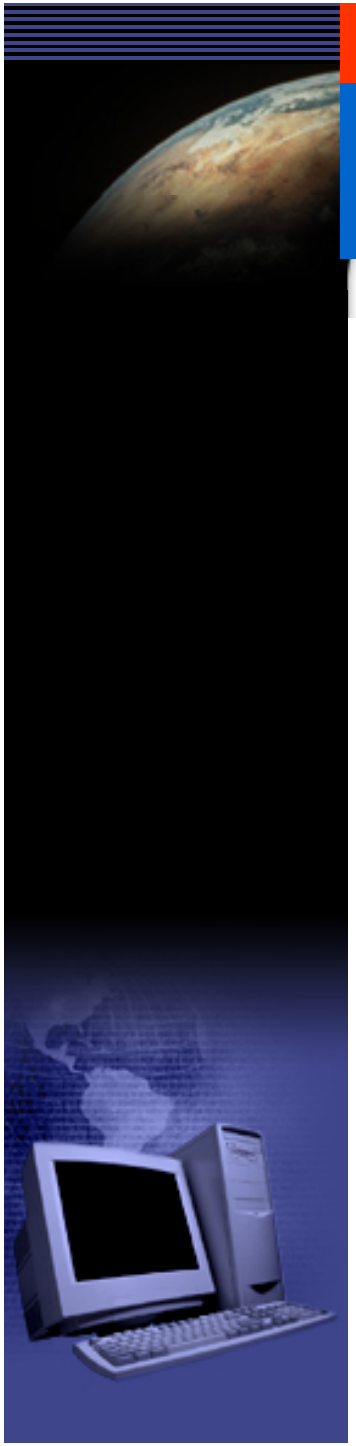
Phân vùng cố định nhiều hàng đợi

Cấp phát liên tục

- Các hệ thống đa chương với phân vùng cố định

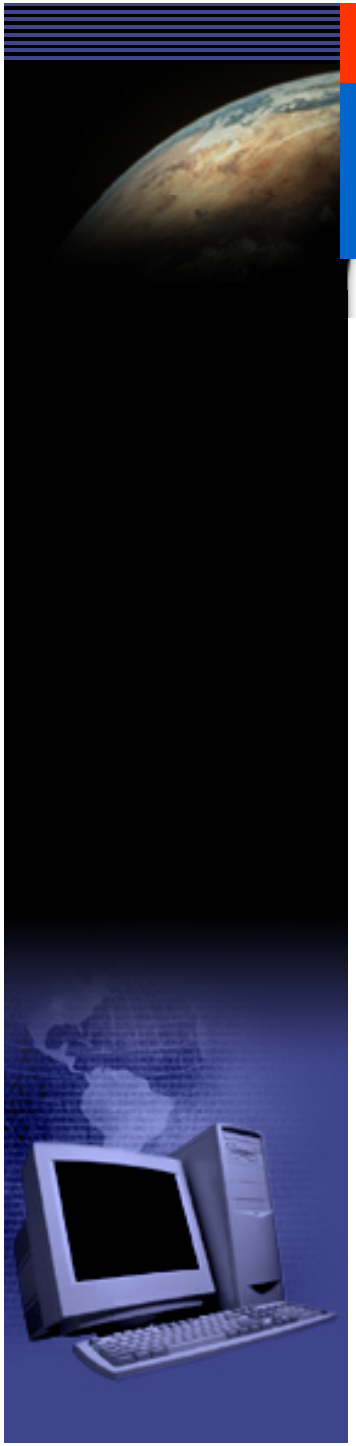


Phân vùng cố định một hàng đợi



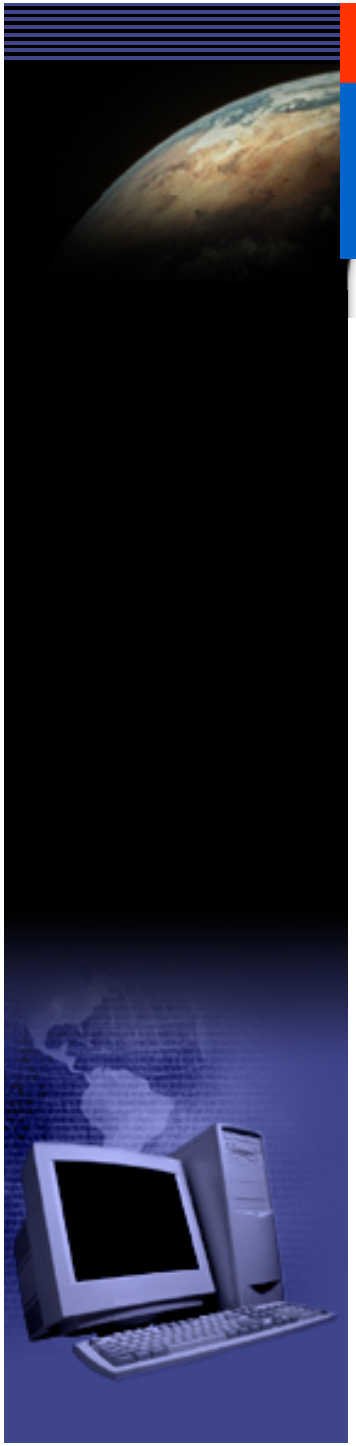
Cấp phát liên tục

- Các hệ thống đa chương với phân vùng cố định
  - Phân vùng cố định nhiều hàng đợi
    - Mỗi phân vùng có một hàng đợi
    - Mỗi tiến trình mới được tạo lập sẽ được đưa vào hàng đợi của phân vùng có kích thước nhỏ nhất đủ để thoả mãn nhu cầu chứa nó.
    - Các hàng đợi của một số phân vùng trống, đầy. Các tiến trình phải chờ được cấp phát bộ nhớ.



Cấp phát liên tục

- Các hệ thống đa chương với phân vùng cố định
- Phân vùng cố định một hàng đợi
  - Tất cả các tiến trình được đặt trong một hàng đợi.
  - Khi có một phân vùng tự do, tiến trình đầu tiên trong hàng đợi có kích thước phù hợp sẽ được đặt vào phân vùng này cho xử lý.
  - Kích thước của tiến trình không đúng bằng kích thước của phân vùng tự do $\Rightarrow$ phân mảnh nội vi
  - Mức độ đa chương bị giới hạn bởi số lượng phân vùng



Cấp phát liên tục

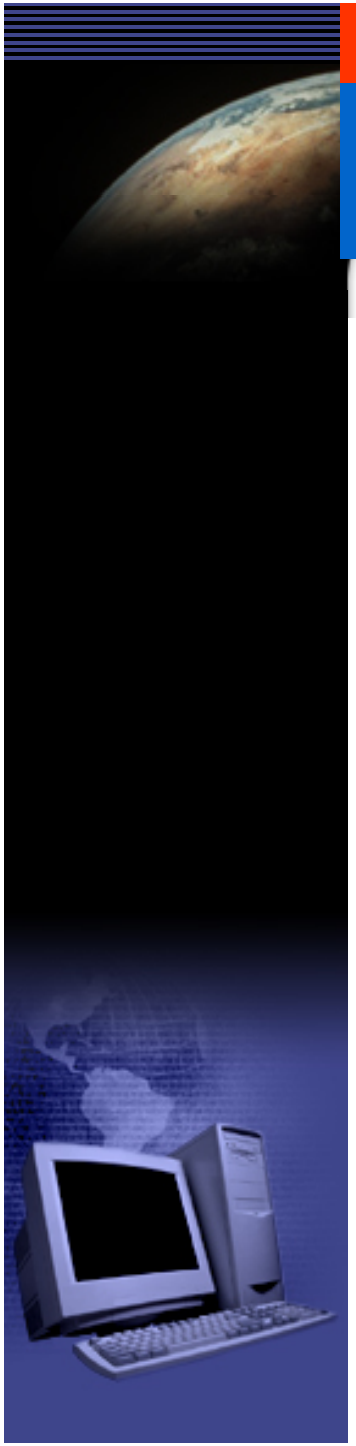
- Các hệ thống đa chương với phân vùng cố định
  - Phân vùng cố định một hàng đợi
    - Giải quyết 2 vấn đề của đa chương: sự tái định vị, sự bảo vệ

Ví dụ: giả sử chương trình truy xuất đến địa chỉ 100 (địa chỉ tương đối), ct được nạp vào phân vùng 1 địa chỉ bắt đầu 100k, thì địa chỉ truy xuất là (100k+100)

- Tái định vị vào thời điểm nạp chương trình

Cấp phát liên tục

- Các hệ thống đa chương với phân vùng cố định
- Phân vùng cố định một hàng đợi
- Sử dụng các thanh ghi đặc biệt: phần cứng
  - Thanh ghi nền (Base Register)
  - Thanh ghi giới hạn (Limit Register)
- Khi một tiến trình được tạo lập, nạp vào thanh ghi nền địa chỉ bắt đầu của phân vùng được nạp, nạp vào thanh ghi giới hạn kích thước của tiến trình.

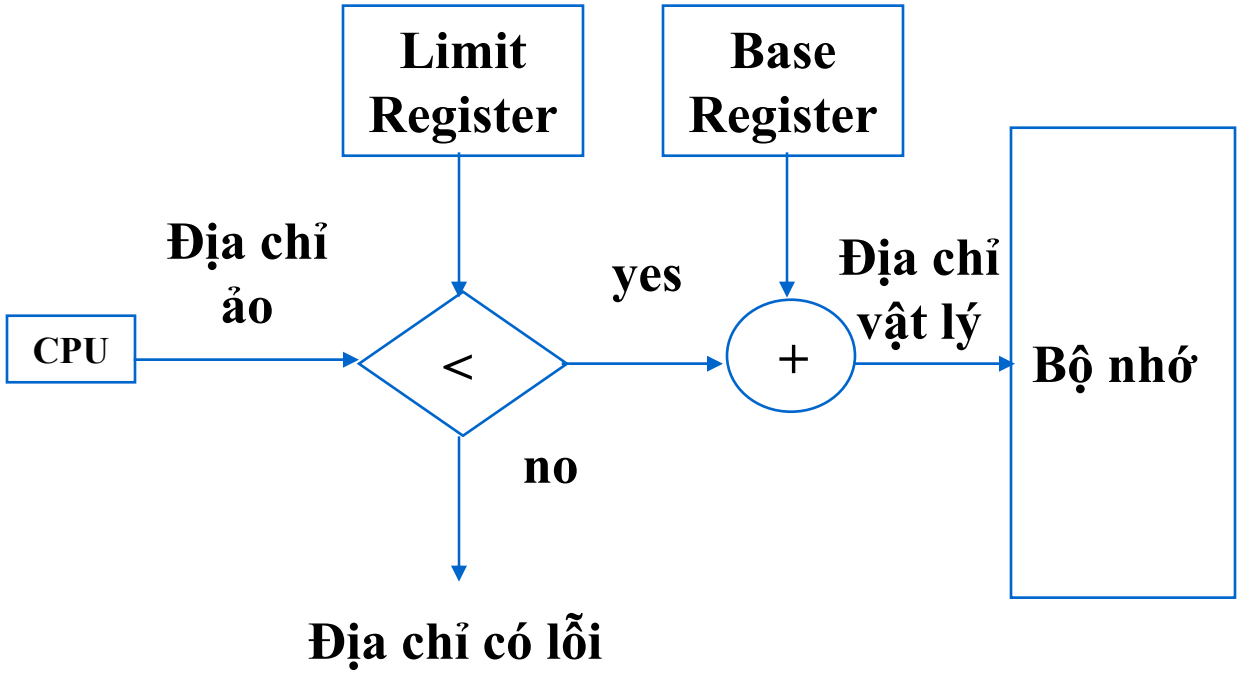


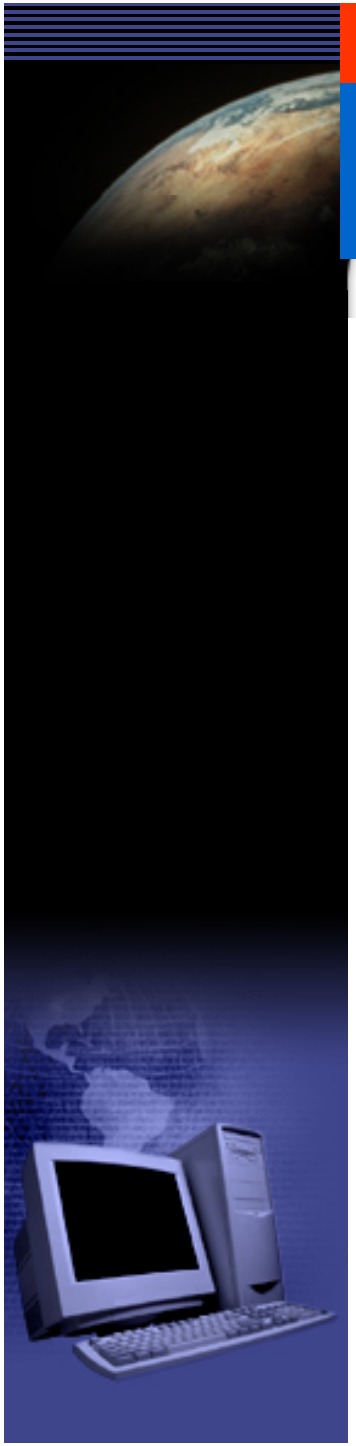
Cấp phát liên tục

- Các hệ thống đa chương với phân vùng cố định
- Phân vùng cố định một hàng đợi
  - Địa chỉ ảo được đối chiếu với thanh ghi giới hạn để bảo đảm tiến trình không truy xuất ngoài phạm vi phân vùng cấp cho nó.
  - Địa chỉ vật lý=địa chỉ ảo+địa chỉ trong thanh ghi nền.
  - Sử dụng thanh ghi nền là có thể di chuyển các chương trình trong bộ nhớ sau khi chúng bắt đầu xử lý. Chỉ cần nạp lại thanh ghi nền.

Cấp phát liên tục

- Các hệ thống đa chương với phân vùng cố định
- Phân vùng cố định một hàng đợi





Cấp phát liên tục

❖ **Các hệ thống đa chương với phân vùng động**

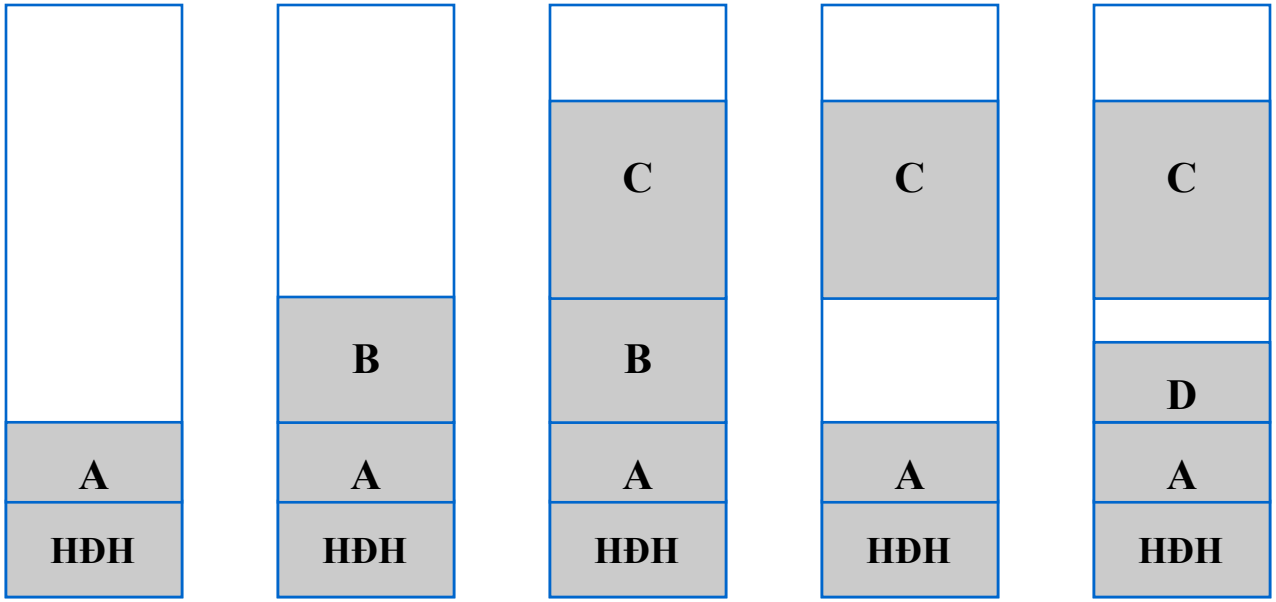
- **Xảy ra hiện tượng phân mảnh ngoại vi**
- **Kỹ thuật "dồn bộ nhớ": kết hợp các mảnh bộ nhớ nhỏ rời rạc thành một vùng nhớ lớn liên tục**

⇒ **Các tiến trình có thể bị di chuyển.**

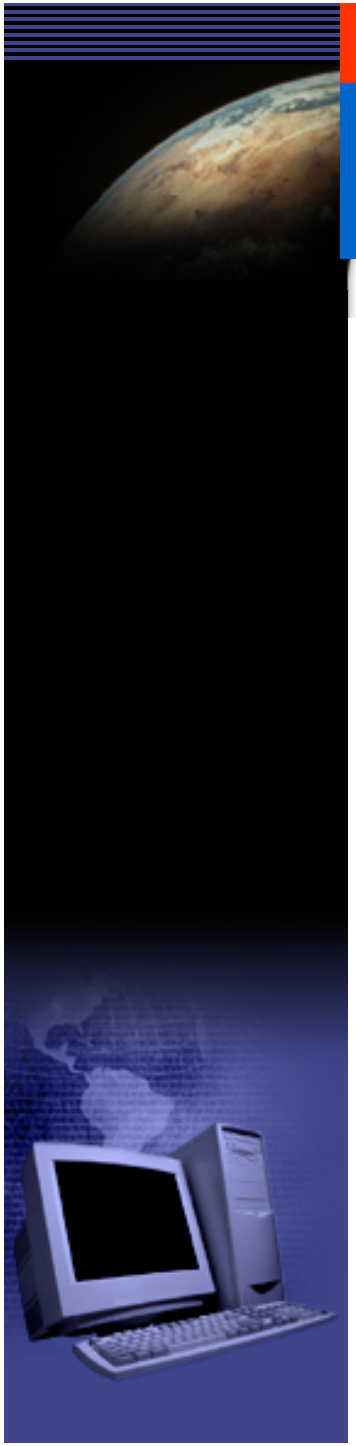
⇒ **Kích thước tiến trình tăng trưởng trong quá trình xử lý mà không còn vùng nhớ trống gần kề (dời chỗ tiến trình, cấp phát dư).**

Cấp phát liên tục

- Các hệ thống đa chương với phân vùng động



Cấp phát các phân vùng động

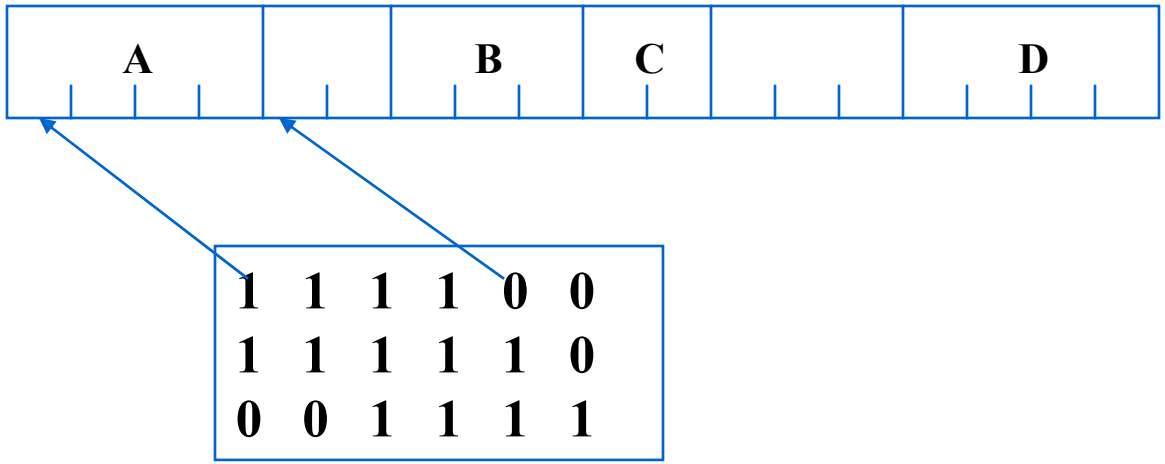


Cấp phát liên tục

- Các hệ thống đa chương với phân vùng động
  - Giải pháp cấp phát động
    - Quản lý bằng một bảng các bit
    - Quản lý bằng danh sách

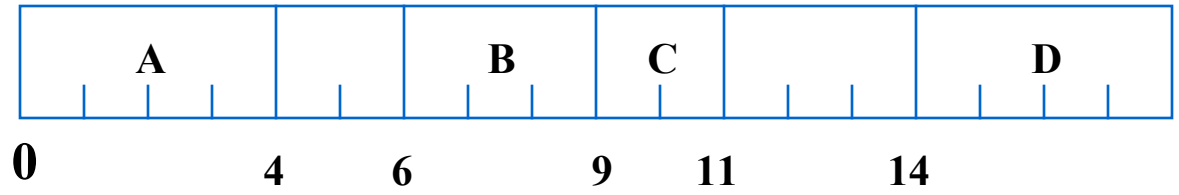
Cấp phát liên tục

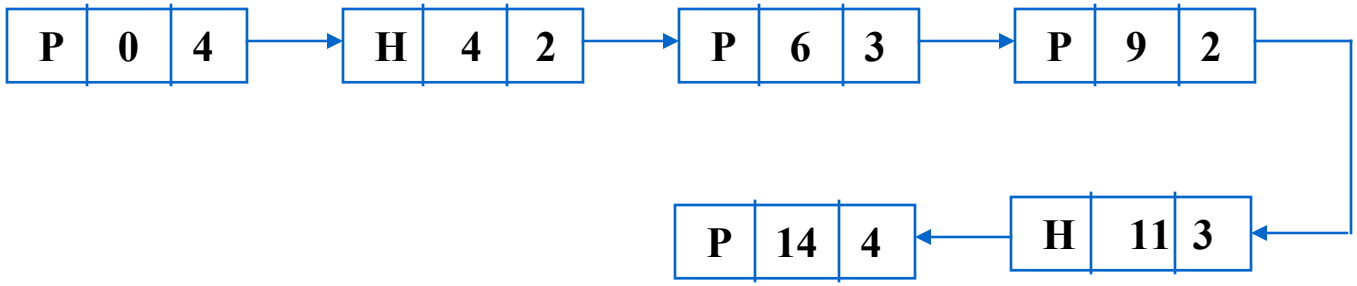
- Các hệ thống đa chương với phân vùng động
- Quản lý bằng một bảng các bit



Cấp phát liên tục

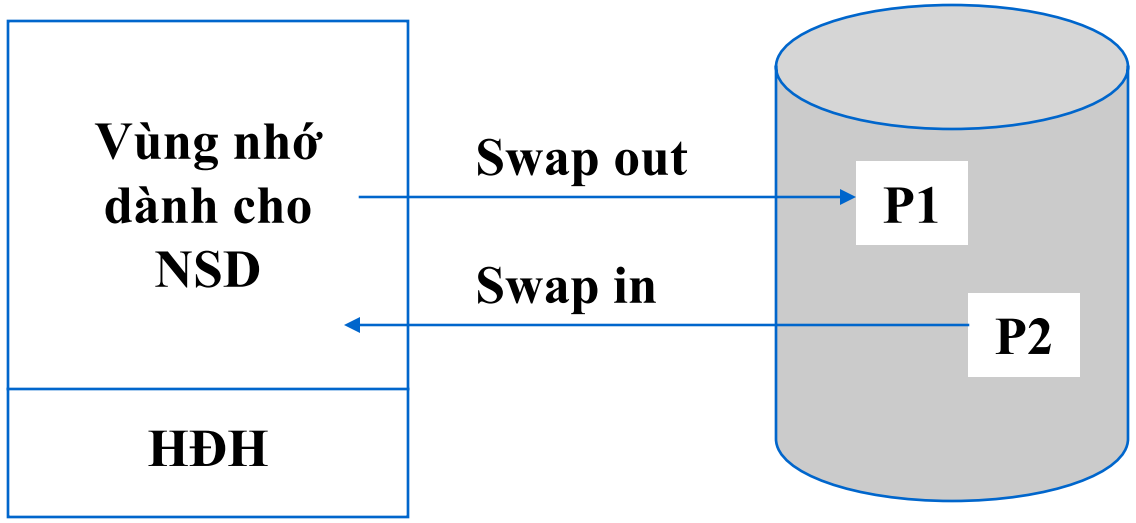
- Các hệ thống đa chương với phân vùng động
- Quản lý bằng danh sách

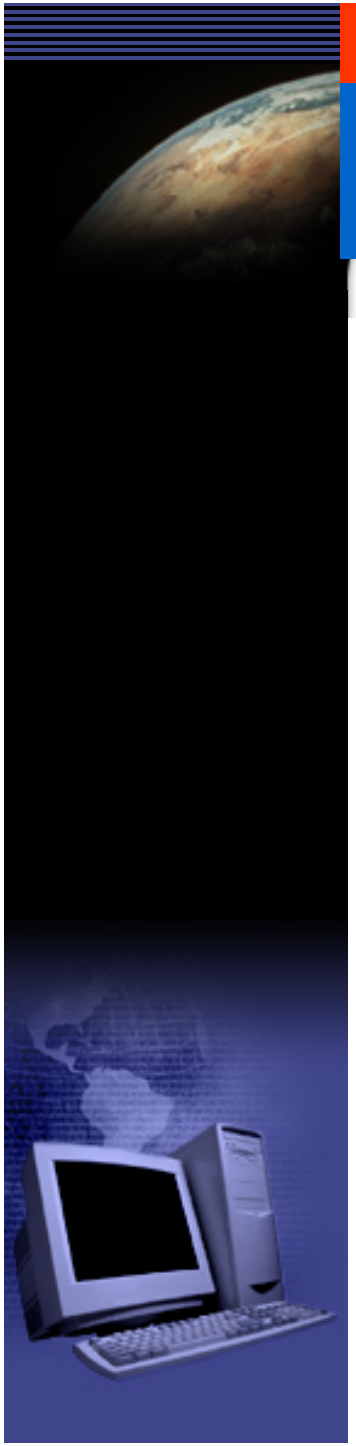




Cấp phát liên tục

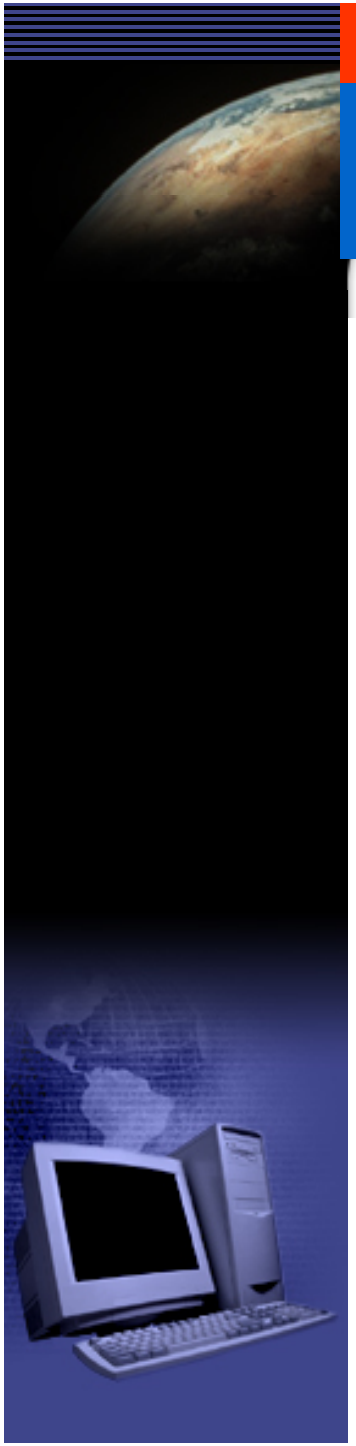
- Các hệ thống đa chương với kỹ thuật "Swapping"





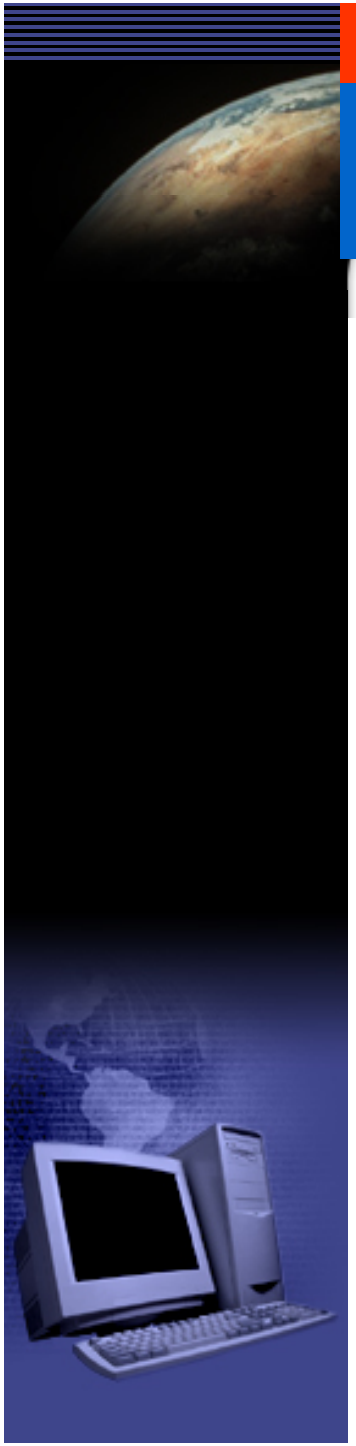
Cấp phát liên tục

- Các hệ thống đa chương với kỹ thuật "Swapping"
  - Chuyển một tiến trình đang ở trạng thái chờ nằm sang bộ nhớ phụ. (swap out)
  - Khi đến lượt nó sẽ được mang trở lại bộ nhớ chính để tiếp tục xử lý. (swap in)
  - Xảy ra hiện tượng phân mảng ngoại vi.



Cấp phát không liên tục

- Phân trang
- Phân đoạn
- Phân đoạn kết hợp phân trang



Cấp phát không liên tục

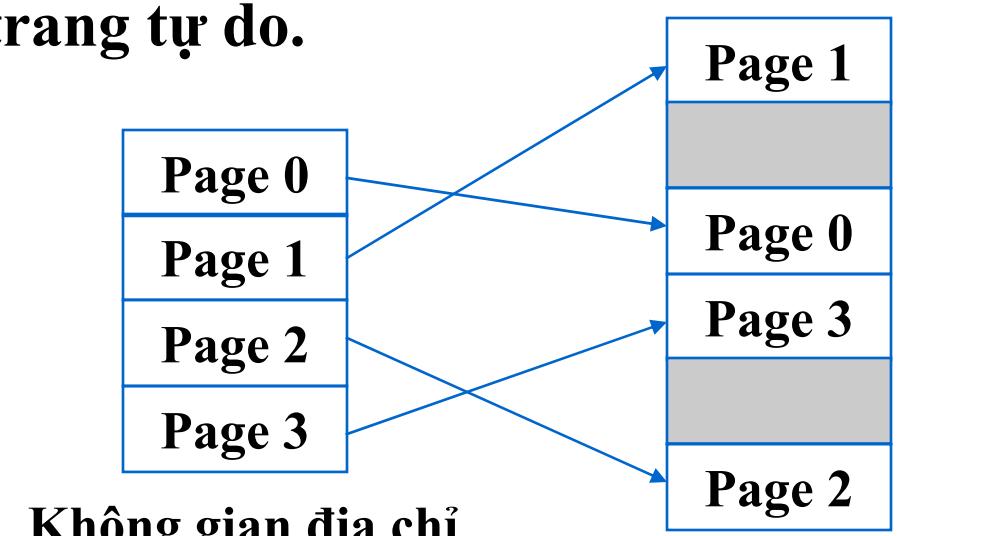
- **Phân trang**
  - **Ý tưởng**
  - **Cơ chế MMU**
  - **Chuyển đổi địa chỉ**
  - **Cài đặt bảng trang**
  - **Tổ chức bảng trang**

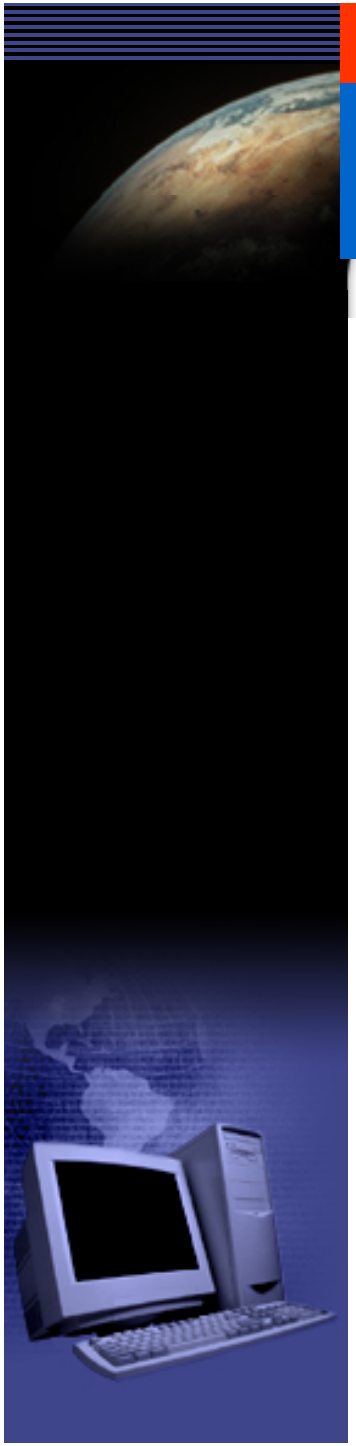
Cấp phát không liên tục

- **Phân trang**
  - **Ý tưởng**
    - **Bộ nhớ vật lý: chia thành các khối (khung trang) có kích thước bằng nhau.**
    - **Không gian địa chỉ: chia thành các khối (trang) có kích thước trùng bằng khung trang.**
    - **Khi cần nạp một tiến trình để xử lý, các trang của tiến trình sẽ được nạp vào các khung trang còn trống.**

Cấp phát không liên tục

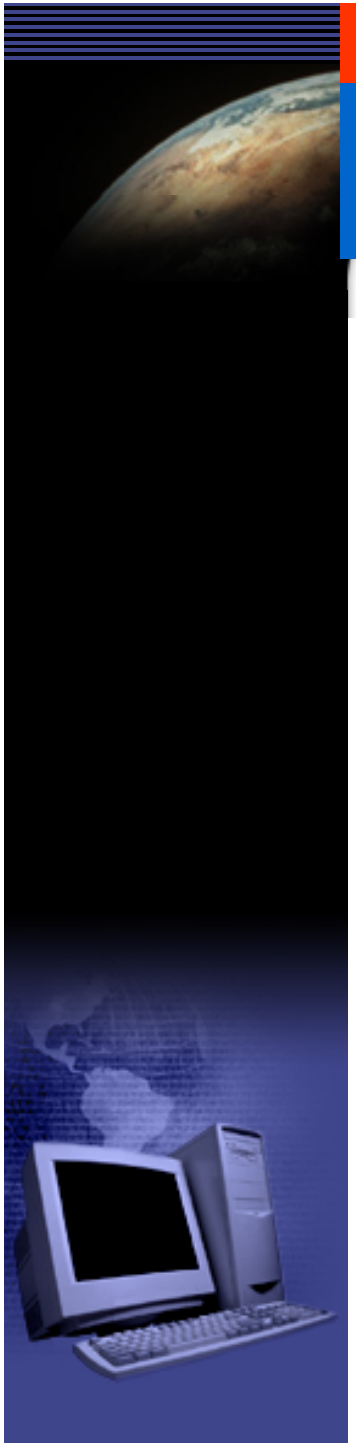
- **Phân trang**
  - **Ý tưởng**
    - **Tiến trình có kích thước N trang, sẽ yêu cầu N khung trang tự do.**





Cấp phát không liên tục

- **Phân trang**
  - **Cơ chế MMU(Memory Management Unit)**
    - **Cơ chế phần cứng hỗ trợ chuyển đổi địa chỉ trong cơ chế phân trang (bảng trang).**
    - **Mỗi phân tử trong bảng trang: địa chỉ bắt đầu lưu trữ trang tương ứng trong bộ nhớ vật lý;số hiệu khung trang tương ứng.**



Cấp phát không liên tục

- **Phân trang**
  - **Chuyển đổi địa chỉ**

- Địa chỉ phát sinh bởi CPU gồm 2 phần: p,d

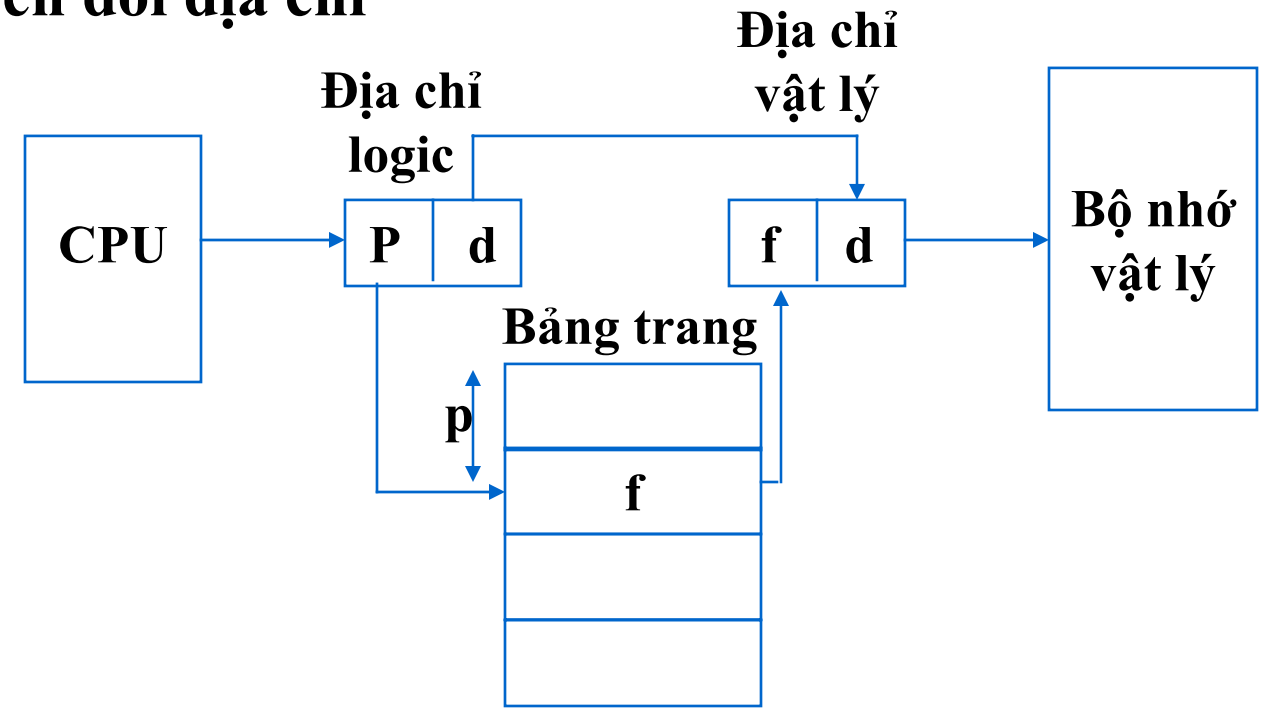
+ p: số hiệu trang

+ d: địa chỉ tương đối

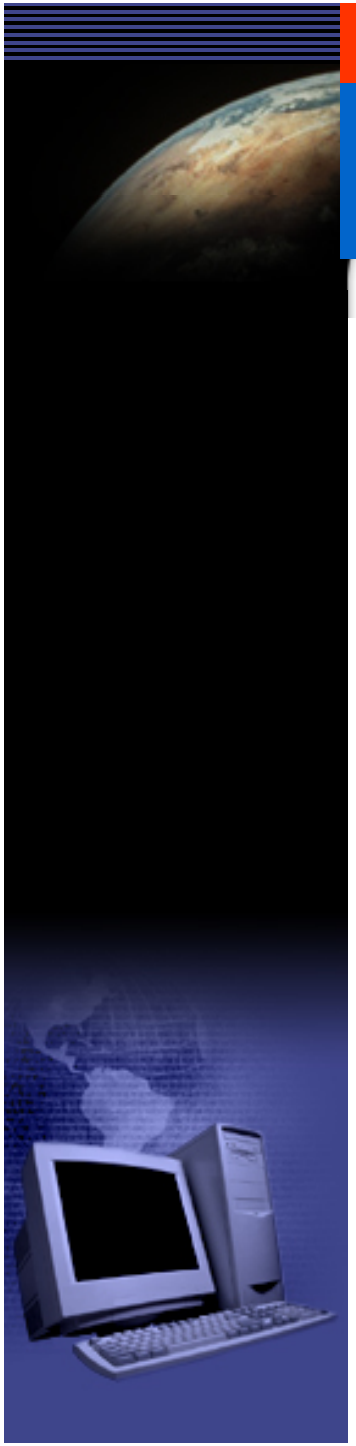
- Địa chỉ vật lý=địa chỉ bắt đầu của trang + d.

Cấp phát không liên tục

- Phân trang
  - Chuyển đổi địa chỉ



Cơ chế phần cứng hỗ trợ phân trang

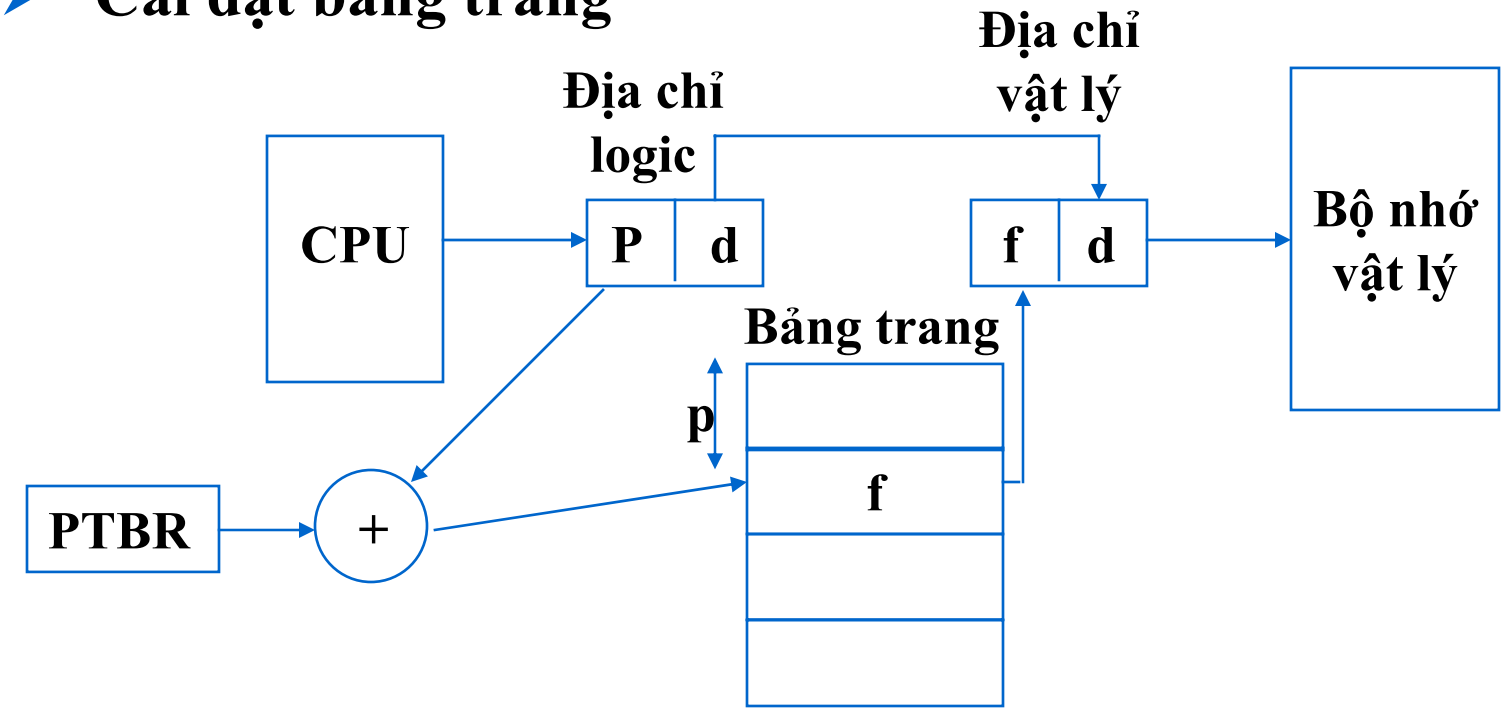


Cấp phát không liên tục

- **Phân trang**
  - **Cài đặt bảng trang**
    - **Sử dụng tập các thanh ghi: bảng trang có kích thước nhỏ.**
    - **Lưu trữ trong bộ nhớ, sử dụng thanh ghi nền(PTBR) để lưu địa chỉ bắt đầu bảng trang. (Page Table Basic Register)**
    - **Sử dụng bộ nhớ kết hợp (TLB), mỗi thanh ghi trong bộ nhớ gồm: (Translation Lookaside Buffers)**
      - **từ khoá: số hiệu trang**
      - **giá trị: số hiệu khung trang**

Cấp phát không liên tục

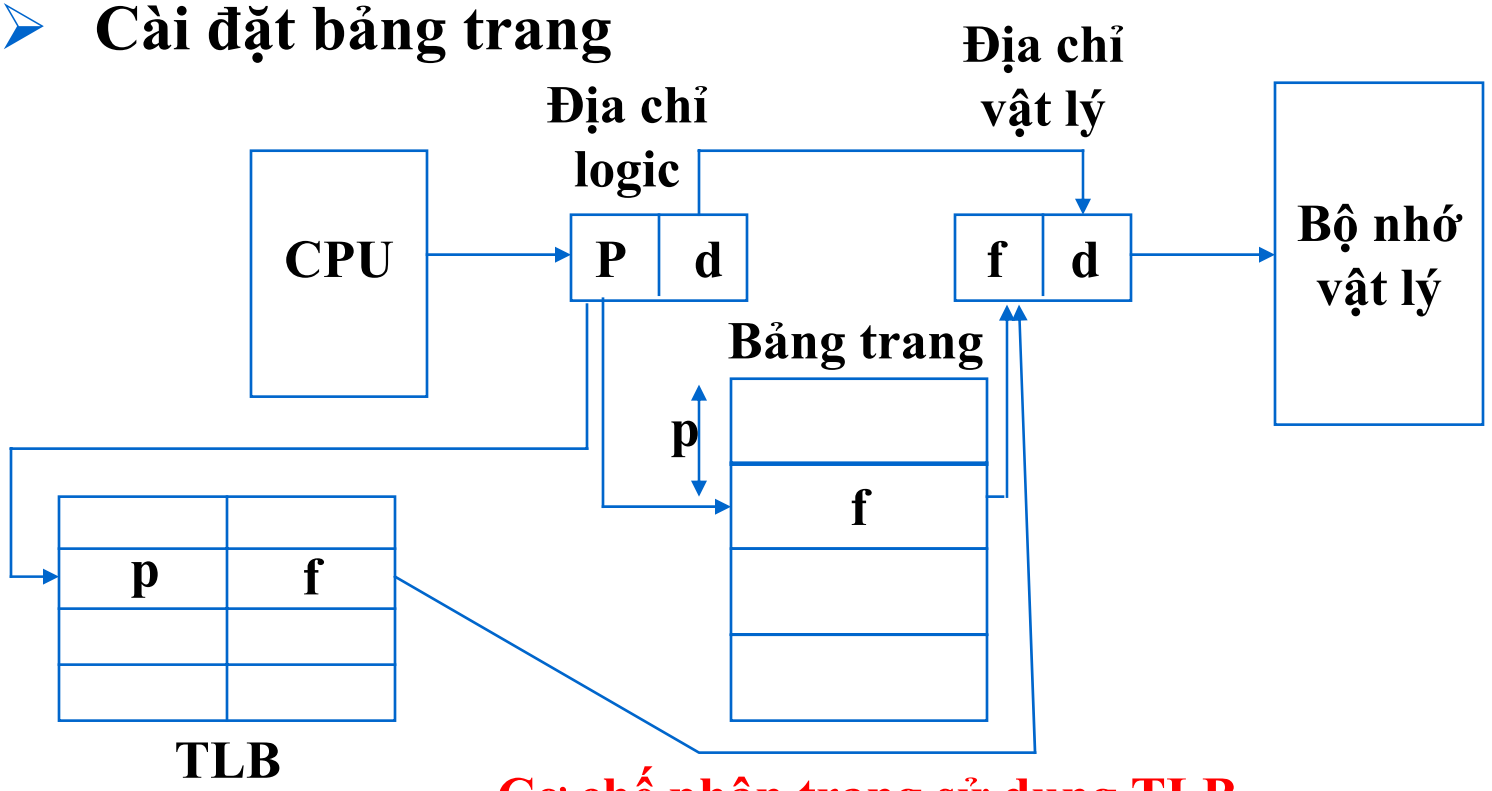
- Phân trang
  - Cài đặt bảng trang



Cơ chế phân trang sử dụng PTBR

Cấp phát không liên tục

- Phân trang
  - Cài đặt bảng trang



Cơ chế phân trang sử dụng TLB

Cấp phát không liên tục

- **Phân trang**
- **Tổ chức bảng trang**

- **Mỗi HĐH có một cách tổ chức bảng trang. Đa số các HĐH cấp cho mỗi tiến trình một bảngtrang**
- **Nếu không gian địa chỉ có dung lượng quá lớn. Bảng trang đòi hỏi một vùng nhớ quá lớn. Có 2 giải pháp:**
  - **Phân trang đa cấp.**

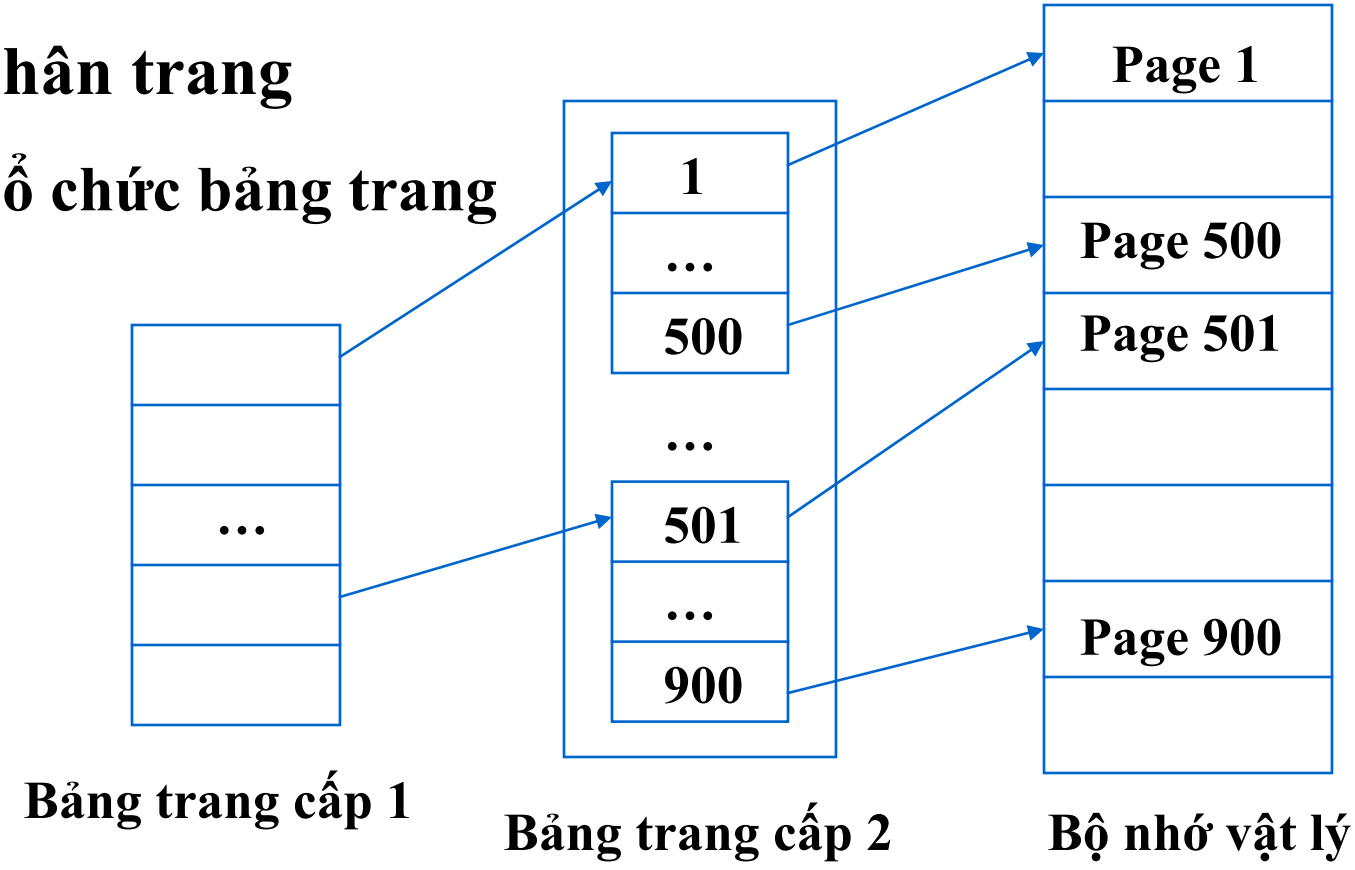
Cấp phát không liên tục

- **Phân trang**
- **Tổ chức bảng trang**
- **Phân trang đa cấp**

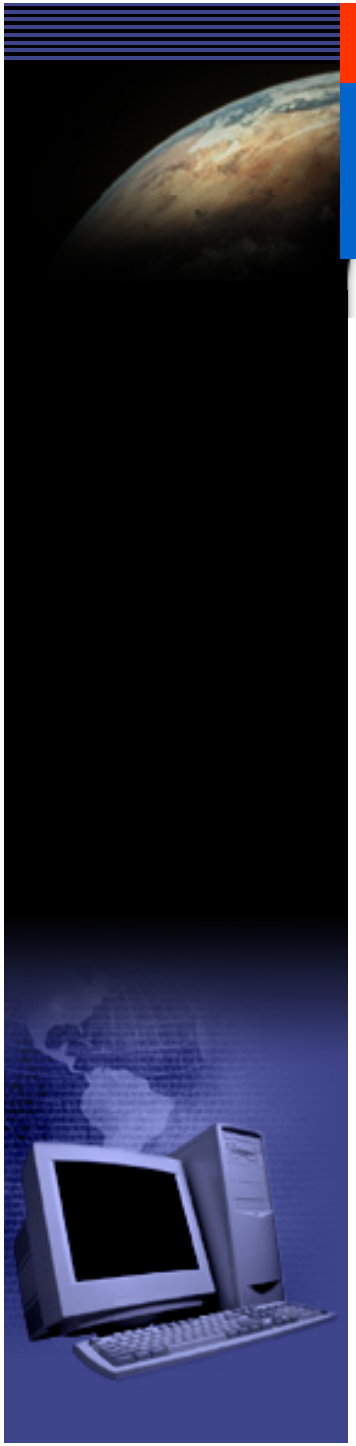
  Phân chia bảng trang thành các phần nhỏ, bản thân bảng trang cũng sẽ được phân trang

Cấp phát không liên tục

- Phân trang
- Tổ chức bảng trang

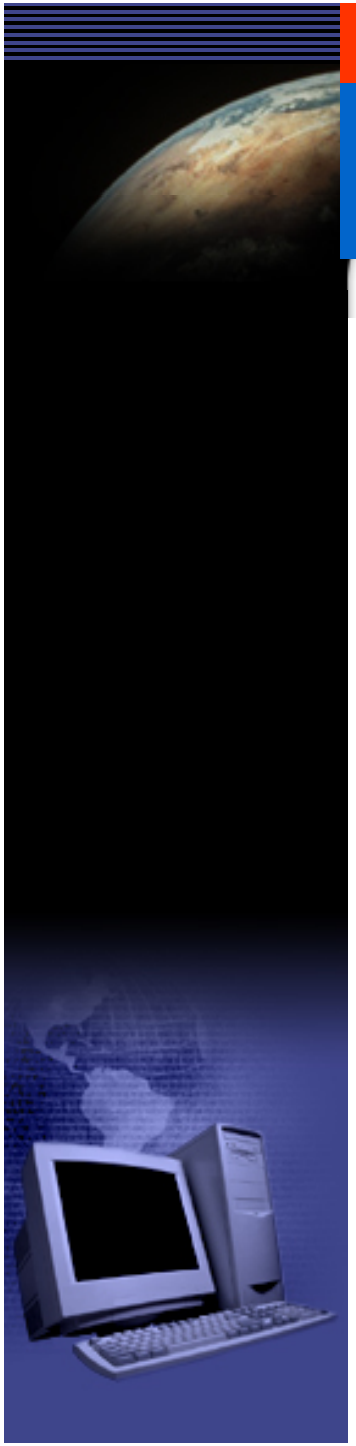


Bảng trang nhị cấp



Cấp phát không liên tục

- **Phân đoạn**
  - **Ý tưởng**
  - **Cơ chế MMU**
  - **Chuyển đổi địa chỉ**
  - **Cài đặt bảng phân đoạn**
  - **Chia sẻ phân đoạn**



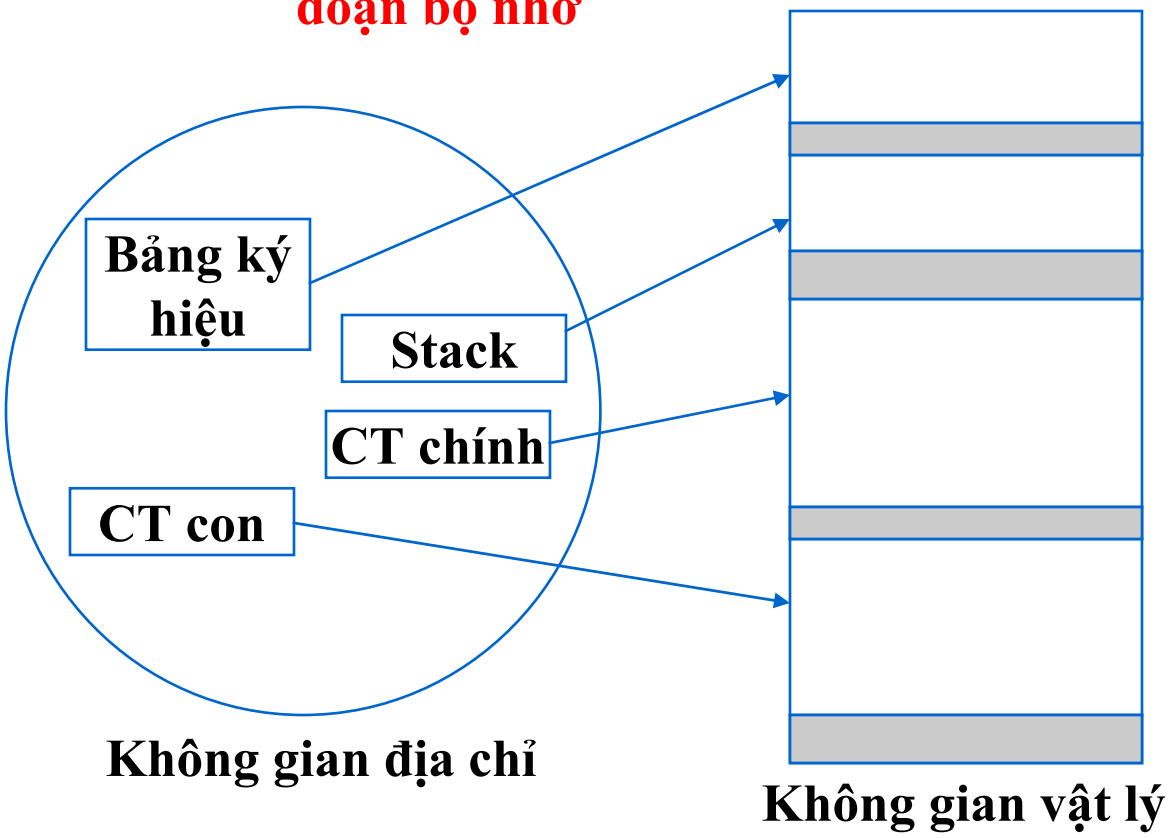
Cấp phát không liên tục

- **Phân đoạn**
  - **Ý tưởng**
    - **Không gian địa chỉ: tập các phân đoạn(segments) có kích thước khác nhau, có liên hệ logic với nhau**
    - **Mỗi phân đoạn: <số hiệu, độ dài>**
    - **Mỗi địa chỉ logic: <số hiệu phân đoạn, offset>**

Cấp phát không liên tục

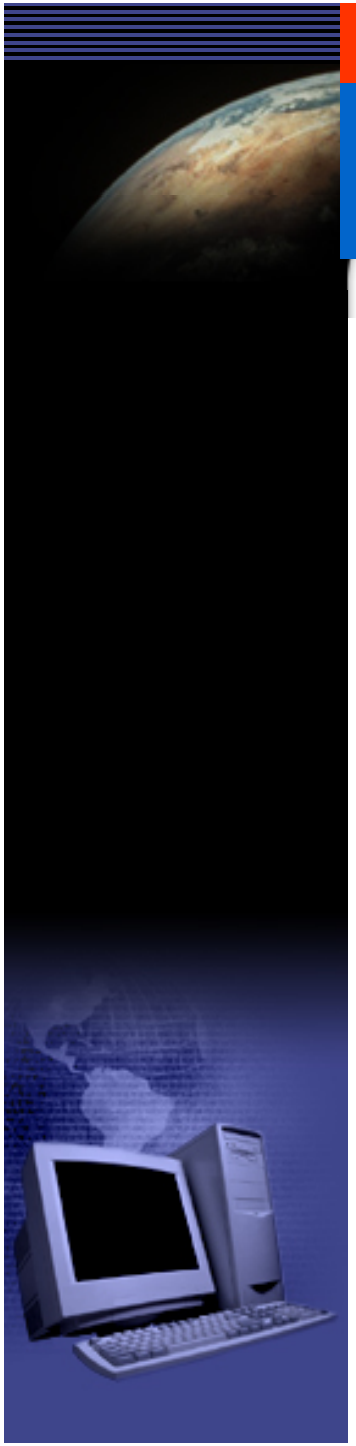
- Phân đoạn
  - Ý tưởng

Mô hình phân đoạn bộ nhớ



Cấp phát không liên tục

- **Phân đoạn**
- **Cơ chế MMU**
- **Sử dụng bảng phân đoạn:**
  - **Thanh ghi nền: địa chỉ vật lý nơi bắt đầu của phân đoạn**
  - **Thanh ghi giới hạn: chiều dài của phân đoạn**

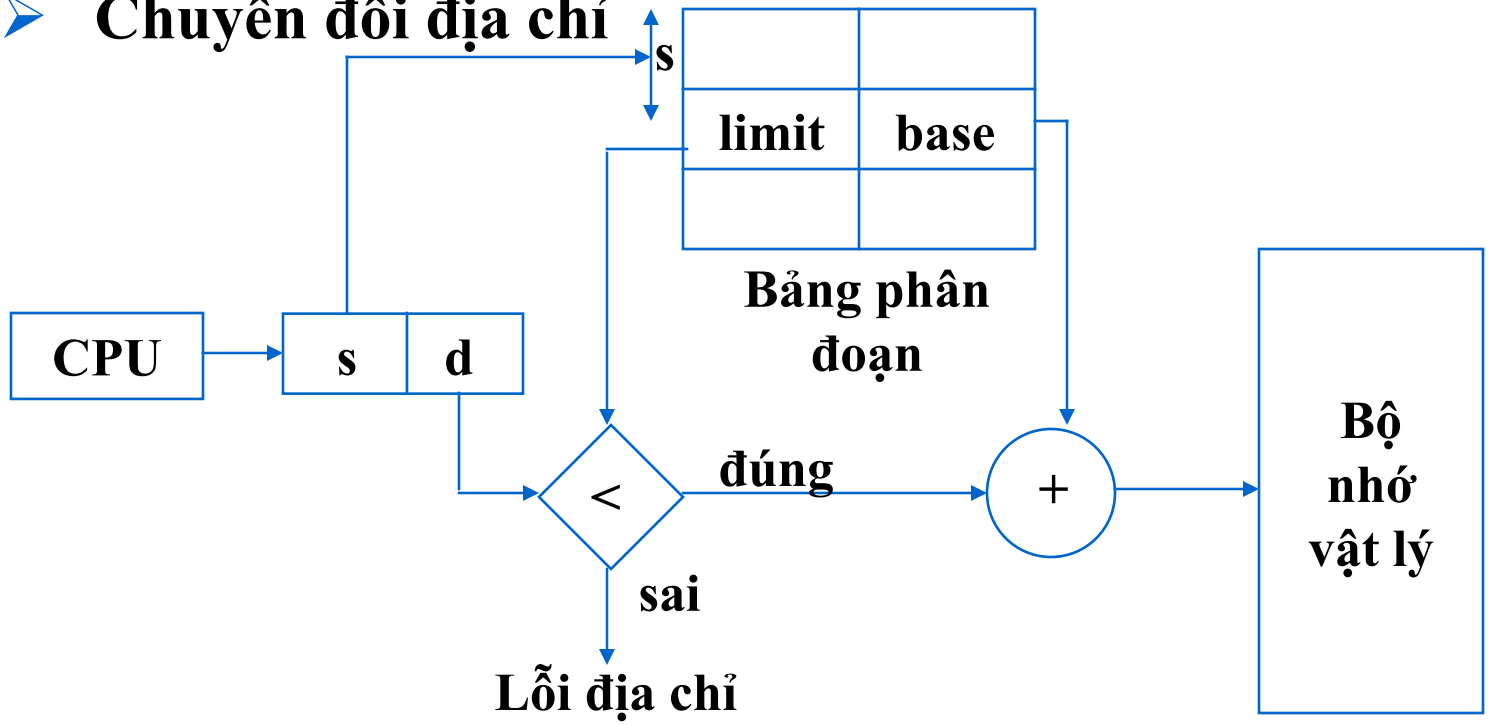


Cấp phát không liên tục

- **Phân đoạn**
  - **Chuyển đổi địa chỉ**
    - **Mỗi địa chỉ logic: <s,d>**
      - **s: số hiệu phân đoạn**
      - **d: địa chỉ tương đối offset, có giá trị từ 0 đến độ dài phân đoạn.**
    - **Địa chỉ vật lý=d+ giá trị chứa trong thanh ghi nền**

Cấp phát không liên tục

- Phân đoạn
  - Chuyển đổi địa chỉ



Cơ chế phần cứng hỗ trợ kỹ thuật phân đoạn

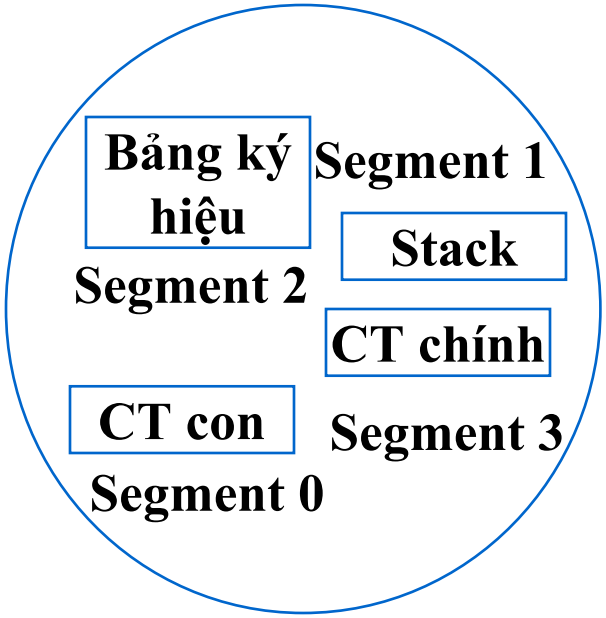
Cấp phát không liên tục

- **Phân đoạn**
  - **Cài đặt bảng phân đoạn**
    - **Sử dụng tập các thanh ghi: bảng phân đoạn có kích thước nhỏ.**
    - **Lưu trữ trong bộ nhớ: bảng phân đoạn có kích thước lớn**
      - **Thanh ghi nền bảng phân đoạn (STBR) để lưu địa chỉ bắt đầu bảng phân đoạn (Segment Table Basic Register)**
      - **Thanh ghi đặc tả kích thước bảng phân đoạn (STLR)**

Cấp phát không liên tục

Phân đoạn

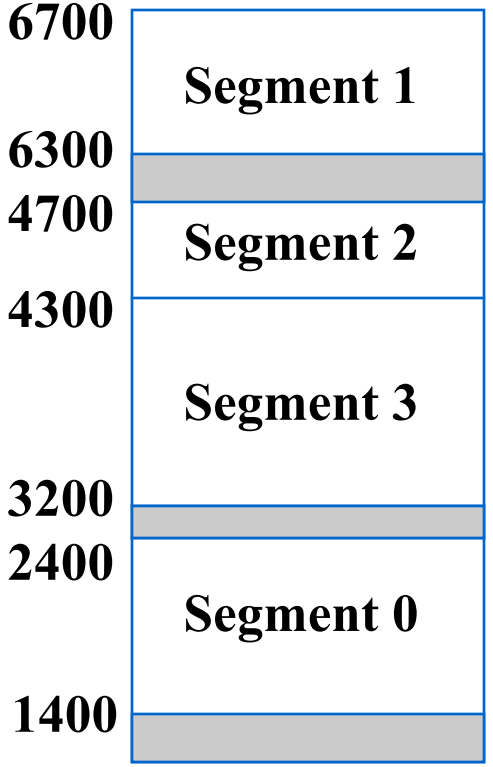
Cài đặt bảng phân đoạn



Không gian địa chỉ

limit	base
1000	1400
400	6300
400	4300
1100	3200

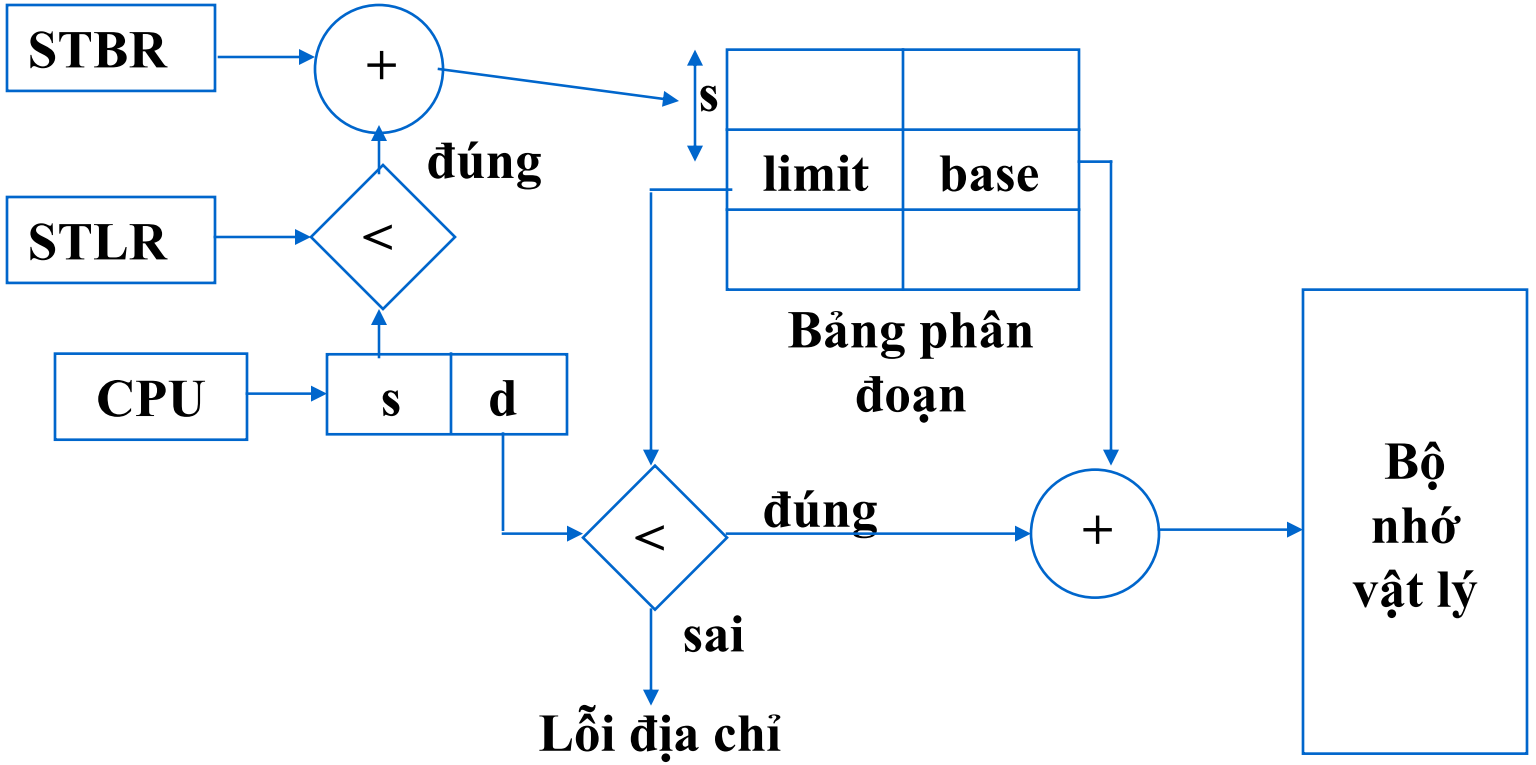
Hệ thống phân đoạn

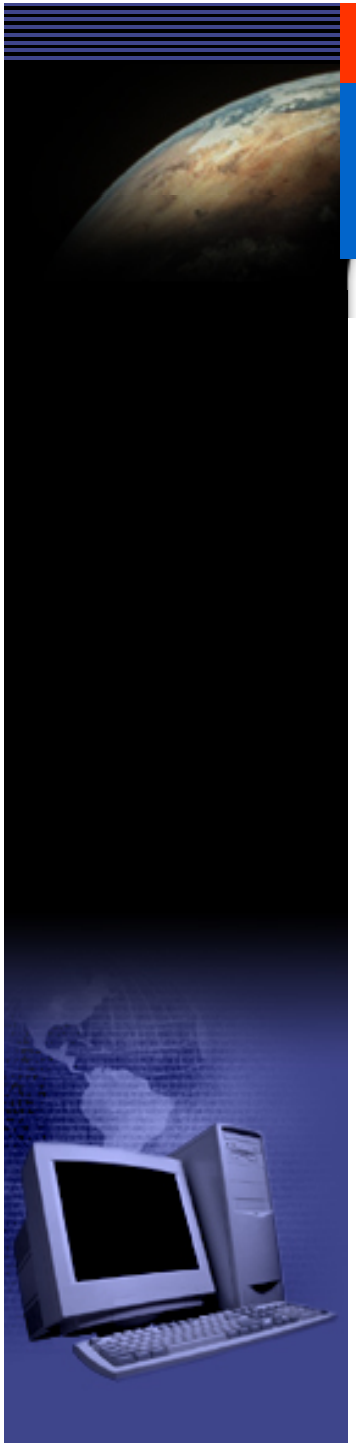


Không gian vật lý

Cấp phát không liên tục

- Phân đoạn
  - Cài đặt bảng phân đoạn



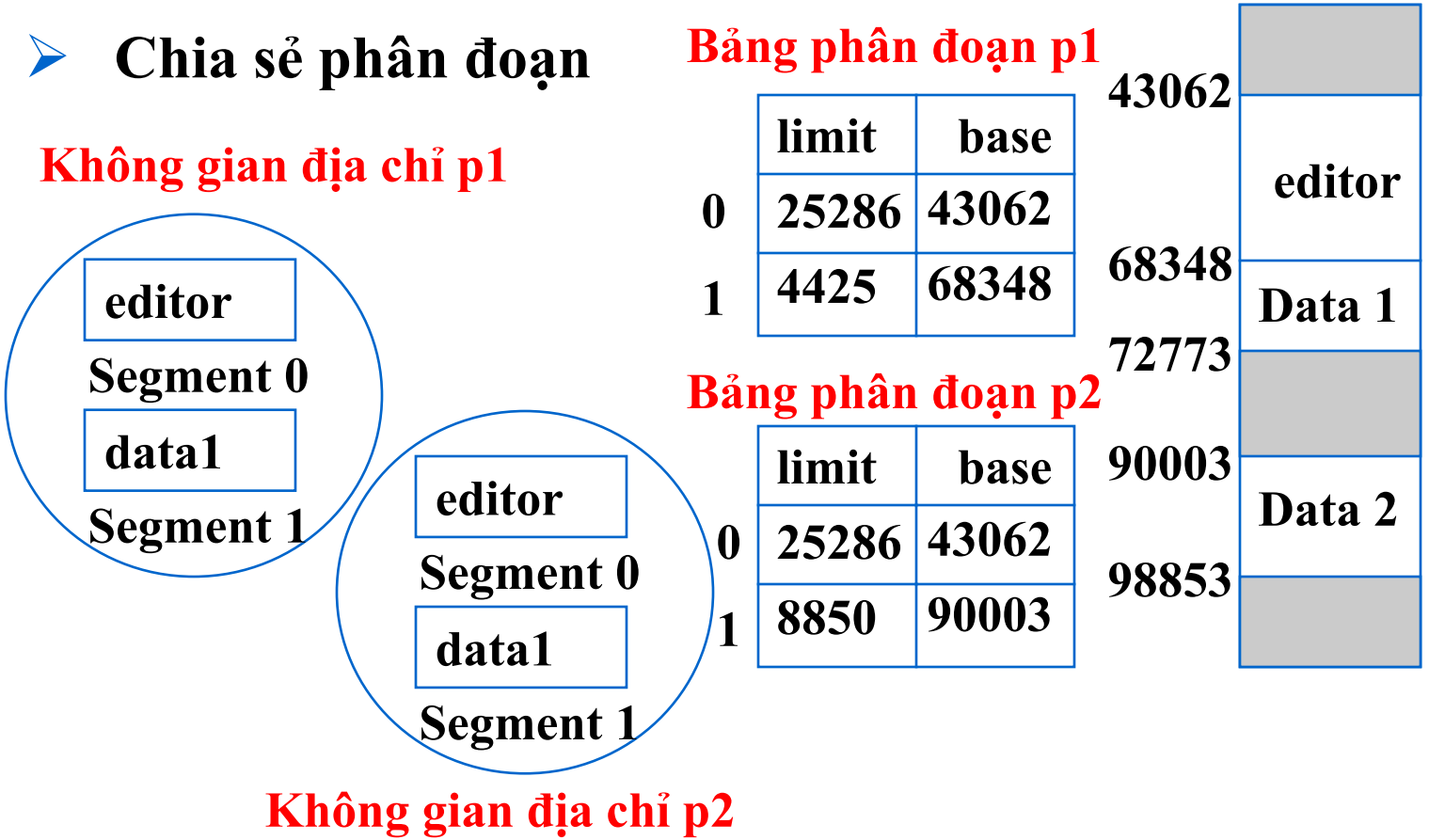


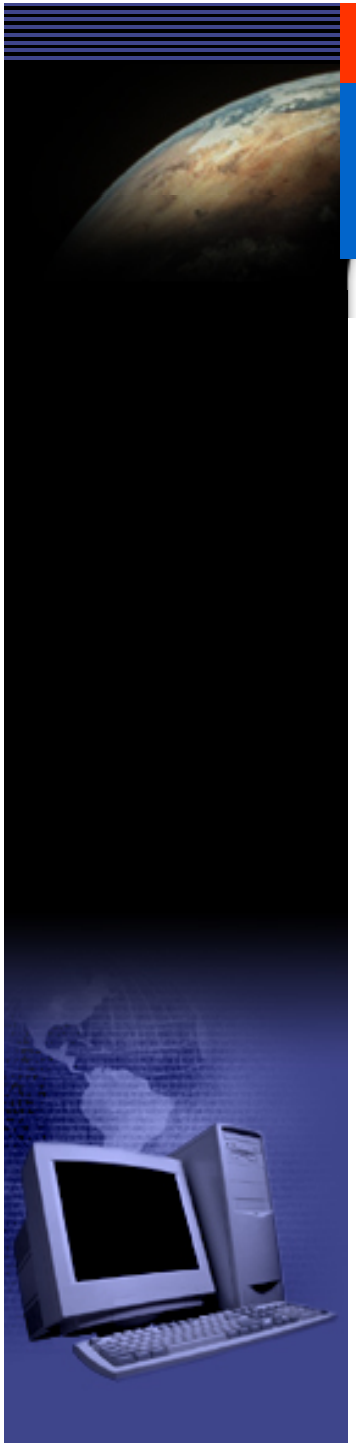
Cấp phát không liên tục

- **Phân đoạn**
- **Chia sẻ phân đoạn**
  - **Khả năng chia xẻ ở mức phân đoạn: chia xẻ các chương trình con.**
  - **Mỗi tiến trình có một bảng phân đoạn riêng.**
  - **Một phân đoạn được chia xẻ khi các phần tử trong bảng phân đoạn của hai tiến trình khác nhau cùng truy xuất đến một địa chỉ vật lý giống nhau**

Cấp phát không liên tục

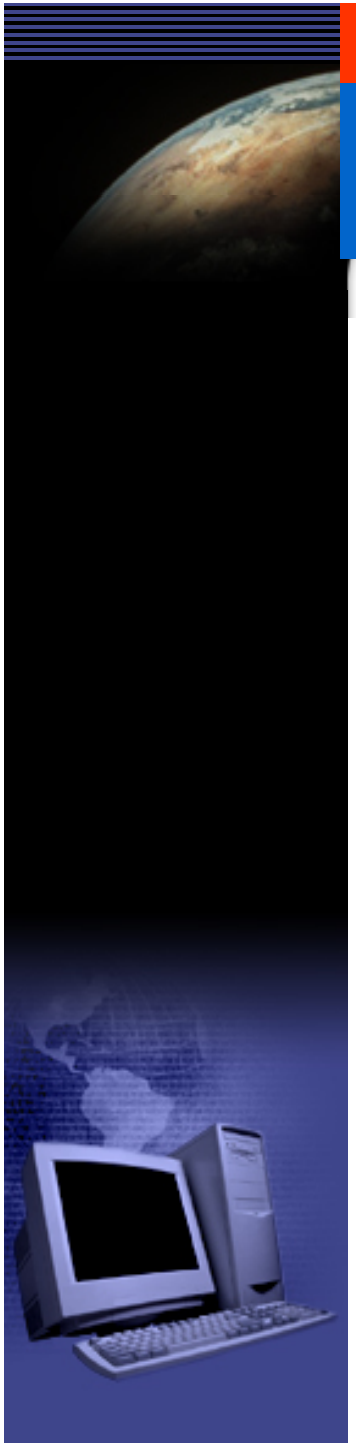
- Phân đoạn
- Chia sẻ phân đoạn





Cấp phát không liên tục

- **Phân đoạn kết hợp phân trang**
  - **Ý tưởng**
  - **Cơ chế MMU**
  - **Chuyển đổi địa chỉ**

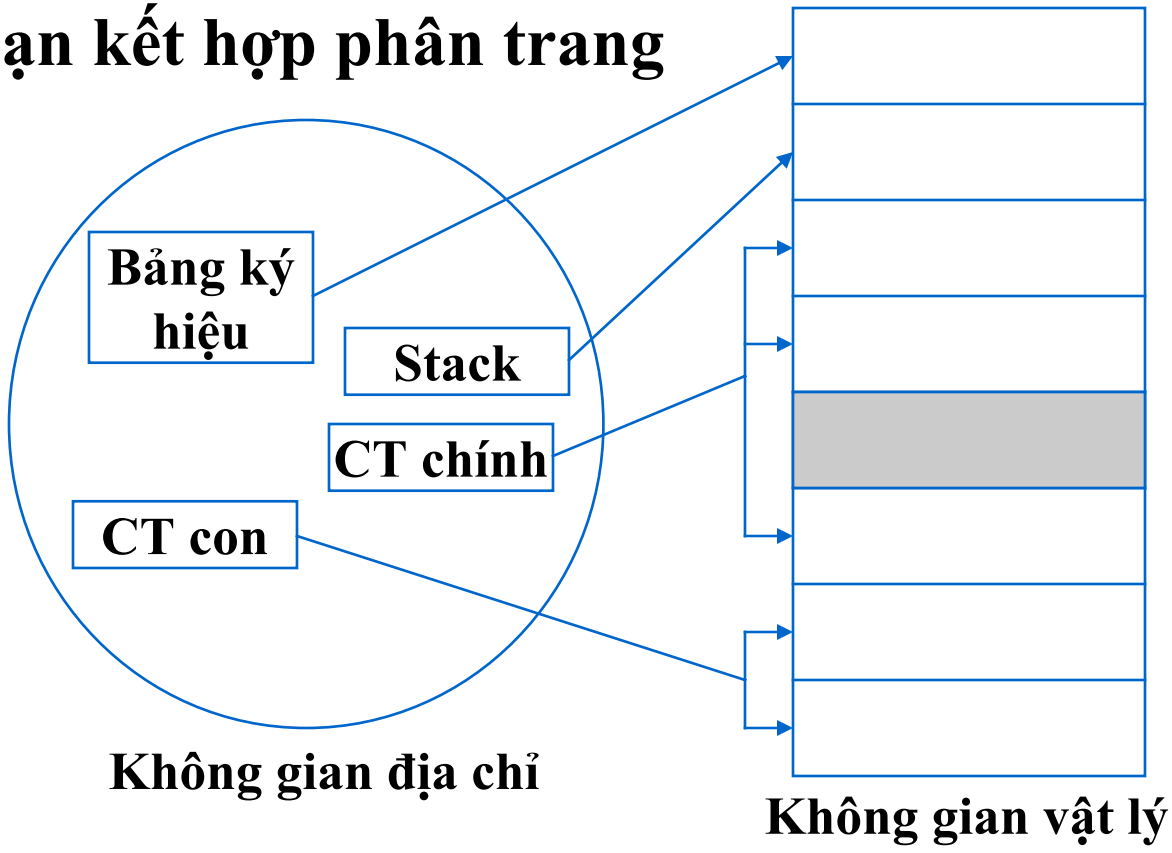


Cấp phát không liên tục

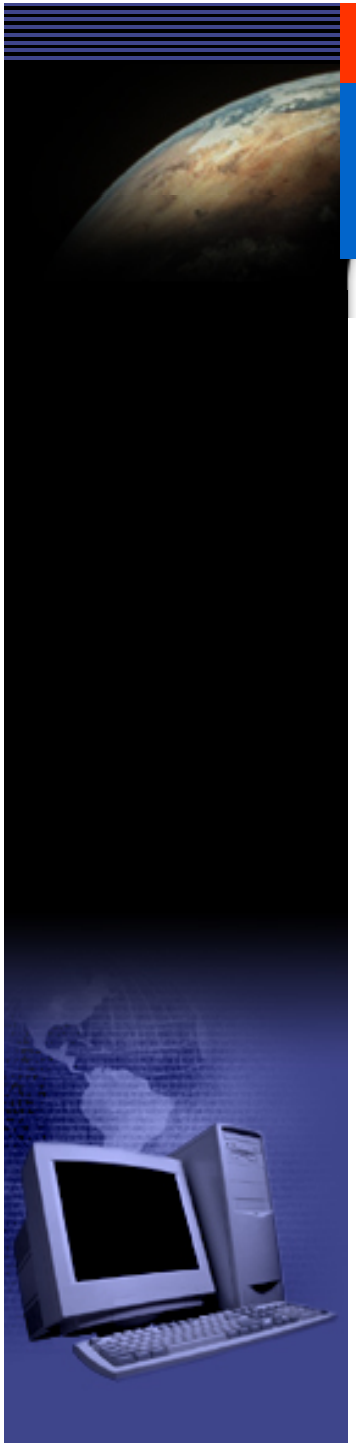
- **Phân đoạn kết hợp phân trang**
  - **Ý tưởng**
    - **Không gian địa chỉ: tập hợp các phân đoạn.**
    - **Mỗi phân đoạn: chia thành nhiều**
    - **Tiến trình được đưa vào hệ thống, HĐH sẽ cấp phát cho tiến trình các trang cần thiết để chứa đủ các phân đoạn của tiến trình**

Cấp phát không liên tục

- Phân đoạn kết hợp phân trang
  - Ý tưởng



Mô hình phân đoạn kết hợp phân trang

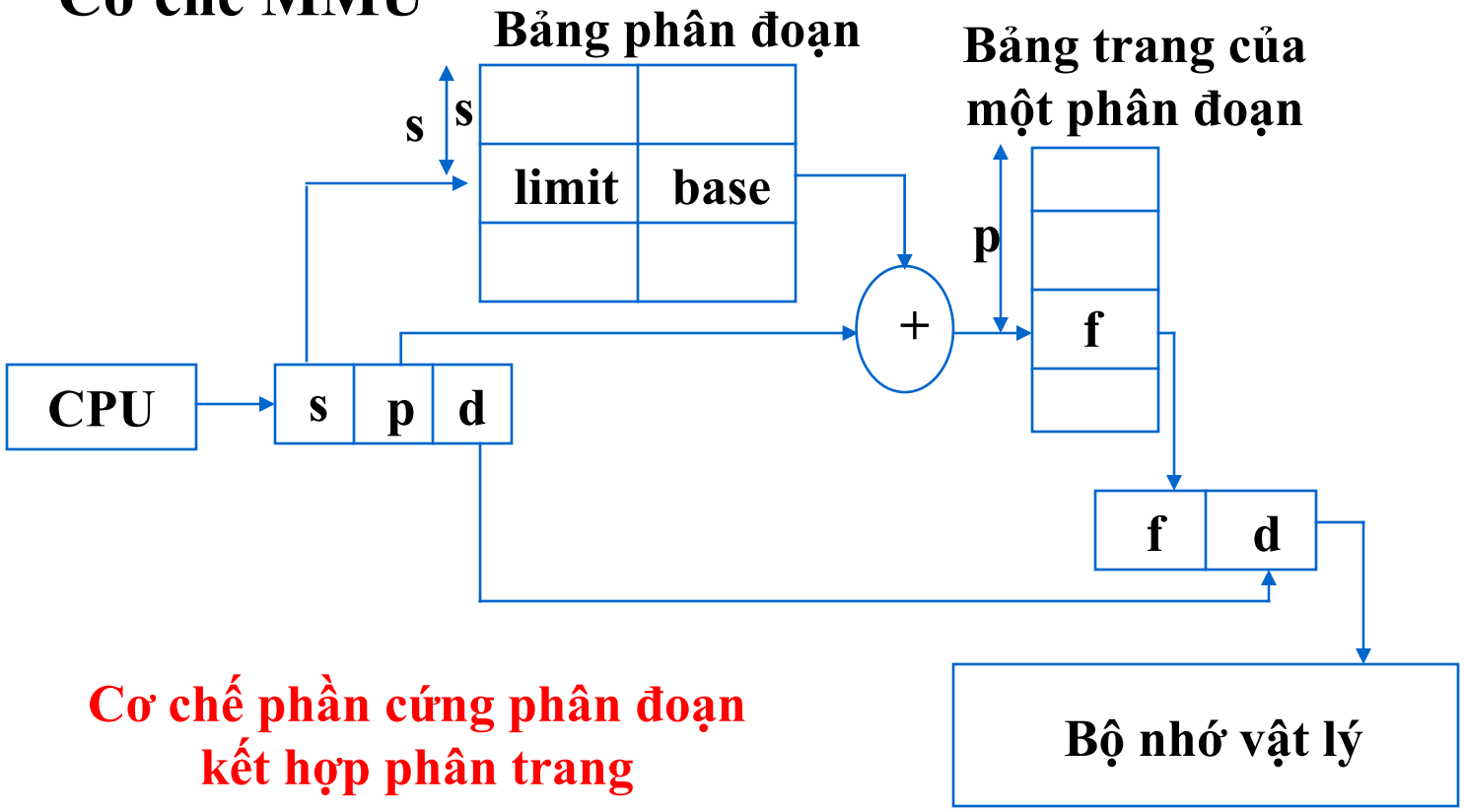


Cấp phát không liên tục

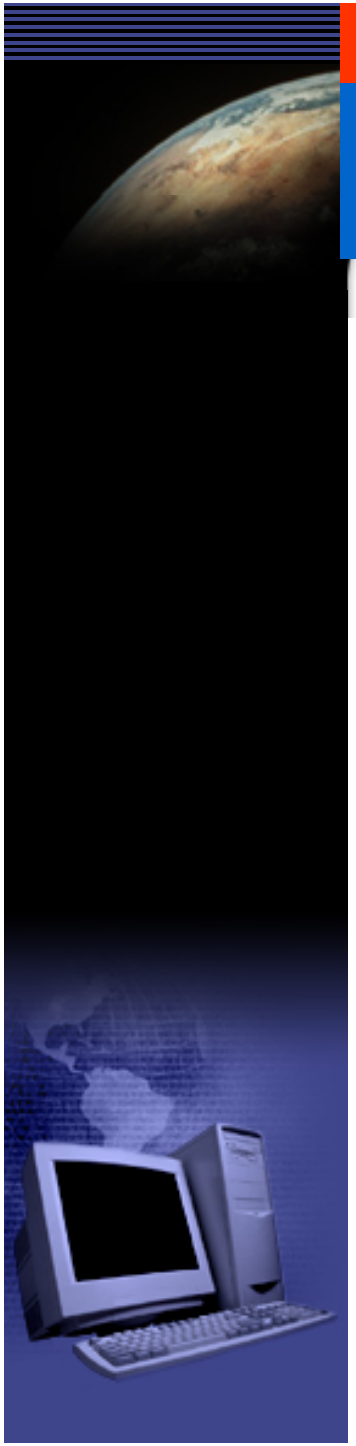
- **Phân đoạn kết hợp phân trang**
- **Chuyển đổi địa chỉ**
  - **Mỗi địa chỉ: <s, p, d>**
    - **S: số hiệu phân đoạn**
    - **P: số hiệu trang**
    - **D: địa chỉ tương đối**

Cấp phát không liên tục

- Phân đoạn kết hợp phân trang
  - Cơ chế MMU

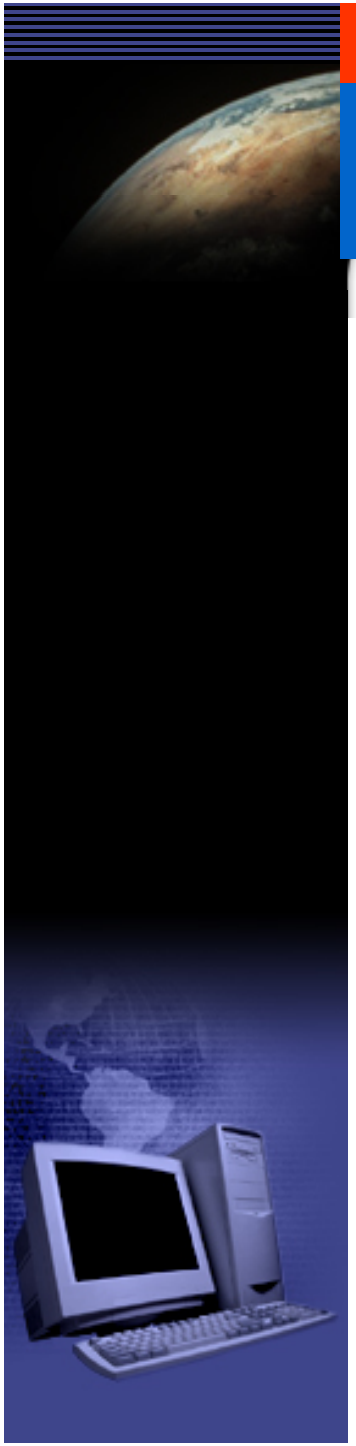


Cơ chế phần cứng phân đoạn kết hợp phân trang



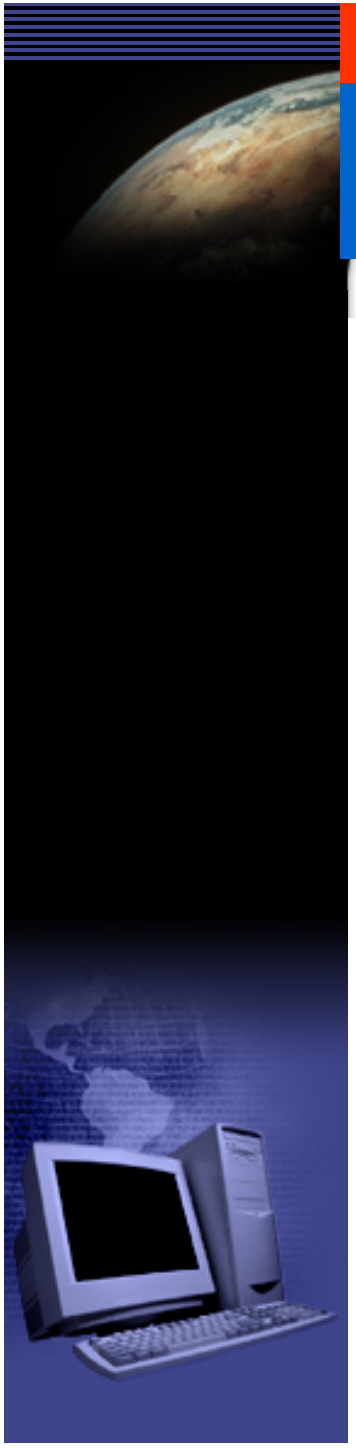
Bộ nhớ ảo

- Nếu đặt toàn thể không gian địa chỉ vào bộ nhớ vật lý thì kích thước của chương trình bị giới hạn bởi kích thước bộ nhớ.
- Nạp từng phần của chương trình.
- Tại một thời điểm, chỉ nạp vào bộ nhớ vật lý các chỉ thị và dữ liệu của ct cần thiết cho việc thi hành lệnh ở thời điểm đó.



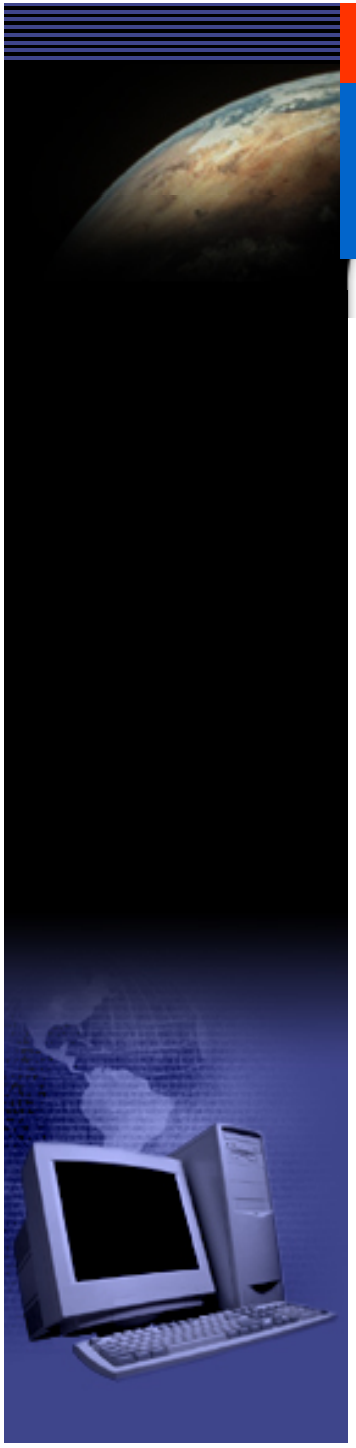
Bộ nhớ ảo

- **Bộ nhớ ảo: kỹ thuật cho phép xử lý một tiến trình k0 được nạp toàn bộ vào bộ nhớ vật lý.**
- **Bộ nhớ ảo: mô hình hoá bộ nhớ như một bảng lưu trữ rất lớn và đông nhất.**
- **NSD làm việc với địa chỉ ảo. Việc chuyển đổi sang địa chỉ vật lý do HĐH đảm nhiệm bằng cơ chế phần cứng**



Mô hình Client-Server

- Hệ thống nguyên khối (Monolithic System)
- Hệ thống phân lớp (Layer System)
- Máy ảo (Virtual Machine)
- Mô hình Client-Server (Client-Server Model)



Mô hình Client-Server

- Hệ thống nguyên khối (Monolithic System)
- Hệ thống phân lớp (Layer System)
- Máy ảo (Virtual Machine)
- Mô hình Client-Server (Client-Server Model)

Mục lục

Chương 1. TỔNG QUAN VỀ HỆ ĐIỀU HÀNH 5
1.1. KHÁI NIỆM VỀ HỆ ĐIỀU HÀNH.....5
1.2. PHÂN LOẠI HỆ ĐIỀU HÀNH.....6
1.2.1. Hệ thống xử lý theo lô 6
1.2.2. Hệ thống xử lý theo lô đa chương 6
1.2.3. Hệ thống chia sẻ thời gian.....7
1.2.4. Hệ thống song song.....7
1.2.5. Hệ thống phân tán.....8
1.2.6. Hệ thống xử lý thời gian thực.....8
Chương 2. LUỒNG VÀ TIẾN TRÌNH.....10
2.1. NHU CẦU XỬ LÝ ĐỒNG THỜI.....10
2.1.1. Tăng hiệu suất sử dụng CPU 10
2.1.2. Tăng tốc độ xử lý 10
2.2. KHÁI NIỆM TIẾN TRÌNH(PROCESS) VÀ MÔ HÌNH ĐA TIẾN TRÌNH (MULTIPROCESS).....10
2.3. KHÁI NIỆM LUỒNG (THREAD) VÀ MÔ HÌNH ĐA LUỒNG (MULTITHREAD).....11
2.3.1. Nguyên lý chung: 12
2.3.2. Phân bổ thông tin lưu trữ.....12
2.3.3. Kernel thread và user thread 13
Chương 3. LẬP LỊCH TIẾN TRÌNH.....14
3.1. Tổ chức quản lý tiến trình 14
3.1.1. Các trạng thái của tiến trình.....14
3.1.2. Chế độ xử lý của tiến trình.....15
3.1.3. Cấu trúc dữ liệu khối quản lý tiến trình.....15
3.1.4. Thao tác trên tiến trình.....16
3.1.4.1. Tạo lập tiến trình.....16
3.1.4.2. Kết thúc tiến trình.....17
3.1.5. Cấp phát tài nguyên cho tiến trình.....17
3.2. Lập lịch tiến trình.....18
3.2.1. Giới thiệu.....19
3.2.1.1. Mục tiêu lập lịch.....19
3.2.1.2. Các đặc điểm của tiến trình.....19
3.2.1.3. Điều phối không độc quyền và điều phối độc quyền (preemptive/nopreemptive).....20
3.2.2.1. Các danh sách sử dụng trong quá trình lập lịch.....21
3.2.2.2. Các cấp độ lập lịch.....22
3.2.3. Các thuật toán lập lịch.....23
3.2.3.1. Chiến lược FIFO.....23
3.2.3.2. Lập lịch xoay vòng (Round Robin).....24
3.2.3.3. Lập lịch với độ ưu tiên.....25
3.2.3.4. Chiến lược công việc ngắn nhất (Shortest-job-first SJF).....26
3.2.3.5. Chiến lược điều phối với nhiều mức độ ưu tiên.....27
3.2.3.6. Chiến lược lập lịch Xổ số (Lottery).....28
Chương 4. TRUYỀN THÔNG VÀ ĐỒNG BỘ TIẾN TRÌNH 29

4.1. LIÊN LẠC TIẾN TRÌNH 29
4.1.1. Nhu cầu liên lạc tiến trình....29
4.1.2. Các vấn đề nảy sinh trong việc liên lạc tiến trình....29
4.2. Các Cơ Chế Thông Tin Liên lạc....30
4.2.1. Tín hiệu (Signal)....30
4.2.2. Pipe....31
4.2.3. Vùng nhớ chia sẻ....32
4.2.4. Trao đổi thông điệp (Message)....32
4.2.5. Sockets....33
4.3. Nhu cầu đồng bộ hóa (synchronisation)....34
4.3.1. Yêu cầu độc quyền truy xuất (Mutual exclusion) 34
4.3.2. Yêu cầu phối hợp (Synchronization)....34
4.3.3. Bài toán đồng bộ hoá 35
4.3.3.1. Vấn đề tranh đoạt điều khiển (race condition)....35
4.3.3.2. Miền găng (critical section)....35
4.4. CÁC GIẢI PHÁP ĐỒNG BỘ HOÁ....37
4.4.1. Giải pháp « busy waiting »....37
4.4.1.1. Sử dụng các biến cờ hiệu(simaphore)....37
4.4.1.2. Sử dụng việc kiểm tra luân phiên 37
4.4.1.3. Giải pháp của Peterson 38
4.4.1.4. Cấm ngắt: 38
4.4.1.5. Chỉ thị TSL (Test-and-Set) 39
4.4.2. Các giải pháp « SLEEP and WAKEUP »....39
4.4.2.1. Semaphore....41
4.4.2.2. Monitors....42
4.4.2.3. Trao đổi thông điệp....44
Chương 5. VẤN ĐỀ KHOÁ CHẾT (DEADLOCK)....46
5.1. Mô hình hệ thống 46
5.2. Đặc điểm deadlock 47
5.2.1. Những điều kiện cần thiết gây ra deadlock 47
5.2.2. Đồ thị cấp phát tài nguyên 47
5.3. Các phương pháp xử lý deadlock 50
5.4. Ngăn chặn deadlock 51
5.4.1. Loại trừ hỗ tương 51
5.4.2. Giữ và chờ cấp thêm tài nguyên 51
5.4.3. Không đòi lại tài nguyên từ quá trình đang giữ chúng 52
5.4.4. Tồn tại chu trình trong đồ thị cấp phát tài nguyên 53
5.5. Tránh deadlock 53
5.5.1. Trạng thái an toàn 54
5.5.2. Giải thuật đồ thị cấp phát tài nguyên 54
5.5.3. Giải thuật của Banker 56
5.5.3.1. Giải thuật an toàn 57
5.5.3.2. Giải thuật yêu cầu tài nguyên 58
Chương 6: QUẢN LÝ BỘ NHỚ TRONG....59
6.1. Ngữ cảnh liên kết bộ nhớ....59
6.2. Không gian địa chỉ lôgic và không gian địa chỉ vật lý....60

6.3. Cấp phát liên tục....60
6.3.1. Mô hình Linker_Loader....60
6.3.2. Mô hình Base &Bound....61
6.4. Cấp phát không liên tục....63
6.4.1. Phân đoạn (Segmentation)....63
6.4.2. Phân trang (Paging)....66
6.4.3. Phân đoạn kết hợp phân trang (Paged segmentation)....72
6.5. Bộ nhớ ảo....74
6.5.1. Cơ chế bộ nhớ ảo....74
6.5.1.1. Định nghĩa....74
6.5.1.2. Cài đặt bộ nhớ ảo....75
6.5.2. Thay thế trang....77
6.5.2.1. Sự thi hành phân trang theo yêu cầu....77
6.5.2.2. Các thuật toán thay thế trang....78
Chương 7. HỆ THỐNG QUẢN LÝ TẬP TIN....83
7.1. CÁC KHÁI NIỆM CƠ BẢN....83
7.1.1. Bộ nhớ ngoài....83
7.1.2. Tập tin và thư mục....83
7.1.3. Hệ thống quản lý tập tin....83
7.2. MÔ HÌNH TỔ CHỨC VÀ QUẢN LÝ CÁC TẬP TIN....84
7.2.1. Mô hình....84
7.2.1.1. Tập tin....84
7.2.1.2. Thư mục:....87
7.2.2. Các chức năng....89
7.2.2.1. Tập tin....89
7.2.2.2. Thư mục....89
7.3. CÁC PHƯƠNG PHÁP CÀI ĐẶT HỆ THỐNG QUẢN LÝ TẬP TIN....91
7.3.1. Bảng quản lý tệp tin, thư mục....91
7.3.1.1. Khái niệm....91
7.3.1.2. Cài đặt....91
7.3.2. Bảng phân phối vùng nhớ....92
7.3.2.1. Khái niệm....92
7.3.2.2. Các phương pháp....92
7.3.3. Tệp tin chia sẻ....94
7.3.4. Độ an toàn của hệ thống quản lý tệp tin....95
7.3.4.1. Quản lý khối bị hỏng....95
7.3.4.2. Sao lưu....96
7.3.4.3. Tính không đổi của hệ thống tệp tin....96
Chương 8. HỆ THỐNG QUẢN LÝ NHẬP/XUẤT....98
8.1. KHÁI NIỆM VỀ HỆ THỐNG QUẢN LÝ NHẬP/XUẤT....98
8.2. PHẦN CỨNG NHẬP/XUẤT....99
8.2.1. Thiết bị I/O....99
8.2.2. Tổ chức của chức năng I/O....99
8.2.3. Bộ điều khiển thiết bị....100
8.2.4. DMA (Direct Memory Access)....101
8.3. PHẦN MỀM NHẬP/XUẤT....102

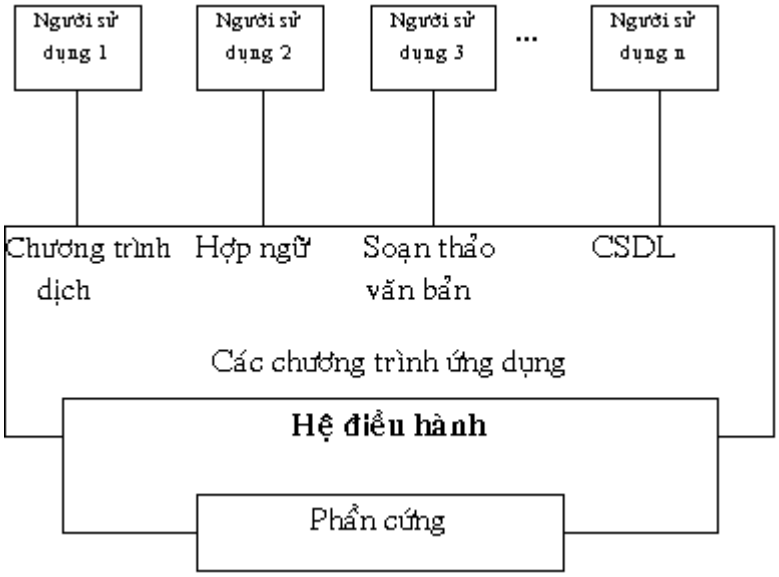
8.3.1. Kiểm soát ngắt 102
8.3.2. Điều khiển thiết bị (device drivers) 103
8.3.3. Phần mềm nhập/xuất độc lập thiết bị 103
8.3.4. Phần mềm nhập/xuất phạm vi người sử dụng 104
Chương 9. GIỚI THIỆU MỘT SỐ HỆ THỐNG I/O....105
9.1. HỆ THỐNG I/O ĐĨA....105
9.1.1. Phần cứng đĩa....105
9.1.2. Các thuật toán đọc đĩa 105
9.1.2.1. Lập lịch FCFS....106
9.1.2.2. Lập lịch SSTF (shortest-seek-time-first)....106
9.1.2.3. Lập lịch SCAN....106
9.1.2.4. Lập lịch C-SCAN....107
9.1.2.5. Lập lịch LOOK....107
9.1.2.6. Lựa chọn thuật toán lập lịch:....108
9.1.3. Quản lý lỗi....108
9.1.4. RAM Disks....108
9.1.5. Interleave 109
9.2. HỆ THỐNG I/O CHUẨN (terminals)....110
9.2.1. Phần cứng terminal....110
9.2.2. Terminal ánh xạ bộ nhớ....111
9.2.3. Phần mềm nhập....112
9.2.4. Phần mềm xuất....113
9.3. CÀI ĐẶT ĐỒNG HỒ....114
9.3.1. Phần cứng đồng hồ....114
9.3.2. Phần mềm đồng hồ 115
Chương 10. BẢO VỆ VÀ AN TOÀN HỆ THỐNG....117
10.1. Mục tiêu bảo vệ hệ thống (Protection)....117
10.2. Miền bảo vệ (Domain of Protection)....117
10.2.1. Khái niệm....117
10.2.2. Cấu trúc của miền bảo vệ 118
10.3. Ma trận quyền truy xuất (Access matrix)....119

CHƯƠNG 1. TỔNG QUAN VỀ HỆ ĐIỀU HÀNH

1.1. KHÁI NIỆM VỀ HỆ ĐIỀU HÀNH

Hệ điều hành là một *chương trình* hay một *hệ chương trình* hoạt động giữa người sử dụng (user) và phần cứng của máy tính. Mục tiêu của hệ điều hành là cung cấp một môi trường để người sử dụng có thể thi hành các chương trình. Nó làm cho máy tính dể sử dụng hơn, thuận lợi hơn và hiệu quả hơn.
Hệ điều hành là một phần quan trọng của hầu hết các hệ thống máy tính. Một hệ thống máy tính thường được chia làm bốn phần chính : phần cứng, hệ điều hành, các chương trình ứng dụng và người sử dụng.

Phần cứng bao gồm CPU, bộ nhớ, các thiết bị nhập xuất, đây là những tài nguyên của máy tính. ***Chương trình ứng dụng*** như các chương trình dịch, hệ thống cơ sở dữ liệu, các trò chơi, và các chương trình thương mại. Các chương trình này sử dụng tài nguyên của máy tính để giải quyết các yêu cầu của người sử dụng. ***Hệ điều hành*** điều khiển và phối hợp việc sử dụng phần cứng cho những ứng dụng khác nhau của nhiều người sử dụng khác nhau. Hệ điều hành cung cấp một môi trường mà các chương trình có thể làm việc hữu hiệu trên đó.



Hình 1.1 Mô hình trừu tượng của hệ thống máy tính

Hệ điều hành có thể được coi như là bộ phân phối tài nguyên của máy tính. Nhiều tài nguyên của máy tính như thời gian sử dụng CPU, vùng bộ nhớ, vùng lưu trữ tập tin, thiết bị nhập xuất v.v… được các chương trình yêu cầu để giải quyết vấn đề. Hệ điều hành hoạt động như một bộ quản lý các tài nguyên và phân phối chúng cho các chương trình và người sử dụng khi cần thiết. Do có rất nhiều yêu cầu, hệ điều hành phải giải quyết vấn đề tranh chấp và phải quyết định ***cấp phát tài nguyên*** cho những yêu cầu theo

thứ tự nào để hoạt động của máy tính là hiệu quả nhất. Một hệ điều hành cũng có thể được coi như là một chương trình kiểm soát việc sử dụng máy tính, đặc biệt là các thiết bị nhập xuất.

Tuy nhiên, nhìn chung chưa có định nghĩa nào là hoàn hảo về hệ điều hành. Hệ điều hành tồn tại để giải quyết các vấn đề sử dụng hệ thống máy tính. Mục tiêu cơ bản của nó là giúp cho việc thi hành các chương trình dễ dàng hơn. Mục tiêu thứ hai là hỗ trợ cho các thao tác trên hệ thống máy tính hiệu quả hơn. Mục tiêu này đặc biệt quan trọng trong những hệ thống nhiều người dùng và trong những hệ thống lớn(phần cứng + quy mô sử dụng). Tuy nhiên hai mục tiêu này cũng có phần tương phản vì vậy lý thuyết về hệ điều hành tập trung vào việc tối ưu hóa việc sử dụng tài nguyên của máy tính.

1.2. PHÂN LOẠI HỆ ĐIỀU HÀNH

1.2.1. Hệ thống xử lý theo lô

- **Bộ giám sát thường trực:**
  Khi một công việc chấm dứt, hệ thống sẽ thực hiện công việc kế tiếp mà không cần sự can thiệp của người lập trình, do đó thời gian thực hiện sẽ mau hơn. Một chương trình, còn gọi là bộ giám sát thường trực được thiết kế để giám sát việc thực hiện dãy các công việc một cách tự động, chương trình này luôn luôn thường trú trong bộ nhớ chính.
  Hệ điều hành theo lô thực hiện các công việc lần lượt theo những chỉ thị định trước.
- **CPU và thao tác nhập xuất:**
  CPU thường hay nhàn rỗi do tốc độ làm việc của các thiết bị nhập xuất (thường là thiết bị cơ) chậm hơn rất nhiều lần so với các thiết bị điện tử. Cho dù là một CPU chậm nhất, nó cũng nhanh hơn rất nhiều lần so với thiết bị nhập xuất. Do đó phải có các phương pháp để đồng bộ hóa việc hoạt động của CPU và thao tác nhập xuất.
- **Xử lý off_line:**
  Xử lý off_line là thay vì CPU phải đọc trực tiếp từ thiết bị nhập và xuất ra thiết bị xuất, hệ thống dùng một *bộ lưu trữ trung gian*. CPU chỉ thao thác với bộ phận này. Việc đọc hay xuất đều đến và từ bộ lưu trữ trung gian.
- **Spooling:**
  Spool (**simultaneous-đồng thời peripheral operation on-line**) là đồng bộ hóa các thao tác bên ngoài on-line. Cơ chế này cho phép xử lý của CPU là on-line, sử dụng đĩa để lưu các dữ liệu nhập cũng như xuất.

1.2.2. Hệ thống xử lý theo lô đa chương

Khi có nhiều công việc cùng truy xuất lên thiết bị, vấn đề lập lịch cho các công việc là cần thiết. Khía cạnh quan trọng nhất trong việc lập lịch là khả năng đa chương. *Đa chương* (multiprogram) gia tăng khai thác CPU bằng cách tổ chức các công việc sao cho CPU luôn luôn phải trong tình trạng làm việc .

Ý tưởng: hệ điều hành lưu giữ một phần của các công việc ở nơi lưu trữ trong bộ nhớ . CPU sẽ lần lượt thực hiện các phần công việc này. Khi đang thực hiện, nếu có yêu cầu truy xuất thiết bị thì CPU không nghỉ mà thực hiện tiếp công việc thứ hai…
Với hệ đa chương hệ điều hành ra quyết định cho người sử dụng vì vậy, ***hệ điều hành đa chương*** rất tinh vi. Hệ phải xử lý các vấn đề lập lịch cho công việc, lập lịch cho bộ nhớ và cho cả CPU nữa.

1.2.3. Hệ thống chia sẻ thời gian

Hệ thống chia sẻ thời gian là một mở rộng logic của hệ đa chương. Hệ thống này còn được gọi là ***hệ thống đa nhiệm*** (multitasking). Nhiều công việc cùng được thực hiện thông qua cơ chế chuyển đổi của CPU như hệ đa chương nhưng thời gian mỗi lần chuyển đổi diễn ra rất nhanh.

Hệ thống chia sẻ được phát triển để cung cấp việc sử dụng bên trong của một máy tính có giá trị hơn. ***Hệ điều hành chia sẻ*** thời gian dùng lập lịch CPU và đa chương để cung cấp cho mỗi người sử dụng một phần nhỏ trong máy tính chia sẻ. Một chương trình khi thi hành được gọi là một tiến trình. Trong quá trình thi hành của một tiến trình, nó phải thực hiện các thao tác nhập xuất và trong khoảng thời gian đó CPU sẽ thi hành một tiến trình khác. Hệ điều hành chia sẻ cho phép nhiều người sử dụng chia sẻ máy tính một cách đồng bộ do thời gian chuyển đổi nhanh nên họ có cảm giác là các tiến trình đang được thi hành cùng lúc.

Hệ điều hành chia sẻ phức tạp hơn hệ điều hành đa chương. Nó phải có các chức năng : quản trị và bảo vệ bộ nhớ, sử dụng bộ nhớ ảo. Nó cũng cung cấp hệ thống tập tin truy xuất on-line…

Hệ điều hành chia sẻ là kiểu của các hệ điều hành hiện đại ngày nay.

1.2.4. Hệ thống song song

Ngoài các hệ thống chỉ có một bộ xử lý còn có các hệ thống có nhiều bộ xử lý cùng chia sẻ hệ thống đường truyền dữ liệu, đồng hồ, bộ nhớ và các thiết bị ngoại vi. Các bộ xử lý này liên lạc bên trong với nhau .

Có nhiều nguyên nhân xây dựng dạng hệ thống này. Với sự gia tăng số lượng bộ xử lý, công việc được thực hiện nhanh chóng hơn, Nhưng không phải theo đúng tỉ lệ thời gian, nghĩa là có n bộ xử lý không có nghĩa là sẽ thực hiện nhanh hơn n lần.
Hệ thống với máy nhiều bộ xử lý sẽ tối ưu hơn hệ thống có nhiều máy có một bộ xử lý vì các bộ xử lý chia sẻ các thiết bị ngoại vi, hệ thống lưu trữ, nguồn … và rất thuận tiện cho nhiều chương trình cùng làm việc trên cùng một tập hợp dữ liệu.
Một lý do nữa là độ tin cậy. Các chức năng được xử lý trên nhiều bộ xử lý và sự hỏng hóc của một bộ xử lý sẽ không ảnh hưởng đến toàn bộ hệ thống.

Hệ thống đa xử lý thông thường sử dụng cách ***đa xử lý đối xứng***, trong cách này mỗi bộ xử lý chạy với một bản sao của hệ điều hành, những bản sao này liên lạc với nhau khi cần thiết. Một số hệ thống sử dụng đa xử lý bất đối xứng, trong đó mỗi bộ xử lý được giao một công việc riêng biệt.. Một bộ xử lý chính kiểm soát toàn bộ hệ thống, các bộ xử lý khác thực hiện theo lệnh của bộ xử lý chính hoặc theo những chỉ thị đã được định nghĩa trước. Mô hình này theo dạng quan hệ chủ tớ. Bộ xử lý chính sẽ lập lịch cho các bộ xử lý khác.

Một ví dụ về hệ thống xử lý đối xứng là version Encore của UNIX cho máy tính Multimax. Hệ thống này có hàng tá bộ xử lý. Ưu điểm của nó là nhiều tiến trình có thể thực hiện cùng lúc . Một hệ thống đa xử lý cho phép nhiều công việc và tài nguyên được chia sẻ tự động trong những bộ xử lý khác nhau.

Hệ thống đa xử lý không đồng bộ thường xuất hiện trong những hệ thống lớn, trong đó hầu hết thời gian hoạt động đều dành cho xử lý nhập xuất.

1.2.5. Hệ thống phân tán

Hệ thống này cũng tương tự như hệ thống chia sẻ thời gian nhưng các bộ xử lý không chia sẻ bộ nhớ và đồng hồ, thay vào đó mỗi bộ xử lý có bộ nhớ cục bộ riêng. Các bộ xử lý thông tin với nhau thông qua các đường truyền thông như những bus tốc độ cao hay đường dây điện thoại.

Các bộ xử lý trong hệ phân tán thường khác nhau về kích thước và chức năng. Nó có thể bao gồm máy vi tính, trạm làm việc, máy mini, và những hệ thống máy lớn. Các bộ xử lý thường được tham khảo với nhiều tên khác nhau như site, node, computer v.v.... tùy thuộc vào trạng thái làm việc của chúng.

Các nguyên nhân phải xây dựng hệ thống phân tán là:

- ***chia sẻ tài nguyên*** : Một người sử dụng A có thể sử dụng máy in laser của người sử dụng B và người sử dụng B có thể truy xuất những tập tin của A. Tổng quát, chia sẻ tài nguyên trong hệ thống phân tán cung cấp một cơ chế để chia sẻ tập tin ở vị trí xa, xử lý thông tin trong một cơ sở dữ liệu phân tán, in ấn tại một vị trí xa, sử dụng những thiết bị ở xa đểõ thực hiện các thao tác.
- ***Tăng tốc độ tính toán*** : Một thao tác tính toán được chia làm nhiều phần nhỏ cùng thực hiện một lúc. Hệ thống phân tán cho phép phân chia việc tính toán trên nhiều vị trí khác nhau để tính toán song song.
- ***An toàn*** : Nếu một vị trí trong hệ thống phân tán bị hỏng, các vị trí khác vẫn tiếp tục làm việc.
- ***Thông tin liên lạc với nhau*** :Có nhiều lúc , chương trình cần chuyển đổi dữ liệu từ vị trí này sang vị trí khác. Ví dụ trong hệ thống Windows, thường có sự chia sẻ và chuyển dữ liệu giữa các cửa sổ. Khi các vị trí được nối kết với nhau trong một hệ thống mạng, việc trao đổi dữ liệu diễn ra rất dễ. Người sử dụng có thể chuyển tập tin hay các E_mail cho nhau từ cùng vị trí hay những vị trí khác.

1.2.6. Hệ thống xử lý thời gian thực

Hệ thống xử lý thời gian thực được sử dụng khi có những đòi hỏi khắt khe về thời gian trên các thao tác của bộ xử lý hoặc dòng dữ liệu, nó thường được dùng điều khiển các thiết bị trong các ứng dụng tận hiến (dedicated). Máy tính phân tích dữ liệu và có thể chỉnh các điều khiển giải quyết cho dữ liệu nhập.

Một hệ điều hành xử lý thời gian thực phải được định nghĩa tốt, thời gian xử lý nhanh. Hệ thống phải cho kết quả chính xác trong khoảng thời gian bị thúc ép nhanh nhất. Có hai hệ thống xử lý thời gian thực là hệ thống thời gian thực cứng và hệ thống thời gian thực mềm..

Hệ thống thời gian thực cứng là công việc được hoàn tất đúng lúc. Lúc đó dữ liệu thường được lưu trong bộ nhớ ngắn hạn hay trong ROM. Việc xử lý theo thời gian thực sẽ xung đột với tất cả hệ thống liệt kê ở trên.

Dạng thứ hai là hệ thống thời gian thực mềm, mỗi công việc có một độ ưu tiên riêng và sẽ được thi hành theo độ ưu tiên đó. Có một số lĩnh vực áp dụng hữu hiệu phương pháp này là multimedia hay thực tại ảo.

CHƯƠNG 2. LUỒNG VÀ TIẾN TRÌNH

2.1. NHU CẦU XỬ LÝ ĐỒNG THỜI

Có 2 động lực chính khiến cho các hệ điều hành hiện đại thường hỗ trợ môi trường đa nhiệm (multitask) trong đó chấp nhận nhiều tác vụ thực hiện đồng thời trên cùng một máy tính :

2.1.1. Tăng hiệu suất sử dụng CPU

Phần lớn các tác vụ (job) khi thi hành đều trải qua nhiều chu kỳ xử lý (sử dụng CPU) và chu kỳ nhập xuất (sử dụng các thiết bị nhập xuất) xen kẽ như sau :

CPU	**IO**	**CPU**	**IO**	**CPU**

Nếu chỉ có 1 tiến trình duy nhất trong hệ thống, thì vào các chu kỳ IO của tác vụ, CPU sẽ hoàn toàn nhàn rỗi. Ý tưởng tăng cường số lượng tác vụ trong hệ thống là để tận dụng CPU : nếu tác vụ 1 xử lý IO, thì có thể sử dụng CPU để thực hiện tác vụ 2...

CPU	**IO**	**CPU**	**IO**	**CPU**

Tác vụ 1

	CPU	**IO**	**CPU**	**IO**

Tác vụ

2.1.2. Tăng tốc độ xử lý

Một số bài toán có bản chất xử lý song song nếu được xây dựng thành nhiều module hoạt động đồng thời thì sẽ tiết kiệm được thời gian xử lý.

Ví dụ : Xét bài toán tính giá trị biểu thức kq = a*b + c*d . Nếu tiến hành tính đồng thời (a*b) và (c*d) thì thời gian xử lý sẽ ngắn hơn là thực hiện tuần tự.

Trong các trường hợp đó, cần có một mô hình xử lý đồng hành thích hợp. Trên máy tính có cấu hình nhiều CPU, hỗ trợ xử lý song song (multiprocessing) thật sự, điều này sẽ giúp tăng hiệu quả thi hành của hệt thống đáng kể.

2.2. KHÁI NIỆM TIẾN TRÌNH(PROCESS) VÀ MÔ HÌNH ĐA TIẾN TRÌNH (MULTIPROCESS)

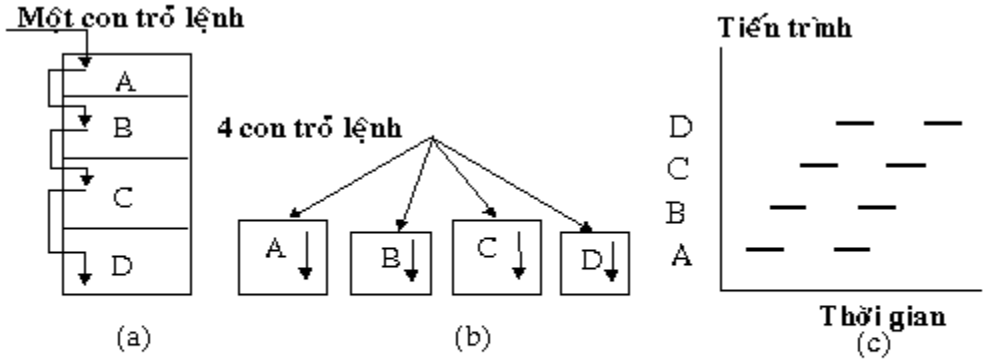
Để hỗ trợ sự đa chương, máy tính phải có khả năng thực hiện nhiều tác vụ đồng thời. Nhưng việc điều khiển nhiều hoạt động song song ở cấp độ phần cứng là rất khó

khăn. Vì thế các nhà thiết kế hệ điều hành đề xuất một mô hình *song song giả lập* bằng cách chuyển đổi bộ xử lý qua lại giữa các chương trình để duy trì hoạt động của nhiều chương trình cùng lúc, điều này tạo cảm giác có nhiều hoạt động được thực hiện đồng thời.

Trong mô hình này, tất cả các phần mềm trong hệ thống được tổ chức thành một số những *tiến trình (process)*. **Tiến trình là một chương trình đang xử lý, sỡ hữu một con trỏ lệnh, tập các thanh ghi và các biến**. Để hoàn thành tác vụ của mình, một tiến trình có thể cần đến một số tài nguyên – như CPU, bộ nhớ chính, các tập tin và thiết bị nhập/xuất.

Cần **phân biệt hai khái niệm *chương trình* và *tiến trình***. Một **chương trình là một thực thể thụ động**, *chứa đựng các chỉ thị điều khiển máy tính để tiến hành một tác vụ* nào đó ; **khi cho thực hiện các chỉ thị này, chương trình chuyển thành tiến trình**, **tiến trình là một thực thể hoạt động**, với con trỏ lệnh xác định chỉ thị kế tiếp sẽ thi hành, kèm theo tập các tài nguyên phục vụ cho hoạt động của tiến trình.

Về mặt ý niệm, có thể xem như mỗi tiến trình sỡ hữu một bộ xử lý ảo cho riêng nó, nhưng trong thực tế, chỉ có một bộ xử lý thật sự được chuyển đổi qua lại giữa các tiến trình. Sự chuyển đổi nhanh chóng này được gọi là *sự đa chương (multiprogramming)* . Hệ điều hành chịu trách nhiệm sử dụng một thuật toán điều phối để quyết định thời điểm cần dừng hoạt động của tiến trình đang xử lý để phục vụ một tiến trình khác, và lựa chọn tiến trình tiếp theo sẽ được phục vụ. Bộ phận thực hiện chức năng này của hệ điều hành được gọi là *bộ điều phối (scheduler)*.



Hình 2.1 (a) Đa chương với 4 chương trình
(b) Mô hình khái niệm với 4 chương trình độc lập
(c) Tại một thời điểm chỉ có một chương trình hoạt động

2.3. KHÁI NIỆM LUỒNG (THREAD) VÀ MÔ HÌNH ĐA LUỒNG (MULTITHREAD)

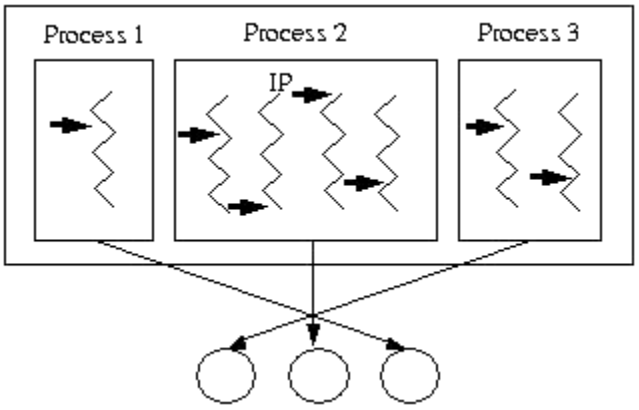
Trong hầu hết các hệ điều hành, **mỗi tiến trình có một không gian địa chỉ và chỉ có một dòng xử lý**. Tuy nhiên, có nhiều tình huống người sử dụng mong muốn có nhiều dòng xử lý cùng chia sẻ một không gian địa chỉ, và các dòng xử lý này hoạt động song song tương tự như các tiến trình phân biệt (ngoại trừ việc chia sẻ không gian địa chỉ).

Ví dụ : Một server quản lý tập tin thỉnh thoảng phải tự khóa để chờ các thao tác truy xuất đĩa hoàn tất.Nếu server có nhiều dòng xử lý, hệ thống có thể xử lý các yêu cầu mới trong

khi một dòng xử lý bị khoá. Như vậy việc thực hiện chương trình sẽ có hiệu quả hơn. Điều này không thể đạt được bằng cách tạo hai tiến trình server riêng biệt vì cần phải chia sẻ cùng một vùng đệm, do vậy bắt buộc phải chia sẻ không gian địa chỉ.
Chính vì các tình huống tương tự, người ta cần có một cơ chế xử lý mới cho phép có nhiều dòng xử lý trong cùng một tiến trình.
Ngày nay đã có nhiều hệ điều hành cung cấp một cơ chế như thế và gọi là *luồng (threads)*.

2.3.1. Nguyên lý chung:

Một luồng là một đơn vị xử lý cơ bản trong hệ thống . Mỗi luồng xử lý tuần tự đoạn code của nó, sỡ hữu một con trỏ lệnh, tập các thanh ghi và một vùng nhớ stack riêng. Các luồng chia sẻ CPU với nhau giống như cách chia sẻ giữa các tiến trình: một luồng xử lý trong khi các luồng khác chờ đến lượtù. Một luồng cũng có thể tạo lập các tiến trình con, và nhận các trạng thái khác nhau như một tiến trình thật sự. Một tiến trình có thể sỡ hữu nhiều luồng.
Các tiến trình tạo thành những thực thể độc lập. Mỗi tiến trình có một tập tài nguyên và một môi trường riêng (một con trỏ lệnh, một Stack , các thanh ghi và không gian địa chỉ). Các tiến trình hoàn toàn độc lập với nhau, chỉ có thể liên lạc thông qua các cơ chế thông tin giữa các tiến trình mà hệ điều hành cung cấp. Ngược lại, các luồng trong cùng một tiến trình lại chia sẻ một không gian địa chỉ chung , điều này có nghĩa là các luồng có thể chia sẻ các biến toàn cục của tiến trình. Một luồng có thể truy xuất đến cả các stack của những luồng khác trong cùng tiến trình. Cấu trúc này không đề nghị một cơ chế bảo vệ nào, và điều này cũng không thật cần thiết vì các luồng trong cùng một tiến trình thuộc về cùng một sỡ hữu chủ đã tạo ra chúng trong ý định cho phép chúng hợp tác với nhau.



Các luồng trong cùng một luồng

2.3.2. Phân bổ thông tin lưu trữ

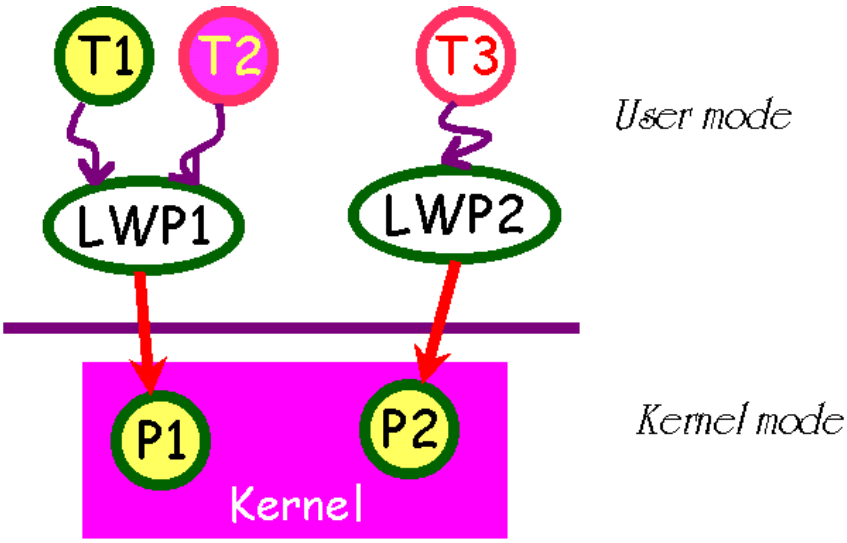
Tiến trình
Không gian địa chỉ
Tài nguyên toàn cục
Các thông tin thống kê

Tiểu trình
Con trỏ lệnh + các thanh ghi
Stack
Tài nguyên cục bộ

Cấu trúc mô tả tiến trình và luồng

2.3.3. Kernel thread và user thread

Khái niệm luồng có thể được cài đặt trong kernel của Hệ điều hành, khi đó đơn vị cơ sở sử dụng CPU để xử lý là luồng, Hệ điều hành sẽ phân phối CPU cho các luồng trong hệ thống. Tuy nhiên đối với một số hệ điều hành, khái niệm luồng chỉ được hỗ trợ như một đối tượng người dùng, các thao tác luồng được cung cấp kèm theo do một bộ thư viện xử lý trong chế độ người dùng không đặc quyền (user mode). Lúc này Hệ điều hành sẽ chỉ biết đến khái niệm tiến trình, do vây cận co cơ chế để liên kết các luồng cùng một tiến trình với tiến trình cha trong kernel_ đối tượng này đôi lúc được gọi là LWP (lightweight process).



CHƯƠNG 3. LẬP LỊCH TIẾN TRÌNH

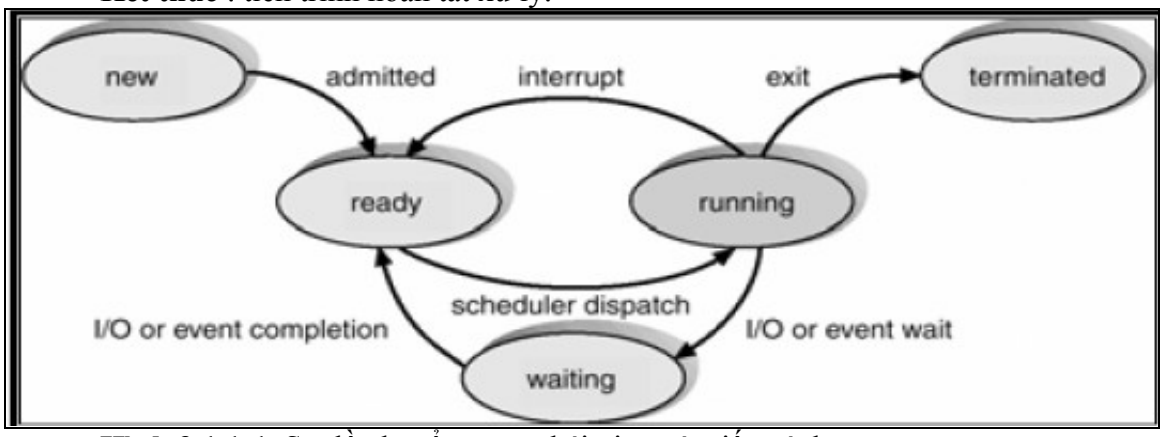
3.1. Tổ chức quản lý tiến trình

3.1.1. Các trạng thái của tiến trình

Trạng thái của tiến trình tại một thời điểm được xác định bởi hoạt động hiện thời của tiến trình tại thời điểm đó. Trong **quá trình sống**, một tiến trình thay đổi trạng thái do nhiều nguyên nhân như : phải chờ một sự kiện nào đó xảy ra, hay đợi một thao tác nhập/xuất hoàn tất, buộc phải dừng hoạt động do đã hết thời gian xử lý …

Tại một thời điểm, một tiến trình có thể nhận trong một các trạng thái sau đây :

- **Mới tạo** : tiến trình đang được tạo lập.
- **Running** : các chỉ thị của tiến trình đang được xử lý.
- **Waiting** : tiến trình chờ được cấp phát một tài nguyên, hay chờ một sự kiện xảy ra .
- **Ready** : tiến trình chờ được cấp phát CPU để xử lý.
- **Kết thúc** : tiến trình hoàn tất xử lý.



Hình 3.1.1-1. Sơ đồ chuyển trạng thái giữa các tiến trình

Tại một thời điểm, chỉ có một tiến trình có thể nhận trạng thái *running* trên một bộ xử lý bất kỳ. Trong khi đó, **nhiều tiến trình có thể ở trạng thái *blocked* hay *ready*.**

Các cung chuyển tiếp trong sơ đồ trạng thái biểu diễn sáu sự chuyển trạng thái có thể xảy ra trong các điều kiện sau :

- *Tiến trình mới tạo được đưa vào hệ thống(bộ nhớ trong)*
- *Bộ điều phối cấp phát cho tiến trình một khoảng thời gian sử dụng CPU*
- *Tiến trình kết thúc*
- *Tiến trình yêu cầu một tài nguyên nhưng chưa được đáp ứng vì tài nguyên chưa sẵn sàng để cấp phát tại thời điểm đó ; hoặc tiến trình phải chờ một sự kiện hay thao tác nhập/xuất.*
- *Bộ điều phối chọn một tiến trình khác để cho xử lý .*

- *Tài nguyên mà tiến trình yêu cầu trở nên sẵn sàng để cấp phát ; hay sự kiện hoặc thao tác nhập/xuất tiến trình đang đợi hoàn tất.*

3.1.2. Chế độ xử lý của tiến trình

Để đảm bảo hệ thống hoạt động đúng đắn, **hệ điều hành cần phải được bảo vệ khỏi sự xâm phạm của các tiến trình. Bản thân các tiến trình và dữ liệu cũng cần được bảo vệ để tránh các ảnh hưởng sai lạc lẫn nhau**. Một cách tiếp cận để giải quyết vấn đề là phân biệt hai chế độ xử lý cho các tiến trình : ***chế độ không đặc quyền*** **và** ***chế độ đặc quyền*** nhờ vào sự trợ giúp của cơ chế phần cứng. ***Tập lệnh của CPU được phân chia thành các lệnh đặc quyền và lệnh không đặc quyền. Cơ chế phần cứng chỉ cho phép các lệnh đặc quyền được thực hiện trong chế độ đặc quyền***. Thông thường chỉ có hệ điều hành hoạt động trong chế độ đặc quyền, các tiến trình của người dùng hoạt động trong chế độ không đặc quyền, không thực hiện được các lệnh đặc quyền có nguy cơ ảnh hưởng đến hệ thống. Như vậy hệ điều hành được bảo vệ. **Khi một tiến trình người dùng gọi đến một lời gọi hệ thống, tiến trình của hệ điều hành xử lý lời gọi này sẽ hoạt động trong chế độ đặc quyền, sau khi hoàn tất thì trả quyền điều khiển về cho tiến trình người dùng trong chế độ không đặc quyền.**

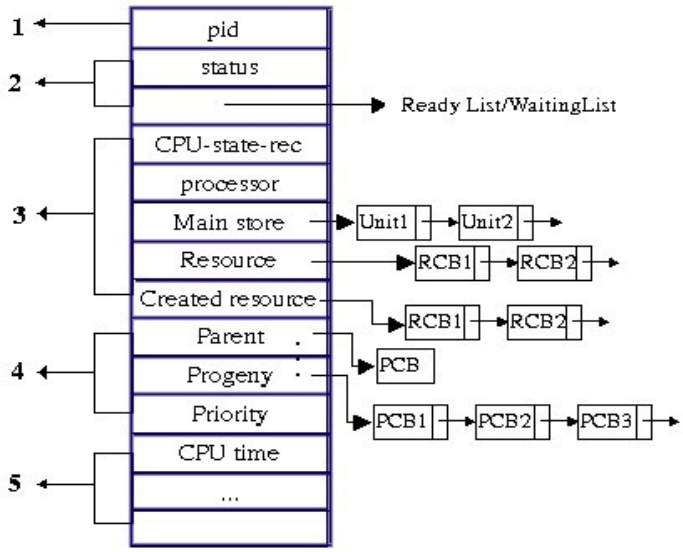
Hình 3.1.2-1. Hai chế độ xử lý

3.1.3. Cấu trúc dữ liệu khối quản lý tiến trình

Hệ điều hành quản lý các tiến trình trong hệ thống thông qua khối quản lý tiến trình (process control block -PCB). ***PCB là một vùng nhớ lưu trữ các thông tin mô tả cho tiến trình, với các thành phần chủ yếu bao gồm*** :

- **Định danh của tiến trình (1)** : giúp phân biệt các tiến trình
- **Trạng thái tiến trình (2)**: xác định hoạt động hiện hành của tiến trình.
- **Ngữ cảnh của tiến trình (3)**: *mô tả các tài nguyên tiến trình* đang trong quá trình, hoặc để phục vụ cho hoạt động hiện tại, hoặc để làm cơ sở phục hồi hoạt động cho tiến trình, bao gồm các thông tin về:
    - *Trạng thái CPU*: bao gồm nội dung các thanh ghi, quan trọng nhất là con trỏ lệnh IP lưu trữ địa chỉ câu lệnh kế tiếp tiến trình sẽ xử lý. Các thông tin này cần được lưu trữ khi xảy ra một ngắt, nhằm có thể cho phép phục hồi hoạt động của tiến trình đúng như trước khi bị ngắt.
    - *Bộ xử lý*: dùng cho máy có cấu hình nhiều CPU, xác định số hiệu CPU mà tiến trình đang sử dụng.
    - *Bộ nhớ chính*: danh sách các khối nhớ được cấp cho tiến trình.
    - *Tài nguyên sử dụng*: danh sách các tài mguyên hệ thống mà tiến trình đang sử dụng.
    - *Tài nguyên tạo lập*: danh sách các tài nguyên được tiến trình tạo lập.

- **Thông tin giao tiếp (4)**: phản ánh các thông tin về quan hệ của tiến trình với các tiến trình khác trong hệ thống :
  - *Tiến trình cha*: của một tiến trình là tiến trình tạo lập ra tiến trình này .
  - *Tiến trình con*: của một tiến trình là các tiến trình do tiến trình này tạo lập .
  - *Độ ưu tiên* : giúp bộ điều phối có thông tin để lựa chọn tiến trình được cấp CPU.
- **Thông tin thống kê (5)**: đây là những thông tin thống kê về hoạt động của tiến trình, như thời gian đã sử dụng CPU,thời gian chờ. Các thông tin này có thể có ích cho công việc đánh giá tình hình hệ thống và dự đoán các tình huống tương lai.



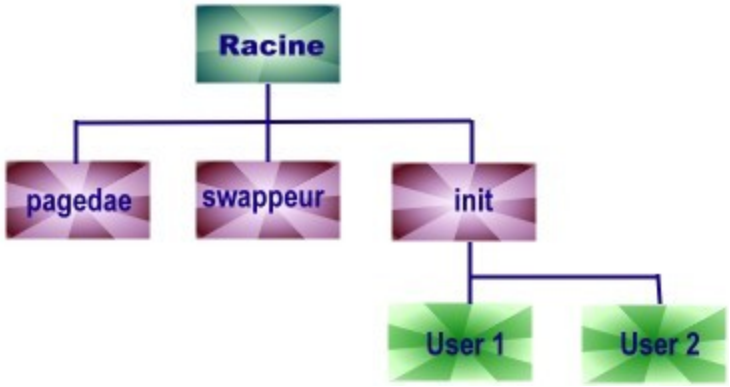
Hình 3.1.3-1. Khối điều khiển tiến trình

3.1.4. Thao tác trên tiến trình

Hệ điều hành cung cấp các thao tác chủ yếu sau đây trên một tiến trình :
- Tạo lập tiến trình (create)
- Kết thúc tiến trình (destroy)
- Tạm dừng tiến trình (suspend)
- Tái kích hoạt tiến trình (resume)
- Thay đổi độ ưu tiên tiến trình

3.1.4.1. Tạo lập tiến trình

Trong quá trình xử lý, một tiến trình có thể tạo lập nhiều tiến trình mới bằng cách sử dụng một lời gọi hệ thống tương ứng. Tiến trình gọi lời gọi hệ thống để tạo tiến trình mới sẽ được gọi là tiến trình *cha*, tiến trình được tạo gọi là tiến trình *con*. Mỗi tiến trình con đến lượt nó lại có thể tạo các tiến trình mới…quá trình này tiếp tục sẽ tạo ra một *cây tiến trình*.



Hình vẽ2.5 Một cây tiến trình trong hệ thống UNIX

Các **công việc hệ điều hành cần thực hiện khi tạo lập tiến trình** bao gồm :

- *Định danh cho tiến trình mới phát sinh*
- *Đưa tiến trình vào danh sách quản lý của hệ thống*
- *Xác định độ ưu tiên cho tiến trình*
- *Tạo PCB cho tiến trình*
- *Cấp phát các tài nguyên ban đầu cho tiến trình*

Khi một tiến trình tạo lập một tiến trình con, tiến trình con có thể sẽ được hệ điều hành trực tiếp cấp phát tài nguyên hoặc được tiến trình cha cho thừa hưởng một số tài nguyên ban đầu.

Khi một tiến trình tạo tiến trình mới, tiến trình ban đầu có thể xử lý theo một trong hai khả năng sau :

- Tiến trình cha tiếp tục xử lý đồng hành với tiến trình con.
- Tiến trình cha chờ đến khi một tiến trình con nào đó, hoặc tất cả các tiến trình con kết thúc xử lý.

Các hệ điều hành khác nhau có thể chọn lựa các cài đặt khác nhau để thực hiện thao tác tạo lập một tiến trình.

3.1.4.2. Kết thúc tiến trình

Một **tiến trình kết thúc xử lý khi nó hoàn tất chỉ thị cuối cùng và sử dụng một lời gọi hệ thống để yêu cầu hệ điều hành hủy bỏ nó(giải phóng CPU)**. Đôi khi một tiến trình có thể yêu cầu hệ điều hành kết thúc xử lý của một tiến trình khác. ***Khi một tiến trình kết thúc, hệ điều hành thực hiện các công việc*** :

- *Thu hồi các tài nguyên hệ thống đã cấp phát cho tiến trình*
- *Hủy tiến trình khỏi tất cả các danh sách quản lý của hệ thống*
- *Hủy bỏ PCB của tiến trình*

Hầu hết các hệ điều hành không cho phép các tiến trình con tiếp tục tồn tại nếu tiến trình cha đã kết thúc. Trong những hệ thống như thế, hệ điều hành sẽ tự động phát sinh một loạt các thao tác kết thúc tiến trình con.

3.1.5. Cấp phát tài nguyên cho tiến trình

Khi có nhiều người sử dụng đồng thời làm việc trong hệ thống, hệ điều hành cần phải cấp phát các tài nguyên theo yêu cầu cho mỗi người sử dụng. Do tài nguyên hệ thống thường rất giới hạn và có khi không thể chia sẻ, nên hiếm khi tất cả các yêu cầu tài

nguyên đồng thời đều được thỏa mãn. Vì thế cần phải nghiên cứu một phương pháp để chia sẻ một số tài nguyên hữu hạn giữa nhiều tiến trình người dùng đồng thời. Hệ điều hành quản lý nhiều loại tài nguyên khác nhau (CPU, bộ nhớ chính, các thiết bị ngoại vi …), với mỗi loại cần có một cơ chế cấp phát và các chiến lược cấp phát hiệu qủa. ***Mỗi tài nguyên được biểu diễn thông qua một cấu trúc dữ liệu, khác nhau về chi tiết cho từng loại tài nguyên, nhưng cơ bản chứa đựng các thông tin sau*** :

- **Định danh tài nguyên**
- **Trạng thái tài nguyên** : đây là các thông tin mô tả chi tiết trạng thái tài nguyên : phần nào của tài nguyên đã cấp phát cho tiến trình, phần nào còn có thể sử dụng ?
- **Hàng đợi trên một tài nguyên** : danh sách các tiến trình đang chờ được cấp phát tài nguyên tương ứng.
- **Bộ cấp phát** : là **đoạn code đảm nhiệm việc cấp phát một tài nguyên** đặc thù. Một số tài nguyên đòi hỏi các giải thuật đặc biệt (như CPU, bộ nhớ chính, hệ thống tập tin), trong khi những tài nguyên khác (như các thiết bị nhập/xuất) có thể cần các giải thuật cấp phát và giải phóng tổng quát hơn.

Hình 3.1.5-1. Khối quản lý tài nguyên

Các **mục tiêu của kỹ thuật cấp phát** :

- *Bảo đảm một số lượng hợp lệ các tiến trình truy xuất đồng thời đến các tài nguyên không chia sẻ được.*
- *Cấp phát tài nguyên cho tiến trình có yêu cầu trong một khoảng thời gian trì hoãn có thể chấp nhận được.*
- *Tối ưu hóa sự sử dụng tài nguyên.*

Để có thể thõa mãn các mục tiêu kể trên, cần phải giải quyết các vấn đề nảy sinh khi có nhiều tiến trình đồng thời yêu cầu một tài nguyên không thể chia sẻ.

3.2. Lập lịch tiến trình

Trong môi trường đa chương, có thể xảy ra tình huống nhiều tiến trình đồng thời sẵn sàng để xử lý. Mục tiêu của các hệ phân chia thời gian (time-sharing) là chuyển đổi CPU qua lại giữa các tiến trình một cách thường xuyên để nhiều người sử dụng có thể tương tác cùng lúc với từng chương trình trong quá trình xử lý.

Để thực hiện được mục tiêu này, **hệ điều hành phải lựa chọn tiến trình được xử lý tiếp theo**. Bộ điều phối sẽ sử dụng một giải thuật điều phối thích hợp để thực hiện nhiệm vụ này. Một thành phần khác của hệ điều hành cũng tiềm ẩn trong công tác điều

phối là *bộ phân phối* (dispatcher). **Bộ phân phối *sẽ chịu trách nhiệm chuyển đổi ngữ cảnh và trao CPU cho tiến trình được chọn bởi bộ điều phối để xử lý*.**

3.2.1. Giới thiệu

3.2.1.1. Mục tiêu lập lịch

Bộ điều phối không cung cấp cơ chế, mà đưa ra các quyết định. Các hệ điều hành xây dựng nhiều chiến lược khác nhau để thực hiện việc điều phối, nhưng tựu chung cần đạt được các mục tiêu sau :

- **Sự công bằng (Fairness):** Các tiến trình chia sẻ CPU một cách công bằng, không có tiến trình nào phải chờ đợi vô hạn để được cấp phát CPU
- **Tính hiệu qủa (Efficiency):** Hệ thống phải tận dụng được CPU 100% thời gian.
- **Thời gian đáp ứng hợp lý (Response time):** Cực tiểu hoá thời gian hồi đáp cho các tương tác của người sử dụng
- **Thời gian lưu lại trong hệ thống (Turnaround Time):** Cực tiểu hóa thời gian hoàn tất các tác vụ xử lý theo lô.
- **Thông lượng tối đa (Throughput):** Cực đại hóa số công việc được xử lý trong một đơn vị thời gian.

Tuy nhiên thường không thể thỏa mãn tất cả các mục tiêu kể trên vì bản thân chúng có sự mâu thuẫn với nhau mà chỉ có thể dung hòa chúng ở mức độ nào đó.

3.2.1.2. Các đặc điểm của tiến trình

Điều phối hoạt động của các tiến trình là một vấn đề rất phức tạp, đòi hỏi hệ điều hành khi giải quyết phải xem xét nhiều yếu tố khác nhau để có thể đạt được những mục tiêu đề ra. Một số đặc tính của tiến trình cần được quan tâm như tiêu chuẩn điều phối :

- **Tính hướng xuất / nhập của tiến trình (I/O-boundedness):**
  Khi một tiến trình nhận được CPU, chủ yếu nó chỉ sử dụng CPU đến khi phát sinh một yêu cầu nhập xuất ? Hoạt động của các tiến trình như thế thường bao gồm nhiều lượt sử dụng CPU , mỗi lượt trong một thời gian khá ngắn.
- **Tính hướng xử lý của tiến trình (CPU-boundedness):**
  Khi một tiến trình nhận được CPU, nó có khuynh hướng sử dụng CPU đến khi hết thời gian dành cho nó ? Hoạt động của các tiến trình như thế thường bao gồm một số ít lượt sử dụng CPU , nhưng mỗi lượt trong một thời gian đủ dài.
- **Tiến trình tương tác hay xử lý theo lô :**
  Người sử dụng theo kiểu tương tác thường yêu cầu được hồi đáp tức thời đối với các yêu cầu của họ, trong khi các tiến trình của tác vụ được xử lý theo lô nói chung có thể trì hoãn trong một thời gian chấp nhận được.
- **Độ ưu tiên của tiến trình :**
  Các tiến trình có thể được phân cấp theo một số tiêu chuẩn đánh giá nào đó, một cách hợp lý, các tiến trình quan trọng hơn (có độ ưu tiên cao hơn) cần được ưu tiên hơn.
- **Thời gian đã sử dụng CPU của tiến trình :**
  Một số quan điểm ưu tiên chọn những tiến trình đã sử dụng CPU nhiều thời gian nhất vì hy vọng chúng sẽ cần ít thời gian nhất để hoàn tất và rời khỏi hệ thống .

Tuy nhiên cũng có quan điểm cho rằng các tiến trình nhận được CPU trong ít thời gian là những tiến trình đã phải chờ lâu nhất, do vậy ưu tiên chọn chúng.

- **Thời gian còn lại tiến trình cần để hoàn tất :**
- Có thể giảm thiểu thời gian chờ đợi trung bình của các tiến trình bằng cách cho các tiến trình cần ít thời gian nhất để hoàn tất được thực hiện trước. Tuy nhiên đáng tiếc là rất hiếm khi biết được tiến trình cần bao nhiêu thời gian nữa để kết thúc xử lý.

3.2.1.3. Điều phối không độc quyền và điều phối độc quyền (preemptive/nopreemptive)

Thuật toán điều phối cần xem xét và quyết định thời điểm chuyển đổi CPU giữa các tiến trình. Hệ điều hành có thể thực hiện cơ chế điều phối theo nguyên lý *độc quyền* hoặc *không độc quyền*.

- **Điều phối độc quyền** : Nguyên lý điều phối *độc quyền* **cho phép một tiến trình khi nhận được CPU sẽ có quyền độc chiếm CPU đến khi hoàn tất xử lý hoặc tự nguyện giải phóng CPU**. Khi đó **quyết định điều phối CPU sẽ xảy ra trong các tình huống** sau:
    - *Khi tiến trình chuyển từ trạng thái đang xử lý(running) sang trạng thái bị khóa blocked (ví dụ chờ một thao tác nhập xuất hay chờ một tiến trình con kết thúc…).*
    - *Khi tiến trình kết thúc.*

  Các giải thuật độc quyền thường đơn giản và dễ cài đặt. Tuy nhiên chúng thường không thích hợp với các hệ thống tổng quát nhiều người dùng, vì nếu cho phép một tiến trình có quyền xử lý bao lâu tùy ý, có nghĩa là tiến trình này có thể giữ CPU một thời gian không xác định, có thể ngăn cản những tiến trình còn lại trong hệ thống có một cơ hội để xử lý.
- **Điều phối không độc quyền** : Ngược với nguyên lý độc quyền, điều phối theo nguyên lý *không độc quyền* cho phép tạm dừng hoạt động của một tiến trình đang sẵn sàng xử lý. Khi một tiến trình nhận được CPU, nó **vẫn được sử dụng CPU đến khi hoàn tất hoặc tự nguyện giải phóng CPU**, nhưng **khi có một tiến trình khác có độ ưu tiên có thể dành quyền sử dụng CPU của tiến trình ban đầu**. Như vậy là tiến trình có thể bị tạm dừng hoạt động bất cứ lúc nào mà không được báo trước, để tiến trình khác xử lý. Các **quyết định điều phối xảy ra khi** :
    - *Khi tiến trình chuyển từ trạng thái đang xử lý (running) sang trạng thái bị khóa blocked (ví dụ chờ một thao tác nhập xuất hay chờ một tiến trình con kết thúc…).*
    - *Khi tiến trình chuyển từ trạng thái đang xử lý (running) sang trạng thái ready (ví dụ xảy ra một ngắt).*
    - *Khi tiến trình chuyển từ trạng thái chờ (blocked) sang trạng thái ready (ví dụ một thao tác nhập/xuất hoàn tất).*
    - *Khi tiến trình kết thúc.*

  Các thuật toán điều phối theo nguyên tắc không độc quyền ngăn cản được tình trạng một tiến trình độc chiếm CPU, nhưng việc tạm dừng một tiến trình có thể dẫn đến các mâu thuẫn trong truy xuất, đòi hỏi phải sử dụng một phương pháp đồng bộ hóa thích hợp để giải quyết.

Trong các hệ thống sử dụng nguyên lý điều phối độc quyền có thể xảy ra tình trạng các tác vụ cần thời gian xử lý ngắn phải chờ tác vụ xử lý với thời gian rất dài hoàn tất! Nguyên lý điều phối độc quyền thường chỉ thích hợp với các hệ xử lý theo lô.

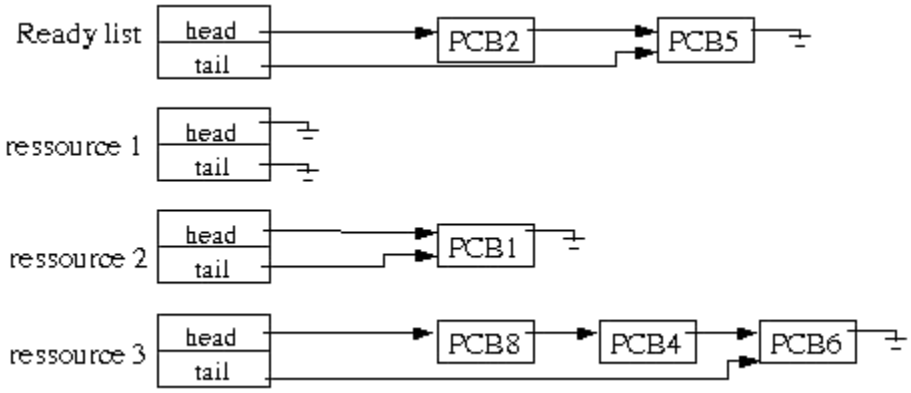
Đối với các hệ thống tương tác(time sharing), các hệ thời gian thực (real time),cần phải sử dụng nguyên lý điều phối không độc quyền để các tiến trình quan trọng có cơ hội hồi đáp kịp thời. Tuy nhiên thực hiện điều phối theo nguyên lý không độc quyền đòi hỏi những cơ chế phức tạp trong việc phân định độ ưu tiên, và phát sinh thêm chi phí khi chuyển đổi CPU qua lại giữa các tiến trình.

3.2.2. Tổ chức lập lịch

3.2.2.1. Các danh sách sử dụng trong quá trình lập lịch

Hệ điều hành sử dụng **hai loại danh sách** để thực hiện điều phối các tiến trình là ***danh sách sẵn sàng (ready list)* và *danh sách chờ đợi(waiting list)***.
Khi một tiến trình bắt đầu đi vào hệ thống, nó được chèn vào danh sách các tác vụ (job list). Danh sách này *bao gồm tất cả các tiến trình của hệ thống*. Nhưng **chỉ các tiến trình đang thường trú trong bộ nhớ chính và ở trạng thái sẵn sàng tiếp nhận CPU để hoạt động mới được đưa vào *danh sách sẵn sàng***.
Bộ điều phối sẽ chọn một tiến trình trong danh sách sẵn sàng và cấp CPU cho tiến trình đó. **Tiến trình được cấp CPU sẽ thực hiện xử lý, và có thể chuyển sang trạng thái chờ khi xảy ra các sự kiện như đợi một thao tác nhập/xuất hoàn tất, yêu cầu tài nguyên chưa được thỏa mãn, được yêu cầu tạm dừng ...Khi đó tiến trình sẽ được chuyển sang một *danh sách chờ đợi***.

Hệ điều hành chỉ sử dụng một danh sách sẵn sàng cho toàn hệ thống, nhưng **mỗi một tài nguyên (thiết bị ngoại vi) có một danh sách chờ đợi riêng bao gồm các tiến trình đang chờ được cấp phát tài nguyên đó**.

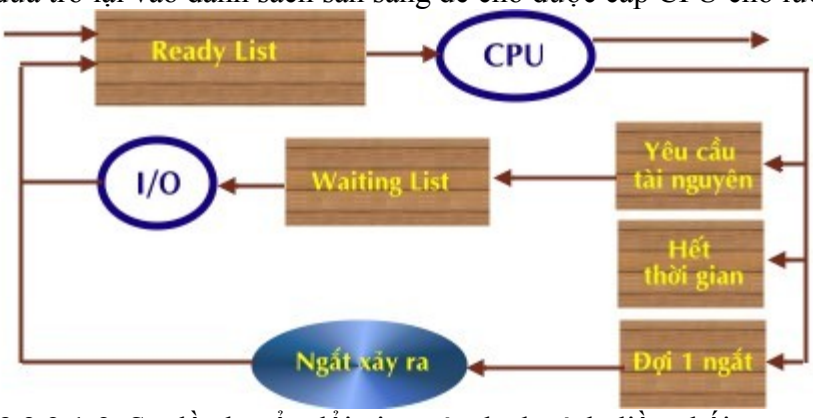


Hình 3.2.2.1-1. Các danh sách điều phối

Quá trình xử lý của một tiến trình trải qua những chu kỳ chuyển đổi qua lại giữa danh sách sẵn sàng và danh sách chờ đợi. Sơ đồ dưới đây mô tả sự điều phối các tiến trình dựa trên các danh sách của hệ thống.

Thoạt đầu tiến trình mới được đặt trong danh sách các tiến trình sẵn sàng (ready list), nó sẽ đợi trong danh sách này cho đến khi được chọn để cấp phát CPU và bắt đầu xử lý. Sau đó có thể xảy ra một trong các tình huống sau :

- Tiến trình phát sinh một yêu cầu một tài nguyên mà hệ thống chưa thể đáp ứng, khi đó tiến trình sẽ được chuyển sang danh sách các tiến trình đang chờ tài nguyên tương ứng.
- Tiến trình có thể bị bắt buộc tạm dừng xử lý do một ngắt xảy ra, khi đó tiến trình được đưa trở lại vào danh sách sẵn sàng để chờ được cấp CPU cho lượt tiếp theo.



Hình 3.2.2.1-2. Sơ đồ chuyển đổi giữa các danh sách điều phối

Trong trường hợp đầu tiên, tiến trình cuối cùng sẽ chuyển từ trạng thái blocked sang trạng thái ready và lại được đưa trở vào danh sách sẵn sàng. Tiến trình lặp lại chu kỳ này cho đến khi hoàn tất tác vụ thì được hệ thống hủy bỏ khỏi mọi danh sách điều phối.

3.2.2.2. Các cấp độ lập lịch

Thực ra công việc điều phối được hệ điều hành thực hiện ở hai mức độ : *điều phối tác vụ (job scheduling)* và *điều phối tiến trình (process scheduling)*.

a) Lập lịch tác vụ

Quyết định lựa chọn tác vụ nào được đưa vào hệ thống, và nạp những tiến trình của tác vụ đó vào bộ nhớ chính để thực hiện. Chức năng điều phối tác vụ ***quyết định mức độ đa chương của hệ thống (số lượng tiến trình trong bộ nhớ chính)***. Khi hệ thống tạo lập một tiến trình, hay có một tiến trình kết thúc xử lý thì chức năng điều phối tác vụ mới được kích hoạt. Vì mức độ đa chương tương đối ổn định nên chức năng điều phối tác vụ có tần suất hoạt động thấp .

Để hệ thống hoạt động tốt, bộ điều phối tác vụ cần biệt tính chất của tiến trình là *hướng nhập xuất (I/O bounded)* hay *hướng xử lý (CPU bounded)*. Một tiến trình được gọi là *hướng nhập xuất* nếu nó chủ yếu nó chỉ sử dụng CPU để thực hiện các thao tác nhập xuất. Ngược lại một tiến trình được gọi là *hướng xử lý* nếu nó chủ yếu nó chỉ sử dụng CPU để thực hiện các thao tác tính toán. Để cân bằng hoạt động của CPU và các thiết bị ngoại vi, bộ điều phối tác vụ nên lựa chọn các tiến trình để nạp vào bộ nhớ sao cho hệ thống là sự pha trộn hợp lý giữa các tiến trình *hướng nhập xuất* và các tiến trình *hướng xử lý*

b) Lập lịch tiến trình

Chọn một tiến trình ở trạng thái sẵn sàng (đã được nạp vào bộ nhớ chính, và có đủ tài nguyên để hoạt động) và cấp phát CPU cho tiến trình đó thực hiện. Bộ điều phối tiến trình có tần suất hoạt động cao, sau mỗi lần xảy ra ngắt (do đồng hồ báo giờ, do các thiết bị ngoại vi...), thường là 1 lần trong khoảng 100ms. Do vậy để nâng cao hiệu suất của hệ thống, cần phải tăng tốc độ xử lý của bộ điều phối tiến trình. **Chức năng**

điều phối tiến trình là một trong chức năng cơ bản, quan trọng nhất của hệ điều hành.

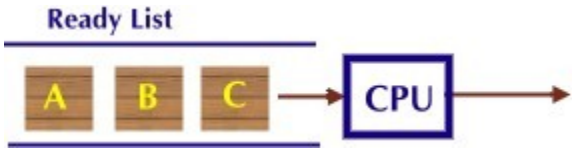
Trong nhiều hệ điều hành, có thể không có bộ điều phối tác vụ hoặc tách biệt rất ít đối với bộ điều phối tiến trình. Một vài hệ điều hành lại đưa ra một cấp độ điều phối trung gian kết hợp cả hai cấp độ điều phối tác vụ và tiến trình

Hình 3.2.2.2-1. Cấp độ điều phối trung gian

3.2.3. Các thuật toán lập lịch

3.2.3.1. Chiến lược FIFO

- **Nguyên tắc** : **CPU được cấp phát cho tiến trình đầu tiên trong danh sách sẵn sàng có yêu cầu, là tiến trình được đưa vào hệ thống sớm nhất**. Đây là thuật toán điều phối **theo nguyên tắc độc quyền**. Một khi CPU được cấp phát cho tiến trình, CPU chỉ được tiến trình tự nguyện giải phóng khi kết thúc xử lý hay khi có một yêu cầu nhập/xuất.



Hình 3.2.3.1-1. Điều phối FIFO

- **Ví dụ :**

Tiến trình	Thời điểm vào RL	Thời gian xử lý
P1	0	24
P2	1	3
P3	2	3

Thứ tự cấp phát CPU cho các tiến trình là :

P1	P2	P3

0	‘24	27 30

thời gian chờ đợi được xử lý là 0 đối với P1, (24 -1) với P2 và (24+3-2) với P3. Thời gian chờ trung bình là (0+23+25)/3 = 16 milisecondes.

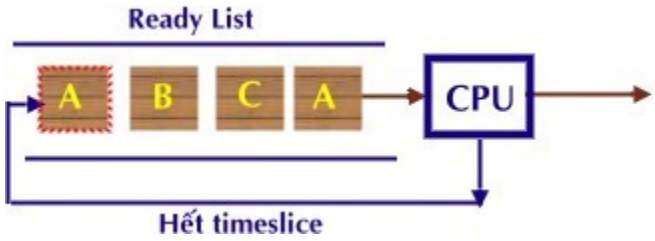
- **Thảo luận** : *Thời gian chờ trung bình không đạt cực tiểu, và biến đổi đáng kể đối với các giá trị về thời gian yêu cầu xử lý và thứ tự khác nhau của các tiến trình trong danh sách sẵn sàng. Có thể xảy ra hiện tượng tích lũy thời gian chờ, khi các tất cả các tiến trình (có thể có yêu cầu thời gian ngắn) phải chờ đợi một tiến trình có yêu cầu thời gian dài kết thúc xử lý.*

  Giải thuật này đặc biệt không phù hợp với các hệ phân chia thời gian, trong các hệ này, cần cho phép mỗi tiến trình được cấp phát CPU đều đặn trong từng khoảng thời gian.

3.2.3.2. Lập lịch xoay vòng (Round Robin)

- **Nguyên tắc** : **Danh sách sẵn sàng được xử lý như một danh sách vòng**, bộ điều phối **lần lượt cấp phát cho từng tiến trình trong danh sách một khoảng thời gian sử dụng CPU gọi là *quantum*(lượng tử thời gian)**. Đây **là một giải thuật điều phối không độc quyền** : khi một ***tiến trình sử dụng CPU đến hết thời gian quantum dành cho nó, hệ điều hành thu hồi CPU và cấp cho tiến trình kế tiếp*** trong danh sách. Nếu ***tiến trình bị khóa hay kết thúc trước khi sử dụng hết thời gian quantum, hệ điều hành cũng lập tức cấp phát CPU cho tiến trình khác. Khi tiến trình tiêu thụ hết thời gian CPU dành cho nó mà chưa hoàn tất, tiến trình được đưa trở lại vào cuối danh sách sẵn sàng*** để đợi được cấp CPU trong lượt kế tiếp.

- **Ví dụ**:



Hình 3.2.3.2-1. Lập lịch Round Robin

Tiến trình	Thời điểm vào RL	Thời gian xử lý
P1	0	24
P2	1	3
P3	2	3

Nếu sử dụng quantum là 4 milisecondes, thứ tự cấp phát CPU sẽ là :

P1	P2	P3	P1	P1	P1	P1	P1
0	‘4	7	10	14	18	22	26 30

Thời gian chờ đợi trung bình sẽ là (0+6+3+5)/3 = 4.66 milisecondes.
Nếu có *n* tiến trìh trong danh sách sẵn sàng và sử dụng quantum *q*, thì mỗi tiến trình sẽ được cấp phát CPU *1/n* trong từng khoảng thời gian *q*. Mỗi tiến trình sẽ không phải đợi quá *(n-1)q* đơn vị thời gian trước khi nhận được CPU cho lượt kế tiếp.

- **Thảo luận** : *Vấn đề đáng quan tâm đối với giải thuật RR là* ***độ dài của quantum****. Nếu thời lượng quantum quá bé sẽ phát sinh quá nhiều sự chuyển đổi giữa các tiến trình và khiến cho việc sử dụng CPU kém hiệu qủa. Nhưng nếu sử dụng quantum quá lớn sẽ làm tăng thời gian hồi đáp và giảm khả năng tương tác của hệ thống.*

3.2.3.3. Lập lịch với độ ưu tiên

- **Nguyên tắc** : Mỗi **tiến trình được gán cho một độ ưu tiên tương ứng, tiến trình có độ ưu tiên cao nhất sẽ được chọn** để cấp phát CPU đầu tiên. Độ ưu tiên có thể được định nghĩa nội tại hay nhờ vào các yếu tố bên ngoài. Độ ưu tiên nội tại sử dụng các đại lượng có thể đo lường để tính toán độ ưu tiên của tiến trình, ví dụ các giới hạn thời gian, nhu cầu bộ nhớ…Độ ưu tiên cũng có thể được gán từ bên ngoài dựa vào các tiêu chuẩn do hệ điều hành như tầm quan trọng của tiến trình, loại người sử dụng sỡ hữu tiến trình…

Giải thuật điều phối với độ ưu tiên có thể theo nguyên tắc độc quyền hay không độc quyền. Khi một tiến trình được đưa vào danh sách các tiến trình sẵn sàng, độ ưu tiên của nó được so sánh với độ ưu tiên của tiến trình hiện hành đang xử lý. Giải thuật điều phối với độ ưu tiên và không độc quyền sẽ thu hồi CPU từ tiến trình hiện hành để cấp phát cho tiến trình mới nếu độ ưu tiên của tiến trình này cao hơn tiến trình hiện hành. Một giải thuật độc quyền sẽ chỉ đơn giản chèn tiến trình mới vào danh sách sẵn sàng, và tiến trình hiện hành vẫn tiếp tục xử lý hết thời gian dành cho nó.

- **Ví dụ:** (độ ưu tiên 1 > độ ưu tiên 2> độ ưu tiên 3)

Tiến trình	Thời điểm vào RL	Độ ưu tiên	Thời gian xử lý
P1	0	3	24
P2	1	1	3
P3	2	2	3

Sử dụng thuật giải độc quyền, thứ tự cấp phát CPU như sau :

P1	P2	P3
0	‘24	27 30

Sử dụng thuật giải không độc quyền, thứ tự cấp phát CPU như sau :

P1	P2	P3	P1
0	‘1	4	7 30

- **Thảo luận** : *Tình trạng 'đói CPU' (starvation) là một vấn đề chính yếu của các giải thuật sử dụng độ ưu tiên. Các giải thuật này có thể để các tiến trình có độ ưu tiên thấp chờ đợi CPU vô hạn ! Để ngăn cản các tiến trình có độ ưu tiên cao chiếm dụng CPU vô thời hạn, bộ điều phối sẽ giảm dần độ ưu tiên của các tiến trình này sau mỗi ngắt đồng hồ. Nếu độ ưu tiên của tiến trình này giảm xuống thấp hơn tiến trình có độ ưu tiên cao thứ nhì, sẽ xảy ra sự chuyển đổi quyền sử dụng CPU.Quá trình này gọi là sự 'lão hóa' (aging) tiến trình.*

3.2.3.4. Chiến lược công việc ngắn nhất (Shortest-job-first SJF)

- **Nguyên tắc** : Đây là một **trường hợp đặc biệt của giải thuật điều phối với độ ưu tiên**. Trong giải thuật này, **độ ưu tiên *p* được gán cho mỗi tiến trình là nghịch đảo của thời gian xử lý *t* mà tiến trình yêu cầu : *p = 1/t***. Khi CPU được tự do, nó sẽ được cấp phát cho tiến trình yêu cầu ít thời gian nhất để kết thúc- tiến trình ngắn nhất. Giải thuật này cũng có thể độc quyền hay không độc quyền. Sự chọn lựa xảy ra khi có một tiến trình mới được đưa vào danh sách sẵn sàng trong khi một tiến trình khác đang xử lý. Tiến trình mới có thể sỡ hữu một yêu cầu thời gian sử dụng CPU cho lần tiếp theo (CPU-burst) ngắn hơn thời gian còn lại mà tiến trình hiện hành cần xử lý. Giải thuật SJF không độc quyền sẽ dừng hoạt động của tiến trình hiện hành, trong khi giải thuật độc quyền sẽ cho phép tiến trình hiện hành tiếp tục xử lý.

- **Ví dụ:**

Tiến trình	Thời điểm vào RL	Thời gian xử lý
P1	0	6
P2	1	8
P3	2	4
P4	3	2

Sử dụng thuật giải SJF độc quyền, thứ tự cấp phát CPU như sau:

P1	P4	P3	P2
0	6	8	12 20

Sử dụng thuật giải SJF không độc quyền, thứ tự cấp phát CPU như sau:

P1	P4	P1	P3	P2
0	3	5	8	12 20

- **Thảo luận** : *Giải thuật này cho phép đạt được thời gian chờ trung bình cực tiểu. Khó khăn thực sự của giải thuật SJF là không thể biết được thời gian yêu cầu xử lý còn lại của tiến trình ? Chỉ có thể dự đoán giá trị này theo cách tiếp cận sau :*

gọi tn là độ dài của thời gian xử lý lần thứ n, □ $_{n+1}$ là giá trị dự đoán cho lần xử lý tiếp theo. Với hy vọng giá trị dự đoán sẽ gần giống với các giá trị trước đó, có thể sử dụng công thức:

□ $_{n+1}$ = □ *tn* + *(1-*□ *)*□ $_{n}$

Trong công thức này,tn chứa đựng thông tin gần nhất ; □ $_{n}$ *chứa đựng các thông tin quá khứ được tích lũy. Tham số* □ *(0* □ □ □ *1) kiểm soát trọng số của hiện tại gần hay quá khứ ảnh hưởng đến công thức dự đón.*

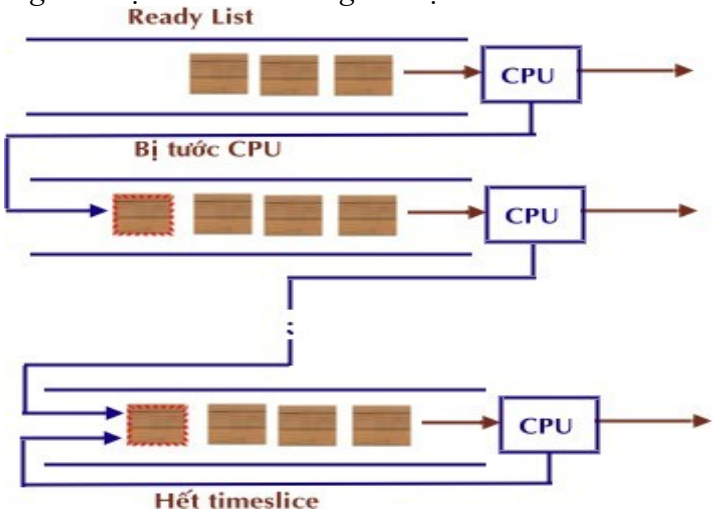
3.2.3.5. Chiến lược điều phối với nhiều mức độ ưu tiên

- **Nguyên tắc** : Ý tưởng chính của giải thuật là **phân lớp các tiến trình tùy theo độ ưu tiên của chúng để có cách thức điều phối thích hợp cho từng nhóm**. **Danh sách sẵn sàng được phân tách thành các danh sách riêng biệt theo cấp độ ưu tiên**, ***mỗi danh sách bao gồm các tiến trình có cùng độ ưu tiên và được áp dụng một giải thuật điều phối thích hợp để điều phối***. Ngoài ra, còn có một giải thuật điều phối giữa các nhóm, thường giải thuật này là giải thuật không độc quyền và sử dụng độ ưu tiên cố định.Một tiến trình thuộc về danh sách ở cấp ưu tiên *i* sẽ chỉ được cấp phát CPU khi các danh sách ở cấp ưu tiên lớn hơn *i* đã trống.

Hình 3.2.3.5-1. Điều phối nhiều cấp ưu tiên

- **Thảo luận** : *Thông thường, một tiến trình sẽ được gán vĩnh viễn với một danh sách ở cấp ưu tiên i khi nó được đưa vào hệ thống. Các tiến trình không di chuyển giữa các danh sách. Cách tổ chức này sẽ làm giảm chi phí điều phối, nhưng lại thiếu linh động và có thể dẫn đến tình trạng 'đói CPU' cho các tiến trình thuộc về những danh sách có độ ưu tiên thấp. Do vậy có thể xây dựng giải thuật điều phối nhiều cấp ưu tiên và xoay vòng. Giải thuật này sẽ chuyển dần một tiến trình từ danh sách có độ ưu tiên cao xuống danh sách có độ ưu tiên thấp hơn sau mỗi lần sử dụng CPU. Cũng vậy, một tiến trình chờ quá lâu trong các danh sách có độ ưu tiên thấp cũng có thể được chuyển dần lên các danh sách có độ ưu tiên cao hơn. Khi xây dựng một giải thuật điều phối nhiều cấp ưu tiên và xoay vòng cần quyếtđịnh các tham số :*
  - *Số lượng các cấp ưu tiên*
  - *Giải thuật điều phối cho từng danh sách ứng với một cấp ưu tiên.*
  - *Phương pháp xác định thời điểm di chuyển một tiến trình lên danh sách có độ ưu tiên cao hơn.*
  - *Phương pháp xác định thời điểm di chuyển một tiến trình lên danh sách có độ ưu tiên thấp hơn.*

- *Phương pháp sử dụng để xác định một tiến trình mới được đưa vào hệ thống sẽ thuộc danh sách ứng với độ tiên nào.*



Hình 3.2.3.5-2. Điều phối Multilevel Feedback

3.2.3.6. Chiến lược lập lịch Xổ số (Lottery)

- **Nguyên tắc** : Ý tưởng chính của giải thuật là phát hành một số vé số và phân phối cho các tiến trình trong hệ thống. Khi đến thời điểm ra quyết định điều phối, **sẽ tiến hành chọn 1 vé "trúng giải", tiến trình nào sỡ hữu vé này sẽ được nhận CPU(chọn ngẫu nhiên)**
- **Thảo luận** : *Giải thuật Lottery cung cấp một giải pháp đơn giản nhưng bảo đảm tính công bằng cho thuật toán điều phối với chi phí thấp để cập nhật độ ưu tiên cho các tiến trình* :

CHƯƠNG 4. TRUYỀN THÔNG VÀ ĐỒNG BỘ TIẾN TRÌNH

4.1. LIÊN LẠC TIẾN TRÌNH

4.1.1. Nhu cầu liên lạc tiến trình

Trong môi trường đa chương, một tiến trình không đơn độc trong hệ thống , mà có thể ảnh hưởng đến các tiến trình khác , hoặc bị các tiến trình khác tác động. Nói cách khác, các tiến trình là những thực thể độc lập , nhưng chúng vẫn có nhu cầu liên lạc với nhau để :

- **Chia sẻ thông tin**: nhiều tiến trình có thể cùng quan tâm đến những dữ liệu nào đó, do vậy hệ điều hành cần cung cấp một môi trường cho phép sự truy cập đồng thời đến các dữ liệu chung.
- **Hợp tác hoàn thành tác vụ**: đôi khi để đạt được một sự xử lý nhanh chóng, người ta phân chia một tác vụ thành các công việc nhỏ có thể tiến hành song song. Thường thì các công việc nhỏ này cần hợp tác với nhau để cùng hoàn thành tác vụ ban đầu, ví dụ dữ liệu kết xuất của tiến trình này lại là dữ liệu nhập cho tiến trình khác …Trong các trường hợp đó, hệ điều hành cần cung cấp cơ chế để các tiến trình có thể trao đổi thông tin với nhau.

4.1.2. Các vấn đề nảy sinh trong việc liên lạc tiến trình

Do mỗi tiến trình sỡ hữu một không gian địa chỉ riêng biệt, nên các tiến trình không thể liên lạc trực tiếp dễ dàng mà phải nhờ vào các cơ chế do hệ điều hành cung cấp. Khi cung cấp cơ chế liên lạc cho các tiến trình, hệ điều hành thường phải tìm giải pháp cho các vấn đề chính yếu sau :

- *Liên kết tường minh hay tiềm ẩn (explicit naming/implicit naming)* : tiến trình có cần phải biết tiến trình nào đang trao đổi hay chia sẻ thông tin với nó ? Mối liên kết được gọi là tường minh khi được thiết lập rõ ràng , trực tiếp giữa các tiến trình, và là tiềm ẩn khi các tiến trình liên lạc với nhau thông qua một qui ước ngầm nào đó.
- *Liên lạc theo chế độ đồng bộ hay không đồng bộ (blocking / non-blocking)*: khi một tiến trình trao đổi thông tin với một tiến trình khác, các tiến trình có cần phải đợi cho thao tác liên lạc hoàn tất rồi mới tiếp tục các xử lý khác ? Các tiến trình liên lạc theo cơ chế đồng bộ sẽ chờ nhau hoàn tất việc liên lạc, còn các tiến trình liên lạc theo cơ chế nonblocking thì không.
- *Liên lạc giữa các tiến trình trong hệ thống tập trung và hệ thống phân tán*: cơ chế liên lạc giữa các tiến trình trong cùng một máy tính có sự khác biệt với việc liên lạc giữa các tiến trình giữa những máy tính khác nhau?

Hầu hết các hệ điều hành đưa ra nhiều cơ chế liên lạc khác nhau, mỗi cơ chế có những đặc tính riêng, và thích hợp trong một hoàn cảnh chuyên biệt.

4.2. Các Cơ Chế Thông Tin Liên lạc

4.2.1. Tín hiệu (Signal)

Giới thiệu: Tín hiệu là một cơ chế phần mềm tương tự như các ngắt cứng tác động đến các tiến trình. Một tín hiệu được sử dụng để thông báo cho tiến trình về một sự kiện nào đó xảy ra. Có nhiều tín hiệu được định nghĩa, mỗi một tín hiệu có một ý nghĩa tương ứng với một sự kiện đặc trưng.

Ví dụ : Một số tín hiệu của UNIX

Tín hiệu	Mô tả
SIGINT	Người dùng nhấn phím DEL để ngắt xử lý tiến trình
SIGQUIT	Yêu cầu thoát xử lý
SIGILL	Tiến trình xử lý một chỉ thị bất hợp lệ
SIGKILL	Yêu cầu kết thúc một tiến trình
SIGFPT	Lỗi floating – point xảy ra (chia cho 0)
SIGPIPE	Tiến trình ghi dữ liệu vào pipe mà không có reader
SIGSEGV	Tiến trình truy xuất đến một địa chỉ bất hợp lệ
SIGCLD	Tiến trình con kết thúc
SIGUSR1	Tín hiệu 1 do người dùng định nghĩa
SIGUSR2	Tín hiệu 2 do người dùng định nghĩa

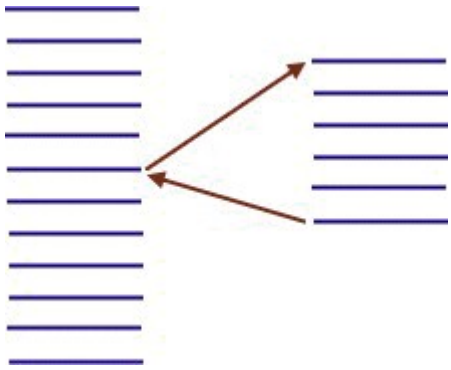
Mỗi tiến trình sỡ hữu một bảng biểu diễn các tín hiệu khác nhau. Với mỗi tín hiệu sẽ có tương ứng một trình xử lý tín hiệu *(signal handler)* qui định các xử lý của tiến trình khi nhận được tín hiệu tương ứng.

Các tín hiệu được gởi đi bởi :

- Phần cứng (ví dụ lỗi do các phép tính số học)
- Hạt nhân hệ điều hành gởi đến một tiến trình (ví dụ lưu ý tiến trình khi có một thiết bị nhập/xuất tự do).
- Một tiến trình gởi đến một tiến trình khác (ví dụ tiến trình cha yêu cầu một tiến trình con kết thúc)
- Người dùng (ví dụ nhấn phím Ctl-C để ngắt xử lý của tiến trình)

Khi một tiến trình nhận một tín hiệu, nó có thể xử sự theo một trong các cách sau :

- Bỏ qua tín hiệu
- Xử lý tín hiệu theo kiểu mặc định
- Tiếp nhận tín hiệu và xử lý theo cách đặc biệt của tiến trình.



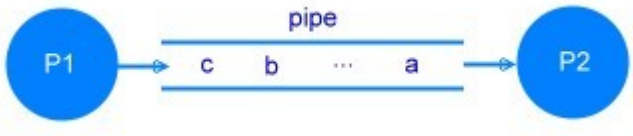
Hình 4.2.1-1. Liên lạc bằng tín hiệu

Thảo luận: Liên lạc bằng tín hiệu mang tính chất *không đồng bộ*, nghĩa là một tiến trình nhận tín hiệu không thể xác định trước thời điểm nhận tính hiệu. Hơn nữa các tiến trình không thể kiểm tra được sự kiện tương ứng với tín hiệu có thật sự xảy ra ? Cuối cùng, các tiến trình chỉ có thể thông báo cho nhau về một biến cố nào đó, mà không trao đổi dữ liệu theo cơ chế này được.

4.2.2. Pipe

Giới thiệu: Một pipe là một kênh liên lạc trực tiếp giữa hai tiến trình : dữ liệu xuất của tiến trình này được chuyển đến làm dữ liệu nhập cho tiến trình kia dưới dạng một dòng các byte.

Khi một pipe được thiết lập giữa hai tiến trình, một trong chúng sẽ ghi dữ liệu vào pipe và tiến trình kia sẽ đọc dữ liệu từ pipe. Thứ tự dữ liệu truyền qua pipe được bảo toàn theo nguyên tắc FIFO. Một pipe có kích thước giới hạn (thường là 4096 ký tự)



Hình 4.2.2-1. Liên lạc qua pipe

Một tiến trình chỉ có thể sử dụng một pipe do nó tạo ra hay kế thừa từ tiến trình cha. Hệ điều hành cung cấp các lời gọi hệ thống read/write cho các tiến trình thực hiện thao tác đọc/ghi dữ liệu trong pipe. Hệ điều hành cũng chịu trách nhiệm đồng bộ hóa việc truy xuất pipe trong các tình huống:

- Tiến trình đọc pipe sẽ bị khóa nếu pipe trống, nó sẽ phải đợi đến khi pipe có dữ liệu để truy xuất.
- Tiến trình ghi pipe sẽ bị khóa nếu pipe đầy, nó sẽ phải đợi đến khi pipe có chỗ trống để chứa dữ liệu.

Thảo luận: Liên lạc bằng pipe là một cơ chế liên lạc *một chiều (unidirectional)*, nghĩa là một tiến trình kết nối với một pipe chỉ có thể thực hiện một trong hai thao tác

đọc hoặc ghi, nhưng không thể thực hiện cả hai. Một số hệ điều hành cho phép thiết lập hai pipe giữa một cặp tiến trình để tạo liên lạc hai chiều. Trong những hệ thống đó, có nguy cơ xảy ra tình trạng *tắc nghẽn* (deadlock) : một pipe bị giới hạn về kích thước, do vậy nếu cả hai pipe nối kết hai tiến trình đều đầy(hoặc đều trống) và cả hai tiến trình đều muốn ghi (hay đọc) dữ liệu vào pipe(mỗi tiến trình ghi dữ liệu vào một pipe), chúng sẽ cùng bị khóa và chờ lẫn nhau mãi mãi !

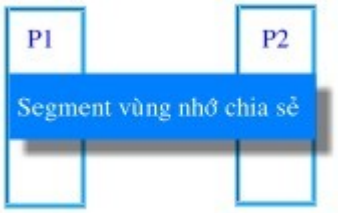
Cơ chế này cho phép truyền dữ liệu với cách thức không cấu trúc.

Ngoài ra, một giới hạn của hình thức liên lạc này là chỉ cho phép kết nối hai tiến trình có quan hệ cha-con, và trên cùng một máy tính.

4.2.3. Vùng nhớ chia sẻ

Giới thiệu: Cách tiếp cận của cơ chế này là cho nhiều tiến trình cùng truy xuất đến một vùng nhớ chung gọi là *vùng nhớ chia sẻ (shared memory)*.Không có bất kỳ hành vi truyền dữ liệu nào cần phải thực hiện ở đây, dữ liệu chỉ đơn giản được đặt vào một vùng nhớ mà nhiều tiến trình có thể cùng truy cập được.

Với phương thức này, các tiến trình chia sẻ một vùng nhớ vật lý thông qua trung gian không gian địa chỉ của chúng. Một vùng nhớ chia sẻ tồn tại độc lập với các tiến trình, và khi một tiến trình muốn truy xuất đến vùng nhớ này, tiến trình phải kết gắn vùng nhớ chung đó vào không gian địa chỉ riêng của từng tiến trình, và thao tác trên đó như một vùng nhớ riêng của mình.



Hình 4.2.3-1. Liên lạc qua vùng nhớ chia sẻ

Thảo luận:. Đây là phương pháp nhanh nhất để trao đổi dữ liệu giữa các tiến trình. Nhưng phương thức này cũng làm phát sinh các khó khăn trong việc bảo đảm sự toàn vẹn dữ liệu (*coherence*) , ví dụ : làm sao biết được dữ liệu mà một tiến trình truy xuất là dữ liệu mới nhất mà tiến trình khác đã ghi ? Làm thế nào ngăn cản hai tiến trình cùng đồng thờighi dữ liệu vào vùng nhớ chung ?…Rõ ràng vùng nhớ chia sẻ cần được bảo vệ bằng những cơ chế đồng bộ hóa thích hợp..

Một khuyết điểm của phương pháp liên lạc này là không thể áp dụng hiệu quả trong các hệ phân tán , để trao đổi thông tin giữa các máy tính khác nhau.

4.2.4. Trao đổi thông điệp (Message)

Giới thiệu: Hệ điều hành còn cung cấp một cơ chế liên lạc giữa các tiến trình không thông qua việc chia sẻ một tài nguyên chung , mà thông qua việc gởi thông điệp. Để hỗ trợ cơ chế liên lạc bằng thông điệp, hệ điều hành cung cấp các hàm IPC chuẩn (Interprocess communication), cơ bản là hai hàm:

- **Send**(message) : gởi một thông điệp
- **Receive**(message) : nhận một thông điệp

Nếu hai tiến trình P và Q muốn liên lạc với nhau, cần phải thiết lập một mối liên kết giữa hai tiến trình, sau đó P, Q sử dụng các hàm IPC thích hợp để trao đổi thông điệp,

cuối cùng khi sự liên lạc chấm dứt mối liên kết giữa hai tiến trình sẽ bị hủy. Có nhiều cách thức để thực hiện sự liên kết giữa hai tiến trình và cài đặt các theo tác send /receive tương ứng : liên lạc trực tiếp hay gián tiếp, liên lạc đồng bộ hoặc không đồng bộ , kích thước thông điệp là cố định hay không … Nếu các tiến trình liên lạc theo kiểu liên kết tường minh, các hàm Send và Receive sẽ được cài đặt với tham số :

- **Send**(destination, message) : gởi một thông điệp đến *destination*
- **Receive**(source,message) : nhận một thông điệp từ *source*

Thảo luận: Đơn vị truyền thông tin trong cơ chế trao đổi thông điệp là một thông điệp, do đó các tiến trình có thể trao đổi dữ liệu ở dạng có cấu trúc.

4.2.5. Sockets

Giới thiệu: Một socket là một thiết bị truyền thông hai chiều tương tự như tập tin, chúng ta có thể đọc hay ghi lên nó, tuy nhiên mỗi socket là một thành phần trong một mối nối nào đó giữa các máy trên mạng máy tính và các thao tác đọc/ghi chính là sự trao đổi dữ liệu giữa các ứng dụng trên nhiều máy khác nhau.

Sử dụng socket có thể mô phỏng hai phương thức liên lạc trong thực tế : liên lạc thư tín (socket đóng vai trò bưu cục) và liên lạc điện thoại (socket đóng vai trò tổng đài) .

Các thuộc tính của socket:

- Domaine: định nghĩa dạng thức địa chỉ và các nghi thức sử dụng. Có nhiều domaines, ví dụ UNIX, INTERNET, XEROX_NS, ...
- Type: định nghĩa các đặc điểm liên lạc:
  a) Sự tin cậy
  b) Sự bảo toàn thứ tự dữ liệu
  c) Lặp lại dữ liệu
  d) Chế độ nối kết
  e) Bảo toàn giới hạn thông điệp
  f) Khả năng gởi thông điệp khẩn

Để thực hiện liên lạc bằng socket, cần tiến hành các thao tác ::

- Tạo lập hay mở một socket
- Gắn kết một socket với một địa chỉ
- Liên lạc : có hai kiểu liên lạc tùy thuộc vào chế độ nối kết:

a) Liên lạc trong chế độ không liên kết **:** liên lạc theo hình thức hộp thư:

- Hai tiến trình liên lạc với nhau không kết nối trực tiếp
- Mỗi thông điệp phải kèm theo địa chỉ người nhận.
  Hình thức liên lạc này có đặc điểm được :
  - Người gởi không chắc chắn thông điệp của học được gởi đến người nhận,
  - Một thông điệp có thể được gởi nhiều lần,
  - Hai thông điệp đượ gởi theo một thứ tự nào đó có thể đến tay người nhận theo một thứ tự khác.

  Một tiến trình sau khi đã mở một socket có thể sử dụng nó để liên lạc với nhiều tiến trình khác nhau nhờ sử hai primitive *send* và *receive*.

b) Liên lạc trong chế độ nối kết**:**

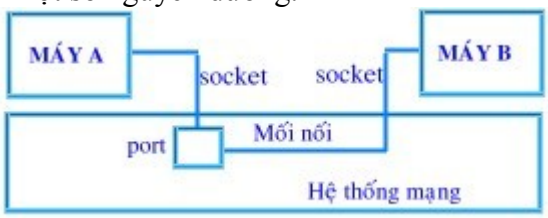
Một liên kết được thành lập giữa hai tiến trình. Trước khi mối liên kết này được thiết lập, một trong hai tiến trình phải đợi có một tiến trình khác yêu cầu kết nối.Có thể sử dụng socket để liên lạc theo mô hình client-serveur. Trong mô hình này, server sử dụng

lời gọi hệ thống listen và accept để nối kết với client, sau đó , client và server có thể trao đổi thông tin bằng cách sử dụng các primitive send và receive.

- Hủy một socket

  Ví dụ :

  Trong nghi thức truyền thông TCP, mỗi mối nối giữa hai máy tính được xác định bởi một port, khái niệm port ở đây không phải là một cổng giao tiếp trên thiết bị vật lý mà chỉ là một khái niệm logic trong cách nhìn của người lập trình, mỗi port được tương ứng với một số nguyên dương.



Hình 4.2.5-1. Các socket và port trong mối nối TCP.

Hình 4.2.5-1 minh họa một cách giao tiếp giữa hai máy tính trong nghi thức truyền thông TCP. Máy A tạo ra một socket và kết buộc (bind) socket nầy với một port X (tức là một số nguyên dương có ý nghĩa cục bộ trong máy A), trong khi đó máy B tạo một socket khác và móc vào (connect) port X trong máy A.

Thảo luận: Cơ chế socket có thể sử dụng để chuẩn hoá mối liên lạc giữa các tiến trình vốn không liên hệ với nhau, và có thể hoạt động trong những hệ thống khác nhau.

4.3. Nhu cầu đồng bộ hóa (synchronisation)

Trong một hệ thống cho phép các tiến trình liên lạc với nhau, bao giờ hệ điều hành cũng cần cung cấp kèm theo những cơ chế đồng bộ hóa để bảo đảm hoạt động của các tiến trình đồng hành không tác động sai lệch đến nhau vì các lý do sau đây:

4.3.1. Yêu cầu độc quyền truy xuất (Mutual exclusion)

Các tài nguyên trong hệ thống được phân thành hai loại: tài nguyên có thể chia sẻ cho phép nhiều tiến trình đồng thời truy xuất, và tài nguyên không thể chia sẻ chỉ chấp nhận một (hay một số lượng hạn chế) tiến trình sử dụng tại một thời điểm. Tính không thể chia sẻ của tài nguyên thường có nguồn gốc từ một trong hai nguyên nhân sau đây:

- Đặc tính cấu tạo phần cứng của tài nguyên không cho phép chia sẻ.
- Nếu nhiều tiến trình sử dụng tài nguyên đồng thời, có nguy cơ xảy ra các kết quả không dự đoán được do hoạt động của các tiến trình trên tài nguyên ảnh hưởng lẫn nhau.

Để giải quyết vấn đề, cần bảo đảm tiến trình độc quyền truy xuất tài nguyên, nghĩa là hệ thống phải kiểm soát sao cho tại một thời điểm, chỉ có một tiến trình được quyền truy xuất một tài nguyên không thể chia sẻ.

4.3.2. Yêu cầu phối hợp (Synchronization)

Nhìn chung, mối tương quan về tốc độ thực hiện của hai tiến trình trong hệ thống là không thể biết trước, vì điều này phụ thuộc vào nhiều yếu tố động như tần suất xảy ra các ngắt của từng tiến trình, thời gian tiến trình được cấp phát bộ xử lý…

Có thể nói rằng các tiến trình hoạt động không đồng bộ với nhau. Như ng có những tình huống các tiến trình cần hợp tác trong việc hoàn thành tác vụ, khi đó cần phải đồng bộ hóa hoạt động của các tiến trình , ví dụ một tiến trình chỉ có thể xử lý nếu một tiến trình khác đã kết thúc một công việc nào đó …

4.3.3. Bài toán đồng bộ hoá

4.3.3.1. Vấn đề tranh đoạt điều khiển (race condition)

Giả sử có hai tiến trình P_1 và P_2 thực hiện công việc của các kế toán, và cùng chia sẻ một vùng nhớ chung lưu trữ biến *taikhoan* phản ánh thông tin về tài khoản. Mỗi tiến trình muốn rút một khoản tiền *tienrut* từ tài khoản:

```
if (taikhoan - tienrut >=0)
   taikhoan = taikhoan - tienrut;
else
   error(« khong the rut tien ! »);
```

Giả sử trong tài khoản hiện còn 800, P_1 muốn rút 500 và P_2 muốn rút 400. Nếu xảy ra tình huống như sau :

- Sau khi đã kiểm tra điều kiện (taikhoan - tienrut >=0) và nhận kết quả là 300, P_1 hết thời gian xử lý mà hệ thống cho phép, hệ điều hành cấp phát CPU cho P_2.
- P_2 kiểm tra cùng điều kiện trên, nhận được kết quả là 400 (do P_1 vẫn chưa rút tiền) và rút 400. Giá trị của *taikhoan* được cập nhật lại là 400.
- Khi P_1 được tái kích hoạt và tiếp tục xử lý, nó sẽ không kiểm tra lại điều kiện (taikhoan - tienrut >=0)-vì đã kiểm tra trong lượt xử lý trước- mà thực hiện rút tiền. Giá trị của *taikhoan* sẽ lại được cập nhật thành -100. Tình huống lỗi xảy ra !

  Các tình huống tương tự như thế - có thể xảy ra khi có nhiều hơn hai tiến trình đọc và ghi dữ liệu trên cùng một vùng nhớ chung, và kết quả phụ thuộc vào sự điều phối tiến trình của hệ thống- được gọi là các tình huống tranh đoạt điều khiển *(race condition)* .

4.3.3.2. Miền găng (critical section)

Để ngăn chặn các tình huống lỗi có thể nảy sinh khi các tiến trình truy xuất đồng thời một tài nguyên không thể chia sẻ, cần phải áp đặt một sự truy xuất độc quyền trên tài nguyên đó : khi một tiến trình đang sử dụng tài nguyên, thì những tiến trình khác không được truy xuất đến tài nguyên.

Đoạn chương trình trong đó có khả năng xảy ra các mâu thuẫn truy xuất trên tài nguyên chung được gọi là *miền găng (critical section)*. Trong ví dụ trên, đoạn mã :

```
if (taikhoan - tienrut >=0)
taikhoan = taikhoan - tienrut;
```

của mỗi tiến trình tạo thành một miền găng.

Có thể giải quyết vấn đề mâu thuẫn truy xuất nếu có thể bảo đảm tại một thời điểm chỉ có duy nhất một tiến trình được xử lý lệnh trong miền găng.

Một phương pháp giải quyết tốt bài toán miền găng cần thõa mãn 4 điều kiện sau :

- Không có hai tiến trình cùng ở trong miền găng cùng lúc.
- Không có giả thiết nào đặt ra cho sự liên hệ về tốc độ của các tiến trình, cũng như về số lượng bộ xử lý trong hệ thống.

- Một tiến trình tạm dừng bên ngoài miền găng không được ngăn cản các tiến trình khác vào miền găng.
- Không có tiến trình nào phải chờ vô hạn để được vào miền găng.

4.4. CÁC GIẢI PHÁP ĐỒNG BỘ HOÁ

4.4.1. Giải pháp « busy waiting »

4.4.1.1. Sử dụng các biến cờ hiệu(simaphore)

Tiếp cận : các tiến trình **chia sẻ một biến chung đóng vai trò « chốt cửa » (lock)** , biến này được khởi động là 0. Một tiến trình muốn vào miền găng trước tiên phải kiểm tra giá trị của biến lock. Nếu lock = 0, tiến trình đặt lại giá trị cho lock = 1 và đi vào miền găng. Nếu lock đang nhận giá trị 1, tiến trình phải chờ bên ngoài miền găng cho đến khi lock có giá trị 0. Như vậy giá trị 0 của lock mang ý nghĩa là không có tiến trình nào đang ở trong miền găng, và lock=1 khi có một tiến trình đang ở trong miền găng.

```
while (TRUE) {
while (lock == 1); // wait
lock = 1;
critical-section ();
lock = 0;
Noncritical-section ();
}
```

Thảo luận : *Giải pháp này có thể vi phạm điều kiện thứ nhất: hai tiến trình có thể cùng ở trong miền găng tại một thời điểm. Giả sử một tiến trình nhận thấy lock = 0 và chuẩn bị vào miền găng, nhưng trước khi nó có thể đặt lại giá trị cho lock là 1, nó bị tạm dừng để một tiến trình khác hoạt động. Tiến trình thứ hai này thấy lock vẫn là 0 thì vào miền găng và đặt lại lock = 1. Sau đó tiến trình thứ nhất được tái kích hoạt, nó gán lock = 1 lần nữa rồi vaò miền găng. Như vậy tại thời điểm đó cả hai tiến trình đều ở trong miền găng*.

4.4.1.2. Sử dụng việc kiểm tra luân phiên

Tiếp cận : Đây là một giải pháp **đề nghị cho hai tiến trình**. Hai tiến trình này **sử dụng chung biến *turn*** (phản ánh phiên tiến trình nào được vào miền găng), được khởi động với giá trị 0. Nếu *turn = 0*, tiến trình A được vào miền găng. Nếu *turn = 1,* tiến trình A đi vào một vòng lặp chờ đến khi *turn* nhận giá trị *0*. Khi tiến trình A rời khỏi miền găng, nó đặt giá trị *turn* về 1 để cho phép tiến trình B đi vào miền găng.

```
while (TRUE) {
while (turn != 0); // wait
critical-section ();
turn = 1;
Noncritical-section ();
}
```

(a) Cấu trúc tiến trình A

```
while (TRUE) {
while (turn != 1); // wait
critical-section ();
turn = 0;
Noncritical-section ();
```

}

Thảo luận: Giải pháp này dựa trên việc thực hiện sự kiểm tra nghiêm nhặt đến lượt tiến trình nào được vào miền găng. Do đó nó có thể ngăn chặn được tình trạng hai tiến trình cùng vào miền găng, nhưng lại có thể vi phạm điều kiện thứ ba: một tiến trình có thể bị ngăn chặn vào miền găng bởi một tiến trình khác không ở trong miền găng. Giả sử tiến trình B ra khỏi miền găng rất nhanh chóng. Cả hai tiến trình đều ở ngoài miền găng, và turn = 0. Tiến trình A vào miền găng và ra khỏi nhanh chóng, đặt lại giá trị của turn là1, rồi lại xử lý đoạn lệnh ngoài miền găng lần nữa. Sau đó, tiến trình A lại kết thúc nhanh chóng đoạn lệnh ngoài miền găng của nó và muốn vào miền găng một lần nữa. Tuy nhiên lúc này B vẫn còn mãi xử lý đoạn lệnh ngoài miền găng của mình, và turn lại mang giá trị 1 ! Như vậy, giải pháp này không có giá trị khi có sự khác biệt lớn về tốc độ thực hiện của hai tiến trình, nó vi phạm cả điều kiện thứ hai.

4.4.1.3. Giải pháp của Peterson

Tiếp cận : Petson đưa ra một giải pháp kết hợp ý tưởng của cả hai giải pháp kể trên. Các tiến trình chia sẻ hai biến chung :

int turn; // đến phiên ai

int interesse[2]; // khởi động là FALSE

Nếu *interesse[i]* = TRUE có nghĩa là tiến trình Pi muốn vào miền găng. Khởi đầu, *interesse[0]=interesse[1]=FALSE* và giá trị của est được khởi động là 0 hay 1. Để có thể vào được miền găng, trước tiên tiến trình Pi đặt giá trị *interesse[i]*=TRUE (xác định rằng tiến trình muốn vào miền găng), sau đó đặt turn=*j* (đề nghị thử tiến trình khác vào miền găng). Nếu tiến trình Pj không quan tâm đến việc vào miền găng (*interesse[j]=FALSE*), thì Pi có thể vào miền găng, nếu không, Pi phải chờ đến khi *interesse[j]=FALSE*. Khi tiến trình Pi rời khỏi miền găng, nó đặt lại giá trị cho *interesse[i]*= FALSE.

```
while (TRUE) {
int j = 1-i; // j là tiến trình còn lại
interesse[i]= TRUE;
turn = j;
while (turn == j && interesse[j]==TRUE);
critical-section ();
interesse[i] = FALSE;
Noncritical-section ();
}
```

Thảo luận: giải pháp này ngăn chặn được tình trạng mâu thuẫn truy xuất : mỗi tiến trình Pi chỉ có thể vào miền găng khi *interesse[j]=FALSE* hoặc *turn = i*. Nếu cả hai tiến trình đều muốn vào miền găng thì *interesse[i] = interesse[j] =TRUE* nhưng giá trị của *turn* chỉ có thể hoặc là 0 hoặc là 1, do vậy chỉ có một tiến trình được vào miền găng.

4.4.1.4. Cấm ngắt:

Là giải pháp phần cứng.

Tiếp cận: cho phép tiến trình cấm tất cả các ngắt trước khi vào miền găng, và phục hồi ngắt khi ra khỏi miền găng. Khi đó, ngắt đồng hồ cũng không xảy ra, do vậy hệ thống không thể tạm dừng hoạt động của tiến trình đang xử lý để cấp phát CPU cho tiến

trình khác, nhờ đó tiến trình hiện hành yên tâm thao tác trên miền găng mà không sợ bị tiến trình nào khác tranh chấp.

Thảo luận: giải pháp này không được ưa chuộng vì rất thiếu thận trọng khi cho phép tiến trình người dùng được phép thực hiện lệnh cấm ngắt. Hơn nữa, nếu hệ thống có nhiều bộ xử lý, lệnh cấm ngắt chỉ có tác dụng trên bộ xử lý đang xử lý tiến trình, còn các tiến trình hoạt động trên các bộ xử lý khác vẫn có thể truy xuất đến miền găng !

4.4.1.5. Chỉ thị TSL (Test-and-Set)

Là giải pháp phần cứng.

Tiếp cận: đây là một giải pháp đòi hỏi sự trợ giúp của cơ chế phần cứng. Nhiều máy tính cung cấp một chỉ thị đặc biệt cho phép kiểm tra và cập nhật nội dung một vùng nhớ trong một thao tác không thể phân chia, gọi là chỉ thị *Test-and-Set Lock* (TSL) và được định nghĩa như sau:

```
Test-and-Setlock(boolean target)
{
Test-and-Setlock = target;
target = TRUE;
}
```

Nếu có hai chỉ thị TSL xử lý đồng thời (trên hai bộ xử lý khác nhau), chúng sẽ được xử lý tuần tự . Có thể cài đặt giải pháp truy xuất độc quyền với TSL bằng cách sử dụng thêm một biến lock, được khởi gán là FALSE. Tiến trình phải kiểm tra giá trị của biến lock trước khi vào miền găng, nếu lock = FALSE, tiến trình có thể vào miền găng.

```
while (TRUE) {
while (Test-and-Setlock(lock));
critical-section ();
lock = FALSE;
Noncritical-section ();
}
```

Thảo luận : cũng giống như các giải pháp phần cứng khác, chỉ thị TSL giảm nhẹ công việc lập trình để giải quyết vấn đề, nhưng lại không dễ dàng để cài đặt chỉ thị TSL sao cho được xử lý một cách không thể phân chia, nhất là trên máy với cấu hình nhiều bộ xử lý.

Tất cả các giải pháp trên đây đều phải thực hiện một vòng lặp để kiểm tra liệu nó có được phép vào miền găng, nếu điều kiện chưa cho phép, tiến trình phải chờ tiếp tục trong vòng lặp kiểm tra này. Các giải pháp buộc tiến trình phải liên tục kiểm tra điều kiện để phát hiện thời điểm thích hợp được vào miền găng như thế được gọi các giải pháp « *busy waiting* ». Lưu ý rằng việc kiểm tra như thế tiêu thụ rất nhiều thời gian sử dụng CPU, do vậy tiến trình đang chờ vẫn chiếm dụng CPU. Xu hướng giải quyết vấn đề đồng bộ hoá là nên tránh các giải pháp « *busy waiting* ».

4.4.2. Các giải pháp « SLEEP and WAKEUP »

Để loại bỏ các bất tiện của giải pháp « busy waiting », chúng ta có thể tiếp cận theo hướng cho một tiến trình chưa đủ điều kiện vào miền găng chuyển sang trạng thái blocked, từ bỏ quyền sử dụng CPU. Để thực hiện điều này, cần phải sử dụng các thủ tục do hệ điều hành cung cấp để thay đổi trạng thái tiến trình. Hai thủ tục cơ bản *SLEEP* và *WAKEUP* thường được sử dụng để phục vụ mục đích này.

SLEEP là một lời gọi hệ thống có tác dụng tạm dừng hoạt động của tiến trình (blocked) gọi nó và chờ đến khi được một tiến trình khác « đánh thức ». Lời gọi hệ thống *WAKEUP* nhận một tham số duy nhất : tiến trình sẽ được tái kích hoạt (đặt về trạng thái ready).
Ý tưởng sử dụng SLEEP và WAKEUP như sau : khi một tiến trình chưa đủ điều kiện vào miền găng, nó gọi *SLEEP* để tự khóa đến khi có một tiến trình khác gọi *WAKEUP* để giải phóng cho nó. Một tiến trình gọi *WAKEUP* khi ra khỏi miền găng để đánh thức một tiến trình đang chờ, tạo cơ hội cho tiến trình này vào miền găng :
int busy; // 1 nếu miền găng đang bị chiếm, nếu không là 0
int blocked; // đếm số lượng tiến trình đang bị khóa

```
while (TRUE) {
if (busy){
   blocked = blocked + 1;
   sleep();
}
else busy = 1;
critical-section ();

busy = 0;
if(blocked){
   wakeup(process);
   blocked = blocked - 1;
}
Noncritical-section ();
}
```

Khi sử dụng SLEEP và WAKEUP cần hết sức cẩn thận, nếu không muốn xảy ra tình trạng mâu thuẫn truy xuất trong một vài tình huống đặc biệt như sau : giả sử tiến trình A vào miền găng, và trước khi nó rời khỏi miền găng thì tiến trình B được kích hoạt. Tiến trình B thử vào miền găng nhưng nó nhận thấy A đang ở trong đó, do vậy B tăng giá trị biến *blocked* và chuẩn bị gọi *SLEEP* để tự khoá. Tuy nhiên trước khi B có thể thực hiện *SLEEP*, tiến trình A lại được tái kích hoạt và ra khỏi miền găng. Khi ra khỏi miền găng A nhận thấy có một tiến trình đang chờ (*blocked*=1) nên gọi *WAKEUP* và giảm giá trị của *blocked*. Khi đó tín hiệu *WAKEUP* sẽ lạc mất do tiến trình B chưa thật sự « ngủ » để nhận tín hiệu đánh thức !Khi tiến trình B được tiếp tục xử lý, nó mới goi *SLEEP* và tự khó vĩnh viễn !

Vấn đề ghi nhận được là tình trạng lỗi này xảy ra do việc kiểm tra tư cách vào miền găng và việc gọi SLEEP hay WAKEUP là những hành động tách biệ, có thể bị ngắt nửa chừng trong quá trình xử lý, do đó có khi tín hiệu WAKEUP gởi đến một tiến trình chưa bị khóa sẽ lạc mất.

Để tránh những tình huống tương tự, hệ điều hành cung cấp những cơ chế đồng bộ hóa dựa trên ý tưởng của chiến lược « SLEEP and WAKEUP » nhưng được xây dựng bao hàm cả phương tiện kiểm tra điều kiện vào miền găng giúp sử dụng an toàn.

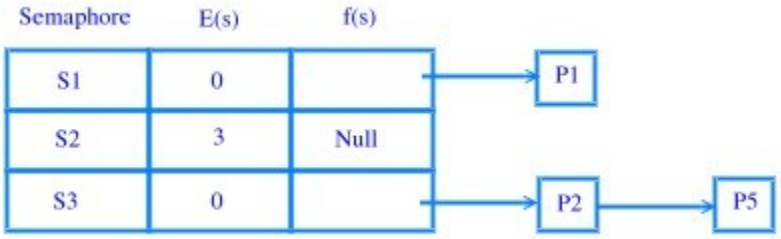
4.4.2.1. Semaphore

Tiếp cận: Được **Dijkstra** đề xuất vào 1965, một semaphore ***s*** là một *biến* có các thuộc tính sau:

- Một giá trị nguyên dương *e(s)*
- Một hàng đợi *f(s)* lưu danh sách các tiến trình đang bị khóa (chờ) trên semaphore *s*
- Chỉ có hai thao tác được định nghĩa trên semaphore

  Down(s): giảm giá trị của semaphore *s* đi 1 đơn vị nếu semaphore có trị e(s) > 0, và tiếp tục xử lý. Ngược lại, nếu e(s) □ 0, tiến trình phải chờ đến khi e(s) >0.

  Up(s): tăng giá trị của semaphore *s* lên 1 đơn vị. Nếu có một hoặc nhiều tiến trình đang chờ trên semaphore *s*, bị khóa bởi thao tác **Down**, thì hệ thống sẽ chọn một trong các tiến trình này để kết thúc thao tác **Down** và cho tiếp tục xử lý.



Hình 4.4.2.1-1. Semaphore s

Cài đặt: Gọi *p* là tiến trình thực hiện thao tác *Down(s)* hay *Up(s)*.

Down(s):

```
e(s) = e(s) - 1;
if e(s) < 0 {
        status(P)= blocked;
        enter(P,f(s));
}
```

Up(s):

```
e(s) = e(s) + 1;
if s = 0 {
        exit(Q,f(s)); //Q là tiến trình đang chờ trên s
        status (Q) = ready;
        enter(Q,ready-list);
}
```

Lưu ý cài đặt này có thể đưa đến một giá trị âm cho semaphore, khi đó trị tuyệt đối của semaphore cho biết số tiến trình đang chờ trên semaphore.

Điều quan trọng là các thao tác này cần thực hiện một cách không bị phân chia, không bị ngắt nữa chừng, có nghĩa là không một tiến trình nào được phép truy xuất đến semaphore nếu tiến trình đang thao tác trên semaphore này chưa kết thúc xử lý hay chuyển sang trạng thái blocked.

Sử dụng: có thể dùng semaphore để giải quyết vấn đề truy xuất độc quyền hay tổ chức phối hợp giữa các tiến trình.

Tổ chức truy xuất độc quyền với Semaphores: khái niệm *semaphore* cho phép bảo đảm nhiều tiến trình cùng truy xuất đến miền găng mà không có sự mâu thuẫn truy

xuất. *n* tiến trình cùng sử dụng một semaphore s, e(s) được khởi gán là 1. Để thực hiện đồng bộ hóa, tất cả các tiến trình cần phải áp dụng cùng cấu trúc chương trình sau đây:

```
while (TRUE) {
        Down(s)
        critical-section ();
        Up(s)
        Noncritical-section ();
}
```

Tổ chức đồng bộ hóa với Semaphores: với semaphore có thể đồng bộ hóa hoạt động của hai tiến trình trong tình huống một tiến trình phải đợi một tiến trình khác hoàn tất thao tác nào đó mới có thể bắt đầu hay tiếp tục xử lý. Hai tiến trình chia sẻ một semaphore s, khởi gán e(s) là 0. Cả hai tiến trình có cấu trúc như sau:

```
P1:
while (TRUE) {
        job1();
        Up(s); //đánh thức P2
}
P2:
        while (TRUE) {
        Down(s); // chờ P1
        job2();
}
```

Thảo luận : Nhờ có thực hiện một các không thể phân chia, semaphore đã giải quyết được vấn đề tín hiệu "đánh thức" bị thất lạc. Tuy nhiên, nếu lập trình viên vô tình đặt các primitive Down và Up sai vị trí, thứ tự trong chương trình, thì tiến trình có thể bị khóa vĩnh viễn.

Ví dụ :

```
while (TRUE) {
        Down(s)
        critical-section                                    ();
        Noncritical-section ();
}
```

Tiến trình trên đây quên gọi Up(s), và kết quả là khi ra khỏi miền găng nó sẽ không cho tiến trình khác vào miền găng !

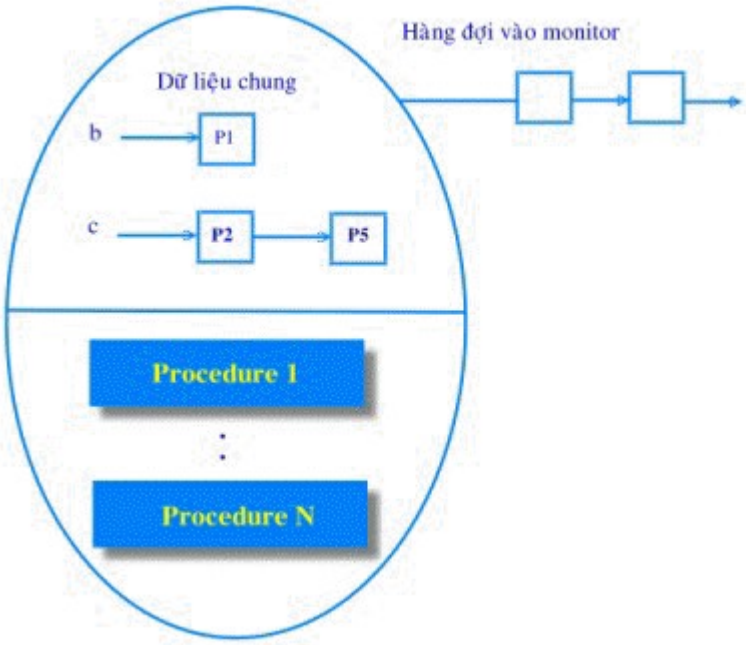
Vì thế việc sử dụng đúng cách semaphore để đồng bộ hóa phụ thuộc hoàn toàn vào lập trình viên và đòi hỏi lập trình viên phải hết sức thận trọng.

4.4.2.2. Monitors

Tiếp cận: Để có thể dễ viết đúng các chương trình đồng bộ hóa hơn, Hoare(1974) và Brinch & Hansen (1975) đã đề nghị một cơ chế cao hơn được cung cấp bởi ngôn ngữ lập trình , là *monitor*. Monitor là một cấu trúc đặc biệt bao gồm các thủ tục, các biến và cấu trúc dữ liệu có các thuộc tính sau :

- Các biến và cấu trúc dữ liệu bên trong monitor chỉ có thể được thao tác bởi các thủ tục định nghĩa bên trong monitor đó. (*encapsulation*).

- Tại một thời điểm, chỉ có một tiến trình duy nhất được hoạt động bên trong một monitor (*mutual exclusive*).
- Trong một monitor, có thể định nghĩa các *biến điều kiện* và hai thao tác kèm theo là **Wait** và **Signal** như sau : gọi ***c*** là biến điều kiện được định nghĩa trong monitor:
    - **Wait(c):** chuyển trạng thái tiến trình gọi sang blocked , và đặt tiến trình này vào hàng đợi trên biến điều kiện ***c***.
    - **Signal(c)**: nếu có một tiến trình đang bị khóa trong hàng đợi của ***c,*** tái kích hoạt tiến trình đó, và tiến trình gọi sẽ rời khỏi monitor.



Hình 4.4.2.2-1. Monitor và các biến điều kiện

Cài đặt : trình biên dịch chịu trách nhiệm thực hiện việc truy xuất độc quyền đến dữ liệu trong monitor. Để thực hiện điều này, một semaphore nhị phân thường được sử dụng. Mỗi monitor có một hàng đợi toàn cục lưu các tiến trình đang chờ được vào monitor, ngoài ra, mỗi biến điều kiện ***c*** cũng gắn với một hàng đợi ***f(c)*** và hai thao tác trên đó được định nghĩa như sau:

Wait(c) :
status(P)= **blocked**;
enter(P,f(c));
Signal(c) :
if (f(c) != NULL){
    exit(Q,f(c)); //Q là tiến trình chờ trên c
    statusQ) = **ready**;
    enter(Q,ready-list);
}

Sử dụng: Với mỗi nhóm tài nguyên cần chia sẻ, có thể định nghĩa một monitor trong đó đặc tả tất cả các thao tác trên tài nguyên này với một số điều kiện nào đó.:

monitor <tên monitor >

```
condition <danh sách các biến điều kiện>;
<déclaration de variables>;

 procedure Action1();
 {

 }
            ....
 procedure Actionn();
 {

 }
end monitor;
```

Các tiến trình muốn sử dụng tài nguyên chung này chỉ có thể thao tác thông qua các thủ tục bên trong monitor được gắn kết với tài nguyên:

```
while (TRUE) {
        Noncritical-section ();
        <monitor>.Actioni; //critical-section();
        Noncritical-section ();
}
```

Thảo luận: Với monitor, việc truy xuất độc quyền được bảo đảm bởi trình biên dịch mà không do lập trình viên, do vậy nguy cơ thực hiện đồng bộ hóa sai giảm rất nhiều. Tuy nhiên giải pháp monitor đòi hỏi phải có một ngôn ngữ lập trình định nghĩa khái niệm monitor, và các ngôn ngữ như thế chưa có nhiều.

4.4.2.3. Trao đổi thông điệp

Tiếp cận: giải pháp này dựa trên cơ sở trao đổi thông điệp với hai primitive Send và Receive để thực hiện sự đồng bộ hóa:

- **Send(destination, message):** gởi một thông điệp đến một tiến trình hay gởi vào hộp thư.
- **Receive(source,message)**: nhận một thông điệp thừ một tiến trình hay từ bất kỳ một tiến trình nào, tiến trình gọi sẽ chờ nếu không có thông điệp nào để nhận.

Sử dụng: Có nhiều cách thức để thực hiện việc truy xuất độc quyền bằng cơ chế trao đổi thông điệp. Đây là một mô hình đơn giản: một tiến trình kiểm soát việc sử dụng tài nguyên và nhiều tiến trình khác yêu cầu tài nguyên này. Tiến trình có yêu cầu tài nguyên sẽ gởi một thông điệp đến tiến trình kiểm soát và sau đó chuyển sang trạng thái blocked cho đến khi nhận được một thông điệp chấp nhận cho truy xuất từ tiến trình kiểm soát tài nguyên.Khi sử dụng xong tài nguyên , tiến trình gởi một thông điệp khác đến tiến trình kiểm soát để báo kết thúc truy xuất. Về phần tiến trình kiểm soát , khi nhận được thông điệp yêu cầu tài nguyên, nó sẽ chờ đến khi tài nguyên sẵn sàng để cấp phát thì gởi một thông điệp đến tiến trình đang bị khóa trên tài nguyên đó để đánh thức tiến trình này.

```
while (TRUE) {
Send(process controler, request message);
Receive(process controler, accept message);
```

```
critical-section ();
Send(process controler, end message);
Noncritical-section ();
}
```

Thảo luận: Các primitive semaphore và monitor có thể giải quyết được vấn đề truy xuất độc quyền trên các máy tính có một hoặc nhiều bộ xử lý chia sẻ một vùng nhớ chung. Nhưng các primitive không hữu dụng trong các hệ thống phân tán, khi mà mỗi bộ xử lý sỡ hữu một bộ nhớ riêng biệt và liên lạc thông qua mạng. Trong những hệ thống phân tán như thế, cơ chế trao đổi thông điệp tỏ ra hữu hiệu và được dùng để giải quyết bài toán đồng bộ hóa.

CHƯƠNG 5. VẤN ĐỀ KHOÁ CHẾT (DEADLOCK)

5.1. Mô hình hệ thống

Một hệ thống chứa số tài nguyên hữu hạn được phân bổ giữa nhiều quá trình cạnh tranh. Các tài nguyên này được phân chia thành nhiều loại, mỗi loại chứa một số thể hiện xác định. Không gian bộ nhớ, các chu kỳ CPU và các thiết bị nhập/xuất (như máy in, đĩa từ) là những thí dụ về loại tài nguyên. Nếu hệ thống có hai CPUs, thì loại tài nguyên CPU có hai thể hiện. Tương tự, loại tài nguyên máy in có thể có năm thể hiện.

Nếu một quá trình yêu cầu một thể hiện của loại tài nguyên thì việc cấp phát bất cứ thể hiện nào của loại tài nguyên này sẽ thoả mãn yêu cầu. Nếu nó không có thì các thể hiện là không xác định và các lớp loại tài nguyên sẽ không được định nghĩa hợp lý. Thí dụ, một hệ thống có thể có hai máy in. Hai loại máy in này có thể được định nghĩa trong cùng lớp loại tài nguyên nếu không có quá trình nào quan tâm máy nào in ra dữ liệu. Tuy nhiên, nếu một máy in ở tầng 9 và máy in khác ở tầng trệt thì người dùng ở tầng 9 không thể xem hai máy in là tương tự nhau và lớp tài nguyên riêng rẻ cần được định nghĩa cho mỗi máy in.

Một quá trình phải yêu cầu một tài nguyên trước khi sử dụng nó, và phải giải phóng sau khi sử dụng nó. Một quá trình có thể yêu cầu nhiều tài nguyên như nó được

yêu cầu để thực hiện tác vụ được gán của nó. Chú ý, số tài nguyên được yêu cầu không vượt quá số lượng tổng cộng tài nguyên sẵn có trong hệ thống. Nói cách khác, một quá trình không thể yêu cầu ba máy in nếu hệ thống chỉ có hai.

Dưới chế độ điều hành thông thường, một quá trình có thể sử dụng một tài nguyên chỉ trong thứ tự sau:

01) **Yêu cầu**: nếu yêu cầu không thể được gán tức thì (thí dụ, tài nguyên đang được dùng bởi quá trình khác) thì quá trình đang yêu cầu phải chờ cho tới khi nó có thể nhận được tài nguyên.

12) **Sử dụng**: quá trình có thể điều hành tài nguyên (thí dụ, nếu tài nguyên là máy in, quá trình có thể in máy in)

23) **Giải phóng**: quá trình giải phóng tài nguyên.

Yêu cầu và giải phóng tài nguyên là các lời gọi hệ thống. Thí dụ như yêu cầu và giải phóng thiết bị, mở và đóng tập tin, cấp phát và giải phóng bộ nhớ. Yêu cầu và giải phóng các tài nguyên khác có thể đạt được thông qua thao tác chờ wait và báo hiệu signal. Do đó, cho mỗi trường hợp sử dụng, hệ điều hành kiểm tra để đảm bảo rằng quá trình sử dụng yêu cầu và được cấp phát tài nguyên. Một bảng hệ thống ghi nhận mỗi quá trình giải phóng hay được cấp phát tài nguyên. Nếu một quá trình yêu cầu tài nguyên mà tài nguyên đó hiện được cấp phát cho một quá trình khác, nó có thể được thêm vào hàng đợi để chờ tài nguyên này.

Một tập hợp quá trình trong trạng thái deadlock khi mỗi quá trình trong tập hợp này chờ sự kiện mà có thể được tạo ra chỉ bởi quá trình khác trong tập hợp. Những sự kiện mà chúng ta quan tâm chủ yếu ở đây là nhận và giải phóng tài nguyên. Các tài nguyên có thể là tài nguyên vật lý (thí dụ, máy in, đĩa từ, không gian bộ nhớ và chu kỳ

CPU) hay tài nguyên luận lý (thí dụ, tập tin, semaphores, monitors). Tuy nhiên, các loại khác của sự kiện có thể dẫn đến deadlock.

Để minh hoạ trạng thái deadlock, chúng ta xét hệ thống với ba ổ đĩa từ. Giả sử mỗi quá trình giữ các một ổ đĩa từ này. Bây giờ, nếu mỗi quá trình yêu cầu một ổ đĩa từ khác thì ba quá trình sẽ ở trong trạng thái deadlock. Mỗi quá trình đang chờ một sự kiện "ổ đĩa từ được giải phóng" mà có thể được gây ra chỉ bởi một trong những quá trình đang chờ. Thí dụ này minh hoạ deadlock liên quan đến cùng loại tài nguyên.

Deadlock cũng liên quan nhiều loại tài nguyên khác nhau. Thí dụ, xét một hệ thống với một máy in và một ổ đĩa từ. Giả sử, quá trình Pi đang giữ ổ đĩa từ và quá trình Pj đang giữ máy in. Nếu Pi yêu cầu máy in và Pj yêu cầu ổ đĩa từ thì deadlock xảy ra.

Một người lập trình đang phát triển những ứng dụng đa luồng phải quan tâm đặc biệt tới vấn đề này: Các chương trình đa luồng là ứng cử viên cho vấn đề deadlock vì nhiều luồng có thể cạnh tranh trên tài nguyên được chia sẻ.

5.2. Đặc điểm deadlock

Trong một deadlock, các quá trình không bao giờ hoàn thành việc thực thi và các tài nguyên hệ thống bị buộc chặt, ngăn chặn các quá trình khác bắt đầu. Trước khi chúng ta thảo luận các phương pháp khác nhau giải quyết vấn đề deadlock, chúng ta sẽ mô tả các đặc điểm mà deadlock mô tả.

5.2.1. Những điều kiện cần thiết gây ra deadlock

Trường hợp deadlock có thể phát sinh nếu bốn điều kiện sau xảy ra cùng một lúc trong hệ thống:

- Loại trừ hỗ tương: ít nhất một tài nguyên phải được giữ trong chế độ không chia sẻ; nghĩa là, chỉ một quá trình tại cùng một thời điểm có thể sử dụng tài nguyên. Nếu một quá trình khác yêu cầu tài nguyên đó, quá trình yêu cầu phải tạm dừng cho đến khi tài nguyên được giải phóng.
- Giữ và chờ cấp thêm tài nguyên: quá trình phải đang giữ ít nhất một tài nguyên và đang chờ để nhận tài nguyên thêm mà hiện đang được giữ bởi quá trình khác.
- Không đòi lại tài nguyên từ quá trình đang giữ chúng: Các tài nguyên không thể bị đòi lại; nghĩa là, tài nguyên có thể được giải phóng chỉ tự ý bởi quá trình đang giữ nó, sau khi quá trình đó hoàn thành tác vụ.
- Tồn tại chu trình trong đồ thị cấp phát tài nguyên: một tập hợp các quá trình {P0, P1,…,Pn} đang chờ mà trong đó P0 đang chờ một tài nguyên được giữ bởi P1, P1 đang chờ tài nguyên đang giữ bởi P2,…,Pn-1 đang chờ tài nguyên đang được giữ bởi quá trình P0.

1

Chúng ta nhấn mạnh rằng tất cả bốn điều kiện phải cùng phát sinh để deadlock xảy ra. Điều kiện chờ đợi ch trình đưa đến điều kiện giữ-và-chờ vì thế bốn điều kiện không hoàn toàn độc lập.

5.2.2. Đồ thị cấp phát tài nguyên

Deadlock có thể mô tả chính xác hơn bằng cách hiển thị đồ thị có hướng gọi là đồ thị cấp phát tài nguyên hệ thống. Đồ thị này chứa một tập các đỉnh V và tập hợp các cạnh

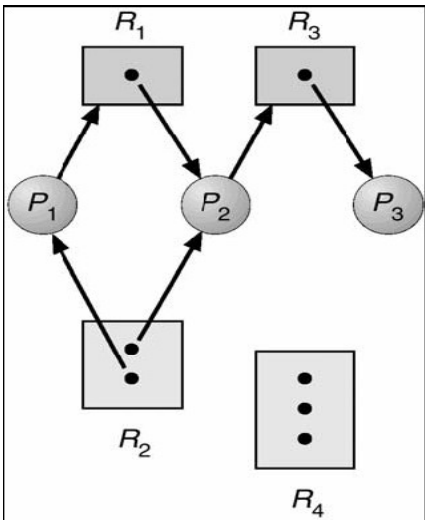
E. Một tập các đỉnh V được chia làm hai loại nút P = {P1, P2,…,Pn} là tập hợp các quá trình hoạt động trong hệ thống, và R = {R1, R2, ..., Rm} là tập hợp chứa tất cả các loại tài nguyên trong hệ thống.

Một cạnh có hướng từ quá trình Pi tới loại tài nguyên Rj được ký hiệu Pi →Rj; nó biểu thị rằng quá trình Pi đã yêu cầu loại tài nguyên Rj và hiện đang chờ loại tài nguyên đó. Một cạnh có hướng từ loại tài nguyên Rj tới quá trình Pi được hiển thị bởi Rj → Pi; nó hiển thị rằng thể hiện của loại tài nguyên Rj đã được cấp phát tới quá trình Pi. Một cạnh có hướng Pi → Rj được gọi là cạnh yêu cầu; một cạnh có hướng Rj → Pi được gọi là cạnh gán.

Bằng hình tượng, chúng ta hiển thị mỗi quá trình Pi là một hình tròn, và mỗi loại tài nguyên Rj là hình chữ nhật. Vì loại tài nguyên Rj có thể có nhiều hơn một thể hiện, chúng ta hiển thị mỗi thể hiện là một chấm nằm trong hình vuông. Chú ý rằng một cạnh yêu cầu trỏ tới chỉ một hình vuông Rj, trái lại một cạnh gán cũng phải gán tới một trong các dấu chấm trong hình vuông.

Khi quá trình Pi yêu cầu một thể hiện của loại tài nguyên Rj, một cạnh yêu cầu được chèn vào đồ thị cấp phát tài nguyên. Khi yêu cầu này có thể được đáp ứng, cạnh yêu cầu lập tức được truyền tới cạnh gán. Khi quá trình không còn cần truy xuất tới tài nguyên, nó giải phóng tài nguyên, và khi đó dẫn đến cạnh gán bị xoá.

Đồ thị cấp phát tài nguyên được hiển thị trong hình VI-1 dưới đây mô tả trường hợp sau:



Đồ thị cấp phát tài nguyên

- Các tập P, R, và E:
  1○ P = {P_1, P_2, P_3}
  2○ R = {R_1, R_2, R_3, R_4}
  3○ E = {$P_1 \rightarrow R_1$, $P_2 \rightarrow R_3$, $R_1 \rightarrow P_2$, $R_2 \rightarrow P_2$, $R_3 \rightarrow P_3$}
- Các thể hiện tài nguyên
  4○ Một thể hiện của tài nguyên loại R_1
  5○ Hai thể hiện của tài nguyên loại R_2
  6○ Một thể hiện của tài nguyên loại R_3

7○ Một thể hiện của tài nguyên loại R_4

- Trạng thái quá trình

8○ Quá trình P_1 đang giữ một thể hiện của loại tài nguyên R_2 và đang chờ một thể hiện của loại tài nguyên R_1.

9○ Quá trình P_2 đang giữ một thể hiện của loại tài nguyên R_1 và R_2 và đang chờ một thể hiện của loại tài nguyên R_3.

10○ Quá trình P_3 đang giữ một thể hiện của R_3

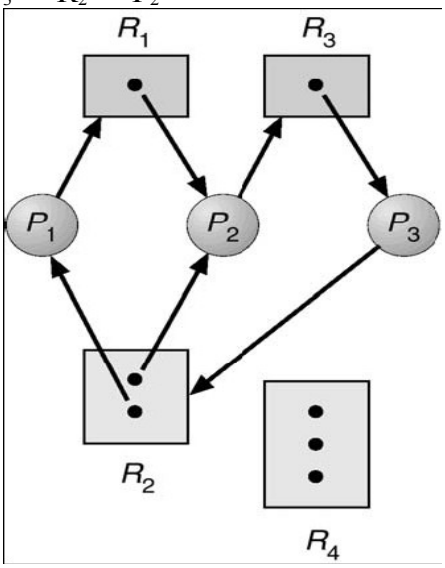
Đồ thị cấp phát tài nguyên hiển thị rằng, nếu đồ thị không chứa chu trình, thì không có quá trình nào trong hệ thống bị deadlock. Nếu đồ thị có chứa chu trình, thì deadlock có thể tồn tại. Nếu mỗi loại tài nguyên có chính xác một thể hiện, thì một chu trình ngụ ý rằng một deadlock xảy ra. Nếu một chu trình bao gồm chỉ một tập hợp các loại tài nguyên, mỗi loại tài nguyên chỉ có một thể hiện thì deadlock xảy ra. Mỗi quá trình chứa trong chu trình bị deadlock. Trong trường hợp này, một chu trình trong đồ thị là điều kiện cần và đủ để tồn tại deadlock.

Nếu mỗi loại tài nguyên có nhiều thể hiện thì chu trình không ngụ ý deadlock xảy. Trong trường hợp này, một chu trình trong đồ thị là điều kiện cần nhưng chưa đủ để tồn tại deadlock.

Để hiển thị khái niệm này, chúng ta xem lại đồ thị ở hình VII-1 ở trên. Giả sử quá trình P3 yêu cầu một thể hiện của loại tài nguyên R2. Vì không có thể hiện tài nguyên hiện có, một cạnh yêu cầu P3 → R2 được thêm vào đồ thị (hình VI-2). Tại thời điểm này, hai chu trình nhỏ tồn tại trong hệ thống:

$P_1 \rightarrow R_1 \rightarrow P_2 \rightarrow R_3 \rightarrow P_3 \rightarrow R_2 \rightarrow P_1$

$P_2 \rightarrow R_3 \rightarrow P_3 \rightarrow R_2 \rightarrow P_2$

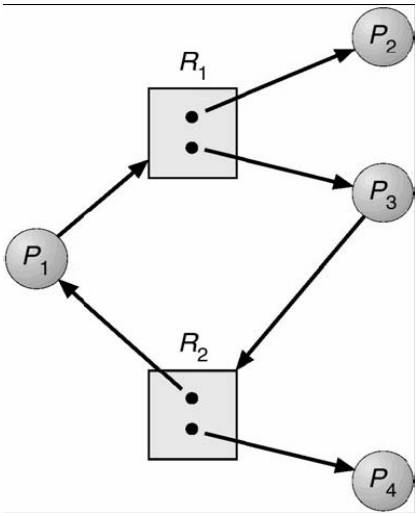


Đồ thị cấp phát tài nguyên với deadlock

Quá trình P1, P2, và P3 bị deadlock. Quá trình P3 đang chờ tài nguyên R3, hiện được giữ bởi quá trình P2. Hay nói cách khác, quá trình P3 đang chờ quá trình P1 hay P2 giải phóng tài nguyên R2. Ngoài ra, quá trình P1 đang chờ quá trình P2 giải phóng tài nguyên R1.

Bây giờ xem xét đồ thị cấp phát tài nguyên trong hình dưới đây. Trong thí dụ này, chúng ta cũng có một chu kỳ

P1 → R1 → P3 → R2 → P1



Đồ thị cấp phát tài nguyên có chu trình nhưng không bị deadlock

Tuy nhiên, không có deadlock. Chú ý rằng quá trình P4 có thể giải phóng thể hiện của loại tài nguyên R2. Tài nguyên đó có thể được cấp phát tới P3 sau đó, chu trình sẽ không còn.

Tóm lại, nếu đồ thị cấp phát tài nguyên không có chu trình thì hệ thống không có trạng thái deadlock. Ngoài ra, nếu có chu trình thì có thể có hoặc không trạng thái deadlock. Nhận xét này là quan trọng khi chúng ta giải quyết vấn đề deadlock.

5.3. Các phương pháp xử lý deadlock

Phần lớn, chúng ta có thể giải quyết vấn đề deadlock theo một trong ba cách:

- Chúng ta có thể sử dụng một giao thức để ngăn chặn hay tránh deadlocks, đảm bảo rằng hệ thống sẽ không bao giờ đi vào trạng thái deadlock
- Chúng ta có thể cho phép hệ thống đi vào trạng thái deadlock, phát hiện nó và phục hồi.
- Chúng ta có thể bỏ qua hoàn toàn vấn đề này và giả vờ deadlock không bao giờ xảy ra trong hệ thống. Giải pháp này được dùng trong nhiều hệ điều hành, kể cả UNIX.
- Chúng ta sẽ tìm hiểu vắn tắt mỗi phương pháp. Sau đó, chúng ta sẽ trình bày các giải thuật một cách chi tiết trong các phần sau đây.

Để đảm bảo deadlock không bao giờ xảy ra, hệ thống có thể dùng kế hoạch ngăn chặn hay tránh deadlock. Ngăn chặn deadlock là một tập hợp các phương pháp để đảm bảo rằng ít nhất một điều kiện cần (trong phần VI.4.1) không thể xảy ra. Các phương pháp này ngăn chặn deadlocks bằng cách ràng buộc yêu cầu về tài nguyên được thực hiện như thế nào. Chúng ta thảo luận phương pháp này trong phần sau.

Ngược lại, tránh deadlock yêu cầu hệ điều hành cung cấp những thông tin bổ sung tập trung vào loại tài nguyên nào một quá trình sẽ yêu cầu và sử dụng trong thời gian sống của nó. Với những kiến thức bổ sung này, chúng ta có thể quyết định đối với mỗi

yêu cầu quá trình nên chờ hay không. Để quyết định yêu cầu hiện tại có thể được thoả mãn hay phải bị trì hoãn, hệ thống phải xem xét tài nguyên hiện có, tài nguyên hiện cấp phát cho mỗi quá trình, và các yêu cầu và giải phóng tương lai của mỗi quá trình.

Nếu một hệ thống không dùng giải thuật ngăn chặn hay tránh deadlock thì trường hợp deadlock có thể xảy ra. Trong môi trường này, hệ thống có thể cung cấp một giải thuật để xem xét trạng thái của hệ thống để xác định deadlock có xảy ra hay không và giải thuật phục hồi từ deadlock.

Nếu hệ thống không đảm bảo rằng deadlock sẽ không bao giờ xảy ra và cũng không cung cấp một cơ chế để phát hiện và phục hồi deadlock thì có thể dẫn đến trường hợp hệ thống ở trong trạng thái deadlock. Trong trường hợp này, deadlock không được phát hiện sẽ làm giảm năng lực hệ thống vì tài nguyên đang được giữ bởi những quá trình mà chúng không thể thực thi, đi vào trạng thái deadlock. Cuối cùng, hệ thống sẽ dừng các chức năng và cần được khởi động lại bằng thủ công.

Mặc dù phương pháp này dường như không là tiếp cận khả thi đối với vấn đề deadlock nhưng nó được dùng trong một số hệ điều hành. Trong nhiều hệ thống, deadlock xảy ra không thường xuyên; do đó phương pháp này là rẻ hơn chi phí cho phương pháp ngăn chặn deadlock, tránh deadlock, hay phát hiện và phục hồi deadlock mà chúng phải được sử dụng liên tục. Trong một số trường hợp, hệ thống ở trong trạng thái cô đặc nhưng không ở trạng thái deadlock. Như thí dụ, xem xét một quá trình thời thực chạy tại độ ưu tiên cao nhất (hay bất cứ quá trình đang chạy trên bộ

5.4. Ngăn chặn deadlock

Để deadlock xảy ra, một trong bốn điều kiện cần phải xảy ra. Bằng cách đảm bảo ít nhất một trong bốn điều kiện này không thể xảy ra, chúng ta có thể ngăn chặn việc xảy ra của deadlock. Chúng ta tìm hiểu tỷ mỹ tiếp cận này bằng cách xem xét mỗi điều kiện cần riêng rẻ nhau.

5.4.1. Loại trừ hỗ tương

Điều kiện loại trừ hỗ tương phải giữ cho tài nguyên không chia sẻ. Thí dụ, một máy in không thể được chia sẻ cùng lúc bởi nhiều quá trình. Ngược lại, các tài nguyên có thể chia sẻ không đòi hỏi truy xuất loại trừ hỗ tương và do đó không thể liên quan đến deadlock. Những tập tin chỉ đọc là một thí dụ tốt cho tài nguyên có thể chia sẻ. Nếu nhiều quá trình cố gắng mở một tập tin chỉ đọc tại cùng một thời điểm thì chúng có thể được gán truy xuất cùng lúc tập tin. Một quá trình không bao giờ yêu cầu chờ tài nguyên có thể chia sẻ. Tuy nhiên, thường chúng ta không thể ngăn chặn deadlock bằng cách từ chối điều kiện loại trừ hỗ tương: một số tài nguyên về thực chất không thể chia sẻ.

5.4.2. Giữ và chờ cấp thêm tài nguyên

Để đảm bảo điều kiện giữ-và-chờ cấp thêm tài nguyên không bao giờ xảy ra trong hệ thống, chúng ta phải đảm bảo rằng bất cứ khi nào một quá trình yêu cầu tài nguyên, nó không giữ bất cứ tài nguyên nào khác. Một giao thức có thể được dùng là đòi hỏi mỗi quá trình yêu cầu và được cấp phát tất cả tài nguyên trước khi nó bắt đầu thực thi. Chúng ta có thể cài đặt sự cung cấp này bằng cách yêu cầu các lời gọi hệ thống yêu cầu tài nguyên cho một quá trình trước tất cả các lời gọi hệ thống khác.

Một giao thức khác cho phép một quá trình yêu cầu tài nguyên chỉ khi quá trình này không có tài nguyên nào. Một quá trình có thể yêu cầu một số tài nguyên và dùng chúng. Tuy nhiên, trước khi nó có thể yêu cầu bất kỳ tài nguyên bổ sung nào, nó phải giải phóng tất cả tài nguyên mà nó hiện đang được cấp phát.

Để hiển thị sự khác nhau giữa hai giao thức, chúng ta xét một quá trình chép dữ liệu từ băng từ tới tập tin đĩa, sắp xếp tập tin đĩa và sau đó in kết quả ra máy in. Nếu tất cả tài nguyên phải được yêu cầu cùng một lúc thì khởi đầu quá trình phải yêu cầu băng từ, tập tin đĩa và máy in. Nó sẽ giữ máy in trong toàn thời gian thực thi của nó mặc dù nó cần máy in chỉ ở giai đoạn cuối.

Phương pháp thứ hai cho phép quá trình yêu cầu ban đầu chỉ băng từ và tập tin đĩa. Nó chép dữ liệu từ băng từ tới đĩa, rồi giải phóng cả hai băng từ và đĩa. Sau đó, quá trình phải yêu cầu lại tập tin đĩa và máy in. Sau đó, chép tập tin đĩa tới máy in, nó giải phóng hai tài nguyên này và kết thúc.

Hai giao thức này có hai nhược điểm chủ yếu. Thứ nhất, việc sử dụng tài nguyên có thể chậm vì nhiều tài nguyên có thể được cấp nhưng không được sử dụng trong thời gian dài. Trong thí dụ được cho, chúng ta có thể giải phóng băng từ và tập tin đĩa, sau đó yêu cầu lại tập tin đĩa và máy in chỉ nếu chúng ta đảm bảo rằng dữ liệu của chúng ta sẽ vẫn còn trên tập tin đĩa. Nếu chúng ta không thể đảm bảo rằng dữ liệu vẫn còn tập tin đĩa thì chúng ta phải yêu cầu tất cả tài nguyên tại thời điểm bắt đầu cho cả hai giao thức. Thứ hai, đói tài nguyên là có thể. Một quá trình cần nhiều tài nguyên phổ biến có thể phải đợi vô hạn định vì một tài nguyên mà nó cần luôn được cấp phát cho quá trình khác.

5.4.3. Không đòi lại tài nguyên từ quá trình đang giữ chúng

Điều kiện cần thứ ba là không đòi lại những tài nguyên đã được cấp phát rồi. Để đảm bảo điều kiện này không xảy ra, chúng ta có thể dùng giao thức sau. Nếu một quá trình đang giữ một số tài nguyên và yêu cầu tài nguyên khác mà không được cấp phát tức thì tới nó (nghĩa là, quá trình phải chờ) thì tất cả tài nguyên hiện đang giữ được đòi lại. Nói cách khác, những tài nguyên này được giải phóng hoàn toàn. Những tài nguyên bị đòi lại được thêm tới danh sách các tài nguyên mà quá trình đang chờ. Quá trình sẽ được khởi động lại chỉ khi nó có thể nhận lại tài nguyên cũ của nó cũng như các tài nguyên mới mà nó đang yêu cầu.

Có một sự chọn lựa khác, nếu một quá trình yêu cầu một số tài nguyên, đầu tiên chúng ta kiểm tra chúng có sẵn không. Nếu tài nguyên có sẵn, chúng ta cấp phát chúng. Nếu tài nguyên không có sẵn, chúng ta kiểm tra chúng có được cấp phát tới một số quá trình khác đang chờ tài nguyên bổ sung. Nếu đúng như thế, chúng ta lấy lại tài nguyên mong muốn đó từ quá trình đang đợi và cấp chúng cho quá trình đang yêu cầu. Nếu tài nguyên không sẵn có hay được giữ bởi một quá trình đang đợi, quá trình đang yêu cầu phải chờ. Trong khi nó đang chờ, một số tài nguyên của nó có thể được đòi lại chỉ nếu quá trình khác yêu cầu chúng. Một quá trình có thể được khởi động lại chỉ khi nó được cấp các tài nguyên mới mà nó đang yêu cầu và phục hồi bất cứ tài nguyên nào đã bị lấy lại trong khi nó đang chờ.

Giao thức này thường được áp dụng tới tài nguyên mà trạng thái của nó có thể được lưu lại dễ dàng và phục hồi lại sau đó, như các thanh ghi CPU và không gian bộ nhớ. Nó thường không thể được áp dụng cho các tài nguyên như máy in và băng từ.

5.4.4. Tồn tại chu trình trong đồ thị cấp phát tài nguyên

Điều kiện thứ tư và cũng là điều kiện cuối cùng cho deadlock là điều kiện tồn tại chu trình trong đồ thị cấp phát tài nguyên. Một cách để đảm bảo rằng điều kiện này không bao giờ xảy ra là áp đặt toàn bộ thứ tự của tất cả loại tài nguyên và đòi hỏi mỗi quá trình trong thứ tự tăng của số lượng.

Gọi R = {R1, R2, …, Rm} là tập hợp loại tài nguyên. Chúng ta gán mỗi loại tài nguyên một số nguyên duy nhất, cho phép chúng ta so sánh hai tài nguyên và xác định tài nguyên này có đứng trước tài nguyên khác hay không trong thứ tự của chúng ta. Thông thường, chúng ta định nghĩa hàm ánh xạ một-một F: R → N, ở đây N là tập hợp các số tự nhiên. Thí dụ, nếu tập hợp các loại tài nguyên R gồm các ổ băng từ, ổ đĩa và máy in thì hàm F có thể được định nghĩa như sau:

F(ổ băng từ) = 1,
F(đĩa từ) = 5,
F(máy in) = 12.

Bây giờ chúng ta xem giao thức sau để ngăn chặn deadlock: mỗi quá trình có thể yêu cầu tài nguyên chỉ trong thứ tự tăng của số lượng. Nghĩa là, một quá trình ban đầu có thể yêu cầu bất cứ số lượng thể hiện của một loại tài nguyên Ri. Sau đó, một quá trình có thể yêu cầu các thể hiện của loại tài nguyên Rj nếu và chỉ nếu F(Rj) > F(Ri). Nếu một số thể hiện của cùng loại tài nguyên được yêu cầu, thì một yêu cầu cho tất cả thể hiện phải được cấp phát. Thí dụ, sử dụng hàm được định nghĩa trước đó, một quá trình muốn dùng ổ băng từ và máy in tại cùng một lúc trước tiên phải yêu cầu ổ băng từ và sau đó yêu cầu máy in.

Nói một cách khác, chúng ta yêu cầu rằng, bất cứ khi nào một quá trình yêu cầu một thể hiện của loại tài nguyên Rj, nó giải phóng bất cứ tài nguyên Ri sao cho F(Ri) ≥ F(Rj).

Nếu có hai giao thức được dùng thì điều kiện tồn tại chu trình không thể xảy ra. Chúng ta có thể giải thích điều này bằng cách cho rằng tồn tại chu trình trong đồ thị cấp phát tài nguyên tồn tại. Gọi tập hợp các quá trình chứa tồn tại chu trình trong đồ thị cấp phát tài nguyên là {P0, P1, … , Pn}, ở đây Pi đang chờ một tài nguyên Ri, mà Ri được giữ bởi quá trình Pi+1. Vì sau đó quá trình Pi+1 đang giữ tài nguyên Ri trong khi yêu cầu tài nguyên Ri+1, nên chúng ta có F(Ri) < F(Ri+1) cho tất cả i. Nhưng điều kiện này có nghĩa là F(R0) < F(R1) < …< F(Rn) < F(R0). Bằng qui tắc bắt cầu F(R0) < F(R0), điều này là không thể. Do đó, không thể có chờ chu trình.

Chú ý rằng hàm F nên được định nghĩa dựa theo thứ tự tự nhiên của việc sử dụng tài nguyên trong hệ thống. Thí dụ, vì ổ băng từ thường được yêu cầu trước máy in nên có thể hợp lý để định nghĩa F(ổ băng từ) < F(máy in).

5.5. Tránh deadlock

Các giải thuật ngăn chặn deadlock, được thảo luận ở VII-6, ngăn chặn deadlock bằng cách hạn chế cách các yêu cầu có thể được thực hiện. Các ngăn chặn đảm bảo rằng ít nhất một trong những điều kiện cần cho deadlock không thể xảy ra. Do đó, deadlock không thể xảy ra. Tuy nhiên, các tác dụng phụ có thể ngăn chặn deadlock bởi phương pháp này là việc sử dụng thiết bị chậm và thông lượng hệ thống bị giảm.

Một phương pháp khác để tránh deadlock là yêu cầu thông tin bổ sung về cách tài nguyên được yêu cầu. Thí dụ, trong một hệ thống với một ổ băng từ và một máy in, chúng ta có thể bảo rằng quá trình P sẽ yêu cầu ổ băng từ trước và sau đó máy in trước khi giải phóng cả hai tài nguyên. Trái lại, quá trình Q sẽ yêu cầu máy in trước và sau đó ổ băng từ. Với kiến thức về thứ tự hoàn thành của yêu cầu và giải phóng cho mỗi quá trình, chúng ta có thể quyết định cho mỗi yêu cầu của quá trình sẽ chờ hay không. Mỗi yêu cầu đòi hỏi hệ thống xem tài nguyên hiện có, tài nguyên hiện được cấp tới mỗi quá trình, và các yêu cầu và giải phóng tương lai của mỗi quá trình, để yêu cầu của quá trình hiện tại có thể được thoả mãn hay phải chờ để tránh khả năng xảy ra deadlock.

Các giải thuật khác nhau có sự khác nhau về lượng và loại thông tin được yêu cầu. Mô hình đơn giản và hữu ích nhất yêu cầu mỗi quá trình khai báo số lớn nhất tài nguyên của mỗi loại mà nó cần. Thông tin trước về số lượng tối đa tài nguyên của mỗi loại được yêu cầu cho mỗi quá trình, có thể xây dựng một giải thuật đảm bảo hệ thống sẽ không bao giờ đi vào trạng thái deadlock. Đây là giải thuật định nghĩa tiếp cận tránh deadlock. Giải thuật tránh deadlock tự xem xét trạng thái cấp phát tài nguyên để đảm bảo điều kiện tồn tại chu trình trong đồ thị cấp phát tài nguyên có thể không bao giờ xảy ra. Trạng thái cấp phát tài nguyên được định nghĩa bởi số tài nguyên sẵn dùng và tài nguyên được cấp phát và số yêu cầu tối đa của các quá trình.

5.5.1. Trạng thái an toàn

Một trạng thái là an toàn nếu hệ thống có thể cấp phát các tài nguyên tới mỗi quá trình trong một vài thứ tự và vẫn tránh deadlock. Hay nói cách khác, một hệ thống ở trong trạng thái an toàn chỉ nếu ở đó tồn tại một thứ tự an toàn. Thứ tự của các quá trình <P1, P2, …, Pn> là một thứ tự an toàn cho trạng thái cấp phát hiện hành nếu đối

với mỗi thứ tự Pi, các tài nguyên mà Pi yêu cầu vẫn có thể được thoả mãn bởi tài nguyên hiện có cộng với các tài nguyên được giữ bởi tất cả Pj, với j<i. Trong trường hợp này, nếu những tài nguyên mà quá trình Pi yêu cầu không sẵn dùng tức thì thì Pi có thể chờ cho đến khi tất cả Pj hoàn thành. Khi chúng hoàn thành, Pi có thể đạt được tất cả những tài nguyên nó cần, hoàn thành các tác vụ được gán, trả về những tài nguyên được cấp phát cho nó và kết thúc. Khi Pi kết thúc, Pi+1 có thể đạt được các tài nguyên nó cần,... Nếu không có thứ tự như thế tồn tại thì trạng thái hệ thống là không an toàn.

Một trạng thái an toàn không là trạng thái deadlock. Do đó, trạng thái deadlock là trạng thái không an toàn. Tuy nhiên, không phải tất cả trạng thái không an toàn là deadlock (hình VI-4). Một trạng thái không an toàn có thể dẫn đến deadlock. Với điều kiện trạng thái là an toàn, hệ điều hành có thể tránh trạng thái không an toàn (và deadlock). Trong một trạng thái không an toàn, hệ điều hành có thể ngăn chặn các quá trình từ những tài nguyên đang yêu cầu mà deadlock xảy ra: hành vi của các quá trình này điều khiển các trạng thái không an toàn.

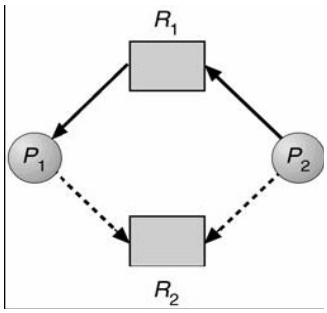
5.5.2. Giải thuật đồ thị cấp phát tài nguyên

Nếu chúng ta có một hệ thống cấp phát tài nguyên với một thể hiện của mỗi loại, một biến dạng của đồ thị cấp phát tài nguyên được định nghĩa trong phần VI.4.2 có thể được dùng để tránh deadlock.

Ngoài các cạnh yêu cầu và gán, chúng ta giới thiệu một loại cạnh mới được gọi là cạnh thỉnh cầu (claim edge). Một cạnh thỉnh cầu Pi → Rj hiển thị quá trình Pi có thể yêu

cầu tài nguyên Rj vào một thời điểm trong tương lai. Cạnh này tương tự cạnh yêu cầu về phương hướng nhưng được hiện diện bởi dấu đứt khoảng. Khi quá trình Pi yêu cầu tài nguyên Rj, cạnh thỉnh cầu Pi → Rj chuyển tới cạnh yêu cầu. Tương tự, khi một tài nguyên Rj được giải phóng bởi Pi, cạnh gán Rj → Pi được chuyển trở lại thành cạnh thỉnh cầu Pi → Rj. Chúng ta chú ý rằng các tài nguyên phải được yêu cầu trước trong hệ thống. Nghĩa là, trước khi Pi bắt đầu thực thi, tất cả các cạnh thỉnh cầu của nó phải xuất hiện trong đồ thị cấp phát tài nguyên. Chúng ta có thể giảm nhẹ điều kiện này bằng cách cho phép một cạnh Pi → Rj để được thêm tới đồ thị chỉ nếu tất cả các cạnh gắn liền với quá trình Pi là các cạnh thỉnh cầu.

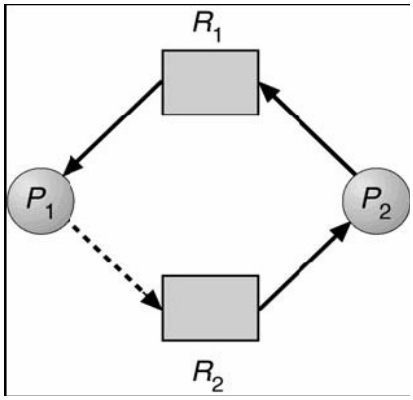
Giả sử rằng Pi yêu cầu tài nguyên Rj. Yêu cầu có thể được gán chỉ nếu chuyển cạnh yêu cầu Pi → Rj tới cạnh gán Rj→Pi không dẫn đến việc hình thành chu trình trong đồ thị cấp phát tài nguyên. Chú ý rằng chúng ta kiểm tra tính an toàn bằng cách dùng giải thuật phát hiện chu trình. Một giải thuật để phát hiện một chu trình trong đồ thị này yêu cầu một thứ tự của n2 thao tác, ở đây n là số quá trình trong hệ thống.



Đồ thị cấp phát tài nguyên để tránh deadlock

Nếu không có chu trình tồn tại, thì việc cấp phát tài nguyên sẽ để lại hệ thống trong trạng thái an toàn. Nếu chu trình được tìm thấy thì việc cấp phát sẽ đặt hệ thống trong trạng thái không an toàn. Do đó, quá trình Pi sẽ phải chờ yêu cầu của nó được thoả.

Để minh hoạ giải thuật này, chúng ta xét đồ thị cấp phát tài nguyên của hình VI-5. Giả sử rằng P2 yêu cầu R2. Mặc dù R2 hiện rảnh nhưng chúng ta không thể cấp phát nó tới P2 vì hoạt động này sẽ tạo ra chu trình trong đồ thị (Hình VI-6). Một chu trình hiển thị rằng hệ thống ở trong trạng thái không an toàn. Nếu P1 yêu cầu R2 và P2 yêu cầu R1 thì deadlock sẽ xảy ra.



Trạng thái không an toàn trong đồ thị cấp phát tài nguyên

5.5.3. Giải thuật của Banker

Giải thuật đồ thị cấp phát tài nguyên không thể áp dụng tới hệ thống cấp phát tài nguyên với nhiều thể hiện của mỗi loại tài nguyên. Giải thuật tránh deadlock mà chúng ta mô tả tiếp theo có thể áp dụng tới một hệ thống nhưng ít hiệu quả hơn cơ chế đồ thị cấp phát tài nguyên. Giải thuật này thường được gọi là giải thuật của Banker. Tên được chọn vì giải thuật này có thể được dùng trong hệ thống ngân hàng để đảm bảo ngân hàng không bao giờ cấp phát tiền mặt đang có của nó khi nó không thể thoả mãn các yêu cầu của tất cả khách hàng.

Khi một quá trình mới đưa vào hệ thống, nó phải khai báo số tối đa các thể hiện của mỗi loại tài nguyên mà nó cần. Số này có thể không vượt quá tổng số tài nguyên trong hệ thống. Khi một người dùng yêu cầu tập hợp các tài nguyên, hệ thống phải xác định việc cấp phát của các tài nguyên này sẽ để lại hệ thống ở trạng thái an toàn hay không. Nếu trạng thái hệ thống sẽ là an toàn, tài nguyên sẽ được cấp; ngược lại quá trình phải chờ cho tới khi một vài quá trình giải phóng đủ tài nguyên.

Nhiều cấu trúc dữ liệu phải được duy trì để cài đặt giải thuật Banker. Những cấu trúc dữ liệu này mã hoá trạng thái của hệ thống cấp phát tài nguyên. Gọi n là số quá trình trong hệ thống và m là số loại tài nguyên trong hệ thống. Chúng ta cần các cấu trúc dữ liệu sau:

0• **Available**: một vector có chiều dài m hiển thị số lượng tài nguyên sẵn dùng của mỗi loại. Nếu Available[j]= k, có k thể hiện của loại tài nguyên R_j sẵn dùng.

1• **Max**: một ma trận n x m định nghĩa số lượng tối đa yêu cầu của mỗi quá trình. Nếu Max[i , j] = k, thì quá trình P_i có thể yêu cầu nhiều nhất k thể hiện của loại tài nguyên R_j.

2• **Allocation**: một ma trận n x m định nghĩa số lượng tài nguyên của mỗi loại hiện được cấp tới mỗi quá trình. Nếu Allocation[i, j] = k, thì quá trình P_i hiện được cấp k thể hiện của loại tài nguyên R_j.

0• **Need**: một ma trận n x m hiển thị yêu cầu tài nguyên còn lại của mỗi quá trình. Nếu Need[i, j] = k, thì quá trình P_i có thể cần thêm k thể hiện của loại tài nguyên R_j để hoàn thành tác vụ của nó. Chú ý rằng, Need[i, j] = Max[i, j] – Allocation [i, j].

Cấu trúc dữ liệu này biến đổi theo thời gian về kích thước và giá trị Để đơn giản việc trình bày của giải thuật Banker, chúng ta thiết lập vài ký hiệu. Gọi X và Y là các vector có chiều dài n. Chúng ta nói rằng $X \leq Y$ nếu và chỉ nếu $X[i] \leq Y[i]$ cho tất cả i = 1, 2, …, n. Thí dụ, nếu X = (1, 7, 3, 2) và Y = (0, 3, 2, 1) thì $Y \leq X$, $Y < X$ nếu $Y \leq X$ và $Y \neq X$.

Chúng ta có thể xem xét mỗi dòng trong ma trận Allocation và Need như là những vectors và tham chiếu tới chúng như Allocationi và Needi tương ứng. Vector Allocationi xác định tài nguyên hiện được cấp phát tới quá trình Pi; vector Needi xác định các tài nguyên bổ sung mà quá trình Pi có thể vẫn yêu cầu để hoàn thành tác vụ của nó.

5.5.3.1. Giải thuật an toàn

Giải thuật để xác định hệ thống ở trạng thái an toàn hay không có thể được mô tả như sau:

11) Gọi Work và Finish là các vector có chiều dài m và n tương ứng. Khởi tạo Work:=Available và Finish[i]:=false cho i = 1, 2, …,n.

22) Tìm i thỏa:

3a) Finish[i] = false

4b) Need $_i \leq$ Work.

5Nếu không có i nào thỏa, di chuyển tới bước 4

63) Work:=Work + Allocation $_i$

Finish[i] := true

Di chuyển về bước 2.

4) Nếu Finish[i] = true cho tất cả i, thì hệ thống đang ở trạng thái an toàn.

2Giải thuật này có thể yêu cầu độ phức tạp mxn^2 thao tác để quyết định trạng thái là an toàn hay không.

5.5.3.2. Giải thuật yêu cầu tài nguyên

Cho $Request_i$ là vector yêu cầu cho quá trình P_i. Nếu $Request_i[j] = k$, thì quá trình P_i muốn k thể hiện của loại tài nguyên R_j. Khi một yêu cầu tài nguyên được thực hiện bởi quá trình P_i, thì các hoạt động sau được thực hiện:

11) Nếu $Request_i \leq Need_i$, di chuyển tới bước 2. Ngược lại, phát sinh một điều kiện lỗi vì quá trình vượt quá yêu cầu tối đa của nó.

22) Nếu $Request_i \leq$ Available, di chuyển tới bước 3. Ngược lại, P_i phải chờ vì tài nguyên không sẵn có.

33) Giả sử hệ thống cấp phát các tài nguyên được yêu cầu tới quá trình P_i bằng cách thay đổi trạng thái sau:

$$Available := Available - Request_i;$$
$$Allocation_i := Allocation_i + Request_i;$$
$$Need_i := Need_i - Request_i;$$

Nếu kết quả trạng thái cấp phát tài nguyên là an toàn, thì giao dịch được hoàn thành và quá trình P_i được cấp phát tài nguyên của nó. Tuy nhiên, nếu trạng thái mới là không an toàn, thì P_i phải chờ $Request_i$ và trạng thái cấp phát tài nguyên cũ được phục hồi.

CHƯƠNG 6: QUẢN LÝ BỘ NHỚ TRONG

Chương này nhằm đề cập những vấn đề cần quan tâm khi thiết kế module quản lý bộ nhớ của Hệ điều hàn . Một số mô hình tổ chức bộ nhớ cũng được giới thiệu và phân tích ưu, khuyết điểm để các bạn có thể hiểu được cách thức cấp phát và thu hồi bộ nhớ diễn ra như thế nào

Bộ nhớ chính là thiết bị lưu trữ duy nhất thông qua đó CPU có thể trao đổi thông tin với môi trường ngoài, do vậy nhu cầu tổ chức, quản lý bộ nhớ là một trong những nhiệm vụ trọng tâm hàng đầu của hệ điều hành . Bộ nhớ chính được tổ chức như một mảng một chiều các từ nhớ (word), mỗi từ nhớ có một địa chỉ . Việc trao đổi thông tin với môi trường ngoài được thực hiện thông qua các thao tác đọc hoặc ghi dữ liệu vào một địa chỉ cụ thể nào đó trong bộ nhớ.

Hầu hết các hệ điều hành hiện đại đều cho phép chế độ đa nhiệm nhằm nâng cao hiệu suất sử dụng CPU. Tuy nhiên kỹ thuật này lại làm nảy sinh nhu cầu chia sẻ bộ nhớ giữa các tiến trình khác nhau . Vấn đề nằm ở chỗ : « *bộ nhớ thì hữu hạn và các yêu cầu bộ nhớ thì vô hạn* ».

Hệ điều hành chịu trách nhiệm cấp phát vùng nhớ cho các tiến trình có yêu cầu. Để thực hiện tốt nhiệm vụ này, hệ điều hành cần phải xem xét nhiều khía cạnh :

- Sự tương ứng giữa địa chỉ *logic* và địa chỉ vật lý (*physic*) : làm cách nào để chuyển đổi một địa chỉ tượng trưng (symbolic) trong chương trình thành một địa chỉ thực trong bộ nhớ chính?
- Quản lý bộ nhớ vật lý: làm cách nào để mở rộng bộ nhớ có sẵn nhằm lưu trữ được nhiều tiến trình đồng thời?
- Chia sẻ thông tin: làm thế nào để cho phép hai tiến trình có thể chia sẻ thông tin trong bộ nhớ?
- Bảo vệ: làm thế nào để ngăn chặn các tiến trình xâm phạm đến vùng nhớ được cấp phát cho tiến trình khác?

Các giải pháp quản lý bộ nhớ phụ thuộc rất nhiều vào đặc tính phần cứng và trải qua nhiều giai đoạn cải tiến để trở thành những giảp pháp khá thỏa đáng như hiện nay.

6.1. Ngữ cảnh liên kết bộ nhớ

Thông thường, một chương trình được lưu trữ trên đĩa như một tập tin nhị phân có thể xử lý. Để thực hiện chương trình, cần nạp chương trình vaò bộ nhớ chính, tạo lập tiến trình tương ứng để xử lý .

Hàng đợi nhập hệ thống là tập hợp các chương trình trên đĩa đang chờ được nạp vào bộ nhớ để tiến hành xử lý.

Các địa chỉ trong chương trình nguồn là địa chỉ tượng trưng , vì thế, một chương trình phải trải qua nhiều giai đoạn xử lý để chuyển đổi các địa chỉ này thành các địa chỉ tuyệt đối trong bộ nhớ chính.

Có thể thực hiện kết buộc các chỉ thị và dữ liệu với các địa chỉ bộ nhớ vào một trong những thời điểm sau :

- **Thời điểm biên dịch**: nếu tại thời điểm biên dịch, có thể biết vị trí mà tiến trình sẽ thường trú trong bộ nhớ, trình biên dịch có thể phát sinh ngay mã với các địa chỉ tuyệt đối. Tuy nhiên, nếu về sau có sự thay đổi vị trí thường trú lúc đầu của chương trình, cần phải biên dịch lại chương trình.
- **Thời điểm nạp**: nếu tại thời điểm biên dịch, chưa thể biết vị trí mà tiến trình sẽ thường trú trong bộ nhớ, trình biên dịch cần phát sinh mã tương đối (translatable). Sự liên kết địa chỉ được trì hoãn đến thời điểm chương trình được nạp vào bộ nhớ, lúc này các địa chỉ tương đối sẽ được chuyển thành địa chỉ tuyệt đối do đã biết vị trí bắt đầu lưu trữ tiến trình. Khi có sự thay đổi vị trí lưu trữ, chỉ cần nạp lại chương trình để tính toán lại các địa chỉ tuyệt đối, mà không cần biên dịch lại.
- **Thời điểm xử lý**: nếu có nhu cầu di chuyển tiến trình từ vùng nhớ này sang vùng nhớ khác trong quá trình tiến trình xử lý, thì thời điểm kết buộc địa chỉ phải trì hoãn đến tận thời điểm xử lý. Để thực hiện kết buộc địa chỉ vào thời điểm xử lý, cần sử dụng cơ chế phần cứng đặc biệt.

6.2. Không gian địa chỉ lôgic và không gian địa chỉ vật lý

Một trong những hướng tiếp cận trung tâm nhằm tổ chức quản lý bộ nhớ một cách hiệu qủa là đưa ra khái niệm không gian địa chỉ được xây dựng trên không gian nhớ vật lý, việc tách rời hai không gian này giúp hệ điều hành dễ dàng xây dựng các cơ chế và chiến lược quản lý bộ nhớ hữu hiệu :

- *Địa chỉ logic* – còn gọi là *địa chỉ ảo* , là tất cả các địa chỉ do bộ xử lý tạo ra.
- *Địa chỉ vật lý* - là địa chỉ thực tế mà trình quản lý bộ nhớ nhìn thấy và thao tác.
- *Không gian địa chỉ* – là tập hợp tất cả các địa chỉ ảo phát sinh bởi một chương trình.
- *Không gian vật lý* – là tập hợp tất cả các địa chỉ vật lý tương ứng với các địa chỉ ảo.

Địa chỉ ảo và địa chỉ vật lý là như nhau trong phương thức kết buộc địa chỉ vào thời điểm biên dịch cũng như vào thời điểm nạp. Nhưng có sự khác biệt giữa địa chỉ ảo và địa chỉ vật lý trong phương thức kết buộc vào thời điểm xử lý.

MMU (*memory-management unit*) là một cơ chế phần cứng được sử dụng để thực hiện chuyển đổi địa chỉ ảo thành địa chỉ vật lý vào thời điểm xử lý.

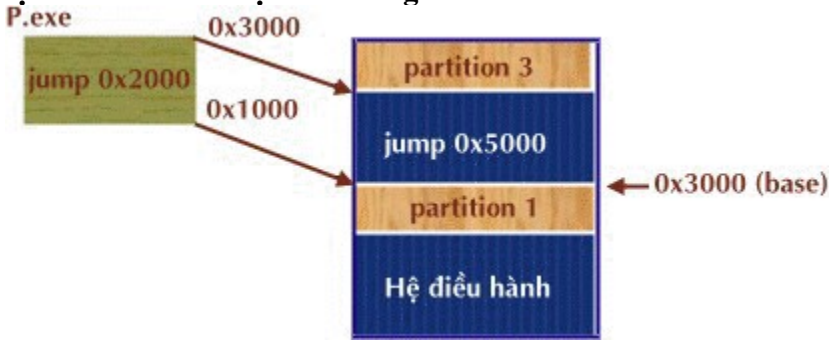
Chương trình của người sử dụng chỉ thao tác trên các địa chỉ ảo, không bao giờ nhìn thấy các địa chỉ vật lý . Địa chỉ thật sự ứng với vị trí của dữ liệu trong bô nhớ chỉ được xác định khi thực hiện truy xuất đến dữ liệu.

6.3. Cấp phát liên tục

6.3.1. Mô hình Linker_Loader

Ý tưởng : Tiến trình được nạp vào một vùng nhớ liên tục đủ lớn để chứa toàn bộ tiến trình. Tại thời điểm biên dịch các địa chỉ bên trong tiến trình vẫn là địa chỉ tương đối. Tại thời điểm nạp, Hệ điều hành sẽ trả về địa chỉ bắt đầu nạp tiến trình, và tính toán để

chuyển các địa chỉ tương đối về địa chỉ tuyệt đối trong bộ nhớ vật lý theo công thức **địa chỉ vật lý = địa chỉ bắt đầu + địa chỉ tương đối**.

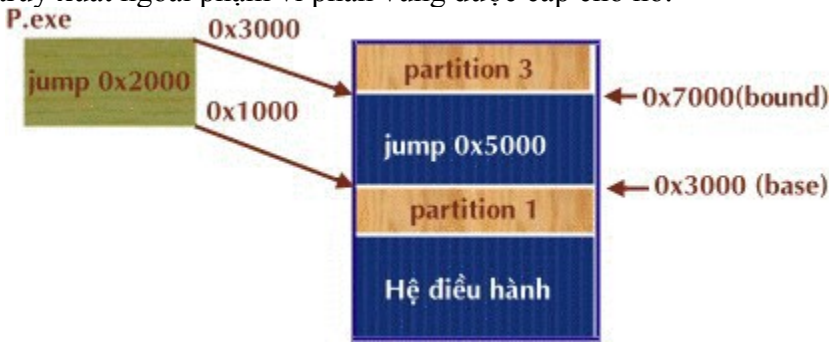


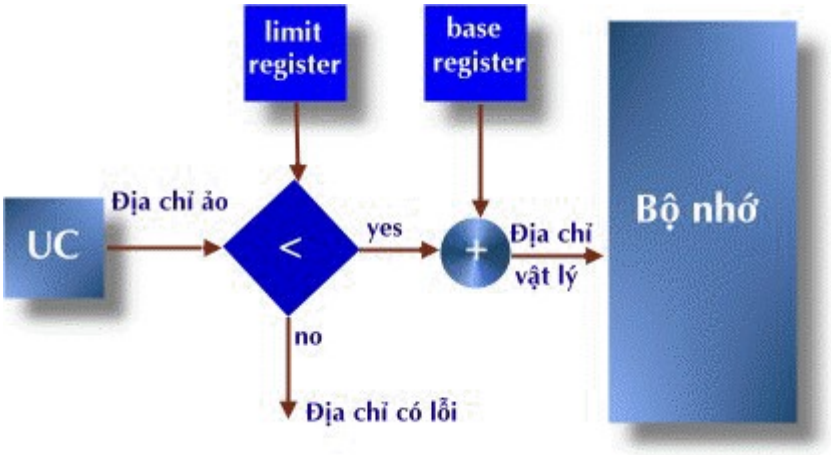
Thảo luận:

- Thời điểm kết buộc địa chỉ là thời điểm nạp, do vậy sau khi nạp không thể dời chuyển tiến trình trong bộ nhớ .
- Không có khả năng kiểm soát địa chỉ các tiến trình truy cập, do vậy không có sự bảo vệ.

6.3.2. Mô hình Base &Bound

Ý tưởng : Tiến trình được nạp vào một vùng nhớ liên tục đủ lớn để chứa toàn bộ tiến trình. Tại thời điểm biên dịch các địa chỉ bên trong tiến trình vẫn là địa chỉ tương đối. Tuy nhiên bổ túc vào cấu trúc phần cứng của máy tính một thanh ghi nền (*base register*) và một thanh ghi giới hạn (*bound register*). Khi một tiến trình được cấp phát vùng nhớ, nạp vào thanh ghi nền địa chỉ bắt đầu của phân vùng được cấp phát cho tiến trình, và nạp vào thanh ghi giới hạn kích thước của tiến trình. Sau đó, mỗi địa chỉ bộ nhớ được phát sinh sẽ tự động được cộng với địa chỉ chứa trong thanh ghi nền để cho ra địa chỉ tuyệt đối trong bộ nhớ, các địa chỉ cũng được đối chiếu với thanh ghi giới hạn để bảo đảm tiến trình không truy xuất ngoài phạm vi phân vùng được cấp cho nó.

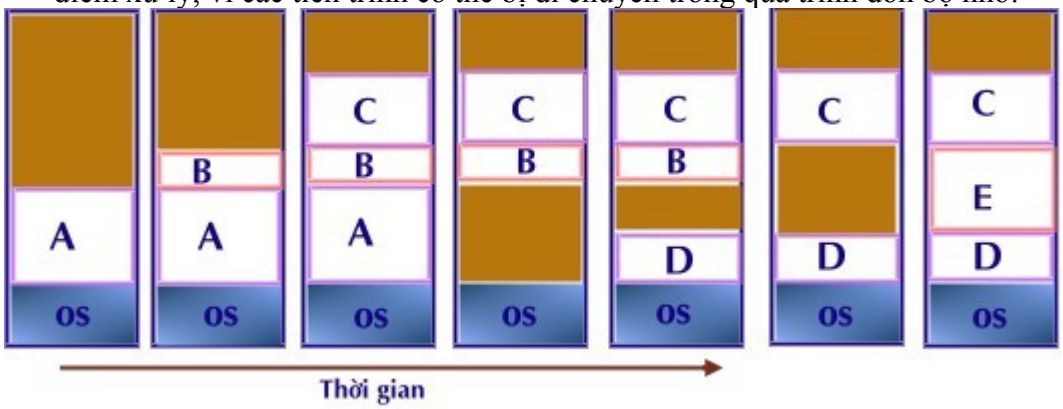




Hai thanh ghi hỗ trợ chuyển đổi địa chỉ

Thảo luận:

- Một ưu điểm của việc sử dụng thanh ghi nền là có thể di chuyển các chương trình trong bộ nhớ sau khi chúng bắt đầu xử lý, mỗi khi tiến trình được di chuyển đến một vị trí mới, chỉ cần nạp lại giá trị cho thanh ghi nền, các địa chỉ tuyệt đối sẽ được phát sinh lại mà không cần cập nhật các địa chỉ tương đối trong chương trình
- Chịu đựng hiện tượng phân mảnh ngoài(*external fragmentation*) : khi các tiến trình lần lượt vào và ra khỏi hệ thống, dần dần xuất hiện các khe hở giữa các tiến trình. Đây là các khe hở được tạo ra do kích thước của tiến trình mới được nạp nhỏ hơn kích thước vùng nhớ mới được giải phóng bởi một tiến trình đã kết thúc và ra khỏi hệ thống. Hiện tượng này có thể dẫn đến tình huống tổng vùng nhớ trống đủ để thoả mãn yêu cầu, nhưng các vùng nhớ này lại không liên tục ! Người ta có thể áp dụng kỹ thuật « dồn bộ nhớ » (*memory compaction*) để kết hợp các mảnh bộ nhớ nhỏ rời rạc thành một vùng nhớ lớn liên tục. Tuy nhiên, kỹ thuật này đòi hỏi nhiều thời gian xử lý, ngoài ra, sự kết buộc địa chỉ phải thực hiện vào thời điểm xử lý, vì các tiến trình có thể bị di chuyển trong quá trình dồn bộ nhớ.



Phân mảnh ngoài:

Vấn đề nảy sinh khi kích thước của tiến trình tăng trưởng trong qúa trình xử lý mà không còn vùng nhớ trống gần kề để mở rộng vùng nhớ cho tiến trình. Có hai cách giải quyết:

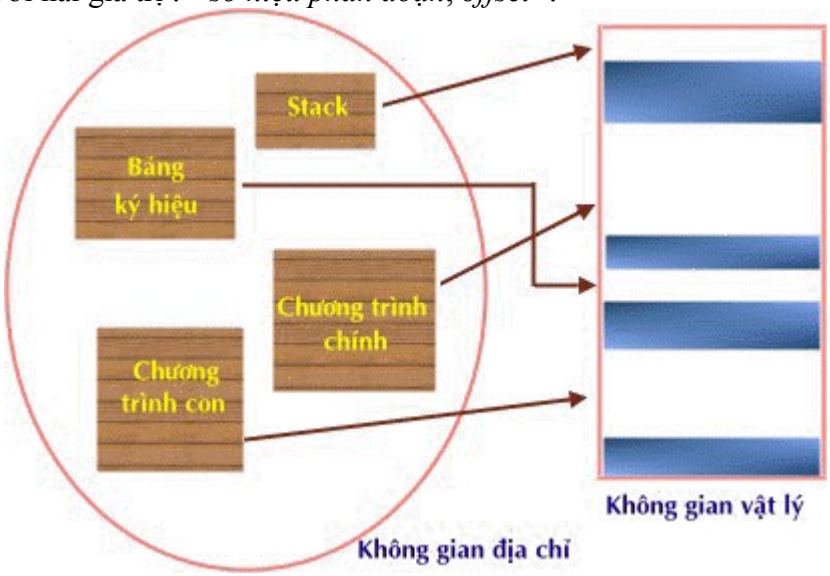
- Dời chỗ tiến trình: di chuyển tiến trình đến một vùng nhớ khác đủ lớn để thỏa mãn nhu cầu tăng trưởng của tiến trình.

- Cấp phát dư vùng nhớ cho tiến trình: cấp phát dự phòng cho tiến trình một vùng nhớ lớn hơn yêu cầu ban đầu của tiến trình.
    - Một tiến trình cần được nạp vào bộ nhớ để xử lý. Trong các phương thức tổ chức trên đây, một tiến trình luôn được lưu trữ trong bộ nhớ suốt quá trình xử lý của nó. Tuy nhiên, trong trường hợp tiến trình bị khóa, hoặc tiến trình sử dụng hết thời gian CPU dành cho nó, nó có thể được chuyển tạm thời ra bộ nhớ phụ và sau này được nạp trở lại vào bộ nhớ chính để tiếp tục xử lý.
    - Các cách tổ chức bộ nhớ trên đây đều phải chịu đựng tình trạng bộ nhớ bị phân mảnh vì chúng đều tiếp cận theo kiểu cấp phát một vùng nhớ liên tục cho tiến trình. Như đã thảo luận, có thể sử dụng kỹ thuật dồn bộ nhớ để loại bỏ sự phân mảnh ngoại vi, nhưng chi phí thực hiện rất cao. Một giải pháp khác hữu hiệu hơn là cho phép không gian địa chỉ vật lý của tiến trình không liên tục, nghĩa là có thể cấp phát cho tiến trình những vùng nhớ tự do bất kỳ, không cần liên tục.

6.4. Cấp phát không liên tục

6.4.1. Phân đoạn (Segmentation)

Ý tưởng: quan niệm không gian địa chỉ là một tập các *phân đoạn (segments)* – các phân đoạn là những phần bộ nhớ *kích thước khác nhau và có liên hệ logic với nhau*. Mỗi phân đoạn có một tên gọi (số hiệu phân đoạn) và một độ dài. Người dùng sẽ thiết lập mỗi địa chỉ với hai giá trị : <*số hiệu phân đoạn*, *offset*>.



Hình 6.4.1-1. Mô hình phân đoạn bộ nhớ

Cơ chế MMU trong kỹ thuật phân đoạn:

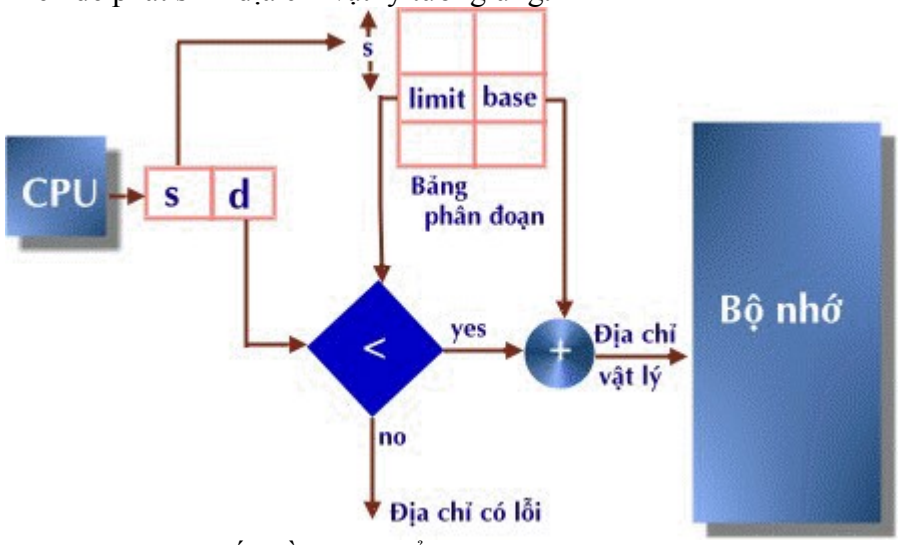
Cần phải xây dựng một ánh xạ để chuyển đổi các địa chỉ 2 chiều được người dùng định nghĩa thành địa chỉ vật lý một chiều. Sự chuyển đổi này được thực hiện qua một *bảng phân đoạn*. Mỗi thành phần trong bảng phân đoạn bao gồm một *thanh ghi nền* và

một *thanh ghi giới hạn*. Thanh ghi nền lưu trữ địa chỉ vật lý nơi bắt đầu phân đoạn trong bộ nhớ, trong khi thanh ghi giới hạn đặc tả chiều dài của phân đoạn.

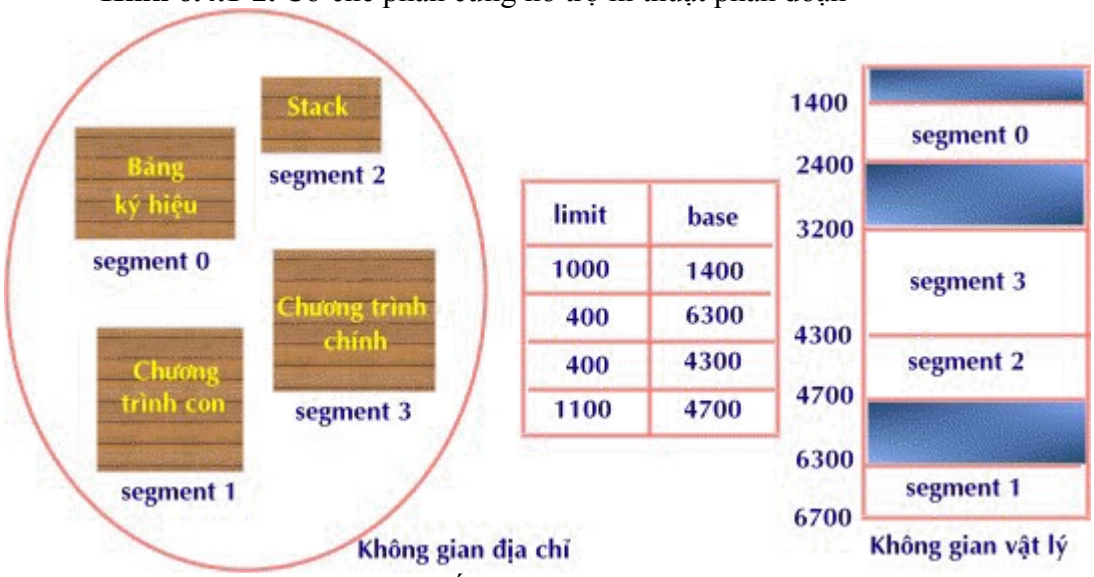
Chuyển đổi địa chỉ:

Mỗi địa chỉ ảo là một bộ <***s,d***> *:*

- *số hiệu phân đoạn s* : được sử dụng như chỉ mục đến bảng phân đoạn
- *địa chỉ tương đối d* : có giá trị trong khoảng từ 0 đến giới hạn chiều dài của phân đoạn. Nếu địa chỉ tương đối hợp lệ, nó sẽ được cộng với giá trị chứa trong thanh ghi nền để phát sinh địa chỉ vật lý tương ứng.



Hình 6.4.1-2. Cơ chế phần cứng hỗ trợ kĩ thuật phân đoạn

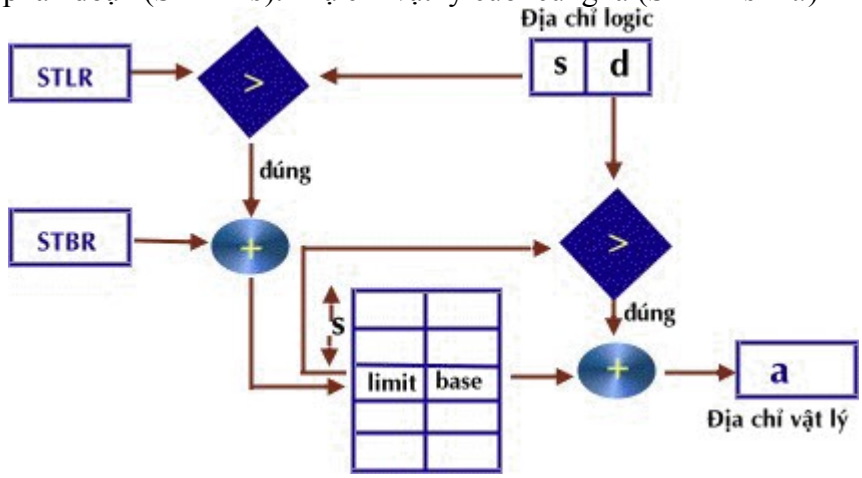


Hình 6.4.1-3. Hệ thống phân đoạn

Cài đặt bảng phân đoạn:

Có thể sử dụng các thanh ghi để lưu trữ bảng phân đoạn nếu số lượng phân đoạn nhỏ. Trong trường hợp chương trình bao gồm quá nhiều phân đoạn, bảng phân đoạn phải được lưu trong bộ nhớ chính. Một *thanh ghi nền bảng phân đoạn* (STBR) chỉ đến địa chỉ bắt đầu của bảng phân đoạn. Vì số lượng phân đoạn sử dụng trong một chương trình biến động, cần sử dụng thêm một *thanh ghi đặc tả kích thước bảng phân đoạn* (STLR).

Với một địa chỉ logic <*s,d*>, trước tiên số hiệu phân đoạn *s* được kiểm tra tính hợp lệ (s <STLR). Kế tiếp, cộng giá trị s với STBR để có được địa chỉ địa chỉ của phần tử thứ *s* trong bảng phân đoạn (STBR+*s*). Địa chỉ vật lý cuối cùng là (STBR+*s* + *d*)

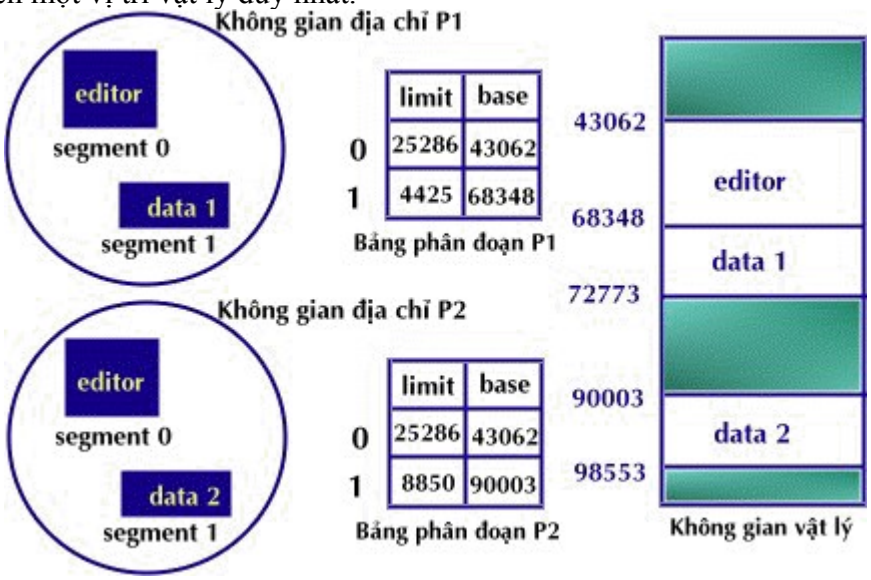


Hình 6.4.1-4. Sử dụng STBR, STLR và bảng phân đoạn

Bảo vệ: Một ưu điểm đặc biệt của cơ chế phân đoạn là khả năng đặc tả thuộc tính bảo vệ cho mỗi phân đoạn. Vì mỗi phân đoạn biểu diễn cho một phần của chương trình với ngữ nghĩa được người dùng xác định, người sử dụng có thể biết được một phân đoạn chứa đựng những gì bên trong, do vậy họ có thể đặc tả các thuộc tính bảo vệ thích hợp cho từng phân đoạn.

Cơ chế phần cứng phụ trách chuyển đổi địa chỉ bộ nhớ sẽ kiểm tra các bit bảo vệ được gán với mỗi phần tử trong bảng phân đoạn để ngăn chặn các thao tác truy xuất bất hợp lệ đến phân đoạn tương ứng.

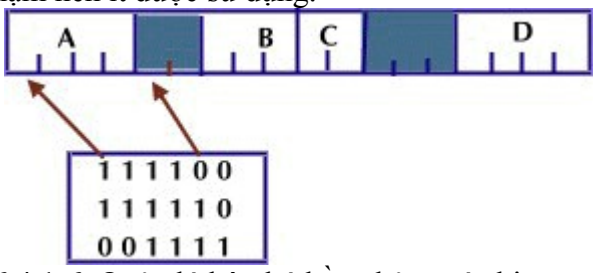
Chia sẻ phân đoạn: Một ưu điểm khác của kỹ thuật phân đoạn là khả năng chia sẻ ở mức độ phân đoạn. Nhờ khả năng này, các tiến trình có thể chia sẻ với nhau từng phần chương trình (ví dụ các thủ tục, hàm), không nhất thiết phải chia sẻ toàn bộ chương trình như trường hợp phân trang. Mỗi tiến trình có một bảng phân đoạn riêng, một phân đoạn được chia sẻ khi các phần tử trong bảng phân đoạn của hai tiến trình khác nhau cùng chỉ đến một vị trí vật lý duy nhất.



Hình 6.4.1-5. Chia sẻ code trong hệ phân đoạn

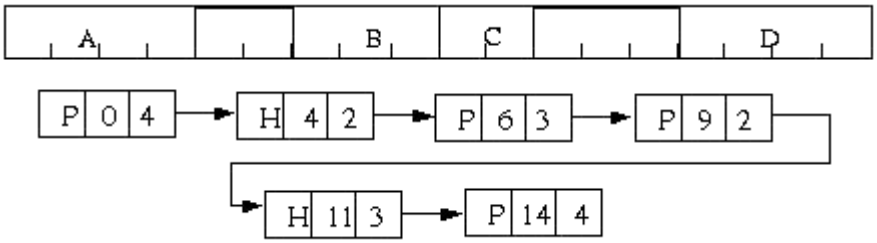
Thảo luận:

- Phải giải quyết vấn đề cấp phát động: làm thế nào để thỏa mãn một yêu cầu vùng nhớ kích thước N ? Cần phải chọn vùng nhớ nào trong danh sách vùng nhớ tự do để cấp phát ? Như vậy cần phải ghi nhớ hiện trạng bộ nhớ để có thể cấp phát đúng. Có hai phương pháp quản lý chủ yếu :
  - Quản lý bằng một bảng các bit : bộ nhớ được chia thành các đơn vị cấp phát, mỗi đơn vị được phản ánh bằng một bit trong bảng các bit, một bit nhận giá trị 0 nếu đơn vị bộ nhớ tương ứng đang tự do, và nhận giá trị 1 nếu đơn vị tương ứng đã được cấp phát cho một tiến trình. Khi cần nạp một tiến trình có kích thước *k* đơn vị, cần phải tìm trong bảng các bit một dãy con *k* bit nhận giá trị 0. Đây là một giải pháp đơn giản, nhưng thực hiện chậm nên ít được sử dụng.



Hình 6.4.1-6. Quản lý bộ nhớ bằng bảng các bit

  - Quản lý bằng danh sách: Tổ chức một danh sách các phân đoạn đã cấp phát và phân đoạn tự do, một phân đoạn có thể là một tiến trình (P) hay vùng nhớ trống giữa hai tiến trình (H).



Hình 6.4.1-7. Quản lý bộ nhớ bằng danh sách

Các thuật toán thông dụng để chọn một phân đoạn tự do trong danh sách để cấp phát cho tiến trình là :

- **First-fit:** cấp phát phân đoạn tự do đầu tiên đủ lớn.
- **Best-fit:** cấp phát phân đoạn tự do nhỏ nhất nhưng đủ lớn để thõa mãn nhu cầu.
- **Worst-fit :** cấp phát phân đoạn tự do lớn nhất.

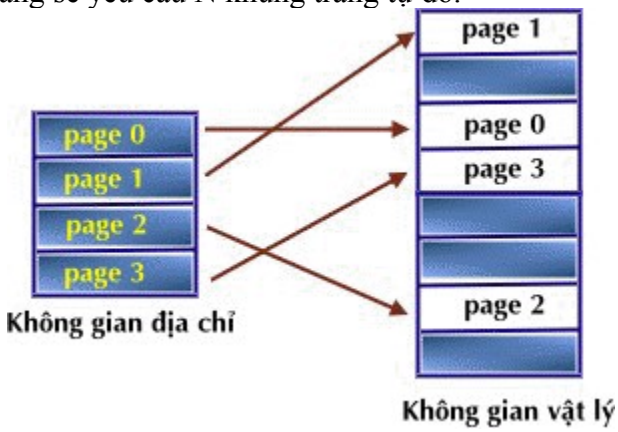
Trong hệ thống sử dụng kỹ thuật phân đoạn , hiện tượng phân mảnh ngoại vi lại xuất hiện khi các khối nhớ tự do đều quá nhỏ, không đủ để chứa một phân đoạn.

6.4.2. Phân trang (Paging)

Ý tưởng:

Phân bộ nhớ vật lý thành các khối (block) có kích thước cố định và bằng nhau, gọi là *khung trang (page frame)*. Không gian địa chỉ cũng được chia thành các khối có cùng kích thước với khung trang, và được gọi là *trang* (*page*). Khi cần nạp một tiến trình

để xử lý, các trang của tiến trình sẽ được nạp vào những khung trang còn trống. Một tiến trình kích thước N trang sẽ yêu cầu N khung trang tự do.



Hình 6.4.2-1. Mô hình bộ nhớ phân trang

Cơ chế MMU trong kỹ thuật phân trang:

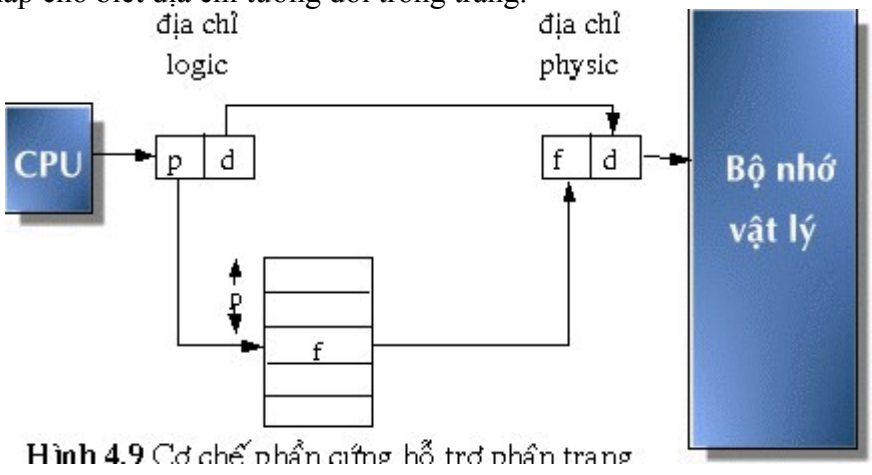
Cơ chế phần cứng hỗ trợ thực hiện chuyển đổi địa chỉ trong cơ chế phân trang là bảng trang (*pages table*). Mỗi phần tử trong bảng trang cho biết các địa chỉ bắt đầu của vị trí lưu trữ trang tương ứng trong bộ nhớ vật lý (số hiệu khung trang trong bộ nhớ vật lý đang chứa trang).

Chuyển đổi địa chỉ:

Mỗi địa chỉ phát sinh bởi CPU được chia thành hai phần:

- *số hiệu trang (p)*: sử dụng như chỉ mục đến phần tử tương ứng trong bảng trang.
- *địa chỉ tương đối trong trang (d)*: kết hợp với địa chỉ bắt đầu của trang để tạo ra địa chỉ vật lý mà trình quản lý bộ nhớ sử dụng.

Kích thước của trang do phần cứng qui định. Để dễ phân tích địa chỉ ảo thành số hiệu trang và địa chỉ tương đối, kích thước của một trang thông thường là một lũy thừa của 2 (biến đổi trong phạm vi 512 bytes và 8192 bytes). Nếu kích thước của không gian địa chỉ là 2^m và kích thước trang là 2^n, thì *m-n* bits cao của địa chỉ ảo sẽ biểu diễn số hiệu trang, và *n* bits thấp cho biết địa chỉ tương đối trong trang.



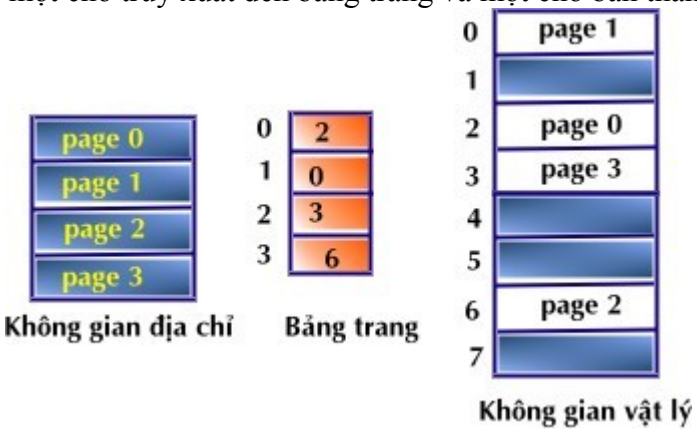
Hình 6.4.2-2. Cơ chế phần cứng hỗ trợ phân trang

Cài đặt bảng trang:

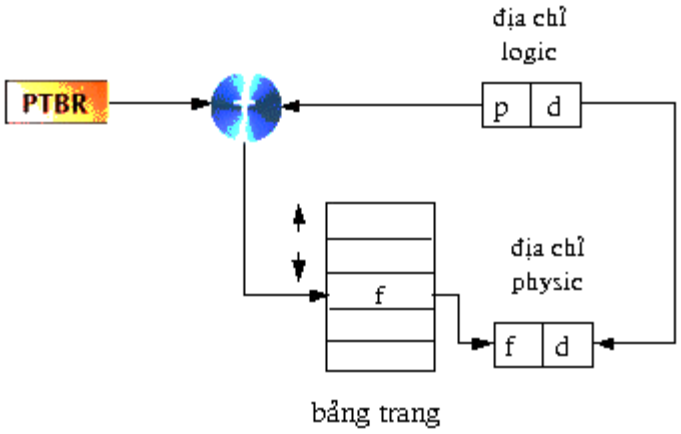
Trong trường hợp đơn giản nhất, bảng trang một tập các thanh ghi được sử dụng để cài đặt bảng trang. Tuy nhiên việc sử dụng thanh ghi chỉ phù hợp với các bảng trang

có kích thước nhỏ, nếu bảng trang có kích thước lớn, nó phải được lưu trữ trong bộ nhớ chính, và sử dụng một thanh ghi để lưu địa chỉ bắt đầu lưu trữ bảng trang (PTBR).

Theo cách tổ chức này, mỗi truy xuất đến dữ liệu hay chỉ thị đều đòi hỏi hai lần truy xuất bộ nhớ : một cho truy xuất đến bảng trang và một cho bản thân dữ liệu!



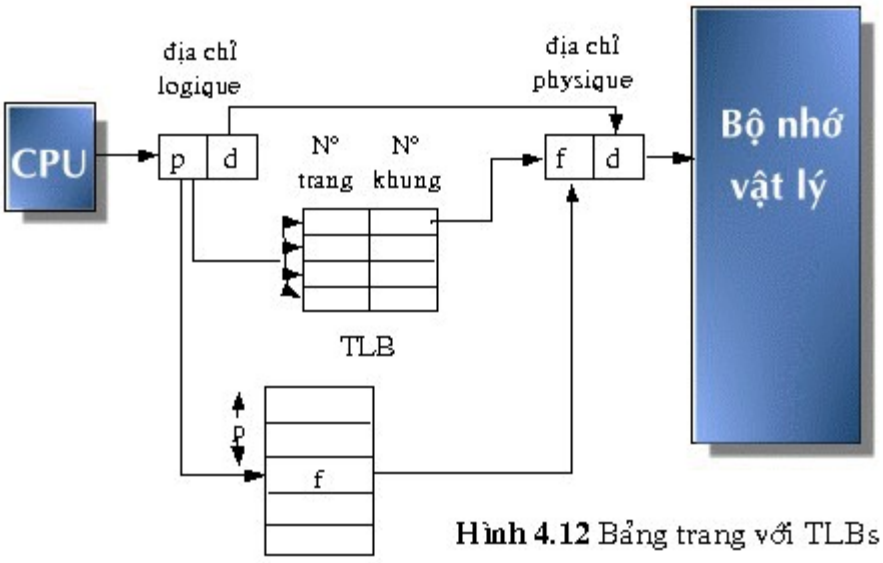
Hình 6.4.2-3. Mô hình bộ nhớ phân trang



Hình 6.4.2-4. Sử dụng thanh ghi nền trỏ đến bảng trang

Có thể né tránh bớt việc truy xuất bộ nhớ hai lần bằng cách sử dụng thêm một vùng nhớ đặc biệt , với tốc độ truy xuất nhanh và cho phép tìm kiếm song song, vùng nhớ cache nhỏ này thường được gọi là bộ nhớ kết hợp (TLBs). Mỗi thanh ghi trong bộ nhớ kết hợp gồm một từ khóa và một giá trị, khi đưa đến bộ nhớ kết hợp một đối tượng cần tìm, đối tượng này sẽ được so sánh cùng lúc với các từ khóa trong bộ nhớ kết hợp để tìm ra phần tử tương ứng. Nhờ đặc tính này mà việc tìm kiếm trên bộ nhớ kết hợp được thực hiện rất nhanh, nhưng chi phí phần cứng lại cao.

Trong kỹ thuật phân trang, TLBs được sử dụng để lưu trữ các trang bộ nhớ được truy cập gần hiện tại nhất. Khi CPU phát sinh một địa chỉ, số hiệu trang của địa chỉ sẽ được so sánh với các phần tử trong TLBs, nếu có trang tương ứng trong TLBs, thì sẽ xác định được ngay số hiệu khung trang tương ứng, nếu không mới cần thực hiện thao tác tìm kiếm trong bảng trang.



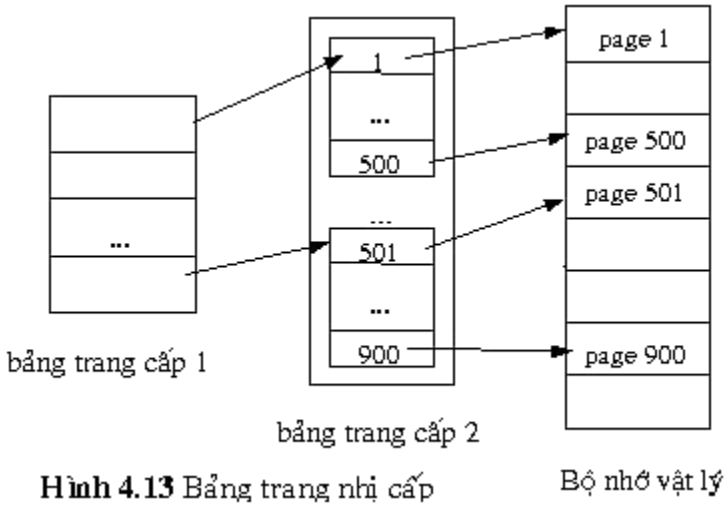
Hình 4.12 Bảng trang với TLBs

Hình 6.4.2-5. Bảng trang với TLBs

Tổ chức bảng trang:

Mỗi hệ điều hành có một phương pháp riêng để tổ chức lưu trữ bảng trang. Đa số các hệ điều hành cấp cho mỗi tiến trình một bảng trang. Tuy nhiên phương pháp này không thể chấp nhận được nếu hệ điều hành cho phép quản lý một không gian địa chỉ có dung lượng quá (2^{32}, 2^{64}): trong các hệ thống như thế, bản thân bảng trang đòi hỏi một vùng nhớ qúa lớn! Có hai giải pháp cho vấn đề này:

Phân trang đa cấp: phân chia bảng trang thành các phần nhỏ, bản thân bảng trang cũng sẽ được phân trang



Hình 4.13 Bảng trang nhị cấp

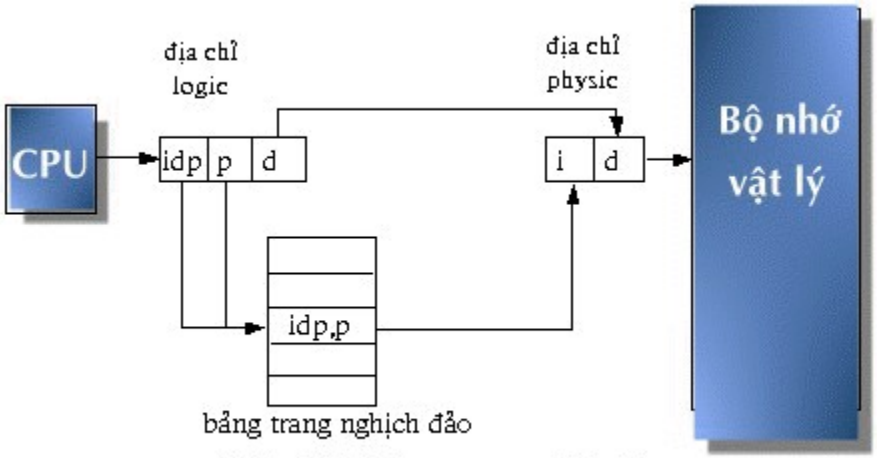
Hình 6.4.2-6. Bảng trang cấp 2

Bảng trang nghịch đảo: sử dụng duy nhất một *bảng trang nghịch đảo* cho tất cả các tiến trình . Mỗi phần tử trong *bảng trang nghịch đảo* phản ánh một khung trang trong bộ nhớ bao gồm địa chỉ logic của một trang đang được lưu trữ trong bộ nhớ vật lý tại khung trang này, cùng với thông tin về tiến trình đang được sỡ hữu trang. Mỗi địa chỉ ảo khi đó là một bộ ba <idp, p, d >

Trong đó : idp là định danh của tiến trình

p là số hiệu trang
d là địa chỉ tương đối trong trang

Mỗi phần tử trong bảng trang nghịch đảo là một cặp <idp, p >. Khi một tham khảo đến bộ nhớ được phát sinh, một phần địa chỉ ảo là <idp, p > được đưa đến cho trình quản lý bộ nhớ để tìm phần tử tương ứng trong bảng trang nghịch đảo, nếu tìm thấy, địa chỉ vật lý <i,d> sẽ được phát sinh. Trong các trường hợp khác, xem như tham khảo bộ nhớ đã truy xuất một địa chỉ bất hợp lệ.



Hình 4.14 Bảng trang nghịch đảo

Hình 6.4.2-7. Bảng trang nghịch đảo

Bảo vệ:

Cơ chế bảo vệ trong hệ thống phân trang được thực hiện với các bit bảo vệ được gắn với mỗi khung trang. Thông thường , các bit này được lưu trong bảng trang , vì mỗi truy xuất đến bộ nhớ đều phải tham khảo đến bảng trang để phát sinh địa chỉ vật lý, khi đó, hệ thống có thể kiểm tra các thao tác truy xuất trên khung trang tương ứng có hợp lệ với thuộc tính bảo vệ của nó không.

Ngoài ra, một bit phụ trội được thêm vào trong cấu trúc một phần tử của bảng trang : bit hợp lệ-bất hợp lệ (valid-invalid).

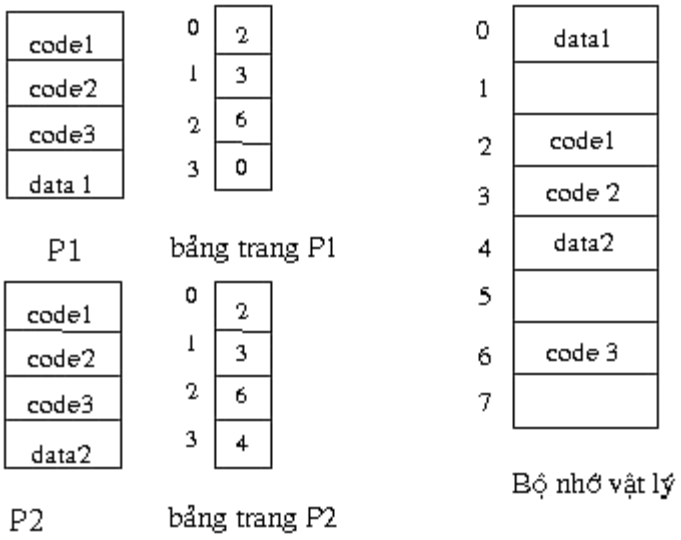
- *Hợp lệ* : trang tương ứng thuộc về không gian địa chỉ của tiến trình.
- *Bất hợp lệ* : trang tương ứng không nằm trong không gian địa chỉ của tiến trình, điều này có nghĩa tiến trình đã truy xuất đến một địa chỉ không được phép.

số hiệu khung trang	bit valid-invalid

Hình 6.4.2-8. Cấu trúc một phần tử trong bảng trang

Chia sẻ bộ nhớ trong cơ chế phân trang:

Một ưu điểm của cơ chế phân trang là cho phép chia sẻ các trang giữa các tiến trình.Trong trường hợp này, sự chia sẻ được thực hiện bằng cách ánh xạ nhiều địa chỉ logic vào một địa chỉ vật lý duy nhất. Có thể áp dụng kỹ thuật này để cho phép có tiến trình chia sẻ một vùng code chung*:* nếu có nhiều tiến trình của cùng một chương trình, chỉ cần lưu trữ một đoạn code của chương trình này trong bộ nhớ, các tiến trình sẽ có thể cùng truy xuất đến các trang chứa code chung này. Lưu ý để có thể chia sẻ một đoạn code, đoạn code này phải có thuộc tính *reenterable* (cho phép một bản sao của chương trình được sử dụng đồng thời bởi nhiều tác vụ).



Hình 4.16 Chia sẻ các trang trong hệ phân trang

Hình 6.4.2-9. Chia sẻ các trang nhớ

Thảo luận:

- Kỹ thuật phân trang loại bỏ được hiện tượng phân mảnh ngoại vi : mỗi khung trang đều có thể được cấp phát cho một tiến trình nào đó có yêu cầu. Tuy nhiên hiện tượng phân mảnh nội vi vẫn có thể xảy ra khi kích thước của tiến trình không đúng bằng bội số của kích thước một trang, khi đó, trang cuối cùng sẽ không được sử dụng hết.
- Một khiá cạnh tích cực rất quan trọng khác của kỹ thuật phân trang là sự phân biệt rạch ròi góc nhìn của người dùng và của bộ phận quản lý bộ nhớ vật lý:
    - *Góc nhìn của người sử dụng:* một tiến trình của người dùng nhìn thấy bộ nhớ như là một không gian liên tục, đồng nhất và chỉ chứa duy nhất bản thân tiến trình này.
    - *Góc nhìn của bộ nhớ vật lý:* một tiến trình của người sử dụng được lưu trữ phân tán khắp bộ nhớ vật lý, trong bộ nhớ vật lý đồng thời cũng chứa những tiến trình khác.
- Phần cứng đảm nhiệm việc chuyển đổi địa chỉ logic thành địa chỉ vật lý . Sự chuyển đổi này là trong suốt đối với người sử dụng.
- Để lưu trữ các thông tin chi tiết về quá trình cấp phát bộ nhớ, hệ điều hành sử dụng một bảng khung trang, mà mỗi phần tử mô tả tình trạng của một khung trang vật lý : tự do hay được cấp phát cho một tiến trình nào đó .

Lưu ý rằng sự phân trang không phản ánh đúng cách thức người sử dụng cảm nhận về bộ nhớ. Người sử dụng nhìn thấy bộ nhớ như một tập các đối tượng của chương trình (segments, các thư viện...) và một tập các đối tượng dữ liệu (biến toàn cục, stack, vùng nhớ chia sẻ...). Vấn đề đặt ra là cần tìm một cách thức biểu diễn bộ nhớ sao cho có thể cung cấp cho người dùng một cách nhìn gần với quan điểm logic của họ hơn và đó là kỹ thuật phân đoạn

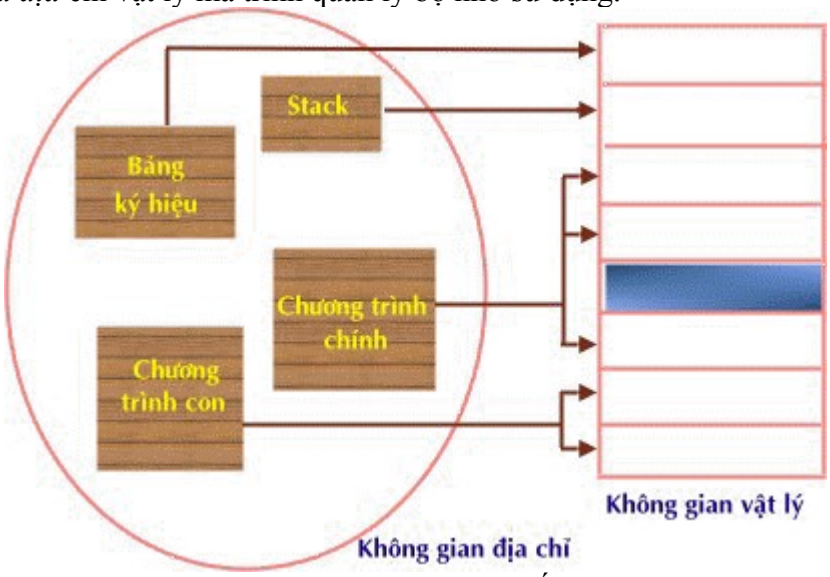
Kỹ thuật phân đoạn thõa mãn được nhu cầu thể hiện cấu trúc logic của chương trình nhưng nó dẫn đến tình huống phải cấp phát các khối nhớ có kích thước khác nhau cho các phân đoạn trong bộ nhớ vật lý. Điều này làm rắc rối vấn đề hơn rất nhiều so với việc

cấp phát các trang có kích thước tĩnh.Một giải pháp dung hoà là kết hợp cả hai kỹ thuật phân trang và phân đoạn : chúng ta tiến hành *phân trang các phân đoạn*.

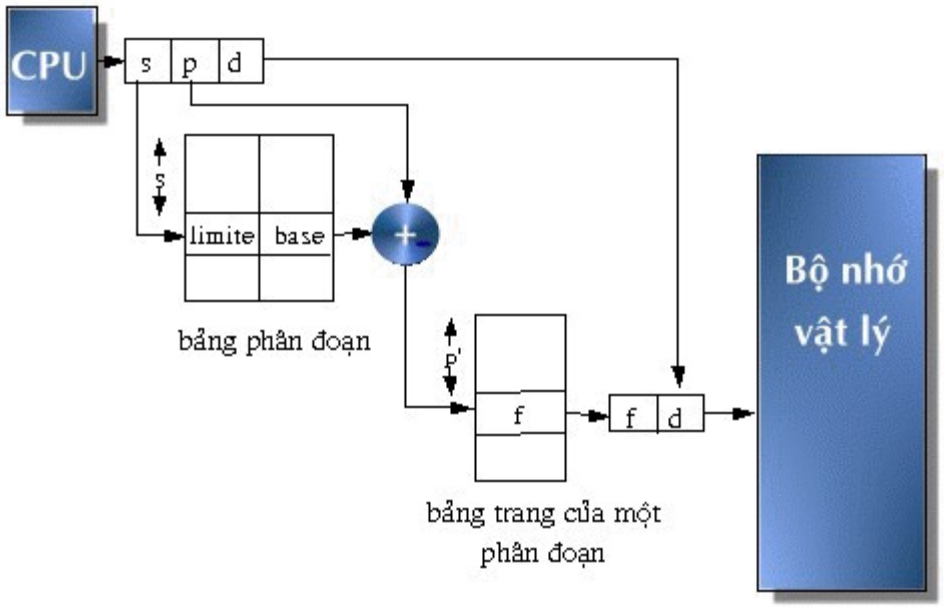
6.4.3. Phân đoạn kết hợp phân trang (Paged segmentation)

Ý tưởng: Không gian địa chỉ là một tập các phân đoạn, mỗi phân đoạn được chia thành nhiều trang. Khi một tiến trình được đưa vào hệ thống, hệ điều hành sẽ cấp phát cho tiến trình các trang cần thiết để chứa đủ các phân đoạn của tiến trình.

- **Cơ chế MMU trong kỹ thuật phân đoạn kết hợp phân trang**:
  Để hỗ trợ kỹ thuật phân đoạn, cần có một *bảng phân đoạn*, nhưng giờ đây mỗi phân đoạn cần có một *bảng trang* phân biệt.
- **Chuyển đổi địa chỉ:**
  Mỗi địa chỉ logic là một bộ ba: <***s,p,d***>
  - *số hiệu phân đoạn (s)*: sử dụng như chỉ mục đến phần tử tương ứng trong bảng phân đoạn.
  - *số hiệu trang (p)*: sử dụng như chỉ mục đến phần tử tương ứng trong bảng trang của phân đoạn.
  - *địa chỉ tương đối trong trang (d): kết hợp với địa chỉ bắt đầu của trang để tạo ra địa* chỉ vật lý mà trình quản lý bộ nhớ sử dụng.



Hình 6.4.3-1. Mô hình phân đoạn kế hợp phân trang



Hình 4.23 Cơ chế phần cứng của sự phân đoạn kết hợp phân trang

Hình 6.4.3-2. Cơ chế phần cứng của sự phân đoạn, phân trang kết hợp

Tất cả các mô hình tổ chức bộ nhớ trên đây đều có khuynh hướng cấp phát cho tiến trình toàn bộ các trang yêu cầu trước khi thật sự xử lý. Vì bộ nhớ vật lý có kích thước rất giới hạn, điều này dẫn đến hai điểm bất tiện sau :

- Kích thước tiến trình bị giới hạn bởi kích thước của bộ nhớ vật lý.
- Khó có thể bảo trì nhiều tiến trình cùng lúc trong bộ nhớ, và như vậy khó nâng cao mức độ đa chương của hệ thống.

6.5. Bộ nhớ ảo

Bộ nhớ ảo là một kỹ thuật hiện đại giúp cho người dùng được giải phóng hoàn toàn khỏi mối bận tâm về giới hạn bộ nhớ. Ý tưởng, ưu điểm và những vấn đề liên quan đến việc tổ chức bộ nhớ ảo sẽ được trình bày trong bài học này.

6.5.1. Cơ chế bộ nhớ ảo

Nếu đặt toàn thể không gian địa chỉ vào bộ nhớ vật lý, thì kích thước của chương trình bị giới hạn bởi kích thước bộ nhớ vật lý.

Thực tế, trong nhiều trường hợp, chúng ta không cần phải nạp toàn bộ chương trình vào bộ nhớ vật lý cùng một lúc, vì tại một thời điểm chỉ có một chỉ thị của tiến trình được xử lý. Ví dụ, các chương trình đều có một đoạn code xử lý lỗi, nhưng đoạn code này hầu như rất ít khi được sử dụng vì hiếm khi xảy ra lỗi, trong trường hợp này, không cần thiết phải nạp đoạn code xử lý lỗi từ đầu.

Từ nhận xét trên, một giải pháp được đề xuất là cho phép thực hiện một chương trình chỉ được nạp từng phần vào bộ nhớ vật lý. Ý tưởng chính của giải pháp này là tại mỗi thời điểm chỉ lưu trữ trong bộ nhớ vật lý các chỉ thị và dữ liệu của chương trình cần thiết cho việc thi hành tại thời điểm đó. Khi cần đến các chỉ thị khác, những chỉ thị mới sẽ được nạp vào bộ nhớ, tại vị trí trước đó bị chiếm giữ bởi các chỉ thị nay không còn cần đến nữa. Với giải pháp này, một chương trình có thể lớn hơn kích thước của vùng nhớ cấp phát cho nó.

Một cách để thực hiện ý tưởng của giải pháp trên đây là sử dụng kỹ thuật *overlay*. Kỹ thuật overlay không đòi hỏi bất kỳ sự trợ giúp đặc biệt nào của hệ điều hành , nhưng trái lại, lập trình viên phải biết cách lập trình theo cấu trúc overlay, và điều này đòi hỏi khá nhiều công sức.

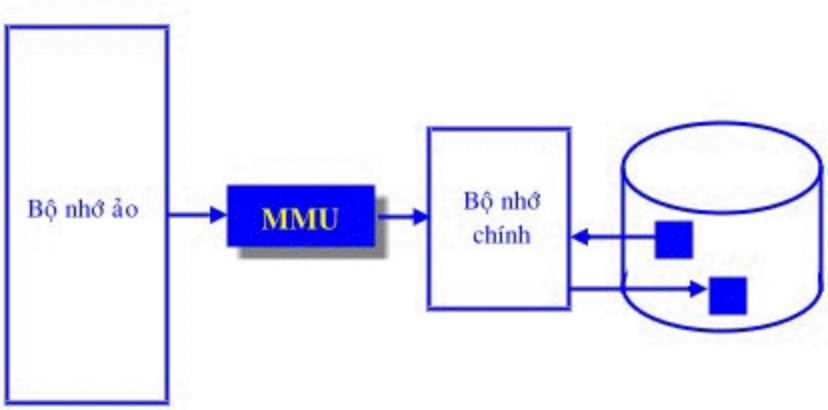
Để giải phóng lập trình viên khỏi các suy tư về giới hạn của bộ nhớ, mà cũng không tăng thêm khó khăn cho công việc lập trình của họ, người ta nghĩ đến các kỹ thuật tự động, cho phép xử lý một chương trình có kích thước lớn chỉ với một vùng nhớ có kích thước nhỏ . Giải pháp được tìm thấy với khái niệm *bộ nhớ ảo (virtual memory).*

6.5.1.1. Định nghĩa

Bộ nhớ ảo là một kỹ thuật cho phép xử lý một tiến trình không được nạp toàn bộ vào bộ nhớ vật lý. Bộ nhớ ảo mô hình hoá bộ nhớ như một bảng lưu trữ rất lớn và đồng nhất, tách biệt hẳn khái niệm không gian địa chỉ và không gian vật lý. Người sử dụng chỉ nhìn thấy và làm việc trong không gian địa chỉ ảo, việc chuyển đổi sang không gian vật lý do hệ điều hành thực hiện với sự trợ giúp của các cơ chế phần cứng cụ thể.

Thảo luận:

- Cần kết hợp kỹ thuật *swapping* đển chuyển các phần của chương trình vào-ra giữa bộ nhớ chính và bộ nhớ phụ khi cần thiết.
- Nhờ việc tách biệt bộ nhớ ảo và bộ nhớ vật lý, có thể tổ chức một bộ nhớ ảo có kích thước lớn hơn bộ nhớ vật lý.
- Bộ nhớ ảo cho phép giảm nhẹ công việc của lập trình viên vì họ không cần bận tâm đến giới hạn của vùng nhớ vật lý, cũng như không cần tổ chức chương trình theo cấu trúc overlays.



Hình 6.5.1.1-1. Bộ nhớ ảo

6.5.1.2. Cài đặt bộ nhớ ảo

Bộ nhớ ảo thường được thực hiện với kỹ thuật *phân trang theo yêu cầu (demand paging)*. Cũng có thể sử dụng kỹ thuật *phân đoạn theo yêu cầu (demand segmentation)* để cài đặt bộ nhớ ảo, tuy nhiên việc cấp phát và thay thế các phân đoạn phức tạp hơn thao tác trên trang, vì kích thước không bằng nhau của các đoạn.

Phân trang theo yêu cầu (demand paging)

Một hệ thống phân trang theo yêu cầu là hệ thống sử dụng kỹ thuật phân trang kết hợp với kỹ thuật swapping. Một tiến trình được xem như một tập các trang, thường trú trên bộ nhớ phụ (thường là đĩa). Khi cần xử lý, tiến trình sẽ được nạp vào bộ nhớ chính. Nhưng thay vì nạp toàn bộ chương trình, chỉ những trang cần thiết trong thời điểm hiện tại mới được nạp vào bộ nhớ. Như vậy một trang chỉ được nạp vào bộ nhớ chính khi có yêu cầu.

Với mô hình này, cần cung cấp một cơ chế phần cứng giúp phân biệt các trang đang ở trong bộ nhớ chính và các trang trên đĩa. Có thể sử dụng lại bit *valid-invalid* nhưng với ngữ nghĩa mới:

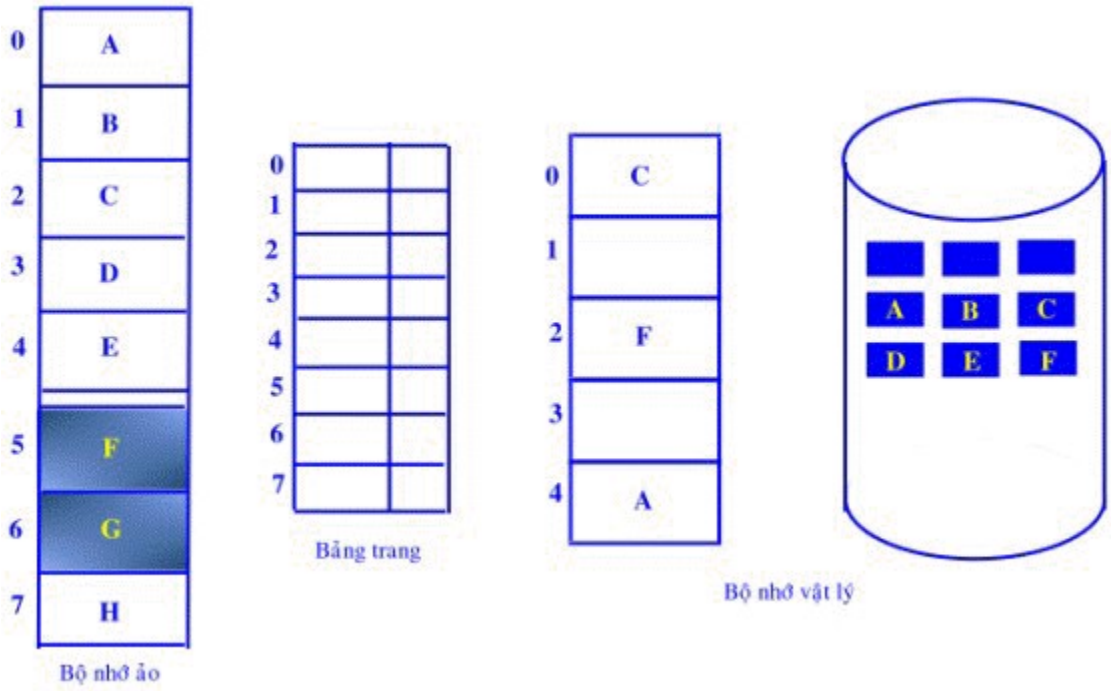
- *valid* : trang tương ứng là hợp lệ và đang ở trong bộ nhớ chính .
- *invalid* : hoặc trang bất hợp lệ (không thuộc về không gian địa chỉ của tiến trình) hoặc trang hợp lệ nhưng đang được lưu trên bộ nhớ phụ.

Một phần tử trong bảng trang mô tả cho một trang không nằm trong bộ nhớ chính, sẽ được đánh dấu *invalid* và chứa địa chỉ của trang trên bộ nhớ phụ.

Cơ chế phần cứng:

Cơ chế phần cứng hỗ trợ kỹ thuật phân trang theo yêu cầu là sự kết hợp của cơ chế hỗ trợ kỹ thuật phân trang và kỹ thuật swapping:

- *Bảng trang*: Cấu trúc bảng trang phải cho phép phản ánh tình trạng của một trang là đang nằm trong bộ nhớ chính hay bộ nhớ phụ.
- *Bộ nhớ phụ*: Bộ nhớ phụ lưu trữ những trang không được nạp vào bộ nhớ chính. Bộ nhớ phụ thường được sử dụng là đĩa, và vùng không gian đĩa dùng để lưu trữ tạm các trang trong kỹ thuật swapping được gọi là *không gian swapping.*

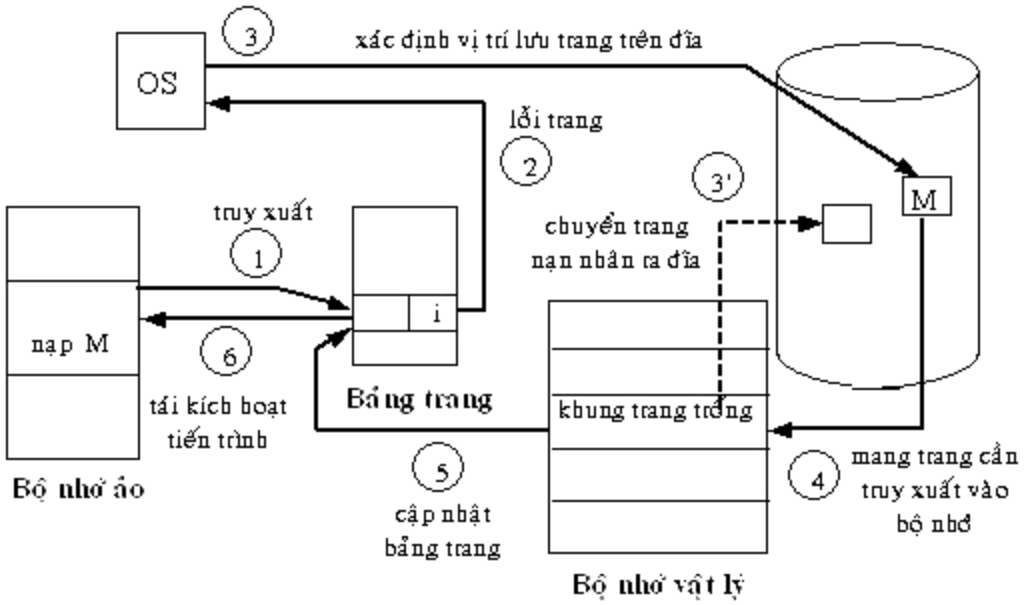


Hình 6.5.1.2-1. Bảng trang với một số trang trên bộ nhớ phụ

Lỗi trang

Truy xuất đến một trang được đánh dấu bất hợp lệ sẽ làm phát sinh một *lỗi trang* (*page fault*). Khi dò tìm trong bảng trang để lấy các thông tin cần thiết cho việc chuyển đổi địa chỉ, nếu nhận thấy trang đang được yêu cầu truy xuất là bất hợp lệ, cơ chế phần cứng sẽ phát sinh một ngắt để báo cho hệ điều hành. Hệ điều hành sẽ xử lý lỗi trang như sau :

- Kiểm tra truy xuất đến bộ nhớ là hợp lệ hay bất hợp lệ
- Nếu truy xuất bất hợp lệ : kết thúc tiến trình
- Ngược lại : đến bước 3
- Tìm vị trí chứa trang muốn truy xuất trên đĩa.
- Tìm một khung trang trống trong bộ nhớ chính:
    - Nếu tìm thấy : đến bước 5
    - Nếu không còn khung trang trống, chọn một khung trang « nạn nhân » và chuyển trang « nạn nhân » ra bộ nhớ phụ (lưu nội dung của trang đang chiếm giữ khung trang này lên đĩa), cập nhật bảng trang tương ứng rồi đến bước 5
- Chuyển trang muốn truy xuất từ bộ nhớ phụ vào bộ nhớ chính : nạp trang cần truy xuất vào khung trang trống đã chọn (hay vừa mới làm trống) ; cập nhật nội dung bảng trang, bảng khung trang tương ứng.
- Tái kích hoạt tiến trình người sử dụng.



Hình 6.5.1.2-2. Các giai đoạn xử lý lỗi trang

6.5.2. Thay thế trang

Khi xảy ra một lỗi trang, cần phải mang trang vắng mặt vào bộ nhớ . Nếu không có một khung trang nào trống, hệ điều hành cần thực hiện công việc *thay thế trang* – chọn một trang đang nằm trong bộ nhớ mà không được sử dụng tại thời điểm hiện tại và chuyển nó ra *không gian swapping* trên đĩa để giải phóng một khung trang dành chỗ nạp trang cần truy xuất vào bộ nhớ.

Như vậy nếu không có khung trang trống, thì mỗi khi xảy ra lỗi trang cần phải thực hiện hai thao tác chuyển trang : chuyển một trang ra bộ nhớ phụ và nạp một trang khác vào bộ nhớ chính. Có thể giảm bớt số lần chuyển trang bằng cách sử dụng thêm một bit *cập nhật* (dirty bit). Bit này được gắn với mỗi trang để phản ánh tình trạng trang có bị cập nhật hay không : giá trị của bit được cơ chế phần cứng đặt là 1 mỗi lần có một từ được ghi vào trang, để ghi nhận nội dung trang có bị sửa đổi. Khi cần thay thế một trang, nếu bit cập nhật có giá trị là 1 thì trang cần được lưu lại trên đĩa, ngược lại, nếu bit cập nhật là 0, nghĩa là trang không bị thay đổi, thì không cần lưu trữ trang trở lại đĩa.

số hiệu trang	bit valid-invalid	dirty bit

Hình 6.5.2-1. Cấu trúc một phần tử trong bảng trang

Sự thay thế trang là cần thiết cho kỹ thuật phân trang theo yêu cầu. Nhờ cơ chế này, hệ thống có thể hoàn toàn tách rời bộ nhớ ảo và bộ nhớ vật lý, cung cấp cho lập trình viên một bộ nhớ ảo rất lớn trên một bộ nhớ vật lý có thể bé hơn rất nhiều lần.

6.5.2.1. Sự thi hành phân trang theo yêu cầu

Việc áp dụng kỹ thuật phân trang theo yêu cầu có thể ảnh hưởng mạnh đến tình hình hoạt động của hệ thống.

Gỉa sử *p* là xác suất xảy ra một lỗi trang (0□ p □ 1):

p = 0 : không có lỗi trang nào

p = 1 : mỗi truy xuất sẽ phát sinh một lỗi trang

Thời gian thật sự cần để thực hiện một truy xuất bộ nhớ (TEA) là:

TEA = (1-p)ma + p (tdp) [+ swap out] + swap in + tái kích hoạt

Trong công thức này, *ma* là thời gian truy xuất bộ nhớ, *tdp* thời gian xử lý lỗi trang.

Có thể thấy rằng, để duy trì ở một mức độ chấp nhận được sự chậm trễ trong hoạt động của hệ thống do phân trang, cần phải duy trì *tỷ lệ phát sinh lỗi trang* thấp.

Hơn nữa, để cài đặt kỹ thuật phân trang theo yêu cầu, cần phải giải quyết hai vấn đề chính yếu : xây dựng một *thuật toán cấp phát khung trang*, và *thuật toán thay thế trang*.

6.5.2.2. Các thuật toán thay thế trang

Vấn đề chính khi thay thế trang là chọn lựa một trang « nạn nhân » để chuyển ra bộ nhớ phụ. Có nhiều thuật toán thay thế trang khác nhau, nhưng tất cả cùng chung một mục tiêu : chọn trang « nạn nhân » là trang mà sau khi thay thế sẽ gây ra ít lỗi trang nhất.

Có thể đánh giá hiệu qủa của một thuật toán bằng cách xử lý trên một *chuỗi các địa chỉ cần truy xuất* và tính toán số lượng lỗi trang phát sinh.

Ví dụ: Giả sữ theo vết xử lý của một tiến trình và nhận thấy tiến trình thực hiện truy xuất các địa chỉ theo thứ tự sau :

0100, 0432, 0101, 0162, 0102, 0103, 0104, 0101, 0611, 0102, 0103,0104, 0101, 0610, 0102, 0103, 0104, 0101, 0609, 0102, 0105

Nếu có kích thước của một trang là 100 bytes, có thể viết lại *chuỗi truy xuất* trên giản lược hơn như sau :

1, 4, 1, 6, 1, 6, 1, 6, 1

Để xác định số các lỗi trang xảy ra khi sử dụng một thuật toán thay thế trang nào đó trên một chuỗi truy xuất cụ thể, còn cần phải biết số lượng khung trang sử dụng trong hệ thống.

Để minh hoạ các thuật toán thay thế trang sẽ trình bày, chuỗi truy xuất được sử dụng là :

7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

Thuật toán FIFO

Tiếp cận: Ghi nhận thời điểm một trang được mang vào bộ nhớ chính. Khi cần thay thế trang, trang ở trong bộ nhớ lâu nhất sẽ được chọn

Ví dụ : sử dụng 3 khung trang , ban đầu cả 3 đều trống :

7	**0**	**1**	**2**	**0**	**3**	**0**	**4**	**2**	**3**	**0**	**3**	**2**	**1**	**2**	**0**	**1**	**7**	**0**	**1**
7	7	7	2	2	2	2	4	4	4	0	0	0	0	0	0	0	7	7	7
	0	0	0	0	3	3	3	2	2	2	2	2	1	1	1	1	1	0	0
		1	1	1	1	0	0	0	3	3	3	3	3	2	2	2	2	2	1
*	*	*	*		*	*	*	*	*	*			*	*			*	*	*

Ghi chú : * : có lỗi trang

Thảo luận:

- Để áp dụng thuật toán FIFO, thực tế không nhất thiết phải ghi nhận thời điểm mỗi trang được nạp vào bộ nhớ, mà chỉ cần tổ chức quản lý các trang trong bộ nhớ trong một danh sách FIFO, khi đó trang đầu danh sách sẽ được chọn để thay thế.
- Thuật toán they thế trang FIFO dễ hiểu, dễ cài đặt. Tuy nhiên khi thực hiện không phải lúc nào cũng có kết qủa tốt : trang được chọn để thay thế có thể là trang chức nhiều dữ liệu cần thiết, thường xuyên được sử dụng nên được nạp sớm, do vậy khi bị chuyển ra bộ nhớ phụ sẽ nhanh chóng gây ra lỗi trang.
- Số lượng lỗi trang xảy ra sẽ tăng lên khi số lượng khung trang sử dụng tăng. Hiện tượng này gọi là *nghịch lý Belady*.

Ví dụ: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

Sử dụng 3 khung trang , sẽ có 9 lỗi trang phát sinh

1	2	3	4	1	2	5	1	2	3	4	5
1	1	1	4	4	4	5	5	5	5	5	5
	2	2	2	1	1	1	1	1	3	3	3
		3	3	3	2	2	2	2	2	4	4
*	*	*	*	*	*	*			*	*	

Sử dụng 4 khung trang , sẽ có 10 lỗi trang phát sinh

1	2	3	4	1	2	5	1	2	3	4	5
1	1	1	1	1	1	5	5	5	5	4	4
	2	2	2	2	2	2	1	1	1	1	5
		3	3	3	3	3	3	2	2	2	2
			4	4	4	4	4	4	3	3	3
*	*	*	*			*	*	*	*	*	*

Thuật toán tối ưu

Tiếp cận: Thay thế trang sẽ lâu được sử dụng nhất trong tương lai.

Ví dụ : sử dụng 3 khung trang, khởi đầu đều trống:

7	**0**	**1**	**2**	**0**	**3**	**0**	**4**	**2**	**3**	**0**	**3**	**2**	**1**	**2**	**0**	**1**	**7**	**0**	**1**
7	7	7	2	2	2	2	2	2	2	2	2	2	2	2	2	2	7	7	7
	0	0	0	0	0	0	4	4	4	0	0	0	0	0	0	0	0	0	0
		1	1	1	3	3	3	3	3	3	3	3	1	1	1	1	1	1	1
*	*	*	*		*		*			*			*				*		

Thảo luận:

Thuật toán này bảo đảm số lượng lỗi trang phát sinh là thấp nhất , nó cũng không gánh chịu nghịch lý Belady, tuy nhiên, đây là một thuật toán không khả thi trong thực tế, vì không thể biết trước chuỗi truy xuất của tiến trình!

Thuật toán « Lâu nhất chưa sử dụng » (Least-recently-used LRU)

Tiếp cận: Với mỗi trang, ghi nhận thời điểm cuối cùng trang được truy cập, trang được chọn để thay thế sẽ là trang lâu nhất chưa được truy xuất.

Ví dụ: sử dụng 3 khung trang, khởi đầu đều trống:

7	**0**	**1**	**2**	**0**	**3**	**0**	**4**	**2**	**3**	**0**	**3**	**2**	**1**	**2**	**0**	**1**	**7**	**0**	**1**
7	7	7	2	2	2	2	4	4	4	0	0	0	1	1	1	1	1	1	1
	0	0	0	0	0	0	0	0	3	3	3	3	3	3	0	0	0	0	0
		1	1	1	3	3	3	2	2	2	2	2	2	2	2	2	7	7	7
*	*	*	*		*		*	*	*	*			*		*		*		

Thảo luận:

- Thuật toán FIFO sử dụng thời điểm nạp để chọn trang thay thế, thuật toán tối ưu lại dùng thời điểm trang *sẽ* được sử dụng, vì thời điểm này không thể xác định trước nên thuật toán LRU phải dùng thời điểm cuối cùng trang được truy xuất – dùng quá khứ gần để dự đoán tương lai.
- Thuật toán này đòi hỏi phải được cơ chế phần cứng hỗ trợ để xác định một thứ tự cho các trang theo thời điểm truy xuất cuối cùng. Có thể cài đặt theo một trong hai cách:
    - **Sử dụng bộ đếm:**
        - thêm vào cấu trúc của mỗi phần tử trong bảng trang một trường ghi nhận thời điểm truy xuất mới nhất, và thêm vào cấu trúc của CPU một bộ đếm.
        - mỗi lần có sự truy xuất bộ nhớ, giá trị của counter tăng lên 1.
        - Mỗi lần thực hiện truy xuất đến một trang, giá trị của counter được ghi nhận vào trường thời điểm truy xuất mới nhất của phần tử tương ứng với trang trong bảng trang.
        - thay thế trang có giá trị trường thời điểm truy xuất mới nhất là nhỏ nhất.
    - **Sử dụng stack:**
        - tổ chức một stack lưu trữ các số hiệu trang
        - mỗi khi thực hiện một truy xuất đến một trang, số hiệu của trang sẽ được xóa khỏi vị trí hiện hành trong stack và đưa lên đầu stack.
        - trang ở đỉnh stack là trang được truy xuất gần nhất, và trang ở đáy stack là trang lâu nhất chưa được sử dụng.

Các thuật toán xấp xỉ LRU

Có ít hệ thống được cung cấp đủ các hỗ trợ phần cứng để cài đặt được thuật toán LRU thật sự. Tuy nhiên, nhiều hệ thống được trang bị thêm một bit *tham khảo* (reference):

- một bit reference, được khởi gán là 0, được gắn với một phần tử trong bảng trang.
- bit reference của một trang được phần cứng đặt giá trị 1 mỗi lần trang tương ứng được truy cập, và được phần cứng gán trở về 0 sau từng chu kỳ qui định trước.
- Sau từng chu kỳ qui định trước, kiểm tra giá trị của các bit reference, có thể xác định được trang nào đã được truy xuất đến và trang nào không, sau khi đã kiểm tra xong, các bit reference được phần cứng gán trở về 0 .

- với bit reference, có thể biết được trang nào đã được truy xuất, nhưng không biết được thứ tự truy xuất. Thông tin không đầy đủ này dẫn đến nhiều thuật toán xấp xỉ LRU khác nhau.

số hiệu trang	Bit valid-invalid	dirty bit	bit reference

Hình 6.5.2.2-1. Cấu trúc một phần tử trong bảng trang

***a)* Thuật toán với các bit reference phụ trợ**

Tiếp cận: Có thể thu thập thêm nhiều thông tin về thứ tự truy xuất hơn bằng cách lưu trữ các bit references sau từng khoảng thời gian đều đặn:

- với mỗi trang, sử dụng thêm 8 bit lịch sử (history)trong bảng trang
- sau từng khoảng thời gian nhất định (thường là100 millisecondes), một ngắt đồng hồ được phát sinh, và quyền điều khiển được chuyển cho hệ điều hành. Hệ điều hành đặt bit reference của mỗi trang vào bit cao nhất trong 8 bit phụ trợ củatrang đó bằng cách đẩy các bit khác sang phải 1 vị trí, bỏ luôn bit thấp nhất.
- như vậy 8 bit thêm vào này sẽ lư u trữ tình hình truy xuất đến trang trong 8 chu kỳ cuối cùng.
- nếu gía trị của 8 bit là 00000000, thì trang tương ứng đã không được dùng đến suốt 8 chu kỳ cuối cùng, ngược lại nếu nó được dùng đến ít nhất 1 lần trong mỗi chu kỳ, thì 8 bit phụ trợ sẽ là 11111111. Một trang mà 8 bit phụ trợ có giá trị11000100 sẽ được truy xuất gần thời điểm hiện tại hơn trang có 8 bit phụ trợ là 01110111.
- nếu xét 8 bit phụ trợ này như một số nguyên không dấu, thì trang LRU là trang có số phụ trợ nhỏ nhất.

Ví dụ :

	0	0	1	0	0	0	1	1	1	0
HR =11000100										
HR =11100010										
HR =01110001										

Thảo luận: Số lượng các bit lịch sử có thể thay đổi tùy theo phần cứng, và phải được chọn sao cho việc cập nhật là nhanh nhất có thể.

***b)* Thuật toán « cơ hội thứ hai »**

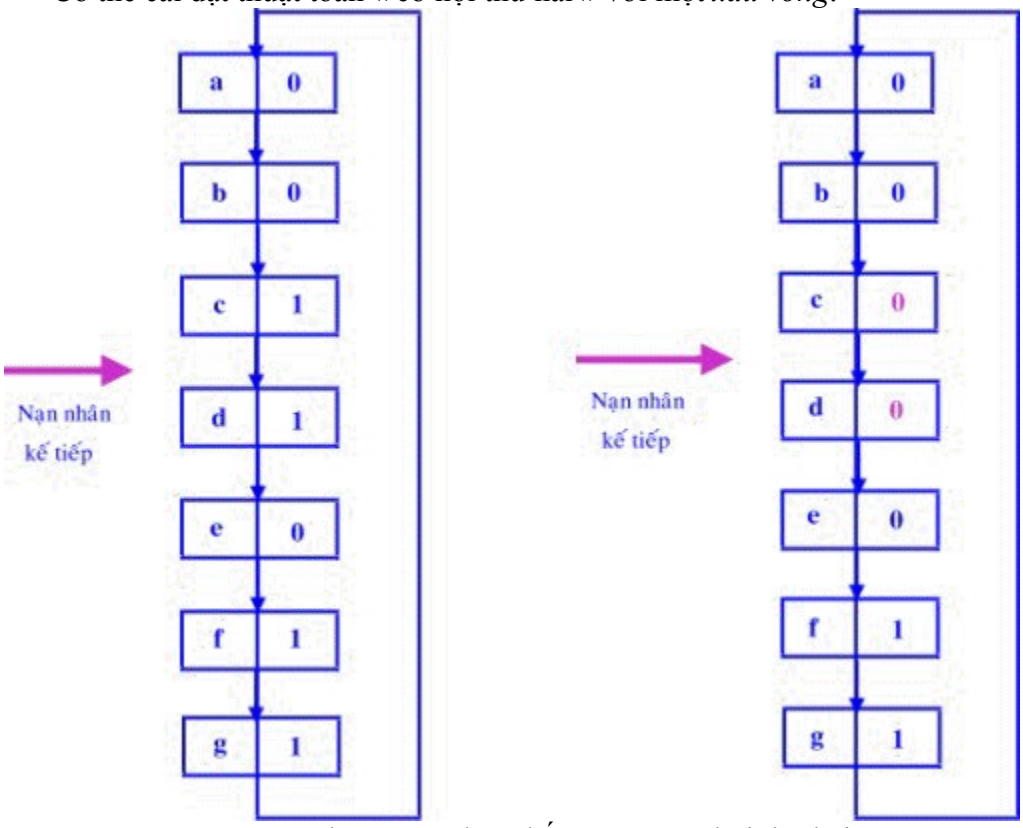
Tiếp cận: Sử dụng một bit reference duy nhất. Thuật toán cơ sở vẫn là FIFO, tuy nhiên khi chọn được một trang theo tiêu chuẩn FIFO, kiểm tra bit reference của trang đó :

- Nếu giá trị của bit reference là 0, thay thế trang đã chọn.
- Ngược lại, cho trang này một cơ hội thứ hai, và chọn trang FIFO tiếp theo.
- Khi một trang được cho cơ hội thứ hai, giá trị của bit reference được đặt lại là 0, và thời điểm vào Ready List được cập nhật lại là thời điểm hiện tại.
- Một trang đã được cho cơ hội thứ hai sẽ không bị thay thế trước khi hệ thống đã thay thế hết những trang khác. Hơn nữa, nếu trang thường xuyên được sử dụng,

bit reference của nó sẽ duy trì được giá trị 1, và trang hầu như không bao giờ bị thay thế.

Thảo luận:

Có thể cài đặt thuật toán « cơ hội thứ hai » với một *xâu vòng*.



Hình 6.5.2.2-2. Thuật toán thay thế trang <<cơ hội thứ hai >>

***c)* Thuật toán « cơ hội thứ hai » nâng cao (Not Recently Used - NRU)**

Tiếp cận : xem các bit reference và dirty bit như một cặp có thứ tự .

- Với hai bit này, có thể có 4 tổ hợp tạo thành 4 lớp sau :
  - (0,0) không truy xuất, không sửa đổi: đây là trang tốt nhất để thay thế.
  - (0,1) không truy xuất gần đây, nhưng đã bị sửa đổi: trường hợp này không thật tốt, vì trang cần được lưu trữ lại trước khi thay thế.
  - (1,0) được truy xuất gần đây, nhưng không bị sửa đổi: trang có thể nhanh chóng được tiếp tục được sử dụng.
  - (1,1) được truy xuất gần đây, và bị sửa đổi: trang có thể nhanh chóng được tiếp tục được sử dụng, và trước khi thay thế cần phải được lưu trữ lại.
- lớp 1 có độ ưu tiên thấp nhất, và lớp 4 có độ ưu tiên cao nhất.
- một trang sẽ thuộc về một trong bốn lớp trên, tuỳ vào bit reference và dirty bit của trang đó.
- trang được chọn để thay thế là trang đầu tiên tìm thấy trong lớp có độ ưu tiên thấp nhất và khác rỗng.

CHƯƠNG 7. HỆ THỐNG QUẢN LÝ TẬP TIN

Trong hầu hết các ứng dụng, tập tin là thành phần chủ yếu. Cho dù mục tiêu của ứng dụng là gì nó cũng phải bao gồm phát sinh và sử dụng thông tin. Thông thường đầu vào của các ứng dụng là tập tin và đầu ra cũng là tập tin cho việc truy xuất của người sử dụng và các chương trình khác sau này. Trong bài học này chúng ta sẽ tìm hiểu những khái niệm và cơ chế của hệ thống quản lý tập tin thông qua các nội dung như sau:

- Các khái niệm cơ bản
- Mô hình tổ chức và quản lý các tập tin

Chương này đề nhằm đưa ra các vấn đề về tệp tin: khái niệm, cách thức tổ chức và quản lý tập tin như thế nào. Từ đó giúp hiểu được các cơ chế cài đặt hệ thống tập tin trên các hệ điều hành.

7.1. CÁC KHÁI NIỆM CƠ BẢN

7.1.1. Bộ nhớ ngoài

Máy tính phải sử dụng thiết bị có khả năng lưu trữ trong thời gian dài (long-term) vì :
Phải chứa những lượng thông tin rất lớn (giữ vé máy bay, ngân hàng...)
Thông tin phải được lưu giữ một thời gian dài trước khi xử lý
Nhiều tiến trình có thể truy cập thông tin cùng lúc.
Giải pháp là sử dụng các thiết bị lưu trữ bên ngoài gọi là bộ nhớ ngoài.

7.1.2. Tập tin và thư mục

Tập tin

Tập tin là đơn vị lưu trữ thông tin của bộ nhớ ngoài. Các tiến trình có thể đọc hay tạo mới tập tin nếu cần thiết. Thông tin trên tập tin là vững bền không bị ảnh hưởng bởi các xử lý tạo hay kết thúc các tiến trình, chỉ mất đi khi user thật sự muốn xóa. Tập tin được quản lý bởi hệ điều hành.

Thư mục

Để lưu trữ dãy các tập tin, hệ thống quản lý tập tin cung cấp thư mục, mà trong nhiều hệ thống có thể coi như là tập tin.

7.1.3. Hệ thống quản lý tập tin

Các tập tin được quản lý bởi hệ điều hành với cơ chế gọi là hệ thống quản lý tập tin. Bao gồm : cách hiển thị, các yếu tố cấu thành tập tin, cách đặt tên, cách truy xuất, cách sử dụng và bảo vệ tập tin, các thao tác trên tập tin. Cách tổ chức thư mục, các đặc tính và các thao tác trên thư mục.

7.2. MÔ HÌNH TỔ CHỨC VÀ QUẢN LÝ CÁC TẬP TIN

7.2.1. Mô hình

7.2.1.1. Tập tin

Tên tập tin :

Tập tin là một cơ chế trừu tượng và để quản lý mỗi đối tượng phải có một tên. Khi tiến trình tạo một tập tin, nó sẽ đặt một tên, khi tiến trình kết thúc tập tin vẫn tồn tại và có thể được truy xuất bởi các tiến trình khác với tên tập tin đó.

Cách đặt tên tập tin của mỗi hệ điều hành là khác nhau, đa số các hệ điều hành cho phép sử dụng 8 chữ cái để đặt tên tập tin như ctdl, caycb, tamhghau v.v…, thường thường thì các ký tự số và ký tự đặc biệt cũng được sử dụng như baitap2…,

Hệ thống tập tin có thể có hay không phân biệt chữ thường và chữ hoa. Ví dụ : UNIX phân biệt chữ thường và hoa còn MS-DOS thì không phân biệt.

Nhiều hệ thống tập tin hỗ trợ tên tập tin gồm 2 phần được phân cách bởi dấu '.' mà phần sau được gọi là phần mở rộng. Ví dụ : vidu.txt. Trong MS-DOS tên tập tin có từ 1 đến 8 ký tư, phần mở rộng có từ 1 đến 3 ký tự. Trong UNIX có thể có nhiều phân cách như prog.c.Z. Một số kiểu mở rộng thông thường là :

.bak, .bas, .bin, .c, .dat, .doc, .ftn, .hlp, .lib, .obj, .pas, .tex, .txt.

Trên thực tế phần mở rộng có hữu ích trong một số trường hợp, ví dụ như có những trình dịch C chỉ nhận biết các tập tin có phần mở rộng là .C

Cấu trúc của tập tin :

Gồm 3 loại :

- Dãy tuần tự các byte không cấu trúc : hệ điều hành không biết nội dung của tập tin:MS-DOS và UNIX sử dụng loại này.
- Dãy các record có chiều dài cố định.
- Cấu trúc cây : gồm cây của những record, không cần thiết có cùng độ dài, mỗi record có một trường khóa giúp cho việc tìm kiếm nhanh hơn.

Kiểu tập tin :

Nếu hệ điều hành nhận biết được loại tập tin, nó có thể thao tác một cách hợp lý trên tập tin đó. Các hệ điều hành hỗ trợ cho nhiều loại tập tin khác nhau bao gồm các kiểu như : tập tin thường, thư mục, tập tin có ký tự đặc biệt, tập tin khối.

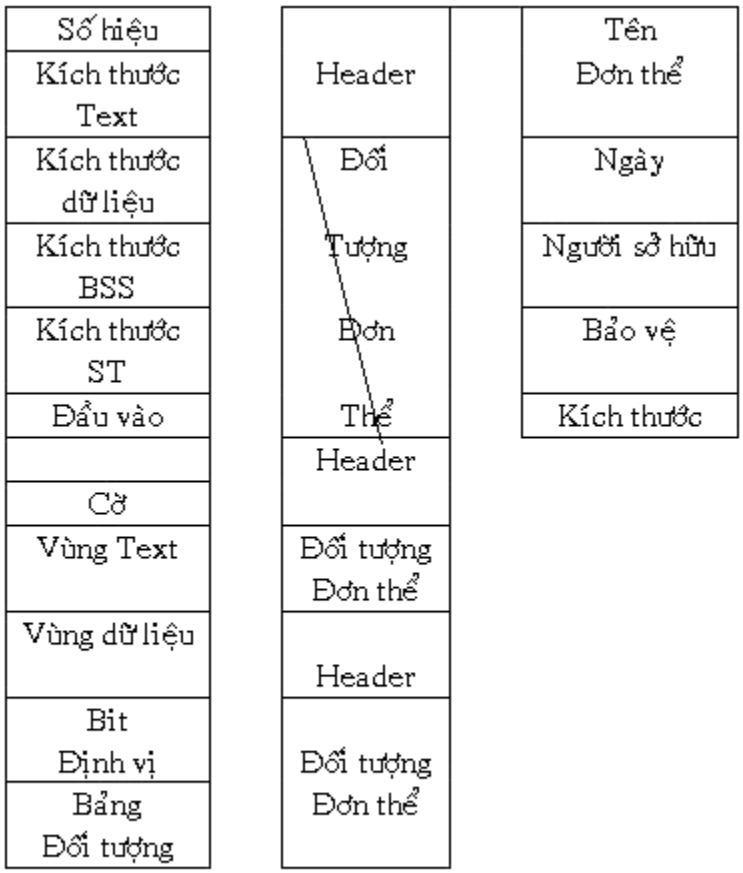
- *Tập tin thường* : là tập tin text hay tập tin nhị phân chứa thông tin của người sử dụng.
- *Thư mục* : là những tập tin hệ thống dùng để lưu giữ cấu trúc của hệ thống tập tin.
- *Tập tin có ký tự đặc biệt* : liên quan đến nhập xuất thông qua các thiết bị nhập xuất tuần tự như màn hình, máy in, mạng.
- *Tập tin khối* : dùng để truy xuất trên thiết bị đĩa.
- Tập tin thường được chia làm hai loại là tập tin văn bản và tập tin nhị phân.

Tập tin văn bản chứa các dòng văn bản cuối dòng có ký hiệu enter. Mỗi dòng có độ dài có thể khác nhau. Ưu điểm của kiểu tập tin này là nó có thể hiển thị, in hay soạn thảo với một editor thông thường.Đa số các chương trình dùng tập tin văn bản để nhập xuất, nó cũng dễ dàng làm đầu vào và đầu ra cho cơ chế pipeline.

Tập tin nhị phân : có cấu trúc khác tập tin văn bản. Mặc dù về mặt kỹ thuật , tập tin nhị phân gồm dãy các byte , nhưng hệ điều hành chỉ thực thi tập tin đó nếu nó có cấu trúc

đúng. Ví dụ một một tập tin nhị phân thi hành được của UNIX. Thường thường nó bao gồm năm thành phần : header, text, data, relocation bits, symbol table. Header bắt đầu bởi byte nhận diện cho biết đó là tập tin thi hành. Sau đó là 16 bit cho biết kích thước các thành phần của tập tin, địa chỉ bắt đầu thực hiện và một số bit cờ. Sau header là dữ liệu và text của tập tin. Nó được nạp vào bộ nhớ và định vị lại bởi những bit relocation. Bảng symbol được dùng để debug.

Một ví dụ khác là tập tin nhị phân kiểu archive. Nó chứa các thư viện đã được dịch nhưng chưa được liên kết. Bao gồm một header cho biết tên, ngày tạo, người sở hữu, mã bảo vệ, và kích thước…



Hình 8.1 Cấu trúc tập tin nhị phân trong UNIX

Hình 7.2.1.1-1. Cấu trúc file trong UNIX

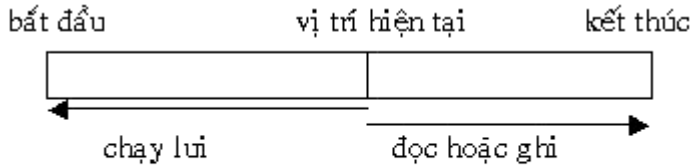
Truy xuất tập tin :

Tập tin lưu trữ các thông tin. Khi tập tin được sử dụng, các thông tin này được đưa vào bộ nhớ của máy tính. Có nhiều cách để truy xuất chúng. Một số hệ thống cung cấp chỉ một phương pháp truy xuất, một số hệ thống khác, như IBM chẳng hạn cho phép nhiều cách truy xuất.

Kiểu truy xuất tập tin đơn giản nhất là truy xuất tuần tự . Tiến trình đọc tất cả các byte trong tập tin theo thứ tự từ đầu. Các trình soạn thảo hay trình biên dịch cũng truy xuất tập tin theo cách này. Hai thao tác chủ yếu trên tập tin là đọc và ghi. Thao tác đọc sẽ đọc một mẫu tin tiếp theo trên tập tin và tự động tăng con trỏ tập tin. Thao tác ghi cũng

tương tự như vậy. Tập tin có thể tự khởi động lại từ vị trí đầu tiên và trong một số hệ thống tập tin cho phép di chuyển con trỏ tập tin đi tới hoặc đi lui n mẫu tin.

Truy xuất kiểu này thuận lợi cho các loại băng từ và cũng là cách truy xuất khá thông dụng. Truy xuất tuần tự cần thiết cho nhiều ứng dụng. Có hai cách truy xuất. Cách truy xuất thứ nhất thao tác đọc bắt đầu ở vị trí đầu tập tin, cách thứ hai có một thao tác đặc biệt gọi là SEEK cung cấp vị trí hiện thời làm vị trí bắt đầu. Sau đó tập tin được đọc tuần tự từ vị trí bắt đầu.



Hình 8.2 Truy xuất tuần tự trên tập tin

Hình 7.2.1.1-2. Truy xuất tuần tự trên File

Một kiểu truy xuất khác là truy xuất trực tiếp. Một tập tin có cấu trúc là các mẫu tin logic có kích thước bằng nhau, nó cho phép chương trình đọc hoặc ghi nhanh chóng mà không cần theo thứ tự. Kiểu truy xuất này dựa trên mô hình của đĩa. Đĩa cho phép truy xuất ngẫu nhiên bất kỳ khối dữ liệu nào của tập tin. Truy xuất trực tiếp được sử dụng trong trường hợp phải truy xuất một khối lượng thông tin lớn như trong cơ sở dữ liệu chẳng hạn. Ngoài ra còn có một số cách truy xuất khác dự trên kiểu truy xuất này như truy xuất theo chỉ mục ...

Thuộc tính tập tin :

Ngoài tên và dữ liệu, hệ điều hành cung cấp thêm một số thông tin cho tập tin gọi là thuộc tính.

Các thuộc tính thông dụng trong một số hệ thống tập tin :

Tên thuộc tính	Ý nghĩa
Bảo vệ	Ai có thể truy xuất được và bằng cách nào
Mật khẩu	Mật khẩu cần thiết để truy xuất tập tin
Người tạo	Id của người tạo tập tin
Người sở hữu	Người sở hữu hiện tại
Chỉ đọc	0 là đọc ghi, 1 là chỉ đọc
Aản	0 là bình thường, 1 là không hiển thị khi liệt kê
Hệ thống	0 là bình thường, 1 là tập tin hệ thống
Lưu trữ	0 đã được backup, 1 cần backup
ASCII/binary	0 là tập tin văn bản, 1 là tập tin nhị phân

Truy xuất ngẫu nhiên	0 truy xuất tuần tự, 1 là truy xuất ngẫu nhiên
Temp	0 là bình thường, 1 là bị xóa khi tiến trình kết thúc
Khóa	0 là không khóa, khác 0 là khóa
Độ dài của record	Số byte trong một record
Vị trí khóa	Offset của khóa trong mỗi record
Giờ tạo	Ngày và giờ tạo tập tin
Thời gian truy cập cuối cùng	Ngày và giờ truy xuất tập tin gần nhất
Thời gian thay đổi cuối cùng	Ngày và giờ thay đổi tập tin gần nhất
Kích thước hiện thời	Số byte của tập tin
Kích thước tối đa.	Số byte tối đa của tập tin

7.2.1.2. Thư mục:

HỆ THỐNG THƯ MỤC THEO CẤP BẬC :

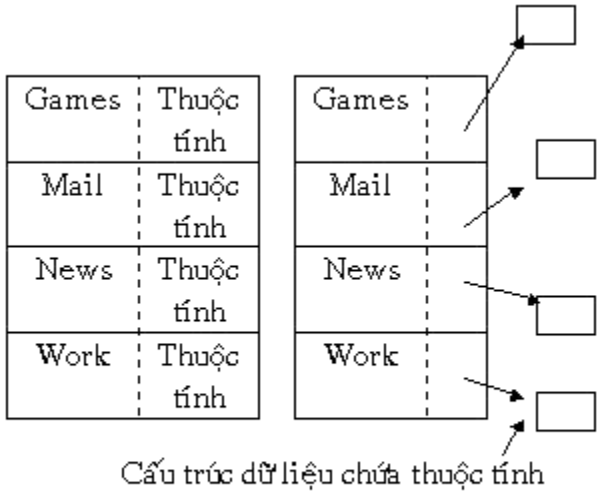
Một thư mục thường thường chứa một số *entry*, mỗi entry cho một tập tin. Mỗi entry chứa tên tập tin, thuộc tính và địa chỉ trên đĩa lưu dữ liệu hoặc một entry chỉ chứa tên tập tin và một con trỏ, trỏ tới một cấu trúc, trên đó có thuộc tính và vị trí lưu trữ của tập tin.

Khi một tập tin được mở, hệ điều hành tìm trên thư mục của nó cho tới khi tìm thấy tên của tập tin được mở. Sau đó nó sẽ xác định thuộc tính cũng như địa chỉ lưu trữ trên đĩa và đưa vào một bảng trong bộ nhớ. Những truy xuất sau đó thực hiện trong bộ nhớ chính.

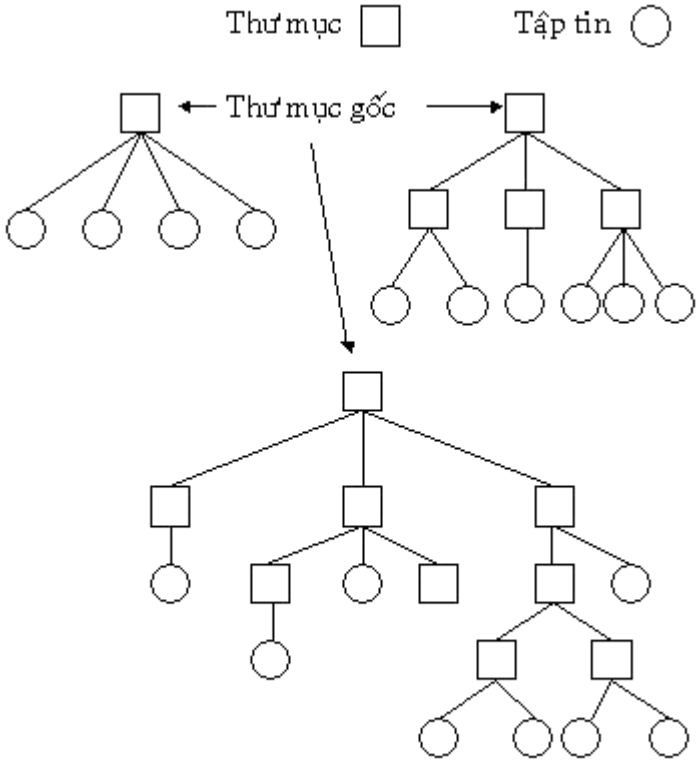
Số lượng thư mục trên mỗi hệ thống là khác nhau. Thiết kế đơn giản nhất là hệ thống chỉ có thư mục đơn(còn gọi là thư mục một cấp), chứa tất cả các tập tin của tất cả người dùng, cách này dễ tổ chức và khai thác nhưng cũng dễ gây ra khó khăn khi có nhiều người sử dụng vì sẽ có nhiều tập tin trùng tên. Ngay cả trong trường hợp chỉ có một người sử dụng, nếu có nhiều tập tin thì việc đặt tên cho một tập tin mới không trùng lắp là một vấn đề khó.

Cách thứ hai là có một thư mục gốc và trong đó có nhiều thư mục con, trong mỗi thư mục con chứa tập tin của người sử dụng (còn gọi là thư mục hai cấp), cách này tránh được trường hợp xung đột tên nhưng cũng còn khó khăn với người dùng có nhiều tập tin. Người sử dụng luôn muốn nhóm các ứng dụng lại một cách logic.

Từ đó, hệ thống thư mục theo cấp bậc (còn gọi là cây thư mục) được hình thành với mô hình một thư mục có thể chứa tập tin hoặc một thư mục con và cứ tiếp tục như vậy hình thành cây thư mục như trong các hệ điều hành DOS, Windows, v. v...
Ngoài ra, trong một số hệ điều hành nhiều người dùng, hệ thống còn xây dựng các hình thức khác của cấu trúc thư mục như cấu trúc thư mục theo đồ thị có chu trình và cấu trúc thư mục theo đồ thị tổng quát. Các cấu trúc này cho phép các người dùng trong hệ thống có thể liên kết với nhau thông qua các thư mục chia sẻ.



Hình 8.4 Hai dạng cấu trúc thư mục



Hình 8.5 Hệ thống thư mục theo cấp bậc.

Hình 7.2.1.2-1. Hệ thống phân cấp thư mục

ĐƯỜNG DẪN :

Khi một hệ thống tập tin được tổ chức thành một ***cây thư mục***, có hai cách để xác định một tên tập tin. Cách thứ nhất là ***đường dẫn tuyệt đối***, mỗi tập tin được gán một đường dẫn từ thư mục gốc đến tập tin. Ví dụ : /usr/ast/mailbox.

Dạng thứ hai là ***đường dẫn tương đối***, dạng này có liên quan đến một khái niệm là ***thư mục hiện hành*** hay thư mục làm việc. Người sử dụng có thể quy định một thư mục là thư mục hiện hành. Khi đó đường dẫn không bắt đầu từ thư mục gốc mà liên quan đến thư mục hiện hành. Ví dụ, nếu thư mục hiện hành là /usr/ast thì tập tin với đường dẫn tuyệt đối /usr/ast/mailbox có thể được dùng đơn giản là mailbox.

Trong phần lớn hệ thống, mỗi tiến trình có một thư mục hiện hành riêng, khi một tiến trình thay đổi thư mục làm việc và kết thúc, không có sự thay đổi để lại trên hệ thống tập tin. Nhưng nếu một hàm thư viện thay đổi đường dẫn và sau đó không đổi lại thì sẽ có ảnh hưởng đến tiến trình.

Hầu hết các hệ điều hành đều hỗ trợ hệ thống thư mục theo cấp bậc với hai entry đặc biệt cho mỗi thư mục là "." và "..". "." chỉ thư mục hiện hành, ".." chỉ thư mục cha.

7.2.2. Các chức năng

7.2.2.1. Tập tin

- *Tạo* : một tập tin được tạo chưa có dữ liệu. Mục tiêu của chức năng này là thông báo cho biết rằng tập tin đã tồn tại và thiết lập một số thuộc tính.
- *Xóa* :khi một tập tin không còn cần thiết nữa, nó được xóa để tăng dung lượng đĩa. Một số hệ điều hành tự động xoá tập tin sau một khoảng thời gian n ngày.
- *Mở* : trước khi sử dụng một tập tin, tiến trình phải mở nó. Mục tiêu của mở là cho phép hệ thống thiết lập một số thuộc tính và địa chỉ đĩa trong bộ nhớ để tăng tốc độ truy xuất.
- *Đóng* : khi chấm dứt truy xuất, thuộc tính và địa chỉ trên đĩa không cần dùng nữa, tập tin được đóng lại để giải phóng vùng nhớ. Một số hệ thống hạn chế tối đa số tập tin mở trong một tiến trình.
- *Đọc* : đọc dữ liệu từ tập tin tại vị trí hiện thời của đầu đọc, nơi gọi sẽ cho biết cần bao nhiêu dữ liệu và vị trí của buffer lưu trữ nó.
- *Ghi* : ghi dữ liệu lên tập tin từ vị trí hiện thời của đầu đọc. Nếu là cuối tập tin,kích thước tập tin sẽ tăng lên, nếu đang ở giữa tập tin, dữ liệu sẽ bị ghi chồng lên.
- *Thêm* : gần giống như WRITE nhưng dữ liệu luôn được ghi vào cuối tập tin.
- *Tìm* :dùng để truy xuất tập tin ngẫu nhiên. Khi xuất hiện lời gọi hệ thống, vị trí con trỏ đang ở vị trí hiện hành được di chuyển tới vị trí cần thiết. Sau đó dữ liệu sẽ được đọc ghi tại vị trí này.
- *Lấy thuộc tính* :lấy thuộc tính của tập tin cho tiến trình
- *Thiết lập thuộc tính* :thay đổi thuộc tính của tập tin sau một thời gian sử dụng.
- *Đổi tên* :thay đổi tên của tập tin đã tồn tại.

7.2.2.2. Thư mục

- *Tạo* : một thư mục được tạo, nó rỗng, ngoại trừ "." và ".." được đặt tự động bởi hệ thống.

- *Xóa* :xoá một thư mục, chỉ có thư mục rỗng mới bị xóa, tư mục chứa "." và ".." coi như là thư mục rỗng.
- *Mở thư mục* :thư mục có thể được đọc. Ví dụ để liệt kê tất cả tập tin trong một thư mục, chương trình liệt kê mở thư mục và đọc ra tên của tất cả tập tin chứa trong đó. Trước khi thư mục được đọc, nó phải được mở ra trước.
- *Đóng thư mục* :khi một thư mục đã được đọc xong, phải đóng thư mục để giải phóng vùng nhớ.
- *Đọc thư mục* :Lệnh này trả về entry tiếp theo trong thư mục đã mở. Thông thường có thể đọc thư mục bằng lời gọi hệ thống READ, lệnh đọc thư mục luôn luôn trả về một entry dưới dạng chuẩn .
- *Đổi tên* :cũng như tập tin, thư mục cũng có thể được đổi tên.
- *Liên kết* :kỹ thuật này cho phép một tập tin có thể xuất hiện trong nhiều thư mục khác nhau. Khi có yêu cầu, một liên kết sẽ được tạo giữa tập tin và một đường dẫn được cung cấp.
- *Bỏ liên kết* :Nếu tập tin chỉ còn liên kết với một thư mục, nó sẽ bị loại bỏ hoàn toàn khỏi hệ thống, nếu nhiều thì nó bị giảm chỉ số liên kết.

7.3. CÁC PHƯƠNG PHÁP CÀI ĐẶT HỆ THỐNG QUẢN LÝ TẬP TIN

Người sử dụng thì quan tâm đến cách đặt tên tập tin, các thao tác trên tập tin, cây thư mục...Nhưng đối người cài đặt thì quan tâm đến tập tin và thư mục được lưu trữ như thế nào, vùng nhớ trên đĩa được quản lý như thế nào và làm sao cho toàn bộ hệ thống làm việc hữu hiệu và tin cậy. Hệ thống tập tin được cài đặt trên đĩa. Để gia tăng hiệu quả trong việc truy xuất, mỗi đơn vị dữ liệu được truy xuất gọi là một khối. Một khối dữ liệu bao gồm một hoặc nhiều sector. Bộ phận tổ chức tập tin quản lý việc lưu trữ tập tin trên những khối vật lý bằng cách sử dụng các bảng có cấu trúc. Trong bài học này chúng ta sẽ tìm hiểu các phương pháp tổ chức quản lý tập tin trên bộ nhớ phụ thông qua các nội dung như sau:

- Bảng quản lý thư mục, tập tin
- Bảng phân phối vùng nhớ
- Tập tin chia sẻ
- Quản lý đĩa
- Độ an toàn của hệ thống tập tin

Chương này đưa ra các đặc điểm cũng như ưu và khuyết điểm của các phương pháp tổ chức quản lý tập tin trên đĩa và một số vấn đề liên quan khác nhờ đó có thể hiểu được cách các hệ điều hành cụ thể quản lý tập tin như thế nào.
Bài học này đòi hỏi những kiến thức về :mô hình tổ chức các tập tin và thư mục cũng và một số cấu trúc dữ liệu.

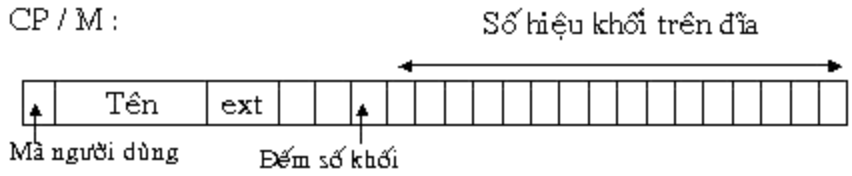
7.3.1. Bảng quản lý tệp tin, thư mục

7.3.1.1. Khái niệm

Trước khi tập tin được đọc, tập tin phải được mở, để mở tập tin hệ thống phải biết đường dẫn do người sử dụng cung cấp và được định vị trong cấu trúc đầu vào thư mục (directory entry). Directory entry cung cấp các thông tin cần thiết để tìm kiếm các khối. Tuỳ thuộc vào mỗi hệ thống, thông tin là địa chỉ trên đĩa của toàn bộ tập tin, số hiệu của khối đầu tiên, hoặc là số I-node.

7.3.1.2. Cài đặt

Bảng này thường được cài đặt ở phần đầu của đĩa. Bảng là dãy các phần tử có kích thước xác định, mỗi phần tử được gọi là một entry. Mỗi entry sẽ lưu thông tin về tên , thuộc tính, vị trí lưu trữ .... của một tập tin hay thư mục.
Ví dụ quản lý thư mục trong CP/M :



Hình 9.1

7.3.2. Bảng phân phối vùng nhớ

7.3.2.1. Khái niệm

Bảng này thường được sử dụn phối hợp với bảng quản lý thư mục tập tin, mục tiêu là cho biết vị trí khối vật lý của một tập tin hay thư mục nào đó nói khác đi là lưu giữ dãy các khối trên đĩa cấp phát cho tập tin lưu dữ liệu hay thư mục. Có một số phương pháp được cài đặt.

7.3.2.2. Các phương pháp

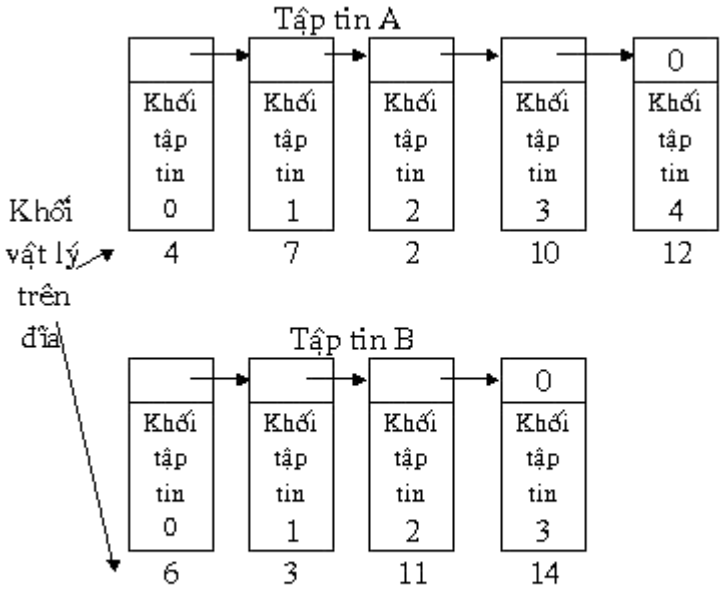
Định vị liên tiếp:

Lưu trữ tập tin trên dãy các khối liên tiếp.

Phương pháp này có 2 ưu điểm : thứ nhất, dể dàng cài đặt. Thứ hai, dể dàng thao tác vì toàn bộ tập tin được đọc từ đĩa bằng thao tác đơn giản không cần định vị lại.

Phương pháp này cũng có 2 khuyết điểm : không linh động trừ khi biết trước kích thước tối đa của tập tin. Sự phân mảnh trên đĩa, gây lãng phí lớn.

Định vị bằng danh sách liên kết:



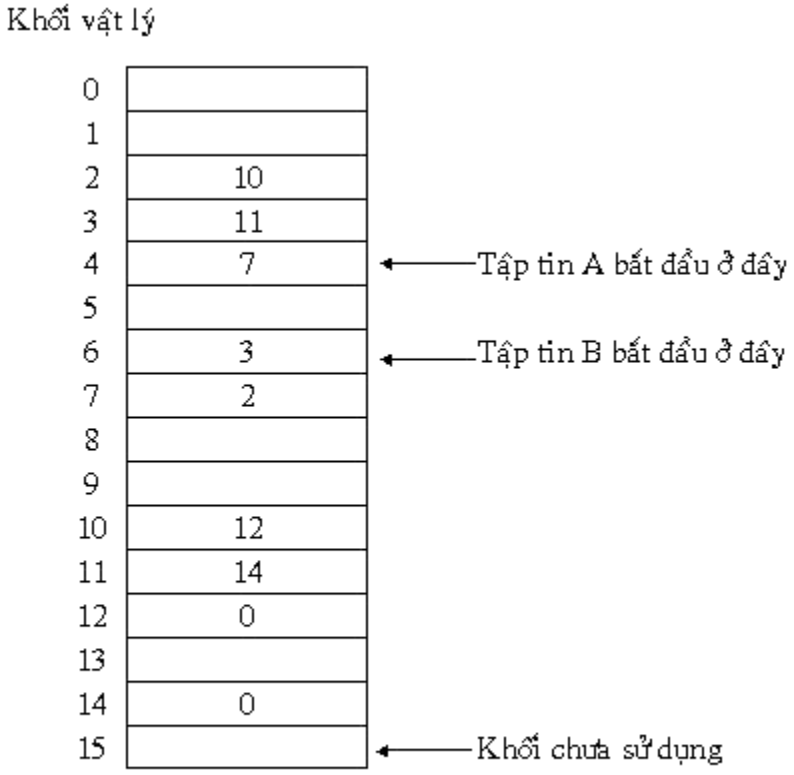
Hình 9.2 Định vị bằng danh sách liên kết

Hình 7.3.2.2-1. Định vị liên kết

Mọi khối đều được cấp phát, không bị lãng phí trong trường hợp phân mảnh và directory entry chỉ cần chứa địa chỉ của khối đầu tiên.

Tuy nhiên khối dữ liệu bị thu hẹp lại và truy xuất ngẫu nhiên sẽ chậm.

Danh sách liên kết sử dụng index :



Hình 9.3 Bảng chỉ mục của danh sách liên kết

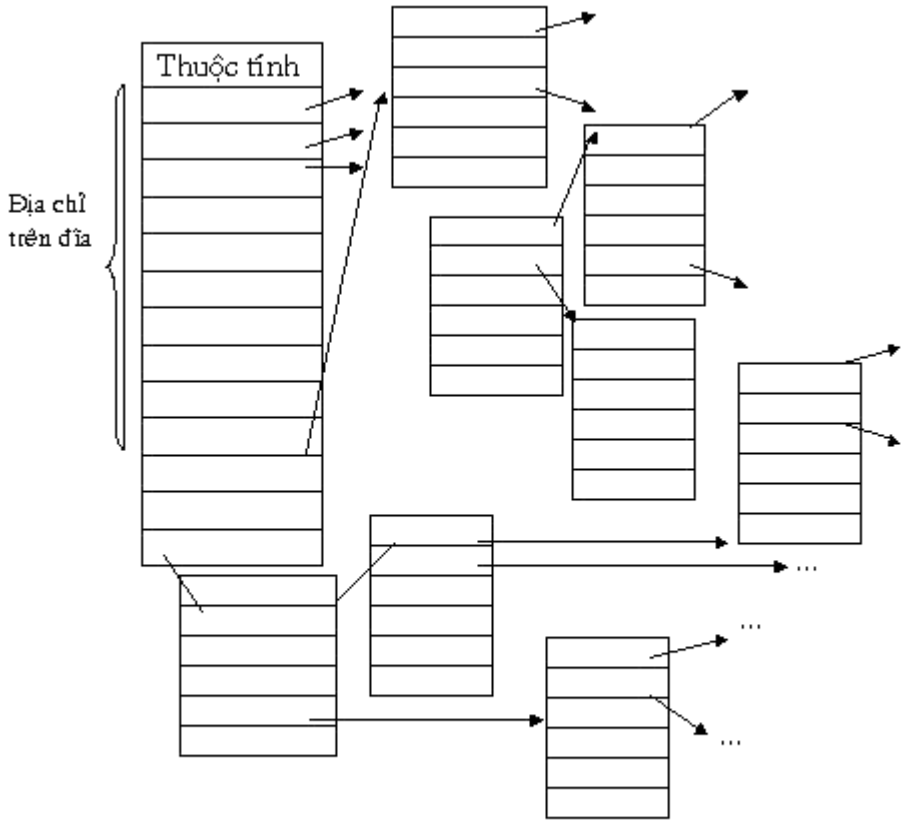
Hình 7.3.2.2-2. Bảng chỉ mục của danh sách liên kết

Tương tự như hai nhưng thay vì dùng con trỏ thì dùng một bảng index. Khi đó toàn bộ khối chỉ chứa dữ liệu. Truy xuất ngẫu nhiên sẽ dễ dàng hơn. Kích thước tập tin được mở rộng hơn. Hạn chế là bản này bị giới hạn bởi kích thước bộ nhớ .

I-nodes :

Một I-node bao gồm hai phần. Phần thứ nhất là thuộc tính của tập tin. Phần này lưu trữ các thông tin liên quan đến tập tin như kiểu, người sở hữu, kích thước, v.v...Phần thứ hai chứa địa chỉ của khối dữ liệu. Phần này chia làm hai phần nhỏ. Phần nhỏ thứ nhất bao gồm 10 phần tử, mỗi phần tử chứa địa chỉ khối dữ liệu của tập tin. Phần tử thứ 11 chứa địa chỉ gián tiếp cấp 1 (single indirect), chứa địa chỉ của một khối, trong khối đó chứa một bảng có thể từ 2^{10} đến 2^{32} phần tử mà mỗi phần tử mới chứa địa chỉ của khối dữ liệu. Phần tử thứ 12 chứa địa chỉ gián tiếp cấp 2 (double indirect), chứa địa chỉ của bảng các khối single indirect. Phần tử thứ 13 chứa địa chỉ gián tiếp cấp 3 (double indirect), chứa địa chỉ của bảng các khối double indirect.

Cách tổ chức này tương đối linh động. Phương pháp này hiệu quả trong trường hợp sử dụng để quán lý những hệ thống tập tin lớn. Hệ điều hành sử dụng phương pháp này là Unix (Ví dụ : BSD Unix)



Hình 9.4 Cấu trúc của I-node

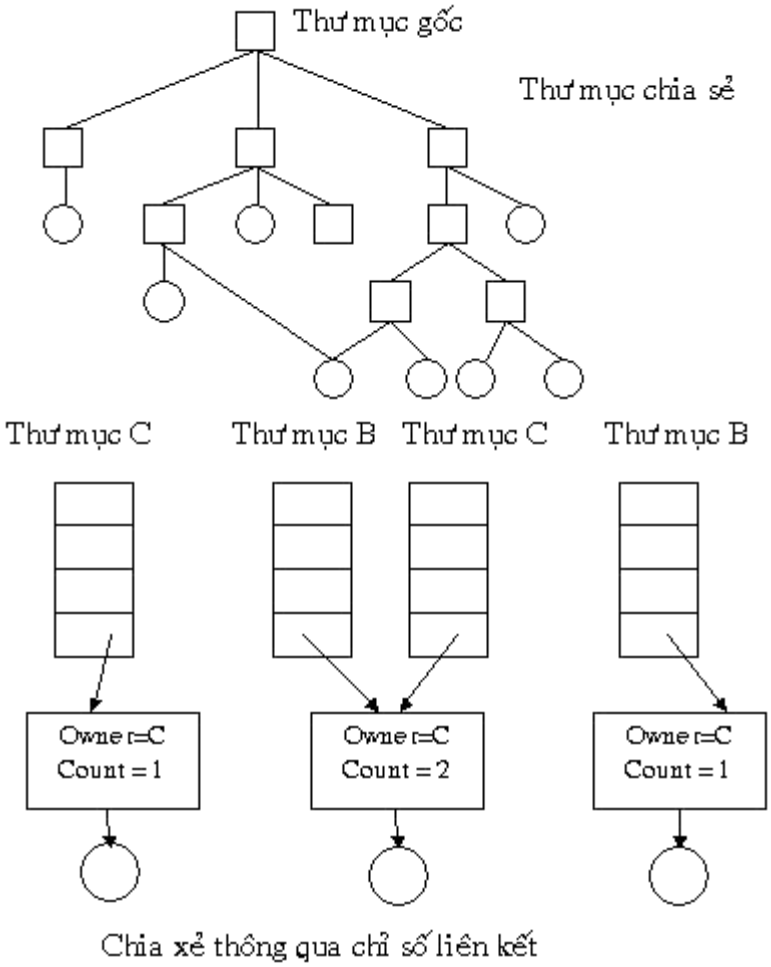
Hình 7.3.2.2-3. Cấu trúc I-node

7.3.3. Tệp tin chia sẻ

Khi có nhiều người sử dụng cùng làm việc trong một đề án, họ cần ***chia sẻ*** các tập tin. Cách chia sẻ thông thường là tập tin xuất hiện trong các thư mục là như nhau nghĩa là một tập tin có thể liên kết với nhiều thư mục khác nhau.

Để cài đặt được, khối đĩa không được liệt kê trong thư mục mà được thay thế bằng một cấu trúc dữ liệu, thư mục sẽ trỏ tới cấu trúc này. Một cách khác là hệ thống tạo một tập tin mới có kiểu LINK, tập tin mới này chỉ chứa đường dẫn của tập tin được liên kết, khi cần truy xuất sẽ dựa trên tập tin LINK để xác định tập tin cần truy xuất, phương pháp này gọi là liên kết hình thức. Mỗi phương pháp đều có những ưu và khuyết điểm riêng.

Ở phương pháp thứ nhất hệ thống biết được có bao nhiêu thư mục liên kết với tập tin nhờ vào chỉ số liên kết. Ở phương pháp thứ hai khi loại bỏ liên kết hình thức, tập tin không bị ảnh hưởng.



Hình 7.3.3-1. Chia sẻ thông tin qua chỉ số liên kết

7.3.4. Độ an toàn của hệ thống quản lý tệp tin

Một hệ thống tập tin bị hỏng còn nguy hiểm hơn máy tính bị hỏng vì những hư hỏng trên thiết bị sẽ ít chi phí hơn là hệ thống tập tin vì nó ảnh hưởng đến các phần mềm trên đó. Hơn nữa hệ thống tập tin không thể chống lại được như hư hòng do phần cứng gây ra, vì vậy chúng phải cài đặt một số chức năng để bảo vệ.

7.3.4.1. Quản lý khối bị hỏng

Đĩa thường có những khối bị hỏng trong quá trình sử dụng đặc biệt đối với đĩa cứng vì khó kiểm tra được hết tất cả.

Có hai giải pháp : phần mềm và phần cứng.

- Phần cứng là dùng một sector trên đĩa để lưu giữ danh sách các khối bị hỏng. Khi bộ kiểm soát tực hiện lần đầu tiên, nó đọc những khối bị hỏng và dùng một khối thừa để lưu giữ. Từ đó không cho truy cập những khối hỏng nữa.
- Phần mềm là hệ thống tập tin xây dựng một tập tin chứa các khối hỏng. Kỹ thuật này loại trừ chúng ra khỏi danh sách các khối trống, do đó nó sẽ không được cấp phát cho tập tin.

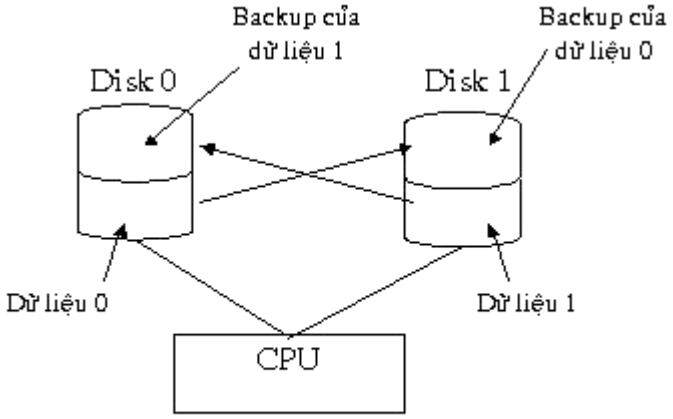
7.3.4.2. Sao lưu

Mặc dù có các chiến lược quản lý các khối hỏng, nhưng một công việc hết sức quan trọng là phải backup tập tin thường xuyên.

Tệp tin trên đĩa mềm được backup bằng cách chép lại toàn bộ qua một đĩa khác.

Dữ liệu trên đĩa cứng nhỏ thì được backup trên các băng từ.

Đối với các đĩa cứng lớn, việc backup thường được tiến hành ngay trên nó. Một chiến lược dể cài đặt nhưng lãng phí một nữa đĩa là chia đĩa cứng làm hai phần một phần dữ liệu và một phần là backup. Mỗi tối, dữ liệu từ phần dữ liệu sẽ được chép sang phần backup.

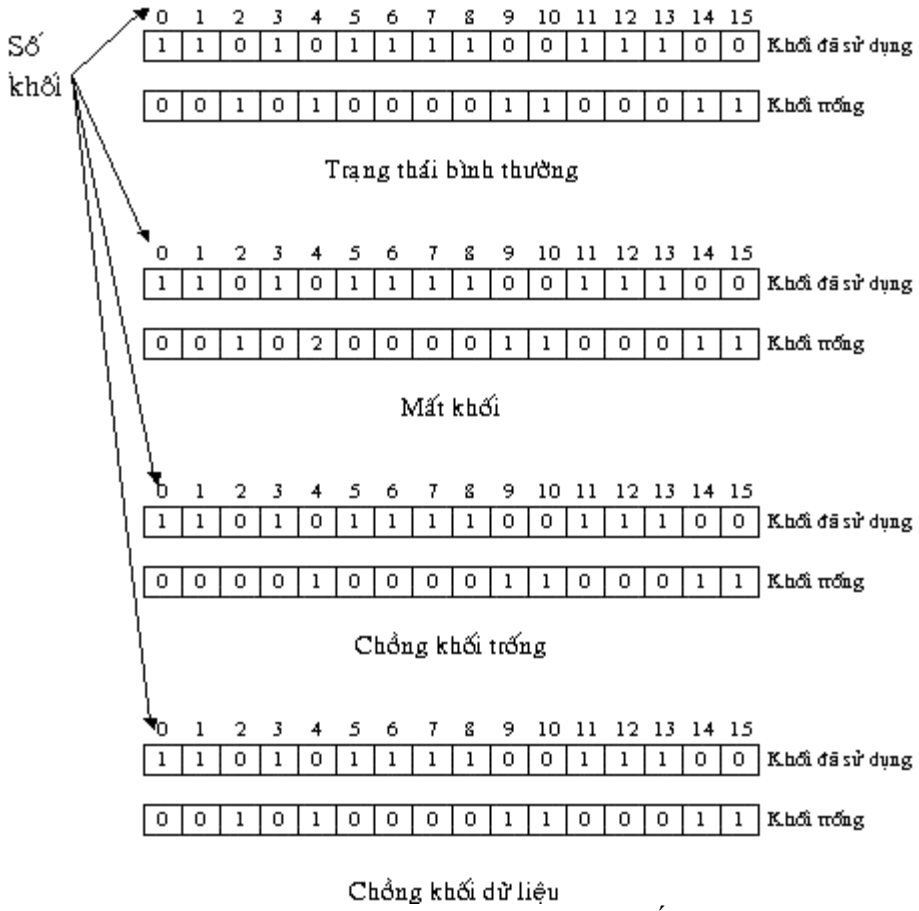


Hình 9.7 Backup

Hình 7.3.4.2-1. Cơ chế sao lưu

7.3.4.3. Tính không đổi của hệ thống tệp tin

Một vấn đề nữa về độ an toàn là ***tính không đổi***. Khi truy xuất một tập tin, trong quá trình thực hiện, nếu có xảy ra những sự cố làm hệ thống ngừng hoạt động đột ngột, lúc đó hàng loạt thông tin chưa được cập nhật lên đĩa. Vì vậy mỗi lân khởi động ,hệ thống sẽ thực hiện việc kiểm tra trên hai phần khối và tập tin. Việc kiểm tra thực hiện , khi phát hiện ra lỗi sẽ tiến hành sữa chữa cho các trường hợp cụ thể:



Hình 7.3.4.3-1. Trạng thái của hệ thống tập tin

CHƯƠNG 8. HỆ THỐNG QUẢN LÝ NHẬP/XUẤT

Một trong những chức năng chính của hệ điều hành là quản lý tất cả những thiết bị nhập/xuất của máy tính. Hệ điều hành phải ra các chỉ thị điều khiển thiết bị, kiểm soát các ngắt và lỗi. Hệ điều hành phải cung cấp một cách giao tiếp đơn giản và tiện dụng giữa các thiết bị và phần còn lại của hệ thống và giao tiếp này phải độc lập với thiết bị. Nội dung chương này tìm hiểu hệ điều hành quản lý nhập/xuất như thế nào với những nội dung sau:

- Khái niệm về hệ thống nhập/ xuất
- Phần cứng nhập / xuất
- Phần mềm nhập / xuất

Qua chương này, chúng ta hiểu được cơ chế quản lý nhập/xuất của hệ điều hành một cách tổng quát. Từ đó chúng ta có thể hiểu rõ hơn quá trình nhập xuất diễn ra trên máy tính thông qua hệ điều hành như thế nào. Bài học này cũng giúp cho việc tìm hiểu cơ chế tương tác giữa hệ điều hành và các thiết bị nhập/xuất cụ thể(được đề cập trong bài học sau) dễ dàng hơn.

8.1. KHÁI NIỆM VỀ HỆ THỐNG QUẢN LÝ NHẬP/XUẤT

Hệ thống quản lý nhập/xuất được tổ chức theo từng lớp, mỗi lớp có một chức năng nhất định và các lớp có giao tiếp với nhau như sơ đồ sau :

CÁC LỚP	CHỨC NĂNG NHẬP/XUẤT
Xử lý của người dùng	Tạo lời gọi nhập/xuất, định dạng nhập/xuất
Phần mềm độc lập thiết bị	Đặt tên, bảo vệ, tổ chức khối, bộ đệm, định vị
Điều khiển thiết bị	Thiết lập thanh ghi thiết bị, kiểm tra trạng thái
Kiểm soát ngắt	Báo cho driver khi nhập/xuất hoàn tất
Phần cứng	Thực hiện thao tác nhập/xuất

Ví dụ: Trong một chương trình ứng dụng, người dùng muốn đọc một khối từ một tập tin, hệ điều hành được kích hoạt để thực hiện yêu cầu này. Phần mềm độc lập thiết bị tìm kiếm trong cache, nếu khối cần đọc không có sẵn, nó sẽ gọi chương trình điều khiển thiết bị gửi yêu cầu đến phần cứng. Tiến trình bị ngưng lại cho đến khi thao tác đĩa hoàn tất. Khi thao tác này hoàn tất, phần cứng phát sinh một ngắt. Bộ phận kiểm soát ngắt kiểm tra biến cố này, ghi nhận trạng thái của thiết bị và đánh thức tiến trình bị ngưng để chấm dứt yêu cầu I/O và cho tiến trình của người sử dụng tiếp tục thực hiện.[TAN]

8.2. PHẦN CỨNG NHẬP/XUẤT

Có nhiều cách nhìn khác nhau về phần cứng nhập/xuất. Các kỹ sư điện tử thì nhìn dưới góc độ là các thiết bị như IC, dây dẫn, bộ nguồn, motor v.v….Các lập trình viên thì nhìn chúng dưới góc độ phần mềm - những lệnh nào thiết bị chấp nhận, chúng sẽ thực hiện những chức năng nào, và thông báo lỗi của chúng bao gồm những gì, nghĩa là chúng ta quan tâm đến lập trình thiết bị chứ không phải các thiết bị này hoạt động như thế nào mặc dù khía cạnh này có liên quan mật thiết với các thao tác bên trong của chúng. Phần này chúng ta đề cập đến một số khái niệm về phần cứng I/O liên quan đến khía cạnh lập trình.

8.2.1. Thiết bị I/O

Các thiết bị nhập xuất có thể chia tương đối thành hai loại là thiết bị khối và thiết bị tuần tự.

Thiết bị khối là thiết bị mà thông tin được lưu trữ trong những khối có kích thước cố định và được định vị bởi địa chỉ. Kích thước thông thường của một khối là khoảng từ 128 bytes đến 1024 bytes. Đặc điểm của thiết bị khối là chúng có thể được truy xuất (đọc hoặc ghi) từng khối riêng biệt, và chương trình có thể truy xuất một khối bất kỳ nào đó. Đĩa là một ví dụ cho loại thiết bị khối.

Một dạng thiết bị thứ hai là thiết bị tuần tự. Ở dạng thiết bị này, việc gửi và nhận thông tin dựa trên là chuỗi các bits, không có xác định địa chỉ và không thể thực hiện thao tác seek được. Màn hình, bàn phím, máy in, card mạng, chuột, và các loại thiết bị khác không phải dạng đĩa là thiết bị tuần tự.

Việc phân chia các lớp như trên không hoàn toàn tối ưu, một số các thiết bị không phù hợp với hai lớp trên, ví dụ : đồng hồ, bộ nhớ màn hình v.v...không thực hiện theo cơ chế tuần tự các bits. Ngoài ra, người ta còn phân loại các thiết bị I/O dưới một tiêu chuẩn khác :

Thiết bị tương tác được với con người : dùng để giao tiếp giữa người và máy. Ví dụ : màn hình, bàn phím, chuột, máy in ...

Thiết bị tương tác trong hệ thống máy tính là các thiết bị giao tiếp với nhau. Ví dụ : đĩa, băng từ, card giao tiếp...

Thiết bị truyền thông : như modem...

Những điểm khác nhau giữa các thiết bị I/O gồm :

- Tốc độ truyền dữ liệu , ví dụ bàn phím : 0.01 KB/s, chuột 0.02 KB/s ...
- Công dụng.
- Đơn vị truyền dữ liệu (khối hoặc ký tự).
- Biểu diễn dữ liệu, điều này tùy thuộc vào từng thiết bị cụ thể.
- Tình trạng lỗi : nguyên nhân gây ra lỗi, cách mà chúng báo về...

8.2.2. Tổ chức của chức năng I/O

Có ba cách để thực hiện I/O :

- Một là, bộ xử lý phát sinh một lệnh I/O đến các đơn vị I/O, sau đó, nó chờ trong trạng thái "busy" cho đến khi thao tác này hoàn tất trước khi tiếp tục xử lý.

- Hai là, bộ xử lý phát sinh một lệnh I/O đến các đơn vị I/O, sau đó, nó tiếp tục việc xử lý cho tới khi nhận được một ngắt từ đơn vị I/O báo là đã hoàn tất, nó tạm ngưng việc xử lý hiện tại để chuyển qua xử lý ngắt.
- Ba là, sử dụng cơ chế DMA (như được đề cập ở sau)

Các bước tiến hóa của chức năng I/O :

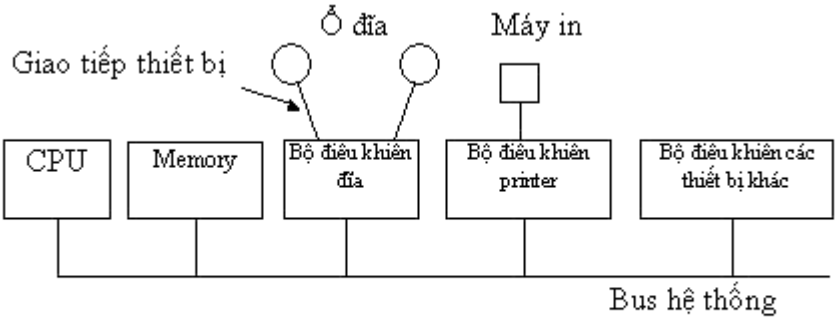
- Bộ xử lý kiểm soát trực tiếp các thiết bị ngoại vi.
- Hệ thống có thêm bộ điều khiển thiết bị. Bộ xử lý sử dụng cách thực hiện nhập xuất thứ nhất. Theo cách này bộ xử lý được tách rời khỏi các mô tả chi tiết của các thiết bị ngoại vi.
- Bộ xử lý sử dụng thêm cơ chế ngắt.
- Sử dụng cơ chế DMA, bộ xử lý truy xuất những dữ liệu I/O trực tiếp trong bộ nhớ chính.

8.2.3. Bộ điều khiển thiết bị

Một đơn vị bị nhập xuất thường được chia làm hai thành phần chính là thành phần cơ và thành phần điện tử. Thành phần điện tử được gọi là bộ phận điều khiển thiết bị hay bộ tương thích, trong các máy vi tính thường được gọi là card giao tiếp. Thành phần cơ chính là bản thân thiết bị.

Một bộ phận điều khiển thường có bộ phận kết nối trên chúng để có thể gắn thiết bị lên đó. Một bộ phận điều khiển có thể quản lý được hai, bốn hay thậm chí tám thiết bị khác nhau. Nếu giao tiếp giữa thiết bị và bộ phận điều khiển là các chuẩn như ANSI, IEEE hay ISO thì nhà sản xuất thiết bị và bộ điều khiển phải tuân theo chuẩn đó, ví dụ : bộ điều khiển đĩa được theo chuẩn giao tiếp của IBM.

Giao tiếp giữa bộ điều khiển và thiết bị là giao tiếp ở mức thấp.



Hình 11.1 Sự kết nối giữa CPU, bộ nhớ, bộ điều khiển và các thiết bị nhập/xuất

Hình 8.2.3-1. Mô hình vào/ra

Chức năng của bộ điều khiển là giao tiếp với hệ điều hành vì hệ điều hành không thể truy xuất trực tiếp với thiết bị. Việc thông tin thông qua hệ thống đường truyền gọi là bus.

Công việc của bộ điều khiển là chuyển đổi dãy các bit tuần tự trong một khối các byte và thực hiện sửa chửa nếu cần thiết. Thông thường khối các byte được tổ chức thành từng bit và đặt trong buffer của bộ điều khiển. Sau khi thực hiện checksum nội dung của buffer sẽ được chuyển vào bộ nhớ chính. Ví dụ : bộ điều khiển cho màn hình đọc các byte của ký tự để hiển thị trong bộ nhớ và tổ chức các tín hiệu để điều khiển các tia của CRT để xuất trên màn ảnh bằng cách quét các tia dọc và ngang. Nếu không có bộ điều

khiển, lập trình viên hệ điều hành phải tạo thêm chương trình điều khiển tín hiệu analog cho đèn hình. Với bộ điều khiển , hệ điều hành chỉ cần khởi động chúng với một số tham số như số ký tự trên một dòng, số dòng trên màn hình và bộ điều khiển sẽ thực hiện điều khiển các tia.

Mỗi bộ điều khiển có một số thanh ghi để liên lạc với CPU. Trên một số máy tính, các thanh ghi này là một phần của bộ nhớ chính tại một địa chỉ xác định gọi là ánh xạ bộ nhớ nhập xuất. Hệ máy PC dành ra một vùng địa chỉ đặc biệt gọi là địa chỉ nhập xuất và trong đó được chia làm nhiều đoạn, mỗi đoạn cho một loại thiết bị như sau :

Bộ điều khiển nhập/ xuất	Địa chỉ nhập/xuất	Vectơ ngắt
Đồng hồ	040 - 043	8
Bàn phím	060 - 063	9
RS232 phụ	2F8 - 2FF	11
Đĩa cứng	320 - 32F	13
Máy in	378 - 37F	15
Màn hình mono	380 - 3BF	-
Màn hình màu	3D0 - 3DF	-
Đĩa mềm	3F0 - 3F7	14
RS232 chính	3F8 - 3FF	12

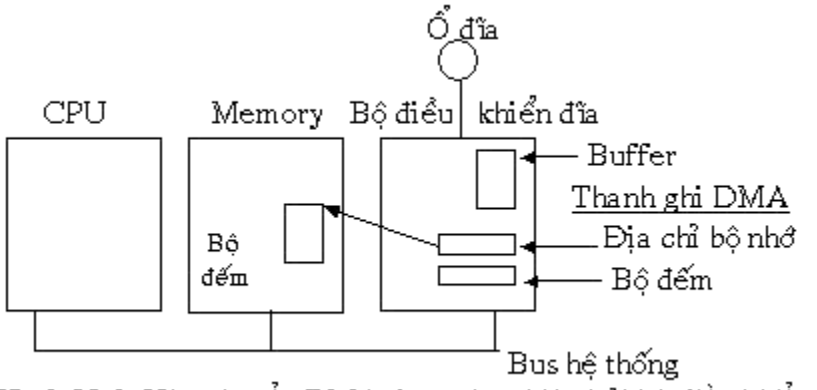
Hệ điều hành thực hiện nhập xuất bằng cách ghi lệnh lên các thanh ghi của bộ điều khiển. Ví dụ : bộ điều khiển đĩa mềm của IBMPC chấp nhận 15 lệnh khác nhau như : READ, WRITE, SEEK, FORMAT, RECALIBRATE, một số lệnh có tham số và các tham số cũng được nạp vào thanh ghi. Khi một lệnh đã được chấp nhận, CPU sẽ rời bộ điều khiển để thực hiện công việc khác. Sau khi thực hiện xong, bộ điều khiển phát sinh một ngắt để báo hiệu cho CPU biết và đến lấy kết quả được lưu giữ trong các thanh ghi.

8.2.4. DMA (Direct Memory Access)

Đa số các loại thiết bị, đặc biệt là các thiết bị dạng khối, hỗ trợ cơ chế DMA (direct memory access). Để hiểu về cơ chế này, trước hết phải xem xét quá trình đọc đĩa mà không có DMA. Trước tiên, bộ điều khiển đọc tuần tự các khối trên đĩa, từng bit từng bit cho tới khi toàn bộ khối được đưa vào buffer của bộ điều khiển. Sau đó máy tính thực hiện checksum để đảm bảo không có lỗi xảy ra. Tiếp theo bộ điều khiển tạo ra một ngắt để báo cho CPU biết. CPU đến lấy dữ liệu trong buffer chuyển về bộ nhớ chính bằng cách tạo một vòng lặp đọc lần lượt từng byte. Thao tác này làm lãng phí thời gian của CPU. Do đó để tối ưu, người ta đưa ra cơ chế DMA.

Cơ chế DMA giúp cho CPU không bị lãng phí thời gian. Khi sử dụng, CPU gửi cho bộ điều khiển một số các thông số như địa chỉ trên đĩa của khối, địa chỉ trong bộ nhớ nơi định vị khối, số lượng byte dữ liệu để chuyển.

Sau khi bộ điều khiển đã đọc toàn bộ dữ liệu từ thiết bị vào buffer của nó và kiểm tra checksum. Bộ điều khiển chuyển byte đầu tiên vào bộ nhớ chính tại địa chỉ được mô tả bởi địa chỉ bộ nhớ DMA. Sau đó nó tăng địa chỉ DMA và giảm số bytes phải chuyển. Quá trình này lập cho tới khi số bytes phải chuyển bằng 0, và bộ điều khiển tạo một ngắt. Như vậy không cần phải copy khối vào trong bộ nhớ, nó đã hiện hữu trong bộ nhớ.



Hình 11.2 Vận chuyển DMA được thực hiện bởi bộ điều khiển

Hình 8.2.4. Kỹ thuật DMA

8.3. PHẦN MỀM NHẬP/XUẤT

Mục tiêu chung của thiết bị logic là dễ biểu diễn. Thiết bị logic được tổ chức thành nhiều lớp. Lớp dưới cùng giao tiếp với phần cứng, lớp trên cùng giao tiếp tốt, thân thiện với người sử dụng. Khái niệm then chốt của thiết bị logic là độc lập thiết bị, ví dụ : có thể viết chương trình truy xuất file trên đĩa mềm hay đĩa cứng mà không cần phải mô tả lại chương trình cho từng loại thiết bị. Ngoài ra, thiết bị logic phải có khả năng kiểm soát lỗi. Thiết bị logic được tổ chức thành bốn lớp : Kiểm soát lỗi, điều khiển thiết bị, phần mềm hệ điều hành độc lập thiết bị, phần mềm mức người sử dụng.

8.3.1. Kiểm soát ngắt

Ngắt là một hiện tượng phức tạp. Nó phải cần được che dấu sâu trong hệ điều hành, và một phần ít của hệ thống biết về chúng. Cách tốt nhất để che dấu chúng là hệ điều hành có mọi tiến trình thực hiện thao tác nhập xuất cho tới khi hoàn tất mới tạo ra một ngắt. Tiến trình có thể tự khóa lại bằng cách thực hiện lệnh WAIT theo một biến điều kiện hoặc RECEIVE theo một thông điệp.

Khi một ngắt xảy ra, hàm xử lý ngắt khởi tạo một tiến trình mới để xử lý ngắt. Nó sẽ thực hiện một tín hiệu trên biến điều kiện và gửi những thông điệp đến cho các tiến trình bị khóa. Tổng quát, chức năng của ngắt là làm cho một tiến trình đang bị khóa được thi hành trở lại.

8.3.2. Điều khiển thiết bị (device drivers)

Tất cả các đoạn mã độc lập thiết bị đều được chuyển đến device drivers. Mỗi device drivers kiểm soát mỗi loại thiết bị, nhưng cũng có khi là một tập hợp các thiết bị liên quan mật thiết với nhau.

Device drivers phát ra các chỉ thị và kiểm tra xem chỉ thị đó có được thực hiện chính xác không. Ví dụ, driver của đĩa là phần duy nhất của hệ điều hành kiểm soát bộ điều khiển đĩa. Nó quản lý sectors, tracks, cylinders, head, chuyển động, interleave, và các thành phần khác giúp cho các thao tác đĩa được thực hiện tốt.

Chức năng của device drivers là nhận những yêu cầu trừu tượng từ phần mềm nhập/xuất độc lập thiết bị ở lớp trên, và giám sát yêu cầu này thực hiện. Nếu driver đang rảnh, nó sẽ thực hiện ngay yêu cầu, ngược lại, yêu cầu đó sẽ được đưa vào hàng đợi.

Ví dụ, bước đầu tiên của yêu cầu nhập/xuất đĩa là chuyển từ trừu tượng thành cụ thể. Driver của đĩa phải biết khối nào cần đọc, kiểm tra sự hoạt động của motor đĩa, xác định vị trí của đầu đọc đã đúng chưa v.v…

Nghĩa là device drivers phải xác định được những thao tác nào của bộ điều khiển phải thi hành và theo trình tự nào. Một khi đã xác định được chỉ thị cho bộ điều khiển, nó bắt đầu thực hiện bằng cách chuyển lệnh vào thanh ghi của bộ điều khiển thiết bị. Bộ điều khiển có thể nhận một hay nhiều chỉ thị liên tiếp và sau đó tự nó thực hiện không cần sự trợ giúp của hệ điều hành. Trong khi lệnh thực hiện. Có hai trường hợp xảy ra : Một là device drivers phải chờ cho tới khi bộ điều khiển thực hiện xong bằng cách tự khóa lại cho tới khi một ngắt phát sinh mở khóa cho nó. Hai là, hệ điều hành chấm dứt mà không chờ, vì vậy driver không cần thiết phải khóa.

Sau khi hệ điều hành hoàn tất việc kiểm tra lỗi và nếu mọi thứ đều ổn driver sẽ chuyển dữ liệu cho phần mềm độc lập thiết bị. Cuối cùng nó sẽ trả về thông tin về trạng thái hay lỗi cho nơi gọi và nếu có một yêu cầu khác ở hàng đợi, nó sẽ thực hiện tiếp, nếu không nó sẽ khóa lại chờ đến yêu cầu tiếp theo.

8.3.3. Phần mềm nhập/xuất độc lập thiết bị

Mặc dù một số phần mềm nhập/xuất mô tả thiết bị nhưng phần lớn chúng là độc lập với thiết bị. Ranh giới chính xác giữa drivers và phần mềm độc lập thiết bị là độc lập về mặt hệ thống, bởi vì một số hàm mà được thi hành theo kiểu độc lập thiết bị có thể được thi hành trên drivers vì lý do hiệu quả hay những lý dó khác nào đó.

Giao tiếp đồng nhất cho device drivers
Đặt tên thiết bị
Bảo vệ thiết bị
Cung cấp khối độc lập thiết bị
Tổ chức buffer
Định vị lưu trữ trên thiết bị khối

Cấp phát và giải phóng thiết bị tận hiến
Báo lỗi

Chức năng cơ bản của phần mềm nhập/xuất độc lập thiết bị là những chức năng chung cho tất cả các thiết bị và cung cấp một giao tiếp đồng nhất cho phần mềm phạm vi người sử dụng.

Trước tiên nó phải có chức năng tạo một ánh xạ giữa thiết bị và một tên hình thức. Ví dụ đối với UNIX, tên /dev/tty0 dành riêng để mô tả I-node cho một file đặc biệt, và I-node này chứa chứa số thiết bị chính, được dùng để xác định driver thích hợp và số thiết bị phụ, được dùng để xác định các tham số cho driver để cho biết là đọc hay ghi.

Thứ hai là bảo vệ thiết bị, là cho phép hay không cho phép người sử dụng truy xuất thiết bị. Các hệ điều hành có thể có hay không có chức năng này.

Thứ ba là cung cấp khối dữ liệu độc lập thiết bị vì ví dụ những đĩa khác nhau sẽ có kích thước sector khác nhau và điều này sẽ gây khó khăn cho các phần mềm người sử dụng ở lớp trên. Chức năng này cung cấp các khối dữ liệu logic độc lập với kích thước sector vật lý.

Thứ tư là cung cấp buffer để hỗ trợ cho đồng bộ hóa quá trình hoạt động của hệ thống. Ví dụ buffer cho bàn phím.

Thứ năm là định vị lưu trữ trên các thiết bị khối.

Thứ sáu là cấp phát và giải phóng các thiết bị tận hiến.

Cuối cùng là thông báo lỗi cho lớp bên trên từ các lỗi do device driver báo về.

8.3.4. Phần mềm nhập/xuất phạm vi người sử dụng

Hầu hết các phần mềm nhập/xuất đều ở bên trong của hệ điều hành và một phần nhỏ của chúng chứa các thư viện liên kết với chương trình của người sử dụng ngay cả những chương trình thi hành bên ngoài hạt nhân.

Lời gọi hệ thống, bao gồm lời gọi hệ thống nhập/xuất thường được thực hiện bởi các hàm thư viện. Ví dụ khi trong chương trình C có lệnh

```
count = write(fd, buffer, nbytes) ;
```

Hàm thư viện write được địch và liên kết dưới dạng nhị phân và nằm trong bộ nhớ khi thi hành. Tập hợp tất cả những hàm thư viện này rõ ràng là một phần của hệ thống nhập/xuất.

Không phải tất cả các phần mềm nhập/xuất đều chứa hàm thư viện, có một loại quan trọng khác gọi là hệ thống spooling dùng để khai thác tối đa thiết bị nhập/xuất trong hệ thống đa chương.

Các hàm thư viện chuyển các tham số thích hợp cho lời gọi hệ thống và hàm thư viện thực hiện việc định dạng cho nhập và xuất như lệnh printf trong C. Thư viện nhập/xuất chuẩn chứa một số hàm có chức năng nhập/xuất và tất cả chạy như chương trình người dùng.

Chức năng của spooling là tránh trường hợp một tiến trình đang truy xuất thiết bị, chiếm giữ thiết bị nhưng sau đó không làm gì cả trong một khoảng thời gian và như vậy các tiến trình khác bị ảnh hưởng vì không thể truy xuất thiết bị đó. Một ví dụ của spooling device là line printer. Spooling còn được sử dụng trong hệ thống mạng như hệ thống e-mail chẳng hạn.

CHƯƠNG 9. GIỚI THIỆU MỘT SỐ HỆ THỐNG I/O

Cơ chế quản lý nhập/xuất(I/O) của hệ điều hành được minh họa cụ thể qua việc điều khiển các thiết bị I/O cụ thể. Trong bài này chúng ta tìm hiểu một số hệ thống I/O sau:

- Hệ thống nhập xuất đĩa
- Hệ thống nhập xuất chuẩn
- Cài đặt đồng hồ

Qua chương này, chúng ta hiểu được cơ chế quản lý nhập/xuất của hệ điều hành được thể hiện cụ thể trên một số thiết bị I/O. Chúng ta cũng nắm được cơ chế tương tác giữa hệ điều hành với các thiết bị đó và trên hết chúng ta thấy được vai trò của độc lập thiết bị.
Bài học này đòi hỏi những kiến thức về : kiến trúc máy tính, hệ thống quản lý I/O của hệ điều hành.

9.1. HỆ THỐNG I/O ĐĨA

Hầu như tất cả các máy tính đều có đĩa để lưu trữ thông tin. Đĩa có ba ưu điểm chính hơn sử dụng bộ nhớ chính để lưu trữ :

- Dung lượng lưu trữ lớn hơn rất nhiều.
- Giá trên một bit rẻ hơn.
- Thông tin không bị mất đi khi không còn cung cấp điện.

9.1.1. Phần cứng đĩa

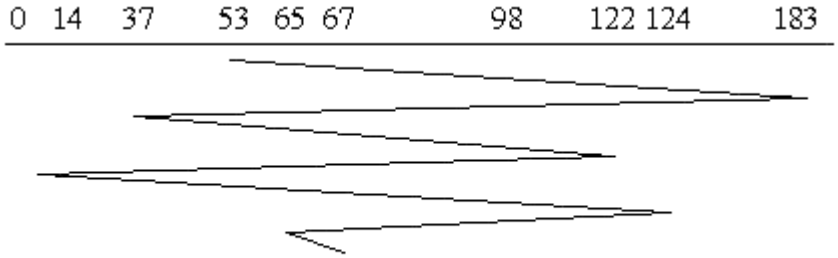
Một đĩa bao gồm nhiều cylinder, mỗi cylinder chứa nhiều track trên các head. Mỗi track được chia làm nhiều sector (từ 8 đến 32). Mỗi sector có số byte là như nhau dù vị trí của nó ở gần tâm hay ở ngoài rìa đĩa, những khoảng trống thừa không dùng đến.
Một đặc điểm thiết bị cài đặt quan trọng cho driver của đĩa là khả năng của bộ điều khiển thực hiện tìm kiếm trên hai hay nhiều driver cùng lúc gọi là tìm kiếm chồng. Trong khi bộ điều khiển và phần mềm đợi việc tìm kiếm hoàn tất trên một đĩa, bộ điều khiển có thể khởi động việc tìm kiếm trên đĩa khác. Các bộ điều khiển không thể cùng lúc đọc hoặc ghi trên hai driver vì khả năng này có thể làm giảm thời gian truy xuất trung bình.

9.1.2. Các thuật toán đọc đĩa

Tất cả mọi công việc đều phụ thuộc vào việc nạp chương trình và nhập xuất tập tin, do đó điều quan trọng là dịch vụ đĩa phải càng nhanh càng tốt. Hệ điều hành có thể tổ chức dịch vụ truy xuất đĩa tốt hơn bằng cách lập lịch yêu cầu truy xuất đĩa.
Tốc độ đĩa bao gồm ba phần. Để truy xuất các khối trên đĩa, trước tiên phải di chuyển đầu đọc đến track hay cylinder thích hợp, thao tác này gọi là seek và thời gian để hoàn tất gọi là *seek time*. Một khi đã đến đúng track, còn phải chờ cho đến khi khối cần thiết đến dưới đầu đọc. Thời gian chờ này gọi là *latency time*. Cuối cùng là vận chuyển dữ liệu giữa đĩa và bộ nhớ chính gọi là *transfer time*. Tổng thời gian cho dịch vụ đĩa chính là tổng của ba khoảng thời gian trên. Trong đó *seek time* và *latency time* là mất nhiều thời gian nhất, do đó để giảm thiểu thời gian truy xuất hệ điều hành đưa ra các thuật toán lập lịch truy xuất.

9.1.2.1. Lập lịch FCFS

Phương pháp lập lịch đơn giản nhất là FCFS(first-come,first-served). Thuật toán này rất dễ lập trình nhưng không cung cấp được một dịch vụ tốt. Ví dụ : cần phải đọc các khối theo thứ tự như sau :
98, 183, 37, 122, 14, 124, 65, và 67
Giả sử hiện tại đầu đọc đang ở vị trí 53. Như vậy đầu đọc lần lượt đi qua các khối 53, 98, 183, 37, 122, 14, 124, 65, và 67 như hình sau :

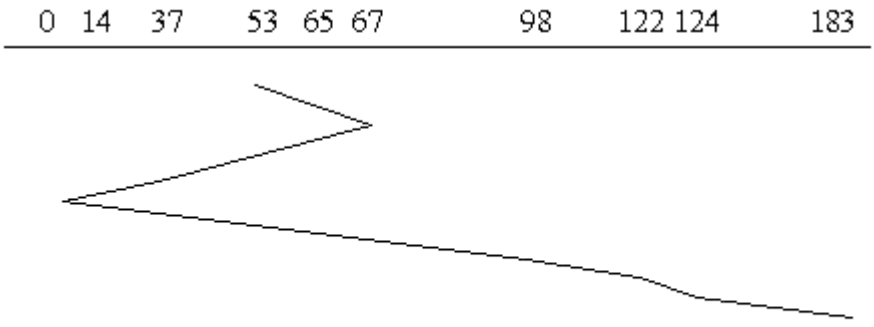


Hình 12.1 Phương pháp FCFS

Hình 9.1.2.1-1. Lập lịch FCFS

9.1.2.2. Lập lịch SSTF (shortest-seek-time-first)

Thuật toán này sẽ di chuyển đầu đọc đến các khối cần thiết theo vị trí lần lượt gần với vị trí hiện hành của đầu đọc nhất. Ví dụ : cần đọc các khối như sau :
98, 183, 37, 122, 14, 124, 65, và 67
Giả sử hiện tại đầu đọc đang ở vị trí 53. Như vậy đầu đọc lần lượt đi qua các khối 53, 65, 67, 37, 14, 98, 122, 124 và 183 như hình sau :



Hình 12.2 Phương pháp SSTF

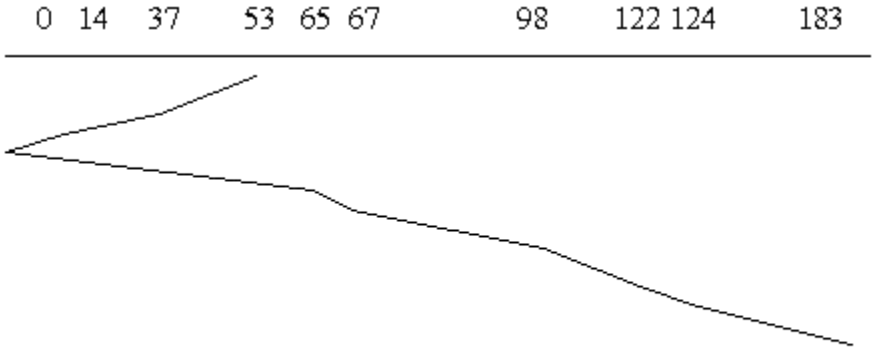
Hình 9.1.2.2-1. Lập lịch SSTF

Với ví dụ này, thuật toán SSTF làm giảm số khối mà đầu đọc phải di chuyển là 208 khối.

9.1.2.3. Lập lịch SCAN

Theo thuật toán này, đầu đọc sẽ di chuyển về một phía của đĩa và từ đó di chuyển qua phía kia. Ví dụ : cần đọc các khối như sau :
98, 183, 37, 122, 14, 124, 65, và 67

Giả sử hiện tại đầu đọc đang ở vị trí 53. Như vậy đầu đọc lần lượt đi qua các khối 53, 37, 14, 0 , 65, 67, 98, 122, 124 và 183 như hình sau :



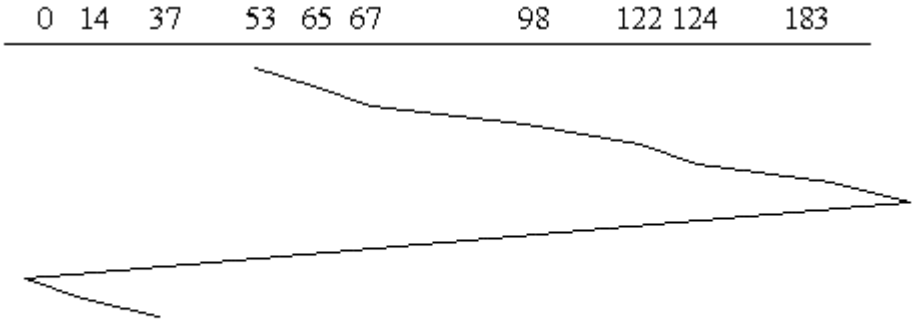
Hình 12.3 Phương pháp SCAN

Hình 9.1.2.3-1. Lập lịch SCAN

Thuật toán này còn được gọi là thuật toán thang máy. Hình ảnh thuật toán giống như hình ảnh của một người quét tuyết, hay quét lá.

9.1.2.4. Lập lịch C-SCAN

Thuật toán này tương tự như thuật toán SCAN, chỉ khác là khi nó di chuyển đến một đầu nào đó của đĩa, nó sẽ lập tức trở về đầu bắt đầu của đĩa. Lấy lại ví dụ trên, khi đó thứ tự truy xuất các khối sẽ là : 53, 65, 67, 98, 122, 124, 183, 199, 0, 14, 37 như hình sau :

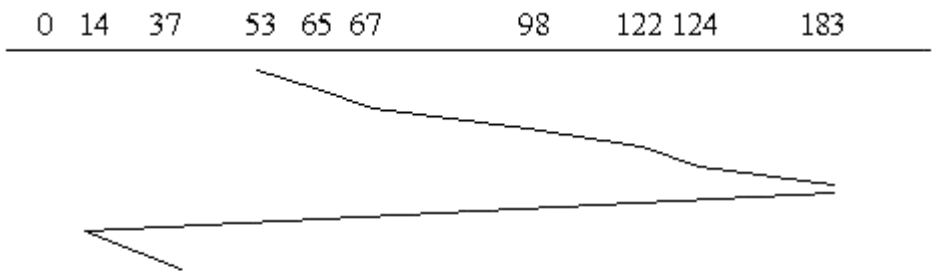


Hình 12.4 Phương pháp C-SCAN

Hình 9.1.2.4-1. Lập lịch C-SCAN

9.1.2.5. Lập lịch LOOK

Nhận xét rằng cả hai thuật toán lập lịch SCAN và C-SCAN luôn luôn chuyển đầu đọc của đĩa từ đầu này sang đầu kia. Nhưng thông thường thì đầu đọc chỉ chuyển đến khối xa nhất ở mỗi hướng chứ không đến cuối. Do đó SCAN và C-SCAN được chỉnh theo thực tế và gọi là lập lịch LOOK. Như hình sau :



Hình 12.5 Phương pháp LOOK

Hình 9.1.2.5-1. Lập lịch LOOK

9.1.2.6. Lựa chọn thuật toán lập lịch:

Với những thuật toán lập lịch, vấn đề là phải lựa chọn thuật toán nào cho hệ thống. Thuật toán SSTF thì rất thông thường. Thuật toán SCAN và C-SCAN thích hợp cho những hệ thống phải truy xuất dữ liệu khối lượng lớn. Với bất kỳ thuật toán lập lịch nào, điều quan trọng là khối lượng về số và kiểu khối cần truy xuất. Ví dụ , nếu số khối cần truy xuất là liên tục thì FCFS là thuật toán tốt.

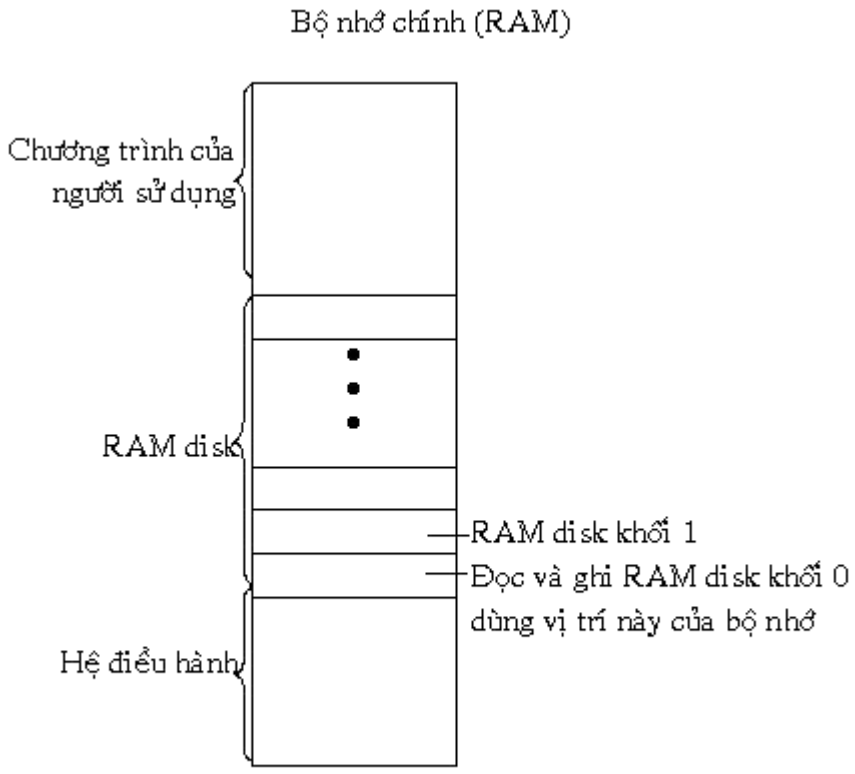
9.1.3. Quản lý lỗi

Đĩa là đối tượng mà khi truy xuất có thể gây nhiều lỗi. Một trong số các lỗi thường gặp là :

- *Lỗi lập trình* : yêu cầu đọc các sector không tồn tại.
  Lỗi lập trình xảy ra khi yêu cầu bộ điều khiển tìm kiếm cylinder không tồn tại, đọc sector không tồn tại, dùng đầu đọc không tồn tại, hoặc vận chuyển vào và ra bộ nhớ không tồn tại. Hầu hết các bộ điều khiển kiểm tra các tham số và sẽ báo lỗi nếu không thích hợp.
- *Lỗi checksum tạm thời* : gây ra bởi bụi trên đầu đọc.
  Bụi tồn tại giữa đầu đọc và bề mặt đĩa sẽ gây ra lỗi đọc. Nếu lỗi tồn tại, khối có thể bị đánh dấu hỏng bởi phần mềm.
- *Lỗi checksum thường trực* : đĩa bị hư vật lý trên các khối.
- *Lỗi tìm kiếm* : ví dụ đầu đọc đến cylinder 7 trong khi đó phải đọc 6.
- *Lỗi điều khiển* : bộ điều khiển từ chối thi hành lệnh.

9.1.4. RAM Disks

Ý tưởng RAM disk khá đơn giản. Thiết bị khối là phần lưu trữ trung gian với hai lệnh : đọc một khối và ghi một khối. Thông thường những khối này được lưu trữ trên đĩa mềm hoặc đĩa cứng. ***RAM disk dùng một phần đã định vị trước của bộ nhớ chính để lưu trữ các khối***. RAM disk có ưu điểm là cho phép truy xuất nhanh chóng (không phải chờ quay hay tìm kiếm). Như vậy nó thích hợp cho việc lưu trữ những chương trình hay dữ liệu được truy xuất thường xuyên.



Hình 12.6 RAM disks

Hình 9.1.4-1. RAM Disk

Hình trên mô tả ý tưởng của RAM disk. Một RAM disk được chia làm nhiều khối, số lượng tùy thuộc vào dung lượng của vùng nhớ. Mỗi khối có cùng kích thước và vừa đúng bằng kích thước của khối thực sự trên đĩa. Khi driver nhận được chỉ thị là đọc hoặc ghi một khối, nó sẽ tìm trong bộ nhớ RAM disk vị trí của khối, và thực hiện việc đọc hay ghi trong đó thay vì từ đĩa mềm hay đĩa cứng.

9.1.5. Interleave

Bộ điều khiển đọc ghi đĩa phải thực hiện hai chức năng là đọc/ghi dữ liệu và chuyển dữ liệu vào hệ thống. Để thực hiện được đồng bộ hai chức năng này, bộ điều khiển đọc đĩa cung cấp chức năng interleave. Trên đĩa các sector số hiệu liên tiếp nhau không nằm kế bên nhau mà có một khoảng cách nhất định, khoảng cách này được xác định bởi quá trình format đĩa. Ví dụ : giả sử hệ thống chỉ có 17 sector, và interleave được chọn là 4 thì các sector được bố trí theo thứ tự như sau :

1, 14, 10, 6, 2, 15, 11, 7, 3, 16, 12, 8, 4, 17, 13, 9, 5

Cách đọc lần lượt như sau :

Lần 1:

1, 14, 10, 6, ***2***, 15, 11, 7, ***3***, 16, 12, 8, ***4***, 17, 13, 9, ***5***

Lần 2:

1, 14, 10, ***6***, 2, 15, 11, ***7***, 3, 16, 12, ***8***, 4, 17, 13, ***9***, 5

Lần 3:

1, 14, ***10***, 6, 2, 15, ***11***, 7, 3, 16, ***12***, 8, 4, 17, ***13***, 9, 5

Lần 4:

1, ***14***, 10, 6, 2, ***15***, 11, 7, 3, ***16***, 12, 8, 4, ***17***, 13, 9, 5

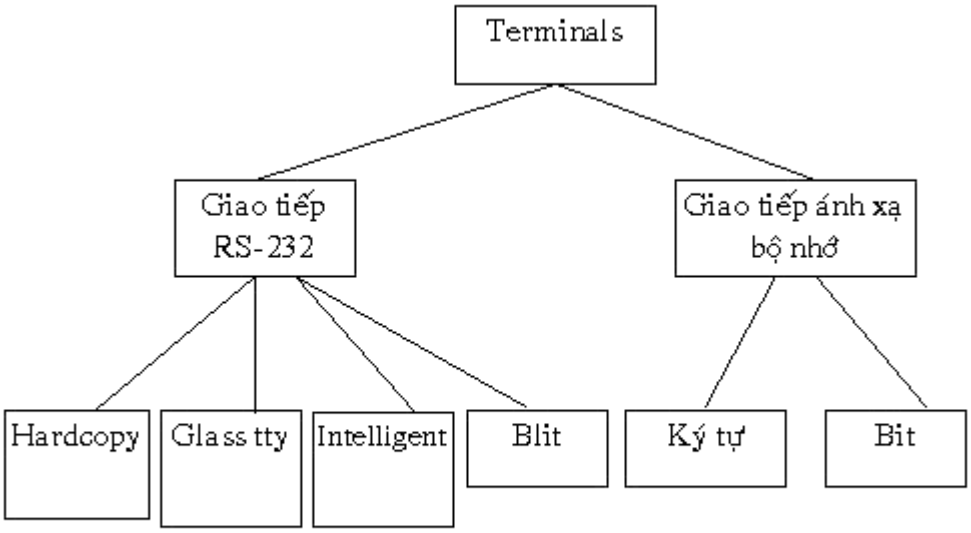
Như vậy sau bốn lần thứ tự các sector đọc được vẫn là từ 1 đến 17

9.2. HỆ THỐNG I/O CHUẨN (terminals)

Mọi máy tính đều liên lạc với một hay nhiều terminals. Terminals có rất nhiều dạng khác nhau. Bộ điều khiển terminals ẩn dấu mọi sự khác biệt, vì vậy phần độc lập thiết bị của hệ điều hành và chương trình người sử dụng không cần thiết phải viết lại cho mỗi loại terminal.

9.2.1. Phần cứng terminal

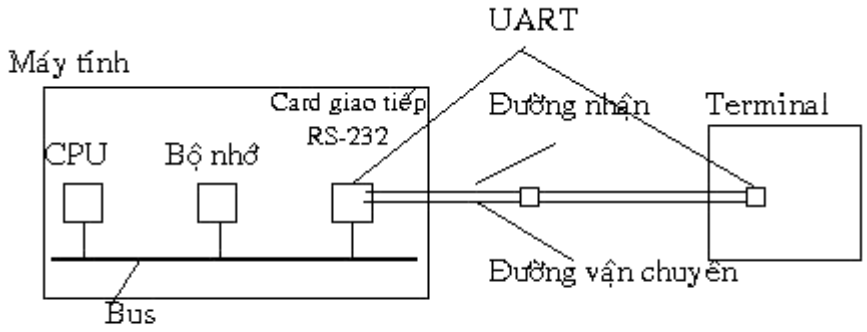
Dưới quan điểm của hệ điều hành, terminal được chia làm hai loại lớn dựa vào cách liên lạc với hệ điều hành. Loại thứ nhất bao gồm những loại terminal giao tiếp theo chuẩn RS-232. Loại thứ hai là những terminal dùng ánh xạ bộ nhớ. Mỗi loại được chia làm nhiều loại nhỏ như hình sau :



Hình 12.7 Các loại terminals

Hình 9.2.1-1. Các loại Terminals

Terminal RS-232 là những thiết bị bao gồm như bàn phím và màn hình. Đây là thiết bị giao tiếp tuần tự, mỗi lần một bit. Những terminals này dùng connector 25-pin, một pin dùng để chuyển dữ liệu, một pin dùng để nhận dữ liệu, một pin là nền, 22 pin còn lại có những chức năng khác nhau, hầu hết thường thường không dùng đến. Để gởi một ký tự cho terminal RS-232, máy tính mỗi lần chuyển một bit, ngoài ra có một bit bắt đầu, và sau đó có 1 hoặc 2 bit kết thúc để giới hạn một ký tự. Thường thường tốc độ vận chuyển là 1200, 2400, 4800, 9600…bps. Vì cả máy tính và terminal đều làm việc với ký tự mà phải liên lạc với nhau bằng bit nên hệ thống phải thiết kế bộ chuyển đổi gọi là UART. Bộ phận này được gắn vào các card giao tiếp của RS-232.



Hình 12.8 Terminal RS-232

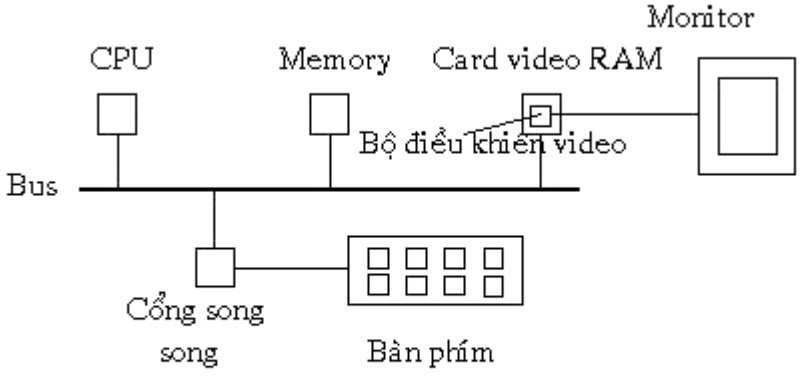
Hình 9.2.1-2. RS-232

Để in một ký tự, bộ điều khiển terminal ghi một ký tự lên card giao tiếp, sau đó sẽ chuyển cho UART.

Terminal RS-232 được chia làm nhiều loại. Dạng đơn giản nhất là terminal hardcopy(printing). Ví dụ các ký tự được nhập vào từ bàn phím và chuyển cho máy tính. Các ký tự từ máy tính xuất ra máy in. Dạng tương tự như vậy nhưng ký tự được xuất trên màn hình gọi là "glass ttys" do đó nó cũng có chức năng tương tự như trên. Terminals intelligent dùng trong máy tính nhỏ. Điểm khác biệt với loại trên dưới quan điểm hệ điều hành là nó sẽ gữi ký tự ASCII ESC sau những ký tự khác nhau dùng để chuyển cursor đến vị trí bất kỳ trên màn hình, chèn một dòng vào giữa màn hình. Blit là một terminal có bộ xử lý mạnh và một màn hình có 1024x800 điểm giao tiếp với máy tính bằng RS-232.

9.2.2. Terminal ánh xạ bộ nhớ

Dạng thứ hai của terminal là terminal ánh xạ bộ nhớ. Loại này không giao tiếp với máy tính qua đường serial. Nó là một phần của của hệ thống máy tính. Terminal ánh xạ bộ nhớ giao tiếp bằng một bộ nhớ đặc biệt gọi là video RAM, là một phần của bộ nhớ chính được định vị bởi CPU.



Hình 12.9 Terminal ánh xạ bộ nhớ

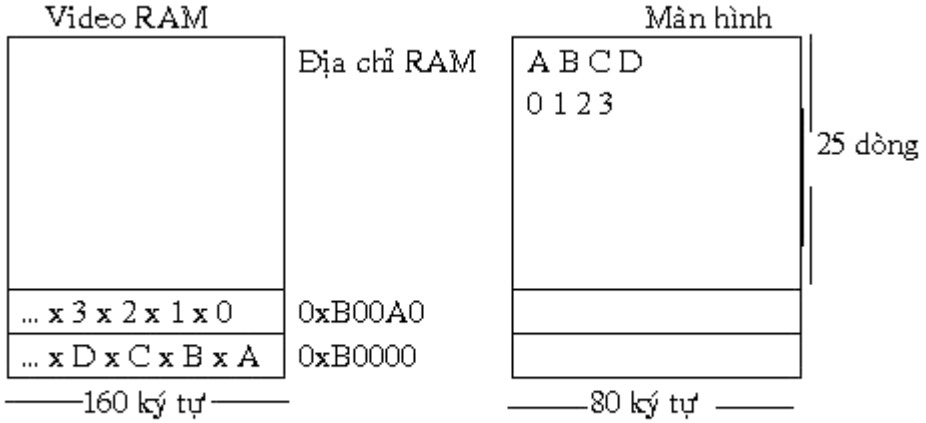
Hình 9.2.2-1. Terminal ánh xạ bộ nhớ

Trên card video RAM có một chip gọi là bộ điều khiển video. Chip này sẽ lấy thông tin từ video RAM và tạo ra tín hiệu video để điều khiển màn hình. Màn hình tạo

những tia điện tử quét từ trên xuống dưới. Thường thường có khoảng từ 200 đến 1200 dòng, trên mỗi dòng có từ 200 đến 1200 điểm. Mỗi điểm được gọi là pixel. Bộ điều khiển tín hiệu sẽ xác định mỗi điểm là sáng hay tối. Màn hình màu sẽ có ba tia là đỏ, lục và xanh.

Thông thường màn hình mono xây dựng một ký tự trong một box có chiều rộng là 9 pixel và chiều cao là 14 pixel (bao gồm khoảng trống giữa những ký tự) như vậy sẽ có 25 dòng và mỗi dòng có 80 ký tự. Mỗi khung được vẽ lại từ 45 đến 70 lần trong một giây. Bộ điều khiển video đặt các dòng 80 ký tự vào trong video RAM.

Một ví dụ về màn hình ánh xạ ký tự trên máy IBM PC. Một phần bộ nhớ chính bắt đầu từ địa chỉ 0xB000 cho màn hình đơn sắc và 0xB800 cho màn hình màu. Mỗi ký tự trên màn hình chiếm hai bytes trong bộ nhớ. Byte thấp chứa giá trị ASCII của ký tự, byte cao chứa thuộc tính như màu sắc, nhấp nháy v.v... Màn hình 80x25 sẽ chiếm 4000 bytes bộ nhớ video RAM



Hình 12.10 Ánh xạ màn hình

Hình 9.2.2-2. Terminal ánh xạ màn hình

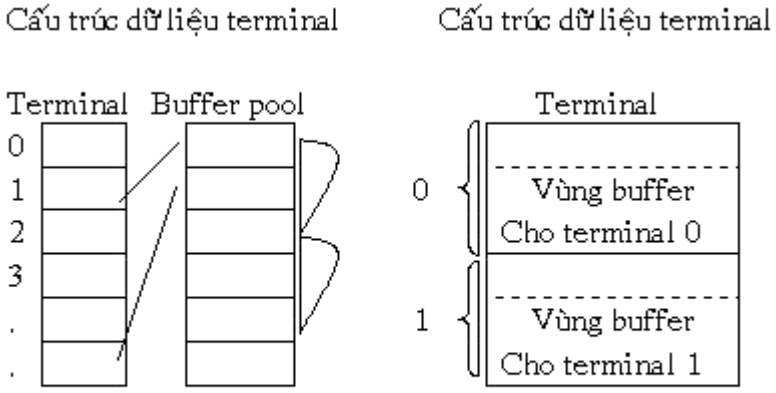
Khi CPU ghi một ký tự vào video RAM, nó xuất hiện trên màn hình theo mỗi lần hiển thị (1/50 giây cho mono, 1/60 cho màu). CPU có thể nạp 4K ảnh màn hình đã được tính trước vào video RAM trong vài phần triệu giây. Với tốc độ 9600 bps, ghi 2000 ký tự vào terminal RS-232 mất khoảng 2083 phần triệu giây. Terminal ánh xạ bộ nhớ cho phép truy xuất rất nhanh.

Terminal bit-map tương tự như vậy, ngoại trừ là mọi bit trong video RAM kiểm soát mỗi điểm trên màn hình. Màn hình có 1024x800 pixel cần dùng 100 K bộ nhớ nhưng khó thiết kế font và kích thước cho ký tự. Bàn phím giao tiếp thông qua cổng song song và giao tiếp RS-232. Mỗi khi gõ phím vào, CPU bị ngắt, bộ điều khiển bàn phím xác định kiểu ký tự được đọc từ cổng I/O. Đôi khi bàn phím chỉ cung cấp số hiệu phím , không phải mã ASCII. Trên IBM PC khi gõ phím A mã ký tự 30 được đưa vào thanh ghi I/O. Bộ điều khiển xác định ký tự là chữ hoa hay chữ thường hay là tổ hợp phím.

9.2.3. Phần mềm nhập

Bàn phím và màn hình hầu như độc lập với thiết bị. Công việc cơ bản của bộ điều khiển bàn phím là tập hợp các dữ liệu nhập từ bàn phím và chuyển cho chương trình của người sử dụng. Khi có một phím được gõ, nó sẽ gây một ngắt, và bộ điều khiển yêu cầu ký tự trong suốt quá trình ngắt này. Nếu ngắt được gây ra bởi một lời gọi ngắt của một

ngôn ngữ lập trình cấp thấp nó sẽ chuyển ký tự này cho chương trình đó. Nó sử dụng một buffer trong bộ nhớ chính và một thông điệp để báo cho bộ điều khiển biết đã có ký tự nhập. Một khi bộ điều khiển nhận một ký tự, nó sẽ bắt đầu xử lý. Nếu dưới dạng mã bàn phím, nó sẽ ánh xạ lại mã ASCII thật. Nếu terminal ở dạng cook, ký tự phải được lưu trữ cho tới khi nhận được hết dòng vì người sử dụng có thể xóa một phần nội dung của nó.
Có hai loại buffer thông thường. Dạng thứ nhất, bộ điều khiển chứa pool chính của buffer, mỗi buffer chứa 16 ký tự. Có một cấu trúc dữ liệu liên kết với nó, trong đó có chứa một con trỏ trỏ tới chuỗi trong buffer. Khi ký tự chuyển cho chương trình, nó sẽ được loại khỏi buffer. Dạng thứ hai là buffer trực tiếp có cấu trúc dữ liệu vì nếu tổ chức theo dạng thứ nhất sẽ không đủ bộ nhớ. Hình sau cho biết sự khác biệt giữa hai cách như hình sau:



Hình 12.11 Hai dạng cấu trúc dữ liệu terminal

Hình 9.2.3-1. Hai dạng cấu trúc dữ liệu Terminal

Mặt dù màn hình và bàn phím là hai thiết bị logic riêng biệt, nhưng mọi người đều quen với việc gõ ký tự và xem nó xuất hiện trên màn hình. Một số terminal cho phép tự động hiển thị lên màn hình những gì vừa gõ hoặc chỉ là những dấu . khi gõ password. Một số terminal không hiển thị ký tự được gõ do đó phải dựa vào phần mềm để hiển thị input, xử lý này gọi là echoing.

Echoing phức tạp vì chương trình phải xuất lên màn hình khi người dùng gõ vào. Bộ điều khiển bàn phím phải kiểm soát không cho ghi chồng lên output của chương trình. Echoing cũng gặp khó khăn khi người nhập gõ nhiều hơn 80 ký tự trên màn hình 80 ký tự một dòng. Một vấn đề khác là xử lý tab. Bộ điều khiển phải tính toán vị trí hiện thời cursor sau đó tính toán để chuyển cho chương trình và cho echoing và tính toán bao nhiêu khoảng trống phải hiển thị. Vấn đề tiếp theo là phải xử lý carriage return và line feed để chuyển cursor qua đầu dòng mới. Việc xử lý này tùy thuộc vào các hệ điều hành khác nhau. Ngoài ra phải kiểm soát tổ hợp ký tư và những ký tự xoá, lùi, hay các phím chức năng.

9.2.4. Phần mềm xuất

Phần mềm xuất thì đơn giản hơn nhập nhưng ở hai dạng thiết bị terminal RS-232 và ánh xạ bộ nhớ là khác nhau. Phương pháp thông thường của terminal RS-232 là có một buffer xuất cho mỗi loại terminal. Dạng buffer có thể là pool như buffer nhập hay là dạng tận hiến như input. Khi chương trình ghi lên terminal, trước tiên nó xuất lên buffer. Sau

khi đã xuất lên buffer, ký tự đầu tiên được xuất, sau đó bộ điều khiển tạm dừng, khi có một ngắt phát sinh, ký tự tiếp theo sẽ được xuất, và cứ tiếp tục như vậy.

Với terminal ánh xạ bộ nhớ, vấn đề đơn giản hơn. Những ký tự được in được xuất một lần từ chương trình người dùng được xuất lên video RAM. Với một số kýtự sẽ được xư lý đặc biệt. Ví dụ : backspace, carriage return, line feed, và bell (CTRL-G). Bộ điều khiển ánh xạ bộ nhớ, lưu giữ trong phần mềm vị trí của video RAM, vì vậy những ký tự in được được xuất trên đó theo thứ tự, các ký tự đặc biệt cũng được cập nhật thích hợp.

Khi một line feed được xuất tại cuối dòng của màn hình, màn hình sẽ cuộn. Thường thường phần cứng cung cấp một số giúp đỡ ở đây. Hầu hết những bộ điều khiển màn hình chứa một thanh ghi xác định vị trí của video RAM để bắt đầu đặt các byte vào dòng đầu tiên của màn hình. Phần mềm soạn thảo màn hình phải có nhiều xử lý phức tạp hơn là chỉ xuống dòng. Để tương thích, một số bộ điều khiển terminal hỗ trợ một số xử lý, thông thường là :

- Di chuyển cursor lên, xuống, trái, phải của một vị trí.
- Di chuyển cursor đến vị trí x,y.
- Chèn một ký tự hay chèn một dòng.
- Xóa một ký tự hay một dòng.
- Cuộn màn hình lên hoặc xuống n dòng.
- Xoá màn hinh từ vị trí cursor đến cuối dòng hoặc màn hình.
- Tạo tương phản, gạch dưới, nhấp nháy, hay mode thường.
- Tạo, hủy, di chuyển quản trị các cửa sổ.

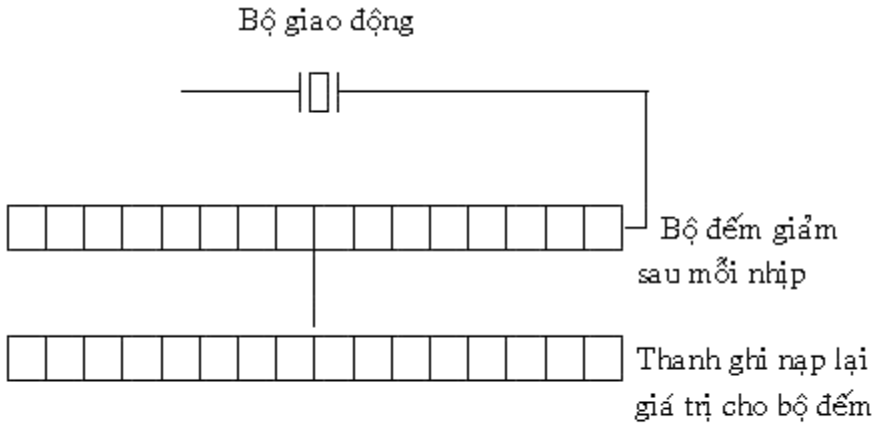
9.3. CÀI ĐẶT ĐỒNG HỒ

Đồng hồ còn được gọi là timer, là bộ phận rất cần thiết cho các thao tác của những hệ thống chia sẻ vì nhiều nguyên nhân khác nhau. Nó kiểm soát thời gian trong ngày và không cho phép một tiến trình nào đó độc chiếm CPU trong khi tồn tại những tiến trình khác. Phần mềm đồng hồ có thể xem như là device driver mặc dù đồng hồ không phải là thiết bị khối như đĩa hay thiết bị tuần tự như bàn phím, màn hình.

9.3.1. Phần cứng đồng hồ

Trong máy tính thường sử dụng hai loại đồng hồ nhưng cả hai đều khác với đồng hồ người sử dụng thông thường. Dạng đơn giản sử dụng đồng hồ với điện thế 110v hay 220v, và tạo ra ngắt theo mỗi chu kỳ của hiệu điện thế, từ 50 đến 60 MHz.

Một dạng khác của đồng hồ được xây dựng dựa trên ba thành phần : bộ dao động bằng thạch anh, bột đếm và bộ thanh ghi lưu trữ như hình vẽ. Dưới tác dụng của dòng điện, tinh thể thạch anh tạo ra dao động. Nhịp dao động rất chính xác theo thời gian, thường thường vào khoảng từ 5 đến 100 MHz tùy theo mỗi loại thạch anh. Tín hiệu này sẽ chuyển cho bộ đếm và bộ đếm sẽ thực hiện việc đếm lùi về 0. Khi bộ đếm có giá trị là 0, nó sẽ gây ra một ngắt CPU. Điều gì xảy ra tiếp theo là do hệ điều hành.



Hình 12.12 Cấu trúc của đồng hồ

Hình 9.3.1-1. Đồng hồ hệ thống

Dạng đồng hồ có thể lập trình có vài dạng thao tác. Thứ nhất là one-shot, khi đồng hồ khởi động, nó sẽ copy giá trị trong thanh ghi lưu trữ vào bộ đếm và sau đó giảm bộ đếm sau mỗi nhịp của thạch anh. Khi bộ đếm đến giá trị 0, nó sẽ gây ra một ngắt và dừng lại cho đến khi phần mềm khởi động lại nó. Thứ hai là square-wave, khi đến giá trị 0, nó sẽ gây ra một ngắt, bộ thanh ghi lưu trữ tự động nạp lại giá trị vào bộ đếm, và tiến trình sẽ được lập lại. Những ngắt phát sinh định kỳ này gọi là clock tick.

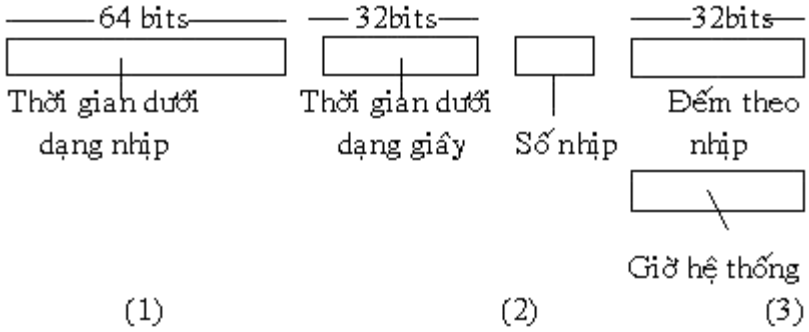
Ưu điểm của đồng hồ có thể lập trình là ngắt định kỳ được điều khiển bởi phần mềm. Nếu sử dụng tin thể thạch anh có tần số 1 MHz, bộ đếm sẽ có nhịp là mỗi micro giây. Với thanh ghi 16 bit, ngắt có thể được lập trình để xảy ra trong khoảng từ 1 đến 65535 msec.

9.3.2. Phần mềm đồng hồ

Tất cả mọi việc mà phần cứng đồng hồ thực hiện tạo ra các ngắt theo từng khoảng thời gian đều đặn. Mọi điều khác đều được thực hiện bởi phần mềm đồng hồ, là driver đồng hồ. Công việc của driver đồng hồ trên mỗi hệ điều hành là khác nhau, nhưng thường bao gồm những chức năng chính như sau :

- Quản lý thời gian trong ngày.
- Không cho phép tiến trình chạy lâu hơn thời gian mà nó được phép.
- Kế toán việc sử dụng CPU.
- Cung cấp watchdog timer cho một phần của chính hệ thống đó.

Chức năng đầu tiên của đồng hồ, quản lý thời gian trong ngày thì không khó. Chỉ cần tăng một bộ đếm sau mỗi nhịp của đồng hồ như đề cập ở trên. Vấn đề lưu ý ở đây là số lượng bit cho bộ counter. Với đồng hồ ở tần số 60 MHz, một bộ đếm 32 bit sẽ bị tràn sau hai năm. Do đó hệ thống không thể lưu trữ thời gian thực sự dưới dạng số nhịp từ 01/01/1970. Có ba cách giải quyết. Thứ nhất, dùng bộ đếm 64 bit, giải pháp này tốn kém. Thứ hai, lưu trữ dưới dạng giây thay vì nhịp vì 2^{32} giây sẽ là 136 năm. Thứ ba, đếm theo nhịp, nhưng liên hệ với thời gian của hệ thống khi khởi động.



Hình 12.13 Tổ chức lưu trữ của đồng hồ

Hình 9.3.2-1. Tổ chức lưu trữ của đồng hồ

Chức năng thứ hai là không cho phép một tiến trình thực hiện quá lâu. Khi nào một tiến trình bắt đầu, bộ lập lịch sẽ khởi gán giá trị cho bộ đếm, mỗi ngắt đồng hồ sẽ giảm giá trị của bộ đếm, khi nào giá trị bằng 0, bộ điều khiển đồng hồ sẽ yêu cầu bộ lập lịch thiết lập giá trị cho một tiến trình khác.

Chức năng thứ ba là kế toán việc sử dụng CPU. Cách thức chính xác nhất là sử dụng một bộ timer thứ hai, khác với timer hệ thống. Bộ timer thứ hai khởi động khi tiến trình bắt đầu và khi tiến trình kết thúc, timer này sẽ cho biết thời gian tiến trình đã thực hiện.

Phần lớn hệ thống cần thiết thiết lập timer. Gọi là watchdog timer. Ví dụ, để sử dụng đĩa mềm, hệ thống phải khởi động motor và chờ khoảng 500msec đạt được tốc độ. Vì vậy, ý tưởng tốt là phải sử dụng watchdog timer để chờ cho thao tác I/O tiếp theo, vào khoảng 3 giây, không tắt motor.

CHƯƠNG 10. BẢO VỆ VÀ AN TOÀN HỆ THỐNG

An toàn và bảo vệ hệ thống là chức năng khoông thể thiếu của các hệ điều hành hiện đại. Trong bài học này, chúng ta sẽ làm quen với các khái niệm về tổ chức an toàn hệ thống, cũng như các cơ chế bảo vệ hỗ trợ việc triển khai các chiến lược này.

10.1. Mục tiêu bảo vệ hệ thống (Protection)

Mục tiêu của việc bảo vệ hệ thống là:

- ***Bảo vệ chống lỗi của tiến trình*** : khi có nhiều tiến trình cùng hoạt động, lỗi của một tiến trình j phải được ngăn chặn không cho lan truyền trên hệ thống làm ảnh hưởng đến các tiến trình khác. Đặc biệt , qua việc phát hiện các lỗi tiềm ẩn trong các thành phần của hệ thống có thể tăng cường độ tin cậy hệ thống (reliability) .
- ***Chống sự truy xuất bất hợp lệ*** : Bảo đảm các bộ phận tiến trình sử dụng tài nguyên theo một cách thức hợp lệ được qui định cho nó trong việc khai thác các tài nguyên này .

Vai trò của bộ phận bảo vệ trong hệ thống là cung cấp một *cơ chế* để áp dụng các *chiến lược* quản trị việc sử dụng tài nguyên . Cần phân biệt khái niệm cơ chế và chiến lược:

- Cơ chế : xác định làm thế nào để thực hiện việc bảo vệ, có thể có các cơ chế phần mềm hoặc cơ chế phần cứng.
- Chiến lược: quyết định việc bảo vệ được áp dụng như thế nào : những đối tượng nào trong hệ thống cần được bảo vệ, và các thao tác thích hợp trên các đối tượng này

Để hệ thống có tính tương thích cao , cần phân tách các cơ chế và chiến lược được sử dụng trong hệ thống. Các chiến lược sử dụng tài nguyên là khác nhau tùy theo ứng dụng, và thường dễ thay đổi . Thông thường các chiến lược được lập trình viên vận dụng vào ứng dụng của mình để chống lỗi truy xuất bất hợp lệ đến các tài nguyên, trong khi đó hệ thống cung cấp các cơ chế giúp người sử dụng có thể thực hiện được chiến lược bảo vệ của mình.

10.2. Miền bảo vệ (Domain of Protection)

10.2.1. Khái niệm

Một hệ thống máy tính được xem như một tập các đối tượng *(objects)*. Một đối tượng có thể là một bộ phận phần cứng (CPU, bộ nhớ, ổ đĩa...) hay một thực thể phần mềm (tập tin, chương trình, semaphore...). Mỗi đối tượng có một định danh duy nhất để phân biệt với các đối tượng khác trong hệ thống, và chỉ được truy xuất đến thông qua các thao tác được định nghĩa chặt chẽ và được qui định ngữ nghĩa rõ ràng. Các thao tác có thể thực hiện được trên một đối tượng được xác định cụ thể tùy vào đối tượng.

Để có thể kiểm soát được tình hình sử dụng tài nguyên trong hệ thống, hệ điều hành chỉ cho phép các tiến trình được truy xuất đến các tài nguyên mà nó có quyền sử dụng, hơn nữa tiến trình chỉ được truy xuất đến các tài nguyên cần thiết trong thời điểm

hiện tại để nó hoàn thành tác vụ (nguyên lý ***need-to-know***) nhằm hạn chế các lỗi truy xuất mà tiến trình có thể gây ra trong hệ thống.

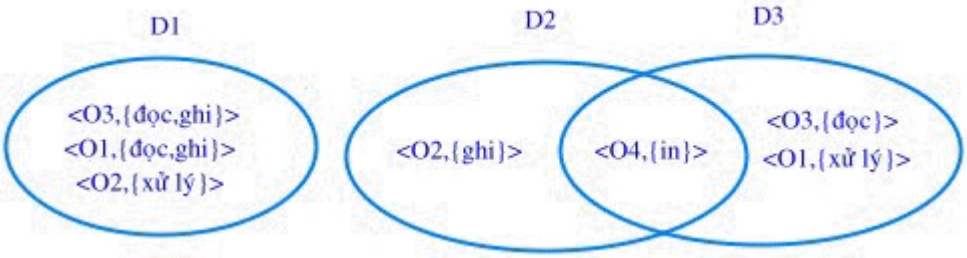
Mỗi tiến trình trong hệ thống đều hoạt động trong một miền bảo vệ *(protection domain)* nào đó. Một miền bảo vệ sẽ xác định các tài nguyên (đối tượng) mà những tiến trình hoạt động trong miền bảo vệ này có thể sử dụng, và các thao tác hợp lệ các tiến trình này có thể thực hiện trên những tài nguyên đó.

Ví dụ : <File F, {read, write}>

10.2.2. Cấu trúc của miền bảo vệ

Các khả năng thao tác trên một đối tượng được gọi là quyền truy xuất *(access right)*. Một *miền bảo vệ* là một tập các quyền truy xuất, mỗi quyền truy xuất được định nghĩa bởi một bộ hai thứ tự *<đối tượng, {quyền thao tác}* >.

Các miền bảo vệ khác nhau có thể giao nhau một số quyền truy xuất :



Hình 10.2.2-1. Hệ thống với 3 miền bảo vệ

Mối liên kết giữa một tiến trình và một miền bảo vệ có thể tĩnh hay động :

- **Liên kết tĩnh** : trong suốt thời gian sống của tiến trình, tiến trình chỉ hoạt động trong một miền bảo vệ . Trong trường hợp tiến trình trải qua các giai đoạn xử lý khác nhau, ở mỗi giai đoạn tiến trình có thể thao tác trên những tập tài nguyên khác nhau bằng các thao tác khác nhau. Tuy nhiên, nếu sử dụng liên kết tĩnh, rõ ràng là ngay từ đầu miền bảo vệ đã phải đặc tả tất cả các quyền truy xuất qua các giai đoạn cho tiến trình , điều này có thể khiến cho tiến trình có dư quyền trong một giai đoạn nào đó, và vi phạm nguyên lý need-to-know. Để có thể tôn trọng nguyên lý này, khi đó cần phải có khả năng cập nhật nội dung miền bảo vệ để có thể phản ánh các quyền tối thiểu của tiến trình trong miền bảo vệ tại một thời điểm!
- **Liên kết động** : cơ chế này cho phép tiến trình chuyển từ miền bảo vệ này sang miền bảo vệ khác trong suốt thời gian sống của nó. Để tiếp tục tuân theo nguyên lý **need-to-know**, thay vì sửa đổi nội dung của miền bảo vệ, có thể tạo ra các miền bảo vệ mới với nội dung thay đổi qua từng giai đoạn xử lý của tiến trình, và chuyển tiến trình sang hoạt động trong miền bảo vệ phù hợp theo từng thời điểm.

Một miền bảo vệ có thể được xây dựng cho:

- Một người sử dụng : trong trường hợp này, tập các đối tượng được phép truy xuất phụ thuộc vào định danh của người sử dụng, miền bảo vệ được chuyển khi thay đổi người sử dụng.
- Một tiến trình : trong trường hợp này, tập các đối tượng được phép truy xuất phụ thuộc vào định danh của tiến trình, miền bảo vệ được chuyển khi quyền điều khiển được chuyển sang tiến trình khác.

- Một thủ tục : trong trường hợp này, tập các đối tượng được phép truy xuất là các biến cục bộ được định nghĩa bên trong thủ tục, miền bảo vệ được chuyển khi thủ tục được gọi.

10.3. Ma trận quyền truy xuất (Access matrix)

Một cách trừu tượng, có thể biểu diễn mô hình bảo vệ trên đây như một ma trận quyền truy xuất (access matrix). Các dòng của ma trận biểu diễn các miền bảo vệ và các cột tương ứng với các đối tượng trong hệ thống. Phần tử *acess[i,j]* của ma trận xác định các quyền truy xuất mà một tiến trình hoạt động trong miền bảo vệ *Di* có thể thao tác trên đối tượng *Oj*.

object Domain	**F_1**	**F_2**	**F_3**	**Máy in**
D_1	đọc		đọc	
D_2				in
D_3		đọc	xử lý	
D_4	đọc ghi		đọc Ghi	

Hình 10.3-1. Ma trận quyền truy xuất

Cơ chế bảo vệ được cung cấp khi ma trận quyền truy xuất được cài đặt (với đầy đủ các thuộc tính ngữ nghĩa đả mô tả trên lý thuyết), lúc này người sử dụng có thể áp dụng các chiến lược bảo vệ bằng cách đặc tả nội dung các phần tử tương ứng trong ma trận _ xác định các quyền truy xuất ứng với từng miền bảo vệ , và cuối cùng, hệ điều hành sẽ quyết định cho phép tiến trình hoạt động trong miền bảo vệ thích hợp.
Ma trận quyền truy xuất cũng cung cấp một cơ chế thích hợp để định nghĩa và thực hiện một sự kiểm soát nghiêm nhặt cho cả phương thức liên kết tĩnh và động các tiến trình với các miền bảo vệ :
Có thể kiểm soát việc chuyển đổi giữa các miền bảo vệ nếu quan niệm miền bảo vệ cũng là một đối tượng trong hệ thống, và bổ sung các cột mô tả cho nó trong ma trận quyền truy xuất.
Khi đó tiến trình được phép chuyển từ miền bảo vệ **Di** sang miền bảo vệ **Dj** nếu phần tử **access**(i,j) chứa đựng quyền « chuyển » (switch).

object domain	**F_1**	**F_2**	**F_3**	**Máy in**	**D_1**	**D_2**	**D_3**	**D_4**
D_1	đọc		đọc			chuyển		
D_2				in			chuyển	chuyển
D_3		đọc	xử lý					
D_4	đọc ghi		đọc ghi		chuyển			

Hình 10.3-2. Ma trận quyền truy xuất với domain là một đối tượng

Có thể kiểm soát việc sửa đổi nội dung ma trận (thay đổi các quyền truy xuất trong một miền bảo vệ) nếu quan niệm bản thân ma trận cũng là một đối tượng.

Các thao tác sửa đổi nội dung ma trận được phép thực hiện bao gồm : sao chép quyền (copy), chuyển quyền (transfer), quyền sở hữu (owner), và quyền kiểm soát (control)

- ***Copy***: nếu một quyền truy xuất *R* trong *access[i,j]* được đánh dấu là *R** thì có thể sao chép nó sang một phần tử *access[k,j]* khác (mở rộng quyền truy xuất *R* trên cùng đối tượng *Oj* nhưng trong miền bảo vệ *Dk*).
- ***Transfer*** : nếu một quyền truy xuất *R* trong *access[i,j]* được đánh dấu là *R+* thì có thể chuyển nó sang một phần tử *access[k,j]* khác (chuyển quyền truy xuất *R+* trên đối tượng *Oj* sang miền bảo vệ *Dk*).
- ***Owner*** : nếu *access[i,j]* chứa quyền truy xuất *owner* thì tiến trình hoạt động trong miền bảo vệ *Di* có thể thêm hoặc xóa các quyền truy xuất trong bất kỳ phần tử nào trên cột *j* (có quyền thêm hay bớt các quyền truy xuất trên đối tượng *Oj* trong những miền bảo vệ khác).
- ***Control*** : nếu *access[i,j]* chứa quyền truy xuất *control* thì tiến trình hoạt động trong miền bảo vệ *Di* có thể xóa bất kỳ quyền truy xuất nào trong các phần tử trên dòng *j* (có quyền bỏ bớt các quyền truy xuất trong miền bảo vệ *Dj*).

object domain	**F_1**	**F_2**	**F_3**
D_1	xử lý		ghi+
D_2	xử lý	đọc*	xử lý
D_3	xử lý		

(a)

object domain	**F_1**	**F_2**	**F_3**
D_1	xử lý		
D_2	xử lý	đọc*	xử lý
D_3	xử lý	đọc	ghi+

(b)

Hình 10.3-3. Ma trận quyền truy xuất với quyền *copy* , *transfer* (a) trước, (b) sau cập nhật

object domain	**F_1**	**F_2**	**F_3**
D_1	owner xử lý		Ghi
D_2		đọc* owner	đọc* owner ghi*

D_3	xử lý		

(a)

object domain	**F_1**	**F_2**	**F_3**
D_1	owner xử lý		
D_2		owner đọc* ghi*	đọc* owner ghi*
D_3		ghi	

(b)

Hình 10.3-4. Ma trận quyền truy xuất với quyền *owner* (a) trước, (b) sau cập nhật

object domain	**F_1**	**F_2**	**F_3**	**Máy in**	**D_1**	**D_2**	**D_3**	**D_4**
D_1	đọc		đọc			chuyển		
D_2				in			chuyển	control chuyển
D_3		đọc	xử lý					
D_4	ghi		Ghi		chuyển			

Hình 10.3-5. Ma trận quyền truy xuất đã sửa đổi nội dung so với H5.3 nhờ quyền *control*

Giáo trình

Nguyên lý các hệ điều hành

CHƯƠNG 1: TỔNG QUAN VỀ HỆ ĐIỀU HÀNH 4
1.1 Khái niệm hệ điều hành 4
1.2 Lịch sử phát triển của hệ điều hành 5
1.3. Phân loại hệ điều hành 7
1.3.1 Hệ điều hành xử lý theo lô đơn giản 7
1.3.2 Hệ điều hành xử lý theo lô đa chương 8
1.3.3 Hệ điều hành chia xẻ thời gian 8
1.3.4 Hệ điều hành đa vi xử lý. 9
1.3.5 Hệ điều hành mạng 9
1.3.6 Hệ điều hành xử lý thời gian thực 9
1.4 Các thành phần của hệ điều hành 10
1.5 Các cấu trúc của hệ thống 15
1.6 Các tính chất cơ bản của hệ điều hành 21
1.7 Nguyên lý xây dựng chương trình hệ điều hành 22
1.8 Các hình thái giao tiếp 24
CHƯƠNG 2 QUẢN LÝ TIẾN TRÌNH 27
2.1 Tổng quan về tiến trình 27
2.1.1 Tiến trình (Process) và mô hình đa tiến trình (Multiprocess) 27
2.1.2 Tiểu trình (Thread) và mô hình đa tiểu trình (Multithread) 28
2.1.3 Phân loại tiến trình 29
2.1.4. Các trạng thái của tiến trình 31
2.1.5. Cấu trúc dữ liệu của khối quản lý tiến trình 32
2.1.6. Các thao tác điều khiển tiến trình 34
2.1.7 Cấp phát tài nguyên cho tiến trình 36
2.2. Điều phối tiến trình 37
2.2.1. Mục tiêu điều phối 38
2.2.2 Điều phối độc quyền và điều phối không độc quyền (preemptive/nopreemptive) 38
2.2.3. Các danh sách sử dụng trong quá trình điều phối. 40
2.2.4. Các chiến lược điều phối 41
2.3. Thông tin liên lạc giữa các tiến trình 48
2.3.1. Nhu cầu liên lạc giữa các tiến trình 48
2.3.2. Các Cơ Chế Thông Tin Liên lạc 49
2.4 Đồng bộ hoá tiến trình 55
2.4.1 Nhu cầu đồng bộ hóa (synchronisation) 55
2.4.2. Bài toán đồng bộ hoá 56
2.4.3 Các giải pháp đồng bộ hoá 59
2.5. Tắc nghẽn (Deadlock) 73
2.5.1. Định nghĩa: 73
2.5.2. Điều kiện xuất hiện tắc nghẽn 74
2.5.3. Các phương pháp xử lý tắc nghẽn 75
2.5.4 Ngăn chặn tắc nghẽn 76

2.5.5. Tránh tắc nghẽn 78
2.5.6. Hiệu chỉnh tắc nghẽn 83
CHƯƠNG 3 :QUẢN LÝ BỘ NHỚ CHÍNH 85
3.1 Tổ chức vùng nhớ 85
3.2 Mục tiêu của việc quản lý vùng nhớ 85
3.3 Không gian địa chỉ và không gian vật lý 86
3.4. Cấp phát liên tục 87
3.4.1 Hệ đơn chương 87
3.4.2 Hệ thống đa chương với phân vùng cố định 88
3.4.3 Hệ thống đa chương với phân vùng động 89
3.5. Cấp phát không liên tục 93
3.5.1 Kỹ thuật phân trang (Paging) 93
3.5.2. Phân đoạn (Segmentation) 101
3.5.3. Phân đoạn kết hợp phân trang (Paged segmentation) 105
3.6 Kỹ thuật bộ nhớ ảo (Virtual Memory) 107
3.6.1. Bộ nhớ ảo 107
3.6.2. Cài đặt bộ nhớ ảo 108
3.6.3.Các thuật toán thay thế trang 111
Chương 4 QUẢN LÝ VÙNG NHỚ PHỤ 116
4.1 Một số khái niệm dùng quản lý đĩa 116
4.2 Hệ thống bảng FAT 121
4.2.1 Quản lý file trên đĩa của MS_DOS 121
4.2.2 Hệ thống NTFS (New Technology File System) 126
4.3 Các thông số và thuật toán truy nhập đĩa 127
4.3.1Các thông số 127
4.3.2 Các thuật toán đọc đĩa 128
Chương 5 QUẢN LÝ VÀO RA 132
5.1 Khái niệm về hệ thống quản lý vào/ ra 132
5.2 Phần cứng vào/ra 132
5.2.1 Các thiết bị vào/ra 132
5.2.2 Tổ chức của chức năng I/O 134
5.2.3 Bộ điều khiển thiết bị 134
5.2.4 Truy nhập bộ nhớ trực tiếp DMA (Direct Memory Access) 136
5.3 Phần mềm vào/ra 136
5.3.1 Kiểm soát ngắt 137
5.3.2 Điều khiển thiết bị (device drivers) 137
5.3.3 Phần mềm nhập/xuất độc lập thiết bị 138
5.3.4 Phần mềm vào/ra phạm vi người sử dụng 139
Chương 6: HỆ THỐNG QUẢN L□FILE 141
6.1 File và các thuộc tính của file 141
6.2 Thư mục: khái niệm, hệ thống thư mục, tổ chức bên trong 143
6.3 Các phương pháp lưu giữ file 146

6.4 Hệ thống quản lý tập tin (File management system). 148
6.5 Các thao tác file.....149
6.6 Tổ chức file, truy nhập file.....150
6.7 Độ an toàn của hệ thống file.....151

không. Những máy này rất lớn với hơn 10000 ống chân không nhưng chậm hơn nhiều so với máy rẻ nhất ngày nay.

Mỗi máy được một nhóm thực hiện tất cả từ thiết kế, xây dựng lập trình, thao tác đến quản lý. Lập trình bằng ngôn ngữ máy tuyệt đối, thường là bằng cách dùng bảng điều khiển để thực hiện các chức năng cơ bản. Ngôn ngữ lập trình chưa được biết đến và hệ điều hành cũng chưa nghe đến.

Vào đầu thập niên 1950, phiếu đục lổ ra đời và có thể viết chương trình trên phiếu thay cho dùng bảng điều khiển.

Thế hệ 2 (1955 – 1965)

Sự ra đời của thiết bị bán dẫn vào giữa thập niên 1950 làm thay đổi bức tranh tổng thể. Máy tính trở nên đủ tin cậy hơn. Nó được sản xuất và cung cấp cho các khách hàng. Lần đầu tiên có sự phân chia rõ ràng giữa người thiết kế, người xây dựng, người vận hành, người lập trình, và người bảo trì.

Để thực hiện một công việc (một chương trình hay một tập hợp các chương trình), lập trình viên trước hết viết chương trình trên giấy (bằng hợp ngữ hay FORTRAN) sau đó đục lỗ trên phiếu và cuối cùng đưa phiếu vào máy. Sau khi thực hiện xong nó sẽ xuất kết quả ra máy in.

Hệ thống xử lý theo lô ra đời, nó lưu các yêu cầu cần thực hiện lên băng từ, và hệ thống sẽ đọc và thi hành lần lượt. Sau đó, nó sẽ ghi kết quả lên băng từ xuất và cuối cùng người sử dụng sẽ đem băng từ xuất đi in.

Hệ thống xử lý theo lô hoạt động dưới sự điều khiển của một chương trình đặc biệt là tiền thân của hệ điều hành sau này. Ngôn ngữ lập trình sử dụng trong giai đoạn này chủ yếu là FORTRAN và hợp ngữ.

Thế hệ 3 (1965 – 1980)

Trong giai đoạn này, máy tính được sử dụng rộng rãi trong khoa học cũng như trong thương mại. Máy IBM 360 là máy tính đầu tiên sử dụng mạch tích hợp (IC). Từ đó kích thước và giá cả của các hệ thống máy giảm đáng kể và máy tính càng phỗ biến hơn. Các thiết bị ngoại vi dành cho máy xuất hiện ngày càng nhiều và thao tác điều khiển bắt đầu phức tạp.

- Cung cấp các cơ chế giao tiếp giữa các tiến trình.

- Cung cấp cơ chế kiểm soát deadlock

b)Thành phần quản lý bộ nhớ chính :

Bộ nhớ là thiết bị lưu trữ duy nhất mà CPU có thể truy xuất trực tiếp. Bộ nhớ chính có thể xem như một mảng kiểu byte hay kiểu word. Mỗi phần tử đều có địa chỉ. Đó là nơi lưu dữ liệu được CPU truy xuất một cách nhanh chóng so với các thiết bị nhập/xuất. CPU đọc những chỉ thị từ bộ nhớ chính. Các thiết bị nhập/xuất cài đặt cơ chế DMA cũng đọc và ghi dữ liệu trong bộ nhớ chính. Thông thường bộ nhớ chính chứa các thiết bị mà CPU có thể định vị trực tiếp. Ví dụ CPU truy xuất dữ liệu từ đĩa, những dữ liệu này được chuyển vào bộ nhớ qua lời gọi hệ thống nhập/xuất.

Một chương trình muốn thi hành trước hết phải được ánh xạ thành địa chỉ tuyệt đối và nạp vào bộ nhớ chính. Khi chương trình thi hành, hệ thống truy xuất các chỉ thị và dữ liệu của chương trình trong bộ nhớ chính. Ngay cả khi tiến trình kết thúc, dữ liệu vẫn còn trong bộ nhớ cho đến khi một tiến trình khác được ghi chồng lên.

Hệ điều hành có những vai trò như sau trong việc quản lý bộ nhớ chính :

- Lưu giữ thông tin về các vị trí trong bộ nhớ đã được sử dụng và tiến trình nào đang sử dụng.

- Quyết định tiến trình nào được nạp vào bộ nhớ chính, khi bộ nhớ đã có thể dùng được.

- Cấp phát và thu hồi bộ nhớ khi cần thiết.

- Bảo vệ bộ nhớ

c) Thành phần quản lý bộ nhớ phụ

Bộ nhớ chính quá nhỏ để có thể lưu giữ mọi dữ liệu và chương trình, ngoài ra dữ liệu sẽ mất khi không còn được cung cấp năng lượng. Hệ thống máy tính ngày nay cung cấp ***hệ thống lưu trữ phụ***. Đa số các máy tính đều dùng đĩa để lưu trữ cả chương trình và dữ liệu. Hầu như tất cả chương trình : chương trình dịch, hợp ngữ, thủ tục, trình soạn thảo, định dạng... đều được lưu trữ trên đĩa cho tới khi nó được thực hiện, nạp vào trong bộ nhớ chính và cũng sử dụng đĩa để chứa dữ liệu và kết quả xử lý. Vai trò của hệ điều hành trong việc quản lý đĩa :

- Quản lý vùng trống trên đĩa.

- Định vị lưu trữ thông tin trên đĩa.

- Lập lịch cho vấn đề ghi/đọc thông tin trên đĩa của đầu từ.

d) Quản lý hệ thống vào/ ra :

Một trong những mục tiêu của hệ điều hành là *che dấu* những đặc thù của các thiết bị phần cứng đối với người sử dụng thay vào đó là một lớp thân thiện hơn, người sử dụng dể thao tác hơn. Một hệ thống vào/ra bao gồm :

- Thành phần quản lý bộ nhớ chứa vùng đệm (buffering), lưu trữ (caching) và spooling (vùng chứa).

- Giao tiếp điều khiển thiết bị (device drivers) tổng quát.

- Bộ điều khiển cho các thiết bị xác định.

Chỉ có bộ điều khiển cho các thiết bị xác định mới hiểu đến cấu trúc đặc thù của thiết bị mà nó mô tả.

e) Thành phần quản lý tập tin :

Máy tính có thể lưu trữ thông tin trong nhiều dạng thiết bị vật lý khác nhau : băng từ, đĩa từ, đĩa quang, ... Mỗi dạng có những đặc thù riêng về mặt tổ chức vật lý. Mỗi thiết bị có một bộ kiểm soát như bộ điều khiển đĩa (disk driver) và có những tính chất riêng. Những tính chất này là tốc độ, khả năng lưu trữ, tốc độ truyền dữ liệu và cách truy xuất.

Để cho việc sử dụng hệ thống máy tính thuận tiện, hệ điều hành cung cấp một cái nhìn logic đồng nhất về hệ thống lưu trữ thông tin. Hệ điều hành định nghĩa một đơn vị lưu trữ logic là tập tin. Hệ điều hành tạo một ánh xạ từ tập tin đến vùng thông tin trên đĩa và truy xuất những tập tin này thông qua thiết bị lưu trữ.

Một tập tin là một tập hợp những thông tin do người tạo ra nó xác định. Thông thường một tập tin đại diện cho một chương trình và dữ liệu. Dữ liệu của tập tin có thể là số, là ký tự, hay ký số.

Vai trò của hệ điều hành trong việc quản lý tập tin :

- Tạo và xoá một tập tin.

- Tạo và xoá một thư mục.

- Hỗ trợ các thao tác trên tập tin và thư mục.

- Tạo mối quan hệ giữa tập tin và bộ nhớ phụ chứa tập tin.

- Sao lưu dự phòng các tập tin trên các thiết bị lưu trữ.

- Bảo vệ tập tin khi có hiện tượng truy xuất đồng thời

-Tạo cơ chế truy xuất tập tin thông qua tên tập tin

f) Hệ thống bảo vệ :

Trong một hệ thống nhiều người sử dụng và cho phép nhiều tiến trình diễn ra đồng thời, các tiến trình phải được bảo vệ đối với những hoạt động khác.Do đó, hệ thống cung cấp cơ chế để đảm bảo rằng tập tin, bộ nhớ, CPU, và những tài nguyên khác chỉ được truy xuất bởi những tiến trình có quyền. Ví dụ, bộ nhớ đảm bảo rằng tiến trình chỉ được thi hành trong phạm vi địa chỉ của nó. Bộ thời gian đảm bảo rằng không có tiến trình nào độc chiếm CPU. Cuối cùng các thiết bị ngoại vi cũng được bảo vệ.

Hệ thống bảo vệ là một cơ chế kiểm soát quá trình truy xuất của chương trình, tiến trình, hoặc người sử dụng với tài nguyên của hệ thống. Cơ chế này cũng cung cấp cách thức để mô tả lại mức độ kiểm soát.

Hệ thống bảo vệ cũng làm tăng độ an toàn khi kiểm tra lỗi trong giao tiếp giữa những hệ thống nhỏ bên trong.

g) Thành phần thông dịch lệnh :

Một trong những phần quan trọng của hệ điều hành là hệ thống thông dịch lệnh, đó là giao tiếp giữa người sử dụng và hệ điều hành. Thành phần này chính là Shell.

Một số hệ điều hành chứa Shell trong nhân của nó, một số hệ điều hành khác thì Shell được thiết kế dưới dạng một chương trình đặc biệt.

Shell là một bộ phận hay một tiến trình đặc biệt của hệ điều hành, nó có nhiệm vụ nhận lệnh của người sử dụng, phân tích lệnh và phát sinh tiến trình mới để thực hiện yêu cầu của lệnh, tiến trình mới này được gọi là tiến trình đáp ứng yêu cầu.

Shell nhận lệnh thông qua cơ chế dòng lệnh, đó chính là nơi giao tiếp giữa người sử dụng và hệ điều hành, mỗi hệ điều hành khác nhau có cơ chế dòng lệnh khác nhau, với MS_DOS đó là con trỏ lệnh và dấu nhắc của hệ điều hành (C:\>), với Windows 9x đó là nút Start/Run. Tập tin Command.com chính là Shell của MS_DOS.

Trong môi trường đơn nhiệm, ví dụ như MS_DOS, khi tiến trình đáp ứng yêu cầu hoạt động thì Shell sẽ chuyển sang trạng thái chờ, để chờ cho đến khi tiến trình đáp ứng yêu cầu kết thúc thì Shell trở lại trạng thái sẵn sang nhận lệnh mới.

Trong môi trường hệ điều hành đa nhiệm, ví dụ như Windows 9x sau khi phát sinh tiến trình đáp ứng yêu cầu và đưa nó vào trạng thái hoạt động thì Shell sẽ chuyển sang trạng thái sẵn sang nhận lệnh mới, nhờ vậy Shell có khả năng khởi tạo nhiều tiến trình đáp ứng yêu cầu để nó hoạt động song song với nhau, hay chính xác hơn trong môi trường hệ điều hành đa nhiệm, người sử dụng có thể khởi tạo nhiều chương trình để nó hoạt động đồng thời với nhau.

Chú ý: hầu hết các ngôn ngữ lập trình đều hỗ trợ các công cụ để người sử dụng hay ngươi lập trình có thể gọi Shell ngay trong các ứng dụng của họ. Khi một ứng dụng cần gọi thực hiện một chương trình nào đó thì:

Trong Assembly, các ứng dụng gọi hàm 4Bh/21h của MS_DOS.

Trong Pascal, các ứng dụng gọi thủ tục Exec

Trong VisualBasic, các ứng dụng gọi hàm/ thủ tục shell. Ví dụ dòng lệnh sau: Shell”C:\Windows\Notepad.exe” có thể gọi thực hiện chương trình Notepad của Windows.

Trong Windows 9x/WindowsNT, các ứng dụng gọi hàm ShellExecute.

Chú ý: Cần phải phân biệt sự khác nhau giữa Shell và System Call. Shell tạo môi trường giao tiếp giữa người sử dụng và hệ điều hành, System call tạo môi trường giao tiếp giữa chương trình người sử dụng và hệ điều hành.

h) Thành phần quản lý mạng

Xem xét đến các vấn đề liên lạc giữa các tiến trình, chia sẻ tài nguyên chung, vấn đề bảo mật trên các tiến trình trong các hệ thống khác nhau.

Mặc dù MS-DOS có cấu trúc nhưng giữa giao diện và chức năng không có sự phân chia rõ rệt. Các chương trình ứng dụng có thể truy xuất trực tiếp các thủ tục nhập xuất cơ bản và ghi trực tiếp lên màn hình hay bộ điều khiển đĩa.

b) Các hệ thống phân lớp

Hệ thống được chia thành một số lớp, mỗi lớp dược xây dựng dựa vào các lớp bên trong. Lớp trong cùng thường là phần cứng, lớp ngoài cùng là giao tiếp với người sử dụng.

Mỗi lớp là một đối tượng trừu tượng chứa đựng bên trong nó các dữ liệu và các thao tác xử lý dữ liệu đó. Lớp n chứa đựng một cấu trúc dữ liệu và các thủ tục có thể được gọi bởi lớp n+1 hoặc ngược lại có thể gọi các thủ tục ở lớp n-1.

Nhận xét:

-Hệ thống này mang tính đơn thể, nên dễ cài đặt, tìm lỗi và kiểm chứng hệ thống.

Ưu điểm là tính module. Các lớp được chọn dựa trên cơ sở lớp trên sử dụng chức năng và các dịch vụ chỉ của lớp dưới nó. Tiếp cận này đơn giản hóa việc gỡ rối và kiểm tra hệ thống. Lớp đầu tiên có thể được gỡ rối mà không có bất cứ sự quan tâm nào cho lớp còn lại của hệ thống. Bởi vì theo định nghĩa, nó chỉ sử dụng phần cứng cơ bản để cài đặt các chức năng của nó. Một khi lớp đầu tiên được gỡ rối, chức năng sửa lỗi của nó có thể được đảm đương trong khi lớp thứ 2 được gỡ rối, ...Nếu một lỗi được tìm thấy trong khi gỡ rối cho một lớp xác định, lỗi phải được nằm trên lớp đó vì các lớp bên dưới đã được gỡ rối rồi. Do đó, thiết kế và cài đặt hệ thống được đơn giản hóa khi hệ thống được phân chia thành nhiều lớp.

Mỗi lớp được cài đặt chỉ với các thao tác được cung cấp bởi các lớp bên dưới. Một lớp không cần biết các thao tác được cài đặt như thế nào; nó chỉ cần biết các thao tác đó làm gì. Do đó, mỗi lớp che giấu sự tồn tại của cấu trúc dữ liệu, thao tác và phần cứng từ các lớp cấp cao hơn.

-Các nhà thiết kế gặp khó khăn trong việc xác định số lượng lớp, thứ tự và chức năng của mỗi lớp.

Khó khăn chính của tiếp cận phân lớp liên quan tới việc định nghĩa cẩn thận các lớp vì một lớp chỉ có thể sử dụng các lớp bên dưới nó. Thí dụ, trình điều khiển thiết bị cho không gian đĩa được dùng bởi các giải thuật bộ nhớ ảo phải nằm ở tại cấp thấp hơn trình điều khiển thiết bị của các thủ tục quản lý bộ nhớ vì quản lý bộ nhớ yêu cầu khả năng sử dụng không gian đĩa.

Các yêu cầu có thể không thật sự rõ ràng. Thường thì các trình điều khiển lưu trữ dự phòng nằm trên bộ định thời CPU vì trình điều khiển cần phải chờ nhập/xuất và CPU có thể được định thời lại trong thời gian này. Tuy nhiên, trên hệ thống lớn, bộ định thời có thể có nhiều thông tin hơn về tất cả quá trình đang hoạt động hơn là có thể đặt vừa trong bộ nhớ. Do đó, thông tin này có thể cần được hoán vị vào và ra bộ nhớ, yêu cầu thủ tục trình điều khiển lưu trữ dự phòng nằm bên dưới bộ định thời CPU.

-Trong một số trường hợp lời gọi thủ tục có thể lan truyền đến các thủ tục khác ở các lớp bên trên nên chi phí cho vấn đề truyền tham số và chuyển đổi ngữ cảnh tăng lên, dẫn đến lời gọi hệ thống trong cấu trúc này thực hiện chậm hơn so với các cấu trúc khác

Vấn đề cuối cùng với các cài đặt phân lớp là chúng có khuynh hướng ít hiệu quả hơn các loại khác. Thí dụ, khi chương trình người dùng thực thi thao tác nhập/xuất, nó thực thi một lời gọi hệ thống. Lời gọi hệ thống này được bẫy (trapped) tới lớp nhập/xuất, nó yêu cầu tầng quản lý bộ nhớ, sau đó gọi tầng định thời CPU, sau đó được truyền tới phần cứng. Tại mỗi lớp, các tham số có thể được hiệu chỉnh, dữ liệu có thể được truyền,…Mỗi tầng thêm chi phí cho lời gọi hệ thống; kết quả thực sự là lời gọi hệ thống mất thời gian lâu hơn khi chúng thực hiện trên hệ thống không phân tầng.

Cấu trúc lớp này lần đầu tiên được thiết kế và áp dụng cho hệ điều hành THE (Technische Hogeschool Eindhoven). Hệ thống này được chia thành sáu lớp như hình sau:

Lớp 5	Chương trình của người sử dụng
Lớp 4	Tạo buffer cho thiết bị nhập xuất
Lớp 3	Device driver thao tác màn hình
Lớp 2	Quản lý bộ nhớ
Lớp 1	Lập lịch CPU
Lớp 0	Phần cứng

Hình 1.3 Cấu trúc của hệ điều hành THE

Các ví dụ khác như cấu trúc lớp của hệ điều hành VENUS và OS/2

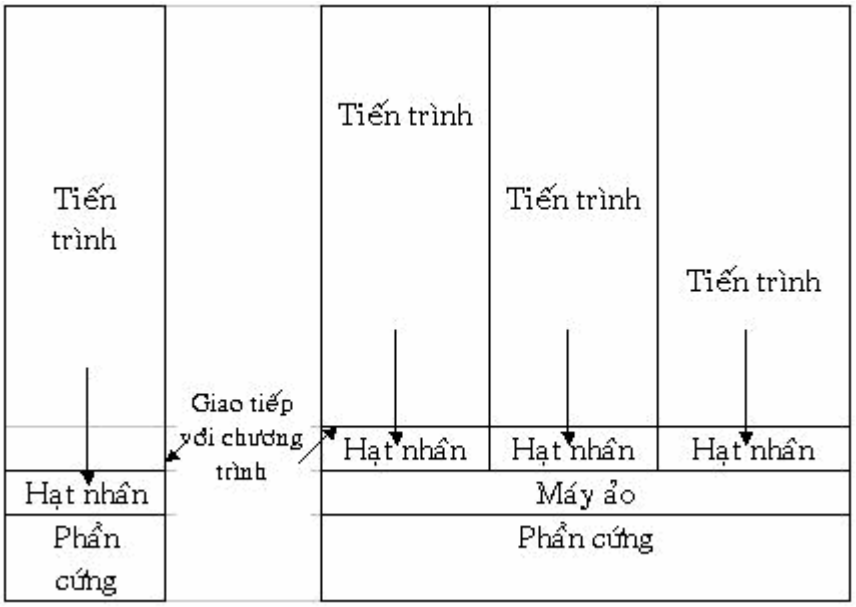
c) Máy ảo (Virtual Machine)

Các máy ảo là những bản sao ảo chính xác các đặc tính phần cứng của máy tính thực sự và cho phép hệ điều hành hoạt động trên đó như trên phần cứng thực sự. Phần nhân hệ thống thực hiện giám sát máy ảo chịu trách nhiệm giao tiếp với phần cứng, chia sẻ tài nguyên hệ thống để tạo ra nhiều máy ảo, hoạt động độc lập với nhau, để cung cấp cho lớp bên trên.

Với cấu trúc này mỗi tiến trình hoạt động trên một máy ảo độc lập và nó có cảm giác như đang sở hữu một máy tính thực sự.

Mục đích của việc sử dụng máy ảo là xây dựng các hệ thống đa chương với nhiều tiến trình thực hiện đồng thời, mỗi tiến trình được cung cấp một máy ảo với đầy đủ tài nguyên, tất nhiên là tài nguyên ảo, để nó thực hiện được.

Vấn đề phức tạp nhất của máy ảo là hệ thống đĩa. Giả sử hệ thống chỉ có ba bộ điều khiển đĩa nhưng có tới bảy máy ảo. Như vậy không thể gán cho mỗi máy ảo một bộ điều khiển đĩa và giải pháp là xây dựng hệ thống đĩa ảo.



Hình 1.4 So sánh giữa máy thực và máy ảo

Nhận xét:

- Việc cài đặt các phần mềm giả lặp phần cứng để tạo ra máy ảo thường rất khó khăn và phức tạp.

- Trong hệ thống này vấn đề bảo vệ tài nguyên hệ thống và tài nguyên đã cấp phát cho tiến trình, sẽ trở nên đơn giản hơn vì mỗi tiến trình thực hiện trên một máy tính (ảo) độc lập với nhau nên việc tranh chấp tài nguyên là không thể xảy ra.

- Nhờ hệ thống máy ảo mà một ứng dụng được xây dựng trên hệ điều hành có thể hoạt động được trên hệ điều hành khác.

Trong môi trường hệ điều hành Windows 9x người sử dụng có thể thực hiện được các ứng dụng được thiết kế để thực hiện trên môi trường MS_DOS, sở dĩ như vậy vì Windows đã cung cấp cho các ứng dụng này một máy ảo DOS (VMD: Virtual Machine DOS) để nó hoạt động như đang hoạt động trong hệ điều hành DOS. Tương tự như trong môi trường hệ điều hành Windows NT người sử dụng có thể thực hiện được các ứng dụng được thiết kế trên một số hệ điều hành khác, có được điều này là nhờ cấu trúc của Windows NT có chứa các hệ thống con (subsystem) môi trường tương thích với các môi trường hệ điều hành khác như : Win32, OS/2,..các ứng dụng khi cần thiết thực hiện trên Windows NT sẽ thực hiện trong các hệ thống con môi trường tương ứng, đúng với môi trường mà ứng dụng đó được tạo ra.

d) Mô hình Client/Server

Các hệ điều hành hiện đại thường chuyển dần các nhiệm vụ của hệ điều hành ra các lớp bên ngoài nhằm thu nhỏ phần cốt lõi của hệ điều hành thành hạt nhân cực tiểu (kernel) sao cho chỉ phần hạt nhân này chỉ phụ thuộc vào phần cứng. Để thực hiện được điều này hệ điều hành xây dựng theo mô hình Client/Server, theo mô hình này hệ điều hành bao gồm nhiều tiến trình đóng vai trò server có các chức năng chuyên biệt như quản lý tiến trình, quản lý bộ nhớ,…phần hạt nhân của hệ điều hành chỉ thực hiện nhiệm vụ tạo cơ chế thông tin liên lạc giữa các tiến trình client và các tiến trình server.

Như vậy các tiến trình trong hệ thống được chia thành 2 loại:

- Tiến trình bên ngoài hay tiến trình của chương trình người sử dụng được gọi là các tiến trình client.

- Tiến trình của hệ điều hành được gọi là các tiến trình server.

Khi cần thực hiện một chức năng hệ thống các tiến trình Client sẽ gởi yêu cầu tới tiến trình server tương ứng, tiến tình server sẽ xử lý và trả lời kết quả cho tiến trình client.

Nhận xét:

-Hệ thống này dễ thay đổi và dễ mở rộng hệ điều hành. Để thay đổi các chức năng của hệ điều hành chỉ cần thay đổi server tương ứng, để mở rộng hệ điều hành chỉ cần thêm các server mới vào hệ thống.

-Các tiến trình server của hệ điều hành hoạt động trong chế độ không đặc quyền nên không thể truy cập trực tiếp tới phần cứng, điều này giúp cho hệ thống được bảo vệ tốt hơn.

Ví dụ: WindowsNT

- Các module đồng cấp quan hệ với nhau thông qua dữ liệu vào và ra.

- Tồn tại quan hệ phân cấp khi các liên kết các module tạo thành những module có khả năng giải quết những vấn đề phức tạp hơn.

b) Nguyên tắc tương đối trong định vị

Các modul chương trình được viết theo địa chỉ tương đối kể từ đầu bộ nhớ. Khi thực hiện chúng mới được định vị tại vùng bộ nhớ cụ thể. Nguyên tắc này cho phép hệ thống sử dụng bộ nhớ một cách linh hoạt và hệ điều hành không bị phụ thuộc vào cấu hình bộ nhớ cụ thể.

c) Nguyên tắc Macroproccessor

Theo nguyên tắc này khi có nhiệm vụ cụ thể hệ thống sẽ xây dựng các phiếu yêu cầu, liệt kê các bước phải thực hiện và trên cơ sở đó xây dựng chương trình tương ứng, sau đó thực hiện chương trình nói trên. Mọi hệ điều hành đều phải xây dựng nguyên lý này trong đối thoại giữa người và máy trên ngôn ngữ vận hành. Dĩ nhiên độ sâu trong việc phân tích và xây dựng chương trình là khác nhau ở những hệ thống khác nhau. Chính nguyên tắc này đã làm cho quá trình đối thoại được linh hoạt mà không cần tới một chương trình dịch phức tạp.

d) Nguyên tắc khởi tạo trong cài đặt

Nguyên tắc Macroproccessor có thể áp dụng không những với từng nhiệm vụ mà còn với toàn bộ HĐH hoặc các thành phần của nó. Người sử dụng được cung cấp các bộ chương trình cài đặt. Chương trình cài đặt sẽ tạo phiên bản làm việc thích hợp với các tham số kỹ thuật hiện có, loại bỏ những modul không cần thiết để có một phiên bản tối ưu cả về cấu trúc lẫn phương thức hoạt động

e) Nguyên tắc lập chức năng

Mỗi công việc bao giờ cũng có nhiều cách thực hiện khác nhau với những tổ hợp modul khác nhau. Nguyên tắc này trước hết đảm bảo độ an toàn của hệ thống cao: vẫn có thể khai thác hệ thống bình thường ngay cả khi thiếu hoặc hỏng nhiều thành phần hệ thống. Ngoài ra, với nguyên tắc này người sử dụng sẽ thoải mái hơn khi giao tiếp với hệ thống: với một công việc, ai nhớ hoặc thích phương tiện nào thì sử dụng phương tiện đó. Như vậy người sử dụng khai thác được cả những hiệu ứng phụ của các modul chương trình. Đôi khi trong hệ thống tồn tại nhiều modul khác nhau cùng giải quyết một vấn đề, chẳng hạn có nhiều chương tình dịch cho một ngôn ngữ thuật toán

b) Hình thái thực đơn

Người sử dụng giao tiếp với hệ điều hành thông qua các thực đơn, các thực đơn thường có dạng trải xuống(popup). Mỗi thực đơn con tương ứng với một chức năng. Các tham số có thể được đưa vào thông qua giao tiếp với người sử dụng.

-Ưu điểm:

Hình thái này không yêu cầu nhớ lệnh

Người sử dụng có thể truy nhập vào thực đơn qua bàn phím hoặc qua chuột

- Nhược điểm:

Hình thái giao tiếp này bị cản trở bởi hàng rào ngôn ngữ.

Đôi khi các từ trên thực đơn không nêu bật được chức năng của nó.

c) Hình thái cửa sổ-biểu tượng

Người sử dụng giao tiếp với hệ điều hành thông qua các thanh công cụ và các biểu tượng. Mỗi biểu tượng tương ứng với một chức năng. Các tham số có thể được đưa vào thông qua giao tiếp với người sử dụmg.

-Ưu điểm:

Hình thái này không yêu cầu nhớ lệnh

Người sử dụng không bị hàng rào ngôn ngữ gây cản trở.

- Nhược điểm:

Có thể có rất nhiều biểu tượng do đó gây sự nhập nhằng về chức năng.

Không thuận lợi khi thao tác bằng bàn phím.

d) Hình thái kết hợp

HĐH thường kết hợp nhiều hình thái giao tiếp để tạo ra tính thân thiện với người sử dụng. Ví dụ: việc kết hợp thực đơn với các biểu tượng, hoặc kết hợp giữa các biểu tượng với các từ gợi ý.

Hình thái giao tiếp kết hợp này khắc phục được các nhược điểm của các hình thái giao tiếp đơn lẻ.

+ Tiến trình song song có quan hệ thông tin: trong quá trình hoạt động các tiến trình thường trao đổi thông tin với nhau, trong một số tường hợp tiến trình gởi thông báo cần phải nhận được tín hiệu từ tiến trình nhận để tiếp tục, điều này dễ dẫn đến bế tắc khi tiến trình nhận tín hiệu không ở trong trạng thái nhận hay tiến trình gửi không ở trong trạng thái nhận thông báo trả lời.

+ Tiến trình song song độc lập: là các tiến trình hoạt động song song nhưng không có quan hệ thông tin với nhau, trong trường hợp này hệ điều hành phải thiết lập cơ chế bảo vệ dữ liệu của các tiến trình, và cấp phát tài nguyên cho các tiến trình một cách hợp lý.

-Tiến trình song song phân cấp: Trong quá trình hoạt động một tiến trình có thể khởi tạo các tiến trình khác hoạt động song song với nó, tiến trình khởi tạo được gọi là tiến trình cha, tiến trình được tạo gọi là tiến trình con. Trong mô hình này hệ điều hành phải giải quyết vấn đề cấp phát tài nguyên cho các tiến trình con. Hệ điều hành đưa ra hai mô hình quản lý tài nguyên: Thứ nhất, mô hình tập trung, hệ điều hành chịu trách nhiệm phân phối tài nguyên cho tất cả các tiến trình trong hệ thống. Thứ hai, mô hình phân tán hệ điều hành cho phép tiến trình con nhận tài nguyên từ tiến trình cha, tức là tiến trình khởi tạo có nhiệm vụ nhận tài nguyên từ hệ điều hành để cấp phát cho các tiến trình mà nó tạo ra, và nó có nhiệm vụ thu hồi lại tài nguyên đã cấp phát trả về cho hệ điều hành trước khi kết thúc.

-Tiến trình song song đồng mức: là các tiến trình hoạt động song song sử dụng chung tài nguyên theo nguyên tắc lần lượt, mỗi tiến trình sau một khoảng thời gian chiếm giữ tài nguyên phải tự động trả lại tài nguyên cho tiến trình kia.

Trong tài liệu này chúng ta chỉ khảo sát sự hoạt động của các tiến trình song song trên các hệ thống đơn bộ xử lý.

Đối với người sử dụng thì trong hệ thống chỉ có 2 nhóm tiến trình. Thứ nhất là các tiến trình của hệ điều hành. Thứ hai, là các tiến trình của chương trình người sử dụng. Các tiến trình của hệ điều hành hoạt động trong chế độ đặc quyền, nhờ đó mà nó có thể truy xuất vào vào các vùng dữ liệu được bảo vệ của hệ thống. Trong khi đó các tiến trình của chương trình người sử dụng hoạt động trong chế độ không đặc quyền, nên nó không thể truy xuất vào hệ thống thông qua các tiến trình của hệ điều hành bằng cách thực hiện một lời gọi hệ thống.

Trạng thái CPU: bao gồm nội dung các thanh ghi, quan trọng nhất là con trỏ lệnh IP lưu trữ địa chỉ câu lệnh kế tiếp tiến trình sẽ xử lý. Các thông tin này cần được lưu trữ khi xảy ra một ngắt, nhằm có thể cho phép phục hồi hoạt động của tiến trình đúng như trước khi bị ngắt.

Bộ xử lý: dùng cho máy có cấu hình nhiều CPU, xác định số hiệu CPU mà tiến trình đang sử dụng.

Bộ nhớ chính: danh sách các khối nhớ được cấp cho tiến trình.

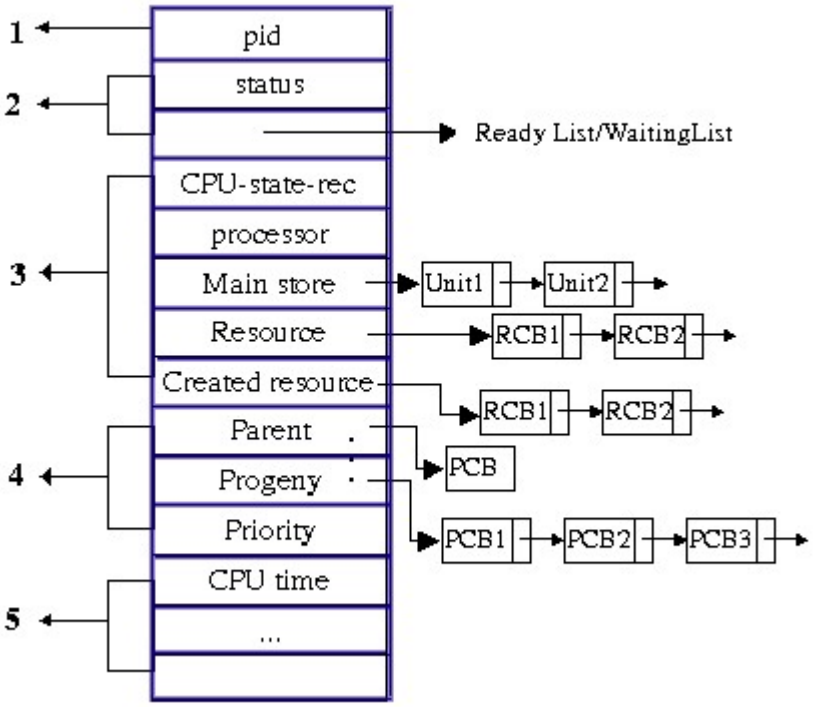
Tài nguyên sử dụng: danh sách các tài mguyên hệ thống mà tiến trình đang sử dụng.

Tài nguyên tạo lập: danh sách các tài nguyên được tiến trình tạo lập.

-Thông tin giao tiếp (4): phản ánh các thông tin về quan hệ của tiến trình với các tiến trình khác trong hệ thống :

Tiến trình cha: tiến trình tạo lập tiến trình này .

Tiến trình con: các tiến trình do tiến trình này tạo lập .



Hình 2.4 Khối mô tả tiến trình

sẽ tạo ra một *cây tiến trình*. Ví dụ trong UNIX: lời gọi hệ thống là fork

Hình 2.5 Một cây tiến trình trong hệ thống UNIX

Các công việc hệ điều hành cần thực hiện khi tạo lập tiến trình bao gồm :

- Định danh cho tiến trình mới phát sinh

-Đưa tiến trình vào danh sách quản lý của hệ thống

- Xác định độ ưu tiên cho tiến trình

- Tạo PCB cho tiến trình

-Cấp phát các tài nguyên ban đầu cho tiến trình

Khi một tiến trình tạo lập một tiến trình con, tiến trình con có thể sẽ được hệ điều hành trực tiếp cấp phát tài nguyên hoặc được tiến trình cha cho thừa hưởng một số tài nguyên ban đầu.

Khi một tiến trình tạo tiến trình mới, tiến trình ban đầu có thể xử lý theo một trong hai khả năng sau :

Tiến trình cha tiếp tục xử lý đồng hành với tiến trình con. Ví dụ UNIX

Tiến trình cha chờ đến khi một tiến trình con nào đó, hoặc tất cả các tiến trình con kết thúc xử lý. Ví dụ MSDOS

Các hệ điều hành khác nhau có thể chọn lựa các cài đặt khác nhau để thực hiện thao tác tạo lập một tiến trình.

b). Kết thúc tiến trình

Một tiến trình kết thúc khi:

- Tiến trình hoàn tất công việc.

- Tiến trình kết thúc khi vượt quá thời hạn

- Tiến trình kết thúc khi sử dụng quá tài nguyên quy định

- Tiến trình kết thúc khi bộ nhớ không đủ.

Điều phối độc quyền : Nguyên lý điều phối *độc quyền* cho phép một tiến trình khi nhận được CPU sẽ có quyền độc chiếm CPU đến khi hoàn tất xử lý hoặc tự nguyện giải phóng CPU. Khi đó quyết định điều phối CPU sẽ xảy ra trong các tình huống sau:

- Khi tiến trình chuyển từ trạng thái đang xử lý(running) sang trạng thái bị khóa blocked (ví dụ chờ một thao tác nhập xuất hay chờ một tiến trình con kết thúc…).

- Khi tiến trình kết thúc.

Các giải thuật độc quyền thường đơn giản và dễ cài đặt. Tuy nhiên chúng thường không thích hợp với các hệ thống tổng quát nhiều người dùng, vì nếu cho phép một tiến trình có quyền xử lý bao lâu tùy ý, có nghĩa là tiến trình này có thể giữ CPU một thời gian không xác định, có thể ngăn cản những tiến trình còn lại trong hệ thống có một cơ hội để xử lý.

Điều phối không độc quyền :

Bộ phận điều phối tiến trình có thể tạm dừng tiến trình đang xử lý để thu hồi processor của nó, để cấp cho tiến trình khác, sao cho phù hợp với công tác điều phối hiện tại

Các quyết định điều phối xảy ra khi :

- Khi tiến trình chuyển từ trạng thái đang xử lý (running) sang trạng thái bị khóa blocked (ví dụ chờ một thao tác nhập xuất hay chờ một tiến trình con kết thúc…).

- Khi tiến trình chuyển từ trạng thái đang xử lý (running) sang trạng thái ready (ví dụ xảy ra một ngắt).

- Khi tiến trình chuyển từ trạng thái chờ (blocked) sang trạng thái ready (ví dụ một thao tác nhập/xuất hoàn tất).

- Khi tiến trình kết thúc.

Các thuật toán điều phối theo nguyên tắc không độc quyền ngăn cản được tình trạng một tiến trình độc chiếm CPU, nhưng việc tạm dừng một tiến trình có thể dẫn đến các mâu thuẫn trong truy xuất, đòi hỏi phải sử dụng một phương pháp đồng bộ hóa thích hợp để giải quyết.

P1	P2	P3
0	‘24	27 30

thời gian chờ đợi được xử lý là 0 đối với P1, (24 -1) với P2 và (24+3-2) với P3. Thời gian chờ trung bình là (0+23+25)/3 = 16 milisecondes.

Thời gian lưu:

P1: 24, P2: 26, P3: 28

Thời gian lưu trung bình là: 26.

Ưu điểm: đơn giản, dễ cài đặt.

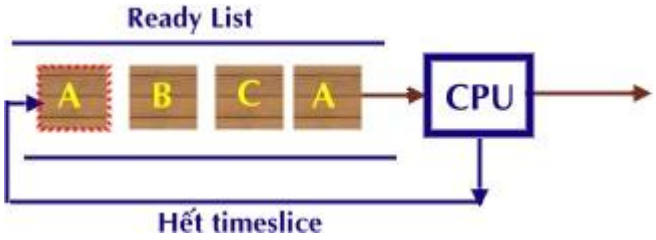
Một số hạn chế:

- Thời gian chờ đợi trung bình lớn nên không phù hợp với các hệ thống chia sẻ thời gian.
- Khả năng tương tác kém khi nó được áp dụng trên các hệ thống uniprocessor
- Nếu các tiến trình ở đầu Readylist cần nhiều thời gian của processor thì các tiến trình ở cuối readylist sẽ phải chờ đợi lâu mới được cấp processor

b). Chiến lược phân phối xoay vòng (RR: Round Robin)

Nguyên tắc : Danh sách sẵn sàng được xử lý như một danh sách vòng, bộ điều phối lần lượt cấp phát cho từng tiến trình trong danh sách một khoảng thời gian tối đa sử dụng CPU cho trước gọi là *quantum*. Tiến trình ở đầu readylist thì được cấp phát CPU trước. Đây là một giải thuật điều phối không độc quyền : khi một tiến trình sử dụng CPU đến hết thời gian quantum dành cho nó, hệ điều hành thu hồi CPU và cấp cho tiến trình kế tiếp trong danh sách. Nếu tiến trình bị khóa hay kết thúc trước khi sử dụng hết thời gian quantum, hệ điều hành cũng lập tức cấp phát CPU cho tiến trình khác. Khi tiến trình tiêu thụ hết thời gian CPU dành cho nó mà chưa hoàn tất, tiến trình được đưa trở lại vào cuối danh sách sẵn sàng để đợi được cấp CPU trong lượt kế tiếp.

Ví dụ :



Hình 2.10 Điều phối Round Robin

Tiến trình	Thời điểm vào RL	Thời gian xử lý
P1	0	24
P2	1	3
P3	2	3

Nếu sử dụng quantum là 4 milisecondes, thứ tự cấp phát CPU sẽ là

P1	P2	P3	P1	P1	P1	P1	P1
0	‘4	7	10	14	18	22	26 30

Thời gian chờ:

P1: 6, P2:3, P3:5

Thời gian chờ đợi trung bình sẽ là (6+3+5)/3 = 4.66 milisecondes.

Đảm bảo tính công bằng.

Nếu có *n* tiến trình trong danh sách sẵn sàng và sử dụng quantum *q*, thì mỗi tiến trình sẽ được cấp phát CPU *1/n* trong từng khoảng thời gian *q*. Mỗi tiến trình sẽ không phải đợi quá *(n-1)q* đơn vị thời gian trước khi nhận được CPU cho lượt kế tiếp.

Hiệu quả điều phối phụ thuộc vào độ dài quantum

Vấn đề đáng quan tâm đối với giải thuật RR là độ dài của quantum. Nếu thời lượng quantum quá bé sẽ phát sinh quá nhiều sự chuyển đổi giữa các tiến trình và khiến cho việc sử dụng CPU kém hiệu qủa. Nhưng nếu sử dụng quantum quá lớn sẽ làm tăng thời gian hồi đáp và giảm khả năng tương tác của hệ thống.

c) Điều phối với độ ưu tiên

Mỗi tiến trình được gán cho một độ ưu tiên nhất định

Độ ưu tiên của tiến trình có thể được phát sinh tự động bởi hệ thống hoặc được gán tường minh trong chương trình của người sử dụng. Độ ưu tiên của tiến trình có 2 loại: thứ nhất là độ ưu tiên tĩnh là độ ưu tiên gán trước cho tiến trình và không thay đổi trong suốt thời gian sống của tiến trình. Thứ hai độ ưu tiện động là độ ưu tiên được gán cho tiến trình trong quá tình hoạt động của nó, hệ điều hành sẽ gán lại độ ưu tiên cho tiến trình khi môi trường xử lý tiến trình thay đổi.

Tại thời điểm điều phối, tiến trình có độ ưu tiên cao nhất sẽ được chọn để cấp phát CPU. Các tiến trình có độ ưu tiên cao nhất bằng nhau thì tiến trình nào đến trước thì sẽ được cấp trước. Giải thuật điều phối với độ ưu tiên có thể theo nguyên tắc độc quyền hay không độc quyền. Khi một tiến trình được đưa vào danh sách các tiến trình sẵn sàng, độ ưu tiên của nó được so sánh với độ ưu tiên của tiến trình hiện hành đang xử lý. Giải thuật điều phối với độ ưu tiên và không độc quyền sẽ thu hồi CPU từ tiến trình hiện hành để cấp phát cho tiến trình mới nếu độ ưu tiên của tiến trình này cao hơn tiến trình hiện hành. Một giải thuật độc quyền sẽ chỉ đơn giản chèn tiến trình mới vào danh sách sẵn sàng tại vị trí thích hợp, và tiến trình hiện hành vẫn tiếp tục xử lý hết thời gian dành cho nó.

Ready list luôn được xếp theo thứ tự giảm dần của độ ưu tiên kể từ đầu danh sách. Điều này có nghĩa là tiến trình được chọn để cấp processor là tiến trình ở đầu readylist.

Ví dụ : (độ ưu tiên 1 > độ ưu tiên 2> độ ưu tiên 3)

Tiến trình	Thời điểm vào RL	Độ ưu tiên	Thời gian xử lý
P1	0	3	24
P2	1	1	3
P3	2	2	3

Sử dụng thuật giải độc quyền, thứ tự cấp phát CPU như sau :

P1	P2	P3
0	‘24	27 30

Thời gian chờ trung bình là 16

Sử dụng thuật giải không độc quyền, thứ tự cấp phát CPU như sau :

P1	P2	P3	P1
0	‘1	4	7 30

Thời gian chờ:

P1: 6, P2:0, P3:2

Thời gian chờ trung bình là 8/3

Thảo luận : Tình trạng 'đói CPU' (starvation) là một vấn đề chính yếu của các giải thuật sử dụng độ ưu tiên. Các giải thuật này có thể để các tiến trình có độ ưu tiên thấp chờ đợi CPU vô hạn ! Để ngăn cản các tiến trình có độ ưu tiên cao chiếm dụng CPU vô thời hạn, bộ điều phối sẽ giảm dần độ ưu tiên của các tiến trình này sau mỗi ngắt đồng hồ. Nếu độ ưu tiên của tiến trình này giảm xuống thấp hơn tiến trình có độ ưu tiên cao thứ nhì, sẽ xảy ra sự chuyển đổi quyền sử dụng CPU.Quá trình này gọi là sự 'lão hóa' *(aging)* tiến trình.

d). Chiến lược công việc ngắn nhất (Shortest-job-first SJF)

Nguyên tắc : Đây là một trường hợp đặc biệt của giải thuật điều phối với độ ưu tiên. Trong giải thuật này, độ ưu tiên *p* được gán cho mỗi tiến trình là nghịch đảo của thời gian xử lý *t* mà tiến trình yêu cầu : $p = 1/t$. Khi CPU được tự do, nó sẽ được cấp phát cho tiến trình yêu cầu ít thời gian nhất để kết thúc- tiến trình ngắn nhất. Giải thuật này cũng có thể độc quyền hay không độc quyền. Sự chọn lựa xảy ra khi có một tiến trình mới được đưa vào danh sách sẵn sàng trong khi một tiến trình khác đang xử lý. Tiến trình mới có thể sỡ hữu một yêu cầu thời gian sử dụng CPU cho lần tiếp theo (CPU-burst) ngắn hơn thời gian còn lại mà tiến trình hiện hành cần xử lý. Giải thuật SJF không độc quyền sẽ dừng hoạt động của tiến trình hiện hành, trong khi giải thuật độc quyền sẽ cho phép tiến trình hiện hành tiếp tục xử lý. Nếu hai tiến trình có cùng thời gian sử dụng CPU, tiến trình đến trước sẽ đươc yêu cầu CPU trước.

Ví dụ :

Tiến trình	Thời điểm vào RL	Thời gian xử lý
P1	0	6
P2	1	8
P3	2	4
P4	3	2

Sử dụng thuật giải SJF độc quyền, thứ tự cấp phát CPU như sau:

P1	P4	P3	P2
0	6	8	12 20

Thời gian chờ:

P1:0, P2: 11, P3: 6, P4:3

Thời gian chờ trung bình là 5.

Sử dụng thuật giải SJF không độc quyền, thứ tự cấp phát CPU như sau:

P1	P4	P1	P3	P2
0	3	5	8	12 20

Thời gian chờ:

P1:2, P2:11, P3: 6, P4:0

Thời gian chờ trung bình là 19/4

Thảo luận : Giải thuật này cho phép đạt được thời gian chờ trung bình cực tiểu. Khó khăn thực sự của giải thuật SJF là không thể biết được thời gian yêu cầu chu kỳ CPU tiếp theo? Chỉ có thể dự đoán giá trị này theo cách tiếp cận sau : gọi *tn* là độ dài của thời gian xử lý lần thứ *n*, t_{n+1} là giá trị dự đoán cho lần xử lý tiếp theo. Với hy vọng giá trị dự đoán sẽ gần giống với các giá trị trước đó, có thể sử dụng công thức:

$$t_{n+1} = a\,tn + (1-a)t_n$$

Trong công thức này,tn chứa đựng thông tin gần nhất ; t_n chứa đựng các thông tin quá khứ được tích lũy. Tham số a (0 <= a <= 1) kiểm soát trọng số của hiện tại gần hay quá khứ ảnh hưởng đến công thức dự đoán.

e) Chiến lược điều phối với nhiều mức độ ưu tiên

Nguyên tắc : Ý tưởng chính của giải thuật là phân lớp các tiến trình tùy theo độ ưu tiên của chúng để có cách thức điều phối thích hợp cho từng nhóm. Danh sách sẵn sàng được phân tách thành các danh sách riêng biệt theo cấp độ ưu tiên, mỗi danh sách bao gồm các tiến trình có cùng độ ưu tiên và được áp dụng một giải thuật điều phối thích hợp để điều phối. Ngoài ra, còn có một giải thuật điều phối giữa các nhóm, thường giải thuật này là giải thuật không độc quyền và sử dụng độ ưu tiên cố định.Một tiến trình thuộc về danh sách ở cấp ưu tiên *i* sẽ chỉ được cấp phát CPU khi các danh sách ở cấp ưu tiên lớn hơn *i* đã trống.

Hình 2.11 Điều phối nhiều cấp ưu tiên

Thông thường, một tiến trình sẽ được gán vĩnh viễn với một danh sách ở cấp ưu tiên *i* khi nó được đưa vào hệ thống. Các tiến trình không di chuyển giữa các danh sách. Cách tổ chức này sẽ làm giảm chi phí điều phối, nhưng lại thiếu linh động và có thể dẫn đến tình trạng 'đói CPU' cho các tiến trình thuộc về những danh sách có độ ưu tiên thấp. Do vậy có thể xây dựng giải thuật điều phối nhiều cấp ưu tiên và xoay vòng. Giải thuật này sẽ chuyển dần một tiến trình từ danh sách có độ ưu tiên cao xuống danh sách có độ ưu tiên thấp hơn sau mỗi lần sử dụng CPU. Cũng vậy, một tiến trình chờ quá lâu trong các danh sách có độ ưu tiên thấp cũng có thể được chuyển dần lên các danh sách có độ ưu tiên cao hơn. Khi xây dựng một giải thuật điều phối nhiều cấp ưu tiên và xoay vòng cần quyếtđịnh các tham số :

Số lượng các cấp ưu tiên

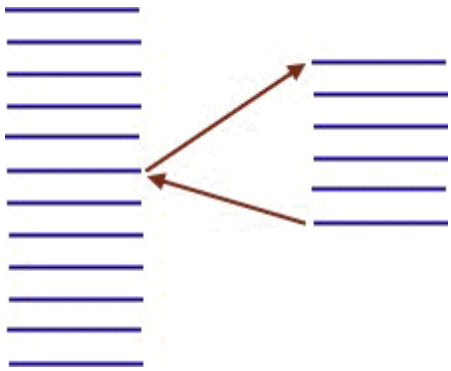
Giải thuật điều phối cho từng danh sách ứng với một cấp ưu tiên.

Phương pháp xác định thời điểm di chuyển một tiến trình lên danh sách có độ ưu tiên cao hơn.

Phương pháp xác định thời điểm di chuyển một tiến trình lên danh sách có độ ưu tiên thấp hơn.

Người dùng (ví dụ nhấn phím Ctl-C để ngắt xử lý của tiến trình)

Khi một tiến trình nhận một tín hiệu, nó có thể xử sự theo một trong các cách sau :



Hình2.13 Liên lạc bằng tín hiệu

Bỏ qua tín hiệu

Xử lý tín hiệu theo kiểu mặc định

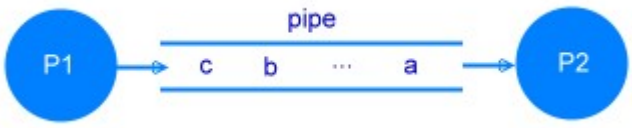
Tiếp nhận tín hiệu và xử lý theo cách đặc biệt của tiến trình.

Liên lạc bằng tín hiệu mang tính chất *không đồng bộ*, nghĩa là một tiến trình nhận tín hiệu không thể xác định trước thời điểm nhận tính hiệu. Hơn nữa các tiến trình không thể kiểm tra được sự kiện tương ứng với tín hiệu có thật sự xảy ra ? Cuối cùng, các tiến trình chỉ có thể thông báo cho nhau về một biến cố nào đó, mà không trao đổi dữ liệu theo cơ chế này được.

b) Pipe

Một pipe là một kênh liên lạc trực tiếp giữa hai tiến trình : dữ liệu xuất của tiến trình này được chuyển đến làm dữ liệu nhập cho tiến trình kia dưới dạng một dòng các byte.

Khi một pipe được thiết lập giữa hai tiến trình, một trong chúng sẽ ghi dữ liệu vào pipe và tiến trình kia sẽ đọc dữ liệu từ pipe. Thứ tự dữ liệu truyền qua pipe được bảo toàn theo nguyên tắc FIFO. Một pipe có kích thước giới hạn (thường là 4096 ký



tự)

Hình 2.14 Liên lạc qua pipe

Một tiến trình chỉ có thể sử dụng một pipe do nó tạo ra hay kế thừa từ tiến trình cha. Hệ điều hành cung cấp các lời gọi hệ thống read/write cho các tiến trình thực hiện thao tác đọc/ghi dữ liệu trong pipe. Hệ điều hành cũng chịu trách nhiệm đồng bộ hóa việc truy xuất pipe trong các tình huống:

Tiến trình đọc pipe sẽ bị khóa nếu pipe trống, nó sẽ phải đợi đến khi pipe có dữ liệu để truy xuất.

Tiến trình ghi pipe sẽ bị khóa nếu pipe đầy, nó sẽ phải đợi đến khi pipe có chỗ trống để chứa dữ liệu.

Liên lạc bằng pipe là một cơ chế liên lạc *một chiều (unidirectional)*, nghĩa là một tiến trình kết nối với một pipe chỉ có thể thực hiện một trong hai thao tác đọc hoặc ghi, nhưng không thể thực hiện cả hai. Một số hệ điều hành cho phép thiết lập hai pipe giữa một cặp tiến trình để tạo liên lạc hai chiều. Trong những hệ thống đó, có nguy cơ xảy ra tình trạng *tắc nghẽn* (deadlock) : một pipe bị giới hạn về kích thước, do vậy nếu cả hai pipe nối kết hai tiến trình đều đầy(hoặc đều trống) và cả hai tiến trình đều muốn ghi (hay đọc) dữ liệu vào pipe(mỗi tiến trình ghi dữ liệu vào một pipe), chúng sẽ cùng bị khóa và chờ lẫn nhau mãi mãi !

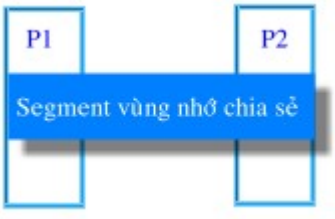
Cơ chế này cho phép truyền dữ liệu với cách thức không cấu trúc.

Ngoài ra, một giới hạn của hình thức liên lạc này là chỉ cho phép kết nối hai tiến trình có quan hệ cha-con, và trên cùng một máy tính.

c) Vùng nhớ chia sẻ

Cách tiếp cận của cơ chế này là cho nhiều tiến trình cùng truy xuất đến một vùng nhớ chung gọi là *vùng nhớ chia sẻ (shared memory)*. Không có bất kỳ hành vi truyền dữ liệu nào cần phải thực hiện ở đây, dữ liệu chỉ đơn giản được đặt vào một vùng nhớ mà nhiều tiến trình có thể cùng truy cập được.

Với phương thức này, các tiến trình chia sẻ một vùng nhớ vật lý thông qua trung gian không gian địa chỉ của chúng. Một vùng nhớ chia sẻ tồn tại độc lập với các tiến trình, và khi một tiến trình muốn truy xuất đến vùng nhớ này, tiến trình phải kết



gắn vùng nhớ chung đó vào không gian địa chỉ riêng của từng tiến trình, và thao tác trên đó như một vùng nhớ riêng của mình.

Hình 2.15 Liên lạc qua vùng nhớ chia sẻ

Đây là phương pháp nhanh nhất để trao đổi dữ liệu giữa các tiến trình. Nhưng phương thức này cũng làm phát sinh các khó khăn trong việc bảo đảm sự toàn vẹn dữ liệu (*coherence*) , ví dụ : làm sao biết được dữ liệu mà một tiến trình truy xuất là dữ liệu mới nhất mà tiến trình khác đã ghi ? Làm thế nào ngăn cản hai tiến trình cùng đồng thờighi dữ liệu vào vùng nhớ chung ?…Rõ ràng vùng nhớ chia sẻ cần được bảo vệ bằng những cơ chế đồng bộ hóa thích hợp..

Một khuyết điểm của phương pháp liên lạc này là không thể áp dụng hiệu quả trong các hệ phân tán , để trao đổi thông tin giữa các máy tính khác nhau.

d) Trao đổi thông điệp (Message)

Hệ điều hành còn cung cấp một cơ chế liên lạc giữa các tiến trình không thông qua việc chia sẻ một tài nguyên chung , mà thông qua việc gởi thông điệp. Để hỗ trợ cơ chế liên lạc bằng thông điệp, hệ điều hành cung cấp các hàm IPC chuẩn (Interprocess communication), cơ bản là hai hàm:

Send(message) : gởi một thông điệp

Receive(message) : nhận một thông điệp

Nếu hai tiến trình P và Q muốn liên lạc với nhau, cần phải thiết lập một mối liên kết giữa hai tiến trình, sau đó P, Q sử dụng các hàm IPC thích hợp để trao đổi thông điệp, cuối cùng khi sự liên lạc chấm dứt mối liên kết giữa hai tiến trình sẽ bị hủy. Có nhiều cách thức để thực hiện sự liên kết giữa hai tiến trình và cài đặt các theo tác send /receive tương ứng : liên lạc trực tiếp hay gián tiếp, liên lạc đồng bộ hoặc không đồng bộ , kích thước thông điệp là cố định hay không … Nếu các tiến trình liên lạc theo kiểu liên kết tường minh, các hàm Send và Receive sẽ được cài đặt với tham số :

Send(destination, message) : gởi một thông điệp đến địa chỉ đích

Receive(source,message) : nhận một thông điệp từ địa chỉ nguồn

Đơn vị truyền thông tin trong cơ chế trao đổi thông điệp là một thông điệp, do đó các tiến trình có thể trao đổi dữ liệu ở dạng có cấu trúc.

Định dạng thông điệp

Loại thông điệp
Địa chỉ đích
Địa chỉ nguồn
Độ dài thông điệp
Các thông tin điều kiển
Nội dung thông điệp

e) Sockets

Giới thiệu: Một socket là một thiết bị truyền thông hai chiều tương tự như tập tin, chúng ta có thể đọc hay ghi lên nó, tuy nhiên mỗi socket là một thành phần trong một mối nối nào đó giữa các máy trên mạng máy tính và các thao tác đọc/ghi chính là sự trao đổi dữ liệu giữa các ứng dụng trên nhiều máy khác nhau.

Sử dụng socket có thể mô phỏng hai phương thức liên lạc trong thực tế : liên lạc thư tín (socket đóng vai trò bưu cục) và liên lạc điện thoại (socket đóng vai trò tổng đài)

Các thuộc tính của socket:

Domaine: định nghĩa dạng thức địa chỉ và các nghi thức sử dụng. Có nhiều domaines, ví dụ UNIX, INTERNET, XEROX_NS, ...

Type: định nghĩa các đặc điểm liên lạc:

Sự tin cậy

Sự bảo toàn thứ tự dữ liệu

Lặp lại dữ liệu

Chế độ nối kết

Bảo toàn giới hạn thông điệp

Khả năng gởi thông điệp khẩn

Để thực hiện liên lạc bằng socket, cần tiến hành các thao tác :

Tạo lập hay mở một socket

Gắn kết một socket với một địa chỉ

Liên lạc : có hai kiểu liên lạc tùy thuộc vào chế độ nối kết:

Liên lạc trong chế độ không liên kết **:** liên lạc theo hình thức hộp thư:

hai tiến trình liên lạc với nhau không kết nối trực tiếp

mỗi thông điệp phải kèm theo địa chỉ người nhận.

Hình thức liên lạc này có đặc điểm được :

người gởi không chắc chắn thông điệp của học được gởi đến người nhận,

một thông điệp có thể được gởi nhiều lần,

hai thông điệp đượ gởi theo một thứ tự nào đó có thể đến tay người nhận theo một thứ tự khác.

Một tiến trình sau khi đã mở một socket có thể sử dụng nó để liên lạc với nhiều tiến trình khác nhau nhờ sử hai primitive *send* và *receive*.

Liên lạc trong chế độ nối kết**:**

Một liên kết được thành lập giữa hai tiến trình. Trước khi mối liên kết này được thiết lập, một trong hai tiến trình phải đợi có một tiến trình khác yêu cầu kết nối.Có thể sử dụng socket để liên lạc theo mô hình client-server. Trong mô hình này, server sử

Tài nguyên găng

Những tài nguyên được hệ điều hành chia sẻ cho nhiều tiến trình hoạt động đồng thời dùng chung, mà có nguy cơ dẫn đến sự chanh chấp giữa các tiến trình khi sử dụng chúng, được gọi là tài nguyên găng. Tài nguyên găng có thể là tài nguyên phần cứng hoặc phần mềm, có thể là tài nguyên phân chia được hoặc không phân chia được, nhưng đa số thường là tài nguyên phân chia được như là: các biến chung, các file chia sẻ.

Ví dụ sau đây cho thấy hậu quả của việc sử dụng tài nguyên găng trong các chương trình có các tiến trình hoạt động đồng thời:

Giả sử có một ứng dụng giao dịch ngân hàng, hoạt động trong môi trường đa nhiệm, đa người dùng. Mỗi người sử dụng trong môi trường này khi cần thực hiện thao tác rút tiền từ trong tài khoản chung thì phải khởi tạo một tiến trình rút tiền, tiến trình rút tiền được chỉ thực hiện được khi số tiền rút nhỏ hơn số tiền còn lại trong tài khoản chung. Trong môi trường này có thể có nhiều người sử dụng đồng thời thực hiện thao tác rút tiền chung từ hệ thống.

Giả sử có hai tiến trình rút tiền P_1 và P_2, có thể hoạt động đồng thời và cùng chia sẻ một vùng nhớ chung lưu trữ biến *taikhoan* phản ánh thông tin về tài khoản. Mỗi tiến trình muốn rút một khoản tiền *tienrut* từ tài khoản:

```
if (taikhoan - tienrut >=0)

taikhoan = taikhoan - tienrut;

else

error(« khong the rut tien ! »);
```

Nếu tại một thời điểm nào đó:

Giả sử trong tài khoản hiện còn 800, P_1 muốn rút 500 và P_2 muốn rút 400.

Tiến trình P1 và P2 đồng thời rút tiền, thì theo nguyên tắc điều trên không xảy ra vì 500+400>800. Nhưng trong môi trường đa nhiệm, đa người sử dụng nếu hệ điều hành không giám sát tốt việc sử dụng tài nguyên dùng chung của các tiến trình hoạt động đồng thời thì điều trên vẫn có thể xảy ra. Tức là, cả 2 tiến trình P1 và P2 đều thành công trong thao tác rút tiền, mà ứng dụng cũng như hệ điều hành không hề phát hiện. Bởi vì, quá trình rút tiền của các tiến trình P1 và P2 có thể diễn ra như sau:

Sau khi đã kiểm tra điều kiện (taikhoan - tienrut >=0) và nhận kết quả là 300, P_1 hết thời gian xử lý mà hệ thống cho phép, hệ điều hành cấp phát CPU cho P_2.

P_2 kiểm tra cùng điều kiện trên, nhận được kết quả là 400 (do P_1 vẫn chưa rút tiền) và rút 400. Giá trị của *taikhoan* được cập nhật lại là 400.

Khi P_1 được tái kích hoạt và tiếp tục xử lý, nó sẽ không kiểm tra lại điều kiện (taikhoan - tienrut >=0)-vì đã kiểm tra trong lượt xử lý trước- mà thực hiện rút tiền. Giá trị của *taikhoan* sẽ lại được cập nhật thành -100. Tình huống lỗi xảy ra !

Nguyên nhân của lỗi này không phải là do hai tiến trình P1 và P2 đồng thời truy xuất biến taifkhoan, mà do hai thao tác: kiểm tra tài khoản và thực hiện rút tiền của các tiến trình này bị tách rời nhau. Nếu hệ điều hành này làm cho hai thao tác này không tách rời nhau thì lỗi sẽ không xảy ra.

Trong trường hợp này tài nguyên găng chính là biến tài khoản

Các tình huống tương tự như thế - có thể xảy ra khi có nhiều hơn hai tiến trình đọc và ghi dữ liệu trên cùng một vùng nhớ chung, và kết quả phụ thuộc vào sự điều phối tiến trình của hệ thống- được gọi là các tình huống tranh đoạt điều khiển *(race condition)* .

Nguyên nhân tiềm ẩn của sự xung đột giữa các tiến trình hoạt động đồng thời khi sử dụng tài nguyên găng là: các tiến trình này hoạt động đồng thời với nhau một cách hoàn toàn độc lập và không trao đổi thông tin với nhau nhưng sự thực thi của các tiến trình này lại ảnh hưởng đến nhau.

b) Miền găng (critical section)

Để ngăn chặn các tình huống lỗi có thể nảy sinh khi các tiến trình truy xuất đồng thời một tài nguyên không thể chia sẻ, cần phải áp đặt một sự truy xuất độc quyền trên tài nguyên đó : khi một tiến trình đang sử dụng tài nguyên, thì những tiến trình khác không được truy xuất đến tài nguyên.

Đoạn chương trình trong đó có khả năng xảy ra các mâu thuẫn truy xuất trên tài nguyên chung được gọi là *miền găng (critical section)*. Trong ví dụ trên, đoạn mã :

```
if (taikhoan - tienrut >=0)

taikhoan = taikhoan - tienrut;
```

}

Hình 3.5 Cấu trúc một chương trình sử dụng biến khóa để đồng bộ

Giải pháp đơn giản, dễ xây dựng, nhưng vẫn xuất hiện hiện tượng chờ đợi tích cực, gây lãng phí thời gian của processor.

Nếu một tiến trình trong đoạn găng không thể ra khỏi đoạn găng, thì các tiến trình chờ ngoài đoạn găng có thể chờ đợi vô hạn (vì lock không được đặt lại bằng 0)

Giải pháp này có thể vi phạm điều kiện thứ nhất: hai tiến trình có thể cùng ở trong miền găng tại một thời điểm. Giả sử một tiến trình nhận thấy lock = 0 và chuẩn bị vào miền găng, nhưng trước khi nó có thể đặt lại giá trị cho lock là 1, nó bị tạm dừng để một tiến trình khác hoạt động. Tiến trình thứ hai này thấy lock vẫn là 0 thì vào miền găng và đặt lại lock = 1. Sau đó tiến trình thứ nhất được tái kích hoạt, nó gán lock = 1 lần nữa rồi vào miền găng. Như vậy tại thời điểm đó cả hai tiến trình đều ở trong miền găng.

b) Sử dụng việc kiểm tra luân phiên :

Tiếp cận : Đây là một giải pháp đề nghị cho hai tiến trình. Hai tiến trình này sử dụng chung biến *turn* (phản ánh phiên tiến trình nào được vào miền găng), được khởi động với giá trị 0. Nếu *turn = 0*, tiến trình A được vào miền găng. Nếu *turn = 1*, tiến trình A đi vào một vòng lặp chờ đến khi *turn* nhận giá trị *0*. Khi tiến trình A rời khỏi miền găng, nó đặt giá trị *turn* về 1 để cho phép tiến trình B đi vào miền găng.

while (TRUE) {

while (turn != 0); // wait
critical-section ();
turn = 1**;**
Noncritical-section ();

}

(a) Cấu trúc tiến trình A

while (TRUE) {

while (turn != 1); // wait
critical-section ();

```
turn = 0;
Noncritical-section ();

                    }
```

(b) Cấu trúc tiến trình B

Hình 3.6 Cấu trúc các tiến trình trong giải pháp kiểm tra luân phiên

Giải pháp này dựa trên việc thực hiện sự kiểm tra nghiêm ngặt đến lượt tiến trình nào được vào miền găng. Do đó nó có thể ngăn chặn được tình trạng hai tiến trình cùng vào miền găng, nhưng lại có thể vi phạm điều kiện thứ ba: một tiến trình có thể bị ngăn chặn vào miền găng bởi một tiến trình khác không ở trong miền găng. Giả sử tiến trình B ra khỏi miền găng rất nhanh chóng. Cả hai tiến trình đều ở ngoài miền găng, và turn = 0. Tiến trình A vào miền găng và ra khỏi nhanh chóng, đặt lại giá trị của turn là1, rồi lại xử lý đoạn lệnh ngoài miền găng lần nữa. Sau đó, tiến trình A lại kết thúc nhanh chóng đoạn lệnh ngoài miền găng của nó và muốn vào miền găng một lần nữa. Tuy nhiên lúc này B vẫn còn mãi xử lý đoạn lệnh ngoài miền găng của mình, và turn lại mang giá trị 1 ! Như vậy, giải pháp này không có giá trị khi có sự khác biệt lớn về tốc độ thực hiện của hai tiến trình, nó vi phạm cả điều kiện thứ hai.

c) Giải pháp của Peterson

Tiếp cận : Petson đưa ra một giải pháp kết hợp ý tưởng của cả hai giải pháp kể trên. Các tiến trình chia sẻ hai biến chung :

```
int turn; // đến phiên ai

int interesse[2]; // khởi động là FALSE
```

Nếu *interesse[i]* = TRUE có nghĩa là tiến trình Pi muốn vào miền găng. Khởi đầu, *interesse[0]=interesse[1]=FALSE* và giá trị của est được khởi động là 0 hay 1. Để có thể vào được miền găng, trước tiên tiến trình Pi đặt giá trị *interesse[i]*=TRUE (xác định rằng tiến trình muốn vào miền găng), sau đó đặt turn=*j* (đề nghị thử tiến trình khác vào miền găng). Nếu tiến trình Pj không quan tâm đến việc vào miền găng (*interesse[j]=FALSE*), thì Pi có thể vào miền găng, nếu không, Pi phải chờ đến khi *interesse[j]=FALSE*. Khi tiến trình Pi rời khỏi miền găng, nó đặt lại giá trị cho *interesse[i]*= FALSE.

```
while (TRUE) {
```

```
        int j = 1-i; // j là tiến trình còn lại
interesse[i]= TRUE;
turn = j;
while (turn == j && interesse[j]==TRUE);
critical-section ();
interesse[i] = FALSE;
Noncritical-section ();

                    }
```

Hình 3.7 Cấu trúc tiến trình Pi trong giải pháp Peterson

giải pháp này ngăn chặn được tình trạng mâu thuẫn truy xuất : mỗi tiến trình Pi chỉ có thể vào miền găng khi *interesse[j]=FALSE* hoặc *turn = i*. Nếu cả hai tiến trình đều muốn vào miền găng thì *interesse[i] = interesse[j] =TRUE* nhưng giá trị của *turn* chỉ có thể hoặc là 0 hoặc là 1, do vậy chỉ có một tiến trình được vào miền găng.

2.4.3.1.2. Các giải pháp phần cứng

a) Cấm ngắt:

Một số vi xử lý cung cấp cấp chỉ thị CLI và STI để người lập trình thực hiện thao tác mở ngắt (STI: Setting Interrupt) và cấm ngắt (CLI: Clean Interrupt) của hệ thống trong lập trình. Người lập trình có thể dùng cặp chỉ thị này để đồng bộ hóa các tiến trình như sau: Trước khi vào miền găng tiến trình thực hiện chỉ thị CLI, để yêu cầu cấm các ngắt trong hệ thống, khi đó ngắt đồng hồ không thể phát sinh, nghĩa là không có một tiến trình nào khác có thể phát sinh, nhờ đó mà tiến trình trong đoạn găng toàn quyền sử dụng tài nguyên găng cho đến hết thời gian xử lý của nó. Khi kết thúc truy xuất tài nguyên găng, tiến trình ra khỏi miền găng, tiến trình thực hiện chỉ thị STI để cho phép ngắt trở lại. Khi đó các tiến trình khác có thể tiếp tục hoạt động và có thể vào miền găng.

Giải pháp đơn giản, dễ cài đặt. Tuy nhiên cần phải có sự hỗ trợ của vi xử lý và dễ gây ra hiện tượng treo toàn bộ hệ thống, khi tiến trình đang ở trong miền găng không có khả năng ra khỏi miền găng. Tiến trình không ra khỏi miền găng nên nó không thể thực hiện chỉ thị STI để mở ngắt cho hệ thống, nên hệ thống bị treo hoàn toàn.

Giải pháp này không thể sử dụng trên các hệ thống đa bộ xử lý, vì CLI chỉ cấm ngắt trên vi xử lý hiện tại chứ không thể cấm ngắt của các vi xử lý khác. Tức sau khi cấm ngắt, tiến trình trong miền găng vẫn có thể bị tranh chấp tài nguyên găng bởi các tiến trình trên các vi xử lý khác trong hệ thống.

b) Chỉ thị TSL (Test-and-Set):

Tiếp cận: đây là một giải pháp đòi hỏi sự trợ giúp của cơ chế phần cứng. Nhiều máy tính cung cấp một chỉ thị đặc biệt cho phép kiểm tra và cập nhật nội dung một vùng nhớ trong một thao tác không thể phân chia, gọi là chỉ thị *Test-and-Set Lock* (TSL) và được định nghĩa như sau:

```
Test-and-Setlock(boolean target)
{

    Test-and-Setlock = target;
target = TRUE;

    }
```

Nếu có hai chỉ thị TSL xử lý đồng thời (trên hai bộ xử lý khác nhau), chúng sẽ được xử lý tuần tự . Có thể cài đặt giải pháp truy xuất độc quyền với TSL bằng cách sử dụng thêm một biến lock, được khởi gán là FALSE. Tiến trình phải kiểm tra giá trị của biến lock trước khi vào miền găng, nếu lock = FALSE, tiến trình có thể vào miền găng.

```
    while (TRUE) {

    while(Test-and-Setlock(lock));
critical-section ();
lock = FALSE;
Noncritical-section ();

    }
```

Hình 3.8 Cấu trúc một chương trình trong giải pháp TSL

Giải pháp đơn giản, dễ cài đặt nhưng cần phải có sự hỗ trợ của vi xử lý. Ngoài ra nó còn một hạn chế lớn là gây lãng phí thời gian xử lý của processor do tồn tại hiện tượng chờ tích cực trong vòng lặp. Hiện tượng chờ đợi tích cực là hiện tượng processor chỉ chờ một sự kiện nào đó xảy ra mà không làm gì cả.

Tất cả các giải pháp trên đây đều phải thực hiện một vòng lặp để kiểm tra liệu nó có được phép vào miền găng, nếu điều kiện chưa cho phép, tiến trình phải chờ tiếp tục trong vòng lặp kiểm tra này. Các giải pháp buộc tiến trình phải liên tục kiểm tra điều kiện để phát hiện thời điểm thích hợp được vào miền găng như thế được gọi các giải pháp « *busy waiting* ». Lưu ý rằng việc kiểm tra như thế tiêu thụ rất nhiều thời gian sử dụng CPU, do vậy tiến trình đang chờ vẫn chiếm dụng CPU. Xu hướng giải quyết vấn đề đồng bộ hoá là nên tránh các giải pháp « *busy waiting* ».

2.4.3.2. Các giải pháp « SLEEP and WAKEUP »

Để loại bỏ các bất tiện của giải pháp « busy waiting », chúng ta có thể tiếp cận theo hướng cho một tiến trình chưa đủ điều kiện vào miền găng chuyển sang trạng thái blocked, từ bỏ quyền sử dụng CPU. Để thực hiện điều này, cần phải sử dụng các thủ tục do hệ điều hành cung cấp để thay đổi trạng thái tiến trình. Hai thủ tục cơ bản *SLEEP* và *WAKEUP* thường được sử dụng để phục vụ mục đích này.

SLEEP là một lời gọi hệ thống có tác dụng tạm dừng hoạt động của tiến trình (blocked) gọi nó và chờ đến khi được một tiến trình khác « đánh thức ». Lời gọi hệ thống *WAKEUP* nhận một tham số duy nhất : tiến trình sẽ được tái kích hoạt (đặt về trạng thái ready).

Ý tưởng sử dụng SLEEP và WAKEUP như sau : khi một tiến trình chưa đủ điều kiện vào miền găng, nó gọi *SLEEP* để tự khóa đến khi có một tiến trình khác gọi *WAKEUP* để giải phóng cho nó. Một tiến trình gọi *WAKEUP* khi ra khỏi miền găng để đánh thức một tiến trình đang chờ, tạo cơ hội cho tiến trình này vào miền găng :

Cấu trúc chương trình trong giải pháp SLEEP and WAKEUP

```
int busy;   // 1 nếu miền găng đang bị chiếm, nếu không là 0
int blocked;   // đếm số lượng tiến trình đang bị khóa
while (TRUE) {
if (busy){
blocked = blocked + 1;
sleep();
}
else busy = 1;
```

critical-section ();

busy = 0**;**
if(blocked){
wakeup(process);
blocked = blocked - 1;
}

Noncritical-section ();

}

Khi sử dụng SLEEP và WAKEUP cần hết sức cẩn thận, nếu không muốn xảy ra tình trạng mâu thuẫn truy xuất trong một vài tình huống đặc biệt như sau : giả sử tiến trình A vào miền găng, và trước khi nó rời khỏi miền găng thì tiến trình B được kích hoạt. Tiến trình B thử vào miền găng nhưng nó nhận thấy A đang ở trong đó, do vậy B tăng giá trị biến *blocked* và chuẩn bị gọi *SLEEP* để tự khoá. Tuy nhiên trước khi B có thể thực hiện *SLEEP*, tiến trình A lại được tái kích hoạt và ra khỏi miền găng. Khi ra khỏi miền găng A nhận thấy có một tiến trình đang chờ (*blocked*=1) nên gọi *WAKEUP* và giảm giá trị của *blocked*. Khi đó tín hiệu *WAKEUP* sẽ lạc mất do tiến trình B chưa thật sự « ngủ » để nhận tín hiệu đánh thức !Khi tiến trình B được tiếp tục xử lý, nó mới goi *SLEEP* và tự khó vĩnh viễn !

Vấn đề ghi nhận được là tình trạng lỗi này xảy ra do việc kiểm tra tư cách vào miền găng và việc gọi SLEEP hay WAKEUP là những hành động tách biệ, có thể bị ngắt nửa chừng trong quá trình xử lý, do đó có khi tín hiệu WAKEUP gởi đến một tiến trình chưa bị khóa sẽ lạc mất.

Để tránh những tình huống tương tự, hệ điều hành cung cấp những cơ chế đồng bộ hóa dựa trên ý tưởng của chiến lược « SLEEP and WAKEUP » nhưng được xây dựng bao hàm cả phương tiện kiểm tra điều kiện vào miền găng giúp sử dụng an toàn.

a) Giải pháp dùng Semaphore (đèn báo)

Giải pháp này được **Dijkstra** đề xuất vào 1965. Semaphore được định nghĩa để sử dụng đồng bộ hóa như sau:

- Semaphore ***s*** là một *biến* nguyên khởi gán bằng một giá trị không âm e(s), đó là khả năng phục vụ của tài nguyên găng tương ứng.

- Ứng với s có một hàng đợi *f(s)*để lưu các tiến trình đang bị blocked (chờ) trên *s*

- Chỉ có hai thao tác được tác động đến semaphore s. Thao tác Down giảm s đi 1 đơn vị, thao tác Up tăng s lên 1 đơn vị

Mỗi tiến trình trước khi vào miền găng phải gọi Down để kiểm tra và xác lập quyền vào đoạn găng. Khi tiến trình gọi Down(s) thì hệ thống sẽ thực hiện như sau: e(s)=e(s)-1, nếu e(s)>=0 thì tiến trình tiếp tục xử lý và vào miền găng. Nếu e(s)<0 thì tiến trình phải vào hàng đợi để chờ cho đến khi e(s)>=0.

Cài đặt: Gọi *p* là tiến trình thực hiện thao tác

Down(s):

e(s) = e(s) - 1;

if e(s) < 0 {

status(P)= **blocked**;

enter(P,f(s));

}

Mỗi tiến trình ngay sau khi ra khỏi miền găng phải gọi Up để kiểm tra xem có tiến trình nào đang đợi trong hàng đợi hay không, nếu có thì đưa tiến trình trong hàng đợi vào miền găng. Khi tiến trình gọi Up thì hệ thống sẽ thực hiện như sau: e(S)=e(S)+1, nếu e(S)<=0 đưa một tiến trình trong hàng đợi vào miền găng

Up(s):

e(s) = e(s) + 1;

if e(s)<= 0 {

exit(Q,f(s)); //Q là tiến trình đang chờ trên s

status (Q) = **ready**;

enter(Q,ready-list);

}

Lưu ý cài đặt này có thể đưa đến một giá trị âm cho semaphore, khi đó trị tuyệt đối của semaphore cho biết số tiến trình đang chờ trên semaphore.

Điều quan trọng là các thao tác này cần thực hiện một cách không bị phân chia, không bị ngắt giữa chừng, có nghĩa là không một tiến trình nào được phép truy xuất đến semaphore nếu tiến trình đang thao tác trên semaphore này chưa kết thúc xử lý hay chuyển sang trạng thái blocked.

Sử dụng: có thể dùng semaphore để giải quyết vấn đề truy xuất độc quyền hay tổ chức phối hợp giữa các tiến trình.

Tổ chức truy xuất độc quyền với Semaphores: khái niệm *semaphore* cho phép bảo đảm nhiều tiến trình cùng truy xuất đến miền găng mà không có sự mâu thuẫn truy xuất. *n* tiến trình cùng sử dụng một semaphore s, e(s) được khởi gán là 1(tài nguyên găng này chỉ có thể đáp ứng cho một tiến trình nên e(s) được khởi gán bằng 1). Để thực hiện đồng bộ hóa, tất cả các tiến trình cần phải áp dụng cùng cấu trúc chương trình sau đây:

```
while (TRUE) {
Down(s)
critical-section ();
Up(s)
Noncritical-section ();
}
```

Ví dụ1:

Tiến trình thực hiện	Thao tác	E(s)	Trạng thái tiến trình
		1	
1. P1	Down(s)	0	P1 hoạt động
2. P2	Down(s)	-1	P2 chờ

3. P1	Up(s)	0	P2 hoạt động
4. P1	Down(s)	-1	P1 chờ
5. P2	Up(s)	0	P1 hoạt động

Ví dụ 2: Nếu trong hệ thống có 6 tiến trình hoạt động đồng thời, cùng sử dụng tài nguyên găng, tài nguyên găng này chỉ cho phép một tiến trình truy xuất đến nó tại một thời điểm. Tức là hệ điều hành phải tổ chức truy xuất độc quyền trên tài nguyên găng này. Thứ tự yêu cầu sử dụng tài nguyên găng của các tiến trình, cùng với thời gian mà tiến trình cần processor khi nó ở trong miền găng (cần tài nguyên găng) và độ ưu tiên của các tiến trình, được mô tả như sau:

Có 6 tiến trình yêu cầu sử dụng tài nguyên găng tương ứng với s lần lượt là: A B C D E F

Độ ưu tiên của các tiến trình là(5 là độ ưu tiên cao nhất):

1 1 2 4 2 5

Thời gian các tiến trình cần sử dụng tài nguyên găng là:

4 2 2 2 1 1

T	Tiến trình	Thao tác	E(s)	CS	F(s)
0			1		
1	A	Down	0	A	
2	B	Down	-1	A	B
3	C	Down	-2	A	C,B
4	D	Down	-3	A	D,C,B
5	A	Up	-2	D	C,B
6	E	Down	-3	D	C,E,B
7	D	Up	-2	C	E,B
8	F	Down	-3	C	F,E,B
9	C	Up	-2	F	E,B
10	F	Up	-1	E	B
11	E	Up	0	B	
12	B	Up	1		

Tiến trình ra khỏi đoạn găng sẽ đánh thức tiến trình có độ ưu tiên cao nhất trong hàng đợi để đưa nó vào miền găng. Tiến trình được đưa vào hàng đợi sau nhưng có độ ưu tiên cao hơn sẽ được đưa vào miền găng trước các tiến trình được đưa vào hàng đợi trước nó.

Sử dụng semaphore để đồng bộ hóa tiến trình mang lại những thuận lợi sau:

- Mỗi tiến trình chỉ kiểm tra quyền vào miền găng một lần, khi chờ nó không làm gì cả, tiến trình ra khỏi đoạn găng phải đánh thức tiến trình đang ngủ.
- Không xuất hiện tượng chờ đợi tích cực, nên khai thác tối đa thời gian xử lý của process .
- Nhờ cơ chế hàng đợi mà hệ điều hành có thể thực hiện gán độ ưu tiên cho các tiến trình khi chúng ở trong hàng đợi.

b) Giải pháp dùng Monitors

Giải pháp này được Hoar đề xuất năm 1974 sau đó vào năm 1975 được Brinch và Hansnen đề xuất lại. Monitor là cấu trúc phần mềm đặc biệt được cung cấp bởi ngôn ngữ lập trình, nó bao gồm các thủ tục, các biến và các cấu trúc dữ liệu được định nghĩa bởi Monitor. Monitor được định nghĩa trong các ngôn ngữ lập trình như pascal+, Modula-2, Modula-3. Monitor có các tính chất sau:

1- Các biến và cấu trúc dữ liệu bên trong monitor chỉ có thể được thao tác bởi các thủ tục định nghĩa bên trong monitor đó. (*encapsulation*).

2-Một tiến trình muốn vào monitor phải gọi một thủ tục của monitor đó.

3-Tại một thời điểm, chỉ có một tiến trình duy nhất được hoạt động bên trong một monitor (*mutual exclusive*). Các tiến trình khác đã gọi monitor phải hoãn lại để chờ monitor rỗi.

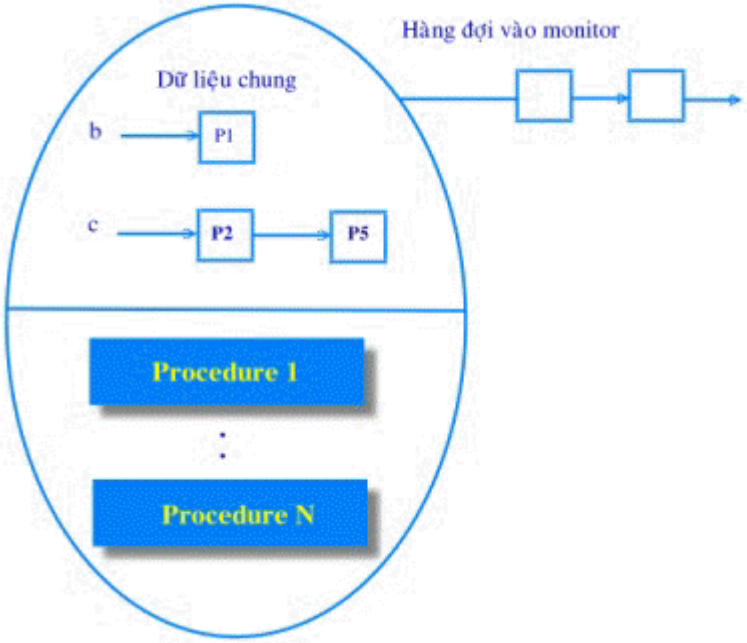
Hai tính chất 1 và 2 tương tự như các tính chất của các đối tượng trong lập trình hướng đối tượng. Như vậy một hệ điều hành hoặc một ngôn ngữ lập trình hướng đối tượng có thể cài đặt monitor như là một đối tượng có các tính chất đặc biệt. Với tính chất 3 monitor có khả năng thực hiện các cơ chế độc quyền, các biến trong monitor có thể được truy xuất chỉ bởi một tiến trình tại một thời điểm. Như vậy các cấu trúc dữ liệu dùng chung bởi các tiến trình có thể được bảo vệ bằng cách đặt chúng bên trong

monitor. Nếu dữ liệu bên trong monitor là tài nguyên găng thì monitor cung cấp sự độc quyền trong việc truy xuất đến tài nguyên găng đó.

Monitor cung cấp các công cụ đồng bộ hóa được định nghĩa như sau: Trong một monitor, có thể định nghĩa các *biến điều kiện* và hai thao tác kèm theo là **Wait** và **Signal(**chỉ có Wait và Signal được tác động đến các biến điều kiện**)** như sau : gọi ***c*** là biến điều kiện được định nghĩa trong monitor:

Wait(c): chuyển trạng thái tiến trình gọi sang blocked , và đặt tiến trình này vào hàng đợi trên biến điều kiện ***c***.

Signal(c): nếu có một tiến trình đang bị khóa trong hàng đợi của ***c,*** tái kích hoạt tiến trình đó, và tiến trình gọi signal sẽ rời khỏi monitor.



Hình 2.18 Monitor và các biến điều kiện

Trình biên dịch chịu trách nhiệm thực hiện việc truy xuất độc quyền đến dữ liệu trong monitor. Để thực hiện điều này, một semaphore nhị phân thường được sử dụng. Mỗi monitor có một hàng đợi toàn cục lưu các tiến trình đang chờ được vào monitor, ngoài ra, mỗi biến điều kiện ***c*** cũng gắn với một hàng đợi ***f(c)*** và hai thao tác trên đó được định nghĩa như sau:

Wait(c) :

status(P)= **blocked**;
enter(P,f(c));

Signal(c) :

if (f(c) != NULL){

exit(Q,f(c)); //Q là tiến trình chờ trên c
statusQ) = **ready**;
enter(Q,ready-list);

}

Sử dụng: Với mỗi nhóm tài nguyên cần chia sẻ, có thể định nghĩa một monitor trong đó đặc tả tất cả các thao tác trên tài nguyên này với một số điều kiện nào đó.:

monitor <tên monitor >

condition <danh sách các biến điều kiện>;

<déclaration de variables>;

procedure $Action_1$();

{

} ....

procedure $Action_n$();

{

}

end monitor;

Hình 3.14 Cấu trúc một monitor

Các tiến trình muốn sử dụng tài nguyên chung này chỉ có thể thao tác thông qua các thủ tục bên trong monitor được gắn kết với tài nguyên:

while (TRUE) {

Noncritical-section ();
<monitor>.Actioni; //critical-section();
Noncritical-section ();

}

Hình 3.15 Cấu trúc tiến trình Pi trong giải pháp monitor

Với monitor, việc truy xuất độc quyền được bảo đảm bởi trình biên dịch mà không do lập trình viên, do vậy nguy cơ thực hiện đồng bộ hóa sai giảm rất nhiều. Tuy nhiên giải pháp monitor đòi hỏi phải có một ngôn ngữ lập trình định nghĩa khái niệm monitor, và các ngôn ngữ như thế chưa có nhiều.

c) Giải pháp trao đổi thông điệp

Khi các tiến trình có sự tương tác với các tiến trình khác, phải yêu cầu cơ bản cần phải được thỏa mãn đó là sự đồng bộ hóa và sự truyền thông. Các tiến trình cần phải thực hiện đồng bộ để thực hiện độc quyền. Các tiến trình hợp tác có thể cần phải trao đổi thông tin. Một hướng tiếp cận để cung cấp cả hai chức năng đó là sự truyền thông điệp. Truyền thông điệp có ưu điểm là có thể thực hiện trên cả hai hệ thống uniprocessor và multiprocessor, khi các hệ thống này hoạt động trên mô hình bộ nhớ chia sẻ.

Giải pháp này dựa trên cơ sở trao đổi thông điệp với hai primitive Send và Receive để thực hiện sự đồng bộ hóa:

Send(destination, message): gởi một thông điệp đến một tiến trình đích.

Receive(source,message): nhận một thông điệp từ một tiến trình nguồn

Tiến trình gọi Receive phải chờ cho đến khi nhận được message từ tiến trình nguồn thì mới có thể tiếp tục được.

Việc sử dụng send và receive để chức đồng bộ hóa các tiến trình được thực hiện như sau:

-Có một tiến trình kiểm soát việc sử dụng tài nguyên găng

-Có nhiều tiến trình khác yêu cầu sử dụng tài nguyên găng này.

- Tiến trình có yêu cầu tài nguyên găng này sẽ gởi một thông điệp đến tiến trình kiểm soát và sau đó chuyển sang trạng thái blocked cho đến khi nhận được một thông điệp chấp nhận cho truy xuất từ tiến trình kiểm soát tài nguyên găng.

chung, các tập tin chia sẻ, hệ điều hành cần phải hỗ trợ sự độc quyền trên tài nguyên này. Với các tài nguyên không chia được thì rất khó vì bản chất tài nguyên gần như cố định. Tuy nhiên đối với một số tài nguyên về kết xuất, hệ điều hành có thể sử dụng kỹ thuật SPOOL(Smulataneous) để tạo ra tài nguyên ảo cung cấp cho các tiến trình đồng thời.

-Đối với điều kiện sự chiếm giữ và yêu cầu thêm tài nguyên: phải bảo đảm rằng mỗi khi tiến trình yêu cầu thêm một tài nguyên thì nó không chiếm giữ các tài nguyên khác. Có thể áp đặt một trong hai cơ chế truy xuất sau :

+ Buộc các tiến trình yêu cầu tất cả các tài nguyên mà nó cần tại một thời điểm, và tiến trình sẽ bị khóa(blocked) cho đến khi yêu cầu tài nguyên của nó được hệ điều hành đáp ứng

=> phương pháp này có khó khăn là tiến trình khó có thể ước lượng chính xác tài nguyên cần sử dụng vì có thể nhu cầu phụ thuộc vào quá trình xử lý . Ngoài ra nếu tiến trình chiếm giữ sẵn các tài nguyên chưa cần sử dụng ngay thì việc sử dụng tài nguyên sẽ kém hiệu quả. Tiến trình phải đợi trong một thời gian dài để có đủ tài nguyên mới có thể chuyển sang hoạt động được.

+ Khi tiến trình yêu cầu một tài nguyên mới và bị từ chối, nó phải giải phóng các tài nguyên đang chiếm giữ, sau đó lại được cấp phát trở lại khi có đủ tài nguyên.

=> phương pháp này làm phát sinh các khó khăn trong việc bảo vệ tính toàn vẹn dữ liệu của hệ thống.

- Với điều kiện không thu hồi tài nguyên: cho phép hệ thống được thu hồi tài nguyên từ các tiến trình bị khoá và cấp phát cho tiến trình khác và cấp phát trở lại cho tiến trình khi có đủ tài nguyên. Tuy nhiên với một số loại tài nguyên, việc thu hồi sẽ rất khó khăn vì vi phạm sự toàn vẹn dữ liệu.

- Với điều kiện tồn tại một chu trình: tránh tạo chu trình trong đồ thị bằng cách cấp phát tài nguyên theo một sự phân cấp như sau :

gọi R = {R_1, R_2,...,Rm} là tập các loại tài nguyên.

Các loại tài nguyên được phân cấp từ 1-N.

Ví dụ : F(đĩa) = 2, F(máy in) = 12

Giải thuật xác định trạng thái an toàn

Cần sử dụng các cấu trúc dữ liệu sau :

int Available[NumResources];

/* Available[r]= số lượng các thể hiện còn tự do của tài nguyên r*/

int Max[NumProcs, NumResources];

/*Max[p,r]= nhu cầu tối đa của tiến trình p về tài nguyên r*/

int Allocation[NumProcs, NumResources];

/* Allocation[p,r] = số lượng tài nguyên r thực sự cấp phát cho p*/

int Need[NumProcs, NumResources];

/* Need[p,r] = Max[p,r] - Allocation[p,r]*/

1.Giả sử có các mảng

int Work[NumResources] = Available;

int Finish[NumProcs] = false; với mọi i

2.Tìm i sao cho

Finish[i] == false

Need[i] <= Work[i]

Nếu có i thì sang bước 3

Nếu không có i như thế, đến bước 4.

3. Work = Work + Allocation[i];

  Finish[i] = true;

 Đến bước 2

4.Nếu Finish[i] == true với mọi i, thì hệ thống ở trạng thái an toàn.

Ngược lại nếu tồn tại i mà finish[i]==false thì hệ thống ở trạng thái không an toàn.

Ví dụ, cho hệ thống sau:

	Max			Allocation			Available		
	R1	R2	R3	R1	R2	R3	R1	R2	R3
P1	6	5	4	2	2	1	2	2	2
P2	5	4	6	3	1	3			
P3	4	3	5	2	1	3			
P4	3	5	2	1	3	1			

Ta tính được bảng Need

	Need		
	R1	R2	R3
P1	4	3	3
P2	2	3	3
P3	2	2	2
P4	2	2	1

B1. Work=(2,2,2)

Finish[i]=false với mọi i từ 1 đến 4.

B2: Tìm thấy tiến trình P3 thỏa mãn 2 điều kiện

Finish[3]=false

$Need_3$<= Work

B3: Work= (2,2,2)+(2,1,3) =(4,3,5)

Finish[3]=true

B2: tìm thấy P1 thỏa mãn 2 điều kiện

B3: Work= (4,3,5)+(2,2,1) =(6,5,6)

Finish[1]=true

B2: tìm thấy P2 thỏa mãn 2 điều kiện

B3: Work= (6,5,6)+(3,1,3) =(9,6,9)

Finish[2]=true

B2: tìm thấy P4 thỏa mãn 2 điều kiện

B3: Work= (9,6,9)+(1,3,1) =(10,9,10)

Finish[4]=true

B2: Không tìm thấy tiến trình nào thỏa mãn điều kiện

B4: finish[i]=true với mọi i từ 1 đến 4 do vậy hệ thống ở trạng thái an toàn.

Giải thuật yêu cầu tài nguyên

Giả sử tiến trình Pi yêu cầu thêm tài nguyên $Request_i$.

1.Nếu $Request_i$. <= Need[i], đến bước 2
Ngược lại, xảy ra tình huống lỗi

2.Nếu $Request_i$. <= Available, đến bước 3
Ngược lại, Pi phải chờ

3.Giả sử hệ thống đã cấp phát cho Pi các tài nguyên mà nó yêu cầu và cập nhật tình trạng hệ thống như sau:

Available = Available - $Request_i$;
$Allocation_i$= $Allocation_i$+ $Request_i$.;
$Need_i$ = $Need_i$ - $Request_i$.;

Nếu trạng thái kết quả là an toàn, lúc này các tài nguyên trên sẽ được cấp phát thật sự cho Pi

Ngược lại, Pi phải chờ

Ví dụ : Giả sử tình trạng hiện hành của hệ thống được mô tả như sau :

	Max			Allocation			Available		
	R1	R2	R3	R1	R2	R3	R1	R2	R3
P1	3	2	2	1	0	0	4	2	2
P2	6	1	3	2	1	1			
P3	3	1	4	2	1	1			
P4	4	2	2	0	0	2			

Nếu tiến trình P2 yêu cầu 4 cho R1, 1 cho R3. hãy cho biết yêu cầu này có thể đáp ứng mà bảo đảm không xảy ra tình trạng deadlock hay không ? Nhận thấy Available[1] =4, Available[3] =2 đủ để thỏa mãn yêu cầu của P2, ta có

	Need			Allocation			Available		
	R1	R2	R3	R1	R2	R3	R1	R2	R3
P1	2	2	2	1	0	0			
P2	0	0	1	6	1	2			
P3	1	0	3	2	1	1			
P4	4	2	0	0	0	2			

Trạng thái kết quả là an toàn, có thể cấp phát.

Phát hiện tắc nghẽn

Cần sử dụng các cấu trúc dữ liệu sau :

int Available[NumResources];

// Available[r]= số lượng các thể hiện còn tự do của tài nguyên r

int Allocation[NumProcs, NumResources];

// Allocation[p,r] = số lượng tài nguyên r thực sự cấp phát cho p

int Request[NumProcs, NumResources];

// Request[p,r] = số lượng tài nguyên r tiến trình p yêu cầu thêm

Giải thuật phát hiện tắc nghẽn

1. int Work[NumResources] = Available;

   int Finish[NumProcs];

for (i = 0; i < NumProcs; i++)

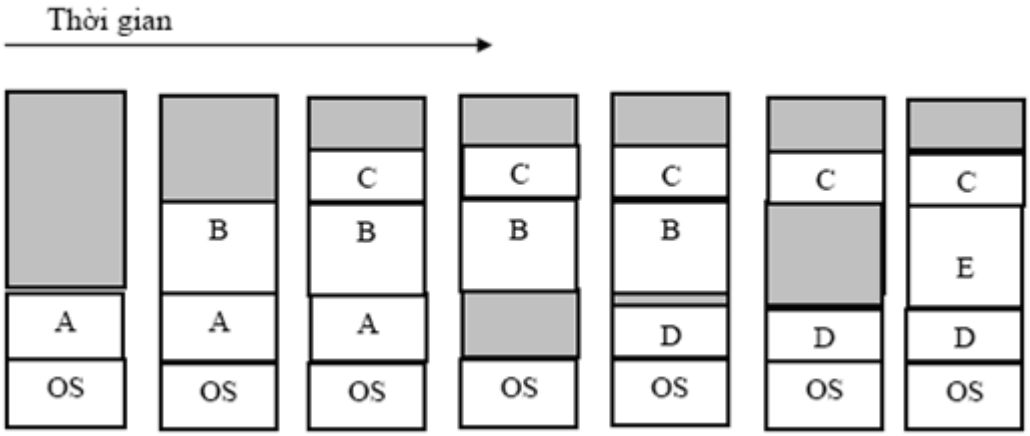
Finish[i] = (Allocation[i] == 0);

2. Tìm i sao cho

Finish[i] == false

Request[i] <= Work

Tình trạng « đói tài nguyên »: làm sao bảo đảm rằng không có một tiến trình luôn luôn bị thu hồi tài nguyên ?



Hình 3.4 Cấp phát đa vùng với phân vùng động

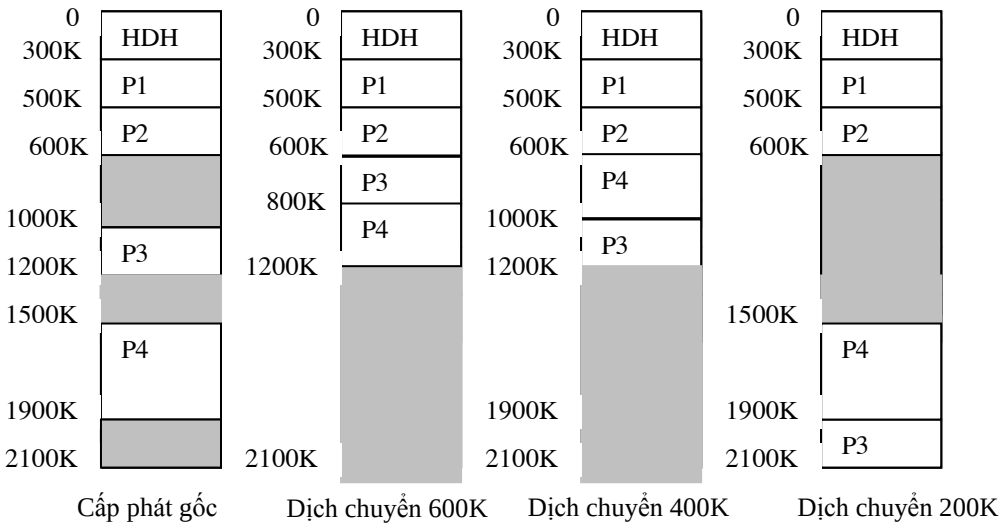
Trong kỹ thuật phân vùng động, số lượng các vùng trên bộ nhớ và kích thước mỗi phân vùng là có thể thay đổi. Tức là phần user program trên bộ nhớ không dược phân chia trước mà nó chỉ được ấn định sau khi dã có một tiến trình được nạp vào bộ nhớ chính. Khi có một tiến trình được nạp vào bộ nhớ nó được hệ điều hành cấp cho nó một không gian vừa đủ, liên tục để chứa tiến trình, phần còn lại để sẵn sàng cấp cho tiến trình khác sau này. Khi một tiến trình kết thúc nó được đưa ra ngoài và phần không gian bộ nhớ mà tiến trình này trả cho hệ điều hành sẽ được hệ điều hành cấp cho tiến trình khác

Xuất hiện hiện tượng phân mảnh ngoại vi(*external fragmentation*) : khi các tiến trình lần lượt vào và ra khỏi hệ thống, dần dần xuất hiện các khe hở giữa các tiến trình. Đây là các khe hở được tạo ra do kích thước của tiến trình mới được nạp nhỏ hơn kích thước vùng nhớ mới được giải phóng bởi một tiến trình đã kết thúc và ra khỏi hệ thống. , không gian bộ nhớ trống bị phân rã thành những mảnh nhớ nhỏ.

Hiện tượng này có thể dẫn đến tình huống tổng vùng nhớ trống đủ để thoả mãn yêu cầu, nhưng các vùng nhớ này lại không liên tục ! Người ta có thể áp dụng kỹ thuật « dồn bộ nhớ » (*memory compaction*) để kết hợp các mảnh bộ nhớ nhỏ rời rạc thành một vùng nhớ lớn liên tục. Tuy nhiên, kỹ thuật này đòi hỏi nhiều thời gian xử lý, ngoài ra, sự kết buộc địa chỉ phải thực hiện vào thời điểm xử lý, vì các tiến trình có thể bị di chuyển trong quá trình dồn bộ nhớ.

Thuật toán đơn giản là dịch chuyển các tiến trình về phía đầu của bộ nhớ.

Ví dụ:



Hình 3.5

Vấn đề cấp phát động

Lựa chọn vùng nhớ tự do trong danh sách các vùng nhớ tự do có kích thước bằng hoặc lớn hơn kích thước của tiến trình để cấp phát cho tiến trình.

Có 3 chiến lược phổ biến nhất được dùng:

• **First-fit**: cấp phát vùng nhớ tự do đầu tiên đủ lớn chứa tiến trình. Tìm kiếm có thể bắt đầu tại đầu từ danh sách tập hợp các vùng trống hay tại điểm kết thúc của tìm kiếm first-fit trước đó. Chúng ta dừng tìm kiếm ngay khi chúng ta tìm thấy một vùng trống đủ lớn.

• **Best-fit**: cấp phát vùng nhớ tự do nhỏ nhất đủ lớn chứa tiến trình. Chúng ta phải tìm toàn bộ danh sách, trừ khi danh sách được xếp thứ tự theo kích thước.

Worst-fit: cấp phát vùng nhớ tự do lớn nhất đủ lớn để chứa tiến trình. Chúng ta phải tìm toàn bộ danh sách trừ khi nó được xếp theo thứ tự kích thước.

Quản lý các khối rỗi bận

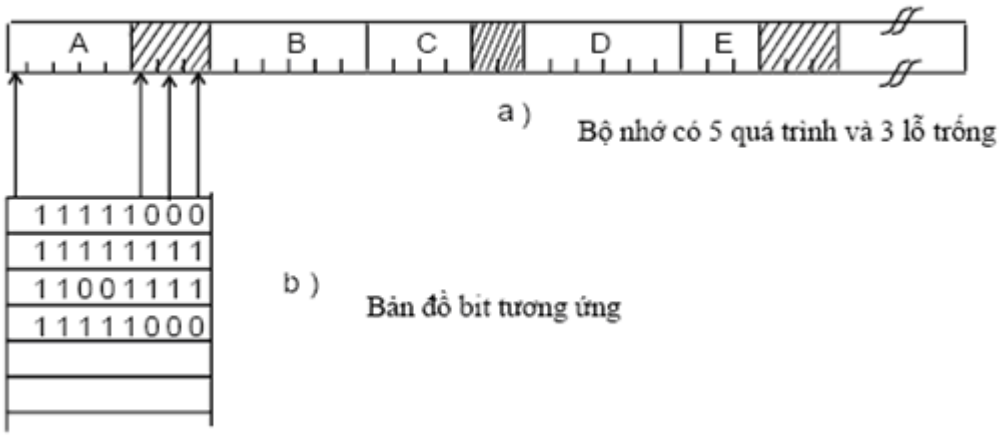
Bộ nhớ được chia thành các đơn vị cấp phát, có kích thước bằng nhau.

- **Quản lý bằng bản đồ bit:**

Mỗi đơn vị cấp phát được ánh xạ tới một bit trong bản đồ bit. Giá trị bit này xác định trạng thái của đơn vị bộ nhớ đó: 0 đang tự do, 1 đã được cấp phát. Khi cần nạp

một tiến trình có kích thước k đơn vị, hệ thống sẽ tìm trong bản đồ bit một dãy k bit có giá trị 0.

Kích thước của đơn vị cấp phát là vấn đề lớn trong thiết kế. Nếu kích thước đơn vị cấp phát nhỏ sẽ làm tăng kích thước của bản đồ bit. Ngược lại, nếu kích thước đơn vị cấp phát lớn có thể gây hao phí cho đơn vị cấp phát sau cùng. Đây là giải pháp đơn giản nhưng thực hiện chậm nên ít được dùng.

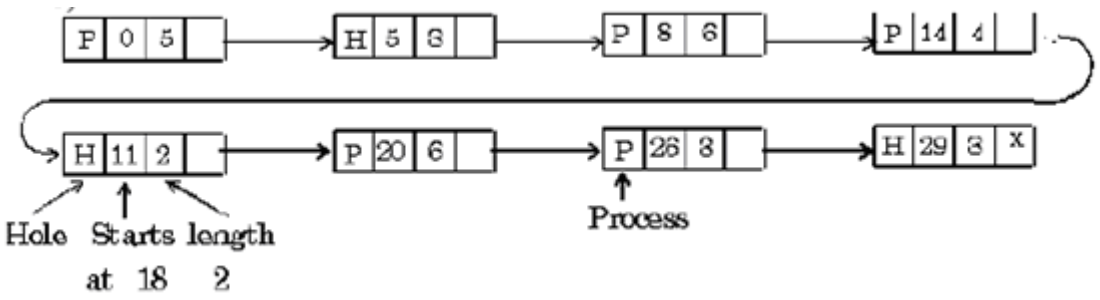


Hình 3.6 Quản lý bộ nhớ bằng bản đồ bit

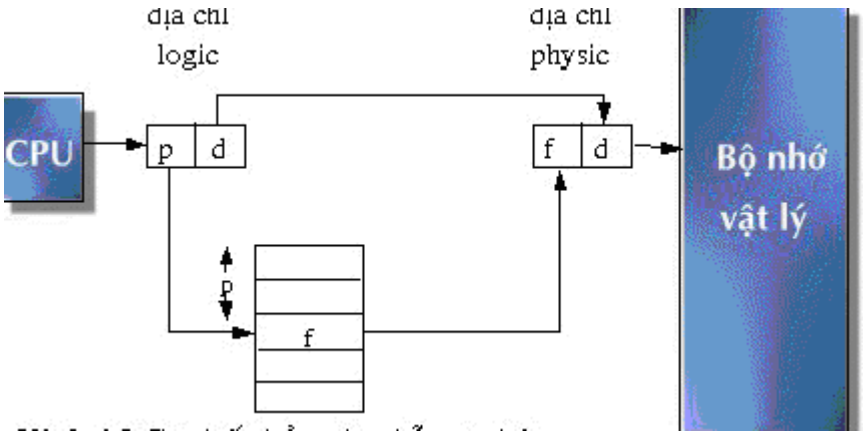
- Quản lý bằng danh sách liên kết:

Dùng một danh sách liên kết để quản lý các phân đoạn bộ nhớ đã cấp phát và phân đoạn tự do. Danh sách liên kết gồm nhiều nút liên tiếp. Mỗi nút gồm 3 trường chính: trường thứ nhất để xác định khối nhớ đã cấp phát (P: tiến trình) hay đang còn trống (H:Hole), trường thứ hai cho biết thứ tự của đơn vị cấp phát đầu tiên trong khối, trường thứ ba cho biết khối gồm bao nhiêu đơn vị cấp phát.

Việc sắp xếp các phân đoạn theo địa chỉ hay theo kích thước tuỳ thuộc vào giải thuật quản lý bộ nhớ.



Hình 3.7 Quản lý bộ nhớ bằng danh sách liên kết



Hình 3.9 Cơ chế phần cứng hỗ trợ phân trang

Trong kỹ thuật này hệ điều hành phải đưa ra các cơ chế thích hợp để theo dõi trạng thái của các khung trang (còn trống hay đã cấp phát) trên bộ nhớ. Hệ điều hành sử dụng một danh sách để ghi số hiệu của các khung trang còn trống trên bộ nhớ, hệ điều hành dựa vào danh này để tìm các khung trang trống trước khi quyết định nạp một tiến trình vào bộ nhớ, danh sách này được cập nhật ngay sau khi hệ điều hành nạp một tiến trình vào bộ nhớ, được kết thúc hoặc swapout ra bên ngoài.

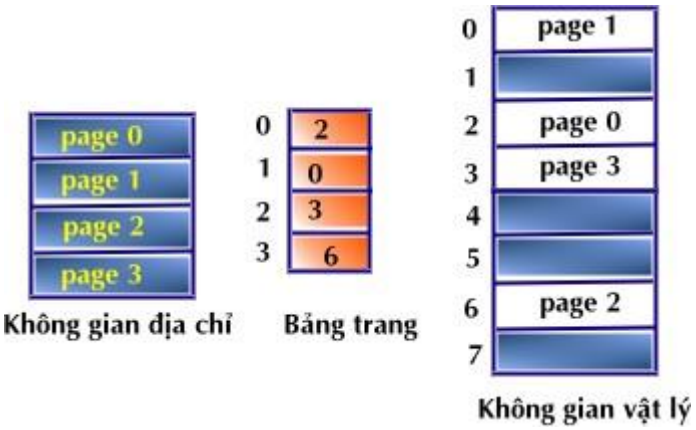
Cơ chế phần cứng hỗ trợ thực hiện chuyển đổi địa chỉ trong cơ chế phân trang là bảng trang (*pages table*). Bảng trang để theo dõi vị trí các trang tiến trình trên bộ nhớ.

Mỗi tiến trình có một bảng trang.

Số phần tử của bảng trang=số trang trong không gian địa chỉ của tiến trình.

Các phần tử được đánh số bắt đầu từ 0

Mỗi phần tử trong bảng trang mô tả một trang cho biết địa chỉ bắt đầu của vị trí lưu trữ trang tương ứng trong bộ nhớ vật lý (số hiệu khung trang trong bộ nhớ vật lý đang chứa trang).



Hình 3.10

Chuyển đổi địa chỉ:

Địa chỉ logic <p, d>

Địa chỉ vật lý <f, d>

số hiệu trang (p): sử dụng như chỉ mục đến phần tử tương ứng trong bảng trang.

địa chỉ tương đối trong trang (d): kết hợp với địa chỉ bắt đầu của khung trang tương ứng để tạo ra địa chỉ vật lý mà trình quản lý bộ nhớ sử dụng.

Số hiệu khung trang (f):địa chỉ bắt đầu của khung trang trong bộ nhớ vật lý.

Kích thước của trang do phần cứng qui định. Để dễ phân tích địa chỉ ảo thành số hiệu trang và địa chỉ tương đối, kích thước của một trang thông thường là một lũy thừa của 2 (biến đổi trong phạm vi 512 bytes và 8192 bytes).

Nếu kích thước của không gian địa chỉ là 2^m và kích thước trang là 2^n, thì *m-n* bits cao của địa chỉ ảo sẽ biểu diễn số hiệu trang, và *n* bits thấp cho biết địa chỉ tương đối trong trang.

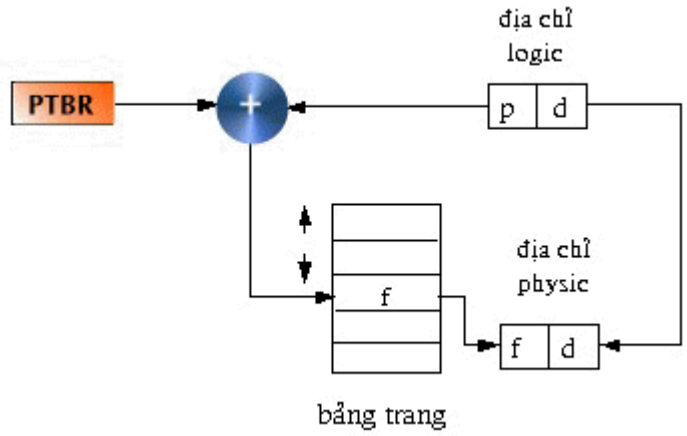
Số trang	Độ dời trang
P	D
m –n	N

Cài đặt bảng trang:

- Với các bảng trang có kích thước nhỏ, trong trường hợp đơn giản nhất, bảng trang được cài đặt trongmột tập các thanh ghi

- Nếu bảng trang có kích thước lớn, nó phải được lưu trữ trong bộ nhớ chính, và sử dụng một thanh ghi để lưu địa chỉ bắt đầu lưu trữ bảng trang (PTBR).

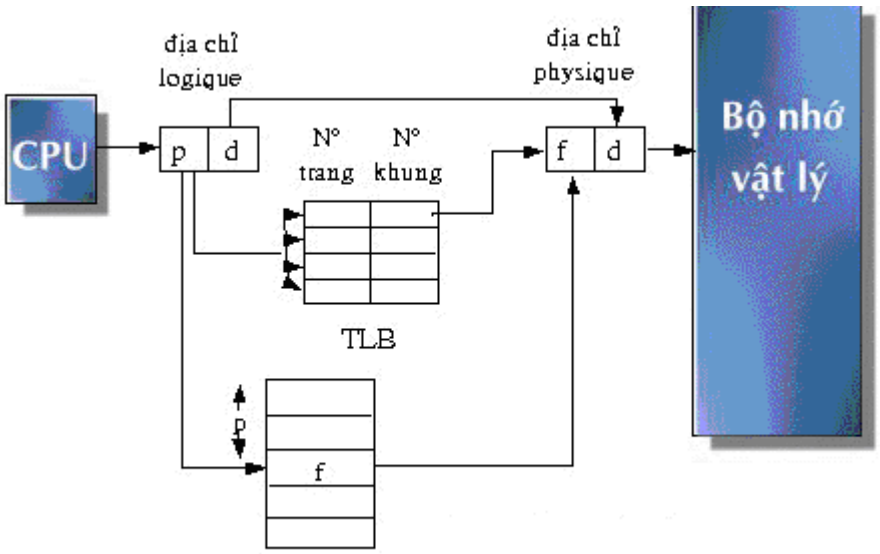
Theo cách tổ chức này, mỗi truy xuất đến dữ liệu hay chỉ thị đều đòi hỏi hai lần truy xuất bộ nhớ : một cho truy xuất đến bảng trang và một cho bản thân dữ liệu, do vậy truy cập chậm.



Hình 3.11 Sử dụng thanh ghi nền trỏ đến bảng trang

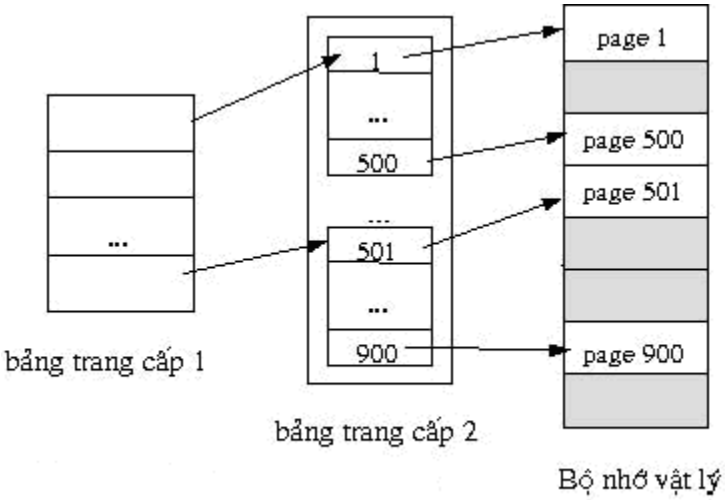
- Để nâng cao tốc độ truy xuất, sử dụng thêm một vùng nhớ đặc biệt , với tốc độ truy xuất nhanh và cho phép tìm kiếm song song, vùng nhớ cache nhỏ này thường được gọi là bộ nhớ kết hợp (translation look-aside buffer TLBs). Mỗi thanh ghi trong bộ nhớ kết hợp chứa số hiệu trang và số hiệu khung trang tương ứng, khi CPU phát sinh một địa chỉ logic, số hiệu trang của địa chỉ sẽ được so sánh cùng lúc với các số hiệu trang trong bộ nhớ kết hợp để tìm ra phần tử tương ứng. Nếu có trang tương ứng trong bộ nhớ kết hợp thì sẽ xác định ngay số hiệu khung trang tương ứng, nếu không mới cần thực hiện thao tác tìm kiếm trong bảng trang. Nhờ đặc tính này mà việc tìm kiếm trên bộ nhớ kết hợp được thực hiện rất nhanh, nhưng chi phí phần cứng lại cao.

Trong kỹ thuật phân trang, TLBs được sử dụng để lưu trữ các trang bộ nhớ được truy cập gần hiện tại nhất.



Hình 3.12 Bảng trang với TLBs

Bảng trang đa cấp:



Hình 3.13 Bảng trang 2 cấp

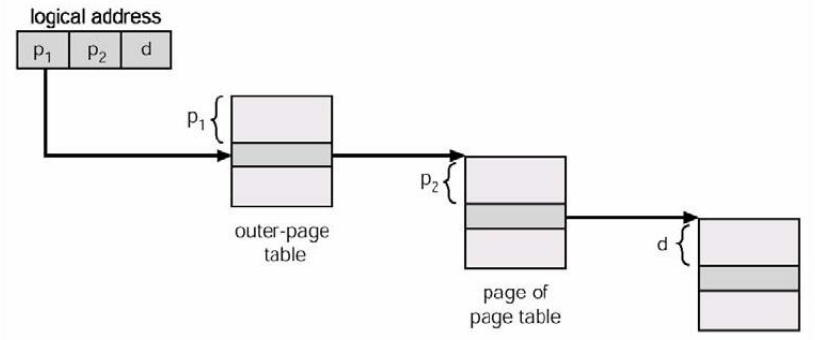
Mỗi hệ điều hành có một phương pháp riêng để tổ chức lưu trữ bảng trang. Đa số các hệ điều hành cấp cho mỗi tiến trình một bảng trang. Tuy nhiên phương pháp này không thể chấp nhận được nếu hệ điều hành cho phép quản lý một không gian địa chỉ có dung lượng quá (2^{31}, 2^{9}): trong các hệ thống như thế, bản thân bảng trang đòi hỏi một vùng nhớ quá lớn(2^{22})!

Phân trang đa cấp: phân chia bảng trang thành các phần nhỏ, bản thân bảng trang cũng sẽ được phân trang

Khi tiến trình thực hiện, chỉ có một phần của bảng trang được nạp vào bộ nhớ chính, đó là phần chứa các phần tử của các trang đang thực hiện tại thời điểm hiện tại.

Ví dụ trong bảng trang 2 cấp cho máy 32 bit với kích thước trang 4KB. Địa chỉ logic được chia thành số trang chứa 20 bit và độ dời trang chứa 12 bit. Vì chúng ta phân trang bảng trang, số trang được chia thành số trang 10 bit và độ dời trang 10-bit. Do đó, một địa chỉ logic như sau:

Số trang		Độ dời trang
P_1	P_2	d
10	10	12



Hình 3.14

Phân trang 3 cấp

Không gian địa chỉ 64 bit, kích thước trang 4KB

Trang bên ngoài cấp 2	Trang bên ngoài	Trang bên trong	Độ dời
P1	P2	P3	D
32	10	10	12

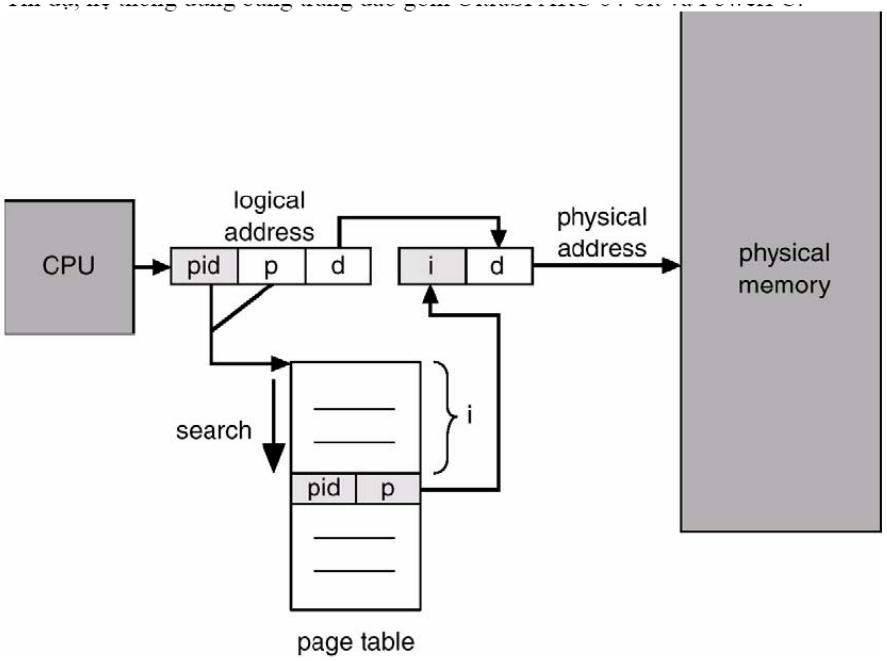
Bảng trang nghịch đảo (inverted page table).

Sử dụng duy nhất một *bảng trang nghịch đảo* cho tất cả các tiến trình. Mỗi phần tử trong *bảng trang nghịch đảo* phản ánh một khung trang trong bộ nhớ bao gồm địa chỉ logic của một trang đang được lưu trữ trong bộ nhớ vật lý tại khung trang này, cùng với thông tin về tiến trình đang được sỡ hữu trang. Mỗi địa chỉ ảo khi đó là một bộ ba <pid,p, d >

Trong đó : pid là định danh của tiến trình

p là số hiệu trang

d là địa chỉ tương đối trong trang



Hình 3.15

Mỗi phần tử trong bảng trang nghịch đảo là một cặp <pid, p >. Khi một tham khảo đến bộ nhớ được phát sinh, một phần địa chỉ ảo là <pid, p > được đưa đến cho trình quản lý bộ nhớ để tìm phần tử tương ứng trong bảng trang nghịch đảo, nếu tìm thấy, địa chỉ vật lý <i,d> sẽ được phát sinh. Trong các trường hợp khác, xem như tham khảo bộ nhớ đã truy xuất một địa chỉ bất hợp lệ.

Bảo vệ:

Cơ chế bảo vệ trong hệ thống phân trang được thực hiện với các bit bảo vệ được gắn với mỗi khung trang. Thông thường, các bit này được lưu trong bảng trang , vì mỗi truy xuất đến bộ nhớ đều phải tham khảo đến bảng trang để phát sinh địa chỉ vật lý, khi đó, hệ thống có thể kiểm tra các thao tác truy xuất trên khung trang tương ứng có hợp lệ với thuộc tính bảo vệ của nó không.

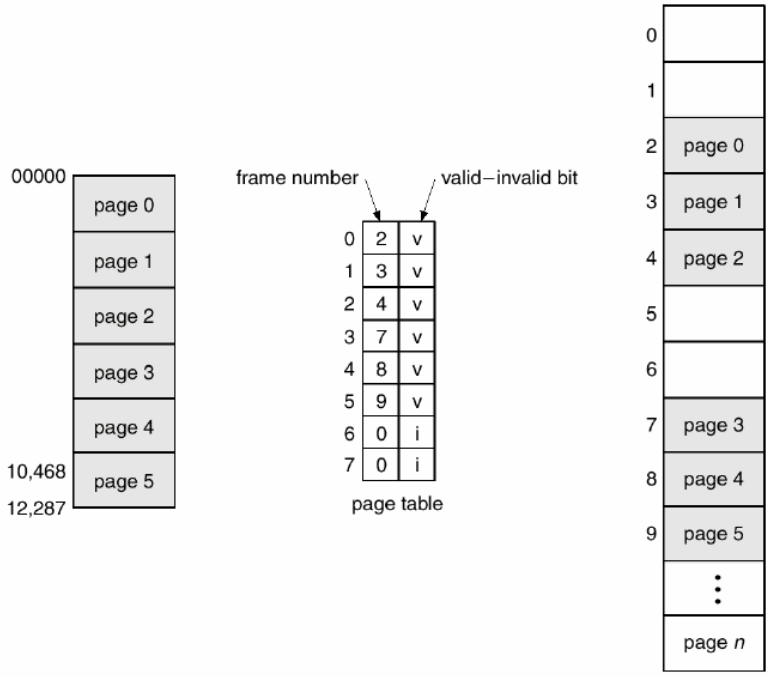
Ngoài ra, một bit phụ trội được thêm vào trong cấu trúc một phần tử của bảng trang : bit hợp lệ-bất hợp lệ (valid-invalid).

Hợp lệ : trang tương ứng thuộc về không gian địa chỉ của tiến trình.

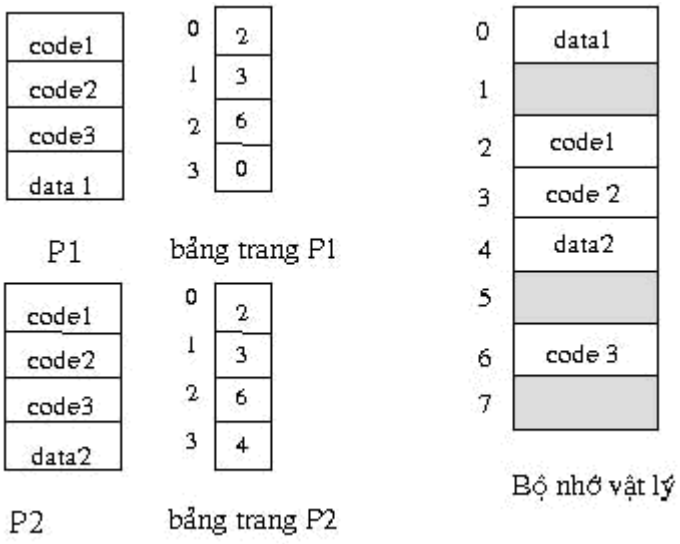
số hiệu khung trang	bit valid-invalid

Bất hợp lệ: trang tương ứng không nằm trong không gian địa chỉ của tiến trình, điều này có nghĩa tiến trình đã truy xuất đến một địa chỉ không được phép.

Hình 3.16 *Cấu trúc một phần tử trong bảng trang*



Chia sẻ bộ nhớ trong cơ chế phân trang:



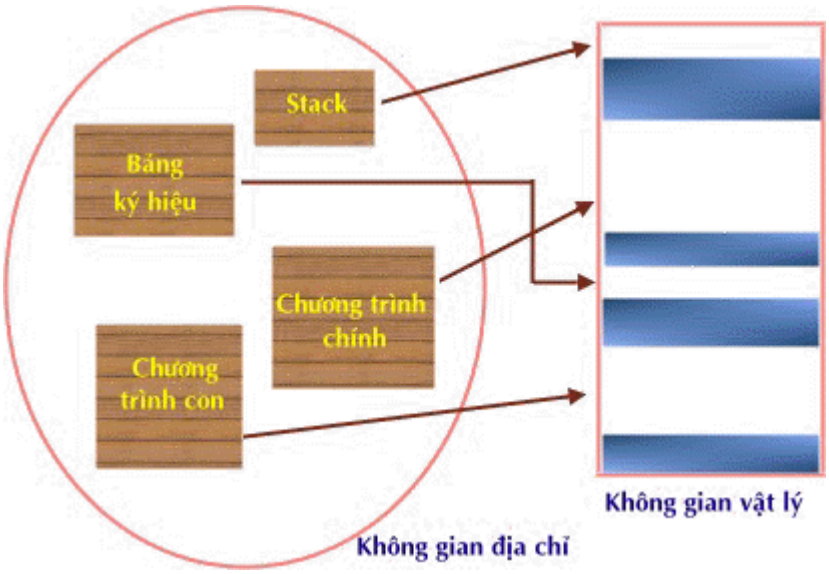
Hình 3.17 Chia sẻ các trang trong hệ phân trang

Một ưu điểm của cơ chế phân trang là cho phép chia sẻ các trang giữa các tiến trình.Trong trường hợp này, sự chia sẻ được thực hiện bằng cách ánh xạ nhiều địa chỉ logic vào một địa chỉ vật lý duy nhất. Có thể áp dụng kỹ thuật này để cho phép các tiến

thường mỗi thành phần của một chương trình/tiến trình như: code, data, stack, subprogram…là một đoạn.

Bộ nhớ được tổ chức cấp phát động

Khi một tiến trình được nạp vào bộ nhớ thì tất cả các đoạn của nó sẽ được nạp vào các phân đoạn còn trống khác nhau trên bộ nhớ. Các phân đoạn này có thể không liên tiếp nhau.



Hình 3.18 Mô hình phân đoạn bộ nhớ

Hình 3.19 Cơ chế phần cứng hổ trợ kĩ thuật phân đoạn

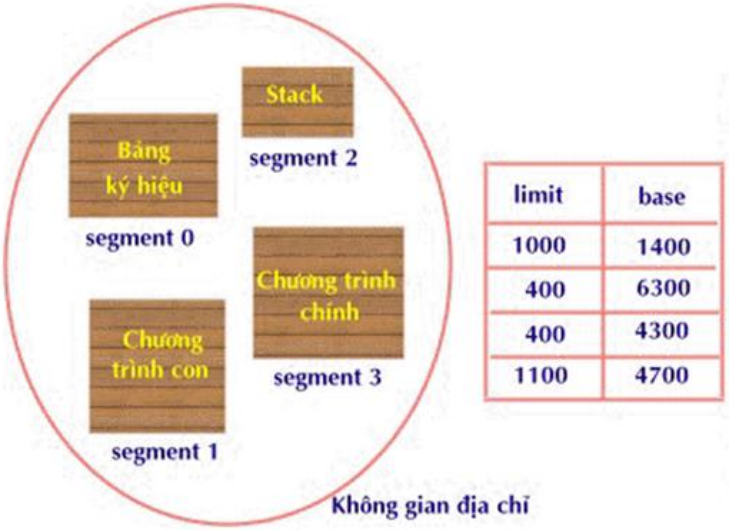
Để theo dõi các đoạn của các tiến trình khác nhau trên bộ nhớ, hệ điều hành sử dụng các bảng phân đoạn tiến trình, thông thường một tiến trình có một bảng phân đoạn riêng. Mỗi phần tử trong bảng phân đoạn gồm tối thiểu 2 trường: trường thứ nhất cho biết địa chỉ cơ sở(base) của phân vùng mà đoạn chương trình tương ứng được nạp, trường thứ 2 cho biết độ dài/giới hạn (limit) của phân đoạn, trường này còn có tác dụng dùng để kiểm soát sự truy xuất bất hợp lệ của các tiến trình.

Chuyển đổi địa chỉ:

Mỗi địa chỉ ảo là một bộ <***s,d***> *:*

số hiệu phân đoạn s : được sử dụng như chỉ mục đến bảng phân đoạn

địa chỉ tương đối d : có giá trị trong khoảng từ 0 đến giới hạn chiều dài của phân đoạn. Nếu địa chỉ tương đối hợp lệ, nó sẽ được cộng với giá trị chứa cơ sở để phát sinh địa chỉ vật lý tương ứng.



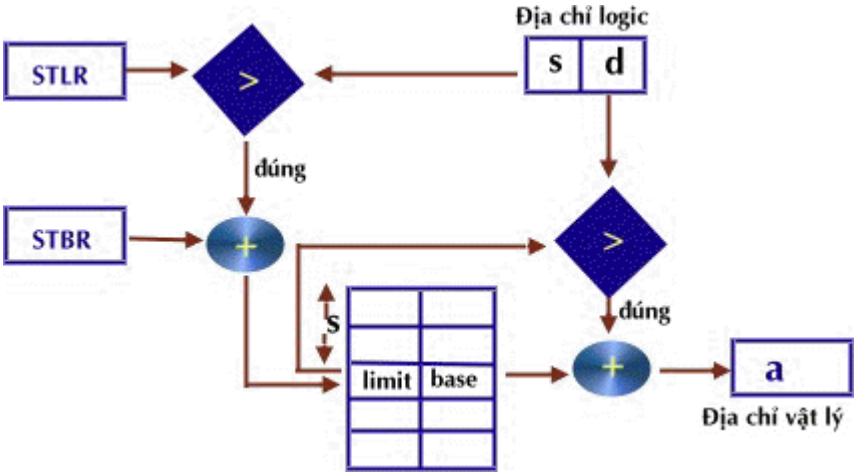
Cài đặt bảng phân đoạn:

Hình 3.20 Hệ thống phân đoạn

Có thể sử dụng các thanh ghi để lưu trữ bảng phân đoạn nếu số lượng phân đoạn nhỏ. Trong trường hợp chương trình bao gồm quá nhiều phân đoạn, bảng phân đoạn phải được lưu trong bộ nhớ chính. Một *thanh ghi nền bảng phân đoạn* (STBR) chỉ đến địa chỉ bắt đầu của bảng phân đoạn. Vì số lượng phân đoạn sử dụng trong một chương trình có thể thay đổi, cần sử dụng thêm một *thanh ghi đặc tả kích thước bảng phân đoạn* (STLR).

Hệ điều hành cũng tổ chức một danh sách riêng để theo dõi các đoạn còn trống trên bộ nhớ.

Với một địa chỉ logic <*s*,*d*>, trước tiên số hiệu phân đoạn *s* được kiểm tra tính hợp lệ (s <STLR). Kế tiếp, cộng giá trị s với STBR để có được địa chỉ địa chỉ của phần



tử thứ *s* trong bảng phân đoạn (STBR+*s*). Địa chỉ vật lý cuối cùng là (STBR+*s* + *d*)

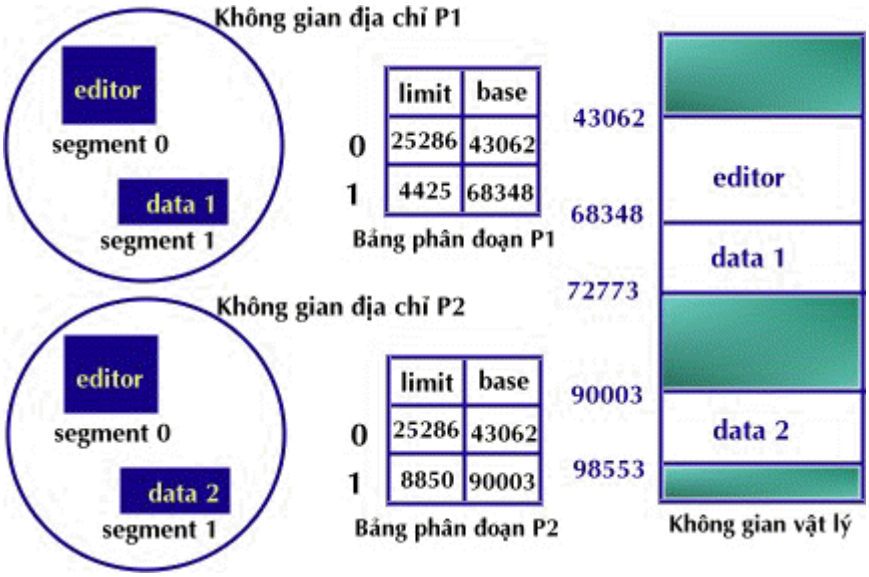
Hình 3.21 Sử dụng STBR, STLR và bảng phân đoạn

Bảo vệ: Một ưu điểm đặc biệt của cơ chế phân đoạn là khả năng đặc tả thuộc tính bảo vệ cho mỗi phân đoạn. Vì mỗi phân đoạn biểu diễn cho một phần của chương trình với ngữ nghĩa được người dùng xác định, người sử dụng có thể biết được một phân đoạn chứa đựng những gì bên trong, do vậy họ có thể đặc tả các thuộc tính bảo vệ thích hợp cho từng phân đoạn.

Cơ chế phần cứng phụ trách chuyển đổi địa chỉ bộ nhớ sẽ kiểm tra các bit bảo vệ được gán với mỗi phần tử trong bảng phân đoạn để ngăn chặn các thao tác truy xuất bất hợp lệ đến phân đoạn tương ứng.

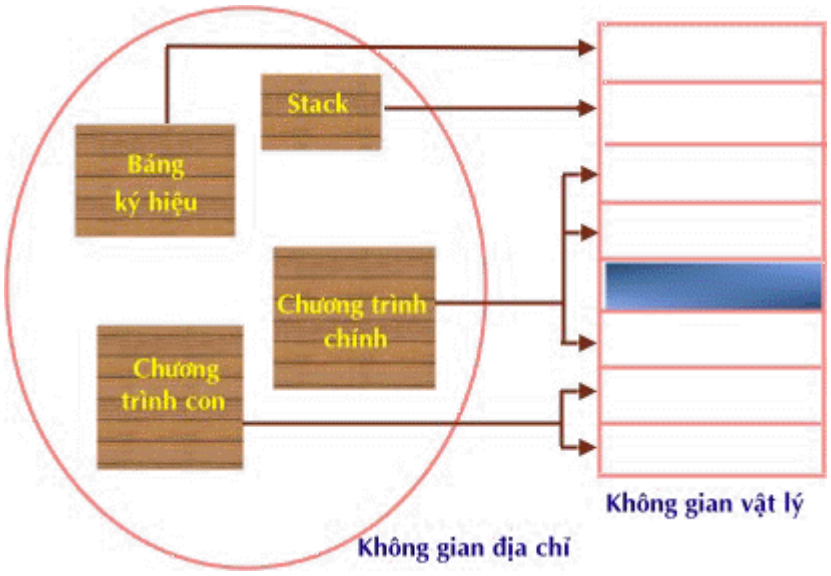
Kỹ thuật phân đoạn dùng chính nó để cài đặt các chính sách bảo vệ và chia sẻ bộ nhớ. Bởi vì mỗi phần tử trong bảng đoạn bao gồm một trường độ dài đoạn và trường chỉ nơi bắt đầu của đoạn, nên một tiến trình trong đoạn không thể truy cập đến một vị trí trong bộ nhớ chính mà vị trí này vượt qua giới hạn của đoạn, ngoiaj trừ đó là truy cập dữ liệu đến một đoạn dữ liệu nào đó.

Chia sẻ phân đoạn:



Hình 3.22 *Chia sẻ code trong hệ phân đoạn*

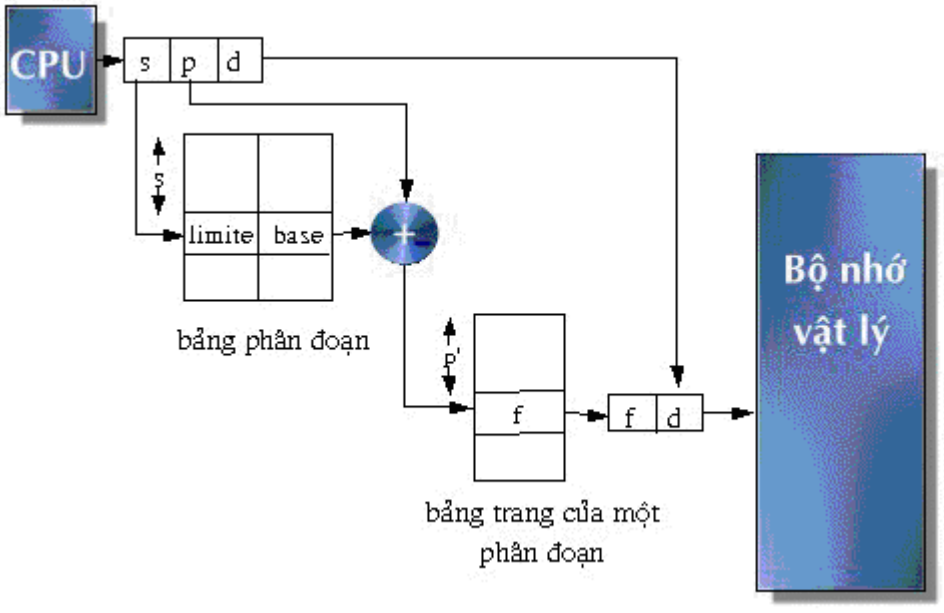
Một ưu điểm khác của kỹ thuật phân đoạn là khả năng chia sẻ ở mức độ phân đoạn. Nhờ khả năng này, các tiến trình có thể chia sẻ với nhau từng phần chương trình (ví dụ các thủ tục, hàm), không nhất thiết phải chia sẻ toàn bộ



Hình 3.23 Mô hình phân đoạn kế hợp phân trang

số hiệu trang (p): sử dụng như chỉ mục đến phần tử tương ứng trong bảng trang của phân đoạn.

địa chỉ tương đối trong trang (d): kết hợp với địa chỉ bắt đầu của khung trang để tạo ra địa chỉ vật lý mà trình quản lý bộ nhớ sử dụng.



Hình 3.24 Cơ chế phần cứng của sự phân đoạn kết hợp phân trang

Tất cả các mô hình tổ chức bộ nhớ trên đây đều có khuynh hướng cấp phát cho tiến trình toàn bộ các trang yêu cầu trước khi thật sự xử lý. Vì bộ nhớ vật lý có kích thước rất giới hạn, điều này dẫn đến hai điểm bất tiện sau :

-Dựa vào các tiêu chuẩn cụ thể để chọn một trang/đoạn nào đó trong số các trang/đoạn đang nằm trong bộ nhớ chính để swap out trong trường hợp cần thiết.

Phân trang theo yêu cầu (demand paging)

Một hệ thống phân trang theo yêu cầu là hệ thống sử dụng kỹ thuật phân trang kết hợp với kỹ thuật swapping. Một tiến trình được xem như một tập các trang, thường trú trên bộ nhớ phụ (thường là đĩa). Khi cần xử lý, tiến trình sẽ được nạp vào bộ nhớ chính. Nhưng thay vì nạp toàn bộ chương trình, chỉ những trang cần thiết trong thời điểm hiện tại mới được nạp vào bộ nhớ. Như vậy một trang chỉ được nạp vào bộ nhớ chính khi có yêu cầu.

Với mô hình này, cần cung cấp một cơ chế phần cứng giúp phân biệt các trang đang ở trong bộ nhớ chính và các trang trên đĩa. Có thể sử dụng lại bit *valid-invalid* nhưng với ngữ nghĩa mới:

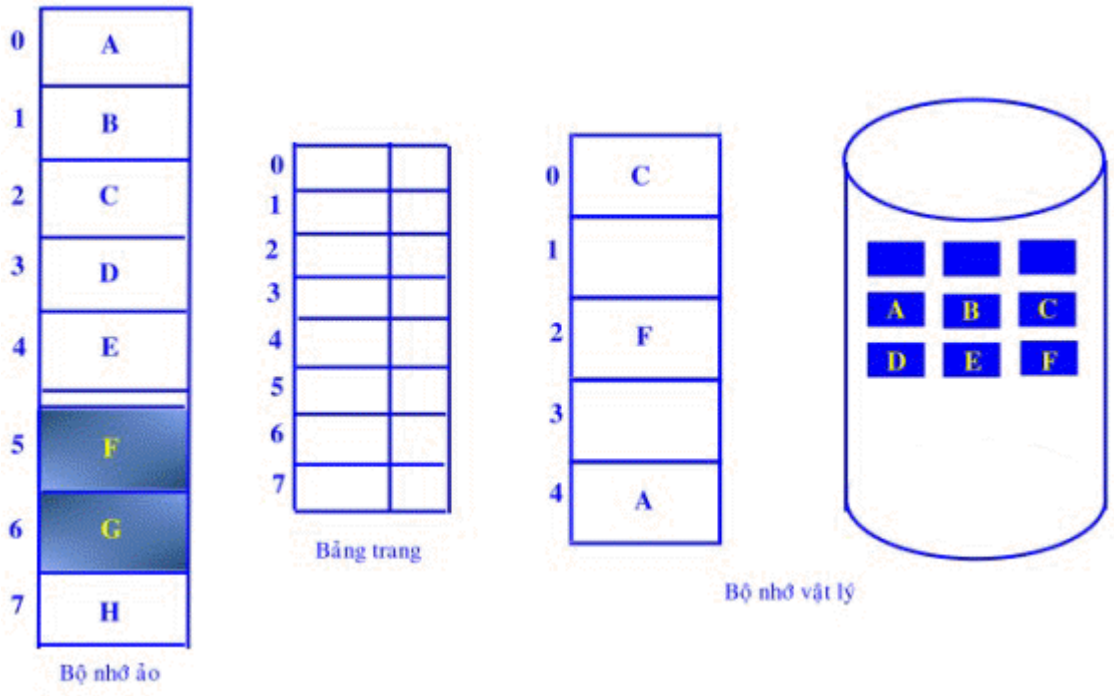
valid : trang tương ứng là hợp lệ và đang ở trong bộ nhớ chính .

invalid : hoặc trang bất hợp lệ (không thuộc về không gian địa chỉ của tiến trình) hoặc trang hợp lệ nhưng đang được lưu trên bộ nhớ phụ.

Một phần tử trong bảng trang mô tả cho một trang không nằm trong bộ nhớ chính, sẽ được đánh dấu *invalid* và chứa địa chỉ của trang trên bộ nhớ phụ.

Cơ chế phần cứng :

Cơ chế phần cứng hỗ trợ kỹ thuật phân trang theo yêu cầu là sự kết hợp của cơ chế hỗ trợ kỹ thuật phân trang và kỹ thuật swapping:



Hình 3.26 Bảng trang với một số trang trên bộ nhớ phụ

Bảng trang: Cấu trúc bảng trang phải cho phép phản ánh tình trạng của một trang là đang nằm trong bộ nhớ chính hay bộ nhớ phụ.

Bộ nhớ phụ: Bộ nhớ phụ lưu trữ những trang không được nạp vào bộ nhớ chính. Bộ nhớ phụ thường được sử dụng là đĩa, và vùng không gian đĩa dùng để lưu trữ tạm các trang trong kỹ thuật swapping được gọi là *không gian swapping.*

Lỗi trang

Truy xuất đến một trang được đánh dấu bất hợp lệ sẽ làm phát sinh một *lỗi trang* (*page fault*). Khi dò tìm trong bảng trang để lấy các thông tin cần thiết cho việc chuyển đổi địa chỉ, nếu nhận thấy trang đang được yêu cầu truy xuất là bất hợp lệ, cơ chế phần cứng sẽ phát sinh một ngắt để báo cho hệ điều hành. Hệ điều hành sẽ xử lý lỗi trang như sau :

Kiểm tra truy xuất đến bộ nhớ là hợp lệ hay bất hợp lệ

Nếu truy xuất bất hợp lệ : kết thúc tiến trình

Ngược lại : đến bước 3

Tìm vị trí chứa trang muốn truy xuất trên đĩa.

Tìm một khung trang trống trong bộ nhớ chính :

Nếu tìm thấy : đến bước 5

Nếu không còn khung trang trống, chọn một khung trang « nạn nhân » và chuyển trang « nạn nhân » ra bộ nhớ phụ (lưu nội dung của trang đang chiếm giữ khung trang này lên đĩa), cập nhật bảng trang tương ứng rồi đến bước 5

Chuyển trang muốn truy xuất từ bộ nhớ phụ vào bộ nhớ chính : nạp trang cần truy xuất vào khung trang trống đã chọn (hay vừa mới làm trống) ; cập nhật nội dung bảng trang, bảng khung trang tương ứng.

Tái kích hoạt tiến trình người sử dụng.

Sự thay thế trang là cần thiết cho kỹ thuật phân trang theo yêu cầu. Nhờ cơ chế này, hệ thống có thể hoàn toàn tách rời bộ nhớ ảo và bộ nhớ vật lý, cung cấp cho lập trình viên một bộ nhớ ảo rất lớn trên một bộ nhớ vật lý có thể bé hơn rất nhiều lần.

Sự thi hành phân trang theo yêu cầu

Việc áp dụng kỹ thuật phân trang theo yêu cầu có thể ảnh hưởng mạnh đến tình hình hoạt động của hệ thống.

Gỉa sử *p* là xác suất xảy ra một lỗi trang (0<= p <=1):

p = 0 : không có lỗi trang nào

p = 1 : mỗi truy xuất sẽ phát sinh một lỗi trang

Thời gian thật sự cần để thực hiện một truy xuất bộ nhớ (TEA) là:

TEA = (1-p)ma + p (tdp) [+ swap out] + swap in + tái kích hoạt

Trong công thức này, *ma* là thời gian truy xuất bộ nhớ, *tdp* thời gian xử lý lỗi trang.

Có thể thấy rằng, để duy trì ở một mức độ chấp nhận được sự chậm trễ trong hoạt động của hệ thống do phân trang, cần phải duy trì *tỷ lệ phát sinh lỗi trang* thấp.

Hơn nữa, để cài đặt kỹ thuật phân trang theo yêu cầu, cần phải giải quyết hai vấn đề chính yếu : xây dựng một *thuật toán cấp phát khung trang*, và *thuật toán thay thế trang*.

Các thuật toán thay thế trang

Vấn đề chính khi thay thế trang là chọn lựa một trang « nạn nhân » để chuyển ra bộ nhớ phụ. Có nhiều thuật toán thay thế trang khác nhau, nhưng tất cả cùng chung một mục tiêu : chọn trang « nạn nhân » là trang mà sau khi thay thế sẽ gây ra ít lỗi trang nhất.

Có thể đánh giá hiệu qủa của một thuật toán bằng cách xử lý trên một *chuỗi các địa chỉ cần truy xuất* và tính toán số lượng lỗi trang phát sinh.

Ví dụ: Giả sử theo vết xử lý của một tiến trình và nhận thấy tiến trình thực hiện truy xuất các địa chỉ theo thứ tự sau :

0100, 0432, 0101, 0162, 0102, 0103, 0104, 0101, 0611, 0102, 0103,0104, 0101, 0610, 0102, 0103, 0104, 0101, 0609, 0102, 0105

Nếu có kích thước của một trang là 100 bytes, có thể viết lại *chuỗi truy xuất* trên giản lược hơn như sau :

1, 4, 1, 6, 1, 6, 1, 6, 1

Để xác định số các lỗi trang xảy ra khi sử dụng một thuật toán thay thế trang nào đó trên một chuỗi truy xuất cụ thể, còn cần phải biết số lượng khung trang sử dụng trong hệ thống.

Để minh hoạ các thuật toán thay thế trang sẽ trình bày, chuỗi truy xuất được sử dụng là :

7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

a) Thuật toán FIFO

Tiếp cận: Ghi nhận thời điểm một trang được mang vào bộ nhớ chính. Khi cần thay thế trang, trang ở trong bộ nhớ lâu nhất sẽ được chọn

Ví dụ : sử dụng 3 khung trang , ban đầu cả 3 đều trống :

7	**0**	**1**	**2**	**0**	**3**	**0**	**4**	**2**	**3**	**0**	**3**	**2**	**1**	**2**	**0**	**1**	**7**	**0**	**1**
7	7	7	2	2	2	2	4	4	4	0	0	0	0	0	0	0	7	7	7
	0	0	0	0	3	3	3	2	2	2	2	2	1	1	1	1	1	0	0
		1	1	1	1	0	0	0	3	3	3	3	3	2	2	2	2	2	1
*	*	*	*		*	*	*	*	*	*			*	*			*	*	*

Ghi chú : * : có lỗi trang

Thảo luận:

Để áp dụng thuật toán FIFO, thực tế không nhất thiết phải ghi nhận thời điểm mỗi trang được nạp vào bộ nhớ, mà chỉ cần tổ chức quản lý các trang trong bộ nhớ trong một danh sách FIFO, khi đó trang đầu danh sách sẽ được chọn để thay thế.

Thuật toán thay thế trang FIFO dễ hiểu, dễ cài đặt. Tuy nhiên khi thực hiện không phải lúc nào cũng có kết quả tốt : trang được chọn để thay thế có thể là trang chức nhiều dữ liệu cần thiết, thường xuyên được sử dụng nên được nạp sớm, do vậy khi bị chuyển ra bộ nhớ phụ sẽ nhanh chóng gây ra lỗi trang.

Số lượng lỗi trang xảy ra sẽ tăng lên khi số lượng khung trang sử dụng tăng. Hiện tượng này gọi là *nghịch lý Belady*.

Ví dụ: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

Sử dụng 3 khung trang , sẽ có 9 lỗi trang phát sinh

1	2	3	4	1	2	5	1	2	3	4	5
1	1	1	4	4	4	5	5	5	5	5	5
	2	2	2	1	1	1	1	1	3	3	3
		3	3	3	2	2	2	2	2	4	4
*	*	*	*	*	*	*			*	*	

Sử dụng 4 khung trang , sẽ có 10 lỗi trang phát sinh

1	2	3	4	1	2	5	1	2	3	4	5
1	1	1	1	1	1	5	5	5	5	4	4
	2	2	2	2	2	2	1	1	1	1	5
		3	3	3	3	3	3	2	2	2	2
			4	4	4	4	4	4	3	3	3
*	*	*	*			*	*	*	*	*	*

b) Thuật toán tối ưu

Tiếp cận: Thay thế trang sẽ lâu được sử dụng nhất trong tương lai.

Ví dụ : sử dụng 3 khung trang, khởi đầu đều trống:

7	**0**	**1**	**2**	**0**	**3**	**0**	**4**	**2**	**3**	**0**	**3**	**2**	**1**	**2**	**0**	**1**	**7**	**0**	**1**
7	7	7	2	2	2	2	2	2	2	2	2	2	2	2	2	2	7	7	7
	0	0	0	0	0	0	4	4	4	0	0	0	0	0	0	0	0	0	0
		1	1	1	3	3	3	3	3	3	3	3	1	1	1	1	1	1	1
*	*	*	*		*		*			*			*				*		

Thảo luận:

Thuật toán này bảo đảm số lượng lỗi trang phát sinh là thấp nhất, nó cũng không gánh chịu nghịch lý Belady, tuy nhiên, đây là một thuật toán không khả thi trong thực tế, vì không thể biết trước chuỗi truy xuất của tiến trình!

c) Thuật toán « Lâu nhất chưa sử dụng » (Least-recently-used LRU)

Tiếp cận: Với mỗi trang, ghi nhận thời điểm cuối cùng trang được truy cập, trang được chọn để thay thế sẽ là trang lâu nhất chưa được truy xuất.

Trang được hệ điều hành chọn để thay thế là trang có khoảng thời gian từ lúc nó được truy xuất gần đây nhất đến thời điểm hiện tại là dài nhất, so với các trang đang trên bộ nhớ chính.

Ví dụ: sử dụng 3 khung trang, khởi đầu đều trống:

7	**0**	**1**	**2**	**0**	**3**	**0**	**4**	**2**	**3**	**0**	**3**	**2**	**1**	**2**	**0**	**1**	**7**	**0**	**1**
7	7	7	2	2	2	2	4	4	4	0	0	0	1	1	1	1	1	1	1
	0	0	0	0	0	0	0	0	3	3	3	3	3	3	0	0	0	0	0
		1	1	1	3	3	3	2	2	2	2	2	2	2	2	2	7	7	7
*	*	*	*		*		*	*	*	*			*		*		*		

Thảo luận:

Thuật toán FIFO sử dụng thời điểm nạp để chọn trang thay thế, thuật toán tối ưu lại dùng thời điểm trang *sẽ* được sử dụng, vì thời điểm này không thể xác định trước nên thuật toán LRU phải dùng thời điểm cuối cùng trang được truy xuất – dùng quá khứ gần để dự đoán tương lai.

Thuật toán này đòi hỏi phải được cơ chế phần cứng hỗ trợ để xác định một thứ tự cho các trang theo thời điểm truy xuất cuối cùng. Có thể cài đặt theo một trong hai cách

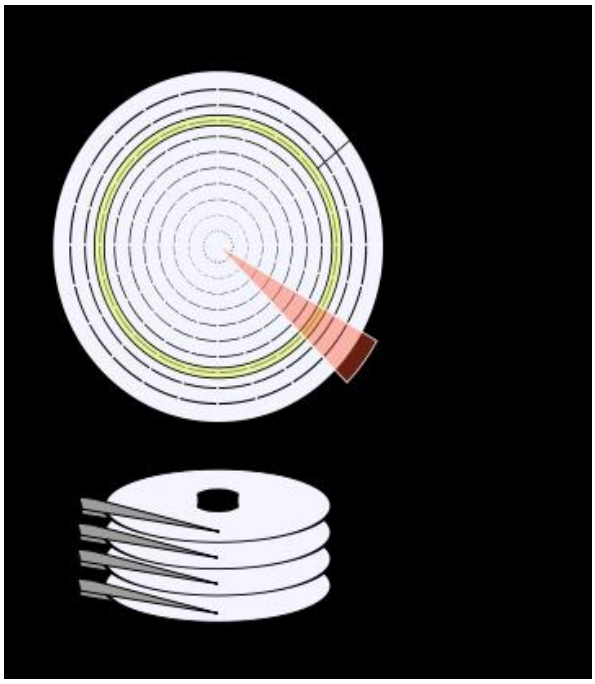
Mặt đĩa	Track	Sector	Sector logic	Thông tin lưu trữ
0	0	1	0	Boot record
0	0	2-5	1-4	FAT
0	0	6-9	5-8	Thư mục gốc
1	0	1-3	9-11	Thư mục gốc
1	0	4-9	12-17	Dữ liệu
0	1	1-9	18-26	Dữ liệu

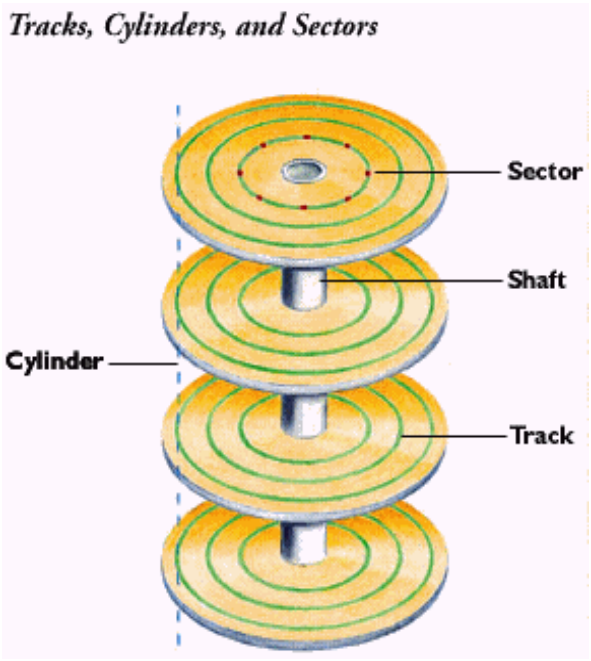
Bảng: Tương ứng giữa sector vật lý và sector logic trên đĩa mềm

Trên bề mặt đĩa tồn tại các sector mà hệ điều hành không thể ghi dữ liệu vào đó hoặc không thể đọc dữ liệu từ đó. Các sector này được gọi là các bad sector. Trong quá trình định dạng đĩa hệ điều hành đánh dấu loại bỏ các bad sector này.

Cylinder(từ trụ): Tập hợp các track có cùng số hiệu trên các mặt đĩa khác nhau của một hệ thống đĩa tạo thành một cylinder. Như vậy mặt đĩa có bao nhiêu track thì có bấy nhiêu cylinder. Cylinder chỉ có trên các ổ dĩa cứng.

Trên một ổ cứng có nhiều cylinder





Hình 4.6

Cluster (liên cung): Một nhóm gồm 2, 4 hoặc 6sector liên tiếp nhau tạo thành một cluster. Kích thước của cluster thường là bội số kích thước một sector. Các cluster được đánh địa chỉ bắt đầu từ 0. Số sector trên một cluster phụ thuộc vào từng loại đĩa. Một số hệ điều hành cho phép người sử dụng quy định số sector trên một cluster. Các hệ điều hành thường tổ chức lưu trữ dữ liệu, nội dung các tập tin trên đĩa theo từng cluster. Trên bề mặt đĩa cũng tồn tại các bad cluster, đó là các cluster có chứa bad sector.

Partition (phân khu): Partition là một tập các sector liền kề trên một đĩa. Mỗi partition có một bảng partition hoặc một cơ sở dữ liệu quản lý đĩa riêng, dùng để lưu trữ sector đầu tiên, kích thước và các đặc tính khác của partition.

Volume: Một volume tương tự một partition logic trên một đĩa, và nó được tạo khi ta định dạng một đĩa hoặc một phần của đĩa theo hệ thống File NTFS. Trong hệ điều hành windowsNT/2000 ta có thể tạo ra một volume trải dài trên nhiều đĩa vật lý khác nhau. Một đĩa có thể có một hoặc nhiều volume. NTFS điều khiển mỗi volume sao cho không phụ thuộc vào các volume khác.

Một volume bao gồm một tập các file cùng với bất kỳ một không gian chưa được cấp phát còn lại trên partition đĩa. Trong hệ thống file FAT, một volume cũng chứa các vùng đặc biệt được định dạng cho việc sử dụng của hệ thống file, như là bitmap, directory và cả system bootstrap trên các file.

Simple volume: là các đối tượng đại diện cho các sector từ một partition đơn, mà các trình điều khiển hệ thống file, quản lý nó như một đơn vị.

Multipartition volume: là các đối tượng đại diện cho các sector từ nhiều partition khác nhau, mà các trình điều khiển hệ thống file quản lý nó như một đơn vị. Các multipartition volume có các đặc tính mà các simple volume không có được như: hiệu suất cao, độ tin cậy cao và khả năng mở rộng kích thước.

Metadata: là một dạng dữ liệu đặc biệt, được lưu trữ trên đĩa, nó hỗ trợ cho các thành phần quản lý các dạng thức hệ thống file khác nhau, dữ liệu của nó có thể là vị trí của các tập tin/thư mục trên ổ đĩa. Metadata không được sử dụng trong các ứng dụng.

File system (hệ thống file): Các dạng thức hệ thống file định nghĩa cách mà dữ liệu file được lưu trữ trên thiết bị lưu trữ và sự tác động của hệ thống file đến các file. Một dạng thức hệ thống file cũng có thể đưa ra các giới hạn về kích thước của các file và các thiết bị lưu trữ mà hệ thống file hỗ trợ. Một vài hệ thống file hỗ trợ cho các file lớn nhỏ, hoặc cả các đĩa lớn và nhỏ.

Một hệ thống file thường bao gồm các thành phần: Sector khởi động, bảng định vị File, bảng thư mục gốc, một tập các file các thư mục và các công cụ quản lý các thành phần này. Các thành phần này có thể có cấu trúc hoặc phương thức tổ chức khác nhau trên các dạng hệ thống file khác nhau. Người ta thường dùng tên của FAT trong hệ thống File để gọi tên của hệ thống file đó.

Hệ điều hành MS_DOS sử dụng hệ thống File FAT 12 và FAT16, hệ điều hành Windows9x sự dụng hệ thống file FAT32 và CDFS, hệ điều hành WindowsNT và Windows 2000 sử dụng các hệ thống file FAT12, FAT16, FAT32, CDFS (CD_ROM File System) UDF (Universal Disk Format) và NTFS(New Technology File System)

- Format

Format cấp thấp (low format) định dạng lại các track, sector, cylinder

Format thông thường: định dạng mức cao (high-level format)

Format nhanh (xóa các kí tự lưu trữ đầu tiên của hdh hay phần mềm)

Format thường (Xóa dữ liệu và kiểm tra khối hư hỏng (bad block))

Đĩa mềm

Mặt	Rãnh	Sector	ý nghĩa
0	0	1	Bootsector
0	0	2,3	FAT1(File Allocation Table)
0	0	4, 5	FAT2(dành trường hợp FAT1 hỏng)
0	0	6,7,8,9	Root directory
1	0	1,2,3	Root directory

Đĩa cứng

Mặt	Rãnh	Sector	ý nghĩa
0	0	1	Bootsector
1	0	1	Cung khởi động

Bảng các phân khu được tạo

Offset	Nội dung	Kích thước
1BEh	Partition1 entry	16 byte
1CEh	Partition1 entry	16 byte
1DEh	Partition1 entry	16 byte
1EEh	Partition1 entry	16 byte

Nội dung 16 byte

địa chỉ	Kích thước	Nội dung
00	1 byte	địa chỉ khởi động
01	3 byte	địa chỉ đầu phân khu
05	3 byte	địa chỉ cuối phân khu
08	4 byte	Số cung trước phân khu
0C	4 byte	Số cung trong phân khu
04	1 byte	Chỉ thị hệ thống

Quản lý không gian đĩa

Để tổ chức lưu trữ nội dung các file trên đĩa, các hệ điều hành đều chia không gian lưu trữ của đĩa thành các phần có kích thước bằng nhau được gọi là khối(block) lưu trữ. Nội dung của file cũng được chia thành các block có kích thước bằng nhau, trừ block cuối cùng, và bằng các kích thước block đĩa. Khi cần lưu trữ File trên đĩa hệ điều hành cấp cho mỗi tập tin một số lượng block vừa đủ để chứa hết nội dung của tập tin, các block đĩa này có thể nằm tại vị trí bất kỳ trên đĩa. Trong quá trình sử dụng file kích thước của file có thể thay đổi, tăng lên hay giảm xuống, do hệ điều hành phải tổ chức cấp phát động các block đĩa cho các file. Khi kích thước của file tăng lên thì hệ điều hành phải cấp phát them block cho nó, khi kích thước file giảm xuống hoặc khi file bị xóa khỏi đĩa thì hệ điều hành phải thu hồi lại các block đĩa đã cấp cho nó để cấp cho các file khác sau này.

Offset	size	ý nghĩa
00h	3 byte	lệnh JUMP, nhẩy về Bootstrap Loader
03h	8 byte	Tên nhà sản xuất và số phiên bản
0Bh	2 byte	Số byte trên một sector
0Dh	1 byte	Số sector trên một cluster
0Eh	2 byte	Số sector dành cho bootsector
10h	1 byte	Số bảng FAT
11h	2 byte	Số phần tử trong thư mục gốc
13h	2 byte	Chỉ tổng số sector trên một tập đĩa (volume)
15h	1byte	Nhận khuôn dạng đĩa (F8: đĩa cứng)
16h	2 byte	Số sector dành cho bảng FAT
18h	2 byte	Số sector trên một track
1Ah	2 byte	Số mặt (đầu từ)
1Ch	4 byte	Số sector dự trữ
1Eh	4 byte	Số sector nếu kích thước >32Mb
22h	1 byte	Số hiệu ổ đĩa (đĩa mềm:0, đĩa cứng: 80)
23h	1 byte	Byte dự trữ
24 h	1 byte	chữ ký của Bootsector mở rộng
25h	4 byte	Số serial của đĩa được tạo ra lúc format
29h	11 byte	Nhãn đĩa
34h	8 byte	Loại FAT: “FAT 12” hoặc “FAT 16”
3Ch-200h	452 byte	Code của chương trình bootstrap loader

VD Bootsector đĩa cứng

EB 3C 90 4D 53 57 49 4E 34 2E 31 M S W I N 4 1 0 0 0 2

Như vậy, ngay sau khi quyền điều khiển được trả về cho bootsector thì hệ thống thì hệ thống sẽ thực hiện lệnh nhẩy (Jmp) ở đầu bootsector (offset 00), để nhảy đến thực hiện đoạn code bootstrap loader ở cuối boot sector (từ offset 3Ch đến offset 200h). Và bootstrap loader sẽ thực hiện nhiệm vụ của nó.

b) FAT

Nội dung của một File cần lưu trữ trên đĩa được chia thành các phần có kích thước bằng nhau và bằng kích thước của một cluster, được gọi là các block file. Các block file của các file được lưu trữ tại các cluster xác định trên đĩa, các cluster chứa nội dung một file có thể không nằm kề nhau. Để theo dõi danh sách các cluster đang chứa nội dung của một file của tất cả các file đang lưu giữ trên đĩa hệ điều hành DOS dùng bảng FAT, hay còn gọi là bảng định vị File. Bảng Fat còn dùng để ghi nhận trạng thái của các cluster trên đĩa: còn trống, đã cấp phát cho các file, bị bad không thể sử dụng hay dành riêng vào hệ điều hành. Trong quá trình khởi động máy tính hệ điều hành nạp bảng FAT vào bộ nhớ để chuẩn bị cho việc đọc/ghi các file sau này.

Khi cần ghi nội dung của một file vào đĩa hoặc đọc nội dung của một file trên đĩa hệ điều hành phải dựa vào bảng FAT, nếu bảng FAT bị hỏng thì hệ điều hành không thể ghi/đọc các file trên đĩa. Do đó, hệ điều hành DOS tạo ra hai bảng FAT hoàn toàn giống nhau là FAT1, FAT2, DOS sử dụng bảng FAT1 và dự phòng FAT2, nếu FAT1 bị hỏng thì DOS sẽ dùng FAT2 để khôi phục lại FAT1. Điều này không đúng với hệ thống file FAT32, FAT32 vẫn tạo ra 2 FAT của DOS, nhưng nếu FAT1 bị hỏng thì hệ điều hành sẽ chuyển sang sử dụng FAT2, sau đó mới khôi phục FAT1, và ngược lại.

Hệ điều hành DOS tổ chức cấp phát động các cluster các cluster cho các file trên đĩa, sau mỗi thao tác cấp phát/thu hồi cluster thì hệ điều hành phai cập nhật nội dung cho cả FAT1 và FAT2. Có thể hệ điều hành chỉ thực hiện cấp phát động cluster cho các file dữ liệu (có kích thước thay đổi), còn đối với các file chương trình, file thư viện, file liên kết động…(có kích thước không thể thay đổi) thì hệ điều hành sẽ thực hiện cấp tĩnh cluster cho nó.

Bảng FAT bao gồm nhiều phần tử (điểm nhập/mục vào), các phần tử được đánh địa chỉ bắt đầu từ 0 để phân biệt. Giá trị dữ liệu tại một phần tử trong bảng FAT cho biết trạng thái của một cluster tương ứng trên vùng dữ liệu. Hệ điều hành DOS có thể định dạng hệ thống File theo một trong 2 loại FAT là FAT12 và FAT16. Mỗi phần tử trong FAT12 rộng 12bit (1.5 byte), mỗi phần tử trong FAT16 rộng 16 bit(2 byte)

FAT 12	FAT 16	Ý nghĩa
FF7h	FFF7h	Cluster tương ứng bị hỏng
FF8h-FFFh	FFF8h-FFFFh	Cluster cuối cùng trong dãy các cluster chứa file
000h	0000h	Cluster tương ứng trống
FF0h- FF6h	FFF0h-FFF6h	Cluster tương ứng dành riêng cho hệ điều hành.
002h-FFEh	0002h-FFFEh	Đây là số hiệu của cluster trong bảng FAT, nó cho biết cluster tiếp theo trong dãy các cluster chứa nội dung một file.

Trong bảng FAT, hai phần tử đầu tiên (00 và 01) không dùng cho việc theo dõi trạng thái cluster và ghi nhận bản đồ cấp phát file, mà nó được sử dụng để chứa một giá trị nhận biết khuôn dạng đĩa, được gọi là byte định danh của đĩa, đây là byte đầu tiên trong bảng FAT. Đối với đĩa cứng thì byte ID=F8h.

Như vậy để đọc được nội dung của một file trên đĩa thì trước hết hệ điều hành phải tìm được dãy các cluster chứa nội dung của một file. Nhưng bảng Fat chỉ cho biết số hiệu các cluster từ cluster thứ hai đến cluster cuối cùng trong dãy nói trên. Cluster

đầu tiên trong dãy các cluster chứa nội dung của một file trên đĩa được tìm thấy trong bảng thư mục gốc.

Lưu giữ File theo FAT

Các phần tử trong bảng FAT tạo thành danh sách móc nối và phần tử cuối cùng của danh sách có giá trị FFF(FFFF). Vì các phần tử tạo thành danh sách móc nối nên chúng không nhất thiết phải nằm cạnh nhau.

Ví dụ : Tệp f1.txt có giá trị Starting cluster=6

0	FF0
1	FFF
2	
3	4
4	8
5	FFF
6	9
7	5
8	7
9	3

Tệp được lưu ở các cluster sau:

6→9→3→4→8→7→5

Thao tác đọc File như trên của DOS là kém hiệu quả, vì ngoài việc đọc nội dung của file tại các cluster trên vùng data của đĩa, hệ điều hành còn phải đọc và phân tích bảng FAT để dò tìm ra dãy các cluster chứa nội dung của một file. Hệ thống File NTFS trong WindowsNT/2000 khắc phục điều này bằng cách lưu danh sách các cluster chứa nội dung của một file vào một vị trí cố định nào đó, nên khi dọc file hệ điều hành chỉ cần đọc nội dung của các cluster trên đĩa theo danh sách ở trên, mà không phải tốn thời gian cho việc dò tìm dãy các cluster chứa nội dung của file của hệ thống file FAT trong DOS.

c) Root Directory (DIR)

Để quản lý thông tin của các file và các thư mục (thư mục con của thư mục gốc) đang được lưu trữ trên thư mục gốc của đĩa mềm hoặc đĩa logic trên đĩa cứng, hệ điều hành DOS sử dụng bảng thư mục gốc.

Bảng thư mục gốc gồm nhiều phần tử, số lượng phần tử trong bảng thư mục gốc được DOS quy định trước trong quá trình format đĩa và được ghi tại word tại offset 11h trong bootsector, giá trị này không thể thay đổi. Do đó tổng số file và thư mục con mà người sử dụng có thể chứ trên thư mục gốc của đĩa là có giới hạn, đây là

một hạn chế của DOS. Trong hệ thống file FAT32 và NTFS số phần tử trong bảng thư mục gốc không bị giới hạn, có thể thay đổi được và có thể được định vị tại một vị trí bất kỳ trên đĩa hoặc chứa trong một tập tin nào đó.

Mỗi phần tử trong bảng thư mục gốc dùng để chứa thông tin về một file hay thư mục nào đó đang được lưu trên thư mục gốc của đĩa. Khi có một file hoặc một thư mục nào được tạo ra trên thư mục gốc của đĩa thì hệ điều hành dùng một phần tử trong bảng thư mục gốc để chứa các thông tin liên quan của nó, khi một file hoặc một thư mục bị xóa/di chuyển khỏi thư mục gốc thì hệ điều hành sẽ thu hồi lại phần tử này để chuẩn bị cấp cho các file thư mục khác sau này.

Mỗi phần tử trong thư mục gốc dài 32 byte, chứa các thông tin sau:

Offset	Nội dung	Độ lớn
00h	Tên chính của file	8 byte
08h	Phần mở rộng của tên file	3 byte
0Bh	Thuộc tính file	1 byte
0Ch	Dự trữ, chưa được sử dụng	10 byte
16h	Giờ thay đổi tập tin cuối cùng	2 byte
18h	Ngày thay đổi tập tin cuối cùng	2 byte
1Ah	Cluster đầu tiên của file	2 byte
1Ch	Kích thước của file	4 byte

Thuộc tính của File

7	6	5	4	3	2	1	0
Dự trữ	Dự trữ	Archive	Directory	volum	system	hiden	readonly

d) Data area

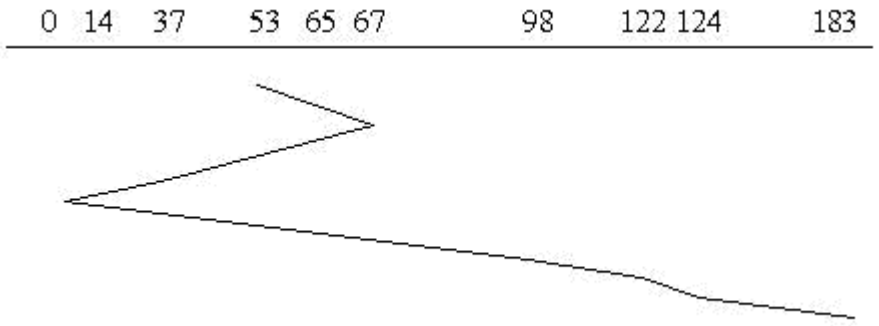
b)Lập lịch SSTF (shortest-seek-time-first)

Thuật toán này sẽ di chuyển đầu đọc đến các khối cần thiết theo vị trí lần lượt gần với vị trí hiện hành của đầu đọc nhất. Ví dụ : cần đọc các khối như sau :

98, 183, 37, 122, 14, 124, 65, và 67

Giả sử hiện tại đầu đọc đang ở vị trí 53. Như vậy đầu đọc lần lượt đi qua các khối

53, 65, 67, 37, 14, 98, 122, 124 và 183 như hình sau :



Hình 4.12 Phương pháp SSTF

Với ví dụ này, thuật toán SSTF làm giảm số khối mà đầu đọc phải di chuyển là 208 khối.

***c)*Lập lịch SCAN**

Theo thuật toán này, đầu đọc sẽ di chuyển về một phía của đĩa và từ đó di chuyển qua phía kia. Ví dụ : cần đọc các khối như sau :

98, 183, 37, 122, 14, 124, 65, và 67

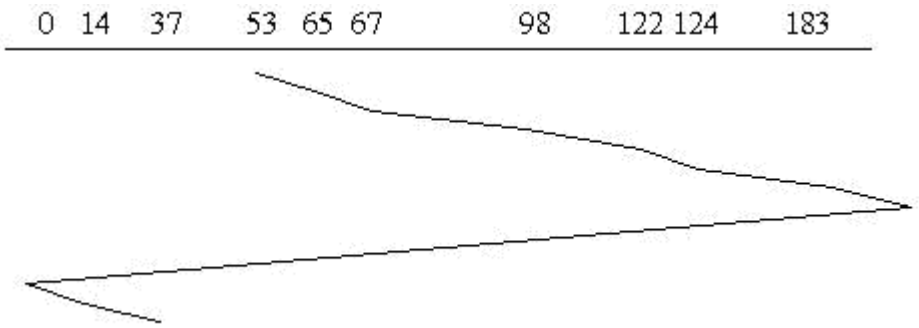
Giả sử hiện tại đầu đọc đang ở vị trí 53. Như vậy đầu đọc lần lượt đi qua các khối

53, 37, 14, 0 , 65, 67, 98, 122, 124 và 183 như hình sau :

Thuật toán này còn được gọi là thuật toán thang máy. Hình ảnh thuật toán giống như hình ảnh của một người quét tuyết, hay quét lá.

d)Lập lịch C-SCAN

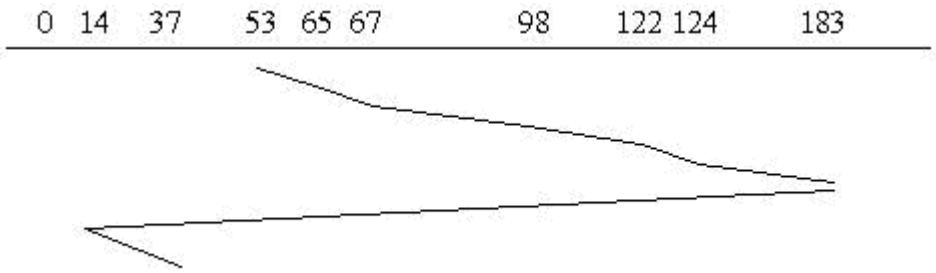
Thuật toán này tương tự như thuật toán SCAN, chỉ khác là khi nó di chuyển đến một đầu nào đó của đĩa, nó sẽ lập tức trở về đầu bắt đầu của đĩa. Lấy lại ví dụ trên, khi đó thứ tự truy xuất các khối sẽ là : 53, 65, 67, 98, 122, 124, 183, 199, 0, 14, 37 như hình sau :



Hình 4.13 Phương pháp C-SCAN

e)Lập lịch LOOK:

Nhận xét rằng cả hai thuật toán lập lịch SCAN và C-SCAN luôn luôn chuyển đầu đọc của đĩa từ đầu này sang đầu kia. Nhưng thông thường thì đầu đọc chỉ chuyển đến khối xa nhất ở mỗi hướng chứ không đến cuối. Do đó SCAN và C-SCAN được chỉnh theo thực tế và gọi là lập lịch LOOK. Như hình sau :



Hình 4.14 Phương pháp LOOK

Lựa chọn thuật toán lập lịch :

Với những thuật toán lập lịch, vấn đề là phải lựa chọn thuật toán nào cho hệ thống. Thuật toán SSTF thì rất thông thường. Thuật toán SCAN và C-SCAN thích hợp cho những hệ thống phải truy xuất dữ liệu khối lượng lớn. Với bất kỳ thuật toán lập lịch nào, điều quan trọng là khối lượng về số và kiểu khối cần truy xuất. Ví dụ , nếu số khối cần truy xuất là liên tục thì FCFS là thuật toán tốt.

Quản lý lỗi

Đĩa là đối tượng mà khi truy xuất có thể gây nhiều lỗi. Một trong số các lỗi thường gặp là

Lỗi lập trình : yêu cầu đọc các sector không tồn tại.

Lỗi lập trình xảy ra khi yêu cầu bộ điều khiển tìm kiếm cylinder không tồn tại, đọc sector không tồn tại, dùng đầu đọc không tồn tại, hoặc vận chuyển vào và ra bộ nhớ không tồn tại. Hầu hết các bộ điều khiển kiểm tra các tham số và sẽ báo lỗi nếu không thích hợp.

Lỗi checksum tạm thời : gây ra bởi bụi trên đầu đọc.

Bụi tồn tại giữa đầu đọc và bề mặt đĩa sẽ gây ra lỗi đọc. Nếu lỗi tồn tại, khối có thể bị đánh dấu hỏng bởi phần mềm.

Lỗi checksum thường trực : đĩa bị hư vật lý trên các khối.

Lỗi tìm kiếm : ví dụ đầu đọc đến cylinder 7 trong khi đó phải đọc 6.

Lỗi điều khiển : bộ điều khiển từ chối thi hành lệnh.

truy xuất (đọc hoặc ghi) từng khối riêng biệt, và chương trình có thể truy xuất một khối bất kỳ nào đó. Đĩa là một ví dụ cho loại thiết bị khối.

- Một dạng thiết bị thứ hai là thiết bị tuần tự. Ở dạng thiết bị này, việc gửi và nhận thông tin dựa trên là chuỗi các bits, không có xác định địa chỉ và không thể thực hiện thao tác seek được. Màn hình, bàn phím, máy in, card mạng, chuột, và các loại thiết bị khác không phải dạng đĩa là thiết bị tuần tự.

Việc phân chia các lớp như trên không hoàn toàn tối ưu, một số các thiết bị không phù hợp với hai lớp trên, ví dụ : đồng hồ, bộ nhớ màn hình v.v...không thực hiện theo cơ chế tuần tự các bits. Ngoài ra, người ta còn phân loại các thiết bị I/O dưới một tiêu chuẩn khác :

- Thiết bị tương tác được với con người : dùng để giao tiếp giữa người và máy. Ví dụ : màn hình, bàn phím, chuột, máy in ...

- Thiết bị tương tác trong hệ thống máy tính là các thiết bị giao tiếp với nhau. Ví dụ : đĩa, băng từ, card giao tiếp...

- Thiết bị truyền thông : như modem...

Những điểm khác nhau giữa các thiết bị I/O gồm :

Tốc độ truyền dữ liệu , ví dụ bàn phím : 0.01 KB/s, chuột 0.02 KB/s ...

Công dụng.

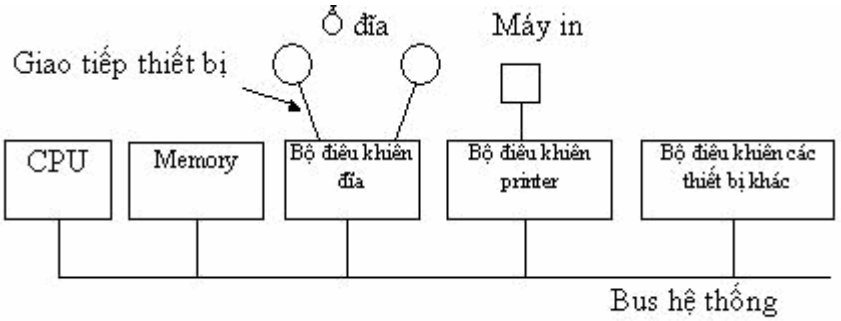
Đơn vị truyền dữ liệu (khối hoặc ký tự).

Biểu diễn dữ liệu, điều này tùy thuộc vào từng thiết bị cụ thể.

Tình trạng lỗi : nguyên nhân gây ra lỗi, cách mà chúng báo về...

Một thiết bị giao tiếp với một hệ thống máy tính bằng cách gửi các tín hiệu qua dây cáp hay thậm chi qua không khí. Các thiết bị giao tiếp với máy tính bằng một điểm nối kết(cổng-port) như cổng tuần tự. Nếu một hay nhiều thiết bị dung một tập hợp dây dẫn, nối kết được gọi là bus. Một bus là một tập hợp dây dẫn và giao thức được định nghĩa chặt chẽ để xác định tập hợp thông điệp có thể được gửi qua dây. Trong thuật ngữ điện tử, các thông điệp được truyền bởi các mẫu điện thế điện tử được áp dụng tới các dây dẫn với thời gian được xác định. Khi thiết bị A có một cáp gán vào thiết bị B, thiết bị B có một cáp gán vào thiết bị C và thiết bị C gán vào một cổng máy tính, sự

Giao tiếp giữa bộ điều khiển và thiết bị là giao tiếp ở mức thấp.



Hình 5.1 Sự kết nối giữa CPU, bộ nhớ, bộ điều khiển và các thiết bị nhập xuất

Chức năng của bộ điều khiển là giao tiếp với hệ điều hành vì hệ điều hành không thể truy xuất trực tiếp với thiết bị. Việc thông tin thông qua hệ thống đường truyền gọi là bus.

Công việc của bộ điều khiển là chuyển đổi dãy các bit tuần tự trong một khối các byte và thực hiện sửa chửa nếu cần thiết. Thông thường khối các byte được tổ chức thành từng bit và đặt trong buffer của bộ điều khiển. Sau khi thực hiện checksum nội dung của buffer sẽ được chuyển vào bộ nhớ chính. Ví dụ : bộ điều khiển cho màn hình đọc các byte của ký tự để hiển thị trong bộ nhớ và tổ chức các tín hiệu để điều khiển các tia của CRT để xuất trên màn ảnh bằng cách quét các tia dọc và ngang. Nếu không có bộ điều khiển, lập trình viên hệ điều hành phải tạo thêm chương trình điều khiển tín hiệu analog cho đèn hình. Với bộ điều khiển , hệ điều hành chỉ cần khởi động chúng với một số tham số như số ký tự trên một dòng, số dòng trên màn hình và bộ điều khiển sẽ thực hiện điều khiển các tia.

Mỗi bộ điều khiển có một số thanh ghi để liên lạc với CPU. Trên một số máy tính, các thanh ghi này là một phần của bộ nhớ chính tại một địa chỉ xác định gọi là ánh xạ bộ nhớ nhập xuất. Hệ máy PC dành ra một vùng địa chỉ đặc biệt gọi là địa chỉ nhập xuất và trong đó được chia làm nhiều đoạn, mỗi đoạn cho một loại thiết bị như sau :

Bộ điều khiển nhập/xuất	Địa chỉ nhập/xuất	Vectơ ngắt
Đồng hồ	040 - 043	8
Bàn phím	060 - 063	9
RS232 phụ	2F8 - 2FF	11
Đĩa cứng	320 - 32F	13
Máy in	378 - 37F	15

spooling device là line printer. Spooling còn được sử dụng trong hệ thống mạng như hệ thống e-mail chẳng hạn.

.bak, .bas, .bin, .c, .dat, .doc, .ftn, .hlp, .lib, .obj, .pas, .tex, .txt.

Trên thực tế phần mở rộng có hữu ích trong một số trường hợp, ví dụ như có những trình dịch C chỉ nhận biết các tập tin có phần mở rộng là .C

Loại File: được thể hiện ở phần mở rộng và cách thực hiện trên file

Loại File	Phần mở rộng	Chức năng
Excutable	Exe, com, bin	sẵn sàng để chạy ngôn ngữ máy
Object	Obj,o	dịch ngôn ngữ máy, không liên kết
Source code	C, pas, asm	Mã nguồn
Batch	Bat, sh	xử lý theo lô
Text	Txt, doc	đocữ liệu văn bản, tài liệu
Library	Lib,a	Thư viện
Print or view	Ps, pdf, gif	In ấn và hiển thị
Archive	Arc, zip, tar	Nhóm các file trong một file, lưu giữ.

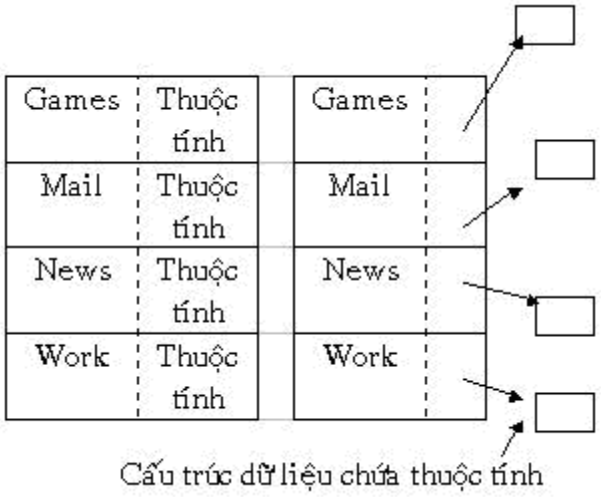
Ngoài tên và dữ liệu, hệ điều hành cung cấp thêm một số thông tin cho tập tin gọi là thuộc tính.

Các thuộc tính thông dụng trong một số hệ thống tập tin :

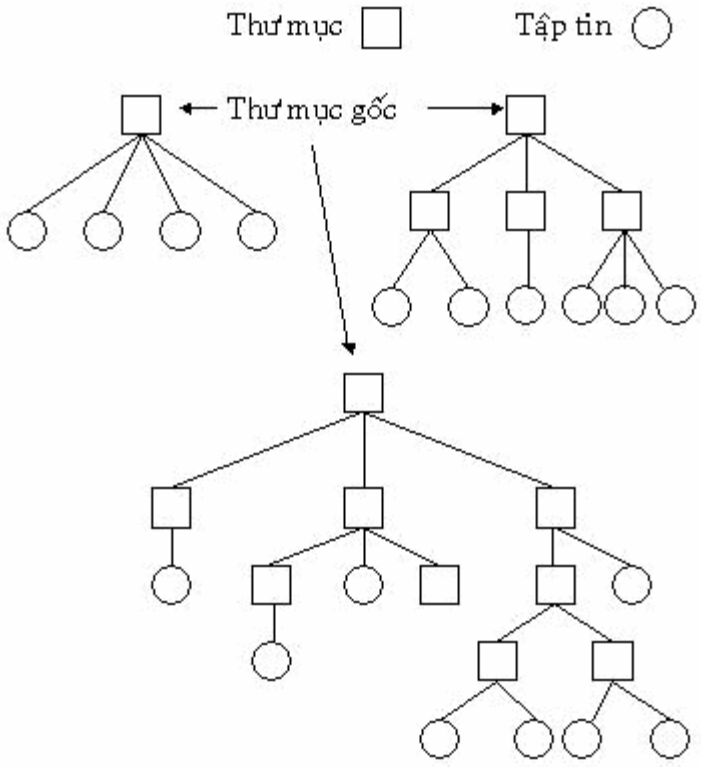
Tên thuộc tính	Ý nghĩa
Bảo vệ	Ai có thể truy xuất được và bằng cách nào
Mật khẩu	Mật khẩu cần thiết để truy xuất tập tin
Người tạo	Id của người tạo tập tin
Người sở hữu	Người sở hữu hiện tại
Chỉ đọc	0 là đọc ghi, 1 là chỉ đọc
ẩn	0 là bình thường, 1 là không hiển thị khi liệt kê
Hệ thống	0 là bình thường, 1 là tập tin hệ thống
Lưu trữ	0 đã được backup, 1 cần backup
ASCII/binary	0 là tập tin văn bản, 1 là tập tin nhị phân
Truy xuất ngẫu nhiên	0 truy xuất tuần tự, 1 là truy xuất ngẫu nhiên
Temp	0 là bình thường, 1 là bị xóa khi tiến trình kết thúc
Khóa	0 là không khóa, khác 0 là khóa
Độ dài của record	Số byte trong một record
Vị trí khóa	Offset của khóa trong mỗi record
Giờ tạo	Ngày và giờ tạo tập tin
Thời gian truy cập	Ngày và giờ truy xuất tập tin gần nhất

Từ đó, hệ thống thư mục theo cấp bậc (còn gọi là cây thư mục) được hình thành với mô hình một thư mục có thể chứa tập tin hoặc một thư mục con và cứ tiếp tục như vậy hình thành cây thư mục như trong các hệ điều hành DOS, Windows, v. v...

Ngoài ra, trong một số hệ điều hành nhiều người dùng, hệ thống còn xây dựng các hình thức khác của cấu trúc thư mục như cấu trúc thư mục theo đồ thị có chu trình và cấu trúc thư mục theo đồ thị tổng quát. Các cấu trúc này cho phép các người dùng trong hệ thống có thể liên kết với nhau thông qua các thư mục chia sẻ.



Hình 6.1



Hình 6.2 Hệ thống thư mục theo cấp bậc

Đường dẫn :

Khi một hệ thống tập tin được tổ chức thành một ***cây thư mục***, có hai cách để xác định một tên tập tin. Cách thứ nhất là ***đường dẫn tuyệt đối***, mỗi tập tin được gán một đường dẫn từ thư mục gốc đến tập tin. Ví dụ : /usr/ast/mailbox.

Dạng thứ hai là ***đường dẫn tương đối***, dạng này có liên quan đến một khái niệm là ***thư mục hiện hành*** hay thư mục làm việc. Người sử dụng có thể quy định một thư mục là thư mục hiện hành. Khi đó đường dẫn không bắt đầu từ thư mục gốc mà liên quan đến thư mục hiện hành. Ví dụ, nếu thư mục hiện hành là /usr/ast thì tập tin với đường dẫn tuyệt đối /usr/ast/mailbox có thể được dùng đơn giản là mailbox.

Trong phần lớn hệ thống, mỗi tiến trình có một thư mục hiện hành riêng, khi một tiến trình thay đổi thư mục làm việc và kết thúc, không có sự thay đổi để lại trên hệ thống tập tin. Nhưng nếu một hàm thư viện thay đổi đường dẫn và sau đó không đổi lại thì sẽ có ảnh hưởng đến tiến trình.

Hầu hết các hệ điều hành đều hỗ trợ hệ thống thư mục theo cấp bậc với hai entry đặc biệt cho mỗi thư mục là "." và "..". "." chỉ thư mục hiện hành, ".." chỉ thư mục cha.

Các thao tác trên thư mục :

Tạo : một thư mục được tạo, nó rỗng, ngoại trừ "." và ".." được đặt tự động bởi hệ thống.

Xóa :xoá một thư mục, chỉ có thư mục rỗng mới bị xóa, tư mục chứa "." và ".." coi như là thư mục rỗng.

Mở thư mục :thư mục có thể được đọc. Ví dụ để liệt kê tất cả tập tin trong một thư mục, chương trình liệt kê mở thư mục và đọc ra tên của tất cả tập tin chứa trong đó. Trước khi thư mục được đọc, nó phải được mở ra trước.

Đóng thư mục :khi một thư mục đã được đọc xong, phải đóng thư mục để giải phóng vùng nhớ.

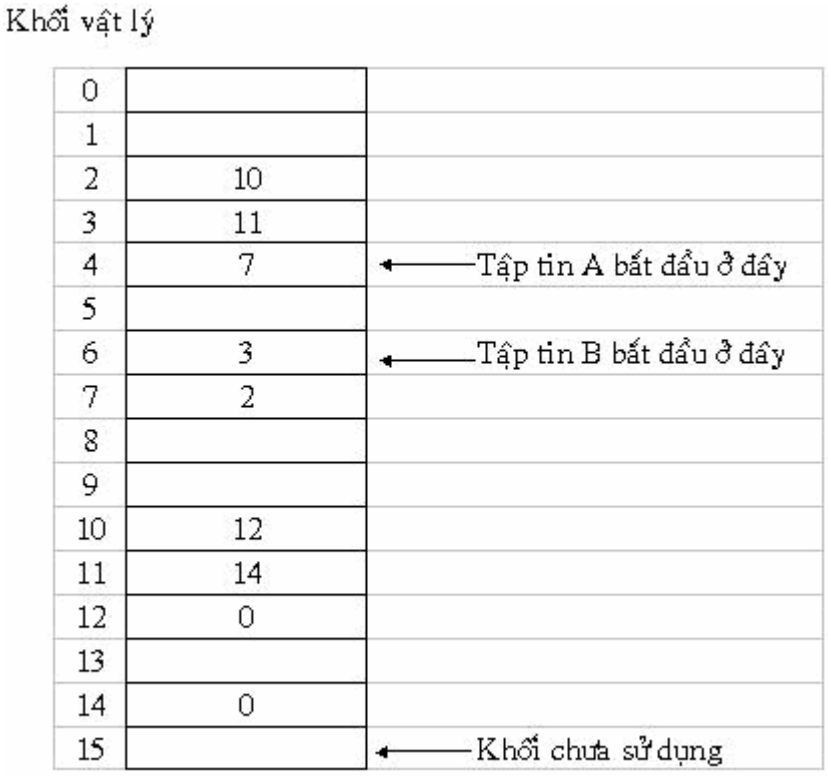
Đọc thư mục :Lệnh này trả về entry tiếp theo trong thư mục đã mở. Thông thường có thể đọc thư mục bằng lời gọi hệ thống READ, lệnh đọc thư mục luôn luôn trả về một entry dưới dạng chuẩn .

Mọi khối đều được cấp phát, không bị lãng phí trong trường hợp phân mảnh và directory entry chỉ cần chứa địa chỉ của khối đầu tiên.

Tuy nhiên khối dữ liệu bị thu hẹp lại và truy xuất ngẫu nhiên sẽ chậm.

Danh sách liên kết sử dụng index :

Tương tự như hai nhưng thay vì dùng con trỏ thì dùng một bảng index. Khi đó toàn bộ khối chỉ chứa dữ liệu. Truy xuất ngẫu nhiên sẽ dễ dàng hơn. Kích thước tập tin được mở rộng hơn. Hạn chế là bản này bị giới hạn bởi kích thước bộ nhớ .



Hình 6.4 Bảng chỉ mục của danh sách liên kết

I-nodes :

Một I-node bao gồm hai phần. Phần thứ nhất là thuộc tính của tập tin. Phần này lưu trữ các thông tin liên quan đến tập tin như kiểu, người sở hữu, kích thước, v.v...Phần thứ hai chứa địa chỉ của khối dữ liệu. Phần này chia làm hai phần nhỏ. Phần nhỏ thứ nhất bao gồm 10 phần tử, mỗi phần tử chứa địa chỉ khối dữ liệu của tập tin. Phần tử thứ 11 chứa địa chỉ gián tiếp cấp 1 (single indirect), chứa địa chỉ của một khối, trong khối đó chứa một bảng có thể từ 2^{10} đến 2^{32} phần tử mà mỗi phần tử mới chứa địa chỉ của khối dữ liệu. Phần tử thứ 12 chứa địa chỉ gián tiếp cấp 2 (double indirect),

Quản lý khối bị hỏng

Đĩa thường có những khối bị hỏng trong quá trình sử dụng đặc biệt đối với đĩa cứng vì khó kiểm tra được hết tất cả.

Có hai giải pháp : phần mềm và phần cứng.

Phần cứng là dùng một sector trên đĩa để lưu giữ danh sách các khối bị hỏng. Khi bộ kiểm soát tực hiện lần đầu tiên, nó đọc những khối bị hỏng và dùng một khối thừa để lưu giữ. Từ đó không cho truy cập những khối hỏng nữa.

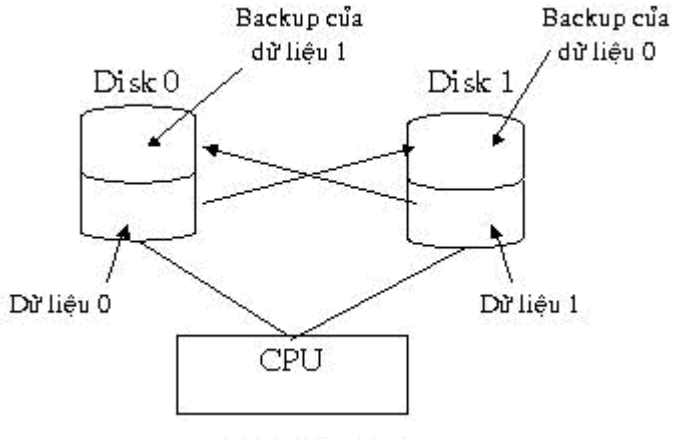
Phần mềm là hệ thống tập tin xây dựng một tập tin chứa các khối hỏng. Kỹ thuật này loại trừ chúng ra khỏi danh sách các khối trống, do đó nó sẽ không được cấp phát cho tập tin.

Backup

Mặc dù có các chiến lược quản lý các khối hỏng, nhưng một công việc hết sức quan trọng là phải backup tập tin thường xuyên.

Tập tin trên đĩa mềm được backup bằng cách chép lại toàn bộ qua một đĩa khác. Dữ liệu trên đĩa cứng nhỏ thì được backup trên các băng từ.

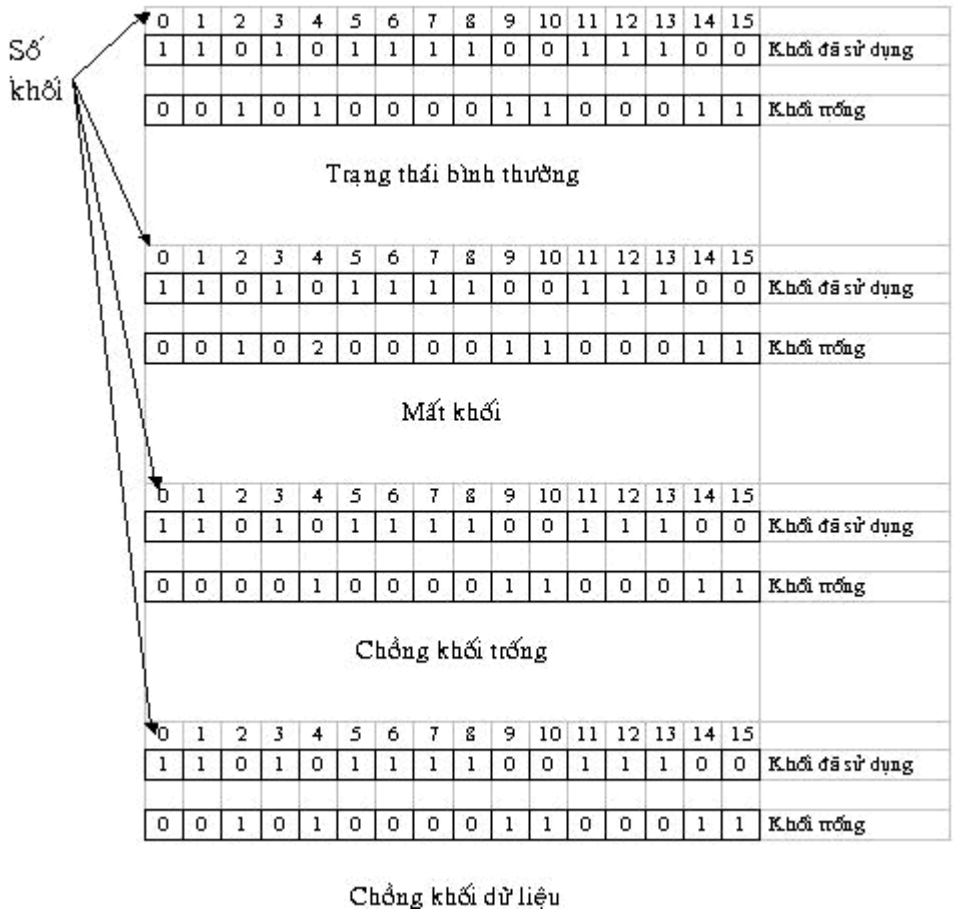
Đối với các đĩa cứng lớn, việc backup thường được tiến hành ngay trên nó. Một chiến lược dể cài đặt nhưng lãng phí một nữa đĩa là chia đĩa cứng làm hai phần một phần dữ liệu và một phần là backup. Mỗi tối, dữ liệu từ phần dữ liệu sẽ được chép sang phần backup.



Hình 6.7 Backup

Tính không đổi của hệ thống tập tin

Một vấn đề nữa về độ an toàn là ***tính không đổi***. Khi truy xuất một tập tin, trong quá trình thực hiện, nếu có xảy ra những sự cố làm hệ thống ngừng hoạt động đột ngột, lúc đó hàng loạt thông tin chưa được cập nhật lên đĩa. Vì vậy mỗi lân khởi động ,hệ thống sẽ thực hiện việc kiểm tra trên hai phần khối và tập tin. Việc kiểm tra thực hiện , khi phát hiện ra lỗi sẽ tiến hành sữa chữa cho các trường hợp cụ thể:



Hình 6.8 Trạng thái của hệ thống tập tin

Tài liệu tham khảo:

[1] "*Tập slide bài giảng*". Bộ môn Các hệ thống thông tin.

[2] "*Operating System Concepts*". A. Silberschatz, P. B. Galvin, G. Gagne, Wiley & Sons, 2002.

[3] "*Operating Systems*", H. M. Deitel, P. J. Deitel and D. R. Choffnes, Pearson Education International, 2004.

[4] "*Modern Operating Systems*". Andrew S. Tanenbaum, Prentice-Hall International, 2001.

[5] "*Operating Systems – Internals and Design Principles*". William Stallings, Pearson Education International, 2005.

Simpo PDF Merge and Split Unregistered Version - http://www.simpopdf.com

Nguyên lý hệ điều hành

Nguyễn Hải Châu
Khoa Công nghệ thông tin
Trường Đại học Công nghệ

1

Mục tiêu của môn học

- Cung cấp những khái niệm cơ bản về hệ điều hành máy tính: phân loại, nguyên lý, cách làm việc, phân tích thiết kế và chi tiết về một số hệ điều hành cụ thể
- Yêu cầu sinh viên: Nắm vững các nguyên lý cơ bản, làm tốt các bài tập để lấy đó làm cơ sở - nguyên lý cho các vấn đề khác trong thiết kế và cài đặt các hệ thống thông tin
- Chú ý liên hệ nội dung môn học với các tình huống thực tế về khía cạnh quản lý, tổ chức

2

Nội dung

- Gồm có 6 phần chính:
  - Tổng quan (3 tiết)
  - Quản lý tiến trình (12 tiết)
  - Quản lý lưu trữ (12 tiết)
  - Hệ vào/ra (9 tiết)
  - Bảo vệ và an ninh (6 tiết)
  - Hệ điều hành Linux (*optional*) + Ôn tập (3 tiết)

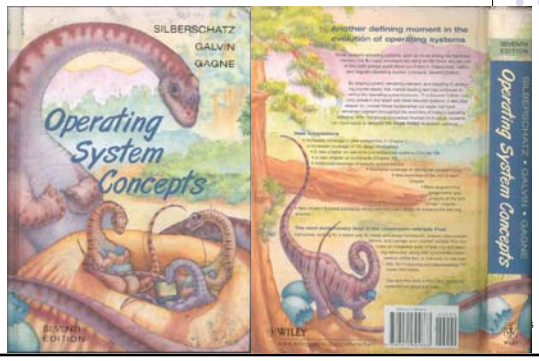
3

Tài liệu tham khảo

- **Abraham Silberschatz, Peter Baer Galvin, Greg Gagne, Operating System Concepts, 7th edition, John Wiley & Sons, Inc., 2005.**
- William Stallings, Operating Systems: Internals and Design Principles 5th edition, Prentice-Hall, 2005.
- Andrew S. Tanenbaum, Modern Operating Systems, 2nd edition, Prentice-Hall, 2001.
- Andrew S. Tanenbaum, Albert S Woodhull, Operating Systems: Design and Implementation, 3rd edition, Prentice-Hall. 2006. (Có mã nguồn kèm theo).
- Hà Quang Thụy, Nguyên lý hệ điều hành, NXB KHKT, 2002.
- Robert Love, Linux Kernel Development, Sams Publishing, 2003.
- Daniel P. Bovet, Marco Cesati, Understanding Linux Kernel, 2nd edition, O'Reilly & Associates, 2002.
- W. Richard Stevens, Advanced Programming in the UNIX Environment, Addison-Wesley, 1992.

4

Giáo trình



5

Bản điện tử của giáo trình

- Website của Bộ môn Các hệ thống thông tin: http://coltech.vnu.edu.vn/httt
- Chọn "Góc học tập" ở menu bên trái
- Chọn "Nguyên lý hệ điều hành" ở phần nội dung chính của trang web
- Download sách theo chỉ dẫn

6

Simpo PDF Merge and Split Unregistered Version - http://www.simpopdf.com

Thi và kiểm tra

- Kiểm tra giữa kỳ: viết, 45-60 phút
  - Là điều kiện bắt buộc để được thi cuối kỳ
  - Sau phần quản lý bộ nhớ/lưu trữ
  - Được sử dụng tài liệu
- Thi cuối kỳ:
  - Thi viết 60-90 phút
  - Được sử dụng tài liệu

7

Giới thiệu

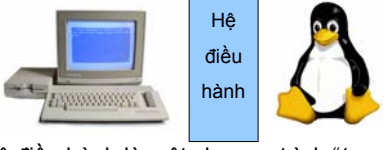
8

Máy tính - tài nguyên máy tính

- Tài nguyên:
  - CPU
  - Bộ nhớ trong
  - Đĩa cứng
  - Thiết bị ngoại vi (máy in, màn hình, bàn phím, card giao tiếp mạng, USB...)

9

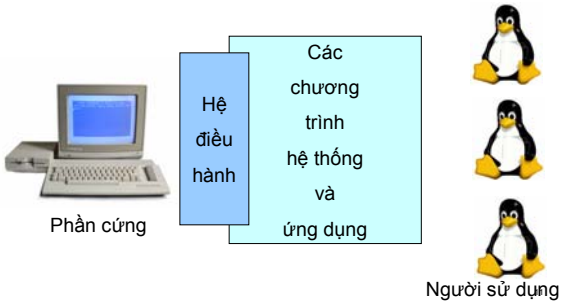
Hệ điều hành là gì?



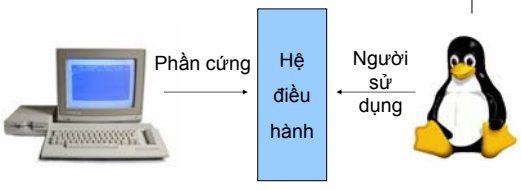
- Hệ điều hành là một chương trình "*trung gian*" (nhân – kernel) giữa NSD và máy tính :
  - *Quản lý* phần cứng máy tính (các tài nguyên)
  - Cung cấp cho NSD môi trường làm việc *tiện lợi* và *hiệu quả*

10

Hệ thống máy tính



Hai cách nhìn hệ điều hành



- Phần cứng: Quản lý & cấp phát tài nguyên để sử dụng tối đa năng lực phần cứng
- Người sử dụng: Dễ sử dụng, hiệu quả, ứng dụng phong phú

12

Simpo PDF Merge and Split Unregistered Version - http://www.simpopdf.com

Một số loại hệ điều hành

- Xử lý theo lô (batch processing)
- Đa chương trình (multiprogramming)
- Phân chia thời gian (time-sharing/multitasking)
- Hệ điều hành cho máy cá nhân
- Xử lý song song (parallel)
- Thời gian thực (real-time)
- Nhúng (embedded)
- Cầm tay (portable)
- Đa phương tiện (multimedia)
- Chuyên dụng (special-purpose)

13

Các hệ xử lý theo lô đơn giản

- Thuật ngữ: *Batch processing*
- Các chương trình được đưa vào hàng chờ
- Máy tính thực hiện tuần tự các chương trình của người sử dụng
- Chương trình không có giao tiếp với người sử dụng

14

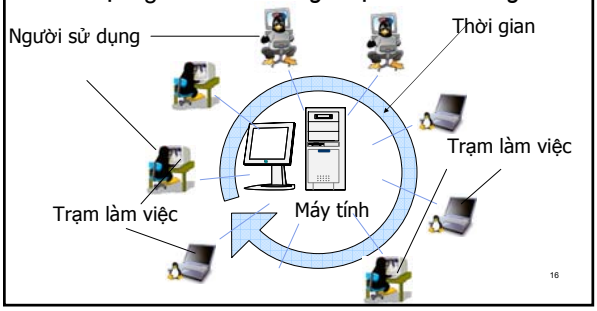
Đa chương trình

- Thuật ngữ: *Multiprogramming*
- Các chương trình được xếp hàng
- Một chương trình được thực hiện và chiếm giữ CPU cho đến khi (1) có yêu cầu vào/ra, hoặc (2) kết thúc
- Khi (1) hoặc (2) xảy ra, chương trình khác sẽ được thực hiện
- Tận dụng CPU tốt hơn xử lý theo lô đơn giản

15

Phân chia thời gian/đa nhiệm

- Thuật ngữ: *time-sharing* hoặc *multitasking*



16

Một số hệ điều hành

- UNIX (UNiplexed Information and Computing Service): (1) AT&T System V (2) Berkeley (BSD)
  - AIX dựa trên System V (IBM)
  - HP-UX dựa trên BSD (Hewlett-Packard)
  - IRIX dựa trên System V (Silicon Graphics Inc.)
  - Linux
  - Solaris, SunOS (Sun Microsystems)
  - Minix

17

Một số hệ điều hành

- Windows (Microsoft): Windows 3.x, Windows 95, Windows 98, Windows 2000, Windows NT, Windows XP, Windows Vista
- Mac OS, Mac OS X (Apple Inc.)
- BeOS
- OS 9
- OS/2
- DOS
- PalmOS, Symbian

18

Simpo PDF Merge and Split Unregistered Version - http://www.simpopdf.com

Cấu trúc hệ điều hành

19

Các thành phần của hệ thống

- Quản lý tiến trình
- Quản lý bộ nhớ trong
- Quản lý tệp
- Quản lý vào/ra
- Quản lý lưu trữ trên bộ nhớ ngoài
- Liên kết mạng
- Bảo vệ và an ninh
- Thông dịch lệnh

20

Các dịch vụ của hệ điều hành

- Giao diện với người sử dụng
- Thực hiện các chương trình
- Thực hiện các thao tác vào/ra
- Quản lý hệ thống tệp
- Truyền thông
- Phát hiện lỗi
- Cấp phát tài nguyên
- "Kế toán"
- Đưa ra các cơ chế bảo vệ và an ninh

21

Các hàm hệ thống

- Các *hàm hệ thống* (system calls) cung cấp giao diện lập trình tới các dịch vụ do hệ điều hành cung cấp
- Ví dụ trong hệ điều hành Unix:
  - Tạo một tiến trình mới: **fork();**
  - Thoát khỏi tiến trình đang thực hiện: **exit(1);**
  - **fork** và **exit** là các hàm hệ thống (Hàm HT)

22

Hàm HT điều khiển tiến trình

- Kết thúc tiến trình bình thường/bất thường
- Nạp, thực hiện tiến trình
- Tạo, kết thúc tiến trình
- Đọc hoặc thiết lập các thuộc tính cho tiến trình
- Yêu cầu tiến trình vào trạng thái chờ
- Cấp phát và giải phóng bộ nhớ
- Xử lý các sự kiện không đồng bộ

23

Hàm HT quản trị tệp

- Tạo, xóa tệp
- Đóng, mở tệp
- Đọc, ghi, định vị con trỏ tệp
- Đọc, thiết lập thuộc tính của tệp

24

Simpo PDF Merge and Split Unregistered Version - http://www.simpopdf.com

Hàm HT quản trị thiết bị

- Yêu cầu sử dụng hoặc thôi sử dụng thiết bị
- Đọc, ghi, định vị con trỏ
- Đọc, thiết lập thuộc tính cho thiết bị
- Attach/detach thiết bị về mặt logic

25

Hàm HT bảo trì thông tin

- Đọc, thiết lập thời gian hệ thống
- Đọc, ghi dữ liệu về hệ thống
- Đọc thuộc tính tệp, thiết bị, tiến trình
- Thiết lập thuộc tính tệp, thiết bị, tiến trình

26

Hàm HT về truyền thông

- Tạo, hủy các kết nối mạng
- Truyền nhận các thông điệp
- Lấy thông tin trạng thái truyền thông
- Attach/detach các thiết bị ở xa

27

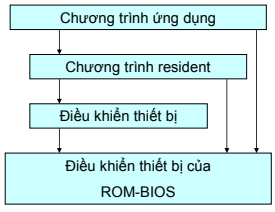
Các chương trình hệ thống

- Các chương trình hệ thống cung cấp môi trường thuận tiện cho việc thực hiện và phát triển chương trình. Chúng được phân loại như sau:
  - Thao tác với tệp
  - Thông tin về trạng thái của hệ thống
  - Sửa đổi tệp
  - Hỗ trợ ngôn ngữ lập trình
  - Nạp và thực hiện chương trình
  - Truyền thông
  - Cách nhìn HĐH của NSD được xác định qua các chương trình hệ thống, không thực sự qua các hàm hệ thống (system calls).

28

Cấu trúc HĐH: Đơn giản

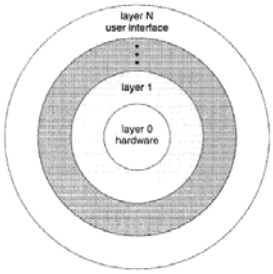
- Thuật ngữ: Simple approach
- Ví dụ MS-DOS. (tương tự: UNIX thời gian đầu)



29

Cấu trúc HĐH: Phân tầng

- Thuật ngữ: Layered apparoach

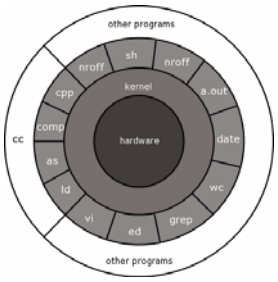


30

Simpo PDF Merge and Split Unregistered Version - http://www.simpopdf.com

Cấu trúc HĐH: Phân tầng

- Ví dụ UNIX



31

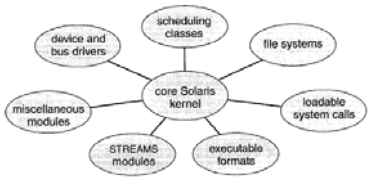
Cấu trúc HĐH: Vi nhân

- Thuật ngữ: Microkernel
- Giữ cho nhân có các đủ các chức năng thiết yếu nhất để giảm cỡ
- Các chức năng khác được đưa ra ngoài nhân
- Ví dụ: Mach, Tru64 UNIX, QNX

32

Cấu trúc HĐH: Module

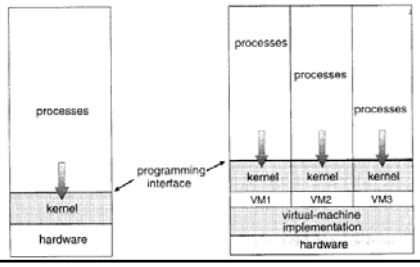
- Thuật ngữ: Module approach
- Hiện tại đây là cách tiếp cận tốt nhất (sử dụng được các kỹ thuật lập trình hướng đối tượng). Ví dụ: Solaris của Sun Microsystem:



33

Máy ảo

- Thuật ngữ (Virtual Machine)
- Ví dụ: VMware (sản phẩm thương mại)



Tóm tắt

- Khái niệm HĐH, nhân
- Hai cách nhìn HĐH từ NSD và hệ thống
- Các khái niệm xử lý theo lô, đa chương trình và phân chia thời gian
- Các thành phần và dịch vụ của HĐH
- Các hàm hệ thống
- Một số cấu trúc phổ biến của HĐH
- Máy ảo

35

Tìm hiểu thêm

- **Không bắt buộc**
- Bổ sung một hàm hệ thống mới vào nhân Linux và sử dụng hàm đó:
  - Đọc hướng dẫn trong giáo trình từ trang 74-78
  - Thử nghiệm trên RedHat Fedora hoặc Ubuntu/Debian

36

Simpo PDF Merge and Split Unregistered Version - http://www.simpopdf.com

Nguyên lý hệ điều hành

Nguyễn Hải Châu
Khoa Công nghệ thông tin
Trường Đại học Công nghệ

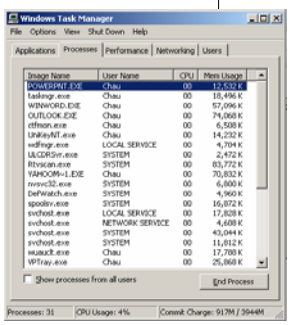
1

Khái niệm tiến trình

2

Tiến trình là gì?

- Thuật ngữ: Process (tiến trình/quá trình)
- Là một *chương trình đang được thực hiện*
- Được xem là đơn vị làm việc trong các HĐH
- Có hai loại tiến trình:
  - Tiến trình của HĐH
  - Tiến trình của NSD

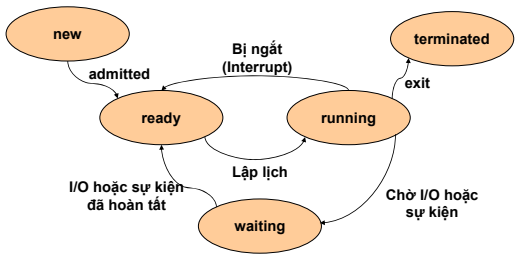


Tiến trình gồm có…

- Đoạn mã lệnh (code, có sách gọi là text)
- Đoạn dữ liệu
- Đoạn ngăn xếp và heap (stack/heap)
- Các *hoạt động hiện tại* được thể hiện qua con đếm lệnh (IP) và nội dung các thanh ghi (registers) của bộ xử lý
- Chú ý:
  - Tiến trình là *thực thể chủ động*
  - Chương trình là *thực thể bị động*

4

Trạng thái tiến trình



5

Khối điều khiển tiến trình

- **Thuật ngữ: Process Control Block (PCB)**
- **Các thông tin:**
  - **Trạng thái tiến trình**
  - **Con đếm**
  - **Các thanh ghi**
  - **Thông tin về lập lịch**
  - **Thông tin về bộ nhớ**
  - **Thông tin accounting**
  - **Thông tin vào/ra**

Con trỏ	Trạng thái tiến trình
Số hiệu tiến trình (Process number)	
Con đếm (program counter)	
Các thanh ghi (registers)	
Giới hạn bộ nhớ	
Danh sách các tệp đang mở	
.....	

6

Simpo PDF Merge and Split Unregistered Version - http://www.simpopdf.com

Lập lịch tiến trình

7

Tại sao phải lập lịch?

- Số lượng NSD, số lượng tiến trình luôn lớn hơn số lượng CPU của máy tính rất nhiều
- Tại một thời điểm, chỉ có duy nhất một tiến trình được thực hiện trên một CPU
- Vấn đề:
  - Số lượng yêu cầu sử dụng nhiều hơn số lượng tài nguyên đang có (CPU)
  - Do đó cần lập lịch để phân phối thời gian sử dụng CPU cho các tiến trình của NSD và hệ thống

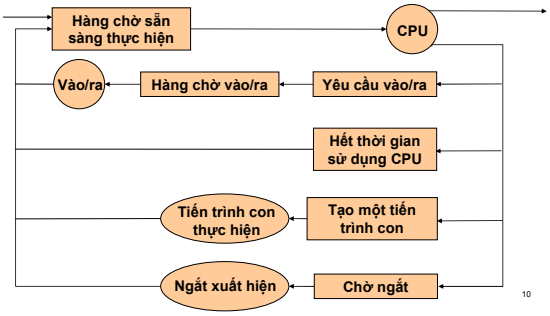
8

Hàng chờ lập lịch

- Thuật ngữ: Queue
- Các tiến trình chưa được phân phối sử dụng CPU sẽ được đưa vào *hàng chờ* (*queue*)
- Có thể có nhiều hàng chờ trong hệ thống: Hàng chờ sử dụng CPU, hàng chờ sử dụng máy in, hàng chờ sử dụng ổ đĩa CD…
- Trong suốt thời gian tồn tại, tiến trình phải di chuyển giữa các hàng chờ

9

Hàng chờ lập lịch tiến trình



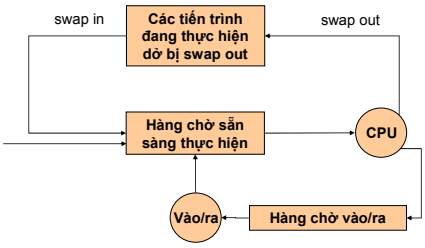
10

Phân loại các bộ lập lịch

- *Bộ lập lịch dài hạn* (long-term scheduler)
  - Thường dùng trong các hệ xử lý theo lô
  - Đưa tiến trình từ spool vào bộ nhớ trong
- *Bộ lập lịch ngắn hạn* (short-term scheduler)
  - Còn gọi là bộ lập lịch CPU
  - Lựa chọn tiến trình tiếp theo được sử dụng CPU
- *Bộ lập lịch trung hạn* (medium-term scheduler)
  - Hay còn gọi là swapping (tráo đổi)
  - Di chuyển tiến trình đang trong trạng thái chờ giữa bộ nhớ trong và bộ nhớ ngoài

11

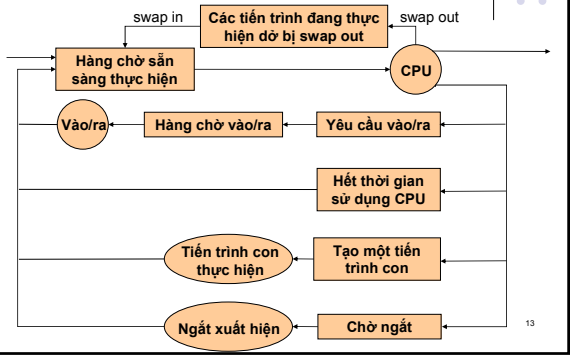
Minh họa bộ lập lịch trung hạn



12

Simpo PDF Merge and Split Unregistered Version - http://www.simpopdf.com

Hàng chờ lập lịch tiến trình



13

Chuyển trạng thái

- Thuật ngữ: Context switch
- Xảy ra khi một tiến trình A bị ngắt ra khỏi CPU, tiến trình B bắt đầu được sử dụng CPU
- Cách thực hiện:
  - Nhân HĐH ghi lại toàn bộ trạng thái của A, lấy từ PCB (khối điều khiển tiến trình) của A
  - Đưa A vào hàng chờ
  - Nhân HĐH nạp trạng thái của B lấy từ PCB của B
  - Thực hiện B

14

Chuyển trạng thái

- Việc chuyển trạng thái, nói chung, là lãng phí thời gian của CPU
- Do đó việc chuyển trạng thái cần được thực hiện càng nhanh càng tốt
- Thông thường thời gian chuyển trạng thái mất khoảng 1-1000 micro giây

15

Các thao tác với tiến trình

16

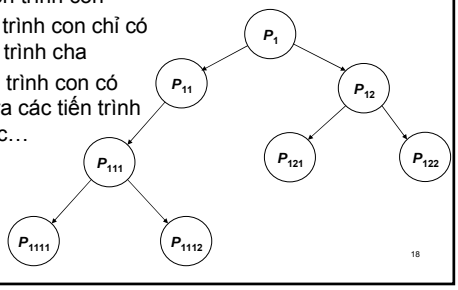
Tạo tiến trình

- HĐH cung cấp hàm **create-process** để tạo một tiến trình mới
  - Tiến trình gọi đến hàm **create-process** là tiến trình cha (parent process)
  - Tiến trình được tạo ra sau khi thực hiện hàm **create-process** là tiến trình con (child process)
- Sau khi tiến trình con được tạo, tiến trình cha có thể:
  - Chờ tiến trình con kết thúc rồi tiếp tục thực hiện
  - Thực hiện "song song" với tiến trình con

17

Cây tiến trình

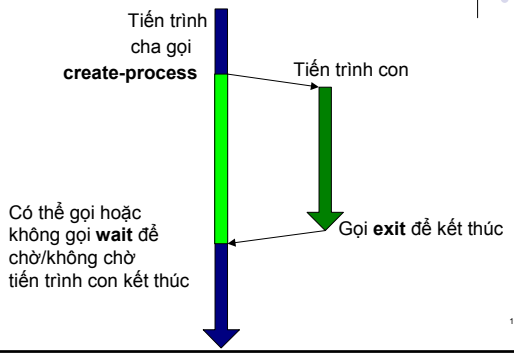
- Tiến trình cha có thể có nhiều tiến trình con
- Mỗi tiến trình con chỉ có một tiến trình cha
- Các tiến trình con có thể tạo ra các tiến trình con khác…



18

Simpo PDF Merge and Split Unregistered Version - http://www.simpopdf.com

Minh họa tiến trình cha và con



19

Kết thúc tiến trình

- Một tiến trình kết thúc khi:
  - Thực hiện xong và gọi hàm hệ thống **exit** (kết thúc bình thường)
  - Gọi đến hàm **abort** hoặc **kill** (kết thúc bất thường khi có lỗi hoặc có sự kiện)
  - Bị hệ thống hoặc tiến trình cha áp dụng hàm **abort** hoặc **kill** do:
    - Sử dụng quá quota tài nguyên
    - Tiến trình con không còn cần thiết
    - Khi tiến trình cha đã kết thúc (trong một số HĐH)

20

Minh họa tiến trình trong UNIX

```
#include <stdio.h>
main()
{
    int pid=fork(); /* Tạo tiến trình mới bằng hàm fork() */
    if (pid<0) { perror("Cannot create process"); return(-1); }
    else if (pid==0) { /* Tiến trình con */
        execlp();
    }
    else { /* Tiến trình cha */
        wait(NULL);  /* Nếu không có lệnh này tiến trình cha thực hiện
                        "song song" với tiến trình con */
        printf("Child completed\n");
        return(0);
    }
}
```

21

Hợp tác giữa các tiến trình

- Các tiến trình có thể hoạt động *độc lập* hoặc *hợp tác* với nhau
- Các tiến trình cần hợp tác khi:
  - Sử dụng chung thông tin
  - Thực hiện một số nhiệm vụ chung
  - Tăng tốc độ tính toán
  - …
- Để hợp tác các tiến trình, cần có các cơ chế truyền thông/liên lạc giữa các tiến trình (Interprocess communication – IPC)

22

Truyền thông giữa các tiến trình (IPC)

23

Các hệ thống truyền thông điệp

- Cho phép các tiến trình truyền thông với nhau qua các toán tử **send** và **receive**
- Các tiến trình cần có *tên* để tham chiếu
- Cần có một kết nối (*logic)* giữa tiến trình *P* và *Q* để truyền thông điệp
- Một số loại truyền thông:
  - Trực tiếp hoặc gián tiếp
  - Đối xứng hoặc không đối xứng
  - Sử dụng vùng đệm tự động hoặc không tự động

24

Simpo PDF Merge and Split Unregistered Version - http://www.simpopdf.com

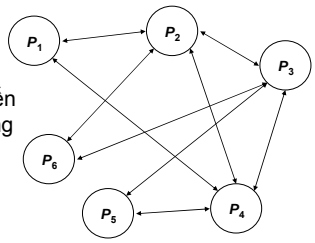
Truyền thông trực tiếp

- Hai toán tử
  - **send**(*P*, *msg*): Gửi *msg* đến tiến trình *P*
  - **receive**(*Q*, *msg*): Nhận *msg* từ tiến trình *Q*
- Cải tiến:
  - **send**(*P*, *msg*): Gửi *msg* đến tiến trình *P*
  - **receive**(*id*, *msg*): Nhận *msg* từ bất kỳ tiến trình nào

25

Minh họa truyền thông trực tiếp

- Mỗi kết nối được thiết lập cho một cặp tiến trình duy nhất
- Mỗi tiến trình chỉ cần biết tên/số hiệu của tiến trình kia là truyền thông được
- Tồn tại duy nhất một kết nối giữa một cặp tiến trình



26

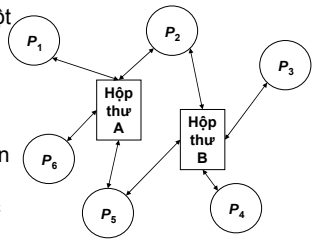
Truyền thông gián tiếp

- Các thông điệp được gửi và nhận qua các *hộp thư (*mailbox) hoặc qua các *cổng* (port)
- Hai toán tử:
  - **send**(*A*, *msg*): Gửi *msg* đến hộp thư *A*
  - **receive**(*B*, *msg*): Nhận *msg* từ hộp thư *B*
- Minh họa: Topo mạng hình sao

27

Minh họa truyền thông gián tiếp

- Hai tiến trình có kết nối nếu sử dụng chung một hộp thư
- Một kết nối có thể sử dụng cho nhiều tiến trình (>=2)
- Nhiều kết nối có thể tồn tại giữa một cặp tiến trình (nếu sử dụng các hộp thư khác nhau)



28

Vấn đề đồng bộ hóa

- Thuật ngữ: Synchronization
- Liên quan tới phương thức cài đặt các toán tử **send** và **receive**:
  - Phương thức có chờ (blocking)
  - Phương thức không chờ (non-blocking)

29

Các phương thức send/receive

	send(*P*, *msg*)	**receive**(*Q*, *msg*)
Blocking	Tiến trình truyền thông điệp chờ đến khi *msg* được nhận hoặc *msg* được phân phát đến hộp thư	Tiến trình nhận tạm dừng thực hiện cho đến khi *msg* được chuyển tới
Non-blocking	Tiến trình truyền không phải chờ *msg* đến đích để tiếp tục thực hiện	Tiến trình nhận trả lại kết quả là *msg* (nếu nhận được) hoặc báo lỗi (nếu chưa nhận được)

30

Simpo PDF Merge and Split Unregistered Version - http://www.simpopdf.com

Vấn đề sử dụng vùng đệm

- Các thông điệp nằm trong hàng chờ tạm thời
- Cỡ của hàng chờ:
  - Chứa được 0 thông điệp: **send** blocking
  - Chứa được *n* thông điệp: **send** non-blocking cho đến khi hàng chờ có *n* thông điệp, sau đó **send** blocking
  - Vô hạn: **send** non-blocking

31

Luồng (thread)

- Sinh viên tự tìm hiểu trong giáo trình trang

32

Tóm tắt

- Khái niệm tiến trình
- Các trạng thái, chuyển trạng thái tiến trình
- Khối điều khiển tiến trình
- Lập lịch tiến trình, các loại bộ lập lịch
- Truyền thông giữa các tiến trình
  - Gián tiếp, trực tiếp
  - Blocking và non-blocking (đồng bộ hóa)
  - Vấn đề sử dụng vùng đệm (buffer)

33

Bài tập

- Viết chương trình C trong Linux/Unix tạo ra 16 tiến trình con. Tiến trình cha chờ cho 16 tiến trình con này kết thúc rồi mới kết thúc bằng hàm **exit.** Sử dụng các hàm **fork** và **wait** để thực hiện yêu cầu.
- Hãy tìm một số ví dụ thực tế minh họa cho các khái niệm lập lịch/hàng chờ trong tình huống có nhiều người sử dụng và ít tài nguyên.

34

Bài tập

- Hãy viết chương trình minh họa cho các cơ chế truyền thông non-blocking, blocking
- Hãy viết chương trình minh họa các cơ chế truyền thông điệp sử dụng buffer có độ dài *n* trong hai trường hợp: *n*>0 và *n*=0
- Chú ý: Để làm hai bài tập trên cần sử dụng hai tiến trình; có thể thực hiện bài tập với UNIX/Linux hoặc Windows

35

Simpo PDF Merge and Split Unregistered Version - http://www.simpopdf.com

Nguyên lý hệ điều hành

Nguyễn Hải Châu
Khoa Công nghệ thông tin
Trường Đại học Công nghệ

1

Lập lịch CPU

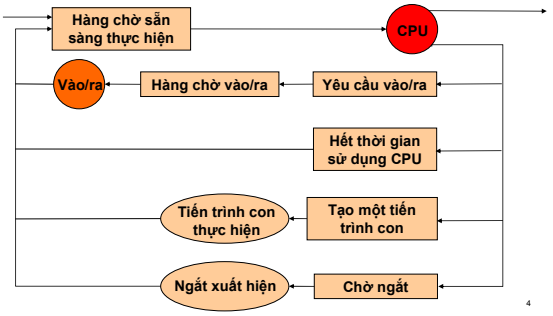
2

Tại sao phải lập lịch CPU?

- Số lượng NSD, số lượng tiến trình luôn lớn hơn số lượng CPU của máy tính rất nhiều
- Tại một thời điểm, chỉ có duy nhất một tiến trình được thực hiện trên một CPU
- Vấn đề:
  - Nhu cầu sử dụng nhiều hơn tài nguyên (CPU) đang có
  - Do đó cần lập lịch để phân phối thời gian sử dụng CPU cho các tiến trình của NSD và hệ thống

3

Hàng chờ lập lịch tiến trình

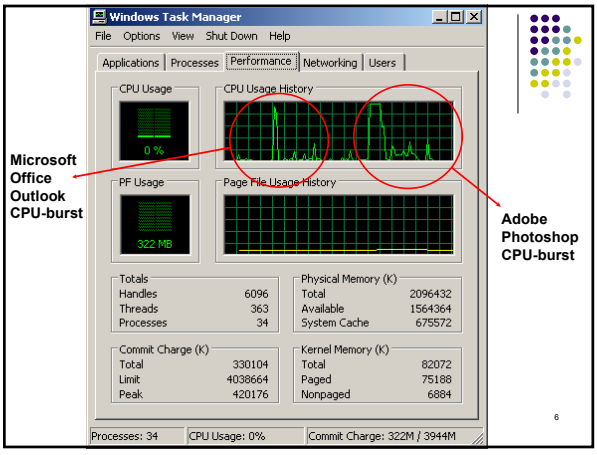


4

CPU-burst và IO-burst

- Trong suốt thời gian tồn tại trong hệ thống, tiến trình được xem như thực hiện hai loại công việc chính:
  - Khi tiến trình ở trạng thái running: Sử dụng CPU (thuật ngữ: *CPU-burst*)
  - Khi tiến trình thực hiện các thao tác vào ra: Sử dụng thiết bị vào/ra (thuật ngữ: *I/O burst*)

5



6

Simpo PDF Merge and Split Unregistered Version - http://www.simpopdf.com

Hai loại tiến trình chính

- Căn cứ theo cách sử dụng CPU của tiến trình, có hai loại tiến trình:
  - Tiến trình loại CPU-bound: Tiến trình có một hoặc nhiều phiên sử dụng CPU dài
  - Tiến trình loại I/O-bound: Tiến trình có nhiều phiên sử dụng CPU ngắn (tức là thời gian vào ra nhiều)

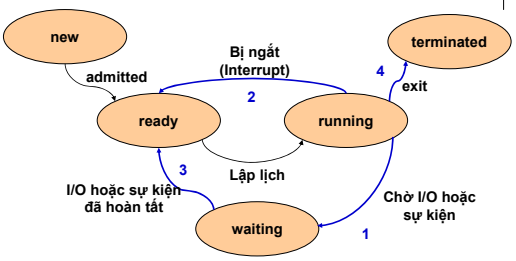
7

Bộ lập lịch ra hoạt động khi…

1. Một tiến trình chuyển từ trạng thái running sang waiting
2. Một tiến trình chuyển từ trạng thái running sang ready
3. Một tiến trình chuyển từ trạng thái waiting sang ready
4. Một tiến trình kết thúc

8

Các phương pháp lập lịch



- 1 và 4: Lập lịch non-preemptive
- Ngược lại: Lập lịch preemptive

9

Lập lịch non-preemptive

- Một tiến trình giữ CPU đến khi nó kết thúc hoặc chuyển sang trạng thái waiting.
- Ví dụ: Microsoft Windows 3.1, Apple Macintosh sử dụng lập lịch non-preemptive
- Có thể sử dụng trên nhiều loại phần cứng vì không đòi hỏi timer

10

Lập lịch preemptive

- Hiệu quả hơn lập lịch non-preemptive
- Thuật toán phức tạp hơn non-preemptive và sử dụng nhiều tài nguyên CPU hơn
- Ví dụ: Microsoft Windows XP, Linux, UNIX sử dụng lập lịch preemptive

11

Bộ điều phối (dispatcher)

- Nhiệm vụ:
  - Chuyển trạng thái (context switch)
  - Chuyển về user-mode
  - Thực hiện tiến trình theo trạng thái đã lưu
- Cần hoạt động hiệu quả (tốc độ nhanh)
- Thời gian cần để bộ điều phối dừng một tiến trình và thực hiện tiến trình khác gọi là độ trễ (latency) của bộ điều phối

12

Simpo PDF Merge and Split Unregistered Version - http://www.simpopdf.com

Các tiêu chí đánh giá lập lịch

- Khả năng tận dụng CPU (*CPU utilization*): Thể hiện qua tải CPU – là một số từ 0% đến 100%.
  - Trong thực tế các hệ thống thường có tải từ 40% (tải thấp) đến 90% (tải cao)
- Thông lượng (*throughput*): Là số lượng các tiến trình hoàn thành trong một đơn vị thời gian

13

Các tiêu chí đánh giá lập lịch

- Thời gian hoàn thành (*turnaround time*):
  - $t_{turnaround} = t_o - t_i$ với t_i là thời điểm tiến trình vào hệ thống, t_o là thời điểm tiến trình ra khỏi hệ thống (kết thúc thực hiện)
  - Như vậy $t_{turnaround}$ là tổng: thời gian tải vào bộ nhớ, thời gian thực hiện, thời gian vào ra, thời gian nằm trong hàng chờ…

14

Các tiêu chí đánh giá lập lịch

- Thời gian chờ (*waiting time*): Là tổng thời gian tiến trình phải nằm trong hàng chờ ready ($t_{waiting}$)
  - Các thuật toán lập lịch CPU không có ảnh hưởng đến tổng thời gian thực hiện một tiến trình mà chỉ có ảnh hưởng đến thời gian chờ của một tiến trình trong hàng chờ ready
  - Thời gian chờ trung bình (*average waiting time*): $t_{averagewaiting}=t_{waiting} / n$, n là số lượng tiến trình trong hàng chờ

15

Các tiêu chí đánh giá lập lịch

- Thời gian đáp ứng (*response time*): Là khoảng thời gian từ khi tiến trình nhận được một yêu cầu cho đến khi *bắt đầu đáp ứng* yêu cầu đó
- $t_{res} = t_r - t_s$, trong đó t_r là thời điểm nhận yêu cầu, t_s là thời điểm bắt đầu đáp ứng yêu cầu

16

Đánh giá các thuật toán lập lịch

Tiêu chí	Giá trị thấp	Giá trị cao
Khả năng tận dụng CPU (CPU utilization)	Xấu	Tốt
Thông lượng (throughput)	Xấu	Tốt
Thời gian hoàn thành (turnaround time)	Tốt	Xấu
Thời gian chờ (waiting time) -> Thời gian chờ trung bình	Tốt	Xấu
Thời gian đáp ứng (response time)	Tốt	Xấu

17

Các thuật toán lập lịch

18

Simpo PDF Merge and Split Unregistered Version - http://www.simpopdf.com

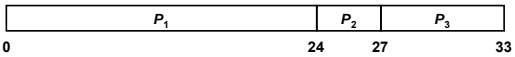
FCFS (First Come First Served)

- Tiến trình nào có yêu cầu sử dụng CPU trước sẽ được thực hiện trước
- Ưu điểm: Thuật toán đơn giản nhất
- Nhược điểm: Hiệu quả của thuật toán phụ thuộc vào *thứ tự* của các tiến trình trong hàng chờ, vì thứ tự này ảnh hưởng rất lớn đến *thời gian chờ trung bình* (*average waiting time*)

19

Ví dụ FCFS 1a

- Giả sử có 3 tiến trình P_1, P_2, P_3 với thời gian thực hiện tương ứng là 24ms, 3ms, 6ms
- Giả sử 3 tiến trình xếp hàng theo thứ tự P_1, P_2, P_3. Khi đó ta có biểu đồ Gantt như sau:

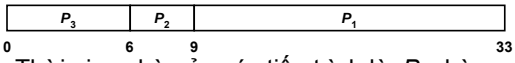


- Thời gian chờ của các tiến trình là: P_1 chờ 0ms, P_2 chờ 24ms, P_3 chờ 27ms
- Thời gian chờ trung bình: (0+24+27)/3=17ms

20

Ví dụ FCFS 1b

- Xét ba tiến trình trong ví dụ 1a với thứ tự xếp hàng P_3, P_2, P_1
- Biểu đồ Gantt:



- Thời gian chờ của các tiến trình là: P_3 chờ 0ms, P_2 chờ 6ms, P_3 chờ 9ms
- Thời gian chờ trung bình: (0+6+9)/3=5ms
- Thời gian chờ trung bình thấp hơn thời gian chờ trung bình trong ví dụ 1a

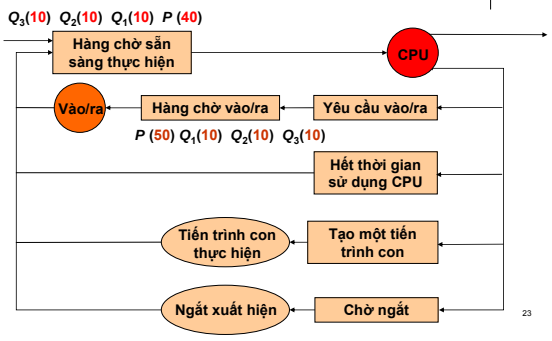
21

Hiện tượng "đoàn hộ tống"

- Thuật ngữ: *convoy effect*
- Xảy ra khi có một tiến trình "lớn" *P* nằm ở đầu hàng chờ và nhiều tiến trình "nhỏ" Q_i xếp hàng sau *P*.
- "Lớn": Sử dụng nhiều thời gian CPU và vào ra
- "Nhỏ": Sử dụng ít thời gian CPU và vào ra
- Thuật toán lập lịch được sử dụng là FCFS.:
- Hiện tượng xảy ra: CPU, thiết bị vào ra có nhiều thời gian rỗi, thời gian chờ trung bình của các tiến trình cao

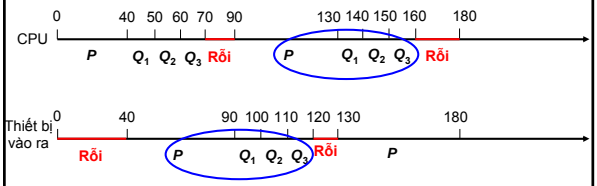
22

Ví dụ convoy effect



23

Convoy effect: Thời gian sử dụng CPU và thiết bị vào ra



- Giả sử *P* là tiến trình "lớn" có chu kỳ sử dụng CPU trong 40ms và vào ra trong 50ms
- Q_1, Q_2, Q_3 là 3 tiến trình "nhỏ" có chu kỳ sử dụng CPU trong 10ms và vào ra trong 10ms
- Thứ tự xếp hàng: *P*, Q_1, Q_2, Q_3

24

Simpo PDF Merge and Split Unregistered Version - http://www.simpopdf.com

SJF (Shortest Job First)

- Với SJF, tham số lập lịch có thêm *độ dài của phiên sử dụng CPU tiếp theo* $t_{nextburst}$
- Tiến trình có $t_{nextburst}$ nhỏ nhất sẽ được lập lịch sử dụng CPU trước
- Nếu hai tiến trình có $t_{nextburst}$ bằng nhau thì FCFS được áp dụng

25

SJF (Shortest Job First)

- Với lập lịch dài hạn: Có thể biết $t_{nextburst}$ vì có tham số chỉ ra thời gian chạy của tiến trình khi đưa tiến trình vào hệ thống (job submit)
- Khó khăn: Chỉ có thể phỏng đoán được $t_{nextburrt}$ với lập lịch ngắn hạn
- Đọc ở nhà: Dự đoán $t_{nextburst}$ bằng công thức trung bình mũ (exponential average) trang 160-162 trong giáo trình
- SJF không tối ưu được với lập lịch ngắn hạn
- SJF thường được áp dụng cho lập lịch dài hạn

26

Lập lịch có độ ưu tiên

- Thuật ngữ: Priority scheduling
- Mỗi tiến trình được gắn một tham số lập lịch gọi là độ ưu tiên *p*, với *p* là một số thực
- Tiến trình có độ ưu tiên cao nhất được sử dụng CPU
- Qui ước trong môn học này:
  - Tiến trình P_1 và P_2 có độ ưu tiên p_1, p_2 tương ứng
  - $p_1 > p_2$ có nghĩa là tiến trình P_1 có độ ưu tiên thấp hơn P_2
- SJF là trường hợp đặc biệt của lập lịch có ưu tiên với ưu tiên của các tiến trình là nghịch đảo độ dài phiên sử dụng

27

Lập lịch có độ ưu tiên

- Hai cách xác định độ ưu tiên:
  - Độ ưu tiên trong: Được tính toán dựa trên các tham số định lượng của tiến trình như giới hạn về thời gian, bộ nhớ, số file đang mở, thời gian sử dụng CPU
  - Độ ưu tiên ngoài: Dựa vào các yếu tố như mức độ quan trọng, chi phí thuê máy tính…
- Chờ không xác định: Một tiến trình có độ ưu tiên thấp có thể nằm trong hàng chờ trong một khoảng thời gian dài nếu trong hàng chờ luôn có các tiến trình có độ ưu tiên cao hơn

28

Lập lịch có độ ưu tiên

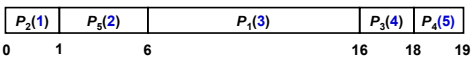
- Để tránh hiện tượng chờ không xác định, có thêm tham số *tuổi* để xác định thời gian tiến trình thời gian nằm trong hàng chờ
- Tham số *tuổi* được sử dụng để làm tăng độ ưu tiên của tiến trình
- Ví dụ thực tế: Khi bảo dưỡng máy tính IBM 7094 của MIT năm 1973, người ta thấy một tiến trình nằm trong hàng chờ từ năm 1967 nhưng vẫn chưa được thực hiện

29

Ví dụ lập lịch có độ ưu tiên

- Xét 5 tiến trình như trong bảng với thứ tự trong hàng chờ là P_1, P_2, P_3, P_4, P_5
- Sử dụng thuật toán lập lịch có ưu tiên, ta có thứ tự thực hiện của các tiến trình như sau biểu đồ dưới
- Thời gian chờ trung bình là (1+6+16+18)/5=8.2ms

Tiến trình	Thời gian thực hiện (ms)	Độ ưu tiên
P_1	10	3
P_2	1	1
P_3	2	4
P_4	1	5
P_5	5	2



30

Simpo PDF Merge and Split Unregistered Version - http://www.simpopdf.com

Round-robin (RR)

- Còn gọi là lập lịch quay vòng
- Được thiết kế để áp dụng cho các hệ phân chia thời gian (time-sharing)
- RR hoạt động theo chế độ preemptive
- Tham số lượng tử thời gian (time quantum) $t_{quantum}$: Mỗi tiến trình được sử dụng CPU trong nhiều nhất bằng $t_{quantum}$, sau đó đến lượt tiến trình khác

31

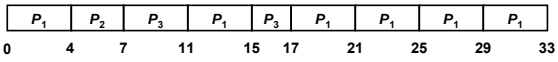
Round-robin (RR)

- Hiệu quả của RR phụ thuộc độ lớn của $t_{quantum}$
  - Nếu $t_{quaptum}$ nhỏ thì hiệu quả của RR giảm vì bộ điều phối phải thực hiện nhiều thao tác chuyển trạng thái, lãng phí thời gian CPU
  - Nếu $t_{quantum}$ lớn thì số thao tác chuyển trạng thái giảm đi
- Nếu $t_{quantum}$ rất nhỏ (ví dụ 1ms) thì RR được gọi là processor sharing
- Nếu $t_{quantum}$ = ∞ thì RR trở thành FCFS

32

Ví dụ RR

- Giả sử có 3 tiến trình P_1, P_2, P_3 với thời gian thực hiện tương ứng là 24ms, 3ms, 6ms, thứ tự trong hàng chờ P_1, P_2, P_3 ,vào hàng chờ cùng thời điểm 0
- Giả sử $t_{quantum}$=4ms
- RR lập lịch các tiến trình như sau:



- Thời gian chờ
  - P_1: 0+(11-4)+(17-15)=9ms
  - P_2: 4ms
  - P_3: 7+(15-11)=11ms
- Thời gian chờ trung bình: (9+4+11)/3=8ms

33

Lập lịch với hàng chờ đa mức

- Thuật ngữ: Multilevel queue scheduling
- Được sử dụng khi ta có thể chia các tiến trình thành nhiều lớp khác nhau để lập lịch theo các tiêu chí khác nhau, ví dụ:
  - Lớp các tiến trình có tương tác (interactive hoặc foreground process) cần có độ ưu tiên cao hơn
  - Lớp các tiến trình chạy nền (background) thường không có tương tác với NSD: Độ ưu tiên thấp hơn

34

Lập lịch với hàng chờ đa mức

- Thuật toán lập lịch với hàng chờ đa mức chia hàng chờ ready thành nhiều hàng chờ con khác nhau, mỗi hàng chờ con được áp dụng một loại thuật toán khác nhau, ví dụ:
  - Hàng chờ các tiến trình background: FCFS
  - Hàng chờ các tiến trình có tương tác:RR
- Các tiến trình được phân lớp dựa vào đặc tính như bộ nhớ, độ ưu tiên, …
- Cần có thuật toán lập lịch cho các hàng chờ con, ví dụ: preemptive có độ ưu tiên cố định

35

Ví dụ hàng chờ đa mức

- Ví dụ các hàng chờ đa mức có độ ưu tiên giảm dần:
  - Hàng chờ các tiến trình hệ thống
  - Hàng chờ các tiến trình có tương tác
  - Hàng chờ các tiến trình là editor
  - Hàng chờ các tiến trình hoạt động theo lô
  - Hàng chờ các tiến trình thực tập của sinh viên

36

Simpo PDF Merge and Split Unregistered Version - http://www.simpopdf.com

Lập lịch với hàng chờ đa mức có phản hồi

- Thuật ngữ: Multilevel feedback-queue scheduling
- Thuật toán lập lịch kiểu này nhằm khắc phục nhược điểm *không mềm dẻo* của lập lịch với hàng chờ đa mức
- Ý tưởng chính: Cho phép tiến trình chuyển từ hàng chờ này sang hàng chờ khác, trong khi lập lịch với hàng chờ đa mức không cho phép điều này

37

Lập lịch với hàng chờ đa mức có phản hồi

- Cách thực hiện:
  - Độ dài phiên sử dụng CPU và thời gian đã nằm trong hàng chờ là tiêu chuẩn chuyển tiến trình giữa các hàng chờ
  - Tiến trình chiếm nhiều thời gian CPU sẽ bị chuyển xuống hàng chờ có độ ưu tiên thấp
  - Tiến trình nằm lâu trong hàng chờ sẽ được chuyển lên hàng chờ có độ ưu tiên cao hơn

38

Các tham số của bộ lập lịch hàng chờ đa mức có phản hồi

- Số lượng các hàng chờ
- Thuật toán lập lịch cho mỗi hàng chờ
- Phương pháp tăng độ ưu tiên cho một tiến trình
- Phương pháp giảm độ ưu tiên cho một tiến trình
- Phương pháp xác định hàng đợi nào để đưa một tiến trình vào

39

Các thuật toán lập lịch khác

- Sinh viên tìm hiểu trong giáo trình:
  - Lập lịch đa xử lý
  - Lập lịch thời gian thực

40

Các phương pháp đánh giá thuật toán lập lịch

41

Mô hình xác định

- Thuật ngữ: Deterministic modeling
- Dựa vào các trường hợp cụ thể (chẳng hạn các ví dụ đã nói trong bài giảng này) để rút ra các kết luận đánh giá
- Ưu điểm: Nhanh và đơn giản
- Nhược điểm: Không rút ra được kết luận đánh giá cho trường hợp tổng quát

42

Simpo PDF Merge and Split Unregistered Version - http://www.simpopdf.com

Mô hình hàng chờ

- Thuật ngữ: Queueing model
- Dựa trên lý thuyết xác suất, quá trình ngẫu nhiên. Tài liệu tham khảo:
  - Leonard Kleinrock, *Queueing Systems: Volume I – Theory* (Wiley Interscience, New York, 1975)
  - Leonard Kleinrock, *Queueing Systems: Volume II – Computer Applications* (Wiley Interscience, New York, 1976)
- Ưu điểm: So sánh được các thuật toán lập lịch trong một số trường hợp
- Hạn chế: Phức tạp về mặt toán học, lớp các thuật toán phân tích so sánh được còn hạn chế

43

Mô phỏng

- Thuật ngữ: Simulation
- Thực hiện các tình huống giả định trên máy tính để đánh giá hiệu quả của các thuật toán lập lịch, kiểm nghiệm các kết quả lý thuyết
- Mô phỏng thường tốn thời gian CPU và không gian lưu trữ
- Được sử dụng nhiều trong công nghiệp
- Ví dụ các hệ mô phỏng mạng (có module hàng chờ): OPNET, NS-2, Qualnet

44

Cài đặt thử trong thực tế

- Mô phỏng có thể xem như "qui nạp không hoàn toàn"
- Có thể xây dựng hệ thống thử trong thực tế
- Ưu điểm: Đánh giá được hiệu quả thực sự khi sử dụng
- Nhược điểm:
  - Chi phí cao
  - Hành vi của người sử dụng có thể thay đổi theo môi trường hệ thống

45

Tóm tắt

- Khái niệm lập lịch, các tiêu chí đánh giá thuật toán lập lịch
- Các phương thức hoạt động preemptive và non-preemptive
- Các thuật toán lập lịch FCFS, SJF, ưu tiên, RR
- Lập lịch với hàng chờ đa mức, có và không có phản hồi
- Các phương pháp đánh giá thuật toán lập lịch

46

Bài tập

- Thực hiện ví dụ RR với lượng tử thời gian 2ms, 6ms và 6ms.
  - Tính thời gian chờ trung bình của các tiến trình trong các trường hợp này.
  - Khi lượng tử thời gian thay đổi, thời gian chờ trung bình thay đổi thế nào?
  - Tính thời gian hoàn thành (turnaround time) của tất cả các tiến trình trong các trường hợp trên
  - Nhận xét về sự thay đổi thời gian hoàn thành của các tiến trình khi lượng tử thời gian thay đổi

47

Bài tập

- Hãy xây dựng một ví dụ về hiện tượng convoy effect sao cho thời gian rỗi của thiết bị vào ra ít hơn thời gian rỗi của CPU. Giả sử có 4 tiến trình, trong đó *P* là tiến trình "lớn", Q_1, Q_2, Q_3 là các tiến trình "nhỏ" được nằm trong hàng chờ theo thứ tự: *P,* Q_1, Q_2, Q_3
- Tìm hai ví dụ trong thực tế về hàng chờ đa mức và hàng chờ đa mức có phản hồi và giải thích các ví dụ này.

48

Simpo PDF Merge and Split Unregistered Version - http://www.simpopdf.com

Nguyên lý hệ điều hành

Nguyễn Hải Châu
Khoa Công nghệ thông tin
Trường Đại học Công nghệ

Đồng bộ hóa tiến trình

Ví dụ đồng bộ hóa (1)

Tiến trình ghi *P*:

```
while (true) {
   while (counter==SIZE) ;
   buf[in] = nextItem;
   in = (in+1) % SIZE;
   counter++;
}
```

buf: Buffer
SIZE: cỡ của buffer
counter: Biến chung

Tiến trình đọc *Q*:

```
while (true) {
   while (counter==0) ;
   nextItem = buf[out];
   out = (out+1) % SIZE;
   counter--;
}
```

- Đây là *bài toán vùng đệm có giới hạn*

Ví dụ đồng bộ hóa (2)

- Các toán tử ++ và -- có thể được cài đặt như sau:

- counter++

```
register1 = counter;
register1 = register1 + 1;
counter = register1;
```

- counter--

```
register2 = counter;
register2 = register2 - 1;
counter = register2;
```

P và Q có thể nhận được các giá trị khác nhau của counter tại cùng 1 thời điểm nếu như đoạn mã xanh và đỏ thực hiện xen kẽ nhau.

Ví dụ đồng bộ hóa (3)

- Giả sử P và Q thực hiện song song với nhau và giá trị của counter là 5:

```
register1 = counter;          // register1=5
register1 = register1 + 1;    // register1=6
register2 = counter;          // register2=5
register2 = register2 - 1;    // register2=4
counter = register1;          // counter=6 !!
counter = register2;          // counter=4 !!
```


Ví dụ đồng bộ hóa (4)

- Lỗi: Cho phép *P* và *Q* **đồng thời thao tác trên biến chung counter**. Sửa lỗi:

```
register1 = counter;          // register1=5
register1 = register1 + 1;    // register1=6
counter = register1;          // counter=6
register2 = counter;          // register2=6
register2 = register2 - 1;    // register2=5
counter = register2;          // counter=5
```

Simpo PDF Merge and Split Unregistered Version - http://www.simpopdf.com

Tương tranh và đồng bộ

- Tình huống xuất hiện khi nhiều tiến trình cùng thao tác trên dữ liệu chung và kết quả các thao tác đó phụ thuộc vào thứ tự thực hiện của các tiến trình trên dữ liệu chung gọi là *tình huống tương tranh* (*race condition*)
- Để tránh các tình huống tương tranh, các tiến trình cần được *đồng bộ* theo một phương thức nào đó ⇒ Vấn đề nghiên cứu: Đồng bộ hóa các tiến trình

7

Khái niệm về đoạn mã găng (1)

- Thuật ngữ: Critical section
- Thuật ngữ tiếng Việt: Đoạn mã găng, đoạn mã tới hạn.
- Xét một hệ có n tiến trình P_0, P_1, ..., P_n, mỗi tiến trình có một đoạn mã lệnh gọi là đoạn mã găng, ký hiệu là CS_i, nếu như trong đoạn mã này, các tiến trình thao tác trên các biến chung, đọc ghi file... (tổng quát: thao tác trên dữ liệu chung)

8

Khái niệm về đoạn mã găng (2)

- Đặc điểm quan trọng mà hệ n tiến trình này cần có là: Khi một tiến trình P_i thực hiện đoạn mã CS_i thì không có tiến trình P_j nào khác được phép thực hiện CS_j
- Mỗi tiến trình P_i phải "xin phép" (entry section) trước khi thực hiện CS_i và thông báo (exit section) cho các tiến trình khác sau khi thực hiện xong CS_i.

9

Khái niệm về đoạn mã găng (3)

- Cấu trúc chung của P_i để thực hiện đoạn mã găng CS_i.

```
do {
   Xin phép (ENTRY_i) thực hiện CS_i; // Entry section
   Thực hiện CS_i;
   Thông báo (EXIT_i) đã thực hiện xong CS_i; // Exit section
   Phần mã lệnh khác (REMAIN_i); // Remainder section
} while (TRUE);
```

10

Khái niệm về đoạn mã găng (4)

- Viết lại cấu trúc chung của đoạn mã găng:

```
do {
   ENTRY_i;        // Entry section
   Thực hiện CS_i; // Critical section
   EXIT_i;         // Exit section
   REMAIN_i;       // Remainder section
} while (TRUE);
```

11

Giải pháp cho đoạn mã găng

- Giải pháp cho đoạn mã găng cần thỏa mãn 3 điều kiện:
  - Loại trừ lẫn nhau (mutual exclusion): Nếu P_i đang thực hiện CS_i thì P_j không thể thực hiện CS_j $\forall j \neq i$.
  - Tiến triển (progress): Nếu không có tiến trình P_i nào thực hiện CS_i và có m tiến trình P_{j1}, P_{j2}, ..., P_{jm} muốn thực hiện CS_{j1}, CS_{j2}, ..., CS_{jm} thì chỉ có các tiến trình không thực hiện $REMAIN_{jk}$ (k=1,...,m) mới được xem xét thực hiện CS_{jk}.
  - Chờ có giới hạn (bounded waiting): *sau khi* một tiến trình P_i có yêu cầu vào CS_i và *trước khi* yêu cầu đó được chấp nhận, số lần các tiến trình P_j (với $j \neq i$) được phép thực hiện CS_j phải bị giới hạn.

12

Simpo PDF Merge and Split Unregistered Version - http://www.simpopdf.com

Ví dụ: giải pháp của Peterson

- Giả sử có 2 tiến trình P_0 và P_1 với hai đoạn mã găng tương ứng CS_0 và CS_1
- Sử dụng một biến nguyên *turn* với giá trị khởi tạo 0 hoặc 1 và mảng boolean *flag*[2]
- *turn* có giá trị *i* có nghĩa là P_i được phép thực hiện CS_i (*i*=0,1)
- nếu flag[i] là TRUE thì tiến trình P_i đã sẵn sàng để thực hiện CS_i

13

Ví dụ: giải pháp của Peterson

- Mã lệnh của P_i:

```
do {
   flag[i] = TRUE;
   turn = j;
   while (flag[j] && turn == j) ;
   CSi;
   flag[j] = FALSE;
   REMAINi;
} while (1);
```

14

Chứng minh giải pháp Peterson

- Xem chứng minh giải pháp của Peterson thỏa mãn 3 điều kiện của đoạn mã găng trong giáo trình (trang 196)
- Giải pháp "kiểu Peterson":
  - Phức tạp khi số lượng tiến trình tăng lên
  - Khó kiểm soát

15

Semaphore

16

Thông tin tham khảo

- Edsger Wybe Dijkstra (người Hà Lan) phát minh ra khái niệm semaphore trong khoa học máy tính vào năm 1972
- Semaphore được sử dụng lần đầu tiên trong cuốn sách "The operating system" của ông

Edsger Wybe Dijkstra (1930-2002)

17

Định nghĩa

- Semaphore là một biến nguyên, nếu không tính đến toán tử khởi tạo, chỉ có thể truy cập thông qua hai toán tử *nguyên tố* là wait (hoặc P) và signal (hoặc V).
  - P: proberen – kiểm tra (tiếng Hà Lan)
  - V: verhogen – tăng lên (tiếng Hà Lan)
- Các tiến trình có thể *sử dụng chung* semaphore
- Các toán tử là nguyên tố để đảm bảo không xảy ra trường hợp như ví dụ đồng bộ hóa đã nêu

18

Simpo PDF Merge and Split Unregistered Version - http://www.simpopdf.com

Toán tử wait và signal

```
wait(S)     // hoặc P(S)
{
   while (S<=0);
   S--;
}
```

- Toán tử wait: Chờ khi semaphore S âm và giảm S đi 1 nếu S>0

```
signal(S)     // hoặc V(S)
{
   S++;
}
```

- Toán tử signal: Tăng S lên 1

19

Sử dụng semaphore (1)

- Với bài toán đoạn mã găng:

```
do {
   wait(mutex); // mutex là semaphore khởi tạo 1
   CS_i;
   signal(mutex);
   REMAIN_i;
} while (1);
```

20

Sử dụng semaphore (2)

- Xét hai tiến trình P_1 và P_2, P_1 cần thực hiện toán tử O_1, P_2 cần thực hiện O_2 và O_2 chỉ được thực hiện sau khi O_1 đã hoàn thành
- Giải pháp: Sử dụng semaphore *synch* = 0

- P_1:

```
...
O1;
signal(synch);
...
```

- P_2:

```
...
wait(synch);
O2;
...
```

21

Cài đặt semaphore cổ điển

- Định nghĩa cổ điển của wait cho ta thấy toán tử này có *chờ bận* (busy waiting), tức là tiến trình phải chờ toán tử wait kết thúc nhưng CPU vẫn phải làm việc: Lãng phí tài nguyên
- Liên hệ cơ chế polling trong kiến trúc máy tính
- Cài đặt semaphore theo định nghĩa cổ điển:
  - Lãng phí tài nguyên CPU với các máy tính 1 CPU
  - Có lợi nếu thời gian chờ wait ít hơn thời gian thực hiện context switch
  - Các semaphore loại này gọi là *spinlock*

22

Cài đặt semaphore theo cấu trúc

- Khắc phục chờ bận: Chuyển vòng lặp chờ thành việc sử dụng toán tử block (tạm dừng)
- Để khôi phục thực hiện từ block, ta có toán tử wakeup
- Khi đó để cài đặt, ta có cấu trúc dữ liệu mới cho semaphore:

```
typedef struct {
   int value; // Giá trị của semaphore
   struct process *L; // Danh sách tiến trình chờ...
} semaphore;
```

23

Cài đặt semaphore theo cấu trúc

```
void wait(semaphore *S)
{
   S->value--;
   if (S->value<0) {
      Thêm tiến trình gọi
      toán tử vào s->L;
      block();
   }
}
```

```
void signal(semaphore *S)
{
   S->value++;
   if (S->value<=0) {
      Xóa một tiến trình P
      ra khỏi s->L;
      wakeup(P);
   }
}
```

24

Simpo PDF Merge and Split Unregistered Version - http://www.simpopdf.com

Semaphore nhị phân

- Là semaphore chỉ nhận giá trị 0 hoặc 1
- Cài đặt semaphore nhị phân đơn giản hơn semaphore không nhị phân (thuật ngữ: counting semaphore)

25

Một số bài toán đồng bộ hóa cơ bản

26

Bài toán vùng đệm có giới hạn

- Đã xét ở ví dụ đầu tiên (the bounded-buffer problem)
- Ta sử dụng 3 semaphore tên là *full*, *empty* và *mutex* để giải quyết bài toán này
- Khởi tạo:
  - *full*: Số lượng phần tử buffer đã có dữ liệu (0)
  - *empty*: Số lượng phần tử buffer chưa có dữ liệu (*n*)
  - *mutex*: 1 (Chưa có tiến trình nào thực hiện đoạn mã găng)

27

Bài toán vùng đệm có giới hạn

Tiến trình ghi *P*:

```
do {
   wait(empty);
   wait(mutex);
   // Ghi một phần tử mới
   // vào buffer
   signal(mutex);
   signal(full);
} while (TRUE);
```

Tiến trình đọc *Q*:

```
do {
   wait(full);
   wait(mutex);
   // Đọc một phần tử ra
   // khỏi buffer
   signal(mutex);
   signal(empty);
} while (TRUE);
```

28

Bài toán tiến trình đọc - ghi

- Thuật ngữ: the reader-writer problem
- Tình huống: Nhiều tiến trình cùng thao tác trên một cơ sở dữ liệu trong đó
  - Một vài tiến trình chỉ đọc dữ liệu (ký hiệu: reader)
  - Một số tiến trình vừa đọc vừa ghi (ký hiệu: writer)
- Khi có đọc/ghi đồng thời của nhiều tiến trình trên cùng một cơ sở dữ liệu, có 2 bài toán:
  - Bài toán 1: reader không phải chờ, trừ khi writer đã được phép ghi vào CSDL (hay các reader không loại trừ lẫn nhau khi đọc)
  - Bài toán 2: Khi writer đã sẵn sàng ghi, nó sẽ được ghi trong thời gian sớm nhất (nói cách khác khi writer đã sẵn sàng, không cho phép các reader đọc dữ liệu)

29

Bài toán tiến trình đọc-ghi số 1

- Sử dụng các semaphore với giá trị khởi tạo: *wrt* (1), *mutex* (1)
- Sử dụng biến *rcount* (khởi tạo 0) để đếm số lượng reader đang đọc dữ liệu
- *wrt*: Đảm bảo loại trừ lẫn nhau khi writer ghi
- *mutex*: Đảm bảo loại trữ lẫn nhau khi cập nhật biến *rcount*

30

Simpo PDF Merge and Split Unregistered Version - http://www.simpopdf.com

Bài toán tiến trình đọc-ghi số 1

- **Tiến trình writer P_w:**

```
do {
    wait(wrt);
    // Thao tác ghi đang được
    // thực hiện
    signal(wrt);
while (TRUE);
```

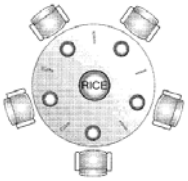
- **Tiến trình reader P_r:**

```
do {
    wait(mutex);
    rcount++;
    if (rcount==1) wait(wrt);
    signal(mutex);
    // Thực hiện phép đọc
    wait(mutex);
    rcount--;
    if (rcount==0) signal(wrt);
    signal(mutex);
} while (TRUE);
```

31

Bữa ăn tối của các triết gia

- Thuật ngữ: the dining-philosophers problem
- Có 5 triết gia, 5 chiếc đũa, 5 bát cơm và một âu cơm bố trí như hình vẽ
- Đây là bài toán cổ điển và là ví dụ minh họa cho một lớp nhiều bài toán tương tranh (concurrency): *Nhiều tiến trình khai thác nhiều tài nguyên chung*



32

Bữa ăn tối của các triết gia

- Các triết gia chỉ làm 2 việc: Ăn và suy nghĩ
  - Suy nghĩ: Không ảnh hưởng đến các triết gia khác, đũa, bát và âu cơm
  - Để ăn: Mỗi triết gia phải có đủ 2 chiếc đũa gần nhất ở bên phải và bên trái mình; chỉ được lấy 1 chiếc đũa một lần và không được phép lấy đũa từ tay triết gia khác
  - Khi ăn xong: Triết gia bỏ cả hai chiếc đũa xuống bàn và tiếp tục suy nghĩ

33

Giải pháp cho bài toán Bữa ăn...

- Biểu diễn 5 chiếc đũa qua mảng semaphore:

  semaphore chopstick[5];

  các semaphore được khởi tạo giá trị 1
- Mã lệnh của triết gia như hình bên
- Mã lệnh này có thể gây bế tắc (deadlock) nếu cả 5 triết gia đều lấy được 1 chiếc đũa và chờ để lấy chiếc còn lại nhưng không bao giờ lấy được!!

- Mã lệnh của triết gia *i*:

```
do {
    wait(chopstick[i]);
    wait(chopstick[(i+1)%5];
    // Ăn...
    signal(chopstick[i]);
    signal(chopstick[(i+1)%5];
    // Suy nghĩ...
} while (TRUE);
```

34

Một số giải pháp tránh bế tắc

- Chỉ cho phép nhiều nhất 4 triết gia đồng thời lấy đũa, dẫn đến có ít nhất 1 triết gia lấy được 2 chiếc đũa
- Chỉ cho phép lấy đũa khi cả hai chiếc đũa bên phải và bên trái đều nằm trên bàn
- Sử dụng giải pháp bất đối xứng: Triết gia mang số lẻ lấy chiếc đũa đầu tiên ở bên trái, sau đó chiếc đũa ở bên phải; triết gia mang số chẵn lấy chiếc đũa đầu tiên ở bên phải, sau đó lấy chiếc đũa bên trái

35

Hạn chế của semaphore

- Mặc dù semaphore cho ta cơ chế đồng bộ hóa tiện lợi song sử dụng semaphore không đúng cách có thể dẫn đến bế tắc hoặc lỗi do trình tự thực hiện của các tiến trình
- Trong một số trường hợp: khó phát hiện bế tắc hoặc lỗi do trình tự thực hiện khi sử dụng semaphore không đúng cách
- Sử dụng không đúng cách gây ra bởi *lỗi lập trình* hoặc do *người lập trình không cộng tác*

36

Simpo PDF Merge and Split Unregistered Version - http://www.simpopdf.com

Ví dụ hạn chế của semaphore (1)

- Xét ví dụ về đoạn mã găng:

Mã đúng:	Mã sai:

...
wait(mutex);
// Đoạn mã găng
signal(mutex);
...
	...
	signal(mutex);
	// Đoạn mã găng
	wait(mutex);
	...
	• Đoạn mã sai này gây ra vi phạm điều kiện loại trừ lẫn nhau

37

Ví dụ hạn chế của semaphore (2)

- Xét ví dụ về đoạn mã găng:

Mã đúng:	Mã sai:

...
wait(mutex);
// Đoạn mã găng
signal(mutex);
...
	...
	wait(mutex);
	// Đoạn mã găng
	wait(mutex);
	...
	• Đoạn mã sai này gây ra bế tắc

38

Ví dụ hạn chế của semaphore (3)

- Nếu người lập trình quên các toán tử wait() hoặc signal() trong trong các đoạn mã găng, hoặc cả hai thì có thể gây ra:
  - Bế tắc
  - Vi phạm điều kiện loại trừ lẫn nhau

39

Ví dụ hạn chế của semaphore (4)

Tiến trình P_1	Tiến trình P_2

...
wait(S);
wait(Q);
...
signal(S);
signal(Q);
	...
	wait(Q);
	wait(S);
	...
	signal(Q);
	signal(S);

- Hai tiến trình P_1 và P_2 đồng thời thực hiện sẽ dẫn tới bế tắc

40

Cơ chế monitor

41

Thông tin tham khảo

- Per Brinch Hansen (người Đan Mạch) là người đầu tiên đưa ra khái niệm và cài đặt monitor năm 1972
- Monitor được sử dụng lần đầu tiên trong ngôn ngữ lập trình Concurrent Pascal

Per Brinch Hansen
(1938-2007)

42

Simpo PDF Merge and Split Unregistered Version - http://www.simpopdf.com

Monitor là gì?

- Thuật ngữ monitor: *giám sát*
- Định nghĩa không hình thức: Là một loại construct trong ngôn ngữ bậc cao dùng để phục vụ các thao tác đồng bộ hóa
- Monitor được nghiên cứu, phát triển để khắc phục các hạn chế của semaphore như đã nêu trên

43

Định nghĩa tổng quát

- Monitor là một cách tiếp cận để đồng bộ hóa các tác vụ trên máy tính khi phải sử dụng các tài nguyên chung. Monitor thường gồm có:
  - Tập các procedure thao tác trên tài nguyên chung
  - Khóa loại trừ lẫn nhau
  - Các biến tương ứng với các tài nguyên chung
  - Một số các giả định bất biến nhằm tránh các tình huống tương tranh
- Trong bài này: Nghiên cứu một loại cấu trúc monitor: Kiểu monitor (monitor type)

44

Monitor type

- Một kiểu (type) hoặc kiểu trừu tượng (abstract type) gồm có các dữ liệu *private* và các phương thức *public*
- Monitor type được đặc trưng bởi tập các toán tử của người sử dụng định nghĩa
- Monitor type có các biến xác định các trạng thái; mã lệnh của các procedure thao tác trên các biến này

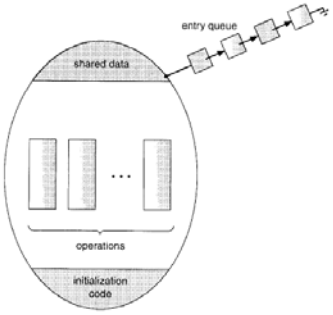
45

Cấu trúc một monitor type

```
monitor tên_monitor {
   // Khai báo các biến chung
   procedure P1(...) { ...
   }
   procedure P2(...) { ...
   }
   ...
   procedure Pn(...) { ...
   }
   initialization_code (..) { ...
   }
}
```

46

Minh họa cấu trúc monitor



47

Cách sử dụng monitor

- Monitor được cài đặt sao cho *chỉ có một tiến trình được hoạt động trong monitor* (*loại trừ lẫn nhau*). Người lập trình không cần viết mã lệnh để đảm bảo điều này
- Monitor như định nghĩa trên chưa đủ mạnh để xử lý mọi trường hợp đồng bộ hóa. Cần thêm một số cơ chế "tailor-made" về đồng bộ hóa
- Các trường hợp đồng bộ hóa "tailor-made": sử dụng kiểu *condition*.

48

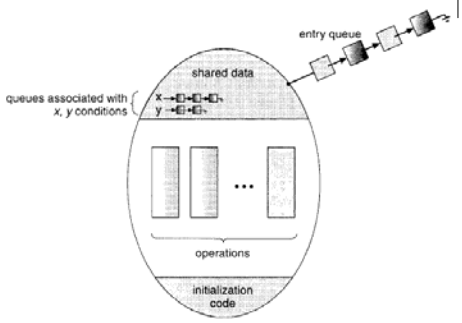
Simpo PDF Merge and Split Unregistered Version - http://www.simpopdf.com

Kiểu condition

- Khai báo:
  condition x, y; // x, y là các biến kiểu condition
- Sử dụng kiểu condition: Chỉ có 2 toán tử là wait và signal
  - x.wait(): tiến trình gọi đến x.wait() sẽ được chuyển sang trạng thái chờ (wait hoặc suspend)
  - x.signal(): tiến trình gọi đến x.signal() sẽ khôi phục việc thực hiện (wakeup) một tiến trình đã gọi đến x.wait()

49

Monitor có kiểu condition



50

Đặc điểm của x.signal()

- x.signal() chỉ đánh thức duy nhất một tiến trình đang chờ
- Nếu không có tiến trình chờ, x.signal() không có tác dụng gì
- x.signal() khác với signal trong semaphore cổ điển: signal cổ điển luôn làm thay đổi trạng thái (giá trị) của semaphore

51

Signal wait/continue

- Giả sử có hai tiến trình P và Q:
  - *Q* gọi đến x.wait(), sau đó *P* gọi đến x.signal()
  - *Q* được phép tiếp tục thực hiện (wakeup)
- Khi đó *P* phải vào trạng thái wait vì nếu ngược lại thì *P* và *Q* cùng thực hiện trong monitor
- Khả năng xảy ra:
  - Signal-and-wait: *P* chờ đến khi *Q* rời monitor hoặc chờ một điều kiện khác (*)
  - Signal-and-continue: *Q* chờ đến khi *P* rời monitor hoặc chờ một điều kiện khác

52

Bài toán Ăn tối.. với monitor

- Giải quyết bài toán Ăn tối của các triết gia với monitor để không xảy ra bế tắc khi hai triết gia ngồi cạnh nhau cùng lấy đũa để ăn
- Trạng thái của các triết gia:
  enum {thinking, hungry, eating} state[5];
- Triết gia *i* chỉ có thể ăn nếu cả hai người ngồi cạnh ông ta không ăn:
  (state[(i+4)%5]!=eating) and (state[(i+1)%5]!=eating)
- Khi triết gia *i* không đủ điều kiện để ăn: cần có biến condition: condition self[5];

53

Monitor của bài toán Ăn tối...

```
monitor dp {
   enum {thinking, hungry, eating} state[5];
   condition self[5];
   void pickup(int i) {
        state[i] = hungry;
        test(i);
        if (state[i] != eating) self[i].wait();
   }
}
```

54

Simpo PDF Merge and Split Unregistered Version - http://www.simpopdf.com

Monitor của bài toán Ăn tối...

```
void putdown(int i) {
    state[i] = thinking;
    test((i+4)%5);
    test((i+1)%5);
}
initialization_code() {
    for (int i=0;i<5;i++) state[i] = thinking;
}
```

55

Monitor của bài toán Ăn tối...

```
void test(int i) {
    if ((state[(i+4)%5] != eating) &&
      (state[i] == hungry) &&
      (state[(i+1)%5] != eating)) {
        state[i] = eating;
        self[i].signal();
    }
}
```

56

Đọc thêm ở nhà

- Khái niệm về miền găng (critical region)
- Cơ chế monitor của Java:

```
public class XYZ {
    ...
    public synchronized void safeMethod() {
      ...
    }
}
```

- Toán tử wait() và notify() trong java.util.package (tương tự toán tử wait() và signal())
- Cách cài đặt monitor bằng semaphore

57

Tóm tắt

- Khái niệm đồng bộ hóa
- Khái niệm đoạn mã găng, ba điều kiện của đoạn mã găng
- Khái niệm semaphore, semaphore nhị phân
- Hiện tượng bế tắc do sử dụng sai semaphore
- Một số bài toán cổ điển trong đồng bộ hóa
- Miền găng
- Cơ chế monitor

58

Bài tập

- Chỉ ra điều kiện nào của đoạn mã găng bị vi phạm trong đoạn mã găng sau của P_i:

```
do {
   while (turn != i) ;
   CS_i;
   turn = j;
   REMAIN_i;
} while (1);
```

59

Bài tập

- Cài đặt giải pháp cho bài toán Bữa ăn tối của các triết gia trong Java bằng cách sử dụng synchronized, wait() và notify()
- Giải pháp monitor cho bài toán Bữa ăn tối... tránh được bế tắc, nhưng có thể xảy ra trường hợp tất cả các triết gia đều không được ăn. Hãy chỉ ra trường hợp này và tìm cách giải quyết bằng cơ chế monitor
- Chú ý: Sinh viên cần làm bài tập để hiểu tốt hơn về đồng bộ hóa

60

Simpo PDF Merge and Split Unregistered Version - http://www.simpopdf.com

Nguyên lý hệ điều hành

Nguyễn Hải Châu
Khoa Công nghệ thông tin
Trường Đại học Công nghệ

1

Bế tắc (Deadlock)

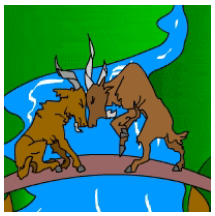
2

Định nghĩa

- Bế tắc là tình huống xuất hiện khi hai hay nhiều "hành động" phải chờ một hoặc nhiều hành động khác để kết thúc, nhưng không bao giờ thực hiện được
- Máy tính: Bế tắc là tình huống xuất hiện khi hai tiến trình phải chờ đợi nhau giải phóng tài nguyên hoặc nhiều tiến trình chờ sử dụng các tài nguyên theo một "vòng tròn" (circular chain)

3

Hai con dê qua cầu: Bế tắc



4

Bế tắc giao thông tại ngã tư

5

Bế tắc trong máy tính

- Tiến trình A:

```
{
   …
   Khóa file F1;
   ...
   Mở file F2;
   …
   Đóng F1 (mở khóa F1);
}
```

- Tiến trình B

```
{
   …
   Khóa file F2;
   ...
   Mở file F1;
   …
   Đóng F1 (mở khóa F1);
}
```

6

Simpo PDF Merge and Split Unregistered Version - http://www.simpopdf.com

Qui trình sử dụng tài nguyên

- Một tiến trình thường sử dụng tài nguyên theo các bước tuần tự sau:
  - Xin phép sử dụng (request)
  - Sử dụng tài nguyên (use)
  - Giải phóng tài nguyên sau khi sử dụng (release)

7

Điều kiện cần để có bế tắc

- Bế tắc xuất hiện nếu 4 điều kiện sau xuất hiện đồng thời (điều kiện cần):
  - C1: Loại trừ lẫn nhau (mutual exclusion)
  - C2: Giữ và chờ (hold and wait)
  - C3: Không có đặc quyền (preemption)
  - C4: Chờ vòng (circular wait)

8

C1: Loại trừ lẫn nhau

- Một tài nguyên bị chiếm bởi một tiến trình, và không tiến trình nào khác có thể sử dụng tài nguyên này

9

C2: Giữ và chờ

- Một tiến trình giữ ít nhất một tài nguyên và chờ một số tài nguyên khác rỗi để sử dụng. Các tài nguyên này đang bị một tiến trình khác chiếm giữ

10

C3: Không có đặc quyền

- Tài nguyên bị chiếm giữ chỉ có thể rỗi khi tiến trình "tự nguyện" giải phóng tài nguyên sau khi đã sử dụng xong.

11

C3: Chờ vòng

- Một tập tiến trình $\{P_0, P_1, ..., P_n\}$ có xuất hiện điều kiện "chờ vòng" nếu P_0 chờ một tài nguyên do P_1 chiếm giữ, P_1 chờ một tài nguyên khác do P_2 chiếm giữ, ..., P_{n-1} chờ tài nguyên do P_n chiếm giữ và P_n chờ tài nguyên do P_0 chiếm giữ

12

Simpo PDF Merge and Split Unregistered Version - http://www.simpopdf.com

Đồ thị cấp phát tài nguyên

- Thuật ngữ: Resource allocation graph
- Để mô tả một cách chính xác bế tắc, chúng ta sử dụng đồ thị có hướng gọi là "đồ thị cấp phát tài nguyên" $G=(V, E)$ với V là tập đỉnh, E là tập cung
- E được chia thành hai tập con $P=\{P_0, P_1, ..., P_n\}$ là tập các tiến trình trong hệ thống và $R=\{R_0, R_1, ..., R_m\}$ là tập các loại tài nguyên trong hệ thống thỏa mãn $P \cup R=E$ và $P \cap R= \varnothing$

13

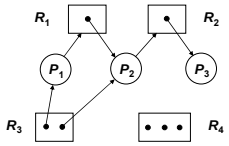
Đồ thị cấp phát tài nguyên

- Cung có hướng từ tiến trình P_i đến tài nguyên R_j, ký hiệu là $P_i \rightarrow R_j$ có ý nghĩa: Tiến trình P_i yêu cầu một *thể hiện* của R_j. Ta gọi $P_i \rightarrow R_j$ là *cung yêu cầu* (*request edge*)
- Cung có hướng từ tài nguyên R_j đến tiến trình P_i ký hiệu là $R_j \rightarrow P_i$ có ý nghĩa: Một thể hiện của tài nguyên Rj đã được cấp phát cho tiến trình Pi. Ta gọi $R_j \rightarrow P_i$ là *cung cấp phát* (*asignment edge*)

14

Đồ thị cấp phát tài nguyên

- Ký hiệu hình vẽ:
  - P_i là hình tròn
  - R_j là các hình chữ nhật với mỗi chấm bên trong là số lượng các thể hiện của tài nguyên
- Minh họa đồ thị cấp phát tài nguyên:



15

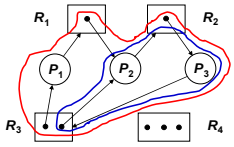
Đồ thị cấp phát tài nguyên

- Nếu không có chu trình trong đồ thị cấp phát tài nguyên: Không có bế tắc. Nếu có chu trình: *Có thể* xảy ra bế tắc.
- Nếu trong một chu trình trong đồ thị cấp phát tài nguyên, mỗi loại tài nguyên chỉ có đúng một thể hiện: Bế tắc đã xảy ra (Điều kiện cần và đủ)
- Nếu trong một chu trình trong đồ thị cấp phát tài nguyên một số tài nguyên có nhiều hơn một thể hiện: Có thể xảy ra bế tắc (Điều kiện cần nhưng không đủ)

16

Ví dụ chu trình dẫn đến bế tắc

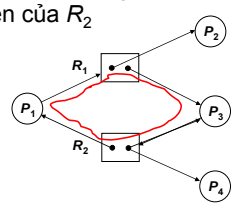
- Giả sử P_3 yêu cầu một thể hiện của R_3
- Khi đó có 2 chu trình xuất hiện:
  - $P_1 \rightarrow R_1 \rightarrow P_2 \rightarrow R_2 \rightarrow P_3 \rightarrow R_3 \rightarrow P_1$, và
  - $P_2 \rightarrow R_2 \rightarrow P_3 \rightarrow R_3 \rightarrow P_2$
- Khi đó các tiến trình P_1, P_2, P_3 bị bế tắc



17

Ví dụ chu trình không dẫn đến bế tắc

- Chu trình: $P_1 \rightarrow R_1 \rightarrow P_3 \rightarrow R_2 \rightarrow P_1$
- Bế tắc không xảy ra vì P_4 có thể giải phóng một thể hiện tài nguyên R_2 và P_3 sẽ được cấp phát một thể hiện của R_2



18

Simpo PDF Merge and Split Unregistered Version - http://www.simpopdf.com

Các phương pháp xử lý bế tắc

Các phương pháp xử lý bế tắc

- Một cách tổng quát, có 3 phương pháp:
  - Sử dụng một giao thức để hệ thống không bao giờ rơi vào trạng thái bế tắc: *Deadlock prevention* (*ngăn chặn bế tắc*) hoặc *Deadlock avoidance* (*tránh bế tắc*)
  - Có thể cho phép hệ thống bị bế tắc, phát hiện bế tắc và khắc phục nó
  - Bỏ qua bế tắc, xem như bế tắc không bao giờ xuất hiện trong hệ thống (*Giải pháp này dùng trong nhiều hệ thống, ví dụ Unix, Windows*!!)

Ngăn chặn bế tắc (Deadlock prevention)

Giới thiệu

- Ngăn chặn bế tắc (deadlock prevention) là phương pháp xử lý bế tắc, không cho nó xảy ra bằng cách làm cho ít nhất một điều kiện cần của bế tắc là C1, C2, C3 hoặc C4 không được thỏa mãn (không xảy ra)
- Ngăn chặn bế tắc theo phương pháp này có tính chất tĩnh (statically)

Ngăn chặn "loại trừ lẫn nhau"

- C1 (Loại trừ lẫn nhau): là điều kiện bắt buộc cho các tài nguyên không sử dụng chung được → *Khó làm cho C1 không xảy ra* vì các hệ thống luôn có các tài nguyên không thể sử dụng chung được

Ngăn chặn "giữ và chờ"

- C2 (Giữ và chờ): Có thể làm cho C2 không xảy ra bằng cách đảm bảo:
  - Một tiến trình luôn yêu cầu cấp phát tài nguyên chỉ khi nó không chiếm giữ bất kỳ một tài nguyên nào, hoặc
  - Một tiến trình chỉ thực hiện khi nó được cấp phát toàn bộ các tài nguyên cần thiết

Simpo PDF Merge and Split Unregistered Version - http://www.simpopdf.com

Ngăn chặn "không có đặc quyền"

- Để ngăn chặn không cho điều kiện này xảy ra, có thể sử dụng giao thức sau:
  - Nếu tiến trình P (đang chiếm tài nguyên R_1, ..., R_{n-1}) yêu cầu cấp phát tài nguyên R_n nhưng không được cấp phát ngay (có nghĩa là P phải chờ) thì tất cả các tài nguyên R_1, ..., R_{n-1} phải được "thu hồi"
  - Nói cách khác, R_1, ..., R_{n-1} phải được "giải phóng" một cách áp đặt, tức là các tài nguyên này phải được đưa vào danh sách các tài nguyên mà P đang chờ cấp phát.

25

Ngăn chặn "không có đặc quyền": mã lệnh

Tiến trình P yêu cầu cấp phát tài nguyên R_1, ..., R_{n-1}

if (R_1, ..., R_{n-1} rỗi)

then cấp phát tài nguyên cho P

else if ({R_i...R_j} được cấp phát cho Q và Q đang trong trạng thái chờ một số tài nguyên S khác)

then thu hồi {R_i...R_j} và cấp phát cho P

else đưa P vào trạng thái chờ tài nguyên R_1, ..., R_{n-1}

26

Ngăn chặn "chờ vòng"

- Một giải pháp ngăn chặn chờ vòng là đánh số thứ tự các tài nguyên và bắt buộc các tiến trình yêu cầu cấp phát tài nguyên theo số thứ tự tăng dần
- Giả sử có các tài nguyên {R_1, ..., R_n}. Ta gán cho mỗi tài nguyên một số nguyên dương duy nhất qua một ánh xạ 1-1

  $f : R \rightarrow N$, với N là tập các số tự nhiên

  Ví dụ: f(ổ cứng) = 1, f(băng từ) = 5, f(máy in) = 11

27

Ngăn chặn "chờ vòng"

- Giao thức ngăn chặn chờ vòng:
  - Khi tiến trình P không chiếm giữ tài nguyên nào, nó có thể yêu cầu cấp phát nhiều *thể hiện* của *một* tài nguyên R_i bất kỳ
  - Sau đó P chỉ có thể yêu cầu các thể hiện của tài nguyên R_j nếu và chỉ nếu $f(R_j) > f(R_i)$. Một cách khác, nếu P muốn yêu cầu cấp phát tài nguyên R_j, nó đã giải phóng tất cả các tài nguyên R_i thỏa mãn $f(R_i) \geq f(R_j)$
  - Nếu P cần được cấp phát nhiều loại tài nguyên, P phải *lần lượt* yêu cầu các thể hiện của *từng* tài nguyên đó

28

Chứng minh giải pháp ngăn chặn chờ vòng

- Sử dụng chứng minh phản chứng
- Giả sử giải pháp ngăn chặn gây ra chờ vòng {P_0, P_1, ..., P_n} trong đó P_i chờ tài nguyên R_i bị chiếm giữ bởi $P_{(i+1) \bmod n}$
- Vì P_{i+1} đang chiếm giữ R_i và yêu cầu R_{i+1}, do đó $f(R_i) < f(R_{(i+1) \bmod n})$ $\forall i$, có nghĩa là ta có:
  - $f(R_0) < f(R_1) < f(R_2) < .. < f(R_n) < f(R_0)$
  - Mâu thuẫn! $\rightarrow$ Giải pháp được chứng minh

29

Ưu nhược điểm của ngăn chặn giải pháp bế tắc

- Ưu điểm: ngăn chặn bế tắc (deadlock prevention) là phương pháp tránh được bế tắc bằng cách làm cho điều kiện cần không được thỏa mãn
- Nhược điểm:
  - Giảm khả năng tận dụng tài nguyên và giảm thông lượng của hệ thống
  - Không mềm dẻo

30

Simpo PDF Merge and Split Unregistered Version - http://www.simpopdf.com

Tránh bế tắc (Deadlock avoidance)

31

Giới thiệu

- Tránh bế tắc là phương pháp sử dụng thêm các thông tin về phương thức yêu cầu cấp phát tài nguyên để ra quyết định cấp phát tài nguyên sao cho bế tắc không xảy ra.
- Có nhiều thuật toán theo hướng này
- Thuật toán đơn giản nhất và hiệu quả nhất là: Mỗi tiến trình *P* đăng ký số thể hiện của mỗi loại tài nguyên mà *P* sẽ sử dụng. Khi đó hệ thống sẽ có đủ thông tin để xây dựng thuật toán cấp phát không gây ra bế tắc

32

Giới thiệu

- Các thuật toán như vậy kiểm tra *trạng thái cấp phát tài nguyên* một cách "động" để đảm bảo điều kiện chờ vòng không xảy ra
- Trạng thái cấp phát tài nguyên được xác định bởi số lượng tài nguyên rỗi, số lượng tài nguyên đã cấp phát và số lượng lớn nhất các yêu cầu cấp phát tài nguyên của các tiến trình
- Hai thuật toán sẽ nghiên cứu: *Thuật toán đồ thị cấp phát tài nguyên* và thuật toán *banker*

33

Trạng thái an toàn (safe-state)

- Một trạng thái (cấp phát tài nguyên) được gọi là an toàn nếu hệ thống có thể cấp phát tài nguyên cho các tiến trình theo một thứ tự nào đó mà vẫn tránh được bế tắc, hay
- Hệ thống ở trong trạng thái an toàn nếu và chỉ nếu tồn tại một *thứ tự an toàn* (safe-sequence)

34

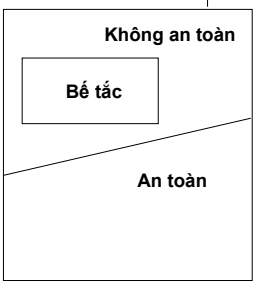
Thứ tự an toàn

- Thứ tự các tiến trình $<P_1,...,P_n>$ gọi là một thứ tự an toàn (safe-sequence) cho trạng thái cấp-phát hiện-tại nếu với mỗi P_i, yêu cầu cấp phát tài nguyên của P_i vẫn có thể được thỏa mãn căn cứ vào trạng thái của:
  - Tất cả các tài nguyên rỗi hiện có, và
  - Tất cả các tài nguyên đang bị chiếm giữ bởi tất cả các P_j $\forall j<i$.

35

Các trạng thái an toàn, không an toàn và bế tắc

- Trạng thái an toàn không là trạng thái bế tắc
- Trạng thái bế tắc là trạng thái không an toàn
- Trạng thái không an toàn có thể là trạng thái bế tắc hoặc không



36

Simpo PDF Merge and Split Unregistered Version - http://www.simpopdf.com

Ví dụ trạng thái an toàn, bế tắc

- Xét một hệ thống có 12 tài nguyên là 12 băng từ và 3 tiến trình P_0, P_1, P_2 với các yêu cầu cấp phát:
  - P_0 yêu cầu nhiều nhất 10 băng từ
  - P_1 yêu cầu nhiều nhất 4 băng từ
  - P_2 yêu cầu nhiều nhất 9 băng từ
- Giả sử tại một thời điểm t_0, P_0 đang chiếm 5 băng từ, P_1 và P_2 mỗi tiến trình chiếm 2 băng từ. Như vậy có 3 băng từ rỗi

37

Ví dụ trạng thái an toàn, bế tắc

	Yêu cầu nhiều nhất	Yêu cầu hiện tại
P_0	10	5
P_1	4	2
P_2	9	2

- Tại thời điểm t_0, hệ thống ở trạng thái an toàn
- Thứ tự <P_1, P_0, P_2> thỏa mãn điều kiện an toàn
- Giả sử ở thời điểm t_1, P_2 có yêu cầu và được cấp phát 1 băng từ: Hệ thống không ở trạng thái an toàn nữa... -> quyết đinh cấp tài nguyên cho P_2 là sai.

38

Thuật toán đồ thị cấp phát tài nguyên

- Giả sử các tài nguyên chỉ có 1 thể hiện
- Sử dụng đồ thị cấp phát tài nguyên như ở slide 16 và thêm một loại cung nữa là *cung báo trước* (claim)
- Cung báo trước $P_i \rightarrow R_j$ chỉ ra rằng P_i có thể yêu cầu cấp phát tài nguyên R_j, được biểu diễn trên đồ thị bằng các đường nét đứt
- Khi tiến trình P_i yêu cầu cấp phát tài nguyên R_j, đường nét đứt trở thành đường nét liền

39

Thuật toán đồ thị cấp phát tài nguyên

- Chú ý rằng các tài nguyên phải được thông báo trước khi tiến trình thực hiện
- Các cung báo trước sẽ phải có trên đồ thị cấp phát tài nguyên
- Tuy nhiên có thể giảm nhẹ điều kiện: cung thông báo $P_i \rightarrow R_j$ được thêm vào đồ thị nếu tất cả các cung gắn với P_i đều là cung thông báo

40

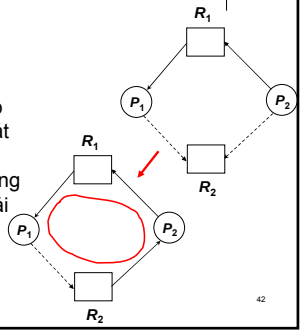
Thuật toán đồ thị cấp phát tài nguyên

- Giả sử P_i yêu cầu cấp phát R_j. Yêu cầu này chỉ có thể được chấp nhận nếu ta chuyển cung báo trước $P_i \rightarrow R_j$ thành cung cấp phát $R_j \rightarrow P_i$ và không tạo ra một chu trình
- Chúng ta kiểm tra bằng cách sử dụng thuật toán phát hiện chu trình trong đồ thị: Nếu có n tiến trình trong hệ thống, thuật toán phát hiện chu trình có độ phức tạp tính toán $O(n^2)$
- Nếu không có chu trình: Cấp phát->trạng thái an toàn, ngược lại: Trạng thái không an toàn

41

Ví dụ

- Giả sử P_1 yêu cầu cấp phát R_2
- Mặc dù R_2 rỗi nhưng chúng ta không thể cấp phát R_2, vì nếu cấp phát ta sẽ có chu trình trong đồ thị và gây ra chờ vòng → Hệ thống ở trạng thái không an toàn



42

Simpo PDF Merge and Split Unregistered Version - http://www.simpopdf.com

Thuật toán banker

- Thuật toán đồ thị phân phối tài nguyên không áp dụng được cho các hệ thống có những tài nguyên có nhiều thể hiện
- Thuật toán *banker* được dùng cho các hệ có tài nguyên nhiều thể hiện, nó kém hiệu quả hơn thuật toán đồ thị phân phối tài nguyên
- Thuật toán banker có thể dùng trong ngân hàng: Không bao giờ cấp phát tài nguyên (tiền) gây nên tình huống sau này không đáp ứng được nhu cầu của tất cả các khách hàng

43

Ký hiệu dùng trong banker

- Tài nguyên rỗi: Vector *m* thành phần *Available*, *Available*[*j*]=*k* nghĩa là có *k* thể hiện của R_j rỗi
- Max: Ma trận *nxm* xác định yêu cầu tài nguyên max của mỗi tiến trình. *Max*[*i*][*j*]=*k* có nghĩa là tiến trình P_i yêu cầu nhiều nhất *k* thể hiện của tài nguyên R_j.

44

Ký hiệu dùng trong banker

- Cấp phát: Ma trận *nxm* xác định số thể hiện của các loại tài nguyên đã cấp phát cho mỗi tiến trình. *Allocation*[*i*][*j*]=*k* có nghĩa là tiến trình P_i được cấp phát *k* thể hiện của R_j.
- Cần thiết: Ma trận *nxm* chỉ ra số lượng thể hiện của các tài nguyên mỗi tiến trình cần cấp phát tiếp. *Need*[*i*][*j*]=*k* có nghĩa là tiến trình P_i còn có thể cần thêm *k* thể hiện nữa của tài nguyên R_j.

45

Ký hiệu dùng trong banker

- Số lượng và giá trị các biến trên biến đổi theo trạng thái của hệ thống
- Qui ước: Nếu hai vector *X*, *Y* thỏa mãn *X*[*i*]≤*Y*[*i*] ∀*i* thì ta ký hiệu *X*≤*Y*.
- Giả sử *Work* và *Finish* là các vector *m* và *n* thành phần.
- *Request*[*i*] là vector yêu cầu tài nguyên của tiến trình P_i. *Request*[*i*][*j*]=*k* có nghĩa là tiến trình P_i yêu cầu *k* thể hiện của tài nguyên R_j

46

Thuật toán trạng thái an toàn

1. Khởi tạo *Work*=*Available* và *Finish*[*i*]=false ∀*i*=1..*n*
2. Tìm *i* sao cho *Finish*[*i*]==false và *Need*[*i*]≤*Work*
   Nếu không tìm được *i*, chuyển đến bước 4
3. *Work*=*Work*+*Allocation*[i], *Finish*[*i*]=true
   Chuyển đến bước 2
4. Nếu *Finish*[*i*]==true ∀*i* thì hệ thống ở trạng thái an toàn

- Độ phức tạp tính toán của thuật toán trạng thái an toàn: $O(m.n^2)$

47

Thuật toán yêu cầu tài nguyên

1. Nếu *Request*[*i*]≤*Need*[*i*], chuyển đến bước 2
   Ngược lại thông báo lỗi (không có tài nguyên rỗi)
2. Nếu *Request*[*i*]≤*Available*, chuyển đến bước 3.
   Ngược lại P_i phải chờ vì không có tài nguyên
3. Nếu việc thay đổi trạng thái giả định sau đây:
   Available=*Availalble*-*Request*[i]
   Allocation=*Allocation*+*Request*[i]
   Need[i]=*Need*[i]-*Request*[i]
   đưa hệ thống vào trạng thái an toàn thì cấp phát tài nguyên cho P_i, ngược lại P_i phải chờ *Request*[*i*] và trạng thái của hệ thống được khôi phục như cũ

48

Simpo PDF Merge and Split Unregistered Version - http://www.simpopdf.com

Ví dụ banker

- Xét một hệ thống các tiến trình và tài nguyên như sau:

	Allocation			Max			Available			Need		
	A	B	C	A	B	C	A	B	C	A	B	C
P0	0	1	0	7	5	3	3	3	2	7	4	3
P1	2	0	0	3	2	2				1	2	2
P2	3	0	2	9	0	2				6	0	0
P3	2	1	1	2	2	2				0	1	1
P4	0	0	2	4	3	3				4	3	1

49

Ví dụ banker

- Hệ thống hiện đang ở trạng thái an toàn
- Thứ tự $<P_1,P_3,P_4,P_2,P_0>$ thỏa mãn tiêu chuẩn an toàn
- Giả sử P_1 có yêu cầu: *Request*[1]=(1,0,2)
- Để quyết định xem có cấp phát tài nguyên theo yêu cầu này không, trước hết ta kiểm tra *Request*[1]≤*Available*: (1,0,2)<(3,3,2): Đúng
- Giả sử yêu cầu này được cấp phát, khi đó trạng thái giả định của hệ thống là:

50

Ví dụ banker

	Allocation			Need			Available		
	A	B	C	A	B	C	A	B	C
P0	0	1	0	7	4	3	3	3	2
P1	3	0	2	0	2	0			
P2	3	0	2	6	0	0			
P3	2	1	1	0	1	1			
P4	0	0	2	4	3	1			

- Thực hiện thuật toán trạng thái an toàn và thấy rằng thứ tự $<P_1,P_3,P_4,P_0,P_2>$. Do đó có thể cấp phát tài nguyên cho P_1 ngay.

51

Ví dụ banker

- Tuy nhiên, nếu hệ thống ở trạng thái sau thì
  - Yêu cầu (3,3,0) của P4 không thể cấp phát ngay vì các tài nguyên không rỗi
  - Yêu cầu (0,2,0) của P0 cũng không thể cấp phát ngay vì mặc dù các tài nguyên rỗi nhưng việc cấp phát sẽ làm cho hệ thống rơi vào trạng thái không an toàn
- **Bài tập**: Thực hiện kiểm tra và quyết định cấp phát hai yêu cầu trên

52

Phát hiện bế tắc

- Nếu không áp dụng phòng tránh hoặc ngăn chặn bế tắc thì hệ thống có thể bị bế tắc
- Khi đó:
  - Cần có thuật toán kiểm tra trạng thái để xem có bế tắc xuất hiện hay không
  - Thuật toán khôi phục nếu bế tắc xảy ra

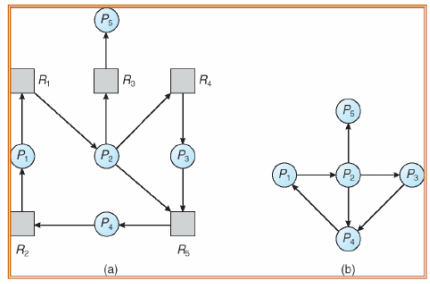
53

Tài nguyên chỉ có một thể hiện

- Sử dụng thuật toán đồ thị chờ: Đồ thị chờ có được từ đồ thị cấp phát tài nguyên bằng cách xóa các đỉnh tài nguyên và nối các cung liên quan
  - Cung $P_i \rightarrow P_j$ có nghĩa là P_i đang chờ P_j giải phóng tài nguyên mà P_i cần
  - Cung $P_i \rightarrow P_j$ tồn tại trong đồ thị chờ nếu và chỉ nếu đồ thị cấp phát tài nguyên tương ứng có hai cung $P_i \rightarrow R_q$ và $R_q \rightarrow P_j$ với R_q là tài nguyên
  - Hệ thống có bế tắc nếu đồ thị chờ có chu trình
  - Để phát hiện bế tắc: Cần cập nhật đồ thị chờ và thực hiện định kỳ thuật toán phát hiện chu trình

54

Simpo PDF Merge and Split Unregistered Version - http://www.simpopdf.com



55

Tài nguyên có nhiều thể hiện

- *Available*: Vector *m* thành phần chỉ ra số lượng thể hiện của mỗi loại tài nguyên
- *Allocation*: Ma trận *nxm* xác định số thể hiện của mỗi loại tài nguyên đang được cấp phát cho các tiến trình
- *Request*: Ma trận *nxm* xác định yêu cầu hiện tại của mỗi tiến trình. Nếu *Request*[*i*][*j*]=*k* thì tiến trình P_i yêu cầu cấp phát *k* thể hiện của tài nguyên R_j.

56

Tài nguyên có nhiều thể hiện

1. Giả sử *Work* và *Finish* là các vector *m* và *n* thành phần. Khởi tạo *Work*=*Available*. Với mỗi *i*=0..*n*-1 gán *Finish*[*i*]=false nếu *Allocation*[*i*]≠0, ngược lại gán *Finish*[*i*]=true
2. Tìm *i* sao cho *Finish*[*i*]==false và *Request*[*i*]≤*Work*. Nếu không tìm thấy *i*, chuyển đến bước 4
3. *Work*=*Work*+*Allocation*, *Finish*[*i*]=true; chuyển đến bước 2
4. Nếu *Finish*[*i*]==false với 0≤*i*≤*n*-1 thì hệ thống đang bị bế tắc (và tiến trình P_i đang bế tắc).

- Độ phức tạp tính toán của thuật toán: $O(m.n^2)$

57

Sử dụng thuật toán phát hiện

- Tần suất sử dụng phụ thuộc:
  - Tần suất xảy ra bế tắc
  - Bao nhiêu tiến trình bị ảnh hưởng bởi bế tắc?
- Sử dụng thuật toán phát hiện:
  - Định kỳ: Có thể có nhiều chu trình trong đồ thị, không biết được tiến trình/request nào gây ra bế tắc
  - Khi có yêu cầu cấp phát tài nguyên: Tốn tài nguyên CPU

58

Khôi phục khi có bế tắc

- Kết thúc tiến trình:
  - Kết thúc toàn bộ các tiến trình bị bế tắc (1)
  - Kết thúc từng tiến trình và dừng quá trình này khi bế tắc chấm dứt (2)
- Tiến trình bị kết thúc ở (2) căn cứ vào:
  - Độ ưu tiên
  - Thời gian đã thực hiện và thời gian còn lại
  - Số lượng và các loại tài nguyên đã sử dụng
  - Các tài nguyên cần cấp phát thêm
  - Số lượng các tiến trình phải kết thúc
  - Tiến trình là tương tác hay xử lý theo lô (batch)

59

Khôi phục khi có bế tắc

- Giải phóng tài nguyên một cách bắt buộc (preemption):
  - Chọn tài nguyên nào và tiến trình nào để thực hiện?
  - Khôi phục trạng thái của tiến trình đã chọn ở (1) như thế nào?
  - Làm thế nào để tránh tình trạng một tiến trình luôn bị bắt buộc giải phóng tài nguyên?

60

Simpo PDF Merge and Split Unregistered Version - http://www.simpopdf.com

Tóm tắt

- Khái niệm bế tắc
- Các điều kiện cần để có bế tắc
- Đồ thị phân phối tài nguyên
- Các phương pháp xử lý bế tắc: Ngăn chặn và tránh bế tắc (thuật toán đồ thị cấp phát tài nguyên và thuật toán banker)
- Khôi phục khi bế tắc đã xảy ra: Kết thúc tiến trình và preemption

61

Bài tập

- Thực hiện lại ví dụ phát hiện bế tắc ở trang 264 trong giáo trình
- Làm bài tập số 7.1 trong giáo trình (trang 268)
- Làm bài tập số 7.11 trong giáo trình (trang 270)

62

Simpo PDF Merge and Split Unregistered Version - http://www.simpopdf.com

Nguyên lý hệ điều hành

Nguyễn Hải Châu
Khoa Công nghệ thông tin
Trường Đại học Công nghệ

1

Quản lý bộ nhớ

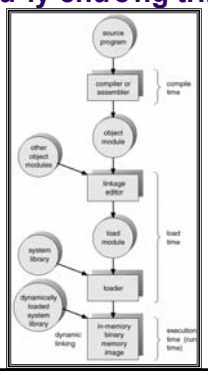
2

Giới thiệu

- Chương trình được HĐH đưa vào bộ nhớ, sau đó tạo tiến trình để thực hiện
- *Input queue* – Là hàng chờ các tiến trình trên đĩa đang chờ được đưa vào bộ nhớ để thực hiện
- Các chương trình của NSD phải qua một số bước chuẩn bị trước khi được thực hiện

3

Các bước xử lý chương trình NSD



4

Chuyển đổi địa chỉ

Có 3 cách chuyển đổi địa chỉ lệnh và dữ liệu của chương trình vào bộ nhớ:

- **Khi dịch chương trình (compile-time)**: Sinh mã có địa chỉ cố định; phải dịch lại nếu cần thay đổi địa chỉ.
- **Khi nạp chương trình (load-time)**: Phải sinh mã có thể định vị lại nếu như địa chỉ bộ nhớ không được biết ở thời điểm dịch chương trình
- **Khi thực hiện chương trình (execution-time)**: Ánh xạ địa chỉ khi chương trình được thực hiện nếu như tiến trình có thể chuyển giữa các segment bộ nhớ. Cần có hỗ trợ từ phần cứng (ví dụ thanh ghi *base* và *limit*)

5

Không gian địa chỉ logic (ảo) và địa chỉ vật lý (địa chỉ thật)

- Để quản lý bộ nhớ một cách hoàn chỉnh, cần có hai cách nhìn địa chỉ khác nhau:
  - *Địa chỉ logic* (*Logical address*) – sinh bởi CPU; còn gọi là địa chỉ ảo (*virtual address*).
  - *Địa chỉ vật lý* (*Physical address*); còn gọi là địa chỉ thật – sinh bởi đơn vị quản lý bộ nhớ
- Địa chỉ thật và ảo giống nhau trong lược đồ ánh xạ địa chỉ "compile-time" và "load-time" và khác nhau trong "execution-time".

6

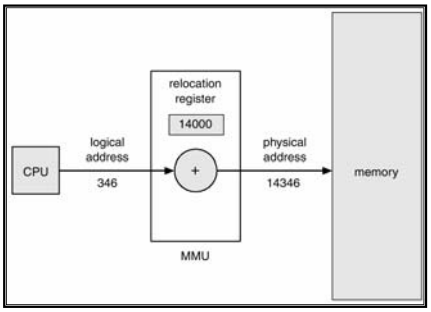
Simpo PDF Merge and Split Unregistered Version - http://www.simpopdf.com

Đơn vị quản lý bộ nhớ (MMU)

- Là thiết bị phần cứng dùng để ánh xạ địa chỉ ảo sang địa chỉ vật lý
- Trong MMU, có thanh ghi relocation (định vị lại) dùng để tính toán địa chỉ thực (vật lý) từ địa ảo của một tiến trình của NSD
- Chương trình của NSD làm việc trên địa chỉ ảo và không bao giờ biết địa chỉ vật lý

7

Sử dụng thanh ghi relocation



8

Nạp chương trình động (Dynamic loading)

- Các hàm, thủ tục không được nạp cho đến khi được sử dụng (được gọi đến)
- Cách nạp động này sử dụng bộ nhớ hiệu quả hơn: Các hàm, thủ tục không dùng đến không bao giờ được nạp vào bộ nhớ
- Hữu ích khi có một đoạn mã lớn được sử dụng với tần suất thấp
- Không cần có các đặc điểm đặc biệt từ hệ điều hành về phần cứng/phần mềm

9

Liên kết động (dynamic linking) và thư viện chung (shared library)

- Liên kết chương trình được thực hiện khi chương trình được thực hiện.
- Một đoạn mã ngắn (stub) được dùng để định vị các hàm tương ứng đã được nạp sẵn trong bộ nhớ
- Stub được thay thế bằng địa chỉ của hàm/thủ tục cần thiết, sau đó thực hiện hàm/thủ tục đó
- HĐH cần kiểm tra các hàm/thủ tục đã được nạp chưa
- Liên kết động rất có lợi khi xây dựng các thư viện chung, khi sửa lỗi (các miếng vá – patch)

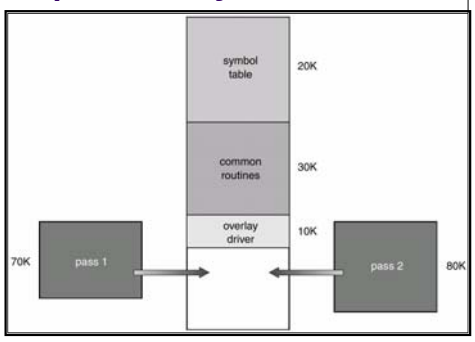
10

Overlays

- Chỉ lưu trong bộ nhớ các phần lệnh và dữ liệu phải sử dụng trong suốt quá trình thực hiện
- Sử dụng khi tiến trình có yêu cầu bộ nhớ lớn hơn dung lượng được cấp phát.
- Cài đặt bởi người sử dụng, lập trình overlays rất phức tạp

11

Ví dụ về overlays



12

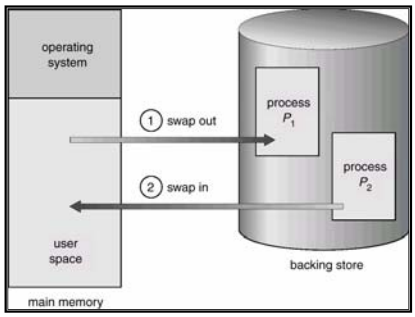
Simpo PDF Merge and Split Unregistered Version - http://www.simpopdf.com

Swapping

- *Swapping*: Đưa một tiến trình ra *backing store* để lưu trữ tạm thời, sau đó đưa trở lại bộ nhớ trong để thực hiện.
  - *Backing store* – Vùng đĩa có tốc độ truy cập cao, đủ lớn để chứa được nhiều tiến trình của NSD, có thể truy cập trực tiếp
- *Roll out, roll in* – Phương án swap dành cho lập lịch có ưu tiên: Tiến trình ưu tiến thấp: *roll out*, ưu tiên cao: *roll in* để tiếp tục thực hiện
- Thời gian swap tỷ lệ thuận với dung lượng bộ nhớ được swap vào/ra
- UNIX, Linux, and Windows sử dụng swapping

13

Minh họa swapping



14

Cấp phát liên tục (Contiguous allocation)

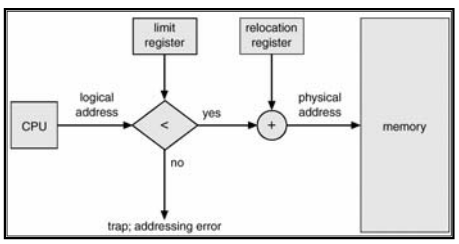
15

Cấp phát bộ nhớ liên tục

- Bộ nhớ trong thường được chia thành 2 phần:
  - Phần dành cho hệ điều hành (resident) thường dùng phần thấp của bộ nhớ với các ngắt
  - NSD dùng phần cao của bộ nhớ. Mỗi tiến trình được cấp phát một *vùng liên tục của bộ nhớ*
- Thanh ghi *relocation* dùng để bảo vệ các tiến trình của NSD và để tránh thay đổi mã và dữ liệu của HĐH
- Thanh ghi *relocation* chứa giá trị nhỏ nhất của địa chỉ vật lý, thanh ghi *limit* chứa độ lớn của miền địa chỉ ảo (*địa chỉ ảo* < *limit*)

16

Minh họa thanh ghi *relocation*, *limit*



17

Cấp phát liên tục (tiếp): MFT

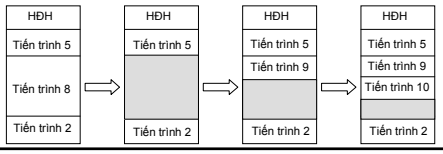
- Bộ nhớ được chia thành các khối với cỡ cố định, mỗi tiến trình được cấp phát một khối
- Khi tiến trình kết thúc, khối bộ nhớ đã cấp phát cho tiến trình được giải phóng để cấp phát cho tiến trình khác
- Mức độ đa chương trình bị hạn chế bởi các khối
- Cỡ của tiến trình bị hạn chế bởi cỡ của khối
- Các HĐH/máy tính sử dụng MFT: IBM/360

18

Simpo PDF Merge and Split Unregistered Version - http://www.simpopdf.com

Cấp phát liên tục (tiếp): MVT

- Cấp phát MVT
  - *Hole* – khối bộ nhớ rỗi; các khối rỗi với kích cỡ khác nhau rải rác trong bộ nhớ
  - Một tiến trình sẽ được cấp phát một khối bộ nhớ đủ lớn để thực hiện
  - HĐH có thông tin về các khối đã cấp phát và khối rỗi



19

Các chiến lược cấp phát

- **First-fit**: Cấp phát khối nhớ đầu tiên thỏa mãn điều kiện.
- **Best-fit**: Cấp phát khối nhớ bé nhất thỏa mãn điều kiện: Phải duyệt toàn bộ danh sách khối nhớ
- **Worst-fit**: Cấp phát khối nhớ lớn nhất thỏa mãn điều kiện: Phải duyệt toàn bộ danh sách khối nhớ
- First-fit và best-fit tốt hơn worst-fit theo nghĩa tốc độ và tận dụng bộ nhớ

20

Vấn đề phân mảnh

- **External Fragmentation (*Phân mảnh ngoài*)**: Tổng dung lượng đáp ứng được nhu cầu cấp phát nhưng các khối không liên tục
- **Internal Fragmentation (*Phân mảnh trong*)** – Dung lượng bộ nhớ đã cấp phát cho tiến trình không được sử dụng hết
- Giảm phân mảnh ngoài: Compaction
  - Xáo trộn các khối để các khối nhớ rỗi nằm liên tục
  - Compaction chỉ thực hiện được khi relocation là động, và được thực hiện ở execution-time
- Ví dụ: Tiện ích Defragmentation của Windows

21

Phân trang (Paging)

22

Phân trang (paging)

- Phân trang là chiến lược cấp phát bộ nhớ cho phép không gian địa chỉ logic của một tiến trình có thể không liên tục; tiến trình được cấp phát bộ nhớ vật lý khi có bộ nhớ rỗi
- Bộ nhớ vật lý được chia thành các frame cỡ cố định, nhỏ (là lũy thừa của 2, ví dụ 512, 1024, 8192)
- Chia bộ nhớ ảo thành các khối *cùng cỡ* gọi là trang (page)
- HĐH có danh sách các frame rỗi
- Để thực hiện một chương trình cỡ *n* trang, cần tìm *n* frame rỗi để nạp chương trình
- Có một bảng trang để ánh xạ trang→frame
- Bảng trang: chung trong HĐH, mỗi tiến trình có một copy

23

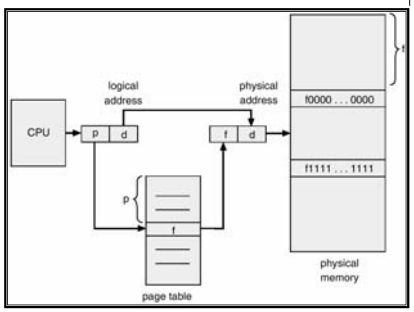
Cách đánh địa chỉ theo trang

- Địa chỉ được đánh một cách phân cấp:
  - *Số hiệu trang* (*Page number* - *p*) – Được sử dụng làm chỉ số đến phần tử trong bảng trang chứa địa chỉ cơ sở của các frame trong bộ nhớ vật lý
  - *Offset trang* (*Page offset* - *d*) – Địa chỉ tương đối trong trang
- Địa chỉ ảo có *m* bit, sử dụng *m-n* bit cao làm số hiệu trang và *n* bit thấp làm offset
- Không có *phân mảnh ngoài*, có *phân mảnh trong*:
  - Giảm cỡ trang→Giảm phân mảnh trong→Giảm hiệu năng
  - Tăng cỡ trang→Tăng hiệu suất→Tăng phân mảnh trong

24

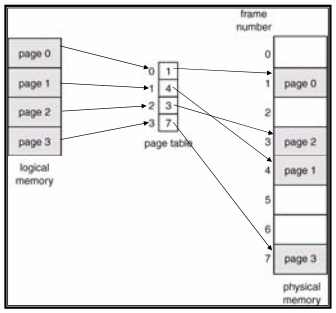
Simpo PDF Merge and Split Unregistered Version - http://www.simpopdf.com

Chuyển đổi địa chỉ



25

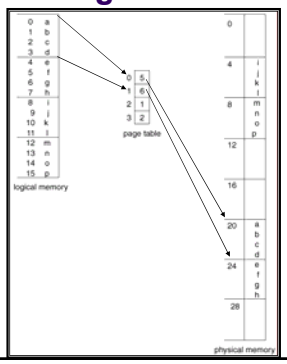
Ví dụ phân trang 1



26

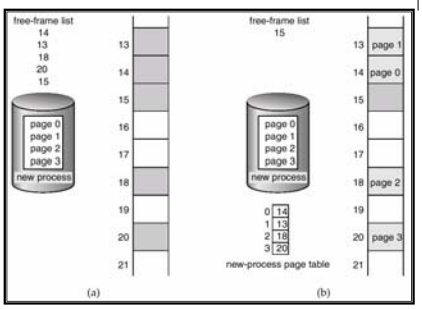
Ví dụ phân trang 2

Cỡ của một trang là 4 bytes



27

Bảng frame rỗi



Trước cấp phát Sau cấp phát

28

Cài đặt bảng trang

- Bảng trang được lưu ở bộ nhớ trong
- *Thanh ghi cơ sở bảng trang* (*page-table base register*) (PTBR) trỏ đến bảng trang
- *Thanh ghi độ dài bảng trang* (*page-table length register*) (PTLR) lưu cỡ bảng trang
- Sử dụng bảng trang, mọi thao tác truy cập dữ liệu/lệnh cần tới 2 lần truy cập bộ nhớ (1 cho bảng trang, 1 cho dữ liệu/lệnh)

29

Cài đặt bảng trang (tiếp)

- Truy cập bộ nhớ hai lần: Giảm tốc độ
- Giải quyết vấn đề 2 lần truy cập bộ nhớ: Sử dụng phần cứng cache có tốc độ truy cập cao gọi là bộ nhớ kết hợp (*associative memory*) hoặc vùng đệm hỗ trợ chuyển đổi (*translation look-aside buffers -TLB*)
- Mỗi phần tử trong TLB có hai phần: khóa và giá trị
- Số lượng các phần tử của TLB thường từ 64 đến 1024

30

Simpo PDF Merge and Split Unregistered Version - http://www.simpopdf.com

Bộ nhớ kết hợp

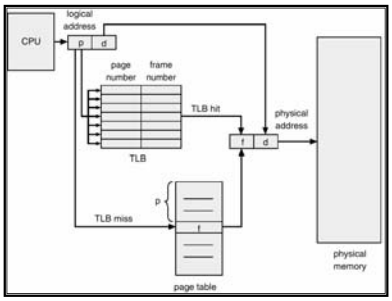
- Bộ nhớ kết hợp

Page #	Frame #

- Chuyển đổi địa chỉ (A´, A´´)
  if A´ nằm trong thanh ghi kết hợp, lấy *frame#*.
  else lấy *frame#* từ bảng trang trong bộ nhớ

31

Phân trang phần cứng với TLB



32

Thời gian truy cập hiệu quả

- Thời gian tìm kiếm ở thanh ghi kết hợp = ε (đơn vị thời gian)
- Thời gian truy cập bộ nhớ là *n* đơn vị thời gian
- Hit ratio: Số phần trăm (%) địa chỉ trang được tìm thấy ở các thanh ghi kết hợp/TLB
- Hit ratio = α
- Thời gian truy cập hiệu quả (EAT):
  EAT = $(n + \varepsilon)\,\alpha + (2n + \varepsilon)(1 - \alpha) = 2n + \varepsilon - \alpha n$

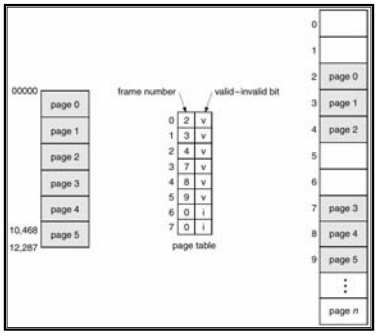
33

Bảo vệ bộ nhớ

- Bộ nhớ được bảo vệ nhờ kết hợp bit bảo vệ trong mỗi phần tử ở bảng trang
- Bit hợp lệ-không hợp lệ (*valid-invalid*) kết nối với mỗi phần tử trong bảng trang:
  - "valid" chỉ ra rằng trang thuộc không gian địa chỉ logic của tiến trình → trang hợp lệ
  - "invalid" chỉ ra rằng trang không thuộc không gian địa chỉ logic của tiến trình

34

Ví dụ bit valid (v)/invalid (i) trong bảng trang



35

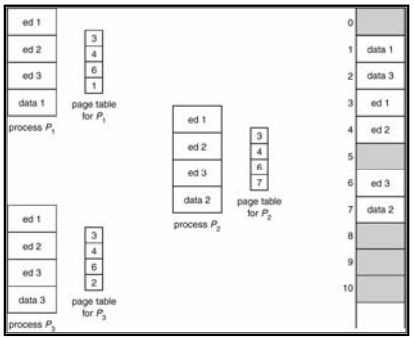
Các trang chung

- Mã dùng chung
  - Nhiều tiến trình (soạn thảo, compiler...) có thể dùng chung các đoạn mã reentrant (đoạn mã không tự thay đổi chính nó)
  - Đoạn mã chung phải xuất hiện ở cùng một vị trí địa chỉ trong không gian địa chỉ logic/ảo của tất cả các tiến trình
- Mã lệnh và dữ liệu riêng
  - Mỗi tiến trình có một bản riêng chứa lệnh và dữ liệu
  - Các trang chứa lệnh và dữ liệu riêng có thể ở bất kỳ vị trí nào trong không gian địa chỉ của tiến trình

36

Simpo PDF Merge and Split Unregistered Version - http://www.simpopdf.com

Ví dụ các trang chung



37

Cấu trúc bảng trang

Bảng trang phân cấp

Bảng trang băm

Bảng trang ngược

38

Bảng trang phân cấp

- Bộ nhớ máy tính lớn (2^{32}-2^{64} bytes): Nếu dùng bảng trang một cấp thì bảng trang có cỡ rất lớn: Tốn bộ nhớ, tìm kiếm chậm
- Không gian địa chỉ logic được quản lý bởi nhiều bảng trang ở nhiều cấp
- Một kỹ thuật đơn giản nhất là bảng trang hai cấp. Có thể có bảng trang hai, ba, bốn cấp

39

Ví dụ bảng trang hai cấp

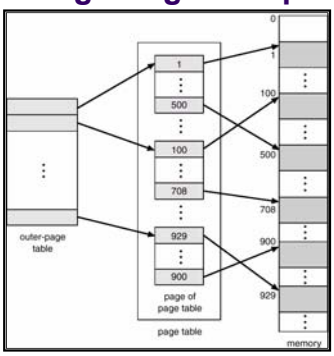
- Địa chỉ logic (trên máy 32-bit, trang cỡ 4K=2^{12}) được chia thành:
  - Địa chỉ trang: 20 bits.
  - Địa chỉ offset: 12 bits.
- Bảng trang 2 cấp (địa chỉ 20 bit) được chia thành:
  - 10-bit địa chỉ trang cấp 1
  - 10-bit địa chỉ trang cấp 2
- Khi đó địa chỉ logic có dạng:

Địa chỉ trang		Offset
p_1	p_2	d
10	10	12

trong đó p_1 là chỉ số đến bảng trang ngoài, p_2 là chỉ số đến trang (thực sự) ở bảng trang ngoài

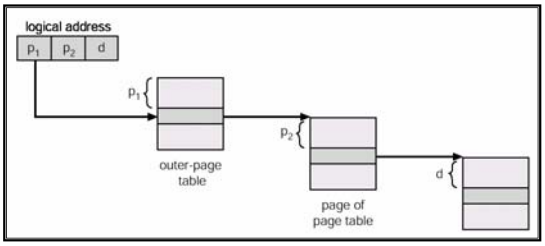
40

Sơ đồ bảng trang hai cấp



41

Tính địa chỉ với bảng trang hai cấp



42

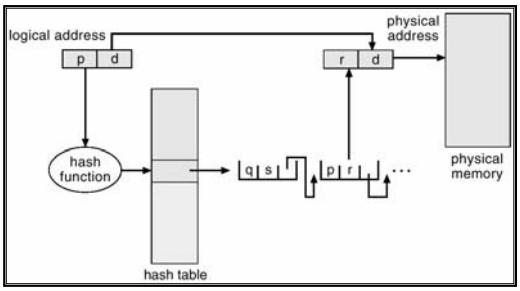
Simpo PDF Merge and Split Unregistered Version - http://www.simpopdf.com

Bảng trang băm

- Thường sử dụng khi địa chỉ > 32 bit
- Số hiệu/địa chỉ trang được băm trong bảng trang. Bảng trang này chứa dãy các phần tử (các trang) được băm ở cùng một vị trí
- Số hiệu trang được so sánh trong dãy các trang được băm ở cùng một vị trí để từ đó tìm ra frame vật lý

43

Bảng trang băm



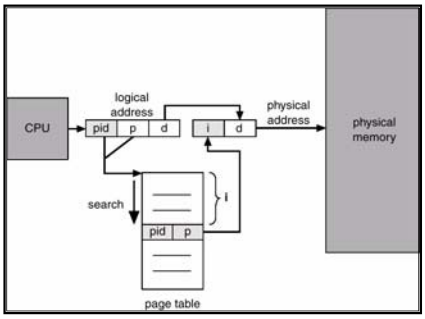
44

Bảng trang ngược

- Giải pháp giảm bộ nhớ lưu các bảng trang
- Mỗi phần tử trong bảng ứng với một frame
- Mỗi phần tử chứa địa chỉ ảo của trang và thông tin về tiến trình đang sử dụng trang đó
- Giảm dung lượng bộ nhớ cần để lưu các bảng trang, nhưng tăng thời gian cần để tìm trong bảng khi cần tham chiếu đến một trang
- Sử dụng bảng băm để hạn chế số lần tìm kiếm trong các phần tử bảng trang

45

Kiến trúc bảng trang ngược



46

Phân đoạn (Segmentation)

47

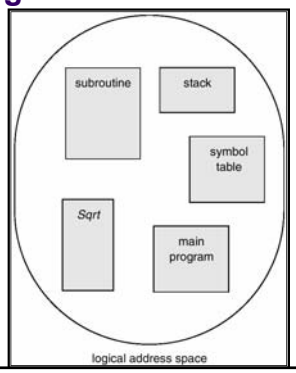
Phân đoạn

- Phương thức quản lý bộ nhớ cho phép NSD "nhìn" bộ nhớ một cách dễ dàng dưới góc độ lập trình
- Một chương trình gồm nhiều phân đoạn, mỗi phân đoạn thể hiện dưới góc độ lập trình ở dạng:
  - main program, // Chương trình chính
  - function, // Các hàm
  - method, // Các phương thức
  - object, // Các đối tượng, lớp
  - local/global variables, // Các biến
  - common block, // Các khối chung
  - stack, // Ngăn xếp
  - symbol table, arrays // Bảng ký hiệu, mảng

48

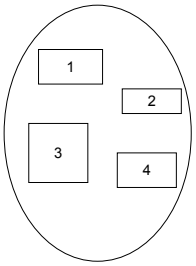
Simpo PDF Merge and Split Unregistered Version - http://www.simpopdf.com

Chương trình nhìn từ NSD

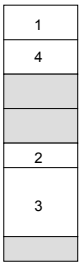


49

Phân đoạn: Cách nhìn logic



Không gian địa chỉ của NSD



Không gian bộ nhớ vật lý

50

Kiến trúc phân đoạn

- Địa chỉ ảo/logic là một bộ đôi:<segment, offset>
- Bảng phân đoạn (*segment table*) – ánh xạ địa chỉ vật lý 2 cấp; mỗi phần tử bảng có:
  - *base*: Địa chỉ vật lý bắt đầu của phân đoạn (segment)
  - *limit*: Độ dài của phân đoạn (segment).

51

Kiến trúc phân đoạn (tiếp)

- Thanh ghi cơ sở bảng phân đoạn (*Segment-table base register STBR*) trỏ đến base
- Thanh ghi độ dài bảng phân đoạn (*Segment-table length register - STLR*) chỉ ra số lượng phân đoạn được sử dụng trong tiến trình;
- Số hiệu phân đoạn *s* là hợp lệ nếu thỏa mãn điều kiện: *s* < STLR.

52

Kiến trúc phân đoạn (tiếp)

- Định vị lại (relocation)
  - Động
  - Sử dụng bảng phân đoạn
- Dùng chung (sharing)
  - Có các phân đoạn dùng chung
  - Sử dụng cùng một số hiệu phân đoạn (segment number)
- Cấp phát (allocation)
  - first fit/best fit
  - Phân mảnh ngoài

53

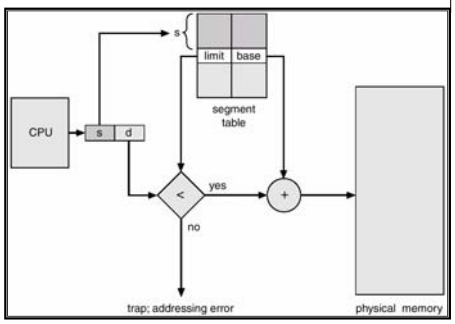
Kiến trúc phân đoạn (tiếp)

- Bảo vệ bộ nhớ:Mỗi phân đoạn có:
  - Bit kiểm tra = 0 ⇒ phân đoạn không hợp lệ
  - read/write/execute privileges
- Protection bits associated with segments; code sharing occurs at segment level.
- Do phân đoạn có cỡ biến đổi → Gặp vấn đề tương tự trong cấp phát bộ nhớ liên tục
- Kết hợp phân đoạn với phân trang để tăng hiệu quả sử dụng bộ nhớ, dễ cấp phát hơn (ví dụ: MULTICS, Intel 386)

54

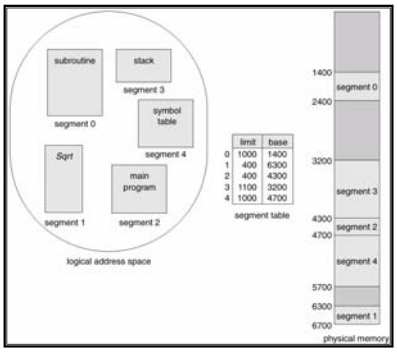
Simpo PDF Merge and Split Unregistered Version - http://www.simpopdf.com

Phần cứng phân đoạn



55

Ví dụ phân đoạn



56

Tóm tắt

- Địa chỉ logic (ảo)/Địa chỉ vật lý (thật)
- Các phương án ánh xạ địa chỉ của chương trình vào bộ nhớ
- Cấp phát bộ nhớ liên tục, phân mảnh, các chiến lược cấp phát first-fit, best-fit, worst-fit
- Phân trang
  - Trang, frame
  - Bảng trang, bảng trang phân cấp, bảng trang ngược
- Phân đoạn, bảng phân đoạn

57

Simpo PDF Merge and Split Unregistered Version - http://www.simpopdf.com

Nguyên lý hệ điều hành

Nguyễn Hải Châu
Khoa Công nghệ thông tin
Trường Đại học Công nghệ

1

Bộ nhớ ảo (Virtual Memory)

Yêu cầu phân trang
Tạo tiến trình
Thay thế trang
Cấp phát frame
Thrashing

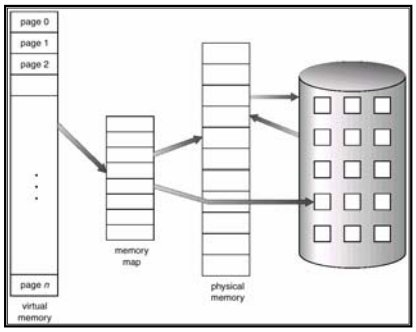
2

Virtual memory (Bộ nhớ ảo)

- **Bộ nhớ ảo** – tách biệt bộ nhớ logic và vật lý.
  - Cho phép tiến trình có cỡ lớn hơn bộ nhớ trong có thể thực hiện được
  - Không gian địa chỉ ảo có thể lớn hơn nhiều so với không gian địa chỉ vật lý (về dung lượng)
  - Cho phép các tiến trình sử dụng chung không gian địa chỉ
  - Cho phép tạo tiến trình hiệu quả hơn
- Bộ nhớ ảo có thể được cài đặt thông qua:
  - Yêu cầu phân trang (demand paging)
  - Yêu cầu phân đoạn (demand segmentation)

3

Minh họa bộ nhớ ảo có dung lượng lớn hơn bộ nhớ vật lý



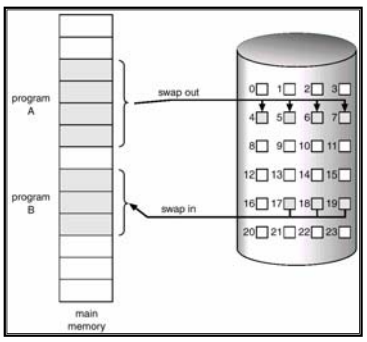
4

Yêu cầu trang (demand paging)

- Chỉ đưa một trang vào bộ nhớ khi cần thiết
  - Giảm thao tác các vào ra
  - Tiết kiệm bộ nhớ
  - Đáp ứng nhanh
  - Tăng được số người sử dụng (tiến trình)
- Khi cần một trang ⇒ tham chiếu đến nó
  - Tham chiếu lỗi ⇒ Hủy bỏ
  - Không nằm trong bộ nhớ ⇒ Đưa trang vào bộ nhớ

5

Chuyển một trang (trong bộ nhớ) ra vùng đĩa liên tục

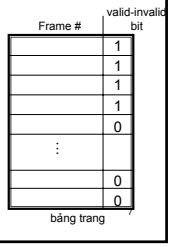


6

Simpo PDF Merge and Split Unregistered Version - http://www.simpopdf.com

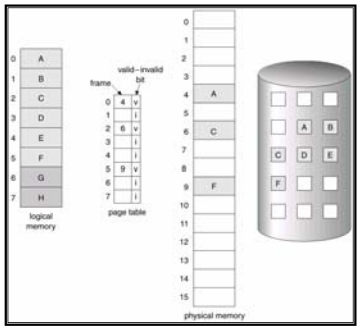
Valid-Invalid Bit

- Mỗi phần tử bảng trang có một bit hợp lệ/không hợp lệ (1: trong bộ nhớ, 0: không trong bộ nhớ)
- Khởi đầu: valid–invalid bằng 0.
- Ví dụ bảng trang+bit invalid/valid:
- Khi tính địa chỉ, nếu valid–invalid ở bảng trang là 0: ⇒ lỗi trang (page-fault trap)



7

Bảng trang với một số trang không nằm trong bộ nhớ



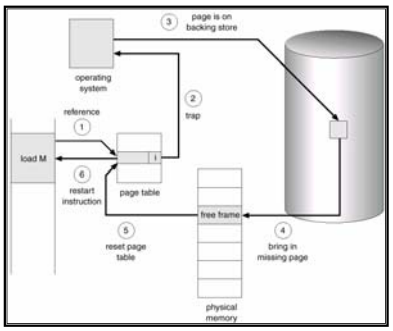
8

Xử lý page-fault (lỗi trang)

- HĐH sẽ kiểm tra nguyên nhân lỗi:
  - Lỗi từ tiến trình: kết thúc tiến trình, hoặc
  - Trang không nằm trong bộ nhớ: Thực hiện tiếp:
- Tìm một frame rỗi và đưa trang vào bộ nhớ
- Sửa lại bảng trang (bit = valid)
- Thực hiện lại lệnh tham chiếu trang
- Vấn đề hiệu năng: Nếu tại một thời điểm có yêu cầu nhiều trang (ví dụ: Một trang cho lệnh và vài trang cho dữ liệu)?

9

Các bước xử lý page-fault



10

Nếu không có frame rỗi?

- Thực hiện thay thế trang – swap out một số trang đang ở trong bộ nhớ nhưng hiện tại không được sử dụng
  - Thuật toán nào tốt?
  - Hiệu năng: Cần một thuật toán có ít page-fault nhất để hạn chế vào/ra
- Nhiều trang có thể được đưa vào bộ nhớ tại cùng một thời điểm.
- Thuật toán thay thế trang: FIFO, Optimal, LRU, LRU-approximation

11

Hiệu năng của yêu cầu trang

- Tỷ lệ page-fault là p: $0 \le p \le 1.0$
  - nếu p = 0: Không có page-fault
  - nếu p = 1, mọi yêu cầu truy cập đến trang đều gây ra page-fault
- Effective Access Time (EAT)

  EAT = (1 – p) x memory access
  + p (page fault overhead
  + [swap page out]
  + swap page in
  + restart overhead)

12

Simpo PDF Merge and Split Unregistered Version - http://www.simpopdf.com

Tạo tiến trình

- Bộ nhớ ảo có ưu điểm khi khởi tạo một tiến trình mới:

  - Copy-on-Write (Chỉ tạo copy của trang khi có thay đổi)

  - Memory-Mapped Files (Các file ánh xạ bộ nhớ)

13

Copy-on-Write

- Copy-on-Write (COW) cho phép tiến trình cha và con dùng chung trang trong bộ nhớ khi mới khởi tạo tiến trình con
- Chỉ khi nào một trong hai tiến trình sửa đổi trang dùng chung, thì trang đó mới được copy một bản mới
- COW làm cho việc tạo tiến trình hiệu quả hơn: Chỉ các trang bị sửa đổi mới được copy
- Các trang rỗi được cấp phát từ một tập hợp (pool) các trang được xóa trắng với số 0

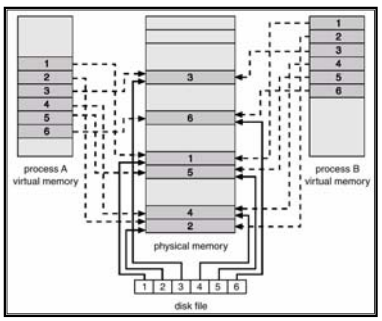
14

Các file ánh xạ bộ nhớ

- Các file được xem như một phần bộ nhớ trong bằng các ánh xạ một khối đĩa vào một trang trong bộ nhớ
- Khởi đầu các file được đọc khi có yêu cầu trang: Một phần của file (cỡ=cỡ trang) được đọc vào bộ nhớ
- Các thao tác đọc/ghi trên file sau đó được xem như đọc/ghi trong bộ nhớ
- Đơn giản hóa việc truy cập file thông qua bộ nhớ hơn là sử dụng các hàm hệ thống **read()** và **write()**.
- Cho phép các tiến trình có thể ánh xạ chung một file, do đó cho phép các trang dùng chung trong bộ nhớ

15

Ví dụ file ánh xạ bộ nhớ



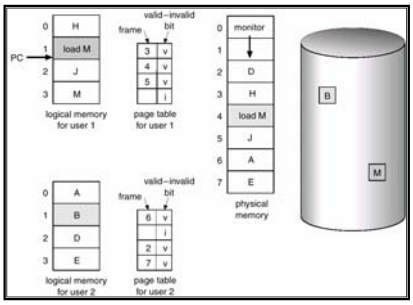
16

Thay thế trang

- Thay thế trang dùng để tránh thực hiện nhiều lần cấp phát mỗi khi có page-fault
- Sử dụng *modify bit* để giảm chi phí (overhead) vào/ra với các trang: Chỉ các trang có thay đổi mới được ghi ra đĩa
- Thay thế trang là một trong các yếu tố xóa đi sự khác biệt của bộ nhớ ảo và thật: Tiến trình lớn hơn dung lượng bộ nhớ trong có thể thực hiện được

17

Ví dụ: Yêu cầu thay thế trang



18

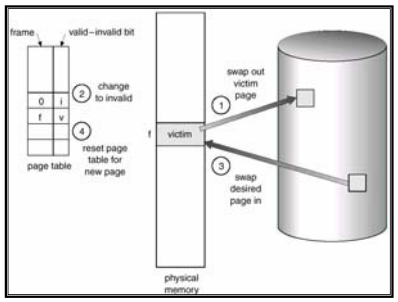
Simpo PDF Merge and Split Unregistered Version - http://www.simpopdf.com

Cơ sở thay thế trang

1. Tìm vị trí của trang *p* cần thay trên đĩa
2. Tìm một frame rỗi *f*:
   1. Nếu có frame rỗi: Sử dụng frame đó
   2. Nếu không có frame rỗi: Sử dụng thuật toán thay thế trang để đưa một trang trong bộ nhớ ra để sử dụng frame ứng với trang đó
3. Đọc trang *p* vào frame *f* vừa tìm được và cập nhật bảng trang, bảng frame
4. Lặp lại quá trình này

19

Thay thế trang



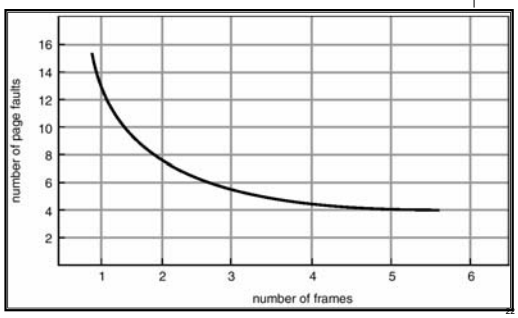
20

Thuật toán thay thế trang

- Cần tỷ lệ page-fault thấp nhất
- Đánh giá thuật toán: Thực hiện trên một danh sách các yêu cầu truy cập bộ nhớ và tính số lượng các page-fault
- Trong tất cả các ví dụ, ta sử dụng danh sách yêu cầu truy cập bộ nhớ:
  1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5.

21

Đồ thị page-fault



22

Thuật toán FIFO

- Yêu cầu truy cập: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
- Bộ nhớ VL có 3 frame: 9 page-faults
- Bộ nhớ VL có 4 frame: 10 page-faults

9 page faults

1	1		4	5
2	2		1	3
3	3		2	4

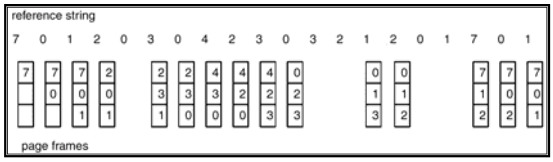
- Thay thế FIFO – Belady's anomaly
  - Có nhiều frame ⇒ ít page-fault

10 page faults

1	1		5	4
2	2		1	5
3	3		2	
4	4		3	

23

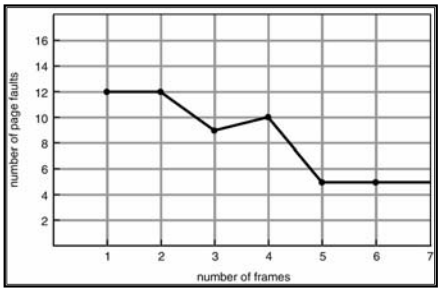
Thay thế trang FIFO



24

Simpo PDF Merge and Split Unregistered Version - http://www.simpopdf.com

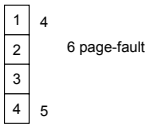
FIFO Illustrating Belady's Anamoly



25

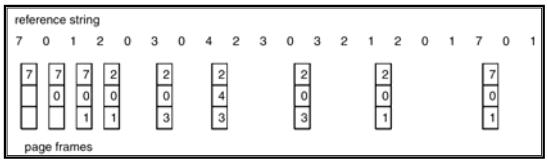
Thuật toán tối ưu

- Thay thế các trang *sẽ không được sử dụng* trong khoảng thời gian dài nhất
- Danh sách yêu cầu truy cập: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5; có 4 frame



26

Thay thế trang tối ưu



27

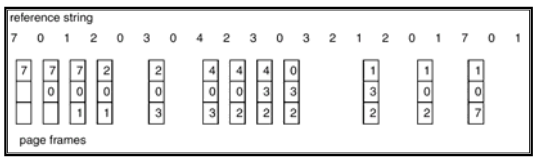
Thuật toán LRU (Least Recently Used)

- Thay thế trang *không được sử dụng lâu nhất*
- Danh sách yêu cầu truy cập: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

- Cài đặt sử dụng biến đếm
  - Mỗi trang có một biến đếm; mỗi khi trang được truy cập, gán giá trị đồng hồ thời gian cho biến đếm.
  - Khi một cần phải thay thế trang, căn cứ vào giá trị các biến đếm của trang: Cần tìm kiếm trong danh sách các trang

28

Thay thế trang LRU



29

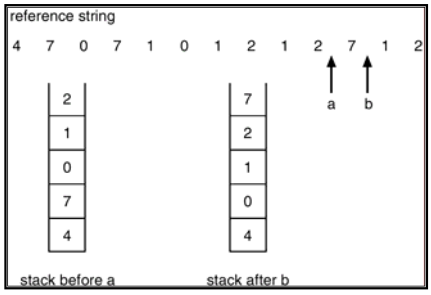
Thuật toán LRU (tiếp)

- Cài đặt sử dụng ngăn xếp: Sử dụng một ngăn xếp (stack) lưu các số hiệu trang ở dạng danh sách móc nối kép:
  - Khi trang được tham chiếu đến:
    - Chuyển trang lên đỉnh ngăn xếp
    - Cần phải thay đổi 6 con trỏ
  - Khi thay thế trang không cần tìm kiếm

30

Simpo PDF Merge and Split Unregistered Version - http://www.simpopdf.com

Sử dụng ngăn xếp để ghi lại trang vừa mới được sử dụng



31

Thuật toán LRU xấp xỉ

- Thuật toán bit tham chiếu
- Sử dụng bit tham chiếu đánh dấu các trang đã được sử dụng/chưa được sử dụng
  - Mỗi trang có 1 bit được khởi tạo bằng 0
  - Khi trang được tham chiếu đến, đặt bit bằng 1
  - Không biết thứ tự sử dụng các trang

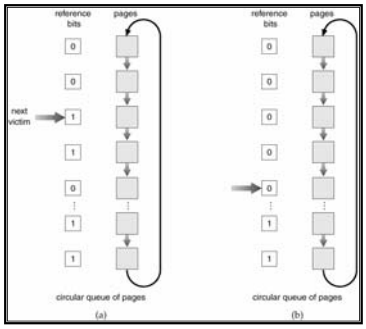
32

Thuật toán LRU xấp xỉ (tiếp)

- Thuật toán "Second-chance"
  - Là một thuật toán kiểu FIFO
  - Cần sử dụng bit tham chiếu
  - Thay thế theo thứ tự thời gian
  - Nếu trang cần được thay thế (theo thứ tự thời gian) có bit tham chiếu là 1 thì:
    - Đặt bit tham chiếu bằng 0, để trang đó trong bộ nhớ, chưa thay thế ngay (*second chance*)
    - Thay thế trang tiếp theo (theo thứ tự thời gian) tuân theo qui tắc tương tự
    - Có thể cài đặt bằng buffer vòng

33

Thuật toán second-chance



34

Các thuật toán đếm

- Sử dụng biến đếm để đếm số lần tham chiếu đến trang
- Thuật toán LFU (Least Frequently Used): Thay thế các trang có giá trị biến đếm nhỏ nhất
- Thuật toán MFU (Most Frequently Used): Ngược lại với LFU, dựa trên cơ sở: Các trang ít được sử dụng nhất (giá trị biến đếm nhỏ nhất) là các trang vừa được đưa vào bộ nhớ trong

35

Cấp phát các frame

- Mỗi tiến trình cần một số lượng tối thiểu các trang để thực hiện được
- Ví dụ: IBM 370 cần 6 để thực hiện lệnh SS MOVE:
  - Lệnh dài 6 bytes, có thể chiếm 2 trang.
  - 2 trang để thao tác **from**.
  - 2 trang để thao tác **to**.
- Hai cách cấp phát:
  - Cấp phát cố định
  - Cấp phát ưu tiên

36

Simpo PDF Merge and Split Unregistered Version - http://www.simpopdf.com

Cấp phát cố định

- Cấp phát bình đẳng: nếu cấp phát 100 frame cho 5 tiến trình, mỗi tiến trình có 20 frame.
- Cấp phát tỷ lệ: Dựa theo cỡ tiến trình.
- Cấp phát tỷ lệ:

s_i = size of process p_i
$S = \sum s_i$
m = total number of frames
a_i = allocation for $p_i = \frac{s_i}{S} \times m$

$m = 64$
$s_i = 10$
$s_2 = 127$
$a_1 = \frac{10}{137} \times 64 \approx 5$
$a_2 = \frac{127}{137} \times 64 \approx 59$

37

Cấp phát ưu tiên

- Sử dụng cấp phát tỷ lệ căn cứ vào độ ưu tiên của tiến trình, không căn cứ vào cỡ tiến trình
- Nếu tiến trình P_i sinh ra page-fault:
  - Thay thế một trong các frame của tiến trình đó
  - Thay thế một trong các frame của tiến trình khác có độ ưu tiên thấp hơn

38

Cấp phát tổng thể và cục bộ

- Thay thế tổng thể – Các trang/frame có thể được chọn để thay thế từ tập hợp tất cả các frame/trang. Tiến trình này có thể dùng lại frame/trang của tiến trình khác
- Thay thế cục bộ – Mỗi tiến trình chỉ thay thế trong các trang/frame của chính nó đã được cấp phát

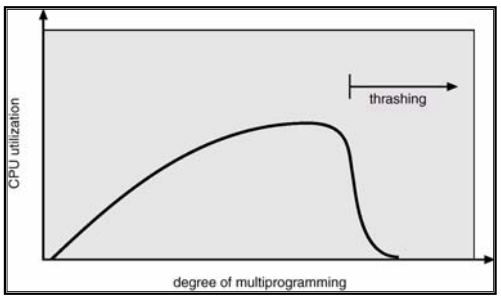
39

Thrashing

- Nếu tiến trình không có đủ trang, tỷ lệ page-fault rất cao, điều đó dẫn tới:
  - Khả năng tận dụng CPU thấp, do đó
  - HĐH có thể tăng mức độ đa chương trình →
  - Các tiến trình được tiếp tục đưa vào hệ thống
- → Hiện tượng thrashing
- Tiến trình thrashing: Một tiến trình luôn bận để swap-in và swap-out

40

Minh họa thrashing



41

Cách hạn chế/ngăn chặn thrashing

- Sử dụng thuật toán thay thế trang cục bộ hoặc thay thế trang ưu tiên để hạn chế thrashing: Chưa giải quyết tốt
- Sử dụng mô hình cục bộ: mô hình working-set
  - Khi tiến trình thực hiện, nó chuyển từ điều kiện cục bộ này sang điều kiện cục bộ khác
  - Điều kiện cục bộ được xác định dựa trên cấu trúc của chương trình và cấu trúc dữ liệu

42

Simpo PDF Merge and Split Unregistered Version - http://www.simpopdf.com

Mô hình working-set

- Sử dụng tham số Δ là tổng số lần tham chiếu trang gần nhất
- Tập các trang sử dụng trong Δ lần gần nhất là working-set
- WSS_i là cỡ working-set của tiến trình P_i: Rõ ràng là WSS_i phụ thuộc vào độ lớn của Δ
  - Nếu Δ quá nhỏ: Không phản ánh đúng tính cục bộ
  - Nếu Δ quá lớn: vượt qua tính cục bộ
  - Nếu $\Delta = \infty$: WSS_i tập toàn bộ các trang trong quá trình thực hiện tiến trình

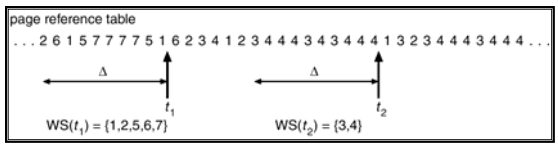
43

Mô hình working-set

- $D = \Sigma\ WSS_i \equiv$ Tổng số các frame được yêu cầu
- Gọi m là tổng số fram rỗi: nếu $D > m$ thì trong hệ thống sẽ xuất hiện thrashing
- Chính sách cấp phát: nếu $D > m$ thì tạm dừng thực hiện một số tiến trình

44

Mô hình working-set



45

Các tệp ánh xạ bộ nhớ

- Sinh viên tự tìm hiểu trong giáo trình từ trang 348 đến trang 353

46

Các vấn đề cần nhớ

- Bộ nhớ ảo
- Yêu cầu trang
- Thay thế trang
- Các thuật toán thay thế trang: FIFO, tối ưu, LRU, LRU xấp xỉ, LFU, MFU, thuật toán đếm
- Thrashing: Định nghĩa, nguyên nhân, cách khắc phục và phòng tránh
- Mô hình working-set

47

Simpo PDF Merge and Split Unregistered Version - http://www.simpopdf.com

Nguyên lý hệ điều hành

Nguyễn Hải Châu
Khoa Công nghệ thông tin
Trường Đại học Công nghệ

Giao diện hệ thống tệp

Khái niệm tệp
Các phương pháp truy cập
Cấu trúc thư mục
Nối hệ thống tệp
Dùng chung tệp
Bảo vệ

Khái niệm tệp

- Không gian địa chỉ logic liên tục
- Các kiểu:
  - Dữ liệu
    - Số
    - Ký tự
    - Nhị phân
  - Chương trình

Cấu trúc tệp

- Chuỗi các từ, byte
- Cấu trúc bản ghi đơn giản: gồm các dòng, độ dài cố định, độ dài thay đổi
- Cấu trúc phức tạp: tài liệu có khuôn dạng, các tệp nạp có định vị lại
- Yếu tố quyết định cấu trúc:
  - Hệ điều hành
  - Chương trình

Thuộc tính tệp

- **Name** – Thông tin người đọc được về tệp
- **Type** – cần cho chương trình, hệ điều hành
- **Location** – Vị trí tệp trên các thiết bị lưu trữ
- **Size** – Cỡ hiện tại của tệp
- **Protection** – Điều khiển các quyền truy cập
- **Time**, **date**, **and user ID** – Dữ liệu về thời gian và định danh người sử dụng
- Thông tin về cách lưu trữ tệp trên thiết bị, được lưu trong cấu trúc của thư mục

Các toán tử trên tệp

- **Create**: Tạo tệp mới
- **Write**: Ghi vào tệp
- **Read**: Đọc từ tệp
- **Seek** – Định vị lại con trỏ tệp
- **Delete**: Xóa tệp
- **Truncate**: Xóa dữ liệu hiện có trong tệp
- **Open**(F_i) – mở tệp F_i (tìm phần tử F_i trong thư mục và đưa nội dung của F_i vào bộ nhớ)
- **Close**(F_i) – đóng tệp F_i (đưa nội dung của F_i trong bộ nhớ ra đĩa)

Simpo PDF Merge and Split Unregistered Version - http://www.simpopdf.com

Mở tệp

- Một số thông tin cần quản lý khi mở tệp:
  - Con trỏ tệp (file pointer): Con trỏ đến vị trí đọc/ghi cuối cùng của mỗi tiến trình
  - Đếm số lượng mở tệp (file-open count): Biến đếm số lần tệp được mở, để cho phép xóa dữ liệu từ bảng mở tệp khi tiến trình cuối cùng đóng tệp
  - Vị trí trên đĩa của tệp: thông tin truy cập dữ liệu của tệp lưu trên đĩa
  - Quyền truy cập (access rights): Thông tin về các quyền truy cập tệp của mỗi tiến trình

7

Mở tệp có khóa

- Một số HĐH có toán tử này
- Dùng để điều khiển truy cập đồng thời đến tệp
- Có hai cách khóa:
  - **Mandatory** – Khóa mang tính chất toàn cục
  - **Advisory** – Khóa mang tính chất hợp tác giữa các tiến trình

8

Kiểu tệp, tên và phần mở rộng

file type	usual extension	function
executable	exe, com, bin or none	read to run machine-language program
object	obj, o	compiled, machine language, not linked
source code	c, cc, java, pas, asm, a	source code in various languages
batch	bat, sh	commands to the command interpreter
text	txt, doc	textual data, documents
word processor	wp, tex, rrf, doc	various word-processor formats
library	lib, a, so, dll, mpeg, mov, rm	libraries of routines for programmers
print or view	arc, zip, tar	ASCII or binary file in a format for printing or viewing
archive	arc, zip, tar	related files grouped into one file, sometimes com-pressed, for archiving or storage
multimedia	mpeg, mov, rm	binary file containing audio or A/V information

9

Các phương pháp truy cập

- **Truy cập tuần tự**

  read next
  write next
  reset
  no read after last write
  (rewrite)

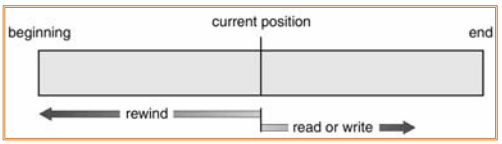
- **Truy cập trực tiếp**

  read *n*
  write *n*
  position to *n*
  read next
  write next
  rewrite *n*

n = relative block number

10

Tệp truy cập tuần tự



11

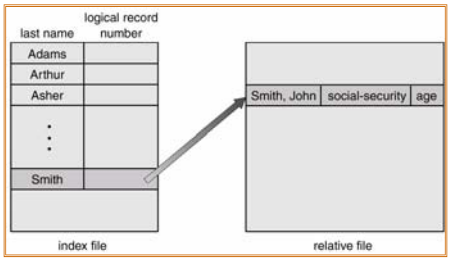
Mô phỏng truy cập tuần tự và trực tiếp

sequential access	implementation for direct access
reset	*cp = 0;*
read next	*read cp;* *cp = cp+1;*
write next	*write cp;* *cp = cp+1;*

12

Simpo PDF Merge and Split Unregistered Version - http://www.simpopdf.com

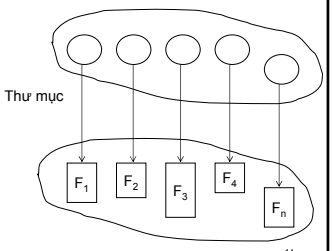
Ví dụ về tệp chỉ số và Relative Files



13

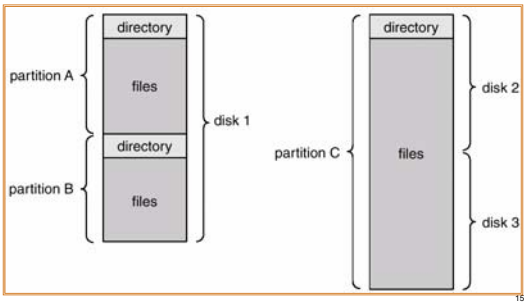
Cấu trúc thư mục

- Thư mục là một tập các node chứa thông tin về tất cả các tệp
- Thư mục và tệp đều nằm trên đĩa
- Backup của thư mục và đĩa có thể nằm trên băng từ



14

Ví dụ một hệ thống tệp



15

Thông tin trên thư mục thiết bị

- Tên
- Kiểu
- Địa chỉ
- Độ dài hiện tại
- Độ dài lớn nhất
- Thời gian của lần truy cập cuối (để lưu trữ)
- Thời gian của lần cập nhật cuối (for dump)
- ID của người chủ tệp
- Các thông tin bảo vệ

16

Các toán tử trên thư mục

- Tìm một tệp
- Tạo một tệp
- Xóa một tệp
- Liệt kê nội dung thư mục
- Đổi tên một tệp
- Duyệt toàn bộ hệ thống tệp

17

Cần tổ chức thư mục để đạt được:

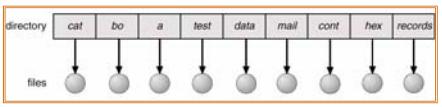
- Tính hiệu quả: tìm thấy một tệp nhanh chóng
- Tên tệp mang lại sự tiện lợi cho người dùng
  - Hai NSD có thể đặt cùng tên cho hai tệp khác nhau
  - Một tệp có thể có nhiều tên khác nhau
- Nhóm tệp: Các tệp có thể được nhóm lại dựa trên thuộc tính (ví dụ nhóm các tệp chương trình nguồn Java, nhóm các tệp thực hiện được...)

18

Simpo PDF Merge and Split Unregistered Version - http://www.simpopdf.com

Thư mục một mức

- Một thư mục cho tất cả NSD

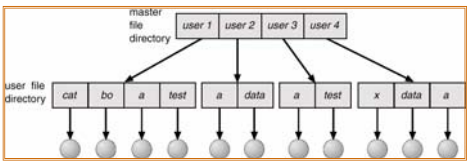


- Vấn đề đặt tên
- Vấn đề nhóm các tệp với nhau

19

Thư mục hai mức

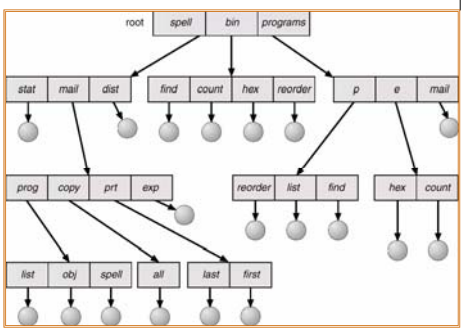
- Mỗi NSD có các thư mục riêng



- Đường dẫn
- Hai NSD có thể đặt cùng tên cho hai tệp khác nhau
- Tìm kiếm tệp hiệu quả
- Không có khả năng nhóm các tệp

20

Thư mục cấu trúc cây



21

Thư mục cấu trúc cây (tiếp)

- Tìm kiếm hiệu quả
- Có khả năng nhóm các tệp
- Thư mục làm việc hiện hành
  - Đổi thư mục làm việc hiện hành

22

Thư mục cấu trúc cây (tiếp)

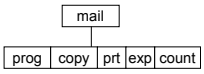
- Đường dẫn tuyệt đối và tương đối
- Tạo một tệp trong thư mục hiện hành
- Xóa tệp

  rm <tên tệp>
- Tạo thư mục con trong thư mục hiện hành

  mkdir <tên thư mục>

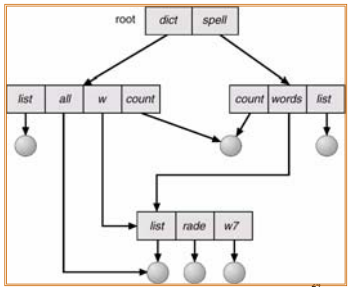
Ví dụ: Nếu thư mục hiện hành là /mail

mkdir count



Thư mục với cấu trúc đồ thị phi chu trình

- Các thư mục có thể có chung thư mục con và tệp



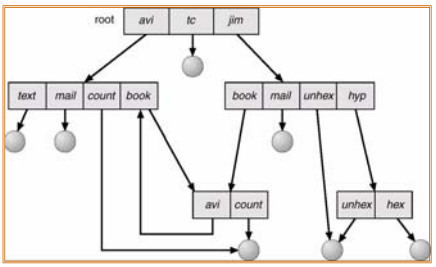
Simpo PDF Merge and Split Unregistered Version - http://www.simpopdf.com

Thư mục với cấu trúc đồ thị phi chu trình (tiếp)

- Tệp hoặc thư mục có thể có các tên khác nhau
  - Shortcut trong Windows
  - Link trong Unix/Linux
- Mềm dẻo hơn cấu trúc cây nhưng phức tạp hơn:
  - Khi xóa một tệp hoặc thư mục có nhiều tên
    - Cần sử dụng con trỏ ngược
    - Sử dụng biến đếm số tên

25

Thư mục đồ thị tổng quát



26

Thư mục đồ thị tổng quát (tiếp)

- Làm cách nào để đảm bảo không có chu trình?
  - Chỉ có phép link tới tệp, không cho link đến thư mục
  - "Dọn dẹp" hệ thống tệp (garbage collection)
  - Mỗi khi có link mới, thực hiện thuật toán phát hiện chu trình

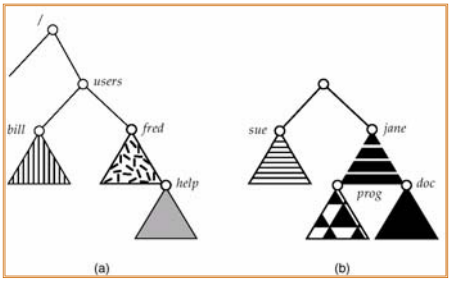
27

Kết nối (mount) hệ thống tệp

- Một hệ thống tệp phải được mount trước khi có thể truy cập tới (sử dụng)
- Một tệp được mount tại điểm kết nối (mount point)

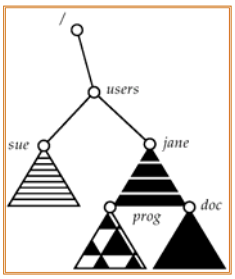
28

Minh họa mount/unmount



29

Điểm kết nối (mount point)



30

Simpo PDF Merge and Split Unregistered Version - http://www.simpopdf.com

Tệp dùng chung

- Cần có tệp dùng chung trên các hệ đa người dung (multi-user)
- Dùng chung tệp có thể thực hiện thông qua một phương pháp bảo vệ
- Với các hệ phân tán, NSD có thể dùng chung tệp trên mạng
- Network File System (NFS) là một phương pháp dùng chung tệp phổ biến

31

Đa người dùng

- **User IDs** định danh NSD để từ đó xác định các quyền và phương pháp bảo vệ
- **Group IDs** xác định nhóm NSD để từ đó xác định các quyền truy cập nhóm

32

Hệ thống tệp từ xa

- Sử dụng tệp để truy cập các hệ thống file ở các vị trí khác nhau
  - Thủ công: ví dụ FTP
  - Tự động: hệ thống tệp phân tán **distributed file systems**
  - Bán tự động: **world wide web**
- Mô hình khách-chủ **(Client-server model)** cho phép máy khách mount hệ thống tệp của máy chủ từ xa
  - Máy chủ có thể phục vụ nhiều máy khách
  - Định danh máy khách và NSD trên máy khách có thể đơn giản (không an toàn - insecure) hoặc rất phức tạp

33

Hệ thống tệp từ xa (tiếp)

- **NFS** là giao thức sử dụng chung tệp chuẩn trên UNIX cho mô hình client-server
- **CIFS** là chuẩn trên Windows
- Các hàm hệ thống chuẩn được chuyển đổi thành lời gọi từ xa (remote call)
- Các dịch vụ đặt tên phân tán (**distributed naming services**) như LDAP, DNS, NIS cho ta cách truy cập thống nhất đến các thông tin cần thiết cho tính toán từ xa

34

Lỗi trong hệ thống tệp từ xa

- Có nhiều nguyên nhân gây lỗi trong hệ thống tệp từ xa: Do lỗi mạng, lỗi server...
- Khôi phục lỗi cần có thông tin trạng thái đối với mỗi yêu cầu phục vụ từ xa
- Các giao thức như NFS lưu đưa toàn bộ các thông tin trạng thái vào mỗi yêu cầu do đó dễ khôi phục, nhưng kém an ninh

35

Nhất quán về ngữ nghĩa

- **Nhất quán ngữ nghĩa** chỉ định cách truy cập đồng thời của nhiều NSD đến tệp dùng chung
  - Andrew File System (AFS) cài đặt hệ thống ngữ nghĩa phức tạp cho hệ thống tệp truy cập từ xa
    - AFS có ngữ nghĩa theo phiên: các toán tử **write** chỉ có tác dụng sau khi tệp được **close**
  - Unix file system (UFS) cài đặt:
    - Các toán tử **write** ngay lập tức có tác dụng trên các tệp chung (người đọc nhìn thấy kết quả của **write**)
    - Dùng chung tệp cho phép nhiều NSD đọc và ghi đồng thời

36

Simpo PDF Merge and Split Unregistered Version - http://www.simpopdf.com

Bảo vệ

- Người tạo tệp (chủ tệp) được phép qui định
  - Các toán tử nào trên tệp có thể được thực hiện...
  - ... và do ai thực hiện
- Các toán tử:
  - Read
  - Write
  - Execute
  - Append
  - Delete
  - List

37

Danh sách và nhóm truy cập

- Các toán tử: read, write, execute
- Ba lớp NSD là **owner**, **group** và **public**

		RWX
a) **owner access** 7	⇒	1 1 1
		RWX
b) **group access** 6	⇒	1 1 0
		RWX
c) **public access** 1	⇒	0 0 1

- Xem thêm quyền truy cập tệp của HĐH Unix/Linux

38

Các vấn đề cần nhớ

- Khái niệm tệp
- Các phương pháp truy cập tệp
- Cấu trúc thư mục một cấp, nhiều cấp, cấu trúc thư mục cây, đồ thị phi chu trình, đồ thị tổng quát
- Nối hệ thống tệp
- Dùng chung tệp
- Hệ thống tệp từ xa
- Quyền truy cập tệp

39

Simpo PDF Merge and Split Unregistered Version - http://www.simpopdf.com

Nguyên lý hệ điều hành

Nguyễn Hải Châu
Khoa Công nghệ Thông tin
Trường Đại học Công nghệ

1

Hệ vào/ra

Phần cứng
Giao diện vào/ra với ứng dụng
Hệ vào/ra của nhân
Chuyển yêu cầu vào/ra thành thao tác phần cứng
Streams
Các vấn đề về hiệu năng

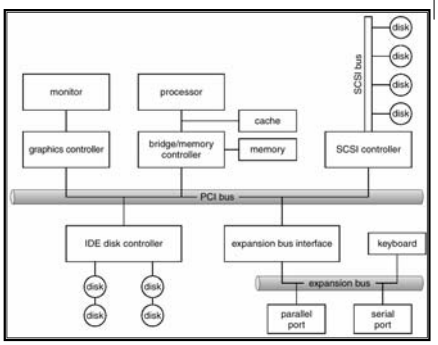
2

Phần cứng vào/ra

- Có rất nhiều loại thiết bị vào/ra
- Các khái niệm chung
  - Port (cổng vào/ra)
  - Bus
  - Controller
- Các vi lệnh điều khiển thiết bị vào/ra
- Thiết bị vào/ra có địa chỉ được sử dụng bởi:
  - Các lệnh vào/ra trực tiếp
  - Vào/ra thông qua ánh xạ bộ nhớ

3

Cấu trúc bus của máy PC



4

Một số địa chỉ vào/ra của PC

I/O address range (hexadecimal)	device
000-00F	DMA controller
020-021	interrupt controller
040-043	timer
200-20F	game controller
2F8-2FF	serial port (secondary)
320-32F	hard-disk controller
378-37F	parallel port
3D0-3DF	graphics controller
3F0-3F7	diskette-drive controller
3F8-3FF	serial port (primary)

5

Polling

- Xác định trạng thái của thiết bị:
  - command-ready (lệnh sẵn sàng?)
  - busy (bận?)
  - Error (lỗi?)
- Thực hiện vòng lặp *chờ bận* để chờ vào/ra với thiết bị

6

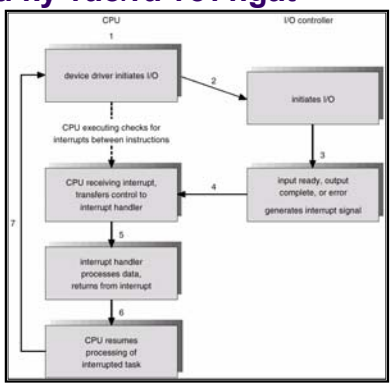
Simpo PDF Merge and Split Unregistered Version - http://www.simpopdf.com

Interrupts (ngắt)

- Thiết bị vào/ra kích hoạt đường yêu cầu ngắt CPU
- Bộ thao tác ngắt nhận ngắt
- CPU có thể bỏ qua hoặc làm trễ việc xử lý một số ngắt
- Vector ngắt giúp CPU tìm được hàm xử lý ngắt
  - Dựa trên độ ưu tiến
  - Một số ngắt là không che được (unmaskable)
- Cơ chế ngắt có thể dùng cho exceptions

7

Chu kỳ vào/ra với ngắt



8

Bảng vector ngắt của BXL Intel

vector number	description
0	divide error
1	debug exception
2	null interrupt
3	breakpoint
4	INTO-detected overflow
5	bound range exception
6	invalid opcode
7	device not available
8	double fault
9	coprocessor segment overrun (reserved)
10	invalid task state segment
11	segment not present
12	stack fault
13	general protection
14	page fault
15	(Intel reserved, do not use)
16	floating-point error
17	alignment check
18	machine check
19Đ31	(Intel reserved, do not use)
32Đ255	maskable interrupts

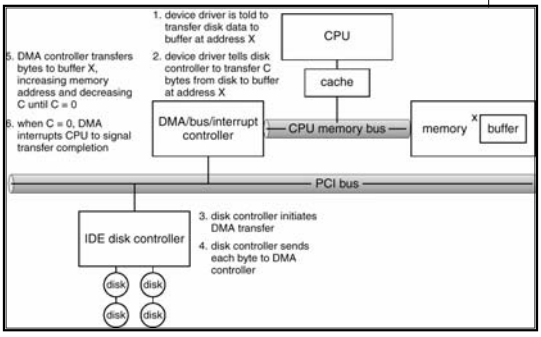
9

Truy cập bộ nhớ trực tiếp

- Thuật ngữ: Direct memory access (DMA)
- Được sử dụng để tránh lập trình vào/ra với dung lượng dữ liệu lớn
- Phần cứng cần có: Bộ điều khiển DMA
- CPU truyền dữ liệu trực tiếp giữa bộ nhớ và thiết bị vào/ra

10

Quá trình 6 bước thực hiện vào/ra theo DMA



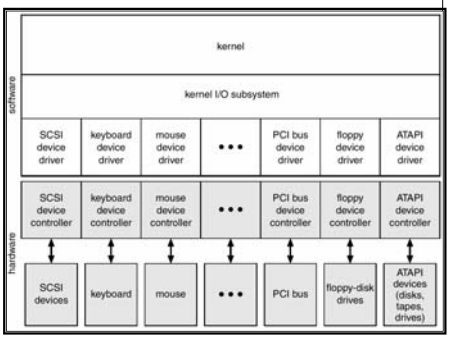
Giao diện vào/ra với ứng dụng

- Các hàm hệ thống vào/ra của một thiết bị được đóng gói trong các class chung
- Tầng điều khiển thiết bị (device-driver layer) che đi sự khác biệt giữa các bộ điều khiển vào/ra
- Có nhiều loại thiết bị căn cứ theo các tiêu chí:
  - Character-stream / block
  - Sequential / random-access
  - Sharable / dedicated
  - Speed of operation
  - read-write / read only / write only

12

Simpo PDF Merge and Split Unregistered Version - http://www.simpopdf.com

Cấu trúc vào ra của nhân



13

Đặc tính các thiết bị vào/ra

aspect	variation	example
data-transfer mode	character block	terminal disk
access method	sequential random	modem CD-ROM
transfer schedule	synchronous asynchronous	tape keyboard
sharing	dedicated sharable	tape keyboard
device speed	latency seek time transfer rate delay between operations	
I/O direction	read only write only readÐwrite	CD-ROM graphics controller disk

14

Các thiết bị *block* và *character*

- Các thiết bị *block* (ví dụ đĩa cứng):
  - Các lệnh làm việc: read, write, seek
  - Có thể thực hiện vào/ra theo chế độ *raw* I/O hoặc thông qua truy cập hệ thống tệp
  - Có thể truy cập qua tệp memory-mapped (ánh xạ bộ nhớ)
- Các thiết bị *character* (ví dụ bàn phím, chuột, cổng COM):
  - Các lệnh làm việc: get, put
  - Bổ sung thư viện cho phép làm việc theo dòng (line)

15

Các thiết bị mạng

- Có thể là thiết bị block hoặc character
- Unix và Windows NT/9*i*/2000 có giao diện lập trình socket
  - Tách biệt giao thức mạng với các thao tác mạng
  - Có tính năng `select`
- Nhiều cách tiếp cận vào/ra (pipes, FIFOs, streams, queues, mailboxes)

16

Đồng hồ (clock) và timer

- Cung cấp thông tin về giờ hiện tại, giờ đã trôi qua, timer
- Nếu phần cứng clock/timer lập trình được:Có thể tạo ngắt định kỳ (Cần cho các hệ time-sharing)

17

Vào/ra blocking và nonblocking

- Blocking – Tiến trình treo đến khi vào/ra hoàn thành
  - Dễ hiểu, dễ sử dụng
  - Không đủ đối với một số loại yêu cầu vào/ra
- Nonblocking – Hàm vào/ra trả lại kết quả ngay không cần vào/ra hoàn thành
  - Giao diện NSD, copy dữ liệu có buffered vào/ra
  - Được cài đặt qua kỹ thuật đa luồng
  - Trả lại ngay số byte được đọc/ghi

18

Simpo PDF Merge and Split Unregistered Version - http://www.simpopdf.com

Vào/ra không đồng bộ

- Asynchronous (không đồng bộ): Tiến trình chạy trong khi vào/ra đang được thực hiện
  - Khó sử dụng
  - Hệ vào/ra gửi tín hiệu cho tiến trình khi vào/ra hoàn thành

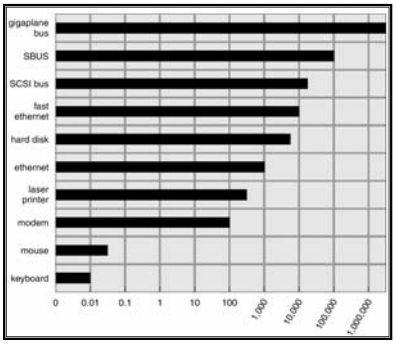
19

Hệ vào/ra của nhân

- Lập lịch
  - Một số yêu cầu vào/ra được phục vụ thông qua hàng chờ vào/ra của từng thiết bị
  - Một số HĐH cố gắng đảm bảo tính công bằng
- Tạo vùng đệm lưu dữ liệu trong bộ nhớ khi truyền dữ liệu giữa các thiết bị:
  - Khắc phục sự khác nhau về tốc độ của các thiết bị
  - Khắc phục sự khác nhau về độ dài gói dữ liệu
  - Để duy trì ngữ nghĩa copy

20

Tốc độ truyền dữ liệu của các thiết bị trên Sun Enterprise 6000



21

Hệ vào/ra của nhân

- Caching – Bộ nhớ tốc độ cao chứa các bản copy của dữ liệu
  - Dữ liệu luôn là bản copy
  - Cải thiện đáng kể hiệu năng hệ thống
- Spooling –Lưu dữ liệu ra (output) cho một thiết bị
  - Sử dụng khi thiết bị chỉ phục vụ được một yêu cầu tại một thời điểm
  - Ví dụ: Máy in

22

Hệ vào/ra của nhân

- Cung cấp khả năng sử dụng "độc quyền" một thiết bị
  - Hàm hệ thống: cấp phát và giải phóng thiết bị
  - Cơ chế chống bế tắc

23

Xử lý lỗi

- HĐH có thể khôi phục lỗi gây ra do đọc đĩa, thiết bị chưa sẵn sàng, ghi lỗi...
- Khi có lỗi vào/ra: Hàm điều khiển trả lại mã lỗi
- Hệ thống có log ghi lại các lỗi vào/ra

24

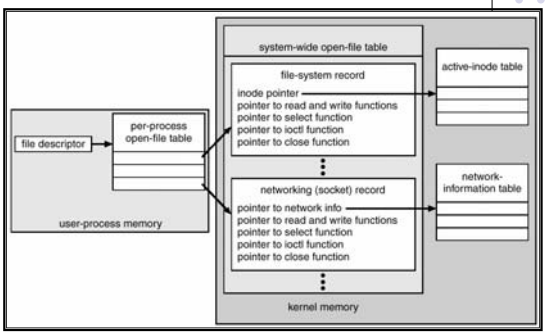
Simpo PDF Merge and Split Unregistered Version - http://www.simpopdf.com

Cấu trúc dữ liệu của nhân

- Nhân giữ các thông tin trạng thái cho các thành phần của hệ vào/ra, bao gồm bảng các mở tệp, kết nối mạng, trạng thái các thiết bị *character*
- Nhiều cấu trúc dữ liệu phức tạp để lưu vết các vùng đệm, cấp phát bộ nhớ, các khối nhớ rỗi...
- Một số HĐH sử dụng phương pháp hướng đối tượng và message-passing để cài đặt hệ vào/ra

25

Cấu trúc vào/ra trong nhân UNIX

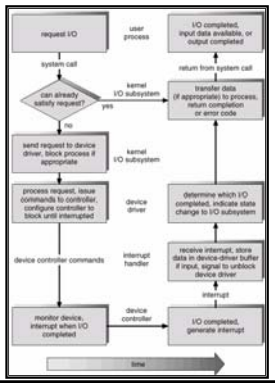


Chuyển đổi yêu cầu vào/ra thành thao tác phần cứng

- Giả sử một tiến trình đọc tệp từ đĩa cứng. Các bước thực hiện như sau:
  - Xác định thiết bị chứa tệp
  - Biến đổi tên tệp thành dạng biểu diễn của tệp trên thiết bị
  - Đọc dữ liệu (vật lý) từ đĩa vào vùng đệm
  - Cho phép tiến trình được đọc dữ liệu từ vùng đệm
  - Trả lại điều khiển cho tiến trình

27

Thực hiện một yêu cầu vào/ra



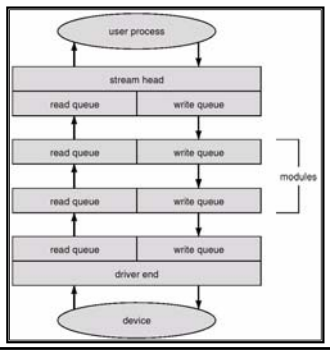
28

STREAMS

- **STREAM** – kênh liên lạc full-duplex giữa một tiến trình của NSD và một thiết bị
- Một STREAM gồm có:
  - **STREAM head** dùng để giao tiếp với tiến trình của NSD
  - **driver end** giao tiếp với thiết bị
  - *n* STREAM module giữa head và end (n≥0).
- Mỗi module có một **read queue** và một **write queue**
- Message passing được sử dụng để truyền thông giữa các queue

29

Cấu trúc STREAMS



30

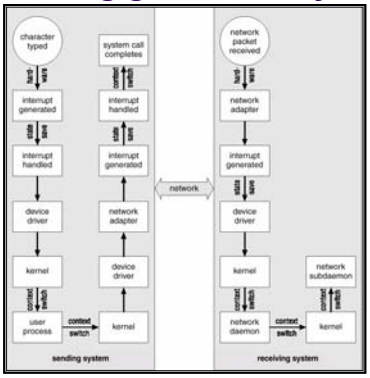
Simpo PDF Merge and Split Unregistered Version - http://www.simpopdf.com

Hiệu năng

- Vào/ra là yếu tố chính ảnh hưởng đến hiệu năng của hệ thống:
  - Yêu cầu CPU thực hiện mã vào/ra của driver, mã vào/ra của kernel
  - Thực hiện Context switch khi có ngắt
  - Copy dữ liệu
  - Truyền dữ liệu mạng

31

Truyền thông giữa các máy tính



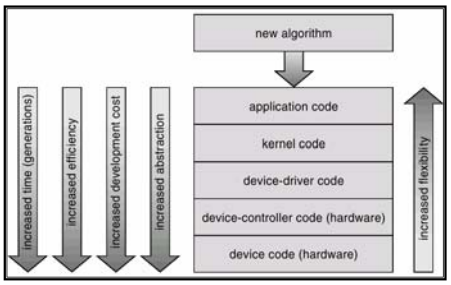
32

Cải tiến hiệu năng

- Giảm số lượng các context switches
- Giảm copy dữ liệu
- Giảm số lần ngắt bằng cách truyền lượng lớn dữ liệu trong mỗi lần vào/ra, dùng controller thông minh...
- Sử dụng DMA
- Cân bằng tải giữa hiệu năng CPU, bộ nhớ, bus, và vào/ra để đạt thông lượng tốt nhất

33

Chức năng các thiết bị



34

Các vấn đề cần nhớ

- Giao diện vào/ra với ứng dụng
- Hệ vào/ra của nhân HĐH
- Streams
- Các vấn đề về hiệu năng hệ thống chịu ảnh hưởng của vào/ra

35

Simpo PDF Merge and Split Unregistered Version - http://www.simpopdf.com

Nguyên lý hệ điều hành

Nguyễn Hải Châu
Khoa Công nghệ Thông tin
Trường Đại học Công nghệ

Các hệ thống lưu trữ

Cấu trúc đĩa
Lập lịch đĩa
Quản lý đĩa
Quản lý không gian swap
Cấu trúc RAID
...

Cấu trúc đĩa

- Các ổ đĩa được đánh địa chỉ như một mảng lớn, 1 chiều với mỗi phần tử là một khối logic – đơn vị truyền nhận nhỏ nhất
- Mảng một chiều nói trên được ánh xạ vào các sector đĩa một cách tuần tự
  - Sector 0 là sector đầu tiên trên rãnh đầu tiên của track nằm ngoài cùng trên đĩa
  - Quá trình ánh xạ theo thứ tự chỉ số: sector, track, cylinder (từ ngoài vào trong)

Lập lịch đĩa (1)

- HĐH cần sử dụng phần cứng một cách hiệu quả - với đĩa: thời gian truy cập nhanh và băng thông lớn
- Thời gian truy cập bị ảnh hưởng bởi:
  - *Seek time* (thời gian dịch đầu đọc): Thời gian chuyển đầu đọc đến cylinder chứa sector cần truy cập
  - *Rotational latency* (Độ trễ quay): Thời gian chờ đĩa quay để đầu đọc gặp sector cần truy cập

Lập lịch đĩa (2)

- Seek time: Càng nhỏ càng tốt
  - Tương đương: Khoảng cách dịch đầu đọc càng nhỏ càng tốt
- Băng thông đĩa là tổng số byte đã được truyền chia cho tổng thời gian giữa yêu cầu đầu tiên và thời gian hoàn thành lần truyền dữ liệu cuối cùng

Lập lịch đĩa (3)

- Có nhiều thuật toán lập lịch đĩa
- Chúng ta minh họa với dãy các yêu cầu (Giả sử đĩa có 200 track từ 0-199):

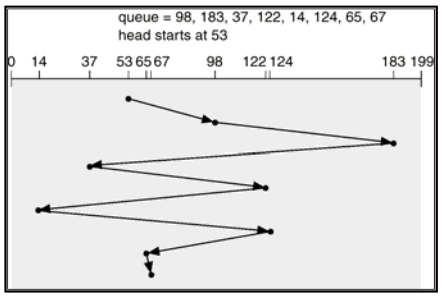
98, 183, 37, 122, 14, 124, 65, 67

Đầu đọc đang nằm ở cylinder 53

Simpo PDF Merge and Split Unregistered Version - http://www.simpopdf.com

FCFS (First come first serve)

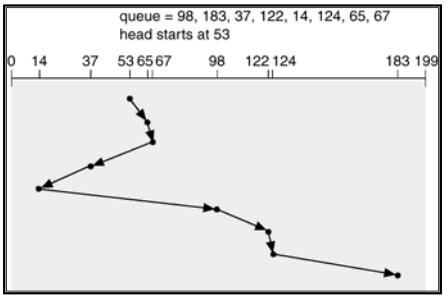
Tổng số bước di chuyển của đầu đọc là: 640 cylinder.



SSTF: Shortest seek time first

- Yêu cầu có seek time nhỏ nhất tính từ vị trí hiện tại của đầu đọc
- Lập lịch SSTF là một dạng của lập lịch SJF có thể gây ra một số yêu cầu không bao giờ được phục vụ (starvation)
- Ví dụ minh họa: Tổng số bước di chuyển của đầu đọc là 236 cylinder.

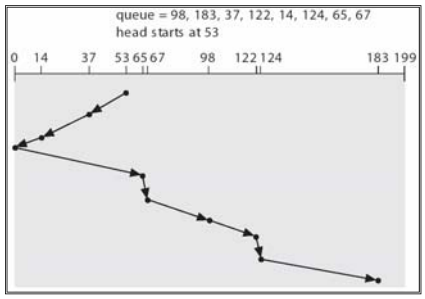
Ví dụ SSTF



SCAN

- Đầu đọc của đĩa di chuyển từ một phía (ví dụ bên ngoài hoặc bên trong đĩa) sang phía kia để phục vụ các yêu cầu đọc, sau đó di chuyển ngược lại... quá trình này lặp đi lặp lại
- Phương thức hoạt động tương tự thang máy nên thuật toán này còn được gọi là thuật toán thang máy (*elevator algorithm*)
- Ví dụ minh họa: Đầu đọc phải dịch chuyển 208 cylinder.

Ví dụ SCAN

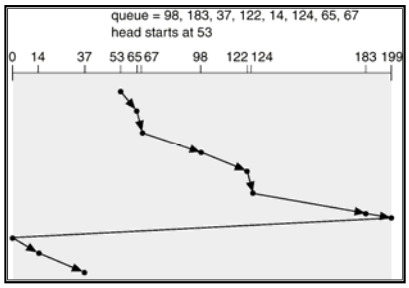


C-SCAN

- Đầu đọc chuyển từ một phía (trong/ngoài) sang phía kia và phục vụ các yêu cầu. Khi sang đến phía kia, đầu đọc quay trở lại nhưng trong khi quay trở lại không phục vụ yêu cầu nào.
- C-SCAN xem các cylinders như một danh sách vòng

Simpo PDF Merge and Split Unregistered Version - http://www.simpopdf.com

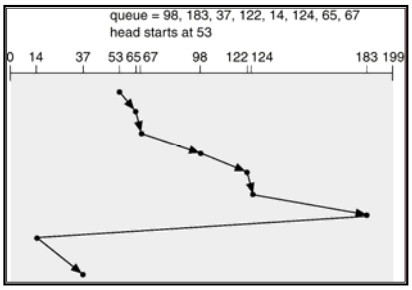
Ví dụ C-SCAN



C-LOOK

- Là một trường hợp của C-SCAN
- C-SCAN: Đầu đọc chuyển giữa cylinder 0 và *n* (cynlinder cuối)
- C-LOOK: Đầu đọc chuyển giữa c_{min} và c_{max} trong đó c_{min} là cynlinder có số thứ tự nhỏ nhất trong số các yêu cầu; c_{max} là cynlinder có số thứ tự nhỏ nhất trong số các yêu cầu
- C-LOOK giảm quãng đường di chuyển của đầu đọc so với C-SCAN

Ví dụ C-LOOK



Chọn thuật toán lập lịch đĩa

- SSTF là phổ biến và "tự nhiên"
- SCAN và C-SCAN thực hiện tốt với các hệ thống đọc ghi đĩa nhiều
- Hiệu năng nói chung phụ thuộc vào số lượng và tính chất của các yêu cầu truy cập đĩa
- Yêu cầu đọc ghi đia có thể bị ảnh hưởng bởi phương pháp cấp phát tệp

Chọn thuật toán lập lịch đĩa

- Các thuật toán lập lịch đĩa nên được cài đặt như một module độc lập của HĐH để dễ thay thế khi cần thiết
- SSTF hoặc LOOK có thể chọn là thuật toán ngầm định

Quản lý đĩa

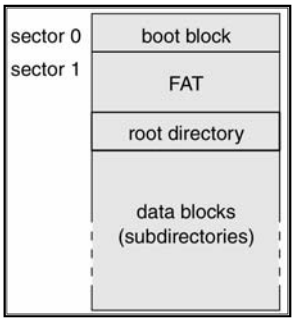
- *Low-level format* hoặc *physical format* — Chia đĩa thành các sector để bộ điều khiển đĩa (disk controller) có thể đọc/ghi
- Để lưu tệp lên đĩa, HĐH cần ghi cấu trúc dữ liệu lên đĩa:
  - HĐH chia đĩa thành các *partition* (phân vùng) – mỗi partition là một nhóm các cylinder
  - HĐH thực hiện *logical formatting* hay tạo hệ thống tệp.

Simpo PDF Merge and Split Unregistered Version - http://www.simpopdf.com

Tổ chức đĩa

- Đĩa cứng có boot block để khởi tạo hệ thống:
  - bootstrap được lưu trong ROM
  - *Bootstrap loader* là một chương trình nhỏ nằm trên boot block của đĩa cứng.
- Ví dụ bootstrap loader:
  - Linux: GRUB, LILO
  - Windows: NTLDR
- Với các sector hỏng: Phương pháp *sector sparing*

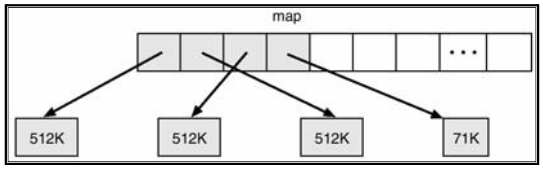
Tổ chức đĩa của MS-DOS



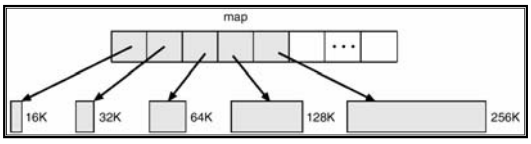
Quản lý không gian swap

- Không gian swap được xem như một phần mở rộng của bộ nhớ trong và nằm trên đĩa
- Không gian swap có thể nằm trên hệ thống tệp hoặc trên một partition riêng
- Quản lý không gian swap
  - UNIX BSD 4.3 cấp phát swap khi tiến trình bắt đầu thực hiện và lưu các segment: text và data
  - Nhân dùng *swap maps* để quản lý việc sử dụng không gian swap.

4.3 BSD Text-Segment Swap Map



4.3 BSD Data-Segment Swap Map



Cấu trúc RAID

- **RAID** – Hệ thống lưu trữ sử dụng nhiều ổ đĩa để tăng độ tin cậy (**reliability)** thông qua sự dư thừa (**redundancy**).
- Có 6 mức RAID.

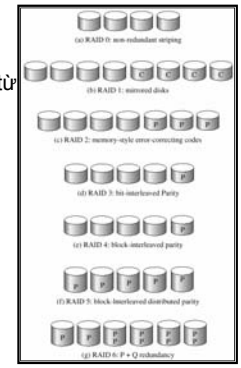
Simpo PDF Merge and Split Unregistered Version - http://www.simpopdf.com

RAID (tiếp)

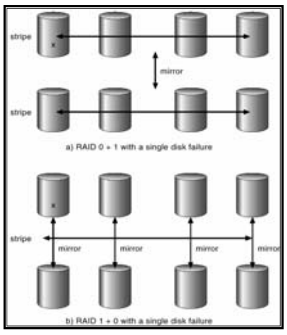
- Disk striping: Sử dụng một tập đĩa như một đĩa
- RAID cải thiện hiệu năng và độ tin cậy của hệ lưu trữ bằng cách lưu trữ có dư thừa
  - *Mirroring/shadowing*: Mỗi đĩa có một bản copy (duplicate).
  - *Block interleaved parity*: Mức độ dư thừa ít hơn mirroring.

Các mức RAID

- Xem chi tiết các mức RAID trong giáo trình từ trang 471 đến 475



RAID (0 + 1) và (1 + 0)

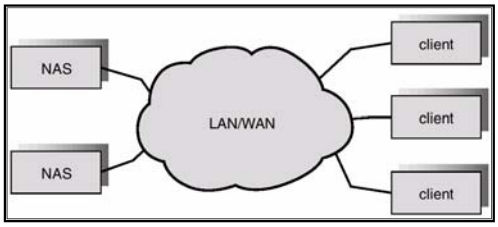


Kết nối đĩa

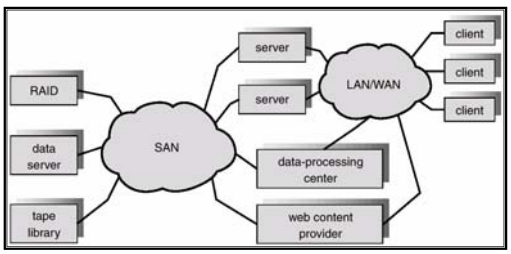
- Có thể kết nối đĩa theo 2 cách:

1. **Host attached** thông qua cổng vào/ra
2. **Network attached** thông qua kết nối mạng

Network-Attached Storage



Storage-Area Network



Simpo PDF Merge and Split Unregistered Version - http://www.simpopdf.com

Cài đặt hệ lưu trữ ổn định

- Write-ahead log scheme requires stable storage.
- Để cài đặt hệ lưu trữ ổn định:
  - Replicate information on more than one nonvolatile storage media with independent failure modes.
  - Update information in a controlled manner to ensure that we can recover the stable data after any failure during data transfer or recovery.

Các thiết bị tertiary

- Thuật ngữ: Tertiary storage
- Đặc tính của tertiary storage là giá rẻ
- Nói chung, tertiary storage được làm để *tháo được*
- Ví dụ: Đĩa mềm, CD-ROM, USB

Đĩa tháo được

- Đĩa mềm: Đĩa phủ từ nằm trong vỏ bảo vệ
  - Hầu hết các đĩa mềm chứa được khoảng 1 MB; một số đĩa sử dụng công nghệ tương tự có thể chứa 1GB.
  - Tốc độ đĩa mềm nhanh nhưng dễ mất dữ liệu do hỏng bề mặt (tiếp xúc)

Đĩa tháo được

- Đĩa quang-từ ghi dữ liệu trên đĩa nhựa cứng bề mặt phủ vật liệu từ.
  - Nhiệt laser được sử dụng để khuếch đại từ trường yếu để ghi 1 bit
  - Ánh sáng laser light is được sử dụng để đọc dữ liệu (hiệu ứng Kerr).
  - Đầu đọc đĩa loại này xa bề mặt đĩa hơn là đầu đọc đĩa từ → giảm hỏng do xước, va chạm.
- Đĩa quang không sử dụng vật liệu từ mà dùng vật liệu đặc biệt có thể bị biến đổi do tia laser

Đĩa WORM

- WORM (“Write Once, Read Many Times”) Ghi một lần, đọc nhiều lần
- Đĩa có 3 lớp: Hai lớp nhựa ở 2 mặt, ở giữa là một lớp phim mỏng chế tạo từ nhôm
- Để ghi 1 bit: Ổ đĩa dùng tia laser đốt một lỗ nhỏ trên lớp phim nhôm.
- Rất bền và tin cậy
- Ví dụ: CD-ROM, DVD-ROM

Băng

- Băng rẻ hơn đĩa, nhưng truy cập ngẫu nhiên chậm hơn
- Băng thường được dùng cho các ứng dụng không yêu cầu truy cập nhanh. Ví dụ: Lưu trữ dữ liệu, backup
- Các hệ thống băng từ lớn sử dụng robot để thay băng: Chuyển băng giữa các ổ băng và thư viện băng
  - stacker – Thư viện băng nhỏ (một vài băng)
  - silo – Thư viện băng lớn (vài nghìn băng)

Simpo PDF Merge and Split Unregistered Version - http://www.simpopdf.com

Các vấn đề của HĐH

- Nhiệm vụ chính của HĐH là quản lý các thiết bị vật lý và cung cấp một máy ảo cho ứng dụng (thông qua trừu tượng hóa)
- Với đĩa cứng có hai mức trừu tượng hóa:
  - Thiết bị – Một mảng các khối dữ liệu
  - Hệ thống tệp – HĐH phục vụ các yêu cầu truy cập đĩa (qua cơ chế hàng chờ/lập lịch) từ nhiều ứng dụng

Tốc độ truy cập đĩa

- Hai yếu tố ảnh hưởng đến tốc độ truy cập đĩa: băng thông bandwidth và độ trễ (latency).
- Băng thông được đo bằng byte/second.
  - Sustained bandwidth: Tốc độ trao đổi dữ liệu trung bình trong một lần đọc/ghi lớn (tổng số byte/thời gian)
  - Effective bandwidth – Băng thông trung bình của toàn bộ các lần vào/ra bao gồm **seek** / **locate**...

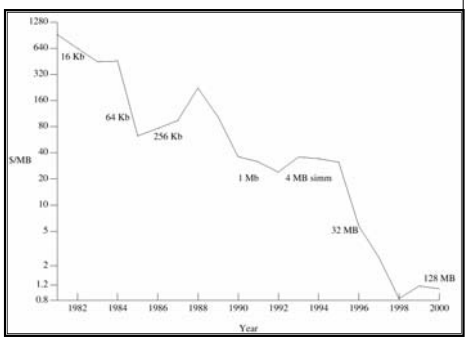
Tốc độ truy cập đĩa

- Độ trễ truy cập: Thời gian cần để định vị dữ liệu trên đĩa.
  - Độ trễ cho đĩa: Chuyển đầu đọc đến cylinder cần thiết + độ trễ quay (thường < 35ms)
  - Độ trễ băng: Cần tua băng → Độ trễ từ vài chục đến vài trăm giây

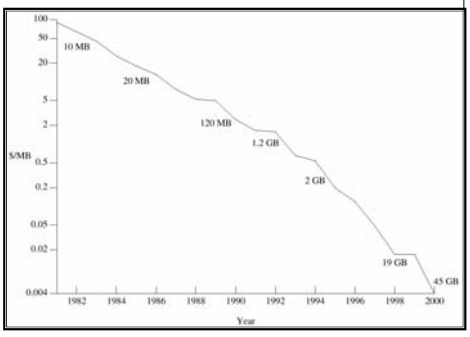
Độ tin cậy

- Đĩa cứng có độ tin cậy cao hơn băng hoặc đĩa tháo được
- Lưu trữ trên đĩa quang tin cậy hơn đĩa từ hoặc băng
- Đầu đọc đĩa cứng hỏng→ mất dữ liệu
- Đầu đọc băng, CD... hỏng không gây mất dữ liệu

Giá 1MB DRAM từ 1981 đến 2000

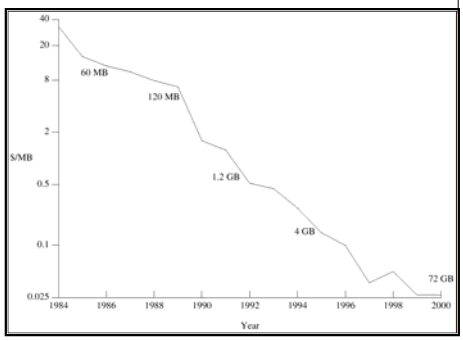


Giá 1MB đĩa cứng từ 1981 đến 2000



Simpo PDF Merge and Split Unregistered Version - http://www.simpopdf.com

Giá 1MB băng, từ 1984 đến 2000



Simpo PDF Merge and Split Unregistered Version - http://www.simpopdf.com

Nguyên lý hệ điều hành

Nguyễn Hải Châu
Khoa Công nghệ Thông tin
Trường Đại học Công nghệ

Bảo vệ và an ninh

Bảo vệ

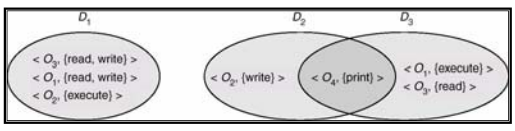
Mục đích bảo vệ
Các miền bảo vệ
Ma trận truy cập
Cài đặt ma trận truy cập
Hủy bỏ quyền truy cập

Bảo vệ

- HĐH gồm một tập các đối tượng, đối tượng: phần cứng hoặc phần mềm
- Mỗi đối tượng có một tên duy nhất và có thể truy cập đến thông qua một số toán tử (hàm hệ thống)
- Bảo vệ: Đảm bảo mỗi đối tượng được truy cập đúng cách và chỉ bởi các tiến trình được phép

Cấu trúc miền bảo vệ

- Quyền truy cập = <*tên đối tượng*, *tập các toán tử*> trong đó *tập các toán tử* là một tập con của tập tất cả các toán tử hợp lệ có thể thực hiện trên *đối tượng*
- Miền = Tập các quyền truy cập



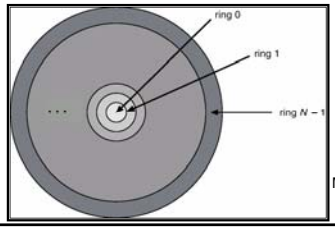
Cài đặt miền trên UNIX

- Hệ Unix có 2 miền:
  - Người sử dụng (user)
  - Người quản trị hệ thống (supervisor/root)
- UNIX
  - Miền = user-id
  - "Chuyển" miền bảo vệ thông qua hệ thống tệp:
    - Mỗi tệp có 1 bit gắn với miền (setuid bit).
    - Khi tệp *f* được thực hiện và bit setuid=1 thì user-id được đặt là owner của tệp *f*. Khi thực hiện xong, user-id được trả lại giá trị cũ

Simpo PDF Merge and Split Unregistered Version - http://www.simpopdf.com

Cài đặt tên miền trên Multics

- Các miền bảo vệ bao nhau (ring)
- Gọi D_i và D_j là hai miền bảo vệ bất kỳ.
- Nếu $j < i \Rightarrow D_i \subseteq D_j$



Multics Rings

Ma trận truy cập

- Biểu diễn các miền bảo vệ dưới dạng ma trận (Ma trận truy cập - *access matrix*). Giả sử ma trận là *access*
- Các hàng biểu diễn các miền
- Các tên cột biểu diễn các đối tượng
- Phần tử *access(i, j)* là tập các toán tử một tiến trình thực hiện trong miền D_i được thao tác trên đối tượng O_j

Ma trận truy cập

domain \ object	F_1	F_2	F_3	printer
D_1	read		read	
D_2				print
D_3		read	execute	
D_4	read write		read write	

Sử dụng ma trận truy cập

- Nếu một tiến trình trong miền D_i muốn thực hiện toán tử "op" trên đối tượng O_j, thì "op" phải nằm trong ma trận truy cập
- Mở rộng: Bảo vệ "động"
  - Các toán tử để thêm, xóa các quyền truy cập
  - Các quyền truy cập đặc biệt:
    - *Chủ của đối tượng O_i*
    - *Sao chép toán tử "op" từ O_i sang O_j*
    - *Quyền điều khiển – D_i có thể sửa đổi quyền truy cập của D_j*
    - *transfer – switch từ miền D_i sang D_j*

Ma trận truy cập với các miền được xem như các đối tượng

domain \ object	F_1	F_2	F_3	laser printer	D_1	D_2	D_3	D_4
D_1	read		read			switch		
D_2				print			switch	switch
D_3		read	execute					
D_4	read write		read write		switch			

Ma trận truy cập với quyền truy cập *Copy*

domain \ object	F_1	F_2	F_3
D_1	execute		write*
D_2	execute	read*	execute
D_3	execute		

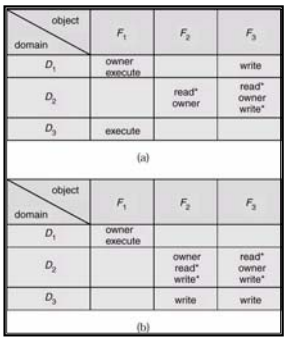
(a)

domain \ object	F_1	F_2	F_3
D_1	execute		write*
D_2	execute	read*	execute
D_3	execute	read	

(b)

Simpo PDF Merge and Split Unregistered Version - http://www.simpopdf.com

Ma trận truy cập với quyền truy cập *Owner*



object / domain	F_1	F_2	F_3
D_1	owner execute		write
D_2		read* owner	read* owner write*
D_3	execute		

(a)

object / domain	F_1	F_2	F_3
D_1	owner execute		
D_2		owner read* write*	read* owner write*
D_3		write	write

(b)

object / domain	F_1	F_2	F_3	laser printer	D_1	D_2	D_3	D_4
D_1	read		read			switch		
D_2				print			switch	switch control
D_3		read	execute					
D_4	write		write		switch			

Cài đặt ma trận truy cập

- Có 4 cách cài đặt:
  - Bảng toàn cục (global table): Phương pháp đơn giản nhất
  - Danh sách truy cập (access list) cho các đối tượng
  - Danh sách khả năng (capability list) cho các miền
  - Cơ chế khóa – chìa (Lock-Key)

Bảng toàn cục

- Là một bảng với các phần tử là bộ ba:
  - <miền, đối tượng, tập các quyền>
- Bảng toàn cục thường lớn nên không nằm toàn bộ trong bộ nhớ trong → Cần nhiều thao tác vào/ra
- Tốn thời gian tìm kiếm trên bảng toàn cục

Danh sách truy cập

- Mỗi cột trong ma trận truy cập có thể được cài đặt thành một danh sách truy cập cho một đối tượng
- Danh sách gồm các phần tử là bộ đôi <miền, tập các quyền>
- Các hệ điều hành: UNIX, Windows sử dụng danh sách truy cập

Danh sách khả năng

- Mỗi hàng trong ma trận truy cập được cài đặt thành một danh sách khả năng cho một miền
- Một danh sách khả năng là một danh sách đối tượng kèm theo các quyền

Simpo PDF Merge and Split Unregistered Version - http://www.simpopdf.com

Cơ chế khóa – chìa

- Là sự kết hợp giữa danh sách truy cập và danh sách khả năng
- Đối tượng có danh sách các mẫu bit gọi là *khóa*
- Mỗi miền có danh sách các mẫu bit gọi là *chìa*
- Một tiến trình thực hiện trong một miền xem như có *chìa* và được thao tác trên đối tượng nếu *chìa* khớp với *khóa*

So sánh các phương pháp cài đặt ma trận truy cập

- Bảng toàn cục: Cài đặt đơn giản, tốn bộ nhớ
- Danh sách truy cập: Liên quan trực tiếp đến nhu cầu NSD, khó xác định quyền truy cập cho các miền
- Danh sách khả năng: Dễ dàng xác định quyền truy cập cho các miền, không liên quan trực tiếp đến nhu cầu NSD
- Khóa-chìa: Kết hợp được ưu điểm của danh sách truy cập và danh sách khả năng

Hủy bỏ quyền truy cập

- Các vấn đề cần xem xét:
  - Hủy ngay hay có trễ? Nếu có trễ→Khi nào?
  - Phạm vi ảnh hưởng: Toàn bộ NSD hay chỉ một nhóm NSD nhất định?
  - Hủy bỏ một số quyền nhất định hay tất cả các quyền?
  - Hủy tạm thời hay vĩnh viễn?

An ninh

Vấn đề an ninh
Xác thực
Các mối đe dọa chương trình và hệ thống
Mã hóa

Vấn đề an ninh

- An ninh: Xem xét môi trường bên ngoài hệ thống để bảo vệ hệ thống khỏi:
  - Truy cập trái phép
  - Sửa đổi hoặc phá hoại hệ thống
  - Vô tình làm hỏng tính nhất quán của hệ thống
- Dễ đảm bảo an ninh tránh các hành động vô ý hơn là đảm bảo anh ninh cho sự phá hoại/truy cập trái phép có mục đích

Xác thực

- Định danh người sử dụng thường được thực hiện qua mật khẩu
- Mật khẩu phải được giữ bí mật
  - Thường xuyên đổi mật khẩu
  - Sử dụng mật khẩu là các chuỗi ký tự khó đoán
  - Ghi lại tất cả những lần login không thành công
- Mật khẩu có thể được mã hóa hoặc sử dụng một lần (ví dụ: SecurID)
- Có thể sử dụng công nghệ mới, ví dụ xác thực sinh trắc học

Simpo PDF Merge and Split Unregistered Version - http://www.simpopdf.com

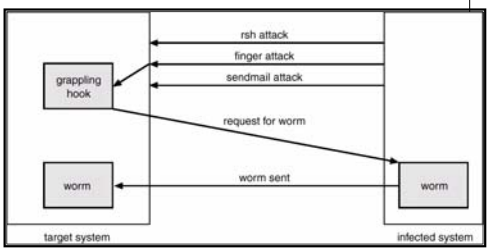
Đe dọa chương trình

- Trojan Horse (Con ngựa thành T'roa)
  - Đoạn mã được sử dụng sai mục đích
  - Khai thác cơ chế setuid
- Trap Door (Cửa sập)
  - Người lập trình để ngỏ một "cửa" chỉ mình anh ta biết để sử dụng sai mục đích, vi phạm an ninh
  - Có thể xuất hiện trong chương trình dịch
- Stack/buffer overflow (tràn bộ đệm/ngăn xếp)

Đe dọa hệ thống

- Worms (Sâu): Chương trình độc lập, có cơ chế tự sinh
- Internet worm (sâu Internet)
  - Khai thác đặc điểm mạng của UNIX (truy cập từ xa) lỗi trong các chương trình *finger* và *sendmail*
  - Grappling hook program uploaded main worm program.
- Viruses (Vi rút) – Đoạn mã ký sinh vào các chương trình khác
  - Chủ yếu ảnh hưởng đến các *máy vi tính*
  - Lây nhiễm qua các phương tiện lưu trữ, qua chương trình
  - *Safe computing*. (Tính toán an toàn)
- Từ chối dịch vụ: Làm cho máy bị tấn công hoạt động quá tải dẫn đến không phục vụ được các yêu cầu

Sâu Internet của Albert Morris (1998)



Kiểm soát các đe dọa

- Kiểm tra các hành động có thể gây mất an ninh (ví dụ liên tục gõ sai mật khẩu)
- Ghi nhật ký hệ thống: Thời gian, NSD các loại truy cập đến các đối tượng – hữu ích cho việc tìm ra cơ chế an ninh tốt hơn cũng như khôi phục việc mất an ninh
- Quét hệ thống định kỳ để tìm ra các lỗ hổng an ninh

Kiểm soát

- Cần kiểm soát:
  - Mật khẩu ngắn và dễ đoán (ví dụ abc123)
  - Các chương trình có setuid và chưa được xác thực
  - Các chương trình chưa được xác thực trong các thư mục hệ thống
  - Các tiến trình thực hiện quá lâu
  - Các thư mục được bảo vệ không đúng cách

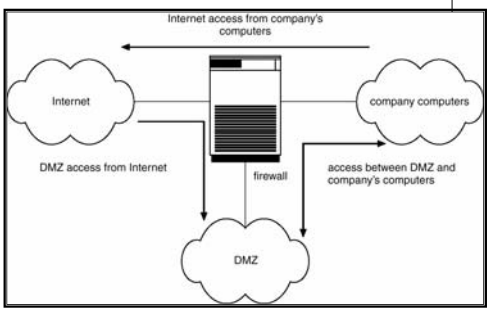
- Các tệp dữ liệu hệ thống được bảo vệ không đúng cách
- Các phần tử "nguy hiểm" trong PATH (ví dụ Trojan horse)
- Kiểm tra các chương trình hệ thống có bị thay đổi hay không thông qua checksum, MD5...

Simpo PDF Merge and Split Unregistered Version - http://www.simpopdf.com

Tường lửa (firewall)

- Tường lửa được đặt (hoạt động) giữa các máy chủ tin cậy và các máy không tin cậy
- Tường lửa hạn chế truy cập qua mạng giữa hai miền an ninh khác nhau

Tường lửa



Phát hiện đột nhập

- Phát hiện các cố gắng đột nhập vào hệ thống máy tính
- Phương pháp phát hiện:
  - "Kiểm toán" và ghi nhật ký
  - Tripwire (Phần mềm của UNIX kiểm tra xem một số tệp và thư mục có bị thay đổi không)
- Kiểm soát các hàm hệ thống

Mã hóa

- Mã hóa: Bản rõ → Bản mã
- Đặc điểm của kỹ thuật mã hóa tốt:
  - Tương đối đơn giản để NSD đã được xác thực có thể sử dụng để mã và giải mã
  - Sơ đồ mã hóa không chỉ phụ thuộc thuật toán mà còn phụ thuộc tham số (ví dụ khóa)
  - Rất khó phát hiện khóa
- Hệ mã hóa công khai sử dụng hai khóa
  - public key – khóa công khai, dùng để mã hóa.
  - private key – khóa bí mật, dùng để giải mã

Ví dụ mã hóa: SSL

- SSL – Secure Socket Layer
- Là giao thức mã hóa cho phép hai máy tính trao đổi dữ liệu an toàn với nhau
- Thường sử dụng giữa web server và browser để trao đổi thông tin một cách an toàn (ví dụ nạp số thẻ tín dụng)
- Web server được kiểm tra qua chứng chỉ
- Sau khi đã thiết lập kết nối an toàn SSL, hai máy tính truyền thông với khóa đối xứng

Phân loại an ninh máy tính

- Bộ Quốc phòng Mỹ chia an ninh máy tính thành 4 mức từ cao đến thấp: A, B, C, D.
- Xem thêm trong giáo trình về 4 mức an ninh này

Simpo PDF Merge and Split Unregistered Version - http://www.simpopdf.com

Các vấn đề cần nhớ

- Phân biệt bảo vệ và an ninh
- Quyền truy cập, miền bảo vệ
- Ma trận truy cập, các phương pháp cài đặt ma trận truy cập, so sánh các phương pháp đó
- An ninh máy tính: Xác thực, các mối đe dọa, kiểm soát, mã hóa

Tổng quan về hệ điều hành

Thuviendientu.org

TỔNG QUAN VỀ HỆ ĐIỀU HÀNH

I Mục tiêu

Sau khi học xong chương này, người học nắm được những kiến thức sau:

- Hiểu vai trò của hệ điều hành trong hệ thống máy tính
- Biết các giai đoạn phát triển của hệ điều hành
- Hiểu sự khác biệt của các hệ điều hành qua từng giai đoạn
- Hiểu cách giải quyết các vấn đề phát sinh trong từng hệ điều hành

II Giới thiệu

Hệ điều hành là một chương trình quản lý phần cứng máy tính. Nó cung cấp nền tảng cho các chương trình ứng dụng và đóng vai trò trung gian giao tiếp giữa người dùng máy tính và phần cứng của máy tính đó. Hệ điều hành thiết lập cho các tác vụ này rất đa dạng. Một vài hệ điều hành thiết kế tiện dụng trong khi một số khác thiết kế hiệu quả hoặc kết hợp cả hai.

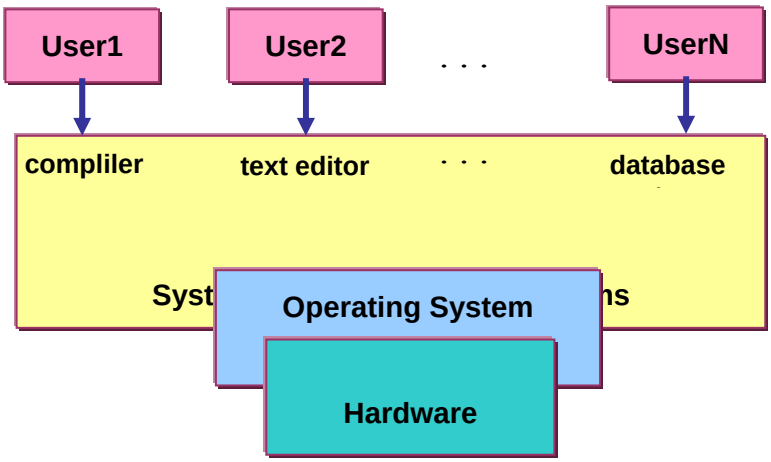
Để hiểu hệ điều hành là gì, trước hết chúng ta phải hiểu chúng được phát triển như thế nào. Trong chương này chúng ta điểm lại sự phát triển của hệ điều hành từ những hệ thử nghiệm đầu tiên tới những hệ đa chương và chia thời. Thông qua những giai đoạn khác nhau chúng ta sẽ thấy cách thức mà những thành phần của hệ điều hành được cải tiến như những giải pháp tự nhiên cho những vấn đề trong những hệ thống máy tính ban đầu. Xem xét những lý do phía sau sự phát triển của hệ điều hành cho chúng ta một đánh giá về những tác vụ gì hệ điều hành làm và cách hệ điều hành thực hiện chúng.

III Hệ điều hành là gì?

Một hệ điều hành là một thành phần quan trọng của mọi hệ thống máy tính. Một hệ thống máy tính có thể được chia thành bốn thành phần: phần cứng, hệ điều hành, các chương trình ứng dụng và người dùng.

- **Phần cứng** (Hardware): bao gồm bộ xử lý trung tâm (CPU), bộ nhớ, thiết bị xuất/nhập,..cung cấp tài nguyên cơ bản cho hệ thống.
- **Các chương trình ứng dụng** (application programs): trình biên dịch (compiler), trình soạn thảo văn bản (text editor), hệ cơ sở dữ liệu (database system), trình duyệt Web,..định nghĩa cách mà trong đó các tài nguyên được sử dụng để giải quyết yêu cầu của người dùng.
- **Người dùng** (user): có nhiều loại người dùng khác nhau, thực hiện những yêu cầu khác nhau, do đó sẽ có nhiều ứng dụng khác nhau.
- **Hệ điều hành** (operating system): hay còn gọi là chương trình hệ thống, điều khiển và hợp tác việc sử dụng phần cứng giữa những chương trình ứng dụng khác nhau cho những người dùng khác nhau. Hệ điều hành có thể được khám phá từ hai phía: người dùng và hệ thống.

Thuviendientu.org



Hình 0-1 Tầm nhìn trừu tượng các thành phần của một hệ thống máy tính

III.1 Tầm nhìn người dùng

Tầm nhìn người dùng của máy tính rất đa dạng bởi giao diện được dùng. Hầu hết những người dùng máy tính ngồi trước máy tính cá nhân gồm có màn hình, bàn phím, chuột và bộ xử lý hệ thống (system unit). Một hệ thống như thế được thiết kế cho một người dùng độc quyền sử dụng tài nguyên của nó để tối ưu hoá công việc mà người dùng đang thực hiện. Trong trường hợp này, hệ điều hành được thiết kế dễ dàng cho việc sử dụng với sự quan tâm về năng lực nhưng không quan tới việc sử dụng tài nguyên. Năng lực thực hiện là quan trọng với người dùng nhưng không là vấn đề nếu hầu hết hệ thống đang rãnh, chờ tốc độ xuất/nhập chậm từ phía người dùng.

Vài người dùng ngồi tại thiết bị đầu cuối (terminal) được nối kết tới máy tính lớn (mainframe) hay máy tính tầm trung (minicomputer). Những người khác đang truy xuất cùng máy tính thông qua các thiết bị đầu cuối khác. Những người dùng này chia sẻ các tài nguyên và có thể trao đổi thông tin. Hệ điều hành được thiết kế để tối ưu hoá việc sử dụng tài nguyên-để đảm bảo rằng tất cả thời gian sẵn dùng của CPU, bộ nhớ và thiết bị xuất nhập được sử dụng hữu hiệu và không cá nhân người dùng sử dụng độc quyền tài nguyên hơn là chia sẻ công bằng.

Những người dùng khác ngồi tại trạm làm việc, được nối kết tới mạng của các trạm làm việc khác và máy chủ. Những người dùng này có tài nguyên tận hiến là trạm làm việc của mình nhưng họ cũng chia sẻ các tài nguyên trên mạng và các máy chủ-tập tin, tính toán và các máy phục vụ in. Do đó, hệ điều hành của họ được thiết kế để thoả hiệp giữa khả năng sử dụng cá nhân và việc tận dụng tài nguyên.

Gần đây, sự đa dạng của máy tính xách tay trở thành thời trang cho những người làm việc trong lãnh vực công nghệ thông tin. Các thiết bị này được sử dụng chỉ bởi cá nhân người dùng. Một vài máy tính này được nối mạng hoặc nối trực tiếp bằng dây hay thông qua các modem không dây. Do sự giới hạn về năng lượng (điện) và giao diện, chúng thực hiện tương đối ít các thao tác ở xa. Hệ điều hành được thiết kế chủ yếu cho việc sử dụng cá nhân nhưng năng lực thực hiện trên thời gian sống của pin cũng là yếu tố quan trọng.

Một số máy tính có rất ít hay không có tầm nhìn người dùng. Thí dụ, các máy tính được nhúng vào các thiết bị gia đình và xe ôtô có thể có một bảng số và các đèn hiển thị trạng thái mở, tắt nhưng hầu hết chúng và các hệ điều hành được thiết kế để chạy mà không cần giao tiếp.

Thuviendientu.org

III.2 Tầm nhìn hệ thống

Từ quan điểm của máy tính, hệ điều hành là chương trình gần gủi với phần cứng. Chúng ta có thể thấy một hệ điều hành như bộ cấp phát tài nguyên. Hệ thống máy tính có nhiều tài nguyên - phần cứng và phần mềm - mà có thể được yêu cầu để giải quyết một vấn đề: thời gian CPU, không gian bộ nhớ, không gian lưu trữ tập tin, các thiết bị xuất/nhập,..Hệ điều hành hoạt động như bộ quản lý tài nguyên. Đương đầu với một lượng lớn các yêu cầu có thể xung đột về tài nguyên, hệ điều hành phải quyết định cách cấp phát tài nguyên tới những chương trình cụ thể và người dùng để có thể điều hành hệ thống máy tính hữu hiệu và công bằng.

Một tầm nhìn khác của hệ điều hành nhấn mạnh sự cần thiết để điều khiển các thiết bị xuất/nhập khác nhau và chương trình người dùng. Một hệ điều hành là một chương trình điều khiển. Chương trình điều khiển quản lý sự thực thi của các chương trình người dùng để ngăn chặn lỗi và việc sử dụng không hợp lý máy tính. Nó đặc biệt quan tâm với những thao tác và điều khiển các thiết bị nhập/xuất.

Nhìn chung, không có định nghĩa hoàn toàn đầy đủ về hệ điều hành. Các hệ điều hành tồn tại vì chúng là cách hợp lý để giải quyết vấn đề tạo ra một hệ thống máy tính có thể sử dụng. Mục tiêu cơ bản của hệ thống máy tính là thực thi chương trình người dùng và giải quyết vấn đề người dùng dễ dàng hơn. Hướng đến mục tiêu này, phần cứng máy tính được xây dựng. Tuy nhiên, chỉ đơn thuần là phần cứng thì không dễ sử dụng và phát triển các chương trình ứng dụng. Các chương trình khác nhau này đòi hỏi những thao tác chung nào đó, chẳng hạn như điều khiển thiết bị xuất/nhập. Sau đó, những chức năng chung về điều khiển và cấp phát tài nguyên được đặt lại với nhau vào một bộ phận phần mềm gọi là hệ điều hành.

Cũng không có định nghĩa bao quát nào được chấp nhận để xác định phần gì thuộc về hệ điều hành, phần gì không. Một quan điểm đơn giản là mọi thứ liên quan khi chúng ta ra lệnh hệ điều hành nên được xem xét. Tuy nhiên, những yêu cầu về bộ nhớ và những đặc điểm bên trong rất khác nhau trong từng hệ thống. Một định nghĩa bao quát hơn về hệ điều hành là một chương trình chạy liên tục trên máy tính (thường gọi là nhân kernel), những chương trình còn lại thuộc về chương trình ứng dụng.

III.3 Mục tiêu hệ thống

Định nghĩa những gì hệ điều hành làm thì dễ hơn xác định hệ điều hành là gì. Mục đích chính của hệ điều hành là dễ dàng sử dụng. Vì sự tồn tại của hệ điều hành hỗ trợ nhiều cho máy tính trong việc đáp ứng các ứng dụng của người dùng. Tầm nhìn này đặc biệt rõ ràng hơn khi nhìn hệ điều hành trên các máy tính cá nhân.

Mục tiêu thứ hai của hệ điều hành là điều hành hữu hiện hệ thống máy tính. Mục tiêu này đặc biệt quan trọng cho các hệ thống lớn, được chia sẻ, nhiều người dùng. Những hệ thống tiêu biểu này khá đắt, khai thác hiệu quả nhất các hệ thống này luôn là điều mong muốn. Tuy nhiên, hai mục tiêu tiện dụng và hữu hiệu đôi khi mâu thuẫn nhau. Trong quá khứ, xem xét tính hữu hiệu thường quan trọng hơn tính tiện dụng. Do đó, lý thuyết hệ điều hành tập trung nhiều vào việc tối ưu hoá sử dụng tài nguyên tính toán. Hệ điều hành cũng phát triển dần theo thời gian. Thí dụ, UNIX bắt đầu với bàn phím và máy in như giao diện của nó giới hạn tính tiện dụng đối với người dùng. Qua thời gian, phần cứng thay đổi và UNIX được gắn vào phần cứng mới với giao diện thân thiện với người dùng hơn. Nhiều giao diện người dùng đồ hoạ GUIs (graphical user interfaces) được bổ sung cho phép tiện dụng hơn với người dùng trong khi vẫn quan tâm tính hiệu quả.

Thuviendientu.org

Thiết kế hệ điều hành là một công việc phức tạp. Người thiết kế gặp phải nhiều sự thoả hiệp trong thiết kế và cài đặt. Nhiều người tham gia không chỉ mang đến hệ điều hành những lợi điểm mà còn liên tục xem xét và nâng cấp.
Để thấy rõ những hệ điều hành là gì và những gì hệ điều hành làm, chúng ta xem xét cách chúng phát triển trong bốn mươi lăm năm qua. Bằng cách lần theo sự tiến triển, chúng ta có thể xác định những thành phần của hệ điều hành và thấy cách thức và lý do hệ điều hành phát triển như chúng có.

Hệ điều hành và kiến trúc máy tính có mối quan hệ khăng khít nhau. Để dễ dàng sử dụng phần cứng, hệ điều hành được phát triển. Sau đó, các người dùng hệ điều hành đề nghị những chuyển đổi trong thiết kế phần cứng để đơn giản chúng. Nhìn lại lịch sử ngắn ngủi này, chú trọng cách giải quyết những vấn đề về hệ điều hành để giới thiệu những đặc điểm phần cứng.

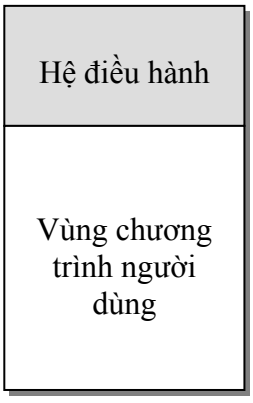
IV Hệ thống mainframe

Những hệ thống máy tính mainframe là những máy tính đầu tiên được dùng để xử lý ứng dụng thương mại và khoa học. Trong phần này, chúng ta lần theo sự phát triển của hệ thống mainframe từ các hệ thống bó (batch systems), ở đó máy tính chỉ chạy một-và chỉ một -ứng dụng, tới các hệ chia sẻ thời gian (time-shared systems), mà cho phép người dùng giao tiếp với hệ thống máy tính.

IV.1 Hệ thống bó

Những máy tính thời kỳ đầu là những máy cực lớn chạy từ một thiết bị cuối (console). Những thiết bị nhập thường là những bộ đọc thẻ và các ổ đĩa băng từ. Các thiết bị xuất thông thường thường là những máy in dòng (line printers), các ổ đĩa từ và các phiếu đục lỗ. Người dùng không giao tiếp trực tiếp với các hệ thống máy tính. Thay vào đó, người dùng chuẩn bị một công việc- chứa chương trình, dữ liệu và các thông tin điều khiển về tính tự nhiên của công việc-và gởi nó đến người điều hành máy tính. Công việc này thường được thực hiện trong các phiếu đục lỗ. Tại những thời điểm sau đó (sau vài phút, giờ hay ngày), dữ liệu xuất xuất hiện. Dữ liệu xuất chứa kết quả chương trình cũng như kết xuất bộ nhớ cuối cùng và nội dung các thanh ghi cho việc gở rối.

Hệ điều hành trong các máy tính thời kỳ đầu này tương đối đơn giản. Tác vụ chính là chuyển điều khiển tự động từ một công việc này sang công việc khác. Hệ điều hành luôn được thường trú trong bộ nhớ.



Hình 0-2 Sắp xếp bộ nhớ cho một hệ thống bó đơn giản

Thuviendientu.org

Để tăng tốc việc xử lý, người điều hành bó các công việc có cùng yêu cầu và chạy chúng thông qua máy tính như một nhóm. Do đó, các lập trình viên sẽ đưa chương trình của họ cho người điều hành. Người điều hành sẽ sắp xếp chương trình thành những bó với cùng yêu cầu và khi máy tính sẵn dùng sẽ chạy mỗi bó này. Dữ liệu xuất từ mỗi công việc sẽ gởi lại cho lập trình viên tương ứng.

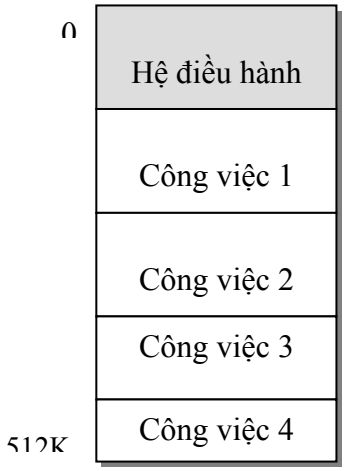
Trong môi trường thực thi này, CPU luôn rảnh vì tốc độ của các thiết bị xuất/nhập dạng cơ thực chất chậm hơn tốc độ của các thiết bị điện. Ngay cả một CPU chậm (với hàng ngàn chỉ thị lệnh được thực thi trên giây) cũng chỉ làm việc trong vài phần trăm giây. Thêm vào đó, một bộ đọc thẻ nhanh có thể đọc 1200 thẻ trong thời gian 1 phút (hay 20 thẻ trên giây). Do đó, sự khác biệt giữa tốc độ CPU và thiết bị xuất/nhập của nó có thể là 3 lần hay nhiều hơn. Dĩ nhiên theo thời gian, sự tiến bộ trong công nghệ dẫn đến sự ra đời những thiết bị nhập/xuất nhanh hơn. Tuy nhiên, tốc độ CPU tăng tới một tỷ lệ lớn hơn vì thế vấn đề không những không được giải quyết mà còn làm tăng.

Giới thiệu công nghệ đĩa cho phép hệ điều hành giữ tất cả công việc trên một đĩa hơn trong một bộ đọc thẻ tuần tự. Với việc truy xuất trực tiếp tới nhiều công việc, hệ điều hành có thể thực hiện định thời công việc, để sử dụng tài nguyên và thực hiện các tác vụ hữu hiệu.

IV.2 Hệ đa chương

Một khía cạnh quan trọng nhất của định thời công việc là khả năng đa chương. Thông thường, một người dùng giữ CPU hay các thiết bị xuất/nhập luôn bận. Đa chương gia tăng khả năng sử dụng CPU bằng cách tổ chức các công việc để CPU luôn có một công việc để thực thi.

Ý tưởng của kỹ thuật đa chương có thể minh hoạ như sau: Hệ điều hành giữ nhiều công việc trong bộ nhớ tại một thời điểm. Tập hợp các công việc này là tập con của các công việc được giữ trong vùng công việc-bởi vì số lượng các công việc có thể được giữ cùng lúc trong bộ nhớ thường nhỏ hơn số công việc có thể có trong vùng đệm. Hệ điều hành sẽ lấy và bắt đầu thực thi một trong các công việc có trong bộ nhớ. Cuối cùng, công việc phải chờ một vài tác vụ như một thao tác xuất/nhập để hoàn thành. Trong hệ thống đơn chương, CPU sẽ chờ ở trạng thái rỗi. Trong hệ thống đa chương, hệ điều hành sẽ chuyển sang thực thi công việc khác. Cuối cùng, công việc đầu tiên kết thúc việc chờ và nhận CPU trở lại. Chỉ cần ít nhất một công việc cần thực thi, CPU sẽ không bao giờ ở trạng thái rỗi.



Hình 0-3 Sắp xếp bộ nhớ cho hệ đa chương

Thuviendientu.org

Đa chương là một trường hợp đầu tiên khi hệ điều hành phải thực hiện quyết định cho những người dùng. Do đó, hệ điều hành đa chương tương đối tinh vi. Tất cả công việc đưa vào hệ thống được giữ trong vùng công việc. Vùng này chứa tất cả quá trình định vị trên đĩa chờ được cấp phát bộ nhớ chính. Nếu nhiều công việc sẵn sàng được mang vào bộ nhớ và nếu không đủ không gian cho tất cả thì hệ điều hành phải chọn một trong chúng. Khi hệ điều hành chọn một công việc từ vùng công việc, nó nạp công việc đó vào bộ nhớ để thực thi. Có nhiều chương trình trong bộ nhớ tại cùng thời điểm yêu cầu phải có sự quản lý bộ nhớ. Ngoài ra, nếu nhiều công việc sẵn sàng chạy cùng thời điểm, hệ thống phải chọn một trong chúng. Thực hiện quyết định này là định thời CPU. Cuối cùng, nhiều công việc chạy đồng hành đòi hỏi hoạt động của chúng có thể ảnh hưởng tới một công việc khác thì bị hạn chế trong tất cả giai đoạn của hệ điều hành bao gồm định thời quá trình, lưu trữ đĩa, quản lý bộ nhớ.

IV.3 Hệ chia thời

Hệ thống bó-đa chương cung cấp một môi trường nơi mà nhiều tài nguyên khác nhau (chẳng hạn như CPU, bộ nhớ, các thiết bị ngoại vi) được sử dụng hiệu quả. Tuy nhiên, nó không cung cấp giao tiếp người dùng với hệ thống máy tính. Chia thời (hay đa nhiệm) là sự mở rộng luận lý của đa chương. CPU thực thi nhiều công việc bằng cách chuyển đổi qua lại giữa chúng, nhưng những chuyển đổi xảy ra quá thường xuyên để người dùng có thể giao tiếp với mỗi chương trình trong khi chạy.

Một hệ thống máy tính giao tiếp (interactive computer) hay thực hành (hands-on computer system) cung cấp giao tiếp trực tuyến giữa người dùng và hệ thống. Người dùng cho những chỉ thị tới hệ điều hành hay trực tiếp tới một chương trình, sử dụng bàn phím hay chuột và chờ nhận kết quả tức thì. Do đó, thời gian đáp ứng nên ngắn-điển hình trong phạm vi 1 giây hay ít hơn.

Một hệ thống chia thời cho phép nhiều người dùng chia sẻ máy tính cùng một thời điểm. Vì mỗi hoạt động hay lệnh trong hệ chia thời được phục vụ ngắn, chỉ một ít thời gian CPU được yêu cầu cho mỗi người dùng. Khi hệ thống nhanh chóng chuyển từ một người dùng này sang người dùng kế, mỗi người dùng được cho cảm giác rằng toàn bộ hệ thống máy tính được tận hiến cho mình, nhưng thật sự máy tính đó đang được chia sẻ cho nhiều người dùng.

Một hệ điều hành chia thời sử dụng định thời CPU và đa chương để cung cấp mỗi người dùng với một phần nhỏ của máy tính chia thời. Mỗi người dùng có ít nhất một chương trình riêng trong bộ nhớ. Một chương trình được nạp vào trong bộ nhớ và thực thi thường được gọi là một quá trình. Khi một quá trình thực thi, điển hình nó thực thi chỉ tại một thời điểm ngắn trước khi nó kết thúc hay cần thực hiện xuất/nhập. Xuất/nhập có thể được giao tiếp; nghĩa là dữ liệu xuất hiển thị trên màn hình cho người dùng và dữ liệu nhập từ bàn phím, chuột hay thiết bị khác. Vì giao tiếp xuất/nhập chủ yếu chạy ở "tốc độ người dùng", nó có thể mất một khoảng thời gian dài để hoàn thành. Thí dụ, dữ liệu nhập có thể bị giới hạn bởi tốc độ nhập của người dùng; 7 ký tự trên giây là nhanh đối với người dùng, nhưng quá chậm so với máy tính. Thay vì để CPU rảnh khi người dùng nhập liệu, hệ điều hành sẽ nhanh chóng chuyển CPU tới một chương trình khác.

Hệ điều hành chia thời phức tạp hơn nhiều so với hệ điều hành đa chương. Trong cả hai dạng, nhiều công việc được giữ cùng lúc trong bộ nhớ vì thế hệ thống phải có cơ chế quản lý bộ nhớ và bảo vệ. Để đạt được thời gian đáp ứng hợp lý, các công việc có thể được hoán vị vào ra bộ nhớ chính. Một phương pháp chung để đạt mục tiêu này là bộ nhớ ảo, là kỹ thuật cho phép việc thực thi của một công việc có thể không hoàn toàn ở trong bộ nhớ. Ưu điểm chính của cơ chế bộ nhớ ảo là các chương

Thuviendientu.org

trình có thể lớn hơn bộ nhớ vật lý. Ngoài ra, nó trừu tượng hoá bộ nhớ chính thành mảng lưu trữ lớn và đồng nhất, chia bộ nhớ luận lý như được thấy bởi người dùng từ bộ nhớ vật lý. Sự sắp xếp này giải phóng lập trình viên quan tâm đến giới hạn lưu trữ của bộ nhớ.

Các hệ chia thời cũng phải cung cấp một hệ thống tập tin. Hệ thống tập tin định vị trên một tập hợp đĩa; do đó quản lý đĩa phải được cung cấp. Hệ chia thời cũng cung cấp cơ chế cho việc thực thi đồng hành, yêu cầu cơ chế định thời CPU tinh vi. Để đảm bảo thứ tự thực thi, hệ thống phải cung cấp các cơ chế cho việc đồng bộ hoá và giao tiếp công việc, và có thể đảm bảo rằng các công việc không bị deadlock, chờ đợi công việc khác mãi mãi.

Ý tưởng chia thời được giới thiệu trong những năm 1960, nhưng vì hệ chia thời là phức tạp và rất đắt để xây dựng, chúng không phổ biến cho tới những năm 1970. Mặc dù xử lý theo bó vẫn được thực hiện nhưng hầu hết hệ thống ngày nay là chia thời. Do đó, đa chương và chia thời là những chủ đế trung tâm của hệ điều hành hiện đại và chúng là chủ đề trọng tâm của giáo trình này.

V Hệ để bàn (Desktop system)

Máy tính cá nhân (PC) xuất hiện vào những năm 1970. Trong suốt thập niên đầu, CPU trong PC thiếu các đặc điểm cần thiết để bảo vệ hệ điều hành từ chương trình người dùng. Do đó, các hệ điều hành PC không là đa người dùng hoặc đa nhiệm. Tuy nhiên, các mục tiêu của hệ điều hành này thay đổi theo thời gian; thay vì tối ưu hoá việc sử dụng CPU và thiết bị ngoại vi, các hệ thống chọn lựa tối ưu hoá sự tiện dụng và đáp ứng người dùng. Các hệ thống này gồm các PC chạy các hệ điều hành Microsoft Windows và Apple Macintosh. Hệ điều hành MS-DOS từ Microsoft được thay thế bằng nhiều ấn bản của Microsoft Windows và IBM đã nâng cấp MS-DOS thành hệ đa nhiệm OS/2. Hệ điều hành Apple Macintosh được gắn nhiều phần cứng hiện đại hơn và ngày nay chứa nhiều đặc điểm mới như bộ nhớ ảo và đa nhiệm. Với sự phát hành MacOS X, lõi của hệ điều hành ngày nay dựa trên Mach và FreeBSD UNIX cho sự mở rộng, năng lực và đặc điểm nhưng nó vẫn giữ lại giao diện đồ hoạ người dùng GUI. LINUX, một hệ điều hành tương tự như UNIX sẵn dùng cho máy PC trở nên phổ biến gần đây.

Hệ điều hành cho các máy tính này có những thuận lợi trong nhiều cách từ sự phát triển của hệ điều hành cho mainframes. Máy vi tính (microcomputer) lập tức có thể được chấp nhận một số công nghệ được phát triển cho hệ điều hành lớn hơn. Thêm vào đó, chi phí phần cứng cho máy vi tính đủ thấp để các cá nhân có thể một mình sử dụng máy tính, và sử dụng CPU không còn quan trọng nữa. Do đó, những quyết định thiết kế được thực hiện trong hệ điều hành cho mainframes có thể không hợp lý cho các hệ thống nhỏ hơn.

Những quyết định thiết kế khác vẫn được áp dụng. Thí dụ, trước hết bảo vệ hệ thống tập tin không cần thiết trên máy cá nhân. Tuy nhiên, hiện nay các máy tính này thường được nối vào các máy tính khác qua mạng cục bộ hay Internet. Khi những máy tính khác và người dùng khác có thể truy xuất các tập tin này trên một PC, bảo vệ tập tin một lần nữa cũng trở thành một đặc điểm cần thiết của hệ điều hành. Sự thiếu bảo vệ tạo điều kiện dễ dàng cho những chương trình hiểm phá huỷ dữ liệu trên những hệ thống như MS-DOS và hệ điều hành Macintosh. Các chương trình này có thể tự nhân bản và phát tán nhanh chóng bằng cơ chế worm hay virus và làm tê liệt mạng của các công ty hay thậm chí mạng toàn cầu. Đặc điểm chia thời được cải tiến như bộ nhớ bảo vệ và quyền tập tin là chưa đủ để bảo vệ một hệ thống từ sự tấn công.

Thuviendientu.org

VI Hệ đa xử lý

Hầu hết các hệ thống ngày nay là các hệ thống đơn xử lý; nghĩa là chỉ có một CPU chính. Tuy nhiên, các hệ thống đa xử lý (hay còn gọi là hệ song song hay hệ kết nối chặt) được phát triển rất quan trọng. Các hệ thống như thế có nhiều hơn một bộ xử lý trong giao tiếp gần, chia sẻ bus máy tính, đồng hồ, đôi khi còn là bộ nhớ hay thiết bị ngoại vi.

Hệ thống đa xử lý có ba ưu điểm chính:

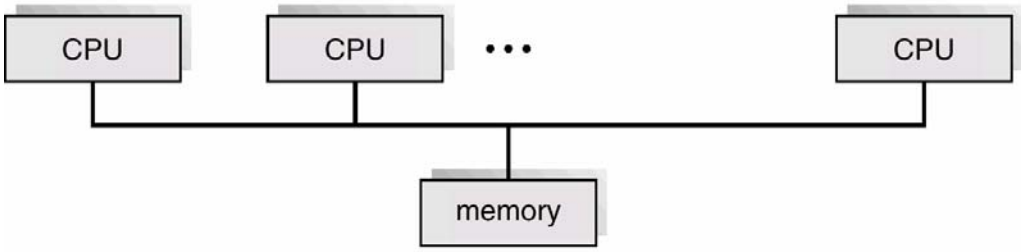
- **Thông lượng được gia tăng**: bằng cách tăng số lượng bộ xử lý, chúng ta hy vọng thực hiện nhiều công việc hơn với thời gian ít hơn. Tỉ lệ giữa sự tăng tốc với N bộ xử lý không là N; đúng hơn nó nhỏ hơn N. Trong khi nhiều bộ xử lý cộng tác trên một công việc, một lượng chi phí phải chịu trong việc giữ các thành phần làm việc phù hợp. Chi phí này cộng với chi phí cạnh tranh tài nguyên được chia sẻ, làm giảm kết quả được mong đợi từ những bộ xử lý bổ sung. Tương tự như một nhóm gồm N lập trình viên làm việc với nhau không dẫn đến kết quả công việc đang đạt được tăng N lần.
- **Tính kinh tế của việc mở rộng**: hệ thống đa xử lý có thể tiết kiệm nhiều chi phí hơn hệ thống đơn bộ xử lý, bởi vì chúng có thể chia sẻ ngoại vi, thiết bị lưu trữ và điện. Nếu nhiều chương trình điều hành trên cùng tập hợp dữ liệu thì lưu trữ dữ liệu đó trên một đĩa và tất cả bộ xử lý chia sẻ chúng sẽ rẻ hơn là có nhiều máy tính với đĩa cục bộ và nhiều bản sao dữ liệu.
- **Khả năng tin cậy được gia tăng**: nếu các chức năng được phân bổ hợp lý giữa các bộ xử lý thì lỗi trên một bộ xử lý sẽ không dừng hệ thống, chỉ năng lực bị giảm. Nếu chúng ta có 10 bộ xử lý và có 1 bộ xử lý bị sự cố thì mỗi bộ xử lý trong 9 bộ xử lý còn lại phải chia sẻ của công việc của bộ xử lý bị lỗi. Do đó, toàn bộ hệ thống chỉ giảm 10% năng lực hơn là dừng hoạt động. Các hệ thống được thiết kế như thế được gọi là hệ thống có khả năng chịu lỗi (fault tolerant).

Việc điều hành vẫn tiếp tục trong sự hiện diện của lỗi yêu cầu một cơ chế cho phép lỗi được phát hiện, chuẩn đoán và sửa lỗi nếu có thể. Hệ thống Tandem sử dụng sự nhân đôi phần cứng và phần mềm để đảm bảo sự điều hành vẫn tiếp tục mặc dù có lỗi xảy ra. Hệ thống này chứa hai bộ xử lý, mỗi bộ xử lý có bộ nhớ cục bộ riêng. Các bộ xử lý được nối kết bởi một bus. Một bộ xử lý chính và bộ xử lý kia là dự phòng. Cả hai bản sao được giữ ở mỗi bộ xử lý: một là chính và một là dự phòng. Tại các điểm kiểm tra (checkpoints) trong việc thực thi của hệ thống, thông tin trạng thái của mỗi công việc-gồm một bản sao hình ảnh bộ nhớ-được chép từ máy chính tới máy dự phòng. Nếu một lỗi được phát hiện, bản sao dự phòng được kích hoạt và được khởi động lại từ điểm kiểm tra mới nhất. Giải pháp này đắt vì nó bao gồm việc nhân đôi phần cứng.

Các hệ thống đa xử lý thông dụng nhất hiện nay sử dụng **đa xử lý đối xứng** (symmetric multiprocessing-SMP). Trong hệ thống này mỗi bộ xử lý chạy bản sao của hệ điều hành và những bản sao này giao tiếp với các bản sao khác khi cần. Vài hệ thống sử dụng **đa xử lý bất đối xứng** (asymmetric multiprocessing). Trong hệ thống này mỗi bộ xử lý được gán một công việc xác định. Một bộ xử lý chủ điều khiển hệ thống; những bộ xử lý còn lại hoặc chờ bộ xử lý chủ ra chỉ thị hoặc có những tác vụ được định nghĩa trước. Cơ chế này định nghĩa mối quan hệ chủ-tớ. Bộ xử lý chính lập thời biểu và cấp phát công việc tới các bộ xử lý tớ.

Thuviendientu.org

Đa xử lý đối xứng có nghĩa tất cả bộ xử lý là ngang hàng; không có mối quan hệ chủ-tớ tồn tại giữa các bộ xử lý. Hình I-4 minh hoạ một kiến trúc đa xử lý đối xứng điển hình. Một thí dụ của đa xử lý đối xứng là ấn bản của Encore của UNIX cho máy tính Multimax. Máy tính này có thể được cấu hình như nó đang thực hiện nhiều bộ xử lý, tất cả bộ xử lý đều chạy bản sao của UNIX. Ưu điểm của mô hình này là nhiều quá trình có thể chạy cùng một lúc - N quá trình có thể chạy nếu có N CPU- không gây ra sự giảm sút to lớn về năng lực. Tuy nhiên, chúng ta phải điều khiển cẩn thận xuất/nhập để đảm bảo rằng dữ liệu dẫn tới bộ xử lý tương ứng. Vì các CPU là riêng rẻ, một CPU có thể đang rảnh trong khi CPU khác quá tải dẫn đến việc sử dụng không hữu hiệu tài nguyên của hệ thống. Sự không hiệu quả này có thể tránh được nếu các bộ xử lý chia sẻ các cấu trúc dữ liệu. Một hệ thống đa xử lý của dạng này sẽ cho phép các quá trình và tài nguyên – như bộ nhớ - được chia sẻ tự động giữa các quá trình khác nhau và có thể làm giảm sự khác biệt giữa các bộ xử lý. Hầu như tất cả hệ điều hành hiện đại - gồm Windows NT, Solaris, Digital UNIX, OS/2 và LINUX - hiện nay cung cấp sự hỗ trợ đa xử lý đối xứng.



Hình 0-4 Kiến trúc đa xử lý đối xứng

Sự khác biệt giữa đa xử lý đối xứng và bất đối xứng có thể là do phần cứng hoặc phần mềm. Phần cứng đặc biệt có thể khác nhau trên nhiều bộ xử lý, hoặc phần mềm có thể được viết để cho phép chỉ một chủ và nhiều tớ. Thí dụ, SunOS ấn bản 4 cung cấp đa xử lý không đối xứng, ngược lại, ấn bản 5 (Solaris 2) là đối xứng trên cùng phần cứng.

Khi các bộ vi xử lý trở nên rẻ hơn và mạnh hơn các chức năng bổ sung của hệ điều hành là chuyển tới bộ xử lý tớ. Thí dụ, tương đối dễ để thêm bộ vi xử lý với bộ nhớ riêng để quản lý hệ thống đĩa. Bộ vi xử lý có thể nhận một chuỗi các yêu cầu từ bộ nhớ chính và cài đặt hàng đợi đĩa riêng và giải thuật định thời. Sự sắp xếp này làm giảm chi phí định thời đĩa của CPU. PC chứa một bộ vi xử lý trong bàn phím để chuyển những phím nóng thành mã để gởi tới CPU. Thực tế, việc sử dụng các bộ vi xử lý trở nên quá phổ biến đến nỗi mà đa xử lý không còn được xem xét.

VII Hệ phân tán

Một mạng, trong thuật ngữ đơn giản nhất, là một đường dẫn truyền thông giữa hai hay nhiều hệ thống. Hệ phân tán phụ thuộc vào mạng với những khả năng của nó. Bằng cách cho phép truyền thông, hệ phân tán có thể chia sẻ các tác vụ tính toán và cung cấp nhiều chức năng tới người dùng.

Các mạng với sự đa dạng về giao thức được dùng, khoảng cách giữa các nút và phương tiện truyền. TCP/IP là giao thức mạng phổ biến nhất mặc dù ATM và các giao thức khác được sử dụng rộng rãi. Tương tự, hệ điều hành hỗ trợ sự đa dạng về giao thức. Hầu hết các hệ điều hành hỗ trợ TCP/IP, gồm Windows và UNIX. Một số hệ điều hành khác hỗ trợ các giao thức riêng phù hợp với yêu cầu của chúng. Đối với một hệ điều hành, một giao thức mạng chỉ cần một thiết bị giao diện – thí dụ: một

Thuviendientu.org

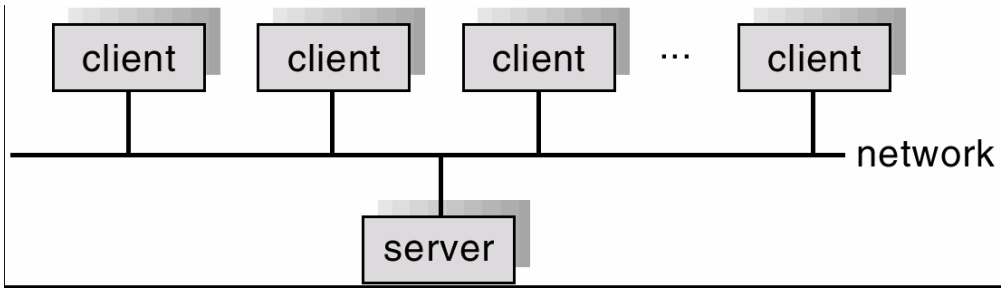
card mạng-với một trình điều khiển thiết bị để quản lý nó và một phần mềm để đóng gói dữ liệu trong giao thức giao tiếp để gởi nó và mở gói để nhận nó.

Mạng thường dựa trên các khoảng cách giữa các nút. Một mạng cục bộ (LAN-Local Area Network) tồn tại trong phạm vi một phòng, một tầng, hay một toà nhà. Một mạng diện rộng (Wide-Area Network) thường tồn tại giữa các toà nhà, các thành phố, các quốc gia. Một công ty toàn cầu có thể có một mạng diện rộng để nối kết tới các văn phòng của nó. Các mạng này có thể chạy với một hay nhiều giao thức. Sự tiến bộ liên tục của công nghệ mới hình thành nhiều dạng mạng khác nhau. Thí dụ, mạng đô thị (MAN-Metropolitan Area Network) cũng liên kết các toà nhà trong cùng một thành phố. Các thiết bị BlueTooth giao tiếp qua một khoảng cách ngắn khoảng vài bộ (feet), chủ yếu tạo ra một mạng phạm vi nhỏ (small-area network).

Các phương tiện truyền thông làm các mạng tương đối khác nhau. Chúng gồm cáp đồng, cáp quang, truyền không dây giữa vệ tinh, các đĩa vi sóng (microware dishes) và sóng radio. Khi các thiết bị tính toán được nối kết tới các điện thoại di động, chúng tạo ra một mạng. Thậm chí rất nhiều giao tiếp hồng ngoại dãy ngắn có thể được dùng cho mạng. Tại cấp độ cơ bản, bất cứ khi nào các máy tính giao tiếp chúng sử dụng hay tạo ra một mạng. Các mạng này cũng rất khác nhau về năng lực và khả năng tin cậy.

VII.1 Hệ khách hàng-máy phục vụ

Một PC có thể trở nên nhanh hơn, mạnh hơn, rẻ hơn nếu người thiết kế thay đổi rất xa từ kiến trúc hệ thống tập trung. Các thiết bị đầu cuối được nối kết tới các hệ tập trung hiện nay đang được thay thế bởi các PC. Tương ứng, chức năng giao diện người dùng được dùng quản lý trực tiếp bởi các hệ tập trung đang được quản lý tăng dần bởi các PC. Do đó, các hệ tập trung ngày nay hoạt động như hệ máy phục vụ để thoả mãn các yêu cầu phát sinh bởi hệ thống khách hàng. Đặc điểm chung của hệ khách hàng- máy phục vụ được mô tả trong hình-I-5:



Hình 0-5 Cấu trúc chung của hệ khách hàng-máy phục vụ

Các hệ máy chủ có thể được phân loại rộng rãi như máy phục vụ tính toán và máy phục vụ tập tin

- **Hệ máy phục vụ tính toán** (Compute-server systems): cung cấp giao diện mà khách hàng có thể gởi các yêu cầu để thực hiện hoạt động. Chúng thực thi hoạt động và gởi kết quả cho khách hàng.
- **Hệ máy phục vụ tập tin** (File-server systems): cung cấp một giao diện hệ thống tập tin nơi khách hàng có thể tạo, cập nhật, đọc và xoá tập tin.

Thuviendientu.org

VII.2 Hệ ngang hàng

Sự phát triển của mạng máy tính-đặc biệt là Internet và Word Wide Web (WWW)-có ảnh hưởng sâu sắc đến sự phát triển gần đây của hệ điều hành. Khi PC được giới thiệu vào những năm 1970, chúng được thiết kế cho việc sử dụng "cá nhân" và thường được xem như là các máy tính đơn lẻ (standalone computer). Với việc bắt đầu sử dụng Internet phổ biến và rộng rãi vào những năm 1980 với e-mail, ftp, gopher, nhiều PC được nối vào mạng máy tính. Với sự giới thiệu dịch vụ Web vào giữa những năm 1990, nối kết mạng trở thành một thành phần quan trọng của một hệ thống mạng máy tính.

Gần như tất cả máy tính PC hiện đại và các trạm làm việc có thể chạy trình duyệt Web để truy xuất tài liệu siêu văn bản trên Web. Các hệ điều hành (như Windows, OS/2, MacOS và UNIX) hiện nay cũng chứa phần mềm hệ thống (như TCP/IP và PPP) cho phép một máy tính truy xuất Internet qua một mạng cục bộ hay nối kết qua đường điện thoại. Nhiều hệ điều hành chứa trình duyệt Web cũng như khách hàng và máy phục vụ e-mail, đăng nhập từ xa, và ftp.

Tương phản với các hệ thống kết nối chặt được thảo luận trong phần I.6, mạng máy tính được dùng trong các ứng dụng này gồm tập hợp các bộ xử lý không chia sẻ bộ nhớ hay đồng hồ. Thay vào đó, mỗi bộ xử lý có bộ nhớ cục bộ riêng. Bộ xử lý giao tiếp với bộ xử lý khác thông qua các đường truyền thông như các bus tốc độ cao hay các đường điện thoại. Các hệ thống này thường được xem như các hệ thống kết nối lỏng (hay hệ thống phân tán).

Vài hệ điều hành thực hiện khái niệm mạng hơn là chú trọng cung cấp nối kết mạng. Một hệ điều hành mạng là một hệ điều hành cung cấp các đặc tính như chia sẻ tập tin qua mạng, nó chứa một cơ chế giao tiếp cho phép các quá trình khác nhau trên các máy khác nhau trao đổi thông điệp. Một máy tính chạy một hệ điều hành mạng hoạt động tự trị từ tất cả máy tính khác trên mạng, mặc dù nó nhận thức sự hiện diện của mạng và có thể giao tiếp với các máy tính được nối mạng khác. Một hệ điều hành phân tán là một môi trường ít tự trị hơn: Các hệ điều hành phân tán giao tiếp đủ gần để cung cấp một hình ảnh mà chỉ một hệ điều hành đơn lẻ điều khiển mạng.

VIII Hệ thống nhóm (Clustered Systems)

Tương tự các hệ song song, hệ thống nhóm tập hợp nhiều CPUs với nhau để thực hiện công việc tính toán. Tuy nhiên, hệ thống nhóm khác hệ thống song song ở điểm chúng được hợp thành từ hai hay nhiều hệ thống đơn được kết hợp với nhau. Định nghĩa của thuật ngữ nhóm (clustered) là không cụ thể. Định nghĩa thông thường có thể chấp nhận là các máy tính nhóm chia sẻ việc lưu trữ và được liên kết gần qua LAN.

Nhóm thường được thực hiện để cung cấp khả năng sẵn dùng cao. Một lớp phần mềm nhóm chạy trên các nút nhóm (cluster nodes). Mỗi nút có thể kiểm soát một hay nhiều hơn một nút (qua mạng LAN). Nếu máy bị kiểm soát gặp sự cố, máy kiểm soát có thể lấy quyền sở hữu việc lưu trữ của nó và khởi động lại (các) ứng dụng mà chúng đang chạy trên máy bị sự cố. Máy bị sự cố vẫn chưa hoạt động nhưng người dùng và khách hàng của ứng dụng chỉ thấy một sự gián đoạn ngắn của dịch vụ.

Trong nhóm bất đối xứng (asymmetric clustering), một máy ở trong chế độ dự phòng nóng (hot standby) trong khi các máy khác đang chạy các ứng dụng. Máy dự phòng không là gì cả ngoại trừ theo dõi server hoạt động. Nếu server đó bị lỗi, máy chủ dự phòng nóng trở thành server hoạt động. Trong chế độ đối xứng (symmetric mode), hai hay nhiều máy chủ đang chạy ứng dụng và chúng đang kiểm soát lẫn nhau. Chế độ

Thuviendientu.org

này chú trọng tính hiệu quả khi nó sử dụng tất cả phần cứng sẵn có. Nó thực hiện yêu cầu nhiều hơn một ứng dụng sẵn dùng để chạy.

Các hình thức khác của nhóm gồm các nhóm song song (parallel clusters) và nhóm qua một WAN. Các nhóm song song cho phép nhiều máy chủ truy xuất cùng dữ liệu trên thiết bị lưu trữ được chia sẻ. Vì hầu hết các hệ điều hành hỗ trợ nghèo nàn việc truy xuất dữ liệu đồng thời bởi nhiều máy chủ, các nhóm song song thường được thực hiện bởi các ấn bản phần mềm đặc biệt và sự phát hành của các ứng dụng đặc biệt. Thí dụ, Oracle Parallel Server là một ấn bản cơ sở dữ liệu của Oracle, và lớp phần mềm ghi vết việc truy xuất tới đĩa được chia sẻ. Mỗi máy có truy xuất đầy đủ tới dữ liệu trong cơ sở dữ liệu.

Mặc dù có nhiều cải tiến trong tính toán phân tán, hầu hết các hệ thống không cung cấp các hệ thống tập tin phân tán mục đích chung (general-purpose distributed file systems). Do đó, hầu hết các nhóm không cho phép truy xuất được chia sẻ tới dữ liệu trên đĩa. Cho mục đích này, các hệ thống tập tin phân tán phải cung cấp điều khiển truy xuất và khoá các tập tin để đảm bảo không có các thao tác xung đột xảy ra. Loại dịch vụ này thường được gọi là bộ quản lý khoá phân tán (distributed lock manager-DLM).

Công nghệ nhóm đang nhanh chóng thay đổi. Những định hướng nhóm gồm các nhóm toàn cục, trong đó các máy có thể định vị bất cứ nơi nào trên thế giới (hay bất cứ nơi nào một WAN đạt tới). Các dự án như thế vẫn là chủ đề cho nghiên cứu và phát triển.

Việc sử dụng hệ thống nhóm và các đặc tính nên mở rộng như mạng vùng lưu trữ (storage-area networks-SANs) chiếm ưu thế. SANs cho phép gán dễ dàng nhiều máy chủ tới nhiều đơn vị lưu trữ. Các nhóm hiện tại thường bị giới hạn tới hai hay bốn máy chủ do sự phức tạp của nối kết các máy chủ tới thiết bị lưu trữ được chia sẻ.

IX Hệ thời thực

Một dạng khác của hệ điều hành có mục đích đặc biệt là hệ thời thực (real-time system). Hệ thời thực được dùng khi các yêu cầu thời gian khắt khe được đặt trên thao tác của một bộ xử lý hay dòng dữ liệu; do đó, nó thường được dùng như một thiết bị điều khiển trong một ứng dụng tận hiến. Các bộ cảm biến mang dữ liệu tới máy tính. Máy tính phải phân tích dữ liệu và có thể thích ứng các điều khiển để hiệu chỉnh các dữ liệu nhập cảm biến. Các hệ thống điều khiển các thí nghiệm khoa học, hệ thống ảnh hoá y tế, hệ thống điều khiển công nghệ và các hệ thống hiển thị,... Các hệ thống phun dầu động cơ ôtô, các bộ điều khiển dụng cụ trong nhà, hệ thống vũ khí cũng là các hệ thống thời thực.

Một hệ thống thời thực có sự ràng buộc cố định, rõ ràng. Xử lý phải được thực hiện trong phạm vi các ràng buộc được định nghĩa hay hệ thống sẽ thất bại. Một hệ thời thực thực hiện đúng chức năng chỉ nếu nó trả về kết quả đúng trong thời gian ràng buộc. Tương phản với yêu cầu này trong hệ chia thời, ở đó nó mong muốn (nhưng không bắt buộc) đáp ứng nhanh, hay đối với hệ thống bó, nó không có ràng buộc thời gian gì cả.

Hệ thời thực có hai dạng: cứng và mềm. Hệ thời thực cứng đảm bảo rằng các tác vụ tới hạn được hoàn thành đúng giờ. Mục tiêu này đòi hỏi tất cả trì hoãn trong hệ thống bị giới hạn, từ việc lấy lại dữ liệu được lưu trữ thời gian hệ điều hành hoàn thành bất cứ yêu cầu cho nó. Các ràng buộc thời gian như thế ra lệnh các phương tiện sẵn có trong hệ thời thực cứng. Thiết bị lưu trữ phụ của bất cứ thứ hạng nào thường bị giới hạn hay bị mất với dữ liệu đang được lưu trong bộ nhớ lưu trữ ngắn hạn (short-term memory) hay trong bộ nhớ chỉ đọc (ROM). Hầu hết các hệ điều hành hiện đại

Thuviendientu.org

không cung cấp đặc điểm này vì chúng có khuynh hướng tách rời người dùng từ phần cứng và sự tách rời này dẫn đến lượng thời gian không xác định mà thao tác sẽ mất. Thí dụ, bộ nhớ ảo hầu như chưa bao giờ thấy trong hệ thời thực. Do đó, những hệ thời thực cứng xung đột với thao tác của hệ chia thời và hai hệ này không thể đan xen nhau. Vì không có hệ điều hành đa mục đích đã có hỗ trợ chức năng thời thực cứng; chúng ta không tập trung với loại hệ thống này trong chương này.

Một loại thời thực ít hạn chế hơn là hệ thời thực mềm, ở đó tác vụ thời thực tới hạn có độ ưu tiên hơn các tác vụ khác và duy trì độ ưu tiên đó cho đến khi chúng hoàn thành. Như trong hệ thời thực cứng, sự trì hoãn nhân (kernel) của hệ điều hành trì hoãn yêu cầu được giới hạn. Một tác vụ thời thực không thể giữ việc chờ không xác định đối với nhân để thực thi. Thời thực mềm là mục tiêu có thể đạt được và có thể được đan xen với các loại hệ thống khác. Tuy nhiên, hệ thời thực mềm có những tiện ích giới hạn hơn hệ thời thực cứng. Vì không hỗ trợ tốt cho thời điểm tới hạn, nên hệ thời thực mềm dễ gây rủi ro khi dùng cho việc kiểm soát công nghệ và tự động hoá. Tuy nhiên, chúng có ích trong nhiều lĩnh vực như đa phương tiện, thực tế ảo, dự án khoa học tiên tiến-như khám phá trong lòng đại dương và khám phá hành tinh. Những hệ thống này cần những đặc điểm hệ điều hành tiên tiến mà không được hỗ trợ bởi hệ thời thực cứng. Vì việc sử dụng chức năng thời thực mềm được mở rộng nên chúng ta đang tìm cách đưa chúng vào trong hầu hết các hệ điều hành hiện tại, gồm các ấn bản chính thức của UNIX.

X Hệ xách tay

Hệ xách tay gồm các máy hỗ trợ cá nhân dùng kỹ thuật số (personal digital assistants-PDAs) như Palm hay điện thoại di động (cellular telephone) với nối kết tới mạng như Internet. Những người phát triển hệ xách tay và ứng dụng gặp phải nhiều thử thách, nhất là sự giới hạn về kích thước của thiết bị. Thí dụ, một PDA điển hình cao khoảng 5 inches và rộng khoảng 3 inches và trọng lượng của nó ít hơn 0.5 pound. Do sự giới hạn về kích thước này, hầu hết các thiết bị xách tay có bộ nhớ nhỏ gồm các bộ xử lý thấp và màn hình hiển thị nhỏ. Bây giờ chúng ta sẽ xem xét mỗi sự giới hạn này.

Nhiều thiết bị xách tay có dung lượng bộ nhớ 512KB và 8 MB (ngược lại, các máy PC hay trạm làm việc có hàng trăm MB bộ nhớ). Do đó, hệ điều hành và các ứng dụng phải quản lý bộ nhớ hiệu quả. Điều này gồm trả về tất cả bộ nhớ được cấp phát tới bộ quản lý bộ nhớ một khi bộ nhớ không còn được dùng nữa. Hiện nay, nhiều thiết bị xách tay không dùng kỹ thuật bộ nhớ ảo do đó buộc người phát triển chương trình làm việc trong phạm vi giới hạn của bộ nhớ vật lý.

Vấn đề thứ hai quan tâm đến người phát triển các thiết bị xách tay là tốc độ của bộ xử lý được dùng trong thiết bị. Các bộ xử lý đối với hầu hết các thiết bị xách tay thường chạy với tốc độ chỉ bằng một phần tốc độ của một bộ xử lý trong máy PC. Các bộ xử lý nhanh hơn yêu cầu điện năng nhiều hơn. Để chứa một bộ xử lý nhanh hơn bên trong thiết bị xách tay nên yêu cầu nhiều pin hơn hơn và phải được nạp lại thường xuyên. Để tối thiểu hoá kích thước của các thiết bị xách tay đòi hỏi bộ xử lý nhỏ hơn, chậm hơn tiêu thụ ít điện năng hơn. Do đó, hệ điều hành và các ứng dụng phải được thiết kế không đòi hỏi sử dụng nhiều bộ xử lý.

Vấn đề cuối cùng gây khó khăn cho người thiết kế chương trình cho các thiết bị xách tay là màn hình hiển thị nhỏ. Trong khi một màn hình cho máy tính ở nhà kích thước có thể 21 inches, màn hình cho thiết bị xách tay thường có diện tích không quá 3 inches. Những tác vụ quen thuộc như đọc e-mail hay hiển thị các trang web, phải được cô đọng vào màn hình nhỏ hơn. Một phương pháp để hiển thị nội dung các trang

Thuviendientu.org

web là cắt xén web (web clipping), ở đó chỉ một tập hợp nhỏ trang web được phân phát và hiển thị trên thiết bị xách tay.

Một số thiết bị xách tay có thể dùng công nghệ không dây như BlueTooth, cho phép truy xuất từ xa tới e-mail và trình duyệt web. Các điện thoại di động với nối kết Internet thuộc loại này. Tuy nhiên, nhiều PDAs hiện tại không cung cấp truy xuất không dây. Để tải dữ liệu xuống các thiết bị này, trước tiên người dùng tải dữ liệu xuống PC hay trạm và sau đó tải dữ liệu xuống PDA. Một số PDA cho phép dữ liệu chép trực tiếp từ một thiết bị này tới thiết bị khác dùng liên kết hồng ngoại. Nhìn chung, các giới hạn trong chức năng của PDA được cân bằng bởi những tiện dụng và linh động của chúng. Việc sử dụng chúng tiếp tục mở rộng khi các nối kết mạng trở nên sẵn dùng và các chọn lựa khác như máy ảnh và MP3 players, mở rộng tiện ích của chúng.

XI Tóm tắt

Hệ điều hành được phát triển hơn 45 năm qua với hai mục đích chính. Mục đích thứ nhất, hệ điều hành cố gắng lập thời biểu các hoạt động tính toán để đảm bảo năng lực thực hiện của hệ thống là tốt. Mục đích thứ hai, nó cung cấp một môi trường tiện dụng để phát triển và thực thi chương trình. Ban đầu, hệ thống máy tính được dùng từ một màn hình và bàn phím (thiết bị đầu cuối). Các phần mềm như bộ hợp ngữ (assembler), bộ nạp (loader), bộ liên kết (linkers) và các trình biên dịch (compiler) cải tiến sự tiện dụng của việc lập trình hệ thống nhưng cũng yêu cầu thời gian thiết lập đáng kể. Để giảm thời gian thiết lập, các phương tiện thuê người điều hành và các công việc tương tự được bó.

Các hệ thống xử lý theo lô cho phép sắp xếp công việc tự động bởi hệ điều hành và cải tiến rất nhiều việc tận dụng toàn bộ máy tính. Máy tính không còn phải chờ các thao tác của người dùng. Tuy nhiên, việc tận dụng CPU vẫn còn thấp vì tốc độ của thiết bị xuất nhập thấp hơn nhiều so với tốc độ của CPU. Thao tác ngoại vi (off-line operation) của các thiết bị chậm cung cấp một phương tiện sử dụng nhiều hệ thống bộ đọc tới băng từ (reader-to-tape) và băng từ tới máy in (tape-to-printer) cho một CPU. Để cải tiến toàn bộ năng lực thực hiện của hệ thống máy tính, người phát triển giới thiệu khái niệm đa chương để mà nhiều công việc có thể được giữ cùng lúc trong bộ nhớ tại cùng một thời điểm. CPU được chuyển qua lại giữa chúng để gia tăng việc tận dụng CPU và giảm toàn bộ thời gian được yêu cầu để thực thi các công việc.
Đa chương cũng cho phép chia sẻ thời gian. Hệ điều hành chia sẻ thời gian cho phép nhiều người dùng (từ một tới vài trăm) sử dụng hệ thống máy tính giao tiếp tại cùng một thời điểm.

PC là máy vi tính; chúng xem như nhỏ hơn và rẻ hơn hệ thống mainframe. Các hệ điều hành cho các máy tính này lợi hơn việc phát triển hệ điều hành cho máy tính mainframe trong nhiều cách. Tuy nhiên, vì mỗi cá nhân là người dùng duy nhất sử dụng máy tính nên việc tận dụng CPU không còn là mối quan tâm chủ yếu. Do đó, một vài quyết định thiết kế được thực hiện cho hệ điều hành cho máy mainframe có thể không phù hợp cho cả hệ thống nhỏ và lớn, hiện nay khi các PCs có thể được nối kết tới các máy tính khác và người dùng thông qua mạng và Web.
Các hệ song song có nhiều hơn một CPU trong giao tiếp gần; các CPU chia sẻ bus máy tính và đôi khi chia sẻ bộ nhớ và các thiết bị ngoại vi. Những hệ thống như thế có thể cung cấp thông lượng và khả năng tin cậy tăng. Các hệ thống phân tán cho phép chia sẻ tài nguyên trên những máy chủ được phân tán về mặt địa lý. Các hệ thống được nhóm cho phép nhiều máy thực hiện việc tính toán trên dữ liệu được chứa trên

Thuviendientu.org

thiết bị lưu trữ chia sẻ và để việc tính toán tiếp tục trong trường hợp lỗi của tập hợp con các thành viên nhóm.

Một hệ thời thực cứng thường được dùng như một thiết bị điều khiển trong một ứng dụng tận hiến. Một hệ điều hành thời thực cứng có ràng buộc hoàn toàn xác định và thời gian cố định. Xử lý phải được thực hiện trong các ràng buộc được xác định hoặc hệ thống sẽ bị lỗi. Các hệ thống thời thực mềm có ràng buộc thời gian ít nghiêm khắc hơn và không hỗ trợ thời biểu tới hạn.

Gần đây sự tác động của Internet và World Wide Web khuyến khích sự phát triển của các hệ điều hành hiện đại. Các hệ điều hành này chứa các trình duyệt Web, mạng và phần mềm truyền thông như là các đặc điểm tích hợp.

Chúng ta đã thể hiện tiến trình luận lý của sự phát hệ điều hành, được định hướng bởi sự bao gồm các đặc điểm trong phần cứng CPU được yêu cầu cho chức năng tiên tiến. Xu hướng này có thể được thấy ngày nay trong cuộc cách mạng của PC, với phần cứng ngày một rẻ hơn và đang được cải tiến đủ để cho phép cải tiến các đặc điểm.

Thuviendientu.org

CẤU TRÚC HỆ ĐIỀU HÀNH

I Mục đích

Sau khi học xong chương này, người học nắm được những kiến thức sau:

- Hiểu hệ điều hành từ ba khía cạnh: người dùng, người lập trình và người thiết kế
- Hiểu các dịch vụ mà hệ điều hànhcung cấp
- Biết các phương pháp được dùng để thiết kế kiến trúc hệ điều hành

II Giới thiệu

Hệ điều hành cung cấp môi trường cho các chương trình thực thi. Nội tại, các hệ điều hành rất khác biệt nhau về kiến trúc, chúng được tổ chức cùng với các dòng khác nhau. Thiết kế một hệ điều hành mới là một công việc quan trọng. Các mục đích của hệ thống phải được định nghĩa rõ ràng trước khi thiết kế bắt đầu. Kiểu hệ thống mong muốn là cơ sở cho việc chọn lựa giữa các giải thuật và chiến lược khác nhau.

Hệ điều hành có thể được nhìn từ nhiều lợi điểm khác nhau. Người này xem xét các dịch vụ mà hệ điều hành cung cấp. Người kia quan tâm đến giao diện mà hệ điều hành mang lại cho người dùng và người lập trình. Người khác lại phân rã hệ thống thành những thành phần và các mối quan hệ bên trong của chúng. Trong chương này chúng ta tìm hiểu cả ba khía cạnh của hệ điều hành, thể hiện ba quan điểm của người dùng, người lập trình và người thiết kế hệ điều hành. Chúng ta xem xét các dịch vụ mà hệ điều hành cung cấp, cách chúng được cung cấp và các phương pháp khác nhau được dùng cho việc thiết kế hệ điều hành.

III Các thành phần hệ thống

Chúng ta có thể tạo ra một hệ thống lớn và phức tạp như hệ điều hành chỉ khi phân chia hệ điều hành thành những phần nhỏ hơn. Mỗi phần nên là một thành phần được mô tả rõ ràng của hệ thống, với xuất, nhập và các chức năng được định nghĩa cẩn thận. Tuy nhiên, nhiều hệ thống hiện đại chia sẻ mục tiêu hỗ trợ các thành phần hệ thống được liệt kê sau đây:

III.1 Quản lý quá trình

Một chương trình không làm gì trừ khi các chỉ thị của nó được thực thi bởi một CPU. Một quá trình có thể được xem như một chương đang thực thi, nhưng định nghĩa của nó sẽ mở rộng khi chúng ta khám phá chi tiết hơn. Một chương trình người dùng được chia thời chẳng hạn như một trình biên dịch là một quá trình. Một chương trình xử lý văn bản đang được thực thi bởi một người dùng trên một PC cũng là một quá trình. Một tác vụ hệ thống, như gởi dữ liệu xuất ra máy in cũng được xem là một quá trình. Bây giờ chúng ta có thể xem xét một quá trình là một công việc hay chương trình chia thời, nhưng chúng ta sẽ nghiên cứu khái niệm này tổng quát hơn trong các chương sau.

Một quá trình cần các tài nguyên xác định-gồm thời gian CPU, bộ nhớ, tập tin, các thiết bị xuất/nhập-để hoàn thành tác vụ của nó. Các tài nguyên này được cấp cho

Thuviendientu.org

quá trình khi nó được tạo ra, hay được cấp phát tới nó khi nó đang chạy. Ngoài ra, các tài nguyên vật lý và luận lý khác nhau mà quá trình nhận được khi nó được tạo, dữ liệu khởi tạo khác nhau (hay nhập) có thể được truyền qua. Thí dụ, xem xét một quá trình có chức năng hiển thị trạng thái của một tập tin trên màn hình của một thiết bị đầu cuối. Quá trình này sẽ được cho dữ liệu vào là tên của tập tin, và sẽ thực thi các chỉ thị thích hợp và các lời gọi hệ thống đạt được và xuất trên thiết bị cuối thông tin mong muốn. Khi quá trình này kết thúc, hệ điều hành sẽ đòi lại bất cứ tài nguyên nào có thể dùng lại.

Chúng ta nhấn mạnh một chương trình chính nó không phải là một quá trình; một chương trình là một thực thể thụ động, như là nội dung của tập tin được lưu trên đĩa, trái lại một quá trình là một thực thể hoạt động, với một bộ đếm chương trình xác định chỉ thị kế tiếp để thực thi. Việc thực thi của quá trình phải là tuần tự. CPU thực thi một chỉ thị của quá trình sau khi đã thực thi một chỉ thực trước đó cho đến khi quá trình hoàn thành. Ngoài ra, tại bất kỳ thời điểm nào, tối đa một chỉ thị được thực thi cho quá trình. Do đó, mặc dù hai quá trình có thể được liên kết với cùng một quá trình, vì thế chúng được xem như hai chuỗi thực thi riêng. Thông thường có một chương trình sinh ra nhiều quá trình khi nó thực thi.

Một quá trình là một đơn vị công việc trong hệ thống. Một hệ thống chứa tập các quá trình, một vài quá trình này là các quá trình hệ điều hành (thực thi mã hệ thống) và các quá trình còn lại là các quá trình người dùng (chúng thực thi mã người dùng). Tất cả các quá trình này có tiềm năng thực thi đồng hành bằng cách đa hợp CPU giữa các quá trình.

Hệ điều hành có nhiệm vụ cho các hoạt động sau khi đề cập đến chức năng quản lý quá trình:

- Tạo và xoá các quá trình người dùng và hệ thống
- Tạm dừng và thực thi tiếp quá trình
- Cung cấp các cơ chế đồng bộ hoá quá trình
- Cung cấp các cơ chế giao tiếp quá trình
- Cung cấp cơ chế quản lý deadlock

III.2 Quản lý bộ nhớ chính

Bộ nhớ chính là trung tâm điều hành của một máy tính hiện đại. Bộ nhớ chính là một mảng các từ (words) hay bytes có kích thước lớn từ hàng trăm ngàn tới hàng tỉ. Mỗi từ hay byte có địa chỉ riêng. Bộ nhớ chính là một kho chứa dữ liệu có khả năng truy xuất nhanh được chia sẻ bởi CPU và các thiết bị xuất/nhập. Bộ xử lý trung tâm đọc các chỉ thị từ bộ nhớ trong chu kỳ lấy chỉ thị, nó đọc và viết dữ liệu từ bộ nhớ chính trong chu kỳ lấy dữ liệu. Bộ nhớ chính thường là thiết bị lưu trữ lớn mà CPU có thể định địa chỉ và truy xuất trực tiếp. Thí dụ, đối với CPU xử lý dữ liệu từ đĩa, dữ liệu trước tiên được chuyển tới bộ nhớ chính bởi lời gọi xuất/nhập được sinh ra bởi CPU. Tương tự, các chỉ thị phải ở trong bộ nhớ cho CPU thực thi chúng.

Đối với một chương trình được thực thi, nó phải được ánh xạ các địa chỉ và được nạp vào bộ nhớ. Khi chương trình thực thi, nó truy xuất các chỉ thị chương trình và dữ liệu từ bộ nhớ bằng cách tạo ra các địa chỉ tuyệt đối này. Cuối cùng, chương trình kết thúc, không gian bộ nhớ của nó được khai báo sẵn, và chương trình có thể được nạp và thực thi.

Để cải tiến việc sử dụng CPU và tốc độ đáp ứng của máy tính cho người dùng, chúng ta phải giữ nhiều chương trình vào bộ nhớ. Nhiều cơ chế quản lý bộ nhớ khác nhau được dùng và tính hiệu quả của các giải thuật phụ thuộc vào từng trường hợp cụ thể. Chọn một cơ chế quản lý bộ nhớ cho một hệ thống xác định phụ thuộc vào nhiều

Thuviendientu.org

yếu tố-đặc biệt trên thiết kế phần cứng của hệ thống. Mỗi giải thuật đòi hỏi sự hỗ trợ phần cứng của nó.

Hệ điều hành có nhiệm vụ cho các hoạt động sau khi đề cập tới việc quản lý bộ nhớ

- Giữ vết về phần nào của bộ nhớ hiện đang được dùng và quá trình nào đang dùng.
- Quyết định quá trình nào được nạp vào bộ nhớ khi không gian bộ nhớ trở nên sẵn dùng.
- Cấp phát và thu hồi không gian bộ nhớ khi được yêu cầu.

III.3 Quản lý tập tin

Quản lý tập tin là một trong những thành phần có thể nhìn thấy nhất của hệ điều hành. Máy tính có thể lưu thông tin trên nhiều loại phương tiện lưu trữ vật lý khác nhau. Băng từ, đĩa từ, đĩa quang là những phương tiện thông dụng nhất. Mỗi phương tiện này có đặc điểm và tổ chức riêng. Mỗi phương tiện được điều khiển bởi một thiết bị, như một ổ đĩa hay ổ băng từ. Các thuộc tính này bao gồm tốc độ truy xuất, dung lượng, tốc độ truyền dữ liệu và phương pháp truy xuất (tuần tự hay ngẫu nhiên).

Nhờ vào việc sử dụng thuận lợi hệ thống máy tính, hệ điều hành cung cấp tầm nhìn luận lý của việc lưu trữ thông tin đồng nhất. Hệ điều hành trừu tượng hoá các thuộc tính vật lý của các thiết bị lưu trữ để định nghĩa một đơn vị lưu trữ luận lý là tập tin. Hệ điều hành ánh xạ các tập tin trên các thiết bị lưu trữ vật lý, và truy xuất các tập tin này bằng các thiết bị lưu trữ.

Tập tin là tập hợp thông tin có quan hệ được định nghĩa bởi người tạo. Thông thường, các tập tin biểu diễn chương trình và dữ liệu. Các tập tin dữ liệu có thể là số, chữ cái, chữ số. Các tập tin có dạng bất kỳ (thí dụ, các tập tin văn bản) hay có thể được định dạng có cấu trúc (thí dụ, các trường cố định). Một tập tin chứa một chuỗi các bits, bytes, các dòng hay các mẫu tin mà ý nghĩa của nó được định nghĩa bởi người tạo. Khái niệm tập tin là một khái niệm cực kỳ thông dụng.

Hệ điều hành cài đặt một khái niệm trừu tượng của tập tin bằng cách quản lý phương tiện lưu trữ như đĩa, băng từ và các thiết bị điều khiển chúng. Các tập tin cũng thường được tổ chức trong các thư mục để dễ dàng sử dụng chúng. Cuối cùng, khi nhiều người dùng truy xuất tập tin, chúng ta muốn kiểm soát ai và trong cách gì (thí dụ: đọc, viết, chèn,..) các tập tin có thể được truy xuất.

Hệ điều hành có nhiệm vụ thực hiện các hoạt động trong việc quản lý hệ thống tập tin:

- Tạo và xoá tập tin
- Tạo và xoá thư mục
- Hỗ trợ các hàm nguyên thuỷ để thao tác tập tin và thư mục
- Ánh xạ các tập tin trên các thiết bị lưu trữ phụ
- Sao lưu dự phòng tập tin trên các phương tiện lưu trữ ổ định

III.4 Quản lý hệ thống xuất/nhập

Một trong những mục đích của hệ điều hành là che giấu sự khác biệt của các thiết bị phần cứng từ người dùng. Thí dụ, trong UNIX sự khác biệt của các thiết bị xuất/nhập bị che giấu từ phần chính của hệ điều hành bởi các hệ thống con xuất/nhập. Hệ thống con xuất/nhập chứa:

- Thành phần quản lý bộ nhớ chứa vùng đệm (buffering), lưu trữ (caching) và spooling (vùng chứa).

Thuviendientu.org

- Giao diện trình điều khiển thiết bị chung.
- Trình điều khiển cho các thiết bị xác định.

Chỉ trình điều khiển thiết bị biết sự khác biệt của các thiết bị xác định mà nó được gán

III.5 Quản lý việc lưu trữ phụ

Mục đích chính của một hệ thống máy tính là thực thi các chương trình. Những chương trình này với dữ liệu chúng truy xuất phải nằm trong bộ nhớ chính hay lưu trữ chính trong quá trình thực thi. Vì bộ nhớ chính quá nhỏ để lưu tất cả dữ liệu và chương trình và vì dữ liệu quản lý bị mất khi mất điện, hệ thống máy tính phải cung cấp việc lưu trữ phụ để lưu dự phòng bộ nhớ chính. Hầu hết các hệ thống máy tính hiện đại dùng đĩa như phương tiện lưu trữ trực tuyến cho cả chương trình và dữ liệu. Hầu hết các chương trình – gồm trình biên dịch, trình dịch hợp ngữ, thủ tục sắp xếp, trình soạn thảo và trình định dạng – được lưu trên đĩa cho tới khi được nạp vào trong bộ nhớ và sau đó dùng đĩa khi cả hai nguồn và đích của việc xử lý. Do đó, quản lý hợp lý việc lưu trữ đĩa có vai trò quan trọng đối với một hệ thống máy tính.

Hệ điều hành có nhiệm vụ thực hiện các hoạt động sau trong việc quản lý đĩa:

- Quản lý không gian trống
- Cấp phát lưu trữ
- Định thời đĩa

Vì lưu trữ phụ được dùng thường xuyên nên nó phải được dùng một cách hiệu quả. Tốc độ toàn bộ của các thao tác của máy tính có thể xoay quanh tốc độ hệ thống con đĩa và các giải thuật thao tác trên hệ thống con đó.

III.6 Mạng

Hệ phân tán là tập hợp các bộ xử lý, chúng không chia sẻ bộ nhớ, các thiết bị ngoại vi hay đồng hồ. Thay vào đó mỗi bộ xử lý có bộ nhớ, đồng hồ và các bộ xử lý giao tiếp với nhau thông qua các đường giao tiếp như bus tốc độ cao hay mạng. Các bộ xử lý trong hệ thống phân tán khác nhau về kích thước và chức năng. Chúng có thể chứa các bộ vi xử lý, trạm làm việc, máy vi tính và các hệ thống máy tính thông thường.

Các bộ xử lý trong hệ thống được nối với nhau thông qua mạng truyền thông có thể được cấu hình trong nhiều cách khác nhau. Mạng có thể được nối kết một phần hay toàn bộ. Thiết kế mạng truyền thông phải xem xét vạch đường thông điệp và các chiến lược nối kết, và các vấn đề cạnh tranh hay bảo mật.

Hệ thống phân tán tập hợp những hệ thống vật lý riêng rẻ, có thể có kiến trúc không đồng nhất thành một hệ thống chặt chẻ, cung cấp người dùng với truy xuất tới các tài nguyên khác nhau mà hệ thống duy trì. Truy xuất tới các tài nguyên chia sẻ cho phép tăng tốc độ tính toán, chức năng, khả năng sẵn dùng của dữ liệu, khả năng tin cậy. Hệ điều hành thường tổng quát hoá việc truy xuất mạng như một dạng truy xuất tập tin, với những chi tiết mạng được chứa trong trình điều khiển thiết bị của giao diện mạng. Các giao thức tạo một hệ thống phân tán có thể có một ảnh hưởng to lớn trên tiện ích và tính phổ biến của hệ thống đó. Sự đổi mới của World Wide Web đã tạo ra một phương pháp truy xuất mới cho thông tin chia sẻ. Nó đã cải tiến giao thức truyền tập tin (File Transfer Protocol-FTP) và hệ thống tập tin mạng (Network File System-NFS) đã có bằng cách xoá yêu cầu cho một người dùng đăng nhập trước khi người dùng đó được phép dùng tài nguyên ở xa. Định nghĩa một giao thức mới, giao thức truyền siêu văn bản (hypertext transfer protocol-http), dùng trong giao tiếp giữa một trình phục vụ web và trình duyệt web. Trình duyệt web chỉ cần gởi yêu cầu thông tin tới một trình

Thuviendientu.org

phục vụ web của máy ở xa, thông tin (văn bản, đồ hoạ, liên kết tới những thông tin khác) được trả về.

III.7 Hệ thống bảo vệ

Nếu một hệ thống máy tính có nhiều người dùng và cho phép thực thi đồng hành của nhiều quá trình, thì các quá trình khác nhau phải được bảo vệ từ các hoạt động của quá trình khác. Cho mục đích này, các cơ chế đảm bảo rằng các tập tin, phân đoạn bộ nhớ, CPU, và các tài nguyên khác có thể được điều hành chỉ bởi các quá trình có quyền phù hợp từ hệ điều hành.

Thí dụ, phần cứng định địa chỉ bộ nhớ đảm bảo rằng một quá trình có thể thực thi chỉ trong không gian địa chỉ của chính nó. Bộ định thời đảm bảo rằng không có quá trình nào có thể đạt được điều khiển của CPU mà cuối cùng không trả lại điều khiển. Các thanh ghi điều khiển thiết bị không thể truy xuất tới người dùng vì thế tính đúng đắn của các thiết bị ngoại vi khác nhau được bảo vệ.

Bảo vệ là một cơ chế để điều khiển truy xuất của các chương trình, quá trình hay người dùng tới tài nguyên được định nghĩa bởi một hệ thống máy tính. Cơ chế này phải cung cấp phương tiện để đặc tả các điều khiển được áp đặt và phương tiện cho việc ép buộc.

Bảo vệ có thể cải tiến khả năng tin cậy bằng cách phát hiện các lỗi tiềm tàng tại các giao diện giữa các hệ thống con thành phần. Phát hiện các lỗi giao diện sớm thường có thể ngăn chặn nguy cơ ảnh hưởng tới hệ thống con bởi một hệ thống con khác. Tài nguyên không được bảo vệ không thể ngăn chặn việc sử dụng bởi người dùng không có quyền. Hệ thống hướng bảo vệ (protection-oriented system) cung cấp một phương tiện để phân biệt giữa việc dùng có quyền và không có quyền.

III.8 Hệ thống thông dịch lệnh

Một trong những chương trình hệ thống quan trọng nhất đối với hệ điều hành là trình thông dịch lệnh. Nó là giao diện giữa người dùng và hệ điều hành. Một vài hệ điều hành chứa trình thông dịch lệnh trong nhân (kernel). Các hệ điều hành khác nhau như MS-DOS và UNIX xem trình thông dịch lệnh như một chương trình đặc biệt đang chạy khi một công việc được khởi tạo hay khi người dùng đăng nhập lần đầu tiên (trên các hệ thống chia thời).

Nhiều lệnh (commands) được cung cấp tới hệ điều hành bởi các lệnh điều khiển (control statements). Khi một công việc mới được bắt đầu trong hệ thống bó, hay khi một người dùng đăng nhập tới hệ thống chia thời, một chương trình đọc và thông dịch các câu lệnh điều khiển được thực thi tự động. Chương trình này còn được gọi trình thông dịch thẻ điều khiển (control-card interpreter) hay trình thông dịch dòng lệnh và thường được biết như **shell**. Chức năng của nó đơn giản là: lấy câu lệnh tiếp theo và thực thi nó.

Các hệ điều hành thường khác nhau trong vùng shell, với một trình thông dịch lệnh thân thiện với người dùng làm cho hệ thống có thể chấp nhập nhiều hơn đối với người dùng. Một dạng giao diện thân thiện người dùng là hệ thống trình đơn-cửa sổ trên cơ sở chuột (mouse-based window-and-menu system) được dùng trong Macintosh và Microsoft Windows. Chuột được di chuyển tới vị trí con trỏ chuột trên ảnh hay biểu tượng trên màn hình biểu diễn các chương trình, tập tin, và các hàm hệ thống. Phụ thuộc vào vị trí con trỏ chuột, nhấn một nút trên chuột có thể nạp một chương trình, chọn một tập tin hay thư mục hay kéo xuống một trình đơn chứa các câu lệnh. Các shell mạnh hơn, phức tạp hơn và khó học hơn được đánh giá cao bởi một số người dùng khác. Trong những shell này, các lệnh được đánh vào từ bàn phím

Thuviendientu.org

được hiển thị trên màn hình hay in ra thiết bị đầu cuối, với phím enter (hay return) chỉ rằng một lệnh hoàn thành và sẵn sàng được thực thi. Shell của MS-DOS và UNIX điều hành theo cách này.

Các câu lệnh giải quyết việc tạo và quản lý quá trình, quản lý xuất/nhập, quản lý việc lưu trữ phụ, quản lý bộ nhớ chính, truy xuất hệ thống tập tin, bảo vệ và mạng.

IV Các dịch vụ hệ điều hành

Hệ điều hành cung cấp một môi trường cho việc thực thi các chương trình. Nó cung cấp các dịch vụ xác định tới chương trình và tới người dùng của các chương trình đó. Dĩ nhiên, các dịch vụ được cung cấp khác nhau từ hệ điều hành này với hệ điều hành kia nhưng chúng có thể xác định các lớp chung. Các dịch vụ hệ điều hành được cung cấp sự tiện dụng cho người lập trình để thực hiện tác vụ lập trình dễ dàng.

- Thực thi chương trình: hệ thống phải có thể nạp chương trình vào bộ nhớ và chạy chương trình đó. Chương trình phải có thể kết thúc việc thực thi của nó bình thường hay không bình thường (hiển thị lỗi).
- Thao tác xuất/nhập: một chương trình đang chạy có thể yêu cầu xuất/nhập. Xuất/nhập này có thể liên quan tới tập tin hay thiết bị xuất/nhập. Đối với các thiết bị cụ thể, các chức năng đặc biệt có thể được mong muốn (như quay lại từ đầu một ổ băng từ, hay xoá màn hình). Đối với tính hiệu quả và tính bảo vệ, người dùng thường không thể điều khiển các thiết bị xuất/nhập trực tiếp. Do đó, hệ điều hành phải cung cấp một phương tiện để thực hiện xuất/nhập..
- Thao tác hệ thống tập tin: hệ thống tập tin có sự quan tâm đặc biệt. Các chương trình cần đọc từ và viết tới các tập tin. Chương trình cũng cần tạo và xoá tập tin bằng tên.
- Giao tiếp: trong nhiều trường hợp, một quá trình cần trao đổi thông tin với các quá trình khác. Giao tiếp như thế có thể xảy ra trong hai cách chính. Cách đầu tiên xảy ra giữa các quá trình được thực thi trên cùng máy tính; cách thứ hai xảy ra giữa hai quá trình đang được thực thi trên các máy tính khác nhau được kết nối với nhau bởi một mạng máy tính. Các giao tiếp có thể được thực hiện bằng bộ nhớ được chia sẻ, hay bằng kỹ thuật truyền thông điệp, trong đó các gói tin được di chuyển giữa các quá trình bởi hệ điều hành.
- Phát hiện lỗi: hệ điều hành liên tục yêu cầu nhận biết các lỗi có thể phát sinh. Các lỗi có thể xảy ra trong CPU và phần cứng bộ nhớ (như lỗi bộ nhớ hay lỗi về điện), trong các thiết bị xuất/nhập (như lỗi chẵn lẻ trên băng từ, lỗi nối kết mạng, hết giấy in) và trong chương trình người dùng (như tràn số học, cố gắng truy xuất một vị trí bộ nhớ không hợp lệ, dùng quá nhiều thời gian CPU). Đối với mỗi loại lỗi, hệ điều hành nên thực hiện một hoạt động hợp lý để đảm bảo tính toán đúng và không đổi.

Ngoài ra, một tập chức năng khác của hệ điều hành tồn tại không giúp người dùng, nhưng đảm bảo các điều hành hữu hiệu của chính hệ thống. Các hệ thống với nhiều người dùng có thể đạt tính hữu hiệu bằng cách chia sẻ tài nguyên máy tính giữa các người dùng.

- Cấp phát tài nguyên: khi nhiều người dùng đăng nhập vào hệ thống hay nhiều công việc đang chạy cùng lúc, tài nguyên phải được cấp tới mỗi người dùng. Nhiều loại tài nguyên khác nhau được quản lý bởi hệ điều hành. Một số tài nguyên (như chu kỳ CPU, bộ nhớ chính, lưu trữ tập tin) có mã cấp

Thuviendientu.org

phát đặt biệt, trái lại các tài nguyên khác (như thiết bị xuất/nhập) có mã yêu cầu và giải phóng thường hơn. Thí dụ, xác định cách tốt nhất để dùng CPU, hệ điều hành có các thủ tục định thời biểu CPU. Các thủ tục này xem xét tốc độ CPU, các công việc phải được thực thi, số thanh ghi sẵn dùng và các yếu tố khác. Cũng có các thủ tục cấp phát ổ băng từ để dùng cho một công việc. Một thủ tục như thế định vị ổ băng từ chưa được dùng và đánh dấu một bảng bên trong để ghi người dùng mới của ổ băng từ. Một thủ tục khác được dùng để xoá bảng đó. Các thủ tục này cũng có thể cấp phát các máy vẽ, modem, các thiết bị ngoại vi khác.

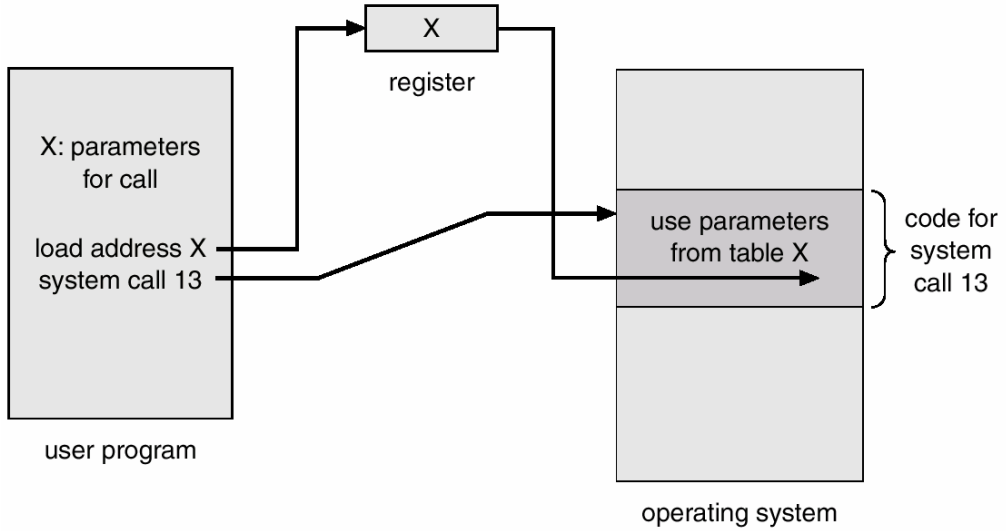
- Tính toán: chúng ta muốn giữ vết người dùng nào sử dụng bao nhiêu và loại tài nguyên máy tính nào. Giữ vết này có thể được dùng để tính toán (tính tiền người dùng) hay đơn giản thống kê sử dụng. Thống kê sử dụng có thể là công cụ có giá trị cho người nghiên cứu muốn cấu hình lại hệ thống để cải tiến các dịch vụ tính toán.
- Bảo vệ: người sở hữu thông tin được lưu trong hệ thống máy tính đa người dùng muốn điều khiển thông tin này. Khi nhiều quá trình riêng rẻ thực thi đồng hành, không thể cho một quá trình can thiệp tới các quá trình khác hay tới chính hệ điều hành. Bảo vệ đảm bảo rằng tất cả truy xuất tài nguyên của hệ thống được kiểm soát. An toàn hệ thống từ người dùng bên ngoài cũng là vấn đề quan trọng. An toàn bắt đầu với mỗi người dùng có quyền đối với hệ thống, thường bằng mật khẩu để được phép truy xuất tài nguyên. Mở rộng việc bảo vệ đối với các thiết bị xuất/nhập bên ngoài, bao gồm modem, card mạng từ những truy xuất không hợp lệ, và ghi lại các nối kết để phát hiện đột nhập vào hệ thống. Nếu hệ thống bảo vệ và bảo mật, những cảnh báo phải được thiết lập xuyên suốt.

V Lời gọi hệ thống

Lời gọi hệ thống cung cấp giao diện giữa một quá trình và hệ điều hành. Các lời gọi này thường sẵn dùng như các chỉ thị hợp ngữ và chúng thường được liệt kê trong những tài liệu hướng dẫn sử dụng được dùng bởi những người lập trình hợp ngữ. Những hệ thống xác định cho phép lời gọi hệ thống được thực hiện trực tiếp từ một chương trình ngôn ngữ cấp cao, trong đó các lời gọi thường tương tự lời gọi hàm hay thủ tục được định nghĩa trước. Chúng có thể tạo ra một lời gọi tới một chương trình con tại thời điểm thực thi cụ thể.

Lời gọi hệ thống xảy ra trong nhiều cách khác nhau, phụ thuộc vào máy tính đang dùng. Thường nhiều thông tin được yêu cầu hơn là đơn giản xác định lời gọi hệ thống mong muốn. Loại chính xác và lượng thông tin khác nhau dựa vào hệ điều hành và lời gọi cụ thể. Thí dụ, để nhập dữ liệu, chúng ta có thể cần xác định tập tin hay thiết bị dùng như nguồn nhập, địa chỉ và chiều dài vùng đệm bộ nhớ mà dữ liệu nhập sẽ được đọc vào. Dĩ nhiên, thiết bị hay tập tin và chiều dài có thể ẩn trong lời gọi.

Thuviendientu.org



Hình 0-1-Truyền tham số như bảng

Có ba phương pháp thông dụng để truyền tham số tới hệ điều hành. Phương pháp đơn giản nhất là truyền tham số trong các thanh ghi. Trong một vài trường hợp, các tham số thường lưu trữ trong một khối hay bảng trong bộ nhớ và địa chỉ của khối được truyền như một tham số trong thanh ghi (Hình II.1). Các tham số cũng có thể được thay thế, hay được đẩy vào trong ngăn xếp bởi chương trình, và được lấy ra khỏi ngăn xếp bởi hệ điều hành. Một vài hệ điều hành dùng phương pháp khối hay ngăn xếp vì các phương pháp này không giới hạn số lượng hay chiều dài của tham số đang được truyền.

VI Các chương trình hệ thống

Một khía cạnh khác của hệ thống hiện đại là tập hợp các chương trình hệ thống. Xem lại hình I.1, nó minh họa cấu trúc phân cấp máy tính luận lý. Tại cấp thấp nhất là phần cứng. Kế đến là hệ điều hành, sau đó các chương trình hệ thống và cuối cùng là các chương trình ứng dụng. Các chương trình hệ thống cung cấp môi trường thuận lợi cho việc phát triển và thực thi chương trình. Vài trong chúng là các giao diện người dùng đơn giản cho các lời gọi hệ thống; các hệ thống còn lại được xem xét phức tạp hơn. Chúng có thể được chia thành các loại sau:

- **Quản lý tập tin**: các chương trình tạo, xóa, chép, đổi tên, in, kết xuất, liệt kê, và các thao tác tập tin thư mục thông thường.
- **Thông tin trạng thái**: một vài chương trình đơn giản yêu cầu hệ thống ngày, giờ, lượng bộ nhớ hay đĩa sẵn dùng, số lượng người dùng, hay thông tin trạng thái tương tự. Sau đó, thông tin được định dạng và được in tới thiết bị đầu cuối hay thiết bị xuất khác hoặc tập tin.
- **Thay đổi tập tin**: nhiều trình soạn thảo văn bản có thể sẵn dùng để tạo và thay đổi nội dung của tập tin được lưu trên đĩa hay băng từ.
- **Hỗ trợ ngôn ngữ lập trình**: trình biên dịch, trình hợp ngữ và trình thông dịch cho các ngôn ngữ lập trình thông dụng (như C, C++, Java, Visual Basic và PERL) thường được cung cấp tới người dùng với hệ điều hành. Hiện nay, một vài chương trình này được cung cấp riêng và có giá cả riêng.

Thuviendientu.org

- **Nạp và thực thi chương trình**: một khi chương trình được tập hợp hay được biên dịch, nó phải được nạp vào bộ nhớ để được thực thi. Hệ thống có thể cung cấp bộ nạp tuyệt đối, bộ nạp có thể tái định vị, bộ soạn thảo liên kết và bộ nạp phủ lắp. Các hệ thống gỡ rối cho các ngôn ngữ cấp cao hay ngôn ngữ máy cũng được yêu cầu.
- **Giao tiếp**: các chương trình này cung cấp cơ chế tạo các nối kết ảo giữa các quá trình, người dùng, các hệ thống máy tính khác. Chúng cho phép người dùng gởi các thông điệp tới màn hình của người dùng khác, hiển thị các trang web, gởi thư điện tử, đăng nhập từ xa hay để chuyển các tập tin từ máy tính này tới máy tính khác.

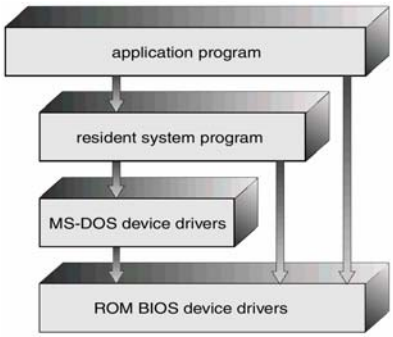
Nhiều hệ điều hành được cung cấp với các chương trình giải quyết các vấn đề giao tiếp thông thường hay thực hiện các thao tác phổ biến. Những chương trình như thế gồm các trình duyệt Web, bộ xử lý văn bản và bộ định dạng văn bản, hệ cơ sở dữ liệu, trình biên dịch, các gói phần mềm đồ họa và phân tích thống kê, trò chơi,..Những chương trình này được gọi là các tiện ích hệ thống hay chương trình ứng dụng.

Hầu hết người dùng nhìn hệ điều hành như các chương trình hệ thống hơn các lời gọi hệ thống thực sự. Nghĩ về việc sử dụng một PC. Khi máy tính của chúng ta chạy hệ điều hành Microsoft Windows, chúng ta có thể thấy một trình thông dịch dòng lệnh MS-DOS hay giao diện cửa sổ và trình đơn đồ họa. Cả hai sử dụng cùng một tập lời gọi hệ thống như lời gọi hệ thống trông rất khác và hoạt động trong các cách khác nhau. Do đó, tầm nhìn của chúng ta về thực chất có thể bị tách rời với cấu trúc hệ thống thực sự. Vì thế, thiết kế một giao diện tiện dụng và thân thiện không là một chức năng trực tiếp của hệ điều hành. Trong giáo trình này chúng ta sẽ tập trung các vấn đề cơ bản của việc cung cấp dịch vụ đầy đủ cho các chương trình người dùng. Từ quan điểm hệ điều hành, chúng ta không phân biệt sự khác nhau giữa các chương trình người dùng và các chương trình hệ thống.

VII Cấu trúc hệ thống

Một hệ thống lớn và phức tạp như một hệ điều hành hiện đại phải được xây dựng cẩn thận nếu nó thực hiện chức năng hợp lý và được hiệu chỉnh dễ dàng. Một phương pháp thông thường là chia tác vụ thành các thành phần nhỏ hơn là có một hệ thống nguyên khối. Mỗi modules này nên là một thành phần hoàn toàn xác định với nhập, xuất, chức năng được định nghĩa cẩn thận. Trong phần này chúng ta sẽ thảo luận về cách thức mà các thành phần được nối kết và trộn lẫn trong một nhân.

VII.1 Cấu trúc đơn giản



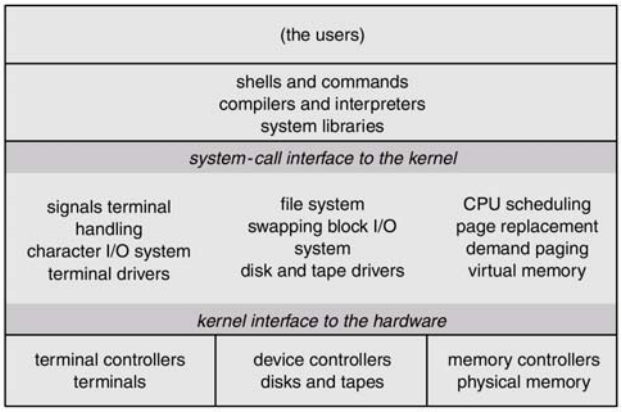
Hình 0-2-Cấu trúc phân tầng của MS-DOS

Thuviendientu.org

Nhiều hệ thống thương mại không có kiến trúc rõ ràng. Thường các hệ điều hành như thế được bắt đầu như các hệ thống nhỏ, đơn giản và có giới hạn. Sau đó chúng lớn lên ngoài giới hạn mã ban đầu của chúng.

MS-DOS là một thí dụ cho hệ thống dạng này. Ban đầu, nó được thiết kế và thực hiện bởi một vài người mà họ không tưởng rằng chúng sẽ trở nên quá phổ biến. Nó được viết để cung cấp các khả năng nhiều nhất trong không gian ít nhất (vì bị giới hạn bởi phần cứng mà nó đang chạy) vì nó không được phân chia thành các modules một cách cẩn thận. Hình II.2, hiển thị cấu trúc của nó.

UNIX là một hệ điều hành khác mà ban đầu nó bị giới hạn bởi chức năng phần cứng. Nó chứa hai phần có thể tách rời nhau: nhân và các chương trình hệ thống. Nhân lại được chia thành một loạt các giao diện và trình điều khiển thiết bị mà chúng được thêm vào và mở rộng qua nhiều năm khi UNIX được cải tiến. Chúng ta có thể hiển thị hệ điều hành UNIX truyền thống khi nó được phân tầng như hình II.3. Mọi thứ bên dưới giao diện lời gọi hệ thống và bên trên phần cứng vật lý là nhân. Nhân cung cấp hệ thống tập tin, bộ định thời CPU, quản lý bộ nhớ và các chức năng khác của hệ điều hành thông qua lời gọi hệ thống. Có rất nhiều chức năng được nối kết trong cấp thứ nhất. Điều này làm cho UNIX khó có thể nâng cấp khi những thay đổi trong phần một ảnh hưởng bất lợi cho những phần khác.



Hình 0-3 – Cấu trúc hệ thống của UNIX

Lời gọi hệ thống định nghĩa giao diện lập trình ứng dụng (API-Application Programming Interface) cho UNIX; tập hợp các chương trình hệ thống thường sẵn dùng định nghĩa giao diện người dùng. Người lập trình và giao diện người dùng định nghĩa ngữ cảnh mà nhân phải hỗ trợ.

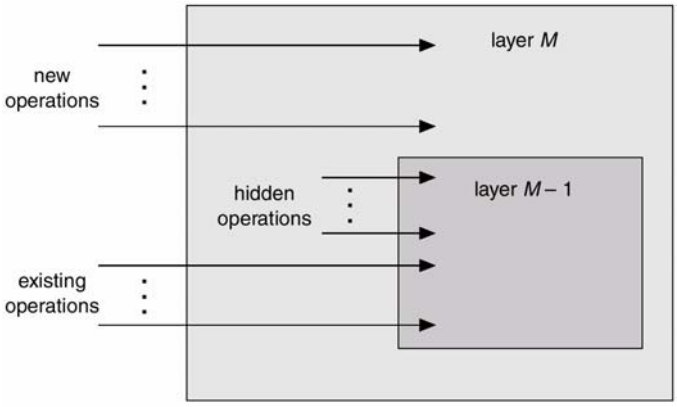
Những ấn bản mới của UNIX được thiết kế để dùng phần cứng tiên tiến hơn. Được cung cấp sự hỗ trợ phần cứng hợp lý, các hệ điều hành có thể được chia thành nhiều phần nhỏ hơn và phù hợp hơn là các hệ thống MS-DOS và UNIX ban đầu. Sau đó, các hệ điều hành có thể giữ lại việc điều khiển lớn hơn qua máy tính và qua các ứng dụng thực hiện việc sử dụng máy tính đó. Những người cài đặt thoải mái hơn trong việc thực hiện những thay đổi các hoạt động bên trong của hệ thống và trong việc tạo các hệ điều hành theo module. Dưới phương pháp từ trên-xuống (top-down), toàn bộ các chức năng và đặc điểm được xác định và được chia thành các thành phần. Sự phân chia này cho phép các người lập trình che giấu thông tin; do đó họ tự do cài đặt các thủ tục cấp thấp khi họ thấy thích hợp, được cung cấp các giao diện bên ngoài của các thủ tục không bị thay đổi do chính thủ tục đó thực hiện các tác vụ thông thường.

Thuviendientu.org

VII.2 Phương pháp phân tầng

Việc phân chia từng phần của một hệ thống có thể được thực hiện trong nhiều cách. Một trong những phương pháp này là thực hiện tiếp cận phân tầng. Trong tiếp cận này hệ điều hành được chia thành nhiều tầng (hay cấp), mỗi tầng được xây dựng trên đỉnh của tầng dưới nó. Tầng cuối cùng (tầng 0) là phần cứng; tầng cao nhất (tầng N) là giao diện người dùng.

Một tầng hệ điều hành là sự cài đặt của một đối tượng trừu tượng. Đối tượng trừu tượng này là sự bao gói dữ liệu và các điều hành có thể thao tác dữ liệu đó. Một tầng hệ điều hành điển hình –tầng M- được mô tả trong hình II.4. Nó chứa các cấu trúc dữ liệu và tập hợp các thủ tục có thể được gọi bởi các tầng cấp cao hơn. Sau đó, tầng M có thể gọi các thao tác trên tầng cấp thấp hơn.



Hình 0-4-Một tầng hệ điều hành

Lợi điểm chủ yếu của tiếp cận phân tầng là tính module. Các tầng được chọn dựa trên cơ sở tầng trên sử dụng chức năng (hay các điều hành) và các dịch vụ chỉ của tầng cấp dưới nó. Tiếp cận này đơn giản hóa việc gỡ rối và kiểm tra hệ thống. Tầng đầu tiên có thể được gỡ rối mà không có bất cứ sự quan tâm nào cho phần còn lại của hệ thống. Bởi vì theo định nghĩa, nó chỉ sử dụng phần cứng cơ bản để cài đặt các chức năng của nó. Một khi tầng đầu tiên được gỡ rối, chức năng sửa lỗi của nó có thể được đảm đương trong khi tầng thứ hai được gỡ rối, …Nếu một lỗi được tìm thấy trong khi gỡ rối cho một tầng xác định, lỗi phải được nằm trên tầng đó vì các tầng bên dưới đã được gỡ rối rồi. Do đó, thiết kế và cài đặt hệ thống được đơn giản hóa khi hệ thống được phân chia thành nhiều tầng.

Mỗi tầng được cài đặt chỉ với các thao tác được cung cấp bởi các tầng bên dưới. Một tầng không cần biết các thao tác được cài đặt như thế nào; nó chỉ cần biết các thao tác đó làm gì. Do đó, mỗi tầng che giấu sự tồn tại của cấu trúc dữ liệu, thao tác và phần cứng từ các tầng cấp cao hơn.

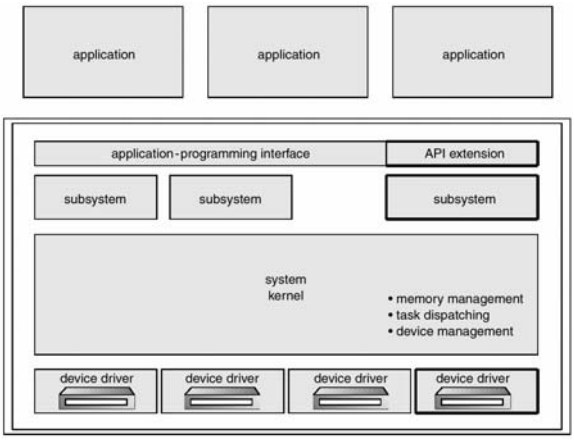
Khó khăn chính của tiếp cận phân tầng liên quan tới việc định nghĩa cẩn thận các tầng vì một tầng chỉ có thể sử dụng các tầng bên dưới nó. Thí dụ, trình điều khiển thiết bị cho không gian đĩa được dùng bởi các giải thuật bộ nhớ ảo phải nằm ở tại cấp thấp hơn trình điều khiển thiết bị của các thủ tục quản lý bộ nhớ vì quản lý bộ nhớ yêu cầu khả năng sử dụng không gian đĩa.

Các yêu cầu có thể không thật sự rõ ràng. Thường thì các trình điều khiển lưu trữ dự phòng nằm trên bộ định thời CPU vì trình điều khiển cần phải chờ nhập/xuất và CPU có thể được định thời lại trong thời gian này. Tuy nhiên, trên hệ thống lớn, bộ

Thuviendientu.org

định thời có thể có nhiều thông tin hơn về tất cả quá trình đang hoạt động hơn là có thể đặt vừa trong bộ nhớ. Do đó, thông tin này có thể cần được hoán vị vào và ra bộ nhớ, yêu cầu thủ tục trình điều khiển lưu trữ dự phòng nằm bên dưới bộ định thời CPU.

Vấn đề cuối cùng với các cài đặt phân tầng là chúng có khuynh hướng ít hiệu quả hơn các loại khác. Thí dụ, khi chương trình người dùng thực thi thao tác nhập/xuất, nó thực thi một lời gọi hệ thống. Lời gọi hệ thống này được bẫy (trapped) tới tầng nhập/xuất, nó yêu cầu tầng quản lý bộ nhớ, sau đó gọi tầng định thời CPU, sau đó được truyền tới phần cứng. Tại mỗi tầng, các tham số có thể được hiệu chỉnh, dữ liệu có thể được truyền,…Mỗi tầng thêm chi phí cho lời gọi hệ thống; kết quả thực sự là lời gọi hệ thống mất thời gian lâu hơn khi chúng thực hiện trên hệ thống không phân tầng.



Hình 0-5 Cấu trúc phân tầng của OS/2

Những giới hạn này gây một phản ứng nhỏ chống lại việc phân tầng trong những năm gần đây. Rất ít các tầng với nhiều chức năng được thiết kế, cung cấp nhiều lợi điểm của mã được module trong khi tránh những vấn đề khó khăn của định nghĩa và giao tiếp tầng. Thí dụ, OS/2 bổ sung thêm tính năng đa tác vụ và điều hành hai chế độ cùng một số đặc điểm mới. Vì tính phức tạp được bổ sung và phần cứng mạnh hơn mà OS/2 được thiết kế, hệ thống được cài đặt trong dạng phân tầng.

VII.3 Vi nhân (Microkernels)

Khi hệ điều hành UNIX được mở rộng, nhân trở nên lớn và khó quản lý. Vào giữa những năm 1980, các nhà nghiên cứu tại đại học Carnegie Mellon phát triển một hệ điều hành được gọi là Match mà module hóa nhân dùng ***tiếp cận vi nhân*** (micro kernel). Phương pháp này định kiến trúc của hệ điều hành bằng xóa tất cả thành phần không quan trọng từ nhân và cài chúng như các chương trình cấp người dùng và hệ thống. Kết quả này làm cho nhân nhỏ hơn. Có rất ít sự nhất trí liên quan đến việc quyết định dịch vụ nào nên để lại trong nhân và dịch vụ nào nên được cài đặt trong không gian người dùng. Tuy nhiên, thường thì các vi nhân điển hình cung cấp quá trình và quản lý bộ nhớ tối thiểu ngoài phương tiện giao tiếp.

Chức năng chính của vi nhân là cung cấp tiện nghi giao tiếp giữa chương trình khách hàng và các dịch vụ khác mà chúng đang chạy trong không gian người dùng. Giao tiếp được cung cấp bằng truyền thông điệp. Thí dụ, nếu chương trình khách hàng muốn truy xuất một tập tin, nó phải giao tiếp với trình phục vụ tập tin (file server).

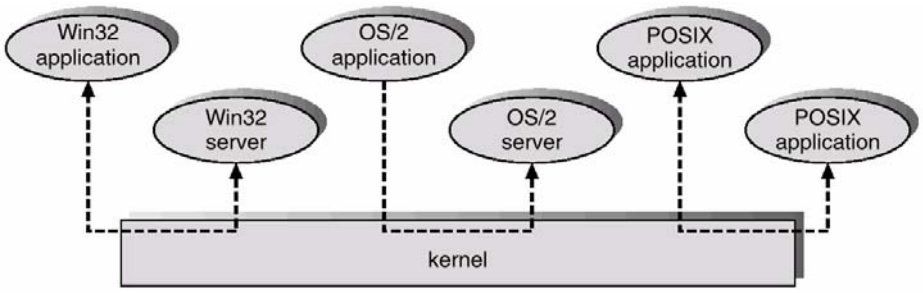
Thuviendientu.org

Chương trình người dùng và dịch vụ không bao giờ giao tiếp trực tiếp. Đúng hơn là chúng giao tiếp gián tiếp bằng cách truyền thông điệp với vi nhân.

Thuận lợi của tiếp cận vi nhân là dễ dàng mở rộng hệ điều hành. Tất cả dịch vụ mới được thêm tới không gian người dùng và do đó không yêu cầu phải hiệu chỉnh nhân. Kết quả là hệ điều hành dễ dàng hơn để chuyển đổi từ thiết kế phần cứng này sang thiết kế phần cứng khác. Vi nhân cũng cung cấp khả năng an toàn và tin cậy hơn vì hầu hết các dịch vụ đang chạy như người dùng –hơn là nhân- các quá trình. Nếu một dịch vụ bị lỗi, phần còn lại của hệ điều hành vẫn không bị ảnh hưởng.
Một số hệ điều hành hiện đại dùng tiếp cận vi nhân. Tru64 UNIX (Digital UNIX trước đây) cung cấp giao diện UNIX tới người dùng, nhưng nó được cài đặt với nhân Mach. Nhân Mach ánh xạ các lời gọi hệ thống vào các thông điệp tới các dịch vụ cấp người dùng tương ứng. Hệ điều hành Apple MacOS Server được dựa trên cơ sở nhân Mach.

QNX là hệ điều hành thời thực cũng dựa trên cơ sở thiết kế vi nhân. Vi nhân QNX cung cấp các dịch vụ cho việc truyền thông điệp và định thời quá trình. Nó cũng quản lý giao tiếp mạng cấp thấp và các ngắt phần cứng. Tất cả dịch vụ khác trong QNX được cung cấp bởi các quá trình chuẩn chạy bên ngoài nhân trong chế độ người dùng.

Windows NT dùng một cấu trúc tổng hợp. Windows NT được thiết kế để chạy các ứng dụng khác nhau, gồm Win32 (ứng dụng thuần Windows), OS/2, và POSIX (Portable Operating System Interface for uniX). Nó cung cấp một server chạy trong không gian người dùng cho mỗi loại ứng dụng. Các chương trình khách hàng cho mỗi loại ứng dụng chạy trong không gian người dùng. Nhân điều phối việc truyền thông điệp giữa các ứng dụng khách hàng và server ứng dụng. Cấu trúc client-server của Windows NT được mô tả trong hình II.6



Hình 0-6 – Cấu trúc client-server của Windows NT

VIII Máy ảo

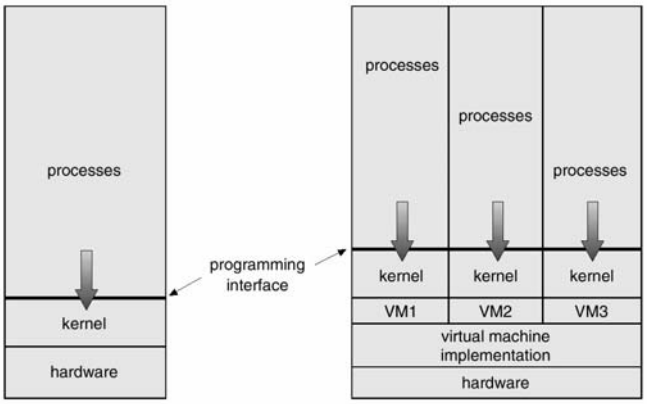
Về mặt khái niệm, một hệ thống máy tính được cấu thành từ các tầng. Phần cứng là cấp thấp nhất trong tất cả hệ thống như thế. Nhân chạy tại cấp kế tiếp dùng các chỉ thị phần cứng để tạo một tập lời gọi hệ thống cho việc sử dụng các tầng bên ngoài. Do đó, các chương trình hệ thống trên nhân có thể dùng các lời gọi hệ thống hay các chỉ thị phần cứng. Trong nhiều trường hợp, các chương trình này không có sự khác biệt giữa hai cách thực hiện. Do đó, mặc dù chúng được truy xuất khác nhau, nhưng cả hai cung cấp chức năng mà chương trình có thể dùng để tạo thậm chí nhiều chức năng tiên tiến hơn. Sau đó, các chương trình hệ thống xem phần cứng và các lời gọi hệ thống như chúng đang ở cùng một cấp.

Thuviendientu.org

Một vài hệ thống thực hiện cơ chế này một cách chi tiết hơn bằng cách cho phép các chương trình hệ thống được gọi dễ dàng bởi các chương trình ứng dụng. Trước đó, mặc dù các chương trình hệ thống ở tại cấp cao hơn các thủ tục khác, nhưng các chương trình ứng dụng có thể hiển thị mọi thứ dưới chúng trong cấu trúc phân cấp như là một phần của chính máy đó. Tiếp cận phân tầng này được đưa đến một kết luận luận lý trong khái niệm **máy ảo** (virtual machine). Một hệ điều hành máy ảo cho các hệ thống IBM là một thí dụ điển hình nhất về khái niệm máy ảo vì IBM tiên phong thực hiện trong lĩnh vực này.

Bằng cách sử dụng bộ định thời CPU và kỹ thuật bộ nhớ ảo, một hệ điều hành có thể tạo một hình ảnh mà một quá trình có bộ xử lý của chính nó với bộ nhớ (ảo) của chính nó. Dĩ nhiên, thường thì một quá trình có các đặc điểm khác nhau, như các lời gọi hệ thống và hệ thống tập tin, mà không được cung cấp bởi phần cứng trơ. Thêm vào đó, tiếp cận máy ảo không cung cấp bất kỳ chức năng bổ sung nào; nhưng đúng hơn là cung cấp một giao diện giống hệt như phần cứng trơ ở bên dưới. Mỗi quá trình được cung cấp với một bản sao (ảo) của máy tính bên dưới (Hình II.7).

Một khó khăn chính với tiếp cận máy ảo liên quan đến hệ thống đĩa. Giả sử rằng máy vật lý có ba ổ đĩa nhưng muốn hỗ trợ bảy máy ảo. Rõ ràng, nó không thể cấp phát một ổ đĩa tới mỗi máy ảo. Nhớ rằng chính phần mềm máy ảo sẽ cần không gian đĩa liên tục để cung cấp bộ nhớ ảo. Giải pháp này cung cấp đĩa ảo, mà nó đúng trong tất cả khía cạnh ngoại trừ kích thước-được thuật ngữ hóa ***đĩa nhỏ*** (minidisks) trong hệ điều hành máy ảo của IBM. Hệ thống cài đặt nhiều đĩa nhỏ bằng cách cấp phát nhiều rảnh ghi trên đĩa vật lý như là các đĩa nhỏ khi cần. Hiển nhiên, tổng kích thước của tất cả đĩa nhỏ là nhỏ hơn kích thước của không gian đĩa vật lý sẵn có.



Hình 0-7-Các mô hình hệ thống. (a) Máy không ảo. (b) máy ảo

Do đó, người dùng được cho máy ảo của chính họ. Sau đó, họ có thể chạy bất kỳ hệ điều hành hay gói phần mềm nào sẵn dùng trên phần cứng bên dưới. Đối với hệ thống IBM VM, một người dùng thường chạy CMS-một hệ điều hành giao tiếp đơn người dùng. Phần mềm máy ảo được quan tâm với đa máy ảo đa chương trên một máy vật lý nhưng không cần xem xét bất cứ phần mềm hỗ trợ người dùng. Việc sắp xếp này có thể cung cấp một sự phân chia hữu ích thành hai phần nhỏ hơn của vấn đề thiết kế một hệ thống giao tiếp đa người dùng.

Thuviendientu.org

VIII.1 Cài đặt

Mặc dù khái niệm máy ảo là hữu ích nhưng rất khó cài đặt. Nhiều công việc được yêu cầu cung cấp một bản sao chính xác của máy bên dưới. Máy bên dưới có hai chế độ: chế độ người dùng và chế độ kiểm soát. Phần mềm máy ảo có thể chạy trong chế độ kiểm soát vì nó là hệ điều hành. Chính máy ảo có thể thực thi chỉ trong chế độ người dùng. Tuy nhiên, chỉ khi máy vật lý có hai chế độ thì nó mới là máy ảo. Do đó, chúng ta phải có một chế độ người dùng ảo và một chế độ kiểm soát ảo. Cả hai đều chạy trong chế độ người dùng vật lý. Các hoạt động đó gây ra sự chuyển từ chế độ người dùng tới chế độ kiểm soát trên một máy thật (như lời gọi hệ thống hay cố gắng thực thi một chỉ thị được cấp quyền) cũng phải gây ra sự chuyển đổi từ chế độ người dùng ảo tới chế độ kiểm soát ảo trên một máy ảo.

VIII.2 Lợi điểm

Có hai lợi điểm chính trong việc sử dụng máy ảo. Thứ nhất, bằng cách bảo vệ hoàn toàn các tài nguyên hệ thống, máy ảo cung cấp mức độ bảo mật cao. Thứ hai, máy ảo cho phép phát triển hệ thống được thực hiện mà không cần phá vỡ hoạt động hệ thống thông thường.

Mỗi máy ảo hoàn toàn bị cô lập từ các máy ảo khác, vì thế chúng ta không gặp phải bất kỳ vấn đề bảo mật nào như tài nguyên hệ thống khác hoàn toàn được bảo vệ. Thí dụ, các ứng dụng không được tin cậy được tải về từ Internet có thể được chạy trong một máy ảo riêng. Một bất lợi của môi trường này là không có sự chia sẻ tài nguyên trực tiếp. Hai tiếp cận cung cấp sự chia sẻ được cài đặt. Thứ nhất, có thể chia sẻ một đĩa nhỏ. Cơ chế này được làm mẫu sau một đĩa được chia sẻ vật lý. Thứ hai, có thể định nghĩa một mạng của các máy ảo, mỗi máy ảo có thể gửi thông tin qua các mạng giao tiếp này nhưng nó được cài đặt bằng phần mềm.

Những hệ thống máy ảo như thế là một phương tiện truyền thông hữu hiệu cho việc nghiên cứu và phát triển hệ điều hành. Thông thường, thay đổi một hệ điều hành là một tác vụ khó. Vì các hệ điều hành là các chương trình lớn và phức tạp, sự thay đổi trên một phần này có thể gây một lỗi khó hiểu trong những phần khác. Sức mạnh của hệ điều hành làm cho trường hợp này là cực kỳ nguy hiểm. Vì hệ điều hành thực thi trong chế độ kiểm soát, một thay đổi sai trong một con trỏ có thể gây lỗi và có thể phá hủy toàn hệ thống tập tin. Do đó, cần phải kiểm tra tất cả thay đổi của hệ điều hành một cách cẩn thận.

Tuy nhiên, hệ điều hành chạy trên máy và điều khiển hoàn toàn máy đó. Do đó, hệ thống hiện hành phải bị dừng và ngừng việc sử dụng trong khi những thay đổi được thực hiện và kiểm tra. Thời điểm này thường được gọi là thời gian phát triển hệ thống. Vì nó làm cho hệ thống không sẵn dùng đối với người sử dụng nên thời gian phát triển hệ thống thường được lập thời biểu vào buổi tối hay cuối tuần, khi tải hệ thống thấp.

Một hệ thống máy ảo có thể loại trừ nhiều vấn đề này. Người lập trình hệ thống được cung cấp chính máy ảo của họ, và phát triển hệ thống được thực hiện trên máy ảo thay vì trên máy vật lý thật sự. Một hệ điều hành thông thường ít khi bị phá vỡ vì phát triển hệ thống. Mặc dù những thuận lợi này, nhưng rất ít cải tiến trên kỹ thuật này được thực hiện gần đây.

Thuviendientu.org

IX Tóm tắt

Hệ điều hành cung cấp một số dịch vụ. Tại cấp thấp nhất, lời gọi hệ thống cho phép một chương trình đang chạy thực hiện yêu cầu trực tiếp từ hệ thống. Tại cấp cao hơn, trình thông dịch cung cấp cơ chế cho người dùng đưa ra yêu cầu mà không viết chương trình. Các lệnh có thể xuất phát từ tập tin trong suốt thời gian thực thi theo chế độ xử lý theo lô, hay trực tiếp từ bàn phím trong chế độ tương tác hay chia sẻ thời gian. Các chương trình hệ thống được cung cấp để thoả mãn nhiều yêu cầu thông thường của người dùng.

Các loại yêu cầu khác nhau dựa theo cấp yêu cầu. Cấp gọi hệ thống phải cung cấp các chức năng cơ bản, như điều khiển quá trình, quản lý tập tin và thiết bị. Các yêu cầu cấp cao hơn được thoả mãn bởi trình thông dịch lệnh và chương trình hệ thống được dịch vào một chuỗi các lời gọi hệ thống. Các dịch vụ hệ thống có thể được phân cấp thành nhiều loại: điều khiển chương trình, yêu cầu trạng thái, yêu cầu nhập/xuất. Lỗi chương trình được xem xét các yêu cầu ẩn cho dịch vụ.

Một khi dịch vụ hệ thống được định nghĩa, cấu trúc của hệ điều hành được phát triển. Các bảng khác nhau cần được ghi thông tin định nghĩa trạng thái của hệ thống máy tính và trạng thái của công việc hệ thống.

Thiết kế một hệ điều hành mới là công việc rất quan trọng. Thiết kế hệ thống như thứ tự của các tầng hay sử dụng vi nhân được xem là một kỹ thuật tốt. Khái niệm máy ảo thực hiện tiếp cận phân tầng và xem nhân của hệ điều hành và phần cứng như là phần cứng của nó. Các hệ điều hành khác có thể được nạp trên đỉnh của máy ảo.

Thuviendientu.org

QUÁ TRÌNH

I Mục đích

Sau khi học xong chương này, người học nắm được những kiến thức sau:

- Hiểu các khái niệm về quá trình
- Hiểu cách lập thời biểu quá trình
- Biết các thao tác trên quá trình
- Hiểu cách giao tiếp liên quá trình

II Giới thiệu

Những hệ thống máy tính ban đầu cho phép chỉ một chương trình được thực thi tại một thời điểm. Chương trình này có toàn quyền điều khiển hệ thống và có truy xuất tới tất cả tài nguyên của hệ thống. Những hệ thống máy tính hiện nay cho phép nhiều chương trình được nạp vào bộ nhớ và được thực thi đồng hành. Sự phát triển này yêu cầu sự điều khiển mạnh mẽ hơn và phân chia nhiều hơn giữa các quá trình. Yêu cầu này dẫn đến khái niệm quá trình, một chương trình đang thực thi. Quá trình là một đơn vị công việc trong một hệ điều hành chia thời hiện đại.

Một hệ điều hành phức tạp hơn được mong đợi nhiều hơn trong việc thực hiện các hành vi của người dùng. Mặc dù quan tâm chủ yếu của hệ điều hành là thực thi chương trình người dùng, nhưng nó cũng quan tâm đến các tác vụ khác nhau bên ngoài nhân. Do đó, một hệ thống chứa tập hợp các quá trình: quá trình hệ điều hành thực thi mã hệ thống, quá trình người dùng thực thi mã người dùng. Tất cả quá trình này có tiềm năng thực thi đồng hành, với một CPU (hay nhiều CPU) được đa hợp giữa chúng. Bằng cách chuyển đổi CPU giữa các quá trình, hệ điều hành có thể làm cho máy tính hoạt động với năng suất cao hơn.

III Khái niệm quá trình

Một vấn đề cần thảo luận là cái gì được gọi trong tất cả hoạt động của CPU? Một hệ thống bó thực thi công việc, trái lại một hệ thống chia thời thực thi chương trình người dùng hay tác vụ. Thậm chí trên hệ thống đơn người dùng như Microsoft Windows và Macintosh OS, một người dùng có thể chạy nhiều chương trình tại một thời điểm: bộ xử lý văn bản, trình duyệt web, e-mail. Thậm chí nếu người dùng có thể thực thi chỉ một quá trình tại một thời điểm, thì một hệ điều hành cần hỗ trợ những hoạt động được lập trình bên trong, như quản lý bộ nhớ. Trong nhiều khía cạnh, tất cả hoạt động là tương tự vì thế chúng ta gọi tất cả chúng là quá trình.

Thuviendientu.org

III.1 Quá trình

Thật vậy, một quá trình là một chương trình đang thực thi. Một quá trình không chỉ là mà chương trình, nó còn bao gồm hoạt động hiện hành như được hiện diện bởi giá trị của bộ đếm chương trình và nội dung các thanh ghi của bộ xử lý. Ngoài ra, một quá trình thường chứa ngăn xếp quá trình, chứa dữ liệu tạm thời (như các tham số phương thức, các địa chỉ trả về, các biến cục bộ) và phần dữ liệu chứa các biến toàn cục.

Chúng ta nhấn mạnh rằng, một chương trình không phải là một quá trình; một chương trình là một thực thể thụ động, như nội dung của các tập tin được lưu trên đĩa, trái lại một quá trình là một thực thể chủ động, với một bộ đếm chương trình xác định chỉ thị lệnh tiếp theo sẽ thực thi và tập hợp tài nguyên có liên quan.

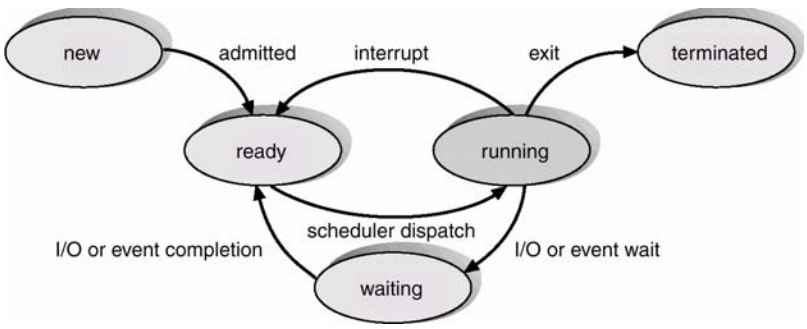
Mặc dù hai quá trình có thể được liên kết với cùng chương trình nhưng chúng được chứa hai thứ tự thực thi riêng rẻ. Thí dụ, nhiều người dùng có thể đang chạy các bản sao của chương trình gởi nhận thư, hay cùng người dùng có thể nạp lên nhiều bản sao của một chương trình soạn thảo văn bản. Mỗi bản sao của chúng là một quá trình riêng và mặc dù các phần văn bản là giống nhau, các phần dữ liệu khác nhau. Ngoài ra, một quá trình có thể tạo ra nhiều quá trình khi nó thực thi.

III.2 Trạng thái quá trình

Khi một quá trình thực thi, nó thay đổi trạng thái. Trạng thái của quá trình được định nghĩa bởi các hoạt động hiện hành của quá trình đó. Mỗi quá trình có thể ở một trong những trạng thái sau:

- Mới (new): quá trình đang được tạo ra
- Đang chạy (running): các chỉ thị đang được thực thi
- Chờ (waiting): quá trình đang chờ sự kiện xảy ra (như hoàn thành việc nhập/xuất hay nhận tín hiệu)
- Sẵn sàng (ready): quá trình đang chờ được gán tới một bộ xử lý.
- Kết thúc (terminated): quá trình hoàn thành việc thực thi
-

Các tên trạng thái này là bất kỳ, và chúng khác nhau ở các hệ điều hành khác nhau. Tuy nhiên, các trạng thái mà chúng hiện diện được tìm thấy trên tất cả hệ thống. Các hệ điều hành xác định mô tả trạng thái quá trình. Chỉ một quá trình có thể đang chạy tức thì trên bất kỳ bộ xử lý nào mặc dù nhiều quá trình có thể ở trạng thái sẵn sàng và chờ.

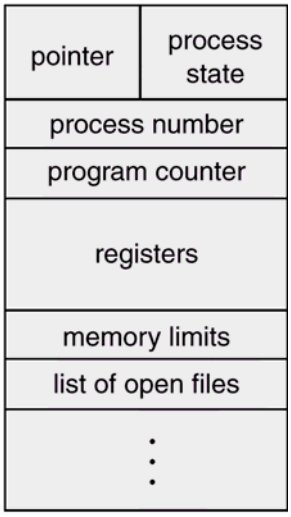


Hình 0-1-Lưu đồ trạng thái quá trình

Thuviendientu.org

III.3 Khối điều khiển quá trình

Mỗi quá trình được hiện diện trong hệ điều hành bởi một **khối điều khiển quá trình** (Process Control Block-PCB) – cũng được gọi khối điều khiển tác vụ. Một PCB được hiển thị trong hình III-2. Nó chứa nhiều phần thông tin được gắn liền với một quá trình xác định, gồm:

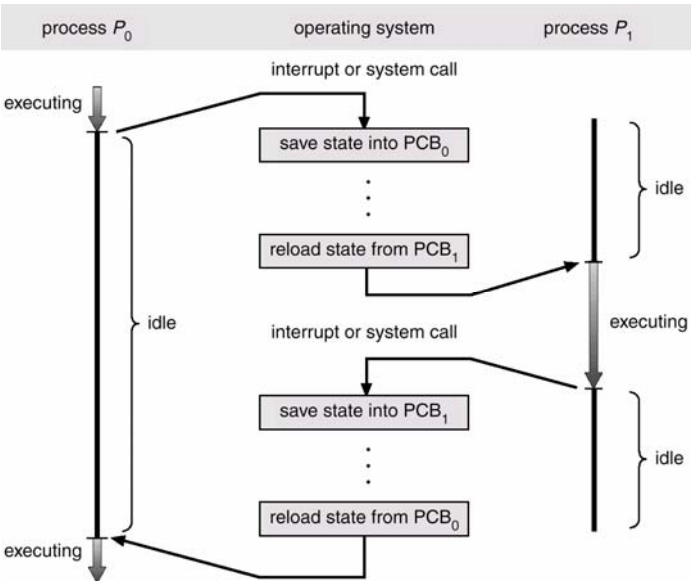


Hình 0-2-Khối điều khiển quá trình

- **Trạng thái quá trình** (process state): trạng thái có thể là mới, sẵn sàng, đang chạy, chờ đợi, kết thúc, …
- **Bộ đếm chương trình** (program counter): bộ đếm hiển thị địa chỉ của chỉ thị kế tiếp được thực thi cho quá trình này.
- **Các thanh ghi** (registers) CPU: các thanh ghi khác nhau về số lượng và loại, phụ thuộc vào kiến trúc máy tính. Chúng gồm các bộ tổng (accumulators), các thanh ghi chỉ mục, các con trỏ ngăn xếp, và các thanh ghi đa năng (general-purpose registers), cùng với thông tin mã điều kiện (condition-code information). Cùng với bộ đếm chương trình, thông tin trạng thái này phải được lưu khi một ngắt xảy ra, cho phép quá trình được tiếp tục một cách phù hợp sau đó (Hình III.3).
- **Thông tin lập thời biểu CPU** (CPU-scheduling information): thông tin gồm độ ưu tiên của quá trình, các con trỏ chỉ tới các hàng đợi lập thời biểu, và bất kỳ tham số lập thời biểu khác.
- **Thông tin quản lý bộ nhớ** (Memory-management information): thông tin này có thể gồm những thông tin như giá trị của các thanh ghi nền và thanh ghi giới hạn, các bảng trang hay các bảng phân đoạn, phụ thuộc hệ thống bộ nhớ được dùng bởi hệ điều hành.
- **Thông tin tính toán** (accounting information): thông tin này gồm lượng CPU và thời gian thực được dùng, công việc hay số quá trình,…
- **Thông tin trạng thái nhập/xuất** (I/O status information): thông tin này gồm danh sách của thiết bị nhập/xuất được cấp phát quá trình này, một danh sách các tập tin đang mở,..

PCB đơn giản phục vụ như kho chứa cho bất cứ thông tin khác nhau từ quá trình này tới quá trình khác.

Thuviendientu.org



Hình 0-3-Lưu đồ hiển thị việc chuyển CPU từ quá trình này tới quá trình khác

III.4 Luồng

Mô hình quá trình vừa được thảo luận ngụ ý rằng một quá trình là một chương trình thực hiện một luồng đơn thực thi. Thí dụ, nếu một quá trình đang chạy một chương trình xử lý văn bản, một luồng đơn của chỉ thị đang được thực thi. Đây là một luồng điều khiển đơn cho phép quá trình thực thi chỉ một tác vụ tại một thời điểm. Thí dụ, người dùng không thể cùng lúc nhập các ký tự và chạy bộ kiểm tra chính tả trong cùng một quá trình. Nhiều hệ điều hành hiện đại mở rộng khái niệm quá trình để cho phép một quá trình có nhiều luồng thực thi. Do đó, chúng cho phép thực hiện nhiều hơn một tác vụ tại một thời điểm.

IV Lập thời biểu quá trình

Mục tiêu của việc đa chương là có vài quá trình chạy tại tất cả thời điểm để tận dụng tối đa việc sử dụng CPU. Mục tiêu của chia thời là chuyển CPU giữa các quá trình thường xuyên để người dùng có thể giao tiếp với mỗi chương trình trong khi đang chạy. Một hệ thống đơn xử lý chỉ có thể chạy một quá trình. Nếu nhiều hơn một quá trình tồn tại, các quá trình còn lại phải chờ cho tới khi CPU rảnh và có thể lập thời biểu lại.

IV.1 Hàng đợi lập thời biểu

Khi các quá trình được đưa vào hệ thống, chúng được đặt vào hàng đợi công việc. Hàng đợi chứa tất cả quá trình trong hệ thống. Các quá trình đang nằm trong bộ nhớ chính sẵn sàng và chờ để thực thi được giữ trên một danh sách được gọi là hàng đợi sẵn sàng. Hàng đợi này thường được lưu như một danh sách liên kết. Đầu của hàng đợi sẵn sàng chứa hai con trỏ: một chỉ đến PCB đầu tiên và một chỉ tới PCB cuối cùng trong danh sách. Chúng ta bổ sung thêm trong mỗi PCB một trường con trỏ chỉ tới PCB kế tiếp trong hàng đợi sẵn sàng.

Hệ điều hành cũng có các hàng đợi khác. Khi một quá trình được cấp phát CPU, nó thực thi một khoảng thời gian và cuối cùng kết thúc, được ngắt, hay chờ một

Thuviendientu.org

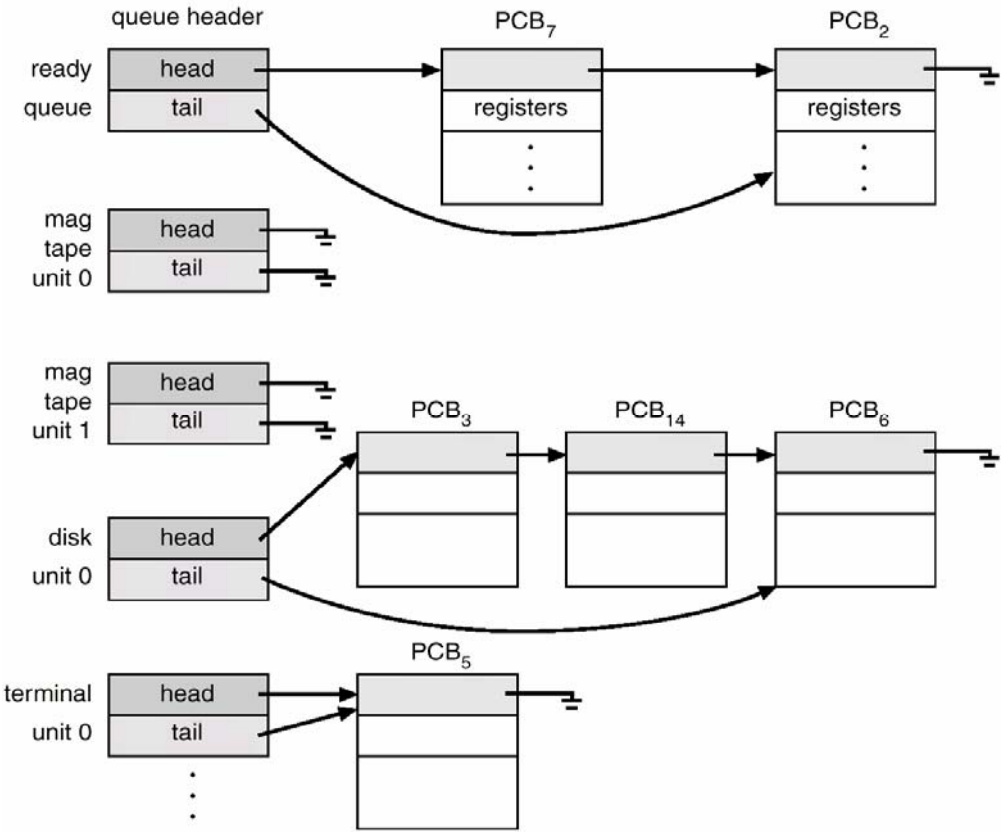
sự kiện xác định xảy ra, chẳng hạn như hoàn thành một yêu cầu nhập/xuất. Trong trường hợp yêu cầu nhập/xuất, một yêu cầu có thể là ổ đĩa băng từ tận hiến hay một thiết bị được chia sẻ như đĩa. Vì hệ thống có nhiều quá trình, đĩa có thể bận với yêu cầu nhập/xuất của các quá trình khác. Do đó, quá trình phải chờ đĩa. Danh sách quá trình chờ một thiết bị nhập/xuất cụ thể được gọi là hàng đợi thiết bị. Mỗi thiết bị có hàng đợi của chính nó như hình III.4.

Một biểu diễn chung của lập thời biểu quá trình là một lưu đồ hàng đợi, như hình III.5. Mỗi hình chữ nhật hiện diện một hàng đợi. Hai loại hàng đợi được hiện diện: hàng đợi sẵn sàng và tập các hàng đợi thiết bị, vòng tròn hiện diện tài nguyên phục vụ hàng đợi, các mũi tên hiển thị dòng các quá trình trong hệ thống.

Một quá trình mới khởi đầu được đặt vào hàng đợi. Nó chờ trong hàng đợi sẵn sàng cho tới khi nó được chọn thực thi. Một khi quá trình được gán tới CPU và đang thực thi, một trong nhiều sự kiện có thể xảy ra:

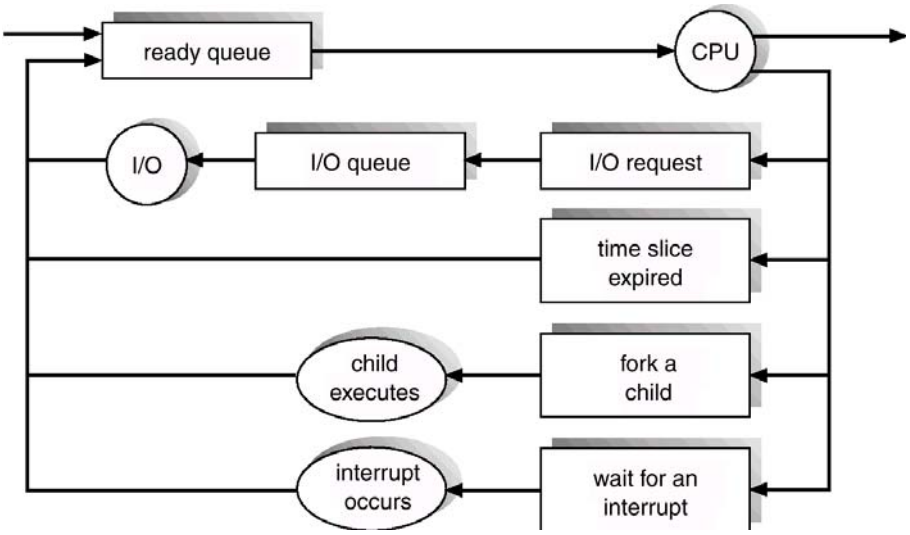
- Quá trình có thể phát ra một yêu cầu nhập/xuất và sau đó được đặt vào trong hàng đợi nhập/xuất.
- Quá trình có thể tạo một quá trình con và chờ cho quá trình con kết thúc
- Khi một ngắt xảy ra, quá trình có thể bị buộc trả lại CPU và được đặt trở lại trong hàng đợi sẵn sàng.

Trong hai trường hợp đầu, cuối cùng quá trình chuyển từ trạng thái chờ tới trạng thái sẵn sàng và sau đó đặt trở lại vào hàng đợi sẵn sàng. Một quá trình tiếp tục chu kỳ này cho tới khi nó kết thúc. Tại thời điểm kết thúc nó bị xoá từ tất cả hàng đợi. Sau đó, PCB và tài nguyên của nó được thu hồi.



Hình 0-4-Hàng đợi sẵn sàng và các hàng đợi nhập/xuất khác nhau

Thuviendientu.org



Hình 0-5-Biểu diễn lưu đồ hàng đợi của lập thời biểu quá trình

IV.2 Bộ định thời biểu

Một quá trình di dời giữa hai hàng đợi định thời khác nhau suốt thời gian sống của nó. Hệ điều hành phải chọn, cho mục đích định thời, các quá trình từ các hàng đợi này. Quá trình chọn lựa này được thực hiện bởi bộ định thời hợp lý.

Trong hệ thống bó, thường nhiều hơn một quá trình được gởi đến hơn là có thể được thực thi tức thì. Các quá trình này được lưu tới thiết bị lưu trữ (như đĩa), nơi chúng được giữ cho việc thực thi sau đó. **Bộ định thời dài** (long-term scheduler) hay bộ định thời công việc (job scheduler), chọn các quá trình từ vùng đệm và nạp chúng vào bộ nhớ để thực thi. **Bộ định thời ngắn** (short-term scheduler) hay bộ định thời CPU chọn một quá trình từ các quá trình sẳn sàng thực thi và cấp phát CPU cho quá trình đó.

Sự khác biệt chủ yếu giữa hai bộ định thời là tính thường xuyên của việc thực thi. Bộ định thời CPU phải chọn một quá trình mới cho CPU thường xuyên. Một quá trình có thể thực thi chỉ một vài mili giây trước khi chờ yêu cầu nhập/xuất. Bộ định thời CPU thường thực thi ít nhất một lần mỗi 100 mili giây. Vì thời gian ngắn giữa việc thực thi nên bộ định thời phải nhanh. Nếu nó mất 10 mili giây để quyết định thực thi một quá trình 100 mili giây thì 10/(100+10) = 9 phần trăm của CPU đang được dùng (hay bị lãng phí) đơn giản cho định thời công việc.

Ngược lại, bộ định thời công việc thực thi ít thường xuyên hơn. Có vài phút giữa việc tạo các quá trình mới trong hệ thống. Bộ định thời công việc điều khiển mức độ đa chương – số quá trình trong bộ nhớ. Nếu mức độ đa chương ổn định thì tốc độ trung bình của việc tạo quá trình phải bằng tốc độ khởi hành trung bình của quá trình rời hệ thống. Vì khoảng thời gian dài hơn giữa việc thực thi nên bộ định thời công việc có thể cấp nhiều thời gian hơn để chọn một quá trình thực thi.

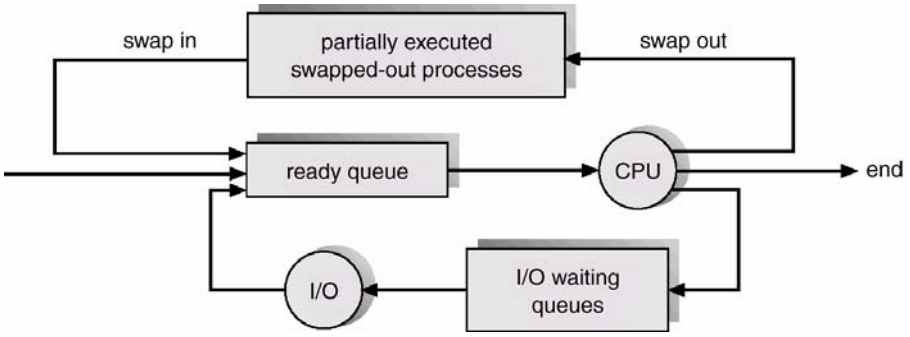
Bộ định thời công việc phải thực hiện một chọn lựa cẩn thận. Thông thường, hầu hết các quá trình có thể được mô tả như là **quá trình hướng nhập/xuất** (I/O-bound proces) hay **quá trình hướng CPU** (CPU-bound process). Một quá trình hướng nhập/xuất mất nhiều thời gian hơn để thực hiện nhập/xuất hơn thời gian tính toán. Ngược lại, một quá trình hướng CPU phát sinh các yêu cầu nhập/xuất không thường xuyên, dùng thời gian để thực hiện việc tính toán hơn một quá trình hướng nhập/xuất dùng. Bộ định thời công việc nên chọn sự kết hợp hài hoà giữa quá trình

Thuviendientu.org

hướng nhập/xuất và quá trình hướng CPU. Nếu tất cả quá trình là hướng nhập/xuất thì hàng đợi sẵn sàng sẽ luôn rỗng và bộ định thời CPU sẽ có ít công việc để thực hiện. Nếu tất cả quá trình là hướng CPU thì hàng đợi nhập/xuất sẽ luôn rỗng, các thiết bị sẽ không được sử dụng và hệ thống sẽ mất cân bằng. Hệ thống với năng lực tốt nhất sẽ có sự kết hợp các quá trình hướng CPU và hướng nhập/xuất.

Trên một vài hệ thống, bộ định thời công việc có thể không có hay rất ít. Thí dụ, các hệ thống chia thời như UNIX thường không có bộ định thời công việc nhưng đơn giản đặt mỗi quá trình mới vào bộ nhớ cho bộ định thời CPU. Khả năng ổn định của hệ thống này phụ thuộc vào giới hạn vật lý (như số lượng thiết bị cuối sẵn dùng) hay tính tự nhiên tự chuyển đổi của người dùng. Nếu năng lực thực hiện giảm tới mức độ không thể chấp nhận được thì một số người dùng sẽ thoát khỏi hệ thống.

Một số hệ thống như hệ chia thời có thể đưa vào một cấp độ định thời bổ sung hay định thời trung gian. **Bộ định thời trung gian** (medium-term process) này (được hiển thị trong lưu đồ hình III-6) xóa các quá trình ra khỏi bộ nhớ (từ sự canh tranh CPU) và do đó giảm mức độ đa chương. Tại thời điểm sau đó, quá trình có thể được đưa trở lại bộ nhớ và việc thực thi của nó có thể được tiếp tục nơi nó bị đưa ra. Cơ chế này được gọi là **hoán vị** (swapping). Quá trình được hoán vị ra và sau đó được hoán vị vào bởi bộ định thời trung gian. Hoán vị là cần thiết để cải tiến sự trộn lẫn quá trình (giữa các quá trình hướng nhập/xuất và hướng CPU), hay vì một thay đổi trong yêu cầu bộ nhớ vượt quá kích thước bộ nhớ sẵn dùng. Hoán vị sẽ được thảo luận trong chương sau.



IV.3 Chuyển ngữ cảnh

Chuyển CPU tới một quá trình khác yêu cầu lưu trạng thái của quá trình cũ và nạp trạng thái được lưu cho quá trình mới. Tác vụ này được xem như **chuyển ngữ cảnh** (context switch). **Ngữ cảnh** của quá trình được hiện diện trong PCB của quá trình; Nó chứa giá trị các thanh ghi, trạng thái quá trình (hình III.1) và thông tin quản lý bộ nhớ. Khi chuyển ngữ cảnh ngữ cảnh xảy ra, nhân lưu ngữ cảnh của quá trình cũ trong PCB của nó và nạp ngữ cảnh được lưu của quá trình mới được định thời để chạy. Thời gian chuyển ngữ cảnh là chi phí thuần vì hệ thống không thực hiện công việc có ích trong khi chuyển. Tốc độ của nó khác từ máy này tới máy khác phụ thuộc vào tốc độ bộ nhớ, số lượng thanh ghi phải được chép và sự tồn tại của các chỉ thị đặc biệt (như chỉ thị để nạp và lưu tất cả thanh ghi). Điển hình dãy tốc độ từ 1 tới 1000 mili giây.

Những lần chuyển đổi ngữ cảnh phụ thuộc nhiều vào hỗ trợ phần cứng. Thí dụ, vài bộ xử lý (như Sun UltraSPARC) cung cấp nhiều tập thanh ghi. Một chuyển ngữ cảnh đơn giản chứa chuyển đổi con trỏ tới tập thanh ghi hiện hành. Dĩ nhiên, nếu quá trình hoạt động vượt quá tập thanh ghi thì hệ thống sắp xếp lại dữ liệu thanh ghi tới và từ bộ nhớ. Cũng vì thế mà hệ điều hành phức tạp hơn và nhiều công việc được làm

Thuviendientu.org

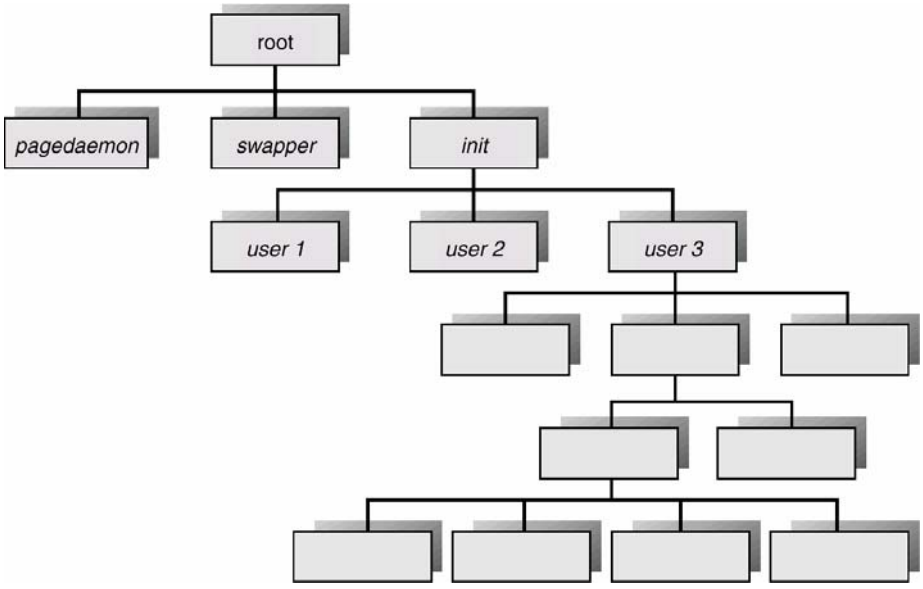
hơn trong khi chuyển ngữ cảnh. Kỹ thuật quản lý bộ nhớ nâng cao có thể yêu cầu dữ liệu bổ sung để được chuyển với mỗi ngữ cảnh. Thí dụ, không gian địa chỉ của quá trình hiện hành phải được lưu khi không gian của tác vụ kế tiếp được chuẩn bị dùng. Không gian địa chỉ được lưu như thế nào và lượng công việc được yêu cầu để lưu nó phụ thuộc vào phương pháp quản lý bộ nhớ của hệ điều hành. Chuyển ngữ cảnh có thể dẫn đến thắt cổ chai năng lực thực hiện vì thế các lập trình viên đang sử dụng các cấu trúc mới để tránh nó bất cứ khi nào có thể.

V Thao tác trên quá trình

Các quá trình trong hệ thống có thể thực thi đồng hành và chúng phải được tạo và xóa tự động. Do đó, hệ điều hành phải cung cấp một cơ chế (hay phương tiện) cho việc tạo quá trình và kết thúc quá trình.

V.1 Tạo quá trình

Quá trình có thể tạo nhiều quá trình mới, bằng một lời gọi hệ thống **create-process**, trong khi thực thi. Quá trình tạo gọi là quá trình cha, ngược lại các quá trình mới được gọi là quá trình con của quá trình đó. Mỗi quá trình mới này sau đó có thể tạo các quá trình khác, hình thành một cây quá trình (hình III-7).



Hình 0-7-Cây quá trình trên một hệ thống UNIX điển hình

Thông thường, một quá trình sẽ cần các tài nguyên xác định (như thời gian CPU, bộ nhớ, tập tin, thiết bị nhập/xuất) để hoàn thành tác vụ của nó. Khi một quá trình tạo một quá trình con, quá trình con có thể nhận tài nguyên của nó trực tiếp từ hệ điều hành hay nó có thể bị ràng buộc tới một tập con các tài nguyên của quá trình cha. Quá trình cha có thể phải phân chia các tài nguyên giữa các quá trình con hay có thể chia sẻ một số tài nguyên (như bộ nhớ và tập tin) giữa nhiều quá trình con. Giới hạn một quá trình con tới một tập con tài nguyên của quá trình cha ngăn chặn bất cứ quá trình từ nạp chồng hệ thống bằng cách tạo quá nhiều quá trình con.

Ngoài ra, khi một quá trình được tạo nó lấy tài nguyên vật lý và luận lý khác nhau, dữ liệu khởi tạo (hay nhập) có thể được truyền từ quá trình cha tới quá trình con. Thí dụ, xét một quá trình với toàn bộ chức năng là hiển thị trạng thái của một tập

Thuviendientu.org

tin F_1 trên màn hình. Khi nó được tạo, nó sẽ lấy dữ liệu vào từ quá trình cha, tên của tập tin F_1 và nó sẽ thực thi dùng dữ liệu đó để lấy thông tin mong muốn.

Nó có thể cũng lấy tên của thiết bị xuất. Một số hệ điều hành chuyển tài nguyên tới quá trình con. Trên hệ thống như thế, quá trình mới có thể lấy hai tập tin đang mở, F_1 và thiết bị cuối, và có thể chỉ cần chuyển dữ liệu giữa hai tập tin.

Khi một quá trình tạo một quá trình mới, hai khả năng có thể tồn tại trong thuật ngữ của việc thực thi:

- Quá trình cha tiếp tục thực thi đồng hành với quá trình con của nó.
- Quá trình cha chờ cho tới khi một vài hay tất cả quá trình con kết thúc.

Cũng có hai khả năng trong thuật ngữ không gian địa chỉ của quá trình mới:

- Quá trình con là bản sao của quá trình cha.
- Quá trình con có một chương trình được nạp vào nó.

Để hiển thị việc cài đặt khác nhau này, chúng ta xem xét hệ điều hành UNIX. Trong UNIX, mỗi quá trình được xác định bởi **danh biểu quá trình** (process identifier), là số nguyên duy nhất. Một quá trình mới được tạo bởi lời gọi hệ thống **fork**. Quá trình mới chứa bản sao của không gian địa chỉ của quá trình gốc. Cơ chế này cho phép quá trình cha giao tiếp dễ dàng với quá trình con. Cả hai quá trình (cha và con) tiếp tục thực thi tại chỉ thị sau khi lời gọi hệ thống **fork**, với một sự khác biệt: mã trả về cho lời gọi hệ thống fork là không cho quá trình mới (con), ngược lại danh biểu quá trình (khác không) của quá trình con được trả về tới quá trình cha.

Điển hình lời gọi hệ thống **execlp** được dùng sau lời gọi hệ thống **fork** bởi một trong hai quá trình để thay thế không gian bộ nhớ với quá trình mới. Lời gọi hệ thống **execlp** nạp tập tin nhị phân vào trong bộ nhớ-xóa hình ảnh bộ nhớ của chương trình chứa lời gọi hệ thống **execlp** - và bắt đầu việc thực thi của nó. Trong cách thức này, hai quá trình có thể giao tiếp và sau đó thực hiện cách riêng của nó. Sau đó, quá trình cha có thể tạo nhiều hơn quá trình con, hay nếu nó không làm gì trong thời gian quá trình con chạy thì nó sẽ phát ra lời gọi hệ thống **wait** để di chuyển nó vào hàng đợi sẵn sàng cho tới khi quá trình con kết thúc. Chương trình C (hình III-8 dưới đây) hiển thị lời gọi hệ thống UNIX được mô tả trước đó. Quá trình cha tạo một quá trình con sử dụng lời gọi hệ thống **fork**. Bây giờ chúng ta có hai quá trình khác nhau chạy một bản sao của cùng chương trình. Giá trị **pid** cho quá trình con là 0; cho quá trình cha là một số nguyên lớn hơn 0. Quá trình con phủ lắp không gian địa chỉ của nó với lệnh của UNIX là **/bin/ls** (được dùng để liệt kê thư mục) dùng lời gọi hệ thống **execlp**. Quá trình cha chờ cho quá trình con hoàn thành với lời gọi hệ thống **wait**. Khi quá trình con hoàn thành, quá trình cha bắt đầu lại từ lời gọi hệ thống **wait** nơi nó hoàn thành việc sử dụng lời gọi hệ thống **exit**.

Ngược lại, hệ điều hành DEC VMS tạo một quá trình mới, nạp chương trình xác định trong quá trình đó và bắt đầu thực thi nó. Hệ điều hành Microsoft Windows NT hỗ trợ cả hai mô hình: không gian địa chỉ của quá trình cha có thể được sao lại hay quá trình cha có thể xác định tên của một chương trình cho hệ điều hành nạp vào không gian địa chỉ của quá trình mới.

Thuviendientu.org


```
#include<stdio.h>
main(int argc, char* argv[])
{
        int     pid;
        /*fork another process*/
        pid     =       fork();
        if(pid<0){      /*error occurred */
                fprintf(stderr, "Fork Failed");
                exit(-1);
}
else if (pid==0){       /*child process*/
                execlp("/bin/ls","ls",NULL);
}
else {  /*parent process*/
        /*parent will wait for the child to complete*/
        wait(NULL);
        printf("Child Complete");
        exit(0);
}
}
```

Hình 0-8-Chương trình C tạo một quá trình riêng rẻ

V.2 Kết thúc quá trình

Một quá trình kết thúc khi nó hoàn thành việc thực thi câu lệnh cuối cùng và yêu cầu hệ điều hành xóa nó bằng cách sử dụng lời gọi hệ thống **exit**. Tại thời điểm đó, quá trình có thể trả về dữ liệu (đầu ra) tới quá trình cha (bằng lời gọi hệ thống **wait**). Tất cả tài nguyên của quá trình –gồm bộ nhớ vật lý và luận lý, các tập tin đang mở, vùng đệm nhập/xuất-được thu hồi bởi hệ điều hành.

Việc kết thúc xảy ra trong các trường hợp khác. Một quá trình có thể gây kết thúc một quá trình khác bằng một lời gọi hệ thống hợp lý (thí dụ, abort). Thường chỉ có quá trình cha bị kết thúc có thể gọi lời gọi hệ thống như thế. Ngược lại, người dùng có thể tùy ý giết (kill) mỗi công việc của quá trình còn lại. Do đó, quá trình cha cần biết các định danh của các quá trình con. Vì thế khi một quá trình tạo một quá trình mới, định danh của mỗi quá trình mới được tạo được truyền tới cho quá trình cha.

Một quá trình cha có thể kết thúc việc thực thi của một trong những quá trình con với nhiều lý do khác nhau:

- Quá trình con sử dụng tài nguyên vượt quá mức được cấp. Điều này yêu cầu quá trình cha có một cơ chế để xem xét trạng thái của các quá trình con.
- Công việc được gán tới quá trình con không còn cần thiết nữa.
- Quá trình cha đang kết thúc và hệ điều hành không cho phép một quá trình con tiếp tục nếu quá trình cha kết thúc. Trên những hệ thống như thế, nếu một quá trình kết thúc (bình thường hoặc không bình thường), thì tất cả quá trình con cũng phải kết thúc. Trường hợp này được xem như kết thúc xếp tầng (cascading termination) thường được khởi tạo bởi hệ điều hành.

Để hiển thị việc thực thi và kết thúc quá trình, xem xét hệ điều hành UNIX chúng ta có thể kết thúc một quá trình dùng lời gọi hệ thống **exit**; nếu quá trình cha có thể chờ cho đến khi quá trình con kết thúc bằng lời gọi hệ thống **wait**. Lời gọi hệ thống **wait** trả về định danh của quá trình con bị kết thúc để quá trình cha có thể xác

Thuviendientu.org

định những quá trình con nào có thể kết thúc. Tuy nhiên, nếu quá trình cha kết thúc thì tất cả quá trình con của nó được gán như quá trình cha mới của chúng, quá trình khởi tạo (init process). Do đó, các quá trình con chỉ có một quá trình cha để tập hợp trạng thái và thống kê việc thực thi.

V.3 Hợp tác quá trình

Các quá trình đồng hành thực thi trong hệ điều hành có thể là những quá trình độc lập hay những quá trình hợp tác. Một quá trình là **độc lập** (independent) nếu nó không thể ảnh hưởng hay bị ảnh hưởng bởi các quá trình khác thực thi trong hệ thống. Rõ ràng, bất kỳ một quá trình không chia sẻ bất cứ dữ liệu (tạm thời hay cố định) với quá trình khác là độc lập. Ngược lại, một quá trình là **hợp tác** (cooperating) nếu nó có thể ảnh hưởng hay bị ảnh hưởng bởi các quá trình khác trong hệ thống. Hay nói cách khác, bất cứ quá trình chia sẻ dữ liệu với quá trình khác là quá trình hợp tác.

Chúng ta có thể cung cấp một môi trường cho phép hợp tác quá trình với nhiều lý do:

- **Chia sẻ thông tin**: vì nhiều người dùng có thể quan tâm cùng phần thông tin (thí dụ, tập tin chia sẻ), chúng phải cung cấp một môi trường cho phép truy xuất đồng hành tới những loại tài nguyên này.
- **Gia tăng tốc độ tính toán**: nếu chúng ta muốn một tác vụ chạy nhanh hơn, chúng ta phải chia nó thành những tác vụ nhỏ hơn, mỗi tác vụ sẽ thực thi song song với các tác vụ khác. Việc tăng tốc như thế có thể đạt được chỉ nếu máy tính có nhiều thành phần đa xử lý (như các CPU hay các kênh I/O).
- **Tính module hóa**: chúng ta muốn xây dựng hệ thống trong một kiểu mẫu dạng module, chia các chức năng hệ thống thành những quá trình hay luồng như đã thảo luận ở chương trước.
- **Tính tiện dụng**: Thậm chí một người dùng đơn có thể có nhiều tác vụ thực hiện tại cùng thời điểm. Thí dụ, một người dùng có thể đang soạn thảo, in, và biên dịch cùng một lúc.

Thực thi đồng hành của các quá trình hợp tác yêu cầu các cơ chế cho phép các quá trình giao tiếp với các quá trình khác và đồng bộ hóa các hoạt động của chúng. Để minh họa khái niệm của các quá trình cộng tác, chúng ta xem xét bài toán người sản xuất-người tiêu thụ, là mô hình chung cho các quá trình hợp tác. Quá trình người sản xuất cung cấp thông tin được tiêu thụ bởi quá trình người tiêu thụ. Thí dụ, một chương trình in sản xuất các ký tự được tiêu thụ bởi trình điều khiển máy in. Một trình biên dịch có thể sản xuất mã hợp ngữ được tiêu thụ bởi trình hợp ngữ. Sau đó, trình hợp ngữ có sản xuất module đối tượng, được tiêu thụ bởi bộ nạp.

Để cho phép người sản xuất và người tiêu thụ chạy đồng hành, chúng ta phải có sẵn một vùng đệm chứa các sản phẩm có thể được điền vào bởi người sản xuất và được lấy đi bởi người tiêu thụ. Người sản xuất có thể sản xuất một sản phẩm trong khi người tiêu thụ đang tiêu thụ một sản phẩm khác. Người sản xuất và người tiêu thụ phải được đồng bộ để mà người tiêu thụ không cố gắng tiêu thụ một sản phẩm mà chưa được sản xuất. Trong trường hợp này, người tiêu thụ phải chờ cho tới khi các sản phẩm mới được tạo ra.

Bài toán người sản xuất-người tiêu thụ với vùng đệm không bị giới hạn (**unbounded-buffer**) thiết lập không giới hạn kích thước của vùng đệm. Người tiêu thụ có thể phải chờ các sản phẩm mới nhưng người sản xuất có thể luôn tạo ra sản phẩm mới. Vấn đề người sản xuất-người tiêu thụ với vùng đệm có kích thước giới hạn

Thuviendientu.org

(**bounded-buffer**) đảm bảo một kích thước cố định cho vùng đệm. Trong trường hợp này, người tiêu thụ phải chờ nếu vùng đệm rỗng, và người sản xuất phải chờ nếu vùng đệm đầy.

Vùng đệm có thể được cung cấp bởi hệ điều hành thông qua việc sử dụng phương tiện **giao tiếp liên quá trình** (interprocess-communication-IPC), hay được mã hóa cụ thể bởi người lập trình ứng dụng với việc sử dụng bộ nhớ được chia sẻ. Để chúng ta hiển thị một giải pháp chia sẻ bộ nhớ đối với vấn đề vùng đệm bị giới hạn (bounded-buffer). Quá trình người sản xuất và người tiêu thụ chia sẻ các biến sau:

```
#define BUFFER_SIZE     10
typedef struct{
        …
} item;
item buffer[BUFFER_SIZE];
int     in      =       0;
int     out     =       0;
```

Vùng đệm được chia sẻ được cài đặt như một mảng vòng với hai con trỏ luận lý: **in** và **out**. Biến **in** chỉ tới vị trí trống kế tiếp trong vùng đệm; **out** chỉ tới vị trí đầy đầu tiên trong vùng đệm. Vùng đệm là rỗng khi **in==out**; vùng đệm là đầy khi **((in + 1)%BUFFER_SIZE) ==out**.

Mã cho quá trình người sản xuất và người tiêu thụ được trình bày dưới đây. Quá trình người sản xuất có một biến **nextProduced** trong đó sản phẩm mới được tạo ra và được lưu trữ:

```
while (1) {
        /*produce an item in nextProduced*/
        while (((in + 1)%BUFFER_SIZE) ==out)
                ; /*do nothing*/
        buffer[in]      =       nextProduced;
        in              =       (in + 1) % BUFFER_SIZE;
}
```

Quá trình người tiêu thụ có biến cục bộ **nextConsumed** trong đó sản phẩm được tiêu thụ và được lưu trữ:

```
while (1){
        while (in==out)
                ; //nothing
        nextConsumed =      buffer[out];
        out     =       (out    +       1)      % BUFFER_SIZE;
        /*consume the item in nextConsumed*/
}
```

Cơ chế này cho phép nhiều nhất BUFFER_SIZE –1 trong vùng đệm tại cùng một thời điểm.

VI Giao tiếp liên quá trình

Trong phần trên chúng ta đã hiển thị cách mà các quá trình hợp tác có thể giao tiếp với nhau trong một môi trường chia sẻ bộ nhớ. Cơ chế yêu cầu các quá trình này chia sẻ nhóm vùng đệm chung và mã cho việc cài đặt vùng đệm được viết trực tiếp bởi người lập trình ứng dụng. Một cách khác đạt được cùng ảnh hưởng cho hệ điều hành là cung cấp phương tiện cho các quá trình hợp tác giao tiếp với nhau bằng một phương tiện giao tiếp liên quá trình (IPC).

Thuviendientu.org

IPC cung cấp một cơ chế cho phép một quá trình giao tiếp và đồng bộ các hoạt động của chúng mà không chia sẻ cùng không gian địa chỉ. IPC đặc biệt có ích trong môi trường phân tán nơi các quá trình giao tiếp có thể thường trú trên các máy tính khác được nối kết qua mạng. Thí dụ chương trình **chat** được dùng trên World Wide Web.

IPC được cung cấp bởi hệ thống truyền thông điệp, và các hệ thống truyền thông điệp có thể được định nghĩa trong nhiều cách. Trong phần này chúng ta sẽ xem xét những vấn đề khác nhau khi thiết kế các hệ thống truyền thông điệp.

VI.1 Hệ thống truyền thông điệp

Chức năng của hệ thống truyền thông điệp là cho phép các quá trình giao tiếp với các quá trình khác mà không cần sắp xếp lại dữ liệu chia sẻ. Chúng ta xem truyền thông điệp được dùng như một phương pháp giao tiếp trong vi nhân. Trong cơ chế này, các dịch vụ được cung cấp như các quá trình người dùng thông thường. Nghĩa là, các dịch vụ hoạt động bên ngoài nhân. Giao tiếp giữa các quá trình người dùng được thực hiện thông qua truyền thông điệp. Một phương tiện IPC cung cấp ít nhất hai hoạt động: **send(message)** và **receive(message)**.

Các thông điệp được gởi bởi một quá trình có thể có kích thước cố định hoặc biến đổi. Nếu chỉ các thông điệp có kích thước cố định được gởi, việc cài đặt cấp hệ thống là đơn giản hơn. Tuy nhiên, hạn chế này làm cho tác vụ lập trình sẽ phức tạp hơn. Ngoài ra, các thông điệp có kích thước thay đổi yêu cầu việc cài đặt mức hệ thống phức tạp hơn nhưng tác vụ lập trình trở nên đơn giản hơn.

Nếu quá trình P và Q muốn giao tiếp, chúng phải gởi các thông điệp tới và nhận thông điệp từ với nhau; một liên kết giao tiếp phải tồn tại giữa chúng. Liên kết này có thể được cài đặt trong những cách khác nhau. Ở đây chúng ta quan tâm đến cài đặt luận lý hơn là cài đặt vật lý. Có vài phương pháp cài đặt một liên kết và các hoạt động send/receive:

- Giao tiếp trực tiếp hay gián tiếp
- Giao tiếp đối xứng hay bất đối xứng
- Gởi bằng bản sao hay tham chiếu
- Thông điệp có kích thước cố định hay thay đổi

VI.2 Đặt tên

Các quá trình muốn giao tiếp phải có cách tham chiếu với nhau. Chúng có thể dùng giao tiếp trực tiếp hay gián tiếp.

VI.2.1 Giao tiếp trực tiếp

Với giao tiếp trực tiếp, mỗi quá trình muốn giao tiếp phải đặt tên rõ ràng người gởi và người nhận của giao tiếp. Trong cơ chế này, các hàm cơ sở send và receive được định nghĩa như sau:

- **Send(P, message)**: gởi một thông điệp tới quá trình P
- **Receive(Q, message)**: nhận một thông điệp từ quá trình Q

Một liên kết giao tiếp trong cơ chế này có những thuộc tính sau:

- Một liên kết được thiết lập tự động giữa mỗi cặp quá trình muốn giao tiếp. Các quá trình cần biết định danh của nhau khi giao tiếp.
- Một liên kết được nối kết với chính xác hai quá trình
- Chính xác một liên kết tồn tại giữa mỗi cặp quá trình.

Thuviendientu.org

Cơ chế này hiển thị tính đối xứng trong việc đánh địa chỉ: nghĩa là, cả hai quá trình gởi và nhận phải biết tên nhau để giao tiếp. Một thay đổi trong cơ chế này thực hiện tính bất đối xứng trong việc đánh địa chỉ. Chỉ người gởi biết tên của người nhận; người nhận không yêu cầu tên của người gởi. Trong cơ chế này các hàm cơ sở được định nghĩa như sau:

- **Send(P, message)**: gởi một thông điệp tới quá trình P
- **Receive(id, message)**: nhận một thông điệp từ bất kỳ quá trình nào; id khác nhau được đặt tên của quá trình mà giao tiếp xảy ra.

Sự bất lợi trong cả hai cơ chế đối xứng và không đối xứng là tính điều chỉnh của việc định nghĩa quá trình bị giới hạn. Thay đổi tên của một quá trình có thể cần xem xét tất cả định nghĩa quá trình khác. Tất cả tham chiếu tới tên cũ phải được tìm thấy để mà chúng có thể được thay đổi thành tên mới. Trường hợp này là không mong muốn từ quan điểm biên dịch riêng.

VI.2.2 Giao tiếp gián tiếp

Với giao tiếp gián tiếp, một thông điệp được gởi tới và nhận từ các hộp thư (mailboxes), hay cổng (ports). Một hộp thư có thể được hiển thị trừu tượng như một đối tượng trong đó các thông điệp có thể được đặt bởi các quá trình và sau đó các thông điệp này có thể được xóa đi. Mỗi hộp thư có một định danh duy nhất. Trong cơ chế này, một quá trình có thể giao tiếp với một vài quá trình khác bằng một số hộp thư khác nhau. Hai quá trình có thể giao tiếp chỉ nếu chúng chia sẻ cùng một hộp thư. Hàm cơ sở **send** và **receive** được định nghĩa như sau:

- **Send(A, message)**: gởi một thông điệp tới hộp thư A.
- **Receive(A, message)**: nhận một thông điệp từ hộp thư A.

Trong cơ chế này, một liên kết giao tiếp có các thuộc tính sau:

- Một liên kết được thiết lập giữa một cặp quá trình chỉ nếu cả hai thành viên của cặp có một hộp thư được chia sẻ.
- Một liên kết có thể được nối kết với nhiều hơn hai quá trình.
- Số các liên kết khác nhau có thể tồn tại giữa mỗi cặp quá trình giao tiếp với mỗi liên kết tương ứng với một hộp thư

Giả sử các quá trình P_1, P_2 và P_3 chia sẻ một hộp thư A. Quá trình P_1 gởi một thông điệp tới A trong khi P_2 và P_3 thực thi việc nhận từ A. Quá trình nào sẽ nhận thông điệp được gởi bởi P_1? Câu trả lời phụ thuộc cơ chế mà chúng ta chọn:

- Cho phép một liên kết được nối kết với nhiều nhất hai quá trình
- Cho phép nhiều nhất một quá trình tại một thời điểm thực thi thao tác nhận.
- Cho phép hệ thống chọn bất kỳ quá trình nào sẽ nhận thông điệp (nghĩa là, hoặc P_1 hoặc P_3 nhưng không phải cả hai sẽ nhận thông điệp). Hệ thống này có thể xác định người nhận tới người gởi.

Một hộp thư có thể được sở hữu bởi một quá trình hay bởi hệ điều hành. Nếu hộp thư được sở hữu bởi một quá trình (nghĩa là, hộp thư là một phần không gian địa chỉ của quá trình), sau đó chúng ta phân biệt giữa người sở hữu (người chỉ nhận thông điệp thông qua hộp thư này) và người dùng (người có thể chỉ gởi thông điệp tới hộp thư).

Thuviendientu.org

Vì mỗi hộp thư có một người sở hữu duy nhất nên không có sự lẫn lộn về người nhận thông điệp được gởi tới hộp thư này. Khi một quá trình sở hữu một hộp thư kết thúc, hộp thư biến mất. Sau đó, bất kỳ quá trình nào gởi thông điệp tới hộp thư này được thông báo rằng hộp thư không còn tồn tại nữa.

Ngoài ra, một hộp thư được sở hữu bởi hệ điều hành độc lập và không được gán tới bất kỳ quá trình xác định nào. Sau đó, hệ điều hành phải cung cấp một cơ chế cho phép một quá trình thực hiện như sau:

- Tạo một hộp thư mới
- Gởi và nhận các thông điệp thông qua hộp thư
- Xóa hộp thư

Mặc định, quá trình tạo hộp thư mới là người sở hữu hộp thư đó. Ban đầu, người sở hữu chỉ là một quá trình có thể nhận thông điệp thông qua hộp thư. Tuy nhiên, việc sở hữu và quyền nhận thông điệp có thể được chuyển tới các quá trình khác thông qua lời gọi hệ thống hợp lý. Dĩ nhiên, sự cung cấp này có thể dẫn đến nhiều người nhận cho mỗi hộp thư.

VI.2.3 Đồng bộ hóa

Giao tiếp giữa hai quá trình xảy ra bởi lời gọi hàm cơ sở **send** và **receive**. Có các tùy chọn thiết kế khác nhau cho việc cài đặt mỗi hàm cơ sở. Truyền thông điệp có thể là nghẽn (block) hay không nghẽn (nonblocking)-cũng được xem như đồng bộ và bất đồng bộ.

- **Hàm send nghẽn**: quá trình gởi bị nghẽn cho đến khi thông điệp được nhận bởi quá trình nhận hay bởi hộp thư.
- **Hàm send không nghẽn**: quá trình gởi gởi thông điệp và thực hiện tiếp hoạt động
- **Hàm receive nghẽn**: người nhận nghẽn cho đến khi thông điệp sẳn dùng
- **Hàm receive không nghẽn**: người nhận nhận lại một thông điệp hợp lệ hay rỗng
-

Sự kết hợp khác nhau giữa send và receive là có thể. Khi cả hai send và receive là nghẽn chúng ta có sự thống nhất giữa người gởi và người nhận.

VI.2.4 Tạo vùng đệm

Dù giao tiếp có thể là trực tiếp hay gián tiếp, các thông điệp được chuyển đổi bởi các quá trình giao tiếp thường trú trong một hàng đợt tạm thời. Về cơ bản, một hàng đợi như thế có thể được cài đặt trong ba cách:

- **Khả năng chứa là 0** (zero capacity): hàng đợi có chiều dài tối đa là 0; do đó liên kết không thể có bất kỳ thông điệp nào chờ trong nó. Trong trường hợp này, người gởi phải nghẽn cho tới khi người nhận nhận thông điệp.
- **Khả năng chứa có giới hạn** (bounded capacity): hàng đợi có chiều dài giới hạn *n*; do đó, nhiều nhất n thông điệp có thể thường trú trong nó. Nếu hàng đợi không đầy khi một thông điệp mới được gởi, sau đó nó được đặt trong hàng đợi (thông điệp được chép hay một con trỏ thông điệp được giữ) và người gởi có thể tiếp tục thực thi không phải chờ.

Thuviendientu.org

Tuy nhiên, liên kết có khả năng chứa giới hạn. Nếu một liên kết đầy, người gởi phải nghẽn cho tới khi không gian là sẵn dùng trong hàng đợi

- **Khả năng chứa không giới hạn** (unbounded capacity): Hàng đợi có khả năng có chiều dài không giới hạn; do đó số lượng thông điệp bất kỳ có thể chờ trong nó. Người gởi không bao giờ nghẽn.

Trường hợp khả năng chứa là 0 thường được xem như hệ thống thông điệp không có vùng đệm; hai trường hợp còn lại được xem như vùng đệm tự động.

VII Tóm tắt

Quá trình là một chương trình đang thực thi. Khi một quá trình thực thi, nó thay đổi trạng thái. Trạng thái của một quá trình được định nghĩa bởi một hoạt động hiện tại của quá trình đó. Mỗi quá trình có thể ở một trong những trạng thái sau: mới (new), sẵn sàng (ready), đang chạy (running), chờ (waiting), hay kết thúc (terminated). Mỗi quá trình được biểu diễn trong hệ điều hành bởi khối điều khiển quá trình của chính nó (PCB).

Một quá trình khi không thực thi, được đặt vào hàng đợi. Hai cấp chủ yếu của hàng đợi trong hệ điều hành là hàng đợi yêu cầu nhập/xuất và hàng đợi sẵn sàng. Hàng đợi sẵn sàng chứa tất cả quá trình sẵn sàng để thực thi và đang chờ CPU. Mỗi quá trình được biểu diễn bởi một PCB và các PCB có thể được liên kết với nhau để hình thành một hàng đợi sẵn sàng. **Định thời biểu dài** (long-term scheduling) (hay định thời biểu công việc) là chọn các quá trình được phép cạnh tranh CPU. Thông thường, định thời biểu dài bị ảnh hưởng nặng nề bởi việc xem xét cấp phát tài nguyên, đặc biệt quản lý bộ nhớ. **Định thời ngắn** (short-term scheduling) là sự chọn lựa một quá trình từ các hàng đợi sẵn sàng.

Các quá trình trong hệ thống có thể thực thi đồng hành. Có nhiều lý do các thực thi đồng hành: chia sẻ thông tin, tăng tốc độ tính toán, hiệu chỉnh và tiện dụng. Thực thi đồng hành yêu cầu cơ chế cho việc tạo và xóa quá trình.

Quá trình thực thi trong hệ điều hành có thể là các quá trình độc lập hay các quá trình hợp tác. Các quá trình hợp tác phải có phương tiện giao tiếp với nhau. Chủ yếu, có hai cơ chế giao tiếp bổ sung cho nhau cùng tồn tại: chia sẻ bộ nhớ và hệ thống truyền thông điệp. Phương pháp chia sẻ bộ nhớ yêu cầu các quá trình giao tiếp chia sẻ một số biến. Các quá trình được mong đợi trao đổi thông tin thông qua việc sử dụng các biến dùng chung này. Trong hệ thống bộ nhớ được chia sẻ, nhiệm vụ cho việc cung cấp giao tiếp tách rời với người lập trình ứng dụng; chỉ hệ điều hành cung cấp hệ thống bộ nhớ được chia sẻ. Phương pháp truyền thông điệp cho phép các quá trình trong đổi thông điệp. Nhiệm vụ cung cấp giao tiếp có thể tách rời với hệ điều hành. Hai cơ chế này không loại trừ hỗ tương và có thể được dùng cùng một lúc trong phạm vi một hệ điều hành.

Thuviendientu.org

CHƯƠNG IV - LUỒNG

IV.1 Mục đích

Sau khi học xong chương này, người học nắm được những kiến thức sau:

- Các khái niệm gán với hệ điều hành đa luồng
- Các vấn đề liên quan với lập trình đa luồng
- Các ảnh hưởng của luồng tới việc thiết kế hệ điều hành
- Cách thức các hệ điều hành hiện đại hỗ trợ luồng

IV.2 Giới thiệu

Mô hình thực thi trong chương 3 giả sử rằng một quá trình là một chương trình đang thực thi với một luồng điều khiển. Nhiều hệ điều hành hiện đại hiện nay cung cấp các đặc điểm cho một quá trình chứa nhiều luồng (thread) điều khiển. Trong chương này giới thiệu các khái niệm liên quan với các hệ thống máy tính đa luồng, gồm thảo luận Pthread API và luồng Java.

Chúng ta sẽ xem xét nhiều vấn đề có liên quan tới lập trình đa luồng và nó ảnh hưởng như thế nào đến thiết kế của hệ điều hành. Cuối cùng, chúng ta sẽ khám phá nhiều hệ điều hành hiện đại hỗ trợ luồng tại cấp độ nhân như thế nào.

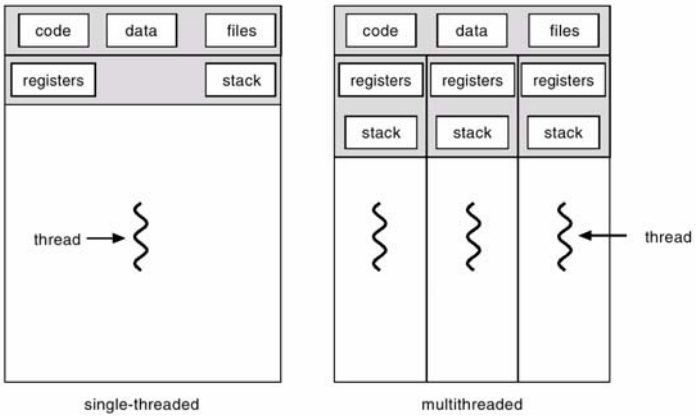
IV.3 Tổng quan

Một luồng thường được gọi là **quá trình nhẹ** (lightweight proces-LWP), là một đơn vị cơ bản của việc sử dụng CPU; nó hình thành gồm: một định danh luồng (thread ID), một bộ đếm chương trình, tập thanh ghi và ngăn xếp. Nó chia sẻ với các luồng khác thuộc cùng một quá trình phần mã, phần dữ liệu, và tài nguyên hệ điều hành như các tập tin đang mở và các tín hiệu. Một quá trình truyền thống (hay quá trình nặng) có một luồng điều khiển đơn. Nếu quá trình có nhiều luồng điều khiển, nó có thể thực hiện nhiều hơn một tác vụ tại một thời điểm. Hình VI.1 hiển thị sự khác nhau giữa quá trình đơn luồng và quá trình đa luồng.

IV.3.1 Sự cơ động

Nhiều gói phần mềm chạy trên các máy để bàn PC là đa luồng. Điển hình, một ứng dụng được cài đặt như một quá trình riêng rẻ với nhiều luồng điều khiển. Một trình duyệt Web có thể có một luồng hiển thị hình ảnh, văn bản trong khi một luồng khác lấy dữ liệu từ mạng. Một trình soạn thảo văn bản có thể có một luồng hiển thị đồ họa, luồng thứ hai đọc sự bấm phím trên bàn phím từ người dùng, một luồng thứ ba thực hiện việc kiểm tra chính tả và từ vựng chạy trong chế độ nền.

Thuviendientu.org



Hình 0-1 Quá trình đơn và đa luồng

Trong những trường hợp cụ thể một ứng dụng đơn có thể được yêu cầu thực hiện nhiều tác vụ đơn. Thí dụ, một trình phục vụ web chấp nhận các yêu cầu khách hàng như trang web, hình ảnh, âm thanh, ..Một trình phục vụ web có thể có nhiều (hàng trăm) khách hàng truy xuất đồng thời nó. Nếu trình phục vụ web chạy như một quá trình đơn luồng truyền thống thì nó sẽ có thể chỉ phục vụ một khách hàng tại cùng thời điểm. Lượng thời gian mà khách hàng phải chờ yêu cầu của nó được phục vụ là rất lớn.

Một giải pháp là có một trình phục vụ chạy như một quá trình đơn chấp nhận các yêu cầu. Khi trình phục vụ nhận một yêu cầu, nó sẽ tạo một quá trình riêng để phục vụ yêu cầu đó. Thật vậy, phương pháp tạo ra quá trình này là cách sử dụng thông thường trước khi luồng trở nên phổ biến. Tạo ra quá trình có ảnh hưởng rất lớn như được trình bày ở chương trước. Nếu quá trình mới sẽ thực hiện cùng tác vụ như quá trình đã có thì tại sao lại gánh chịu tất cả chi phí đó? Thường sẽ hiệu quả hơn cho một quá trình chứa nhiều luồng phục vụ cùng một mục đích. Tiếp cận này sẽ đa luồng quá trình trình phục vụ web. Trình phục vụ sẽ tạo một luồng riêng lắng nghe các yêu cầu người dùng; khi yêu cầu được thực hiện nó không tạo ra quá trình khác mà sẽ tạo một luồng khác phục vụ yêu cầu.

Luồng cũng đóng một vai trò quan trọng trong hệ thống lời gọi thủ tục xa (remote process call-RPC). Như đã trình bày ở chương trước, RPCs cho phép giao tiếp liên quá trình bằng cách cung cấp cơ chế giao tiếp tương tự như các lời gọi hàm hay thủ tục thông thường. Điển hình, các trình phục vụ RPCs là đa luồng. Khi một trình phục vụ nhận một thông điệp, nó phục vụ thông điệp dùng một luồng riêng. Điều này cho phép phục vụ nhiều yêu cầu đồng hành.

IV.3.2 Thuận lợi

Những thuận lợi của lập trình đa luồng có thể được chia làm bốn loại:

- **Sự đáp ứng**: đa luồng một ứng dụng giao tiếp cho phép một chương trình tiếp tục chạy thậm chí nếu một phần của nó bị khóa hay đang thực hiện một thao tác dài, do đó gia tăng sự đáp ứng đối với người dùng. Thí dụ, một trình duyệt web vẫn có thể đáp ứng người dùng bằng một luồng trong khi một ảnh đang được nạp bằng một luồng khác.
- **Chia sẻ tài nguyên**: Mặc định, các luồng chia sẻ bộ nhớ và các tài nguyên của các quá trình mà chúng thuộc về. Thuận lợi của việc chia sẽ mã là nó cho phép

Thuviendientu.org

một ứng dụng có nhiều hoạt động của các luồng khác nhau nằm trong cùng không gian địa chỉ.

- **Kinh tế**: cấp phát bộ nhớ và các tài nguyên cho việc tạo các quá trình là rất đắt. Vì các luồng chia sẻ tài nguyên của quá trình mà chúng thuộc về nên nó kinh tế hơn để tạo và chuyển ngữ cảnh giữa các luồng. Khó để đánh giá theo kinh nghiệm sự khác biệt chi phí cho việc tạo và duy trì một quá trình hơn một luồng, nhưng thường nó sẽ mất nhiều thời gian để tạo và quản lý một quá trình hơn một luồng. Trong Solaris 2, tạo một quá trình chậm hơn khoảng 30 lần tạo một luồng và chuyển đổi ngữ cảnh chậm hơn 5 lần.
- **Sử dụng kiến trúc đa xử lý**: các lợi điểm của đa luồng có thể phát huy trong kiến trúc đa xử lý, ở đó mỗi luồng thực thi song song trên một bộ xử lý khác nhau. Một quá trình đơn luồng chỉ có thể chạy trên một CPU. Đa luồng trên một máy nhiều CPU gia tăng tính đồng hành. Trong kiến trúc đơn xử lý, CPU thường chuyển đổi qua lại giữa mỗi luồng quá nhanh để tạo ra hình ảnh của sự song song nhưng trong thực tế chỉ một luồng đang chạy tại một thời điểm.

IV.3.3 Luồng người dùng và luồng nhân

Chúng ta vừa mới thảo luận là xem xét luồng như một chiều hướng chung. Tuy nhiên, hỗ trợ luồng được cung cấp hoặc ở cấp người dùng, cho các luồng người dùng hoặc ở cấp nhân, cho các luồng nhân.

- **Luồng người dùng**: được hỗ trợ dưới nhân và được cài đặt bởi thư viện luồng tại cấp người dùng. Thư viện cung cấp hỗ trợ cho việc tạo luồng, lập thời biểu, và quản lý mà không có sự hỗ trợ từ nhân. Vì nhân không biết các luồng cấp người dùng, tất cả việc tạo luồng và lập thời biểu được thực hiện trong không gian người dùng mà không cần sự can thiệp của nhân. Do đó, các luồng cấp người dùng thường tạo và quản lý nhanh, tuy nhiên chúng cũng có những trở ngại. Thí dụ, nếu nhân là đơn luồng thì bất cứ luồng cấp người dùng thực hiện một lời gọi hệ thống nghẽn sẽ làm cho toàn bộ quá trình bị nghẽn, thậm chí nếu các luồng khác sẵn dùng để chạy trong ứng dụng. Các thư viện luồng người dùng gồm các luồng POSIX Pthreads, Mach C-threads và Solaris 2 UI-threads.
- **Luồng nhân**: được hỗ trợ trực tiếp bởi hệ điều hành. Nhân thực hiện việc tạo luồng, lập thời biểu, và quản lý không gian nhân. Vì quản lý luồng được thực hiện bởi hệ điều hành, luồng nhân thường tạo và quản lý chậm hơn luồng người dùng. Tuy nhiên, vì nhân được quản lý các luồng nếu một luồng thực hiện lời gọi hệ thống nghẽn, nhân có thể lập thời biểu một luồng khác trong ứng dụng thực thi. Trong môi trường đa xử lý, nhân có thể lập thời biểu luồng trên một bộ xử lý khác. Hầu hết các hệ điều hành hiện nay như Windows NT, Windows 2000, Solaris 2, BeOS và Tru64 UNIX (trước Digital UNIX)-hỗ trợ các luồng nhân.

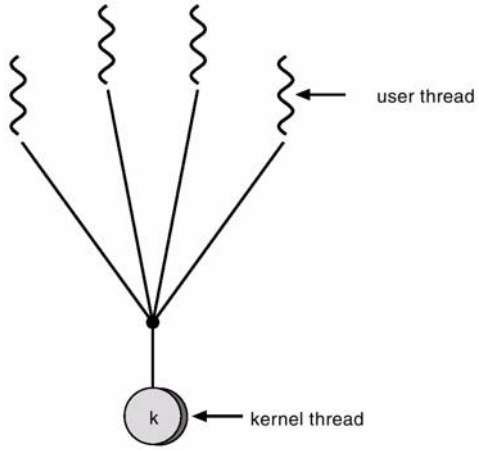
IV.4 Mô hình đa luồng

Nhiều hệ thống cung cấp sự hỗ trợ cả hai luồng nhân và luồng người dùng nên tạo ra nhiều mô hình đa luồng khác nhau. Chúng ta sẽ xem xét ba loại cài đặt luồng thông thường

Thuviendientu.org

IV.4.1 Mô hình nhiều-một

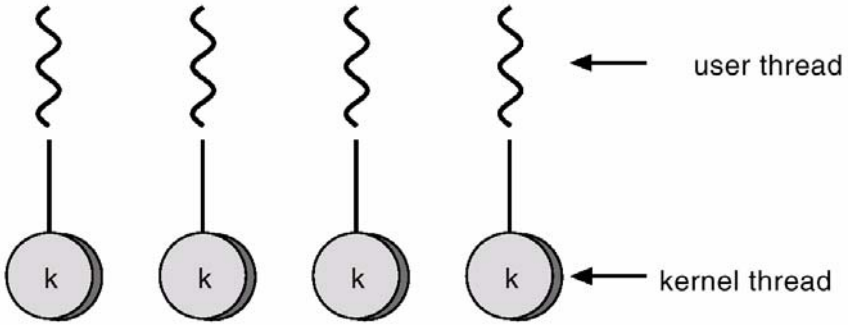
Mô hình nhiều-một (như hình IV.2) ánh xạ nhiều luồng cấp người dùng tới một luồng cấp nhân. Quản lý luồng được thực hiện trong không gian người dùng vì thế nó hiệu quả nhưng toàn bộ quá trình sẽ bị khóa nếu một luồng thực hiện lời gọi hệ thống khóa. Vì chỉ một luồng có thể truy xuất nhân tại một thời điểm nên nhiều luồng không thể chạy song song trên nhiều bộ xử lý. Green threads-một thư viện luồng được cài đặt trên các hệ điều hành không hỗ trợ luồng nhân dùng mô hình nhiều-một.



Hình 0-2-Mô hình nhiều-một

IV.4.2 Mô hình một-một

Mô hình một-một (hình IV.3) ánh xạ mỗi luồng người dùng tới một luồng nhân. Nó cung cấp khả năng đồng hành tốt hơn mô hình nhiều-một bằng cách cho một luồng khác chạy khi một luồng thực hiện lời gọi hệ thống nghẽn; nó cũng cho phép nhiều luồng chạy song song trên các bộ xử lý khác nhau. Chỉ có một trở ngại trong mô hình này là tạo luồng người dùng yêu cầu tạo một luồng nhân tương ứng. Vì chi phí cho việc tạo luồng nhân có thể đè nặng lên năng lực thực hiện của ứng dụng, các cài đặt cho mô hình này giới hạn số luồng được hỗ trợ bởi hệ thống. Windows NT, Windows 2000 và OS/2 cài đặt mô hình một-một này.

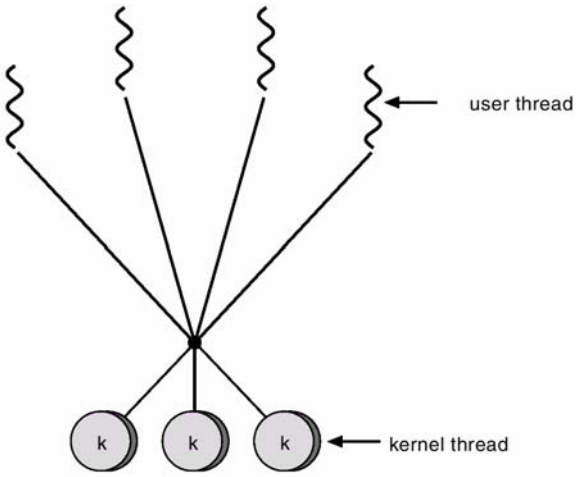


Hình 0-3-Mô hình một-một

Thuviendientu.org

IV.4.3 Mô hình nhiều-nhiều

Mô hình nhiều-nhiều (như hình VI.4) đa hợp nhiều luồng cấp người dùng tới số lượng nhỏ hơn hay bằng các luồng nhân. Số lượng các luồng nhân có thể được xác định hoặc một ứng dụng cụ thể hay một máy cụ thể (một ứng dụng có thể được cấp nhiều luồng nhân trên một bộ đa xử lý hơn trên một bộ đơn xử lý). Trong khi mô hình nhiều-một cho phép người phát triển tạo nhiều luồng người dùng như họ muốn, thì đồng hành thật sự là không đạt được vì nhân có thể lập thời biểu chỉ một luồng tại một thời điểm. Mô hình một-một cho phép đồng hành tốt hơn nhưng người phát triển phải cẩn thận không tạo ra quá nhiều luồng trong một ứng dụng. Mô hình nhiều-nhiều gặp phải một trong hai vấn đề khiếm khuyết: người phát triển có thể tạo nhiều luồng người dùng khi cần thiết và các luồng nhân tương ứng có thể chạy song song trên một bộ đa xử lý. Khi một luồng thực hiện một lời gọi hệ thống khóa, nhân có thể lập thời biểu một luồng khác thực thi. Solaris 2, IRIX, HP-UX, và Tru64 UNIX hỗ trợ mô hình này.



Hình 0-4-Mô hình nhiều-nhiều

IV.5 Cấp phát luồng

Trong phần này chúng ta thảo luận các cấp phát xem xét với các chương trình đa luồng.

IV.5.1 Lời gọi hệ thống fork và exec

Trong chương trước chúng ta mô tả lời gọi hệ thống *fork* được dùng để tạo một quá trình bản sao riêng như thế nào. Trong một chương trình đa luồng, ngữ nghĩa của các lời gọi hệ thống *fork* và *exec* thay đổi. Nếu một luồng trong lời gọi chương trình fork thì quá trình mới sao chép lại quá trình tất cả luồng hay là một quá trình đơn luồng mới? Một số hệ thống UNIX chọn hai ấn bản fork, một sao chép lại tất cả luồng và một sao chép lại chỉ luồng được nạp lên lời gọi hệ thống fork. Lời gọi hệ thống exec điển hình thực hiện công việc trong cùng một cách như được mô tả trong chương trước. Nghĩa là, nếu một luồng nạp lời gọi hệ thống *exec*, chương trình được xác định trong tham số exec sẽ thay thế toàn bộ quá trình-chứa tất cả luồng và các quá trình tải nhẹ.

Thuviendientu.org

Việc sử dụng hai ấn bản fork phụ thuộc vào ứng dụng. Nếu exec bị hủy tức thì sau khi phân nhánh (forking) thì sự sao chép lại tất cả luồng là không cần thiết khi chương trình được xác định trong các tham số exec sẽ thay thế quá trình. Trong trường hợp này, việc sao chép lại chỉ gọi luồng hợp lý. Tuy nhiên, nếu quá trình riêng biệt này không gọi exec sau khi phân nhánh thì quá trình riêng biệt này nên sao chép lại tất cả luồng.

IV.5.2 Sự hủy bỏ luồng

Hủy một luồng là một tác vụ kết thúc một luồng trước khi nó hoàn thành.Thí dụ, nếu nhiều luồng đang tìm kiếm đồng thời thông qua một cơ sở dữ liệu và một luồng trả về kết quả, các luồng còn lại có thể bị hủy. Một trường hợp khác có thể xảy ra khi người dùng nhấn một nút trên trình duyệt web để dừng trang web đang được tải. Thường một trang web được tải trong một luồng riêng. Khi người dùng nhấn nút stop, luồng đang nạp trang bị hủy bỏ.

Một luồng bị hủy thường được xem như luồng đích. Sự hủy bỏ một luồng đích có thể xảy ra hai viễn cảnh khác nhau:

- **Hủy bất đồng bộ**: một luồng lập tức kết thúc luồng đích
- **Hủy trì hoãn**: luồng đích có thể kiểm tra định kỳ nếu nó sắp kết thúc, cho phép luồng đích một cơ hội tự kết thúc trong một cách có thứ tự.

Sự khó khăn của việc hủy này xảy ra trong những trường hợp khi tài nguyên được cấp phát tới một luồng bị hủy hay một luồng bị hủy trong khi việc cập nhật dữ liệu xảy ra giữa chừng, nó đang chia sẻ với các luồng khác. Điều này trở nên đặc biệt khó khăn với sự hủy bất đồng bộ. Hệ điều hành thường đòi lại tài nguyên hệ thống từ luồng bị hủy nhưng thường nó sẽ không đòi lại tất cả tài nguyên. Do đó, việc hủy một luồng bất đồng bộ có thể không giải phóng hết tài nguyên hệ thống cần thiết.

Một chọn lựa khác, sự hủy trì hoãn thực hiện bằng một luồng báo hiệu rằng một luồng đích bị hủy. Tuy nhiên, sự hủy sẽ xảy ra chỉ khi luồng đích kiểm tra để xác định nếu nó được hủy hay không. Điều này cho phép một luồng kiểm tra nếu nó sẽ bị hủy tại điểm nó có thể an toàn bị hủy. Pthreads gọi những điểm như thế là các điểm hủy (cancellation points).

Hầu hết hệ điều hành cho phép một quá trình hay một luồng bị hủy bất đồng bộ. Tuy nhiên, Pthread API cung cấp sự hủy trì hoãn. Điều này có nghĩa rằng một hệ điều hành cài đặt Pthread API sẽ cho phép sự hủy có trì hoãn.

IV.5.3 Tín hiệu quản lý

Một **tín hiệu** (signal) được dùng trong hệ điều hành UNIX thông báo một sự kiện xác định xảy ra. Một tín hiệu có thể được nhận hoặc đồng bộ hoặc bất đồng bộ phụ thuộc mã và lý do cho sự kiện đang được báo hiệu. Một tín hiệu hoặc đồng bộ hoặc bất đồng bộ đều theo sau cùng mẫu:

- Tín hiệu được phát sinh bởi sự xảy ra của một sự kiện xác định.
- Tín hiệu được phát sinh được phân phát tới một quá trình.
- Khi được phân phát xong, tín hiệu phải được quản lý.

Một thí dụ của tín hiệu đồng bộ gồm một truy xuất bộ nhớ không hợp lệ hay chia cho 0. Trong trường hợp này, nếu một chương trình đang chạy thực hiện một trong các hoạt động này, một tín hiệu được phát sinh. Các tín hiệu đồng bộ được phân phát tới cùng một quá trình thực hiện thao tác gây ra tín hiệu (do đó lý do chúng được xem đồng bộ).

Thuviendientu.org

Khi một tín hiệu được phát sinh bởi một sự kiện bên ngoài tới một quá trình đang chạy, quá trình đó nhận tín hiệu bất đồng bộ. Thí dụ, tín hiệu kết thúc quá trình với phím xác định (như <control><C>) hay thời gian hết hạn. Điển hình, một tín hiệu bất đồng bộ được gởi tới quá trình khác.

Mỗi tín hiệu có thể được quản lý bởi một trong hai bộ quản lý:

- Bộ quản lý tín hiệu mặc định
- Bộ quản lý tín hiệu được định nghĩa bởi người dùng

Mỗi tín hiệu có một bộ quản lý tín hiệu mặc định được thực thi bởi nhân khi quản lý tín hiệu. Hoạt động mặc định có thể được ghi đè bởi một hàm quản lý tín hiệu được định nghĩa bởi người dùng. Trong trường hợp này, hàm được định nghĩa bởi người dùng được gọi để quản lý tín hiệu hơn là hoạt động mặc định. Cả hai tín hiệu đồng bộ và bất đồng bộ có thể được quản lý trong các cách khác nhau. Một số tín hiệu có thể được bỏ qua (như thay đổi kích thước của cửa sổ); các tín hiệu khác có thể được quản lý bằng cách kết thúc chương trình (như truy xuất bộ nhớ không hợp lệ).

Quản lý tín hiệu trong những chương trình đơn luồng không phức tạp; các tín hiệu luôn được phân phát tới một quá trình. Tuy nhiên, phân phát tín hiệu là phức tạp hơn trong những chương trình đa luồng, như một quá trình có nhiều luồng. Một tín hiệu nên được phân phát ở đâu?

Thông thường, các tuỳ chọn sau tồn tại:

- Phân phát tín hiệu tới luồng mà tín hiệu áp dụng
- Phân phát tín hiệu tới mỗi luồng trong quá trình.
- Phân phát tín hiệu tới các luồng cụ thể trong quá trình.
- Gán một luồng xác định để nhận tất cả tín hiệu cho quá trình.

Phương pháp cho việc phân phát tín hiệu phụ thuộc vào loại tín hiệu được phát sinh. Thí dụ, các tín hiệu đồng bộ cần được phân phát tới luồng đã phát sinh ra tín hiệu và không phân phát tới luồng nào khác trong quá trình. Tuy nhiên, trường hợp với tín hiệu bất đồng bộ là không rõ ràng. Một số tín hiệu bất đồng bộ - như tín hiệu kết thúc một quá trình (thí dụ:<control><c>)- nên được gởi tới tất cả luồng. Một số ấn bản đa luồng của UNIX cho phép một luồng xác định tín hiệu nào sẽ được chấp nhận và tín hiệu nào sẽ bị khoá. Do đó, một vài tín hiệu bất đồng bộ có thể được phân phát tới chỉ các luồng không khoá tín hiệu. Tuy nhiên, vì tín hiệu cần được quản lý chỉ một lần, điển hình một tín hiệu được phân phát chỉ luồng đầu tiên được tìm thấy trong một luồng mà không nghẽn tín hiệu. Solaris 2 cài đặt bốn tuỳ chọn; nó tạo một luồng xác định trong mỗi quá trình cho quản lý tín hiệu. Khi một tín hiệu bất đồng bộ được gởi tới một quá trình, nó được gởi tới luồng xác định, sau đó nó phân phát tín hiệu tới luồng đầu tiên không khoá tín hiệu.

Mặc dù Windows 2000 không cung cấp rõ sự hỗ trợ tín hiệu, nhưng chúng có thể được mô phỏng sử dụng lời gọi thủ tục bất đồng bộ (asynchronous produce calls-APC). Tiện ích APC cho phép luồng người dùng xác định hàm được gọi khi luồng người dùng nhận thông báo về một sự kiện xác định. Như được hiển thị bởi tên của nó, một APC rất giống tín hiệu bất đồng bộ trong UNIX. Tuy nhiên, UNIX phải đấu tranh với cách giải quyết tín hiệu trong môi trường đa luồng, phương tiện APC phức tạp hơn như một APC được phân phát tới luồng xác định hơn quá trình.

Thuviendientu.org

IV.5.4 Nhóm luồng

Trong phần VI.3, chúng ta mô tả kịch bản đa luồng của một trình phục vụ web. Trong trường hợp này, bất cứ khi nào trình phục vụ nhận một yêu cầu, nó tạo một luồng riêng để phục vụ yêu cầu đó. Ngược lại, tạo một luồng riêng thật sự cao hơn tạo một quá trình riêng, dù sao một trình phục vụ đa luồng có thể phát sinh vấn đề. Quan tâm đầu tiên là lượng thời gian được yêu cầu để tạo luồng trước khi phục vụ yêu cầu, và lượng thời gian xoá luồng khi nó hoàn thành. Vấn đề thứ hai là vấn đề khó giải quyết hơn: nếu chúng ta cho phép tất cả yêu cầu đồng hành được phục vụ trong một luồng mới, chúng ta không thay thế giới hạn trên số lượng luồng hoạt động đồng hành trong hệ thống. Những luồng không giới hạn có thể làm cạn kiệt tài nguyên hệ thống, như thời gian CPU và bộ nhớ. Một giải pháp cho vấn đề này là sử dụng nhóm luồng.

Ý tưởng chung nằm sau nhóm luồng là tạo số lượng luồng tại thời điểm khởi động và đặt chúng vào nhóm, nơi chúng ngồi và chờ công việc. Khi một trình phục vụ nhận một yêu cầu, chúng đánh thức một luồng từ nhóm- nếu một luồng sẵn dùng – truyền nó yêu cầu dịch vụ. Một khi luồng hoàn thành dịch vụ của nó, nó trả về nhóm đang chờ công việc kế. Nếu nhóm không chứa luồng sẵn dùng, trình phục vụ chờ cho tới khi nó rảnh.

Nói cụ thể, các lợi ích của nhóm luồng là:

1) Thường phục vụ yêu cầu nhanh hơn với luồng đã có hơn là chờ để tạo luồng.
2) Một nhóm luồng bị giới hạn số lượng luồng tồn tại bất kỳ thời điểm nào. Điều này đặc biệt quan trọng trên những hệ thống không hỗ trợ số lượng lớn các luồng đồng hành.

Số lượng luồng trong nhóm có thể được đặt theo kinh nghiệm (heuristics) dựa trên các yếu tố như số CPU trong hệ thống, lượng bộ nhớ vật lý và số yêu cầu khách hàng đồng hành. Kiến trúc nhóm luồng tinh vi hơn có thể tự điều chỉnh số lượng luồng trong nhóm dựa theo các mẫu sử dụng. Những kiến trúc như thế cung cấp lợi điểm xa hơn của các nhóm luồng nhỏ hơn-do đó tiêu tốn ít bộ nhớ hơn-khi việc nạp trên hệ thống là chậm.

IV.5.5 Dữ liệu đặc tả luồng

Các luồng thuộc một quá trình chia sẻ dữ liệu của quá trình. Thật vậy, chia sẻ dữ liệu này cung cấp một trong những lợi điểm của lập trình đa luồng. Tuy nhiên, mỗi luồng có thể cần bản sao dữ liệu xác định của chính nó trong một vài trường hợp. Chúng ta sẽ gọi dữ liệu như thế là dữ liệu đặc tả luồng. Thí dụ, trong một hệ thống xử lý giao dịch, chúng ta có thể phục vụ mỗi giao dịch trong một luồng. Ngoài ra, mỗi giao dịch có thể được gán một danh biểu duy nhất. Để gán mỗi luồng với định danh duy nhất của nó chúng ta có thể dùng dữ liệu đặc tả dữ liệu. Hầu hết thư viện luồng gồm Win32 và Pthread – cung cấp một số biểu mẫu hỗ trợ cho dữ liệu đặc tả luồng. Java cũng cung cấp sự hỗ trợ như thế.

IV.6 Pthreads

Pthreads tham chiếu tới chuẩn POSIX (IEEE 1003.1c) định nghĩa API cho việc tạo và đồng bộ luồng. Đây là một đặc tả cho hành vi luồng không là một cài đặt.

Thuviendientu.org

Người thiết kế hệ điều hành có thể cài đặt đặc tả trong cách mà họ muốn. Thông thường, các thư viện cài đặt đặc tả Pthread bị giới hạn đối với các hệ thống dựa trên cơ sở của UNIX như Solaris 2. Hệ điều hành Windows thường không hỗ trợ Pthreads mặc dù các ấn bản shareware là sẵn dùng trong phạm vi công cộng.

Trong phần này chúng ta giới thiệu một số Pthread API như một thí dụ cho thư viện luồng cấp người dùng. Chúng ta sẽ xem nó như thư viện cấp người dùng vì không có mối quan hệ khác biệt giữa một luồng được tạo dùng Pthread và luồng được gắn với nhân. Chương trình C hiển thị trong hình dưới đây, mô tả một Pthread API cơ bản để xây dựng một chương trình đa luồng.

Chương trình hiển thị trong hình tạo một luồng riêng xác định tính tổng của một số nguyên không âm. Trong chương trình Pthread, các luồng riêng bắt đầu thực thi trong một hàm xác định. Trong hình, đây là một hàm runner. Khi chương trình này bắt đầu, một luồng riêng điều khiển bắt đầu trong main. Sau khi khởi tạo, main tạo ra luồng thứ hai bắt đầu điều khiển trong hàm runner.

Bây giờ chúng ta sẽ cung cấp tổng quan của chương trình này chi tiết hơn. Tất cả chương trình Pthread phải chứa tập tin tiêu đề pthread.h. pthread_t tid khai báo danh biểu cho luồng sẽ được tạo. Mỗi luồng có một tập các thuộc tính gồm kích thước ngăn xếp và thông tin định thời. Khai báo pthread_attr_t attr hiện diện các thuộc tính cho luồng. Chúng ta sẽ thiết lập các thuộc tính trong gọi hàm pthread_attr_init(&attr). Vì chúng ta không thiết lập rõ thuộc tính, chúng ta sẽ dùng thuộc tính mặc định được cung cấp. Một luồng riêng được tạo với lời gọi hàm pthread_create. Ngoài ra, để truyền định danh của luồng và các thuộc tính cho luồng, chúng ta cũng truyền tên của hàm, nơi một luồng mới sẽ bắt đầu thực thi, trong trường hợp này là hàm runner. Cuối cùng chúng ta sẽ truyền số nguyên được cung cấp tại dòng lệnh, argv[1].

Tại điểm này, chương trình có hai luồng: luồng khởi tạo trong main và luồng thực hiện việc tính tổng trong hàm runner. Sau khi tạo luồng thứ hai, luồng main sẽ chờ cho luồng runner hoàn thành bằng cách gọi hàm pthread_join. Luồng runner sẽ hoàn thành khi nó gọi hàm pthread_exit.

```
#include<pthread>
#include<stdio.h>
int sum: /*Dữ liệu này được chia sẻ bởi thread(s)*/
void *runner(void *param); /*luồng*/

main(int argc, char *argv[])
{
        pthread_t tid; /*định danh của luồng*/
        pthread_attr_t attr; /*tập hợp các thuộc tính*/
        if(argc !=2){
        fprintf(stderr, "usage: a.out <integer value>");
        exit();
}
if (atoi(argv[1] < 0)){
        fprintf(stderr,"%d must be >= 0 \n", atoi(argv[1]));
        exit();
}
/*lấy các thuộc tính mặc định*/
pthread_attr_init(&attr);
/*tạo một luồng*/
pthread_create(&tid,&attr,runner, argv[1]);
```

Thuviendientu.org

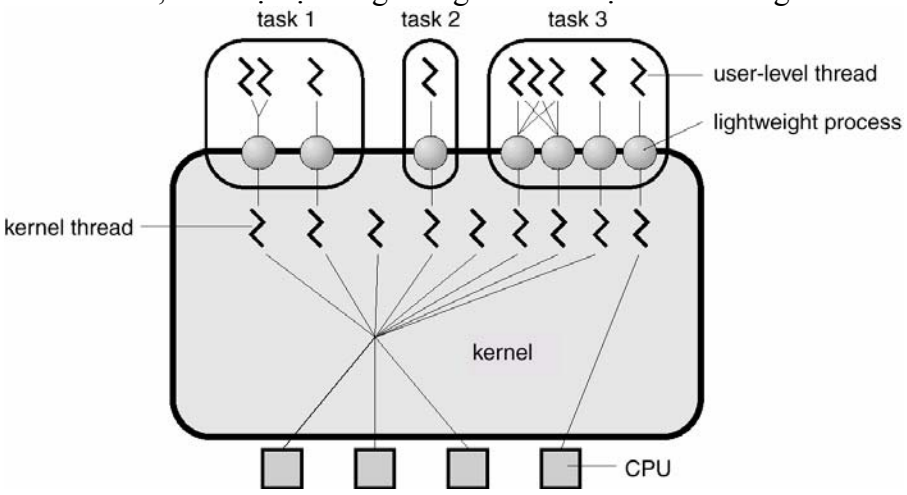

```
/*bây giờ chờ luồng kết thúc*/
pthread_join(tid,NULL);
printf("sum = %d\n",sum);
/*Luồng sẽ bắt đầu điều khiển trong hàm này*/
void *runner(void *param)
{
        int upper = atoi(param);
        int i;
        sum = 0;
        if (upper > 0){
        sum+= i;
}
pthread_exit(0);
}
}
```

Hình 0-5-Chương trình C đa luồng dùng Pthread API

IV.7 Luồng Solaris 2

Solaris 2 là một ấn bản của UNIX với hỗ trợ luồng tại cấp độ nhân và cấp độ người dùng, đa xử lý đối xứng (SMP) và định thời thời thực. Solaris 2 cài đặt Pthread API hỗ trợ luồng cấp người dùng với thư viện chứa APIs cho việc tạo và quản lý luồng (được gọi luồng UI). Sự khác nhau giữa hai thư viện này rất lớn, mặc dù hầu hết người phát triển hiện nay chọn thư viện Pthread. Solaris 2 cũng định nghĩa một cấp độ luồng trung gian. Giữa luồng cấp nhân và cấp người dùng là các quá trình nhẹ (lightweight process- LWPs). Mỗi quá trình chứa ít nhất một LWP. Thư viện luồng đa hợp luồng người dùng trên nhóm LWP cho quá trình và chỉ luồng cấp người dùng hiện được nối kết tới một LWP hoàn thành công việc. Các luồng còn lại bị khoá hoặc chờ cho một LWP mà chúng có thể thực thi trên nó.

Luồng cấp nhân chuẩn thực thi tất cả thao tác trong nhân. Mỗi LWP có một luồng cấp nhân, và một số luồng cấp nhân (kernel) chạy trên một phần của nhân và không có LWP kèm theo (thí dụ, một luồng phục vụ yêu cầu đĩa). Các luồng cấp nhân chỉ là những đối tượng được định thời trong hệ thống. Solaris 2 cài mô hình nhiều-nhiều; toàn bộ hệ thống luồng của nó được mô tả trong hình dưới đây:



Hình 0-6-Luồng Solaris 2

Thuviendientu.org

Các luồng cấp người dùng có thể giới hạn hay không giới hạn. Một luồng cấp người dùng giới hạn được gán vĩnh viễn tới một LWP. Chỉ luồng đó chạy trên LWP và yêu cầu LWP có thể được tận hiến tới một bộ xử lý đơn (xem luồng trái nhất trong hình trên). Liên kết một luồng có ích trong trường hợp yêu cầu thời gian đáp ứng nhanh, như ứng dụng thời thực. Một luồng không giới hạn gán vĩnh viễn tới bất kỳ LWP nào. Tất cả các luồng không giới hạn được đa hợp trong một nhóm cac LWP sẵn dùng cho ứng dụng. Các luồng không giới hạn là mặc định. Solaris 8 cũng cung cấp một thư viện luồng thay đổi mà mặc định chúng liên kết tới tất cả các luồng với một LWP.

Xem xét hệ thống trong hoạt động: bất cứ một quá trình nào có thể có nhiều luồng người dùng. Các luồng cấp người dùng này có thể được định thời và chuyển đổi giữa LWPs bởi thư viện luồng không có sự can thiệp của nhân. Các luồng cấp người dùng cực kỳ hiệu quả vì không có sự hỗ trợ nhân được yêu cầu cho việc tạo hay huỷ, hay thư viện luồng chuyển ngữ cảnh từ luồng người dùng này sang luồng khác.

Mỗi LWP được nối kết tới chính xác một luồng cấp nhân, ngược lại mỗi luồng cấp người dùng là độc lập với nhân. Nhiều LWPs có thể ở trong một quá trình, nhưng chúng được yêu cầu chỉ khi luồng cần giao tiếp với một nhân. Thí dụ, một LWP được yêu cầu mỗi luồng có thể khoá đồng hành trong lời gọi hệ thống. Xem xét năm tập tin khác nhau-đọc các yêu cầu xảy ra cùng một lúc. Sau đó, năm LWPs được yêu cầu vì chúng đang chờ hoàn thành nhập/xuất trong nhân. Nếu một tác vụ chỉ có bốn LWPs thì yêu cầu thứ năm sẽ không phải chờ một trong những LWPs để trả về từ nhân. Bổ sung một LWP thứ sáu sẽ không đạt được gì nếu chỉ có đủ công việc cho năm.

Các luồng nhân được định thời bởi bộ lập thời biểu của nhân và thực thi trên một hay nhiều CPU trong hệ thống. Nếu một luồng nhân khoá (trong khi chờ một thao tác nhập/xuất hoàn thành), thì bộ xử lý rảnh để thực thi luồng nhân khác. Nếu một luồng bị khoá đang chạy trên một phần của LWP thì LWP cũng khoá. Ở trên vòng, luồng cấp người dùng hiện được gán tới LWP cũng bị khoá. Nếu một quá trình có nhiều hơn một LWP thì nhân có thể định thời một LWP khác.

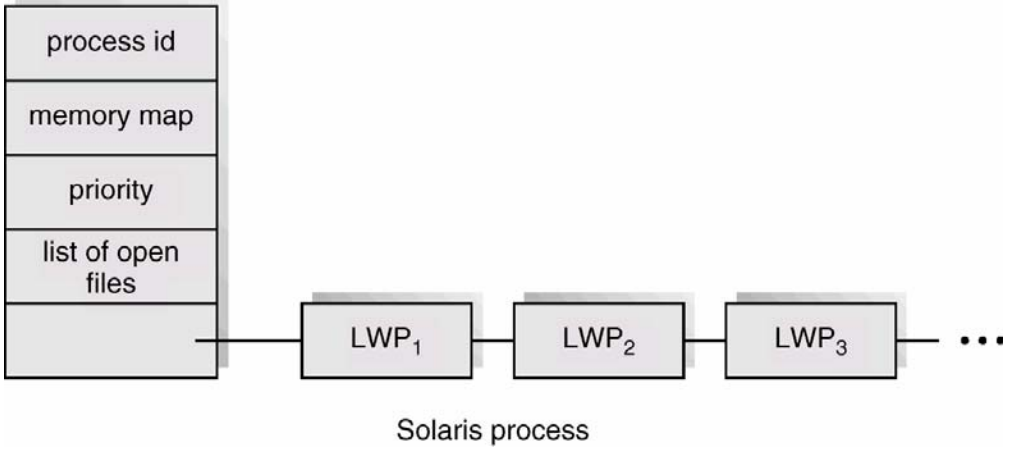
Thư viện luồng tự động thay đổi số lượng LWPs trong nhóm để đảm bảo năng lực thực hiện tốt nhất cho ứng dụng. Thí dụ, nếu tất cả LWPs trong một quá trình bị khoá bởi những luồng có thể chạy thì thư viện tự tạo một LWP khác được gán tới một luồng đang chờ. Do đó, một chương trình được ngăn chặn từ một chương trình khác bởi sự nghèo nàn của những LWPs không bị khoá. LWPs là những tài nguyên nhân đắt để duy trì nếu chúng không được dùng. Thư viện luồng “ages” LWPs và xoá chúng khi chúng không được dùng cho khoảng thời gian dài, điển hình khoảng 5 phút. Các nhà phát triển dùng những cấu trúc dữ liệu cài đặt luồng trên Solaris 2:

- Luồng cấp người dùng chứa một luồng ID; tập thanh ghi (gồm một bộ đếm chương trình và con trỏ ngăn xếp); ngăn xếp; và độ ưu tiên (được dùng bởi thư viện cho mục đích định thời). Không có cấu trúc dữ liệu nào là tài nguyên nhân; tất cả chúng tồn tại trong không gian người dùng.
- Một LWP có một tập thanh ghi cho luồng cấp nhân nó đang chạy cũng như bộ nhớ và thông tin tính toán. Một LWP là một cấu trúc dữ liệu nhân và nó nằm trong không gian nhân
- Một luồng nhân chỉ có một cấu trúc dữ liệu nhân và ngăn xếp. Cấu trúc dữ liệu gồm bản sao các thanh ghi nhân, con trỏ tới LWP mà nó được gán, độ ưu tiên và thông tin định thời.

Mỗi quá trình trong Solaris 2 gồm nhiều thông tin được mô tả trong khối điều khiển quá trình (Process Control Block-PCB). Trong thực tế, một quá trình Solaris 2

Thuviendientu.org

chứa một định danh quá trình (Process ID-PID); bản đồ bộ nhớ; danh sách các tập tin đang mở, độ ưu tiên; và con trỏ của các luồng nhân vơi quá trình (Hình).



Hình 0-7-Quá trình Solaris 2

IV.8 Luồng Windows 2000

Windows 2000 cài đặt Win32 API. Win32 API là một API chủ yếu cho họ hệ điều hành Windows (Windows 95/98/NT và Windows 2000). Thực vậy, những gì được đề cập trong phần phần này áp dụng tới họ hệ điều hành này

Ứng dụng Windows chạy như một quá trình riêng rẻ nơi mỗi quá trình có thể chứa một hay nhiều luồng. Windows 2000 dùng ánh xạ một-một nơi mà mỗi luồng cấp người dùng ánh xạ tới luồng nhân được liên kết tới.Tuy nhiên, Windows cũng cung cấp sự hỗ trợ cho một thư viện có cấu trúc (fiber library) cung cấp chức năng của mô hình nhiều-nhiều. Mỗi luồng thuộc về một quá trình có thể truy xuất một không gian địa chỉ ảo của quá trình.

Những thành phần thông thường của một luồng gồm:

- ID của luồng định danh duy nhất luồng
- Tập thanh ghi biểu diễn trạng thái của bộ xử lý
- Ngăn xếp người dùng khi luồng đang chạy ở chế độ người dùng. Tương tự, mỗi luồng cũng có một ngăn xếp nhân được dùng khi luồng đang chạy trong chế độ nhân
- Một vùng lưu trữ riêng được dùng bởi nhiều thư viện thời gian thực và thự viện liên kết động (DLLs).

Tập thanh ghi, ngăn xếp và vùng lưu trữ riêng được xem như ngữ cảnh của luồng và được đặc tả kiến trúc tới phần cứng mà hệ điều hành chạy trên đó. Cấu trúc dữ liệu chủ yếu của luồng gồm:

- RTHREAD (executive thread block-khối luồng thực thi).
- KTHREAD (kernel thread-khối luồng nhân)
- TEB (thread environment block-khối môi trường luồng)

Các thành phần chủ yếu của RTHREAD gồm một con trỏ chỉ tới quá trình nào luồng thuộc về và địa chỉ của thủ tục mà luồng bắt đầu điều khiển trong đó. ETHREAD cũng chứa một con trỏ chỉ tới KTHREAD tương ứng.

Thuviendientu.org

KTHREAD gồm thông tin định thời và đồng bộ hóa cho luồng. Ngoài ra, KTHREAD chứa ngăn xếp nhân (được dùng khi luồng đang chạy trong chế độ nhân) và con trỏ chỉ tới TEB.

ETHREAD và KTHREAD tồn tại hoàn toàn ở không gian nhân; điều này có nghĩa chỉ nhân có thể truy xuất chúng. TEB là cấu trúc dữ liệu trong không gian người dùng được truy xuất khi luồng đang chạy ở chế độ người dùng. Giữa những trường khác nhau, TEB chứa ngăn xếp người dùng và một mảng cho dữ liệu đặc tả luồng (mà Windows gọi là lưu trữ cục bộ luồng)

IV.9 Luồng Linux

Nhân Linux được giới thiệu trong ấn bản 2.2. Linux cung cấp một lời gọi hệ thống *fork* với chức năng truyền thống là tạo bản sao một quá trình. Linux cũng cung cấp lời gọi hệ thống *clone* mà nó tương tự như tạo một luồng. c*lone* có hành vi rất giống như *fork*, ngoại trừ thay vì tạo một bản sao của quá trình gọi, nó tạo một quá trình riêng chia sẻ không gian địa chỉ của quá trình gọi. Nó chấm dứt việc chia sẻ không gian địa chỉ của quá trình cha mà một tác vụ được nhân bản đối xử giống rất nhiều một luồng riêng rẻ.

Chia sẻ không gian địa chỉ được cho phép vì việc biểu diễn của một quá trình trong nhân Linux. Một cấu trúc dữ liệu nhân duy nhất tồn tại cho mỗi quá trình trong hệ thống. Một cấu trúc dữ liệu nhân duy nhất tồn tại cho mỗi quá trình trong hệ thống. Tuy nhiên, tốt hơn lưu trữ dữ liệu cho mỗi quá trình trong cấu trúc dữ liệu là nó chứa các con trỏ chỉ tới các cấu trúc dữ liệu khác nơi dữ liệu này được được lưu. Thí dụ, cấu trúc dữ liệu trên quá trình chứa các con trỏ chỉ tới các cấu trúc dữ liệu khác hiện diện danh sách tập tin đang mở, thông tin quản lý tín hiệu, và bộ nhớ ảo. Khi fork được gọi, một quá trình mới được tạo cùng với một bản sao của tất cả cấu trúc dữ liệu của quá trình cha được liên kết tới. Khi lời gọi hệ thống *clone* được thực hiện, một quá trình mới chỉ tới cấu trúc dữ liệu của quá trình cha, do đó cho phép quá trình con chia sẻ bộ nhớ và tài nguyên của quá trình cha. Một tập hợp cờ được truyền như một tham số tới lời gọi hệ thống *clone*. Tập hợp cờ này được dùng để hiển thị bao nhiêu quá trình cha được chia sẻ với quá trình con. Nếu không có cờ nào được đặt, không có chia sẻ xảy ra và clone hoạt động giống như fork. Nếu tất cả năm cờ được đặt, quá trình con chia sẻ mọi thứ với quá trình cha. Sự kết hợp khác của cờ cho phép các cấp độ chia sẻ khác nhau giữa hai mức độ cao nhất này.

Điều thú vị là Linux không phân biệt giữa quá trình và luồng. Thật vậy, Linux thường sử dụng thuật ngữ tác vụ-hơn là quá trình hay luồng-khi tham chiếu tới dòng điều khiển trong chương trình. Ngoài quá trình được nhân bản, Linux không hỗ trợ đa luồng, cấu trúc dữ liệu riêng hay thủ tục nhân. Tuy nhiên, những cài đặt Pthreads là sẵn dùng cho đa luồng cấp người dùng.

IV.10 Luồng Java

Như chúng ta đã thấy, hỗ trợ cho luồng có thể được cung cấp tại cấp người dùng với một thư viện như Pthread. Hơn nữa, hầu hết hệ điều hành cung cấp sự hỗ trợ cho luồng tại cấp nhân. Java là một trong số nhỏ ngôn ngữ cung cấp sự hỗ trợ tại cấp ngôn ngữ cho việc tạo và quản lý luồng. Tuy nhiên, vì các luồng được quản lý bởi máy ảo Java, không bởi một thư viện cấp người dùng hay nhân, rất khó để phân cấp luồng Java như cấp độ người dùng hay cấp độ nhân. Trong phần này chúng ta trình bày các luồng Java như một thay đổi đối với mô hình người dùng nghiêm ngặt hay mô hình

Thuviendientu.org

cấp nhân. Phần sau chúng ta sẽ thảo luận một luồng Java có thể được ánh xạ tới luồng nhân bên dưới như thế nào.

Tất cả chương trình tạo ít nhất một luồng điều khiển đơn. Thậm chí một chương trình Java chứa chỉ một phương thức main chạy như một luồng đơn trong máy ảo Java. Ngoài ra, Java cung cấp các lệnh cho phép người phát triển tạo và thao tác các luồng điều khiển bổ sung trong chương trình.

IV.10.1 Tạo luồng

Một cách để tạo một luồng rõ ràng là tạo một lớp mới được phát sinh từ lớp Thread và viết đè phương thức run của lớp Thread. Tiếp cận này được hiển thị trong hình sau, ấn bản Java của chương trình đa luồng xác định tổng các số nguyên không âm.

Một đối tượng của lớp phát sinh sẽ chạy như một luồng điều khiển đơn trong máy ảo Java. Tuy nhiên, tạo một đối tượng được phát sinh từ lớp Thread không tạo một luồng mới, trái lại phương thức start mới thật sự tạo luồng mới. Gọi phương thức start cho đối tượng mới thực hiện hai thứ:

1) Nó cấp phát bộ nhớ và khởi tạo một luồng mới trong máy ảo Java.
2) Nó gọi phương thức run, thực hiện luồng thích hợp để được chạy bởi máy ảo Java. (Chú ý, không thể gọi phương thức run trực tiếp, gọi phương thức start sẽ gọi phương thức run)

```
Class Summation extends Thread
{
        public Summation (int n){
                upper   = n;
        }
        public void run(){
                int sum = 0;

                if (upper>0){
                        for(int i = 1; i<= upper; i++){
                                sum+=i;
                }
                System.out.println("The sum of "+upper+ " is " + sum);
        }
        private int upper;
}

public class ThreadTester
{
        public static void main(String args[]){
                if(args.length>0){
                        if(Integer.parseInt(args[0])<0)
                                System.err.println(args[0] + " must be >= 0.");
                        else{
                                Summation thrd = new Summation
(Integer.parseInt(args[0]));
                                Thrd.start();
                        }
```

Thuviendientu.org


```
                }
                Else
                        System.out.println("Usage: summation < integer value");
        }
}
```

Hình 0-8- Chương trình Java để tính tổng số nguyên không âm

Khi chương trình tính tổng thực thi, hai luồng được tạo bởi JVM. Luồng đầu tiên là luồng được nối kết với ứng dụng-luồng này bắt đầu thực thi tại phương thức main. Luồng thứ hai là luồng Summation được tạo rõ ràng với phương thức start. Luồng Summation bắt đầu thực thi trong phương thức run của nó. Luồng kết thúc khi nó thoát khỏi phương thức run của nó.

IV.10.2 JVM và hệ điều hành chủ

Cài đặt điển hình của JVM ở trên đỉnh của hệ điều hành chủ (host operating system). Thiết lập này cho phép JVM che giấu chi tiết cài đặt của hệ điều hành bên dưới và cung cấp môi trường không đổi, trừu tượng cho phép chương trình Java hoạt động trên bất kỳ phần cứng nào hỗ trợ JVM. Đặc tả cho JVM không hiển thị các luồng Java được ánh xạ tới hệ điều hành bên dưới như thế nào để thay thế việc quên đi việc quyết định cài đặt cụ thể của JVM. Windows 95/98/NT và Windows 2000 dùng mô hình một-một; do đó, mỗi luồng Java cho một JVM chạy trên các hệ điều hành này ánh xạ tới một luồng nhân. Solaris 2 khởi đầu cài đặt JVM dùng mô hình nhiều-một. Tuy nhiên, ấn bản 1.1 của JVM với Solaris 2.6 được cài đặt dùng mô hình nhiều-nhiều.

IV.11 Tóm tắt

Luồng là một dòng điều khiển trong phạm vi một quá trình. Quá trình đa luồng gồm nhiều dòng điều khiển khác nhau trong cùng không gian địa chỉ. Những lợi điểm của đa luồng gồm đáp ứng nhanh đối với người dùng, chia sẻ tài nguyên trong quá trình, tính kinh tế, và khả năng thuận lợi trong kiến trúc đa xử lý.

Luồng cấp người dùng là các luồng được nhìn thấy bởi người lập trình và không được biết bởi nhân. Thư viện luồng trong không gian người dùng điển hình quản lý luồng cấp người dùng. Nhân của hệ điều hành hỗ trợ và quản lý các luồng cấp nhân. Thông thường, luồng cấp người dùng nhanh hơn luồng cấp nhân trong việc tạo và quản lý. Có ba loại mô hình khác nhau liên quan đến luồng cấp người dùng và luồng cấp nhân. Mô hình nhiều-một ánh xạ nhiều luồng người dùng tới một luồng nhân. Mô hình một-một ánh xạ mỗi luồng người dùng tới một luồng nhân tương ứng. Mô hình nhiều-nhiều đa hợp nhiều luồng người dùng tới một số lượng nhỏ hơn hay bằng luồng nhân.

Những chương trình đa luồng giới thiệu nhiều thử thách cho việc lập trình, gồm ngữ nghĩa của lời gọi hệ thống fork và exec. Những vấn đề khác gồm huỷ bỏ luồng, quản lý tín hiệu, và dữ liệu đặc tả luồng. Nhiều hệ điều hành hiện đại cung cấp nhân hỗ trợ luồng như Windows NT, Windows 2000, Solaris 2 và Linux. Pthread API cung cấp tập hợp các hàm để tạo và quản lý luồng tại cấp người dùng. Java cung cấp một API tương tự cho việc hỗ trợ luồng. Tuy nhiên, vì các luồng Java được quản lý bởi JVM và không phải thư viện luồng cấp người dùng hay nhân, chúng không rơi vào loại luồng người dùng hay nhân.

Thuviendientu.org

ĐỊNH THỜI BIỂU CPU

I Mục tiêu

Sau khi học xong chương này, người học nắm được những kiến thức sau:

- Hiểu các khái niệm cơ bản về định thời
- Hiểu các giải thuật định thời biểu CPU
- Vận dụng một giải thuật định thời cho một hệ thống cụ thể

II Giới thiệu

Định thời biểu là cơ sở của các hệ điều hành đa chương. Bằng cách chuyển đổi CPU giữa các quá trình, hệ điều hành có thể làm máy tính hoạt động nhiều hơn. Trong chương này, chúng ta giới thiệu các khái niệm định thời cơ bản và trình bày các giải thuật định thời biểu CPU khác nhau. Chúng ta cũng xem xét vấn đề chọn một giải thuật cho một hệ thống xác định.

III Các khái niệm cơ bản

Mục tiêu của đa chương là có nhiều quá trình chạy cùng thời điểm để tối ưu hóa việc sử dụng CPU. Trong hệ thống đơn xử lý, chỉ một quá trình có thể chạy tại một thời điểm; bất cứ quá trình nào khác đều phải chờ cho đến khi CPU rảnh và có thể được định thời lại.

Ý tưởng của đa chương là tương đối đơn giản. Một quá trình được thực thi cho đến khi nó phải chờ yêu cầu nhập/xuất hoàn thành. Trong một hệ thống máy tính đơn giản thì CPU sẽ rảnh rỗi; tất cả thời gian chờ này là lãng phí. Với đa chương, chúng ta cố gắng dùng thời gian này để CPU có thể phục vụ cho các quá trình khác. Nhiều quá trình được giữ trong bộ nhớ tại cùng thời điểm. Khi một quá trình phải chờ, hệ điều hành lấy CPU từ quá trình này và cấp CPU tới quá trình khác.

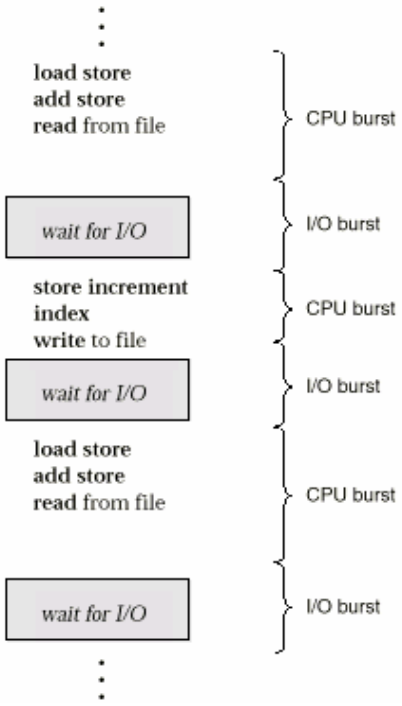
Định thời biểu là chức năng cơ bản của hệ điều hành. Hầu hết tài nguyên máy tính được định thời biểu trước khi dùng. Dĩ nhiên, CPU là một trong những tài nguyên máy tính ưu tiên. Do đó, định thời biểu là trọng tâm trong việc thiết kế hệ điều hành.

III.1 Chu kỳ CPU-I/O

Sự thành công của việc định thời biểu CPU phụ thuộc vào thuộc tính được xem xét sau đây của quá trình. Việc thực thi quá trình chứa một **chu kỳ** (cycle) thực thi CPU và chờ đợi nhập/xuất. Các quá trình chuyển đổi giữa hai trạng thái này. Sự thực thi quá trình bắt đầu với một **chu kỳ CPU** (CPU burst), theo sau bởi một **chu kỳ nhập/xuất** (I/O burst), sau đó một chu kỳ CPU khác, sau đó lại tới một chu kỳ nhập/xuất khác khác,..Sau cùng, chu kỳ CPU cuối cùng sẽ kết thúc với một yêu cầu hệ thống để kết thúc việc thực thi, hơn là với một chu kỳ nhập/xuất khác, được mô tả như hình IV.1. Một chương trình **hướng nhập/xuất** (I/O-bound) thường có nhiều chu kỳ CPU ngắn. Một chương trình **hướng xử lý** (CPU-bound) có thể có một nhiều chu

Thuviendientu.org

kỳ CPU dài. Sự phân bổ này có thể giúp chúng ta chọn giải thuật định thời CPU hợp lý.



Hình 0-1-Thay đổi thứ tự của CPU và I/O burst

III.2 Bộ định thời CPU

Bất cứ khi nào CPU rảnh, hệ điều hành phải chọn một trong những quá trình trong hàng đợi sẵn sàng để thực thi. Chọn quá trình được thực hiện bởi **bộ định thời biểu ngắn** (short-term scheduler) hay bộ định thời CPU. Bộ định thời này chọn các quá trình trong bộ nhớ sẵn sàng thực thi và cấp phát CPU tới một trong các quá trình đó.

Hàng đợi sẵn sàng không nhất thiết là hàng đợi vào trước, ra trước (FIFO). Xem xét một số giải thuật định thời khác nhau, một hàng đợi sẵn sàng có thể được cài đặt như một hàng đợi FIFO, một hàng đợi ưu tiên, một cây, hay đơn giản là một danh sách liên kết không thứ tự. Tuy nhiên, về khái niệm tất cả các quá trình trong hàng đợi sẵn sàng được xếp hàng chờ cơ hội để chạy trên CPU. Các mẫu tin trong hàng đợi thường là khối điều khiển quá trình của quá trình đó.

III.3 Định thời biểu trưng dụng

Quyết định định thời biểu CPU có thể xảy ra một trong 4 trường hợp sau:

- Khi một quá trình chuyển từ trạng thái chạy sang trạng thái chờ (thí dụ: yêu cầu nhập/xuất, hay chờ kết thúc của một trong những quá trình con).
- Khi một quá trình chuyển từ trạng thái chạy tới trạng thái sẵn sàng (thí dụ: khi một ngắt xảy ra)
- Khi một quá trình chuyển từ trạng thái chờ tới trạng thái sẵn sàng (thí dụ: hoàn thành nhập/xuất)

Thuviendientu.org

- Khi một quá trình kết thúc

Trong trường hợp 1 và 4, không cần chọn lựa loại định thời biểu. Một quá trình mới (nếu tồn tại trong hàng đợi sẵn sàng) phải được chọn để thực thi. Tuy nhiên, có sự lựa chọn loại định thời biểu trong trường hợp 2 và 3.

Khi định thời biểu xảy ra chỉ trong trường hợp 1 và 4, chúng ta nói cơ chế **định thời không trưng dụng** (nonpreemptive); ngược lại, khi định thời biểu xảy ra chỉ trong trường hợp 2 và 3, chúng ta nói cơ chế **định thời trưng dụng** (preemptive). Trong định thời không trưng dụng, một khi CPU được cấp phát tới một quá trình, quá trình giữ CPU cho tới khi nó giải phóng CPU hay bởi kết thúc hay bởi chuyển tới trạng thái sẵn sàng. Phương pháp định thời biểu này được dùng bởi các hệ điều hành Microsoft Windows 3.1 và bởi Apple Macintosh. Phương pháp này chỉ có thể được dùng trên các nền tảng phần cứng xác định vì nó không đòi hỏi phần cứng đặc biệt (thí dụ, một bộ đếm thời gian) được yêu cầu để định thời biểu trưng dụng.

Tuy nhiên, định thời trưng dụng sinh ra một chi phí. Xét trường hợp 2 quá trình chia sẻ dữ liệu. Một quá trình có thể ở giữa giai đoạn cập nhật dữ liệu thì nó bị chiếm dụng CPU và một quá trình thứ hai đang chạy. Quá trình thứ hai có thể đọc dữ liệu mà nó hiện đang ở trong trạng thái thay đổi. Do đó, những kỹ thuật mới được yêu cầu để điều phối việc truy xuất tới dữ liệu được chia sẻ.

Sự trưng dụng cũng có một ảnh hưởng trong thiết kế nhân hệ điều hành. Trong khi xử lý lời gọi hệ thống, nhân có thể chờ một hoạt động dựa theo hành vi của quá trình. Những hoạt động như thế có thể liên quan với sự thay đổi dữ liệu nhân quan trọng (thí dụ: các hàng đợi nhập/xuất). Điều gì xảy ra nếu quá trình bị trưng dụng CPU ở trong giai đoạn thay đổi này và nhân (hay trình điều khiển thiết bị) cần đọc hay sửa đổi cùng cấu trúc? Sự lộn xộn chắc chắn xảy ra. Một số hệ điều hành, gồm hầu hết các ấn bản của UNIX, giải quyết vấn đề này bằng cách chờ lời gọi hệ thống hoàn thành hay việc nhập/xuất bị nghẽn, trước khi chuyển đổi ngữ cảnh. Cơ chế này đảm bảo rằng cấu trúc nhân là đơn giản vì nhân sẽ không trưng dụng một quá trình trong khi các cấu trúc dữ liệu nhân ở trong trạng thái thay đổi. Tuy nhiên, mô hình thực thi nhân này là mô hình nghèo nàn để hỗ trợ tính toán thời thực và đa xử lý.

Trong trường hợp UNIX, các phần mã vẫn là sự rủi ro. Vì các ngắt có thể xảy ra bất cứ lúc nào và vì các ngắt này không thể luôn được bỏ qua bởi nhân, nên phần mã bị ảnh hưởng bởi ngắt phải được đảm bảo từ việc sử dụng đồng thời. Hệ điều hành cần chấp nhận hầu hết các ngắt, ngược lại dữ liệu nhập có thể bị mất hay dữ liệu xuất bị viết chồng. Vì thế các phần mã này không thể được truy xuất đồng hành bởi nhiều quá trình, chúng vô hiệu hóa ngắt tại lúc nhập và cho phép các ngắt hoạt động trở lại tại thời điểm việc nhập kết thúc. Tuy nhiên, vô hiệu hóa và cho phép ngắt tiêu tốn thời gian, đặc biệt trên các hệ thống đa xử lý.

III.4 Bộ phân phát

Một thành phần khác liên quan đến chức năng định thời biểu CPU là **bộ phân phát** (dispatcher). Bộ phân phát là một module có nhiệm vụ trao điều khiển CPU tới quá trình được chọn bởi bộ định thời biểu ngắn (short-term scheduler). Chức năng này liên quan:

- Chuyển ngữ cảnh
- Chuyển chế độ người dùng
- Nhảy tới vị trí hợp lý trong chương trình người dùng để khởi động lại quá trình

Thuviendientu.org

Bộ phân phát nên nhanh nhất có thể, và nó được nạp trong mỗi lần chuyển quá trình. Thời gian mất cho bộ phân phát dừng một quá trình này và bắt đầu chạy một quá trình khác được gọi là thời gian trễ cho việc điều phối (dispatch latency).

IV Các tiêu chuẩn định thời

Các giải thuật định thời khác nhau có các thuộc tính khác nhau và có xu hướng thiên vị cho một loại quá trình hơn một quá trình. Trong việc chọn giải thuật nào sử dụng trong trường hợp nào, chúng ta phải xét các thuộc tính của các giải thuật khác nhau.

Nhiều tiêu chuẩn được đề nghị để so sánh các giải thuật định thời biểu. Những đặc điểm được dùng để so sánh có thể tạo sự khác biệt quan trọng trong việc xác định giải thuật tốt nhất. Các tiêu chuẩn gồm:

- **Việc sử dụng CPU**: chúng ta muốn giữ CPU bận nhiều nhất có thể. Việc sử dụng CPU có thể từ 0 đến 100%. Trong hệ thống thực, nó nên nằm trong khoảng từ 40% (cho hệ thống được nạp tải nhẹ) tới 90% (cho hệ thống được nạp tải nặng).
- **Thông lượng**: nếu CPU bận thực thi các quá trình thì công việc đang được thực hiện. Thước đo của công việc là số lượng quá trình được hoàn thành trên một đơn vị thời gian gọi là **thông lượng** (throughput). Đối với các quá trình dài, tỉ lệ này có thể là 1 quá trình trên 1 giờ; đối với các giao dịch ngắn, thông lượng có thể là 10 quá trình trên giây.
- **Thời gian hoàn thành**: từ quan điểm của một quá trình cụ thể, tiêu chuẩn quan trọng là mất bao lâu để thực thi quá trình đó. Khoảng thời gian từ thời điểm gởi quá trình tới khi quá trình hoàn thành được gọi là **thời gian hoàn thành** (turnaround time). Thời gian hoàn thành là tổng các thời gian chờ đưa quá trình vào bộ nhớ, chờ hàng đợi sẵn sàng, thực thi CPU và thực hiện nhập/xuất.
- **Thời gian chờ**: giải thuật định thời CPU không ảnh hưởng lượng thời gian quá trình thực thi hay thực hiện nhập/xuất; nó ảnh hưởng chỉ lượng thời gian một quá trình phải chờ trong hàng đợi sẵn sàng. **Thời gian chờ** (waiting time) là tổng thời gian chờ trong hàng đợi sẵn sàng.
- **Thời gian đáp ứng**: trong một hệ thống giao tiếp, thời gian hoàn thành không là tiêu chuẩn tốt nhất. Thông thường, một quá trình có thể tạo ra dữ liệu xuất tương đối sớm và có thể tiếp tục tính toán các kết quả mới trong khi các kết quả trước đó đang được xuất cho người dùng. Do đó, một thước đo khác là thời gian từ lúc gởi yêu cầu cho tới khi đáp ứng đầu tiên được tạo ra. Thước đo này được gọi là **thời gian đáp ứng** (response time), là lượng thời gian mất đi từ lúc bắt đầu đáp ứng nhưng không là thời gian mất đi để xuất ra đáp ứng đó. Thời gian hoàn thành thường bị giới hạn bởi tốc độ của thiết bị xuất.

Chúng ta muốn tối ưu hóa việc sử dụng CPU và thông lượng, đồng thời tối thiểu hóa thời gian hoàn thành, thời gian chờ, và thời gian đáp ứng. Trong hầu hết các trường hợp, chúng ta tối ưu hóa thước đo trung bình. Tuy nhiên, trong một vài trường hợp chúng ta muốn tối ưu giá trị tối thiểu hay giá trị tối đa hơn là giá trị trung bình. Thí dụ, để đảm bảo rằng tất cả người dùng nhận dịch vụ tốt, chúng ta muốn tối thiểu thời gian đáp ứng tối đa.

Thuviendientu.org

Đối với những hệ thống tương tác (như các hệ thống chia thời), một số nhà phân tích đề nghị rằng sự thay đổi trong thời gian đáp ứng quan trọng hơn tối thiểu hóa thời gian đáp ứng trung bình. Một hệ thống với thời gian đáp ứng phù hợp và có thể đoán trước được quan tâm nhiều hơn hệ thống chạy nhanh hơn mức trung bình nhưng biến đổi cao. Tuy nhiên, gần như không có công việc nào được thực hiện trên các giải thuật định thời biểu CPU để tối thiểu hóa các thay đổi.

Khi chúng ta thảo luận các giải thuật định thời biểu CPU khác nhau, chúng ta muốn hiển thị các hoạt động của chúng. Một hình ảnh chính xác nên thông báo tới nhiều quá trình, mỗi quá trình là một chuỗi của hàng trăm chu kỳ CPU và I/O. Để đơn giản việc hiển thị, chúng ta xem chỉ một chu kỳ CPU (trong mili giây) trên quá trình trong các thí dụ của chúng ta. Thước đo của việc so sánh là thời gian chờ đợi trung bình.

V Các giải thuật định thời

Định thời biểu CPU giải quyết vấn đề quyết định quá trình nào trong hàng đợi sẵn sàng được cấp phát CPU. Trong phần này chúng ta mô tả nhiều giải thuật định thời CPU đang có.

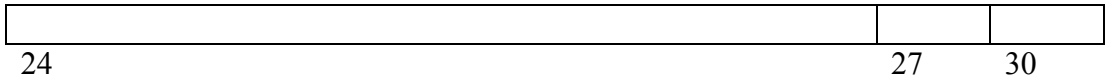
V.1 Định thời đến trước được phục vụ trước

Giải thuật định thời biểu CPU đơn giản nhất là **đến trước, được phục vụ trước** (first-come, first-served-FCFS). Với cơ chế này, quá trình yêu cầu CPU trước được cấp phát CPU trước. Việc cài đặt chính sách FCFS được quản lý dễ dàng với hàng đợi FIFO. Khi một quá trình đi vào hàng đợi sẵn sàng, PCB của nó được liên kết tới đuôi của hàng đợi. Khi CPU rảnh, nó được cấp phát tới một quá trình tại đầu hàng đợi. Sau đó, quá trình đang chạy được lấy ra khỏi hàng đợi. Mã của giải thuật FCFS đơn giản để viết và hiểu.

Tuy nhiên, thời gian chờ đợi trung bình dưới chính sách FCFS thường là dài. Xét tập hợp các quá trình sau đến tại thời điểm 0, với chiều dài thời gian chu kỳ CPU được cho theo mini giây.

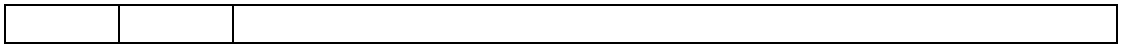
Quá trình	Thời gian xử lý
P1	24
P2	3
P3	3

Nếu các quá trình đến theo thứ tự P_1, P_2, P_3 và được phục vụ theo thứ tự FCFS, chúng ta nhận được kết quả được hiển thị trong **lưu đồ Gantt** như sau:



Thời gian chờ là 0 mili giây cho quá trình P_1, 24 mili giây cho quá trình P_2 và 27 mili giây cho quá trình P_3. Do đó, thời gian chờ đợi trung bình là (0+24+27)/3=17 mili giây. Tuy nhiên, nếu các quá trình đến theo thứ tự P_2, P_3, P_1 thì các kết quả được hiển thị trong **lưu đồ Gannt** như sau:

Thuviendientu.org



Thời gian chờ đợi trung bình bây giờ là (6+0+3)/3=3 mili giây. Việc cắt giảm này là quan trọng. Do đó, thời gian chờ đợi trung bình dưới chính sách FCFS thường không là tối thiểu và có sự thay đổi rất quan trọng nếu các thời gian CPU dành cho các quá trình khác nhau rất lớn.

Ngoài ra, xét năng lực của định thời FCFS trong trường hợp động. Giả sử chúng ta có một quá trình hướng xử lý (CPU-bound) và nhiều quá trình hướng nhập/xuất (I/O bound). Khi các quá trình đưa đến quanh hệ thống, ngữ cảnh sau có thể xảy ra. Quá trình hướng xử lý sẽ nhận CPU và giữ nó. Trong suốt thời gian này, tất cả quá trình khác sẽ kết thúc việc nhập/xuất của nó và chuyển vào hàng đợi sẵn sàng, các thiết bị nhập/xuất ở trạng thái rảnh. Cuối cùng, quá trình hướng xử lý kết thúc chu kỳ CPU của nó và chuyển tới thiết bị nhập/xuất. Tất cả các quá trình hướng xử lý có chu kỳ CPU rất ngắn sẽ nhanh chóng thực thi và di chuyển trở về hàng đợi nhập/xuất. Tại thời điểm này CPU ở trạng thái rảnh. Sau đó, quá trình hướng xử lý sẽ di chuyển trở lại hàng đợi sẵn sàng và được cấp CPU. Một lần nữa, tất cả quá trình hướng nhập/xuất kết thúc việc chờ trong hàng đợi sẵn sàng cho đến khi quá trình hướng xử lý được thực hiện. Có một **tác dụng phụ** (convoy effect) khi tất cả các quá trình khác chờ một quá trình lớn trả lại CPU. Tác dụng phụ này dẫn đến việc sử dụng thiết bị và CPU thấp hơn nếu các quá trình ngắn hơn được cấp trước.

Giải thuật FCSF là giải thuật định thời không trưng dụng CPU. Một khi CPU được cấp phát tới một quá trình, quá trình đó giữ CPU cho tới khi nó giải phóng CPU bằng cách kết thúc hay yêu cầu nhập/xuất. Giải thuật FCFS đặc biệt không phù hợp đối với hệ thống chia sẻ thời gian, ở đó mỗi người dùng nhận được sự chia sẻ CPU với những khoảng thời gian đều nhau.

V.2 Định thời biểu công việc ngắn nhất trước

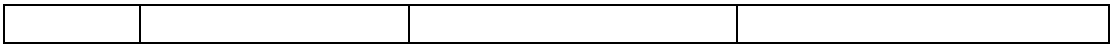
Một tiếp cận khác đối với việc định thời CPU là giải thuật định thời **công việc ngắn nhất trước** (shortest-job-first-SJF). Giải thuật này gán tới mỗi quá trình chiều dài của chu kỳ CPU tiếp theo cho quá trình sau đó. Khi CPU sẵn dùng, nó được gán tới quá trình có chu kỳ CPU kế tiếp ngắn nhất. Nếu hai quá trình có cùng chiều dài chu kỳ CPU kế tiếp, định thời FCFS được dùng. Chú ý rằng thuật ngữ phù hợp hơn là chu kỳ CPU kế tiếp ngắn nhất (shortest next CPU burst) vì định thời được thực hiện bằng cách xem xét chiều dài của chu kỳ CPU kế tiếp của quá trình hơn là toàn bộ chiều dài của nó. Chúng ta dùng thuật ngữ SJF vì hầu hết mọi người và mọi sách tham khảo tới nguyên lý của loại định thời biểu này như SJF.

Thí dụ, xét tập hợp các quá trình sau, với chiều dài của thời gian chu kỳ CPU được tính bằng mili giây:

Quá trình	Thời gian xử lý
P1	6
P2	8
P3	7
P4	3

Dùng định thời SJF, chúng ta định thời biểu cho các quá trình này theo **lưu đồ Gannt** như sau:

Thuviendientu.org



Thời gian chờ đợi là 3 mili giây cho quá trình P_1, 16 mili giây cho quá trình P_2, 9 mili giây cho quá trình P_3, và 0 mili giây cho quá trình P_4. Do đó, thời gian chờ đợi trung bình là (3+16+9+0)/4 = 7 mili giây. Nếu chúng ta dùng cơ chế định thời FCFS thì thời gian chờ đợi trung bình là 10.23 mili giây.

Giải thuật SJF có thể là tối ưu, trong đó nó cho thời gian chờ đợi trung bình nhỏ nhất cho các quá trình được cho. Bằng cách chuyển một quá trình ngắn trước một quá trình dài thì thời gian chờ đợi của quá trình ngắn giảm hơn so với việc tăng thời gian chờ đợi của quá trình dài. Do đó, thời gian chờ đợi trung bình giảm.

Khó khăn thật sự với giải thuật SJF là làm thế nào để biết chiều dài của yêu cầu CPU tiếp theo. Đối với định thời dài trong hệ thống bó, chúng ta có thể dùng chiều dài như giới hạn thời gian xử lý mà người dùng xác định khi gởi công việc. Do đó, người dùng được cơ động để ước lượng chính xác giới hạn thời gian xử lý vì giá trị thấp hơn có nghĩa là đáp ứng nhanh hơn. Định thời SJF được dùng thường xuyên trong định thời dài.

Mặc dù SJF là tối ưu nhưng nó không thể được cài đặt tại cấp định thời CPU ngắn vì không có cách nào để biết chiều dài chu kỳ CPU tiếp theo. Một tiếp cận là khác gần đúng định thời SJF được thực hiện. Chúng ta có thể không biết chiều dài của chu kỳ CPU kế tiếp nhưng chúng ta có đoán giá trị của nó. Chúng ta mong muốn rằng chu kỳ CPU kế tiếp sẽ tương tự chiều dài những chu kỳ CPU trước đó. Do đó, bằng cách tính toán mức xấp xỉ chiều dài của chu kỳ CPU kế tiếp, chúng ta chọn một quá trình với chu kỳ CPU được đoán là ngắn nhất.

Chu kỳ CPU kế tiếp thường được đoán như trung bình số mũ của chiều dài các chu kỳ CPU trước đó. Gọi t_n là chiều dài của chu kỳ CPU thứ n và gọi T_{n+1} giá trị được đoán cho chu kỳ CPU kế tiếp. Thì đối với α, với $0 \le \alpha \le 1$, định nghĩa

$$T_{n+1} = \alpha\, t_n + (1- \alpha)\, T_n$$

Công thức này định nghĩa một giá trị trung bình số mũ. Giá trị của t_n chứa thông tin mới nhất; T_n lưu lịch sử quá khứ. Tham số α điều khiển trọng số liên quan giữa lịch sử quá khứ và lịch sử gần đây trong việc đoán. Nếu α=0 thì $T_{n+1}=T_n$ và lịch sử gần đây không có ảnh hưởng (điều kiện hiện hành được đảm bảo là ngắn); nếu α =1 thì $T_{n+1}=t_n$ và chỉ chu kỳ CPU gần nhất có ảnh hưởng (lịch sử được đảm bảo là cũ và không phù hợp). Thông dụng hơn, α=1/2 thì lịch sử gần đây và lịch sử quá khứ có trọng số tương đương. Giá trị khởi đầu T_0 có thể được định nghĩa như một hằng số hay như toàn bộ giá trị trung bình hệ thống. Hình IV.2 dưới đây hiển thị giá trị trung bình dạng mũ với α=1/2 và T_0=10.

Để hiểu hành vi của giá trị trung bình dạng mũ, chúng ta có thể mở rộng công thức cho T_{n+1} bằng cách thay thế T_n để tìm

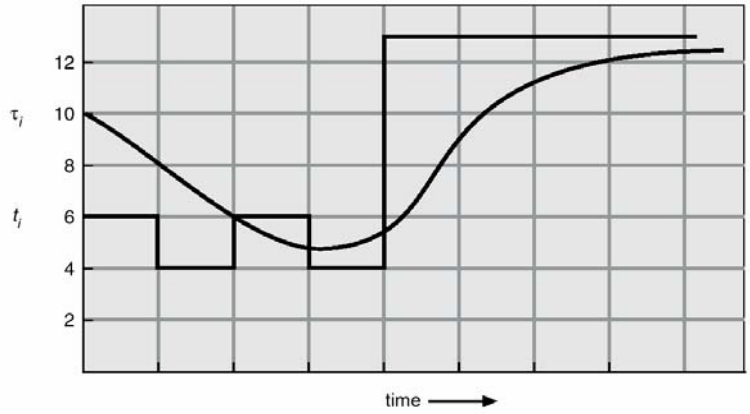
$T_{n+1}=\alpha\, t_n+(1-\alpha)\, \alpha\, t_{n-1}+\ldots+(1-\alpha)^j\, \alpha\, t_{n-j}+\ldots+(1-\alpha)^{n-1} T_0$

Vì cả hai α và (1- α) là nhỏ hơn hay bằng 1, mỗi số hạng tiếp theo có trọng số nhỏ hơn số hạng trước đó.

Giải thuật SJF có thể trưng dụng hoặc không trưng dụng CPU. Chọn lựa này phát sinh khi một quá trình mới đến tại hàng đợi sẵn sàng trong khi một quá trình trước đó đang thực thi. Một quá trình mới có thể có chu kỳ CPU tiếp theo ngắn hơn chu kỳ CPU được để lại của quá trình thực thi hiện tại. Giải thuật SJF trưng dụng sẽ trưng dungj CPU của quá trình đang thực thi hiện tại, trong khi giải thuật SJF không trưng dụng sẽ cho phép quá trình đang thực thi kết thúc chu kỳ CPU của nó. Định thời

Thuviendientu.org

SJF trưng dụng còn được gọi là **định thời thời gian còn lại ngắn nhất trước** (shortest-remaining-time-first).



CPU burst (t_i)		6	4	6	4	13	13	13 ...
"guess" (τ_i)	10	8	6	6	5	9	11	12 ...

Thí dụ, xét 4 quá trình sau với chiều dài của thời gian chu kỳ CPU được cho tính bằng mili giây

Quá trình	Thời gian đến	Thời gian xử lý
P1	0	8
P2	1	4
P3	2	9
P4	3	5

Nếu các quá trình đi vào hàng đợi sẵn sàng tại những thời điểm và cần thời gian xử lý được hiển thị trong bảng trên thì thời biểu SJF trưng dụng được mô tả trong **lưu đồ Gannt** như sau:

P1	P2	P3		
0	1 5	10	17	26

Quá trình P_1 được bắt đầu tại thời điểm 0, vì nó là quá trình duy nhất trong hàng đợi. Quá trình P_2 đến tại thời điểm 1. Thời gian còn lại cho P_1 (7 mili giây) là lớn hơn thời gian được yêu cầu bởi quá trình P_2 (4 mili giây) vì thế quá trình P_1 bị trưng dụng CPU và quá trình P_2 được định thời biểu. Thời gian chờ đợi trung bình cho thí dụ này là: ((10-1) + (1-1) + (17-2) + (5-3))/4 = 6.5 mili giây. Định thời SJF không trưng dụng cho kết quả thời gian chờ đợi trung bình là 7.75 mili giây.

V.3 Định thời theo độ ưu tiên

Giải thuật SJF là trường hợp đặc biệt của **giải thuật định thời theo độ ưu tiên** (priority-scheduling algorithm). Độ ưu tiên được gán với mỗi quá trình và CPU được cấp phát tới quá trình với độ ưu tiên cao nhất. Quá trình có độ ưu tiên bằng nhau được định thời trong thứ tự FCFS.

Giải thuật SJF là giải thuật ưu tiên đơn giản ở đó độ ưu tiên **p** là nghịch đảo với chu kỳ CPU được đoán tiếp theo. Chu kỳ CPU lớn hơn có độ ưu tiên thấp hơn và ngược lại.

Thuviendientu.org

Bây giờ chúng ta thảo luận định thời có độ ưu tiên cao và thấp. Các độ ưu tiên thường nằm trong dãy số cố định, chẳng hạn 0 tới 7 hay 0 tới 4,095. Tuy nhiên, không có sự thoả thuận chung về 0 là độ ưu tiên thấp nhất hay cao nhất. Một vài hệ thống dùng số thấp để biểu diễn độ ưu tiên thấp; ngược lại các hệ thống khác dùng các số thấp cho độ ưu tiên cao. Sự khác nhau này có thể dẫn đến sự lẫn lộn. Trong giáo trình này chúng ta dùng các số thấp để biểu diễn độ ưu tiên cao.

Thí dụ, xét tập hợp quá trình sau đến tại thời điểm 0 theo thứ tự P_1, P_2,…, P_5 với chiều dài thời gian chu kỳ CPU được tính bằng mili giây:

Quá trình	Thời gian xử lý	Độ ưu tiên
P1	10	3
P2	1	1
P3	2	4
P4	1	5
P5	5	2

Sử dụng định thời theo độ ưu tiên, chúng ta sẽ định thời các quá trình này theo **lưu đồ Gannt** như sau:

P2	P5	P1	P3	P4
0	1 6	16	18	19

Thời gian chờ đợi trung bình là 8.2 mili giây.

Độ ưu tiên có thể được định nghĩa bên trong hay bên ngoài. Độ ưu tiên được định nghĩa bên trong thường dùng định lượng hoặc nhiều định lượng có thể đo để tính toán độ ưu tiên của một quá trình. Thí dụ, các giới hạn thời gian, các yêu cầu bộ nhớ, số lượng tập tin đang mở và tỉ lệ của chu kỳ nhập/xuất trung bình với tỉ lệ của chu kỳ CPU trung bình. Các độ ưu tiên bên ngoài được thiết lập bởi các tiêu chuẩn bên ngoài đối với hệ điều hành như sự quan trọng của quá trình, loại và lượng chi phí đang được trả cho việc dùng máy tính, văn phòng hỗ trợ công việc, ..

Định thời biểu theo độ ưu tiên có thể trưng dụng hoặc không trưng dụng CPU. Khi một quá trình đến hàng đợi sẵn sàng, độ ưu tiên của nó được so sánh với độ ưu tiên của quá trình hiện đang chạy. Giải thuật định thời theo độ ưu tiên trưng dụng sẽ chiếm CPU nếu độ ưu tiên của quá trình mới đến cao hơn độ ưu tiên của quá trình đang thực thi. Giải thuật định thời theo độ ưu tiên không trưng dụng sẽ đơn giản đặt quá trình mới tại đầu hàng đợi sẵn sàng.

Vấn đề chính với giải thuật định thời theo độ ưu tiên là **nghẽn không hạn định** (indefinite blocking) hay **đói CPU** (starvation). Một quá trình sẵn sàng chạy nhưng thiếu CPU có thể xem như bị nghẽn-chờ đợi CPU. Giải thuật định thời theo độ ưu tiên có thể để lại nhiều quá trình có độ ưu tiên thấp chờ CPU không hạn định. Trong một hệ thống máy tính tải cao, dòng đều đặn các quá trình có độ ưu tiên cao hơn có thể ngăn chặn việc nhận CPU của quá trình có độ ưu tiên thấp.. Thông thường, một trong hai trường hợp xảy ra. Cuối cùng, một quá trình sẽ được chạy (lúc 2 a.m chủ nhật là thời điểm cuối cùng hệ thống nạp các quá trình nhẹ), hay cuối cùng hệ thống máy tính sẽ đổ vỡ và mất tất cả các quá trình có độ ưu tiên thấp chưa được kết thúc.

Một giải pháp cho vấn đề nghẽn không hạn định này là **sự hoá già** (aging). Hóa già là kỹ thuật tăng dần độ ưu tiên của quá trình chờ trong hệ thống một thời gian dài. Thí dụ, nếu các độ ưu tiên nằm trong dãy từ 127 (thấp) đến 0 (cao), chúng ta giảm độ ưu tiên của quá trình đang chờ xuống 1 mỗi 15 phút. Cuối cùng, thậm chí một quá

Thuviendientu.org

trình với độ ưu tiên khởi đầu 127 sẽ đạt độ ưu tiên cao nhất trong hệ thống và sẽ được thực thi. Thật vậy, một quá trình sẽ mất không quá 32 giờ để đạt được độ ưu tiên từ 127 tới 0.

V.4 Định thời luân phiên

Giải thuật định thời luân phiên (round-robin scheduling algorithm-RR) được thiết kế đặc biệt cho hệ thống chia sẻ thời gian. Tương tự như định thời FCFS nhưng sự trưng dụng CPU được thêm vào để chuyển CPU giữa các quá trình. Đơn vị thời gian nhỏ được gọi là định mức thời gian (time quantum) hay phần thời gian (time slice) được định nghĩa. Định mức thời gian thường từ 10 đến 100 mili giây. Hàng đợi sẵn sàng được xem như một hàng đợi vòng. Bộ định thời CPU di chuyển vòng quanh hàng đợi sẵn sàng, cấp phát CPU tới mỗi quá trình có khoảng thời gian tối đa bằng một định mức thời gian.

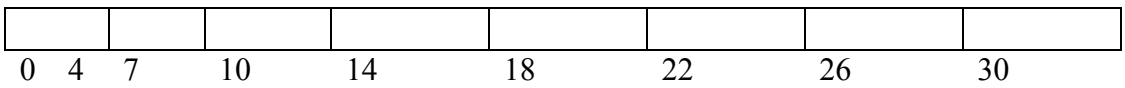
Để cài đặt định thời RR, chúng ta quản lý hàng đợi sẵn sàng như một hàng đợi FIFO của các quá trình. Các quá trình mới được thêm vào đuôi hàng đợi. Bộ định thời CPU chọn quá trình đầu tiên từ hàng đợi sẵn sàng, đặt bộ đếm thời gian để ngắt sau 1 định mức thời gian và gởi tới quá trình.

Sau đó, một trong hai trường hợp sẽ xảy ra. Quá trình có 1 chu kỳ CPU ít hơn 1 định mức thời gian. Trong trường hợp này, quá trình sẽ tự giải phóng. Sau đó, bộ định thời biểu sẽ xử lý quá trình tiếp theo trong hàng đợi sẵn sàng. Ngược lại, nếu chu kỳ CPU của quá trình đang chạy dài hơn 1 định mức thời gian thì độ đếm thời gian sẽ báo và gây ra một ngắt tới hệ điều hành. Chuyển đổi ngữ cảnh sẽ được thực thi và quá trình được đặt trở lại tại đuôi của hàng đợi sẵn sàng. Sau đó, bộ định thời biểu CPU sẽ chọn quá trình tiếp theo trong hàng đợi sẵn sàng.

Tuy nhiên, thời gian chờ đợi trung bình dưới chính sách RR thường là quá dài. Xét một tập hợp các quá trình đến tại thời điểm 0 với chiều dài thời gian CPU-burst được tính bằng mili giây:

Quá trình	Thời gian xử lý
P1	24
P2	3
P3	3

Nếu sử dụng định mức thời gian là 4 mili giây thì quá trình P_1 nhận 4 mili giây đầu tiên. Vì nó yêu cầu 20 mili giây còn lại nên nó bị trưng dụng CPU sau định mức thời gian đầu tiên và CPU được cấp tới quá trình tiếp theo trong hàng đợi, quá trình P_2. Vì P_2 không cần tới 4 mili giây nên nó kết thúc trước khi định mức thời gian của nó hết hạn. Sau đó, CPU được cho tới quá trình kế tiếp, quá trình P_3. Một khi mỗi quá trình nhận 1 định mức thời gian thì CPU trả về quá trình P_1 cho định mức thời gian tiếp theo. Thời biểu RR là:



Thời gian chờ đợi trung bình là 17/3=5.66 mili giây.

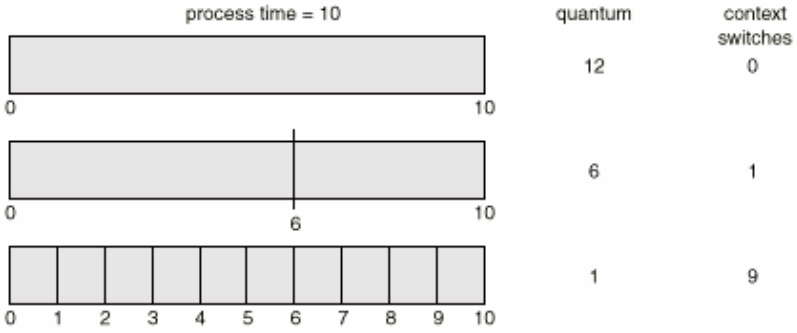
Trong giải thuật RR, không quá trình nào được cấp phát CPU cho nhiều hơn 1 định mức thời gian trong một hàng. Nếu chu kỳ CPU của quá trình vượt quá 1 định

Thuviendientu.org

mức thời gian thì quá trình đó bị trưng dụng CPU và nó được đặt trở lại hàng đợi sẵn sàng. Giải thuật RR là giải thuật trưng dụng CPU.

Nếu có n quá trình trong hàng đợi sẵn sàng và định mức thời gian là q thì mỗi quá trình nhận 1/n thời gian CPU trong các phần, nhiều nhất q đơn vị thời gian. Mỗi quá trình sẽ chờ không dài hơn (n-1)x q đơn vị thời gian cho tới khi định mức thời gian tiếp theo của nó. Thí dụ, nếu có 5 quá trình với định mức thời gian 20 mili giây thì mỗi quá trình sẽ nhận 20 mili giây sau mỗi 100 mili giây.

Năng lực của giải thuật RR phụ thuộc nhiều vào kích thước của định mức thời gian. Nếu định mức thời gian rất lớn (lượng vô hạn) thì chính sách RR tương tự như chính sách FCFS. Nếu định mức thời gian là rất nhỏ (1 mili giây) thì tiếp cận RR được gọi là **chia sẻ bộ xử lý** (processor sharing) và xuất hiện (trong lý thuyết) tới người dùng như thể mỗi quá trình trong n quá trình có bộ xử lý riêng của chính nó chạy tại 1/n tốc độ của bộ xử lý thật.



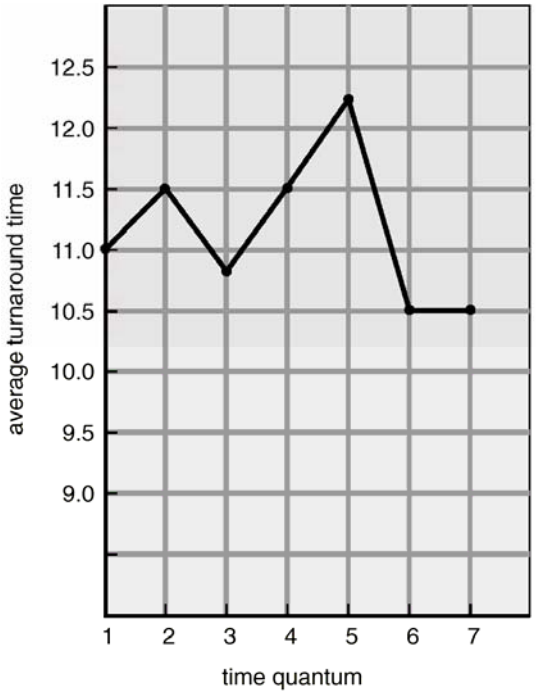
Hình 0-3 Hiển thị một định mức thời gian nhỏ hơn tăng chuyển đổi ngữ cảnh như thế nào

Tuy nhiên, trong phần mềm chúng ta cũng cần xem xét hiệu quả của việc chuyển đổi ngữ cảnh trên năng lực của việc định thời RR. Chúng ta giả sử rằng chỉ có 1 quá trình với 10 đơn vị thời gian. Nếu một định mức là 12 đơn vị thời gian thì quá trình kết thúc ít hơn 1 định mức thời gian, với không có chi phí nào khác. Tuy nhiên, nếu định mức là 6 đơn vị thời gian thì quá trình cần 2 định mức thời gian, dẫn đến 1 chuyển đổi ngữ cảnh. Nếu định mức thời gian là 1 đơn vị thời gian thì 9 chuyển đổi ngữ cảnh sẽ xảy ra, việc thực thi của quá trình bị chậm như được hiển thị trong hình IV.3 .

Do đó chúng ta mong muốn định mức thời gian lớn đối với thời gian chuyển ngữ cảnh. Nếu thời gian chuyển ngữ cảnh chiếm 10% định mức thời gian thì khoảng 10% thời gian CPU sẽ được dùng cho việc chuyển ngữ cảnh.

Thời gian hoàn thành cũng phụ thuộc kích thước của định mức thời gian. Chúng ta có thể thấy trong hình IV.4, thời gian hoàn thành trung bình của tập hợp các quá trình không cần cải tiến khi kích thước định mức thời gian tăng. Thông thường, thời gian hoàn thành trung bình có thể được cải tiến nếu hầu hết quá trình kết thúc chu kỳ CPU kế tiếp của chúng trong một định mức thời gian. Thí dụ, cho 3 quá trình có 10 đơn vị thời gian cho mỗi quá trình và định mức thời gian là 1 đơn vị thời gian, thì thời gian hoàn thành trung bình là 29. Tuy nhiên, nếu định mức thời gian là 10 thì thời gian hoàn thành trung bình giảm tới 20. Nếu thời gian chuyển ngữ cảnh được thêm vào thì thời gian hoàn thành trung bình gia tăng đối với định mức thời gian nhỏ hơn vì các chuyển đổi ngữ cảnh thêm nữa sẽ được yêu cầu.

Thuviendientu.org



process	time
P_1	6
P_2	3
P_3	1
P_4	7

Hình 0-4 Hiển thị cách thời gian hoàn thành biến đổi theo định mức thời gian

Ngoài ra, nếu định mức thời gian quá lớn thì người thiết kế việc định thời RR bao gồm chính sách FCFS. Qui tắc là định mức thời gian nên dài hơn 80% chu kỳ CPU.

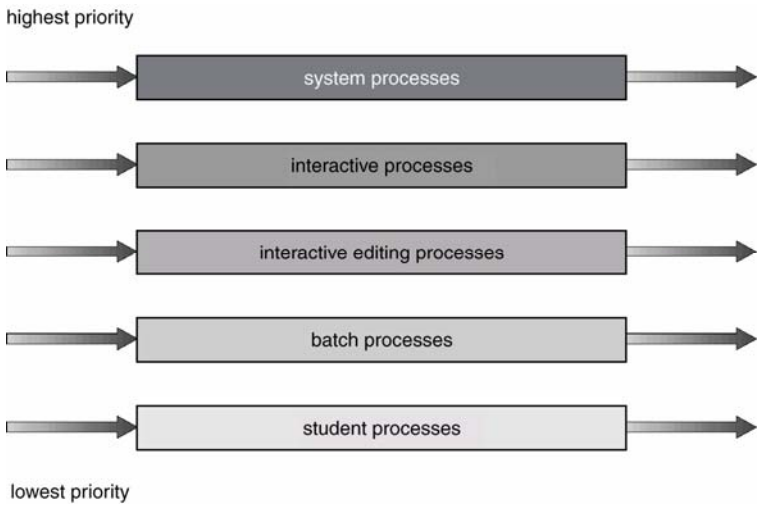
V.5 Định thời biểu với hàng đợi nhiều cấp

Một loại giải thuật định thời khác được tạo ra cho những trường hợp mà trong đó các quá trình được phân lớp thành các nhóm khác nhau. Thí dụ: việc phân chia thông thường được thực hiện giữa các quá trình chạy ở chế độ giao tiếp (foreground hay interactive) và các quá trình chạy ở chế độ nền hay dạng bó (background hay batch). Hai loại quá trình này có yêu cầu đáp ứng thời gian khác nhau và vì thế có yêu cầu về định thời biểu khác nhau. Ngoài ra, các quá trình chạy ở chế độ giao tiếp có độ ưu tiên (hay được định nghĩa bên ngoài) cao hơn các quá trình chạy ở chế độ nền.

Một **giải thuật định thời hàng đợi nhiều cấp** (multilevel queue-scheduling algorithm) chia hàng đợi thành nhiều hàng đợi riêng rẻ (hình IV.5). Các quá trình được gán vĩnh viễn tới một hàng đợi, thường dựa trên thuộc tính của quá trình như kích thước bộ nhớ, độ ưu tiên quá trình hay loại quá trình. Mỗi hàng đợi có giải thuật định thời của chính nó. Thí dụ: các hàng đợi riêng rẻ có thể được dùng cho các quá trình ở chế độ nền và chế độ giao tiếp. Hàng đợi ở chế độ giao tiếp có thể được định thời bởi giải thuật RR trong khi hàng đợi ở chế độ nền được định thời bởi giải thuật FCFS.

Ngoài ra, phải có việc định thời giữa các hàng đợi, mà thường được cài đặt như định thời trưng dụng với độ ưu tiên cố định. Thí dụ, hàng đợi ở chế độ giao tiếp có độ ưu tiên tuyệt đối hơn hàng đợi ở chế độ nền.

Thuviendientu.org



Chúng ta xét một thí dụ của giải thuật hàng đợi nhiều mức với năm hàng đợi:

- Các quá trình hệ thống
- Các quá trình giao tiếp
- Các quá trình soạn thảo giao tiếp
- Các quá trình bó
- Các quá trình sinh viên

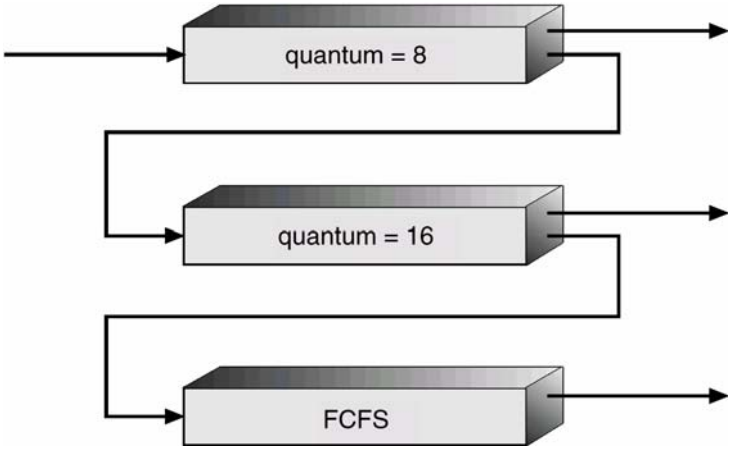
Mỗi hàng đợi có độ ưu tiên tuyệt đối hơn hàng đợi có độ ưu tiên thấp hơn. Thí dụ: không có quá trình nào trong hàng đợi bó có thể chạy trừ khi hàng đợi cho các quá trình hệ thống, các quá trình giao tiếp và các quá trình soạn thảo giao tiếp đều rỗng. Nếu một quá trình soạn thảo giao tiếp được đưa vào hàng đợi sẵn sàng trong khi một quá trình bó đang chạy thì quá trình bó bị trưng dụng CPU. Solaris 2 dùng dạng giải thuật này.

Một khả năng khác là phần (slice) thời gian giữa hai hàng đợi. Mỗi hàng đợi nhận một phần thời gian CPU xác định, sau đó nó có thể định thời giữa các quá trình khác nhau trong hàng đợi của nó. Thí dụ, trong hàng đợi giao tiếp-nền, hàng đợi giao tiếp được cho 80% thời gian của CPU cho giải thuật RR giữa các quá trình của nó, ngược lại hàng đợi nền nhận 20% thời gian CPU cho các quá trình của nó theo cách FCFS.

V.6 Định thời hàng đợi phản hồi đa cấp

Thông thường, trong giải thuật hàng đợi đa cấp, các quá trình được gán vĩnh viễn tới hàng đợi khi được đưa vào hệ thống. Các quá trình không di chuyển giữa các hàng đợi. Nếu có các hàng đợi riêng cho các quá trình giao tiếp và các quá trình nền thì các quá trình không di chuyển từ một hàng đợi này tới hàng đợi khác vì các quá trình không thay đổi tính tự nhiên giữa giao tiếp và nền. Cách tổ chức có ích vì chi phí định thời thấp nhưng thiếu linh động và có thể dẫn đến tình trạng "đói CPU".

Thuviendientu.org



Hình 0-6 Các hàng đợi phản hồi nhiều cấp

Tuy nhiên, **định thời hàng đợi phản hồi đa cấp** (multilevel feedback queue scheduling) cho phép một quá trình di chuyển giữa các hàng đợi. Ý tưởng là tách riêng các quá trình với các đặc điểm chu kỳ CPU khác nhau. Nếu một quá trình dùng quá nhiều thời gian CPU thì nó sẽ được di chuyển tới hàng đợi có độ ưu tiên thấp. Cơ chế này để lại các quá trình hướng nhập/xuất và các quá trình giao tiếp trong các hàng đợi có độ ưu tiên cao hơn. Tương tự, một quá trình chờ quá lâu trong hàng đợi có độ ưu tiên thấp hơn có thể được di chuyển tới hàng đợi có độ ưu tiên cao hơn. Đây là hình thức của sự hóa già nhằm ngăn chặn sự đói CPU.

Thí dụ, xét một bộ định thời hàng đợi phản hồi nhiều cấp với ba hàng đợi được đánh số từ 0 tới 2 (như hình IV.6). Bộ định thời trước tiên thực thi tất cả quá trình chứa trong hàng đợi 0. Chỉ khi hàng đợi 0 rỗng nó sẽ thực thi các quá trình trong hàng đợi 1. Tương tự, các quá trình trong hàng đợi 2 sẽ được thực thi chỉ nếu hàng đợi 0 và 1 rỗng. Một quá trình đến hàng đợi 1 sẽ ưu tiên hơn quá trình đến hàng đợi 2. Tương tự, một quá trình đến hàng đợi 0 sẽ ưu tiên hơn một quá trình vào hàng đợi 1.

Một quá trình đưa vào hàng đợi sẵn sàng được đặt trong hàng đợi 0. Một quá trình trong hàng đợi 0 được cho một định mức thời gian là 8 mili giây. Nếu nó không kết thúc trong thời gian này thì nó sẽ di chuyển vào đuôi của hàng đợi 1. Nếu hàng đợi 0 rỗng thì quá trình tại đầu của hàng đợi 1 được cho định mức thời gian là 16 mili giây. Nếu nó không hoàn thành thì nó bị chiếm CPU và được đặt vào hàng đợi 2. Các quá trình trong hàng đợi 2 được chạy trên cơ sở FCFS chỉ khi hàng đợi 0 và 1 rỗng.

Giải thuật định thời này cho độ ưu tiên cao nhất tới bất cứ quá trình nào với chu kỳ CPU 8 mili giây hay ít hơn. Một quá trình như thế sẽ nhanh chóng nhận CPU, kết thúc chu kỳ CPU của nó và bỏ đi chu kỳ I/O kế tiếp của nó. Các quá trình cần hơn 8 mili giây nhưng ít hơn 24 mili giây được phục vụ nhanh chóng mặc dù với độ ưu tiên thấp hơn các quá trình ngắn hơn. Các quá trình dài tự động rơi xuống hàng đợi 2 và được phục vụ trong thứ tự FCFS với bất cứ chu kỳ CPU còn lại từ hàng đợi 0 và 1.

Nói chung, một bộ định thời hàng đợi phản hồi nhiều cấp được định nghĩa bởi các tham số sau:

- Số lượng hàng đợi
- Giải thuật định thời cho mỗi hàng đợi
- Phương pháp được dùng để xác định khi nâng cấp một quá trình tới hàng đợi có độ ưu tiên cao hơn.
- Phương pháp được dùng để xác định khi nào chuyển một quá trình tới hàng đợi có độ ưu tiên thấp hơn.

Thuviendientu.org

- Phương pháp được dùng để xác định hàng đợi nào một quá trình sẽ đi vào và khi nào quá trình đó cần phục vụ.

Định nghĩa bộ định thời biểu dùng hàng đợi phản hồi nhiều cấp trở thành giải thuật định thời CPU phổ biến nhất. Bộ định thời này có thể được cấu hình để thích hợp với hệ thống xác định. Tuy nhiên, bộ định thời này cũng yêu cầu một vài phương tiện chọn lựa giá trị cho tất cả tham số để định nghĩa bộ định thời biểu tốt nhất. Mặc dù một hàng đợi phản hồi nhiều cấp là cơ chế phổ biến nhất nhưng nó cũng là cơ chế phức tạp nhất.

VI Định thời biểu đa bộ xử lý

Phần trên thảo luận chúng ta tập trung vào những vấn đề định thời biểu CPU trong một hệ thống với một bộ vi xử lý đơn. Nếu có nhiều CPU, vấn đề định thời tương ứng sẽ phức tạp hơn. Nhiều khả năng đã được thử nghiệm và như chúng ta đã thấy với định thời CPU đơn bộ xử lý, không có giải pháp tốt nhất. Trong phần sau đây, chúng ta sẽ thảo luận vắn tắt một số vấn đề tập trung về định thời biểu đa bộ xử lý. Chúng ta tập trung vào những hệ thống mà các bộ xử lý của nó được xác định (hay đồng nhất) trong thuật ngữ chức năng của chúng; bất cứ bộ xử lý nào sẵn có thì có thể được dùng để chạy bất quá trình nào trong hàng đợi. Chúng ta cũng cho rằng truy xuất bộ nhớ là đồng nhất (uniform memory access-UMA). Chỉ những chương trình được biên dịch đối với tập hợp chỉ thị của bộ xử lý được cho mới có thể được chạy trên chính bộ xử lý đó.

Ngay cả trong một bộ đa xử lý đồng nhất đôi khi có một số giới hạn cho việc định thời biểu. Xét một hệ thống với một thiết bị nhập/xuất được gán tới một đường bus riêng của một bộ xử lý. Các quá trình muốn dùng thiết bị đó phải được định thời biểu để chạy trên bộ xử lý đó, ngược lại thiết bị đó là không sẵn dùng.

Nếu nhiều bộ xử lý xác định sẵn dùng thì chia sẻ tải có thể xảy ra. Nó có thể cung cấp một hàng đợi riêng cho mỗi bộ xử lý. Tuy nhiên, trong trường hợp này, một bộ xử lý có thể rảnh với hàng đợi rỗng, trong khi bộ xử lý khác rất bận. Để ngăn chặn trường hợp này, chúng ta dùng một hàng đợi sẵn sàng chung. Tất cả quá trình đi vào một hàng đợi và được định thời biểu trên bất cứ bộ xử lý sẵn dùng nào.

Trong một cơ chế như thế, một trong hai tiếp cận định thời biểu có thể được dùng. Trong tiếp cận thứ nhất, mỗi bộ xử lý định thời chính nó. Mỗi bộ xử lý xem xét hàng đợi sẵn sàng chung và chọn một quá trình để thực thi. Nếu chúng ta có nhiều bộ xử lý cố gắng truy xuất và cập nhật một cấu trúc dữ liệu chung thì mỗi bộ xử lý phải được lập trình rất cẩn thận. Chúng ta phải đảm bảo rằng hai bộ xử lý không chọn cùng quá trình và quá trình đó không bị mất từ hàng đợi. Tiếp cận thứ hai tránh vấn đề này bằng cách đề cử một bộ xử lý như bộ định thời cho các quá trình khác, do đó tạo ra cấu trúc chủ-tớ (master-slave).

Một vài hệ thống thực hiện cấu trúc này từng bước bằng cách tất cả quyết định định thời, xử lý nhập/xuất và các hoạt động hệ thống khác được quản lý bởi một bộ xử lý đơn-một server chủ. Các bộ xử lý khác chỉ thực thi mã người dùng. Đa xử lý không đối xứng (asymmetric multiprocessing) đơn giản hơn đa xử lý đối xứng (symmetric multiprocessing) vì chỉ một quá trình truy xuất các cấu trúc dữ liệu hệ thống, làm giảm đi yêu cầu chia sẻ dữ liệu. Tuy nhiên, nó cũng không hiệu quả. Các quá trình giới hạn nhập/xuất có thể gây thắt cổ chai (bottleneck) trên một CPU đang thực hiện tất cả các hoạt động. Điển hình, đa xử lý không đối xứng được cài đặt trước trong một hệ điều hành và sau đó được nâng cấp tới đa xử lý đối xứng khi hệ thống tiến triển.

Thuviendientu.org

VII Định thời thời gian thực

Trong chương đầu chúng ta đã tìm hiểu tổng quan về hệ điều hành thời thực và thảo luận tầm quan trọng của nó. Ở đây, chúng ta tiếp tục thảo luận bằng cách mô tả các điều kiện thuận lợi định thời cần để hỗ trợ tính toán thời thực trong hệ thống máy tính đa mục đích.

Tính toán thời thực được chia thành hai loại: hệ thống thời thực cứng (hardware real-time systems) được yêu cầu để hoàn thành một tác vụ tới hạn trong lượng thời gian được đảm bảo. Thông thường, một quá trình được đưa ra xem xét cùng với khai báo lượng thời gian nó cần để hoàn thành hay thực hiện nhập/xuất. Sau đó, bộ định thời biểu nhận được quá trình, đảm bảo rằng quá trình sẽ hoàn thành đúng giờ hay từ chối yêu cầu khi không thể. Điều này được gọi là đặt trước tài nguyên (resource reservation). Để đảm bảo như thế đòi hỏi bộ định thời biết chính xác bao lâu mỗi loại chức năng hệ điều hành mất để thực hiện và do đó mỗi thao tác phải được đảm bảo để mất lượng thời gian tối đa. Một đảm bảo như thế là không thể trong hệ thống với lưu trữ phụ và bộ nhớ ảo vì các hệ thống con này gây ra sự biến đổi không thể tránh hay không thể thấy trước trong lượng thời gian thực thi một quá trình xác định. Do đó, hệ thống thời thực cứng được hình thành từ nhiều phần mềm có mục đích đặc biệt chạy trên phần cứng tận hiến cho các quá trình tới hạn, và thiếu chức năng đầy đủ của các máy tính và các hệ điều hành hiện đại.

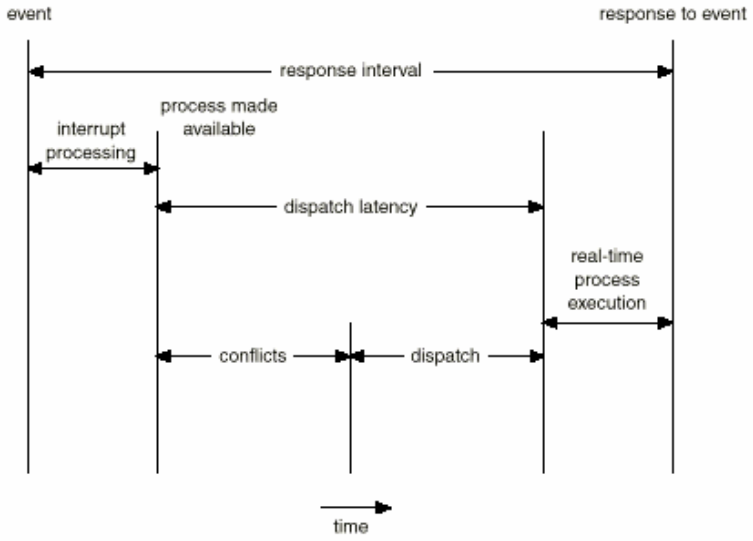
Tính toán thời gian thực mềm (soft real-time computing) ít nghiêm khắc hơn. Nó yêu cầu các quá trình tới hạn nhận độ ưu tiên cao hơn các quá trình khác. Mặc dù thêm chức năng thời thực mềm tới hệ chia sẻ thời gian có thể gây ra việc cấp phát tài nguyên không công bằng và có thể dẫn tới việc trì hoãn lâu hơn hay thậm chí đói tài nguyên đối với một số quá trình, nhưng nó ít có thể đạt được. Kết quả là hệ thống mục đích chung cũng có thể hỗ trợ đa phương tiện, đồ họa giao tiếp tốc độ cao, và nhiều tác vụ khác nhưng không hỗ trợ tính toán thời thực mềm.

Cài đặt chức năng thời thực mềm đòi hỏi thiết kế cẩn thận bộ định thời biểu và các khía cạnh liên quan của hệ điều hành. Trước tiên, hệ thống phải có định thời trưng dụng và các quá trình thời thực phải có độ ưu tiên cao nhất. Độ ưu tiên của các quá trình thời thực phải không giảm theo thời gian mặc dù độ ưu tiên của các quá trình không thời thực có thể giảm. Thứ hai, độ trễ của việc điều phối phải nhỏ. Một quá trình thời thực nhỏ hơn, nhanh hơn có thể bắt đầu thực thi một khi nó có thể chạy.

Quản trị các thuộc tính đã được xem xét ở trên là tương đối đơn giản. Thí dụ, chúng ta có thể không cho phép một quá trình hóa già trên các quá trình thời thực, do đó đảm bảo rằng độ ưu tiên của các quá trình không thay đổi. Tuy nhiên, đảm bảo thuộc tính sau đây phức tạp hơn. Vấn đề là nhiều hệ điều hành gồm hầu hết ấn bản của UNIX bị bắt buộc chờ lời gọi hệ thống hoàn thành hay nghẽn nhập/xuất xảy ra trước khi thực hiện chuyển ngữ cảnh. Độ trễ điều phối trong những hệ thống như thế có thể dài vì một số lời gọi hệ thống phức tạp và một vài thiết bị nhập/xuất chậm. Để giữ độ trễ điều phối chậm, chúng ta cần cho phép các lời gọi hệ thống được trưng dụng. Có nhiều cách để đạt mục đích này. Cách thứ nhất là chèn các điểm trưng dụng (preemption points) trong những lời gọi hệ thống có khoảng thời gian dài, kiểm tra để thấy quá trình ưu tiên cao cần được thực thi hay không. Nếu đúng, thì chuyển ngữ cảnh xảy ra và khi quá trình có độ ưu tiên kết thúc, quá trình bị ngắt tiếp tục với lời gọi hệ thống. Các điểm trưng dụng chỉ có thể được đặt tại vị trí "an toàn" trong nhân-nơi mà những cấu trúc dữ liệu hiện tại không được cập nhật. Ngay cả với độ trễ điều phối trưng dụng có thể lớn vì chỉ một vài điểm trưng dụng có thể được thêm vào nhân trong thực tế.

Thuviendientu.org

Một phương pháp khác để giải quyết sự trưng dụng là làm toàn bộ nhân có thể trưng dụng. Để đảm bảo các hoạt động thực hiện đúng, tất cả cấu trúc dữ liệu nhân phải được bảo vệ thông qua việc sử dụng các cơ chế đồng bộ hóa. Với phương pháp này, nhân luôn có thể trưng dụng vì bất cứ dữ liệu nhân được cập nhật được bảo vệ từ việc sửa đổi bởi quá trình có độ ưu tiên cao. Đây là một phương pháp hiệu quả nhất trong việc sử dụng rộng rãi; nó được dùng trong Solaris 2.



Hình 0-7 Độ trễ gửi

Nhưng điều gì xảy ra nếu quá trình có độ ưu tiên cao cần đọc hay sửa đổi dữ liệu nhân hiện đang được truy xuất bởi quá trình khác có độ ưu tiên thấp hơn? Quá trình có độ ưu tiên cao đang chờ quá trình có độ ưu tiên thấp kết thúc. Trường hợp này được gọi là đảo ngược độ ưu tiên (prioprity inversion). Thật vậy, một chuỗi các quá trình đang truy xuất tài nguyên mà quá trình có độ ưu tiên cao cần. Vấn đề này có thể giải quyết bằng giao thức kế thừa độ ưu tiên (priority-inheritance protocol) trong đó tất cả quá trình này (các quá trình này truy xuất tài nguyên mà quá trình có độ ưu tiên cao cần) kế thừa độ ưu tiên cao cho đến khi chúng được thực hiện với tài nguyên trong câu hỏi. Khi chúng kết thúc, độ ưu tiên của chúng chuyển trở lại giá trị ban đầu của nó.

Trong hình IV.7, chúng ta hiển thị sự thay đổi của độ trễ điều phối. Giai đoạn xung đột (conflict phase) của độ trễ điều phối có hai thành phần:

- Sự trưng dụng của bất cứ quá trình nào đang chạy trong nhân
- Giải phóng tài nguyên các quá trình có độ ưu tiên thấp được yêu cầu bởi quá trình có độ ưu tiên cao

Thí dụ, trong Solaris 2 độ trễ điều phối với sự trưng dụng bị vô hiệu hóa khi vượt qua 100 mili giây. Tuy nhiên, độ trễ điều phối với sự trưng dụng được cho phép thường được giảm xuống tới 2 mili giây.

Thuviendientu.org

VIII Đánh giá giải thuật

Chúng ta chọn một giải thuật định thời CPU cho một hệ thống xác định như thế nào? Có nhiều giải thuật định thời, mỗi giải thuật với các tham số của riêng nó. Do đó, chọn một giải thuật có thể là khó.

Vấn đề đầu tiên là định nghĩa các tiêu chuẩn được dùng trong việc chọn một giải thuật. Các tiêu chuẩn thường được định nghĩa trong thuật ngữ khả năng sử dụng CPU, thời gian đáp ứng hay thông lượng. Để chọn một giải thuật, trước hết chúng ta phải định nghĩa trọng số quan trọng của các thước đo này. Tiêu chuẩn của chúng ta có thể gồm các thước đo như:

- Khả năng sử dụng CPU tối đa dưới sự ràng buộc thời gian đáp ứng tối đa là 1 giây.
- Thông lượng tối đa như thời gian hoàn thành (trung bình) tỉ lệ tuyến tính với tổng số thời gian thực thi.

Một khi các tiêu chuẩn chọn lựa được định nghĩa, chúng ta muốn đánh giá các giải thuật khác nhau dưới sự xem xét. Chúng ta mô tả các phương pháp đánh giá khác nhau trong những phần dưới đây

VIII.1 Mô hình quyết định

Một loại quan trọng của phương pháp đánh giá được gọi là đánh giá phân tích (analytic evaluation). Đánh giá phân tích dùng giải thuật được cho và tải công việc hệ thống để tạo ra công thức hay số đánh giá năng lực của giải thuật cho tải công việc đó.

Một dạng đánh giá phân tích là **mô hình xác định** (deterministic modeling). Phương pháp này lấy tải công việc đặc biệt được xác định trước và định nghĩa năng lực của mỗi giải thuật cho tải công việc đó.

Thí dụ, giả sử rằng chúng ta có tải công việc được hiển thị trong bảng dưới. Tất cả 5 quá trình đến tại thời điểm 0 trong thứ tự được cho, với chiều dài của thời gian chu kỳ CPU được tính bằng mili giây.

Quá trình	Thời gian xử lý
P1	10
P2	29
P3	3
P4	7
P5	12

Xét giải thuật định thời FCFS, SJF và RR (định mức thời gian=10 mili giây) cho tập hợp quá trình này. Giải thuật nào sẽ cho thời gian chờ đợi trung bình tối thiểu?

Đối với giải thuật FCFS, chúng ta sẽ thực thi các quá trình này như sau:

P1	P2	P3	P4	P5
0 10	39	42	49	61

Thời gian chờ đợi là 0 mili giây cho quá trình P_1, 32 mili giây cho quá trình P_2, 39 giây cho quá trình P_3, 42 giây cho quá trình P_4 và 49 mili giây cho quá trình P_5. Do đó, thời gian chờ đợi trung bình là (0 + 10 + 39 + 42 + 49)/5= 28 mili giây.

Thuviendientu.org

Với định thời không trưng dụng SJF, chúng ta thực thi các quá trình như sau:

P3	P4	P1	P5	P2
0	3 10	20	32	61

Thời gian chờ đợi là 10 mili giây cho quá trình P_1, 32 mili giây cho quá trình P_2, 0 mili giây cho quá trình P_3, 3 mili giây cho quá trình P_4, và 20 giây cho quá trình P_5. Do đó, thời gian chờ đợi trung bình là (10 + 32 + 0 + 3 + 20)/5= 13 mili giây.

Với giải thuật RR, chúng ta thực thi các quá trình như sau:

P1	P2	P3	P4	P5	P2	P5	P2
0 10	20	23	30	40	50	52	61

Thời gian chờ đợi là 0 mili giây cho quá trình P_1, 32 mili giây cho quá trình P_2, 20 mili giây cho quá trình P_3, 23 mili giây cho quá trình P_4, và 40 mili giây cho quá trình P_5. Do đó, thời gian chờ đợi trung bình là (0 + 32 + 20 + 23 + 40)/5 = 23 mili giây.

Trong trường hợp này, chúng ta thấy rằng, chính sách SJF cho kết quả ít hơn ½ thời gian chờ đợi trung bình đạt được với giải thuật FCFS; giải thuật RR cho chúng ta giá trị trung gian.

Mô hình xác định là đơn giản và nhanh. Nó cho các con số chính xác, cho phép các giải thuật được so sánh với nhau. Tuy nhiên, nó đòi hỏi các số đầu vào chính xác và các trả lời của nó chỉ áp dụng cho những trường hợp đó. Việc dùng chủ yếu của mô hình xác định là mô tả giải thuật định thời và cung cấp các thí dụ. Trong các trường hợp, chúng ta đang chạy cùng các chương trình lặp đi lặp lại và có thể đo các yêu cầu xử lý của chương trình một cách chính xác, chúng ta có thể dùng mô hình xác định để chọn giải thuật định thời. Qua tập hợp các thí dụ, mô hình xác định có thể hiển thị khuynh hướng được phân tích và chứng minh riêng. Thí dụ, có thể chứng minh rằng đối với môi trường được mô tả (tất cả quá trình và thời gian của chúng sẵn dùng tại thời điểm 0), chính sách SJF sẽ luôn cho kết quả thời gian chờ đợi là nhỏ nhất.

Tuy nhiên, nhìn chung mô hình xác định quá cụ thể và yêu cầu tri thức quá chính xác để sử dụng nó một cách có ích.

VIII.2 Mô hình hàng đợi

Các quá trình được chạy trên nhiều hệ thống khác nhau từ ngày này sang ngày khác vì thế không có tập hợp quá trình tĩnh (và thời gian) để dùng cho mô hình xác định. Tuy nhiên, những gì có thể được xác định là sự phân bổ chu kỳ CPU và I/O. Sự phân bổ này có thể được đo và sau đó được tính xấp xỉ hay ước lượng đơn giản. Kết quả là một công thức toán mô tả xác suất của một chu kỳ CPU cụ thể. Thông thường, sự phân bổ này là hàm mũ và được mô tả bởi giá trị trung bình của nó. Tương tự, sự phân bổ thời gian khi các quá trình đến trong hệ thống-phân bổ thời gian đến-phải được cho.

Hệ thống máy tính được mô tả như một mạng các server. Mỗi server có một hàng đợi cho các quá trình. CPU là một server với hàng đợi sẵn sàng của nó, như là một hệ thống nhập/xuất với các hàng đợi thiết bị. Biết tốc độ đến và tốc độ phục vụ, chúng ta có thể tính khả năng sử dụng, chiều dài hàng đợi trung bình, thời gian chờ

Thuviendientu.org

trung bình,..Lĩnh vực nghiên cứu này được gọi là phân tích mạng hàng đợi (queueing-network analysis).

Thí dụ, gọi n là chiều dài hàng đợi trung bình (ngoại trừ các quá trình đang được phục vụ), gọi *W* là thời gian chờ đợi trung bình trong hàng đợi và λ là tốc độ đến trung bình cho các quá trình mới trong hàng đợi (chẳng hạn 3 quá trình trên giây). Sau đó, chúng ta mong đợi trong suốt thời gian *W* một quá trình chờ, λ x *W* các quá trình mới sẽ đến trong hàng đợi. Nếu hệ thống ở trong trạng thái đều đặn thì số lượng quá trình rời hàng đợi phải bằng số lượng quá trình đến. Do đó,

$$n = \lambda \times W$$

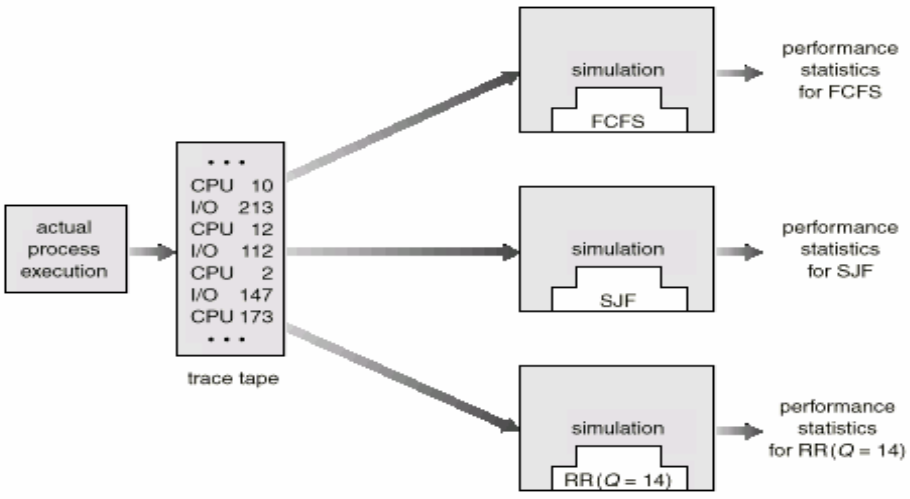
Công thức này được gọi là công thức Little. Công thức Little là đặc biệt có ích vì nó phù hợp cho bất cứ giải thuật định thời và sự phân bổ các quá trình đến.

Chúng ta sử dụng công thức Little để tính một trong ba biến, nếu chúng ta biết hai biến khác. Thí dụ, nếu chúng ta biết có 7 quá trình đến mỗi giây (trung bình) và thường có 14 quá trình trong hàng đợi thì chúng ta có thể tính thời gian chờ đợi trung bình trên mỗi quá trình là 2 giây.

Phân tích hàng đợi có thể có ích trong việc so sánh các giải thuật định thời nhưng nó cũng có một số giới hạn. Hiện nay, các loại giải thuật và sự phân bổ được quản lý là tương đối giới hạn. Tính toán của các giải thuật phức tạp và sự phân bổ là rất khó để thực hiện. Do đó, phân bổ đến và phục vụ thường được định nghĩa không thực tế, nhưng dễ hướng dẫn về mặt tính toán. Thông thường cần thực hiện một số giả định độc lập có thể không chính xác. Do đó, để chúng sẽ có thể tính câu trả lời, các mô hình hàng đợi thường chỉ xấp xỉ hệ thống thật. Vì thế, độ chính xác của các kết quả tính toán có thể là sự nghi vấn.

VIII.3 Mô phỏng

Để đạt được sự đánh giá các giải thuật định thời chính xác hơn, chúng ta có thể dùng mô phỏng (simulations). Mô phỏng liên quan đến lập trình một mô hình hệ thống máy tính. Cấu trúc dữ liệu phần mềm biểu diễn các thành phần quan trọng của hệ thống. Bộ mô phỏng có một biến biểu diễn đồng hồ; khi giá trị của biến này tăng, bộ mô phỏng sửa đổi trạng thái hệ thống để phản ánh các hoạt động của các thiết bị, các quá trình và các bộ định thời. Khi sự mô phỏng thực thi, các thống kê hiển thị năng lực của giải thuật được tập hợp và in ra.



Hình 0-8 Đánh giá các bộ định thời CPU bằng mô phỏng

Thuviendientu.org

Dữ liệu để định hướng sự mô phỏng có thể được sinh ra trong nhiều cách. Cách thông dụng nhất dùng bộ sinh số ngẫu nhiên, được lập trình để sinh ra các quá trình, thời gian chu kỳ CPU, đến, đi của quá trình,..dựa trên phân bổ xác suất. Sự phân bổ này có thể được định nghĩa dạng toán học (đồng nhất, hàm mũ, phân bổ Poisson) hay theo kinh nghiệm. Nếu sự phân bổ được định nghĩa theo kinh nghiệm thì các thước đo của hệ thống thật dưới sự nghiên cứu là lấy được. Các kết quả được dùng để định nghĩa sự phân bổ thật sự các sự kiện trong hệ thống thực và sau đó sự phân bổ này có thể được dùng để định hướng việc mô phỏng.

Tuy nhiên, một mô phỏng hướng phân bổ có thể không chính xác do mối quan hệ giữa các sự kiện tiếp theo trong hệ thống thực. Sự phân bổ thường xuyên hiển thị chỉ bao nhiêu sự kiện xảy ra; nó không hiển thị bất cứ thứ gì về thứ tự xảy ra của chúng. Để sửa chữa vấn đề này, chúng ta dùng băng từ ghi vết (trace tapes). Chúng ta tạo một băng từ ghi vết bằng cách giám sát hệ thống thực, ghi lại chuỗi các sự kiện thật (như hình IV.8). Sau đó, thứ tự này được dùng để định hướng việc mô phỏng. Băng từ ghi vết cung cấp cách tuyệt vời để so sánh chính xác hai giải thuật trên cùng một tập hợp dữ liệu vào thật. Phương pháp này có thể cung cấp các kết quả chính xác cho dữ liệu vào của nó.

Tuy nhiên, mô phỏng có thể rất đắt và thường đòi hỏi hàng giờ máy tính để thực hiện. Một mô phỏng chi tiết hơn cung cấp các kết quả chính xác hơn nhưng cũng yêu cầu nhiều thời gian máy tính hơn. Ngoài ra, các băng từ ghi vết có thể yêu cầu lượng lớn không gian lưu trữ. Cuối cùng, thiết kế, mã, gỡ rối của bộ mô phỏng là một tác vụ quan trọng.

VIII.4 Cài đặt

Ngay cả mô phỏng cũng cho độ chính xác có giới hạn. Chỉ có cách chính xác hoàn toàn để đánh giá giải thuật định thời là mã hóa (code) nó, đặt nó vào trong hệ điều hành và xem nó làm việc như thế nào. Tiếp cận này đặt một giải thuật thật sự vào hệ thống thật để đánh giá dưới điều kiện hoạt động thật sự.

Khó khăn chủ yếu là chi phí của tiếp cận. Chi phí bao gồm không chỉ mã hóa giải thuật và sửa đổi hệ điều hành để hỗ trợ nó cũng như các cấu trúc dữ liệu được yêu cầu mà còn phản ứng của người dùng đối với sự thay đổi liên tục hệ điều hành. Hầu hết người dùng không quan tâm việc xây dựng một hệ điều hành tốt hơn; họ chỉ đơn thuần muốn biết các quá trình của họ thực thi và dùng các kết quả của chúng. Một hệ điều hành thay đổi liên tục không giúp cho người dùng nhận thấy công việc của họ được thực hiện. Một dạng của phương pháp này được dùng phổ biến cho việc cài đặt máy tính mới. Thí dụ, một tiện ích Web mới có thể mô phỏng tải người dùng được phát sinh trước khi nó "sống" (goes live), để xác định bất cứ hiện tượng thắt cổ chai trong tiện ích và để ước lượng bao nhiêu người dùng hệ thống có thể hỗ trợ.
Một khó khăn khác với bất cứ việc đánh giá giải thuật nào là môi trường trong đó giải thuật được dùng sẽ thay đổi. Môi trường sẽ thay đổi không chỉ trong cách thông thường như những chương trình mới được viết và các loại vấn đề thay đổi, mà còn kết quả năng lực của bộ định thời. Nếu các quá trình được cho với độ ưu tiên ngắn thì người dùng có thể tách các quá trình lớn thành tập hợp các quá trình nhỏ hơn. Nếu quá trình giao tiếp được cho độ ưu tiên vượt qua các quá trình không giao tiếp thì người dùng có thể chuyển tới việc dùng giao tiếp.

Thí dụ, trong DEC TOPS-20, hệ thống được phân loại các quá trình giao tiếp và không giao tiếp một cách tự động bằng cách xem lượng nhập/xuất thiết bị đầu cuối. Nếu một quá trình không có nhập hay xuất tới thiết bị đầu cuối trong khoảng thời gian 1 phút thì quá trình được phân loại là không giao tiếp và được di chuyển tới

Thuviendientu.org

hàng đợi có độ ưu tiên thấp. Chính sách này dẫn đến trường hợp một người lập trình sửa đổi chương trình của mình để viết một ký tự bất kỳ tới thiết bị đầu cuối tại khoảng thời gian đều đặn ít hơn 1 phút. Hệ thống này cho những chương trình này có độ ưu tiên cao mặc dù dữ liệu xuất của thiết bị đầu cuối là hoàn toàn không có ý nghĩa.
Các giải thuật có khả năng mềm dẻo nhất có thể được thay đổi bởi người quản lý hay người dùng. Trong suốt thời gian xây dựng hệ điều hành, thời gian khởi động, thời gian chạy, các biến được dùng bởi các bộ định thời có thể được thay đổi để phản ánh việc sử dụng của hệ thống trong tương lai. Yêu cầu cho việc định thời biểu mềm dẻo là một trường hợp khác mà ở đó sự tách riêng các cơ chế từ chính sách là có ích. Thí dụ, nếu các hóa đơn cần được xử lý và in lập tức nhưng thường được thực hiện như công việc bó có độ ưu tiên thấp, hàng đợi bó được cho tạm thời độ ưu tiên cao hơn. Tuy nhiên, rất ít hệ điều hành chấp nhận loại định thời này.

IX Tóm tắt

Định thời CPU là một tác vụ chọn một quá trình đang chờ từ hàng đợi sẵn sàng và cấp phát CPU tới nó. CPU được cấp phát tới quá trình được chọn bởi bộ cấp phát. Định thời đến trước, được phục vụ trước (FCFS) là giải thuật định thời đơn giản nhất, nhưng nó có thể gây các quá trình ngắn chờ các quá trình quá trình quá dài. Định thời ngắn nhất, phục vụ trước (SJF) có thể tối ưu, cung cấp thời gian chờ đợi trung bình ngắn nhất. Cài đặt định thời SJF là khó vì đoán trước chiều dài của chu kỳ CPU kế tiếp là khó. Giải thuật SJF là trường hợp đặc biệt của giải thuật định thời trưng dụng thông thường. Nó đơn giản cấp phát CPU tới quá trình có độ ưu tiên cao nhất. Cả hai định thời độ ưu tiên và SJF có thể gặp phải trở ngại của việc đói tài nguyên.

Định thời quay vòng (RR) là hợp lý hơn cho hệ thống chia sẻ thời gian. Định thời RR cấp phát CPU tới quá trình đầu tiên trong hàng đợi sẵn sàng cho q đơn vị thời gian, ở đây q là định mức thời gian. Sau q đơn vị thời gian, nếu quá trình này không trả lại CPU thì nó bị chiếm và quá trình này được đặt vào đuôi của hàng đợi sẵn sàng. Vấn đề quan trọng là chọn định mức thời gian. Nếu định mức quá lớn, thì định thời RR giảm hơn định thời FCFS ; nếu định mức quá nhỏ thì chi phí định thời trong dạng thời gian chuyển ngữ cảnh trở nên thừa.

Giải thuật FCFS là không ưu tiên; giải thuật RR là ưu tiên. Các giải thuật SJF và ưu tiên có thể ưu tiên hoặc không ưu tiên.

Các giải thuật hàng đợi nhiều cấp cho phép các giải thuật khác nhau được dùng cho các loại khác nhau của quá trình. Chung nhất là hàng đợi giao tiếp ở chế độ hiển thị dùng định thời RR và hàng đợi bó chạy ở chế độ nền dùng định thời FCFS. Hàng đợi phản hồi nhiều cấp cho phép các quá trình di chuyển từ hàng đợi này sang hàng đợi khác.

Vì có nhiều giải thuật định thời sẵn dùng, chúng ta cần các phương pháp để chọn giữa chúng. Các phương pháp phân tích dùng cách thức phân tích toán học để xác định năng lực của giải thuật. Các phương pháp mô phỏng xác định năng lực bằng cách phỏng theo giải thuật định thời trên những mẫu ‘đại diện’ của quá trình và tính năng lực kết quả.

Thuviendientu.org

ĐỒNG BỘ HOÁ QUÁ TRÌNH

I Mục tiêu

Sau khi học xong chương này, người học nắm được những kiến thức sau:

- Hiểu vấn đề vùng tương trục
- Hiểu cơ chế hoạt động hiệu báo Semaphores để đồng bộ hóa quá trình
- Hiểu cơ chế hoạt động của Monitors để đồng bộ hóa quá trình
- Vận dụng các giải pháp để giải quyết các bài toán đồng bộ hóa cơ bản

II Giới thiệu

Một **quá trình hợp tác** là một quá trình có thể gây ảnh hưởng hay bị ảnh hưởng tới quá trình khác đang thực thi trong hệ thống. Các quá trình hợp tác có thể chia sẻ trực tiếp không gian địa chỉ luận lý (mã và dữ liệu), hay được phép chia sẻ dữ liệu thông qua các tập tin. Trường hợp đầu đạt được thông qua việc sử dụng các quá trình có trọng lượng nhẹ hay luồng. Truy xuất đồng hành dữ liệu được chia sẻ có thể dẫn tới việc không đồng nhất dữ liệu. Trong chương này chúng ta sẽ thảo luận các cơ chế đảm bảo việc thực thi có thứ tự của các quá trình hợp tác chia sẻ không gian địa chỉ để tính đúng đắn của dữ liệu luôn được duy trì.

III Tổng quan

Trong chương trước, chúng ta phát triển một mô hình hệ thống chứa số lượng quá trình hợp tác tuần tự, tất cả chúng chạy bất đồng bộ và có thể chia sẻ dữ liệu. Chúng ta hiển thị mô hình này với cơ chế vùng đệm có kích thước giới hạn, được đại diện cho hệ điều hành.

Chúng ta xét giải pháp bộ nhớ được chia sẻ cho bài toán vùng đệm có kích thước giới hạn. Giải pháp này cho phép có nhiều nhất BUFFER_SIZE –1 sản phẩm trong vùng đệm tại cùng thời điểm. Giả sử rằng chúng ta muốn hiệu chỉnh giải thuật để giải quyết sự thiếu sót này. Một khả năng là thêm một biến đếm số nguyên **counter**, được khởi tạo bằng 0. counter được tăng mỗi khi chúng ta thêm một sản phẩm tới vùng đệm và bị giảm mỗi khi chúng ta lấy một sản phẩm ra khỏi vùng đệm. Mã cho quá trình người sản xuất có thể được hiệu chỉnh như sau:

```
while (1){/*tạo sản phẩm trong nextProduced*/
        while (counter==BUFFER_SIZE);  /*không làm gì cả*/
        buffer[in] = nextProduced;
        in          = ( in + 1 ) % BUFFER_SIZE;
        counter++;
}
```

Mã cho quá trình người tiêu dùng có thể được hiệu chỉnh như sau:

```
while (1){
        while (counter == 0)  ;                 /*không làm gì cả*/
        nextConsumed = buffer[out];
```

Thuviendientu.org


```
        out             = ( out + 1 ) % BUFFER_SIZE;
        counter--;
        /*tiêu thụ sản phẩm trong nextConsumed*/
}
```

Mặc dù cả hai thủ tục người sản xuất và người tiêu dùng thực thi đúng khi tách biệt nhau nhưng chúng không thực hiện đúng chức năng khi thực thi đồng hành. Như minh hoạ dưới đây, giả sử rằng giá trị của biến counter hiện tại là 5 và thủ tục người sản xuất và người tiêu dùng thực thi đồng hành câu lệnh "counter++" và "counter--". Theo sau việc thực thi hai câu lệnh này, giá trị của biến counter có thể là 4, 5 hay 6! Kết quả chỉ đúng khi biến counter==5, được tạo ra đúng nếu quá trình người sản xuất và người tiêu dùng thực thi riêng biệt.

Chúng ta có thể minh hoạ giá trị của counter có thể không đúng như sau. Chú ý, câu lệnh "counter++" có thể được cài đặt bằng ngôn ngữ máy (trên một máy điển hình) như sau:

$register_1$ = counter
$register_1$ = $register_1$ + 1
counter = $register_1$

Ở đây $register_1$ là một thanh ghi CPU cục bộ. Tương tự, câu lệnh "counter--" được cài đặt như sau:

$register_2$ = counter
$register_2$ = $register_2$ - 1
counter = $register_2$

Ở đây $register_2$ là thanh ghi CPU cục bộ. Dù là $register_1$ và $register_2$ có thể dùng cùng thanh ghi vật lý, nhưng nội dung của thanh ghi sẽ được lưu lại và lấy lại bởi bộ quản lý ngắt.

Thực thi đồng hành của "counter++" và "counter--" là tương tự như thực thi tuần tự ở đây các câu lệnh cấp thấp hơn được hiện diện trước bị phủ lắp trong thứ tự bất kỳ (nhưng thứ tự bên trong mỗi câu lệnh cấp cao được lưu giữ). Một sự phủ lắp là:

T_0: producer	thực thi	$register_1$ = counter	{$register_1$ = 5}
T_1: producer	thực thi	$register_1$ = $register_1$ + 1	{$register_1$ = 6}
T_2: consumer	thực thi	$register_2$ = counter	{$register_2$ = 5}
T_3: consumer	thực thi	$register_2$ = $register_2$ – 1	{$register_2$ = 4}
T_4: producer	thực thi	counter = $register_1$	{counter = 6}
T_5: consumer	thực thi	counter = $register_2$	{counter = 4}

Chú ý rằng, chúng ta xem xét tình trạng không đúng "counter==4" theo đó có 4 vùng đệm đầy, nhưng thực tế khi đó có 5 vùng đệm đầy. Nếu chúng đổi ngược lại thứ tự của câu lệnh T_4 và T_5, chúng ta sẽ có trạng thái không đúng "counter ==6".

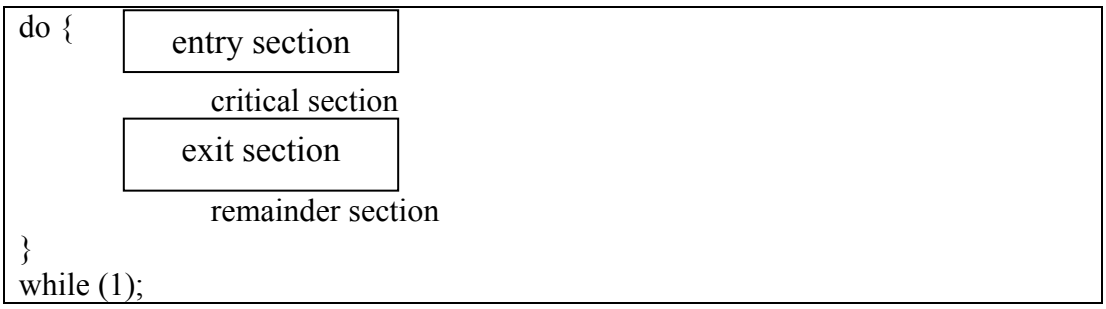
Chúng ta đi đến trạng thái không đúng này vì chúng ta cho phép cả hai quá trình thao tác đồng thời trên biến counter. Trường hợp tương tự, ở đây nhiều quá trình truy xuất và thao tác cùng dữ liệu đồng hành và kết quả của việc thực thi phụ thuộc vào thứ tự xác định trong đó việc truy xuất xảy ra, được gọi là **điều kiện cạnh tranh** (race condition). Để ngăn chặn điều kiện cạnh tranh ở trên, chúng ta cần đảm bảo rằng chỉ một quá trình tại một thời điểm có thể được thao tác biến counter. Để thực hiện việc đảm bảo như thế, chúng ta yêu cầu một vài hình thức đồng bộ hoá quá trình. Những

Thuviendientu.org

trường hợp như thế xảy ra thường xuyên trong các hệ điều hành khi các phần khác nhau của hệ thống thao tác các tài nguyên và chúng ta muốn các thay đổi không gây trở ngại một sự thay đổi khác. Phần chính của chương này là tập trung vào vấn đề đồng bộ hoá và cộng tác quá trình.

IV Vấn đề vùng tương trục

Xét một hệ thống gồm n quá trình (P_0, P_1, … ,P_{n-1}). Mỗi quá trình có một phân đoạn mã, được gọi là **vùng tương trục** (critical section), trong đó quá trình này có thể thay đổi những biến dùng chung, cập nhật một bảng, viết đến tập tin,.. Đặc điểm quan trọng của hệ thống là ở chỗ, khi một quá trình đang thực thi trong vùng tương trục, không có quá trình nào khác được phép thực thi trong vùng tương trục của nó. Do đó, việc thực thi của các vùng tương trục bởi các quá trình là sự loại trừ hỗ tương. Vấn đề vùng tương trục là thiết kế một giao thức mà các quá trình có thể dùng để cộng tác. Mỗi quá trình phải yêu cầu quyền để đi vào vùng tương trục của nó. Vùng mã thực hiện yêu cầu này là **phần đi vào** (entry section). Vùng tương trục có thể được theo sau bởi một **phần kết thúc** (exit section). Mã còn lại là **phần còn lại** (remainder section).



Hình 0-1 Cấu trúc chung của một quá trình điển hình Pi

Một giải pháp đối với vấn đề vùng tương trục phải thoả mãn ba yêu cầu sau:

- **Loại trừ hỗ tương** (Mutual Exclusion): Nếu quá trình P_i đang thực thi trong vùng tương trục của nó thì không quá trình nào khác đang được thực thi trong vùng tương trục đó.
- **Tiến trình** (Progress): nếu không có quá trình nào đang thực thi trong vùng tương trục và có vài quá trình muốn vào vùng tương trục thì chỉ những quá trình không đang thực thi phần còn lại mới có thể tham gia vào việc quyết định quá trình nào sẽ đi vào vùng tương trục tiếp theo và chọn lựa này không thể trì hoãn vô hạn định.
- **Chờ đợi có giới hạn** (bounded wait): giới hạn số lần các quá trình khác được phép đi vào miền tương trục sau khi một quá trình thực hiện yêu cầu để đi vào miền tương trục của nó và trước khi yêu cầu đó được gán.

Chúng ta giả sử rằng mỗi quá trình đang thực thi với tốc độ khác 0. Tuy nhiên, chúng ta có thể thực hiện rằng không có giả thuyết nào được quan tâm về tốc tương đối của n quá trình.

Trong phần tiếp theo chúng ta nghiên cứu để nắm được các giải pháp thoả ba yêu cầu này. Những giải pháp này không quan tâm đến các chỉ thị phần cứng hay số lượng bộ xử lý mà phần cứng hỗ trợ. Tuy nhiên chúng ta giả sử rằng những chỉ thị ngôn ngữ máy cơ bản (chỉ thị cơ bản như **load**, **store** và **test**) được thực hiện mang

Thuviendientu.org

tính nguyên tử (atomically). Nghĩa là, nếu hai chỉ thị như thế được thực thi đồng hành thì kết quả tương tự như thực thi tuần tự trong thứ tự không xác định. Do đó, nếu chỉ thị **load** và **store** được thực thi đồng hành thì **load** sẽ nhận giá trị cũ hay mới như không có sự kết hợp vừa cũ vừa mới.

Khi trình bày một giải thuật, chúng ta định nghĩa chỉ những biến được dùng cho mục đích đồng bộ và mô tả chỉ một quá trình điển hình P_i mà cấu trúc của nó được hiển thị trong hình V.1. Phần đi vào và kết thúc được bao trong hình chữ nhật để nhấn mạnh các đoạn mã quan trọng.

```
do {
        while (turn!=i) ;
                critical section
        turn = j;
                remainder section
}
while (1);
```

Hình 0-2-Cấu trúc của quá trình P_i trong giải thuật 1

V Giải pháp

Có nhiều giải pháp để thực hiện việc loại trừ hỗ tương. Các giải pháp này, tuỳ thuộc vào cách tiếp cận trong xử lý của quá trình bị khoá, được phân biệt thành hai lớp: chờ đợi bận (busy waiting) và nghẽn và đánh thức (sleep and wakeup)

V.1 Giải pháp "chờ đợi bận"

V.1.1 Giải pháp hai quá trình (two-Process Solution)

Trong phần này, chúng ta giới hạn việc quan tâm tới những giải thuật có thể áp dụng chỉ hai quá trình cùng một lúc. Những quá trình này được đánh số P_0 và P_1. Để thuận lợi, khi trình bày P_i, chúng ta dùng P_j để chỉ quá trình còn lại, nghĩa là j = 1 – i

.V.1.1.1 Giải thuật 1

Tiếp cận đầu tiên của chúng ta là để hai quá trình chia sẻ một biến số nguyên chung **turn** được khởi tạo bằng 0 (hay 1). Nếu turn == 0 thì quá trình P_i được phép thực thi trong vùng tương trục của nó. Cấu trúc của quá trình P_i được hiển thị trong Hình V.-2.

Giải pháp này đảm bảo rằng chỉ một quá trình tại một thời điểm có thể ở trong vùng tương trục của nó. Tuy nhiên, nó không thoả mãn yêu cầu tiến trình vì nó yêu cầu sự thay đổi nghiêm khắc của các quá trình trong việc thực thi của vùng tương trục. Thí dụ, nếu turn == 0 và P_1 sẵn sàng đi vào vùng tương trục của nó thì P_1 không thể đi vào vùng tương trục thậm chí khi P_0 đang ở trong phần còn lại của nó.

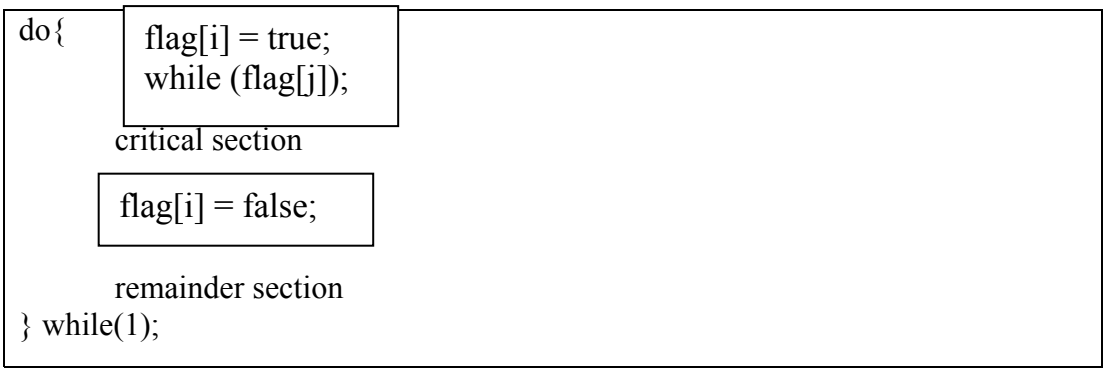
.V.1.1.2 Giải thuật 2

Vấn đề với giải thuật 1 là nó không giữ lại đủ thông tin về trạng thái của mỗi quá trình; nó nhớ chỉ quá trình nào được phép đi vào miền tương trục. Để giải quyết vấn đề này, chúng ta có thể thay thế biến turn với mảng sau:

```
Boolean flag[2];
```

Thuviendientu.org

Các phần tử của mảng được khởi tạo tới flase. Nếu flag[i] là true, giá trị này hiển thị rằng P_i sẵn sàng đi vào vùng tương trục. Cấu trúc của quá trình P_i được hiển thị trong hình V.-3 dưới đây:



Hình 0-3 –Cấu trúc của quá trình P_i trong giải thuật 2

Trong giải thuật này, quá trình P_i trước tiên thiết lập flag[i] tới true, hiển thị rằng nó sẵn sàng đi vào miền tương trục. Sau đó, P_i kiểm tra rằng quá trình quá trình P_j cũng không sẵn sàng đi vào miền tương trục của nó. Nếu P_j sẵn sàng thì P_i sẽ chờ cho tới khi P_j hiển thị rằng nó không còn cần ở trong vùng tương trục nữa (nghĩa là cho tới khi flag[j] là false). Tại thời điểm này, P_i sẽ đi vào miền tương trục. Thoát ra khỏi miền tương trục, P_i sẽ đặt flag[i] là false, cho phép quá trình khác (nếu nó đang chờ) đi vào miền tương trục của nó.

Trong giải pháp này, yêu cầu loại trừ hỗ tương sẽ được thoả mãn. Tuy nhiên, yêu cầu tiến trình không được thoả mãn. Để minh hoạ vấn đề này, chúng ta xem xét thứ tự thực thi sau:

T_0: P_0 thiết lập flag[0] = true;
T_1: P_1 thiết lập flag[1] = true;

Bây giờ P_0 và P_1 được lập mãi mãi trong câu lệnh while tương ứng của chúng.

Giải thuật này phụ thuộc chủ yếu vào thời gian chính xác của hai quá trình. Thứ tự này được phát sinh trong môi trường nơi có nhiều bộ xử lý thực thi đồng hành hay nơi một ngắt (chẳng hạn như một ngắt định thời) xảy ra lập tức sau khi bước T_0 được thực thi và CPU được chuyển từ một quá trình này tới một quá trình khác.

Chú ý rằng chuyển đổi thứ tự của các chỉ thị lệnh để thiết lập flag[i] và kiểm tra giá trị của flag[j] sẽ không giải quyết vấn đề của chúng ta. Hơn nữa chúng ta sẽ có một trường hợp đó là hai quá trình ở trong vùng tương trục cùng một lúc, vi phạm yêu cầu loại trừ hỗ tương.

.V.1.1.3 Giải thuật 3

Giải thuật 3 còn gọi là giải pháp Peterson. Bằng cách kết hợp hai ý tưởng quan trọng trong giải thuật 1 và 2, chúng ta đạt được một giải pháp đúng tới với vấn đề vùng tương trục, ở đó hai yêu cầu được thoả. Các quá trình chia sẻ hai biến:

Boolean flag[2]
Int turn;

Khởi tạo flag[0] = flag[1] = false và giá trị của turn là không xác định (hoặc là 0 hay 1). Cấu trúc của quá trình P_i được hiển thị trong hình sau:

Thuviendientu.org


```
do{
        flag[i] = true;
        turn = j;
        while (flag[j] &&turn ==j);
        critical section
        flag[i] = false;
        remainder section
} while (1);
```

Hình 0-4 Cấu trúc của quá trình P_i trong giải thuật 3

Để đi vào miền tương trục, quá trình P_i trước tiên đặt flag[i] là true sau đó đặt turn tới giá trị j, do đó xác định rằng nếu quá trình khác muốn đi vào miền tương trục nó. Nếu cả hai quá trình đi vào miền tương trục cùng một lúc turn sẽ đặt cả hai i và j tại xấp xỉ cùng một thời điểm. Chỉ một trong hai phép gán này là kết quả cuối cùng. Giá trị cuối cùng của turn quyết định quá trình nào trong hai quá trình được cho phép đi vào miền tương trục trước.

Bây giờ chúng ta chứng minh rằng giải pháp này là đúng. Chúng ta cần hiển thị rằng:

1) Loại trừ hỗ tương được bảo toàn
2) Yêu cầu tiến trình được thoả
3) Yêu cầu chờ đợi có giới hạn cũng được thoả

Chứng minh thuộc tính 1, chúng ta chú ý rằng mỗi P_i đi vào miền tương trục của nó chỉ nếu flag[j] ==false hay turn ==i. Cũng chú ý rằng, nếu cả hai quá trình có thể đang thực thi trong vùng tương trục của chúng tại cùng thời điểm thì flag[0] == flag[1] ==true. Hai nhận xét này ngụ ý rằng P_0 và P_1 không thể thực thi thành công trong vòng lặp while của chúng tại cùng một thời điểm vì giá trị turn có thể là 0 hay 1. Do đó, một trong các quá trình-P_j phải được thực thi thành công câu lệnh while, ngược lại P_i phải thực thi ít nhất câu lệnh bổ sung ("turn==j"). Tuy nhiên, vì tại thời điểm đó, flag[j] ==true và turn ==j, và điều kiện này sẽ không đổi với điều kiện là P_j ở trong vùng miền tương trục của nó, kết quả sau việc loại trừ hỗ tương được bảo vệ

```
do {
        flag[i] = true;
        turn = j;
        while (flag[j] && turn ==j);
            critical section
        flag[i] = false;
          Remainder section
}while (1);
```

Hình 0-5-Cấu trúc của quá trình Pi trong giải thuật 3

Để chứng minh thuộc tính 2 và 3, chúng ta chú ý rằng một quá trình P_i có thể được ngăn chặn từ việc đi vào miền tương truc chỉ nếu nó bị kẹt trong vòng lặp while với điều kiện flag[j] == true và turn == j. Nếu P_j không sẵn sàng đi vào miền tương trục thì flag[j] == false và P_i có thể đi vào miền tương trục của nó. Nếu P_j đặt flag[j] là true và nó cũng đang thực thi trong câu lệnh while của nó thì turn == i hay turn == j. Nếu turn == i thì P_i sẽ đi vào miền tương trục. Nếu turn ==j thì P_j sẽ đi vào miền tương trục. Tuy nhiên, một khi P_j ở trong vùng tương trục của nó thì nó sẽ đặt lại

Thuviendientu.org

flag[j] tới false, cho phép P_i đi vào miền tương trục của nó. Nếu P_j đặt lại flag[j] tới true, nó cũng phải đặt turn tới i. Do đó, vì P_i không thay đổi giá trị của biến turn trong khi thực thi câu lệnh while, nên P_i sẽ đi vào miền tương trục (tiến trình) sau khi nhiều nhất chỉ P_j đi vào (chờ có giới hạn).

V.1.2 Giải pháp nhiều quá trình

Giải thuật 3 giải quyết vấn đề miền tương trục cho hai quá trình. Bây giờ chúng ta phát triển một giải thuật để giải quyết vấn đề miền tương trục cho n quá trình. Giải thuật này được gọi là giải thuật Bakery và nó dựa trên cơ sở của giải thuật định thời thường được dùng trong cửa hiệu bánh mì, cửa hàng kem,..nơi mà thứ tự rất hỗn độn. Giải thuật này được phát triển cho môi trường phân tán, nhưng tại thời điểm này chúng ta tập trung chỉ những khía cạnh của giải thuật liên quan tới môi trường tập trung.

Đi vào một cửa hàng, mỗi khách hàng nhận một số. Khách hàng với số thấp nhất được phục vụ tiếp theo. Tuy nhiên, giải thuật Bakery không thể đảm bảo hai quá trình (khách hàng) không nhận cùng số. Trong trường hợp ràng buộc, một quá trình với tên thấp được phục vụ trước. Nghĩa là, nếu P_i và P_j nhận cùng một số và nếu (i < j) thì P_i được phục vụ trước. Vì tên quá trình là duy nhất và được xếp thứ tự nên giải thuật là hoàn toàn mang tính "may rủi" (deterministic).

Cấu trúc dữ liệu chung là

```
boolean choosing[n];
int     number[n];
```

Đầu tiên, các cấu trúc dữ liệu này được khởi tạo tới false và 0 tương ứng. Để tiện dụng, chúng ta định nghĩa các ký hiệu sau:

- (a, b) < (c, d) nếu a< c hay nếu a==c và b< d
- $\max(a_0,\ldots,a_{n-1})$ là số $k \geq a_i$ với i = 0,…,n-1

Cấu trúc của quá trình P_i được dùng trong giải thuật Bakery, được hiển thị trong hình dưới đây.

```
do {
    choosing[i] = true;
    number[i] = max(number[0], number[i],…,number[n-1]) + 1;
    choosing[i] = false;
    for (j=0; j < n; j++){
            while (choosing[j]);
            while ((number[j]!=0)&&((number[ j ], j ) <(number[i], i)));
    }
    Critical section
    Number[i] = 0;
}   While (1);
```

Hình 0-6 Cấu trúc của giải thuật P_i trong giải thuật Bakery

Kết quả được cho này thể hiện rằng loại trừ hỗ tương được tuân theo. Thật vậy, xét P_i trong vùng tương trục của nó và P_k cố gắng đi vào vùng tương trục P_k. Khi quá trình P_k thực thi câu lệnh while thứ hai cho j==i, nhận thấy rằng

- number[i] != 0
- (number[i], i) < (number[k], k).

Thuviendientu.org

Do đó, nó tiếp tục vòng lặp trong câu lệnh while cho đến khi P_i rời khỏi vùng tương trục P_i.

Giải thuật trên đảm bảo rằng yêu cầu về tiến trình, chờ đợi có giới hạn và đảm bảo sự công bằng, vì các quá trình đi vào miền tương trục dựa trên cơ sở tới trước được phục vụ trước.

V.1.3 Phần cứng đồng bộ hoá

Như các khía cạnh khác của phần mềm, các đặc điểm phần cứng có thể làm các tác vụ lập trình dễ hơn và cải tiến tính hiệu quả của hệ thống. Trong phần này, chúng ta trình bày một số chỉ thị phần cứng đơn giản sẵn dùng trên nhiều hệ thống và trình bày cách chúng được dùng hiệu quả trong việc giải quyết vấn đề miền tương trục.

```
boolean        TestAndSet( boolean &target){
        boolean rv = target;
        target      = true;
        return  rv;
}
```

Hình 0-7 Định nghĩa của chỉ thị TestAndSet

Vấn đề miền tương trục có thể được giải quyết đơn giản trong môi trường chỉ có một bộ xử lý nếu chúng ta cấm các ngắt xảy ra khi một biến chia sẻ đang được thay đổi. Trong cách này, chúng ta đảm bảo rằng chuỗi chỉ thị hiện hành có thể được cho phép thực thi trong thứ tự không trưng dụng. Không có chỉ thị nào khác có thể chạy vì thế không có bất cứ sự thay đổi nào có thể được thực hiện trên các biến được chia sẻ.

Tuy nhiên, giải pháp này là không khả thi trong một môi trường có nhiều bộ xử lý. Vô hiệu hoá các ngắt trên đa bộ xử lý có thể mất nhiều thời gian khi một thông điệp muốn truyền qua tất cả bộ xử lý. Việc truyền thông điệp này bị trì hoãn khi đi vào miền tương trục và tính hiệu quả của hệ thống bị giảm.

Do đó nhiều máy cung cấp các chỉ thị phần cứng cho phép chúng ta kiểm tra hay thay đổi nội dung của một từ (word) hay để thay đổi nội dung của hai từ tuân theo tính **nguyên tử** (atomically)-như là một đơn vị không thể ngắt. Chúng ta có thể sử dụng các chỉ thị đặc biệt này để giải quyết vấn đề miền tương trục trong một cách tương đối đơn giản.

Chỉ thị TestAndSet có thể được định nghĩa như trong hình V.-7. Đặc điểm quan trọng của chỉ thị này là việc thực thi có tính nguyên tử. Do đó, nếu hai chỉ thị TestAndSet được thực thi cùng một lúc (mỗi chỉ thị trên một CPU khác nhau), thì chúng sẽ được thực thi tuần tự trong thứ tự bất kỳ.

```
do{
        while (TestAndSet(lock));
        Critical section
        lock:= false
        remainder section
} while (1);
```

Hình 0-8: Cài đặt loại trừ hỗ tương với TestAndSet

Thuviendientu.org

Nếu một máy hỗ trợ chỉ thị TestAndSet thì chúng ta có thể loại trừ hỗ tương bằng cách khai báo một biến khoá kiểu luận lý và được khởi tạo tới false. Cấu trúc của quá trình P_i được hiển thị trong hình V.-9 ở trên.

Chỉ thị Swap được định như hình V.-9 dưới đây, thao tác trên nội dung của hai từ; như chỉ thị TestAndSet, nó được thực thi theo tính nguyên tử.

```
void Swap(boolean &a, boolean &b){
        boolean temp = a;
        a = b;
        b = temp;
}
```

Hình 0-9: Định nghĩa chỉ thị Swap

Nếu một máy hỗ trợ chỉ thị Swap, thì việc loại trừ hỗ tương có thể được cung cấp như sau. Một biến luận lý toàn cục lock được khai báo và được khởi tạo tới false. Ngoài ra, mỗi quá trình cũng có một biến luận lý cục bộ key. Cấu trúc của quá trình P_i được hiển thị trong hình V.-10 dưới đây.

```
do{
        key = true;
        while (key ==  true)  Swap(lock, key);
        Critical section
        lock = false;
        Remainder section
} while(1);
```

Hình 0-10: Cài đặt loại trừ hỗ tương với chỉ thị Swap

Các giải thuật này không thoả mãn yêu cầu chờ đợi có giới hạn. Chúng ta hiển thị giải thuật sử dụng chỉ thị TestAndSet trong hình V.-11 dưới đây. Giải thuật này thoả mãn tất cả các yêu cầu miền tương trục.

```
do{
        Waiting[i] = true;
        key = true;
        while (waiting[i] && key)
                key = TestAndSet(lock);
        waiting[i] = false;
        Critical section
        j = (i + 1) % n;
        while ((j != i ) && !waiting[j])
                j = (j + 1 ) % n;
        if (j == i)
                lock = false;
        else
                waiting[j] = false;
        Remainder section
} while(1);
```

Hình 0-11 Loại trừ hỗ tương chờ đợi có giới hạn với TestAndSet

Thuviendientu.org

Cấu trúc dữ liệu thông thường là:

```
boolean     waiting[n];
boolean     lock;
```

Cấu trúc dữ liệu này được khởi tạo tới false. Để chứng minh rằng loại trừ hỗ tương được thoả, chúng ta chú ý rằng quá trình P_i có thể đưa vào miền tương trục chỉ nếu hoặc waiting[i] ==false hay key == false. Giá trị của key có thể trở thành false chỉ nếu TestAndSet được thực thi. Đối với quá trình đầu tiên, để thực thi TestAndSet sẽ tìm key == false; tất cả quá trình khác phải chờ. Biến waiting[i] có thể trở thành false chỉ nếu quá trình khác rời khởi miền tương trục của nó; chỉ một waiting[i] được đặt false, duy trì yêu cầu loại trừ hỗ tương.

Để chứng minh yêu cầu tiến trình được thoả, chúng ta chú ý rằng các đối số được hiện diện cho việc loại trừ hỗ tương cũng áp dụng được ở đây, vì thế một quá trình thoát khỏi miền tương trục hoặc đặt lock bằng false hay đặt waiting[j] bằng false. Cả hai trường hợp đều cho phép một quá trình đang chờ để đi vào miền tương trục được xử lý.

Để chứng minh yêu cầu chờ đợi được giới hạn được thoả, chúng ta chú ý rằng khi một quá trình rời miền tương trục của nó, nó duyệt qua mảng waiting trong thứ tự tuần hoàn (i + 1, i + 2, …, n – 1, 0, …, i - 1). Nó định rõ quá trình đầu tiên trong thứ tự này mà thứ tự đó ở trong phần đi vào (waiting[j] == true) khi quá trình tiếp theo đi vào miền tương trục. Bất cứ quá trình nào đang chờ để đi vào miền tương trục sẽ thực hiện n – 1 lần. Tuy nhiên, đối với người thiết kế phần cứng, cài đặt các chỉ thị nguyên tử TestAndSet trên bộ đa xử lý không là tác vụ thử nghiệm.

Những giải pháp trên đều phải thực hiện một vòng lặp để kiểm tra liệu nó có được phép vào miền tương trục hay không. Nếu điều kiện chưa thoả, quá trình phải chờ tiếp tục trong vòng lặp kiểm tra này. Các giải pháp buộc quá trình phải liên tục kiểm tra điều kiện để phát hiện thời điểm thích hợp được vào miền tương trục như thế được gọi là các ***giải pháp chờ đợi bận*** "busy waiting". Lưu ý, việc kiểm tra như thế tiêu thụ rất nhiều thời gian sử dụng CPU, do vậy quá trình đang chờ vẫn chiếm dụng CPU. Xu hướng giải quyết vấn đề đồng bộ hoá là nên tránh các giải pháp chờ đợi bận.

V.2 Các giải pháp "SLEEP and WAKEUP"

Để loại bỏ các bất tiện của của giải pháp chờ đợi bận, chúng ta có thể tiếp cận theo hướng cho một quá trình chưa đủ điều kiện vào miền tương trục chuyển sang trạng thái nghẽn, từ bỏ quyền sử dụng CPU. Để thực hiện điều này, cần phải sử dụng các thủ tục do hệ điều hành cung cấp để thay đổi trạng thái quá trình. Hai thủ tục cơ bản SLEEP và WAKEUP thường được sử dụng cho mục đích này.

SLEEP là một lời gọi hệ thống có tác dụng làm "***nghẽn"*** (blocked) hoạt động của quá trình gọi nó và chờ đến khi được một tiến trình khác "***đánh thức".*** Lời gọi hệ thống WAKEUP nhận một tham số duy nhất: quá trình sẽ được kích hoạt trở lại (đặt về trạng thái sẵn sàng).

Ý tưởng sử dụng SLEEP và WAKEUP như sau: khi một quá trình chưa đủ điều kiện vào miền tương trục, nó gọi SLEEP để tự khoá đến khi có một quá trình khác gọi WAKEUP để giải phóng nó. Một quá trình gọi WAKEUP khi ra khỏi miền tương trục để đánh thức một quá trình đang chờ, tạo cơ hội cho quá trình này vào miền tương trục.

```
int     busy;   // 1 nếu miền tương trục đang bị chiếm
int     blocked;        // đếm số lượng quá trình đang bị khoá
```

Thuviendientu.org


```
do{
if (busy) {
        blocked = blocked + 1;
        sleep();
}
else    busy = 1;
}while (1);
Critical section
busy    = 0;
if (blocked){
        wakeup(process);
        blocked = blocked -1;
}
Remainder section
```

Hình 0-12 Cấu trúc chương trình trong giải pháp SLEEP and WAKEUP

Khi sử dụng SLEEP và WAKEUP cần hết sức cẩn thận, nếu không muốn xảy ra tình trạng mâu thuẩn truy xuất trong một vài tình huống như sau: giả sử quá trình A vào miền tương trục, và trước khi nó rời miền tương trục thì quá trình B được kích hoạt. Quá trình B thử vào miền tương trục nhưng nó nhận thấy A đang ở trong đó, do vậy B tăng giá trị biến blocked lên 1 và chuẩn bị gọi SLEEP để tự nghẽn. Tuy nhiên, trước khi B có thể thực hiện SLEEP, quá trình A được kích hoạt trở lại và ra khỏi miền tương trục. Khi ra khỏi miền tương trục, quá trình A nhận thấy có một quá trình đang chờ (blocked=1) nên gọi WAKEUP và giảm giá trị blocked xuống 1. Khi đó tín hiệu WAKEUP sẽ lạc mất do quá trình B chưa thật sự "ngủ" để nhận tín hiệu đánh thức! Khi quá trình B được tiếp tục xử lý, nó mới gọi SLEEP và tự nghẽn vĩnh viễn!

Vấn đề ghi nhận được là tình trạng lỗi này xảy ra do việc kiểm tra trạng thái miền tương trục và việc gọi SLEEP hay WAKEUP là những hành động tách biệt, có thể bị ngắt nửa chừng trong quá trình xử lý, do đó có khi tín hiệu WAKEUP gởi đến một quá trình chưa bị nghẽn sẽ lạc mất. Để tránh những tình huống tương tự, hệ điều hành cung cấp những cơ chế đồng bộ hoá dựa trên ý tưởng của chiến lược "SLEEP and WAKEUP" nhưng chưa được xây dựng bao gồm cả phương tiện kiểm tra điều kiện vào miền tương trục giúp sử dụng an toàn.

V.2.1 Semaphore

Tiếp cận Semaphore được. Dijkstra đề xuất vào năm 1965. Một semaphore S là một biến số nguyên (integer) được truy xuất chỉ thông qua hai thao tác nguyên tử: **wait** và **signal**. Các thao tác này được đặt tên **P** (cho wait - chờ để kiểm tra) và **V** (cho signal- báo hiệu để tăng). Định nghĩa cơ bản của wait trong mã giả là:

```
        wait(S){
                while (S≤0)
                        ;//no-op
                S--;
}
```

Định nghĩa cơ bản của signal trong mã giả là

```
        signal(S){
                S++;
}
```

Thuviendientu.org

Những sửa đổi đối với giá trị integer của semaphore trong các thao tác wait và signal phải được thực thi không bị phân chia. Nghĩa là khi một quá trình sửa đổi giá trị semaphore, không có quá trình nào cùng một lúc có thể sửa đổi cùng biến semaphore đó. Ngoài ra, trong trường hợp của biến wait(S), kiểm tra giá trị integer của S ($S \leq 0$) và sửa đổi có thể của nó (S--) cũng phải được thực thi mà không bị ngắt.

.V.2.1.1 Cách dùng

Chúng ta có thể sử dụng semaphores để giải quyết vấn đề miền tương trục với n quá trình. N quá trình chia sẻ một biến semaphore, mutex (viết tắt từ mutual exclusion) được khởi tạo 1. Mỗi quá trình P_i được tổ chức như được hiển thị trong hình dưới đây.

```
do{
        wait(mutex)
        critical section
        Signal(mutex)
        remainder section
}while(1);
```

Hình 0-13 Cài đặt loại trừ hỗ tương với semaphores

Chúng ta cũng sử dụng semaphores để giải quyết các vấn đề đồng bộ khác nhau. Thí dụ, để xem xét hai quá trình đang thực thi đồng hành: P_1 với câu lệnh S_1 và P_2 với câu lệnh S_2. Giả sử chúng ta yêu cầu rằng S_2 được thực thi chỉ sau khi S_1 hoàn thành. Chúng ta có thể hoàn thành cơ chế này một cách dễ dàng bằng cách P_1 và P_2 chia sẻ một semaphore chung **synch**, được khởi tạo 0 và bằng cách chèn các câu lệnh:

S_1;
signal(sync);

vào quá trình P_1 và các câu lệnh:

wait(synch);
S_2;

vào trong quá trình P_2. Vì synch được khởi tạo 0. P_2 sẽ thực thi S_2 chỉ sau khi P_1 nạp signal(synch) mà sau đó là S_1;

.V.2.1.2 Cài đặt

Nhược điểm chính của các giải pháp loại trừ hỗ tương trong phần V.-5.1 và của semaphore được cho ở đây là tất cả chúng đều đòi hỏi sự chờ đợi bận.Để giải quyết yêu cầu cho việc chờ đợi bận, chúng ta có thể hiệu chỉnh định nghĩa của các thao tác wait và signal của semaphore. Khi một quá trình thực thi thao tác wait và nhận thấy rằng nếu giá trị của semaphore không dương, nó phải chờ. Tuy nhiên, thay vì chờ đợi bận, quá trình có thể nghẽn chính nó. Thao tác nghẽn đặt quá trình vào một hàng đợi gắn liền với semaphore và trạng thái quá trình được chuyển tới trạng thái chờ. Sau đó, điều khiển được chuyển tới bộ định thời biểu và bộ định thời biểu chọn một quá trình khác để thực thi.

Một quá trình bị nghẽn chờ trên biến semaphore nên được khởi động lại khi quá trình khác thực thi thao tác signal. Quá trình được khởi động lại bởi thao tác wakeup và chuyển quá trình từ trạng thái chờ sang trạng thái sẵn sàng. Sau đó, quá trình này được đặt vào hàng đợi sẵn sàng. (CPU có thể hay không thể được chuyển từ quá trình đang chạy tới quá trình sẵn sàng mới nhất phụ thuộc vào giải thuật định thời biểu CPU).

Thuviendientu.org

Để cài đặt semaphore dưới định nghĩa này, chúng ta định nghĩa một semaphore như một cấu trúc được viết bằng C như sau:

```
typedef struct{
        int     value;
        struct process *L;
} semaphore;
```

Mỗi semaphore có một số integer **value** và một danh sách các quá trình **L**. Khi một quá trình phải chờ trên một semaphore, nó được thêm vào danh sách các quá trình L. Một thao tác signal xoá một quá trình ra khỏi danh sách các quá trình đang chờ và đánh thức quá trình đó.

Thao tác semaphore wait bây giờ được định nghĩa như sau:

```
void wait(semaphore S){
        S.value--;
        If (S.value < 0){
                Thêm quá trình này tới danh sách các quá trình S.L;
                block();
        }
}
```

Thao tác semaphore signal bây giờ có thể được định nghĩa như sau:

```
void signal(semaphore S){
        S.value++;
        if(S.value <= 0){
        xoá một quá trình ra khỏi hàng đợi S.L;
        wakeup(P);
        }
}
```

Thao tác **block()** tạm dừng quá trình gọi thao tác đó. Thao tác **wakeup(P)** tiếp tục thực thi quá trình bị nghẽn P. Hai thao tác này được cung cấp bởi hệ điều hành như những lời gọi hệ thống cơ bản.

Chú ý rằng, mặc dù dưới sự định nghĩa kinh điển của semaphores với sự chờ đợi bận là giá trị semaphore không bao giờ âm. Cài đặt này có thể có giá trị semaphore âm. Nếu giá trị semaphore âm thì tính chất trọng yếu của nó là số lượng quá trình chờ trên semaphore đó. Sự thật này là kết quả của việc chuyển thứ tự của việc giảm và kiểm tra trong việc cài đặt thao tác **wait**. Danh sách các quá trình đang chờ có thể được cài đặt dễ dàng bởi một trường liên kết trong mỗi khối điều khiển quá trình (PCB). Mỗi cách thêm và xoá các quá trình từ danh sách, đảm bảo việc chờ đợi có giới hạn sẽ sử dụng hàng đợi FIFO, ở đó semaphore chứa hai con trỏ đầu (head) và đuôi (tail) chỉ tới hàng đợi. Tuy nhiên, danh sách có thể dùng bất cứ chiến lược hàng đợi nào. Sử dụng đúng semaphores không phụ thuộc vào chiến lược hàng đợi cho danh sách semaphore.

Khía cạnh quyết định của semaphores là chúng được thực thi theo tính nguyên tử. Chúng ta phải đảm bảo rằng không có hai quá trình có thể thực thi các thao tác wait và signal trên cùng một semaphore tại cùng một thời điểm. Trường hợp này là vấn đề vùng tương trục và có thể giải quyết bằng một trong hai cách.

Trong môi trường đơn xử lý (nghĩa là chỉ có một CPU tồn tại), đơn giản là chúng ta có thể ngăn chặn các ngắt trong thời gian các thao tác wait và signal xảy ra. Cơ chế này làm việc trong một môi trường đơn xử lý vì một khi ngắt bị ngăn chặn, các chỉ thị từ các quá trình khác không thể được chen vào. Chỉ quá trình đang chạy hiện tại thực thi cho tới khi các ngắt được cho phép sử dụng trở lại và bộ định thời có thể thu hồi quyền điều khiển.

Thuviendientu.org

Trong môi trường đa xử lý, ngăn chặn ngắt không thể thực hiện được. Các chỉ thị từ các quá trình khác nhau (chạy trên các bộ xử lý khác nhau) có thể được chen vào trong cách bất kỳ. Nếu phần cứng không cung cấp bất cứ các chỉ thị đặc biệt nào, chúng ta có thể tận dụng các giải pháp phần cứng phù hợp cho vấn đề vùng tương trục (phần V.-4), ở đó các vùng tương trục chứa cá thủ tục wait và signal.

Vấn đề quan trọng là chúng ta không xoá hoàn toàn chờ đợi bận, với định nghĩa này cho các thao tác wait và signal. Dĩ nhiên là chúng ta xoá chờ đợi bận từ việc đi vào vùng tương trục của chương trình ứng dụng. Ngoài ra, chúng ta hạn chế việc chờ đợi bận chỉ các miền tương trục với thao tác wait và signal và các vùng này là ngắn (nếu được mã hợp lý, chúng nên không quá 10 chỉ thị). Do đó, miền tương trục hầu như không bao giờ bị chiếm và sự chờ đợi bận rất hiếm khi xảy ra và sau đó chỉ cho thời gian ngắn. Một trường hợp hoàn toàn khác xảy ra với những chương trình ứng dụng có miền tương trục dài (vài phút hay thậm chí vài giờ) hay có thể hầu như luôn bị chiếm. Trong trường hợp này, chờ đợi bận là cực kỳ kém hiệu quả.

.V.2.1.3 Sự khoá chết (deadlocks) và đói tài nguyên

Cài đặt semaphore với một hàng đợi có thể dẫn đến trường hợp hai hay nhiều quá trình đang chờ không hạn định một sự kiện mà có thể được gây ra chỉ bởi một trong những quá trình đang chờ. Sự kiện đặt ra là sự thực thi của thao tác signal. Khi một trạng thái như thế xảy ra, những quá trình này được nói là bị khoá chết.

Để hiển thị điều này, chúng ta xét một hệ thống chứa hai quá trình P_0 và P_1, mỗi truy xuất hai semaphore, S và Q, được đặt giá trị 1.

P_0	P_1
wait(S);	wait(Q);
wait(Q);	wait(S);
.	.
.	.
signal(S);	signal(Q);
Signal(Q);	signal(S);

Giả sử rằng P_0 thực thi wait(S) và sau đó P_1 thực thi wait(Q). Khi P_0 thực thi wait(Q), nó phải chờ cho đến khi P_1 thực thi signal(Q). Tương tự, khi P_1 thực thi wait(S), nó phải chờ cho tới khi P_0 thực thi signal(S). Vì các thao tác signal này không thể được thực thi nên P_0 và P_1 bị khoá chết.

Chúng ta nói rằng một tập hợp các quá trình trong trạng thái khoá chết khi mọi quá trình trong tập hợp đang chờ một sự kiện được gây ra chỉ bởi một quá trình khác trong tập hợp. Những sự kiện mà chúng ta quan tâm chủ yếu ở đây là việc chiếm tài nguyên và giải phóng tài nguyên. Tuy nhiên, các loại sự kiện khác cũng có thể dẫn đến việc khoá chết. Chúng ta sẽ xem trong chương VI. Trong chương đó, chúng ta sẽ mô tả các cơ chế khác nhau để giải quyết vấn đề khoá chết.

Một vấn đề khoá chết liên quan tới khoá chết là **nghẽn hay đói tài nguyên không hạn định** (indefinite blocking or starvation), ở đó các quá trình chờ đợi không hạn định trong semaphore. Nghẽn không hạn định có thể xảy ra nếu chúng ta thêm vào và lấy ra các quá trình từ danh sách được nối kết với một semaphore trong thứ tự vào sau ra trước (LIFO).

Thuviendientu.org

.V.2.1.4 Semaphore nhị phân

Xây dựng semaphore được mô tả trong phần trước được gọi là **semaphore đếm** (counting semaphore) vì giá trị nguyên có thể trãi dài một phạm vi không giới hạn. Một **semaphore nhị phân** (binary semaphore) là một semaphore với một giá trị nguyên mà trải dài từ 0 và 1. Semaphore nhị phân có thể đơn giản hơn trong cài đặt so với semaphore đếm và phụ thuộc vào kiến trúc phần cứng nằm bên dưới. Chúng sẽ hiển thị cách một semaphore đếm có thể được cài đặt sử dụng semaphore nhị phân dưới đây:

Giả sử S là một semaphore đếm. Để cài đặt nó trong dạng semaphore nhị phân chúng ta cần các cấu trúc dữ liệu như sau:

```
Binary-semaphore S1, S2;
int C;
```

Khởi tạo S1 = 1, S2 = 0 và giá trị nguyên C được đặt tới giá trị khởi tạo của semaphore đếm S.

Thao tác wait trên semaphore đếm S có thể được cài đặt như sau:

```
wait(S);
C--;
If (C<0) {
        signal(S1);
        wait(S2);
}
signal(S1);
```

Thao tác signal trên semaphore đếm S có thể được cài đặt như sau:

```
wait(S1);
C++;
if (C<=0)
        signal(S2);
else
        signal(S1);
```

V.2.2 Monitors

Để có thể dễ viết đúng các chương trình đồng bộ hoá hơn, Hoare (1974) và Brinch & Hansen (1975) đề nghị một cơ chế đồng bộ hoá cấp cao hơn được cung cấp bởi ngôn ngữ lập trình là **monitor**. Một monitor được mô tả bởi một tập hợp của các toán tử được định nghĩa bởi người lập trình. Biểu diễn kiểu của một monitor bao gồm việc khai báo các biến mà giá trị của nó xác định trạng thái của một thể hiện kiểu, cũng như thân của thủ tục hay hàm mà cài đặt các thao tác trên kiểu đó. Cú pháp của monitor được hiển thị trong hình dưới đây:

```
Monitor <tên monitor>
        {
                khai báo các biến được chia sẻ
                procedure P1 (…){
        …
        }

        procedure P2 (…){
        …
```

Thuviendientu.org

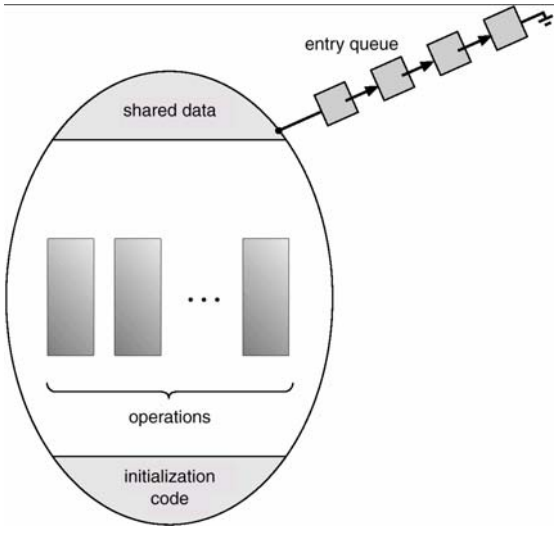

```
}
.
.
.
procedure Pn (…){
…
}

{
mã khởi tạo
}
}
```

Hình 0-14 Cú pháp của monitor

Biểu diễn kiểu monitor không thể được dùng trực tiếp bởi các quá trình khác nhau. Do đó, một thủ tục được định nghĩa bên trong một monitor chỉ có thể truy xuất những biến được khai báo cục bộ bên trong monitor đó và các tham số chính thức của nó. Tương tự, những biến cục bộ của monitor có thể được truy xuất chỉ bởi những thủ tục cục bộ.

Xây dựng monitor đảm bảo rằng chỉ một quá trình tại một thời điểm có thể được kích hoạt trong monitor. Do đó, người lập trình không cần viết mã ràng buộc đồng bộ hoá như hình V-15 dưới đây:



Hình 0-15 Hình ảnh dưới dạng biểu đồ của monitor

Tuy nhiên, xây dựng monitor như được định nghĩa là không đủ mạnh để mô hình hoá các cơ chế đồng bộ. Cho mục đích này, chúng ta cần định nghĩa các cơ chế đồng bộ hoá bổ sung. Những cơ chế này được cung cấp bởi construct ***condition***. Người lập trình có thể định nghĩa một hay nhiều biến của kiểu ***condition***:

```
condition x, y;
```

Chỉ những thao tác có thể gọi lên trên các biến điều kiện là wait và signal. Thao tác

Thuviendientu.org

x.wait();

có nghĩa là quá trình gọi trên thao tác này được tạm dừng cho đến khi quá trình khác gọi

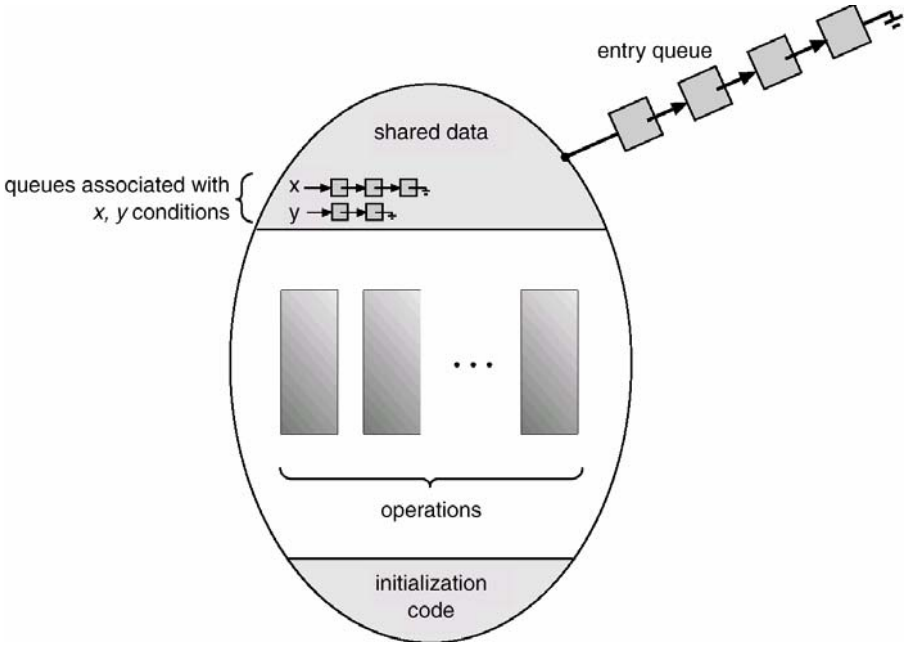
x.signal();

thao tác x.signal() thực thi tiếp một cách chính xác một quá trình tạm dừng. Nếu không có quá trình tạm dừng thì thao tác signal không bị ảnh hưởng gì cả; nghĩa là trạng thái x như thể thao tác chưa bao giờ được thực thi (như hình V.-16). Ngược lại, với thao tác signal được gán cùng với semaphores luôn ảnh hưởng tới trạng thái của semaphore.

Bây giờ giả sử rằng, khi thao tác x.signal() được gọi bởi một quá trình P thì có một quá trình Q gán với biến điều kiện x bị tạm dừng. Rõ ràng, nếu quá trình Q được phép thực thi tiếp thì quá trình P phải dừng. Nếu không thì cả hai quá trình P và Q hoạt động cùng một lúc trong monitor. Tuy nhiên, về khái niệm hai quá trình có thể tiếp tục việc thực thi của chúng. Hai khả năng có thể xảy ra:

P chờ cho đến khi Q rời khỏi monitor hoặc chờ điều kiện khác.

Q chờ cho đến khi P rời monitor hoặc chờ điều kiện khác.



Hình 0-16 Monitor với các biến điều kiện

Có các luận cứ hợp lý trong việc chấp nhận khả năng 1 hay 2. Vì P đã thực thi trong monitor rồi, nên chọn khả năng 2 có vẻ hợp lý hơn. Tuy nhiên, nếu chúng ta cho phép quá trình P tiếp tục, biến điều kiện "***luận lý***" mà Q đang chờ có thể không còn quản lý thời gian Q được tiếp tục. Chọn khả năng 1 được tán thành bởi Hoare vì tham số đầu tiên của nó chuyển trực tiếp tới các qui tắc chứng minh đơn giản hơn. Thoả hiệp giữa hai khả năng này được chấp nhận trong ngôn ngữ đồng hành C. Khi quá trình P thực thi thao tác signal thì quá trình Q lập tức được tiếp tục. Mô hình này không mạnh hơn mô hình của Hoare vì một quá trình không thể báo hiệu nhiều lần trong một lời gọi thủ tục đơn.

Bây giờ chúng ta xem xét cài đặt cơ chế monitor dùng semaphores. Đối với mỗi monitor, một biến semaphore ***mutex*** (được khởi tạo 1) được cung cấp. Một quá

Thuviendientu.org

trình phải thực thi ***wait(mutex)*** trước khi đi vào monitor và phải thực thi ***signal(mutex)*** sau khi rời monitor.

Vì quá trình đang báo hiệu phải chờ cho đến khi quá trình được bắt đầu lại rời hay chờ, một biến semaphore bổ sung ***next*** được giới thiệu, được khởi tạo 0 trên quá trình báo hiệu có thể tự tạm dừng. Một biến số nguyên ***next_count*** cũng sẽ được cung cấp để đếm số lượng quá trình bị tạm dừng trên ***next***. Do đó, mỗi thủ tục bên ngoài F sẽ được thay thế bởi

```
wait(mutex);
        . . .
thân của F
if (next_count > 0)
        signal(next);
else
        signal(mutex);
```

Loại trừ hỗ tương trong monitor được đảm bảo.

Bây giờ chúng ta mô tả các biến điều kiện được cài đặt như thế nào. Đối với mỗi biến điều kiện ***x***, chúng ta giới thiệu một biến semaphore ***x_sem*** và biến số nguyên ***x_count***, cả hai được khởi tạo tới 0. Thao tác ***x.wait*** có thể được cài đặt như sau:

```
x_count++;
if ( next_count > 0)
        signal(next);
else
        signal(mutex);
wait(x_sem);
x_count--;
```

Thao tác ***x.signal()*** có thể được cài đặt như sau:

```
if ( x_count > 0){
        next_count++;
        signal(x_sem);
        wait(next);
        next_count--;
}
```

Cài đặt này có thể áp dụng để định nghĩa của monitor được cho bởi cả hai Hoare và Brinch Hansen. Tuy nhiên, trong một số trường hợp tính tổng quát của việc cài đặt là không cần thiết và yêu cầu có một cải tiến hiệu quả hơn.

Bây giờ chúng ta sẽ trở lại chủ đề thứ tự bắt đầu lại của quá trình trong monitor. Nếu nhiều quá trình bị trì hoãn trên biến điều kiện ***x*** và thao tác ***x.signal*** được thực thi bởi một vài quá trình thì thứ tự các quá trình bị trì hoãn được thực thi trở lại như thế nào? Một giải pháp đơn giản là dùng thứ tự FCFS vì thế quá trình chờ lâu nhất sẽ được thực thi tiếp trước. Tuy nhiên, trong nhiều trường hợp, cơ chế định thời biểu như thế là không đủ. Cho mục đích này cấu trúc ***conditional-wait*** có thể được dùng; nó có dạng

```
        x.wait(c);
```

ở đây ***c*** là một biểu thức số nguyên được định giá khi thao tác ***wait*** được thực thi. Giá trị ***c***, được gọi là ***số ưu tiên***, được lưu với tên quá trình được tạm dừng. Khi ***x.signal*** được thực thi, quá trình với số ưu tiên nhỏ nhất được thực thi tiếp.

Thuviendientu.org

Để hiển thị cơ chế mới này, chúng ta xem xét monitor được hiển thị như hình dưới đây, điều khiển việc cấp phát của một tài nguyên đơn giữa các quá trình cạnh tranh. Mỗi quá trình khi yêu cầu cấp phát tài nguyên của nó, xác định thời gian tối đa nó hoạch định để sử dụng tài nguyên. Monitor cấp phát tài nguyên tới quá trình có yêu cầu thời gian cấp phát ngắn nhất.

```
Monitor ResourceAllocation
{
        boolean     busy;
        condition   x;

        void acquire(int time){
                if (busy)       x.wait(time);
                busy = true;
        }

        void release(){
                busy = false;
                x.signal();
        }

        void init(){
                busy = false;
        }
}
```

Hình 0-17 Một monitor cấp phát tới một tài nguyên

Một quá trình cần truy xuất tài nguyên phải chú ý thứ tự sau:

```
R.acquire(t);
…
        truy xuất tài nguyên
...
R.release();
```

ở đây ***R*** là thể hiện của kiểu ***ResourceAllocation***.

Tuy nhiên, khái niệm monitor không đảm bảo rằng các thứ tự truy xuất trước sẽ được chú ý. Đặc biệt,

- Một quá trình có thể truy xuất tài nguyên mà không đạt được quyền truy xuất trước đó.
- Một quá trình sẽ không bao giờ giải phóng tài nguyên một khi nó được gán truy xuất tới tài nguyên đó.
- Một quá trình có thể cố gắng giải phóng tài nguyên mà nó không bao giờ yêu cầu.
- Một quá trình có thể yêu cầu cùng tài nguyên hai lần (không giải phóng tài nguyên đó trong lần đầu)

Việc sử dụng monitor cũng gặp cùng những khó khăn như xây dựng miền tương trục. Trong phần trước, chúng ta lo lắng về việc sử dụng đúng semaphore. Bây giờ, chúng ta lo lắng về việc sử dụng đúng các thao tác được định nghĩa của người lập trình cấp cao mà các trình biên dịch không còn hỗ trợ chúng ta.

Thuviendientu.org

Một giải pháp có thể đối với vấn đề trên là chứa các thao tác truy xuất tài nguyên trong monitor ***ResourceAllocation***. Tuy nhiên, giải pháp này sẽ dẫn đến việc định thời được thực hiện dựa theo giải thuật định thời monitor được xây dựng sẵn hơn là được viết bởi người lập trình.

Để đảm bảo rằng các quá trình chú ý đến thứ tự hợp lý, chúng ta phải xem xét kỹ tất cả chương trình thực hiện việc dùng monitor ResourceAllocation và những tài nguyên được quản lý của chúng. Có hai điều kiện mà chúng ta phải kiểm tra để thiết lập tính đúng đắn của hệ thống. Đầu tiên, các quá trình người dùng phải luôn luôn thực hiện các lời gọi của chúng trên monitor trong thứ tự đúng. Thứ hai, chúng ta phải đảm bảo rằng một quá trình không hợp tác không đơn giản bỏ qua cổng (gateway) loại trừ hỗ tương được cung cấp bởi monitor và cố gắng truy xuất trực tiếp tài nguyên được chia sẻ mà không sử dụng giao thức truy xuất. Chỉ nếu hai điều kiện này có thể được đảm bảo có thể chúng ta đảm bảo rằng không có lỗi ràng buộc thời gian nào xảy ra và giải thuật định thời sẽ không bị thất bại.

Mặc dù việc xem xét này có thể cho hệ thống nhỏ, tĩnh nhưng nó không phù hợp cho một hệ thống lớn hay động. Vấn đề kiểm soát truy xuất có thể được giải quyết chỉ bởi một cơ chế bổ sung khác.

VI Các bài toán đồng bộ hoá nguyên thuỷ

Trong phần này, chúng ta trình bày một số bài toán đồng bộ hoá như những thí dụ về sự phân cấp lớn các vấn đề điều khiển đồng hành. Các vấn đề này được dùng cho việc kiểm tra mọi cơ chế đồng bộ hoá được đề nghị gần đây. Semaphore được dùng cho việc đồng bộ hoá trong các giải pháp dưới đây.

VI.1 Bài toán người sản xuất-người tiêu thụ

Bài toán người sản xuất-người tiêu thụ (Producer-Consumer) thường được dùng để hiển thị sức mạnh của các hàm cơ sở đồng bộ hoá. Hai quá trình cùng chia sẻ một vùng đệm có kích thước giới hạn n. Biến semaphore ***mutex*** cung cấp sự loại trừ hỗ tương để truy xuất vùng đệm và được khởi tạo với giá trị ***1***. Các biến semaphore ***empty*** và ***full*** đếm số khe trống và đầy tương ứng. Biến semaphore ***empty*** được khởi tạo tới giá trị ***n***; biến semaphore ***full*** được khởi tạo tới giá trị ***0***.

Mã cho người quá trình sản xuất được hiển thị trong hình V.-18:

```
do{
        …
        sản xuất sản phẩm trong nextp
        …
        wait(empty);
        wait(mutex);
        …
        thêm nextp tới vùng đệm
        …
        signal(mutex);
        signal(full);
} while (1);
```

Hình 0-18 Cấu trúc của quá trình người sản xuất

Mã cho quá trình người tiêu thụ được hiển thị trong hình dưới đây:

Thuviendientu.org


```
do{
        wait(full);
        wait(mutex);
        …
        lấy một sản phẩm từ vùng đệm tới nextc
        …
        signal(mutex);
        signal(empty);
} while (1);
```

Hình 0-19 Cấu trúc của quá trình người tiêu thụ

VI.2 Bài toán bộ đọc-bộ ghi

Bộ đọc-bộ ghi (Readers-Writers) là một đối tượng dữ liệu (như một tập tin hay mẫu tin) được chia sẻ giữa nhiều quá trình đồng hành. Một số trong các quá trình có thể chỉ cần đọc nội dung của đối tượng được chia sẻ, ngược lại một vài quá trình khác cần cập nhật (nghĩa là đọc và ghi) trên đối tượng được chia sẻ. Chúng ta phân biệt sự khác nhau giữa hai loại quá trình này bằng cách gọi các quá trình chỉ đọc là bộ đọc và các quá trình cần cập nhật là bộ ghi. Chú ý, nếu hai bộ đọc truy xuất đối tượng được chia sẻ cùng một lúc sẽ không có ảnh hưởng gì. Tuy nhiên, nếu một bộ ghi và vài quá trình khác (có thể là bộ đọc hay bộ ghi) truy xuất cùng một lúc có thể dẫn đến sự hỗn độn.

Để đảm bảo những khó khăn này không phát sinh, chúng ta yêu cầu các bộ ghi có truy xuất loại trừ lẫn nhau tới đối tượng chia sẻ. Việc đồng bộ hoá này được gọi là bài toán bộ đọc-bộ ghi. Bài toán bộ đọc-bộ ghi có một số biến dạng liên quan đến độ ưu tiên. Dạng đơn giản nhất là bài toán ***bộ đọc trước-bộ ghi*** (first reader-writer). Trong dạng này yêu cầu không có bộ đọc nào phải chờ ngoại trừ có một bộ ghi đã được cấp quyền sử dụng đối tượng chia sẻ. Nói cách khác, không có bộ đọc nào phải chờ các bộ đọc khác để hoàn thành đơn giản vì một bộ ghi đang chờ. Bài toán ***bộ đọc sau-bộ ghi*** (second readers-writers) yêu cầu một khi bộ ghi đang sẵn sàng, bộ ghi đó thực hiện việc ghi của nó sớm nhất có thể. Nói một cách khác, nếu bộ ghi đang chờ truy xuất đối tượng, không có bộ đọc nào có thể bắt đầu việc đọc.

Giải pháp cho bài toán này có thể dẫn đến việc đói tài nguyên. Trong trường hợp đầu, các bộ ghi có thể bị đói; trong trường hợp thứ hai các bộ đọc có thể bị đói. Trong giải pháp cho bài toán bộ đọc trước-bộ ghi, các quá trình bộ đọc chia sẻ các cấu trúc dữ liệu sau:

```
semaphore  mutex, wrt;
int readcount;
```

Biến semaphore ***mutex*** và ***wrt*** được khởi tạo **1**; biến ***readcount*** được khởi tạo **0**. Biến semaphore ***wrt*** dùng chung cho cả hai quá trình bộ đọc và bộ ghi. Biến semaphore ***mutex*** được dùng để đảm bảo loại trừ hỗ tương khi biến ***readcount*** được cập nhật. Biến ***readcount*** ghi vết có bao nhiêu quá trình hiện hành đang đọc đối tượng. Biến semaphore ***wrt*** thực hiện chức năng như một biến semaphore loại trừ hỗ tương cho các bộ đọc. Nó cũng được dùng bởi bộ đọc đầu tiên hay bộ đọc cuối cùng mà nó đi vào hay thoát khỏi miền tương trục. Nó cũng không được dùng bởi các bộ đọc mà nó đi vào hay thoát trong khi các bộ đọc khác đang ở trong miền tương trục. Mã cho quá trình bộ viết được hiển thị như hình V.-20:

```
wait(wrt);
```

Thuviendientu.org


```
…
Thao tác viết được thực hiện
signal(wrt);
```

Hình 0-20 Cấu trúc của quá trình viết

Mã của quá trình đọc được hiển thị như hình V.-21:

```
wait(mutex);
readcount++;
if (readcount == 1)
        wait(wrt);
signal(mutex);
…
Thao tác đọc được thực hiện
wait(mutex);
readcount--;
if (readcount == 0)
        signal(wrt);
signal(mutex);
```

Hình 0-21 Cấu trúc của bộ đọc

Chú ý rằng, nếu bộ viết đang ở trong miền tương trục và n bộ đọc đang chờ thì một bộ đọc được xếp hàng trên wrt, và n-1 được xếp hàng trên mutex. Cũng cần chú ý thêm, khi một bộ viết thực thi signal(wrt) thì chúng ta có thể thực thi tiếp việc thực thi của các quá trình đọc đang chờ hay một quá trình viết đang chờ. Việc chọn lựa này có thể được thực hiện bởi bộ định thời biểu.

VI.3 Bài toán các triết gia ăn tối

Có năm nhà triết gia, vừa suy nghĩ vừa ăn tối. Các triết gia ngồi trên cùng một bàn tròn xung quanh có năm chiếc ghế, mỗi chiếc ghế được ngồi bởi một triết gia. Chính giữa bàn là một bát cơm và năm chiếc đũa được hiển thị như hình VI-22:

VII

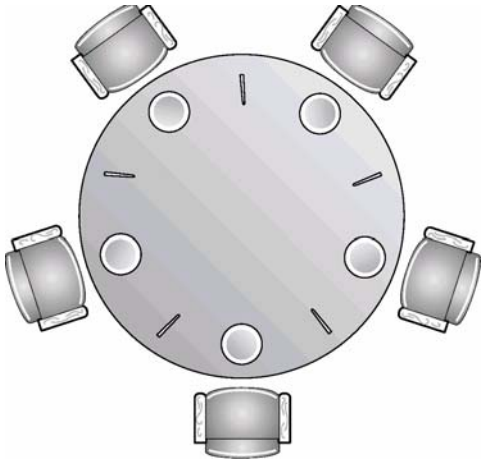
VIII

IX

X

XI

XII



Hình 0-22 Tình huống các triết gia ăn tối

Thuviendientu.org

Khi một triết gia suy nghĩ, ông ta không giao tiếp với các triết gia khác. Thỉnh thoảng, một triết gia cảm thấy đói và cố gắng chọn hai chiếc đũa gần nhất (hai chiếc đũa nằm giữa ông ta với hai láng giềng trái và phải). Một triết gia có thể lấy chỉ một chiếc đũa tại một thời điểm. Chú ý, ông ta không thể lấy chiếc đũa mà nó đang được dùng bởi người láng giềng. Khi một triết gia đói và có hai chiếc đũa cùng một lúc, ông ta ăn mà không đặt đũa xuống. Khi triết gia ăn xong, ông ta đặt đũa xuống và bắt đầu suy nghĩ tiếp.

Bài toán các triết gia ăn tối được xem như một bài toán đồng bộ hoá kinh điển. Nó trình bày yêu cầu cấp phát nhiều tài nguyên giữa các quá trình trong cách tránh việc khoá chết và đói tài nguyên.

Một giải pháp đơn giản là thể hiện mỗi chiếc đũa bởi một biến semaphore. Một triết gia cố gắng chiếm lấy một chiếc đũa bằng cách thực thi thao tác wait trên biến semaphore đó; triết gia đặt hai chiếc đũa xuống bằng cách thực thi thao tác signal trên các biến semaphore tương ứng. Do đó, dữ liệu được chia sẻ là:

```
semaphore chopstick[5];
```

ở đây tất cả các phần tử của chopstick được khởi tạo 1. Cấu trúc của philosopher i được hiển thị như hình dưới đây:

```
do{
        wait(chopstick[ i ]);
        wait(chopstick[  ( i + 1 ) % 5 ]);
        …
        ăn
        …
        signal(chopstick[ i ]);
        signal(chopstick[  ( i + 1 ) % 5 ]);
        …
        suy nghĩ
        …
} while (1);
```

Hình 0-23 Cấu trúc của triết gia thứ i

Mặc dù giải pháp này đảm bảo rằng không có hai láng giềng nào đang ăn cùng một lúc nhưng nó có khả năng gây ra khoá chết. Giả sử rằng năm triết gia bị đói cùng một lúc và mỗi triết gia chiếm lấy chiếc đũa bên trái của ông ta. Bây giờ tất cả các phần tử chopstick sẽ là 0. Khi mỗi triết gia cố gắng dành lấy chiếc đũa bên phải, triết gia sẽ bị chờ mãi mãi.

Nhiều giải pháp khả thi đối với vấn đề khoá chết được liệt kê tiếp theo. Giải pháp cho vấn đề các triết gia ăn tối mà nó đảm bảo không bị khoá chết.

- Cho phép nhiều nhất bốn triết gia đang ngồi cùng một lúc trên bàn
- Cho phép một triết gia lấy chiếc đũa của ông ta chỉ nếu cả hai chiếc đũa là sẵn dùng (để làm điều này ông ta phải lấy chúng trong miền tương trục).
- Dùng một giải pháp bất đối xứng; nghĩa là một triết gia lẽ chọn đũa bên trái đầu tiên của ông ta và sau đó đũa bên phải, trái lại một triết gia chẵn chọn chiếc đũa bên phải và sau đó chiếc đũa bên phải của ông ta.

Tóm lại, bất cứ một giải pháp nào thoả mãn đối với bài toán các triết gia ăn tối phải đảm bảo dựa trên khả năng một trong những triết gia sẽ đói chết. Giải pháp giải quyết việc khoá chết không cần thiết xoá đi khả năng đói tài nguyên.

Thuviendientu.org

XIII Tóm tắt

Một tập hợp các quá trình tuần tự cộng tác chia sẻ dữ liệu, loại trừ hỗ tương phải được cung cấp. Một giải pháp đảm bảo rằng vùng tương trục của mã đang sử dụng chỉ bởi một quá trình hay một luồng tại một thời điểm. Các giải thuật khác tồn tại để giải quyết vấn đề miền tương trục, với giả thuyết rằng chỉ khoá bên trong việc lưu trữ là sẵn dùng.

Sự bất lợi chủ yếu của các giải pháp được mã hoá bởi người dùng là tất cả chúng đều yêu cầu sự chờ đợi bận. Semaphore khắc phục sự bất lợi này. Semaphores có thể được dùng để giải quyết các vấn đề đồng bộ khác nhau và có thể được cài đặt hiệu quả, đặc biệt nếu phần cứng hỗ trợ các thao tác nguyên tử.

Các bài toán đồng bộ khác (chẳng hạn như bài toán người sản xuất-người tiêu dùng, bài toán bộ đọc, bộ ghi và bài toán các triết gia ăn tối) là cực kỳ quan trọng vì chúng là thí dụ của phân lớp lớn các vấn đề điều khiển đồng hành. Vấn đề này được dùng để kiểm tra gần như mọi cơ chế đồng bộ được đề nghị gần đây.

Hệ điều hành phải cung cấp phương tiện để đảm bảo chống lại lỗi thời gian. Nhiều cấu trúc dữ liệu được đề nghị để giải quyết các vấn đề này. Các vùng tương trục có thể được dùng để cài đặt loại trừ hỗ tương và các vấn đề đồng bộ an toàn và hiệu quả. Monitors cung cấp cơ chế đồng bộ cho việc chia sẻ các loại dữ liệu trừu tượng. Một biến điều kiện cung cấp một phương thức cho một thủ tục monitor khoá việc thực thi của nó cho đến khi nó được báo hiệu tiếp tục.

Thuviendientu.org

DEADLOCK

I Mục đích

Sau khi học xong chương này, người học nắm được những kiến thức sau:

- Hiểu mô hình hệ thống về deadlock
- Hiểu các đặc điểm của deadlock
- Hiểu các phương pháp quản lý deadlock
- Hiểu cách ngăn chặn deadlock
- Hiểu cách tránh deadlock
- Hiểu cách phát hiện deadlock
- Hiểu cách phục hồi từ deadlock

II Giới thiệu

Trong môi trường đa chương, nhiều quá trình có thể cạnh tranh một số giới hạn tài nguyên. Một quá trình yêu cầu tài nguyên, nếu tài nguyên không sẵn dùng tại thời điểm đó, quá trình đi vào trạng thái chờ. Quá trình chờ có thể không bao giờ chuyển trạng thái trở lại vì tài nguyên chúng yêu cầu bị giữ bởi những quá trình đang chờ khác. Trường hợp này được gọi là deadlock (khoá chết).

Trong chương này chúng ta sẽ mô tả các phương pháp mà hệ điều hành có thể dùng để ngăn chặn hay giải quyết deadlock. Hầu hết các hệ điều hành không cung cấp phương tiện ngăn chặn deadlock nhưng những đặc điểm này sẽ được thêm vào sau đó. Vấn đề deadlock chỉ có thể trở thành vấn đề phổ biến, xu hướng hiện hành gồm số lượng lớn quá trình, chương trình đa luồng, nhiều tài nguyên trong hệ thống và đặc biệt các tập tin có đời sống dài và những máy phục vụ cơ sở dữ liệu hơn là các hệ thống bó.

III Mô hình hệ thống

Một hệ thống chứa số tài nguyên hữu hạn được phân bổ giữa nhiều quá trình cạnh tranh. Các tài nguyên này được phân chia thành nhiều loại, mỗi loại chứa một số thể hiện xác định. Không gian bộ nhớ, các chu kỳ CPU và các thiết bị nhập/xuất (như máy in, đĩa từ) là những thí dụ về loại tài nguyên. Nếu hệ thống có hai CPUs, thì loại tài nguyên CPU có hai thể hiện. Tương tự, loại tài nguyên máy in có thể có năm thể hiện.

Nếu một quá trình yêu cầu một thể hiện của loại tài nguyên thì việc cấp phát bất cứ thể hiện nào của loại tài nguyên này sẽ thoả mãn yêu cầu. Nếu nó không có thì các thể hiện là không xác định và các lớp loại tài nguyên sẽ không được định nghĩa hợp lý. Thí dụ, một hệ thống có thể có hai máy in. Hai loại máy in này có thể được định nghĩa trong cùng lớp loại tài nguyên nếu không có quá trình nào quan tâm máy nào in ra dữ liệu. Tuy nhiên, nếu một máy in ở tầng 9 và máy in khác ở tầng trệt thì người dùng ở tầng 9 không thể xem hai máy in là tương tự nhau và lớp tài nguyên riêng rẻ cần được định nghĩa cho mỗi máy in.

Một quá trình phải yêu cầu một tài nguyên trước khi sử dụng nó, và phải giải phóng sau khi sử dụng nó. Một quá trình có thể yêu cầu nhiều tài nguyên như nó được

Thuviendientu.org

yêu cầu để thực hiện tác vụ được gán của nó. Chú ý, số tài nguyên được yêu cầu không vượt quá số lượng tổng cộng tài nguyên sẵn có trong hệ thống. Nói cách khác, một quá trình không thể yêu cầu ba máy in nếu hệ thống chỉ có hai.

Dưới chế độ điều hành thông thường, một quá trình có thể sử dụng một tài nguyên chỉ trong thứ tự sau:

1) **Yêu cầu**: nếu yêu cầu không thể được gán tức thì (thí dụ, tài nguyên đang được dùng bởi quá trình khác) thì quá trình đang yêu cầu phải chờ cho tới khi nó có thể nhận được tài nguyên.
2) **Sử dụng**: quá trình có thể điều hành tài nguyên (thí dụ, nếu tài nguyên là máy in, quá trình có thể in máy in)
3) **Giải phóng**: quá trình giải phóng tài nguyên.

Yêu cầu và giải phóng tài nguyên là các lời gọi hệ thống. Thí dụ như yêu cầu và giải phóng thiết bị, mở và đóng tập tin, cấp phát và giải phóng bộ nhớ. Yêu cầu và giải phóng các tài nguyên khác có thể đạt được thông qua thao tác chờ **wait** và báo hiệu **signal**. Do đó, cho mỗi trường hợp sử dụng, hệ điều hành kiểm tra để đảm bảo rằng quá trình sử dụng yêu cầu và được cấp phát tài nguyên. Một bảng hệ thống ghi nhận mỗi quá trình giải phóng hay được cấp phát tài nguyên. Nếu một quá trình yêu cầu tài nguyên mà tài nguyên đó hiện được cấp phát cho một quá trình khác, nó có thể được thêm vào hàng đợi để chờ tài nguyên này.

Một tập hợp quá trình trong trạng thái deadlock khi mỗi quá trình trong tập hợp này chờ sự kiện mà có thể được tạo ra chỉ bởi quá trình khác trong tập hợp. Những sự kiện mà chúng ta quan tâm chủ yếu ở đây là nhận và giải phóng tài nguyên. Các tài nguyên có thể là tài nguyên vật lý (thí dụ, máy in, đĩa từ, không gian bộ nhớ và chu kỳ CPU) hay tài nguyên luận lý (thí dụ, tập tin, semaphores, monitors). Tuy nhiên, các loại khác của sự kiện có thể dẫn đến deadlock.

Để minh hoạ trạng thái deadlock, chúng ta xét hệ thống với ba ổ đĩa từ. Giả sử mỗi quá trình giữ các một ổ đĩa từ này. Bây giờ, nếu mỗi quá trình yêu cầu một ổ đĩa từ khác thì ba quá trình sẽ ở trong trạng thái deadlock. Mỗi quá trình đang chờ một sự kiện "ổ đĩa từ được giải phóng" mà có thể được gây ra chỉ bởi một trong những quá trình đang chờ. Thí dụ này minh hoạ deadlock liên quan đến cùng loại tài nguyên.

Deadlock cũng liên quan nhiều loại tài nguyên khác nhau. Thí dụ, xét một hệ thống với một máy in và một ổ đĩa từ. Giả sử, quá trình P_i đang giữ ổ đĩa từ và quá trình P_j đang giữ máy in. Nếu P_i yêu cầu máy in và P_j yêu cầu ổ đĩa từ thì deadlock xảy ra.

Một người lập trình đang phát triển những ứng dụng đa luồng phải quan tâm đặc biệt tới vấn đề này: Các chương trình đa luồng là ứng cử viên cho vấn đề deadlock vì nhiều luồng có thể cạnh tranh trên tài nguyên được chia sẻ.

IV Đặc điểm deadlock

Trong một deadlock, các quá trình không bao giờ hoàn thành việc thực thi và các tài nguyên hệ thống bị buộc chặt, ngăn chặn các quá trình khác bắt đầu. Trước khi chúng ta thảo luận các phương pháp khác nhau giải quyết vấn đề deadlock, chúng ta sẽ mô tả các đặc điểm mà deadlock mô tả.

Thuviendientu.org

IV.1 Những điều kiện cần thiết gây ra deadlock

Trường hợp deadlock có thể phát sinh nếu bốn điều kiện sau xảy ra cùng một lúc trong hệ thống:

1) **Loại trừ hỗ tương**: ít nhất một tài nguyên phải được giữ trong chế độ không chia sẻ; nghĩa là, chỉ một quá trình tại cùng một thời điểm có thể sử dụng tài nguyên. Nếu một quá trình khác yêu cầu tài nguyên đó, quá trình yêu cầu phải tạm dừng cho đến khi tài nguyên được giải phóng.
2) **Giữ và chờ cấp thêm tài nguyên**: quá trình phải đang giữ ít nhất một tài nguyên và đang chờ để nhận tài nguyên thêm mà hiện đang được giữ bởi quá trình khác.
3) **Không đòi lại tài nguyên từ quá trình đang giữ chúng**: Các tài nguyên không thể bị đòi lại; nghĩa là, tài nguyên có thể được giải phóng chỉ tự ý bởi quá trình đang giữ nó, sau khi quá trình đó hoàn thành tác vụ.
4) **Tồn tại chu trình trong đồ thị cấp phát tài nguyên**: một tập hợp các quá trình $\{P_0, P_1,\ldots,P_n\}$ đang chờ mà trong đó P_0 đang chờ một tài nguyên được giữ bởi P_1, P_1 đang chờ tài nguyên đang giữ bởi $P_2,\ldots,P_{n-1}$ đang chờ tài nguyên đang được giữ bởi quá trình P_0.

Chúng ta nhấn mạnh rằng tất cả bốn điều kiện phải cùng phát sinh để deadlock xảy ra. Điều kiện chờ đợi ch trình đưa đến điều kiện giữ-và-chờ vì thế bốn điều kiện không hoàn toàn độc lập.

IV.2 Đồ thị cấp phát tài nguyên

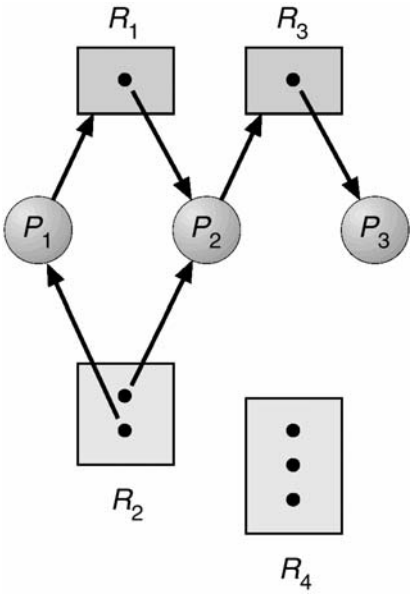
Deadlock có thể mô tả chính xác hơn bằng cách hiển thị đồ thị có hướng gọi là đồ thị cấp phát tài nguyên hệ thống. Đồ thị này chứa một tập các đỉnh V và tập hợp các cạnh E. Một tập các đỉnh V được chia làm hai loại nút $P = \{P_1, P_2,\ldots,P_n\}$ là tập hợp các quá trình hoạt động trong hệ thống, và $R = \{R_1, R_2, ..., R_m\}$ là tập hợp chứa tất cả các loại tài nguyên trong hệ thống.

Một cạnh có hướng từ quá trình P_i tới loại tài nguyên R_j được ký hiệu $P_i \rightarrow R_j$; nó biểu thị rằng quá trình P_i đã yêu cầu loại tài nguyên R_j và hiện đang chờ loại tài nguyên đó. Một cạnh có hướng từ loại tài nguyên R_j tới quá trình P_i được hiển thị bởi $R_j \rightarrow P_i$; nó hiển thị rằng thể hiện của loại tài nguyên R_j đã được cấp phát tới quá trình P_i. Một cạnh có hướng $P_i \rightarrow R_j$ được gọi là cạnh yêu cầu; một cạnh có hướng $R_j \rightarrow P_i$ được gọi là cạnh gán.

Bằng hình tượng, chúng ta hiển thị mỗi quá trình P_i là một hình tròn, và mỗi loại tài nguyên R_j là hình chữ nhật. Vì loại tài nguyên R_j có thể có nhiều hơn một thể hiện, chúng ta hiển thị mỗi thể hiện là một chấm nằm trong hình vuông. Chú ý rằng một cạnh yêu cầu trỏ tới chỉ một hình vuông R_j, trái lại một cạnh gán cũng phải gán tới một trong các dấu chấm trong hình vuông.

Khi quá trình P_i yêu cầu một thể hiện của loại tài nguyên R_j, một cạnh yêu cầu được chèn vào đồ thị cấp phát tài nguyên. Khi yêu cầu này có thể được đáp ứng, cạnh yêu cầu lập tức được truyền tới cạnh gán. Khi quá trình không còn cần truy xuất tới tài nguyên, nó giải phóng tài nguyên, và khi đó dẫn đến cạnh gán bị xoá.
Đồ thị cấp phát tài nguyên được hiển thị trong hình VI-1 dưới đây mô tả trường hợp sau:

Thuviendientu.org



Hình 0-1 Đồ thị cấp phát tài nguyên

- Các tập P, R, và E:
  - $P = \{P_1, P_2, P_3\}$
  - $R = \{R_1, R_2, R_3, R_4\}$
  - $E = \{P_1 \rightarrow R_1, P_2 \rightarrow R_3, R_1 \rightarrow P_2, R_2 \rightarrow P_2, R_3 \rightarrow P_3\}$
- Các thể hiện tài nguyên
  - Một thể hiện của tài nguyên loại R_1
  - Hai thể hiện của tài nguyên loại R_2
  - Một thể hiện của tài nguyên loại R_3
  - Một thể hiện của tài nguyên loại R_4
- Trạng thái quá trình
  - Quá trình P_1 đang giữ một thể hiện của loại tài nguyên R_2 và đang chờ một thể hiện của loại tài nguyên R_1.
  - Quá trình P_2 đang giữ một thể hiện của loại tài nguyên R_1 và R_2 và đang chờ một thể hiện của loại tài nguyên R_3.
  - Quá trình P_3 đang giữ một thể hiện của R_3

Đồ thị cấp phát tài nguyên hiển thị rằng, nếu đồ thị không chứa chu trình, thì không có quá trình nào trong hệ thống bị deadlock. Nếu đồ thị có chứa chu trình, thì deadlock có thể tồn tại.

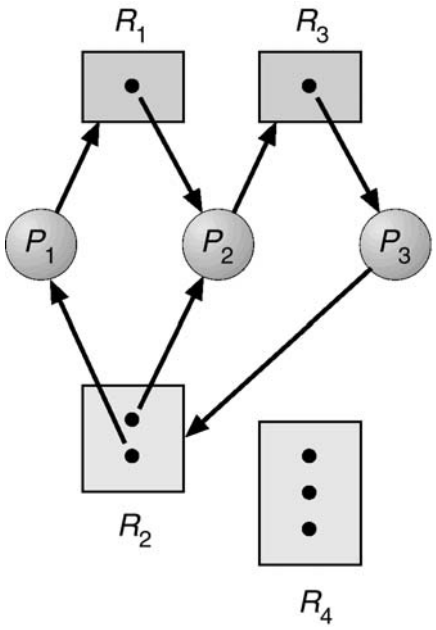
Nếu mỗi loại tài nguyên có chính xác một thể hiện, thì một chu trình ngụ ý rằng một deadlock xảy ra. Nếu một chu trình bao gồm chỉ một tập hợp các loại tài nguyên, mỗi loại tài nguyên chỉ có một thể hiện thì deadlock xảy ra. Mỗi quá trình chứa trong chu trình bị deadlock. Trong trường hợp này, một chu trình trong đồ thị là điều kiện cần và đủ để tồn tại deadlock.

Nếu mỗi loại tài nguyên có nhiều thể hiện thì chu trình không ngụ ý deadlock xảy. Trong trường hợp này, một chu trình trong đồ thị là điều kiện cần nhưng chưa đủ để tồn tại deadlock.

Để hiển thị khái niệm này, chúng ta xem lại đồ thị ở hình VII-1 ở trên. Giả sử quá trình P_3 yêu cầu một thể hiện của loại tài nguyên R_2. Vì không có thể hiện tài nguyên hiện có, một cạnh yêu cầu $P_3 \rightarrow R_2$ được thêm vào đồ thị (hình VI-2). Tại thời điểm này, hai chu trình nhỏ tồn tại trong hệ thống:

Thuviendientu.org

$P_1 \rightarrow R_1 \rightarrow P_2 \rightarrow R_3 \rightarrow P_3 \rightarrow R_2 \rightarrow P_1$
$P_2 \rightarrow R_3 \rightarrow P_3 \rightarrow R_2 \rightarrow P_2$

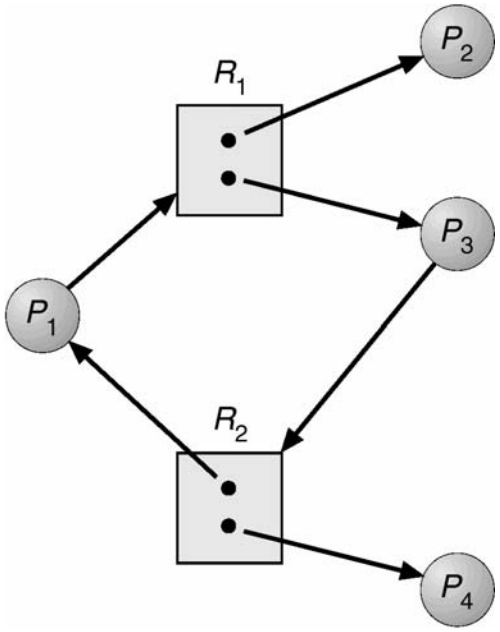


Hình 0-2 Đồ thị cấp phát tài nguyên với deadlock

Quá trình P_1, P_2, và P_3 bị deadlock. Quá trình P_3 đang chờ tài nguyên R_3, hiện được giữ bởi quá trình P_2. Hay nói cách khác, quá trình P_3 đang chờ quá trình P_1 hay P_2 giải phóng tài nguyên R_2. Ngoài ra, quá trình P_1 đang chờ quá trình P_2 giải phóng tài nguyên R_1.

Bây giờ xem xét đồ thị cấp phát tài nguyên trong hình VI-3 dưới đây. Trong thí dụ này, chúng ta cũng có một chu kỳ

$P_1 \rightarrow R_1 \rightarrow P_3 \rightarrow R_2 \rightarrow P_1$



Hình 0-3 Đồ thị cấp phát tài nguyên có chu trình nhưng không bị deadlock

Thuviendientu.org

Tuy nhiên, không có deadlock. Chú ý rằng quá trình P_4 có thể giải phóng thể hiện của loại tài nguyên R_2. Tài nguyên đó có thể được cấp phát tới P_3 sau đó, chu trình sẽ không còn.

Tóm lại, nếu đồ thị cấp phát tài nguyên không có chu trình thì hệ thống không có trạng thái deadlock. Ngoài ra, nếu có chu trình thì có thể có hoặc không trạng thái deadlock. Nhận xét này là quan trọng khi chúng ta giải quyết vấn đề deadlock.

V Các phương pháp xử lý deadlock

Phần lớn, chúng ta có thể giải quyết vấn đề deadlock theo một trong ba cách:

- Chúng ta có thể sử dụng một giao thức để ngăn chặn hay tránh deadlocks, đảm bảo rằng hệ thống sẽ không bao giờ đi vào trạng thái deadlock
- Chúng ta có thể cho phép hệ thống đi vào trạng thái deadlock, phát hiện nó và phục hồi.
- Chúng ta có thể bỏ qua hoàn toàn vấn đề này và giả vờ deadlock không bao giờ xảy ra trong hệ thống. Giải pháp này được dùng trong nhiều hệ điều hành, kể cả UNIX.
- Chúng ta sẽ tìm hiểu vắn tắt mỗi phương pháp. Sau đó, chúng ta sẽ trình bày các giải thuật một cách chi tiết trong các phần sau đây.

Để đảm bảo deadlock không bao giờ xảy ra, hệ thống có thể dùng kế hoạch ngăn chặn hay tránh deadlock. Ngăn chặn deadlock là một tập hợp các phương pháp để đảm bảo rằng ít nhất một điều kiện cần (trong phần VI.4.1) không thể xảy ra. Các phương pháp này ngăn chặn deadlocks bằng cách ràng buộc yêu cầu về tài nguyên được thực hiện như thế nào. Chúng ta thảo luận phương pháp này trong phần sau.

Ngược lại, tránh deadlock yêu cầu hệ điều hành cung cấp những thông tin bổ sung tập trung vào loại tài nguyên nào một quá trình sẽ yêu cầu và sử dụng trong thời gian sống của nó. Với những kiến thức bổ sung này, chúng ta có thể quyết định đối với mỗi yêu cầu quá trình nên chờ hay không. Để quyết định yêu cầu hiện tại có thể được thoả mãn hay phải bị trì hoãn, hệ thống phải xem xét tài nguyên hiện có, tài nguyên hiện cấp phát cho mỗi quá trình, và các yêu cầu và giải phóng tương lai của mỗi quá trình.

Nếu một hệ thống không dùng giải thuật ngăn chặn hay tránh deadlock thì trường hợp deadlock có thể xảy ra. Trong môi trường này, hệ thống có thể cung cấp một giải thuật để xem xét trạng thái của hệ thống để xác định deadlock có xảy ra hay không và giải thuật phục hồi từ deadlock.

Nếu hệ thống không đảm bảo rằng deadlock sẽ không bao giờ xảy ra và cũng không cung cấp một cơ chế để phát hiện và phục hồi deadlock thì có thể dẫn đến trường hợp hệ thống ở trong trạng thái deadlock. Trong trường hợp này, deadlock không được phát hiện sẽ làm giảm năng lực hệ thống vì tài nguyên đang được giữ bởi những quá trình mà chúng không thể thực thi, đi vào trạng thái deadlock. Cuối cùng, hệ thống sẽ dừng các chức năng và cần được khởi động lại bằng thủ công.

Mặc dù phương pháp này dường như không là tiếp cận khả thi đối với vấn đề deadlock nhưng nó được dùng trong một số hệ điều hành. Trong nhiều hệ thống, deadlock xảy ra không thường xuyên; do đó phương pháp này là rẻ hơn chi phí cho phương pháp ngăn chặn deadlock, tránh deadlock, hay phát hiện và phục hồi deadlock mà chúng phải được sử dụng liên tục. Trong một số trường hợp, hệ thống ở trong trạng thái cô đặc nhưng không ở trạng thái deadlock. Như thí dụ, xem xét một quá trình thời thực chạy tại độ ưu tiên cao nhất (hay bất cứ quá trình đang chạy trên bộ

Thuviendientu.org

định thời biểu không trưng dụng) và không bao giờ trả về điều khiển đối với hệ điều hành. Do đó, hệ thống phải có phương pháp phục hồi bằng thủ công cho các điều kiện không deadlock và có thể đơn giản sử dụng các kỹ thuật đó cho việc phục hồi deadlock.

VI Ngăn chặn deadlock

Để deadlock xảy ra, một trong bốn điều kiện cần phải xảy ra. Bằng cách đảm bảo ít nhất một trong bốn điều kiện này không thể xảy ra, chúng ta có thể ngăn chặn việc xảy ra của deadlock. Chúng ta tìm hiểu tỷ mỹ tiếp cận này bằng cách xem xét mỗi điều kiện cần riêng rẻ nhau.

VI.1 Loại trừ hỗ tương

Điều kiện loại trừ hỗ tương phải giữ cho tài nguyên không chia sẻ. Thí dụ, một máy in không thể được chia sẻ cùng lúc bởi nhiều quá trình. Ngược lại, các tài nguyên có thể chia sẻ không đòi hỏi truy xuất loại trừ hỗ tương và do đó không thể liên quan đến deadlock. Những tập tin chỉ đọc là một thí dụ tốt cho tài nguyên có thể chia sẻ. Nếu nhiều quá trình cố gắng mở một tập tin chỉ đọc tại cùng một thời điểm thì chúng có thể được gán truy xuất cùng lúc tập tin. Một quá trình không bao giờ yêu cầu chờ tài nguyên có thể chia sẻ. Tuy nhiên, thường chúng ta không thể ngăn chặn deadlock bằng cách từ chối điều kiện loại trừ hỗ tương: một số tài nguyên về thực chất không thể chia sẻ.

VI.2 Giữ và chờ cấp thêm tài nguyên

Để đảm bảo điều kiện giữ-và-chờ cấp thêm tài nguyên không bao giờ xảy ra trong hệ thống, chúng ta phải đảm bảo rằng bất cứ khi nào một quá trình yêu cầu tài nguyên, nó không giữ bất cứ tài nguyên nào khác. Một giao thức có thể được dùng là đòi hỏi mỗi quá trình yêu cầu và được cấp phát tất cả tài nguyên trước khi nó bắt đầu thực thi. Chúng ta có thể cài đặt sự cung cấp này bằng cách yêu cầu các lời gọi hệ thống yêu cầu tài nguyên cho một quá trình trước tất cả các lời gọi hệ thống khác.

Một giao thức khác cho phép một quá trình yêu cầu tài nguyên chỉ khi quá trình này không có tài nguyên nào. Một quá trình có thể yêu cầu một số tài nguyên và dùng chúng. Tuy nhiên, trước khi nó có thể yêu cầu bất kỳ tài nguyên bổ sung nào, nó phải giải phóng tất cả tài nguyên mà nó hiện đang được cấp phát.

Để hiển thị sự khác nhau giữa hai giao thức, chúng ta xét một quá trình chép dữ liệu từ băng từ tới tập tin đĩa, sắp xếp tập tin đĩa và sau đó in kết quả ra máy in. Nếu tất cả tài nguyên phải được yêu cầu cùng một lúc thì khởi đầu quá trình phải yêu cầu băng từ, tập tin đĩa và máy in. Nó sẽ giữ máy in trong toàn thời gian thực thi của nó mặc dù nó cần máy in chỉ ở giai đoạn cuối.

Phương pháp thứ hai cho phép quá trình yêu cầu ban đầu chỉ băng từ và tập tin đĩa. Nó chép dữ liệu từ băng từ tới đĩa, rồi giải phóng cả hai băng từ và đĩa. Sau đó, quá trình phải yêu cầu lại tập tin đĩa và máy in. Sau đó, chép tập tin đĩa tới máy in, nó giải phóng hai tài nguyên này và kết thúc.

Hai giao thức này có hai nhược điểm chủ yếu. Thứ nhất, việc sử dụng tài nguyên có thể chậm vì nhiều tài nguyên có thể được cấp nhưng không được sử dụng trong thời gian dài. Trong thí dụ được cho, chúng ta có thể giải phóng băng từ và tập tin đĩa, sau đó yêu cầu lại tập tin đĩa và máy in chỉ nếu chúng ta đảm bảo rằng dữ liệu của chúng ta sẽ vẫn còn trên tập tin đĩa. Nếu chúng ta không thể đảm bảo rằng dữ liệu

Thuviendientu.org

vẫn còn tập tin đĩa thì chúng ta phải yêu cầu tất cả tài nguyên tại thời điểm bắt đầu cho cả hai giao thức. Thứ hai, đói tài nguyên là có thể. Một quá trình cần nhiều tài nguyên phổ biến có thể phải đợi vô hạn định vì một tài nguyên mà nó cần luôn được cấp phát cho quá trình khác.

VI.3 Không đòi lại tài nguyên từ quá trình đang giữ chúng

Điều kiện cần thứ ba là không đòi lại những tài nguyên đã được cấp phát rồi. Để đảm bảo điều kiện này không xảy ra, chúng ta có thể dùng giao thức sau. Nếu một quá trình đang giữ một số tài nguyên và yêu cầu tài nguyên khác mà không được cấp phát tức thì tới nó (nghĩa là, quá trình phải chờ) thì tất cả tài nguyên hiện đang giữ được đòi lại. Nói cách khác, những tài nguyên này được giải phóng hoàn toàn. Những tài nguyên bị đòi lại được thêm tới danh sách các tài nguyên mà quá trình đang chờ. Quá trình sẽ được khởi động lại chỉ khi nó có thể nhận lại tài nguyên cũ của nó cũng như các tài nguyên mới mà nó đang yêu cầu.

Có một sự chọn lựa khác, nếu một quá trình yêu cầu một số tài nguyên, đầu tiên chúng ta kiểm tra chúng có sẵn không. Nếu tài nguyên có sẵn, chúng ta cấp phát chúng. Nếu tài nguyên không có sẵn, chúng ta kiểm tra chúng có được cấp phát tới một số quá trình khác đang chờ tài nguyên bổ sung. Nếu đúng như thế, chúng ta lấy lại tài nguyên mong muốn đó từ quá trình đang đợi và cấp chúng cho quá trình đang yêu cầu. Nếu tài nguyên không sẵn có hay được giữ bởi một quá trình đang đợi, quá trình đang yêu cầu phải chờ. Trong khi nó đang chờ, một số tài nguyên của nó có thể được đòi lại chỉ nếu quá trình khác yêu cầu chúng. Một quá trình có thể được khởi động lại chỉ khi nó được cấp các tài nguyên mới mà nó đang yêu cầu và phục hồi bất cứ tài nguyên nào đã bị lấy lại trong khi nó đang chờ.

Giao thức này thường được áp dụng tới tài nguyên mà trạng thái của nó có thể được lưu lại dễ dàng và phục hồi lại sau đó, như các thanh ghi CPU và không gian bộ nhớ. Nó thường không thể được áp dụng cho các tài nguyên như máy in và băng từ.

VI.4 Tồn tại chu trình trong đồ thị cấp phát tài nguyên

Điều kiện thứ tư và cũng là điều kiện cuối cùng cho deadlock là điều kiện tồn tại chu trình trong đồ thị cấp phát tài nguyên. Một cách để đảm bảo rằng điều kiện này không bao giờ xảy ra là áp đặt toàn bộ thứ tự của tất cả loại tài nguyên và đòi hỏi mỗi quá trình trong thứ tự tăng của số lượng.

Gọi $R = \{R_1, R_2, \ldots, R_m\}$ là tập hợp loại tài nguyên. Chúng ta gán mỗi loại tài nguyên một số nguyên duy nhất, cho phép chúng ta so sánh hai tài nguyên và xác định tài nguyên này có đứng trước tài nguyên khác hay không trong thứ tự của chúng ta. Thông thường, chúng ta định nghĩa hàm ánh xạ một-một $F: R \rightarrow N$, ở đây N là tập hợp các số tự nhiên. Thí dụ, nếu tập hợp các loại tài nguyên R gồm các ổ băng từ, ổ đĩa và máy in thì hàm F có thể được định nghĩa như sau:

F(ổ băng từ) = 1,
F(đĩa từ) = 5,
F(máy in) = 12.

Bây giờ chúng ta xem giao thức sau để ngăn chặn deadlock: mỗi quá trình có thể yêu cầu tài nguyên chỉ trong thứ tự tăng của số lượng. Nghĩa là, một quá trình ban đầu có thể yêu cầu bất cứ số lượng thể hiện của một loại tài nguyên R_i. Sau đó, một quá trình có thể yêu cầu các thể hiện của loại tài nguyên R_j nếu và chỉ nếu $F(R_j) > F(R_i)$. Nếu một số thể hiện của cùng loại tài nguyên được yêu cầu, thì một yêu cầu cho tất cả thể hiện phải được cấp phát. Thí dụ, sử dụng hàm được định nghĩa trước đó,

Thuviendientu.org

một quá trình muốn dùng ổ băng từ và máy in tại cùng một lúc trước tiên phải yêu cầu ổ băng từ và sau đó yêu cầu máy in.

Nói một cách khác, chúng ta yêu cầu rằng, bất cứ khi nào một quá trình yêu cầu một thể hiện của loại tài nguyên R_j, nó giải phóng bất cứ tài nguyên R_i sao cho $F(R_i) \geq F(R_j)$.

Nếu có hai giao thức được dùng thì điều kiện tồn tại chu trình không thể xảy ra. Chúng ta có thể giải thích điều này bằng cách cho rằng tồn tại chu trình trong đồ thị cấp phát tài nguyên tồn tại. Gọi tập hợp các quá trình chứa tồn tại chu trình trong đồ thị cấp phát tài nguyên là $\{P_0, P_1, \ldots, P_n\}$, ở đây P_i đang chờ một tài nguyên R_i, mà R_i được giữ bởi quá trình P_{i+1}. Vì sau đó quá trình P_{i+1} đang giữ tài nguyên R_i trong khi yêu cầu tài nguyên R_{i+1}, nên chúng ta có $F(R_i) < F(R_{i+1})$ cho tất cả i. Nhưng điều kiện này có nghĩa là $F(R_0) < F(R_1) < \ldots < F(R_n) < F(R_0)$. Bằng qui tắc bắt cầu $F(R_0) < F(R_0)$, điều này là không thể. Do đó, không thể có chờ chu trình.

Chú ý rằng hàm F nên được định nghĩa dựa theo thứ tự tự nhiên của việc sử dụng tài nguyên trong hệ thống. Thí dụ, vì ổ băng từ thường được yêu cầu trước máy in nên có thể hợp lý để định nghĩa F(ổ băng từ) < F(máy in).

VII Tránh deadlock

Các giải thuật ngăn chặn deadlock, được thảo luận ở VII-6, ngăn chặn deadlock bằng cách hạn chế cách các yêu cầu có thể được thực hiện. Các ngăn chặn đảm bảo rằng ít nhất một trong những điều kiện cần cho deadlock không thể xảy ra. Do đó, deadlock không thể xảy ra. Tuy nhiên, các tác dụng phụ có thể ngăn chặn deadlock bởi phương pháp này là việc sử dụng thiết bị chậm và thông lượng hệ thống bị giảm.

Một phương pháp khác để tránh deadlock là yêu cầu thông tin bổ sung về cách tài nguyên được yêu cầu. Thí dụ, trong một hệ thống với một ổ băng từ và một máy in, chúng ta có thể bảo rằng quá trình P sẽ yêu cầu ổ băng từ trước và sau đó máy in trước khi giải phóng cả hai tài nguyên. Trái lại, quá trình Q sẽ yêu cầu máy in trước và sau đó ổ băng từ. Với kiến thức về thứ tự hoàn thành của yêu cầu và giải phóng cho mỗi quá trình, chúng ta có thể quyết định cho mỗi yêu cầu của quá trình sẽ chờ hay không. Mỗi yêu cầu đòi hỏi hệ thống xem tài nguyên hiện có, tài nguyên hiện được cấp tới mỗi quá trình, và các yêu cầu và giải phóng tương lai của mỗi quá trình, để yêu cầu của quá trình hiện tại có thể được thoả mãn hay phải chờ để tránh khả năng xảy ra deadlock.

Các giải thuật khác nhau có sự khác nhau về lượng và loại thông tin được yêu cầu. Mô hình đơn giản và hữu ích nhất yêu cầu mỗi quá trình khai báo số lớn nhất tài nguyên của mỗi loại mà nó cần. Thông tin trước về số lượng tối đa tài nguyên của mỗi loại được yêu cầu cho mỗi quá trình, có thể xây dựng một giải thuật đảm bảo hệ thống sẽ không bao giờ đi vào trạng thái deadlock. Đây là giải thuật định nghĩa tiếp cận tránh deadlock. Giải thuật tránh deadlock tự xem xét trạng thái cấp phát tài nguyên để đảm bảo điều kiện tồn tại chu trình trong đồ thị cấp phát tài nguyên có thể không bao giờ xảy ra. Trạng thái cấp phát tài nguyên được định nghĩa bởi số tài nguyên sẵn dùng và tài nguyên được cấp phát và số yêu cầu tối đa của các quá trình.

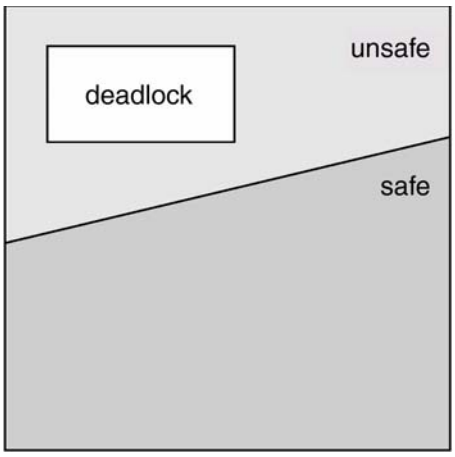
VII.1 Trạng thái an toàn

Một trạng thái là an toàn nếu hệ thống có thể cấp phát các tài nguyên tới mỗi quá trình trong một vài thứ tự và vẫn tránh deadlock. Hay nói cách khác, một hệ thống ở trong trạng thái an toàn chỉ nếu ở đó tồn tại một thứ tự an toàn. Thứ tự của các quá trình $<P_1, P_2, \ldots, P_n>$ là một thứ tự an toàn cho trạng thái cấp phát hiện hành nếu đối

Thuviendientu.org

với mỗi thứ tự P_i, các tài nguyên mà P_i yêu cầu vẫn có thể được thoả mãn bởi tài nguyên hiện có cộng với các tài nguyên được giữ bởi tất cả P_j, với j<i. Trong trường hợp này, nếu những tài nguyên mà quá trình P_i yêu cầu không sẵn dùng tức thì thì P_i có thể chờ cho đến khi tất cả P_j hoàn thành. Khi chúng hoàn thành, P_i có thể đạt được tất cả những tài nguyên nó cần, hoàn thành các tác vụ được gán, trả về những tài nguyên được cấp phát cho nó và kết thúc. Khi P_i kết thúc, P_{i+1} có thể đạt được các tài nguyên nó cần,... Nếu không có thứ tự như thế tồn tại thì trạng thái hệ thống là không an toàn.

Một trạng thái an toàn không là trạng thái deadlock. Do đó, trạng thái deadlock là trạng thái không an toàn. Tuy nhiên, không phải tất cả trạng thái không an toàn là deadlock (hình VI-4). Một trạng thái không an toàn có thể dẫn đến deadlock. Với điều kiện trạng thái là an toàn, hệ điều hành có thể tránh trạng thái không an toàn (và deadlock). Trong một trạng thái không an toàn, hệ điều hành có thể ngăn chặn các quá trình từ những tài nguyên đang yêu cầu mà deadlock xảy ra: hành vi của các quá trình này điều khiển các trạng thái không an toàn.



Hình 0-4 Không gian trạng thái an toàn, không an toàn, deadlock

Để minh hoạ, chúng ta xét một hệ thống với 12 ổ băng từ và 3 quá trình: P_0, P_1, P_2. Quá trình P_0 yêu cầu 10 ổ băng từ, quá trình P_1 có thể cần 4 và quá trình P_2 có thể cần tới 9 ổ băng từ. Giả sử rằng tại thời điểm t_0, quá trình P_0 giữ 5 ổ băng từ, quá trình P_1 giữ 2 và quá trình P_2 giữ 2 ổ băng từ. (Do đó, có 3 ổ băng từ còn rảnh).

	Nhu cầu tối đa	Nhu cầu hiện tại
P0	10	5
P1	4	2
P2	9	2

Tại thời điểm t_0, hệ thống ở trạng thái an toàn. Thứ tự $<P_1,P_0, P_2>$ thoả điều kiện an toàn vì quá trình P_1 có thể được cấp phát tức thì tất cả các ổ đĩa từ và sau đó trả lại chúng (sau đó hệ thống có 5 ổ băng từ sẵn dùng), sau đó quá trình P_0 có thể nhận tất cả ổ băng từ và trả lại chúng (sau đó hệ thống sẽ có 10 ổ băng từ sẵn dùng), và cuối cùng quá trình P_2 có thể nhận tất cả ổ băng từ của nó và trả lại chúng (sau đó hệ thống sẽ có tất cả 12 ổ băng từ sẵn dùng).

Một hệ thống có thể đi từ trạng thái an toàn tới một trạng thái không an toàn. Giả sử rằng tại thời điểm t_1, quá trình P_2 yêu cầu và được cấp 1 ổ băng từ nữa. Hệ thống không còn trong trạng thái an toàn. Tại điểm này, chỉ quá trình P_1 có thể được cấp tất cả ổ băng từ của nó. Khi nó trả lại chúng, chỉ quá trình P_1 có thể được cấp phát tất cả ổ băng từ. Khi nó trả lại chúng, hệ thống chỉ còn 4 ổ băng từ sẵn có. Vì quá

Thuviendientu.org

trình P_0 được cấp phát 5 ổ băng từ, nhưng có tối đa 10, quá trình P_0 phải chờ. Tương tự, quá trình P_2 có thể yêu cầu thêm 6 ổ băng từ và phải chờ dẫn đến deadlock.

Lỗi của chúng ta là gán yêu cầu từ quá trình P_2 cho 1 ổ băng từ nữa. Nếu chúng ta làm cho P_2 phải chờ cho đến khi các quá trình khác kết thúc và giải phóng tài nguyên của nó thì chúng ta có thể tránh deadlock.

Với khái niệm trạng thái an toàn được cho, chúng ta có thể định nghĩa các giải thuật tránh deadlock. Ý tưởng đơn giản là đảm bảo hệ thống sẽ luôn còn trong trạng thái an toàn. Khởi đầu, hệ thống ở trong trạng thái an toàn. Bất cứ khi nào một quá trình yêu cầu một tài nguyên hiện có, hệ thống phải quyết định tài nguyên có thể được cấp phát tức thì hoặc quá trình phải chờ. Yêu cầu được gán chỉ nếu việc cấp phát để hệ thống trong trạng thái an toàn.

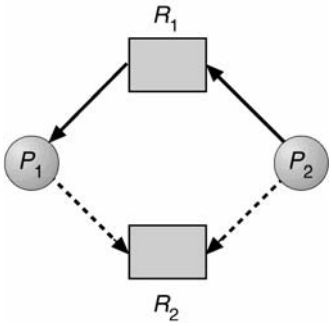
Trong mô hình này, nếu quá trình yêu cầu tài nguyên đang có, nó có thể vẫn phải chờ. Do đó, việc sử dụng tài nguyên có thể chậm hơn mà không có giải thuật tránh deadlock.

VII.2 Giải thuật đồ thị cấp phát tài nguyên

Nếu chúng ta có một hệ thống cấp phát tài nguyên với một thể hiện của mỗi loại, một biến dạng của đồ thị cấp phát tài nguyên được định nghĩa trong phần VI.4.2 có thể được dùng để tránh deadlock.

Ngoài các cạnh yêu cầu và gán, chúng ta giới thiệu một loại cạnh mới được gọi là ***cạnh thỉnh cầu*** (claim edge). Một cạnh thỉnh cầu $P_i \rightarrow R_j$ hiển thị quá trình P_i có thể yêu cầu tài nguyên R_j vào một thời điểm trong tương lai. Cạnh này tương tự cạnh yêu cầu về phương hướng nhưng được hiện diện bởi dấu đứt khoảng. Khi quá trình P_i yêu cầu tài nguyên R_j, cạnh thỉnh cầu $P_i \rightarrow R_j$ chuyển tới cạnh yêu cầu. Tương tự, khi một tài nguyên R_j được giải phóng bởi P_i, cạnh gán $R_j \rightarrow P_i$ được chuyển trở lại thành cạnh thỉnh cầu $P_i \rightarrow R_j$. Chúng ta chú ý rằng các tài nguyên phải được yêu cầu trước trong hệ thống. Nghĩa là, trước khi P_i bắt đầu thực thi, tất cả các cạnh thỉnh cầu của nó phải xuất hiện trong đồ thị cấp phát tài nguyên. Chúng ta có thể giảm nhẹ điều kiện này bằng cách cho phép một cạnh $P_i \rightarrow R_j$ để được thêm tới đồ thị chỉ nếu tất cả các cạnh gắn liền với quá trình P_i là các cạnh thỉnh cầu.

Giả sử rằng P_i yêu cầu tài nguyên R_j. Yêu cầu có thể được gán chỉ nếu chuyển cạnh yêu cầu $P_i \rightarrow R_j$ tới cạnh gán $R_j \rightarrow P_i$ không dẫn đến việc hình thành chu trình trong đồ thị cấp phát tài nguyên. Chú ý rằng chúng ta kiểm tra tính an toàn bằng cách dùng giải thuật phát hiện chu trình. Một giải thuật để phát hiện một chu trình trong đồ thị này yêu cầu một thứ tự của n^2 thao tác, ở đây n là số quá trình trong hệ thống.



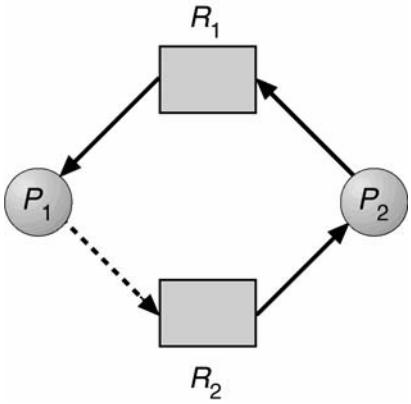
Hình 0-5 Đồ thị cấp phát tài nguyên để tránh deadlock

Nếu không có chu trình tồn tại, thì việc cấp phát tài nguyên sẽ để lại hệ thống trong trạng thái an toàn. Nếu chu trình được tìm thấy thì việc cấp phát sẽ đặt hệ thống

Thuviendientu.org

trong trạng thái không an toàn. Do đó, quá trình P_i sẽ phải chờ yêu cầu của nó được thoả.

Để minh hoạ giải thuật này, chúng ta xét đồ thị cấp phát tài nguyên của hình VI-5. Giả sử rằng P_2 yêu cầu R_2. Mặc dù R_2 hiện rảnh nhưng chúng ta không thể cấp phát nó tới P_2 vì hoạt động này sẽ tạo ra chu trình trong đồ thị (Hình VI-6). Một chu trình hiển thị rằng hệ thống ở trong trạng thái không an toàn. Nếu P_1 yêu cầu R_2 và P_2 yêu cầu R_1 thì deadlock sẽ xảy ra.



Hình 0-6 Trạng thái không an toàn trong đồ thị cấp phát tài nguyên

VII.3 Giải thuật của Banker

Giải thuật đồ thị cấp phát tài nguyên không thể áp dụng tới hệ thống cấp phát tài nguyên với nhiều thể hiện của mỗi loại tài nguyên. Giải thuật tránh deadlock mà chúng ta mô tả tiếp theo có thể áp dụng tới một hệ thống nhưng ít hiệu quả hơn cơ chế đồ thị cấp phát tài nguyên. Giải thuật này thường được gọi là giải thuật của **Banker**. Tên được chọn vì giải thuật này có thể được dùng trong hệ thống ngân hàng để đảm bảo ngân hàng không bao giờ cấp phát tiền mặt đang có của nó khi nó không thể thoả mãn các yêu cầu của tất cả khách hàng.

Khi một quá trình mới đưa vào hệ thống, nó phải khai báo số tối đa các thể hiện của mỗi loại tài nguyên mà nó cần. Số này có thể không vượt quá tổng số tài nguyên trong hệ thống. Khi một người dùng yêu cầu tập hợp các tài nguyên, hệ thống phải xác định việc cấp phát của các tài nguyên này sẽ để lại hệ thống ở trạng thái an toàn hay không. Nếu trạng thái hệ thống sẽ là an toàn, tài nguyên sẽ được cấp; ngược lại quá trình phải chờ cho tới khi một vài quá trình giải phóng đủ tài nguyên.

Nhiều cấu trúc dữ liệu phải được duy trì để cài đặt giải thuật Banker. Những cấu trúc dữ liệu này mã hoá trạng thái của hệ thống cấp phát tài nguyên. Gọi n là số quá trình trong hệ thống và m là số loại tài nguyên trong hệ thống. Chúng ta cần các cấu trúc dữ liệu sau:

- **Available**: một vector có chiều dài m hiển thị số lượng tài nguyên sẵn dùng của mỗi loại. Nếu Available[j]= k, có k thể hiện của loại tài nguyên R_j sẵn dùng.
- **Max**: một ma trận n x m định nghĩa số lượng tối đa yêu cầu của mỗi quá trình. Nếu Max[i , j] = k, thì quá trình P_i có thể yêu cầu nhiều nhất k thể hiện của loại tài nguyên R_j.
- **Allocation**: một ma trận n x m định nghĩa số lượng tài nguyên của mỗi loại hiện được cấp tới mỗi quá trình. Nếu Allocation[i, j] = k, thì quá trình P_i hiện được cấp k thể hiện của loại tài nguyên R_j.

Thuviendientu.org

- **Need**: một ma trận n x m hiển thị yêu cầu tài nguyên còn lại của mỗi quá trình. Nếu Need[i, j] = k, thì quá trình P_i có thể cần thêm k thể hiện của loại tài nguyên R_j để hoàn thành tác vụ của nó. Chú ý rằng, Need[i, j] = Max[i, j] – Allocation [i, j].

Cấu trúc dữ liệu này biến đổi theo thời gian về kích thước và giá trị

Để đơn giản việc trình bày của giải thuật Banker, chúng ta thiết lập vài ký hiệu. Gọi X và Y là các vector có chiều dài n. Chúng ta nói rằng X ≤ Y nếu và chỉ nếu X[i] ≤ Y[i] cho tất cả i = 1, 2, …, n. Thí dụ, nếu X = (1, 7, 3, 2) và Y = (0, 3, 2, 1) thì Y ≤ X, Y < X nếu Y ≤ X và Y ≠ X.

Chúng ta có thể xem xét mỗi dòng trong ma trận Allocation và Need như là những vectors và tham chiếu tới chúng như $Allocation_i$ và $Need_i$ tương ứng. Vector $Allocation_i$ xác định tài nguyên hiện được cấp phát tới quá trình P_i; vector $Need_i$ xác định các tài nguyên bổ sung mà quá trình P_i có thể vẫn yêu cầu để hoàn thành tác vụ của nó.

VII.3.1 Giải thuật an toàn

Giải thuật để xác định hệ thống ở trạng thái an toàn hay không có thể được mô tả như sau:

1) Gọi Work và Finish là các vector có chiều dài m và n tương ứng. Khởi tạo Work:=Available và Finish[i]:=false cho i = 1, 2, …,n.
2) Tìm i thỏa:
   a) Finish[i] = false
   b) $Need_i \le Work$.

   Nếu không có i nào thỏa, di chuyển tới bước 4
3) Work:=Work + $Allocation_i$
   Finish[i] := true
   Di chuyển về bước 2.
4) Nếu Finish[i] = true cho tất cả i, thì hệ thống đang ở trạng thái an toàn.

Giải thuật này có thể yêu cầu độ phức tạp mxn^2 thao tác để quyết định trạng thái là an toàn hay không.

VII.3.2 Giải thuật yêu cầu tài nguyên

Cho $Request_i$ là vector yêu cầu cho quá trình P_i. Nếu $Request_i[j] = k$, thì quá trình P_i muốn k thể hiện của loại tài nguyên R_j. Khi một yêu cầu tài nguyên được thực hiện bởi quá trình P_i, thì các hoạt động sau được thực hiện:

1) Nếu $Request_i \le Need_i$, di chuyển tới bước 2. Ngược lại, phát sinh một điều kiện lỗi vì quá trình vượt quá yêu cầu tối đa của nó.
2) Nếu $Request_i \le Available$, di chuyển tới bước 3. Ngược lại, P_i phải chờ vì tài nguyên không sẵn có.
3) Giả sử hệ thống cấp phát các tài nguyên được yêu cầu tới quá trình P_i bằng cách thay đổi trạng thái sau:

   $Available := Available – Request_i$;
   $Allocation_i := Allocation_i + Request_i$;
   $Need_i := Need_i – Request_i$;

Nếu kết quả trạng thái cấp phát tài nguyên là an toàn, thì giao dịch được hoàn thành và quá trình P_i được cấp phát tài nguyên của nó. Tuy nhiên, nếu trạng thái mới là không an toàn, thì P_i phải chờ $Request_i$ và trạng thái cấp phát tài nguyên cũ được phục hồi.

Thuviendientu.org

VII.3.3 Thí dụ minh họa

Xét một hệ thống với 5 quá trình từ P_0 tới P_4, và 3 loại tài nguyên A, B, C. Loại tài nguyên A có 10 thể hiện, loại tài nguyên B có 5 thể hiện và loại tài nguyên C có 7 thể hiện. Giả sử rằng tại thời điểm T_0 trạng thái hiện tại của hệ thống như sau:

	Allocation			Max			Available		
	A	B	C	A	B	C	A	B	C
P0	0	1	0	7	5	3	3	3	2
P1	2	0	0	3	2	2			
P2	3	0	2	9	0	2			
P3	2	1	1	2	2	2			
P4	0	0	2	4	3	3			

Nội dung ma trận ***Need*** được định nghĩa là ***Max-Allocation*** và là

	Need		
	A	B	C
P0	7	4	3
P1	1	2	2
P2	6	0	2
P3	0	1	1
P4	4	3	1

Chúng ta khẳng định rằng hệ thống hiện ở trong trạng thái an toàn. Thật vậy, thứ tự $<P_1, P_3, P_4, P_2, P_0>$ thỏa tiêu chuẩn an toàn. Giả sử bây giờ P_1 yêu cầu thêm một thể hiện loại A và hai thể hiện loại C, vì thế $Request_1 = (1, 0, 2)$. Để quyết định yêu cầu này có thể được cấp tức thì hay không, trước tiên chúng ta phải kiểm tra $Request_1 \le Available$ (nghĩa là, $(1, 0, 2)) \le (3, 3, 2)$) là đúng hay không. Sau đó, chúng ta giả sử yêu cầu này đạt được và chúng ta đi đến trạng thái mới sau:

	Allocation			Max			Available		
	A	B	C	A	B	C	A	B	C
P0	0	1	0	7	4	3	2	3	0
P1	3	0	2	0	2	0			
P2	3	0	2	6	0	0			
P3	2	1	1	0	1	1			
P4	0	0	2	4	3	1			

Chúng ta phải xác định trạng thái mới này là an toàn hay không. Để thực hiện điều này, chúng ta thực thi giải thuật an toàn của chúng ta và tìm thứ tự $<P_1, P_3, P_4, P_0, P_2>$ thỏa yêu cầu an toàn. Do đó, chúng ta có thể cấp lập tức yêu cầu của quá trình P_1.

Tuy nhiên, chúng ta cũng thấy rằng, khi hệ thống ở trong trạng thái này, một yêu cầu (3, 3, 0) bởi P_4 không thể được gán vì các tài nguyên là không sẵn dùng. Một yêu cầu cho (0, 2, 0) bởi P_0 không thể được cấp mặc dù tài nguyên là sẵn dùng vì trạng thái kết quả là không an toàn.

Thuviendientu.org

VIII Phát hiện Deadlock

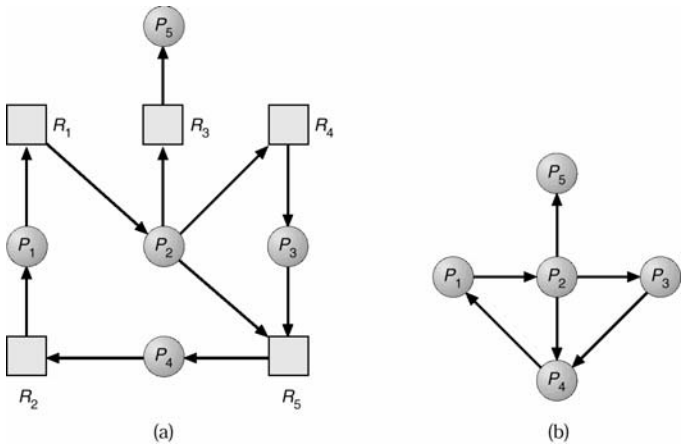
Nếu một hệ thống không thực hiện giải thuật ngăn chặn deadlock hay tránh deadlock thì trường hợp deadlock có thể xảy ra. Trong môi trường này, hệ thống phải cung cấp:

- Giải thuật xem xét trạng thái của hệ thống để quyết định deadlock có xảy ra hay không.
- Giải thuật phục hồi từ deadlock

Trong thảo luận dưới đây, chúng ta thảo luận chi tiết về hai yêu cầu khi chúng liên quan đến những hệ thống với chỉ một thể hiện của mỗi loại tài nguyên cũng như đối với hệ thống có nhiều thể hiện cho mỗi loại tài nguyên. Tuy nhiên, tại thời điểm này chúng ta chú ý lược đồ phát hiện và phục hồi yêu cầu chi phí bao gồm không chỉ chi phí tại thời điểm thực thi cho việc duy trì thông tin cần thiết và thực thi giải thuật phát hiện mà còn các lãng phí có thể phát sinh trong việc phát hiện từ deadlock.

VIII.1 Một thể hiện của mỗi loại tài nguyên

Nếu tất cả tài nguyên chỉ có một thể hiện thì chúng ta có thể định nghĩa giải thuật phát hiện deadlock dùng một biến dạng của đồ thị cấp phát tài nguyên, được gọi là đồ thị chờ (wait-for). Chúng ta đạt được đồ thị này từ đồ thị cấp phát tài nguyên bằng cách gỡ bỏ các nút của loại tài nguyên và xóa các cạnh tương ứng.



Hình 0-7 a) Đồ thị cấp phát tài nguyên. b) Đồ thị chờ tương ứng

Chính xác hơn, một cạnh từ P_i tới P_j trong đồ thị chờ hiển thị rằng quá trình P_i đang chờ một quá trình P_j để giải phóng tài nguyên mà P_i cần. Cạnh $P_i \rightarrow P_j$ tồn tại trong đồ thị chờ nếu và chỉ nếu đồ thị cấp phát tài nguyên tương ứng chứa hai cạnh $P_i \rightarrow R_q$ và $R_q \rightarrow P_j$ đối với một số tài nguyên R_q. Thí dụ, trong hình VI-7 dưới đây, chúng ta trình bày đồ thị cấp phát tài nguyên và đồ thị chờ tương ứng.

Như đã đề cập trước đó, deadlock tồn tại trong hệ thống nếu và chỉ nếu đồ thị chờ chứa chu trình. Để phát hiện deadlock, hệ thống cần duy trì đồ thị chờ và định kỳ gọi giải thuật để tìm kiếm chu trình trong đồ thị.

Một giải thuật phát hiện chu trình trong đồ thị yêu cầu độ phức tạp n^2 thao tác, ở đây n là số cạnh của đồ thị.

Thuviendientu.org

VIII.2 Nhiều thể hiện của một loại tài nguyên

Lược đồ đồ thị chờ không thể áp dụng đối với hệ thống cấp phát tài nguyên với nhiều thể hiện cho mỗi loại tài nguyên. Giải thuật phát hiện deadlock mà chúng ta mô tả sau đây có thể áp dụng cho hệ thống này. Giải thuật thực hiện nhiều cấu trúc dữ liệu thay đổi theo thời gian mà chúng tương tự như được dùng trong giải thuật Banker.

- **Available**: một vector có chiều dài m hiển thị số lượng tài nguyên sẵn có của mỗi loại.
- **Allocation**: ma trận nxm định nghĩa số lượng tài nguyên của mỗi loại hiện được cấp phát tới mỗi quá trình.
- **Request**: ma trận nxm hiển thị yêu cầu hiện tại của mỗi quá trình. Nếu Request[i,j] = k, thì quá trình P_i đang yêu cầu k thể hiện nữa của loại tài nguyên R_j.

Quan hệ ≤ giữa hai vectors được định nghĩa trong phần VI.7.3. Để ký hiệu đơn giản, chúng ta sẽ xem những hàng trong ma trận Allocation và Request như các vector, và sẽ tham chiếu chúng như $Allocation_i$ và $Request_i$ tương ứng. Giải thuật phát hiện được mô tả ở đây đơn giản khảo sát mọi thứ tự cấp phát có thể đối với các quá trình còn lại để được hoàn thành. So sánh giải thuật này với giải thuật Banker.

1) Gọi Work và Finish là các vector có chiều dài m và n tương ứng. Khởi tạo Work:=Available. Cho i = 1, 2, …,n, nếu $Allocation_i \neq 0$, thì Finish[i]:= false; ngược lại Finish[i]:= true.
2) Tìm chỉ số i thỏa:
   a) Finish[i] = false
   b) $Request_i \leq Work$.

   Nếu không có i nào thỏa, di chuyển tới bước 4
3) Work:=Work + $Allocation_i$
   Finish[i] := true
   Di chuyển về bước 2.
4) Nếu Finish[i] = false cho một vài i, $1 \leq i \leq n$ thì hệ thống đang ở trạng thái deadlock. Ngoài ra, nếu Finish[i] = false thì quá trình P_i bị deadlock.

Giải thuật này yêu cầu độ phức tạp mxn^2 để phát hiện hệ thống có ở trong trạng thái deadlock hay không.

Câu hỏi đặt ra là tại sao chúng ta trả lại tài nguyên của quá trình P_i (trong bước 3) ngay sau khi chúng ta xác định rằng $Request_i \leq Work$ (trong bước 2b). Chúng ta biết rằng P_i hiện tại không liên quan deadlock (vì $Request_i \leq Work$). Do đó, chúng ta lạc quan và khẳng định rằng P_i sẽ không yêu cầu thêm tài nguyên nữa để hoàn thành công việc của nó; do đó nó sẽ trả về tất cả tài nguyên hiện được cấp phát tới hệ thống. Nếu giả định của chúng ta không đúng, deadlock có thể xảy ra sao đó. Deadlock sẽ được phát hiện tại thời điểm kế tiếp mà giải thuật phát hiện deadlock được nạp.

Để minh hoạ giải thuật này, chúng ta xét hệ thống với 5 quá trình P_0 đến P_4 và 3 loại tài nguyên A, B, C. Loại tài nguyên A có 7 thể hiện, loại tài nguyên B có 2 thể hiện và loại tài nguyên C có 6 thể hiện. Giả sử rằng tại thời điểm T_0, chúng ta có trạng thái cấp phát tài nguyên sau:

	Allocation			Request			Available		
	A	B	C	A	B	C	A	B	C
P0	0	1	0	0	0	0	0	0	0
P1	2	0	0	2	0	2			
P2	3	0	3	0	0	0			
P3	2	1	1	1	0	0			

Thuviendientu.org

P4	0	0	2	0	0	2			

Chúng ta khẳng định rằng hệ thống không ở trong trạng thái deadlock. Thật vậy, nếu chúng ta thực thi giải thuật, chúng ta sẽ tìm ra thứ tự <P_0, P_2, P_3, P_1, P_4> sẽ dẫn đến trong Finish[i] = true cho tất cả i.

Bây giờ giả sử rằng quá trình P_2 thực hiện yêu cầu thêm thể hiện loại C. Ma trận yêu cầu được hiệu chỉnh như sau:

	Need		
	A	B	C
P0	0	0	0
P1	2	0	2
P2	0	0	1
P3	1	0	0
P4	0	0	2

Chúng ta khẳng định rằng hệ thống bây giờ bị deadlock. Mặc dù chúng ta có thể đòi lại tài nguyên được giữ bởi quá trình P_0, số lượng tài nguyên sẵn dùng không đủ để hoàn thành các yêu cầu của các quá trình còn lại. Do đó, deadlock tồn tại và bao gồm các quá trình P_1, P_2, P_3, P_4.

VIII.3 Sử dụng giải thuật phát hiện deadlock

Khi nào thì chúng ta nên nạp giải thuật phát hiện deadlock? Câu trả lời phụ thuộc vào hai yếu tố:

1) Deadlock có khả năng xảy ra thường xuyên như thế nào?
2) Bao nhiêu quá trình sẽ bị ảnh hưởng bởi deadlock khi nó sẽ ra?

Nếu deadlock xảy ra thường xuyên thì giải thuật phát hiện nên được nạp lên thường xuyên. Những tài nguyên được cấp phát để các quá trình bị deadlock sẽ rảnh cho đến khi deadlock có thể bị phá vỡ. Ngoài ra, số lượng quá trình liên quan trong chu trình deadlock có thể tăng lên.

Deadlock xảy ra chỉ khi một số quá trình thực hiện yêu cầu mà không được cấp tài nguyên tức thì. Yêu cầu này có thể là yêu cầu cuối hoàn thành một chuỗi các quá trình đang yêu cầu. Ngoài ra, chúng ta có thể nạp giải thuật phát hiện mọi khi một yêu cầu cho việc cấp phát không thể được cấp tức thì. Trong trường hợp này, chúng ta không chỉ định nghĩa tập hợp các quá trình bị deadlock, mà còn xác định quá trình đã gây ra deadlock. (Trong thực tế, mỗi quá trình trong suốt quá trình bị deadlock là một liên kết trong chu trình của đồ thị tài nguyên, vì thế tất cả chúng gây ra deadlock). Nếu có nhiều loại tài nguyên khác nhau, một yêu cầu có thể gây chu trình trong đồ thị tài nguyên, mỗi chu trình hoàn thành bởi yêu cầu mới nhất và "được gây ra" bởi một quá trình có thể xác định.

Dĩ nhiên, nạp giải thuật phát hiện deadlock cho mỗi yêu cầu có thể gây ra một chi phí có thể xem xét trong thời gian tính toán. Một thay đổi ít đắt hơn là nạp giải thuật tại thời điểm ít thường xuyên hơn- thí dụ, một lần một giờ hay bất cứ khi nào việc sử dụng CPU rơi xuống thấp hơn 40%. Nếu giải thuật phát hiện deadlock được nạp trong những thời điểm bất kỳ, thì có nhiều chu trình trong đồ thị tài nguyên. Chúng ta không thể nói quá trình nào của nhiều quá trình bị deadlock gây ra deadlock.

Thuviendientu.org

IX Phục hồi deadlock

Khi giải thuật phát hiện xác định rằng deadlock tồn tại, nhiều thay đổi tồn tại. Một khả năng là thông báo người điều hành rằng deadlock xảy ra và để người điều hành giải quyết deadlock bằng thủ công. Một khả năng khác là để hệ thống tự động phục hồi. Có hai tuỳ chọn cho việc phá vỡ deadlock. Một giải pháp đơn giản là huỷ bỏ một hay nhiều quá trình để phá vỡ việc tồn tại chu trình trong đồ thị cấp phát tài nguyên. Tuỳ chọn thứ hai là lấy lại một số tài nguyên từ một hay nhiều quá trình bị deadlock.

IX.1 Kết thúc quá trình

Để xóa deadlock bằng cách hủy bỏ quá trình, chúng ta dùng một trong hai phương pháp. Trong cả hai phương pháp, hệ thống lấy lại tài nguyên được cấp phát đối với quá trình bị kết thúc.

- **Huỷ bỏ tất cả quá trình bị deadlock**: phương pháp này rõ ràng sẽ phá vỡ chu trình deadlock, nhưng chi phí cao; các quá trình này có thể đã tính toán trong thời gian dài, và các kết quả của các tính toán từng phần này phải bị bỏ đi và có thể phải tính lại sau đó.
- **Hủy bỏ một quá trình tại thời điểm cho đến khi chu trình deadlock bị xóa**:phương pháp này chịu chi phí có thể xem xét vì sau khi mỗi quá trình bị hủy bỏ, một giải thuật phát hiện deadlock phải được nạp lên để xác định có quá trình nào vẫn đang bị deadlock.

Hủy bỏ quá trình có thể không dễ. Nếu một quá trình đang ở giữa giai đoạn cập nhật một tập tin, kết thúc nó sẽ để tập tin đó trong trạng thái không phù hợp. Tương tự, nếu quá trình đang ở giữa giai đoạn in dữ liệu ra máy in, hệ thống phải khởi động lại trạng thái đúng trước khi in công việc tiếp theo.

Nếu phương pháp kết thúc một phần được dùng thì với một tập hợp các quá trình deadlock được cho, chúng ta phải xác định quá trình nào (hay các quá trình nào) nên được kết thúc trong sự cố gắng để phá vỡ deadlock. Việc xác định này là một quyết định chính sách tương tự như các vấn đề lập thời biểu CPU. Câu hỏi về tính kinh tế là chúng ta nên hủy bỏ quá trình nào thì sự kết thúc của quá trình đó sẽ chịu chi phí tối thiểu. Tuy nhiên, thuật ngữ chi phí tối thiểu là không chính xác. Nhiều yếu tố có thể xác định quá trình nào được chọn bao gồm:

1) Độ ưu tiên của quá trình là gì.
2) Quá trình đã được tính toán bao lâu và bao lâu nữa quá trình sẽ tính toán trước khi hoàn thành tác vụ được chỉ định của nó.
3) Bao nhiêu và loại tài nguyên gì quá trình đang sử dụng.
4) Bao nhiêu tài nguyên nữa quá trình cần để hoàn thành
5) Bao nhiêu quá trình sẽ cần được kết thúc.
6) Quá trình là giao tiếp hay dạng bó

Thuviendientu.org

IX.2 Lấy lại tài nguyên

Để xóa deadlock sử dụng việc trả lại tài nguyên, sau khi chúng ta đòi một số tài nguyên từ các quá trình và cho các tài nguyên này tới các quá trình khác cho đến khi chu trình deadlock bị phá vỡ.

Nếu việc đòi lại được yêu cầu để giải quyết deadlock thì ba vấn đề cần được xác định:

- **Chọn nạn nhân**: những tài nguyên nào và những quá trình nào bị đòi lại? Trong khi kết thúc quá trình, chúng ta phải xác định thứ tự đòi lại để tối thiểu chi phí. Các yếu tố chi phí có thể gồm các tham số như số lượng tài nguyên một quá trình deadlock đang giữ, và lượng thời gian một quá trình deadlock dùng trong sự thực thi của nó.
- **Trở lại trạng thái trước deadlock**: Nếu chúng ta đòi lại tài nguyên từ một quá trình, điều gì nên được thực hiện với quá trình đó? Rõ ràng, nó không thể tiếp tục việc thực thi bình thường; nó đang bị mất một số tài nguyên được yêu cầu. Chúng ta phải phục hồi quá trình tới trạng thái an toàn và khởi động lại từ trạng thái gần nhất trước đó.

Thông thường, rất khó để xác định trạng thái gì là an toàn vì thế giải pháp đơn giản nhất là phục hồi toàn bộ: hủy quá trình và sau đó khởi động lại nó. Tuy nhiên, hữu hiệu hơn để phục hồi quá trình chỉ đủ xa cần thiết để phá vỡ deadlock. Ngoài ra, phương pháp này yêu cầu hệ thống giữ nhiều thông tin hơn về trạng thái của tất cả các quá trình đang chạy.

Đói tài nguyên: chúng ta đảm bảo như thế nào việc đói tài nguyên không xảy ra? Nghĩa là, chúng ta có thể đảm bảo rằng tài nguyên sẽ không luôn bị đòi lại từ cùng một quá trình.

Trong một hệ thống việc chọn nạn nhân ở đâu dựa trên cơ sở các yếu tố chi phí, nó có thể xảy ra cùng quá trình luôn được chọn như là nạn nhân. Do đó, quá trình này không bao giờ hoàn thành tác vụ được chỉ định của nó, một trường hợp đói tài nguyên cần được giải quyết trong bất kỳ hệ thống thực tế. Rõ ràng, chúng ta phải đảm bảo một quá trình có thể được chọn như một nạn nhân chỉ một số lần xác định (nhỏ). Giải pháp chung nhất là bao gồm số lượng phục hồi trong yếu tố chi phí.

X Tóm tắt

Trạng thái deadlock xảy ra khi hai hay nhiều quá trình đang chờ không xác định một sự kiện mà có thể được gây ra chỉ bởi một trong những quá trình đang chờ. Về nguyên tắc, có ba phương pháp giải quyết deadlock:

- Sử dụng một số giao thức để ngăn chặn hay tránh deadlock, đảm bảo rằng hệ thống sẽ không bao giờ đi vào trạng thái deadlock.
- Cho phép hệ thống đi vào trạng thái deadlock, phát hiện và sau đó phục hồi.
- Bỏ qua vấn đề deadlock và giả vờ deadlock chưa bao giờ xảy ra trong hệ thống. Giải pháp này là một giải pháp được dùng bởi hầu hết các hệ điều hành bao gồm UNIX.

Trường hợp deadlock có thể xảy ra nếu và chỉ nếu bốn điều kiện cần xảy ra cùng một lúc trong hệ thống: loại trừ hỗ tương, giữ và chờ cấp thêm tài nguyên, không đòi lại tài nguyên, và tồn tại chu trình trong đồ thị cấp phát tài nguyên. Để ngăn chặn deadlock, chúng ta đảm bảo rằng ít nhất một điều kiện cần không bao giờ xảy ra.

Một phương pháp để tránh deadlock mà ít nghiêm ngặt hơn giải thuật ngăn chặn deadlock là có thông tin trước về mỗi quá trình sẽ đang dùng tài nguyên như thế nào. Thí dụ, giải thuật Banker cần biết số lượng tối đa của mỗi lớp tài nguyên có thể được

Thuviendientu.org

yêu cầu bởi mỗi quá trình. Sử dụng thông tin này chúng ta có thể định nghĩa giải thuật tránh deadlock.

Nếu hệ thống không thực hiện một giao thức để đảm bảo rằng deadlock sẽ không bao giờ xảy ra thì lược đồ phát hiện và phục hồi phải được thực hiện. Giải thuật phát hiện deadlock phải được nạp lên để xác định deadlock có thể xảy ra hay không. Nếu deadlock được phát hiện hệ thống phải phục hồi bằng cách kết thúc một số quá trình bị deadlock hay đòi lại tài nguyên từ một số quá trình bị deadlock.

Trong một hệ thống mà nó chọn các nạn nhân để phụv hồi về trạng thái trước đó chủ yếu dựa trên cơ sở yếu tố chi phí, việc đói tài nguyên có thể xảy ra. Kết quả là quá trình được chọn không bao giờ hoàn thành tác vụ được chỉ định của nó.

Thuviendientu.org

QUẢN LÝ BỘ NHỚ

I Mục đích

Sau khi học xong chương này, người học nắm được những kiến thức sau:

- Hiểu các cách khác nhau để quản lý bộ nhớ
- Hiểu tiếp cận quản lý bộ phân trang và phân đoạn
- Vận dụng một tiếp cận quản lý bộ nhớ phù hợp với hệ thống xác định

II Giới thiệu

Trong chương này chúng ta sẽ thảo luận nhiều cách khác nhau để quản lý bộ nhớ. Các giải thuật quản lý bộ nhớ từ tiếp cận máy trơ cơ bản (primitive bare-machine) là chiến lược phân trang và phân đoạn. Mỗi tiếp cận có lợi điểm và nhược của chính nó. Chọn phương pháp quản lý bộ nhớ cho một hệ thống xác định phụ thuộc vào nhiều yếu tố, đặc biệt trên thiết kế phần cứng của hệ thống. Chúng ta sẽ thấy nhiều giải thuật yêu cầu hỗ trợ phần cứng mặc dù các thiết kế gần đây đã tích hợp phần cứng và hệ điều hành.

III Đặt vấn đề

Bộ nhớ là trung tâm để điều hành hệ thống máy tính hiện đại. Bộ nhớ chứa một mảng lớn các từ (words) hay các bytes, mỗi phần tử với địa chỉ của chính nó. CPU lấy các chỉ thị từ bộ nhớ dựa theo giá trị của thanh đếm chương trình. Các chỉ thị này có thể gây việc nạp bổ sung các từ và lưu trữ tới các địa chỉ bộ nhớ xác định.

III.1 Liên kết địa chỉ

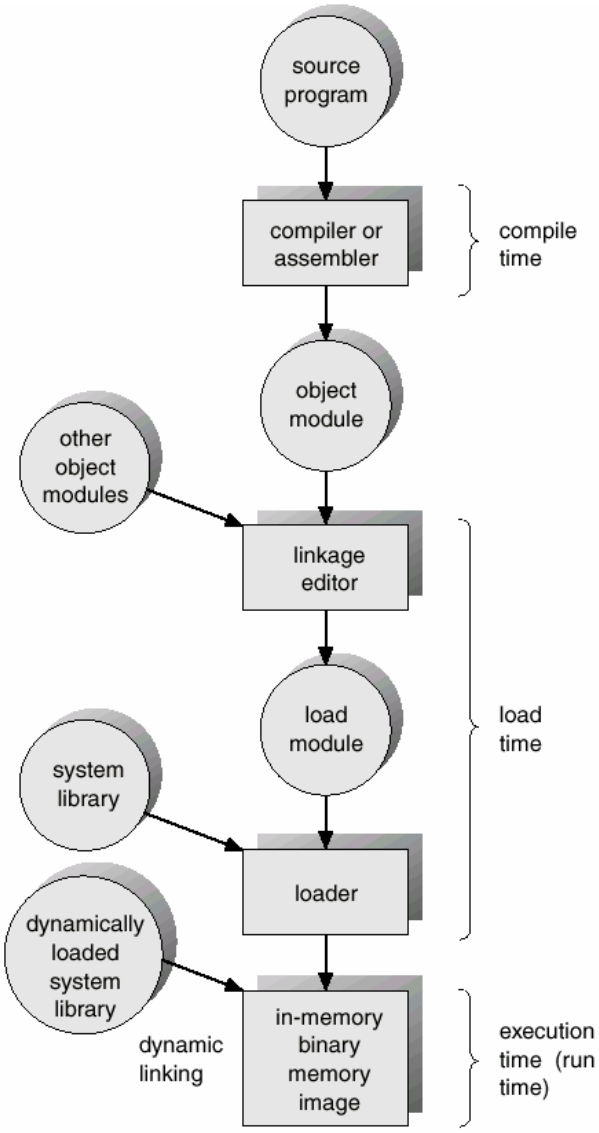
Thông thường, một chương trình nằm trên đĩa như một tập tin có thể thực thi dạng nhị phân. Chương trình này được mang vào trong bộ nhớ và được đặt trong một quá trình để nó được thực thi. Phụ thuộc vào việc quản lý bộ nhớ đang dùng, quá trình có thể được di chuyển giữa đĩa và bộ nhớ trong khi thực thi. Tập hợp các quá trình trên đĩa đang chờ được mang vào bộ nhớ để thực thi hình thành một **hàng đợi nhập** (input queue).

Thủ tục thông thường là chọn một trong những quá trình trong hàng đợi nhập và nạp quá trình đó vào trong bộ nhớ. Khi một quá trình được thực thi, nó truy xuất các chỉ thị và dữ liệu từ bộ nhớ. Cuối cùng, một quá trình kết thúc và không gian bộ nhớ của nó được xác định là trống.

Hầu hết các hệ thống cho phép một quá trình người dùng nằm ở bất cứ phần nào của bộ nhớ vật lý. Do đó, mặc dù không gian địa chỉ của máy tính bắt đầu tại 00000, nhưng địa chỉ đầu tiên của quá trình người dùng không cần tại 00000. Sắp xếp này ảnh hưởng đến địa chỉ mà chương trình người dùng có thể dùng. Trong hầu hết các trường hợp, một chương trình người dùng sẽ đi qua một số bước- một vài trong chúng có thể là tuỳ chọn-trước khi được thực thi (hình VII-1). Các địa chỉ có thể được hiện diện trong những cách khác trong những bước này. Các địa chỉ trong chương trình nguồn thường là những danh biểu. Một trình biên dịch sẽ liên kết các địa chỉ danh biểu tới các địa chỉ có thể tái định vị (chẳng hạn như 14 bytes từ vị trí bắt đầu của

Thuviendientu.org

module này). Bộ soạn thảo liên kết hay bộ nạp sẽ liên kết các địa chỉ có thể tái định vị tới địa chỉ tuyệt đối (chẳng hạn 74014). Mỗi liên kết là một ánh xạ từ một không gian địa chỉ này tới một không gian địa chỉ khác



Hình 0-1 Xử lý nhiều bước của chương trình người dùng

Về truyền thống, liên kết các chỉ thị và dữ liệu tới các địa chỉ có thể được thực hiện tại bất cứ bước nào theo cách sau đây:

- **Thời gian biên dịch**: nếu tại thời điểm biên dịch có thể biết quá trình nằm ở đâu trong bộ nhớ thì mã tuyệt đối có thể được phát sinh. Thí dụ, nếu biết trước quá trình người dùng nằm tại vị trí R thì mã trình biên dịch được phát sinh sẽ bắt đầu tại vị trí đó và mở rộng từ đó. Nếu tại thời điểm sau đó, vị trí bắt đầu thay đổi thì sẽ cần biên dịch lại mã này. Các chương trình định dạng .COM của MS-DOS là mã tuyệt đối giới hạn tại thời điểm biên dịch.
- **Thời điểm nạp**: nếu tại thời điểm biên dịch chưa biết nơi quá trình sẽ nằm ở đâu trong bộ nhớ thì trình biên dịch phải phát sinh mã có thể tái định vị. Trong trường hợp này, liên kết cuối cùng được trì hoãn cho tới thời điểm

Thuviendientu.org

nạp. Nếu địa chỉ bắt đầu thay đổi, chúng ta chỉ cần nạp lại mã người dùng để hợp nhất giá trị được thay đổi này.

- **Thời gian thực thi**: nếu quá trình có thể được di chuyển trong thời gian thực thi từ một phân đoạn bộ nhớ này tới một phân đoạn bộ nhớ khác thì việc liên kết phải bị trì hoãn cho tới thời gian chạy. Phần cứng đặc biệt phải sẵn dùng cho cơ chế này để thực hiện công việc. Hầu hết những hệ điều hành này dùng phương pháp này.

Phần chủ yếu của chương này được dành hết để hiển thị các liên kết khác nhau có thể được cài đặt hữu hiệu trong một hệ thống máy tính và thảo luận sự hỗ trợ phần cứng tương ứng.

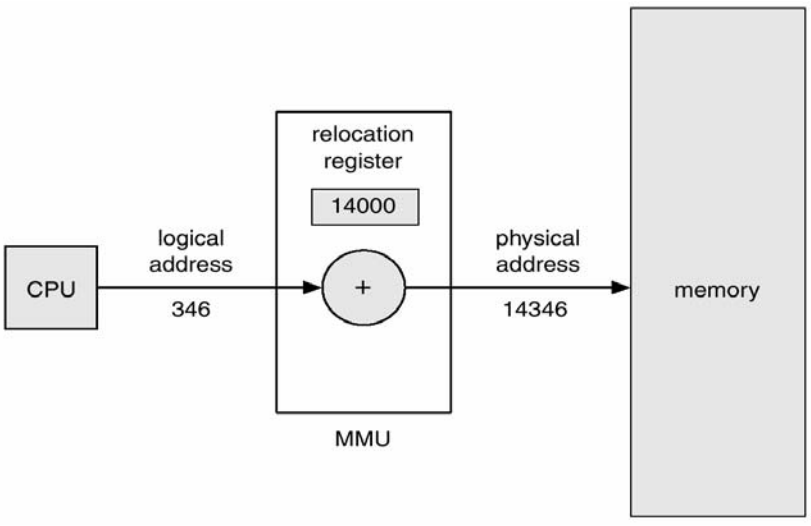
III.2 Không gian địa chỉ luận lý và không gian địa chỉ vật lý

Một địa chỉ được tạo ra bởi CPU thường được gọi là **địa chỉ luận lý** (logical address), ngược lại một địa chỉ được xem bởi đơn vị bộ nhớ-nghĩa là, một địa chỉ được nạp vào thanh ghi địa chỉ bộ nhớ-thường được gọi là **địa chỉ vật lý** (physical address).

Các phương pháp liên kết địa chỉ thời điểm biên dịch và thời điểm nạp tạo ra địa chỉ luận lý và địa chỉ vật lý xác định. Tuy nhiên, cơ chế liên kết địa chỉ tại thời điểm thực thi dẫn đến sự khác nhau giữa địa chỉ luận lý và địa chỉ vật lý. Trong trường hợp này, chúng ta thường gọi địa chỉ luận lý như là **địa chỉ ảo** (virtual address). Tập hợp tất cả địa chỉ luận lý được tạo ra bởi chương trình là **không gian địa chỉ luận lý** ; tập hợp tất cả địa chỉ vật lý tương ứng địa chỉ luận lý này là **không gian địa chỉ vật lý**. Do đó, trong cơ chế liên kết địa chỉ tại thời điểm thực thi, không gian địa chỉ luận lý và không gian địa chỉ vật lý là khác nhau.

Việc ánh xạ tại thời điểm thực thi từ địa chỉ ảo tới địa chỉ vật lý được thực hiện bởi một thiết bị phần cứng được gọi là **bộ quản lý bộ nhớ MMU** (memory-management unit). Chúng ta có thể chọn giữa những phương pháp khác nhau để thực hiện việc ánh xạ.

Như được hiển thị trong hình VII-2 ở trên, phương pháp này yêu cầu sự hỗ trợ phần cứng. Thanh ghi nền bây giờ được gọi là **thanh ghi tái định vị**. Giá trị trong thanh ghi tái định vị được cộng vào mỗi địa chỉ được tạo ra bởi quá trình người dùng tại thời điểm nó được gởi tới bộ nhớ. Thí dụ, nếu giá trị nền là 14000, thì việc cố gắng bởi người dùng để xác định vị trí 0 được tự động tái định vị tới vị trí 14000; một truy xuất tới địa chỉ 346 được ánh xạ tới vị trí 14346.



Hình 0-2 định vị tự động dùng thanh ghi tái định vị

Thuviendientu.org

III.3 Nạp động

Trong thảo luận của chúng ta gần đây, toàn bộ chương trình và dữ liệu của một quá trình phải ở trong bộ nhớ vật lý để quá trình thực thi. Kích thước của quá trình bị giới hạn bởi kích thước của bộ nhớ vật lý. Để đạt được việc sử dụng không gian bộ nhớ tốt hơn, chúng ta có thể sử dụng **nạp động** (dynamic loading). Với nạp động, một thủ tục không được nạp cho tới khi nó được gọi. Tất cả thủ tục được giữ trên đĩa trong định dạng nạp có thể tái định vị. Chương trình chính được nạp vào bộ nhớ và được thực thi. Khi một thủ tục cần gọi một thủ tục khác, thủ tục gọi trước hết kiểm tra để thấy thủ tục khác được nạp hay không. Nếu không, bộ nạp liên kết có thể tái định vị được gọi để nạp thủ tục mong muốn vào bộ nhớ và cập nhật các bảng địa chỉ của chương trình để phản ánh thay đổi này. Sau đó, điều khiển này được truyền tới thủ tục mới được nạp.

Thuận lợi của nạp động là ở đó một thủ tục không được dùng thì không bao giờ được nạp. Phương pháp này đặc biệt có ích khi lượng lớn mã được yêu cầu quản lý các trường hợp xảy ra không thường xuyên, chẳng hạn như các thủ tục lỗi. Trong trường hợp này, mặc dù kích thước toàn bộ chương trình có thể lớn, nhưng phần được dùng (và do đó được nạp) có thể nhỏ hơn nhiều.

Nạp động không yêu cầu hỗ trợ đặc biệt từ hệ điều hành. Nhiệm vụ của người dùng là thiết kế các chương trình của họ để đạt được sự thuận lợi đó. Tuy nhiên, hệ điều hành có thể giúp người lập trình bằng cách cung cấp các thủ tục thư viện để cài đặt nạp tự động.

III.4 Liên kết động và các thư viện được chia sẻ

Trong hình VII-1 cũng hiển thị thư viện được liên kết động. Một số hệ điều hành hỗ trợ chỉ liên kết tĩnh mà trong đó các thư viện ngôn ngữ hệ thống được đối xử như bất kỳ module đối tượng khác và được kết hợp bởi bộ nạp thành hình ảnh chương trình nhị phân. Khái niệm liên kết động là tương tự như khái niệm nạp động. Liên kết bị trì hoãn hơn là việc nạp bị trì hoãn cho tới thời điểm thực thi. Đặc điểm này thường được dùng với các thư viện hệ thống như các thư viện chương trình con của các ngôn ngữ. Không có tiện ích này, tất cả chương trình trên một hệ thống cần có một bản sao thư viện của ngôn ngữ của chúng (hay ít nhất thư viện được tham chiếu bởi chương trình) được chứa trong hình ảnh có thể thực thi. Yêu cầu này làm lãng phí cả không gian đĩa và bộ nhớ chính. Với liên kết động, một **stub** là một đoạn mã hiển thị cách định vị chương trình con trong thư viện cư trú trong bộ nhớ hay cách nạp thư viện nếu chương trình con chưa hiện diện.

Khi stub này được thực thi, nó kiểm tra để thấy chương trình con được yêu cầu đã ở trong bộ nhớ hay chưa. Nếu chưa, chương trình này sẽ nạp chương trình con vào trong bộ nhớ. Dù là cách nào, stub thay thế chính nó với địa chỉ của chương trình con và thực thi chương trình con đó. Do đó, thời điểm tiếp theo phân đoạn mã đạt được, chương trình con trong thư viện được thực thi trực tiếp mà không gây ra bất kỳ chi phí cho việc liên kết động. Dưới cơ chế này, tất cả các quá trình sử dụng một thư viện ngôn ngữ thực thi chỉ một bản sao của mã thư viện.

Thuviendientu.org

III.5 Phủ lắp

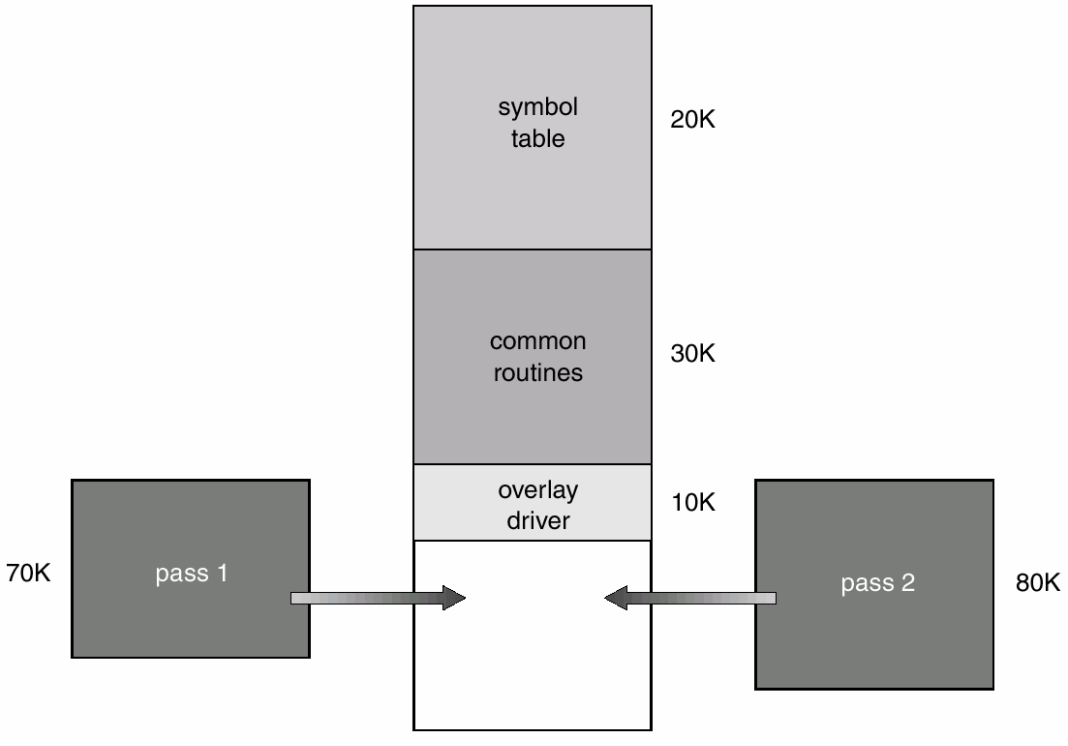
Để cho phép một quá trình lớn hơn lượng bộ nhớ được cấp phát cho nó, chúng ta sử dụng **cơ chế phủ lắp** (overlays). Ý tưởng phủ lắp là giữ trong bộ nhớ những chỉ thị và dữ liệu được yêu cầu tại bất kỳ thời điểm nào được cho. Khi những chỉ thị đó được yêu cầu, chúng được nạp vào không gian được chiếm trước đó bởi các chỉ thị mà chúng không còn cần nữa.

Thí dụ, xét trình dịch hợp ngữ hai lần (two-pass assembler). Trong suốt lần thứ 1, nó xây dựng bảng danh biểu; sau đó, trong lần thứ 2, nó tạo ra mã máy. Chúng ta có thể phân chia trình dịch hợp ngữ thành mã lần 1, mã lần 2, bảng danh biểu, và những thủ tục hỗ trợ chung được dùng bởi lần 1 và lần 2. Giả sử kích thước của các thành phần này như sau:

Lần 1	70 KB
Lần 2	80 KB
Bảng danh biểu	20 KB
Các thủ tục chung	30 KB

Để nạp mọi thứ một lần, chúng ta cần 200KB bộ nhớ. Nếu chỉ có 150KB sẵn có, chúng ta không thể chạy quá trình của chúng ta. Tuy nhiên, chú ý rằng lần 1 và lần 2 không cần ở trong bộ nhớ cùng một lúc. Do đó, chúng ta định nghĩa hai phủ lắp. Phủ lắp A là bảng danh biểu, các thủ tục chung, lần 1, và phủ lắp B là bảng biểu tượng, các thủ tục chung và lần 2.

Chúng ta bổ sung trình điều khiển phủ lắp (10 KB) và bắt đầu với phủ lắp A trong bộ nhớ. Khi chúng ta kết thúc lần 1, chúng ta nhảy tới trình điều khiển phủ lắp, trình điều khiển này sẽ đọc phủ lắp B vào trong bộ nhớ, viết chồng lên phủ lắp B và sau đó chuyển điều khiển tới lần 2. Phủ lắp A chỉ cần 120KB, ngược lại phủ lắp B cần 130KB (hình VII-3). Bây giờ chúng ta có thể chạy trình hợp ngữ trong 150KB bộ nhớ. Nó sẽ nạp nhanh hơn vì rất ít dữ liệu cần được chuyển trước khi việc thực thi bắt đầu. Tuy nhiên, nó sẽ chạy chậm hơn do nhập/xuất phụ đọc mã mã cho phủ lắp A qua mã cho phủ lắp B.



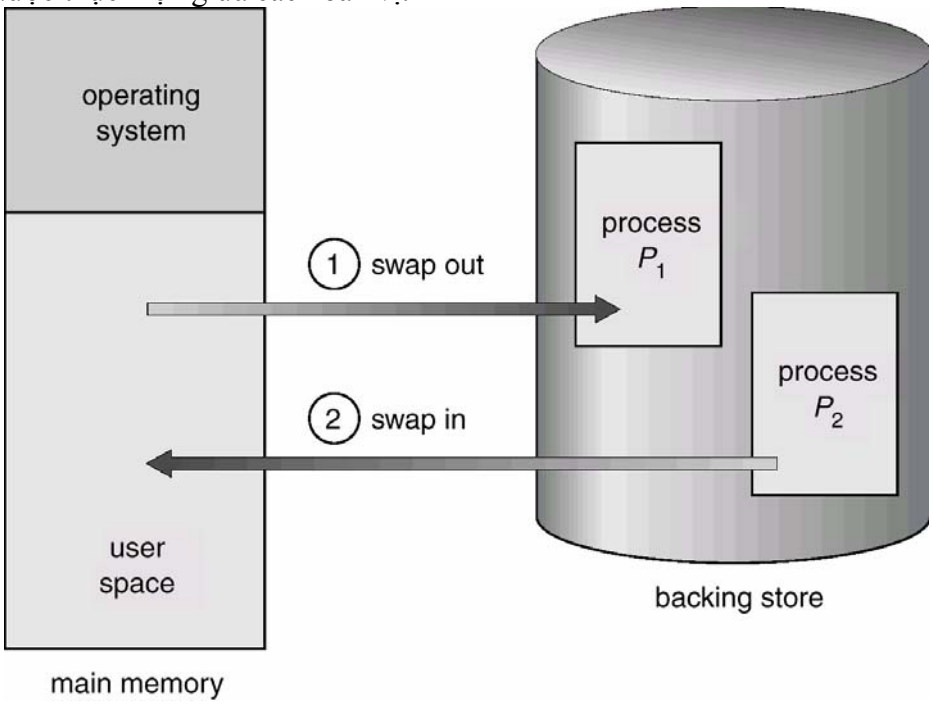
Hình 0-3- Các phủ lắp cho một bộ hợp ngữ dịch hai lần

Thuviendientu.org

Mã cho phủ lắp A và mã cho phủ lắp B được giữ trên đĩa như những hình ảnh bộ nhớ tuyệt đối, và được đọc bởi trình điều khiển phủ lắp khi cần. Tái định vị đặc biệt và các giải thuật liên kết được yêu cầu xây dựng các phủ lắp.

IV Hoán vị

Một quá trình cần ở trong bộ nhớ để được thực thi. Tuy nhiên, một quá trình có thể được hoán vị (swapped) tạm thời khỏi bộ nhớ tới vùng lưu trữ phụ ***backing store***, sau đó mang trở lại bộ nhớ để việc thực thi được tiếp tục. Thí dụ, giả sử một môi trường đa chương với giải thuật lập thời biểu CPU round-robin. Khi định mức thời gian hết, bộ quản lý bộ nhớ sẽ bắt đầu **hoán vị ra** (swap out) vùng lưu trữ phụ quá trình vừa mới kết thúc và **hoán vị vào** (swap in) một quá trình khác tới không gian bộ nhớ được trống (hình VII-4). Do đó, bộ định thời biểu CPU sẽ cấp những phần thời gian tới những quá trình khác trong bộ nhớ. Lý tưởng, bộ quản lý sẽ hoán vị các quá trình đủ nhanh để một vài quá trình sẽ ở trong bộ nhớ, sẵn sàng thực thi, khi bộ định thời CPU muốn định thời lại CPU. Định mức cũng phải đủ lớn để phù hợp lượng tính toán được thực hiện giữa các hoán vị.



Hình 0-4- Hoán vị hai quá trình dùng đĩa như là backing store

Một biến thể của chính sách hoán vị này được dùng cho các giải thuật định thời trên cơ sở ưu tiên. Nếu một quá trình có độ ưu tiên cao hơn đến và muốn phụ vụ, bộ quản lý bộ nhớ có thể hoán vị ra quá trình có độ ưu tiên thấp hơn để mà nó có thể nạp và thực thi quá trình có độ ưu tiên cao hơn. Khi quá trình có độ ưu tiên cao hơn kết thúc, quá trình có độ ưu tiên thấp hơn có thể được hoán vị vào trở lại và được tiếp tục. Biến thể của hoán vị này thường được gọi là **cuộn ra** (roll out), và **cuộn vào** (roll in).

Thông thường, một quá trình được hoán vị ra sẽ được hoán vị trở lại vào cùng không gian bộ nhớ mà nó đã chiếm trước đó. Sự giới hạn này được sai khiến bởi phương pháp liên kết địa chỉ. Nếu liên kết địa chỉ được thực hiện tại thời điểm hợp dịch hay nạp thì quá trình không thể được di chuyển vào không gian bộ nhớ khác vì các địa chỉ vật lý được tính trong thời gian thực thi.

Thuviendientu.org

Hoán vị yêu cầu một **vùng lưu trữ phụ** (backing store). Vùng lưu trữ phụ này thường là một đĩa tốc độ cao. Nó phải đủ lớn để chứa các bản sao của tất cả hình ảnh bộ nhớ cho tất cả người dùng, và nó phải cung cấp truy xuất trực tiếp tới các hình ảnh bộ nhớ này. Hệ thống này duy trì một **hàng đợi sẵn sàng** chứa tất cả quá trình mà các hình ảnh bộ nhớ của nó ở trong vùng lưu trữ phụ hay trong bộ nhớ và sẵn sàng để thực thi. Bất cứ khi nào bộ định thời CPU quyết định thực thi một quá trình, nó gọi **bộ phân phát** (dispacher). Bộ phân phát kiểm tra để thấy quá trình tiếp theo trong hàng đợi ở trong bộ nhớ không. Nếu không, và không có vùng bộ nhớ trống, bộ phân phát hoán vị ra một quá trình hiện hành trong bộ nhớ và hoán vị vào một quá trình mong muốn. Sau đó, nó nạp lại các thanh ghi và chuyển điều khiển tới quá trình được chọn. Trong các hệ hoán vị, thời gian chuyển đổi giữa các tác vụ cần được quan tâm. Mỗi quá trình cần được phân chia một khoảng thời gian sử dụng CPU đủ lớn để không thấy rõ sự chậm trễ do các thao tác hoán vị gây ra. Nếu không, hệ thống sẽ dùng phần lớn thời gian để hoán vị các quá trình vào ra bộ nhớ chính, CPU như vậy sẽ không sử dụng hiệu quả.

Hoán vị cũng bị ràng buộc bởi yếu tố khác. Nếu chúng ta muốn hoán vị một quá trình, chúng ta phải đảm bảo rằng nó hoàn toàn rỗi. Quan tâm đặc biệt là việc chờ nhập/xuất. Một quá trình có thể đang chờ thao tác nhập/xuất khi chúng ta hoán vị quá trình đó tới nơi trống bộ nhớ của nó. Tuy nhiên, nếu nhập/xuất đang truy xuất không đồng bộ bộ nhớ người dùng cho nhập/xuất vùng đệm, thì quá trình không thể được hoán vị. Giả sử rằng thao tác nhập/xuất đang xếp hàng chờ vì thiết bị đang bận. Sau đó, nếu chúng ta hoán vị quá trình P_1 ra và hoán vị P_2 vào thì thao tác nhập/xuất có thể cố gắng sử dụng bộ nhớ hiện thuộc về quá trình P_2. Hai giải pháp chủ yếu cho quá trình này là không bao giờ hoán vị quá trình đang chờ nhập/xuất hay thực thi các thao tác nhập/xuất chỉ ở trong vùng đệm của hệ điều hành. Chuyển giữa các vùng đệm của hệ điều hành và bộ nhớ quá trình thì chỉ xảy ra khi quá trình được hoán vị vào.

V Cấp phát bộ nhớ liên tục

Bộ nhớ chính phải cung cấp cho cả hệ điều hành và các quá trình người dùng khác nhau. Do đó, chúng ta cần cấp phát những phần khác nhau của bộ nhớ chính trong những cách hiệu quả nhất có thể. Phần này chúng ta giải thích một phương pháp thông dụng, cấp phát bộ nhớ liên tục.

Bộ nhớ thường được phân chia thành hai phân khu, một cho hệ điều hành định vị và một cho các quá trình người dùng. Chúng ta có thể đặt hệ điều hành ở bộ nhớ cao hay bộ nhớ thấp. Yếu tố quan trọng ảnh hưởng tới quyết định này là vị trí của vector ngắt. Vì vector ngắt thường ở trong bộ nhớ thấp nên các lập trình viên thường cũng đặt hệ điều hành trong bộ nhớ thấp. Do đó, trong giáo trình này chúng ta sẽ thảo luận chỉ trường hợp hệ điều hành định vị trong bộ nhớ thấp. Phát triển của trường hợp khác là tương tự.

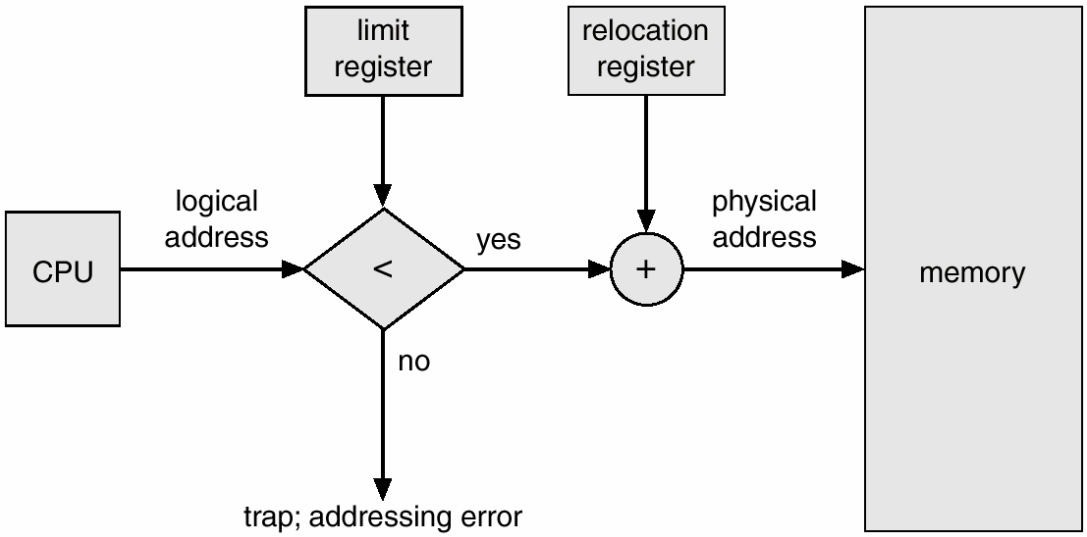
Chúng ta thường muốn nhiều quá trình người dùng định vị trong bộ nhớ tại cùng thời điểm. Do đó, chúng ta cần xem xét cách cấp phát bộ nhớ trống tới những quá trình ở trong hàng đợi nhập đang chờ được mang vào bộ nhớ. Trong cấp phát bộ nhớ liên tục, mỗi quá trình được chứa trong một phần bộ nhớ liên tục.

V.1 Bảo vệ bộ nhớ

Trước khi thảo luận cấp phát bộ nhớ chúng ta phải thảo luận vấn đề bảo vệ bộ nhớ-bảo vệ hệ điều hành từ quá trình người dùng, và bảo vệ các quá trình từ một quá trình khác. Chúng ta có thể cung cấp bảo vệ này bằng cách dùng thanh ghi tái định vị.

Thuviendientu.org

Thanh ghi tái định vị chứa giá trị địa chỉ vật lý nhỏ nhất; thanh ghi giới hạn chứa dãy các định chỉ luận lý (thí dụ: tái định vị = 100040 và giới hạn = 74600). Với các thanh ghi tái định vị và giới hạn, mỗi địa chỉ luận lý phải ít hơn thanh ghi giới hạn; MMU ánh xạ địa chỉ luận lý động bằng cách cộng giá trị trong thanh ghi tái định vị. Địa chỉ được tái định vị này được gửi tới bộ nhớ (như hình VII-5).



Hình 0-5 Hỗ trợ phần cứng cho các thanh ghi tái định vị và các giới hạn

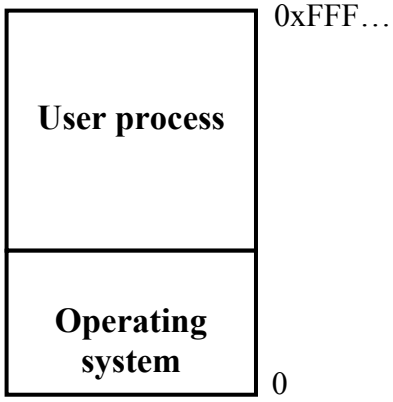
Khi bộ định thời CPU chọn một quá trình thực thi, bộ phân phát nạp thanh ghi tái định vị và giới hạn với các giá trị đúng như một phần của chuyển đổi ngữ cảnh. Vì mọi địa chỉ được phát sinh bởi CPU được kiểm tra dựa trên các thanh ghi này, chúng ta có thể bảo vệ hệ điều hành và các chương trình và dữ liệu người dùng khác từ việc sửa đổi bởi quá trình đang chạy này.

Cơ chế dùng thanh ghi tái định vị cung cấp một cách hiệu quả để cho phép kích thước hệ điều hành thay đổi động. Khả năng mềm dẽo này có thể mong muốn trong nhiều trường hợp. Thí dụ, hệ điều hành chứa mã và không gian vùng đệm cho trình điều khiển thiết bị. Nếu một trình điều khiển thiết bị (hay dịch vụ hệ điều hành khác) không được dùng phổ biến, nó không muốn giữ mã và dữ liệu trong bộ nhớ, khi chúng ta có thể dùng không gian đó cho mục đích khác. Những mã như thế thường được gọi là mã hệ điều hành tạm thời (transient operating system code); nó đến và đi khi được yêu cầu. Do đó, dùng mã này thay đổi kích thước của hệ điều hành trong khi thực thi chương trình.

Thuviendientu.org

V.2 Hệ thống đơn chương

Trong phương pháp này bộ nhớ được chia sẻ cho hệ điều hành và một chương trình duy nhất của người sử dụng. Tại một thời điểm, một phần của bộ nhớ sẽ do hệ điều hành chiếm giữ, phần còn lại thuộc về quá trình người dùng duy nhất trong hệ thống (Hình VII-6). Quá trình này được toàn quyền sử dụng bộ nhớ dành cho nó.



Hình 0-6 Tổ chức bộ nhớ trong hệ thống đơn chương

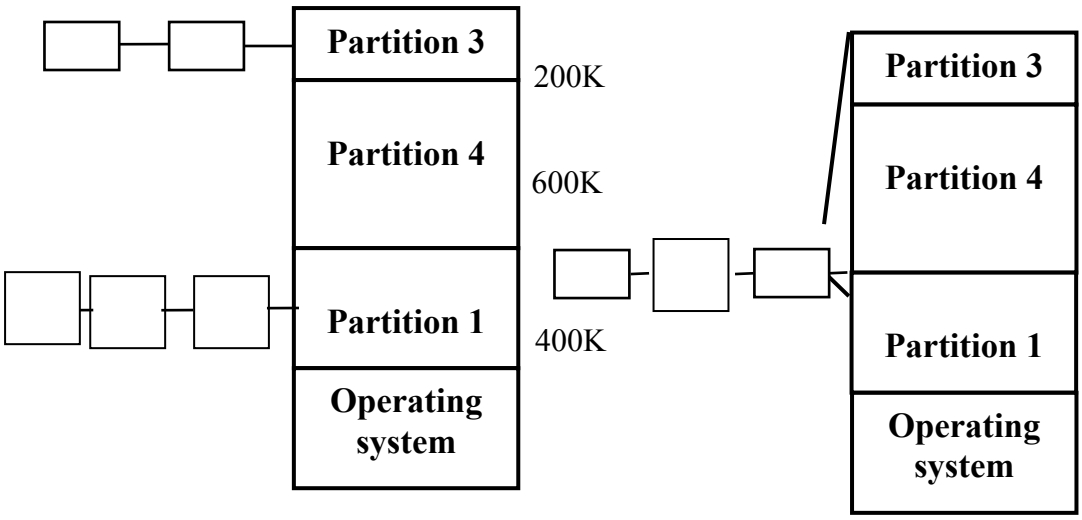
Khi bộ nhớ được tổ chức theo cách thức này, chỉ có thể xử lý một chương trình tại một thời điểm. Quan sát hoạt động của các quá trình, có thể nhận thấy rất nhiều tiến trình trải qua phần lớn thời gian để chờ các thao tác nhập/xuất hoàn thành. Trong suốt thời gian này, CPU ở trạng thái rỗi. Trong trường hợp như thế, hệ thống đơn chương không cho phép sử dụng hiệu quả CPU. Ngoài ra, sự đơn chương không cho phép nhiều người sử dụng làm việc đồng thời theo cơ chế tương tác. Để nâng cao hiệu suất sử dụng CPU, cần cho phép chế độ đa chương mà trong đó các quá trình chia sẻ CPU với nhau để hoạt động đồng hành.

V.3 Hệ thống đa chương với phân khu cố định

Một trong những phương pháp đơn giản nhất để cấp phát bộ nhớ là chia bộ nhớ thành những phân khu có kích thước cố định. Mỗi phân khu có thể chứa chính xác một quá trình. Do đó, cấp độ đa chương được giới hạn bởi số lượng phân khu. Trong phương pháp đa phân khu, khi một phân khu rảnh, một quá trình được chọn từ hàng đợi nhập và được nạp vào phân khu trống. Khi quá trình kết thúc, phân khu trở nên sẵn dùng cho một quá trình khác. Có hai tiếp cận để tổ chức hàng đợi:

- **Sử dụng nhiều hàng đợi**: mỗi phân khu sẽ có một hàng đợi tương ứng (hình VII-7a). Khi một quá trình mới được tạo ra, nó được đưa vào hàng đợi của phân khu có kích thước nhỏ nhất thoả nhu cầu chứa nó. Cách tổ chức này có khuyết điểm trong trường hợp các hàng đợi của một số phân khu trống trong khi các hàng đợi của các phân khu khác lại đầy, buộc các quá trình trong những hàng đợi này phải chờ được cấp phát bộ nhớ.
- **Sử dụng một hàng đợi**: tất cả các quá trình được đặt trong hàng đợi duy nhất (hình VII-7b). Khi có một phân khu trống, quá trình đầu tiên trong hàng đợi có kích thước phù hợp sẽ được đặt vào phân khu và cho xử lý.

Thuviendientu.org



a. Sử dụng nhiều hàng đợi b. Sử dụng một hàng đợi

Hình 0-7 Cấp phát phân khu có kích thước cố định

Khi sử dụng giải thuật này người ta muốn tránh sự hao phí một phân khu lớn cho một công việc nhỏ, nhưng lại xảy ra bất bình đẳng, bất lợi đối với các công việc nhỏ. Để giải quyết người ta thêm vào qui luật là một công việc sẽ không bị bỏ qua nữa nếu nó đã bị bỏ qua **k** lần qui định. Mỗi lần một công việc bị bỏ qua nó được đánh dấu một điểm. Khi đạt được số điểm qui định, nó sẽ không bị bỏ qua nữa, sẽ được nạp vào và thực hiện mặc dầu có thể trên một phân khu lớn hơn.

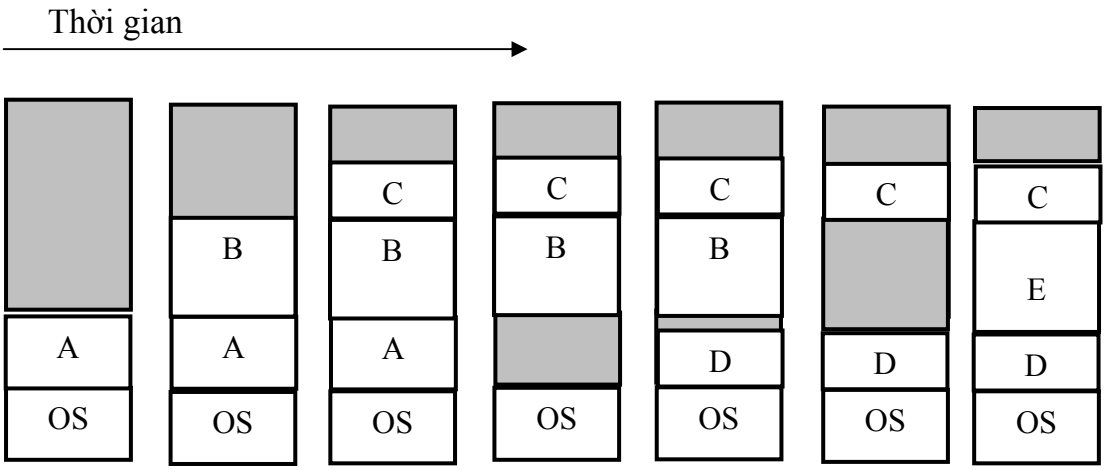
Phương pháp này ban đầu được sử dụng bởi hệ điều hành IBM OS/360, nó được gọi là MFT (Multiprogramming with Fixed number of Tasks). Hiện nay nó không còn sử dụng nữa.

V.4 Hệ thống đa chương với phân khu động

Cơ chế này là tổng quát của cơ chế phân khu cố định. Nó được dùng chủ yếu trong môi trường xử lý theo lô. Nhiều ý tưởng được trình bày ở đây cũng có thể áp dụng tới môi trường chia thời mà trong đó phân đoạn thuần được dùng cho việc quản lý bộ nhớ.

Hệ điều hành giữ một bảng hiển thị những phần nào của bộ nhớ là sẵn dùng và phần nào đang bận. Ban đầu, tất cả bộ nhớ là sẵn dùng cho quá trình người dùng, và được xem như một khối lớn bộ nhớ sẵn dùng hay một lỗ. Khi một quá trình đến và cần bộ nhớ, chúng ta tìm kiếm một lỗ trống đủ lớn cho quá trình này. Nếu chúng ta tìm thấy, chúng ta cấp phát chỉ phần bộ nhớ nhiều bằng lượng được yêu cầu, phần còn lại sẵn dùng để thoả mãn những yêu cầu tương lai (Hình VII-8).

Thuviendientu.org



Hình VIII-8 Cấp phát các phân khu có kích thước thay đổi

Khi các quá trình đi vào hệ thống, chúng được đặt vào hàng đợi nhập. Hệ điều hành xem xét yêu cầu bộ nhớ của mỗi quá trình và lượng không gian bộ nhớ sẵn có để xác định các quá trình nào được cấp phát bộ nhớ. Khi một quá trình được cấp không gian, nó được nạp vào bộ nhớ và sau đó nó có thể cạnh tranh CPU. Khi một quá trình kết thúc, nó giải phóng bộ nhớ của nó, sau đó hệ điều hành có thể đặt một quá trình khác từ hàng đợi nhập.

Tại bất cứ thời điểm được cho, chúng ta có một danh sách kích thước khối trống và hàng đợi nhập. Hệ điều hành có thể xếp hàng đợi nhập dựa theo giải thuật định thời. Bộ nhớ được cấp phát tới quá trình cho đến khi các yêu cầu bộ nhớ của quá trình kế tiếp không thể được thoả; không có khối bộ nhớ trống (hay lỗ) đủ lớn để quản lý quá trình đó. Sau đó, hệ điều hành có thể chờ cho đến khi khối đủ lớn sẵn dùng hay nó có thể di chuyển xuống hàng đợi nhập để xem các yêu cầu bộ nhớ nhỏ hơn của các quá trình khác có thể được thoả hay không.

Thông thường, một tập hợp các lỗ có kích thước khác nhau được phân tán khắp bộ nhớ tại bất cứ thời điểm được cho. Khi một quá trình đến và yêu cầu bộ nhớ, hệ thống tìm tập hợp này một lỗ trống đủ lớn cho quá trình này. Nếu lỗ trống quá lớn, nó được chia làm hai: một phần được cấp tới quá trình đến; phần còn lại được trả về tập hợp các lỗ. Nếu lỗ mới nằm kề với các lỗ khác, các lỗ nằm kề này được gom lại để tạo thành một lỗ lớn hơn. Tại thời điểm này, hệ thống cần kiểm tra có quá trình nào đang chờ bộ nhớ và bộ nhớ trống mới hay bộ nhớ vừa được kết hợp lại có thể thoả yêu cầu của bất cứ quá trình đang chờ này không.

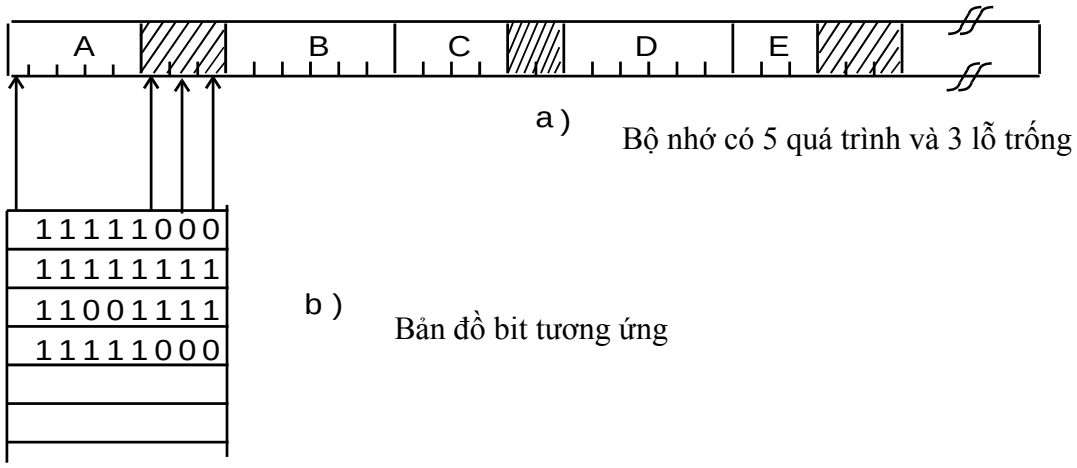
Thủ tục này là một trường hợp đặc biệt của **vấn đề cấp phát lưu trữ động** là làm cách nào để thoả mãn một yêu cầu có kích thước n từ danh sách lỗ trống. Có hai giải pháp chủ yếu cho vấn đề này.

1) **Quản lý bằng bản đồ bit**: bộ nhớ được chia thành các đơn vị cấp phát, mỗi đơn vị được ánh xạ tới một bit trong bản đồ bit (Hình VII-9). Giá trị bit này xác định trạng thái của đơn vị bộ nhớ đó: 0 đang tự do, 1 đã được cấp phát. Khi cần nạp một quá trình có kích thước k đơn vị, hệ thống sẽ tìm trong bản đồ bit một dãy k bit có giá trị 0.

Kích thước của đơn vị cấp phát là vấn đề lớn trong thiết kế. Nếu kích thước đơn vị cấp phát nhỏ sẽ làm tăng kích thước của bản đồ bit. Ngược lạ, nếu kích thước đơn

Thuviendientu.org

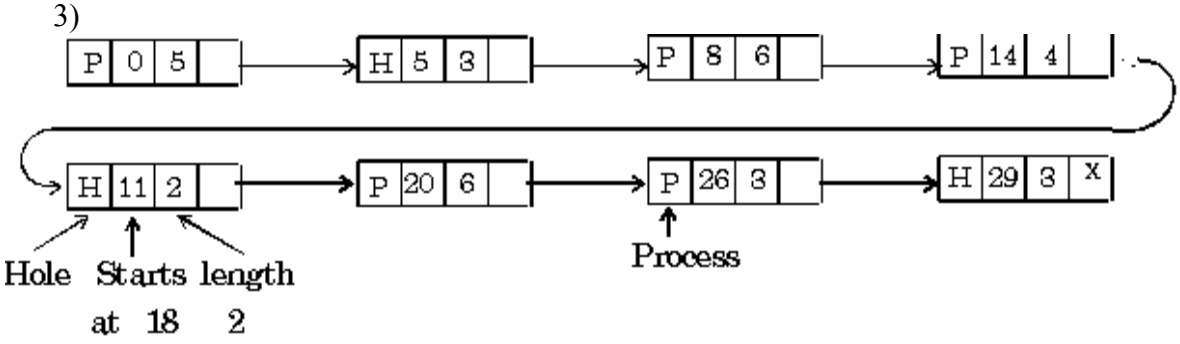
vị cấp phát lớn có thể gây hao phí cho đơn vị cấp phát sau cùng. Đây là giải pháp đơn giản nhưng thực hiện chậm nên ít được dùng.



Hình 0-9 Quản lý bộ nhớ bằng bản đồ bit

2) **Quản lý bằng danh sách liên kết**: dùng một danh sách liên kết để quản lý các phân đoạn bộ nhớ đã cấp phát và phân đoạn tự do, một phân đoạn có thể là một quá trình hay một vùng nhớ trống giữa hai quá trình. Danh sách liên kết gồm nhiều mục từ liên tiếp. Mỗi mục từ gồm 1 bit đầu để xác định phân đoạn đó là lỗ trống (H) hay một quá trình (P), sau đó là 3 từ để chỉ địa chỉ bắt đầu, chiều dài và chỉ điểm tới mục kế tiếp. Việc sắp xếp các phân đoạn theo địa chỉ hay theo kích thước tuỳ thuộc vào giải thuật quản lý bộ nhớ. Sơ đồ quản lý bằng danh sách liên kết tương ứng với sơ đồ quản lý bằng bản đồ bit được minh hoạ trong hình VII-10.

3)



Hình 0-10 Quản lý bộ nhớ bằng danh sách liên kết

Tập hợp các lỗ trống được tìm thấy để xác định lỗ nào là tốt nhất để cấp phát. Các chiến lược first-fit, best-fit, worst-fit là những chiến lược phổ biến nhất được dùng để chọn một lỗ trống từ tập hợp các lỗ trống.

- **First-fit**: cấp phát lỗ trống đầu tiên đủ lớn. Tìm kiếm có thể bắt đầu tại đầu tập hợp các lỗ trống hay tại điểm kết thúc của tìm kiếm first-fit trước đó. Chúng ta dừng tìm kiếm ngay khi chúng ta tìm thấy một lỗ trống đủ lớn.
- **Best-fit**: cấp phát lỗ trống nhỏ nhất đủ lớn. Chúng ta phải tìm toàn bộ danh sách, trừ khi danh sách được xếp thứ tự theo kích thước. Chiến lược này tạo ra lỗ trống nhỏ nhất còn thừa lại.

Thuviendientu.org

- **Worst-fit**: cấp phát lỗ trống lớn nhất. Chúng ta phải tìm toàn danh sách trừ khi nó được xếp theo thứ tự kích thước. Chiến lược này tạo ra lỗ trống còn lại lớn nhất mà có thể có ích hơn lỗ trống nhỏ từ tiếp cận best-fit.

Các mô phỏng hiển thị rằng cả first-fit và best-fit là tốt hơn worst-fit về việc giảm thời gian và sử dụng lưu trữ. Giữa first-fit và best-fit không thể xác định rõ chiến lược nào tốt hơn về sử dụng lưu trữ, nhưng first-fit có tốc độ nhanh hơn.

Tuy nhiên, các giải thuật này gặp phải vấn đề **phân mãnh ngoài** (external fragmentation). Khi các quá trình được nạp và được xoá khỏi bộ nhớ, không gian bộ nhớ trống bị phân rã thành những mãnh nhỏ. Phân mãnh ngoài tồn tại khi tổng không gian bộ nhớ đủ để thoả mãn một yêu cầu, nhưng nó không liên tục; vùng lưu trữ bị phân mãnh thành một số lượng lớn các lỗ nhỏ. Vấn đề phân mãnh này có thể rất lớn. Trong trường hợp xấu nhất, chúng có thể có một khối bộ nhớ trống nằm giữa mỗi hai quá trình. Nếu tất cả bộ nhớ này nằm trong một khối trống lớn, chúng ta có thể chạy nhiều quá trình hơn.

Chọn lựa first-fit so với best-fit có thể ảnh hưởng tới lượng phân mãnh. (First-fit là tốt hơn đối với một số hệ thống, ngược lại best fit là tốt hơn cho một số hệ thống khác). Một yếu tố khác là phần cuối của khối trống nào được cấp phát. (phần còn dư nào-phần trên đỉnh, hay phần ở dưới đáy?). Vấn đề không do giải thuật nào được dùng mà do phân mãnh ngoài.

V.5 Quản lý bộ nhớ với hệ thống bạn thân

Như ta đã thấy trong phần trước, việc quản lý các lỗ hổng trên những bảng liệt kê được sắp xếp theo kích thước làm cho việc cấp phát bộ nhớ rất nhanh, nhưng lại làm chậm cho việc ngưng cấp phát bởi vì ta phải chú ý đến các láng giềng. **Hệ thống bạn thân** (Buddy System) là một giải thuật quản lý bộ nhớ tận dụng thuận lợi của việc máy tính dùng những số nhị phân cho việc đánh địa chỉ để tăng tốc độ kết hợp những lỗ hổng sát nhau khi một quá trình hoàn thành hoặc được hoán vị ra ngoài.

Với phương pháp này, bộ quản lý bộ nhớ sẽ có một bảng liệt kê những khối còn tự do có kích thước 1, 2, 4, 16...bytes đến kích thước của bộ nhớ, tức là có kích thước bằng lũy thừa của 2. Khi có một quá trình cần cấp phát bộ nhớ, một lỗ hổng có kích thước bằng luỹ thừa của 2 đủ chứa quá trình sẽ được cấp phát. Nếu không có lỗ hổng yêu cầu, các lỗ hổng sẽ được phân đôi cho đến khi có được lỗ hỗng cần thiết. Khi một quá trình chấm dứt, các lỗ hổng kế nhau có kích thước bằng nhau sẽ được nhập lại để tạo thành lỗ hổng lớn hơn. Do đó, giải thuật này được gọi là hệ thống bạn thân.

Thí du: với bộ nhớ 1M, cần phải có 21 bảng liệt kê như thế sắp từ 1 bytes (2^0) đến 1 byte (2^{20}). Khởi đầu toàn bộ bộ nhớ còn tự do và bảng liệt kê 1M có một mục từ độc nhất chứa đựng một lỗ hổng 1M, tất cả các bảng liệt kê khác đều rỗng. Cấu hình bộ nhớ lúc khởi đầu được chỉ ra trong hình VII-11.

Memory

	0	128 K	256 K	384 K	512K	640 K	768 K	896 K	1M	
Initially										**1 Hole**
request 70	**A**	**128**		**256**			**512**			**3**
request 35	**A**	**B**	**64**	**256**			**512**			**3**
request 80	**A**	**B**	**64**	**C**	**128**		**512**			**3**
return A	**128**	**B**	**64**	**C**	**128**		**512**			**4**

Thuviendientu.org

request 60	**128**	**B**	**D**	**C**	**128**	**512**	**4**
return B	**128**	**64**	**D**	**C**	**128**	**512**	**4**
return D	**256**			**C**	**128**	**512**	**3**
return C	**1024**						**1**

Hình 0-11Hệ thống bạn thân với kích thước 1M

Bây giờ chúng ta hãy xem cách hệ thống buddy làm việc khi một quá trình 70K được hoán vị vào bộ nhớ trống 1M. Do những lỗ hổng chỉ có thể có kích thước là lũy thừa của 2, 128K sẽ được yêu cầu, bởi vì đó chính là lũy thừa nhỏ nhất của 2 đủ lớn. Không có khối 128K sẵn, cũng không có các khối 256K và 512K. Vì vậy khối 1M sẽ được chia làm hai khối 512K, được gọi là những bạn thân; một tại địa chỉ 0 và một tại địa chỉ 512K. Sau đó khối tại địa chỉ thấp hơn, chính là khối tại 0 lại được phân làm hai khối bạn thân 256K, một tại 0 và một tại 256K. Cái thấp hơn của chúng lại được phân làm hai khối 128K, và khối tại 0, đánh dấu là A trong hình được cấp phát cho quá trình.

Kế đến, một quá trình 35K được hoán vị vào. Khi đó ta cần khối 64K, nhưng cũng không có sẵn, vì thế phải phân phối khối 128K thành hai khối bạn thân 64K, một tại địa chỉ 128K, một tại 192K. Khối tại 128K được cấp cho quá trình, trong hình là B. Yêu cầu thứ ba là 80K.

Bây giờ ta hãy xem những gì xảy ra khi một khối được trả lại. Giả sử tại thời điểm này khối 128K (mà chỉ dùng có 70K) được tự do. Khi đó khối 128K sẽ được đưa vào bảng tự do. Bây giờ cần một khối 60K. Sau khi kiểm tra, khối 64K tại 192K được cấp phát và nó được đánh dấu là C.

Bây giờ khối B được trả lại. Tại thời điểm này có hai khối 128K tự do nhưng chúng không được kết hợp lại. Chú ý rằng ngay cả khi khối 128K tại 0 được phân ra làm 2, khối tại 9 được dùng và khối tại 84K còn tự do, sự kết hợp cũng không xãy ra. Khi D được trả lại, sẽ có sự kết hợp lại thành khối 256K tại 0. Cuối cùng, khi C được trả lại, sẽ có kết hợp tạo thành 1 lỗ hổng 1M như ban đầu.

Hệ thống bạn thân có sự thuận lợi so với những giải thuật cùng sắp xếp theo kích thước của khối. Sự thuận lợi này là khi có một khối 2^k bytes tự do, bộ quản lý bộ nhớ chỉ cần tìm trong bảng liệt kê lỗ hổng có kích thước 2^k để xem chúng có khả năng kết hợp được hay không. Với những giải thuật khác mà trong đó cho phép các khối bộ nhớ được phân chia một cách tùy ý, việc tìm kiếm phải diễn ra trên tất cả các bảng liệt kê. Do dó, hệ thống bạn thân làm việc nhanh hơn.

Đáng tiếc, nó lại cực kỳ kém hiệu quả trong việc sử dụng bộ nhớ. Một quá trình 35K phải được cấp phát đến 64K, hao phí đến 29K. Sự hao phí này được gọi là sự **phân mảnh trong** (internal fragmentation), bởi vì phần bộ nhớ hao phí nằm bên trong đoạn được cấp phát. Còn trong các phần trước ta thấy những lỗ hổng ở giữa các đoạn, nhưng không có sự hao phí bên trong các đoạn, do đó kiểu này được coi là sự phân mảnh ngoài.

Thuviendientu.org

V.6 Phân mãnh

Phân mãnh bộ nhớ có thể là phân mãnh trong hoặc phân mãnh ngoài. Xét cơ chế cấp phát nhiều phân khu với một lỗ trống có kích thước 18,464 bytes. Giả sử rằng quá trình tiếp theo yêu cầu 18,462 bytes. Nếu chúng ta cấp phát chính xác khối được yêu cầu, chúng ta để lại một lỗ trống có kích thước 2 bytes. Chi phí để giữ vết của lỗ này sẽ lớn hơn kích thước của lỗ trống. Tiếp cận thông thường là chia bộ nhớ vật lý thành những khối có kích thước cố định, và cấp phát bộ nhớ dựa theo đơn vị của kích thước khối. Với tiếp cận này, bộ nhớ được cấp phát tới một quá trình có thể là lớn hơn một chút so với khối được yêu cầu. Sự chênh lệnh giữa hai số này là phân mãnh trong-bộ nhớ bị phân mãnh trong đối với một phân khu thì không thể được dùng.

Một giải pháp đối với phân mãnh ngoài là **kết lại thành khối** (compaction). Mục tiêu là di chuyển nội dung bộ nhớ để đặt tất cả bộ nhớ trống với nhau trong một khối lớn. Kết khối không phải luôn thực hiện được. Nếu việc tái định vị là tĩnh và được thực hiện tại thời điểm hợp dịch và nạp thì việc kết khối là không thể thực hiện được. Kết khối chỉ có thể thực hiện được chỉ nếu tái định vị là động và được thực hiện tại thời điểm thực thi. Nếu địa chỉ được tái định vị động, tái định vị yêu cầu chỉ di chuyển chương trình và dữ liệu, sau đó thay đổi thanh ghi nền để phản ánh địa chỉ nền mới. Khi kết khối là có thể, chúng ta phải xác định chi phí của nó. Giải thuật kết khối đơn giản nhất là di chuyển tất cả quá trình tới cuối bộ nhớ; tất cả lỗ trống di chuyển theo hướng ngược lại, tạo ra một lỗ trống lớn của bộ nhớ sẵn dùng. Cơ chế này có thể đắt.

Một giải pháp khác cho vấn đề phân mãnh ngoài là cho phép không gian địa chỉ luận lý của một quá trình là không liên tục, do đó cho phép một quá trình được cấp phát bộ nhớ vật lý bất cứ đâu sau khi sẵn dùng. Hai kỹ thuật bù trừ để đạt giải pháp này là phân trang và phân đoạn.

VI Cấp phát không liên tục

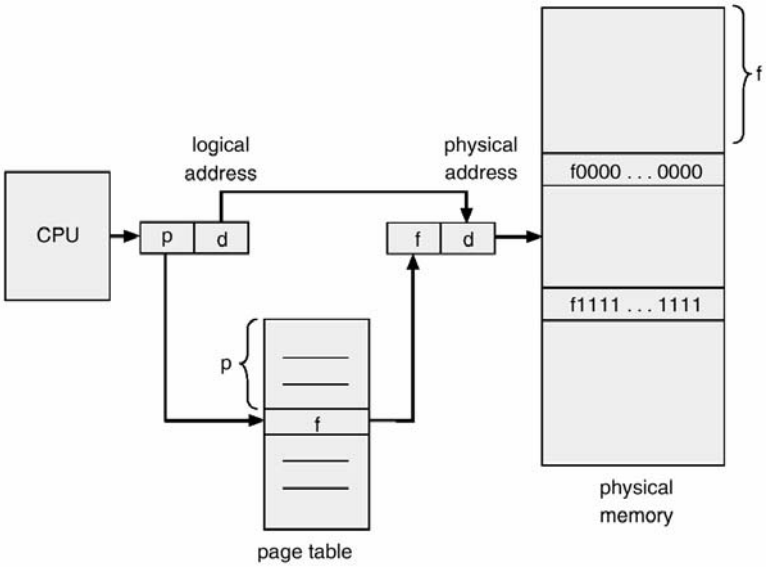
VI.1 Phân trang

Phân trang là cơ chế quản lý bộ nhớ cho phép không gian địa chỉ vật lý của quá trình là không kề nhau. Phân trang tránh vấn đề đặt vừa khít nhóm bộ nhớ có kích thước thay đổi vào vùng lưu trữ phụ (backing store) mà hầu hết các cơ chế quản lý bộ nhớ trước đó gặp phải. Khi phân đoạn mã và dữ liệu nằm trong bộ nhớ được hoán vị ra, không gian phải được tìm thấy trên vùng lưu trữ phụ. Vấn đề phân mãnh được thảo luận trong sự kết nối với bộ nhớ chính cũng thông dụng như với vùng lưu trữ phụ, ngoại trừ truy xuất thấp hơn nhiều, vì thế kết khối là không thể. Vì lợi điểm của nó so với các phương pháp trước đó nên phân trang trong những dạng khác nhau được dùng phổ biến trong hầu hết các hệ điều hành.

Về truyền thống, hỗ trợ phân trang được quản lý bởi phần cứng. Tuy nhiên, những thiết kế gần đây cài đặt phân trang bằng cách tích hợp chặt chẽ phần cứng và hệ điều hành, đặc biệt trên các bộ vi xử lý 64-bit.

Thuviendientu.org

VI.1.1 Phương pháp cơ bản



Hình 0-12 Phần cứng phân trang

Bộ nhớ vật lý được chia thành các khối có kích thước cố định được gọi là các **khung** (frames). Bộ nhớ luận lý cũng được chia thành các khối có cùng kích thước được gọi là các **trang** (pages). Khi một quá trình được thực thi, các trang của nó được nạp vào các khung bộ nhớ sẵn dùng từ vùng lưu trữ phụ. Vùng lưu trữ phụ được chia thành các khối có kích thước cố định và có cùng kích thước như các khung bộ nhớ.

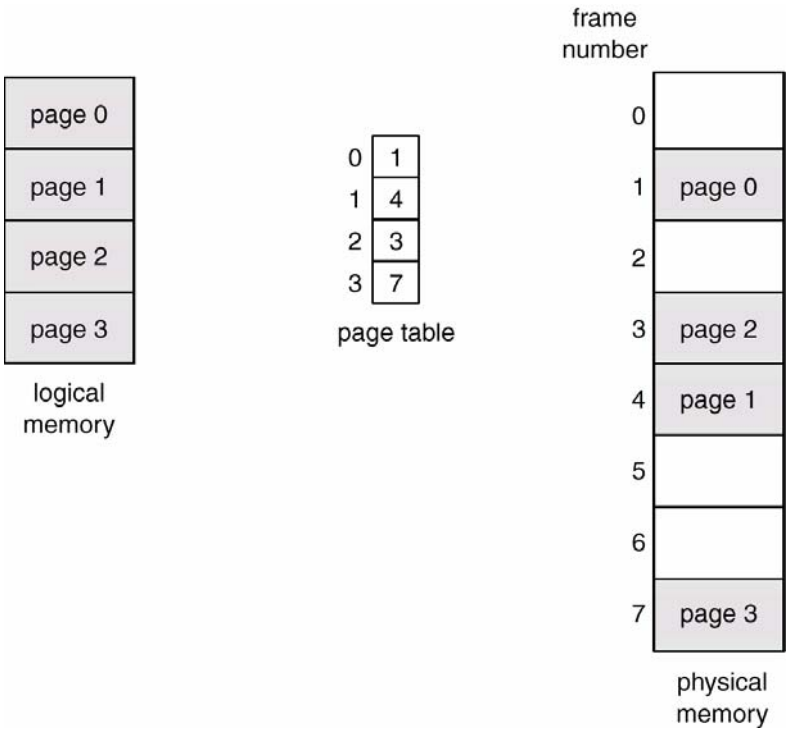
Hỗ trợ phần cứng cho phân trang được hiển thị trong hình VII-12. Mỗi địa chỉ được tạo ra bởi CPU được chia thành hai phần: số trang (p) và độ dời trang (d). Số trang được dùng như chỉ mục vào bảng trang. Bảng trang chứa địa chỉ nền của mỗi trang trong bộ nhớ vật lý. Địa chỉ nền này được kết hợp với độ dời trang để định nghĩa địa chỉ bộ nhớ vật lý mà nó được gởi đến đơn vị bộ nhớ. Mô hình phân trang bộ nhớ được hiển thị như hình VII-13.

Kích thước trang (giống như kích thước khung) được định nghĩa bởi phần cứng. Kích thước của một trang điển hình là luỹ thừa của 2, từ 512 bytes đến 16MB trên trang, phụ thuộc vào kiến trúc máy tính. Chọn luỹ thừa 2 cho kích thước trang thực hiện việc dịch địa chỉ luận lý thành số trang và độ dời trang rất dễ dàng. Nếu kích thước không gian địa chỉ là 2^m, và kích thước trang là 2^n đơn vị địa chỉ (byte hay từ) thì m-n bits cao của địa chỉ luận lý chỉ số trang, n bits thấp chỉ độ dời trang. Do đó, địa chỉ luận lý như sau:

Số trang	Độ dời trang
P	D
m –n	N

ở đây p là chỉ mục trong bảng trang và d là độ dời trong trang.

Thuviendientu.org



Hình 0-13 Mô hình phân trang của bộ nhớ luận lý và vật lý

Thí dụ: xét bộ nhớ trong hình VII-14. Sử dụng kích thước trang 4 bytes và bộ nhớ vật lý 32 bytes (có 8 trang), chúng ta hiển thị cách nhìn bộ nhớ của người dùng có thể được ánh xạ tới bộ nhớ vật lý như thế nào. Địa chỉ luận lý 0 là trang 0, độ dời 0. Chỉ mục trong bảng trang, chúng ta thấy rằng trang 0 ở trong khung 5. Do đó, địa chỉ luận lý 0 ánh xạ tới địa chỉ vật lý 20 (=(5x4)+0). Địa chỉ luận lý 3 (trang 0, độ dời 3) ánh xạ tới địa chỉ vật lý 23 (=(5x4)+3). Địa chỉ luận lý 4 ở trang 1, độ dời 0; dựa theo bảng trang, trang 1 được ánh xạ tới khung 6. Do đó, địa chỉ luận lý 4 ánh xạ tới địa chỉ 24(=(6x4)+0). Địa chỉ luận lý 13 ánh xạ tới địa chỉ vật lý 9.

Có thể chú ý rằng phân trang là một dạng của tái định vị động. Mỗi địa chỉ luận lý được giới hạn bởi phần cứng phân trang tới địa chỉ vật lý. Sử dụng phân trang tương tự sử dụng một bảng các thanh ghi nền (hay tái định vị), một thanh ghi cho mỗi khung bộ nhớ.

Khi chúng ta sử dụng một cơ chế phân trang, chúng ta không có phân mãnh bên ngoài: bất kỳ khung trống có thể được cấp phát tới một quá trình cần nó. Tuy nhiên, chúng ta có thể có phân mãnh trong. Chú ý rằng các khung được cấp phát như các đơn vị. Nếu các yêu cầu bộ nhớ của một quá trình không xảy ra để rơi trên giới hạn của trang, thì khung cuối cùng được cấp phát có thể không đầy hoàn toàn. Thí dụ, nếu các trang là 2048 bytes, một quá trình 72,766 bytes sẽ cần 35 trang cộng với 1086 bytes. Nó được cấp phát 36 khung, do đó phân mãnh trong là 2048 - 1086 = 962 bytes. Trong trường hợp xấu nhất, một quá trình cần n trang cộng với 1 byte. Nó sẽ được cấp phát n+1 khung, dẫn đến phân mãnh trong gần như toàn bộ khung.

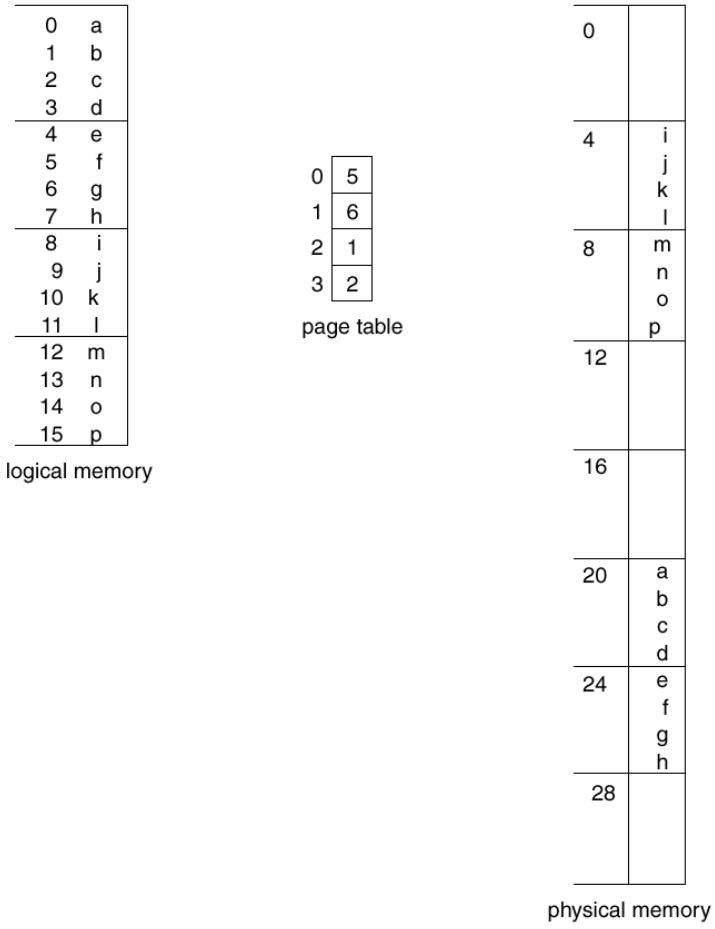
Nếu kích thước quá trình độc lập với kích thước của trang, thì chúng ta mong muốn phân mãnh trong trung bình là ½ trang trên một quá trình. Xem xét này đề nghị rằng kích thước trang nhỏ là mong muốn. Tuy nhiên, chi phí liên quan tới mỗi mục từ bảng trang và chi phí này giảm khi kích thước trang tăng. Vì thế nhập/xuất đĩa là hiệu quả hơn khi số lượng dữ liệu được truyền lớn hơn. Thường thì kích thước trang lớn lên theo thời gian khi các quá trình, tập hợp dữ liệu, bộ nhớ chính trở nên lớn hơn. Ngày nay, các trang điển hình nằm trong khoảng 4 KB tới 8 KB, và một số hệ thống hỗ trợ kích thước trang lớn hơn. CPU và nhân thậm chí hỗ trợ nhiều kích thước khác

Thuviendientu.org

nhau. Thí dụ, Solaris dùng 8 KB và 4 MB kích thước trang, phụ thuộc dữ liệu được lưu bởi trang. Hiện nay, các nhà nghiên cứu đang phát triển việc hỗ trợ kích thước trang khác nhau.

Mỗi mục từ bảng trang thường dài 4 bytes, nhưng kích thước có thể thay đổi. Một mục từ 32-bit có thể chỉ tới một khung trang vật lý 2^{32}. Nếu một khung là 4 KB, thì hệ thống với những mục từ 4 bytes có thể đánh địa chỉ cho 2^{44} bytes (hay 16 TB) bộ nhớ vật lý.

Khi một quá trình đi vào hệ thống để được thực thi, kích thước của nó, được diễn tả trong các trang, được xem xét. Mỗi trang của quá trình cần trên một khung. Do đó, nếu quá trình yêu cầu n trang, ít nhất n khung phải sẵn dùng trong bộ nhớ. Nếu n khung là sẵn dùng, chúng được cấp phát tới quá trình đang đi vào này. Trang đầu tiên của quá trình được nạp vào một trong những khung được cấp phát, và số khung được đặt vào trong bảng trang cho quá trình này. Trang kế tiếp được nạp vào một khung khác, và số khung của nó được đặt vào trong bảng trang, …(hình VII-15).



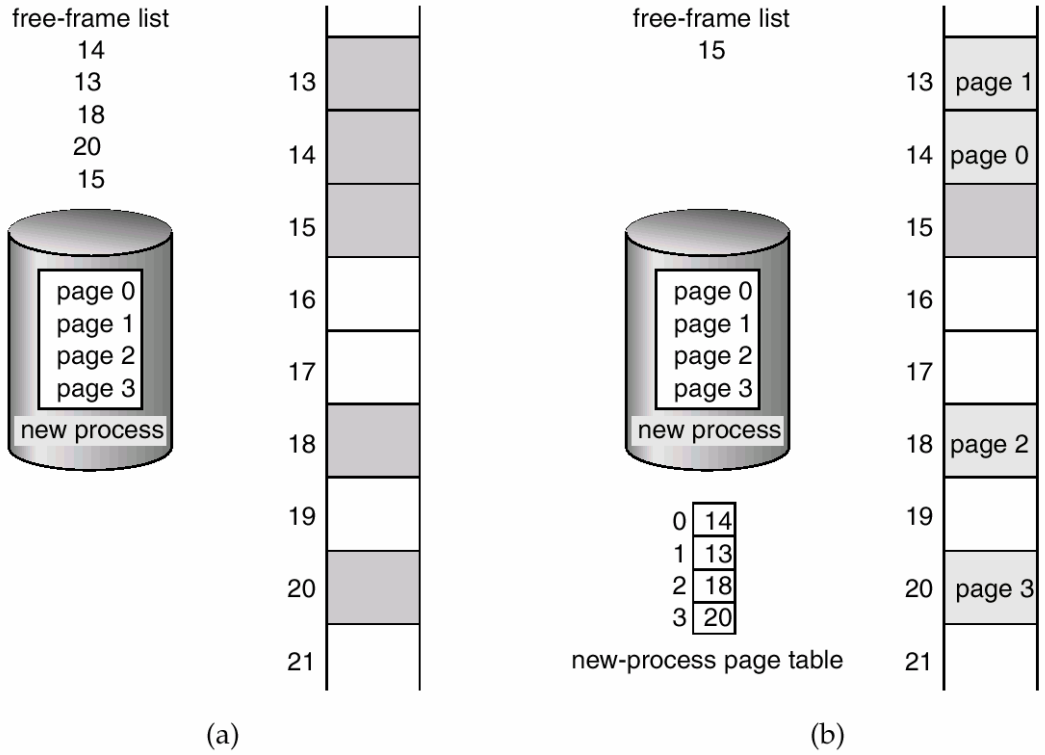
Hình 0-14 Thí dụ phân trang cho bộ nhớ 32 bytes với các trang có kích thước 4 bytes

Một khía cạnh quan trọng của phân trang là sự phân chia rõ ràng giữa tầm nhìn bộ nhớ của người dùng và bộ nhớ vật lý thật sự. Chương trình người dùng nhìn bộ nhớ như một không gian liên tục, chứa chỉ một chương trình. Sự thật, chương trình người dùng được phân bố khắp bộ nhớ vật lý mà nó cũng quản lý các quá trình khác. Sự khác nhau giữa tầm nhìn bộ nhớ của người dùng và bộ nhớ vật lý thật sự được làm cho tương thích bởi phần cứng dịch địa chỉ. Địa chỉ luận lý được dịch thành địa chỉ vật lý. Ánh xạ này được che giấu từ người dùng và được điều khiển bởi hệ điều hành. Chú ý rằng như định nghĩa, quá trình người dùng không thể truy xuất bộ nhớ mà nó

Thuviendientu.org

không sở hữu. Không có cách định địa chỉ bộ nhớ bên ngoài bảng trang của nó và bảng chứa chỉ những trang mà quá trình sở hữu.

Vì hệ điều hành đang quản lý bộ nhớ vật lý nên nó phải hiểu những chi tiết cấp phát bộ nhớ vật lý; khung nào được cấp phát, khung nào còn trống, tổng khung hiện có là bao nhiêu,…Thông tin này được giữ trong một cấu trúc dữ liệu được gọi là bảng khung. Bảng khung chỉ có một mục từ cho mỗi khung trang vật lý, hiển thị khung trang đó đang rảnh hay được cấp phát. Nếu khung trang được cấp phát, thì xác định trang nào của quá trình nào được cấp.



Hình 0-15 các khung trống. (a) trước khi cấp phát. (b) sau khi cấp phát

Ngoài ra, hệ điều hành phải biết rằng quá trình người dùng hoạt động trong không gian người dùng, và tất cả địa chỉ luận lý phải được ánh xạ để phát sinh địa chỉ vật lý. Nếu người dùng thực hiện lời gọi hệ thống (thí dụ: để thực hiện nhập/xuất) và cung cấp địa chỉ như một tham số (thí dụ: vùng đệm), địa chỉ đó phải được ánh xạ để sinh ra địa chỉ vật lý đúng. Hệ điều hành duy trì một bản sao của bảng trang cho mỗi quá trình, như nó duy trì bản sao của bộ đếm chỉ thị lệnh và nội dung thanh ghi. Bản sao này được dùng để dịch địa chỉ luận lý thành địa chỉ vật lý bất cứ khi nào hệ điều hành phải ánh xạ địa chỉ luận lý tới địa chỉ vật lý dạng thủ công. Nó cũng được dùng bởi bộ phân phát CPU để địa chỉ bảng trang phần cứng khi một quá trình được cấp phát CPU. Do đó, trang gia tăng thời gian chuyển đổi ngữ cảnh.

VI.1.2 Hỗ trợ phần cứng

Mỗi hệ điều hành có phương pháp riêng để lưu trữ các bảng trang. Hầu hết đều cấp phát một bảng trang cho mỗi quá trình. Một con trỏ chỉ tới một bảng trang được lưu trữ với những giá trị thanh ghi thanh ghi khác nhau (giống như bộ đếm chỉ thị lệnh) trong khối điều khiển quá trình. Khi bộ phân phát được yêu cầu bắt đầu một quá trình, nó phải nạp lại các thanh ghi người dùng và định nghĩa các giá trị bảng trang phần cứng phù hợp từ bảng trang người dùng được lưu trữ.

Thuviendientu.org

Cài đặt phần cứng của bảng trang có thể được thực hiện trong nhiều cách. Cách đơn giản nhất, bảng trang được cài đặt như tập hợp các thanh ghi tận hiến. Các thanh ghi này nên được xây dựng với tính logic tốc độ rất cao để thực hiện việc dịch địa chỉ trang hiệu quả. Mọi truy xuất tới bộ nhớ phải kiểm tra kỹ lưỡng bảng đồ trang, vì vậy tính hiệu quả là vấn đề xem xét chủ yếu. Bộ phân phát CPU nạp lại các thanh ghi này chỉ khi nó nạp lại các thanh ghi khác. Dĩ nhiên, các chỉ thị để nạp hay hiệu chỉnh các thanh ghi bảng trang phải được cấp quyền để mà chỉ hệ điều hành có thể thay đổi bản đồ bộ nhớ. DEC PDP-11 là một thí dụ về kiến trúc như thế. Địa chỉ chứa 16 bits, và kích thước trang là 8 KB. Do đó, bảng trang chứa 8 mục từ mà chúng được giữ trong các thanh ghi nhanh.

Sử dụng các thanh ghi cho bảng trang chỉ phù hợp nếu bảng trang có kích thước nhỏ (thí dụ: 256 mục từ). Tuy nhiên, hầu hết các máy tính tương thời cho phép bảng trang rất lớn (thí dụ, 1 triệu mục từ). Đối với những máy này, việc sử dụng các thanh ghi nhanh để cài đặt bảng trang là không khả thi. Hay đúng hơn là, bảng trang được giữ trong bộ nhớ chính, và thanh ghi nền bảng trang (page-table base register-PTBR) chỉ tới thanh ghi bảng trang. Thay đổi các bảng trang yêu cầu thay đổi chỉ một thanh ghi, về căn bản cắt giảm thời gian chuyển ngữ cảnh.

Vấn đề với tiếp cận này là thời gian được yêu cầu để truy xuất vị trí bộ nhớ người dùng. Nếu chúng ta muốn truy xuất vị trí i, đầu tiên chúng ta phải xác định chỉ mục trong bảng trang, sử dụng giá trị trong độ dời PTBR bởi số trang cho i. Tác vụ này yêu cầu một truy xuất bộ nhớ. Nó cung cấp chúng ta số khung được nối kết với độ dời trang để sinh ra địa chỉ thực. Sau đó, chúng ta có thể truy xuất tới nơi được mong muốn trong bộ nhớ. Với cơ chế này, hai truy xuất bộ nhớ được yêu cầu để truy xuất một byte (một cho mục từ bảng trang, một cho byte đó). Do đó, truy xuất bộ nhớ bị chậm bởi một trong hai yếu tố đó. Sự trì hoãn này không thể chấp nhận dưới hầu hết trường hợp vì thế chúng ta phải sử dụng đến hoán vị!

Giải pháp chuẩn cho vấn đề này là dùng bộ lưu trữ (cache) phần cứng đặc biệt, nhỏ, tìm kiếm nhanh được gọi là translation look-aside buffer (TLB). TLB là bộ nhớ kết hợp tốc độ cao. Mỗi mục từ trong TLB chứa hai phần: một khoá (key) và một giá trị (value). Khi bộ nhớ kết hợp được biểu diễn với một thành phần, nó được so sánh với tất cả khoá cùng một lúc. Nếu thành phần được tìm thấy, trường giá trị tương ứng được trả về. Tìm kiếm nhanh nhưng phần cứng đắt. Điển hình, số lượng mục từ trong TLB nhỏ, thường từ 64 đến 1024.

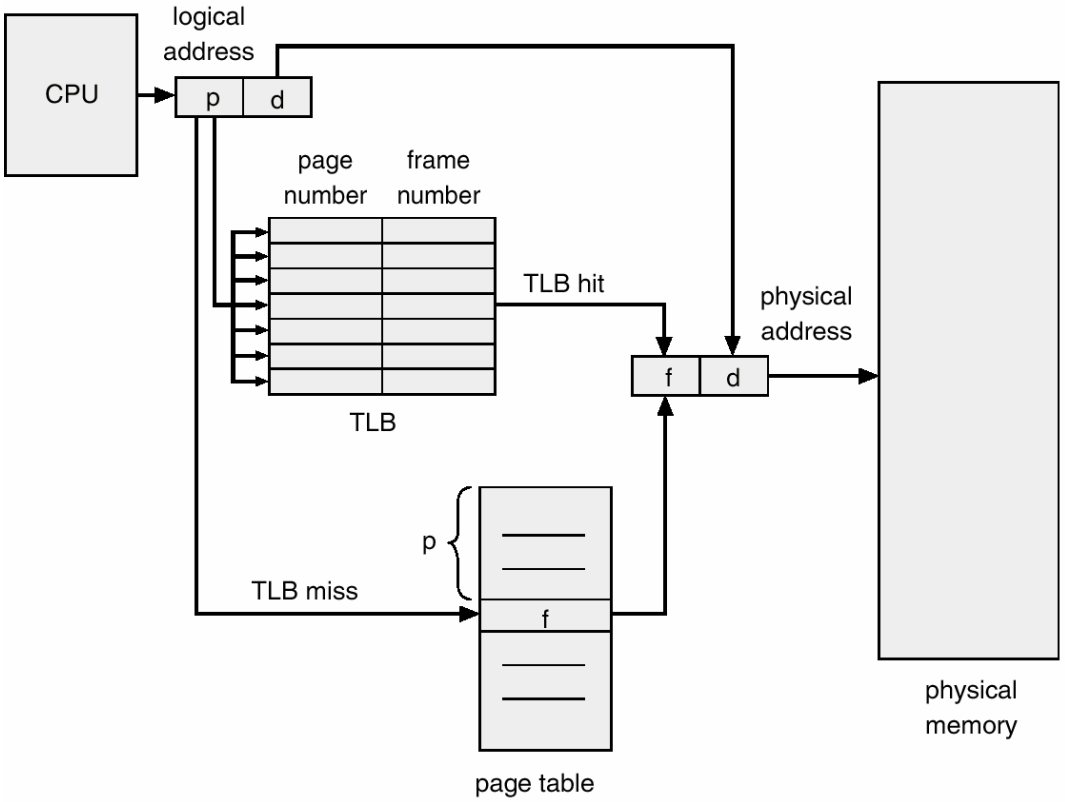
TLB được dùng với các bảng trang trong cách sau. TLB chứa chỉ một vài mục từ bảng trang. Khi một địa chỉ luận lý được phát sinh bởi CPU, số trang của nó được hiện diện trong TLB. Nếu số trang được tìm thấy, khung của nó lập tức sẵn dùng và được dùng để truy xuất bộ nhớ. Toàn bộ tác vụ có thể mất ít hơn 10% thời gian nếu dùng tham chiếu bộ nhớ không được ánh xạ.

Nếu số trang không ở trong TLB (còn gọi là mất TLB), tham chiếu bộ nhớ tới bảng trang phải được thực hiện. Khi số khung đạt được, chúng ta có thể dùng nó để truy xuất bộ nhớ (như hình VII-16). Ngoài ra, chúng ta thêm số trang và số khung tới TLB để mà chúng có thể được tìm thấy nhanh trên tham chiếu tiếp theo. Nếu một TLB đã đầy các mục từ, hệ điều hành phải chọn một mục từ để thay thế. Các chính sách thay thế rất đa dạng từ ít được sử dụng gần đây nhất (least recently used-LRU) tới chọn ngẫu nhiên. Ngoài ra, một số TLB cho phép các mục từ được ***wired down***. Nghĩa là, chúng không thể được xoá khỏi TLB. Điển hình, các mục từ cho nhân thường được wired down.

Một số TLB lưu trữ các định danh không gian địa chỉ (address-space identifers-ASID) trong mỗi mục từ của TLB. Một ASID định danh duy nhất mỗi quá trình và được dùng để cung cấp việc bảo vệ không gian địa chỉ cho quá trình đó. Khi TLB cố

Thuviendientu.org

gắng phân giải các số trang ảo, nó đảm bảo ASID cho mỗi quá trình hiện đang chạy trùng khớp với ASID được nối kết với trang ảo. Nếu các ASID không khớp, chúng được xem như mất TLB. Ngoài ra, để cung cấp việc bảo vệ không gian địa chỉ, một ASID cho phép TLB chứa các mục từ cho nhiều quá trình khác nhau cùng một lúc. Nếu TLB không hỗ trợ các ASID riêng thì mỗi lần một bảng trang được chọn (thí dụ, mỗi chuyển ngữ cảnh), một TLB phải được đẩy (hay được xoá) để đảm bảo rằng các quá trình đang thực thi tiếp theo không sử dụng thông tin dịch sai. Ngược lại, có những mục từ cũ trong TLB chứa các địa chỉ ảo nhưng có các địa chỉ không đúng hay không hợp lệ để lại từ quá trình trước.



Hình 0-16 phần cứng phân trang với TBL

Phần trăm thời gian mà số trang xác định được tìm thấy trong TLB được gọi là tỉ lệ chập (hit ratio). Tỉ lệ chập 80% có nghĩa là chúng ta tìm số trang mong muốn trong TLB 80% thời gian. Nếu mất 20 nano giây để tìm TLB và 100 nano giây để truy xuất bộ nhớ, thì một truy xuất bộ nhớ được ánh xạ mất 120 nano giây khi số trang ở trong TLB. Nếu chúng ta không tìm số trang trong TLB (20 nano giây) thì trước hết chúng ta phải truy xuất bộ nhớ cho bảng trang và số khung (100 nano giây), thì sau đó truy xuất byte mong muốn trong bộ nhớ (100 nano giây), tổng thời gian là 220 nano giây. Để tìm thời gian truy xuất bộ nhớ hiệu quả, chúng ta phải đo mỗi trường hợp với xác suất của nó:

Thời gian truy xuất hiệu quả = 0.80 x 120 + 0.2 x 220 = 140 nano giây

Trong thí dụ này, chúng ta gặp phải 40% chậm lại trong thời gian truy xuất bộ nhớ (từ 100 tới 140 nano giây).

Đối với một tỉ lệ chậm 98%, chúng ta có:

Thời gian truy xuất hiệu quả = 0.98 x 120 + 0.02 x 220 = 122 nano giây

Tỉ lệ chập được tăng này chỉ tạo ra 22% chậm lại trong thời gian truy xuất.

Thuviendientu.org

VI.1.3 Sự bảo vệ

Bảo vệ bộ nhớ trong môi trường phân trang được thực hiện bởi các bit bảo vệ gán liền với mỗi khung. Thông thường, các bit này được giữ trong bảng trang. Một bit có thể định nghĩa một trang để đọc-viết hay chỉ đọc. Mỗi tham chiếu tới bộ nhớ sẽ tìm khắp bảng trang để xác định số khung tương ứng. Tại cùng thời điểm địa chỉ vật lý được tính , các bit bảo vệ có thể được kiểm tra để thẩm định rằng không có thao tác viết nào đang được thực hiện tới trang chỉ đọc. Cố gắng viết tới một trang chỉ đọc gây ra một trap phần cứng tới hệ điều hành (hay xung đột bộ nhớ bảo vệ).

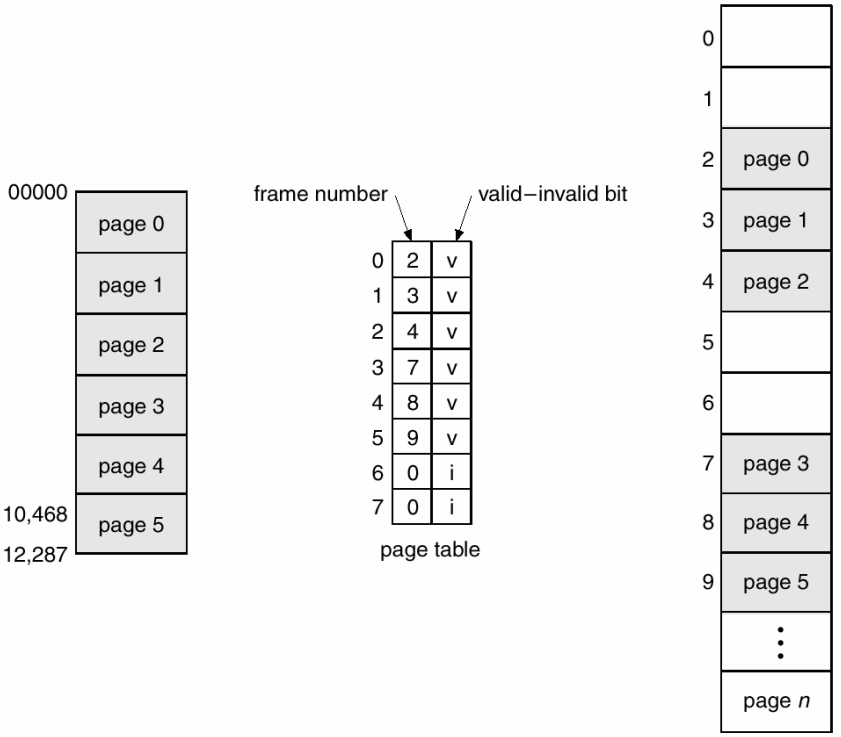
Chúng ta có thể dễ dàng mở rộng tiếp cận này để cung cấp một cấp độ bảo vệ chi tiết hơn. Chúng ta có thể tạo phần cứng để cung cấp bảo vệ chỉ đọc, đọc viết, chỉ thực thi. Hay bằng cách cung cấp các bit bảo vệ riêng cho mỗi loại truy xuất, chúng ta có thể cho phép bất cứ kết hợp của các truy xuất này; các cố gắng không hợp lệ sẽ được trap tới hệ điều hành.

Một bit nữa thường được gán tới mỗi mục từ trong bảng trang: một **bit hợp lệ-không hợp lệ**. Khi bit này được đặt là "hợp lệ" thì giá trị này hiển thị rằng trang được gán trong không gian địa chỉ luận lý bộ nhớ là trang hợp lệ. Nếu bit này được đặt là "không hợp lệ", giá trị này hiển thị trang đó không ở trong không gian địa chỉ luận lý của quá trình. Các địa chỉ không hợp lệ được trap bằng cách sử dụng bit hợp lệ-không hợp lệ. Hệ điều hành thiết lập bit này cho mỗi trang để cho phép hay không cho phép truy xuất tới trang này. Thí dụ, trong một hệ thống với không gian địa chỉ 14 bit (0 tới 16383), chúng ta có thể có một chương trình sử dụng chỉ địa chỉ 0 tới 10468. Cho kích thước trang 2KB, chúng ta xem trường hợp trong hình VII-17. Địa chỉ trong các trang 0, 1, 2, 3, 4, và 5 thường được ánh xạ khắp bảng trang. Tuy nhiên, bất cứ những nỗ lực để tạo ra một địa chỉ trong trang 6 hay 7 nhận thấy rằng bit hợp lệ-không hợp lệ được đặt là không hợp lệ và máy tính sẽ trap tới hệ điều hành (tham chiếu trang không hợp lệ).

Vì chương trình mở rộng chỉ tới địa chỉ 10468, bất cứ tham chiếu vượt ra ngoài địa chỉ đó là không hợp lệ. Tuy nhiên, các tham chiếu tới trang 5 được xem là hợp lệ vì thế những địa chỉ tới 12287 là hợp lệ. Chỉ những địa chỉ từ 12288 tới 16383 là không hợp lệ. Vấn đề này là do kích thước trang 2KB và phản ánh phân mãnh trong của việc phân trang.

Rất hiếm khi một quá trình dùng tất cả dãy địa chỉ của nó. Thật vậy, nhiều quá trình dùng chỉ một phần nhỏ không gian địa chỉ còn trống cho chúng. Nó sẽ lãng phí rất nhiều trong những trường hợp này để tạo một bảng trang với các mục từ cho mỗi trang trong dãy địa chỉ. Hầu hết bảng này sẽ không được dùng nhưng sẽ mang đến không gian bộ nhớ có giá trị. Một số hệ thống cung cấp phần cứng, trong dạng một thanh ghi có chiều dài bảng trang (page-table length register-PTLR) để hiển thị kích thước của bảng trang. Giá trị này được kiểm tra dựa trên mỗi địa chỉ luận lý để thẩm định địa chỉ đó nằm trong dãy địa chỉ hợp lệ cho quá trình. Lỗi của việc kiểm tra này gây ra một trap lỗi tới hệ điều hành.

Thuviendientu.org



Hình 0-17 Bit hợp lệ (v) và không hợp lệ (i) trong một bảng trang

VI.1.4 Cấu trúc bảng trang

Trong phần này chúng ta sẽ xem xét một số kỹ thuật thông dụng nhất để xây dựng cấu trúc bảng trang.

.VI.1.4.1 Bảng trang phân cấp

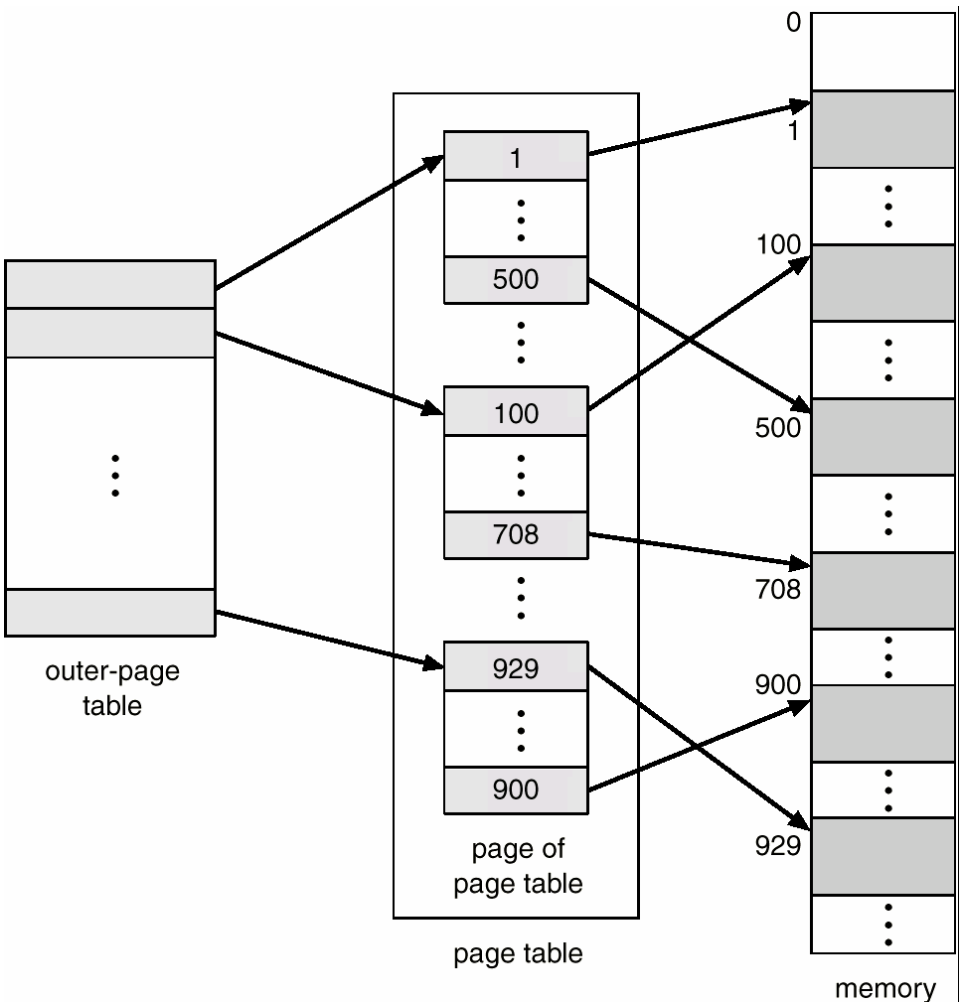
Hầu hết các hệ thống máy tính hiện đại hỗ trợ một không gian địa chỉ luận lý lớn (2^{32} tới 2^{64}). Trong môi trường như thế, bảng trang trở nên quá lớn. Thí dụ, xét một hệ thống với không gian địa chỉ luận lý 32 bit. Nếu kích thước trang 4KB thì bảng trang có thể chứa tới 1 triệu mục từ ($2^{32}/2^{12}$). Giả sử rằng mỗi mục từ chứa 4 bytes, mỗi quá trình có thể cần tới 4MB không gian địa chỉ vật lý cho một bảng trang. Rõ ràng, chúng ta sẽ không muốn cấp phát bảng trang liên tiếp nhau. Một giải pháp đơn giản cho vấn đề này là chia bảng trang thành những phần nhỏ hơn. Có nhiều cách để đạt được sự phân chia này.

Một cách là dùng giải thuật phân trang hai cấp, trong đó bảng trang cũng được phân trang như hình VII-18.

Đây là thí dụ cho máy 32 bit với kích thước trang 4KB. Địa chỉ luận lý được chia thành số trang chứa 20 bit và độ dời trang chứa 12 bit. Vì chúng ta phân trang bảng trang, số trang được chia thành số trang 10 bit và độ dời trang 10-bit. Do đó, một địa chỉ luận lý như sau:

Số trang		Độ dời trang
P_1	P_2	d
10	10	12

Thuviendientu.org



Hình 0-18 Cơ chế bảng trang hai cấp

Ở đây p_1 là chỉ mục trong bảng trang bên ngoài và p_2 là độ dời trong trang của bảng trang bên ngoài. Phương pháp dịch địa chỉ cho kiến trúc này được hiển thị trong hình VII-19. Vì dịch địa chỉ thực hiện từ những phần trong bảng trang bên ngoài, cơ chế này cũng được gọi là **bảng trang được ánh xạ chuyển tiếp** (forward-mapped page table). Petium-II sử dụng kiến trúc này.

Kiến trúc VAX cũng hỗ trợ một biến dạng của phân trang hai cấp. VAX là máy 32-bit với kích thước trang 512 bytes. Không gian địa chỉ luận lý của một quá trình được chia làm 4 phần bằng nhau, mỗi phần chứa 2^{30} bytes. Mỗi phần biểu diễn một phần khác nhau của không gian địa chỉ luận lý của một quá trình. Hai bit cao đầu tiên của địa chỉ luận lý chỉ rõ phần tương ứng. 21 bits tiếp theo biểu diễn số trang luận lý của phần đó, và 9 bits cuối biểu diễn độ dời trong trang mong muốn. Bằng cách chia bảng trang như thế, hệ điều hành có thể để những phân khu không được dùng cho tới khi một quá trình yêu cầu chúng. Một địa chỉ trên kiến trúc VAX như sau:

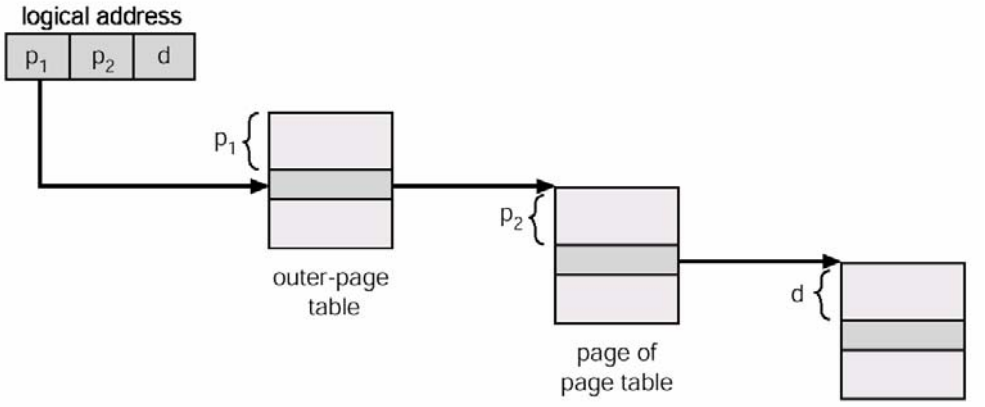
Phần	Trang	Độ dời
S	P	D
2	21	9

ở đây s chỉ rõ số phần, p là chỉ mục trong bảng trang và d là độ dời trong trang.

Thuviendientu.org

Kích thước của bảng trang cấp một cho một quá trình VAX dùng một phần vẫn là 2^{21} bits * 4 bytes/trang = 8 MB. Để việc sử dụng bộ nhớ chính bị giảm nhiều hơn, VAX phân trang các bảng trang quá trình người dùng.

Đối với các hệ thống có không gian địa chỉ luận lý 64 bits, cơ chế phân trang hai cấp không còn phù hợp nữa. Để thể hiện điểm này, chúng ta giả sử rằng kích thước trang trong hệ thống là 4 KB (2^{12}). Trong trường hợp này, bảng trang sẽ chứa tới 2^{52} mục từ. Nếu chúng ta dùng cơ chế phân trang hai cấp thì các bảng bên trong có thể là một trang dài chứa 2^{10} mục từ. Các địa chỉ sẽ như thế này:



Hình 0-19 Dịch địa chỉ cho kiến trúc phân trang hai cấp 32-bit

Trang bên ngoài	Trang bên trong	Độ dời
P1	P2	D
42	10	12

Bảng trang bên ngoài sẽ chứa 242 mục từ, hay 244 bytes. Các phương pháp được chú trọng để tránh để trang lớn là chia bảng trang bên ngoài thành những phần nhỏ hơn. Tiếp cận này cũng được dùng trên một vài bộ xử lý 32-bit để thêm khả năng mềm dẽo và hiệu quả.

Chúng ta có thể chia bảng trang bên ngoài thành cơ chế phân trang 3 cấp. Giả sử rằng bảng trang bên ngoài được tạo ra từ các trang có kích thước chuẩn (210 mục từ, hay 212 bytes); một không gian địa chỉ 64 bit vẫn có kích thước rất lớn:

Trang bên ngoài cấp 2	Trang bên ngoài	Trang bên trong	Độ dời
P1	P2	P3	D
32	10	10	12

Bảng trang bên ngoài vẫn lớn 232.

Bước tiếp theo sẽ là cơ chế phân trang cấp bốn, ở đây bảng trang bên ngoài cấp hai cũng được phân trang. Kiến trúc SPARC (với 32-bit đánh địa chỉ) hỗ trợ cơ chế phân trang cấp ba, trái lại kiến trúc Motorola 68030 32-bit hỗ trợ cơ chế phân trang bốn cấp.

Tuy nhiên, đối với kiến trúc 64-bit, các bảng trang phân cấp thường được xem xét là không phù hợp. Thí dụ, UltraSPARC 64-bit sẽ yêu cầu phân trang bảy cấp – một số truy xuất bộ nhớ không được phép để dịch mỗi địa chỉ luận lý.

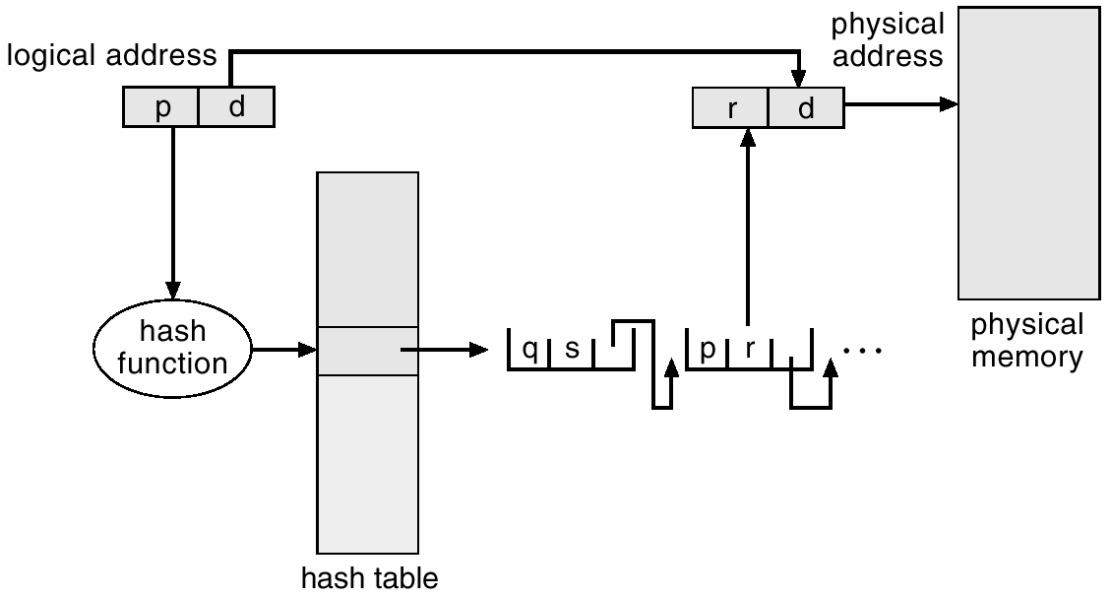
Thuviendientu.org

.VI.1.4.2 Bảng trang được băm

Một tiếp cận thông thường cho việc quản lý không gian địa chỉ lớn hơn 32-bit là dùng **bảng trang được băm** (hashed page table), với giá trị băm là số trang ảo. Mỗi mục từ trong bảng trang chứa một danh sách liên kết của các phần tử. Danh sách này băm tới cùng vị trí (để quản lý đụng độ). Mỗi phần tử chứa ba trường: (a) số trang ảo, (b) giá trị khung trang được ánh xạ và con trỏ chỉ tới phần tử kế tiếp trong danh sách liên kết.

Giải thuật thực hiện như sau: số trang ảo trong địa chỉ ảo được băm tới bảng băm. Số trang ảo được so sánh tới trường (a) trong phần tử đầu tiên của danh sách liên kết. Nếu có phần tử trùng khớp, khung trang tương ứng (trường (b) được dùng để hình thành địa chỉ vật lý mong muốn). Nếu không có phần tử nào trùng khớp, các mục từ tiếp theo trong danh sách liên kết được tìm kiếm số trang ảo trùng khớp. Cơ chế này được hiển thị trong hình VII-20 dưới đây:

Một biến thể đối với cơ chế này cho không gian địa chỉ 64-bit được đề nghị. Bảng trang được nhóm (Clustered page tables) tương tự như bảng băm ngoại trừ mỗi mục từ trong bảng băm tham chiếu tới nhiều trang (chẳng hạn như 16) hơn là một trang. Do đó, mục từ bảng trang đơn có thể lưu những ánh xạ cho nhiều khung trang vật lý. Bảng trang được nhóm đặc biệt có ích cho không gian địa chỉ rời nhau (**spare**), ở đó các tham chiếu bộ nhớ là không liên tục và tập hợp khắp không gian bộ nhớ.



Hình 0-20 Bảng trang được băm

.VI.1.4.3 Bảng trang đảo

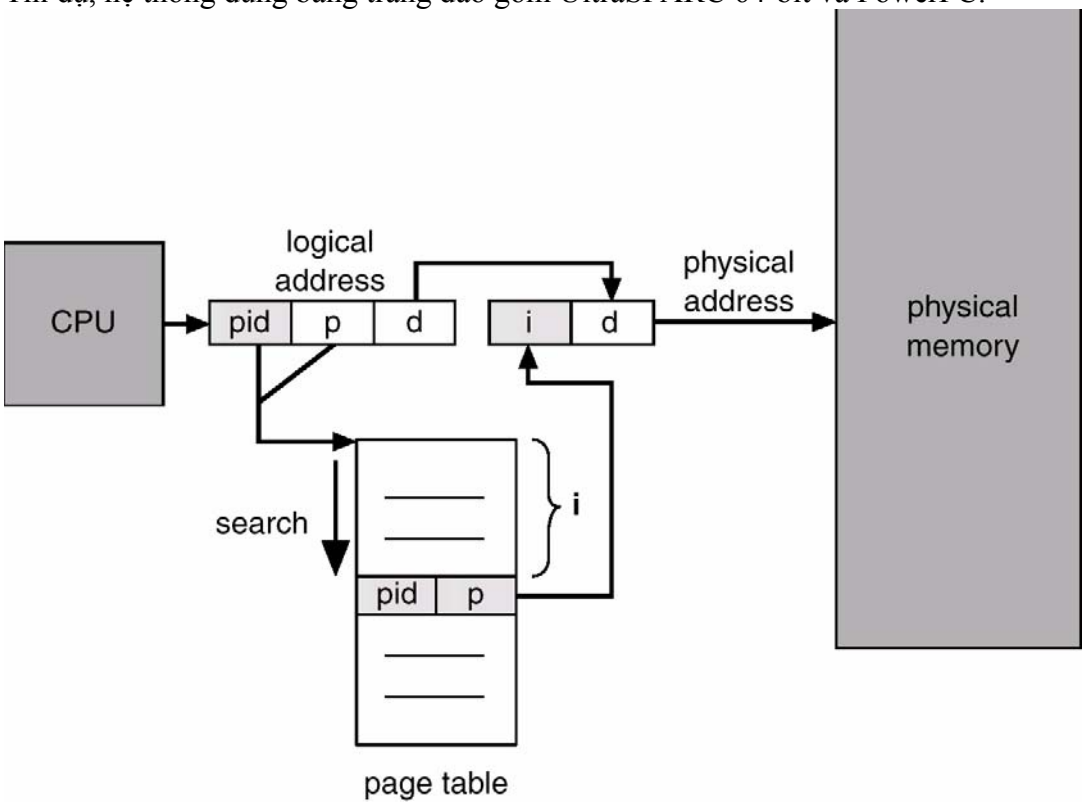
Thông thường, mỗi quá trình có một trang gán liền với nó. Bảng trang có một mục từ cho mỗi trang mà quá trình đó đang sử dụng (hay một khe cho mỗi địa chỉ ảo, không phụ thuộc tính hợp lệ sau đó). Biểu diễn bảng trang này là biểu diễn tự nhiên vì tham chiếu quá trình phân trang thông qua các địa chỉ ảo của trang. Sau đó, hệ điều hành phải dịch tham chiếu này vào một địa chỉ bộ nhớ vật lý. Vì bảng này được sắp xếp bởi địa chỉ ảo, hệ điều hành có thể tính toán nơi trong bảng mà mục từ địa chỉ vật lý được nối kết tới và sử dụng giá trị đó trực tiếp. Một trong những khó khăn của phương pháp này là mỗi bảng trang có thể chứa hàng triệu mục từ. Các bảng này có

Thuviendientu.org

thể tiêu tốn lượng lớn bộ nhớ vật lý, được yêu cầu chỉ để giữ vết của bộ nhớ vật lý khác đang được sử dụng như thế nào.

Để giải quyết vấn đề này chúng ta có thể sử dụng một **bảng trang đảo** (inverted page table). Bảng trang đảo có một mục từ cho mỗi trang thật (hay khung) của bộ nhớ. Mỗi mục từ chứa địa chỉ ảo của trang được lưu trong vị trí bộ nhớ thật đó, với thông tin về quá trình sở hữu trang đó. Do đó, chỉ một bảng trang trong hệ thống và nó chỉ có một mục từ cho mỗi trang của bộ nhớ vật lý. Hình VII-21 dưới đây hiển thị hoạt động của bảng trang đảo.

So sánh nó với hình VII-6, mô tả hoạt động của một bảng trang chuẩn. Vì chỉ một bảng trang trong hệ thống còn có nhiều không gian địa chỉ khác ánh xạ bộ nhớ vật lý, nên các bảng trang đảo thường yêu cầu một định danh không gian địa chỉ được lưu trong mỗi mục từ của bảng trang. Lưu trữ định danh không gian địa chỉ đảm bảo rằng ánh xạ của trang luận lý cho một quá trình xác định tới khung trang vật lý tương ứng. Thí dụ, hệ thống dùng bảng trang đảo gồm UltraSPARC 64-bit và PowerPC.



Hình 0-21 Bảng trang đảo

Để hiển thị phương pháp này, chúng ta mô tả một ấn bản được đơn giản hoá cài đặt bảng trang đảo dùng trong IBM RT. Mỗi địa chỉ ảo trong hệ thống chứa bộ ba: <process-id, page-number, offset>.

Mỗi mục từ bảng trang đảo là một cặp <process-id, page-number>, ở đây process-id đảm bảo vai trò định danh không gian địa chỉ. Khi một tham chiếu bộ nhớ xảy ra, một phần của địa chỉ ảo, gồm <process-id, page-number>, được hiện diện trong hệ thống bộ nhớ. Sau đó, bảng trang đảo được tìm kiếm sự trùng khớp. Nếu sự trùng khớp được tìm thấy tại mục từ i thì địa chỉ vật lý <i, offset> được tạo ra. Nếu không tìm thấy thì một truy xuất địa chỉ không hợp lệ được cố gắng thực hiện.

Mặc dù cơ chế này giảm lượng bộ nhớ được yêu cầu để lưu mỗi bảng trang, nhưng nó gia tăng lượng thời gian cần cho việc tìm kiếm bảng khi có một tham chiếu

Thuviendientu.org

xảy ra. Vì bảng trang đảo được lưu bởi địa chỉ vật lý nhưng tìm kiếm xảy ra trên địa chỉ ảo, toàn bộ bảng trang có thể cần được tìm kiếm sự trùng khớp. Sự tìm kiếm này có thể mất thời gian quá dài. Để làm giảm vấn đề này, chúng ta sử dụng một bảng băm được mô tả trong hình dưới đây để giới hạn việc tìm kiếm. Dĩ nhiên, mỗi truy xuất tới bảng băm thêm một tham chiếu tới thủ tục, để mà một tham chiếu bộ nhớ ảo yêu cầu ít nhất hai thao tác đọc bộ nhớ thật: một cho mục từ bảng băm và một cho bảng trang. Để cải tiến năng lực thực hiện, TLB được tìm kiếm đầu tiên, trước khi bảng băm được tra cứu.

VI.1.5 Trang được chia sẻ

Một thuận lợi khác của phân trang là khả năng chia sẻ mã chung. Việc xem xét này đặc biệt quan trọng trong môi trường chia thời. Xét một hệ thống hỗ trợ 40 người dùng, mỗi người dùng thực thi một trình soạn thảo văn bản. Nếu trình soạn thảo văn bản chứa 150 KB mã và 50 KB dữ liệu, chúng ta sẽ cần 8000 KB để hỗ trợ 40 người dùng. Tuy nhiên, nếu mã là mã tái sử dụng (reentrant code), nó có thể được chia sẻ như được hiển thị trong hình VII-22. Ở đây chúng ta thấy một bộ soạn thảo ba trang-mỗi trang có kích thước 50 KB; kích thước trang lớn được dùng để đơn giản hoá hình này-đang được chia sẻ giữa ba quá trình. Mỗi quá trình có trang dữ liệu riêng của nó.

Mã tái sử dụng (hay thuần mã-pure code) là mã không thay đổi bởi chính nó. Nếu mã là tái sử dụng thì nó không bao giờ thay đổi trong quá trình thực thi. Do đó, hai hay nhiều quá trình có thể thực thi cùng mã tại cùng thời điểm. Mỗi quá trình có bản sao thanh ghi của chính nó và lưu trữ dữ liệu để quản lý dữ liệu cho việc thực thi của quá trình. Dĩ nhiên, dữ liệu cho hai quá trình khác nhau sẽ khác nhau cho mỗi quá trình.

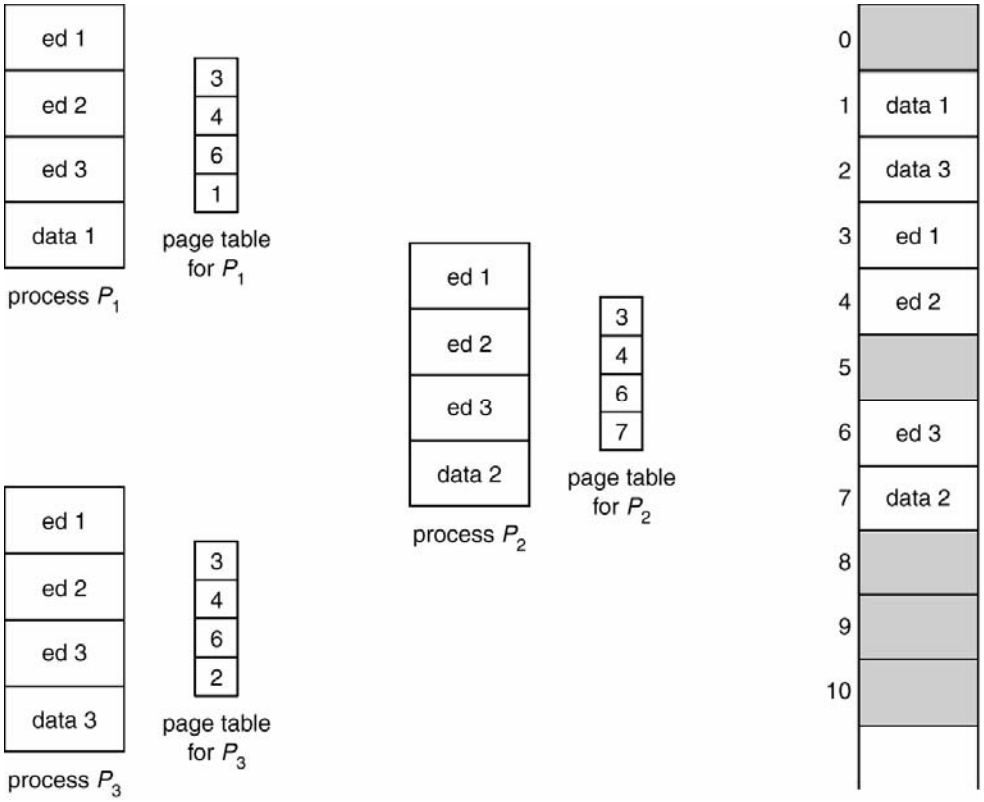
Chỉ một bản sao của bộ soạn thảo cần được giữ trong bộ nhớ vật lý. Mỗi bảng trang của người dùng ánh xạ tới cùng bản sao vật lý của bộ soạn thảo nhưng các trang dữ liệu được ánh xạ tới các khung khác nhau. Do đó, để hỗ trợ 40 người dùng, chúng ta cần chỉ một bản sao của bộ soạn thảo (150 KB) cộng với 40 bản sao của 50 KB không gian dữ liệu trên một người dùng. Bây giờ toàn bộ không gian được yêu cầu là 2150 KB thay vì 8000 KB-một tiết kiệm lớn.

Những chương trình được dùng nhiều khác cũng có thể được chia sẻ - trình biên dịch, hệ thống cửa sổ, thư viện thời điểm thực thi, hệ cơ sở dữ liệu,…Để có thể chia sẻ, mã phải được tái sử dụng. Tính tự nhiên chỉ đọc của mã được chia sẻ sẽ không được phó mặc cho tính đúng đắn của mã; hệ điều hành nên tuân theo thuộc tính này. Chia sẻ bộ nhớ giữa các quá trình trên hệ điều hành tương tự chia sẻ không gian địa chỉ của một tác vụ bởi luồng. Ngoài ra, bộ nhớ được chia sẻ như một phương pháp giao tiếp liên quá trình. Một số hệ điều hành cài đặt bộ nhớ được chia sẻ dùng các trang được chia sẻ.

Hệ điều hành dùng bảng trang bên trong gặp khó khăn khi cài đặt bộ nhớ được chia sẻ. Bộ nhớ được chia sẻ thường được cài đặt như nhiều địa chỉ ảo (một địa chỉ cho mỗi quá trình chia sẻ bộ nhớ) mà chúng được ánh xạ tới một địa chỉ vật lý. Tuy nhiên, phương pháp chuẩn này không thể được dùng khi có chỉ một mục từ trang ảo cho mỗi trang vật lý vì thế một trang vật lý không thể có hai (hay nhiều) địa chỉ ảo được chia sẻ.

Tổ chức bộ nhớ dựa theo trang cung cấp nhiều lợi điểm khác để cho phép nhiều quá trình chia sẻ cùng trang vật lý.

Thuviendientu.org



Hình 0-22 chia sẻ mã trong môi trường phân trang

VI.2 Phân đoạn

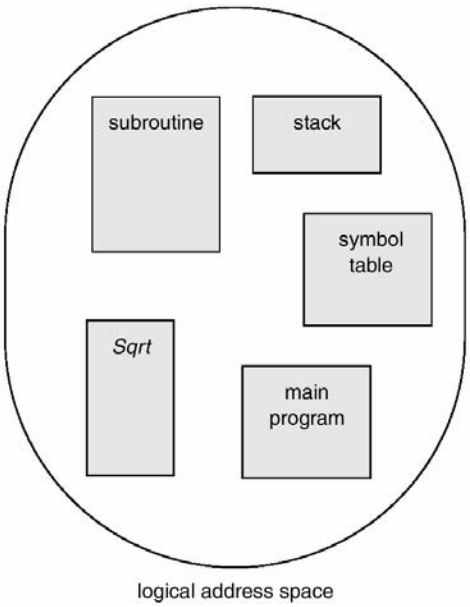
Một khía cạnh quan trọng của việc quản lý bộ nhớ mà trở nên không thể tránh với phân trang là ngăn cách tầm nhìn bộ nhớ của người dùng và bộ nhớ vật lý thật sự. Tầm nhìn bộ nhớ của người dùng không giống như bộ nhớ vật lý. Tầm nhìn người dùng được ánh xạ vào bộ nhớ vật lý. Việc ánh xạ cho phép sự khác nhau giữa bộ nhớ luận lý và bộ nhớ vật lý.

VI.2.1 Phương pháp cơ bản

Người dùng nghĩ bộ nhớ như mảng tuyến tính các byte, một số byte chứa chỉ thị lệnh, một số khác chứa dữ liệu hay không? Hầu hết mọi người nói không. Đúng hơn là, người dùng thích nhìn bộ nhớ như tập hợp các phân đoạn có kích thước thay đổi, và không cần xếp thứ tự giữa các phân đoạn (như hình VII-23).

Chúng ta nghĩ như thế nào về một chương trình khi chúng ta đang viết nó? Chúng ta nghĩ nó như một chương trình chính với một tập hợp các chương trình con, thủ tục, hàm, hay các module. Có thể có các cấu trúc dữ liệu khác nhau: bảng, mảng, ngăn xếp, biến,..Mỗi module hay thành phần dữ liệu này được tham chiếu bởi tên. Chúng ta nói "bảng danh biểu", "hàm sqrt", "chương trình chính" không quan tâm đến địa chỉ trong bộ nhớ mà những phần tử này chiếm. Chúng ta không quan tâm bảng danh biểu được lưu trữ trước hay sau hàm sqrt. Mỗi phân đoạn này có chiều dài thay đổi; thực chất chiều dài được định nghĩa bởi mục đích của phân đoạn trong chương trình. Các phần tử trong một phân đoạn được định nghĩa bởi độ dời của chúng từ điểm bắt đầu của phân đoạn: lệnh đầu tiên của chương trình, mục từ thứ mười bảy trong bảng danh biểu, chỉ thị thứ năm của hàm sqrt,…

Thuviendientu.org



Hình 0-23 Tầm nhìn chương trình của người dùng

Phân đoạn là một cơ chế quản lý bộ nhớ hỗ trợ tầm nhìn bộ nhớ của người dùng. Không gian địa chỉ luận lý là tập hợp các phân đoạn. Mỗi phân đoạn có tên và chiều dài. Các địa chỉ xác định tên phân đoạn và độ dời trong phân đoạn. Do đó, người dùng xác định mỗi địa chỉ bằng hai lượng: tên phân đoạn và độ dời. (tương phản cơ chế này với cơ chế phân trang, trong đó người dùng chỉ xác định một địa chỉ đơn, được chia bởi phần cứng thành số trang và độ dời, tất cả không thể nhìn thấy đối với người lập trình).

Để đơn giản việc cài đặt, các phân đoạn được đánh số và được tham chiếu tới bởi số phân đoạn, hơn là bởi tên phân đoạn. Do đó, địa chỉ luận lý chứa một bộ hai:

<số phân đoạn, độ dời>

Thông thường, chương trình người dùng được biên dịch, và trình biên dịch tự động tạo ra các phân đoạn phản ánh chương trình nhập. Một chương trình Pascal có thể tạo các phân đoạn riêng như sau:

1) Các biến toàn cục;
2) Ngăn xếp gọi thủ tục, để lưu trữ các tham số và trả về các địa chỉ;
3) Phần mã của mỗi thủ tục hay hàm;
4) Các biến cục bộ của mỗi thủ tục và hàm

Một trình biên dịch có thể tạo một phân đoạn riêng cho mỗi khối chung. Các mảng có thể được gán các phân đoạn riêng. Bộ nạp có thể mang tất cả phân đoạn này và gán chúng số phân đoạn.

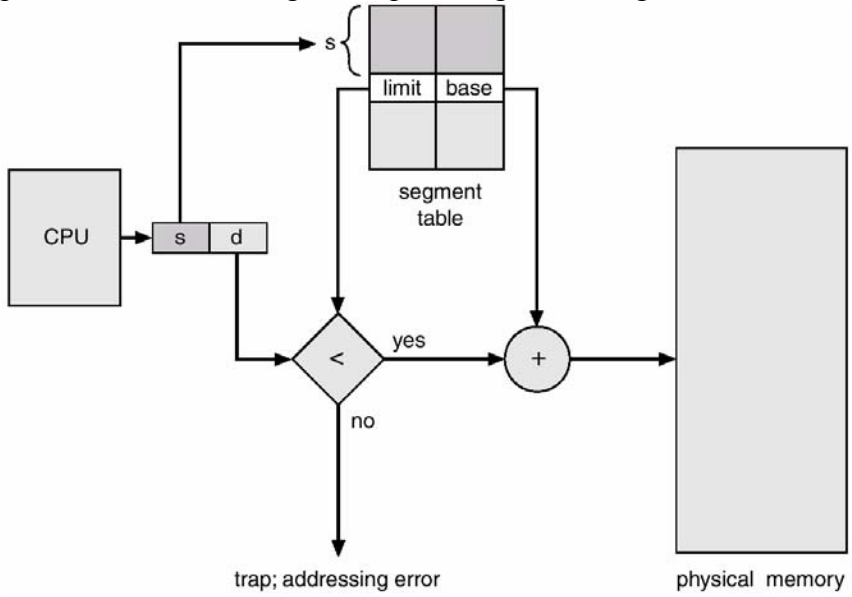
VI.2.2 Phần cứng

Mặc dù người dùng có thể tham chiếu tới các đối tượng trong chương trình bởi một địa chỉ hai chiều, bộ nhớ vật lý là chuỗi một chiều các byte. Do đó, chúng ta phải xác định việc cài đặt để ánh xạ địa chỉ hai chiều được định nghĩa bởi người dùng vào địa chỉ vật lý một chiều. Ánh xạ này được tác động bởi một bảng phân đoạn. Mỗi mục từ của bảng phân đoạn có một nền phân đoạn (segment base) và giới hạn phân đoạn

Thuviendientu.org

(segment limit). Nền phân đoạn chứa địa chỉ vật lý bắt đầu, nơi phân đoạn định vị trong bộ nhớ, ngược lại giới hạn phân đoạn xác định chiều dài của phân đoạn.

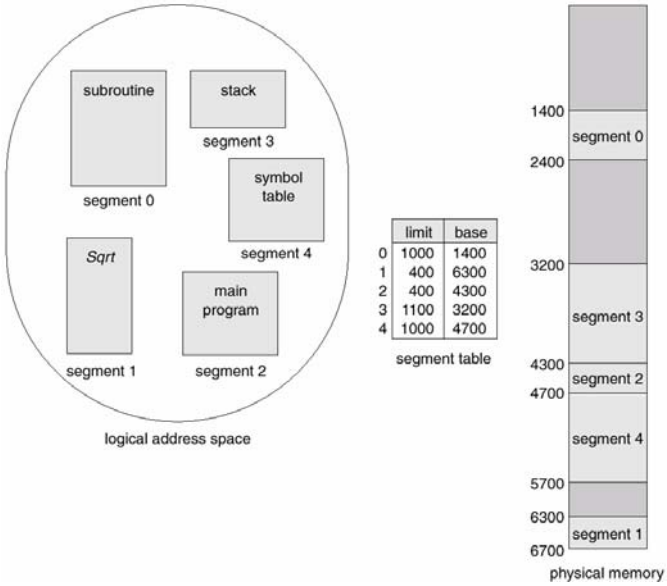
Sử dụng bảng phân đoạn được hiển thị như hình VII-24. Một địa chỉ luận lý có hai phần: số phân đoạn s và độ dời phân đoạn d. Số phân đoạn được dùng như chỉ mục trong bảng đoạn. Độ dời d của địa chỉ luận lý phải ở trong khoảng từ 0 tới giới hạn đoạn. Nếu không chúng ta sẽ trap tới hệ điều hành (địa chỉ vật lý vượt qua điểm cuối của phân đoạn). Nếu độ dời này là hợp lệ thì nó được cộng thêm giá trị nền của phân đoạn để tạo ra địa chỉ trong bộ nhớ vật lý của byte mong muốn. Do đó, bảng phân đoạn là một mảng của cặp thanh ghi nền và giới hạn.



Hình 0-24 Phần cứng phân đoạn

Xét trường hợp như hình VII-25. Chúng ta có năm phân đoạn được đánh số từ 0 đến 4. Các phân đoạn được lưu trong bộ nhớ vật lý như được hiển thị. Bảng phân đoạn có một mục từ riêng cho mỗi phân đoạn, cho địa chỉ bắt đầu của phân đoạn trong bộ nhớ vật lý (hay nền) và chiều dài của phân đoạn đó (hay giới hạn). Thí dụ, phân đoạn 2 dài 400 bytes và bắt đầu tại vị trí 4300. Do đó, một tham chiếu byte 53 của phân đoạn 2 được ánh xạ tới vị trí 4300 + 53 = 4353. Một tham chiếu tới phân đoạn 3, byte 852, được ánh xạ tới 3200 (giá trị nền của phân đoạn 3) +852=4052. Một tham chiếu tới byte 1222 của phân đoạn 0 dẫn đến một trap tới hệ điều hành, khi phân đoạn này chỉ dài 1000 bytes.

Thuviendientu.org



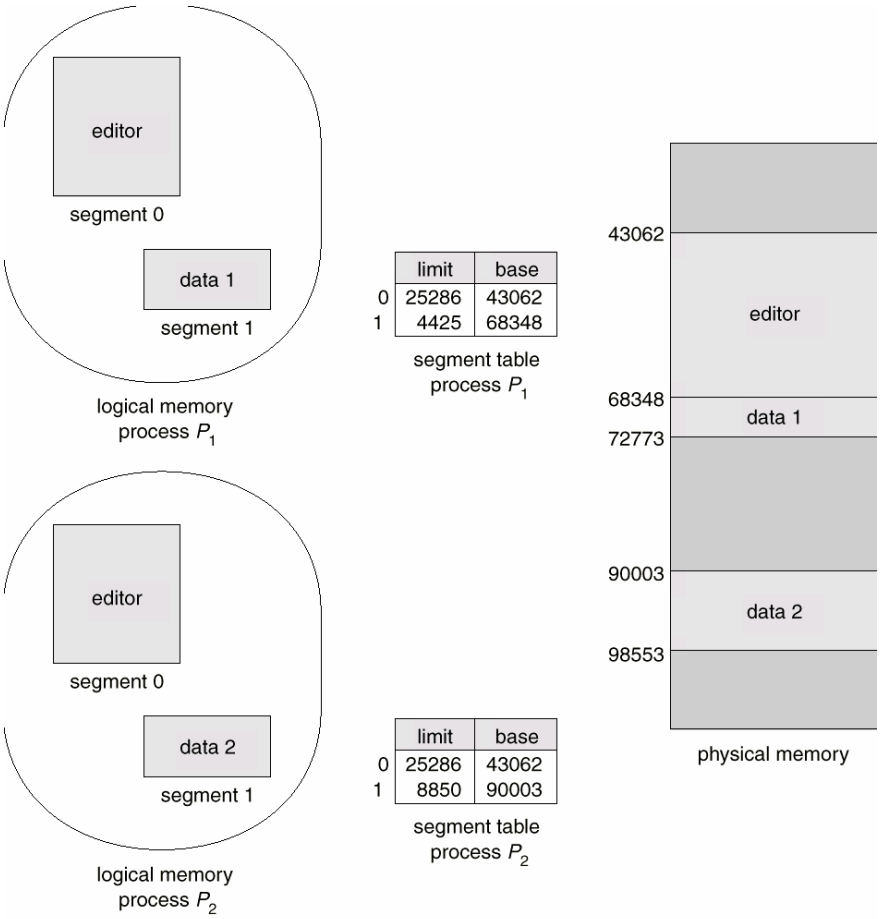
Hình 0-25 Thí dụ về phân đoạn

VI.2.3 Bảo vệ và chia sẻ

Lợi điểm đặc biệt của phân đoạn là sự gắn liền bảo vệ với các phân đoạn. Vì các phân đoạn biểu diễn một phần được định nghĩa của chương trình, tương tự như tất cả mục từ trong phân đoạn sẽ được dùng cùng một cách. Do đó, một số phân đoạn là chỉ thị, trong khi một số phân đoạn khác là dữ liệu. Trong một kiến trúc hiện đại, các chỉ thị không hiệu chỉnh chính nó vì thế các phân đoạn chỉ thị có thể được định nghĩa như chỉ đọc hay chỉ thực thi. Phần cứng ánh xạ bộ nhớ sẽ kiểm tra các bits bảo vệ được gắn với mỗi mục từ trong bảng phân đoạn để ngăn chặn các truy xuất không hợp lệ tới bộ nhớ, như cố gắng viết tới phân đoạn chỉ đọc hay sử dụng những phân đoạn chỉ đọc như dữ liệu. Bằng cách thay thế một mảng trong phân đoạn của chính nó, phần cứng quản lý bộ nhớ sẽ tự động kiểm tra các chỉ số của mảng là hợp lệ và không vượt ra ngoài giới hạn của mảng. Do đó, nhiều lỗi chương trình sẽ được phát hiện bởi phần cứng trước khi chúng có thể gây ra tác hại lớn.

Một lợi điểm khác liên quan đến chia sẻ mã hay dữ liệu. Mỗi quá trình có một bảng phân đoạn gắn với nó. Bộ phân phát dùng bảng phân đoạn này để định nghĩa phân đoạn phần cứng khi một quá trình được cấp CPU. Các phân đoạn được chia sẻ khi các mục từ trong bảng phân đoạn của hai quá trình khác nhau chỉ tới cùng một vị trí vật lý (như hình VII-26).

Thuviendientu.org



Hình 0-26 Chia sẻ các phân đoạn trong một hệ thống bộ nhớ được phân đoạn

Chia sẻ xảy ra tại cấp phân đoạn. Do đó, bất cứ thông tin có thể được chia sẻ nếu nó được định nghĩa là một phân đoạn. Một số phân đoạn có thể được chia sẻ vì thế một chương trình được hình thành từ nhiều phân đoạn có thể được chia sẻ.

Thí dụ, xét việc sử dụng một trình soạn thảo văn bản trong hệ thống chia thời. Trình soạn thảo hoàn chỉnh có thể rất lớn, được hình thành từ nhiều phân đoạn có thể được chia sẻ giữa tất cả người dùng, giới hạn địa chỉ vật lý được yêu cầu hỗ trợ các tác vụ soạn thảo. Thay vì n bản sao của trình soạn thảo, chúng ta chỉ cần một bản sao. Đối với mỗi người dùng, chúng ta vẫn cần các phân đoạn riêng, duy nhất để lưu các biến cục bộ. Dĩ nhiên, các phân đoạn này sẽ không được chia sẻ.

Chúng ta cũng có thể chia sẻ một số phần chương trình. Thí dụ, các gói chương trình con dùng chung có thể được chia sẻ giữa nhiều người dùng nếu chúng được định nghĩa như các phân đoạn chia sẻ, chỉ đọc. Thí dụ, hai chương trình Fortran có thể dùng cùng hàm Sqrt, nhưng chỉ một bản sao vật lý của hàm Sqrt được yêu cầu.

Mặc dù việc chia sẻ này có vẻ đơn giản, nhưng có những xem xét tinh tế. Điển hình, phân đoạn mã chứa các tham chiếu tới chính nó. Thí dụ, một lệnh nhảy (jump) có điều kiện thường có một địa chỉ chuyển gồm số phân đoạn và độ dời. Số phân đoạn của địa chỉ chuyển sẽ là số phân đoạn của phân đoạn mã. Nếu chúng ta cố gắng chia sẻ phân đoạn này, tất cả quá trình chia sẻ phải định nghĩa phân đoạn mã được chia sẻ để có cùng số phân đoạn.

Thí dụ, nếu chúng ta muốn chia sẻ hàm Sqrt và một quá trình muốn thực hiện nó phân đoạn 4 và một quá trình khác muốn thực hiện nó phân đoạn 17, hàm Sqrt nên tham chiếu tới chính nó như thế nào? Vì chỉ có một bản sao vật lý của Sqrt, nó phải được tham chiếu tới chính nó trong cùng cách cho cả hai người dùng-nó phải có một

Thuviendientu.org

số phân đoạn duy nhất. Khi số người dùng chia sẻ tăng do đó khó khăn trong việc tìm số phân đoạn có thể chấp nhận cũng tăng.

Các phân đoạn chỉ đọc không chứa con trỏ vật lý có thể được chia sẻ như số phân đoạn khác nhau, như các phân đoạn mã tham chiếu chính nó không trực tiếp. Thí dụ, các nhánh điều kiện xác định địa chỉ nhánh như một độ dời từ bộ đếm chương trình hiện hành hay quan hệ tới thanh ghi chứa số phân đoạn hiện hành nên cho phép mã tránh tham chiếu trực tiếp tới số phân đoạn hiện hành.

VI.2.4 Sự phân mãnh

Bộ định thời biểu dài phải tìm và cấp phát bộ nhớ cho tất cả các phân đoạn của chương trình người dùng. Trường hợp này tương tự như phân trang ngoại trừ các phân đoạn có chiều dài thay đổi; các trang có cùng kích thước. Do đó, với cơ chế phân khu có kích thước thay đổi, cấp phát bộ nhớ là một vấn đề cấp phát lưu trữ động, thường giải quyết với giải thuật best-fit hay first-fit.

Phân đoạn có thể gây ra sự phân mãnh, khi tất cả khối bộ nhớ trống là quá nhỏ để chứa một phân đoạn. Trong trường hợp này, quá trình có thể phải chờ cho đến khi nhiều bộ nhớ hơn (hay ít nhất một lỗ lớn hơn) trở nên sẵn dùng, hay cho tới khi việc hợp nhất các lỗ nhỏ để tạo một lỗ lớn hơn. Vì sự phân đoạn dùng giải thuật tái định vị động nên chúng ta có thể gom bộ nhớ bất cứ khi nào chúng ta muốn. Nếu bộ định thời biểu CPU phải chờ một quá trình vì vấn đề cấp phát bộ nhớ, nó có thể (hay không thể) bỏ qua hàng đợt CPU để tìm một quá trình nhỏ hơn, có độ ưu tiên thấp hơn để chạy.

Phân mãnh ngoài đối với cơ chế phân đoạn là vấn đề quan trọng như thế nào? Định thời biểu theo thuật ngữ dài với sự cô đặc sẽ giúp giải quyết vấn đề phân mãnh phải không? Câu trả lời phụ thuộc vào kích thước trung bình của phân đoạn. Ở mức độ cao nhất, chúng ta có thể định nghĩa mỗi quá trình là một phân đoạn. Tiếp cận này cắt giảm cơ chế phân khu có kích thước thay đổi. Ở cấp độ khác, mỗi byte có thể được đặt trong chính phân đoạn của nó và được cấp phát riêng. Sắp xếp này xoá đi việc phân mãnh bên ngoài; tuy nhiên mỗi byte cần một thanh ghi nền cho mỗi tái định vị của nó, gấp đôi bộ nhớ được dùng! Dĩ nhiên, bước luận lý tiếp theo-các phân đoạn nhỏ có kích thước cố định-là phân trang. Thông thường, nếu kích thước phân đoạn trung bình là nhỏ, phân mãnh ngoài cũng sẽ nhỏ. Vì cá nhân các phân đoạn là nhỏ hơn toàn bộ quá trình nên chúng có vẻ thích hợp hơn để đặt vào trong các khối bộ nhớ sẵn dùng.

VI.3 Phân đoạn với phân trang

Cả hai phân đoạn và phân trang có những lợi điểm và nhược điểm. Thật vậy, hai bộ vi xử lý phổ biến nhất hiện nay là: dòng Motorola 68000 được thiết kế dựa trên cơ sở không gian địa chỉ phẳng, ngược lại, họ Intel 80x86 và Petium dựa trên cơ sở phân đoạn. Cả hai là mô hình bộ nhớ hợp nhất hướng tới sự kết hợp của phân trang và phân đoạn. Chúng ta có thể kết hợp hai phương pháp để tận dụng lợi điểm của chúng. Sự kết hợp này được thể hiện tốt nhất bởi kết trúc của Intel 386.

Ấn bản IBM OS/2 32-bit là một hệ điều hành chạy trên đỉnh của kiến trúc Intel 386 (hay cao hơn). Intel 386 sử dụng phân đoạn với phân trang cho việc quản lý bộ nhớ. Số tối đa các phân đoạn trên quá trình là 16KB và mỗi phân đoạn có thể lớn tới 4GB. Kích thước trang là 4 KB. Chúng ta sẽ không cho một mô tả đầy đủ về cấu trúc quản lý bộ nhớ của Intel 386 trong giáo trình này. Thay vào đó, chúng ta sẽ trình bày các ý tưởng quan trọng.

Không gian địa chỉ luận lý của quá trình được chia thành hai phân khu. Phân khu thứ nhất chứa tới 8 KB các phân đoạn dành riêng cho quá trình đó. Phân khu thứ

Thuviendientu.org

hai chứa tới 8 KB các phân đoạn được chia sẻ giữa tất cả quá trình. Thông tin về phân khu thứ nhất được giữ trong bảng mô tả cục bộ (Local Descriptor Table-LDT), thông tin về phân khu thứ hai được giữ trong bảng mô tả toàn cục (Global Descriptor Table-GDL). Mỗi mục từ trong LDT và GDT chứa 8 bytes, với thông tin chi tiết về phân đoạn xác định gồm vị trí nền và chiều dài của phân đoạn đó.

Địa chỉ luận lý là một cặp (bộ chọn, độ dời), ở đây bộ chọn là một số 16-bit

S	G	P
13	1	2

Trong đó: s gán tới số phân đoạn, g hiển thị phân đoạn ở trong GDT hay LDT, và p giải quyết vấn đề bảo vệ. Độ dời là một số 32-bit xác định vị trí của byte (hay từ) trong phân đoạn.

Máy này có 6 thanh ghi, cho phép 6 phân đoạn được xác định tại bất cứ thời điểm nào bởi một quá trình. Nó có 6 thanh ghi vi chương trình 8-byte để quản lý bộ mô tả tương ứng từ LDT hay GDT. Bộ lưu trữ này để Intel 386 tránh phải đọc bộ mô tả từ bộ nhớ cho mỗi lần tham chiếu bộ nhớ.

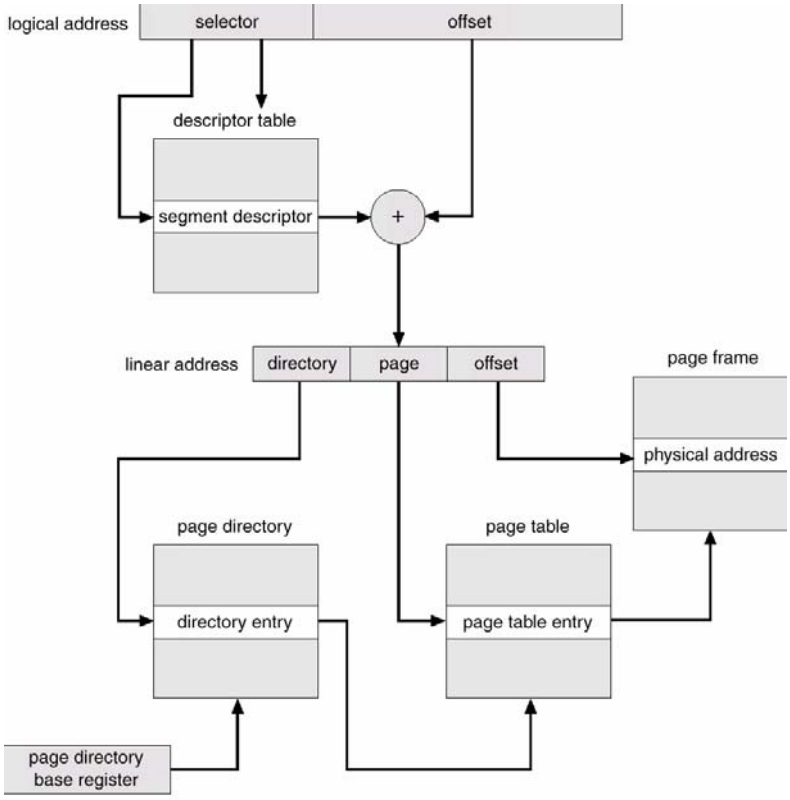
Địa chỉ vật lý trên 386 dài 32 bits và được hình thành như sau. Thanh ghi đoạn chỉ tới mục từ tương ứng trong LDT hay GDT. Thông tin nền và giới hạn về phân đoạn được dùng để phát sinh một địa chỉ tuyến tính. Đầu tiên, giới hạn được dùng để kiểm tra tính hợp lệ của địa chỉ. Nếu địa chỉ không hợp lệ, lỗi bộ nhớ được tạo ra, dẫn đến một trap tới hệ điều hành. Nếu nó là hợp lệ, thì giá trị của độ dời được cộng vào giá trị của nền, dẫn đến địa chỉ tuyến tính 32-bit. Sau đó, địa chỉ này được dịch thành địa chỉ vật lý.

Như được nêu trước đó, mỗi phân đoạn được phân trang và mỗi trang có kích thước 4 KB. Do đó, bảng trang có thể chứa tới 1 triệu mục từ. Vì mỗi mục từ chứa 4 bytes nên mỗi quá trình có thể cần 4 MB không gian địa chỉ vật lý cho một bảng trang. Rõ ràng, chúng ta không muốn cấp phát bảng trang liên tục trong bộ nhớ. Giải pháp này được thông qua trong Intel 386 để dùng cơ chế phân trang 2 cấp. Địa chỉ tuyến tính được chia thành số trang chứa 20 bits, và độ dời trang chứa 12 bits. Vì chúng ta phân trang bảng trang, số trang được chia nhỏ thành con trỏ thư mục trang 10-bit và con trỏ bảng trang 10-bit. Địa chỉ luận lý như sau:

P1	P2	D
10	10	12

Cơ chế dịch địa chỉ cho kiến trúc này tương tự như cơ chế được hiển thị trong hình VII-18. Dịch địa chỉ Intel được hiển thị chi tiết hơn trong hình VII-27dưới đây.

Thuviendientu.org



Hình 0-27 Dịch địa chỉ Intel 386

Để cải tiến tính hiệu quả của việc sử dụng bộ nhớ vật lý, bảng trang Intel 386 có thể được hoán vị tới đĩa. Trong trường hợp này, mỗi bit được dùng trong mục từ thư mục trang để hiển thị bảng mà mục từ đang chỉ tới ở trong bộ nhớ hay trên đĩa. Nếu bảng ở trên đĩa, hệ điều hành có thể dùng 31 bit còn lại để xác định vị trí đĩa của bảng; sau đó bảng có thể được mang vào bộ nhớ theo yêu cầu.

VII Tóm tắt

Các giải thuật quản lý bộ nhớ cho hệ điều hành đa chương trải dài từ tiếp cận hệ thống người dùng đơn tới phân đoạn được phân trang. Yếu tố quyết định lớn nhất của phương pháp được dùng trong hệ thống cụ thể là phần cứng được cải thiện. Mỗi địa chỉ bộ nhớ được được tạo ra bởi CPU phải được kiểm tra hợp lệ và có thể được ánh xạ tới một địa chỉ vật lý. Kiểm tra không thể được cài đặt (hữu hiệu) bằng phần mềm. Do đó, chúng ta bị ràng buộc bởi tính sẵn dùng phần cứng.

Các giải thuật quản lý bộ nhớ được thảo luận (cấp phát liên tục, phân trang, phân đoạn, và sự kết hợp của phân trang và phân đoạn) khác nhau trong nhiều khía cạnh. Trong so sánh các chiến lược quản lý bộ nhớ, chúng ta nên sử dụng các xem xét sau:

- **Hỗ trợ phần cứng**: thanh ghi nền hay cặp thanh ghi nền và thanh ghi giới hạn là đủ cho cơ chế phân khu đơn và đa, ngược lại phân trang và phân đoạn cần bảng ánh xạ để xác định ánh xạ địa chỉ.
- **Năng lực**: khi giải thuật quản lý bộ nhớ trở nên phức tạp hơn thì thời gian được yêu cầu để ánh xạ địa chỉ luận lý tới địa chỉ vật lý tăng. Đối với các hệ thống đơn giản, chúng ta chỉ cần so sánh hay cộng địa chỉ luận lý-các thao tác này phải nhanh. Phân trang và phân đoạn có thể nhanh như nếu

Thuviendientu.org

bảng được cài đặt trong các thanh ghi nhanh. Tuy nhiên, nếu bảng ở trong bộ nhớ thì về thực chất truy xuất bộ nhớ của người dùng có thể bị giảm. Một TLB có thể hạn chế việc giảm năng lực tới mức có thể chấp nhận.

- **Phân mãnh**: một hệ thống đa chương sẽ thực hiện hiệu quả hơn nếu nó có cấp độ đa chương cao hơn. Đối với một tập hợp quá trình được cho, chúng ta có thể tăng cấp độ đa chương chỉ bằng cách đóng gói nhiều quá trình hơn trong bộ nhớ. Để hoàn thành tác vụ này, chúng ta phải giảm sự lãng phí hay phân mãnh bộ nhớ. Các hệ thống với các đơn vị cấp phát có kích thước cố định, như cơ chế đơn phân khu và phân trang, gặp phải sự phân mãnh trong. Các hệ thống với đơn vị cấp phát có kích thước thay đổi như cơ chế đa phân khu và phân đoạn, gặp phải sự phân mãnh ngoài.
- **Tái định vị**: một giải pháp cho vấn đề phân mãnh ngoài là cô đặc. Cô đặc liên quan đến việc chuyển dịch một chương trình trong bộ nhớ không chú ý những thay đổi của chương trình. Xem xét này yêu cầu địa chỉ luận lý được tái định vị động tại thời điểm thực thi. Nếu địa chỉ được tái định vị chỉ tại thời điểm nạp, chúng ta không thể lưu trữ dạng cô đặc.
- **Hoán vị**: bất cứ giải thuật có thể có hoán vị được thêm tới nó. Tại những khoảng thời gian được xác định bởi hệ điều hành, thường được mô tả bởi các chính xác định thời, các quá trình được chép từ bộ nhớ chính tới vùng lưu trữ phụ và sau đó được chép trở lại tới bộ nhớ chính. Cơ chế này cho phép nhiều quá trình được chạy hơn là có thể đặt vào bộ nhớ tại cùng một thời điểm.
- **Chia sẻ**: một phương tiện khác để gia tăng cấp độ đa chương là chia sẻ mã và dữ liệu giữa các người dùng khác nhau. Thường việc chia sẻ yêu cầu phân trang hay phân đoạn được dùng, để cung cấp những gói thông tin nhỏ (các trang hay các đoạn) có thể được chia sẻ. Chia sẻ là một phương tiện chạy nhiều quá trình với lượng bộ nhớ giới hạn nhưng các chương trình và dữ liệu được chia sẻ phải được thiết kế cẩn thận.
- **Bảo vệ**: nếu phân trang hay phân đoạn được cung cấp, các phần khác nhau của chương trình người dùng có thể được khai báo chỉ thực thi, chỉ đọc, hay đọc-viết. Sự hạn chế này là cần thiết với mã và dữ liệu được chia sẻ và thường có ích trong bất cứ trường hợp nào để cung cấp việc kiểm tra tại thời gian thực thi cho các lỗi lập trình thông thường.

Thuviendientu.org

BỘ NHỚ ẢO

I Mục đích

Sau khi học xong chương này, người học nắm được những kiến thức sau:

- Hiểu kỹ thuật bộ nhớ ảo
- Hiểu bộ nhớ ảo ở dạng phân trang theo yêu cầu
- Hiểu độ phức tạp và chi phí trong từng kỹ thuật để cài đặt bộ nhớ ảo

II Giới thiệu

Trong chương trước, chúng ta thảo luận các chiến lược quản lý bộ nhớ được dùng trong hệ thống máy tính. Tất cả những chiến lược này có cùng mục đích: giữ nhiều quá trình trong bộ nhớ cùng một lúc để cho phép đa chương. Tuy nhiên, chúng có khuynh hướng yêu cầu toàn bộ quá trình ở trong bộ nhớ trước khi quá trình có thể thực thi.

Bộ nhớ ảo là một kỹ thuật cho phép việc thực thi của quá trình mà quá trình có thể không hoàn toàn ở trong bộ nhớ. Một lợi điểm quan trọng của cơ chế này là các chương trình có thể lớn hơn bộ nhớ vật lý. Ngoài ra, bộ nhớ ảo phóng đại bộ nhớ chính thành bộ nhớ luận lý cực lớn khi được hiển thị bởi người dùng. Kỹ thuật này giải phóng người lập trình từ việc quan tâm đến giới hạn kích thước bộ nhớ. Bộ nhớ ảo cũng cho phép các quá trình dễ dàng chia sẻ tập tin và không gian địa chỉ, cung cấp cơ chế hữu hiện cho việc tạo quá trình.

Tuy nhiên, bộ nhớ ảo không dễ cài đặt và về thực chất có thể giảm năng lực nếu nó được dùng thiếu thận trọng. Trong chương này, chúng ta thảo luận bộ nhớ ảo trong dạng phân trang theo yêu cầu và xem xét độ phức tạp và chi phí.

III Kiến thức nền

Các giải thuật quản lý bộ nhớ trong chương trước là cần thiết vì một yêu cầu cơ bản: các chỉ thị đang được thực thi phải ở trong bộ nhớ vật lý. Tiếp cận đầu tiên để thoả mãn yêu cầu này đặt toàn bộ không gian địa chỉ luận lý trong bộ nhớ vật lý. Phủ lắp và nạp động có thể giúp làm giảm hạn chế này nhưng chúng thường yêu cầu sự đề phòng đặc biệt và công việc phụ thêm bởi người lập trình. Hạn chế này dường như cần thiết và phù hợp nhưng nó không may mắn vì nó giới hạn kích thước của một chương trình đối với kích thước bộ nhớ vật lý.

Thật vậy, xem xét các chương trình thực thi chúng ta nhận thấy rằng trong nhiều trường hợp toàn bộ chương trình là không cần thiết. Thậm chí trong những trường hợp toàn bộ chương trình được yêu cầu nhưng không phải tất cả chương trình được yêu cầu cùng một lúc.

Khả năng thực thi chương trình chỉ một phần chương trình ở trong bộ nhớ có nhiều lợi điểm:

- Chương trình sẽ không còn bị ràng buộc bởi không gian bộ nhớ vật lý sẵn có. Người dùng có thể viết chương trình có không gian địa chỉ ảo rất lớn, đơn giản hoá tác vụ lập trình.
- Vì mỗi chương trình người dùng có thể lấy ít hơn bộ nhớ vật lý nên nhiều chương trình hơn có thể được thực thi tại một thời điểm. Điều này giúp gia

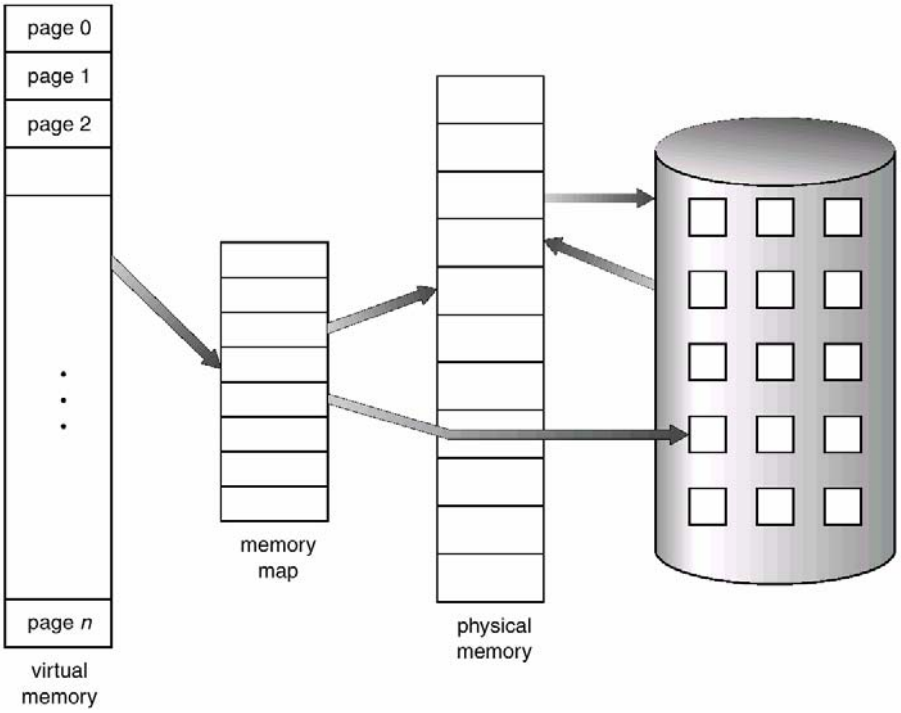
Thuviendientu.org

tăng việc sử dụng CPU và thông lượng nhưng không tăng thời gian đáp ứng.

- Yêu cầu ít nhập/xuất hơn để nạp hay hoán vị mỗi chương trình người dùng trong bộ nhớ vì thế mỗi chương trình người dùng sẽ chạy nhanh hơn.

Do đó, chạy một chương trình mà nó không nằm hoàn toàn trong bộ nhớ có lợi cho cả người dùng và hệ thống.

Bộ nhớ ảo là sự tách biệt bộ nhớ luận lý từ bộ nhớ vật lý. Việc tách biệt này cho phép bộ nhớ ảo rất lớn được cung cấp cho người lập trình khi chỉ bộ nhớ vật lý nhỏ hơn là sẵn dùng (hình VIII-1). Bộ nhớ ảo thực hiện tác vụ lập trình dễ hơn nhiều vì người lập trình không cần lo lắng về lượng bộ nhớ vật lý sẵn có nữa hay về mã gì có thể được thay thế trong việc phủ lắp; thay vào đó, người lập trình có thể quan tâm vấn đề được lập trình. Trên những hệ thống hỗ trợ bộ nhớ ảo, việc phủ lắp hầu như biến mất.



Hình 0-1 Lưu đồ minh hoạ bộ nhớ ảo lơn hơn bộ nhớ vật lý

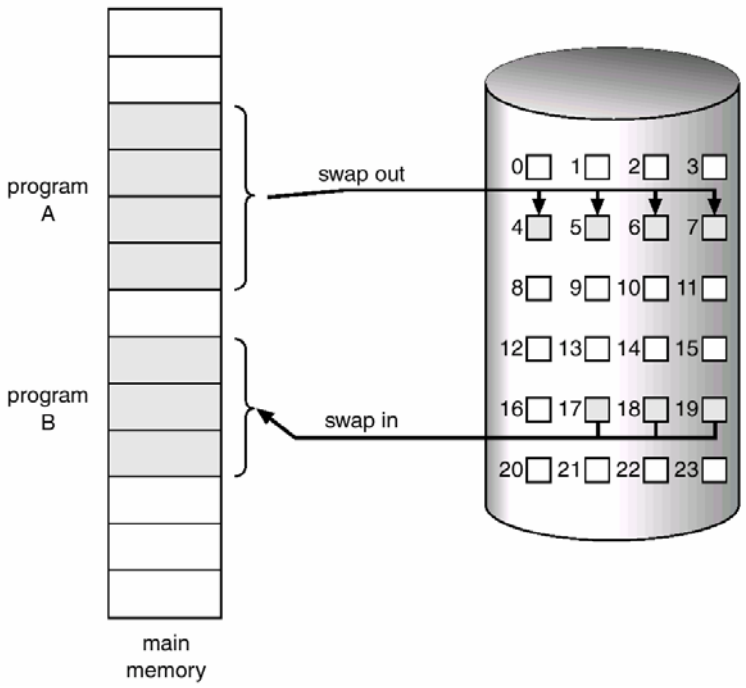
Thêm vào đó, việc tách biệt bộ nhớ luận lý từ bộ nhớ vật lý, bộ nhớ ảo cũng cho phép các tập tin và bộ nhớ được chia sẻ bởi những quá trình khác nhau thông qua việc chia sẻ trang. Ngoài ra, chia sẻ trang cho phép cải tiến năng lực trong khi tạo quá trình.

Bộ nhớ ảo thường được cài đặt bởi **phân trang theo yêu cầu** (demand paging). Nó cũng có thể được cài đặt trong cơ chế phân đoạn. Một vài hệ thống cung cấp cơ chế phân đoạn được phân trang. Trong cơ chế này các phân đoạn được chia thành các trang. Do đó, tầm nhìn người dùng là phân đoạn, nhưng hệ điều hành có thể cài đặt tầm nhìn này với cơ chế phân trang theo yêu cầu. Phân đoạn theo yêu cầu cũng có thể được dùng để cung cấp bộ nhớ ảo. Các hệ thống máy tính của Burrough dùng phân đoạn theo yêu cầu. Tuy nhiên, các giải thuật thay thế đoạn phức tạp hơn các giải thuật thay thế trang vì các đoạn có kích thước thay đổi. Chúng ta không đề cập phân đoạn theo yêu cầu trong giáo trình này.

Thuviendientu.org

IV Phân trang theo yêu cầu

Một hệ thống phân trang theo yêu cầu tương tự một hệ thống phân trang với hoán vị (hình VIII-2). Các quá trình định vị trong bộ nhớ phụ (thường là đĩa). Khi chúng ta muốn thực thi một quá trình, chúng ta hoán vị nó vào bộ nhớ. Tuy nhiên, thay vì hoán vị toàn bộ quá trình ở trong bộ nhớ, chúng ta dùng một **bộ hoán vị lười** (lazy swapper). Bộ hoán vị lười không bao giờ hoán vị một trang vào trong bộ nhớ trừ khi trang đó sẽ được yêu cầu. Vì bây giờ chúng ta xem một quá trình như một chuỗi các trang hơn là một không gian địa chỉ liên tục có kích thước lớn, nên dùng hoán vị là không phù hợp về kỹ thuật. Một bộ hoán vị thao tác toàn bộ quá trình, ngược lại một **bộ phân trang** (pager) được quan tâm với các trang riêng rẻ của một quá trình. Do đó, chúng ta dùng bộ phân trang (hơn là bộ hoán vị) trong nối kết với phân trang theo yêu cầu.

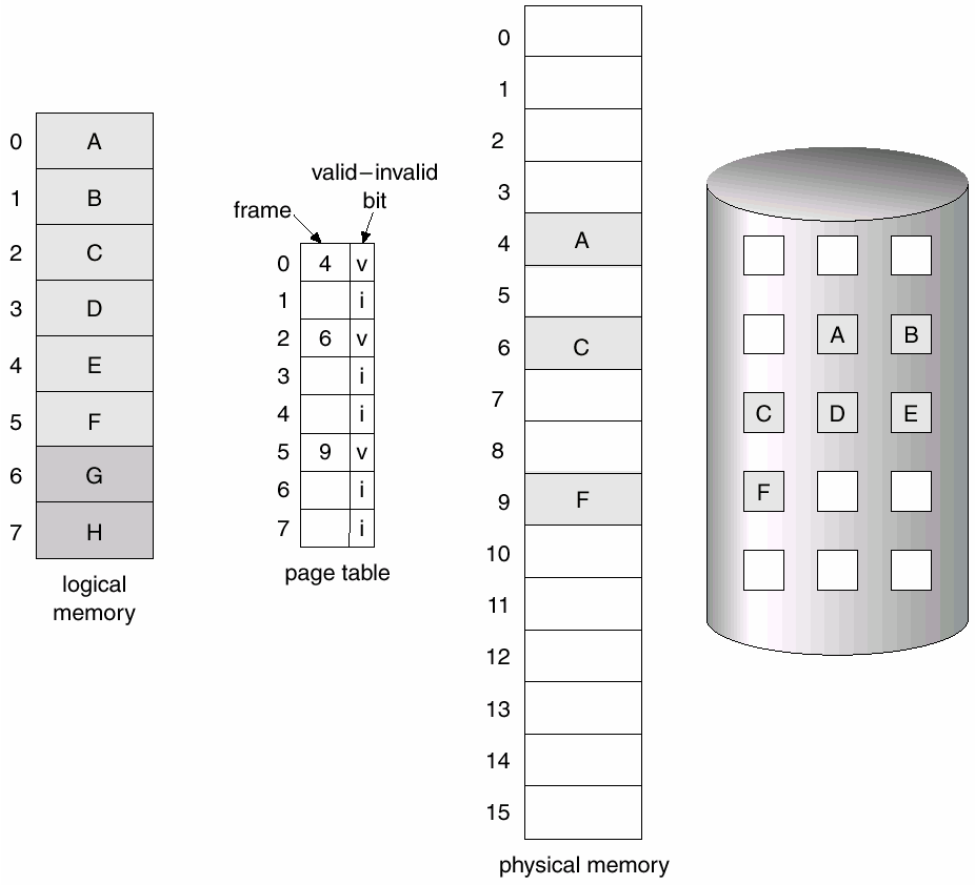


Hình 0-2 Chuyển bộ nhớ được phân trang tới không gian đĩa liên tục

IV.1 Các khái niệm cơ bản

Với cơ chế này, chúng ta cần một số dạng phần cứng hỗ trợ để phân biệt giữa các trang ở trong bộ nhớ và các trang ở trên đĩa. Cơ chế bit hợp lệ-không hợp lệ có thể được dùng cho mục đích này. Tuy nhiên, thời điểm này khi bit được đặt "hợp lệ", giá trị này hiển thị rằng trang được tham chiếu tới là hợp lệ và ở đang trong bộ nhớ. Nếu một bit được đặt "không hợp lệ", giá trị này hiển thị rằng trang không hợp lệ (nghĩa là trang không ở trong không gian địa chỉ của quá trình) hoặc hợp lệ nhưng hiện đang ở trên đĩa. Mục từ bảng trang cho trang không ở trong bộ nhớ đơn giản được đánh dấu không hợp lệ, hay chứa địa chỉ của trang trên đĩa. Trường hợp này được mô tả trong hình VIII-3.

Thuviendientu.org



Hình 0-3 Bảng trang khi một số trang không ở trong bộ nhớ chính

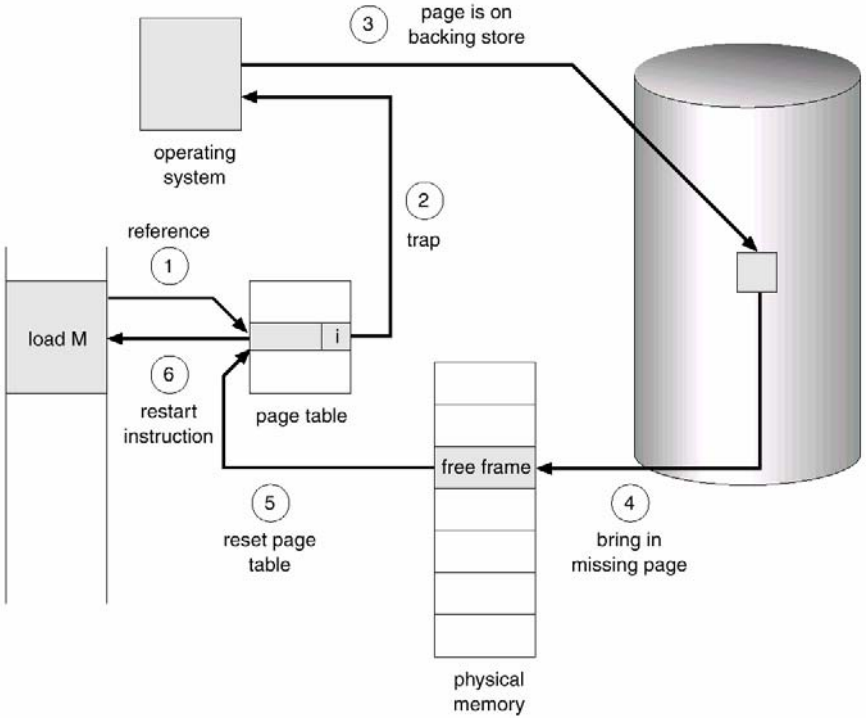
Chú ý rằng, đánh dấu một trang là "không hợp lệ" sẽ không có tác dụng nếu quá trình không bao giờ truy xuất trang đó. Do đó, nếu chúng ta đoán đúng và tất cả những trang thật sự cần đều ở trong bộ nhớ, quá trình sẽ chạy chính xác như khi chúng ta mang tất cả trang vào bộ nhớ. Trong khi quá trình thực thi và truy xuất trang đang định vị trong bộ nhớ, việc thực thi xử lý bình thường.

Nhưng điều gì xảy ra nếu quá trình cố gắng truy xuất trang mà trang đó không được mang vào bộ nhớ? Truy xuất một trang được đánh dấu là "không hợp lệ" gây ra một **trap lỗi trang** (page-fault trap). Phần cứng phân trang, dịch địa chỉ thông qua bảng trang, sẽ thông báo rằng bit không hợp lệ được đặt, gây ra một trap tới hệ điều hành. Trap này là kết quả lỗi của hệ điều hành mang trang được mong muốn vào bộ nhớ (trong một cố gắng tối thiểu chi phí chuyển đĩa và yêu cầu bộ nhớ) hơn là lỗi địa chỉ không hợp lệ như kết quả của việc cố gắng dùng một địa chỉ bộ nhớ không hợp lệ (như một ký hiệu mảng không hợp lệ). Do đó, chúng ta phải sửa trường hợp sơ xuất này. Thủ tục cho việc quản lý lỗi trang này là không phức tạp (hình VIII-4).

1) Chúng ta kiểm tra bảng bên trong (thường được giữ với khối điều khiển quá trình) cho quá trình này, để xác định tham chiếu là truy xuất bộ nhớ hợp lệ hay không hợp lệ.
2) Nếu tham chiếu là không hợp lệ, chúng ta kết thúc quá trình. Nếu nó là hợp lệ, nhưng chúng ta chưa mang trang đó vào bộ nhớ, bây giờ chúng ta mang trang đó vào.
3) Chúng ta tìm khung trống (thí dụ, bằng cách mang một trang từ danh sách khung trống).

Thuviendientu.org

4) Chúng ta lập thời biểu thao tác đĩa để đọc trang mong muốn vào khung trang vừa mới được cấp phát.
5) Khi đọc đĩa hoàn thành, chúng ta sửa đổi bảng bên trong với quá trình và bảng trang để hiển thị rằng trang bây giờ ở trong bộ nhớ.
6) Chúng ta khởi động lại chỉ thị mà nó bị ngắt bởi trap địa chỉ không hợp lệ. Bây giờ quá trình có thể truy xuất trang mặc dù nó luôn ở trong bộ nhớ.



Hình 0-4 Các bước quản lý lỗi trang

Vì chúng ta lưu trạng thái (thanh ghi, mã điều kiện, bộ đếm chỉ thị lệnh) của quá trình bị ngắt khi lỗi trang xảy ra, nên chúng ta có thể khởi động lại quá trình chính xác nơi và trạng thái, ngoại trừ trang mong muốn hiện ở trong bộ nhớ và có thể truy xuất. Trong cách này, chúng ta có thể thực thi một quá trình mặc dù các phần của nó chưa ở trong bộ nhớ. Khi quá trình cố gắng truy xuất các vị trí không ở trong bộ nhớ, phần cứng trap tới hệ điều hành (lỗi trang). Hệ điều hành đọc trang được yêu cầu vào bộ nhớ và khởi động lại quá trình như thể trang luôn ở trong bộ nhớ.

Trong trường hợp xấu nhất, chúng ta bắt đầu thực thi một quá trình với không trang nào ở trong bộ nhớ. Khi hệ điều hành đặt con trỏ chỉ thị lệnh tới chỉ thị đầu tiên của quá trình. Tuy nhiên, chỉ thị này ở trên trang không nằm trong bộ nhớ, quá trình lập tức báo lỗi đối với trang đó. Sau khi trang được mang vào trong bộ nhớ, quá trình tiếp tục thực thi, báo lỗi khi cần cho tới khi mỗi trang nó cần ở trong bộ nhớ. Tại thời điểm đó, nó có thể thực thi với không có lỗi nào nữa. Cơ chế này là **thuần phân trang yêu cầu** (pure demand paging): không bao giờ mang trang vào bộ nhớ cho tới khi nó được yêu cầu.

Về lý thuyết, một số quá trình có thể truy xuất nhiều trang mới của bộ nhớ với mỗi sự thực thi chỉ thị (một trang cho một chỉ thị và nhiều trang cho dữ liệu), có thể gây ra lỗi nhiều trang trên chỉ thị. Trường hợp này sẽ dẫn đến năng lực thực hiện hệ thống không thể chấp nhận. May thay, phân tích các quá trình thực thi thể hiện rằng hành vi này là không hoàn toàn xảy ra. Các chương trình có khuynh hướng tham chiếu cục bộ dẫn đến năng lực phù hợp từ phân trang yêu cầu.

Phần cứng hỗ trợ phân trang theo yêu cầu là tương tự như phần cứng phân trang và hoán vị.

Thuviendientu.org

- **Bảng trang**: bảng này có khả năng đánh dấu mục từ không hợp lệ thông qua bit hợp lệ-không hợp lệ hay giá trị đặc biệt của các bit bảo vệ
- **Bộ nhớ phụ**: bộ nhớ này quản lý các trang không hiện diện trong bộ nhớ chính. Bộ nhớ phụ thường là đĩa tốc độ cao. Nó được xem như là thiết bị hoán vị và phần đĩa được dùng cho mục đích này được gọi là không gian hoán vị.

Ngoài sự hỗ trợ phần cứng này, phần mềm có thể xem xét được yêu cầu. Ràng buộc kiến trúc phải được áp đặt. Ràng buộc quan trọng được yêu cầu là có thể khởi động lại bất cứ chỉ thị nào sau khi lỗi trang. Trong hầu hết các trường hợp, yêu cầu này là dễ dàng thoả mãn. Lỗi trang có thể xảy ra tại bất cứ tham chiếu bộ nhớ nào. Nếu lỗi trang xảy ra trên việc lấy chỉ thị, chúng ta có thể khởi động lại bằng cách lấy lại chỉ thị. Nếu lỗi trang xảy ra trong khi chúng ta đang lấy một toán hạng, chúng ta phải lấy và giải mã lại chỉ thị, và sau đó lấy toán hạng.

IV.2 Năng lực của phân trang theo yêu cầu

Phân trang theo yêu cầu có thể có một ảnh hưởng lớn trên năng lực của một hệ thống máy tính. Để thấy tại sao, chúng ta tính **thời gian truy xuất hiệu quả** (effective access time) cho bộ nhớ được phân trang theo yêu cầu. Đối với hầu hết các hệ thống máy tính, thời gian truy xuất bộ nhớ, được ký hiệu *ma*, nằm trong khoảng từ 10 đến 200 nano giây. Với điều kiện là chúng ta không có lỗi trang, thời gian truy xuất hiệu quả là bằng với thời gian truy xuất bộ nhớ. Tuy nhiên, nếu lỗi trang xảy ra, trước hết chúng ta phải đọc trang tương ứng từ đĩa và sau đó truy xuất từ mong muốn.

Gọi p là xác suất của lỗi trang ($0 \le p \le 1$). Chúng ta mong đợi p gần bằng 0; nghĩa là chỉ có một vài lỗi trang. Thời gian truy xuất hiệu quả là:

Thời gian truy xuất hiệu quả = (1 – p) x *ma* + p x thời gian lỗi trang

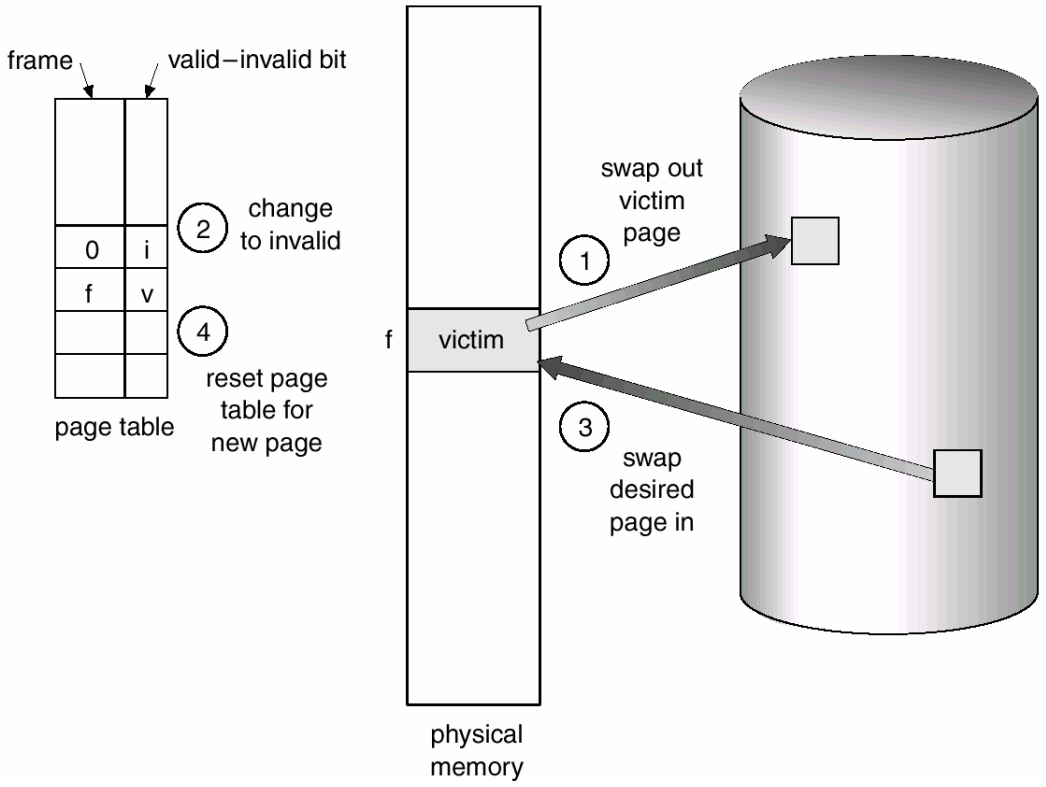
Để tính toán thời gian truy xuất hiệu quả, chúng ta phải biết phải mất bao lâu để phục vụ một lỗi trang. Để duy trì ở mức độ chấp nhận được sự chậm trễ trong hoạt động của hệ thống do phân trang, cần phải duy trì tỷ lệ phát sinh lỗi trang thấp.

V Thay thế trang

Thay thế trang thực hiện tiếp cận sau. Nếu không có khung trống, chúng ta tìm một khung hiện không được dùng và giải phóng nó. Khi chúng ta giải phóng một khung bằng cách viết nội dung của nó tới không gian hoán vị và thay đổi bảng trang (và các bảng trang khác) để hiển thị rằng trang không còn ở trong bộ nhớ (hình VIII-5). Bây giờ chúng ta có thể dùng khung được giải phóng để quản lý trang cho quá trình bị lỗi. Chúng ta sửa đổi thủ tục phục vụ lỗi trang để chứa thay thế trang:

1) Tìm vị trí trang mong muốn trên đĩa
2) Tìm khung trang trống
    a) Nếu có khung trống, dùng nó.
    b) Nếu không có khung trống, dùng một giải thuật thay thế trang để chọn khung “nạn nhân”
    c) Viết trang “nạn nhân” tới đĩa; thay đổi bảng trang và khung trang tương ứng.
3) Đọc trang mong muốn vào khung trang trống; thay đổi bảng trang và khung trang.
4) Khởi động lại quá trình.

Thuviendientu.org



Hình 0-5 Thay thế trang

Chúng ta phải giải quyết hai vấn đề chính để cài đặt phân trang theo yêu cầu: chúng ta phát triển giải thuật cấp phát khung và giải thuật thay thế trang. Nếu chúng ta có nhiều quá trình trong bộ nhớ, chúng ta phải quyết định bao nhiêu khung cấp phát tới quá trình. Ngoài ra, khi thay thế trang được yêu cầu, chúng ta phải chọn các khung để được thay thế. Thiết kế các giải thuật hợp lý để giải quyết vấn đề này là một tác vụ quan trọng vì nhập/xuất đĩa là rất đắt. Thậm chí một cải tiến nhỏ trong các phương pháp phân trang theo yêu cầu sinh ra một lượng lớn năng lực hệ thống.

Có nhiều giải thuật thay thế trang khác nhau. Mỗi hệ điều hành có thể có cơ chế thay thế của chính nó. Chúng ta chọn một giải thuật thay thế trang như thế nào? Thông thường, chúng ta muốn một giải thuật tỉ lệ lỗi trang nhỏ nhất.

Chúng ta đánh giá một giải thuật bằng cách chạy nó trên một chuỗi các tham chiếu bộ nhớ cụ thể và tính số lượng lỗi trang. Chuỗi các tham chiếu bộ nhớ được gọi là chuỗi tham chiếu. Chúng ta có thể phát sinh chuỗi tham chiếu giả tạo (thí dụ, bằng bộ phát sinh số ngẫu nhiên). Chọn lựa sau đó tạo ra số lượng lớn dữ liệu (trên thứ tự 1 triệu địa chỉ trên giây). Để làm giảm số lượng dữ liệu này, chúng ta có hai cách

Cách thứ nhất, đối với kích thước trang được cho (và kích thước trang thường được cố định bởi phần cứng hay hệ thống), chúng ta cần xét chỉ số trang hơn là toàn địa chỉ. Cách thứ hai, nếu chúng ta có một tham chiếu tới trang p, thì bất cứ những tham chiếu tức thì theo sau tới trang p sẽ không bao giờ gây lỗi trang. Trang p sẽ ở trong bộ nhớ sau khi tham chiếu đầu tiên; các tham chiếu theo sau tức thì sẽ không bị lỗi.

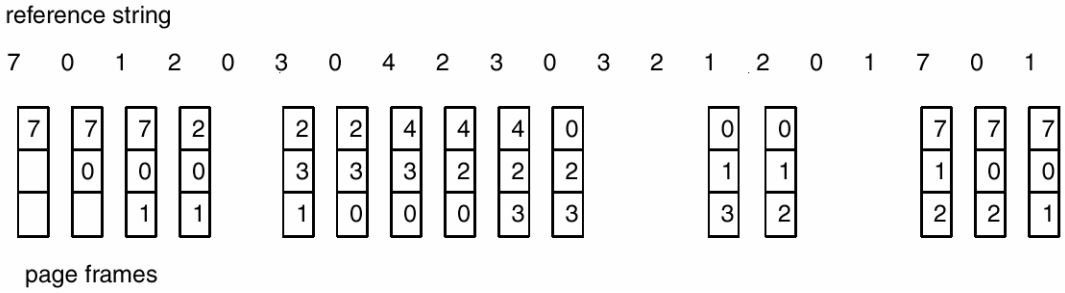
V.1 Thay thế trang FIFO

Giải thuật thay thế trang đơn giản nhất là giải thuật FIFO. Giải thuật này gắn với mỗi trang thời gian khi trang đó được mang vào trong bộ nhớ. Khi một trang phải

Thuviendientu.org

được thay thế, trang cũ nhất sẽ được chọn. Chú ý rằng, nó không yêu cầu nghiêm ngặt để ghi thời gian khi trang được mang vào. Chúng ta có thể tạo một hàng đợi FIFO để quản lý tất cả trang trong bộ nhớ. Chúng ta thay thế trang tại đầu hàng đợi. Khi trang được mang vào bộ nhớ, chúng ta chèn nó vào đuôi của hàng đợi.

Cho một thí dụ về chuỗi tham khảo, 3 khung của chúng ta ban đầu là rỗng. 3 tham khảo đầu tiên (7, 0, 1) gây ra lỗi trang và được mang vào các khung rỗng này. Tham khảo tiếp theo (2) thay thế trang 7, vì trang 7 được mang vào trước. Vì 0 là tham khảo tiếp theo và 0 đã ở trong bộ nhớ rồi, chúng ta không có lỗi trang cho tham khảo này. Tham khảo đầu tiên tới 3 dẫn đến trang 0 đang được thay thế vì thế nó là trang đầu tiên của 3 trang trong bộ nhớ (0, 1, 2) để được mang vào. Bởi vì thay thế này, tham khảo tiếp theo, tới 0, sẽ bị lỗi. Sau đó, trang 1 được thay thế bởi trang 0. Quá trình này tiếp tục như được hiển thị trong hình VIII-6. Mỗi khi một lỗi xảy ra, chúng ta hiển thị các trang ở trong 3 khung của chúng ta. Có 15 lỗi cả thảy.

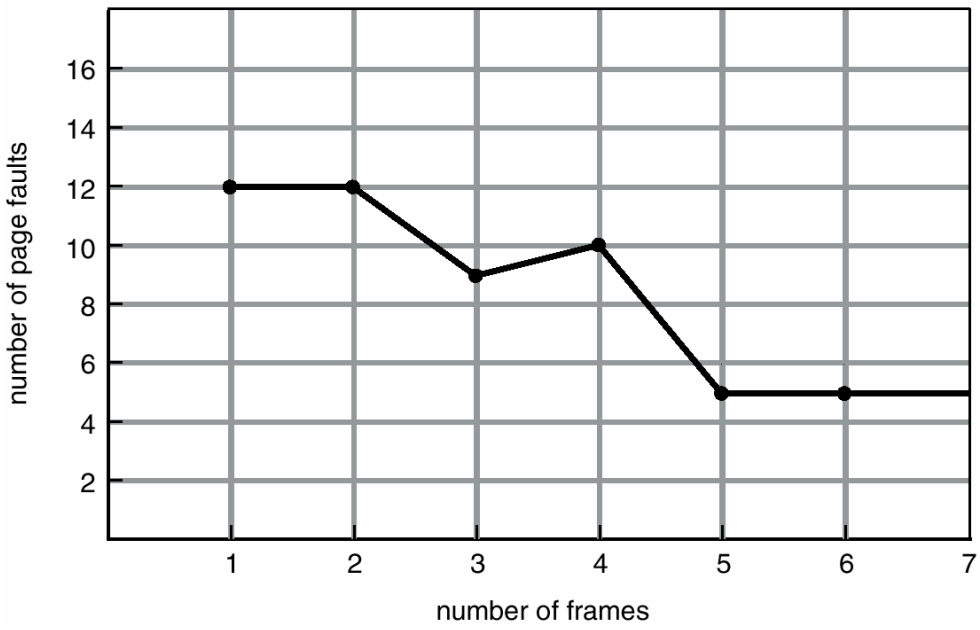


Hình 0-6 giải thuật thay thế trang FIFO

Giải thuật thay thế trang FIFO rất dễ hiểu và lập trình. Tuy nhiên, năng lực của nó không luôn tốt. Trang được cho để thay thế có thể là trang chức nhiều dữ liệu cần thiết, thường xuyên được sử dụng nên được nạp sớm, do vậy khi chuyển ra bộ nhớ phụ sẽ nhanh chóng gây ra lỗi trang.

Để hiển thị các vấn đề có thể phát sinh với giải thuật thay thế trang FIFO, chúng ta xem xét chuỗi tham khảo sau: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5. Hình VIII-7 hiển thị đường cong lỗi trang khi so sánh với số khung sẵn dùng. Chúng ta chú ý rằng số lượng lỗi cho 4 khung (10) là lớn hơn số lượng lỗi cho 3 khung (9). Hầu hết các kết quả không mong đợi này được gọi là sự nghịch lý Belady; đối với một số giải thuật thay thế trang, tỉ lệ lỗi trang có thể tăng khi số lượng khung được cấp phát tăng. Chúng ta sẽ mong muốn rằng cho nhiều bộ nhớ hơn tới một quá trình sẽ cải tiến năng lực của nó. Trong một vài nghiên cứu trước đây, các nhà điều tra đã kết luận rằng giả thuyết này không luôn đúng. Sự không bình thường của Belady được phát hiện như là một kết quả.

Thuviendientu.org



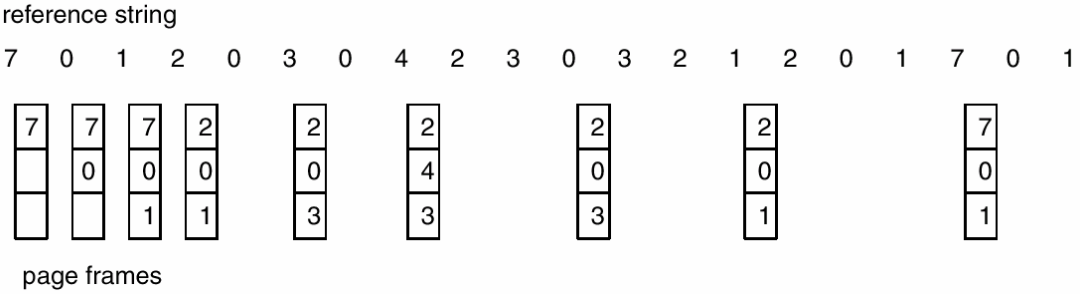
Hình 0-7 Đường cong lỗi trang cho thay thế FIFO trên chuỗi tham khảo

V.2 Thay thế trang tối ưu hoá

Kết quả phát hiện sự nghịch lý của Belady là tìm ra một giải thuật thay thế trang tối ưu. Giải thuật thay thế trang tối ưu có tỉ lệ lỗi trang thấp nhất trong tất cả các giải thuật và sẽ không bao giờ gặp phải sự nghịch lý của Belady. Giải thuật như thế tồn tại và được gọi là OPT hay MIN. Nó đơn giản là: thay thế trang mà nó không được dùng cho một khoảng thời gian lâu nhất. Sử dụng giải thuật thay thế trang đảm bảo tỉ lệ lỗi trang nhỏ nhất có thể cho một số lượng khung cố định.

Thí dụ, trên một chuỗi tham khảo mẫu, giải thuật thay thế trang tối ưu sẽ phát sinh 9 lỗi trang, như được hiển thị trong hình VIII-8. 3 tham khảo đầu tiên gây ra lỗi điền vào 3 khung trống. Tham khảo tới trang 2 thay thế trang 7 vì 7 sẽ không được dùng cho tới khi tham khảo 18, trái lại trang 0 sẽ được dùng tại 5 và trang 1 tại 14. Tham khảo tới trang 3 thay thế trang 1 khi trang 1 sẽ là trang cuối cùng của 3 trang trong bộ nhớ được tham khảo lần nữa. Với chỉ 9 lỗi trang, thay thế tối ưu là tốt hơn nhiều giải thuật FIFO, có 15 lỗi. (Nếu chúng ta bỏ qua 3 lỗi đầu mà tất cả giải thuật phải gặp thì thay thế tối ưu tốt gấp 2 lần thay thế FIFO.) Thật vậy, không có giải thuật thay thế nào có thể xử lý chuỗi tham khảo trong 3 khung với ít hơn 9 lỗi.

Tuy nhiên, giải thuật thay thế trang tối ưu là khó cài đặt vì nó yêu cầu kiến thức tương lai về chuỗi tham khảo. Do đó, giải thuật tối ưu được dùng chủ yếu cho nghiên cứu so sánh. Thí dụ, nó có thể có ích để biết rằng, mặc dù một giải thuật không tối ưu nhưng nó nằm trong 12.3% của tối ưu là tệ, và trong 4.7% là trung bình.

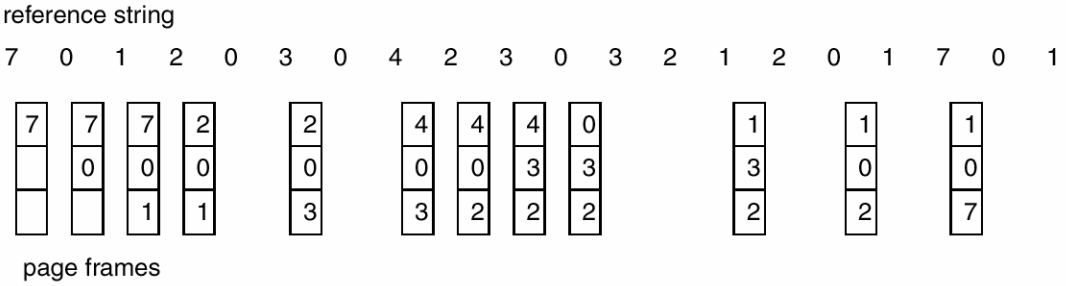


Hình 0-8 giải thuật thay thế trang tối ưu

Thuviendientu.org

V.3 Thay thế trang LRU

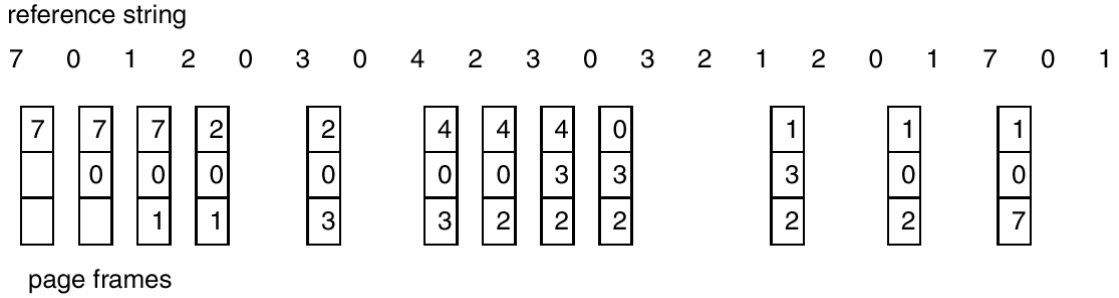
Nếu giải thuật tối ưu là không khả thi, có lẽ một xấp xỉ giải thuật tối ưu là có thể. Sự khác biệt chủ yếu giữa giải thuật FIFO và OPT là FIFO dùng thời gian khi trang được mang vào bộ nhớ; giải thuật OPT dùng thời gian khi trang được sử dụng. Nếu chúng ta sẽ dụng quá khứ gần đây như một xấp xỉ của tương lai gần thì chúng ta sẽ thay thế trang mà nó không được dùng cho khoảng thời gian lâu nhất (hình VIII-9). Tiếp cận này là giải thuật ít được dùng gần đây nhất (least-recently-used (LRU)).



Hình 0-9 giải thuật thay thế trang LRU

Thay thế trang LRU gắn với mỗi trang thời gian sử dụng cuối cùng của trang. Khi một trang phải được thay thế, LRU chọn trang không được dùng trong một khoảng thời gian lâu nhất. Chiến lược này là giải thuật thay thế trang tối ưu tìm kiếm lùi theo thời gian hơn là hướng tới. (gọi S^R là trình tự ngược của chuỗi tham khảo S thì tỉ lệ lỗi trang cho giải thuật OPT trên S là tương tự như tỉ lệ lỗi trang cho giải thuật OPT trên S^R. Tương tự, tỉ lệ lỗi trang đối với giải thuật LRU trên S là giống như tỉ lệ lỗi trang cho giải thuật LRU trên S^R)

Kết quả ứng dụng thay thế LRU đối với chuỗi tham khảo điển hình được hiển thị trong hình VIII-10. Giải thuật LRU sinh ra 12 lỗi. 5 lỗi đầu tiên là giống như thay thế tối ưu. Tuy nhiên, khi tham chiếu tới trang 4 xảy ra thay thế LRU thấy rằng 3 khung trong bộ nhớ, trang 2 được dùng gần đây nhất. Trang được dùng gần đây nhất là trang 0, và chỉ trước khi trang 3 được dùng. Do đó, giải thuật LRU thay thế trang 2, không biết rằng trang 2 để được dùng. Sau đó, khi nó gây lỗi trang 2, giải thuật LRU thay thế trang 3, của 3 trang trong bộ nhớ {0, 3, 4} trang 3 ít được sử dụng gần đây nhất. Mặc dù vấn đề này nhưng thay thế LRU với 12 lỗi vẫn tốt hơn thay thế FIFO với 15.



Hình 0-10 giải thuật thay thế trang

Chính sách LRU thường được dùng như giải thuật thay thế trang và được xem là tốt. Vấn đề chính là cách cài đặt thay thế LRU. Một giải thuật thay thế trang LRU có thể yêu cầu sự trợ giúp phần cứng. Vấn đề là xác định thứ tự cho các khung được định nghĩa bởi thời gian sử dụng gần nhất. Hai cách cài đặt khả thi là:

- **Bộ đếm**: trong trường hợp đơn giản nhất, chúng ta gắn mỗi mục từ bảng trang một trường số lần sử dụng và thêm CPU một đồng hồ luận lý hay bộ

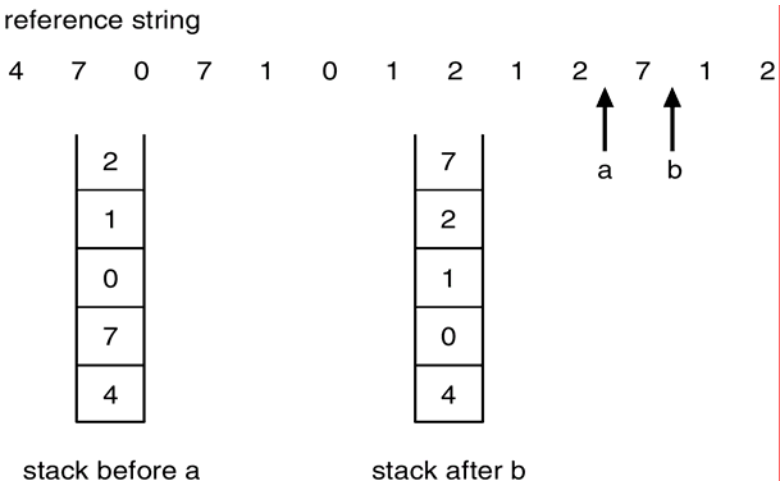
Thuviendientu.org

đếm. Đồng hồ được tăng cho mỗi tham khảo bộ nhớ. Bất cứ khi nào một tham khảo tới trang được thực hiện, các nội dung của thanh ghi đồng hồ được chép tới trường số lần sử dụng trong mục từ bảng trang cho trang đó. Trong cách này, chúng ta luôn có thời gian của tham khảo cuối cùng tới mỗi trang. Chúng ta thay thế trang với giá trị số lần sử dụng nhỏ nhất. Cơ chế này yêu cầu tìm kiếm bảng trang để tìm ra trang LRU và viết tới bộ nhớ (tới trường thời gian dùng trong bảng trang) cho mỗi truy xuất bộ nhớ. Số lần cũng phải được duy trì khi các bảng trang bị thay đổi (do định thời CPU). Vượt quá giới hạn của đồng hồ phải được xem xét.

- **Ngăn xếp**: một tiếp cận khác để cài đặt thay thế LRU là giữ ngăn xếp số trang. Bất cứ khi nào một trang được tham khảo, nó bị xoá từ ngăn xếp và đặt trên đỉnh. Trong cách này, đỉnh của ngăn xếp luôn là trang được dùng gần nhất và đáy là trang LRU (hình VIII-11). Vì các mục từ phải được xoá từ giữa ngăn xếp, nó được cài đặt tốt nhất bởi một danh sách được liên kết kép với con trỏ đầu và đuôi. Xoá một trang và đặt nó trên đỉnh của ngăn xếp sau đó yêu cầu thay đổi 6 con trỏ trong trường hợp xấu nhất. Mỗi cập nhật là ít chi phí hơn nhưng không cần tìm một thay thế; con trỏ đuôi chỉ tới đáy của ngăn xếp là trang LRU. Tiếp cận này đặc biệt phù hợp cho cài đặt phần mềm hay vi mã của thay thế LRU.

Thay thế tối ưu hoá và LRU không gặp phải sự nghịch lý của Belady. Có một lớp giải thuật thay thế trang được gọi là giải thuật ngăn xếp, mà nó không bao giờ hiển thị sự nghịch lý Belady. Một giải thuật ngăn xếp là một giải thuật mà nó có thể được hiển thị rằng tập hợp trang trong bộ nhớ đối với n khung trang luôn là tập hợp con của tập hợp các trang mà nó ở trong bộ nhớ với n + 1 khung. Đối với thay thế LRU, tập hợp trang trong bộ nhớ là n trang được tham khảo gần đây nhất. Nếu số trang được gia tăng thì n trang này sẽ vẫn là những trang được tham khảo gần đây nhất và vì thế sẽ vẫn ở trong bộ nhớ.

Chú ý rằng cài đặt LRU sẽ có thể không có sự trợ giúp phần cứng ngoại trừ thanh ghi TLB. Cập nhật các trường đồng hồ hay ngăn xếp phải được thực hiện cho mỗi tham khảo bộ nhớ. Nếu chúng ta sử dụng ngắt cho mỗi tham khảo bộ nhớ, cho phép phần mềm cập nhật cấu trúc dữ liệu thì nó sẽ làm chậm mỗi tham khảo bộ nhớ gần 1 phần 10. Rất ít hệ thống có thể chịu cấp độ chi phí đó cho việc quản lý bộ nhớ.



Hình 0-11 sử dụng ngăn xếp để ghi những tham khảo trang gần nhất

Thuviendientu.org

V.4 Giải thuật thay thế trang xấp xỉ LRU

Rất ít hệ thống máy tính cung cấp đầy đủ hỗ trợ phần cứng cho thay thế trang LRU. Một số hệ thống không cung cấp bất cứ sự hỗ trợ phần cứng và giải thuật thay thế trang khác (như giải thuật FIFO) phải được dùng. Tuy nhiên, nhiều hệ thống cung cấp một vài hỗ trợ trong dạng 1 bit tham khảo. Bit tham khảo cho một trang được đặt bởi phần cứng, bất cứ khi nào trang đó được tham khảo (đọc hay viết tới bất cứ bit nào trong trang). Các bit tham khảo gắn liền với mỗi mục từ trong bảng trang.

Ban đầu, tất cả bit được xoá (tới 0) bởi hệ điều hành. Khi một quá trình người dùng thực thi, bit được gán với mỗi trang được tham khảo được đặt (tới 1) bởi phần cứng. Sau thời gian đó, chúng có thể xác định trang nào được dùng và trang nào không được dùng bằng cách xem xét các bit tham khảo. Chúng ta không biết thứ tự sử dụng nhưng chúng ta biết trang nào được dùng và trang nào không được dùng. Thông tin thứ tự từng phần dẫn tới nhiều giải thuật thay thế trang xấp xỉ thay thế LRU.

V.4.1 Giải thuật các bit tham khảo phụ

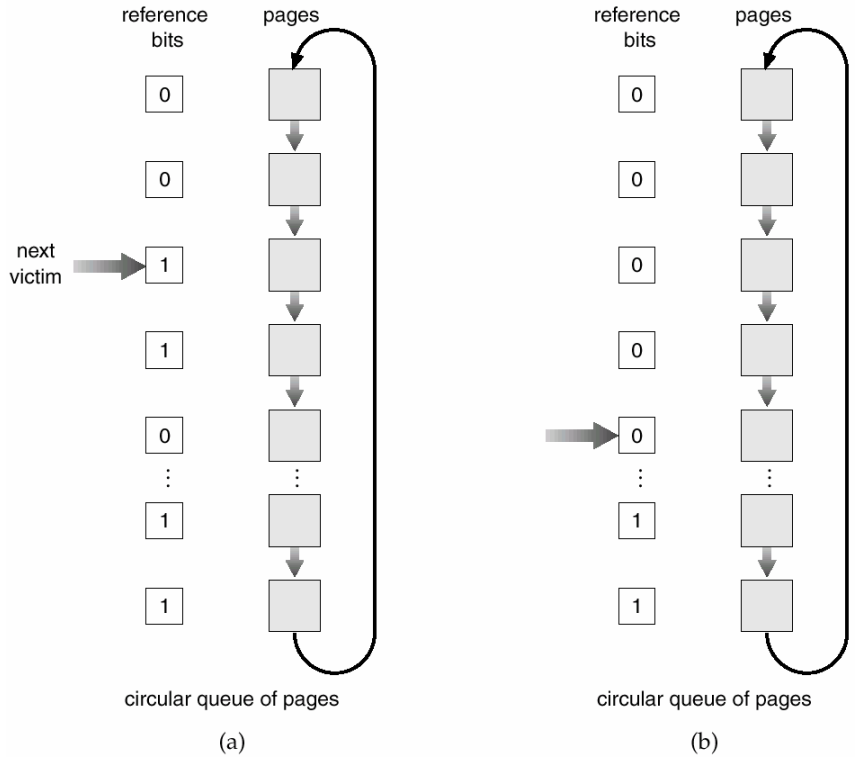
Chúng ta có thể nhận thêm thông tin thứ tự bằng cách ghi nhận các bit tham khảo tại những khoảng thời gian đều đặn. Chúng ta có thể giữ một byte cho mỗi trang trong một bảng nằm trong bộ nhớ. Tại những khoảng thời gian đều đặn (mỗi 100 mili giây), một ngắt thời gian chuyển điều khiển tới hệ điều hành. Hệ điều hành chuyển bit tham khảo cho mỗi trang vào bit có trọng số lớn nhất của byte, dịch các bit còn lại sang phải 1 bit. Xoá bit có trọng số thấp nhất. Thanh ghi dịch 8 bit có thể chứa lịch sử của việc sử dụng trang đối với 8 lần gần nhất. Nếu thanh ghi dịch chứa 00000000, thì trang không được dùng cho 8 thời điểm; một trang được dùng ít nhất một lần mỗi thời điểm sẽ có giá trị thanh ghi dịch là 11111111.

Một thanh ghi với giá trị thanh ghi lịch sử là 11000100 được dùng gần đây hơn một trang với 01110111. Nếu chúng ta thông dịch 8 bit này như số nguyên không dấu, trang với số thấp nhất là trang LRU và nó có thể được thay thế. Tuy nhiên, các số này không đảm bảo duy nhất, chúng ta thay thế tất cả trang với giá trị nhỏ nhất hay dùng FIFO để chọn giữa chúng.

Dĩ nhiên, số lượng bit lịch sử có thể khác nhau và có thể được chọn (phụ thuộc phần cứng sẵn có) để thực hiện cập nhật nhanh nhất có thể. Trong trường hợp cực độ, số có thể được giảm về 0, chỉ bit tham khảo chính nó. Giải thuật này được gọi là **giải thuật thay thế trang cơ hội thứ hai** (second-chance page-replacement algorithm).

Thuviendientu.org

V.4.2 Giải thuật cơ hội thứ hai



Hình 0-12 giải thuật thay thế trang cơ hội thứ hai

Giải thuật thay thế trang cơ hội thứ hai cơ bản là giải thuật thay thế FIFO. Tuy nhiên, khi một trang được chọn, chúng ta xét bit tham khảo của nó. Nếu giá trị bit này là 0, chúng ta xử lý để thay thế trang này. Tuy nhiên, nếu bit tham khảo được đặt tới 1, chúng ta cho trang đó một cơ hội thứ hai và di chuyển để chọn trang FIFO kế tiếp. Khi một trang nhận cơ hội thứ hai, bit tham khảo của nó được xoá và thời gian đến của nó được đặt lại là thời gian hiện hành. Do đó, một trang được cho cơ hội thứ hai sẽ không được thay thế cho đến khi tất cả trang khác được thay thế (hay được cho cơ hội thứ hai). Ngoài ra, nếu một trang được dùng đủ thường xuyên để giữ bit tham khảo của nó được đặt, nó sẽ không bao giờ bị thay thế.

Một cách để cài đặt giải thuật cơ hội thứ hai như một hàng đợi vòng. Một con trỏ hiển thị trang nào được thay thế tiếp theo. Khi một khung được yêu cầu, con trỏ tăng cho tới khi nó tìm được trang với bit tham khảo 0. Khi nó tăng, nó xoá các bit tham khảo (hình VIII-12). Một khi trang nạn nhân được tìm thấy, trang được thay thế và trang mới được chèn vào hàng đợi vòng trong vị trí đó. Chú ý rằng, trong trường hợp xấu nhất khi tất cả bit được đặt, con trỏ xoay vòng suốt toàn hàng đợi, cho mỗi trang một cơ hội thứ hai. Thay thế cơ hội thứ hai trở thành thay thế FIFO nếu tất cả bit được đặt.

V.4.3 Giải thuật cơ hội thứ hai nâng cao

Chúng ta có thể cải tiến giải thuật cơ hội thứ hai bằng cách xem xét cả hai bit tham khảo và sửa đổi như một cặp được xếp thứ tự. Với hai bit này , chúng ta có 4 trường hợp có thể:

1) (0,0) không được dùng mới đây và không được sửa đổi-là trang tốt nhất để thay thế.
2) (0,1) không được dùng mới đây nhưng được sửa đổi-không thật tốt vì trang cần được viết ra trước khi thay thế.

Thuviendientu.org

3) (1,0) được dùng mới đây nhưng không được sửa đổi-nó có thể sẽ nhanh chóng được dùng lại.
4) (1,1) được dùng mới đây và được sửa đổi-trang có thể sẽ nhanh chóng được dùng lại và trang sẽ cần được viết ra đĩa trước khi nó có thể được thay thế.

Khi thay thế trang được yêu cầu, mỗi trang ở một trong bốn trường hợp. Chúng ta dùng cùng một cơ chế như giải thuật đồng hồ, nhưng thay vì xem xét trang chúng ta đang trỏ tới có bit tham khảo được đặt tới 1 hay không, chúng ta xem xét trường hợp mà trang đó đang thuộc về. Chúng ta thay thế trang đầu tiên được gặp trong trường hợp thấp nhất không rỗng. Có thể chúng ta phải quét hàng đợi vòng nhiều lần trước khi chúng ta tìm một trang được thay thế.

Giải thuật này được dùng trong cơ chế quản lý bộ nhớ ảo của Macintosh. Sự khác nhau chủ yếu giữa giải thuật này và giải thuật đồng hồ đơn giản hơn là chúng ta cho tham khảo tới các trang đó mà chúng được sửa đổi để cắt giảm số lượng nhập/xuất được yêu cầu.

V.4.4 Thay thế trang dựa trên cơ sở đếm

Có nhiều giải thuật khác có thể được dùng để thay thế trang. Thí dụ, chúng ta có thể giữ bộ đếm số lần tham khảo đối với mỗi trang và phát triển hai cơ chế sau:

- **Giải thuật thay thế trang được dùng ít thường xuyên nhất** (the least frequently used (LFU) page-replacement algorithm) yêu cầu trang với số đếm nhỏ nhất được thay thế. Lý do cho sự chọn lựa này là trang được dùng nên có bộ đếm tham khảo lớn. Giải thuật này gặp phải trường hợp: trang được dùng nhiều trong quá trình khởi tạo nhưng không bao giờ được dùng lại. Vì nó được dùng nhiều nên nó có bộ đếm lớn và vẫn ở trong bộ nhớ mặc dù nó không còn cần nữa. Một giải pháp là dịch bộ đếm sang phải 1 bit tại khoảng thời gian đều đặn, hình thành một bộ đếm sử dụng trung bình giảm theo hàm mũ.
- **Giải thuật thay thế trang được dùng thường xuyên nhất** (the most frequently used (MFU) page-replacement algorithm) thay thế trang có giá trị đếm lớn nhất, nghĩa là trang được sử dụng nhiều nhất.

VI Cấp phát khung trang

Chúng ta cấp phát lượng bộ nhớ trống cố định giữa các quá trình khác nhau như thế nào? Nếu chúng ta có 93 khung trang trống và 2 quá trình, bao nhiêu khung trang mỗi quá trình sẽ nhận?

Trường hợp đơn giản nhất của bộ nhớ ảo là hệ thống đơn nhiệm. Xét một hệ thống đơn nhiệm với 128 KB bộ nhớ được hình thành từ các trang có kích thước 1 KB. Do đó, có 128 khung trang. Hệ điều hành có thể lấy 35 KB, còn lại 93 khung trang cho quá trình người dùng. Dưới thuần phân trang yêu cầu, tất cả 93 khung trang đầu tiên được đặt vào danh sách khung trống. Khi một quá trình người dùng bắt đầu thực thi, nó sinh ra một chuỗi lỗi trang. Những lỗi trang 93 đầu tiên nhận những khung trống từ danh sách khung trống. Khi danh sách khung trống hết, một giải thuật thay thế trang được dùng để chọn một trong 93 trang đang ở trong bộ nhớ để thay thế với trang thứ 94, …Khi một quá trình kết thúc, khung trang 93 một lần nữa được thay thế trên danh sách khung trang trống.

Có nhiều thay đổi trên chiến lược đơn giản này. Chúng ta có thể yêu cầu hệ điều hành cấp phát tất cả vùng đệm của nó và không gian bảng từ danh sách khung trống.

Thuviendientu.org

Khi không gian này không được dùng bởi hệ điều hành, nó có thể được dùng để hỗ trợ phân trang người dùng. Chúng ta có thể cố gắng giữ 3 khung trang trống được dự trữ trên danh sách khung trang trống tại tất cả thời điểm. Do đó, khi lỗi trang xảy ra có một khung trống sẵn có đối với trang. Trong khi hoán vị trang xảy ra, một thay thế có thể được chọn, sau đó trang được viết tới đĩa khi quá trình người dùng tiếp tục thực thi.

Một thay đổi khác cũng có thể thực hiện trên chiến lược cơ bản là quá trình người dùng được cấp phát bất cứ khung trang nào trống.

Một vấn đề khác phát sinh khi phân trang yêu cầu được kết hợp với đa chương. Đa chương đặt hai hay nhiều quá trình trong bộ nhớ tại cùng một thời điểm.

VI.1 Số khung trang tối thiểu

Những chiến lược cấp phát khung trang bị ràng buộc trong nhiều cách khác nhau. Chúng ta không thể cấp phát nhiều hơn toàn bộ số khung trang sẵn có (nếu không có chia sẻ trang). Chúng ta cũng cấp phát ít nhất số khung trang tối thiểu. Chú ý, khi số khung trang được cấp phát tới mỗi quá trình giảm, tỉ lệ lỗi trang tăng, giảm việc thực thi quá trình.

Ngoài ra, năng lực thực hiện việc cấp phát ngoài mong muốn chỉ có một vài khung trang, có số khung trang tối thiểu phải được cấp phát. Số lượng tối thiểu. Số tối thiểu này được qui định bởi kiến trúc máy tính. Nhớ rằng, khi lỗi trang xảy ra trước khi chỉ thị thực thi hoàn thành, chỉ thị phải bắt đầu lại. Do đó, chúng ta phải có đủ khung trang để giữ tất cả trang khác nhau mà bất cứ chỉ thị đơn có thể tham khảo.

Thí dụ, xét một máy trong đó tất cả chỉ thị tham khảo bộ nhớ chỉ có một địa chỉ bộ nhớ. Do đó, chúng ta cần ít nhất một khung trang cho chỉ thị và một khung trang cho tham khảo bộ nhớ. Ngoài ra, nếu định địa chỉ gián tiếp cấp 1 được phép (thí dụ, một chỉ thị load trên trang 16 có thể tham khảo tới một địa chỉ bộ nhớ trên trang 0, mà nó tham khảo gián tiếp tới trang 23), thì phân trang yêu cầu ít nhất 3 khung trên quá trình. Điều gì có thể xảy ra nếu một quá trình chỉ có hai khung trang.

VI.1.1 Các giải thuật cấp phát trang

Có hai tiếp cận:

1. **Cấp phát cố định**
   - Cấp phát công bằng: nếu có m khung trang và n quá trình, mỗi quá trình được cấp m/n khung trang
   - Cấp phát theo tỉ lệ: dựa vào kích thước của tiến trình để cấp phát số khung trang:
     i. Gọi s_i = kích thước của bộ nhớ ảo cho quá trình p_i
     ii. $S = \sum s_i$
     iii. m = tổng số khung trang có thể sử dụng
     iv. Cấp phát a_i khung trang tới quá trình p_i: $a_i = (s_i / S)\ m$
2. **Cấp phát theo độ ưu tiên**

Sử dụng ý tưởng cấp phát theo tỷ lệ, nhưng lượng khung trang cấp cho quá trình phụ thuộc vào độ ưu tiên của quá trình hơn là phụ thuộc kích thước quá trình

Nếu quá trình p_i phát sinh lỗi trang, chọn một trong các khung trang của nó để thay thế, hoặc chọn một khung trang của quá trình khác với độ ưu tiên thấp hơn để thay thế.

- Thay thế trang toàn cục hay cục bộ

  Có thể phân các thuật toán thay thế trang thành hai lớp chính:

Thuviendientu.org

i. Thay thế toàn cục: khi lỗi trang xảy ra với một quá trình, chọn trang "nạn nhân" từ tập tất cả các khung trang trong hệ thống, bất kể khung trang đó đang được cấp phát cho một quá trình khác.
ii. Thay thế cục bộ: yêu cầu chỉ được chọn trang thay thế trong tập các khung trang được cấp cho quá trình phát sinh lỗi trang

Một khuyết điểm của giải thuật thay thế toàn cục là các quá trình không thể kiểm soát được tỷ lệ phát sinh lỗi trang của mình. Vì thế, tuy giải thuật thay thế toàn cục nhìn chung cho phép hệ thống có nhiều khả năng xử lý hơn, nhưng nó có thể dẫn hệ thống đến tình trạng trì trệ toàn bộ hệ thống (thrashing).

VII Trì trệ toàn hệ thống

Nếu một quá trình không có đủ các khung trang để chứa những trang cần thiết cho xử lý thì nó sẽ thường xuyên phát sinh lỗi trang và vì thế phải dùng đến rất nhiều thời gian sử dụng CPU để thực hiện thay thế trang. Một hoạt động phân trang như thế được gọi là sự trì trệ (thrashing). Một quá trình lâm vào trạng thái trì trệ nếu nó sử dụng nhiều thời gian để thay thế hơn là để xử lý.

Hiện tượng này ảnh hưởng nghiêm trọng đến hoạt động hệ thống, xét tình huống sau:

1) Hệ điều hành giám sát việc sử dụng CPU
2) Nếu hiệu suất sử dụng CPU quá thấp, hệ điều hành sẽ nâng mức độ đa chương bằng cách đưa thêm một quá trình mới vào hệ thống.
3) Hệ thống có thể sử dụng giải thuật thay thế toàn cục để chọn các trang nạn nhân thuộc một tiến trình bất kỳ để có chỗ nạp quá trình mới, có thể sẽ thay thế cả các trang của tiến trình đang xử lý hiện hành.
4) Khi có nhiều quá trình trong hệ thống hơn, thì một quá trình sẽ được cấp ít khung trang hơn và do đó phát sinh nhiều lỗi trang hơn.
5) Khi các quá trình phát sinh nhiều lỗi trang, chúng phải trải qua nhiều thời gian chờ các thao tác thay thế trang hoàn tất, lúc đó hiệu suất sử dụng CPU lại giảm.
6) Hệ điều hành lại quay trở lại bước 1.

Theo kịch bản trên đây, hệ thống sẽ lâm vào tình trạng luẩn quẩn của việc giải phóng các trang để cấp phát thêm khung trang cho một quá trình, và các quá trình khác lại thiếu khung trang..và các quá trình không thể tiếp tục xử lý. Đây chính là tình trạng trì trệ toàn bộ hệ thống. Khi tình trạng trì trệ này xảy ra, hệ thống gần như mất khả năng xử lý, tốc độ phát sinh lỗi trang tăng rất cao không công việc nào có thể kết thúc vì tất cả quá trình đều bận rộn với việc phân trang.

Để ngăn cản tình trạng trì trệ này xảy ra, cần phải cấp cho quá trình đủ các khung trang cần thiết để hoạt động. Vấn đề cần giải quyết là làm sao biết được quá trình cần bao nhiêu trang?

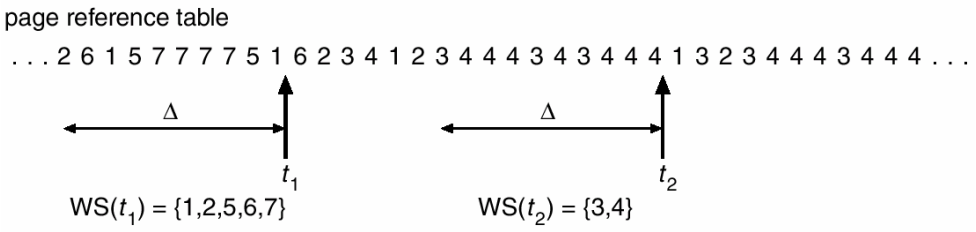
VII.1 Mô hình cục bộ

Theo lý thuyết cục bộ thì khi một quá trình xử lý nó có khuynh hướng di chuyển từ nhóm trang cục bộ này đến nhóm trang cục bộ khác. Một nhóm trang cục bộ là một tập các trang đang được quá trình dùng đến trong một khoảng thời gian. Một chương trình thường bao gồm nhiều nhóm trang cục bộ khác nhau và chúng có thể giao nhau.

Thuviendientu.org

VII.2 Mô hình tập làm việc

Mô hình tập làm việc (working set model) này dựa trên cơ sở lý thuyết cục bộ. Mô hình này sử dụng tham số Δ để định nghĩa cửa sổ cho tập làm việc. Giả sử, khảo sát Δ đơn vị thời gian (lần truy xuất trang) cuối cùng, tập các trang được quá trình truy xuất đến trong Δ lần truy cập cuối cùng được gọi là tập làm việc của quá trình tại thời điểm hiện tại. Nếu một trang đang được quá trình truy xuất tới, nó sẽ nằm trong tập làm việc nếu nó không sử dụng nữa, nó sẽ bị loại khỏi tập làm việc của quá trình sau Δ đơn vị thời gian kể từ lần truy xuất cuối cùng đến nó. Như vậy, tập làm việc chính là sự xấp xỉ của khái niệm nhóm trang cục bộ.



Hình 0-13 Mô hình tập làm việc

Thuộc tính rất quan trọng của tập làm việc là kích thước của nó. Nếu tính toán kích thước tập làm việc WSS_i, cho mỗi tiến trình trong hệ thống thì có thể xem:

$$D = \sum WSS_i$$

Với D là tổng số khung trang yêu cầu cho toàn hệ thống. Mỗi quá trình sử dụng các trang trong tập làm việc của nó, nghĩa là quá trình i yêu cầu WSS_i khung trang. Nếu tổng số trang yêu cầu vượt quá tổng số trang có thể sử dụng trong hệ thống (D > m), thì sẽ xảy ra tình trạng trì trệ toàn bộ.

Dùng mô hình tập làm việc là đơn giản. Hệ điều hành kiểm soát tập làm việc của mỗi quá trình và cấp phát cho quá trình tối thiểu các khung trang để chứa đủ tập làm việc của nó. Nếu có đủ khung trang bổ sung thì quá trình khác có thể được khởi tạo. Nếu tổng kích thước tập làm việc gia tăng vượt quá tổng số khung sẵn có, hệ điều hành chọn một quá trình để tạm dừng. Những trang của quá trình này được viết ra đĩa và các khung trang của nó được cấp phát lại cho quá trình khác. Quá trình được tạm dừng có thể khởi động lại sau đó.

Chiến lược tập làm việc ngăn chặn sự trì trệ trong khi giữ cấp độ đa chương cao nhất có thể. Do đó, nó tối ưu việc sử dụng CPU.

Khó khăn với mô hình tập làm việc này là giữ vết của tập làm việc. Cửa sổ tập làm việc là một cửa sổ di chuyển. Tại mỗi tham khảo bộ nhớ, một tham khảo mới xuất hiện khi một tham khảo trước đó kết thúc và tham khảo cũ nhất trở thành điểm kết thúc khác. Một trang ở trong tập làm việc nếu nó được tham khảo bất cứ nơi nào trong cửa sổ tập làm việc. Chúng ta có thể xem mô hình tập làm việc gần xấp xỉ với ngắt đồng hồ sau từng chu kỳ cố định và bit tham khảo.

VII.3 Tần suất lỗi trang

Tần suất lỗi trang rất cao khiến tình trạng trì trệ hệ thống xảy ra. Khi tần suất lỗi trang quá cao, quá trình cần thêm một số khung trang. Ngược lại, khi tần suất quá thấp, quá trình có thể sở hữu nhiều khung trang hơn mức cần thiết. Có thể thiết lập một giá trị cận trên và cận dưới cho tần suất xảy ra lỗi trang và trực tiếp ước lượng và kiểm soát tần suất lỗi trang để ngăn chặn tình trạng trì trệ xảy ra:

Thuviendientu.org

- Nếu tần suất lỗi trang vượt quá cận trên, cấp cho quá trình thêm một khung trang
- Khi tần suất lỗi trang thấp hơn cận dưới, thu hồi bớt một khung trang từ quá trình.

Với chiến lược tập làm việc, chúng ta có thể có phải tạm dừng một quá trình. Nếu tỉ lệ lỗi trang tăng và không có trang nào trống, chúng ta phải chọn một số quá trình và tạm dừng nó. Sau đó, những khung trang được giải phóng sẽ được phân phối lại cho các quá trình với tỉ lệ lỗi trang cao.

VIII Các vấn đề khác

VIII.1 Kích thước trang

Kích thước trang thông thường được xác định bởi phần cứng. Không có sự chọn lựa lý tưởng cho kích thước trang:

- Kích thước trang càng lớn thì kích thước bảng trang càng giảm
- Kích thước trang càng nhỏ thì cho phép tổ chức nhóm trang cục bộ tốt hơn và giảm sự phân mãnh trong
- Thời gian nhập xuất nhỏ khi kích thước trang lớn
- Kích thước trang nhỏ thì có thể giảm số lượng thao tác nhập xuất cần thiết vì có thể xác định các nhóm trang cục bộ chính xác hơn
- Kích thước trang lớn sẽ giảm tần xuất lỗi trang

Đa số các hệ thống chọn kích thước trang là 4 KB.

VIII.2 Cấu trúc chương trình

Về nguyên tắc, kỹ thuật phân trang theo yêu cầu được thiết kế nhằm giúp người dùng khỏi bận tâm đến việc sử dụng bộ nhớ một cách hiệu quả. Tuy nhiên, nếu hiểu rõ tổ chức bộ nhớ trong kỹ thuật phân trang, lập trình viên có thể giúp cho hoạt động của hệ thống tốt hơn với chương trình được xây dựng phù hợp.

Thí dụ, giả sử 1 trang có kích thước 128 bytes, một chương trình khởi tạo và gán giá trị mảng có kích thước 128x128 như sau:

```
Var A: array[1..128] of array [1..128] of byte;
  For i:= 1 to 128 do
        For j:=1 to 128 do
              A[i][j]:=0;
```

Trong Pascal, C, PL/I, mảng trên đây được lưu trữ theo thứ tự dòng, mỗi dòng mảng chiếm một trang bộ nhớ, do đó tổng số lỗi trang phát sinh sẽ là 128.

Trong Fortran, mảng trên đây lại được lưu trữ theo thứ tự cột, do đó tổng số lỗi trang phát sinh sẽ là 128x128 = 1638.

VIII.3 Neo các trang trong bộ nhớ chính

Khi áp dụng kỹ thuật phân trang đôi lúc có nhu cầu "neo" trong bộ nhớ chính một số trang quan trọng hoặc thường được sử dụng hoặc không thể chuyển ra bộ nhớ phụ để bảo toàn dữ liệu.

Khi đó sử dụng thêm một bit khoá gán tương ứng cho từng khung trang. Một khung trang có bit khoá được đặt sẽ không bị chọn để thay thế.

Thuviendientu.org

IX Tóm tắt

Mong muốn có thể thực thi một quá trình có không gian địa chỉ luận lý lớn hơn không gian địa chỉ vật lý sẵn có. Người lập trình có thể làm một quá trình như thế có thể thực thi bằng cách cấu trúc lại nó dùng cơ chế phủ lắp, nhưng thực hiện điều này thường là một tác vụ lập trình khó. Bộ nhớ ảo là một kỹ thuật cho phép không gian địa chỉ luận lý được ánh xạ vào bộ nhớ vật lý nhỏ hơn. Bộ nhớ ảo cho phép những quá trình cực lớn được chạy và cũng cho phép cấp độ đa chương được gia tăng, tăng khả năng sử dụng CPU. Ngoài ra, nó giải phóng người lập trình ứng dụng từ việc lo lắng khả năng sẵn có của bộ nhớ.

Thuần phân trang theo yêu cầu mang vào một trang cho tới khi trang đó được tham khảo. Tham khảo đầu tiên gây ra lỗi trang tới hệ điều hành. Hệ điều hành xem xét bảng trang bên trong để xác định nơi trang được định vị trên vùng bộ nhớ phụ. Bảng trang được cập nhật để phản ánh sự thay đổi này, cho phép một quá trình chạy mặc dù toàn bộ hình ảnh bộ nhớ của nó không ở trong bộ nhớ chính. Khi tỉ lệ lỗi trang tương đối thấp, năng lực có thể chấp nhận.

Chúng ta có thể dùng phân trang theo yêu cầu để giảm số khung trang được cấp phát tới quá trình. Sắp xếp này có thể tăng cấp độ đa chương (cho phép nhiều quá trình sẵn sằng thực thi tại một thời điểm). Nó cũng cho phép các quá trình được thực thi mặc dù yêu cầu bộ nhớ vượt quá toàn bộ bộ nhớ vật lý sẵn có. Những quá trình như thế chạy trong bộ nhớ ảo.

Nếu tổng số yêu cầu bộ nhớ vượt quá bộ nhớ vật lý, thì nó cần thay thế trang từ bộ nhớ tới các khung trang trống cho những trang mới. Những giải thuật thay thế trang khác nhau được dùng. Thay thế trang FIFO là dễ dàng đối với chương trình nhưng gặp phải lỗi Belady. Thay thế trang tối ưu yêu cầu kiến thức tương lai. Thay thế LRU là xấp xỉ tối ưu nhưng nó rất khó cài đặt. Hầu hết các giải thuật thay thế trang như giải thuật cơ hội thứ hai là xấp xỉ thay thế LRU.

Ngoài ra đối với giải thuật thay thế trang, chính sách cấp phát khung trang được yêu cầu. Cấp phát có thể cố định, đề nghị thay thế trang cục bộ, hay động, đề nghị thay thế toàn cục. Mô hình tập làm việc cho rằng các quá trình thực thi trong các vị trí. Tập làm việc là tập các trang trong các vị trí hiện hành. Theo đó, mỗi quá trình nên được cấp phát đủ các khung cho tập làm việc hiện hành của nó.

Nếu một quá trình không có đủ bộ nhớ cho tập làm việc của nó, nó sẽ bị trì trệ. Cung cấp đủ khung cho mỗi quá trình để tránh trì trệ có thể yêu cầu quá trình hoán vị và định thời.

Ngoài ra, để yêu cầu chúng ta giải quyết các vấn đề chính của thay thế trang và cấp phát khung trang, thiết kế hợp lý hệ thống phân trang yêu cầu chúng ta xem xét kích thước trang, nhập/xuất, khoá, phân lại trang, tạo quá trình, cấu trúc chương trình, sự trì trệ,.. Bộ nhớ ảo có thể được xem như một cấp của cơ chế phân cấp trong các cấp lưu trữ trong hệ thống máy tính. Mỗi cấp có thời gian truy xuất, kích thước và tham số chi phí của chính nó.

Thuviendientu.org

HỆ THỐNG TẬP TIN

I Mục đích

Sau khi học xong chương này, người học nắm được những kiến thức sau:

- Hiểu các khía cạnh khác nhau của tập tin và cấu trúc thư mục
- Hiểu các cơ chế quản lý, kiểm soát, bảo vệ tập tin khi có nhiều người cùng truy xuất
- Hiểu cách chia sẻ tập tin giữa nhiều quá trình, người dùng và máy tính

II Giới thiệu

Đối với hầu hết người dùng, hệ thống tập tin là diện mạo dễ nhìn thấy nhất của hệ điều hành. Nó cung cấp cơ chế cho việc lưu trữ trực tuyến và truy xuất dữ liệu, chương trình của hệ điều hành và tất cả người dùng của hệ thống máy tính. Hệ thống tập tin chứa hai phần riêng biệt: tập hợp ***các tập tin*** (files), mỗi tập tin lưu trữ dữ liệu có liên quan và ***cấu trúc thư mục*** (directory structure) mà nó tổ chức và cung cấp thông tin về tất cả tập tin trong hệ thống. Một số hệ thống tập tin còn có thêm phần thứ ba , ***các phân khu*** (partitions) mà nó được dùng để tách rời tập hợp các thư mục lớn luận lý và vật lý.

Trong chương này chúng ta xét các khía cạnh khác nhau của tập tin và cấu trúc thư mục. Chúng ta cũng thảo luận các cách để quản lý ***việc bảo vệ tập tin*** (file protection), cần thiết khi nhiều người dùng truy xuất các tập tin và chúng ta muốn kiểm soát ai và cách gì truy xuất tập tin. Cuối cùng, chúng ta thảo luận việc chia sẻ giữa nhiều quá trình, người dùng, và máy tính.

III Khái niệm tập tin

Các máy tính lưu trữ thông tin trên nhiều phương tiện lưu trữ khác nhau, như đĩa từ, băng từ, đĩa quang. Để hệ thống máy tính tiện dụng, hệ điều hành cung cấp một tầm nhìn luận lý không đổi của việc lưu trữ thông tin. Hệ điều hành trừu tượng từ các thuộc tính vật lý của các thiết bị lưu trữ của nó đến định nghĩa một đơn vị lưu trữ luận lý là **tập tin** (file). Tập tin được ánh xạ bởi hệ điều hành trên các thiết bị vật lý. Các thiết bị lưu trữ được dùng thường ổn định vì thế nội dung không bị mất khi mất điện hay khởi động lại hệ thống.

Một tập tin là một tập thông tin có liên quan được ghi trên thiết bị lưu trữ phụ. Từ quan điểm người dùng, một tập tin là phần nhỏ nhất của thiết bị lưu trữ phụ luận lý; nghĩa là dữ liệu không thể được viết tới thiết bị lưu trữ phụ trừ khi chúng ở trong một tập tin. Các tập tin dữ liệu có thể là số, chữ, ký tự số hay nhị phân. Các tập tin có thể có dạng bất kỳ như tập tin văn bản, hay có thể được định dạng không đổi. Thông thường, một tập tin là một chuỗi các bits, bytes, dòng hay mẫu tin,..được định nghĩa bởi người tạo ra nó. Do đó, khái niệm tập tin là cực kỳ tổng quát.

Thông tin trong một tập tin được định nghĩa bởi người tạo. Nhiều loại thông tin khác nhau có thể được lưu trữ trong một tập tin-chương trình nguồn, chương trình đối tượng, chương trình thực thi, dữ liệu số, văn bản, mẫu tin, hình ảnh đồ hoạ, âm thanh,..Một tập tin có một cấu trúc được định nghĩa cụ thể dựa theo loại của nó. Một tập tin văn bản là một chuỗi các ký tự được tổ chức thành những dòng. Một tập tin

Thuviendientu.org

nguồn là một chuỗi các thủ tục và hàm, được tổ chức khi khai báo được theo sau bởi các câu lệnh có thể thực thi. Một tập tin đối tượng là một chuỗi các bytes được tổ chức thành các khối có thể hiểu được bởi bộ liên kết của hệ thống. Một tập tin có thể thực thi là một chuỗi các phần mã mà bộ nạp có thể mang vào bộ nhớ và thực thi.

III.1 Thuộc tính tập tin

Để tiện cho người dùng, một tập tin được đặt tên và được tham khảo bởi tên của nó. Một tên thường là một chuỗi các ký tự, thí dụ: example.c. Một số hệ thống có sự phân biệt giữa ký tự hoa và thường trong tên, ngược lại các hệ thống khác xem hai trường hợp đó là tương đương. Khi một tập tin được đặt tên, nó trở nên độc lập với quá trình, người dùng, và thậm chí với hệ thống tạo ra nó. Thí dụ, một người dùng có thể tạo tập tin example.c, ngược lại người dùng khác có thể sửa tập tin đó bằng cách xác định tên của nó. Người sở hữu tập tin có thể ghi tập tin tới đĩa mềm, gởi nó vào email hay chép nó qua mạng và có thể vẫn được gọi example.c trên hệ thống đích.

Một tập tin có một số thuộc tính khác mà chúng rất khác nhau từ một hệ điều hành này tới một hệ điều hành khác, nhưng điển hình chúng gồm:

- **Tên** (name): tên tập tin chỉ là thông tin được lưu ở dạng mà người dùng có thể đọc
- **Định danh** (identifier): là thẻ duy nhất, thường là số, xác định tập tin trong hệ thống tập tin; nó là tên mà người dùng không thể đọc
- **Kiểu** (type): thông tin này được yêu cầu cho hệ thống hỗ trợ các kiểu khác nhau
- **Vị trí** (location): thông tin này là một con trỏ chỉ tới một thiết bị và tới vị trí tập tin trên thiết bị đó.
- **Kích thước** (size): kích thước hiện hành của tập tin (tính bằng byte, word hay khối) và kích thước cho phép tối đa chứa trong thuộc tính này.
- **Giờ** (time), **ngày** (date) và **định danh người dùng** (user identification): thông tin này có thể được lưu cho việc tạo, sửa đổi gần nhất, dùng gần nhất. Dữ liệu này có ích cho việc bảo vệ, bảo mật, và kiểm soát việc dùng.

Thông tin về tất cả tập tin được giữ trong cấu trúc **thư mục** (directory) nằm trong thiết bị lưu trữ phụ. Điển hình, mục từ thư mục chứa tên tập tin và định danh duy nhất của nó. Định danh lần lượt xác định thuộc tính tập tin khác. Trong hệ thống có nhiều tập tin, kích thước của chính thư mục có thể là Mbyte. Bởi vì thư mục giống tập tin, phải bền, chúng phải được lưu trữ trên thiết bị và mang vào bộ nhớ khi cần.

III.2 Thao tác tập tin

Tập tin là ***kiểu dữ liệu trừu tượng***. Để định nghĩa một tập tin hợp lý, chúng ta cần xem xét các thao tác có thể được thực hiện trên các tập tin. Hệ điều hành cung cấp lời gọi hệ thống để thực hiện các thao tác này

- **Tạo tập tin**: hai bước cần thiết để tạo một tập tin. Thứ nhất, không gian trong hệ thống tập tin phải được tìm cho tập tin. Thứ hai, một mục từ cho tập tin mới phải được tạo trong thư mục. Mục từ thư mục ghi tên tập tin và vị trí trong hệ thống tập tin, và các thông tin khác.
- **Mở**: trước khi mở tập tin, quá trình phải mở nó. Mục tiêu của việc mở là cho phép hệ thống thiết lập một số thuộc tính và địa chỉ đĩa trong bộ nhớ để tăng tốc độ truy xuất.

Thuviendientu.org

- **Đóng**: khi chấm dứt truy xuất, thuộc tính và địa chỉ trên đĩa không còn dùng nữa, tập tin được đóng lại để giải phóng vùng nhớ.
- **Ghi**: để ghi một tập tin, chúng ta thực hiện lời gọi hệ thống xác định tên tập tin và thông tin được ghi tới tập tin. Với tên tập tin, hệ thống tìm thư mục để xác định vị trí của tập tin. Hệ thống phải giữ một con trỏ viết tới vị trí trong tập tin nơi mà thao tác viết tiếp theo sẽ xảy ra. Con trỏ viết phải được cập nhật bất cứ khi nào thao tác viết xảy ra.
- **Chèn cuối**: giống thao tác ghi nhưng dữ liệu luôn được ghi vào cuối tập tin
- **Đọc**: để đọc từ một tập tin, chúng ta dùng lời gọi hệ thống xác định tên tập tin và nơi (trong bộ nhớ) mà khối tiếp theo của tập tin được đặt. Thư mục được tìm mục từ tương ứng và hệ thống cần giữ con trỏ đọc tới vị trí trong tập tin nơi thao tác đọc tiếp theo xảy ra.
- **Xoá**: để xoá một tập tin, chúng ta tìm kiếm thư mục với tên tập tin được cho. Tìm mục từ tương ứng, giải phóng không gian tập tin để không gian này có thể dùng lại bởi tập tin khác và xoá mục từ thư mục.
- **Tìm**: thư mục được tìm mục từ tương ứng và vị trí con trỏ hiện hành được đặt tới giá trị được cho
- **Lấy thuộc tính**: lấy thuộc tính tập tin cho quá trình
- **Đổi tên**: thay đổi tên tập tin đã tồn tại

III.3 Các kiểu tập tin

Khi thiết kế một hệ thống tập tin, chúng ta luôn luôn xem xét hệ điều hành nên tổ chức và hỗ trợ các kiểu tập tin nào. Nếu hệ điều hành nhận biết kiểu của một tập tin, nó có thể thao tác trên tập tin đó trong các cách phù hợp.

Một kỹ thuật chung cho việc cài đặt các kiểu tập tin là chứa kiểu đó như một phần của tên tập tin. Tên tập tin được chia làm hai phần-tên và phần mở rộng, thường được ngăn cách bởi dấu chấm. Trong trường hợp này, người dùng và hệ điều hành có thể biết kiểu tập tin là gì từ tên.

Các hệ điều hành thường hỗ trợ các kiểu tập tin sau:

- **Tập tin thường**: là tập tin văn bản hay tập tin nhị phân chứa thông tin của người sử dụng
- **Thư mục**: là những tập tin hệ thống dùng để lưu giữ cấu trúc của hệ thống tập tin
- **Tập tin có ký tự đặc biệt**: liên quan đến nhập/xuất thông qua các thiết bị nhập/xuất tuần tự như màn hình, máy in,..
- **Tập tin khối**: dùng để truy xuất trên thiết bị đĩa

III.4 Cấu trúc tập tin

Các kiểu tập tin cũng có thể được dùng để hiển thị cấu trúc bên trong của một tập tin. Ngoài ra, các tập tin cụ thể phải phù hợp cấu trúc được yêu cầu để hệ điều hành có thể hiểu. Một số hệ điều hành mở rộng ý tưởng này thành tập hợp các cấu trúc tập tin được hỗ trợ hệ thống, với những tập hợp thao tác đặc biệt cho việc thao tác các tập tin với những cấu trúc đó.

Các hệ điều hành thường hỗ trợ ba cấu trúc tập tin thông dụng là:

- **Không có cấu trúc**: tập tin là một dãy tuần tự các byte
- **Có cấu trúc**: tập tin là một dãy các mẫu tin có kích thước cố định

Thuviendientu.org

- **Cấu trúc cây**: tập tin gồm một cây của những mẫu tin không cần thiết có cùng chiều dài, mỗi mẫu tin có một trường khoá giúp việc tìm kiếm nhanh hơn

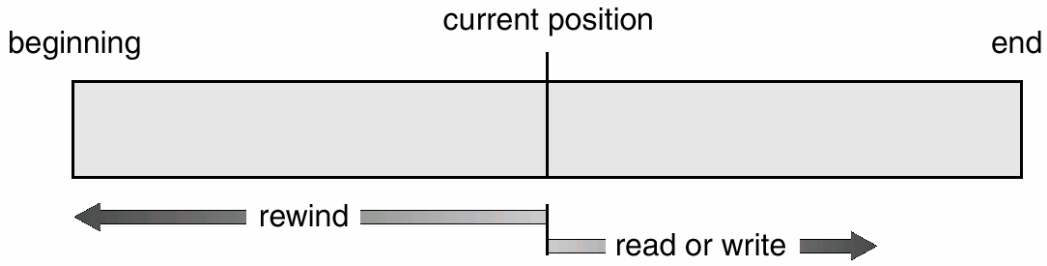
IV Các phương pháp truy xuất

Các tập tin lưu trữ thông tin. Khi nó được dùng, thông tin này phải được truy xuất và đọc vào bộ nhớ máy tính. Thông tin trong tập tin có thể được truy xuất trong nhiều cách.

IV.1 Truy xuất tuần tự

Một phương pháp đơn giản nhất là truy xuất tuần tự. Thông tin trong tập tin được xử lý có thứ tự, một mẫu tin này sau mẫu tin kia. Chế độ truy xuất này là thông dụng nhất. Thí dụ, bộ soạn thảo và biên dịch thường truy xuất các tập tin trong cách thức này.

Nhóm các thao tác trên một tập tin là đọc và viết. Một thao tác đọc đọc phần tiếp theo của tập tin và tự động chuyển con trỏ tập tin để ghi vết vị trí nhập/xuất. Tương tự, một thao tác viết chèn vào cuối tập tin và chuyển tới vị trí cuối của tài liệu vừa được viết (cuối tập tin mới). Trên một vài hệ thống, một tập tin như thế có thể được đặt lại tới vị trí bắt đầu và một chương trình có thể nhảy tới hay lùi n mẫu tin. Truy xuất tuần tự được mô tả như hình IX-1.



Hình 0-1 Truy xuất tập tin tuần tự

IV.2 Truy xuất trực tiếp

Một phương pháp khác là truy xuất trực tiếp (hay truy xuất tương đối). Một tập tin được hình thành từ các mẫu tin luận lý có chiều dài không đổi. Các mẫu tin này cho phép người lập trình đọc và viết các mẫu tin nhanh chóng không theo thứ tự. Phương pháp truy xuất trực tiếp dựa trên mô hình đĩa của tập tin, vì đĩa cho phép truy xuất ngẫu nhiên tới bất cứ khối tập tin. Để truy xuất trực tiếp, tập tin được hiển thị như một chuỗi các khối hay mẫu tin được đánh số. Tập tin truy xuất trực tiếp cho phép các khối bất kỳ được đọc hay viết. Do đó, chúng ta có thể đọc khối 14, sau đó đọc khối 53 và sau đó viết khối 7. Không có bất kỳ sự hạn chế nào trên thứ tự đọc hay viết cho một tập tin truy xuất trực tiếp

Các tập tin truy xuất trực tiếp được dùng nhiều cho truy xuất tức thời tới một lượng lớn thông tin. Cơ sở dữ liệu thường là loại này. Khi một truy vấn tập trung một chủ đề cụ thể, chúng ta tính khối nào chứa câu trả lời và sau đó đọc khối đó trực tiếp để cung cấp thông tin mong muốn.

Không phải tất cả hệ điều hành đều hỗ trợ cả hai truy xuất tuần tự và trực tiếp cho tập tin. Một số hệ thống cho phép chỉ truy xuất tập tin tuần tự; một số khác cho

Thuviendientu.org

phép chỉ truy xuất trực tiếp. Một số hệ điều hành yêu cầu một tập tin được định nghĩa như tuần tự hay trực tiếp khi nó được tạo ra; như tập tin có thể được truy xuất chỉ trong một cách không đổi với khai báo của nó. Tuy nhiên, chúng ta dễ dàng mô phỏng truy xuất tuần tự trên tập tin truy xuất trực tiếp. Nếu chúng ta giữ một biến ***cp*** để xác định vị trí hiện tại thì chúng ta có thể mô phỏng các thao tác tập tin tuần tự như được hiển thị trong hình IX-2. Mặc dù, không đủ và không gọn để mô phỏng một tập tin truy xuất trực tiếp trên một tập tin truy xuất tuần tự.

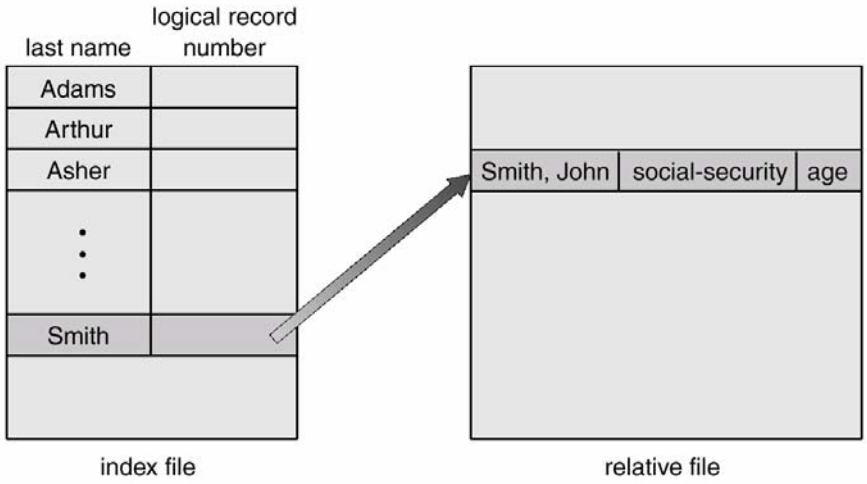
sequential access	implementation for direct access
reset	*cp* = 0;
read next	*read cp*; *cp* = *cp*+1;
write next	*write cp*; *cp* = *cp*+1;

Hình 0-2 Mô phỏng truy xuất tuần tự trên truy xuất trực tiếp

IV.3 Các phương pháp truy xuất khác

Các phương pháp truy xuất khác có thể được xây dựng trên cơ sở của phương pháp truy xuất trực tiếp. Các phương pháp khác thường liên quan đến việc xây dựng chỉ mục cho tập tin. Chỉ mục chứa các con trỏ chỉ tới các khối khác. Để tìm một mẫu tin trong tập tin, trước hết chúng ta tìm chỉ mục và sau đó dùng con trỏ để truy xuất tập tin trực tiếp và tìm mẫu tin mong muốn.

Với những tập tin lớn, chỉ mục tập tin có thể trở nên quá lớn để giữ trong bộ nhớ. Một giải pháp là tạo chỉ mục cho tập tin chỉ mục. Tập tin chỉ mục chính chứa các con trỏ chỉ tới các tập tin chỉ mục thứ cấp mà nó chỉ tới các thành phần dữ liệu thật sự.



Hình 0-3 Thí dụ về chỉ mục và các tập tin liên quan

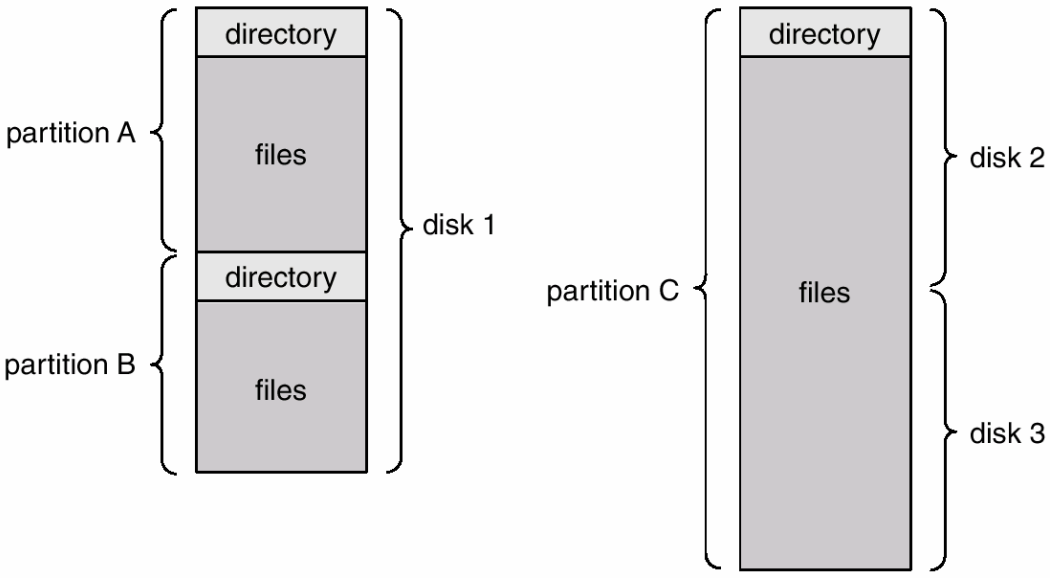
Thuviendientu.org

V Cấu trúc thư mục

Các hệ thống tập tin của máy tính có thể rất lớn về số lượng. Một số hệ thống lưu trữ hàng triệu tập tin trên các terabytes đĩa. Để quản lý tất cả dữ liệu này, chúng ta cần tổ chức lại chúng. Việc tổ chức này thường được thực hiện hai phần.

Thứ nhất, đĩa được chia thành một hay nhiều phân khu (partition) hay phân vùng (volumes). Điển hình, mỗi đĩa trên hệ thống chứa ít nhất một phân khu. Phân khu này là cấu trúc cấp thấp mà các tập tin và thư mục định vị. Thỉnh thoảng các phân khu được dùng để cung cấp nhiều vùng riêng rẻ trong một đĩa, mỗi phân khu được xem như một thiết bị lưu trữ riêng, trái lại các hệ thống khác cho phép các phân khu có dung lượng lớn hơn một đĩa để nhóm các đĩa vào một cấu trúc luận lý và cấu trúc tập tin, và có thể bỏ qua hoàn toàn những vấn đề cấp phát không gian vật lý cho các tập tin. Cho lý do này, các phân khu có thể được xem như các đĩa ảo. Các phân khu cũng có thể lưu trữ nhiều hệ điều hành, cho phép hệ thống khởi động và chạy nhiều hơn một hệ điều hành.

Thứ hai, mỗi phân khu chứa thông tin về các tập tin trong nó. Thông tin này giữ trong những mục từ trong một thư mục thiết bị hay bảng mục lục phân vùng (volume table of contents). Thư mục thiết bị (được gọi đơn giản là thư mục) ghi thông tin-như tên, vị trí, kích thước và kiểu-đối với tất cả tập tin trên phân khu (như hình IX-4).



Hình 0-4 tổ chức hệ thống tập tin điển hình

Thư mục có thể được hiển thị như một bảng danh biểu dịch tên tập tin thành các mục từ thư mục. Các thư mục có thể được tổ chức trong nhiều cách. Chúng ta muốn có thể chèn mục từ, xoá mục từ, tìm kiếm một mục từ và liệt kê tất cả mục từ trong thư mục. Trong phần này, chúng ta xem xét nhiều cơ chế định nghĩa cấu trúc luận lý của hệ thống thư mục. Khi xem xét một cấu trúc thư mục cụ thể, chúng ta cần nhớ các thao tác được thực hiện trên một thư mục.

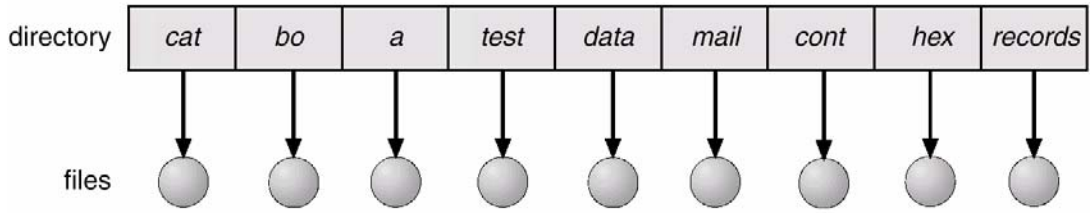
Thuviendientu.org

- **Tìm kiếm tập tin**: chúng ta cần tìm trên cấu trúc thư mục để xác định mục từ cho một tập tin cụ thể.
- **Tạo tập tin**: một tập tin mới cần được tạo và được thêm tới thư mục.
- **Xoá tập tin**: khi một tập tin không còn cần, chúng ta muốn xoá nó ra khỏi thư mục.
- **Liệt kê thư mục**: chúng ta có thể liệt kê các tập tin trong thư mục và nội dung của mục từ thư mục cho mỗi tập tin trong danh sách.
- **Đổi tên tập tin**: vì tên tập tin biểu diễn nội dung của nó đối với người dùng, tên có thể thay đổi khi nội dung hay việc sử dụng tập tin thay đổi. Đổi tên tập tin có thể cho phép vị trí của nó trong cấu trúc thư mục được thay đổi.
- **Duyệt hệ thống tập tin**: chúng ta muốn truy xuất mỗi thư mục và mỗi tập tin trong cấu trúc thư mục.

Chúng ta sẽ mô tả các cơ chế thông dụng nhất để định nghĩa cấu trúc luận lý của một thư mục.

V.1 Cấu trúc thư mục dạng đơn cấp

Cấu trúc thư mục đơn giản nhất là thư mục đơn cấp. Tất cả tập tin được chứa trong cùng thư mục như được hiển thị trong hình IX-5 dưới đây:



Hình 0-5 Thư mục đơn cấp

Tuy nhiên, thư mục đơn cấp có nhiều hạn chế khi số lượng tập tin tăng hay khi hệ thống có nhiều hơn một người dùng. Vì tất cả tập tin được chứa trong cùng thư mục, chúng phải có tên khác nhau. Nếu hai người dùng đặt tên tập tin dữ liệu của họ là test thì qui tắc tên duy nhất bị xung đột. Mặc dù các tên tập tin thường được chọn để phản ánh nội dung của tập tin, chúng thường bị giới hạn chiều dài. Hệ điều hành MS-DOS cho phép chỉ 11 ký tự cho tên; UNIX cho phép 255 ký tự.

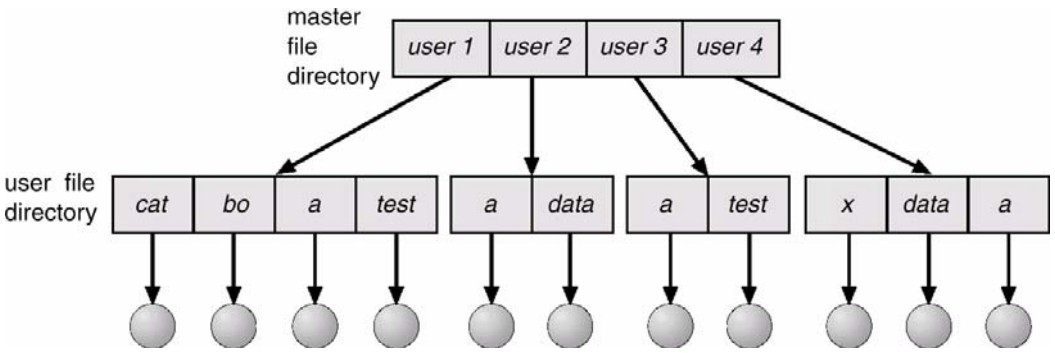
Người dùng trên thư mục đơn cấp có thể gặp phải khó khăn để nhớ tên của tất cả tập tin, khi số tập tin tăng. Nếu trên một hệ thống máy tính có hàng trăm tập tin thì việc ghi lại vết của quá nhiều tập tin là một tác vụ nặng nề.

V.2 Cấu trúc thư mục dạng hai cấp

Một thư mục đơn cấp dẫn đến sự lẫn lộn giữa tên các tập tin của nhiều người dùng khác nhau. Giải pháp chuẩn là tạo một thư mục riêng cho mỗi người dùng.

Trong cấu trúc thư mục hai cấp, mỗi người dùng có thư mục tập tin riêng cho họ (user file directory-UFD). Mỗi UFD có một cấu trúc tương tự nhưng các danh sách chứa các tập tin của một người dùng. Khi công việc của người dùng bắt đầu hay người dùng đăng nhập, thư mục tập tin chính của hệ thống (master file directory) được tìm kiếm. MFD được lập chỉ mục bởi tên người dùng hay số tài khoản và mỗi mục từ chỉ tới UFD cho người dùng đó (như hình IX-6)

Thuviendientu.org



Hình 0-6 thư mục hai cấp

Khi người dùng tham khảo tới một tập tin cụ thể, chỉ UFD của chính người dùng đó được tìm kiếm. Do đó, các người dùng khác nhau có thể có các tập tin với cùng một tên, với điều kiện là tất cả tên tập tin trong mỗi UFD là duy nhất.

Để tạo một tập tin cho một người dùng, hệ điều hành chỉ tìm UFD của người dùng đó để xác định một tập tin khác cùng tên có tồn tại hay không. Để xóa một tập tin, hệ điều hành giữ lại việc tìm kiếm của nó tới UFD cục bộ; do đó, nó không thể xóa nhầm tập tin của người dùng khác có cùng tên.

Các thư mục người dùng phải được tạo và xóa khi cần thiết. Một chương trình hệ thống đặc biệt được chạy với tên người dùng hợp lý và thông tin tài khoản. Chương trình này tạo một UFD mới và thêm một mục từ cho nó tới MFD. Việc thực thi chương trình này có thể bị giới hạn bởi người quản trị hệ thống.

Mặc dù cấu trúc thư mục hai cấp giải quyết vấn đề xung đột tên nhưng nó cũng có những bất lợi. Cấu trúc này cô lập một người dùng từ người dùng khác. Việc cô lập này là lợi điểm khi các người dùng hoàn toàn độc lập nhau nhưng sẽ bất lợi khi các người dùng muốn hợp tác trên một số công việc và để truy xuất các tập tin của người dùng khác. Một số hệ thống đơn giản không cho phép tập tin người dùng cục bộ được truy xuất bởi người dùng khác.

Nếu truy xuất được cho phép, một người dùng phải có khả năng đặt tên một tập tin trong một thư mục của người dùng khác. Để đặt tên một tập tin xác định duy nhất trong thư mục hai cấp, chúng ta phải cho cả hai tên người dùng và tên tập tin. Một thư mục hai cấp có thể được xem như một cây hay ít nhất một cây đảo ngược hay có chiều cao bằng 2. Gốc của cây là UFD. Hậu duệ trực tiếp của nó là MFD. Hậu duệ của UFD là các tập tin. Các tập tin này là lá của cây. Xác định tên người dùng và tên tập tin định nghĩa đường dẫn trong cây từ gốc (MFD) tới một lá (tập tin xác định). Do đó, tên người dùng và tên tập tin định nghĩa tên đường dẫn. Mọi tập tin trong hệ thống có một đường dẫn. Để đặt tên một tập tin duy nhất người dùng phải biết tên đường dẫn của tập tin mong muốn.

Trường hợp đặc biệt xảy ra cho các tập tin hệ thống. Các chương trình này cung cấp một phần hệ thống như: bộ nạp, bộ hợp ngữ, bộ biên dịch, các thủ tục,..thường được định nghĩa như các tập tin. Khi các lệnh tương ứng được gọi tới hệ điều hành, các tập tin này được đọc bởi bộ nạp và được thực thi. Một số bộ thông dịch lệnh hoạt động bằng cách xem lệnh như là tên tập để nạp và thực thi. Với hệ thống thư mục được định nghĩa hiện tại, tên tập tin này được tìm kiếm trong UFD hiện hành. Một giải pháp cho vấn đề này là chép các tập tin hệ thống vào mỗi UFD. Tuy nhiên, chép tất cả tập tin hệ thống sẽ lãng phí lượng lớn không gian.

Giải pháp chuẩn là làm phức tạp thủ tục tìm kiếm một chút. Một thư mục người dùng đặc biệt được định nghĩa để chứa các tập tin hệ thống (thí dụ, user0). Bất cứ khi nào một tên tập tin được cho để được nạp, trước tiên hệ điều hành tự tìm thư mục người dùng cục bộ. Nếu không tìm thấy, hệ điều hành tự tìm trong thư mục

Thuviendientu.org

người dùng đặc biệt này. Một chuỗi các thư mục được tìm khi một tập tin được đặt tên là **đường dẫn tìm kiếm**. Ý tưởng này có thể được mở rộng, đường dẫn tìm kiếm chứa danh sách các thư mục không giới hạn để tìm khi tên một lệnh được cho. Phương pháp này được dùng nhiều nhất trong UNIX và MS-DOS.

V.3 Cấu trúc thư mục dạng cây

Thư mục cấu trúc cây là trường hợp tổng quát của thư mục hai cấp. Sự tổng quát này cho phép người dùng tạo thư mục con và tổ chức các tập tin của họ. Thí dụ, hệ thống MS-DOS có cấu trúc cây. Thật vậy, một cây là cấu trúc thư mục phổ biến nhất. Cây có thư mục gốc. Mỗi tập tin trong hệ thống có tên đường dẫn duy nhất. Tên đường dẫn là đường dẫn từ gốc xuống tất cả thư mục con tới tập tin xác định.

Một thư mục (hay thư mục con) chứa tập hợp các tập tin hay thư mục con. Một thư mục đơn giản là tập tin nhưng nó được đối xử trong một cách đặc biệt. Tất cả thư mục có cùng định dạng bên trong. Một bit trong mỗi mục từ thư mục định nghĩa mục từ như một tập tin (0) hay như một thư mục con (1). Các lời gọi hệ thống đặc biệt được dùng để tạo và xoá thư mục.

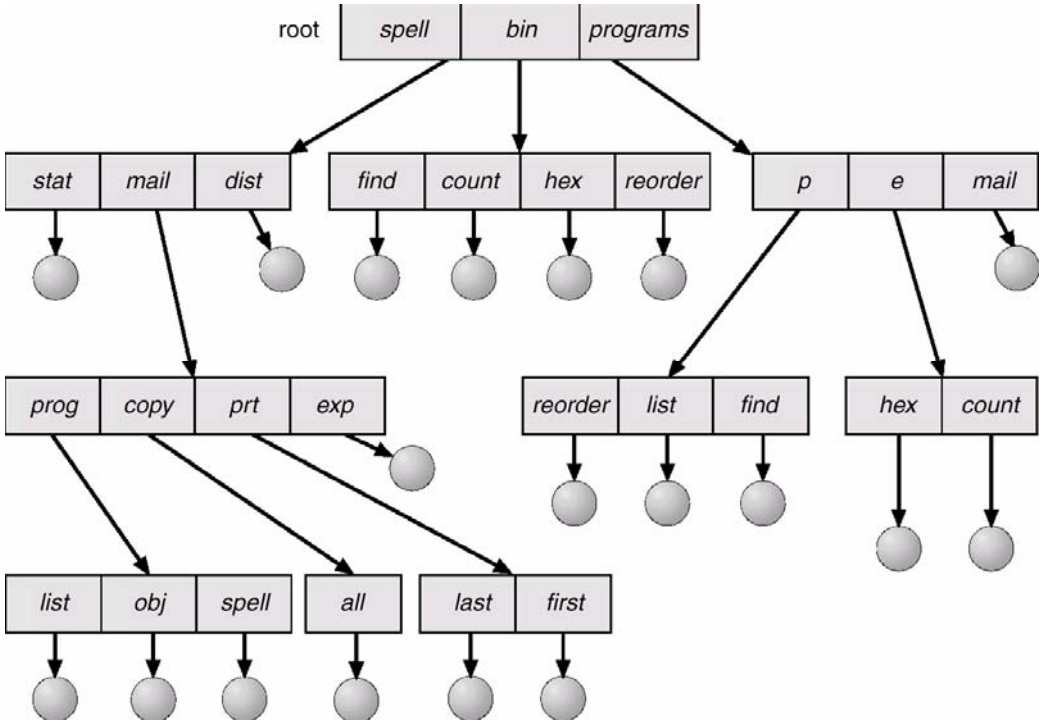
Thường thì mỗi người dùng có ***thư mục hiện hành***. Thư mục hiện hành chứa hầu hết các tập tin người dùng hiện đang quan tâm. Khi tham khảo được thực hiện tới tập tin, thư mục hiện hành được tìm. Nếu một tập tin được yêu cầu mà nó không có trong thư mục hiện hành thì người dùng phải xác định tên đường dẫn hay chuyển thư mục hiện hành tới thư mục quản lý tập tin đó. Để thay đổi thư mục, một lời gọi hệ thống được cung cấp kèm theo tên thư mục như là tham số và dùng nó để định nghĩa lại thư mục hiện hành. Do đó, người dùng có thể thay đổi thư mục hiện hành bất cứ khi nào người dùng muốn.

Thư mục hiện hành khởi đầu của người dùng được gán khi công việc người dùng bắt đầu hay người dùng đăng nhập vào hệ thống. Hệ điều hành tìm tập tin tính toán để xác định mục từ cho người dùng này. Trong tập tin tính toán là con trỏ chỉ tới thư mục khởi đầu của người dùng. Con trỏ này được chép tới một biến cục bộ cho người dùng xác định thư mục hiện hành khởi đầu.

Tên đường dẫn có hai kiểu: ***tên đường dẫn tuyệt đối*** và ***tên đường dẫn tương đối***. Một đường dẫn tuyệt đối bắt đầu từ gốc và theo sau là đường dẫn xuống tới tập tin xác định, cho tên các thư mục trên đường dẫn. Tên đường dẫn tương đối định nghĩa một đường dẫn từ thư mục hiện hành. Thí dụ, trong hệ thống tập tin có cấu trúc cây như hình IX-7, nếu thư mục hiện hành là *root/spell/mail* thì tên đường dẫn tương đối *prt/first* tham chiếu tới cùng tập tin nhưng tên đường dẫn tuyệt đối *root/spell/mail/prt/first*.

Quyết định một chính sách trong cấu trúc thư mục cây là cách để quản lý việc xoá một thư mục. Nếu một thư mục rỗng, mục từ của nó trong thư mục chứa bị xoá. Tuy nhiên, giả sử thư mục bị xoá không rỗng, nhưng chứa nhiều tập tin và thư mục con; một trong hai tiếp cận có thể được thực hiện. Một số hệ thống như MS-DOS sẽ không xoá một thư mục nếu nó không rỗng. Do đó, để xoá một thư mục, người dùng trước hết phải xoá tất cả tập tin trong thư mục đó. Nếu bất cứ thư mục con tồn tại, thủ tục này phải được áp dụng đệ qui tới chúng để mà chúng có thể bị xoá. Tiếp cận này dẫn đến lượng công việc lớn.

Thuviendientu.org



Hình 0-7 cấu trúc thư mục dạng cây

Một tiếp cận khác được thực hiện bởi lệnh **rm** của UNIX cung cấp tuỳ chọn mà khi một yêu cầu được thực hiện để xoá một thư mục, tất cả tập tin và thư mục con của thư mục đó cũng bị xoá. Tiếp cận này tương đối đơn giản để cài đặt; chọn lựa này là một chính sách. Chính sách sau đó tiện dụng hơn nhưng nguy hiểm hơn vì toàn bộ cấu trúc thư mục có thể bị xoá với một lệnh. Nếu lệnh đó được cấp phát bị lỗi, một số lượng lớn tập tin và thư mục cần được phục hồi từ các băng từ sao lưu.

Với một hệ thống thư mục cấu trúc cây, người dùng có thể truy xuất tới các tập tin của họ và các tập tin của người dùng khác. Thí dụ, người dùng B có thể truy xuất các tập tin của người dùng A bằng cách xác định tên đường dẫn của chúng. Người dùng B có thể xác định tên đường dẫn tương đối hay tuyệt đối. Người dùng B có thể chuyển thư mục hiện hành tới thư mục của người dùng A và truy xuất các tập tin bằng tên của chúng. Một số hệ thống cũng cho phép người dùng định nghĩa đường dẫn tìm kiếm của chính họ. Trong trường hợp này, người dùng B có thể định nghĩa đường dẫn tìm kiếm của mình là (1) thư mục cục bộ của mình, (2) thư mục tập tin hệ thống và (3) thư mục của người dùng A, theo thứ tự đó. Tập tin có thể được tham khảo đơn giản bằng tên với điều kiện tên tập tin của người dùng A không xung đột với tên của một tập tin cục bộ hay tập tin hệ thống.

V.4 Cấu trúc thư mục dạng đồ thị không chứa chu trình

Xét hai người lập trình đang làm việc trên một dự án chung. Các tập tin gắn với dự án đó có thể được lưu trong thư mục con, tách rời chúng từ các dự án khác và các tập tin của hai người lập trình. Nhưng vì cả hai người lập trình có trách nhiệm ngang nhau trong dự án, cả hai muốn thư mục con ở trong các thư mục của chính họ. Thư mục con nên được chia sẻ. Một thư mục hay tập tin sẽ tồn tại trong hệ thống tập tin trong hai (hay nhiều hơn) nơi tại một thời điểm.

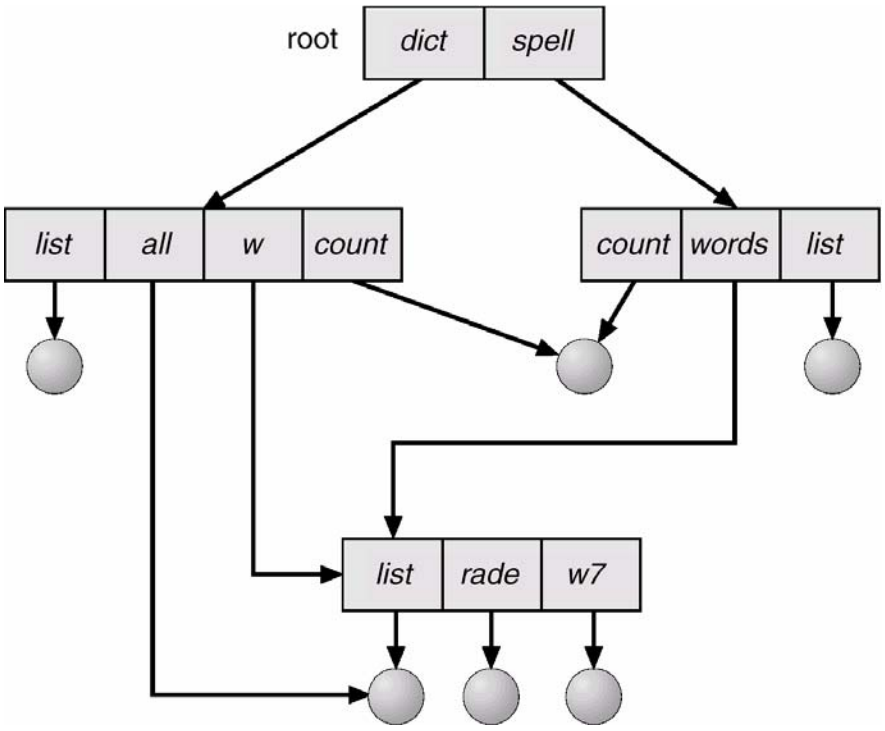
Cấu trúc cây ngăn cản việc chia sẻ các tập tin và thư mục. Một **đồ thị không chứa chu trình** (acyclic graph) cho phép thư mục chia sẻ thư mục con và tập tin (hình

Thuviendientu.org

IX-8). Cùng tập tin và thư mục con có thể ở trong hai thư mục khác nhau. Một đồ thị không chứa chu trình là trường hợp tổng quát của cơ chế thư mục có cấu trúc cây.

Một tập tin (hay thư mục) được chia sẻ không giống như hai bản sao của một tập tin. Với hai bản sao, mỗi người lập trình có thể thích hiển thị bản sao hơn bản gốc, nhưng nếu một người lập trình thay đổi nội dung tập tin, những thay đổi sẽ không xuất hiện trong bản sao của người còn lại. Với một tập tin được chia sẻ, chỉ một tập tin thực sự tồn tại vì thế bất cứ sự thay đổi được thực hiện bởi một người này lập tức nhìn thấy bởi người dùng khác. Việc chia sẻ là rất quan trọng cho các thư mục con; một tập tin mới được tạo bởi người này sẽ tự động xuất hiện trong tất cả thư mục con được chia sẻ.

Khi nhiều người đang làm việc như một nhóm, tất cả tập tin họ muốn chia sẻ có thể đặt vào một thư mục. Các UFD của tất cả thành viên trong nhóm chứa thư mục của tập tin được chia sẻ như một thư mục con. Ngay cả khi có một người dùng, tổ chức tập tin của người dùng này yêu cầu rằng một số tập tin được đặt vào các thư mục con khác nhau. Thí dụ, một chương trình được viết cho một dự án nên đặt trong thư mục của tất cả chương trình và trong thư mục cho dự án đó.



Hình 0-8 cấu trúc đồ thị không chứa chu trình

Các tập tin và thư mục con được chia sẻ có thể được cài đặt trong nhiều cách. Cách thông dụng nhất được UNIX dùng là tạo một mục từ thư mục được gọi là liên kết. Một liên kết là một con trỏ chỉ tới một tập tin hay thư mục con khác. Thí dụ, một liên kết có thể được cài đặt như tên đường dẫn tuyệt đối hay tương đối. Khi một tham chiếu tới tập tin được thực hiện, chúng ta tìm kiếm thư mục. Nếu mục từ thư mục được đánh dấu như một liên kết thì tên tập tin thật sự (hay thư mục) được cho. Chúng ta phân giải liên kết bằng cách sử dụng tên đường dẫn để định vị tập tin thật sự. Những liên kết được xác định dễ dàng bởi định dạng trong mục từ thư mục và được định rõ bằng các con trỏ gián tiếp. Hệ điều hành bỏ qua các liên kết này khi duyệt qua cây thư mục để lưu giữ cấu trúc không chứa chu trình của hệ thống.

Thuviendientu.org

Một tiếp cận khác để cài đặt các tập tin được chia sẻ là nhân bản tất cả thông tin về chúng trong cả hai thư mục chia sẻ. Do đó, cả hai mục từ là giống hệt nhau. Một liên kết rất khác từ mục từ thư mục gốc. Tuy nhiên, nhân bản mục từ thư mục làm cho bản gốc và bản sao không khác nhau. Một vấn đề chính với nhân bản mục từ thư mục là duy trì tính không đổi nếu tập tin bị sửa đổi.

Một cấu trúc thư mục đồ thị không chứa chu trình linh hoạt hơn cấu trúc cây đơn giản nhưng nó cũng phức tạp hơn. Một số vấn đề phải được xem xét cẩn thận. Một tập tin có nhiều tên đường dẫn tuyệt đối. Do đó, các tên tập tin khác nhau có thể tham chiếu tới cùng một tập tin. Trường hợp này là tương tự như vấn đề bí danh cho các ngôn ngữ lập trình. Nếu chúng ta đang cố gắng duyệt toàn bộ hệ thống tập tin-để tìm một tập tin, để tập hợp các thông tin thống kê trên tất cả tập tin, hay chép tất cả tập tin tới thiết bị lưu dự phòng-vấn đề này trở nên lớn vì chúng ta không muốn duyệt các cấu được chia sẻ nhiều hơn một lần.

Một vấn đề khác liên quan đến việc xoá. Không gian được cấp phát tới tập tin được chia sẻ bị thu hồi và sử dụng lại khi nào? một khả năng là xoá bỏ tập tin bất cứ khi nào người dùng xoá nó, nhưng hoạt động này để lại con trỏ chỉ tới một tập tin không tồn tại. Trong trường hợp xấu hơn, nếu các con trỏ tập tin còn lại chứa địa chỉ đĩa thật sự và không gian được dùng lại sau đó cho các tập tin khác, các con trỏ này có thể chỉ vào phần giữa của tập tin khác.

Trong một hệ thống mà việc chia sẻ được cài đặt bởi **liên kết biểu tượng**, trường hợp này dễ dàng quản lý hơn. Việc xoá một liên kết không cần tác động tập tin nguồn, chỉ liên kết bị xoá. Nếu chính tập tin bị xoá, không gian cho tập tin này được thu hồi, để lại các liên kết chơi vơi. Chúng ta có thể tìm các liên kết này và xoá chúng, nhưng nếu không có danh sách các liên kết được nối kết, việc tìm kiếm này sẽ tốn rất nhiều chi phí. Một cách khác, chúng ta có thể để lại các liên kết này cho đến khi nó được truy xuất. Tại thời điểm đó, chúng ta xác định rằng tập tin của tên được cho bởi liên kết không tồn tại và có thể bị lỗi để phục hồi tên liên kết; truy xuất này được đối xử như bất cứ tên tập tin không hợp lệ khác. Trong trường hợp UNIX, các liên kết biểu tượng được để lại khi một tập tin bị xoá và nó cho người dùng nhận thấy rằng tập tin nguồn đã mất hay bị thay thế. Microsoft Windows (tất cả ấn bản) dùng cùng tiếp cận.

Một tiếp cận khác đối với việc xoá là giữ lại tập tin cho tới khi tất cả tham chiếu tới nó bị xoá. Để cài đặt tiếp cận này, chúng ta phải có một số cơ chế để xác định rằng tham chiếu cuối cùng tới tập tin bị xoá. Chúng ta giữ danh sách của tất cả tham chiếu tới một tập tin (các mục từ thư mục hay các liên kết biểu tượng). Khi một liên kết hay bản sao của mục từ thư mục được thiết lập, một mục từ mới được thêm tới danh sách tham chiếu tập tin. Khi một mục từ thư mục hay liên kết bị xoá, chúng ta gỡ bỏ mục từ của nó trên danh sách. Tập tin này bị xoá khi danh sách tham chiếu tập tin của nó là rỗng.

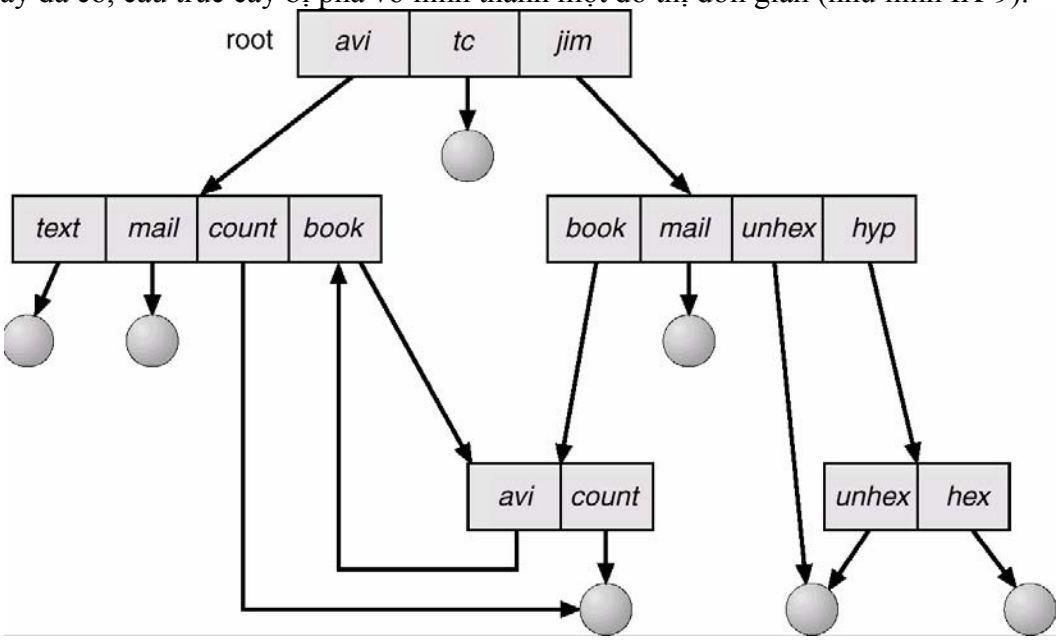
Trở ngại với tiếp cận này là kích thước của danh sách tham chiếu thay đổi và có thể rất lớn. Tuy nhiên, chúng ta thật sự không cần giữ toàn bộ danh sách-chúng ta chỉ cần giữ số đếm của số tham chiếu. Một liên kết mới hay mục từ thư mục mới sẽ tăng số đếm tham chiếu; xoá một liên kết hay mục từ sẽ giảm số đếm. Khi số đếm là 0, tập tin có thể được xoá; không còn tham chiếu nào tới nó. Hệ điều hành UNIX dùng tiếp cận này cho các liên kết không biểu tượng (hay liên kết cứng), giữ một số đếm tham chiếu trong khối thông tin tập tin (hay inode). Bằng cách ngăn cản hiệu quả nhiều tham chiếu tới các thư mục, chúng ta duy trì cấu trúc đồ thị không chứa chu trình.

Thuviendientu.org

Để tránh vấn đề này, một số hệ thống không cho phép thư mục hay liên kết được chia sẻ. Thí dụ, trong MS-DOS, cấu trúc thư mục là một cấu trúc cây hơn là đồ thị không chứa chu trình.

V.5 Cấu trúc thư mục dạng đồ thị tổng quát

Một vấn đề lớn trong việc dùng cấu trúc đồ thị không chứa chu trình là đảm bảo rằng không có chu trình trong đồ thị. Nếu chúng ta bắt đầu với thư mục hai cấp và cho phép người dùng tạo thư mục con, một thư mục có cấu trúc cây tạo ra. Dễ thấy rằng thêm các tập tin và thư mục con mới tới một thư mục có cấu trúc cây đã có vẫn bảo đảm tính tự nhiên của cấu trúc cây. Tuy nhiên, khi chúng ta liên kết một thư mục cấu trúc cây đã có, cấu trúc cây bị phá vỡ hình thành một đồ thị đơn giản (như hình IX-9).



Hình 0-9 thư mục đồ thị tổng quát

Một lợi điểm chính của đồ thị không chứa chu trình là tương đối đơn giản trong giải thuật duyệt đồ thị và xác định khi không có tham chiếu nữa tới tập tin. Chúng ta muốn tránh duyệt các phần được chia sẻ của đồ thị không chứa chu trình hai lần để tăng năng lực. Nếu chúng ta chỉ tìm một thư mục con được chia sẻ cho một tập tin xác định, chúng ta muốn tránh việc tìm kiếm thư mục con đó một lần nữa; tìm kiếm lần hai sẽ lãng phí thời gian.

Nếu các chu trình được cho phép tồn tại trong thư mục, chúng ta muốn tránh tìm kiếm bất cứ thành phần nào hai lần vì tính đúng đắn cũng như năng lực. Một giải thuật được thiết kế nghèo nàn dẫn tới vòng lặp vô tận trên các chu trình. Một giải pháp là giới hạn số lượng thư mục sẽ được truy xuất trong quá trình tìm kiếm.

Một vấn đề tương tự tồn tại khi chúng ta cố gắng xác định khi nào một tập tin có thể bị xoá. Như với cấu trúc thư mục đồ thị không chứa chu trình, một giá trị 0 trong số đếm tham chiếu có nghĩa là không còn tham chiếu nữa tới tập tin hay thư mục và tập tin có thể bị xoá. Tuy nhiên, khi có chu trình tồn tại, số đếm tham chiếu khác 0 ngay cả khi không còn tham chiếu tới thư mục hay tập tin. Sai sót này dẫn tới khả năng tham chiếu chính nó (hay chu trình) trong cấu trúc thư mục. Trong trường hợp này, chúng ta cần dùng cơ chế thu dọn rác (garbage colection) để xác định khi tham chiếu cuối cùng bị xoá và không gian đĩa có thể được cấp phát lại. Thu dọn rác

Thuviendientu.org

liên quan đến việc duyệt toàn bộ hệ thống tập tin, đánh dấu mọi thứ có thể được truy xuất. Sau đó, duyệt lần hai tập hợp mọi thứ không được đánh dấu trên danh sách không gian trống. Tuy nhiên, thu dọn rác cho một đĩa dựa trên hệ thống tập tin rất mất thời gian và do đó hiếm khi được thực hiện.

Thu dọn rác cần thiết chỉ vì có chu trình trong đồ thị. Do đó, cấu trúc đồ thị không chứa chu trình dễ hơn nhiều. Khó khăn là tránh chu trình khi các liên kết mới được thêm vào cấu trúc. Chúng ta biết như thế nào và khi nào một liên kết mới sẽ hình thành chu trình? Có nhiều giải thuật phát hiện các chu trình trong đồ thị; tuy nhiên chi phí tính toán cao, đặc biệt khi đồ thị ở trên đĩa lưu trữ. Một giải thuật đơn giản hơn trong trường hợp đặc biệt của thư mục và liên kết là bỏ qua liên kết khi duyệt qua thư mục. Chu trình có thể tránh được và không có chi phí thêm xảy ra.

VI Gắn hệ thống tập tin

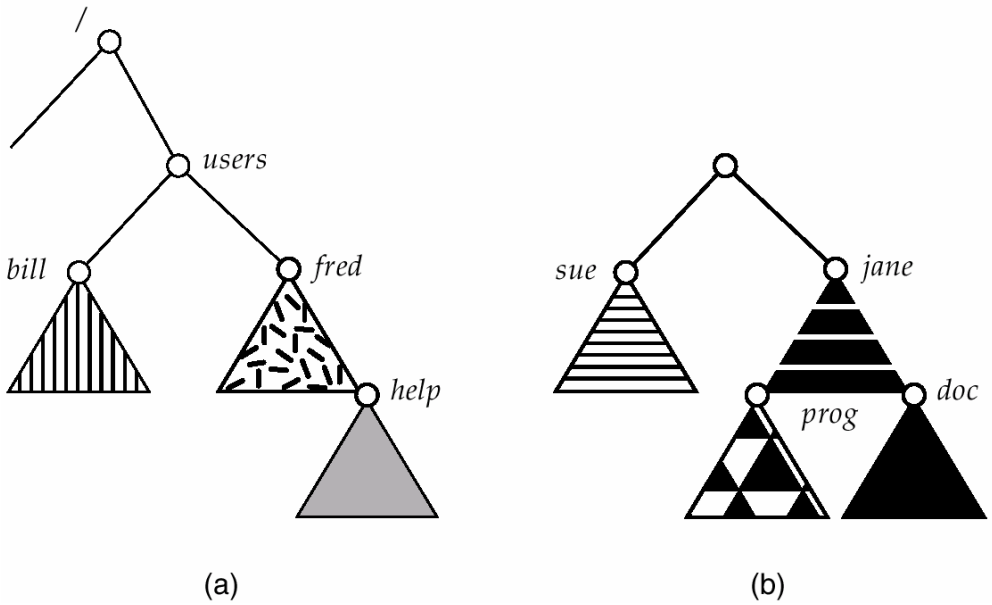
Giống như một tập tin phải được mở trước khi nó được dùng, một hệ thống tập tin phải được ***gắn vào*** (mounted) trước khi nó có thể sẵn dùng cho các quá trình trên hệ thống. Đặc biệt hơn, cấu trúc thư mục có thể được xây dựng trên nhiều phân khu, mà phải được gắn vào để làm cho chúng sẵn dùng trong không gian tên hệ thống tập tin.

Thủ tục gắn vào không phức tạp. Hệ điều hành được cho tên của thiết bị và vị trí trong cấu trúc tập tin tại nơi nó được gán vào hệ thống tập tin (điểm gắn-mount point). Điển hình, một điểm gắn thường là thư mục rỗng nơi hệ thống tập tin sẽ được gắn vào. Thí dụ, trên hệ thống UNIX, một hệ thống tập tin chứa các thư mục dành riêng cho người dùng (home directory) có thể được gắn vào như /home; sau đó để truy xuất tới cấu trúc tập tin trong hệ thống tập tin đó, người dùng có thể đến trước các tên thư mục /home, như trong /home/jane. Gắn vào hệ thống tập tin đó dưới /users sẽ dẫn tới tên đường dẫn /users/jane để đạt cùng thư mục.

Tiếp theo, hệ điều hành kiểm tra thiết bị chứa một hệ thống tập tin hợp lệ không. Nó thực hiện như thế bằng cách yêu cầu trình điều khiển thiết bị đọc thư mục thiết bị và kiểm tra thư mục có định dạng như mong muốn. Cuối cùng, hệ điều hành ghi nhận trong cấu trúc thư mục của nó rằng hệ thống tập tin được gắn vào tại điểm gắn xác định. Cơ chế này cho phép hệ điều hành duyệt cấu trúc thư mục của nó, chuyển qua lại giữa các hệ thống tập tin một cách hợp lý.

Để hiển thị việc gắn tập tin, xem xét hệ thống tập tin được mô tả như hình IX-10, ở đây hình này hình tam giác hiển thị cây con của các thư mục đang quan tâm. Trong hình (a) hiển thị hệ thống tập tin đã có, trong hình (b) hiển thị một phân khu chưa được gắn vào /device/dsk. Tại điểm này, chỉ các tập tin trên hệ thống tập tin đã tồn tại mới có thể được truy xuất. Trong hình IX-11, hiển thị các ảnh hưởng của việc gắn vào phân khu này trên /device/dsk qua /users. Nếu phân khu này chưa được gắn, hệ thống tập tin được phục hồi tới trường hợp được mô tả như hình IX-10.

Thuviendientu.org



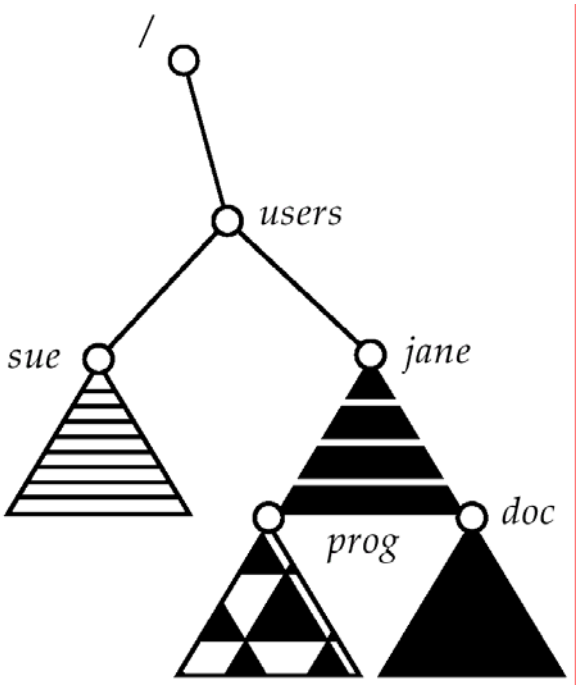
Hình 0-10 Hệ thống tập tin. (a) đã tồn tại. (b) phân khu chưa được gắn

Hệ thống này áp đặt các ngữ nghĩa để phân cấp chức năng. Thí dụ, một hệ thống không được phép vượt qua một thư mục chứa các tập tin hay làm hệ thống tập tin được gắn vào sẵn dùng tại thư mục đó và che đậy các tập tin đã có của thư mục cho đến khi hệ thống tập tin được gỡ ra (unmounted), kết thúc việc sử dụng hệ thống tập tin và cho phép truy xuất tới các tập tin trong thư mục đó.

Xét các hoạt động của hệ điều hành Macintosh. Bất cứ khi nào hệ thống gặp một đĩa tại thời điểm đầu tiên (đĩa cứng tại thời điểm khởi động hay đĩa mềm tại thời điểm chèn đĩa vào ổ), hệ điều hành Macintosh tìm kiếm một hệ thống tập tin cho thiết bị đó. Nếu nó tìm thấy một hệ thống tập tin, nó tự động gắn hệ thống tập tin tại mức gốc, thêm biểu tượng thư mục trên màn hình được ghi nhãn với tên của hệ thống tập tin (như được lưu trong thư mục thiết bị). Sau đó, người dùng có thể nhấp vào biểu tượng để hiển thị hệ thống tập tin vừa được gắn vào.

Họ hệ điều hành Microsoft Windows (95, 98, NT, 2000) duy trì một cấu trúc thư mục hai cấp mở rộng với các thiết bị và phân khu được gán một ký tự ổ đĩa. Các phân khu có cấu trúc thư mục đồ thị tổng quát được gắn liền với ký tự ổ đĩa. Sau đó, đường dẫn tới một tập tin xác định có dạng ***ký tự ổ đĩa:\đường dẫn\tới\tập tin***. Những hệ điều hành này tự phát hiện tất cả thiết bị và gắn vào tất cả hệ thống tập tin được định vị tại thời điểm khởi động. Trong một vài hệ thống, như UNIX, các lệnh gắn vào là hiện (explicit). Một tập tin cấu hình hệ thống chứa danh sách thiết bị và các điểm gắn vào để tự động gắn vào tại thời điểm khởi động, nhưng những gắn vào khác phải được thực hiện thủ công.

Thuviendientu.org



Hình 0-11 Điểm gắn vào

VII Chia sẻ tập tin

VII.1 Nhiều người dùng

Khi hệ điều hành cung cấp nhiều người dùng, các vấn đề chia sẻ tập tin, đặt tên tập tin, và bảo vệ tập tin trở nên quan trọng. Đối với một cấu trúc thư mục cho phép các tập tin được chia sẻ bởi nhiều người dùng, hệ thống phải dàn xếp việc chia sẻ tập tin. Mặc định, hệ thống có thể cho phép một người dùng truy xuất các tập tin của người dùng khác hay nó yêu cầu rằng một người dùng gán quyền truy xuất cụ thể tới các tập tin.

Để cài đặt chia sẻ và bảo vệ, hệ thống phải duy trì nhiều thuộc tính tập tin và thư mục hơn trên hệ thống đơn người dùng. Mặc dù, có nhiều tiếp cận cho chủ đề này, hầu hết các hệ thống đưa ra khái niệm người sở hữu (owner) và nhóm (group) tập tin/thư mục. Người sở hữu là người dùng có thể thay đổi các thuộc tính, gán truy xuất, và có hầu hết điều khiển qua tập tin và thư mục. Thuộc tính nhóm của tập tin được dùng để định nghĩa tập hợp con các người dùng có thể chia sẻ truy xuất tới tập tin.

VII.2 Hệ thống tập tin ở xa

Sự phát triển của mạng cho phép giao tiếp giữa các máy tính ở xa. Mạng cho phép chia sẻ các tài nguyên trải rộng trong một khu hay thậm chí khắp thế giới. Một tài nguyên quan trọng để chia sẻ là dữ liệu ở dạng tập tin. Thông quan sự phát triển mạng và công nghệ tập tin, phương pháp chia sẻ tập tin thay đổi. Trong phương pháp đầu tiên được cài đặt, người dùng truyền tập tin giữa các máy tính bằng chương trình gọi là ftp. Phương pháp quan trọng thứ hai là hệ thống tập tin phân tán (distributed file system-DFS) trong đó, các thư mục ở xa có thể được nhìn thấy từ máy cục bộ. Trong một số cách, phương pháp thứ ba, World Wide Web là một sự trở lại của phương

Thuviendientu.org

pháp đầu tiên. Một trình duyệt được yêu cầu để đạt được truy xuất các tập tin từ xa và các thao tác riêng biệt được dùng để truyền tập tin.

VIII Bảo vệ

Khi thông tin được giữ trong một hệ thống máy tính, chúng ta muốn giữ nó an toàn từ hỏng hóc vật lý (khả năng tin cậy) và những truy xuất không hợp lý (bảo vệ). Khả năng tin cậy thường được cung cấp bởi nhân bản các tập tin. Nhiều máy tính có các chương trình hệ thống tự động chép các tập tin trên đĩa tới băng từ tại những khoảng thời gian đều đặn để duy trì một bản sao. Hệ thống tập tin có thể bị hỏng bởi phần cứng, thay đổi đột ngột về điện, nhiệt độ tăng cao,..các tập tin có thể bị xoá do rủi ro. Những con bọ (bugs) trong phần mềm hệ thống tập tin có thể làm cho nội dung tập tin bị mất.

Bảo vệ có thể được cung cấp trong nhiều cách. Đối với một hệ thống người dùng đơn nhỏ, chúng ta có thể cung cấp sự bảo vệ bằng cách gỡ bỏ các đĩa mềm và khoá chúng trong ngăn kéo. Tuy nhiên, trong hệ thống đa người dùng, những cơ chế khác được yêu cầu.

VIII.1 Các kiểu truy xuất

Nhu cầu bảo vệ tập tin là một kết quả trực tiếp của khả năng để truy xuất tập tin. Hệ thống không cho phép truy xuất các tập tin của người dùng khác thì không cần bảo vệ. Do đó, chúng ta có thể cung cấp sự bảo vệ toàn diện bằng cách cấm truy xuất. Một cách khác, chúng ta có thể cung cấp truy xuất thoải mái và không cần bảo vệ. Cả hai tiếp cận là quá cực đoan cho các sử dụng thông thường. Yêu cầu truy xuất được kiểm soát là gì?

Các cơ chế bảo vệ cung cấp truy xuất được kiểm soát bằng cách giới hạn kiểu truy xuất tập tin có thể thực hiện. Truy xuất được phép hay bị từ chối phụ thuộc nhiều yếu tố, một trong những yếu tố là kiểu truy xuất được yêu cầu. Nhiều kiểu thao tác có thể được kiểm soát:

- Đọc (Read): đọc từ tập tin.
- Viết (Write): viết hay viết lại tập tin.
- Thực thi (Execute): nạp tập tin vào bộ nhớ và thực thi nó.
- Chèn cuối (Append): viết thông tin mới vào cuối tập tin.
- Xoá (Delete): xoá tập tin và giải phóng không gian để có thể dùng lại
- Liệt kê (List): liệt kê tên và thuộc tính của tập tin

Những thao tác khác như đổi tên, chép, soạn thảo tập tin có thể cũng được kiểm soát. Tuy nhiên, đối với nhiều hệ thống, các chức năng cao hơn này có thể được cài đặt bởi một chương trình hệ thống thực hiện lời gọi hệ thống cấp thấp hơn. Bảo vệ được cung cấp chỉ tại cấp thấp hơn. Thí dụ, chép một tập tin có thể được cài đơn giản bởi một chuỗi các yêu cầu đọc. Trong trường hợp này, người dùng với truy xuất đọc cũng có thể làm cho tập tin được chép, in,..

Nhiều cơ chế bảo vệ được đề nghị. Mỗi cơ chế có lợi điểm và nhược điểm và phải phù hợp cho ứng dụng được dự định. Một hệ thống máy tính nhỏ được dùng chỉ bởi một vài thành viên của một nhóm nghiên cứu có thể không cần cùng kiểu bảo vệ như máy tính của công ty lớn.

Thuviendientu.org

VIII.2 Kiểm soát truy xuất

Tiếp cận thông dụng nhất đối với vấn đề bảo vệ là thực hiện truy xuất phụ thuộc định danh của người dùng. Những người dùng khác nhau cần các kiểu truy xuất khác nhau tới một tập tin hay thư mục. Cơ chế thông dụng nhất để cài đặt truy xuất phụ thuộc định danh là gắn với mỗi tập tin và thư mục một danh sách kiểm soát truy xuất (access-control list-ACL) xác định tên người dùng và kiểu truy xuất được phép cho mỗi người dùng. Khi một người dùng yêu cầu truy xuất tới một tập tin cụ thể, hệ điều hành kiểm tra danh sách truy xuất được gắn tới tập tin đó. Nếu người dùng đó được liệt kê cho truy xuất được yêu cầu, truy xuất được phép. Ngược lại, sự vi phạm bảo vệ xảy ra và công việc của người dùng đó bị từ chối truy xuất tới tập tin.

Tiếp cận này có lợi điểm của việc cho phép các phương pháp truy xuất phức tạp. Vấn đề chính với các danh sách truy xuất là chiều dài của nó. Nếu chúng ta muốn cho phép mọi người dùng đọc một tập tin, chúng ta phải liệt kê tất cả người dùng với truy xuất đọc. Kỹ thuật này có hai kết quả không mong muốn:

- Xây dựng một danh sách như thế có thể là một tác vụ dài dòng và không đáng, đặc biệt nếu chúng ta không biết trước danh sách người dùng trong hệ thống.
- Mục từ thư mục, trước đó có kích thước cố định, bây giờ có kích thước thay đổi, dẫn đến việc quản lý không gian phức tạp hơn

Những vấn đề này có thể được giải quyết bởi việc dùng ấn bản cô đọng của danh sách truy xuất.

Để cô đọng chiều dài của danh sách kiểm soát truy xuất, nhiều hệ thống nhận thấy 3 sự phân cấp người dùng trong nối kết với mỗi tập tin:

- Người sở hữu (Owner): người dùng tạo ra tập tin đó
- Nhóm (Group): tập hợp người dùng đang chia sẻ tập tin và cần truy xuất tương tự là nhóm hay nhóm làm việc
- Người dùng khác (universe): tất cả người dùng còn lại trong hệ thống

Tiếp cận phổ biến gần đây nhất là kết hợp các danh sách kiểm soát truy xuất với người sở hữu, nhóm và cơ chế kiểm soát truy xuất được mô tả ở trên

Để cơ chế này làm việc hợp lý, các quyền và danh sách truy xuất phải được kiểm soát chặt chẽ. Kiểm soát này có thể đạt được trong nhiều cách. Thí dụ, trong hệ thống UNIX, các nhóm có thể được tạo và sửa đổi chỉ bởi người quản lý của tiện ích. Do đó, kiểm soát này đạt được thông qua giao tiếp người dùng.

Với việc phân cấp bảo vệ được giới hạn hơn, chỉ có ba trường được yêu cầu để xác định bảo vệ. Mỗi trường thường là một tập hợp các bit, mỗi trường cho phép hay ngăn chặn truy xuất được gắn với nó. Thí dụ, hệ thống UNIX định nghĩa 3 trường 3 bit-rwx, ở đây r kiểm soát truy xuất đọc, w kiểm soát truy xuất viết, x kiểm soát truy xuất thực thi. Một trường riêng rẻ được giữ cho người sở hữu, cho nhóm tập tin và cho tất cả người dùng khác. Trong cơ chế này, 9 bits trên tập tin được yêu cầu để ghi lại thông tin bảo vệ.

VIII.3 Các tiếp cận bảo vệ khác

Một tiếp cận khác cho vấn đề bảo vệ là gắn mật khẩu với mỗi tập tin. Giống như truy xuất tới hệ thống máy tính thường được kiểm soát bởi một mật khẩu, truy xuất tới mỗi tập tin có thể được kiểm soát bởi một mật khẩu. Nếu các mật khẩu được chọn một cách ngẫu nhiên và thường được thay đổi thì cơ chế này có thể hiệu quả trong truy xuất có giới hạn tới tập tin cho những người dùng biết mật khẩu. Tuy nhiên, cơ chế

Thuviendientu.org

này có nhiều nhược điểm. Thứ nhất, số lượng mật khẩu mà người dùng cần nhớ quá nhiều, làm cho cơ chế này không thực tế. Thứ hai, nếu chỉ một mật khẩu được dùng cho tất cả tập tin thì một khi nó bị phát hiện tất cả tập tin có thể truy xuất. Một số hệ thống cho phép người dùng gắn một mật khẩu tới một thư mục con hơn là với từng tập tin riêng rẻ để giải quyết vấn đề này. Thứ ba, thường chỉ một mật khẩu gắn với tất cả tập tin người dùng. Do đó, bảo vệ dựa trên cơ sở tất cả hay không có gì (all-or-nothing). Để cung cấp sự bảo vệ trên cấp độ chi tiết hơn chúng ta phải dùng nhiều mật khẩu.

IX Tóm tắt

Một tập tin là một kiểu dữ liệu trừu tượng được định nghĩa và được cài đặt bởi hệ điều hành. Nó là một chuỗi các mẫu tin luận lý. Một mẫu tin luận lý có thể là một byte, một dòng (có chiều dài cố định hay thay đổi), hay có thành phần dữ liệu phức tạp hơn. Hệ điều hành có thể hỗ trợ nhiều kiểu mẫu tin khác nhau hay để sự hỗ trợ đó tới một chương trình ứng dụng.

Mỗi thiết bị trong một tập tin giữ một bảng volume nội dung hay thư mục thiết bị liệt kê vị trí các tập tin trên thiết bị. Ngoài ra, nó có ích để tạo các thư mục cho phép các tập tin được tổ chức trong thư mục đó. Một thư mục đơn cấp trong hệ thống đơn người dùng gây ra các vấn đề đặt tên vì mỗi tập tin phải có tên duy nhất. Thư mục hai cấp giải quyết vấn đề này bằng cách tạo một thư mục riêng cho mỗi người dùng. Mỗi người dùng có thư mục riêng, chứa tập tin riêng. Thư mục liệt kê các tập tin bằng tên và chứa những thông tin như vị trí tập tin trên đĩa, chiều dài, kiểu, người sở hữu, thời gian tạo, thời điểm dùng gần nhất,…

Tổng quát hóa tính tự nhiên của thư mục hai cấp là thư mục có cấu trúc cây. Thư mục có cấu trúc cây cho phép một người dùng tạo thư mục con để tổ chức các tập tin. Cấu trúc thư mục đồ thị không chứa chu trình cho phép các thư mục con và tập tin được chia sẻ nhưng tìm kiếm và xóa phức tạp. Một cấu trúc đồ thị tổng quát linh động hơn trong việc chia sẻ tập tin và thư mục, nhưng yêu cầu thu dọn rác để phục hồi không gian đĩa không được dùng.

Đĩa được phân chia thành một hay nhiều phân khu, mỗi phân khu chứa một hệ thống tập tin. Hệ thống tập tin này được gán vào cấu trúc đặt tên của hệ thống để làm cho chúng sẵn dùng. Cơ chế đặt tên khác nhau bởi các hệ điều hành khác nhau. Một khi được gán vào, các tập tin trong phân khu là sẵn dùng. Các hệ thống tập tin có thể được gỡ ra (unmount) để vô hiệu hóa truy xuất hay để bảo trì.

Chia sẻ tập tin phụ thuộc vào ngữ nghĩa được cung cấp bởi hệ thống. Các tập tin có nhiều người đọc, viết hay bị giới hạn việc chia sẻ. Hệ thống tập tin phân tán cho phép máy khách hàng gắn các phân khu hay thư mục vào từ nhiều server. Với điều kiện chúng có thể truy xuất nhau qua mạng. Các hệ thống tập tin ở xa có những thách thức về khả năng tin cậy, năng lực và bảo mật. Hệ thống thông tin được phân tán duy trì người dùng, máy chủ, thông tin truy xuất như khách hàng và thông tin trạng thái chia sẻ servers để quản lý việc sử dụng và truy xuất.

Vì các tập tin là cơ chế lưu trữ thông tin quan trọng trong hầu hết các hệ thống máy tính nên bảo vệ tập tin là cần thiết. Truy xuất tới các tập tin được kiểm soát riêng cho mỗi loại truy xuất-đọc, viết, thực thi, chèn cuối, xóa, liệt kê thư mục,..Bảo vệ tập tin có thể được cung cấp bởi mật khẩu, bởi danh sách truy xuất hay bởi những kỹ thuật phức tạp.

Thuviendientu.org

CÀI ĐẶT HỆ THỐNG TẬP TIN

I Mục đích

Sau khi học xong chương này, người học nắm được những kiến thức sau:

- Hiểu việc lưu trữ các tập tin và truy xuất các tập tin trên các thiết bị lưu trữ phụ.
- Hiểu các phương pháp để thiết lập việc sử dụng tập tin
- Hiểu cách cấp phát không gian đĩa, phục hồi không gian trống, ghi vết vị trí dữ liệu

II Giới thiệu

Trong chương trước chúng ta thấy rằng, hệ thống tập tin cung cấp cơ chế cho việc lưu trữ trực tuyến (on-line storage) và truy xuất tới nội dung tập tin, gồm dữ liệu và chương trình. Hệ thống tập tin định vị vĩnh viễn trên thiết bị lưu trữ phụ. Các thiết bị này được thiết kế để quản lý lượng lớn thông tin không thay đổi.

Chương này tập trung chủ yếu với những vấn đề xoay quanh việc lưu trữ tập tin và truy xuất trên các thiết bị lưu trữ phụ. Chúng ta khám phá các cách để xây dựng cấu trúc sử dụng tập tin, cấp phát không gian đĩa và phục hồi không gian trống để ghi lại vị trí dữ liệu và để giao tiếp với các phần khác của hệ điều hành tới thiết bị lưu trữ phụ. Các vấn đề về năng lực được xem xét thông qua chương này.

III Cấu trúc hệ thống tập tin

Đĩa cung cấp số lượng thiết bị lưu trữ phụ mà trên đó hệ thống tập tin được duy trì. Có hai đặc điểm làm đĩa trở thành phương tiện tiện dụng cho việc lưu trữ nhiều tập tin:

- Chúng có thể được viết lại bằng cách thay thế; có thể đọc một khối từ đĩa, sửa một khối và viết nó ngược trở lại đĩa trong cùng vị trí.
- Chúng có thể được truy xuất trực tiếp bất cứ khối thông tin nào trên đĩa.

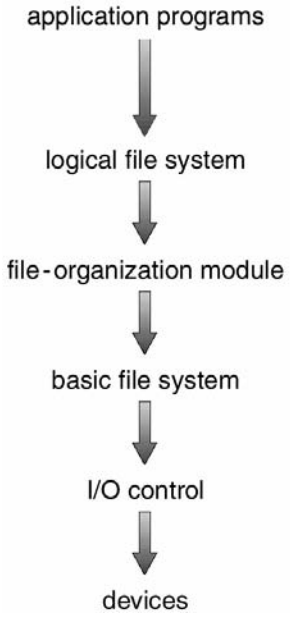
Để cải tiến tính hiệu quả nhập/xuất, thay vì chuyển một byte tại một thời điểm, nhập/xuất chuyển giữa bộ nhớ và đĩa được thực hiện trong đơn vị khối. Mỗi khối là một hay nhiều **cung từ** (sector). Phụ thuộc ổ đĩa, các cung từ biến đổi từ 32 bytes tới 4096 bytes; thường là 512 bytes.

Để cung cấp việc truy xuất hiệu quả và tiện dụng tới đĩa, hệ điều hành áp đặt một hay nhiều **hệ thống tập tin** để cho phép dữ liệu được lưu trữ, định vị và truy xuất lại dễ dàng. Một hệ thống tập tin đặt ra hai vấn đề thiết kế rất khác nhau. Vấn đề đầu tiên là định nghĩa hệ thống tập tin nên quan tâm đến người dùng như thế nào. Tác vụ này liên quan đến việc định nghĩa một tập tin và thuộc tính của nó, các thao tác được phép trên một tập tin và các giải thuật và cấu trúc cho việc tổ chức tập tin. Vấn đề thứ hai là tạo giải thuật và cấu trúc dữ liệu để ánh xạ hệ thống tập tin luận lý vào các thiết bị lưu trữ phụ.

Hệ thống tập tin thường được tạo thành từ nhiều cấp khác nhau. Cấu trúc được hiển thị trong hình X-1 là một thí dụ của thiết kế phân cấp. Mỗi cấp trong thiết kế

Thuviendientu.org

dùng các đặc điểm của cấp thấp hơn để tạo các đặc điểm mới cho việc sử dụng bởi cấp cao hơn.



Hình 0-1 hệ thống tập tin phân tầng

- **Điều khiển nhập/xuất** (I/O control): là cấp thấp nhất chứa các trình điều khiển thiết bị và các bộ quản lý ngắt để chuyển thông tin giữa bộ nhớ chính và hệ thống đĩa. Trình điều khiển thiết bị thường viết các mẫu bit xác định tới các vị trí trong bộ nhớ của bộ điều khiển nhập/xuất để báo với bộ điều khiển vị trí trên thiết bị nào và hoạt động gì xảy ra.
- **Hệ thống tập tin cơ bản** (basic file system) chỉ cần phát ra các lệnh thông thường tới các trình điều khiển thiết bị tương ứng để đọc và viết các khối vật lý trên đĩa. Mỗi khối vật lý được xác định bởi địa chỉ đĩa (thí dụ, đĩa 1, cyclinder 73, track 2, sector 10).
- **Module tổ chức tập tin** (file-organization module) biết các tập tin và các khối luận lý cũng như các khối vật lý. Bằng cách biết kiểu cấp phát tập tin được dùng và vị trí của tập tin, module tổ chức tập tin có thể dịch các địa chỉ khối luận lý thành các địa chỉ khối vật lý cho hệ thống tập tin cơ bản để truyền. Các khối luận lý của mỗi tập tin được đánh số từ 0 (hay 1) tới N, ngược lại các khối vật lý chứa dữ liệu thường không khớp với các số luận lý vì thế một thao tác dịch được yêu cầu để định vị mỗi khối. Module tổ chức tập tin cũng chứa **bộ quản lý không gian trống** (free-space manager), mà nó ghi vết các khối không được cấp phát và cung cấp các khối này tới module tổ chức tập tin khi được yêu cầu.
- **Hệ thống tập tin luận lý** (logical file system) quản lý thông tin siêu dữ liệu (metadata). Metadata chứa tất cả cấu trúc hệ thống tập tin, ngoại trừ dữ liệu thật sự (hay nội dung của các tập tin). Hệ thống tập tin luận lý quản lý cấu trúc thư mục để cung cấp module tổ chức tập tin những thông tin yêu cầu sau đó, được cho tên tập tin ký hiệu. Nó duy trì cấu trúc tập tin bằng khối điều khiển tập tin. Một khối điều khiển tập tin (file control block-FCB) chứa thông tin về tập tin, gồm người sở hữu, quyền và vị trí của nội dung tập tin.

Thuviendientu.org

Nhiều hệ thống tập tin được cài đặt hiện nay. Hầu hết hệ điều hành hỗ trợ nhiều hơn một hệ thống tập tin. Mỗi hệ điều hành có hệ thống tập tin dựa trên cơ sở đĩa. UNIX dùng hệ thống tập tin UNIX (UNIX file system-UFS) như là cơ sở. Windows NT hỗ trợ các định dạng tập tin FAT, FAT32 và NTFS cũng như CD-ROM, DVD và các định dạng hệ thống tập tin đĩa mềm. Bằng cách dùng cấu trúc phân cấp cho việc cài đặt hệ thống tập tin, nên nhân bản mã là tối thiểu. Điều khiển nhập/xuất và mã hệ thống tập tin cơ bản có thể được dùng bởi nhiều hệ thống tập tin. Mỗi hệ thống tập tin có hệ thống tập tin luận lý và module tổ chức tập tin của chính nó.

IV Cài đặt hệ thống tập tin

Trong phần này chúng ta sẽ nghiên cứu các cấu trúc và các thao tác được dùng để cài đặt các thao tác của hệ thống tập tin.

IV.1 Tổng quan

Nhiều cấu trúc trên đĩa và trên bộ nhớ được dùng để cài đặt một hệ thống tập tin. Các cấu trúc này thay đổi dựa trên hệ điều hành và hệ thống tập tin nhưng có một số nguyên tắc chung được áp dụng. Trên đĩa, hệ thống tập tin chứa thông tin về cách khởi động hệ điều hành được lưu trữ ở đó, tổng số khối, số và vị trí của các khối trống, cấu trúc thư mục, các tập tin riêng biệt.

Các cấu trúc trên đĩa gồm:

- **Khối điều khiển khởi động** (boot control block) có thể chứa thông tin được yêu cầu bởi hệ thống để khởi động một hệ điều hành từ phân khu đó. Nếu đĩa không chứa hệ điều hành thì khối này là rỗng. Điển hình, nó là khối đầu tiên của đĩa. Trong UFS, khối này được gọi là khối khởi động; trong NTFS, nó là cung khởi động phân khu (partition boot sector).
- **Khối điều khiển phân khu** (partition control block) chứa chi tiết về phân khu, như số lượng khối trong phân khu, kích thước khối, bộ đếm khối trống và con trỏ khối trống, bộ đếm FCB trống và con trỏ FCB. Trong UFS khối này được gọi là **siêu khối** (superblock); trong NTFS, nó là **bảng tập tin chính** (Master File Table)
- Một cấu trúc tập tin được dùng để tổ chức các tập tin
- Một FCB chứa nhiều chi tiết tập tin gồm các quyền tập tin, người sở hữu, kích thước, và vị trí của các khối dữ liệu. Trong UFS khối này được gọi là inode. Trong NTFS, thông tin này được lưu trong Master File Table dùng cấu trúc cơ sở dữ liệu quan hệ với một dòng cho một tập tin.

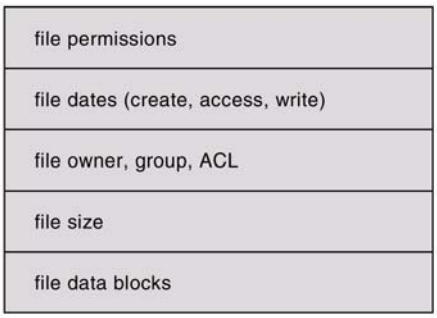
Thông tin trong bộ nhớ được dùng cho việc quản lý hệ thống tập tin và cải tiến năng lực qua lưu trữ (caching). Các cấu trúc này có thể bao gồm:

- Bảng phân khu trong bộ nhớ chứa thông tin về mỗi phân khu được gắn vào.
- Cấu trúc thư mục trong bộ nhớ quản lý thông tin thư mục của những thư mục vừa được truy xuất. (đối với các thư mục nơi mà các phân khu được gắn vào, nó có thể chứa một con trỏ chỉ tới bảng phân khu.)
- Bảng tập tin đang mở của hệ thống (system-wide open-file table) chứa bản sao của FCB của mỗi tập tin đang mở cũng như các thông tin khác.

Thuviendientu.org

- Bảng tập tin đang mở trên quá trình (per-process open-file table) chứa con trỏ chỉ tới mục từ tương ứng trong bảng tập tin đang mở của hệ thống cũng như những thông tin khác.

Để tạo một tập tin mới, một chương trình ứng dụng gọi hệ thống tập tin luận lý. Hệ thống tập tin luận lý biết định dạng của các cấu trúc thư mục. Để tạo một tập tin mới, nó cấp phát một FCB mới, đọc thư mục tương ứng vào bộ nhớ, cập nhật nó với tên tập tin mới và FCB, và viết nó trở lại đĩa. Một FCB điển hình được hiển thị trong hình X-2.

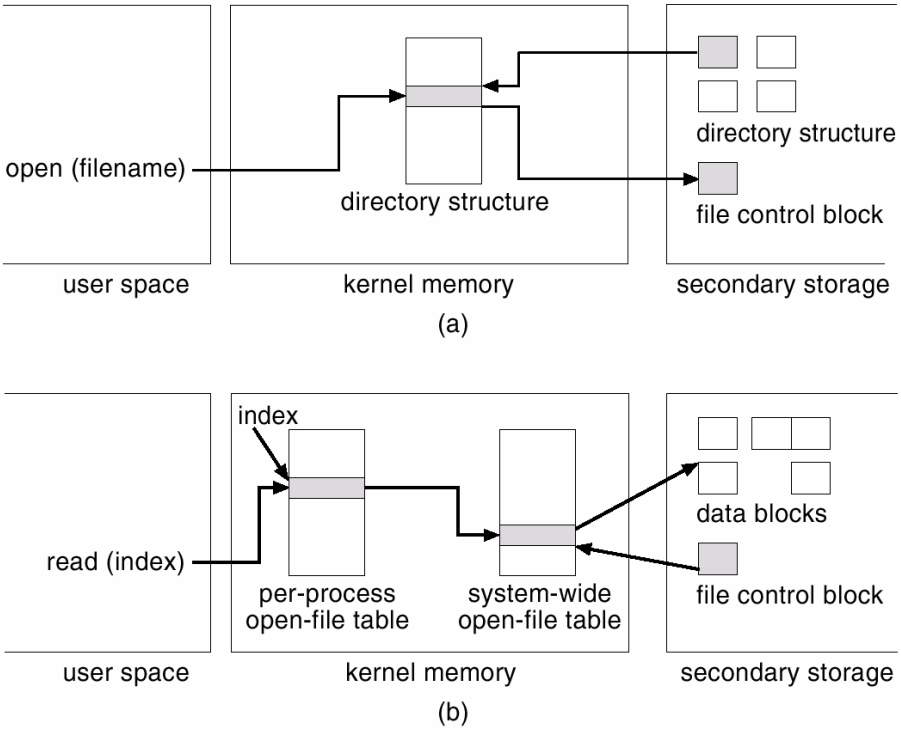


Hình 0-2 Một khối điều khiển tập tin điển hình

Một số hệ điều hành như UNIX xem một thư mục như là một tập tin-một tập tin với một trường kiểu hiển thị rằng nó là một thư mục. Các hệ điều hành khác như Windows NT cài đặt các lời gọi hệ thống riêng cho tập tin và thư mục và xem các thư mục như các thực thể tách rời từ các tập tin. Đối với cấu trúc lớn hơn, hệ thống tập tin luận lý có thể gọi module tổ chức tập tin để ánh xạ nhập/xuất thư mục vào số khối đĩa mà chúng được truyền trên cơ sở hệ thống tập tin và hệ thống điều khiển nhập/xuất. Module tổ chức tập tin cũng cấp phát các khối cho việc lưu trữ dữ liệu của tập tin.

Một tập tin được tạo, nó có thể được dùng cho nhập/xuất. Đầu tiên, nó phải được mở. Lời gọi *open* truyền tên tập tin tới hệ thống tập tin. Khi một tập tin được mở, cấu trúc thư mục thường được lưu vào bộ nhớ để tăng tốc độ các thao tác thư mục. Một khi tập tin được tìm thấy, FCB được chép vào bảng tập tin đang mở của hệ thống trong bộ nhớ. Bảng này không chỉ chứa FCB mà còn có các mục từ cho số đếm của số quá trình có mở tập tin.

Thuviendientu.org



Hình 0-3 Cấu trúc hệ thống trong bộ nhớ. (a) mở tập tin. (b) đọc tập tin.

Tiếp theo, một mục từ được tạo trong bảng tập tin đang mở trên quá trình, với một con trỏ chỉ tới mục từ trong bảng hệ thống tập tin đang mở của hệ thống và một số trường khác. Các trường khác này có thể chứa con trỏ chỉ tới vị trí hiện hành trong tập tin (cho các thao tác *read* hay *write* tiếp theo) và chế độ truy xuất trong tập tin được mở. Lời gọi *open* trả về một con trỏ chỉ tới mục từ tương ứng trong bảng hệ thống tập tin trên quá trình. Sau đó, tất cả thao tác tập tin được thực hiện bằng con trỏ này. Tên tập tin không phải là một phần của bảng tập tin đang mở, hệ thống không dùng nó một khi FCB tương ứng được định vị trên đĩa. Tên được cho đối với mục từ rất đa dạng. Các hệ thống UNIX chỉ tới nó như một bộ mô tả tập tin (file discriptor); Windows 2000 chỉ tới nó như một bộ quản lý tập tin (file handle). Do đó, với điều kiện là tập tin không đóng, tất cả các thao tác tập tin được thực hiện trên bảng tập tin đang mở.

Khi một quá trình đóng tập tin, mục từ trong bảng trên quá trình bị xoá và bộ đếm số lần mở của mục từ hệ thống giảm. Khi tất cả người dùng đóng tập tin, thông tin tập tin được cập nhật sẽ được chép trở lại tới cấu trúc thư mục dựa trên đĩa và mục từ bảng tập tin đang mở bị xóa.

Trong thực tế, lời gọi hệ thống *open* đầu tiên tìm bảng tập tin đang mở hệ thống để thấy nếu tập tin được sử dụng rồi bởi một quá trình khác. Nếu nó là mục từ bảng tập tin đang mở trên quá trình được tạo để chỉ tới bảng tập tin đang mở hệ thống đã có. Giải thuật này có thể tiết kiệm chi phí khi các tập tin đã mở rồi.
Các cấu trúc điều hành của việc cài đặt hệ thống tập tin được tóm tắt như hình X-3.

Thuviendientu.org

IV.2 Hệ thống tập tin ảo

Một phương pháp nổi bật cho việc cài đặt nhiều loại hệ thống tập tin là viết các chương trình con thư mục và tập tin cho mỗi loại. Đúng hơn là hầu hết các hệ điều hành, gồm UNIX, dùng các kỹ thuật hướng đối tượng để đơn giản hóa, tổ chức, và module hóa việc cài đặt. Sử dụng các phương pháp này cho phép nhiều loại hệ thống tập tin khác nhau được cài đặt trong cùng cấu trúc, gồm các hệ thống tập tin mạng như NFS. Người dùng có thể truy xuất các tập tin được chứa trong nhiều hệ thống tập tin trên đĩa cục bộ, hay ngay cả trên các hệ thống tập tin sẵn dùng qua mạng.

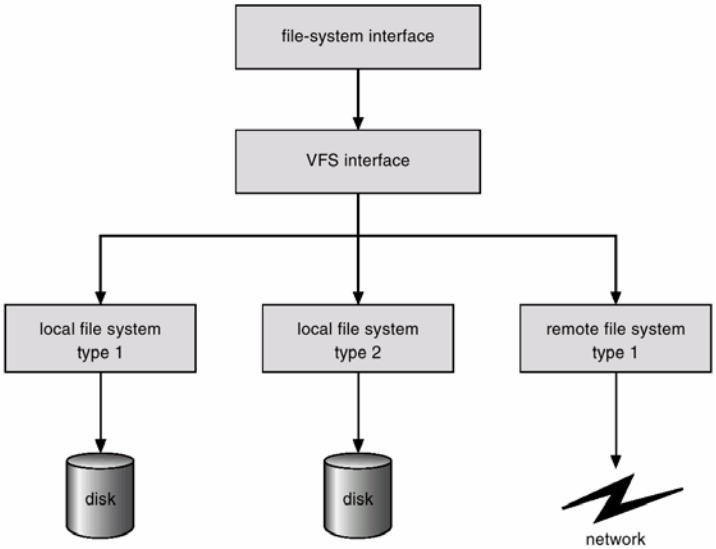
Các cấu trúc dữ liệu và thủ tục được dùng để cô lập chức năng lời gọi hệ thống cơ bản từ các chi tiết cài đặt. Do đó, cài đặt hệ thống tập tin chứa ba tầng chính; nó được mô tả dưới dạng lưu đồ trong hình X-4. Tầng đầu tiên là giao diện hệ thống tập tin dựa trên cơ sở lời gọi *open, read, write, close* và các bộ mô tả tập tin.

Tầng thứ hai được gọi là **hệ thống tập tin ảo** (Virtual File System-VFS); nó phục vụ hai chức năng quan trọng:

1) Nó tách biệt các thao tác hệ thống tập tin giống nhau từ việc cài đặt bằng cách định nghĩa một giao diện VFS rõ ràng. Nhiều cài đặt cho giao diện VFS có thể cùng tồn tại trên cùng một máy, cho phép truy xuất trong suốt tới các loại hệ thống tập tin khác nhau được gắn cục bộ.
2) VFS dựa trên cấu trúc biểu diễn tập tin, được gọi là **vnode**, chứa một bộ gắn bằng số (numerical designator) cho tập tin duy nhất qua mạng. (Inode của UNIX là duy nhất chỉ trong một hệ thống tập tin đơn). Tính duy nhất qua mạng được yêu cầu để hỗ trợ các hệ thống tập tin mạng. Nhân duy trì một cấu trúc vnode cho mỗi nút hoạt động (tập tin hay thư mục).

Do đó, VFS có sự khác biệt các tập tin cục bộ với các tập tin ở xa, và các tập tin cục bộ được phân biệt dựa theo loại hệ thống tập tin của chúng.

VFS kích hoạt các thao tác đặc tả hệ thống tập tin để quản lý các yêu cầu cục bộ dựa theo các loại hệ thống tập tin và ngay cả các lời gọi các thủ tục giao thức NFS cho các yêu cầu ở xa. Quản lý tập tin được xây dựng từ vnode tương ứng và được truyền như các tham số tới các thủ tục này. Tầng cài đặt kiểu hệ thống tập tin, hay giao thức hệ thống tập tin ở xa, là tầng dưới cùng của kiến trúc.



Hình 0-4 Hình ảnh dạng lưu đồ của hệ thống tập tin ảo

Thuviendientu.org

V Cài đặt thư mục

Chọn giải thuật cấp phát thư mục và quản lý thư mục có tác động lớn đến tính hiệu quả, năng lực, khả năng tin cậy của hệ thống tập tin. Do đó, chúng ta cần hiểu sự thoả hiệp liên quan trong các giải thuật này.

V.1 Danh sách tuyến tính

Phương pháp đơn giản nhất cho việc cài đặt thư mục là dùng một danh sách tuyến tính chứa tên tập tin với con trỏ chỉ tới các khối dữ liệu. một danh sách tuyến tính với các mục từ thư mục yêu cầu tìm kiếm tuyến tính để xác định một mục từ cụ thể. Phương pháp này đơn giản để lập trình nhưng mất nhiều thời gian để thực thi.

- Để tạo tập tin mới, trước tiên chúng ta phải tìm thư mục để đảm bảo rằng không có tập tin nào tồn tại với cùng một tên. Sau đó, chúng ta thêm một mục từ mới vào cuối thư mục.
- Để xoá một tập tin, chúng ta tìm kiếm thư mục cho tập tin được xác định bởi tên, sau đó giải phóng không gian được cấp phát tới nó.
- Để dùng lại mục từ thư mục, chúng ta có thể thực hiện một vài bước. Chúng ta có thể đánh dấu mục từ như không được dùng (bằng cách gán nó một tên đặc biệt, như một tên trống hay với một bit xác định trạng thái được dùng hoặc không được dùng trong mỗi mục từ), hay chúng ta có thể gán nó tới một danh sách của các mục từ thư mục trống. Một thay đổi thứ ba là chép mục từ cuối cùng trong thư mục vào vị trí trống và giảm chiều dài của thư mục. Một danh sách liên kết có thể được dùng để giảm thời gian xoá một tập tin.

Bất lợi thật sự của danh sách tuyến tính chứa các mục từ thư mục là tìm kiếm tuyến tính để tìm một tập tin. Thông tin thư mục được dùng thường xuyên và người dùng nhận thấy việc truy xuất tới tập tin là chậm. Để khắc phục nhược điểm này, nhiều hệ điều hành cài đặt một vùng lưu trữ phần mềm (software cache) để lưu hầu hết những thông tin thư mục được dùng gần nhất. Một chập dữ liệu được lưu trữ sẽ tránh đọc lại liên tục thông tin từ đĩa. Một danh sách được sắp xếp cho phép tìm kiếm nhị phân và giảm thời gian tìm kiếm trung bình. Tuy nhiên, yêu cầu mà một danh sách phải được sắp xếp có thể phức tạp việc tạo và xoá tập tin vì chúng ta phải di chuyển lượng thông tin liên tục để duy trì một thư mục được xếp thứ tự. Một cấu trúc dữ liệu cây tinh vi hơn như B-tree có thể giúp giải quyết vấn đề. Lợi điểm của danh sách được sắp xếp là liệt kê một thư mục có thứ tự mà không cần một bước sắp xếp riêng.

V.2 Bảng băm

Một cấu trúc dữ liệu khác thường được dùng cho một thư mục tập tin là bảng băm (hash table). Trong phương pháp này, một danh sách tuyến tính lưu trữ các mục từ thư mục nhưng một cấu trúc bảng băm cũng được dùng. Bảng băm lấy một giá trị được tính từ tên tập tin và trả về con trỏ chỉ tới tên tập tin trong danh sách tuyến tính. Do đó, nó có thể giảm rất lớn thời gian tìm kiếm thư mục. Chèn và xoá cũng tương đối đơn giản mặc dù có thể phát sinh đụng độ-những trường hợp có hai tên tập tin được băm cùng vị trí. Khó khăn chính với một bảng băm là kích thước của nó thường cố định và phụ thuộc vào hàm băm trên kích thước đó.

Thí dụ, giả sử rằng chúng ta thực hiện một bảng băm thăm dò tuyến tính quản lý 64 mục từ. Hàm băm chuyển các tập tin thành các số nguyên từ 0 tới 63, thí dụ bằng cách dùng số dư của phép chia cho 64. Sau đó, nếu chúng ta cố tạo tập tin thứ

Thuviendientu.org

65, chúng ta phải mở rộng bảng băm thư mục-tới 128 mục từ. Kết quả là chúng ta cần hàm băm mới phải ánh xạ tới dãy 0-127 và chúng ta phải sắp xếp lại các mục từ thư mục đã có để phản ánh giá trị hàm băm mới. Một cách khác, một bảng băm vòng có thể được dùng. Mỗi mục từ băm có thể là danh sách liên kết thay vì chỉ một giá trị riêng và chúng ta có thể giải quyết các đụng độ bằng cách thêm mục từ mới vào danh sách liên kết. Tìm kiếm có thể chậm vì tìm kiếm một tên có thể yêu cầu từ bước thông qua một danh sách liên kết của các mục từ bảng đụng độ; nhưng điều này vẫn nhanh hơn tìm kiếm tuyến tính qua toàn thư mục.

VI Các phương pháp cấp phát

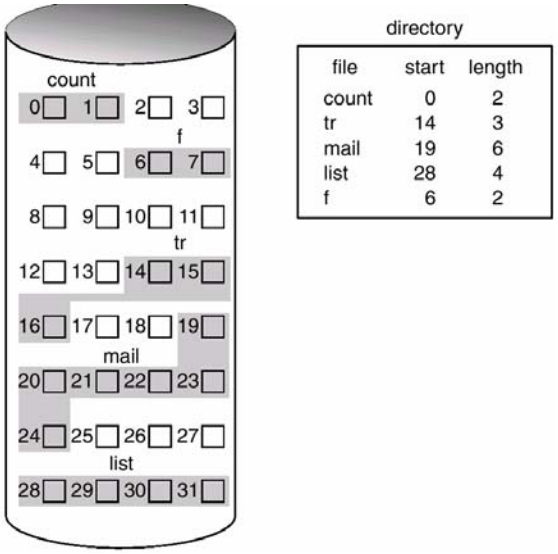
Tính tự nhiên của truy xuất trực tiếp đĩa cho phép chúng ta khả năng linh hoạt trong việc cài đặt tập tin. Trong hầu hết mọi trường hợp, nhiều tập tin sẽ được lưu trên cùng đĩa. Vấn đề chính là không gian cấp phát tới các tập tin này như thế nào để mà không gian đĩa được sử dụng hiệu quả và các tập tin có thể được truy xuất nhanh chóng. Ba phương pháp quan trọng cho việc cấp phát không gian đĩa được sử dụng rộng rãi: cấp phát kề, liên kết và chỉ mục. Mỗi phương pháp có ưu và nhược điểm. Một số hệ thống hỗ trợ cả ba. Thông dụng hơn, một hệ thống sẽ dùng một phương pháp cụ thể cho tất cả tập tin.

VI.1 Cấp phát kề

Phương pháp cấp phát kề yêu cầu mỗi tập tin chiếm một tập hợp các khối kề nhau trên đĩa. Các địa chỉ đĩa định nghĩa một thứ tự tuyến tính trên đĩa. Với thứ tự này, giả sử rằng chỉ một công việc đang truy xuất đĩa, truy xuất khối b+1 sau khi khối b không yêu cầu di chuyển trước. Khi di chuyển đầu đọc được yêu cầu (từ cung từ cuối cùng của cylinder tới cung từ đầu tiên của cylinder tiếp theo), nó chỉ di chuyển một rãnh (track). Do đó, số lượng tìm kiếm đĩa được yêu cầu cho truy xuất kề tới các tập tin được cấp phát là nhỏ nhất, như là thời gian tìm kiếm khi tìm kiếm cuối cùng được yêu cầu. Hệ điều hành IBM VM/CMS dùng cấp phát kề.

Cấp phát kề của một tập tin được định nghĩa bởi địa chỉ đĩa và chiều dài (tính bằng đơn vị khối) của khối đầu tiên. Nếu tập tin có n khối và bắt đầu tại khối b thì nó chiếm các khối b, b+1, b+2,..,b+n-1. Mục từ thư mục cho mỗi tập tin hiển thị địa chỉ của khối bắt đầu và chiều dài của vùng được cấp phát cho tập tin này (hình X-5).

Thuviendientu.org



Hình 0-5 Không gian đĩa được cấp phát kề

Truy xuất một tập tin được cấp phát kề rất dễ. Đối với truy xuất tuần tự, hệ thống tập tin nhớ địa chỉ đĩa của khối cuối cùng được tham chiếu và đọc khối tiếp theo. Để truy xuất khối i của tập tin mà bắt đầu tại khối b, chúng ta có thể lập tức truy xuất khối b+i. Do đó, cả truy xuất tuần tự và truy xuất trực tiếp có thể được hỗ trợ bởi cấp phát kề.

Tuy nhiên, cấp phát kề có một số vấn đề. Một khó khăn là tìm không gian cho một tập tin mới. Việc cài đặt của hệ thống quản lý không gian trống xác định tác vụ này được hoàn thành như thế nào. Bất cứ hệ thống quản lý nào có thể được dùng nhưng nhanh chậm khác nhau.

Vấn đề cấp phát không gian kề có thể được xem là vấn đề cấp phát lưu trữ động của ứng dụng để thoả mãn yêu cầu kích thước n từ danh sách các lỗ trống. First fit và best fit là những chiến lược chung nhất được dùng để chọn lỗ trống từ tập hợp các lỗ trống sẵn dùng. Những mô phỏng hiển thị rằng cả hai first fit và best fit hiệu quả hơn worst fit về thời gian và sử dụng không gian lưu trữ. First fit hay best fit đều không phải là giải thuật tốt nhất nhưng thường thì first fit nhanh hơn best fit.

Các giải thuật này gặp phải vấn đề phân mãnh ngoài. Khi các tập tin được cấp phát và xoá, không gian đĩa trống bị chia thành những mảnh nhỏ. Phân mãnh ngoài tồn tại bất cứ khi nào không gian trống được chia thành những đoạn. Nó trở thành một vấn đề khi đoạn kề lớn nhất không đủ cho một yêu cầu; lưu trữ được phân thành nhiều lỗ, không lỗ nào đủ lớn để lưu dữ liệu. Phụ thuộc vào tổng lượng lưu trữ đĩa và kích thước tập tin trung bình, phân mãnh ngoài có thể là vấn đề chính hay phụ.

Một số hệ thống vi tính cũ dùng cấp phát kề trên đĩa mềm. Để ngăn ngừa lượng lớn không gian đĩa phân mãnh ngoài, người dùng phải chạy thủ tục ***đóng gói lại***. Thủ tục này chép toàn bộ hệ thống tập tin tới một đĩa khác hay trên băng từ. Kế đến, đĩa mềm ban đầu được giải phóng hoàn toàn, tạo một không gian trống kề lớn. Sau đó, thủ tục này chép các tập tin trở lại trên đĩa mềm bằng cách cấp phát không gian kề từ một lỗ lớn. Cơ chế này hiệu quả trong việc hợp nhất (compacts) tất cả không gian trống thành một không gian trống kề, giải quyết vấn đề phân mãnh. Chi phí cho việc hợp nhất là thời gian. Chi phí thời gian phục vụ cho các đĩa cứng lớn dùng cấp phát kề, ở đây hợp nhất tất cả không gian mất hàng giờ. Trong thời gian này, các thao tác hệ thống thông thường không thể được phép vì thế hợp nhất tránh được tất cả chi phí do có thay đổi về dữ liệu.

Thuviendientu.org

Một vấn đề khác với cấp phát kề là xác định bao nhiêu không gian được yêu cầu cho một tập tin. Khi một tập tin được tạo, toàn bộ không gian nó cần phải được tìm kiếm và được cấp phát. Người tạo (chương trình hay người) biết kích thước tập tin được tạo như thế nào? Trong một số trường hợp việc xác định này tương đối đơn giản (thí dụ chép một tập tin đã có); tuy nhiên, kích thước của tập tin xuất có thể khó để ước lượng.

Nếu chúng ta cấp quá ít không gian tới một tập tin, chúng ta thấy rằng tập tin không thể mở rộng. Đặc biệt với một chiến lược cấp phát best fit, không gian trên cả hai phía của tập tin đang được dùng. Do đó, chúng ta không thể làm cho tập tin lớn hơn. Hai khả năng có thể. Thứ nhất, chương trình người dùng có thể được kết thúc với một thông báo lỗi hợp lý. Sau đó, người dùng phải cấp phát nhiều không gian hơn và chạy chương trình lại. Việc lặp này có thể gây ra chi phí. Để ngăn chặn chúng, người dùng sẽ mô phỏng nhiều hơn lượng không gian được yêu cầu. Điều này dẫn đến không gian bị lãng phí.

Một khả năng khác là tìm một lỗ trống lớn hơn, chép nội dung của tập tin tới không gian trống mới, và giải phóng không gian trước đó. Một loạt các hoạt động này có thể được lặp lại với điều kiện là không gian tồn tại mặc dù nó tiêu tốn nhiều thời gian. Tuy nhiên, trong trường hợp này người dùng không bao giờ yêu cầu được thông báo về những gì đang xảy ra; hệ thống tiếp tục mặc dù vấn đề phát sinh.
Ngay cả nếu toàn bộ không gian được yêu cầu cho tập tin được biết trước, cấp phát trước là không đủ. Một tập tin sẽ lớn lên trong khoảng thời gian dài phải được cấp phát đủ không gian cho kích thước cuối cùng của nó mặc dù không gian đó có thể không được dùng cho khoảng thời gian dài. Do đó, tập tin có lượng lớn phân mãnh trong.

Để tối thiểu các khó khăn này, một số hệ điều hành dùng một cơ chế cấp phát kề được hiệu chỉnh. Trong cơ chế này đoạn không gian kề được cấp phát trước và sau đó khi lượng không gian đó không đủ lớn, một đoạn không gian kề khác, một ***đoạn mở rộng*** (extent), được thêm vào cấp phát ban đầu. Sau đó, vị trí của các khối tập tin được ghi lại như một vị trí và một bộ đếm khối cộng với một liên kết tới khối đầu tiên của đoạn mở rộng tiếp theo. Trên một số hệ thống, người sở hữu tập tin có thể đặt kích thước đoạn mở rộng, nhưng việc đặt này có thể không hiệu quả nếu người sở hữu không đúng. Phân mãnh trong vẫn còn là vấn đề nếu đoạn mở rộng quá lớn và phân mãnh ngoài có thể là vấn đề khi các đoạn mở rộng có kích thước khác nhau được cấp phát và thu hồi.

VI.2 Cấp phát liên kết

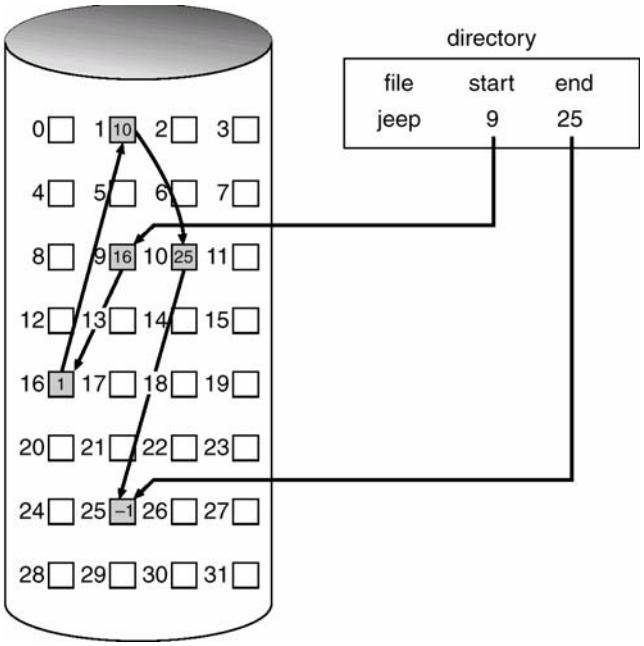
Cấp phát liên kết giải quyết vấn đề của cấp phát kề. Với cấp phát liên kết, mỗi tập tin là một danh sách các khối đĩa được liên kết; các khối đĩa có thể được phân tán khắp nơi trên đĩa. Thư mục chứa một con trỏ chỉ tới khối đầu tiên và các khối cuối cùng của tập tin. Thí dụ, một tập tin có 5 khối có thể bắt đầu tại khối số 9, tiếp tục là khối 16, sau đó khối 1, khối 10 và cuối cùng khối 25 (như hình X-6). Mỗi khối chứa một con trỏ chỉ tới khối kế tiếp. Các con trỏ này không được làm sẵn dùng cho người dùng. Do đó, nếu mỗi khối là 512 bytes, và địa chỉ đĩa (con trỏ) yêu cầu 4 bytes thì phần chứa dữ liệu của khối là 508 bytes.

Để tạo một tập tin mới, chúng ta đơn giản tạo một mục từ mới trong thư mục. Với cấp phát liên kết, mỗi mục từ thư mục có một con trỏ chỉ tới khối đĩa đầu tiên của tập tin. Con trỏ này được khởi tạo tới *nil* (giá trị con trỏ cuối danh sách) để chỉ một tập tin rỗng. Trường kích thước cũng được đặt tới 0. Một thao tác viết tới tập tin làm một khối trống được tìm thấy bằng hệ thống quản lý không gian trống, sau đó khối

Thuviendientu.org

mới này được viết tới và được liên kết tới cuối tập tin. Để đọc một tập tin, chúng ta đơn giản đọc các khối bằng cách lần theo các con trỏ từ khối này tới khối khác.

Không có sự phân mãnh ngoài với cấp phát liên kết, và bất cứ khối trống trên danh sách không gian trống có thể được dùng để thoả mãn yêu cầu. Kích thước của một tập tin không cần được khai báo khi tập tin đó được tạo. Một tập tin có thể tiếp tục lớn lên với điều kiện là các khối trống sẵn có. Do đó, nó không bao giờ cần thiết để hợp nhất không gian trống.



Hình 0-6 cấp phát không gian đĩa liên kết

Tuy nhiên, cấp phát liên kết có một vài nhược điểm. Vấn đề chủ yếu là nó có thể được dùng hiệu quả chỉ cho các tập tin truy xuất tuần tự. Để tìm khối thứ i của tập tin, chúng ta phải bắt đầu tại điểm bắt đầu của tập tin đó, và lần theo con trỏ cho đến khi chúng ta nhận được khối thứ i. Mỗi truy xuất tới con trỏ yêu cầu một thao tác đọc đĩa, và đôi khi là một tìm kiếm đĩa. Do đó, nó không đủ hỗ trợ một khả năng truy xuất trực tiếp cho các tập tin cấp phát liên kết.

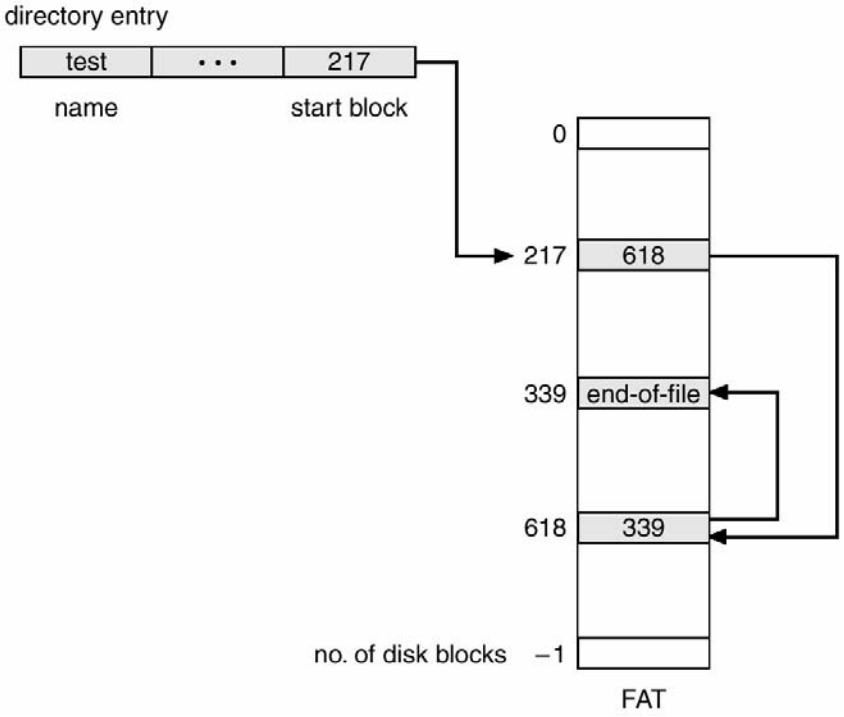
Một nhược điểm khác của cấp phát liên kết là không gian được yêu cầu cho các con trỏ. Nếu một con trỏ yêu cầu 4 bytes của khối 512 bytes thì 0.77% của đĩa được dùng cho các con trỏ thay vì là thông tin.

Một giải pháp thông thường để giải quyết vấn đề này là tập hợp các khối vào các **nhóm** (clusters) và cấp phát các nhóm hơn là các khối. Thí dụ, hệ thống tập tin có thể định nghĩa nhóm gồm 4 khối và thao tác trên đĩa chỉ trong đơn vị nhóm thì các con trỏ dùng % nhỏ hơn của không gian của tập tin. Phương pháp này cho phép ánh xạ khối luận lý tới vật lý vẫn còn đơn giản, nhưng cải tiến thông lượng đĩa và giảm không gian được yêu cầu cho cấp phát khối và quản lý danh sách trống. Chi phí của tiếp cận này là tăng phân mãnh trong vì nhiều không gian hơn bị lãng phí nếu một nhóm chỉ đầy một phần hơn là một khối đầy một phần. Các nhóm có thể được dùng để cải tiến thời gian truy xuất đĩa cho nhiều giải thuật khác nhau vì thế chúng được dùng trong hầu hết các hệ điều hành.

Thuviendientu.org

Một vấn đề khác của cấp phát liên kết là khả năng tin cậy. Vì các tập tin được liên kết với nhau bởi các con trỏ được phân tán khắp đĩa, xem xét điều gì xảy ra nếu một con trỏ bị mất hay bị phá hỏng. Một con bọ (bug) trong phần mềm hệ điều hành hay lỗi phần cứng đĩa có thể dẫn tới việc chọn con trỏ sai. Lỗi này có thể dẫn tới việc liên kết vào danh sách không gian trống hay vào một tập tin khác. Các giải pháp một phần là dùng các danh sách liên kết đôi hay lưu tên tập tin và số khối tương đối trong mỗi khối; tuy nhiên, các cơ chế này yêu cầu nhiều chi phí hơn cho mỗi tập tin.

Một thay đổi quan trọng trên phương pháp cấp phát liên kết là dùng **bảng cấp phát tập tin** (file allocation table-FAT). Điều này đơn giản nhưng là phương pháp cấp phát không gian đĩa hiệu quả được dùng bởi hệ điều hành MS-DOS và OS/2. Một phần đĩa tại phần bắt đầu của mỗi phân khu được thiết lập để chứa bảng này. Bảng này có một mục từ cho mỗi khối đĩa và được lập chỉ mục bởi khối đĩa. FAT được dùng nhiều như là một danh sách liên kết. Mục từ thư mục chứa số khối của khối đầu tiên trong tập tin. Mục từ bảng được lập chỉ mục bởi số khối đó sau đó chứa số khối của khối tiếp theo trong tập tin. Chuỗi này tiếp tục cho đến khi khối cuối cùng, có giá trị cuối tập tin đặc biệt như mục từ bảng. Các khối không được dùng được hiển thị bởi giá trị bảng 0. Cấp phát một khối mới tới một tập tin là một vấn đề đơn giản cho việc tìm mục từ bảng có giá trị 0 đầu tiên và thay thế giá trị kết thúc tập tin trước đó với địa chỉ của khối mới. Sau đó, số 0 được thay thế với giá trị kết thúc tập tin. Một thí dụ minh hoạ là cấu trúc FAT của hình X-7 cho một tập tin chứa các khối đĩa 217, 618 và 339.



Hình 0-7 Bảng cấp phát tập tin

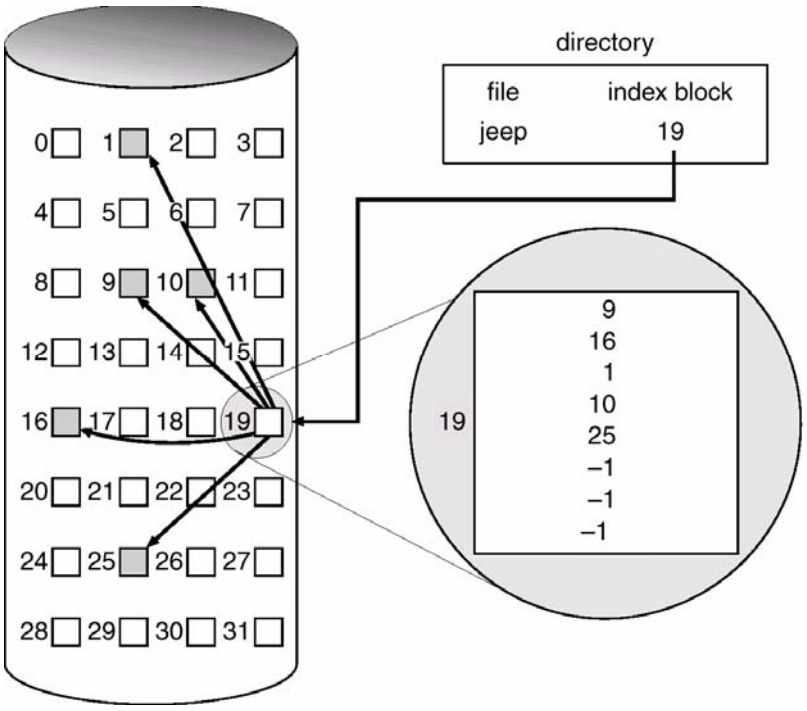
Cơ chế cấp phát FAT có thể dẫn tới số lượng lớn tìm kiếm đầu đọc đĩa nếu FAT không được lưu trữ(cache). Đầu đọc đĩa phải di chuyển tới điểm bắt đầu của phân khu để đọc FAT và tìm vị trí khối sau đó di chuyển tới vị trí của chính khối đĩa đó. Trong trường hợp xấu nhất, cả hai di chuyển xảy ra cho mỗi khối đĩa. Lợi điểm là thời gian truy xuất ngẫu nhiên được cải tiến vì đầu đọc đĩa có thể tìm vị trí của bất cứ khối nào bằng cách đọc thông tin trong FAT.

Thuviendientu.org

VI.3 Cấp phát được lập chỉ mục

Cấp phát liên kết giải quyết việc phân mãnh ngoài và vấn đề khai báo kích thước của cấp phát kề. Tuy nhiên, cấp phát liên kết không hỗ trợ truy xuất trực tiếp hiệu quả vì các con trỏ chỉ tới các khối được phân tán với chính các khối đó qua đĩa và cần được lấy lại trong thứ tự. Cấp phát được lập chỉ mục giải quyết vấn đề này bằng cách mang tất cả con trỏ vào một vị trí: **khối chỉ mục** (index block).

Mỗi tập tin có khối chỉ mục của chính nó, khối này là một mảng các địa chỉ khối đĩa. Mục từ thứ i trong khối chỉ mục chỉ tới khối i của tập tin. Thư mục chứa địa chỉ của khối chỉ mục (như hình X-8). Để đọc khối i, chúng ta dùng con trỏ trong mục từ khối chỉ mục để tìm và đọc khối mong muốn. Cơ chế này tương tự như cơ chế phân trang.



Hình 0-8 Cấp phát không gian đĩa được lập chỉ mục

Khi một tập tin được tạo, tất cả con trỏ trong khối chỉ mục được đặt tới nil. Khi khối thứ i được viết đầu tiên, khối được chứa từ bộ quản lý không gian trống và địa chỉ của nó được đặt trong mục từ khối chỉ mục.

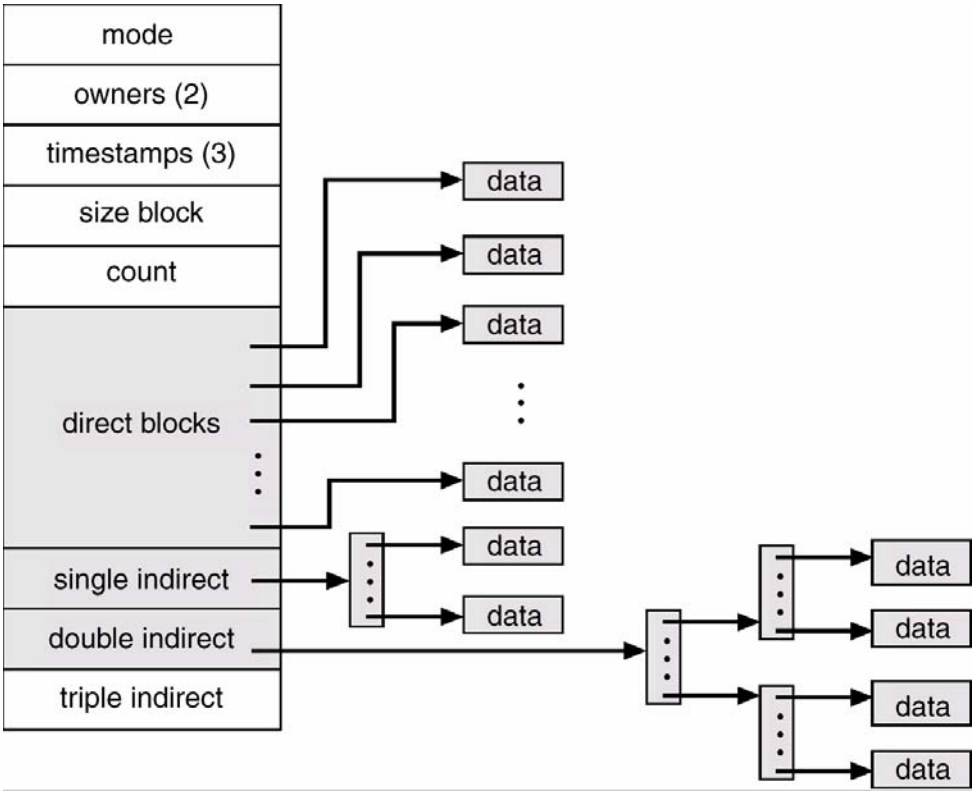
Cấp phát được lập chỉ mục hỗ trợ truy xuất trực tiếp, không gặp phải sự phân mãnh ngoài vì bất cứ khối trống trên đĩa có thể đáp ứng yêu cầu thêm không gian. Cấp phát được lập chỉ mục gặp phải sự lãng phí không gian. Chi phí con trỏ của khối chỉ mục thường lớn hơn chi phí con trỏ của cấp phát liên kết. Xét trường hợp thông thường trong đó chúng ta có một tập tin với chỉ một hoặc hai khối. Với cấp phát liên kết, chúng ta mất không gian của chỉ một con trỏ trên khối (một hay hai con trỏ). Với cấp phát được lập chỉ mục, toàn bộ khối chỉ mục phải được cấp phát thậm chí nếu một hay hai con trỏ là khác nil.

Điểm này sinh ra câu hỏi khối chỉ mục nên lớn bao nhiêu? Mỗi tập tin phải có một khối chỉ mục để mà chúng ta muốn khối chỉ mục nhỏ nhất có thể. Tuy nhiên, nếu khối chỉ mục quá nhỏ nó không thể quản lý đủ các con trỏ cho một tập tin lớn và một cơ chế sẽ phải sẵn có để giải quyết vấn đề này:

Thuviendientu.org

- **Cơ chế liên kết** (linked scheme): một khối chỉ mục thường là một khối đĩa. Do đó, nó có thể được đọc và viết trực tiếp bởi chính nó. Để cho phép đối với các tập tin lớn, chúng ta có thể liên kết nhiều khối chỉ mục với nhau. Thí dụ, một khối chỉ mục có thể chứa một header nhỏ cho tên tập tin và một tập hợp của các địa chỉ 100 khối đĩa đầu tiên. Địa chỉ tiếp theo (từ cuối cùng trong khối chỉ mục) là nil (đối với một tập tin nhỏ) hay một con trỏ tới khối chỉ mục khác (cho một tập tin lớn)
- **Chỉ mục nhiều cấp** (multilevel index): một biến dạng của biểu diễn liên kết là dùng khối chỉ mục cấp 1 để chỉ tới khối chỉ mục cấp 2. Khối cấp 2 chỉ tới các khối tập tin. Để truy xuất một khối, hệ điều hành dùng chỉ mục cấp 1 để tìm một khối chỉ mục cấp 2 và khối đó tìm khối dữ liệu mong muốn. Tiếp cận này có thể được tiếp tục tới cấp 3 hay cấp 4, tuỳ thuộc vào kích thước tập tin lớn nhất được mong muốn. Với khối có kích thước 4,096 bytes, chúng ta có thể lưu 1,024 con trỏ 4 bytes trong một khối chỉ mục. Chỉ mục hai cấp cho phép 1,048,576 khối dữ liệu, cho phép tập tin có kích thước tới 4GB.
- **Cơ chế kết hợp** (combined scheme): một biến dạng khác được dùng trong UFS là giữ 15 con trỏ đầu tiên của khối chỉ mục trong inode của tập tin. 12 con trỏ đầu tiên của 15 con trỏ này chỉ tới khối trực tiếp (direct blocks); nghĩa là chúng chứa các địa chỉ của khối mà chứa dữ liệu của tập tin. Do đó, dữ liệu đối với các tập tin nhỏ (không lớn hơn 12 khối) không cần một khối chỉ mục riêng. Nếu kích thước khối là 4 KB, thì tới 48 KB dữ liệu có thể truy xuất trực tiếp. 3 con trỏ tiếp theo chỉ tới các khối gián tiếp (indirect blocks). Con trỏ khối gián tiếp thứ nhất là địa chỉ của khối gián tiếp đơn (single indirect blocks). Khối gián tiếp đơn là một khối chỉ mục không chứa dữ liệu nhưng chứa địa chỉ của các khối chứa dữ liệu. Sau đó, có con trỏ khối gián tiếp đôi (double indirect block) chứa địa chỉ của một khối mà khối này chứa địa chỉ của các khối chứa con trỏ chỉ tới khối dữ liệu thật sự. Con trỏ cuối cùng chứa chứa địa chỉ của khối gián tiếp ba (triple indirect block). Với phương pháp này, số khối có thể được cấp phát tới một tập tin vượt quá lượng không gian có thể đánh địa chỉ bởi các con trỏ tập tin 4 bytes hay 4 GB. Nhiều cài đặt UNIX gồm Solaris và AIX của IBM hỗ trợ tới 64 bit con trỏ tập tin. Các con trỏ có kích thước này cho phép các tập tin và hệ thống tập tin có kích thước tới terabytes. Một inode được hiển thị trong hình X-9:

Thuviendientu.org



Hình 0-9 Inode của UNIX

Cơ chế cấp phát lập chỉ mục gặp một số khó khăn về năng lực như cấp phát liên kết. Đặc biệt, các khối chỉ mục có thể được lưu trữ (cache) trong bộ nhớ; nhưng các khối dữ liệu có thể được trãi rộng khắp phân khu.

VI.4 Năng lực

Các phương pháp cấp phát ở trên khác nhau về tính hiệu quả lưu trữ và thời gian truy xuất khối dữ liệu. Cả hai yếu tố này là tiêu chuẩn quan trọng trong việc chọn phương pháp hợp lý hay các phương pháp cho một hệ điều hành cài đặt.
Trước khi chọn một phương pháp, chúng ta cần xác định hệ thống sẽ được dùng như thế nào. Một hệ thống với hầu hết truy xuất tuần tự nên dùng một phương pháp khác từ hệ thống với hầu hết truy xuất ngẫu nhiên. Đối với bất cứ loại truy xuất nào, cấp phát kề yêu cầu chỉ một truy xuất để đạt được một khối đĩa. Vì chúng ta có thể giữ dễ dàng địa chỉ khởi đầu của tập tin trong bộ nhớ, chúng ta có thể tính lập tức địa chỉ đĩa của khối thứ i (hay khối kế tiếp) và đọc nó trực tiếp.

Đối với cấp phát liên kết, chúng ta cũng có thể giữ địa chỉ khối kế tiếp trong bộ nhớ và đọc nó trực tiếp. Phương pháp này là tốt cho truy xuất tuần tự; tuy nhiên, đối với truy xuất trực tiếp một truy xuất tới khối thứ i phải yêu cầu đọc I đĩa. Vấn đề này minh hoạ lý do cấp phát liên kết không được dùng cho một ứng dụng yêu cầu truy xuất trực tiếp.

Do đó, một số hệ thống hỗ trợ các tập tin truy xuất trực tiếp bằng cách dùng cấp phát kề và truy xuất tuần tự bởi cấp phát liên kết. Đối với các hệ thống này, loại truy xuất được thực hiện phải được khai báo khi tập tin được tạo. Một tập tin được tạo cho truy xuất tuần tự sẽ được liên kết và không thể được dùng cho truy xuất trực tiếp. Một tập tin được tạo cho truy xuất trực tiếp sẽ kề nhau và có thể hỗ trợ cả hai truy xuất trực tiếp và truy xuất tuần tự nhưng chiều dài tối đa của nó phải được khai báo

Thuviendientu.org

khi nó được tạo. Trong trường hợp này, hệ điều hành phải có cấu trúc dữ liệu hợp lý và các giải thuật để hỗ trợ cả hai phương pháp cấp phát. Các tập tin có thể được chuyển từ một kiểu này sang một kiểu khác bằng cách tạo một tập tin mới của loại mong muốn và các nội dung của tập tin cũ được chép vào tập tin mới. Sau đó, tập tin cũ có thể bị xoá và tập tin mới được đổi tên.

Cấp phát dạng chỉ mục phức tạp hơn. Nếu khối chỉ mục đã ở trong bộ nhớ rồi thì truy xuất có thể được thực hiện trực tiếp. Tuy nhiên, giữ khối chỉ mục trong bộ nhớ yêu cầu không gian có thể xem xét. Nếu không gian bộ nhớ này không sẵn dùng thì chúng ta phải đọc trước khối chỉ mục và sau đó khối dữ liệu mong muốn. Đối với chỉ mục hai cấp, đọc hai khối chỉ mục là cần thiết. Đối với tập tin rất lớn, truy xuất một khối gần cuối tập tin yêu cầu đọc tất cả khối chỉ mục để lần theo chuỗi con trỏ trước khi khối dữ liệu được yêu cầu cuối cùng được đọc. Do đó, năng lực của cấp phát chỉ mục phụ thuộc cấu trúc chỉ mục trên kích thước tập tin và vị trí của khối mong muốn.

Một số hệ thống kết hợp cấp phát kề và cấp phát chỉ mục bằng cách dùng cấp phát kề cho các tập tin nhỏ (ba hay bốn khối) và tự động chuyển tới cấp phát chỉ mục nếu tập tin lớn lên. Vì hầu hết các tập tin là nhỏ và cấp phát kề là hiệu quả cho các tập tin nhỏ, năng lực trung bình là rất tốt.

Nhiều tối ưu khác là có thể và đang được dùng. Với sự chênh lệch tốc độ giữa CPU và đĩa, nó là không hợp lý để thêm hàng ngàn chỉ thị tới hệ điều hành để tiết kiệm chỉ một vài di chuyển của đầu đọc. Ngoài ra, sự chênh lệch này tăng theo thời gian, tới điểm nơi mà hàng trăm của hàng ngàn chỉ thị phù hợp có thể được dùng để tối ưu sự di chuyển của đầu đọc.

VII Quản lý không gian trống

Vì không gian trống là giới hạn nên chúng ta cần dùng lại không gian từ các tập tin bị xoá cho các tập tin mới nếu có thể. Để giữ vết của không gian đĩa trống, hệ thống duy trì một danh sách không gian trống. Danh sách không gian trống ghi lại tất cả khối đĩa trống. Để tạo tập tin, chúng ta tìm trong danh sách không gian trống lượng không gian được yêu cầu và cấp phát không gian đó tới tập tin mới. Sau đó, không gian này được xoá từ danh sách không gian trống. Khi một tập tin bị xoá, không gian đĩa của nó được thêm vào danh sách không gian trống. Mặc dù tên của nó là danh sách nhưng danh sách không gian trống có thể không được cài như một danh sách.

VII.1 Bit vector

Thường thì danh sách không gian trống được cài đặt như một bản đồ bit (bit map) hay một vector bit (bit vector). Mỗi khối được biểu diễn bởi 1 bit. Nếu khối là trống, bit của nó được đặt là 1, nếu khối được cấp phát bit của nó được đặt là 0.
Thí dụ, xét một đĩa khi các khối 2, 3, 4, 5, 8, 9, 10, 11, 12, 13, 17, 18, 25, 26, và 27 là trống và các khối còn lại được cấp phát. Bản đồ bit không gian trống sẽ là:
00111100111111000110000001110000 0…

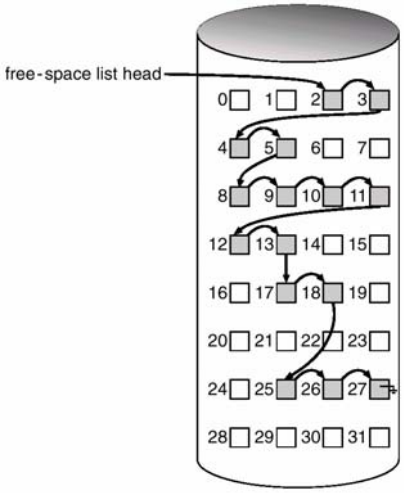
Lợi điểm chính của tiếp cận này là tính tương đối đơn giản và hiệu quả của nó trong việc tìm khối trống đầu tiên, hay n khối trống tiếp theo trên đĩa.

Một lần nữa, chúng ta thấy các đặc điểm phần cứng định hướng chức năng phần mềm. Tuy nhiên, các vector bit là không đủ trừ khi toàn bộ vector được giữ trong bộ nhớ chính. Giữ nó trong bộ nhớ chính là có thể cho các đĩa nhỏ hơn, như trên các máy vi tính nhưng không thể cho các máy lớn hơn. Một đĩa 1.3 GB với khối 512

Thuviendientu.org

bytes sẽ cần một bản đồ bit 332 KB để ghi lại các khối trống. Gom bốn khối vào một nhóm có thể giảm số này xuống còn 83 KB trên đĩa.

VII.2 Danh sách liên kết



Hình 0-10 danh sách không gian trống được liên kết trên đĩa

Một tiếp cận khác để quản lý bộ nhớ trống là liên kết tất cả khối trống, giữ một con trỏ tới khối trống đầu tiên trong một vị trí đặc biệt trên đĩa và lưu nó trong bộ nhớ. Khối đầu tiên này chứa con trỏ chỉ tới khối đĩa trống tiếp theo,..Trong thí dụ trên, chúng ta có thể giữ một con trỏ chỉ tới khối 2 như là khối trống đầu tiên. Khối 2 sẽ chứa một con trỏ chỉ tới khối 3, khối này sẽ chỉ tới khối 4,…(như hình X-10). Tuy nhiên, cơ chế này không hiệu quả để duyệt danh sách, chúng ta phải đọc mỗi khối, yêu cầu thời gian nhập/xuất đáng kể. Tuy nhiên, duyệt danh sách trống không là hoạt động thường xuyên. Thường thì, hệ điều hành cần một khối trống để mà nó có thể cấp phát khối đó tới một tập tin, vì thế khối đầu tiên trong danh sách trống được dùng. Phương pháp FAT kết hợp với đếm khối trống thành cấu trúc dữ liệu cấp phát.

VII.3 Nhóm

Thay đổi tiếp cận danh sách trống để lưu địa chỉ của n khối trống trong khối trống đầu tiên. n-1 khối đầu tiên này thật sự là khối trống. Khối cuối cùng chứa địa chỉ của n khối trống khác, …Sự quan trọng của việc cài đặt này là địa chỉ của một số lượng lớn khối trống có thể được tìm thấy nhanh chóng, không giống như trong tiếp cận danh sách liên kết chuẩn.

VII.4 Bộ đếm

Một tiếp cận khác đạt được lợi điểm trong thực tế là nhiều khối kề có thể được cấp phát và giải phóng cùng lúc, đặc biệt khi không gian được cấp phát với giải thuật cấp phát kề hay thông qua nhóm. Do đó, thay vì giữ một danh sách n địa chỉ đĩa trống, chúng ta có thể giữ địa chỉ của khối trống đầu tiên và số n khối kề trống theo sau khối đầu tiên. Mỗi mục từ trong danh sách không gian trống sau đó chứa một địa chỉ đĩa và bộ đếm. Mặc dù mỗi mục từ yêu cầu nhiều không gian hơn một địa chỉ đĩa đơn, nhưng toàn bộ danh sách sẽ ngắn hơn với điều kiện là bộ đếm lớn hơn 1.

Thuviendientu.org

VIII Tóm tắt

Hệ thống tập tin định vị không đổi trên thiết bị lưu trữ phụ được thiết kế để quản lý một lượng lớn dữ liệu không đổi. Phương tiện lưu trữ phụ phổ biến nhất là đĩa.

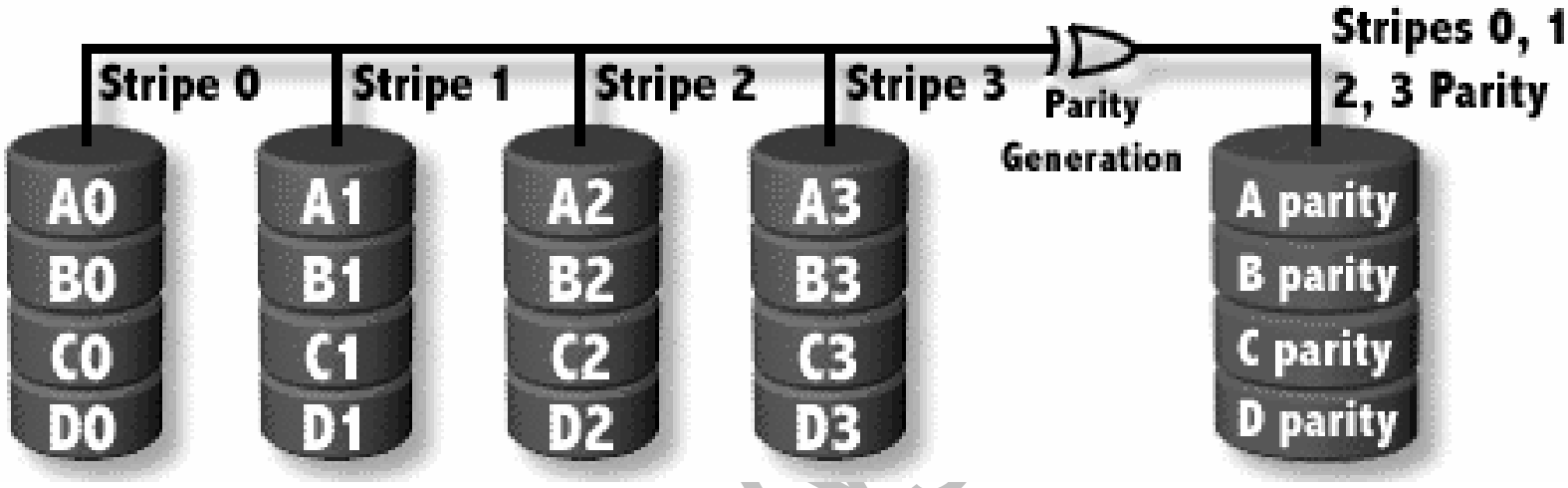
Đĩa vật lý có thể được chia thành nhiều phân khu để điều khiển việc sử dụng phương tiện và cho phép nhiều hệ thống tập tin (có thể khác nhau) trên đĩa. Các hệ thống tập tin này được gắn vào kiến trúc hệ thống tập tin luận lý để làm cho chúng sẵn dùng. Các hệ thống tập tin thường được cài đặt trong một kiến trúc phân tầng hay module. Những cấp thấp hơn giải quyết các thuộc tính vật lý của các thiết bị lưu trữ. Cấp cao hơn giải quyết các tên tập tin biểu tượng và các thuộc tính luận lý của tập tin. Các cấp trung gian ánh xạ các khái niệm tập tin luận lý thành các thuộc tính thiết bị vật lý.

Mỗi kiểu hệ thống tập tin có các cấu trúc và giải thuật khác nhau. Một tầng VFS cho phép các tầng cao hơn giải quyết mỗi kiểu hệ thống tập tin khác nhau trong cùng một cách. Ngay cả các hệ thống tập tin ở xa có thể được tích hợp vào cấu trúc thư mục của hệ thống và được hoạt động trên các lời gọi hệ thống chuẩn bằng giao diện VFS.

Những tập tin khác nhau có thể được cấp phát không gian trên đĩa trong 3 cách: kề, liên kết hay chỉ mục. Cấp phát kề có thể gặp phải sự phân mãnh ngoài. Truy xuất trực tiếp là kém hiệu quả với cấp phát liên kết. Cấp phát chỉ mục yêu cầu chi phí đáng kể cho khối chỉ mục của nó. Các giải thuật này có thể tối ưu trong nhiều cách. Không gian kề có thể lớn lên thông qua đoạn mở rộng để tăng khả năng linh hoạt và giảm phân mãnh ngoài. Cấp phát chỉ mục có thể được thực hiện trong việc nhóm nhiều khối để tăng thông lượng và giảm số lượng các mục từ chỉ mục được yêu cầu. Lập chỉ mục trong các nhóm là tương tự như cấp phát kề với các đoạn mở rộng.

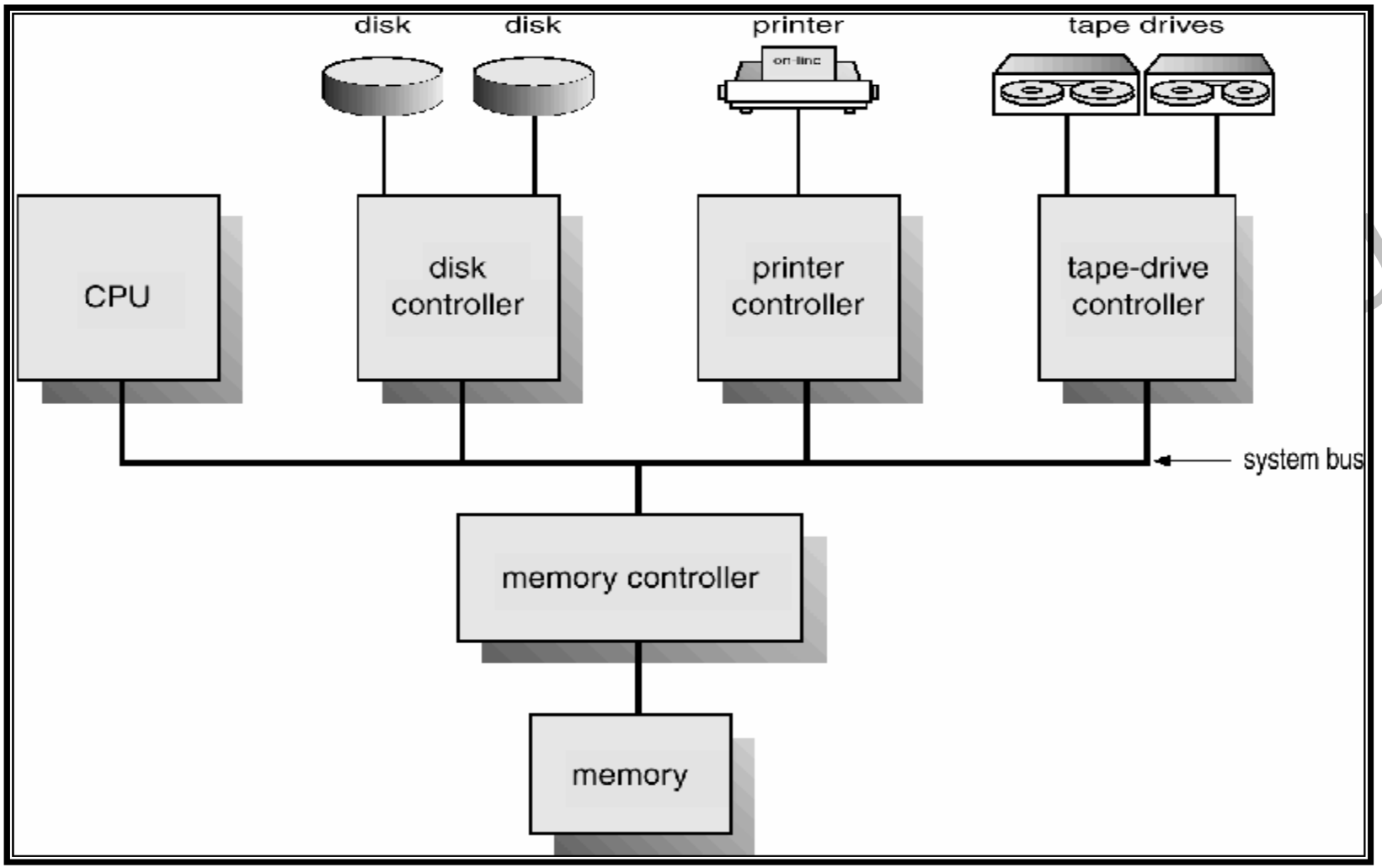
Các phương pháp cấp phát không gian trống cũng ảnh hưởng tới tính hiệu quả của không gian đĩa, năng lực hệ thống tập tin và khả năng tin cậy của thiết bị lưu trữ phụ. Các phương pháp được dùng gồm các vector bit và các danh sách liên kết. Các tối ưu gồm nhóm, đếm và FAT, mà đặt danh sách liên kết trong một vùng kề.

NGUYÊN LÝ HỆ ĐIỀU HÀNH



ThS. Nguyễn Đức Thuần
Bộ môn Hệ Thống Thông Tin
Đ.H THUỶ SẢN

Chương 0: Cấu trúc hệ thống máy tính



Cơ chế vận hành của hệ thống

Một hệ thống thông thường gồm CPU và các bộ điều khiển thiết bị (BĐKTB - device controller). Các tác vụ của CPU và BĐKTB có thể thực thi song hành và tranh chấp bộ nhớ chia sẻ qua sự điều khiển đồng bộ của memory controller.

Mỗi BĐKTB phụ trách một loại thiết bị cụ thể (video adapter, disk drive,...)

Mỗi BĐKTB có một buffer riêng.

CPU chuyển dữ liệu từ bộ nhớ chính đến buffer riêng của các BĐKTB và ngược lại. Dữ liệu của các tác vụ I/O được chuyển từ thiết bị đến buffer riêng của BĐKTB tương ứng.

Mỗi khi kết thúc một tác vụ I/O, BĐKTB thông báo cho CPU bằng cách gây ra *ngắt* (interrupts).

Ng Duc Thuan

Ngắt (interrupts)

Ngắt (interrupt) là cơ chế cho phép tạm ngưng công việc đang được CPU xử lý để phục vụ một yêu cầu khác.

Khi có ngắt xảy ra, quyền điều khiển được chuyển giao cho trình phục vụ ngắt (interrupt service routine) thông qua bảng vector ngắt.

Các đặc điểm của cơ chế ngắt :

- tăng hiệu suất xử lý công việc – đặc biệt là các tác vụ I/O
- ngắt thường được phát khởi bởi một thiết bị phần cứng
- thực hiện theo cơ chế bất đồng bộ (asynchronously) với process bị ngắt.

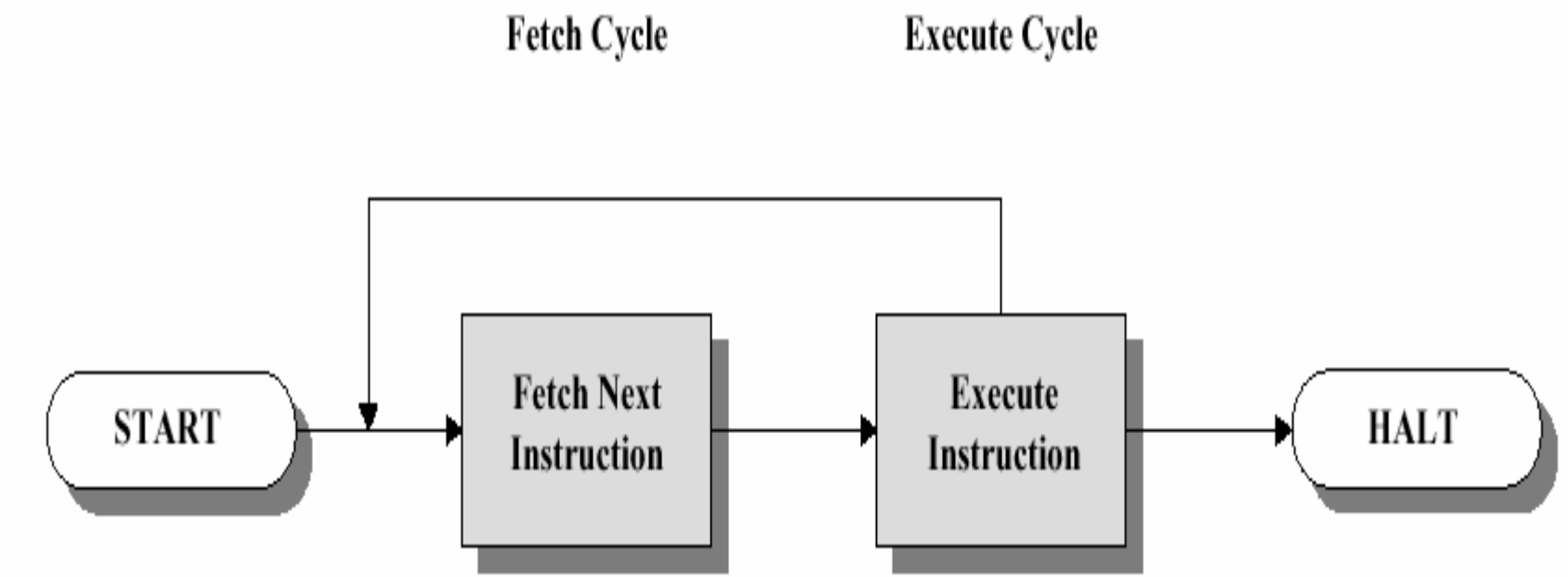
Trap: ngắt mềm, phát khởi bởi một lỗi nào đó hoặc do user yêu cầu. Ví dụ:

- Lỗi chia cho 0 (divided-by-zero), memory page fault…

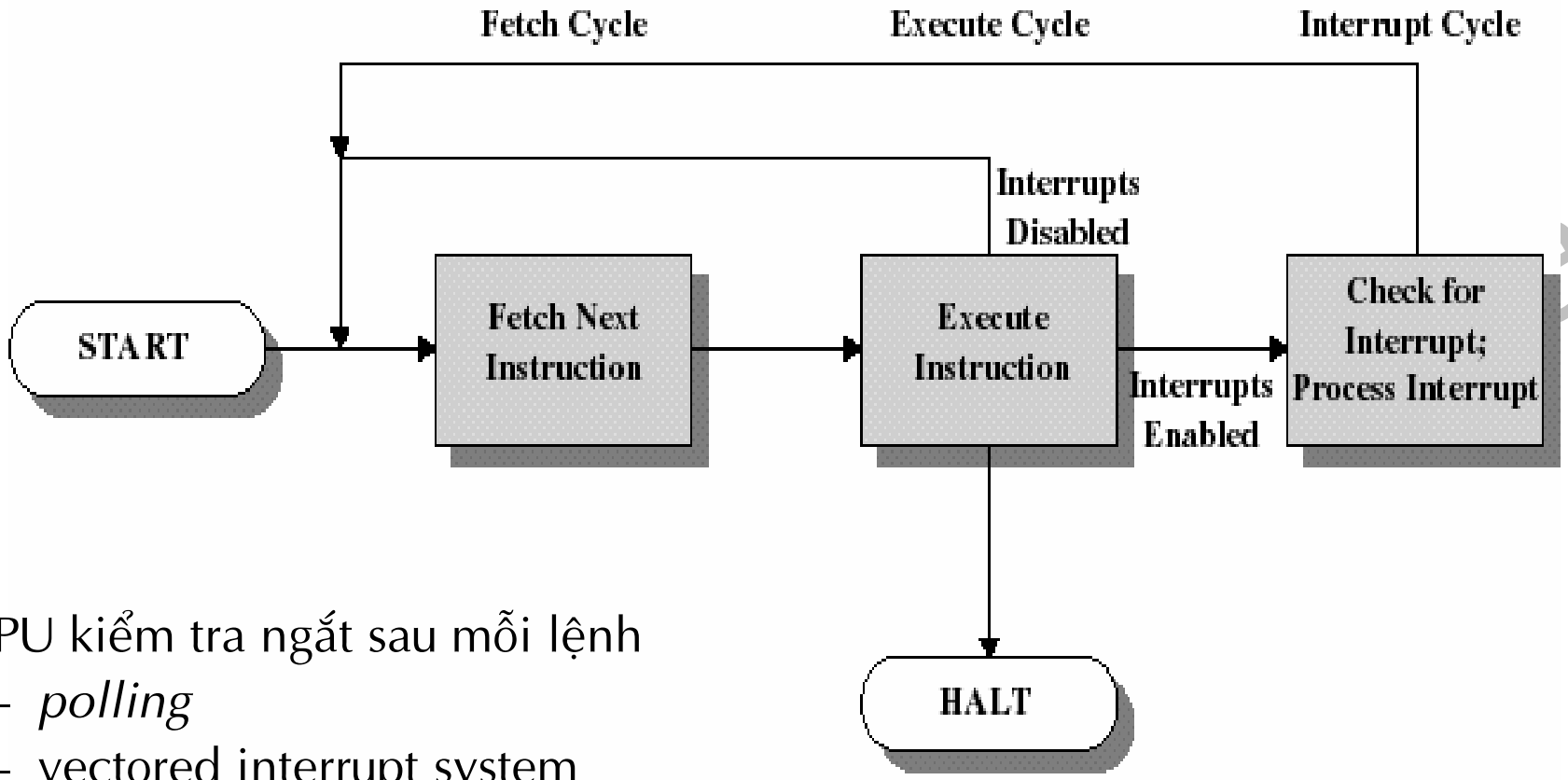
Hầu hết OS là interrupt-driven operating systems

Ng Duc Thuan

Chu trình thực hiện lệnh



Chu trình lệnh với ngắt quãng



CPU kiểm tra ngắt sau mỗi lệnh

- *polling*
- vectored interrupt system

Nếu không có ngắt, bắt đầu một chu kỳ lệnh mới

Nếu có ngắt đang chờ, tạm ngưng và lưu trạng thái chương trình đang thực thi (Program Check, Program Status Word, các thanh ghi dữ liệu,...), chuyển qua thực thi trình điều khiển xử lý ngắt (interrupt handler)

I/O structure

Hai phương thức thực hiện I/O

Sau khi bắt đầu tác vụ I/O, quyền điều khiển chỉ được trả về khi và chỉ khi tác vụ I/O hoàn tất.

- CPU phải chờ đến khi tác vụ I/O hoàn tất
- Trong một thời điểm, có nhiều nhất là 1 tác vụ I/O được thực hiện, không xử lý được đồng thời nhiều yêu cầu I/O.

Sau khi bắt đầu tác vụ I/O, quyền điều khiển được trả về ngay mà không cần chờ hoàn tất tác vụ I/O

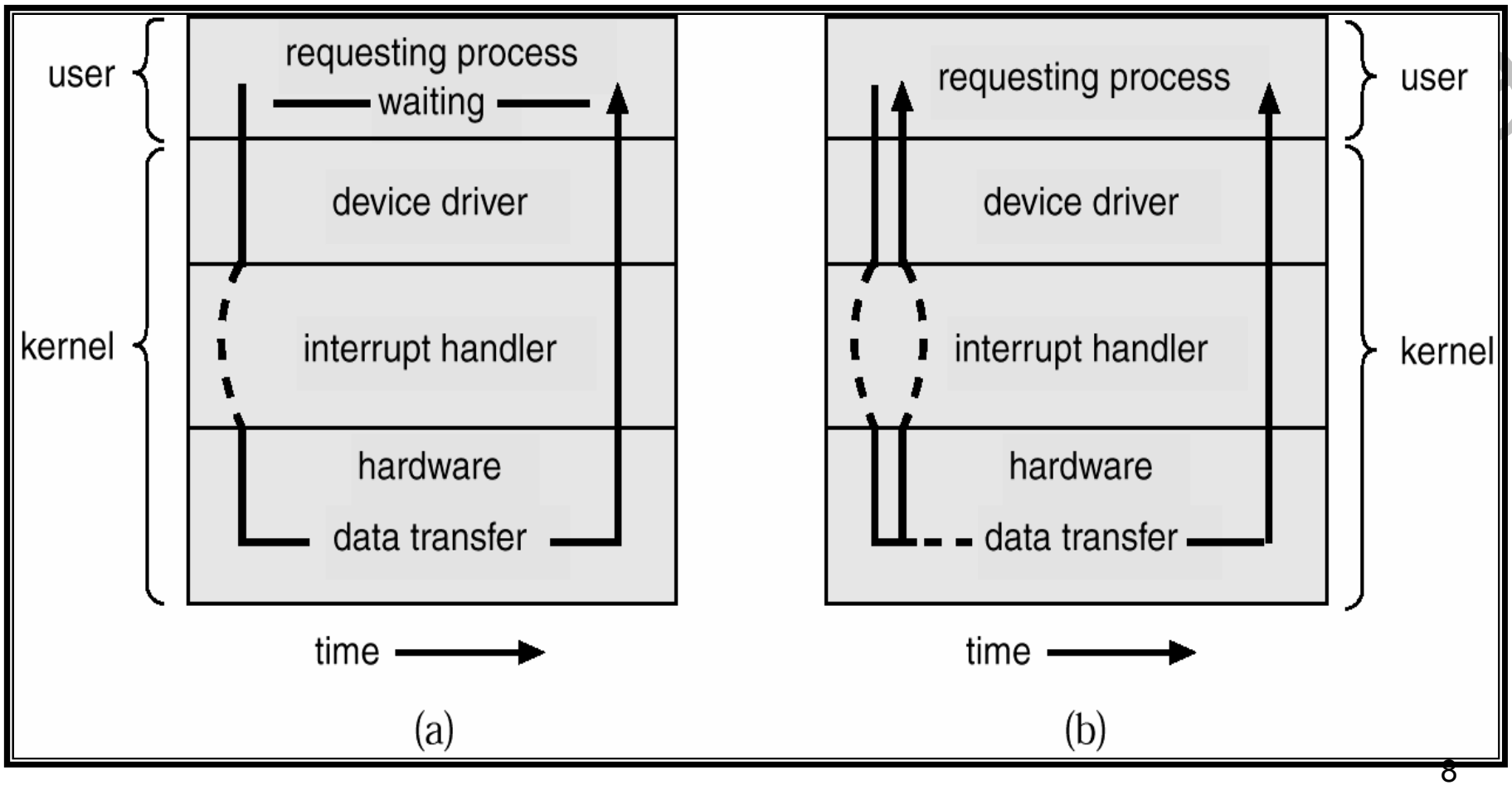
- CPU có thể thực thi lệnh kế tiếp mà không phải chờ đến khi tác vụ I/O hoàn tất
- Trong một thời điểm, có thể có nhiều tác vụ I/O được thực hiện đồng thời.

Ng Duc Thuan

Hai phương thức I/O

Synchronous I/O

Asynchronous I/O



Các kỹ thuật thực hiện I/O

Programmed I/O (PIO)

- Không dùng ngắt, CPU phải chờ tác vụ I/O hoàn tất

Interrupt-driven I/O

- CPU có thể tiếp tục thực thi trong suốt quá trình xảy ra I/O, khi tác vụ I/O hoàn tất thì sẽ phát ra một ngắt báo hiệu cho CPU.

Direct Memory Access (DMA)

- Một khối dữ liệu được chuyển trực tiếp giữa bộ nhớ và thiết bị I/O mà không cần qua CPU
- Thường sử dụng cho các thiết bị I/O có tốc độ cao.

Ng Duc Thuan

Storage structure

Bộ nhớ chính (main memory): CPU chỉ có thể truy cập trực tiếp thanh ghi (registers) và bộ nhớ RAM.

Bộ nhớ phụ (secondary storage): hệ thống lưu trữ thông tin bền vững (nonvolatile storage).

Đĩa từ (magnetic disks)

- Bề mặt đĩa chia thành các rãnh (*tracks*), các rãnh này được chia nhỏ hơn thành các cung từ (*sectors*).
- *Cylinder*: tập các track tạo thành một hình trụ
- *Disk controller*: bộ điều khiển quá trình giao tiếp giữa CPU và đĩa.

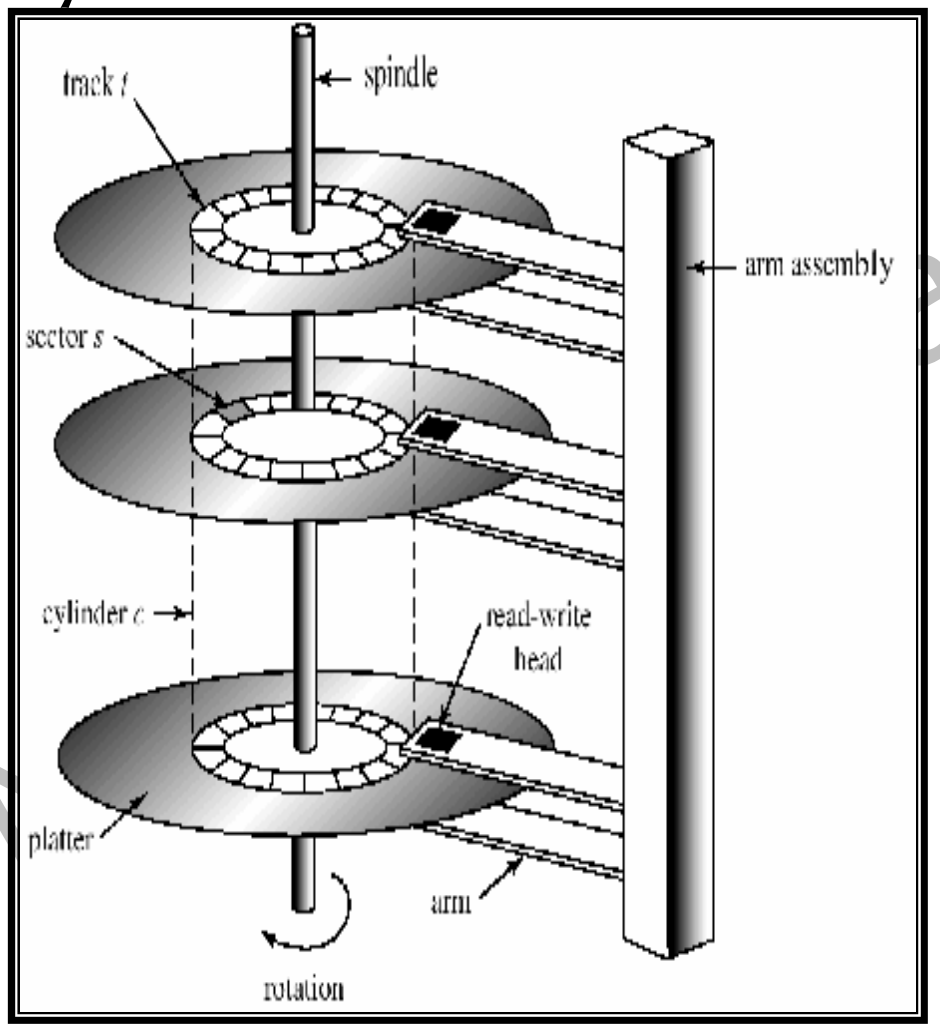
Ng Duc Thuan

Cơ chế truy xuất đĩa

Thời gian định vị dữ liệu - positioning time (random-access time) bao gồm

- Seek time: thời gian di chuyển đầu đọc đến cylinder cần đọc
- Rotational latency: thời gian tìm được sector cần đọc dữ liệu.

Transfer rate: tốc độ chuyển dữ liệu giữa đĩa và CPU



Phân cấp hệ thống lưu trữ

Hệ thống lưu trữ có thể phân cấp theo các tiêu chuẩn

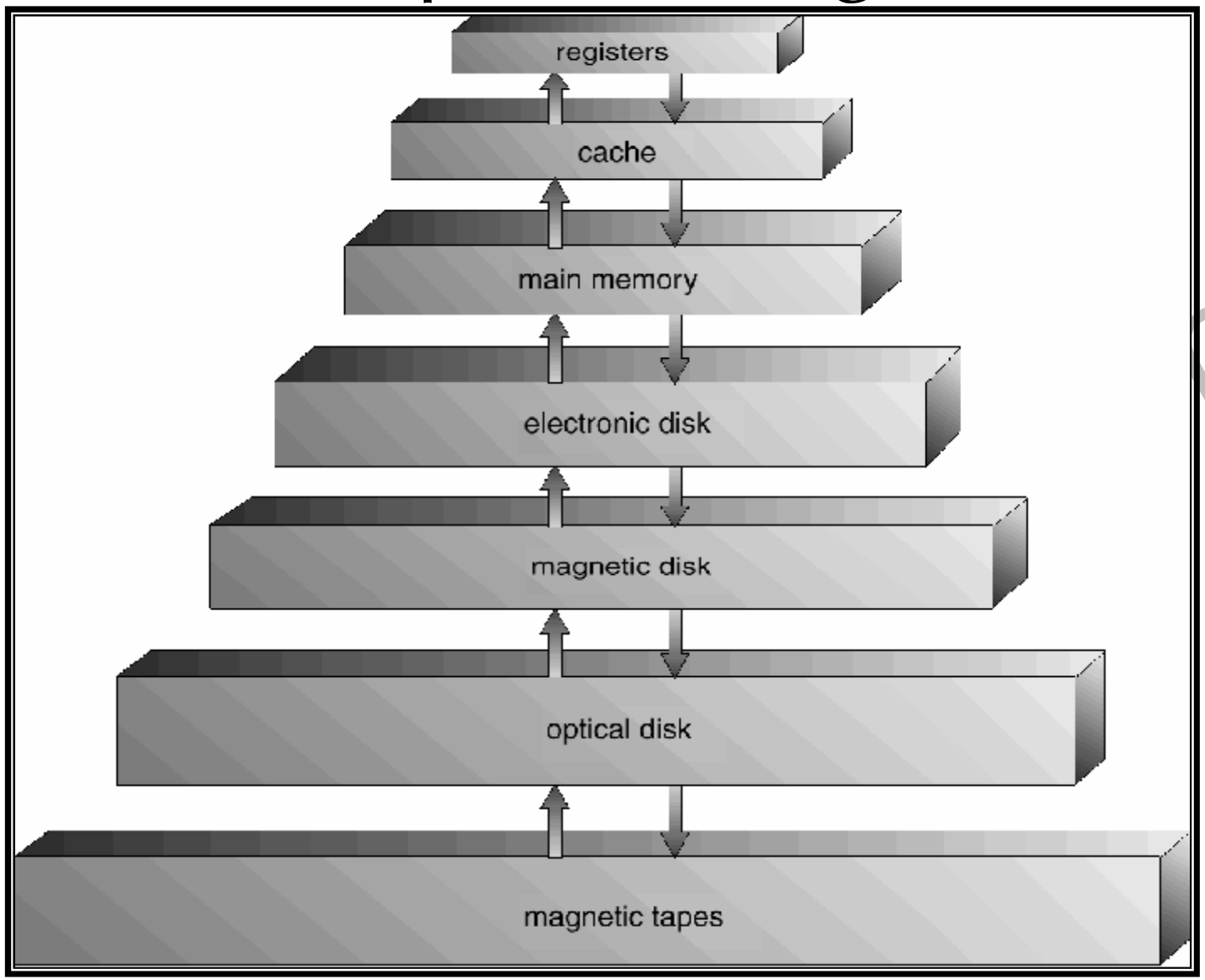
- Tốc độ (speed, access time).
- Dung lượng (capacity)
- Giá cả (cost)
- Độ bền vững dữ liệu (volatility)

Hệ thống phân cấp từ trên xuống theo

- Giá thành giảm dần
- Dung lượng tăng dần
- Thời gian truy cập tăng dần
- Tần suất truy cập của CPU giảm dần
- Chế độ lưu trữ bền vững (non-volatile)

Ng Duc Thuan

Phân cấp hệ thống lưu trữ



huan

Cơ chế caching

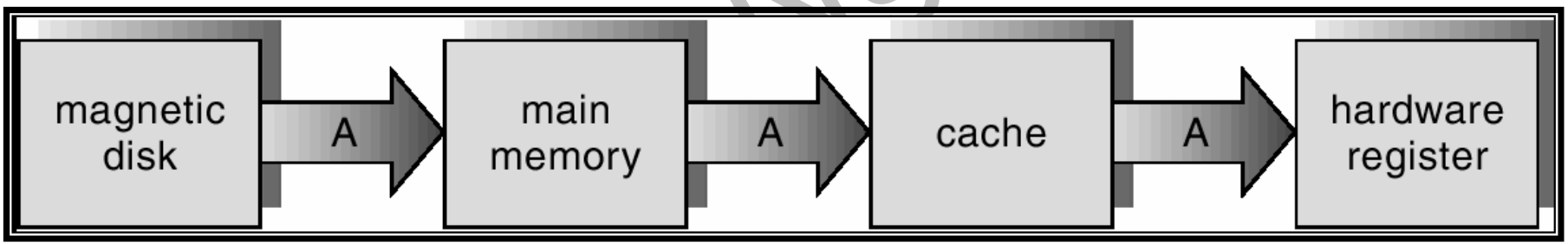
Caching: copy dữ liệu vào thiết bị lưu trữ tốc độ cao hơn.

Tại sao phải dùng bộ nhớ cache

- Tốc độ CPU và bộ nhớ chính khác nhau.
- Khai thác nguyên lý "locality"

Cần có chiến lược quản lý bộ nhớ cache thích hợp

Trong cơ chế caching, một dữ liệu có thể được lưu trữ nhiều nơi → phải bảo đảm tính nhất quán dữ liệu (consistent): bài toán cache coherency.



Ng Duc Thuan

Bảo vệ phần cứng

Chức năng của hệ điều hành là kiểm tra toàn bộ hệ thống.

Hệ điều hành cung cấp chức năng chia sẻ tài nguyên cho nhiều chương trình, vì vậy phải có cơ chế kiểm tra và xử lý lỗi ứng với mỗi một chương trình. Các lỗi chương trình do phần cứng phát hiện và thông báo cho hệ điều hành để ra quyết định chấm dứt và thông báo

Bảo vệ phần cứng bao gồm cơ chế bảo vệ CPU, Bộ nhớ, Thiết bị nhập xuất....

Ng Duc Thuan

Chương 1:Tổng quan về Hệ điều hành

- Khái niệm về hệ điều hành
- Phân loại các hệ điều hành
- Cấu trúc của hệ điều hành
- Lịch sử phát triển của hệ điều hành

Ng Duc Thuan

Chương 1:Tổng quan về Hệ điều hành

- Khái niệm về hệ điều hành
  - Các thành phần cộng tác trong Hệ thống CNTT gồm: phần cứng, HĐH, các ứng dụng và người sử dụng (user)
  - Phần cứng (CPU, bộ nhớ, thiết bị IO…): tài nguyên máy tính
  - Chương trình ứng dụng (trình biên dịch, hệ quản trị CSDL, phần mềm thương mại, trò chơi…) sử dụng tài nguyên máy tính để giải quyết các yêu cầu của user.
  - HĐH là chương trình hoạt động giữa user và phần cứng máy tính, điều khiển quản lý tài nguyên và phối hợp sử dụng phần cứng cho những ứng dụng khác nhau, giúp giao tiếp người-máy thuận lợi hiệu quả.

Vũ Đức Thuan

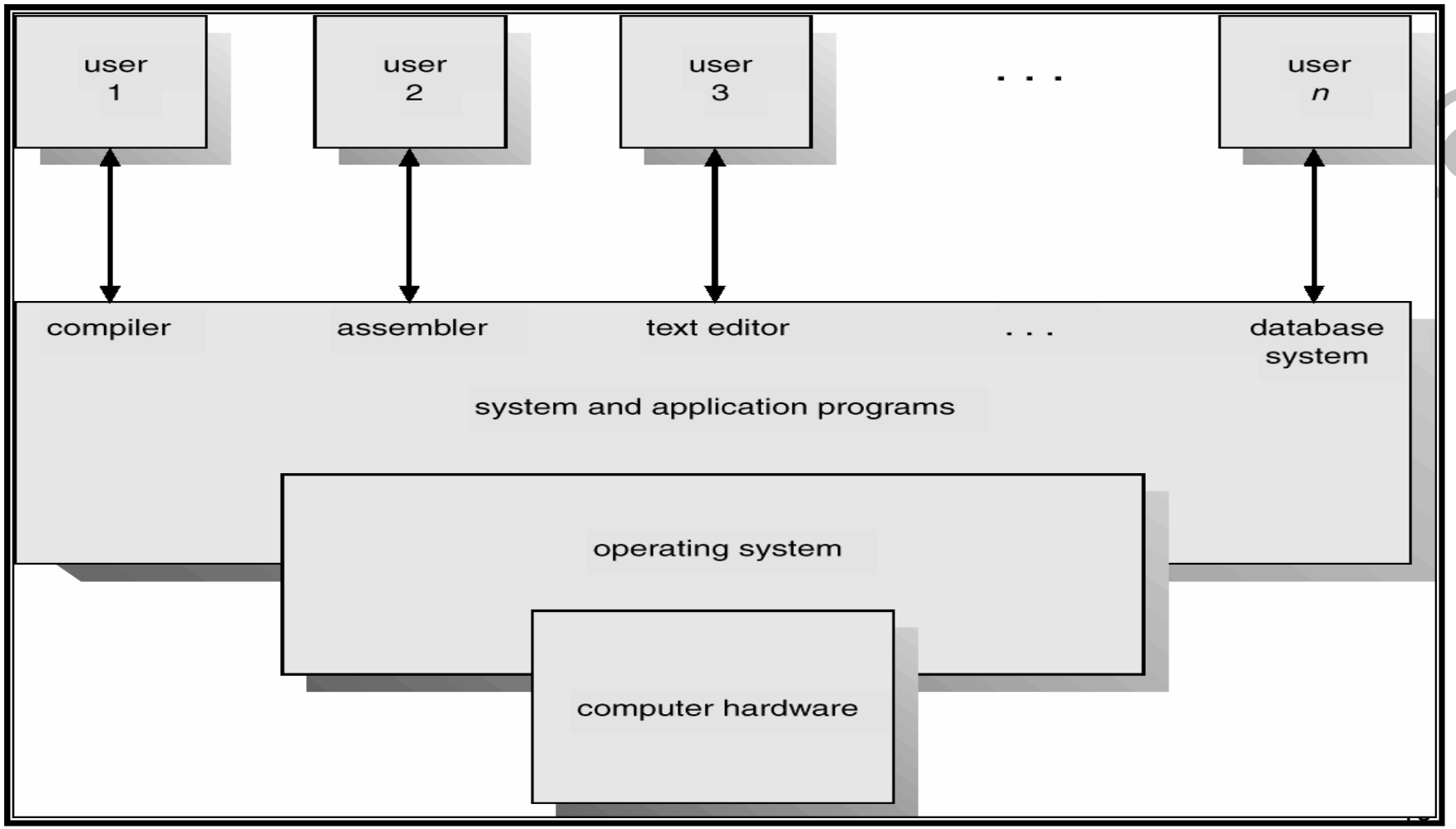
Chương 1:Tổng quan về Hệ điều hành

Users

Duc Thuan

Chương 1:Tổng quan về Hệ điều hành

Mô hình trừu tượng của 1 máy tính



Chương 1:Tổng quan về Hệ điều hành

- Khái niệm về hệ điều hành (tt)
  - HĐH là bộ điều phối tài nguyên của máy tính (thời gian sử dụng CPU, bộ nhớ, đĩa, thiết bị IO) cho các ứng dụng
  - Khi có nhiều yêu cầu khai thác tài nguyên, HĐH phải giải quyết vấn đề tranh chấp và quyết định cấp phát tài nguyên như thế nào là hiệu quả nhất
  - Để gia tăng hiệu quả khai thác tài nguyên, HĐH phải quản lý các bộ ĐKTB của nhà sản xuất (vd: VGA card, sound card, modem, printer, HDD…)

➔ Mục tiêu cơ bản của HĐH:

  - Giúp cho việc thi hành các chương trình dễ dàng hơn
  - Hỗ trợ các thao trên hệ thống máy tính hiệu quả hơn

Ng Duc Thuan

Chương 1:Tổng quan về Hệ điều hành

- Phân loại các hệ điều hành
  - Hệ thống xử lý theo lô (batch) đơn giản
  - Hệ thống xử lý theo lô đa chương
  - Hệ thống chia xẻ thời gian
  - Hệ thống song song
  - Hệ thống phân tán
  - Hệ thống xử lý thời gian thực

Ng Duc Thuan

Chương 1:Tổng quan về Hệ điều hành

1. ***Hệ thống xử lý theo lô đơn giản (đơn nhiệm, đơn chương)***

   Hệ điều hành xử lý theo lô thực hiện các công việc lần lượt theo những chỉ thị định trước. Việc thực hiện dãy các công việc một cách tự động nhờ một chương trình luôn nằm thường trú trong bộ nhớ máy tính được gọi *bộ giám sát thường trực*

   Ưu điểm:Thời gian thực hiện chương trình nhanh.

   Nhược điểm: CPU còn nhiều thời gian nhàn rỗi khi làm việc thiết bị nhập xuất:

   Khắc phục: Xử lý off_line, Spooling.

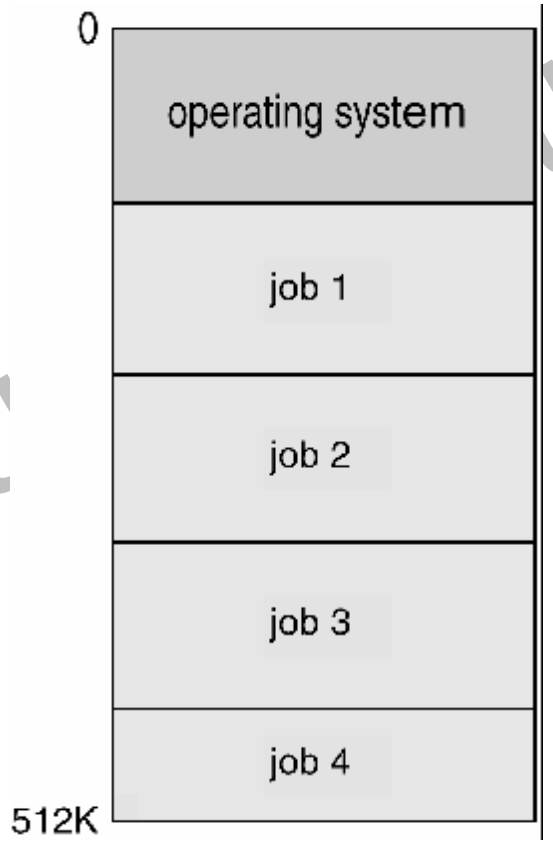
Ng Duc Thuan

Chương 1:Tổng quan về Hệ điều hành

Multi-programmed Systems

2. Hệ thống xử lý đa chương
   - Job pool: cấu trúc dữ liệu cho phép OS lựa chọn công việc được thực thi kế tiếp
   - Nhiều công việc được nạp vào bộ nhớ chính cùng lúc, thời gian xử lý của CPU được phân chia giữa các công việc đó
   - Tận dùng được thời gian rảnh, khi một công việc nào đó phải chờ I/O thì phải nhường CPU cho công việc khác (overlapping CPU - I/O).

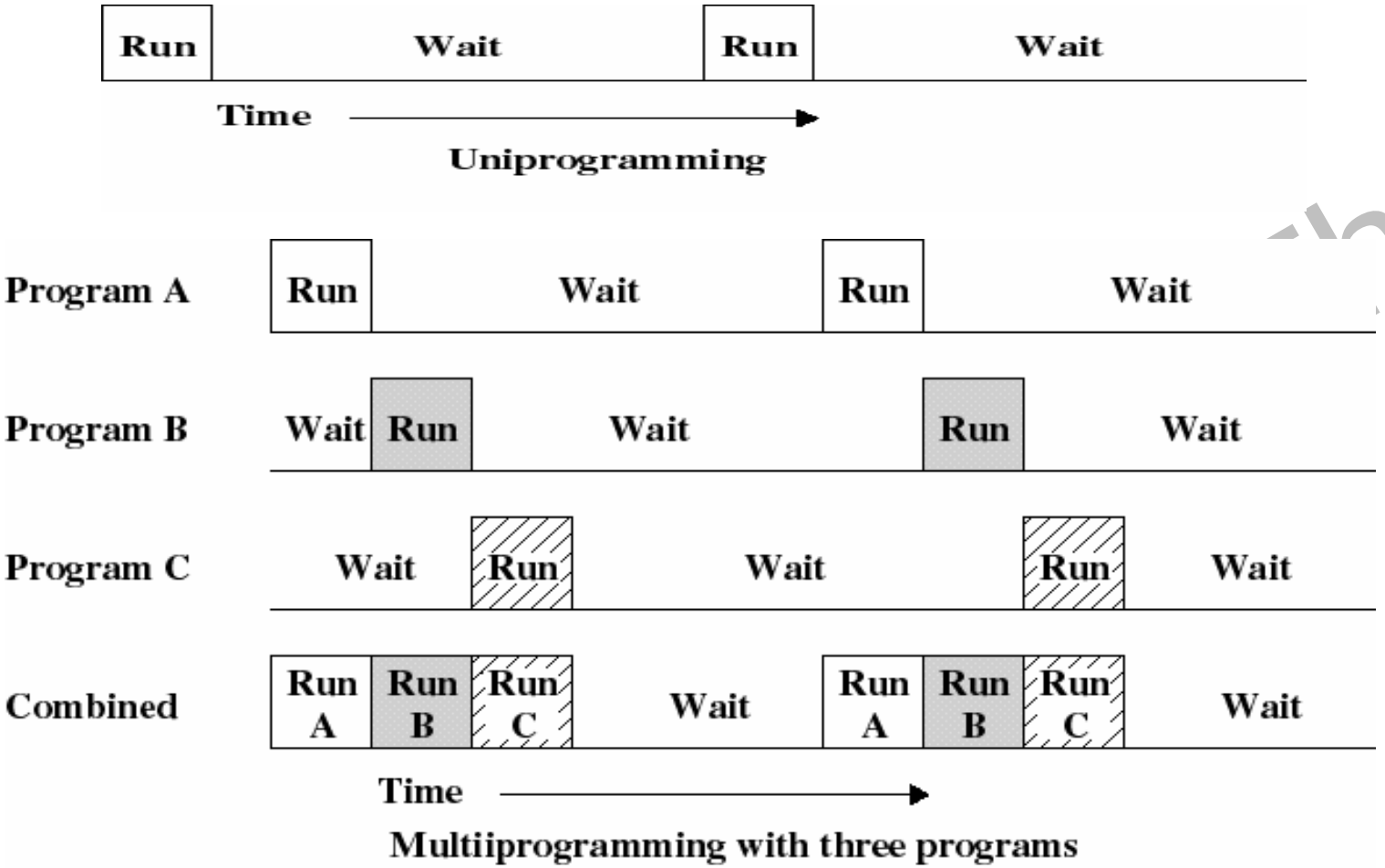
Yêu cầu đối với OS
   - Job Scheduling
   - Memory management
   - CPU scheduling
   - Allocation of devices
   - Protection



Ng Du

Chương 1:Tổng quan về Hệ điều hành

So sánh multi-programming và uni-programming



Chương 1:Tổng quan về Hệ điều hành

3. ***Hệ thống chia sẻ thời gian*** Time-Sharing (multitasking) Systems
   - Multi-programmed systems không cung cấp khả năng tương tác với users
   - CPU luân phiên chuyển đổi thực thi giữa các công việc

     Quá trình chuyển đổi xảy ra ***thường xuyên hơn***, mỗi công việc chỉ được chia một phần nhỏ thời gian CPU

     Cung cấp sự tương tác giữa hệ thống với user
     - Khi kết thúc thực thi một lệnh, OS sẽ chờ lệnh kế tiếp từ bàn phím chứ không phải từ card reader

     Một công việc chỉ được chiếm CPU để xử lý khi nó nằm trong bộ nhớ chính

     Khi cần thiết, một công việc nào đó có thể được chuyển từ bộ nhớ chính ra thiết bị lưu trữ, nhường bộ nhớ chính cho công việc khác.

Ng Duc Thuan

Chương 1:Tổng quan về Hệ điều hành

Yêu cầu đối với OS trong Time-Sharing Systems

- Định thời công việc (job scheduling)
- Quản lý bộ nhớ (Memory Management)
  - Các công việc được hoán chuyển giữa bộ nhớ chính và đĩa
  - Virtual memory: cho phép một công việc có thể được thực thi mà không cần phải nạp hoàn toàn vào bộ nhớ chính
- Quản lý các process (Process Management)
  - Định thời CPU (CPU scheduling)
  - Đồng bộ các công việc (synchronization)
  - Tương tác giữa các công việc (process communication)
  - Tránh Deadlock
- Quản lý hệ thống file, hệ thống lưu trữ (disk management)
- Phân bổ các thiết bị, tài nguyên
- Cơ chế bảo vệ (protection)

Ng Duc Thuan

Chương 1:Tổng quan về Hệ điều hành

4. ***Hệ thống song song***

- Gồm nhiều bộ vi xử lý cùng chia sẻ hệ thống đường dẫn dữ liệu, đồng bộ, bộ nhớ và các thiết bị ngoại vi. Các bộ vi xử lý liên lạc bên trong với nhau.
- Với nhiều bộ vi xử lý công việc thực hiện được thực hiện sẽ nhanh hơn, những không phải có n vi xử lý nhanh hơn gấp n lần so hệ thống 1 vi xử lý.
- Độ tin cậy trong hệ thống nhiều vi xử lý là rất cao.
- Hệ thống đa xử lý thường sử dụng cách đa xử lý đối xứng. Một số hệ thống đa xử lý bất đối xứng.

Symmetric multiprocessing (SMP)

  - Các processor vận hành cùng một hệ điều hành duy nhất.
  - Nhiều ứng dụng thực thi cùng lúc với performance cao.
  - Đa số các hệ điều hành hỗ trợ SMP.

Asymmetric multiprocessing

  - Mỗi processor thực thi một công việc khác nhau, master processor định thời và phân công việc cho các slave processors.

Ng Duc Thuan

Chương 1:Tổng quan về Hệ điều hành

5. Hệ thống phân tán.

- Hệ thống phân tán: trên mỗi máy trong mạng cài đặt một hệ điều hành khác nhau. Máy chủ chủ yếu thực hiện một số chức năng sau: Quản lý các kho dữ liệu tập trung, cung cấp một số dịch vụ truyền dữ liệu, tổ chức xử lý khi có yêu cầu từ máy trạm. Mô hình khách chủ(Client-Server) phân chia quản lý.
- Nguyên nhân xây dựng HĐH phân tán:
  - Chia sẻ tài nguyên.
  - Tăng tốc độ tính toán.
  - An toàn
  - Thông tin liên lạc được với nhau

  Yêu cầu cơ sở hạ tầng mạng máy tính
  – LAN,WAN

  Dựa trên mô hình client-server hoặc peer-to-peer

Ng Duc Thuan

Chương 1:Tổng quan về Hệ điều hành

6. Hệ thống thời gian thực

Hệ thống thời gian thực (Real-Time Systems)

- Thường dùng trong các thiết bị chuyên dụng như điều khiển các thử nghiệm khoa học, điều khiển trong y khoa, dây chuyền công nghiệp.
- Ràng buộc tương đối chặt chẽ về thời gian: hard và soft real-time.

Hard real-time:

- Hạn chế (hoặc không có) bộ nhớ phụ, tất cả dữ liệu nằm trong bộ nhớ chính (RAM) hoặc ROM
- Yêu cầu thời gian đáp ứng, xử lý rất nghiêm ngặt, thường sử dụng trong điều khiển công nghiệp, công nghệ robotics.

Soft real-time

- Thường xuất hiện trong lĩnh vực multimedia, thực tế ảo (virtual reality) với yêu cầu mềm dẻo hơn về thời gian.

Ng Duc Thuan

Chương 1:Tổng quan về Hệ điều hành

1.3.1 Các thành phần của hệ thống gồm:

- Quản lý tiến trình
- Quản lý bộ nhớ chính
- Quản lý bộ nhớ phụ
- Quản lý hệ thống nhập xuất
- Quản lý hệ thống tập tin
- Hệ thống bảo vệ
- Hệ thống cơ chế dòng lệnh

Ng Duc Thuan

Chương 1:Tổng quan về Hệ điều hành

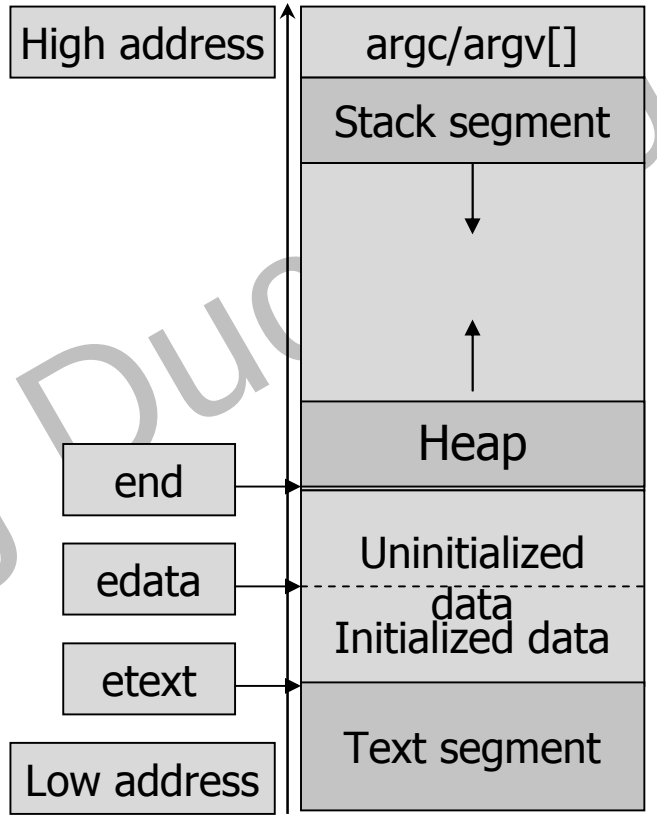
- Quản lý tiến trình
  - Tiến trình là một loạt các công việc được thi hành (proram, batch processing, spooling, connecting…)
  - Tiến trình sử dụng tài nguyên máy tính (CPU, bộ nhớ, tập tin, thiết bị…) để phục vụ công việc của nó
  - Khi tiến trình khởi tạo, nó đòi hỏi nhiều tài nguyên hệ thống.
  - Khi tiến trình kết thúc, HĐH phải thu hồi hoặc tái tạo tài nguyên để có thể dùng lại cho các tiến trình khác
  - Tiến trình được biên dịch thành các tập tin thụ động trên đĩa
  - Khi tiến trình được kích hoạt, HĐH sẽ khởi tạo tài nguyên ban đầu theo yêu cầu, nạp tập chỉ thị vào bộ nhớ và thi hành theo cơ chế tuần tự. Tiến trình chuyển sang hoạt động

Vũ Đức Thuan

Chương 1:Tổng quan về Hệ điều hành

Cấu trúc 1 tiến trình của UNIX

```
1.  int a = 0, b, *c;
2.  int main( int argc, char *argv[ ] ) {
3.        b= increase(a);
4.        c =(int*)malloc(10*sizeof(int));
5.        c[5]= b;
6.  }
7.  int  increase(int x) {
8.        return x ++;
9.  }
```



Ng Duc

Chương 1:Tổng quan về Hệ điều hành

- Quản lý bộ nhớ chính
  - Bộ nhớ chính là trung tâm của các thao tác và xử lý
  - Về mặt vật lý, bộ nhớ là các chip nhớ tĩnh điện
  - Về mặt luận lý, bộ nhớ là mảng các bit nhớ tổ chức theo đơn vị lưu trữ (byte, word hay dword)
  - Mỗi ô nhớ được HĐH định vị bằng cơ chế đánh địa chỉ riêng
  - Các bộ ĐKTB, HĐH, ứng dụng… đều lưu trữ dữ liệu vào bộ nhớ
  - Các chương trình muốn thi hành phải được ánh xạ thành địa chỉ tuyệt đối, nạp vào bộ nhớ chính để hệ thống truy xuất đến
  - Khi tiến trình kết thúc, dữ liệu vẫn còn trong bộ nhớ cho đến khi một tiến trình khác ghi chồng lên (hoặc tắt máy)

Ng Đức Thuan

Chương 1:Tổng quan về Hệ điều hành

- Quản lý bộ nhớ chính (tt)
  - Do ứng dụng có nhu cầu sử dụng bộ nhớ khác nhau, nên HĐH phải có nhiều kế hoạch quản trị bộ nhớ hiệu quả
  - Kế hoạch quản trị bộ nhớ của HĐH phụ thuộc vào đặc điểm phần cứng và nhu cầu sử dụng của user
  - Vai trò của HĐH trong việc quản lý bộ nhớ chính:
    - Cấp phát và thu hồi bộ nhớ khi cần thiết
    - Quyết định tiến trình nào được nạp vào bộ nhớ chính, địa chỉ nạp
    - Lưu giữ thông tin về các vị trí trong bộ nhớ đã sử dụng, tiến trình nào đang sử dụng

Ng Duc Thuan

Chương 1:Tổng quan về Hệ điều hành

- Quản lý bộ nhớ phụ
  - Nhược điểm của bộ nhớ chính:
    - Quá nhỏ để có thể lưu giữ mọi dữ liệu và chương trình
    - Mất dữ liệu khi tắt máy
  - Hệ thống lưu trữ phụ dùng đĩa lưu trữ chương trình, dữ liệu
  - Vai trò của HĐH trong việc quản lý đĩa:
    - Định vị lưu trữ, truy xuất đĩa
    - Quản lý vùng trống
    - Lập lịch cho đĩa
  - Hiệu năng của hệ thống tùy thuộc rất nhiều vào tốc độ đọc/ghi. Vì vậy HĐH phải có cơ chế quản lý đĩa hiệu quả.

Ng Duc Thuan

Chương 1:Tổng quan về Hệ điều hành

- Quản lý hệ thống nhập xuất
  - Mỗi nhà sản xuất đều cung cấp các device driver ĐKTB
  - Nhiệm vụ của HĐH là tạo mặt giao tiếp thân thiện giữa user và thiết bị thông qua các giao thức ĐKTB tổng quát
  - Một hệ thống nhập xuất bao gồm:
    - Hệ thống buffer cahing
    - Giao tiếp điều khiển thiết bị tổng quát
    - Bộ điều khiển cho các thiết bị phần cứng

Ng Duc Thuan

Chương 1:Tổng quan về Hệ điều hành

- Quản lý hệ thống tập tin
  - Máy tính có thể lưu trữ thông tin trên nhiều dạng thiết bị vật lý khác nhau (băng từ, đĩa từ, đĩa quang, thẻ nhớ…)
  - Mỗi dạng thiết bị lưu trữ có đặc điểm riêng về tổ chức vật lý, tốc độ truy xuất, khả năng lưu trữ, tốc độ truyền…
  - HĐH định nghĩa đơn vị lưu trữ logic là tập tin cùng với cấu trúc đĩa luận lý để truy xuất thông qua bộ điều khiển đĩa
  - Tập tin: tập hợp thông tin của chương trình và/hoặc dữ liệu
  - Các loại tập tin:
    - Thi hành: chứa ứng dụng, phần mềm. Vd: COM, EXE
    - Văn bản: chứa ký tự ASCII, chương trình nguồn
    - Nhị phân: chứa dữ liệu của các ứng dụng

Ng Duc Thuan

Chương 1:Tổng quan về Hệ điều hành

- Quản lý hệ thống tập tin (tt)
  - Vai trò của HĐH trong việc quản lý tập tin:
    - Tạo, xóa tập tin
    - Tạo, xóa thư mục
    - Hỗ trợ các thao tác trên tập tin và thư mục
    - Ánh xạ tập tin trên hệ thống lưu trữ phụ
    - Backup tập tin trên các thiết bị lưu trữ

Ng Duc Thuan

Chương 1:Tổng quan về Hệ điều hành

- Hệ thống bảo vệ
  - Hệ thống có nhiều user, nhiều tiến trình đồng thời
  - HĐH cung cấp cơ chế đảm bảo tài nguyên (tập tin, bộ nhớ, CPU, đồng hồ, thiết bị ngoại vi…) chỉ được truy xuất bởi những tíên trình có quyền
  - Hệ thống bảo vệ là cơ chế kiểm soát quá trình truy xuất của chương trình, tiến trình, user trên tài nguyên hệ thống
  - Gia tăng độ an toàn khi kiểm tra lỗi của các giao tiếp giữa những hệ thống nhỏ bên trong

Ng Duc Thuan

Chương 1:Tổng quan về Hệ điều hành

- Hệ thống cơ chế dòng lệnh
  - Mỗi HĐH có những giao tiếp khác nhau: cơ chế dòng lệnh, giao diện có các biểu tượng, cửa sổ thao tác dùng chuột…
  - Cơ chế dòng lệnh là giao tiếp tương tác lệnh giữa user và HĐH
  - Các HĐH thiết kế cơ chế dòng lệnh bên trong hạt nhân hoặc tách cơ chế dòng lệnh thành một ứng dụng đặc biệt:
    - DOS: thi hành khi bắt đầu công việc (COMMAND.COM)
    - Unix: thi hành khi user login lần đầu tiên
  - Các lệnh được đưa vào HĐH nhờ bộ thông dịch lệnh qua cơ chế dòng lệnh hoặc Shell và được thực hiện tuần tự
  - Các lệnh có quan hệ với việc tạo và quản lý các tiến trình, kiểm soát nhập xuất, quản lý bộ nhớ, quản lý đĩa, truy xuất tập tin và cơ chế bảo vệ.

Ng Duc Thuan

Chương 1:Tổng quan về Hệ điều hành

- Các dịch vụ của hệ điều hành
  - Thi hành chương trình: Nạp chương trình, chấm dứt bình thường hay bất thường (lỗi nghiêm trọng)
  - Thao tác nhập xuất: cung cấp dịch vụ hỗ trợ nhập xuất
  - Thao tác trên hệ thống tập tin, truy xuất đĩa
  - Truyền thông điệp giữa các tiến trình (máy đơn, mạng)
  - Phát hiện lỗi do CPU, bộ nhớ, thiết bị, tiến trình gây ra. Mỗi dạng lỗi được HĐH giải quyết tương ứng

Ng Duc Thuan

Chương 1:Tổng quan về Hệ điều hành

- Lời gọi hệ thống
  - Cung cấp giao tiếp giữa tiến trình và HĐH. Có 2 dạng:
    - Cấp thấp: các lệnh hợp ngữ
    - Cấp cao: các hàm/thủ tục thiết kế bằng NNLT cấp cao.
  - Trong các NLT cấp cao, user không quan tâm đến chi tiết mà chỉ cần thông qua các hàm hay các lệnh để gọi thực hiện
  - Có 3 phương pháp chuyển tham số cho HĐH: thanh ghi, ngăn xếp hoặc cấu trúc bảng
  - Các loại lời gọi hệ thống: kiểm soát tiến trình, tao tác tập tin, thao tác thiết bị, truyền thông điệp

Ng Duc Thuan

Chương 1:Tổng quan về Hệ điều hành

- Các chương trình hệ thống
  - Thao tác tập tin, mô tả tập tin
  - Thông tin trạng thái (ngày giờ, dung lượng đĩa, bộ nhớ…)
  - Hỗ trợ các NNLT: trình biên dịch, thông dịch tích hợp trong nhân hay phát hành riêng
  - Nạp và thi hành chương trình: nạp, định vị, liên kết, debug
  - Thông điệp: dùng liên lạc giữa các tiến trình, các máy
  - Các chương trình ứng dụng đi kèm: định dạng, sao chép đĩa, soạn thảo văn bản, vẽ hình đơn giản…

Ng Duc Thuan

Chương 1:Tổng quan về Hệ điều hành

- Cấu trúc hệ thống
  - Cấu trúc đơn giản
  - Cấu trúc theo lớp
  - Cấu trúc Máy ảo
  - Mô hình client-server

Ng Duc Thuan

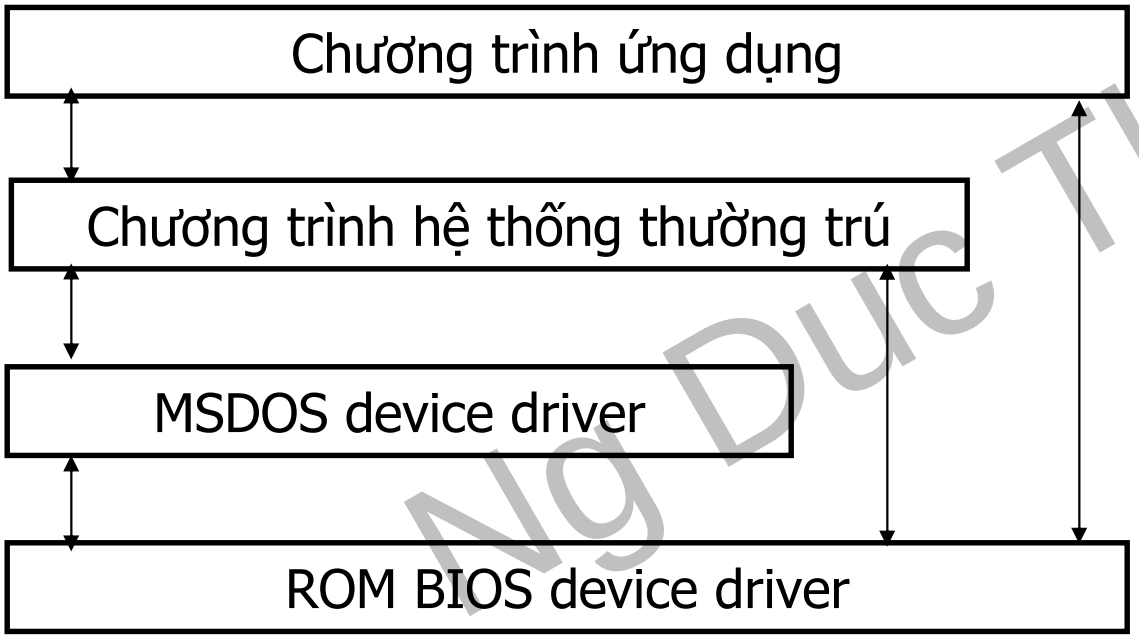
Chương 1:Tổng quan về Hệ điều hành

- Cấu trúc đơn giản
  - Khởi đầu của HĐH là hệ thống nhỏ, đơn giản và có giới hạn
  - Các HĐH cấu trúc đơn giản quen thuộc: MSDOS, UNIX
  - Hệ điều hành MSDOS:
    - HĐH cấu trúc đơn giản, không chia thành những đơn thể rõ rệt
    - Cung cấp những chức năng cần thiết nhất trong không gian nhỏ, chạy trên phần cứng giới hạn
    - Giữa giao diện và chức năng không phân chia rõ rệt
    - Các ứng dụng có thể truy xuất trực tiếp thủ tục nhập xuất

Ng Duc Thuan

Chương 1:Tổng quan về Hệ điều hành

Cấu trúc của MS-DOS



Ng Duc Thuan

Chương 1:Tổng quan về Hệ điều hành

- Cấu trúc đơn giản
  - HĐH UNIX: cấu trúc gồm hạt nhân và chương trình hệ thống
    - Hạt nhân gồm chuỗi giao tiếp và bộ ĐKTB (device driver)
    - Hạt nhân cung cấp hệ thống tập tin, lập lịch CPU, quản trị bộ nhớ và những chức năng khác thông qua lời gọi hệ thống
    - Lời gọi hệ thống định nghĩa giao tiếp lập trình cho UNIX
    - Các chương trình hệ thống dùng lời gọi hệ thống do hạt nhân hỗ trợ để cung cấp những chức năng hữu ích như biên dịch và thao tác tập tin

Ng Đức Thuan

Chương 1:Tổng quan về Hệ điều hành
Cấu trúc UNIX

<table>
<tr><td colspan="3">Người sử dụng</td></tr>
<tr><td colspan="3">Shell và lệnh
Biên dịch và thông dịch
Thư viện hệ thống</td></tr>
<tr><td colspan="3">Giao tiếp lời gọi hệ thống với hạt nhân</td></tr>
<tr><td>Tín hiệu kiểm soát hệ thống, nhập xuất tuần tự của bộ điều khiển terminal</td><td>Hệ thống tập tin chuyển đổi giữa hệ thống nhập xuất khối và bộ điều khiển đĩa</td><td>Lập lịch CPU, thay thế phân trang, yêu cầu phân trang trong bộ nhớ ảo</td></tr>
<tr><td colspan="3">Giao tiếp giữa hạt nhân với phần cứng</td></tr>
<tr><td>Bộ kiểm soát terminal</td><td>Bộ kiểm soát đĩa</td><td>Bộ kiểm soát bộ nhớ</td></tr>
</table>

Ng Duc Thuan

Chương 1:Tổng quan về Hệ điều hành

- Cấu trúc theo lớp
  - Chia chức năng hệ thống thành nhiều phần nhỏ
  - Cấm ứng dụng sử dụng các hàm truy xuất cấp thấp
  - HĐH được chia thành nhiều lớp: lớp phần cứng, lớp hệ thống, lớp giao tiếp user…
  - Mỗi lớp gồm một số CTDL và các hàm được gọi từ lớp trên. Bản thân lớp chỉ gọi được các chức năng của lớp dưới hỗ trợ
  - Mỗi lớp không cần biết HĐH và các lớp khác được cài đặt như thế nào. Nó chỉ biết nhiệm vụ và các thao tác trên lớp

Ng Duc Thuan

Chương 1:Tổng quan về Hệ điều hành

Cấu trúc của Hệ điều hành THE

- HĐH Technische Hogesschool Eindhoven
  - Lớp 5: Chương trình ứng dụng
  - Lớp 4: Tạo buffer cho thiết bị nhập xuất
  - Lớp 3: Device driver thao tác màn hình
  - Lớp 2: Quản lý bộ nhớ ảo
  - Lớp 1: Lập lịch CPU
  - Lớp 0: Phần cứng

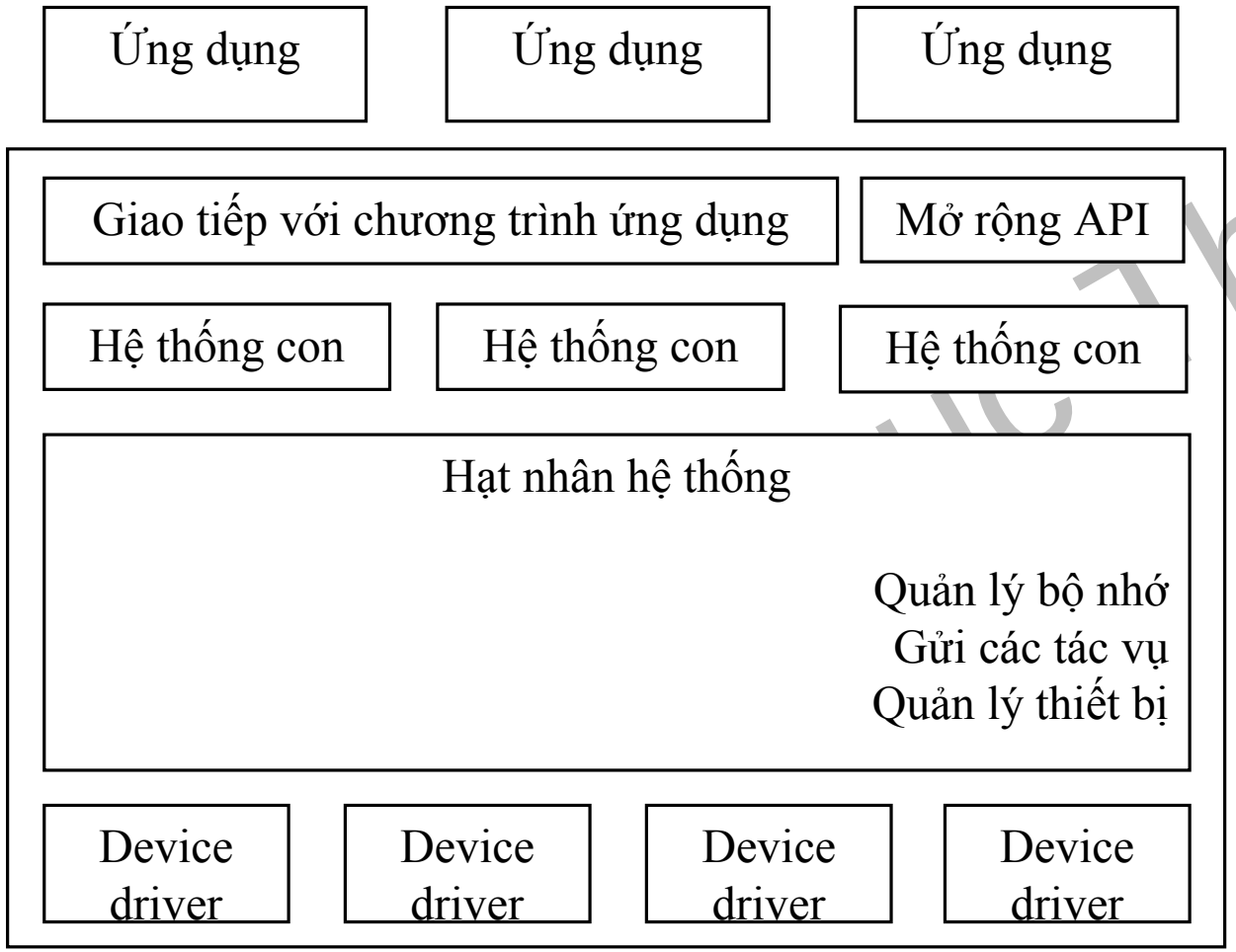
Ng Duc Thuan

Chương 1:Tổng quan về Hệ điều hành

- Các lớp của HĐH VENUS
  - Lớp 6: Chương trình ứng dụng
  - Lớp 5: Device driver và bộ lập lịch
  - Lớp 4: Bộ nhớ ảo
  - Lớp 3: Kênh nhập xuất
  - Lớp 2: Lập lịch CPU
  - Lớp 1: Thông dịch các địa chỉ
  - Lớp 0: Phần cứng

Ng Duc Thuan

Chương 1:Tổng quan về Hệ điều hành



Thuan

Chương 1:Tổng quan về Hệ điều hành

- Máy ảo (Virtual Machine)
  - HĐH = <Ứng dụng>|<Hệ thống>|<Nhân>|<Phần cứng>
  - Nhân HĐH dùng chỉ thị của phần cứng để tạo thư viện Lời gọi hệ thống
  - Các chương trình hệ thống có thể sử dụng lời gọi hệ thống hoặc chỉ thị phần cứng: (SysCall+HWare) ∈ SysClass
  - Các ứng dụng có thể gọi các chương trình hệ thống: (SysFunc+SysClass) ∈ Virtual Machine
  - Bằng kỹ thuật lập lịch CPU và bộ nhớ ảo, HĐH có thể tạo nhiều tiến trình phức ảo hoạt động như một máy tính có bộ xử lý và bộ nhớ riêng.

Ng Đức Thuan

Chương 1:Tổng quan về Hệ điều hành

- Máy ảo (tt)
  - Các tiến trình của máy ảo hoạt động như lời gọi hệ thống cùng hệ thống tập tin không truy xuất trực tiếp phần cứng
  - Tài nguyên của máy tính được chia xẻ để tạo các máy ảo: CPU ảo, bộ nhớ ảo, thiết bị ảo, tập tin (và đĩa) ảo.
  - Máy ảo thực hiện ở 2 mức: giám sát và sử dụng
  - Vấn đề cơ bản của máy ảo: hệ thống đĩa (ảo), vận chuyển dữ liệu và thời gian thi hành
  - Lợi thế của máy ảo: mỗi ứng dụng hoạt động độc lập vì được cung cấp môi trường như một máy riêng (CPU, bộ nhớ, bộ chỉ thị, thiết bị ngoại vi…).

Ng Duc Thuan

Chương 1:Tổng quan về Hệ điều hành

- Mô hình client-server
  - Các đoạn mã hệ thống có xu hướng chuyển dần lên lớp cao.
  - Chức năng của hạt nhân được chuyển bớt cho các tiến trình của user, chỉ còn lại các chức năng cơ bản (vd: kiểm soát quá trình thông tin giữa client-server)
  - HĐH được chia thành các phần nhỏ, mỗi phần kiểm soát một mặt của hệ thống
  - Mô hình hoạt động: client gửi yêu cầu, server thực hiện, gửi kết quả cho client
  - Ưu điểm: tương thích với mô hình hệ thống phân tán

Ng Duc Thuan

Chương 1:Tổng quan về Hệ điều hành

- Lịch sử phát triển các hệ điều hành
  - Thế hệ 1: 1945-1955
  - Thế hệ 2: 1955-1965
  - Thế hệ 3: 1965-1980
  - Thế hệ 4: 1980 - nay

Ng Duc Thuan

Chương 1:Tổng quan về Hệ điều hành

- Thế hệ 1: 1945-1955
  - Máy tính điện tử dùng đèn do Howard Aiken và John von Neuman thiết kế. Kích thước lớn. Tốc độ tính toán chậm
  - Chưa có NNLT và HĐH. Chưa có sự phân công: người thiết kế, lập trình, thao tác, quản lý, sử dụng… đều là một
  - Lập trình bằng ngôn ngữ máy tuyệt đối
  - Sử dụng bảng điều khiển thực hiện các tính năng cơ bản
  - 1950: phiếu đục lỗ ra đời, thay cho bảng điều khiển

Ng Duc Thuan

Chương 1:Tổng quan về Hệ điều hành

- Thế hệ 2: 1955-1965
  - Máy tính điện tử bán dẫn được sản xuất cho khách hàng
  - Kích thước và tốc độ được cải thiện
  - Có sự phân chia rõ ràng giữa người thiết kế, xây dựng, vận hành, lập trình, bảo trì.
  - NNLT: sử dụng Hợp ngữ, FORTRAN mã hóa chỉ thị trên phiếu đục lỗ. Xuất kết quả ra máy in
  - Hệ thống xử lý theo lô (lưu trên băng từ) ra đời
  - Chương trình điều khiển hệ xử lý theo lô là tiền thân của các HĐH ngày nay

Ng Duc Thuan

Chương 1:Tổng quan về Hệ điều hành

- Thế hệ 3: 1965-1980
  - Máy IBM360 được trang bị mạch tích hợp IC
  - Kích thước giảm thiểu. Tốc độ gia tăng.
  - Các thiết bị ngoại vi bắt đầu xuất hiện. Thao tác điều khiển dần dần phức tạp hơn.
  - Hệ điều hành ra đời, từ đơn giản đến phức tạp: hệ đa chương, hệ chia xẻ thời gian
  - Hệ máy mini bắt đầu phát triển

Ng Duc Thuan

Chương 1:Tổng quan về Hệ điều hành

- Thế hệ 4: 1980-nay
  - Máy tính cá nhân (PC-Personal Computer) ra đời
  - Hệ máy IBMPC sử dụng HĐH MS-DOS chiếm lĩnh thị trường máy cá nhân trên toàn thế giới
  - HĐH mạng, HĐH phân tán phát triển mạnh
  - Giao diện đồ họa (GUI-Graphic User Interface), Multimedia, Internet…

Ng Duc Thuan

Chương 2: Giới thiệu 1 số Hệ điều hành

- Hệ điều hành MSDOS
- Hệ điều hành Windows
- Hệ điều hành Windows NT
- Hệ điều hành Novel Netware
- Hệ điều hành Unix

Ng Duc Thuan

Chương 2: Giới thiệu 1 số Hệ điều hành

- Hệ điều hành MSDOS
  - Khái niệm
  - Lịch sử phát triển
  - Quá trình khởi động
  - Tập lệnh của MSDOS

Ng Duc Thuan

Chương 2: Giới thiệu 1 số Hệ điều hành

- Khái niệm về MSDOS
  - Microsoft Disk Operating System là một trong những HĐH được sử dụng rộng rãi nhất trên thế giới
  - MSDOS (gọi tắt là DOS) là hệ điều hành có cấu trúc đơn giản, đơn nhiệm (monotasking), một người dùng, truy xuất hệ thống nhiều cấp từ cao đến thấp
  - Giao tiếp dòng lệnh dùng tham số (Shell từ DOS 4.0)
  - Đòi hỏi cấu hình thấp: CPU AT/XT, 640KB bộ nhớ, màn hình màu CGA/EGA/VGA hoặc đơn sắc. Cho phép mở rộng thiết bị

Ng Duc Thuan

Chương 2: Giới thiệu 1 số Hệ điều hành

- Khái niệm về MSDOS (tt)
  - Tổ chức các chương trình của MSDOS bao gồm:
    - Chương trình khởi động
    - Chương trình Shell
    - Chương trình chức năng
    - Chương trình nhập xuất
    - Hệ thống các tiện ích

Ng Duc Thuan

Chương 2: Giới thiệu 1 số Hệ điều hành

- Lịch sử phát triển của MSDOS
  - Tháng 8-1981, Microsoft phát hành HĐH MSDOS 16 bits cho máy PC có CPU 8088 theo đơn đặt hàng của IBM
  - Version 1.0 có 4000 dòng Hợp ngữ, 12KB bộ nhớ, tổ chức thành 3 tập tin:
    - IBMBIO.COM: hệ thống nhập xuất tuần tự và đĩa
    - IBMMSDOS.COM: hệ thống tập tin, kiểm soát IO, giao tiếp
    - COMMAND.COM: trình xử lý lệnh (nội trú, ngoại trú)
  - Đề xuất khái niệm thiết bị IO độc lập (CON, PRN, AUX), định dạng tập tin ứng dụng COM, EXE. Hỗ trợ xử lý lệnh theo lô
  - Version 1.1: Đĩa mềm lưu trữ 5.2 inch (360 KB, 2 mặt).

Ng Duc Thuan

Chương 2: Giới thiệu 1 số Hệ điều hành

- Lịch sử phát triển của MSDOS (tt)
  - Version 2.0: phát hành vào 3-1983 gồm 20,000 dòng lệnh, cài trên đĩa cứng 10MB.
  - Chuyển IBMMIO.COM thành tập danh sách liên kết thiết bị.
  - Sử dụng cấu trúc đĩa thứ bậc, thẻ file, đường ống, lọc sắp xếp, device driver cấu trúc mở…
  - Cài đặt thiết bị, định dạng ngày tháng, tiền tệ, mã quốc gia vào lúc khởi động trong CONFIG.SYS.
  - Version 2.11 được chuyển thành hơn 60 ngôn ngữ

Võ Đức Thuan

Chương 2: Giới thiệu 1 số Hệ điều hành

- Lịch sử phát triển của MSDOS (tt)
  - Version 3.0 phát hành 8-1984 gồm 40,000 dòng lệnh.
  - Hỗ trợ máy IBMPC AT, 20MB đĩa cứng, đĩa mềm 1.2MB
  - Sử dụng cấu hình CMOS, đĩa ảo VDISK
  - Version 3.1 (11-1984) giới thiệu Microsoft Network
  - Version 3.2 (1986) hỗ trợ đĩa mềm 3.5". Cung cấp các lệnh REPLACE, XCOPY, DRIVER.SYS
  - Version 3.3 hỗ trợ cổng truyền thông, ổ đĩa 1.44MB, đĩa cứng 32MB trên máy AT 80286, 80386
  - Version 4.0 (7-1988) hỗ trợ đĩa cứng đến 2GB. Sử dụng RAM mở rộng làm đĩa ảo. Giới thiệu MS-DOS Shell.

Ng Duc Thuan

Chương 2: Giới thiệu 1 số Hệ điều hành

- Lịch sử phát triển của MSDOS (tt)
  - Version 5.0 (4-1991), sử dụng RAM mở rộng để lưu user device driver. Tận dụng vùng nhớ 640-1MB để giải phóng bớt vùng nhớ cơ sở
  - Cải tiến Shell cho phép nạp nhiều ứng dụng cùng lúc
  - Chú trọng nhiều hơn đến user. Bổ sung Help mở rộng
  - Version 6.0 (1993): tăng dung lượng đĩa bằng DBLSPACE. Tạo cache đĩa bằng SMARTDRV. Tối ưu bộ nhớ bằng MEMMAKER. Chống virus với MSAV. Phục hồi tập tin bằng UNDELETE. Kiểm tra đĩa bằng SCANDISK.

NG Duc Thuan

Chương 2: Giới thiệu 1 số Hệ điều hành

- Quá trình khởi động. Các khái niệm cơ bản của DOS
  - Lệnh nội trú: thư viện xử lý các lệnh phổ biến, thường trú trong bộ nhớ
  - Lệnh ngoại trú: lệnh đặc biệt, ít dùng, lưu trên đĩa dưới dạng các tập tin thi hành
  - Các dạng thông báo lỗi: không tìm thấy lệnh, lệnh sai cú pháp, không thể thực hiện lệnh
  - Khởi động máy bằng đĩa mềm hoặc đĩa cứng.
  - Mục tiêu của quá trình khởi động:
    - Kiểm tra sự hoạt động của thiết bị
    - Cài đặt chương trình của HĐH

Ng Duc Thuan

Chương 2: Giới thiệu 1 số Hệ điều hành

- Quá trình khởi động máy tính
  - POST-Power On Self Test: do chương trình trong ROM thực hiện (trước khi HĐH được kích hoạt):
    - Kiểm tra thiết bị
    - Cài đặt thông số hệ thống vào RAM
    - Nạp Master boot/Boot sector vào bộ nhớ. Chuyển điều khiển
  - Khởi động từ đĩa mềm: POST-Boot sector-HĐH
  - Khởi động từ đĩa cứng: POST-Master boot-Boot sector-HĐH
  - Master boot chứa đoạn chương trình nạp Boot sector vào bộ nhớ chính và bảng phân hoạch đĩa cứng
  - Boot sector chứa đoạn chương trình nạp HĐH vào bộ nhớ chính và bảng tham số đĩa

Ng Đức Thuan

Chương 2: Giới thiệu 1 số Hệ điều hành

- Quá trình khởi động HĐH MSDOS:
  - Cài đặt MSDOS vào RAM
  - Khởi tạo các vector ngắt tương ứng
  - Tìm đọc tập tin cấu hình CONFIG.SYS:
    - Nếu tìm thấy, thiết lập cấu hình theo chỉ dẫn.
    - Nếu không thấy, thiết lập cấu hình chuẩn
  - Nạp DOS Shell, chuyển điều khiển
  - Thực hiện tập tin lô AUTOEXEC.BAT
  - Xuất dấu đợi lệnh (DOS Prompt):
    - Khởi động từ đĩa mềm: A:\>_
    - Khởi động từ đĩa cứng: C:\>_

Ng Đức Thuan

Chương 2: Giới thiệu 1 số Hệ điều hành

- Các tập lệnh của MSDOS:
  - Thông tin hệ thống
  - Làm việc với đĩa
  - Làm việc với thư mục
  - Thao tác với tập tin
  - Quản lý nhập xuất
  - Thiết lập môi trường

Ng Duc Thuan

Chương 2: Giới thiệu 1 số Hệ điều hành

- Hệ điều hành Windows
  - Khái niệm
  - Lịch sử phát triển
  - Quá trình khởi động
  - Tập lệnh

Ng Duc Thuan

Chương 2: Giới thiệu 1 số Hệ điều hành

- Khái niệm về Hệ điều hành Windows
  - HĐH đa nhiệm của Microsoft. Giao diện đồ họa thân thiện, dễ sử dụng, thích hợp cho mọi đối tượng người dùng
  - Yêu cầu phần cứng không cao: 386SX, VGA, 4MB RAM, 80MB HDD. Quản lý thiết bị tốt, trong suốt với user
  - Tương thích với máy IBMPC. Các HĐH khác có thể cùng hoạt động với Windows. DOS trở thành ứng dụng của Windows. HĐH có tính ổn định cao, ít hỏng hóc.
  - Khai thác được khả năng tối đa của máy. Cài đặt thay đổi cấu hình hệ thống dễ dàng: Plug and Play (Windows 95)

Ng Duc Thuan

Chương 2: Giới thiệu 1 số Hệ điều hành

- Khái niệm về Hệ điều hành Windows (tt)
  - Giao diện cửa sổ, biểu tượng. Hỗ trợ tên file dài, giao tiếp dữ liệu OLE.
  - Chạy trong chế độ bảo vệ (protected mode). Định vị bộ nhớ phẳng. Phá vỡ rào cản 640 MB bộ nhớ cơ sở.
  - Hỗ trợ các ứng dụng 32 bits. Giao tiếp trực tiếp với các thiết bị thông tin. Liên kết trong các hệ thống mạng.
  - Dễ học, dễ sử dụng, hỗ trợ lập trình tốt… Windows đã đưa Microsoft trở thành 1 trong những công ty hàng đầu thế giới trong lĩnh vực CNTT

Ng Đức Thuan

Chương 2: Giới thiệu 1 số Hệ điều hành

- Lịch sử phát triển của HĐH Windows
  - 11-1983 công bố. 11-1985 phát hành version 1.0
  - Version 2.0 (11-1987) thay đổi giao diện, cửa sổ phủ lấp. Mở rộng giao tiếp bàn phím, chuột cho menu và dialog box
  - 1987-1990: Windows 2.x (80386 Virtual mode). Multitask các ứng dụng DOS, Windows. Truy xuất trực tiếp phần cứng
  - 5-1990: Windows 3.0 protect mode 80x86. Truy cập 16MB bộ nhớ. Shell (Program, Task, File Manager) hoàn chỉnh.
  - 4-1992: Windows 3.1 protect mode. Hỗ trợ font True type, multimedia, OLE, common dialog box.
  - 9-1995: Windows 95 GUI hoàn chỉnh. Độc lập thiết bị, hỗ trợ CD-ROM, Plug&Play. Hỗ trợ mạng cục bộ, remote control.

Ng Đức Thuan

Chương 2: Giới thiệu 1 số Hệ điều hành

- Quá trình khởi động của HĐH Windows
  - POST-Power On Self Test
  - Nạp các thành phần cho chế độ thực
  - Nạp VxD tĩnh và chế độ bảo vệ
  - Nạp HĐH Windows và khởi tạo desktop

Ng Duc Thuan

Chương 2: Giới thiệu 1 số Hệ điều hành

- Nạp các thành phần cho chế độ thực (real-mode)
  - Boot sector nạp IO.SYS. Chuyển điều khiển
  - IO.SYS có chức năng quản lý cấu hình real-mode, graphic mode. IO.SYS nạp các tập tin tối thiểu, phân tích tập tin CONFIG.SYS, thi hành đồ họa (real-mode, protected mode)
  - Xác định tiểu sử phần cứng (hardware profile). Khởi tạo thiết bị phần cứng cho chế độ thực. Nạp MSDOS.SYS
  - MSDOS.SYS định vị địa chỉ các tập tin và các chức năng khởi động Windows 95.
  - Xử lý CONFIG.SYS. Nạp COMMAND.COM để xử lý lệnh ở chế độ DOS thực. Thi hành AUTOEXEC.BAT. Chuyển điều khiển cho quá trình nạp bộ quản lý máy ảo VMM32 ở chế độ thực

Ng Duc Thuan

Chương 2: Giới thiệu 1 số Hệ điều hành

- Nạp VxD tĩnh và chế độ bảo vệ (protected-mode)
  - VMM32 nạp các thiết bị mô tả trong SYSTEM.INI. Tìm driver cho thiết bị. Nếu tìm không thấy, nạp driver mặc định
  - Chuyển từ chế thực sang chế độ bảo vệ
  - Khởi động bộ quản lý cấu hình chế độ bảo vệ. Khởi tạo các VxD có sẵn. Xác định các VxD cần nạp thêm
  - Nhận dạng thông tin phần cứng
  - Giải quyết các xung đột về tài nguyên
  - Khởi tạo các thiết bị

Ng Duc Thuan

Chương 2: Giới thiệu 1 số Hệ điều hành

- Nạp Windows 95 files, khởi tạo desktop
  - Nạp SYSTEM.INI: thông tin cấu hình hệ thống
  - Nạp KERNEL32.DLL: hạt nhân của Windows 95
  - Nạp GDI.EXE, GDI32.DLL: cơ chế đồ họa của Windows 95
  - Nạp USER.EXE, USER32.DLL giao tiếp user, quản lý cửa sổ
  - Liên kết tài nguyên, font
  - Nạp WIN.INI chứa các cấu hình về chương trình và user
  - Nạp Shell (EXPLORER.EXE), các thành phần của desktop
  - Nếu sử dụng mạng, xuất hiện hộp thoại login.

Ng Duc Thuan

Chương 2: Giới thiệu 1 số Hệ điều hành

- Tập lệnh (Shell) Windows 95
  - Thao tác với các đối tượng
    - Các đối tượng: file, folder, program, printer, modem…
    - Tên: dùng mô tả và xác định đối tượng
    - Thuộc tính: kích thước, vị trí, tên, thời gian…
  - Các thành phần của desktop
    - My Computer: Drives, Control Panel, Printer, DialUp
    - Shortcut: Name, Address, Icon, Application…
    - Network Neighborhood: Workgroup, Server, Printer, Shared Folder…
  - Recycled Bin: Restore Deleted Items
  - Taskbar: Start button, Quick Launch, System Tray
  - Start: Programs, Documents, Settings, Find, Run, Help, Shutdown

Ng Duc Thuan

Chương 2: Giới thiệu 1 số Hệ điều hành

- Hệ điều hành Windows NT. Khái niệm
  - HĐH mạng dành cho LAN và Workgroup của Microsoft
  - Đúc kết từ ưu điểm của các HĐH khác như Unix, OS/2…
  - Windows NT có 2 phiên bản: Server và Workstation
  - Windows NT Workstation phục vụ nhiều chức năng mạng: print server, router TCP/IP…
  - Windows NT sử dụng profiles lưu trữ thông tin chi tiết về môi trường của user qua các phiên làm việc
  - Vừa là HĐH máy đơn, vừa là file server, application server, print server…, Windows NT là HĐH thân thiện, mạnh mẽ và linh hoạt

Ng Duc Thuan

Chương 2: Giới thiệu 1 số Hệ điều hành

- Hệ điều hành Windows NT. Đặc điểm
  - Dễ tương thích (portability):
    - NT được viết bằng ngôn ngữ C
    - Các phần được xây dựng trên lớp phần cứng trừu tượng
    - Chạy được trên các máy Intel x86, RISC (MIPS, AXP, PowerPC)
  - Khả năng bộ nhớ: định vị đến 4GB địa chỉ bộ nhớ
  - Đa nhiệm: thực hiện nhiều tiến trình cùng lúc
  - Hỗ trợ nhiều CPU:
    - Workstation 4.0: 2 CPU
    - Server 4.0: 4 CPU
    - Chuyên dụng: 255 CPU

Ng Duc Thuan

Chương 2: Giới thiệu 1 số Hệ điều hành

- Hệ điều hành Windows NT. Đặc điểm (tt)
  - Hệ thống tập tin NTFS (New Technology File System):
    - Tên file dài đến 255 ký tự. Phân biệt chữ hoa, thường
    - Logical Partition đến 1064 GB
    - Nén file/folder giảm kích thước 40-50%
  - An toàn: nhiều cấp độ an tòan trên mức chia xẻ, quyền truy cập dữ liệu mạng
  - Tương thích: hỗ trợ nhiều loại ứng dụng từ nhiều HĐH khác nhau (MSDOS, Windows 16bits, Windows 32 bits, OS/2, POSIX)

Ng Duc Thuan

Chương 2: Giới thiệu 1 số Hệ điều hành

- Hệ điều hành Windows NT. Đặc điểm (tt)
  - Hỗ trợ mạng:
    - Dễ quản trị: user profiles, quản trị domain, lập lịch tác vụ…
    - Hỗ trợ API (Application Programming Interface): NetBIOS , Socket…
    - Hỗ trợ giao thức truyền thông: NetBEUI, NWLink, TCP/IP, DLC
    - Giao tiếp với Netware: các phần mềm client có thể truy xuất đến server Netware
    - Dịch vụ truy xuất từ xa (RAS-Remote Access Service) cho phép user kết nối vào LAN thông qua modem, vào WAN sử dụng X25
    - Dịch vụ Internet: hỗ trợ cho mạng TCP/IP các dịch vụ world wide web, FTP server, Gopher server…

Ng Duc Thuan

Chương 2: Giới thiệu 1 số Hệ điều hành

- Quản trị user, nhóm user trên Windows NT
  - Bộ quản lý User Manager for Domain
  - Tạo shortcut cho User Manager for Domain
  - Lựa chọn Domain
  - Thêm user account
  - Copy user account
  - Thay đổi thuộc tính của user acount
  - Gán user vào các nhóm
  - Quản lý nhóm

Ng Duc Thuan

Chương 2: Giới thiệu 1 số Hệ điều hành

- Sử dụng Windows NT Workstation Client
  - Windows NT Workstation có 3 thành phần chính:
    - Nghi thức mạng
    - Định hướng lại network client
    - Cấu hình tài nguyên
  - Logon/Logoff Windows NT Workstation
    - Logon cục bộ
    - Logon domain

Ng Duc Thuan

Chương 2: Giới thiệu 1 số Hệ điều hành

- Hệ điều hành Novell Netware
  - 1983: mạng S-Net ra đời, là tiền thân của Novell Netware
  - Các năm sau, Novell giới thiệu Netware 86 (file server, truy xuất hệ thống tập tin an toàn, hỗ trợ quản trị mạng)
  - 1986: Advanced Netware hỗ trợ LAN, cho phép nối nhiều kiểu mạng trong file server hoặc workstation bên ngoài
  - Advanced Netware 286 xây dựng trên CPU 80286, multitasking, protected mode. Server hỗ trợ 4 kiểu mạng
  - Version Netware 386 3.0 (1989) là HĐH 32 bits thích hợp cho các mạng lớn. An toàn, hiệu quả, linh động, tin cậy

Ng Duc Thuan

Chương 2: Giới thiệu 1 số Hệ điều hành

- Hệ điều hành Novell Netware (tt)
  - 1991: Netware 3.11 hỗ trợ tập tin DOS, Macintosh, Windows, OS/2, Unix và các dịch vụ in ấn
  - 1995: Netware 4.1 cung cấp các dịch vụ thiết yếu: thư mục, tập tin, thông điệp hợp nhất, định hướng đa nghi thức, quản trị mạng, an tòan, in ấn.
  - Các sản phẩm của Novell: Netware SFT III 3.12, Netware 4.1 for OS/2, Netware for Unix.

Ng Duc Thuan

Chương 2: Giới thiệu 1 số Hệ điều hành

- Đặc điểm của HĐH Netware 4.1
  - HĐH mạng 32 bit CPU 80386, cung cấp các dịch vụ cho trạm
  - Dịch vụ xác minh user và login
  - Dịch vụ tập tin: user được gán quyền truy cập server, thư mục, tập tin, quản lý hệ thống tập tin, quyền truy cập
  - Bảo vệ an toàn dữ liệu: kiểm tra ghi dữ liệu, tạo mirror, duplicate thư mục, FAT, HotFix, TTS (Transaction Tracking System), giám sát SFT (System Fault Tolerance)
  - Định hướng: server có thể có nhiều card giao tiếp mạng (NIC-Netware Interface Connector). Netware hỗ trợ định hướng thông tin giữa những người sử dụng trên mạng

Ng Duc Thuan

Chương 2: Giới thiệu 1 số Hệ điều hành

- Đặc điểm của HĐH Netware 4.1 (tt)
  - Thông điệp: cung cấp hệ thống quản trị thông điệp
  - Quản trị: cung cấp nhiều tiện ích quản trị, giám sát và tối ưu
  - Dịch vụ in ấn: cho phép chia xẻ tập tin, in ấn trên mạng
  - Internet: hỗ trợ WAN và mobile device
  - Cấu hình năng động: sử dụng bộ nhớ tạo cache cho thư mục, buffer định hướng, chỉ mục Turbo FAT, TTS
  - Quản trị bộ nhớ hiệu quả
  - Hệ thống tập tin: tìm kiếm theo cơ chế thang máy, cache file, ghi background, Turbo FAT, nén file, kích thước file đến 4GB, 2 triệu thư mục/volume, mở cùng lúc 10,000 files…

Ng Duc Thuan

Chương 2: Giới thiệu 1 số Hệ điều hành

- Hệ điều hành Unix
  - Sản phẩm HĐH 32 bits của Bell Lab dành cho máy trạm, máy mini, notebook và các hệ máy đặc biệt
  - Cung cấp môi trường tốt cho phát triển ứng dụng (xử lý văn bản, xử lý tính toán, CASE/CAD/CAM)
  - Dễ chuyển đổi giữa các hệ thống, kết nối các chương trình. Từ khi IBMPC ra đời, Unix được chuyển từ máy mainframe, mini sang PC với hệ Microsoft's XENIX, AT&T System V.
  - Multitasking, multi-user, Unix là HĐH được sử dụng rộng rãi trong các chiến lược mở rộng hệ thống
  - Unix truyền thống có cơ chế dòng lệnh. Ngày nay Unix đã có các HĐH giao diện đồ họa thân thiện hơn.

Ng Đức Thuan

Chương 2: Giới thiệu 1 số Hệ điều hành

- Lịch sử phát triển của HĐH Unix
  - 1965-1969: Bell Lab phát triển hệ Multics trên mainframe
  - 1969: Ken Thompson thiết kế lại cho máy DEC PDP-7
  - 1970: Unix ra đời cho hệ máy PDP-11. Chưa hỗ trợ multitasking, multiprogramming. 24 (16+8) KB bộ nhớ. File tối đa 64KB. Soạn thảo văn bản. Chưa hỗ trợ bảo vệ lưu trữ.
  - 1973: Denis Ritchie viết lại Unix bằng C
  - 1975: Unix mã nguồn mở phổ biến trong các trường đại học
  - 1979: Unix với hệ thống chia xẻ thời gian ra đời. File tối đa 1 tỷ byte. Ngôn ngữ C mở rộng. Bổ sung nhiều shell mạnh hỗ trợ chuỗi, lập trình cấu trúc, chuyển file giữa các máy.

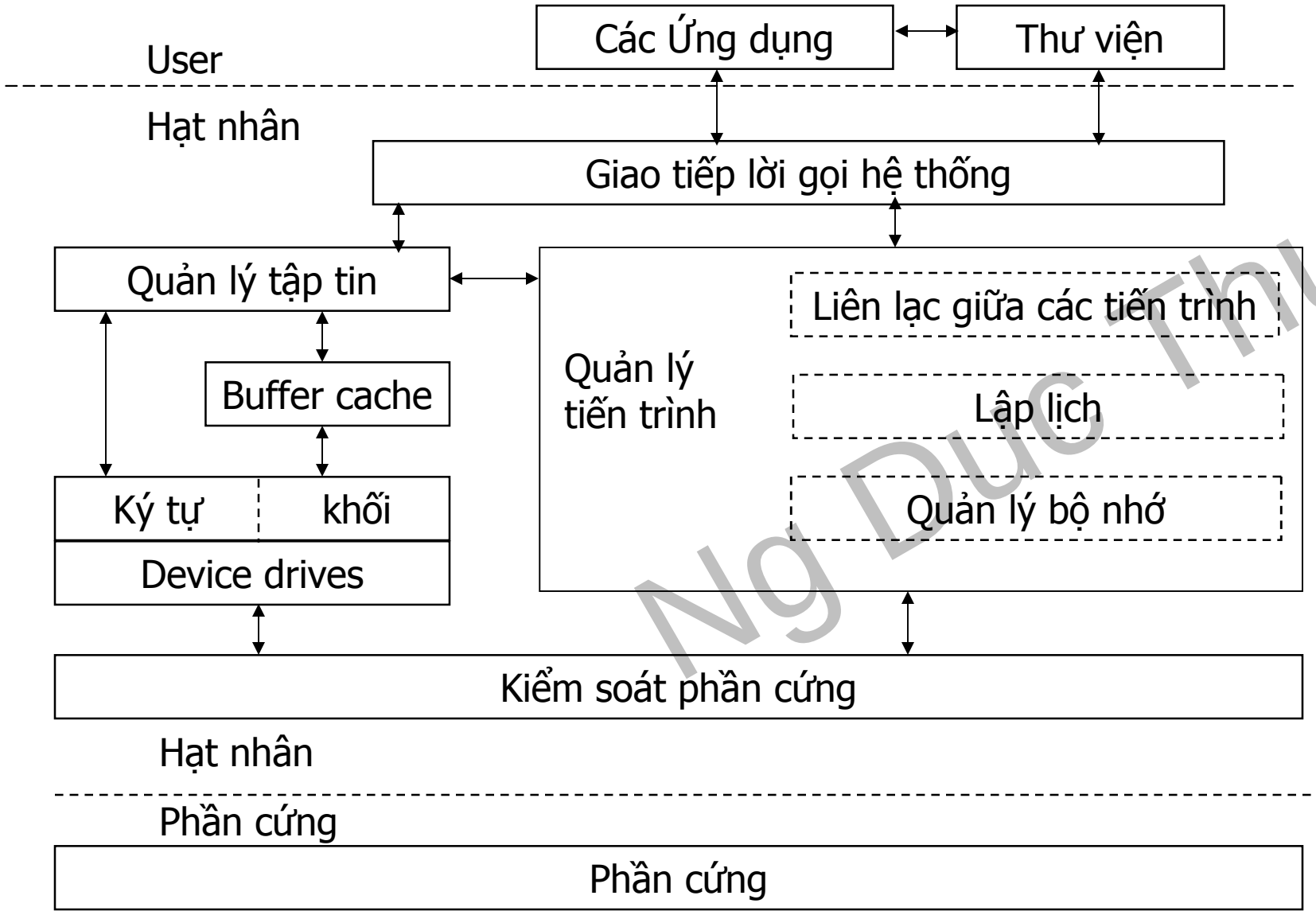
Ng Duc Thuan

Chương 2: Giới thiệu 1 số Hệ điều hành

- Lịch sử phát triển của HĐH Unix (tt)
  - 1980: XENIX, hệ Unix của Microsoft cho bộ xử lý 16 bits. Có đặc tính sửa chữa khôi phục tập tin tự động, chia sẻ dữ liệu, tăng cường xử lý bên trong
  - Unix của đại học Berkeley: HĐH chia sẻ thời gian, hỗ trợ không gian địa chỉ lớn hơn. Bộ nhớ ảo phân trang, hệ thống tập tin nhanh và mạnh hơn. Xử lý thông tin nội tại. Hỗ trợ mạng cục bộ. Soạn thảo toàn màn hình và shell
  - 1982: Unix System III, V của AT&T hỗ trợ truy xuất công việc từ xa, điều khiển mã nguồn
  - Có rất nhiều công ty cùng tham gia xây dựng Unix. Ngày nay Unix chiếm khoảng 20% trên các hệ thống máy tính

Ng Đức Thuan

Hạt nhân UNIX



Ng Duc Thuan

Chương 2: Giới thiệu 1 số Hệ điều hành

- Cấu trúc HĐH Unix: user, hạt nhân và phần cứng
  - Các phần chính của nhân: Quản lý tập tin, quản lý tiến trình
  - Các thư viện liên kết với lời gọi hệ thống để truy cập HĐH
  - Các nhóm Lời gọi hệ thống: nhóm tập tin, nhóm tiến trình
  - Hệ thống quản lý bộ nhớ đồng bộ hóa tiến trình, liên lạc giữa các tiến trình, cấp phát bộ nhớ, lập lịch tiến trình
  - Hệ thống quản lý tập tin và hệ thống quản lý tiến trình cùng tương tác khi nạp tập tin vào bộ nhớ
- Unix Shell, chương trình giao tiếp của user với Unix
  - Unix có nhiều loại shell (Bourne:sh, Korn: ksh, C: csh)
  - Mỗi shell có dấu nhắc khác nhau ($, %…)
  - Các shell có thể thi hành cùng lúc, 1 shell trên 1 shell khác…

Ng Đức Thuan

Chương 2: Giới thiệu 1 số Hệ điều hành

- Tập lệnh Unix
  - Các lệnh cơ bản
  - Các lệnh liên quan đến tập tin, thư mục
  - Các lệnh soạn thảo
  - Các lệnh tiện ích
  - Các lệnh về mạng
  - Các lệnh chuyển tập tin
  - Các lệnh thông tin liên lạc

Ng Duc Thuan

Chương 3: Quản lý tiến trình

2.1 Khái niệm

- Tiến trình là một chương trình đang xử lý, sở hữu một con trỏ lệnh, tập thanh ghi và các biến.
- Chương trình là thực thể thụ động chứa đựng các tín hiệu điều khiển máy tính để thực hiện tác vụ nào đó.
- HĐH hỗ trợ đa chương, đa nhiệm. Trong HĐH có nhiều tiến trình cùng hoạt động. Vì vậy việc sử dụng thuật toán để điều phối các tiến trình là nhiệm vụ của HĐH.
- Bộ phận thực hiện chức năng này gọi là bộ điều phối.

Ng Duc Thuan

Chương 3: Quản lý tiến trình

Trạng thái của một tiến trình:

- Running: *các chỉ thị tiến trình đang được xử lý.*
- Blocked: *tiến trình chờ cấp phát tài nguyên, hay sự kiện nào đó xảy ra.*
- Ready: *tiến trình chờ cấp phát CPU.*

(Create: tạo mới, Destroy: kết thúc)

Chế độ xử lý tiến trình:

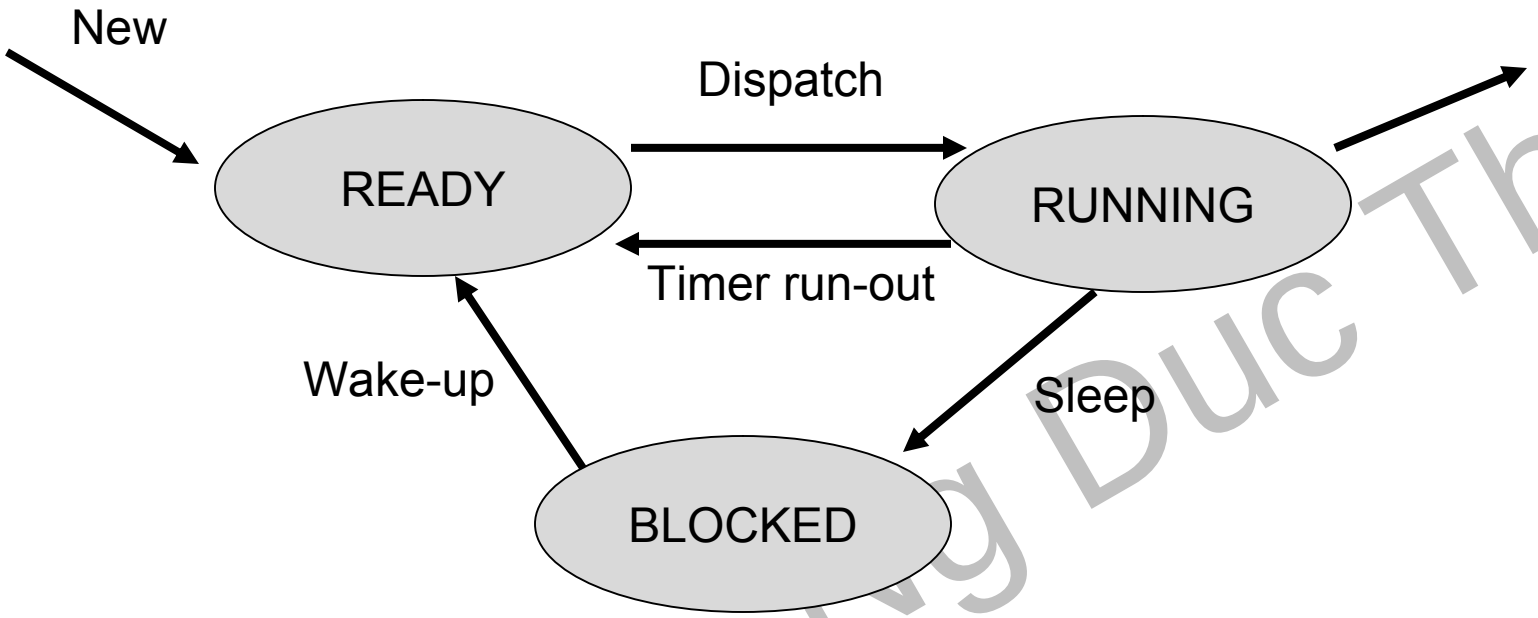
HĐH có hai chế độ xử lý tiến trình:

Chế độ đặc quyền: chỉ có HĐH mới hoạt động được với chế độ độc quyền, nhờ sự giúp đỡ phần cứng.

Chế độ không đặc quyền: người sử dụng

Ng Duc Thuan

Chương 3: Quản lý tiến trình
Chuyển trạng thái tiến trình



Ng Duc Thuan

Chương 3: Quản lý tiến trình

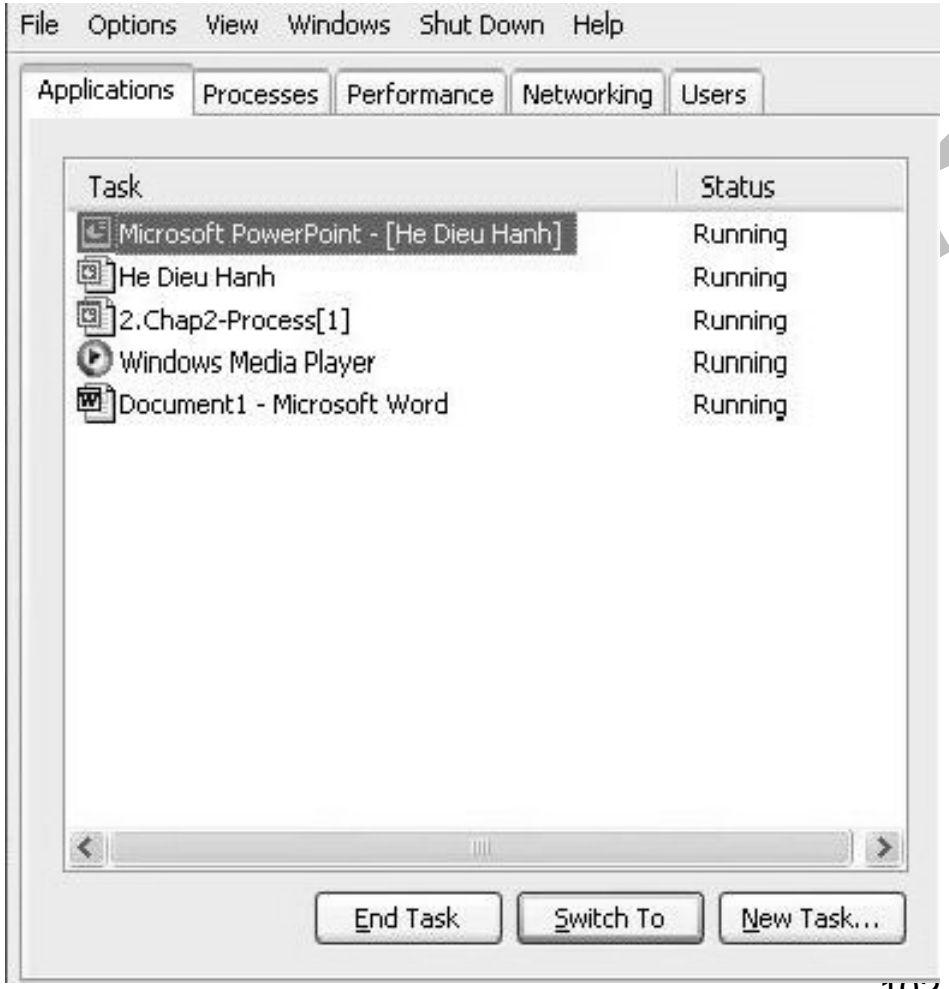
- HĐH quản lý thông tin tiến trình qua khối điều khiển (PCB: Process Control Block). Cấu trúc dữ liệu của HĐH để quản lý quá trình
- Chứa thông tin nhận dạng, trạng thái, định vị tài nguyên cho quá trình bao gồm thông tin sau:
  - ✓ Danh định cho quá trình (PID)
  - ✓ Bộ đếm chương trình
  - ✓ Vùng lưu giá trị thanh ghi CPU
  - ✓ Độ ưu tiên của quá trình
  - ✓ Thông tin định vị bộ nhớ quá trình
  - ✓ Thông tin bảo mật
  - ✓ Con trỏ đến các quá trình cha, con
  - ✓ .....

Ng Duc Thuan

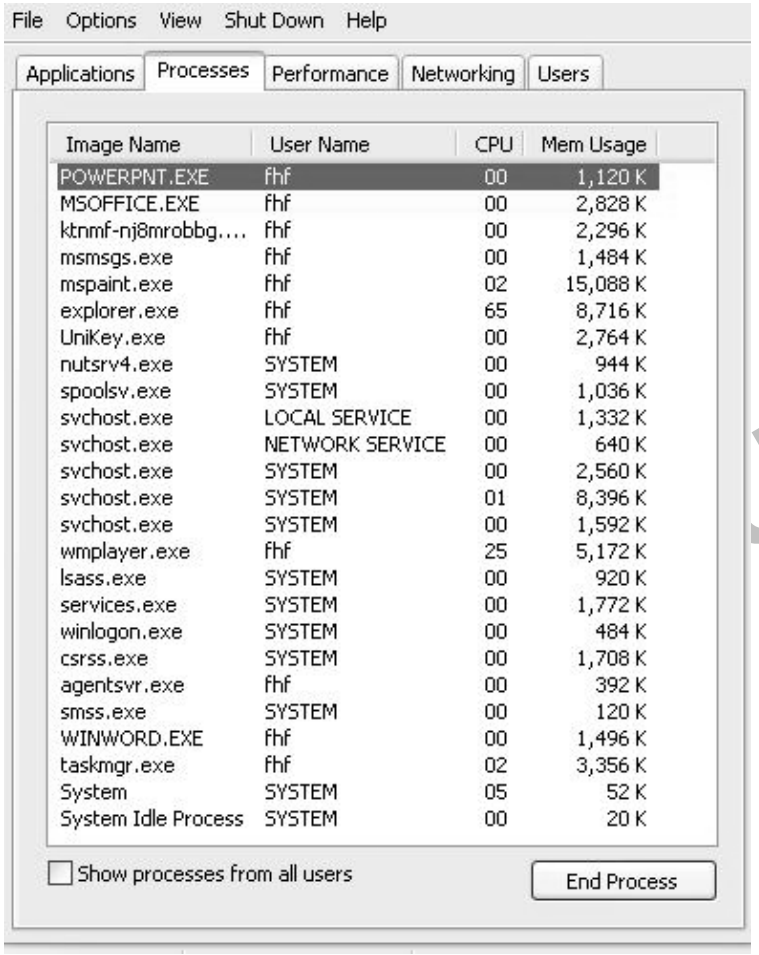
Chương 3: Quản lý tiến trình

Ví dụ

Trong Task Manager



Chương 3: Quản lý tiến trình



uc Thuan

Chương 3: Quản lý tiến trình

Thao tác trên tiến trình

HĐH cung cấp các thao tác chủ yếu cho tiến trình là:

☞ Create: tạo lập tiến trình
- ✓ *Định danh tiến trình*
- ✓ *Đưa tiến trình vào danh sách quản lý*
- ✓ *Xác định mức độ ưu tiên*
- ✓ *Cung cấp tài nguyên ban đầu*
- ✓ *Tạo khối điều khiển tiến trình (PCB)*

☞ Destroy: kết thúc tiến trình
- ✓ *Thu hồi tài nguyên đã cấp*
- ✓ *Huỷ bỏ tiến trình khỏi danh sách quản lý*
- ✓ *Huỷ bỏ khối điều khiển tiến trình*

(Lưu ý: tiến trình con không thể tiếp tục khi tiến trình cha kết thúc)

Ng Duc Thuan

Chương 3: Quản lý tiến trình

2.2 Tiểu trình (threads)

- Mỗi tiến trình thông thường có một không gian địa chỉ và dòng xử lý.
- Trong trường hợp người sử dụng muốn nhiều dòng xử lý cùng chia sẻ một không gian địa chỉ và các dòng dữ liệu xử lý song song như các tiến trình riêng biệt. HĐH cung cấp cơ chế như vậy gọi là tiểu trình.
- Một tiểu trình là một đơn vị xử lý cơ bản trong hệ thống.
- Một tiểu trình có thể tạo nhiều tiến trình con.
- Một tiến trình có thể sở hữu nhiều tiểu trình.

Ng Duc Thuan

Chương 3: Quản lý tiến trình

2.3 Điều phối tiến trình

Trong môi trường đa chương nhiều tiến trình đồng thời sẵn sàng nhận xử lý. Tiến trình tiếp theo nào được chọn để xử lý cần có giải thuật thích hợp để thực hiện nhiệm vụ này. HĐH có bộ phận thực hiện nhiệm vụ này được gọi bộ điều phối tiến trình.(dispatcher)

1. Mục tiêu điều phối là:

- ✡ Sự công bằng(Fairness)
- ✡ Tính hiệu quả (Efficiency)
- ✡ Thời gian đáp ứng hợp lý(Response time)
- ✡ Thời gian lưu lại trong hệ thống (Turaround Time).
- ✡ Thông lượng tối đa (throughput)

Ng Duc Thuan

Chương 3: Quản lý tiến trình

2. Các đặc điểm của tiến trình

Mục tiêu cơ bản trong điều phối tiến trình:

- Hướng xuất/nhập của tiến trình
- Hướng xử lý của tiến trình
- Chương trình tương tác hay xử lý theo lô
- Độ ưu tiên của tiến trình
- Thời gian đã sử dụng CPU của tiến trình.
- Thời gian còn lại của tiến trình cần giải quyết

Điều phối độc quyền và không độc quyền

Điều phối độc quyền: Tiến trình chiếm CPU đến khi hoàn tất hoặc tự nguyện giải phóng.

Điều phối không độc quyền: Tiến trình có thể bị tạm dừng bất cứ khi nào.

Ng Duc Thuan

Chương 3: Quản lý tiến trình

3. Điều phối: HĐH sử dụng 2 loại danh sách trong điều phối:

- Danh sách sẵn sàng
- Danh sách hàng đợi

Danh sách sẵn sàng: Chứa những chương trình thường trú trong bộ nhớ chính. Ở trạng thái săn sàng

Danh sách hàng đợi: Chứa những tiến trình trạng thái Ready.

Các cấp độ điều phối:

- *Điều phối tác vụ*: quyết định số tiến trình vào bộ nhớ
- *Điều phối tiến trình*: quyết định tiến trình nào tiếp theo nhận CPU

Ng Đức Thuan

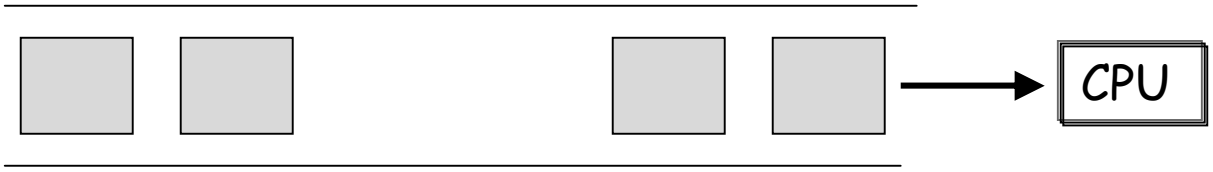
Chương 3: Quản lý tiến trình

4. Các chiến lược điều phối

A. Chiến lược FIFO(First In First Out)

Nguyên tắc: CPU sẽ được cấp phát cho tiến trình đầu tiên trong danh sách sẵn sàng. Đây thuật toán điều phối theo nguyên tắc độc quyền. CPU chỉ có thể thoát khỏi tiến trình khi kết thúc hay có yêu cầu xuất / nhập.

Danh sách sẵn sàng



Ng Duc Thuan

Chương 3: Quản lý tiến trình

Ví dụ: chiến lược điều phối FIFO

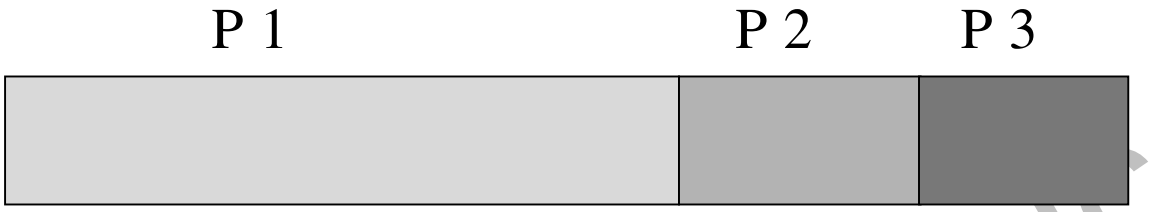
Tiến trình	Thời điểm vào	Thời gian xử lý
P1	0	24
P2	1	3
P3	2	3

Ng Duc Thuan

Chương 3: Quản lý tiến trình

Ví dụ: chiến lược điều phối FIFO

Thứ tự cấp phát:



Thời gian chờ: P1 là 0

P2 là 24

P3 là 27

Thời gian chờ trung bình:(0+24+27)/3=17ms

Ng Duc Thuan

Chương 3: Quản lý tiến trình

Nhận xét: chiến lược điều phối FIFO

- Thời gian chờ không đạt cực tiểu. Có thể xảy hiện tượng tích lũy thời gian chờ.
- Thay đổi đáng kể thời gian chờ nếu thay đổi thứ tự dãy.
- Giải thuật không phù hợp với việc phân chia thời gian

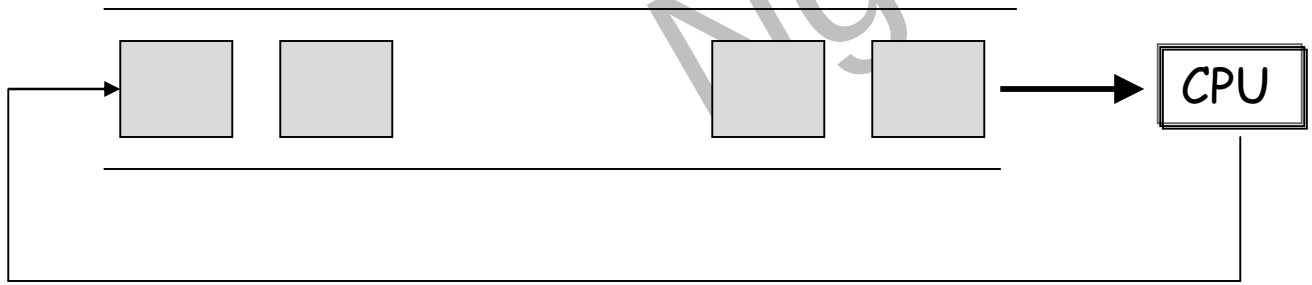
Ng Duc Thuan

Chương 3: Quản lý tiến trình

B. Chiến lược phân phối vòng quay(Round Robin)

Nguyên tắc: Danh sách sẵn sàng xử lý là danh sách vòng, bộ điều phối lần lượt cấp phát từng tiến trình trong danh sách khoảng thời gian t.(Giải thuật điều phối không độc quyền)

Danh sách sẵn sàng



Ng Duc Thuan

Chương 3: Quản lý tiến trình

Ví dụ: chiến lược điều phối RR

Tiến trình	Thời điểm vào	Thời gian xử lý
P1	0	24
P2	1	3
P3	2	3

Ng Duc Thuan

Chương 3: Quản lý tiến trình

Ví dụ: chiến lược điều phối RR

Thời gian chờ trung bình:(0+4+7+6)/3=5,66ms

Nhận xét:

☞ Giải thuật phụ thuộc vào rất nhiều giá trị t

☞ t quá nhỏ -> sử dụng CPU kém hiệu quả

☞ t quá lớn -> tăng khả năng hồi đáp, giám khả năng tương tác hệ thống.

Ng Duc Thuan

Chương 3: Quản lý tiến trình

C. Chiến lược Điều phối với độ ưu tiên

Nguyên tắc: Mỗi tiến trình khi vào hệ thống được gán giá trị ưu tiên. Tiến trình có độ ưu tiên cao sẽ được nhận CPU trước. Độ ưu tiên dựa trên rất nhiều tiêu chí như: thời gian xử lý, yêu cầu bộ nhớ.

- Giải thuật này có thể theo nguyên tắc độc quyền và không độc quyền

Ng Đức Thuan

Chương 3: Quản lý tiến trình

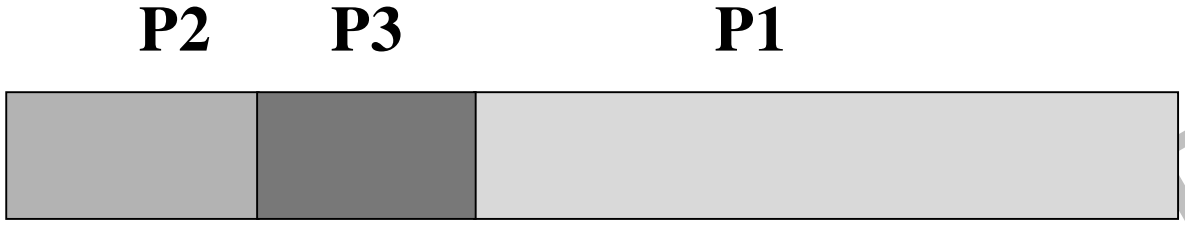
Ví dụ: chiến lược điều phối Ưu tiên

Tiến trình	Độ ưu tiên	Thời gian xử lý
P1	3	24
P2	1	3
P3	2	3

Ng Duc Thuan

Chương 3: Quản lý tiến trình

Ví dụ: chiến lược điều phối Ưu tiên



Thời gian chờ trung bình:(0+3+6)/3=2ms

Nhận xét:

- Có thể dẫn đến tiến trình có mức ưu tiên cấp thấp chờ vô hạn.
- Khắc phục bộ điều phối tiến hành giảm độ ưu tiên xuống sau mỗi ngắt đồng hồ cho đến khi tiến trình có mức ưu tiên thấp hơn tiến trình kế cận và chuyển CPU.

Ng Duc Thuan

Chương 3: Quản lý tiến trình

D. Chiến lược công việc ngắn nhất

(SJF: Shortest Job First)

Nguyên tắc: Điều phối với độ ưu tiên, độ ưu tiên p được gán cho tiến trình là nghịch đảo của thời gian xử lý t mà tiến trình yêu cầu: p=1/t. Tiến trình kế tiếp là tiến trình yêu cầu ít thời gian nhất để kết thúc.

☞ Giải thuật này cũng có thể là độc quyền hay không độc quyền

Ng Đức Thuan

Chương 3: Quản lý tiến trình

Ví dụ: chiến lược điều phối SJF

Tiến trình	Thời gian xử lý
P1	6
P2	8
P3	7
P4	3

Ng Duc Thuan

Chương 3: Quản lý tiến trình

Ví dụ: chiến lược điều phối SJF

Thứ tự cấp phát:

P4 **P1** **P3** **P2**

Thời gian chờ trung bình:(3+9+16+0)/4=7ms

Nhận xét:

- Cho phép đạt thời gian chờ trung bình là cực tiểu
- Xác định thời gian yêu cầu xử lý còn lại là rất khó khăn:

Biểu thức dự đoán $\tau_{n+1} = \alpha t_n + (1-\alpha)\tau_n$

Ng Duc Thuan

Chương 3: Quản lý tiến trình

F. Chiến lược điều phối nhiều mức độ ưu tiên

Nguyên tắc: Phân lớp tiến trình tuỳ theo độ ưu tiên của chúng để có cách điều phối thích hợp cho từng nhóm. Danh sách sẵn sàng được phân tách thành các danh sách riêng biệt theo cấp độ ưu tiên. Mỗi danh sách bao gồm các tiến trình có cùng mức độ ưu tiên và được áp dụng giải thuật điều phối thích hợp để điều phối.

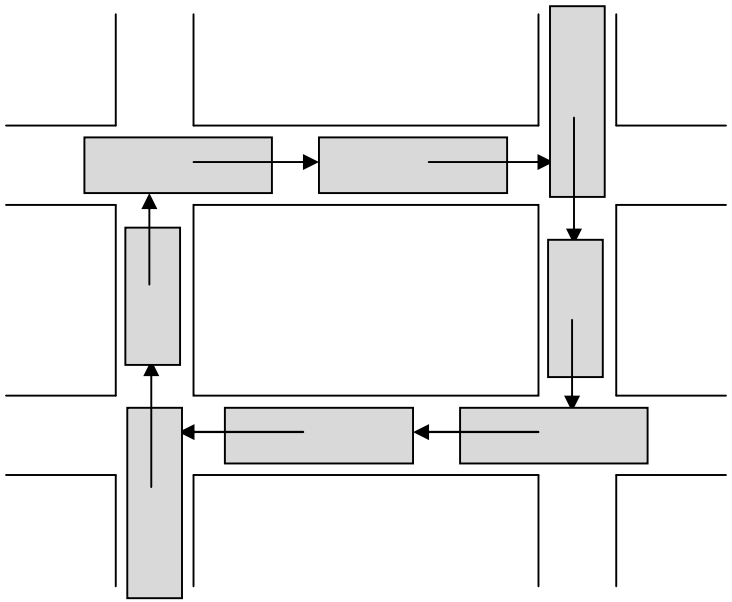
Giải thuật sử dụng là giải thuật không độc quyền.

Ng Đức Thuan

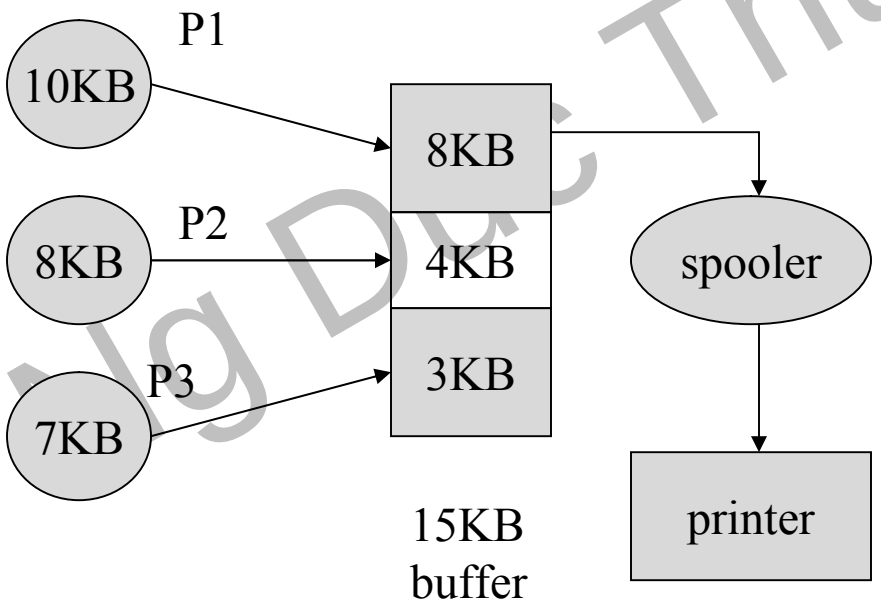
Chương 3: Quản lý tiến trình

Tiến trình deadlock : đợi một sự kiện không bao giờ xảy ra.

Một hệ thống bị deadlock : có tiến trình bị deadlock.



Tắc nghẽn trong giao thông



Tắc nghẽn trong quản lý in ấn

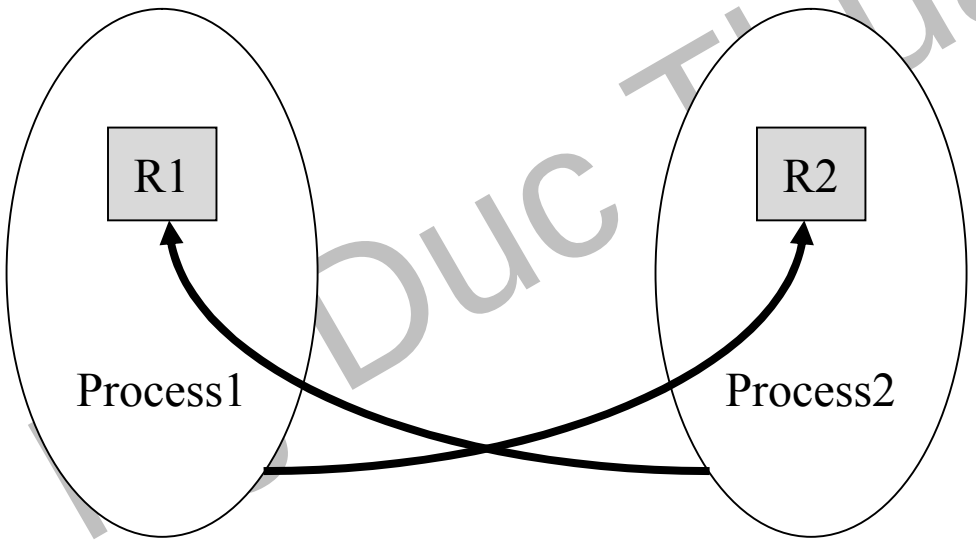
Vũ Đức Thuận

Chương 3: Quản lý tiến trình

Hai tiến trình bị deadlock:

Process 1	Process2
P(S1);	P(S2);
P(S2);	P(S1);
Critical Section;	*Critical Section;*
V(S1);	V(S2);
V(S2);	V(S1);

Dạng deadlock:



Tô Đức Thuận

Chương 3: Quản lý tiến trình

ĐIỀU KIỆN XẢY RA DEADLOCK

Điều kiện **mutual exclusion**: các tiến trình cần thực hiện loại trừ tương hỗ trên vùng tranh chấp

Điều kiện **hold & wait**: tiến trình đang giữ tài nguyên có thể yêu cầu thêm tài nguyên khác

Điều kiện **no-preemption**: tài nguyên chỉ được giải phóng khi tiến trình dùng xong

Điều kiện **circular-wait**: các tiến trình giữ và đợi tài nguyên tạo thành vòng luẩn quẩn

Ng Đức Thuan

Chương 3: Quản lý tiến trình

GIẢI QUYẾT DEADLOCK

Ngăn ngừa deadlock ***(deadlock prevention)***

- Qui định cấp , dùng tài nguyên nghiêm ngặt
- Không cho các điều kiện deadlock xảy ra

Tránh deadlock ***(deadlock avoidance)***

- Vẫn cho các điều kiện deadlock tồn tại
- Cấp tài nguyên hợp lý, an toàn

Phát hiện deadlock ***(deadlock detection)***

Phục hồi deadlock ***(deadlock recovery)***

Ng Duc Thuan

Chương 3: Quản lý tiến trình

Cấm điều kiện **multual-exclusion** ?

Cấm điều kiện **hold & wait**

- Tiến trình yêu cầu tất cả tài nguyên một lần
- Chỉ được xử lý khi đã đủ tất cả tài nguyên cần thiết

Cấm điều kiện **no-preemption**

- Nếu yêu cầu tài nguyên không được, tiến trình phải giải phóng tất cả tài nguyên đang giữ và yêu cầu lại.

Loại bỏ **circular-wait**

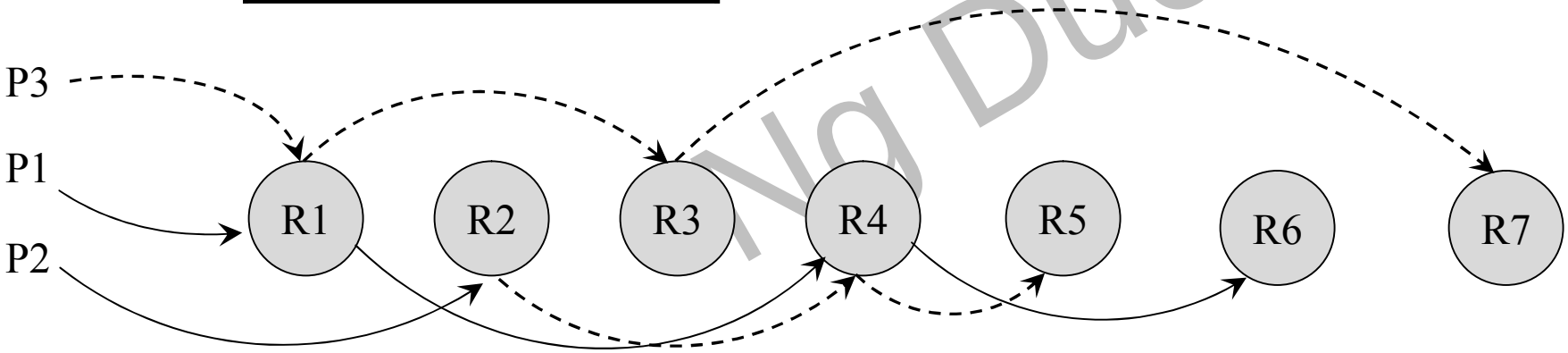
- Sắp xếp tài nguyên theo trật tự và chung cấp cho tiến trình theo đúng trật tự đó

Ng Duc Thuan

Chương 3: Quản lý tiến trình

Ví dụ

Tiến trình	Yêu cầu thực tế
P1	R6, R4, R1
P2	R2, R5, R4
P3	R3, R7, R1



Ng Duc Thuan

Chương 3: Quản lý tiến trình

Giải thuật nhà băng (Banker's Algorithm)

- Hệ điều hành ~ Nhà Băng
- Tiến trình ~ Khách hàng
- Tài nguyên ~ Vốn vay

Ràng buộc

- Yêu cầu vay cực đại ≤ vốn nhà băng
- Khách không trả vốn nếu vay chưa đủ yêu cầu cực đại
- Khi vay đủ, khách phải trả đủ vốn sau thời gian hữu hạn

Trạng thái nhà băng

- An toàn (**Safe**): thỏa yêu cầu mọi khách, ngân hàng thu vốn đủ
- Không an toàn (**Unsafe**) : ngược lại ➔ có thể deadlock

Hệ thống phải cấp phát tài nguyên sao cho không rơi vào trạng thái Unsafe

Ng Duc Thuan

Chương 3: Quản lý tiến trình

Trạng thái sau là an toàn. Tại sao ?

	Đang mượn	Cần tối đa	Cần thêm
P1	1	4	3
P2	4	6	2
P3	5	8	3
Vôn 12 , còn lại 2			

Trạng thái sau là không an toàn. Tại sao ?

	Đang mượn	Cần tối đa	Cần thêm
P1	8	10	2
P2	2	5	3
P3	1	3	2
Vôn 12 , còn lại 1			

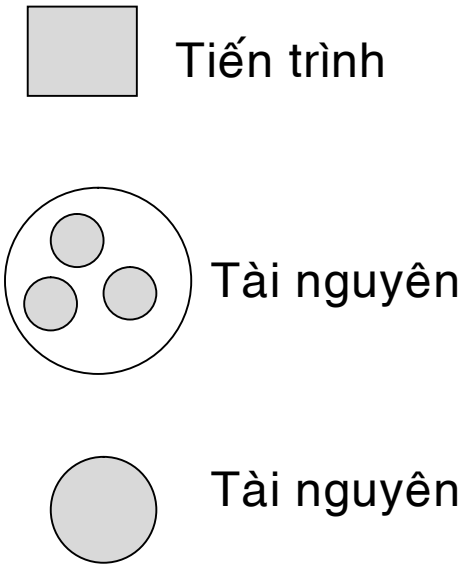
Ng Duc Thuan

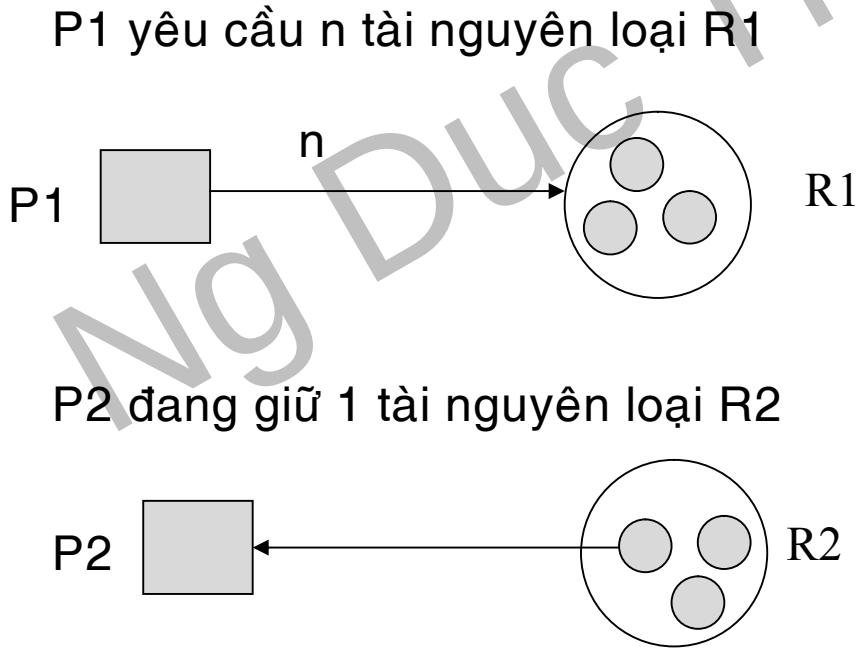
Chương 3: Quản lý tiến trình

PHÁT HIỆN DEADLOCK

Ghi nhận, theo dõi yêu cầu, sự cấp phát tài nguyên cho các tiến trình.

Dùng đồ thị RAG (*Resource Allocation Graph*)





Ng Duc Thuan

Chương 3: Quản lý tiến trình

Giản lược RAG

1. Tài nguyên rảnh → cấp cho tiến trình yêu cầu
2. Tiến trình đủ tài nguyên
   → xoá mọi cạnh vào, xoá tiến trình
3. Lặp lại 1 với các tiến trình khác đến khi tối giản

Khi giải thuật dừng

– RAG không còn cạnh: không có deadlock
– RAG có chu trình (cycle): deadlock

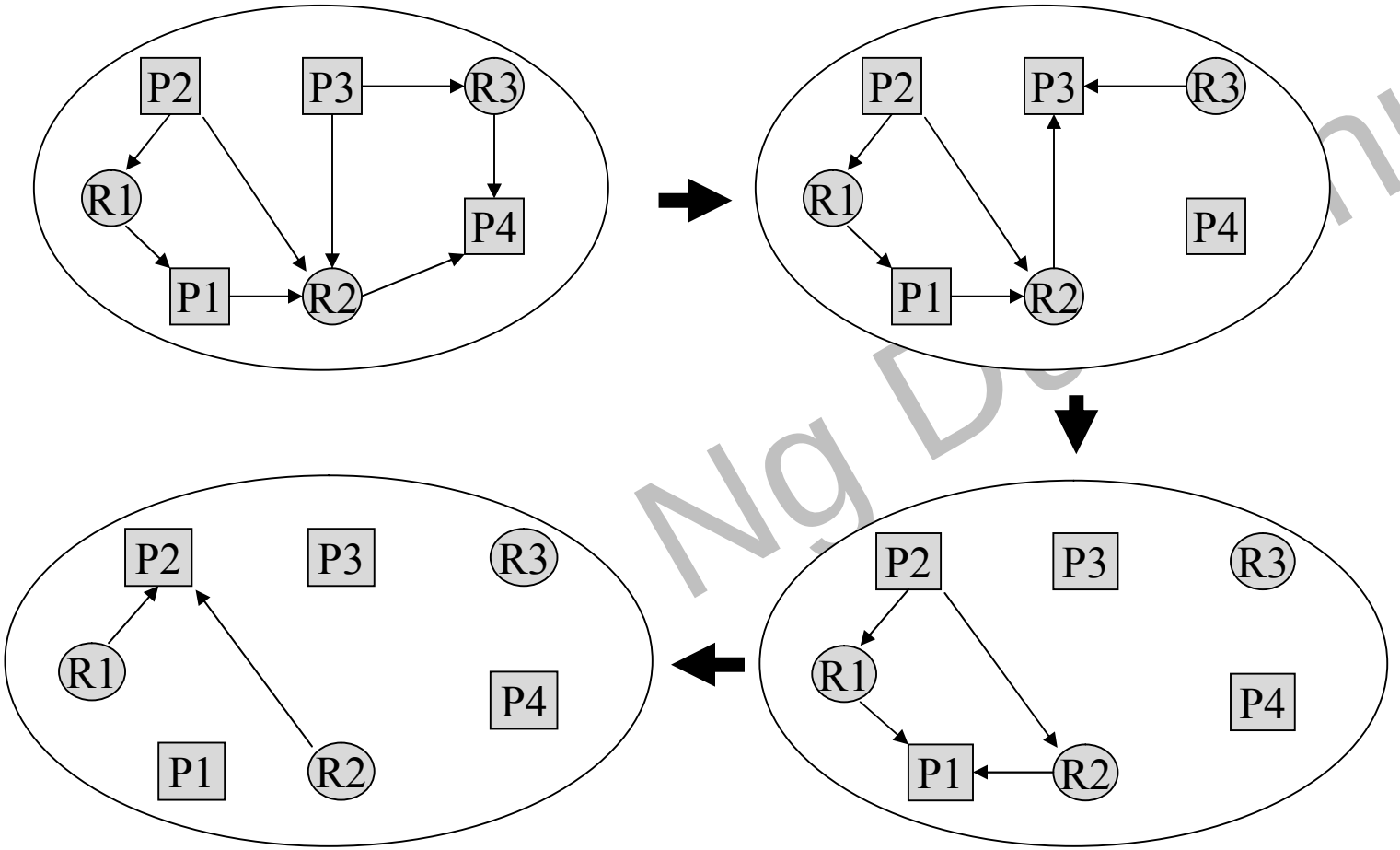
Nhận xét

– Phí tổn lớn

Ng Duc Thuan

Chương 3: Quản lý tiến trình

VÍ DỤ 1 : GIẢN ƯỚC RAG



Ng Đ huan

Chương 4: Hệ thống quản lý File

4.1 TỔNG QUAN VỀ DỮ LIỆU & FILE

Yêu cầu lưu trữ dữ liệu của người dùng:

- ✓ Lưu trữ lâu dài
- ✓ Truy cập nhanh
- ✓ Lưu được nhiều dữ liệu
- ✓ Chia sẻ và bảo vệ tốt
- ✓ Dễ sử dụng

Cần sự hỗ trợ của phần cứng và OS để dữ liệu lưu trữ ra bộ nhớ ngoài.

Tập tin và thư mục

- ✓ *Tập tin:* đơn vị lưu trữ thông tin do HĐH, các ứng dụng, người dùng quản lý.
- ✓ *Thư mục:* cấu trúc tổ chức tập tin trên đĩa.

Ng Duc Thuan

Chương 4: Hệ thống quản lý File

4.2 MÔ HÌNH QLÝ VÀ TỔ CHỨC TẬP TIN

Tên tập tin:

- ✓ Tạo bởi HĐH, tiến trình hoặc user
- ✓ Cách đặt tên tập tin của mỗi HĐH là khác nhau: Số ký tự đặt tên (chữ cái, chữ số, ký tự đặc biệt), phân biệt/không phân biệt chữ hoa/thường.
- ✓ Nhiều hệ tạo tên file nhiều cụm, ngăn cách bởi dấu chấm '.'

Cấu trúc tập tin, gồm 3 loại:

- ✓ Dãy tuần tự các byte không cấu trúc.
- ✓ Dãy các record có chiều dài cố định.
- ✓ Cấu trúc cây (mỗi record có trường khóa giúp tìm kiếm nhanh).

Ng Duc Thuan

Chương 4: Hệ thống quản lý File

4.2 MÔ HÌNH QLÝ VÀ TỔ CHỨC TẬP TIN

Kiểu tập tin

✓*Tập tin thường*: tập tin text/nhị phân chứa thông tin của người dùng.

✓*Thư mục:* (tập tin) lưu giữ cấu trúc hệ thống tập tin.

✓*Tập tin có ký tự đặc biệt*: liên quan đến nhập xuất (màn hình, máy in, mạng)

✓*Tập tin khối*: truy xuất thiết bị khối dạng đĩa

Các loại tập tin

Tập tin văn bản: tập các ký tự

Tập tin nhị phân: gồm dãy các byte tổ chức theo định dạng xác định (vd: header, text, data, relocation bits, symbol table)

Ng Duc Thuan

Chương 4: Hệ thống quản lý File

4.2 MÔ HÌNH QLÝ VÀ TỔ CHỨC TẬP TIN

Truy xuất tập tin:

- *Truy xuất tuần tự:* thích hợp cho các thiết bị lưu trữ tuần tự (băng từ).
- *Truy xuất ngẫu nhiên:* thích hợp cho các thiết bị lưu trữ ngẫu nhiên (đĩa từ, CD-ROM...)

Thuộc tính tập tin: Bảo vệ, Mật khẩu, Người tạo, Người sở hữu, Chỉ đọc, Ẩn, Hệ thống, Lưu trữ, ASCII/Binary, Truy xuất ngẫu nhiên/Tuần tự, Temp, Khóa (lock), Độ dài record, Vị trí khóa, Ngày giờ tạo, Thời điểm truy cập, Thời điểm thay đổi, Kích thước hiện tại, Kích thước tối đa.

Ng Duc Thuan

Chương 4: Hệ thống quản lý File

4.2 MÔ HÌNH QLÝ VÀ TỔ CHỨC TẬP TIN

Thư mục: hệ thống lưu trữ theo cấp bậc

- Thư mục chứa các đề mục tên + thuộc tính + địa chỉ tập tin.
- Khi có yêu cầu mở file, HĐH tìm trong thư mục tên file cần mở, xác định thuộc tính và địa chỉ file, đọc file vào bộ nhớ chính.
- Số lượng thư mục trên mỗi hệ thống là khác nhau. Để tiện cho việc truy xuất, sử dụng, lưu trữ, đa số các HĐH đều tổ chức thư mục theo cấu trúc phân cấp hình cây.

Đường dẫn: cách xác định địa chỉ tập tin

- ✓Thư mục hiện hành [.], thư mục cha [..]
- ✓Đường dẫn tuyệt đối: /usr/ast/mailbox
- ✓Đường dẫn tương đối: [.]/dir1/dir2

Ng Đức Thuận

Chương 4: Hệ thống quản lý File

4.2 MÔ HÌNH QLÝ VÀ TỔ CHỨC TẬP TIN

Các chức năng của hệ thống tập tin

- _*Tập tin*: Tạo, Xóa, Mở, Đọc, Ghi, Thêm, Đóng, Tìm, Lấy thuộc tính, Thiết lập thuộc tính, Đổi tên.
- _*Thư mục*: Tạo, Xóa, Mở thư mục (vd: mở trước khi liệt kê), Đóng thư mục, Đổi tên, Liên kết (file có thể xuất hiện trong nhiều thư mục khác nhau), Bỏ liên kết.

Ng Đức Thuan

Chương 4: Hệ thống quản lý File

CÁC PHƯƠNG PHÁP ĐỊNH VỊ BẢNG PHÂN PHỐI ĐĨA

Định vị liên tiếp:

- Dễ cài đặt, dễ thao tác
- Không linh động, đĩa bị phân mảnh

Định vị bằng danh sách liên kết

- Entry chỉ chứa địa chỉ đầu. Mọi khối đều được cấp phát, ít lãng phí.
- Khối dữ liệu bị thu hẹp. Truy xuất ngẫu nhiên chậm

Danh sách liên kết sử dụng index

- Tương tự như cách 2 nhưng thay con trỏ bằng bảng index
- Truy xuất ngẫu nhiên dễ dàng hơn
- Bị giới hạn bởi kích thước bộ nhớ

Ng Duc Thuan

Chương 4: Hệ thống quản lý File

QUẢN LÝ ĐĨA

Có 2 phương pháp lưu trữ:

✓Lưu tuần tự trên N byte liên tiếp: không hiệu quả khi file lớn

✓Lưu dữ liệu trên đĩa theo đơn vị khối

Kích thước khối thông thường là 512byte, 1 hoặc 2 KB

Lưu giữ các khối trống. Có 2 phương pháp

✓Sử dụng danh sách liên kết của khối đĩa

✓Sử dụng bitmap: đĩa N khối ánh xạ thành N bit (1:trống, 0: đã dùng). Đĩa 20M cần 20Kbit để lưu trữ = 2.5 ~3 khối

Ng Duc Thuan

Chương 4: Hệ thống quản lý File

Độ an toàn của hệ thống tập tin

Quản lý khối bị hỏng:

✓Giải pháp phần mềm: xây dựng tập tin chứa các khối bị hỏng

✓Giải pháp phần cứng: dùng sector trên đĩa lưu giữ danh sách khối hỏng

Backup:

✓Chép dự phòng bản sao thứ cấp dữ liệu (đĩa mềm, băng từ...)

✓Chia đĩa cứng làm 2 phần: dữ liệu hoạt động và backup.

Tính không đổi của hệ thống tập tin

✓Ngừng hệ thống đột ngột có thể gây mất dữ liệu

✓Hệ thống phải có cơ chế kiểm tra tính toàn vẹn dữ liệu trên 2 phần khối và tập tin

Ng Duc Thuan

Chương 4: Hệ thống quản lý File

Truy xuất hệ thống tập tin theo MSDOS

Nhập xuất theo thẻ file

Thẻ file (Handle file): là đối tượng mà MSDOS dùng để quản lý các file đang được mở trên bộ nhớ. Khi file được mở DOS sẽ gán cho file một số xác định. (AX, BX)

- ✓Tạo: 3C
- ✓Mở: 3D
- ✓Đóng: 3E
- ✓Hủy: 41
- ✓Đặt con trỏ: 42
- ✓Đọc từ tập tin/thiết bị: 3F
- ✓Ghi lên tập tin/thiết bị: 40
- ✓IOCTL: 44
- ✓Định hướng lại: 46

Ng Duc Thuan

Chương 4: Hệ thống quản lý File

Truy xuất hệ thống tập tin theo MSDOS

Các chức năng về tập tin và thư mục

- ✓ Tạo thư mục: 39
- ✓ Hủy thư mục: 3A
- ✓ Chuyển thư mục: 3B
- ✓ Đổi tên tập tin: 56
- ✓ Thiết lập ngày giờ: 57
- ✓ Hỏi kích thước file: 42
- ✓ Thay đổi thuộc tính: 43
- ✓ Tìm tập tin: 4E
- ✓ Tìm thư mục: 11

Ng Duc Thuan

Chương 4: Hệ thống quản lý File

Truy xuất hệ thống tập tin theo MSDOS

Các chức năng thi hành:

- ✓ Thi hành: 4B
- ✓ Tạp PSP: 26
- ✓ Lấy địa chỉ PSP: 62
- ✓ Kết thúc thường trú: Int 27
- ✓ Kết thúc chương trình: Int 20
- ✓ Kết thúc gửi mã thoát về tiến trình cha: 4C
- ✓ Lấy mã thoát của tiến trình kết thúc: 4D
- ✓ Kết thúc qua Ctr-Brk: Int 23

Ng Duc Thuan

Chương 4: Hệ thống quản lý File

Truy xuất hệ thống tập tin theo MSDOS

Mở tập tin mới

- ✓ AH=3Ch
- ✓ CL= (0: đọc; 1 ghi; 2 đọc/ghi)
- ✓ DS:DX: là địa chỉ xâu tên file (kết thúc 0).
- ✓ => AX chứa thẻ file

Đóng tập tin

- ✓ AH=3Eh
- ✓ BX= thẻ file

Đọc nội dung tập tin

- ✓ AH=3Fh CX= số byte cần đọc
- ✓ BX= thẻ file DS:DX: vùng đệm lưu

Ghi nội dung tập tin

- ✓ AH=40h CX= số byte cần ghi
- ✓ BX= thẻ file DS:DX: vùng đệm chứa dliệu

Ng Duc Thuan

Chương 4: Hệ thống quản lý File

RAID (*Redundant Array of Inexpensive Disks*)

Tập hợp các đĩa cứng được hệ điều hành xem như một thiết bị lưu trữ

Dữ liệu được phân bố trên tất cả các đĩa

Các mục tiêu chính

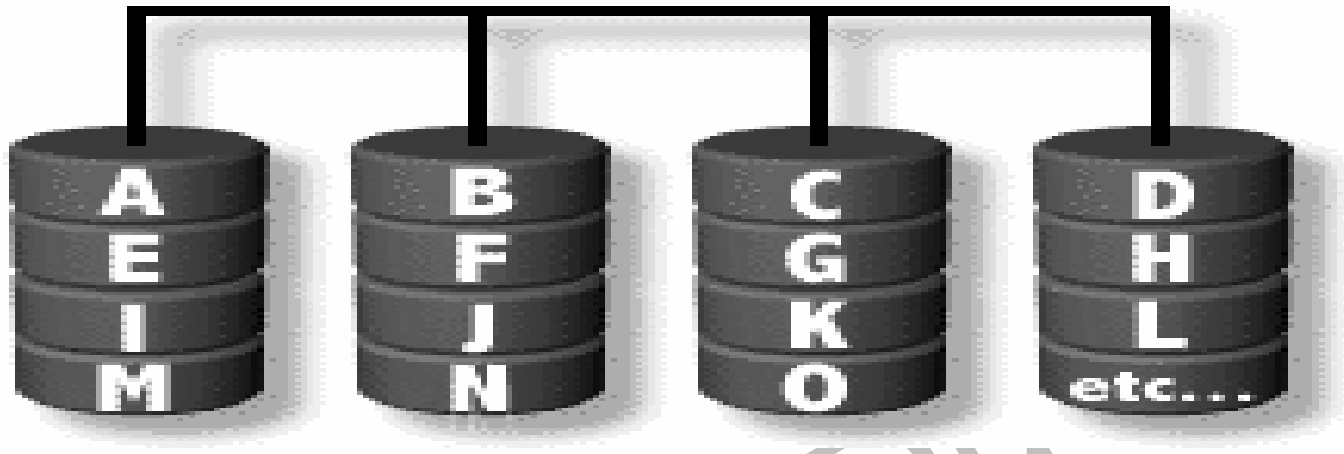
- ❖Tăng dung lượng lưu trữ
- ❖Tăng hiệu suất I/O
- ❖Tăng tính sẵn sàng cao
- ❖Tăng khả năng phục hồi hệ thống

Các loại RAID

- ❖RAID 0 → RAID 10 (phổ biến RAID 0, 1, 3, 5)
- ❖Software RAID/ Hardware RAID

Ng Duc Thuan

Chương 4: Hệ thống quản lý File

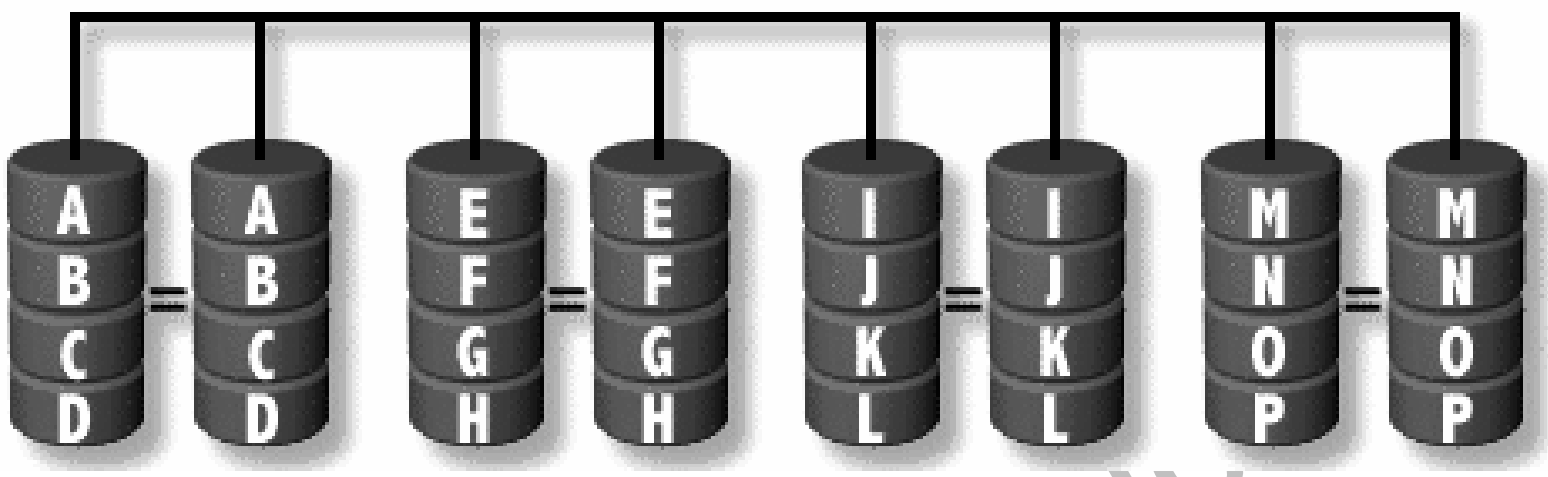


RAID-0

- ✓ Dữ liệu lưu trữ trải đều trên các đĩa
- ✓ Tăng không gian lưu trữ
- ✓ Tăng hiệu suất hệ thống
- ✓ Tính sẵn sàng của dữ liệu thấp

Ng Đ Xuan

Chương 4: Hệ thống quản lý File

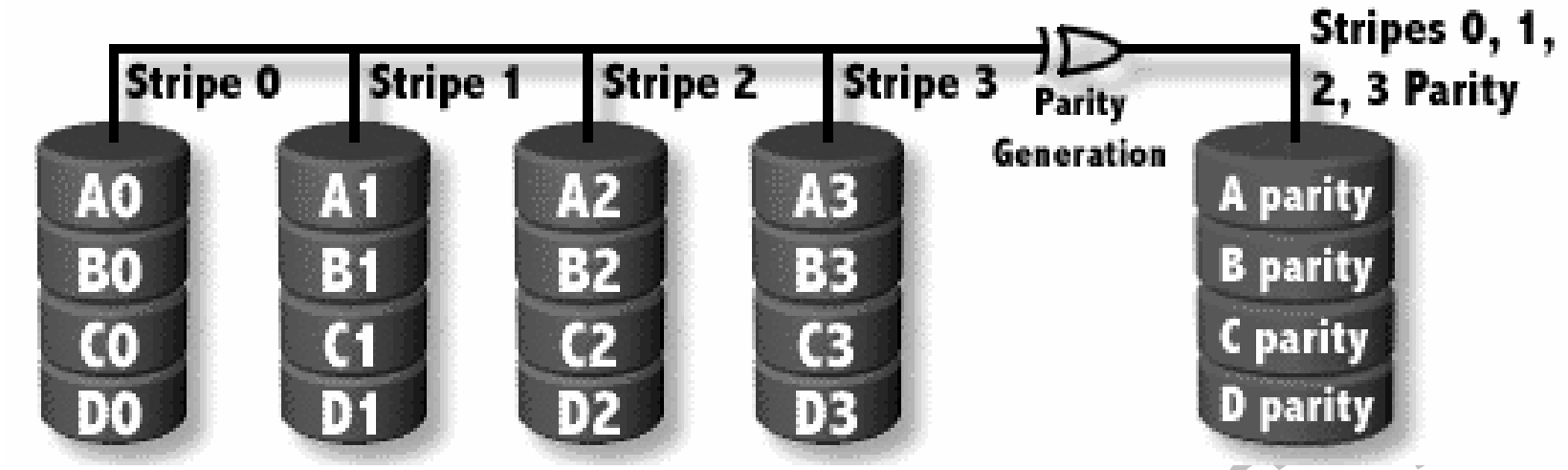


RAID-1

- ✓ Nhân bản dữ liệu trên các đĩa tách biệt
- ✓ Tính sẵn sàng & tốc độ đọc dữ liệu rất cao
- ✓ Yêu cầu dung lượng đĩa gấp đôi
- ✓ Tốc độ ghi chậm hơn

Ng Duc ...an

Chương 4: Hệ thống quản lý File

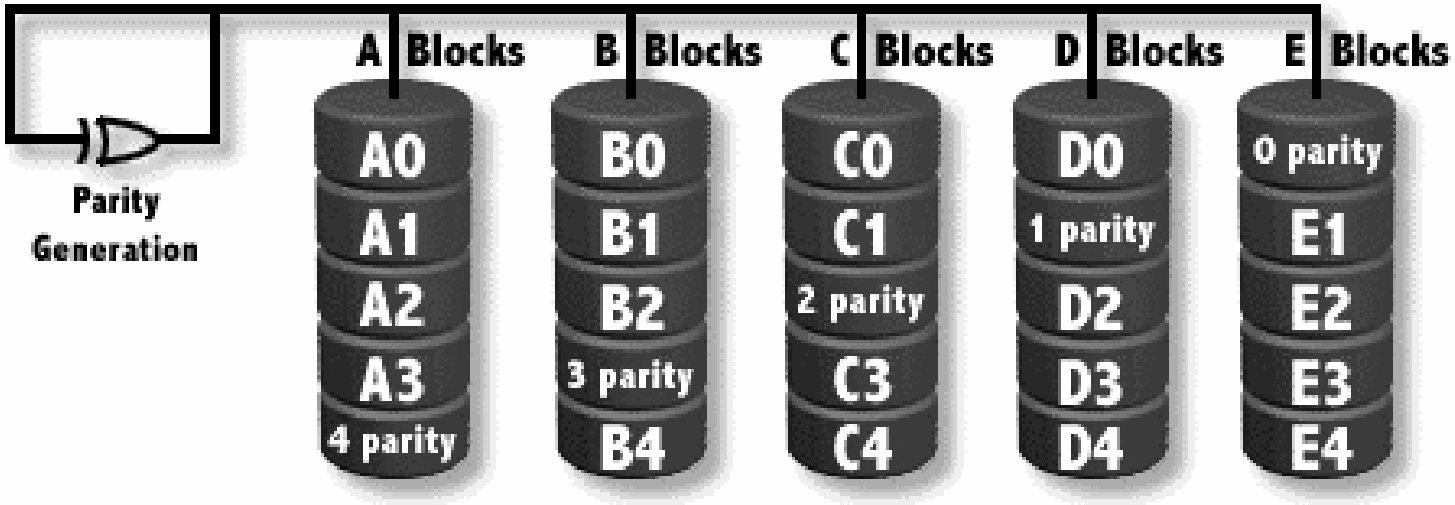


RAID-3

- ✓ Lưu dữ liệu trải đều trên các đĩa
- ✓ Sử dụng một đĩa lưu thông tin kiểm tra dữ liệu
- ✓ Tính sẵn sàng cao, chi phí hợp lý
- ✓ Hiệu suất I/O thấp

Ng Duc ... uan

Chương 4: Hệ thống quản lý File



RAID-5

- ✓ Dữ liệu, thông tin kiểm tra được lưu trải đều trên các đĩa
- ✓ Tính sẵn sàng dữ liệu trung bình, chi phí hợp lý
- ✓ Tốc độ ghi thấp
- ✓ Yêu cầu phần cứng đặc biệt

Ng Duc Thuan

Chương 5: Quản lý bộ nhớ

5.1 Đặt vấn đề

HĐH ngày nay cho phép chạy chế độ đa nhiệm=> Nhu cầu chia sẻ bộ nhớ giữa các tiến trình khác nhau.

- Hệ điều hành có nhiệm vụ cấp pháp bộ nhớ cho các tiến trình khi có yêu cầu.
- Để thực hiện tốt nhiệm vụ này, HĐH xem xét bộ nhớ dựa trên nhiều khía cạnh:
  - Sự tương tác giữa địa chỉ logic và vật lý.
  - Quản lý bộ nhớ vật lý.
  - Chia sẻ thông tin giữa các tiến trình qua bộ nhớ.
  - Bảo mọi sự truy xuất bất hợp pháp.

Ng Duc Thuan

Chương 5: Quản lý bộ nhớ

5.1 Đặt vấn đề

Quá trình ánh xạ địa chỉ tượng trưng của 1 chương trình nguồn vào bộ nhớ chính vào 3 thời điểm:

- Thời điểm biên dịch
- Thời điểm nạp
- Thời điểm xử lý

➡Không gian địa chỉ và không gian vật lý

✓ Địa chỉ logic- địa chỉ ảo do bộ xử lý tạo ra.

✓ Địa chỉ vật lý- địa chỉ thực.

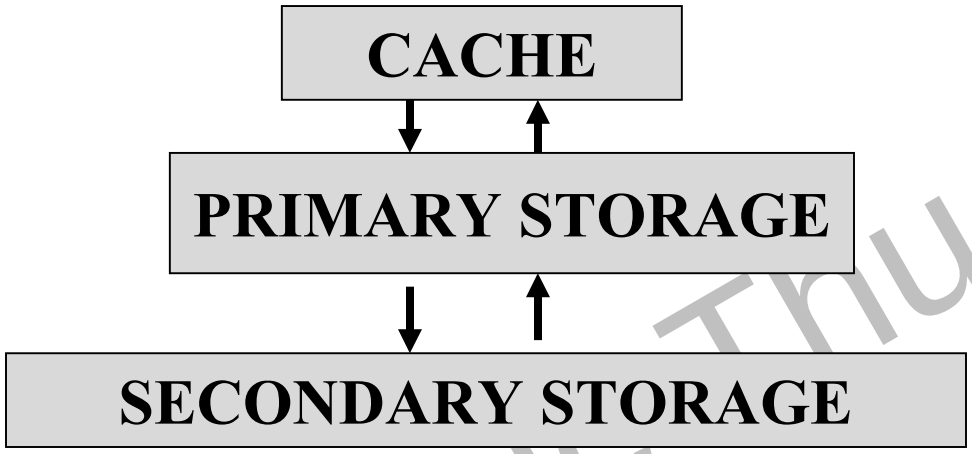
✓ Không gian địa chỉ - là tập hợp địa chỉ ảo phát sinh bởi 1 chương trình.

✓ Không gian vật lý – là tập hợp địa chỉ vật lý tương ứng đại chỉ ảo.

Ng Duc Thuan

Chương 5: Quản lý bộ nhớ
PHÂN CẤP BỘ NHỚ

- **Từ trên xuống**
  - ✓ Tốc độ giảm
  - ✓ Dung lượng tăng
  - ✓ Giá thành giảm



- **Các vấn đề quan tâm**

✓ Bộ nhớ chính chứa 1 hay nhiều tiến trình ?

✓ Các quy trình dùng vùng nhớ như nhau / khác nhau ?

✓ Bảo vệ vùng nhớ của OS và của từng tiến trình ?

✓ Vùng nhớ của quy trình là liên tục / gián đoạn ?

Ng Duc Thuan

Chương 5: Quản lý bộ nhớ

CHIẾN LƯỢC QUẢN LÝ BỘ NHỚ

Chiến lược nạp (*fetch strategies*)

- Nạp phần nào của tiến trình vào bộ nhớ và khi nào nạp ?
- Nạp theo yêu cầu & nạp tiên đoán

Chiến lược sắp đặt (*placement strategies*)

Nạp tiến trình mới vào đâu ?

Chiến lược thay thế (*replacement strategies*)

Đưa tiến trình nào ra bộ nhớ phụ ?

Ng Duc Thuan

Chương 5: Quản lý bộ nhớ

TỔ CHỨC CẤP PHÁT BỘ NHỚ

Cấp phát bộ nhớ liên tục

- Đơn lập trình
- Đa lập trình phân đoạn cố định
- Đa lập trình phân đoạn thay đổi
- Đa lập trình có thay thế vùng nhớ

Cấp phát bộ nhớ không liên tục

Ng Duc Thuan

Chương 5: Quản lý bộ nhớ

5.2 Cấp phát liên tục

Các hệ đơn chương trình

Ý tưởng: Bộ nhớ chỉ chia sẻ cho hệ điều hành và một chương trình duy nhất người sử dụng. Một phần bộ nhớ do HĐH chiếm giữ phần còn lại thuộc về tiến trình người dùng.

Thảo luận:

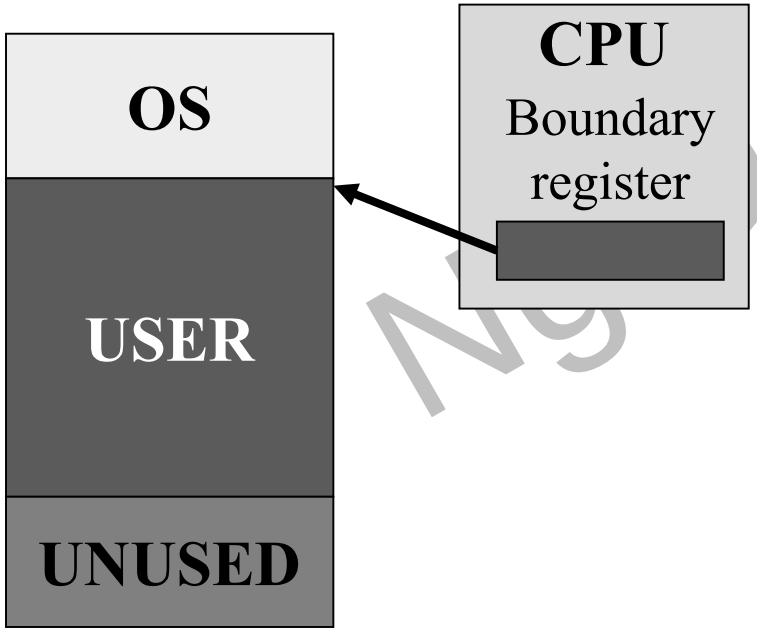
☞ Cần bảo vệ vùng nhớ khỏi sự xâm phạm tiến trình người dùng.(sử dụng thanh ghi giới hạn)

☞Tại một thời điểm chỉ có thể đáp ứng một tiến trình.

Ng Duc Thuan

Chương 5: Quản lý bộ nhớ

✓ Mỗi lần tiến trình người dùng truy xuất cần kiểm tra với nội dung thanh ghi giới hạn =>Tốc độ truy xuất không cao.
✓Sử dụng CPU không hiệu quả.



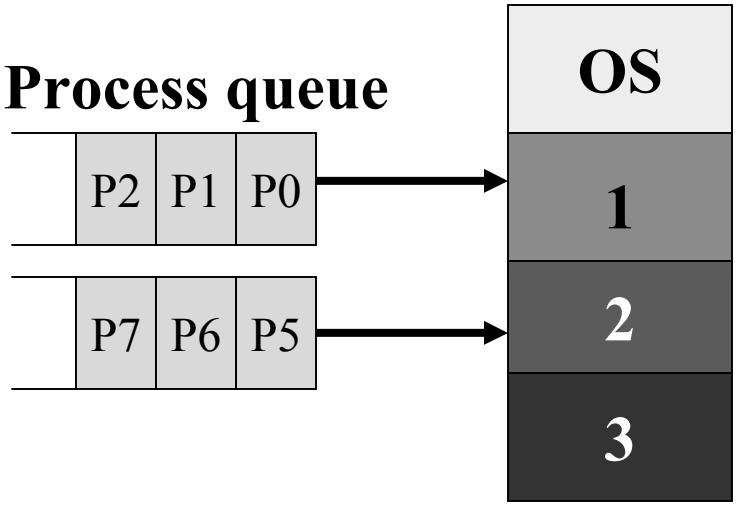
Ngọc Thuan

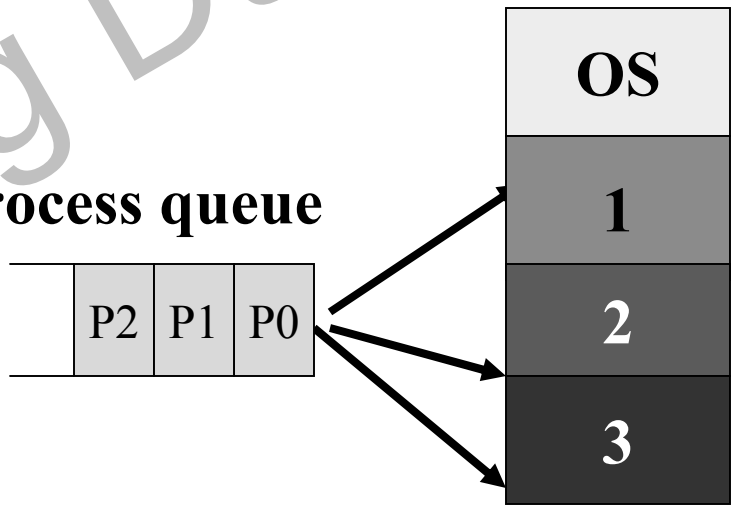
Chương 5: Quản lý bộ nhớ

2. Hệ thống đa chương trình với phân vùng cố định.

Ý tưởng: Bộ nhớ được chia thành n phân vùng có kích thước cố định (các phân vùng có kích thước khác nhau). Tiến trình có yêu cầu bộ nhớ được lưu trữ trong hàng đợi. Hàng đợi được tổ chức:

- Sử dụng mỗi phân vùng một hàng đợi.
- Sử dụng một hàng đợi duy nhất.



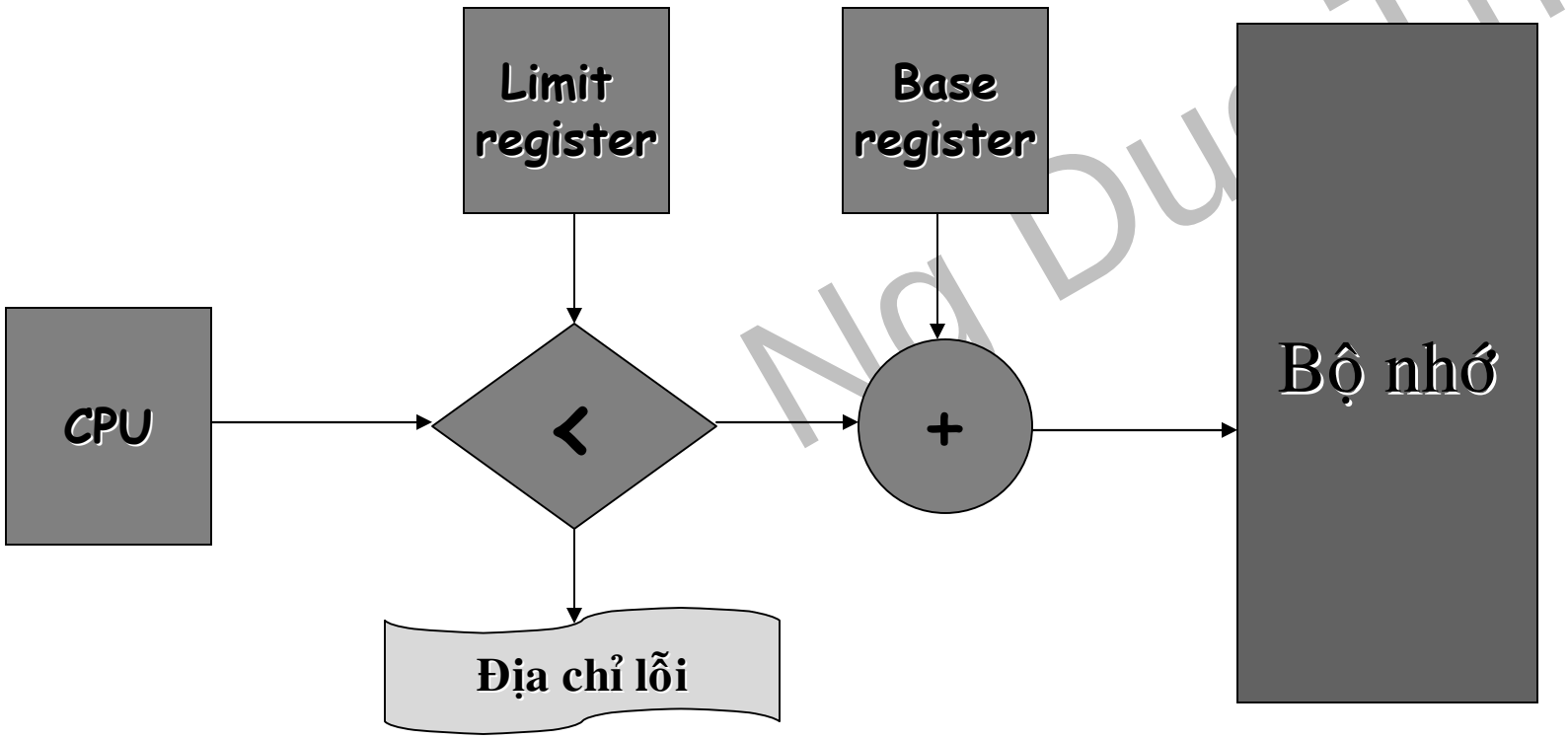


Ng Đức Thuận

Chương 5: Quản lý bộ nhớ

Thảo luận:

- ✓ Kích thước tiến trình khác nhau => phân mảnh nội.
- ✓ Mức độ đa chương của hệ thống bị giới hạn bởi phân vùng.

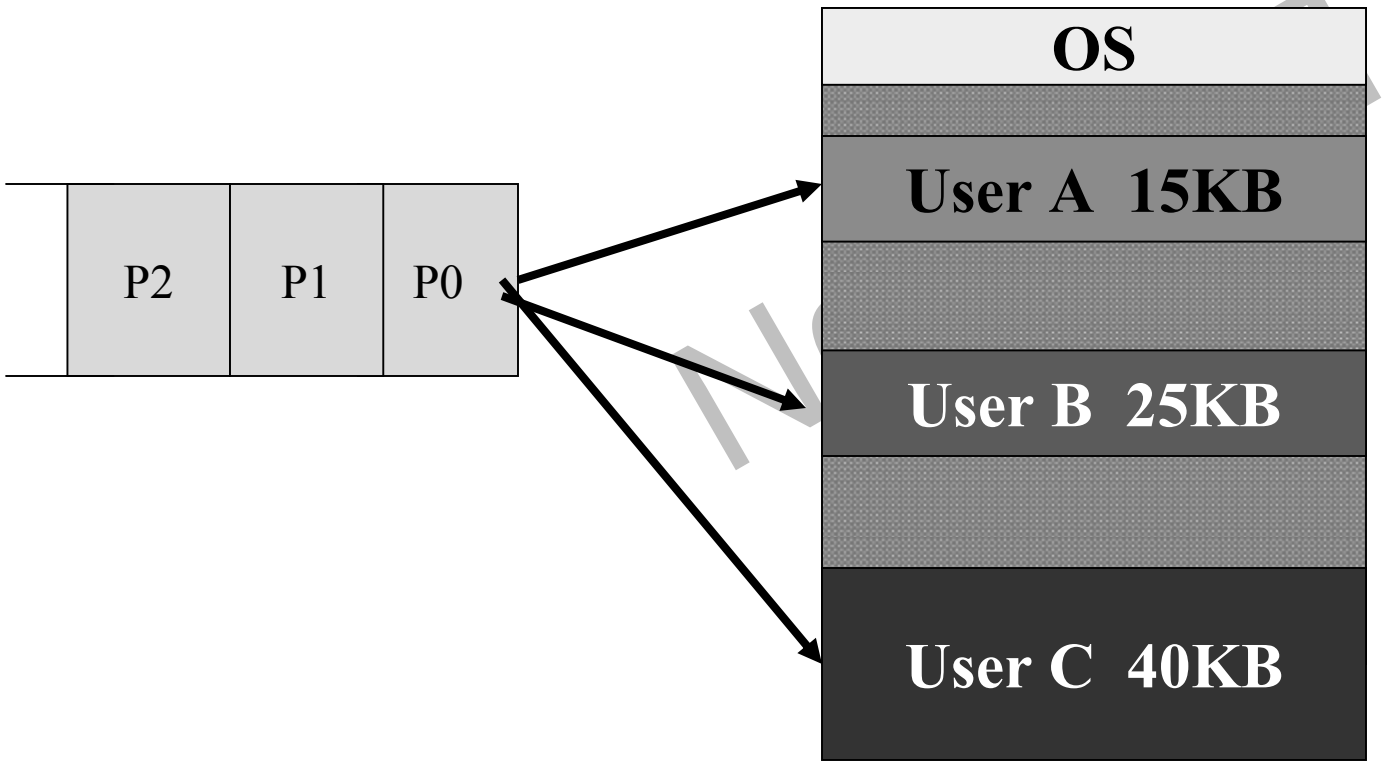


Ng Duc Thuan

Chương 5: Quản lý bộ nhớ

3. Hệ thống đa chương trình với phân vùng động.

Ý tưởng: Tiến trình được đưa vào hệ thống, cấp phát một vùng nhớ vừa đúng kích thước tiến trình. Phần còn lại cấp cho tiến trình khác.



Ng Thuan

Chương 5: Quản lý bộ nhớ

Thảo luận:

➢Không có hiện tượng phân mảnh nội vi, nhưng xuất hiện phân mảnh ngoại vi.

➢Khi kích thước tiến trình tăng trưởng trong quá trình xử lý mà không còn vùng nhớ trống kề nhau đủ rộng.

➢ Ghi lại hiện trạng bộ nhớ để cấp phát để cấp phát động cho đúng.

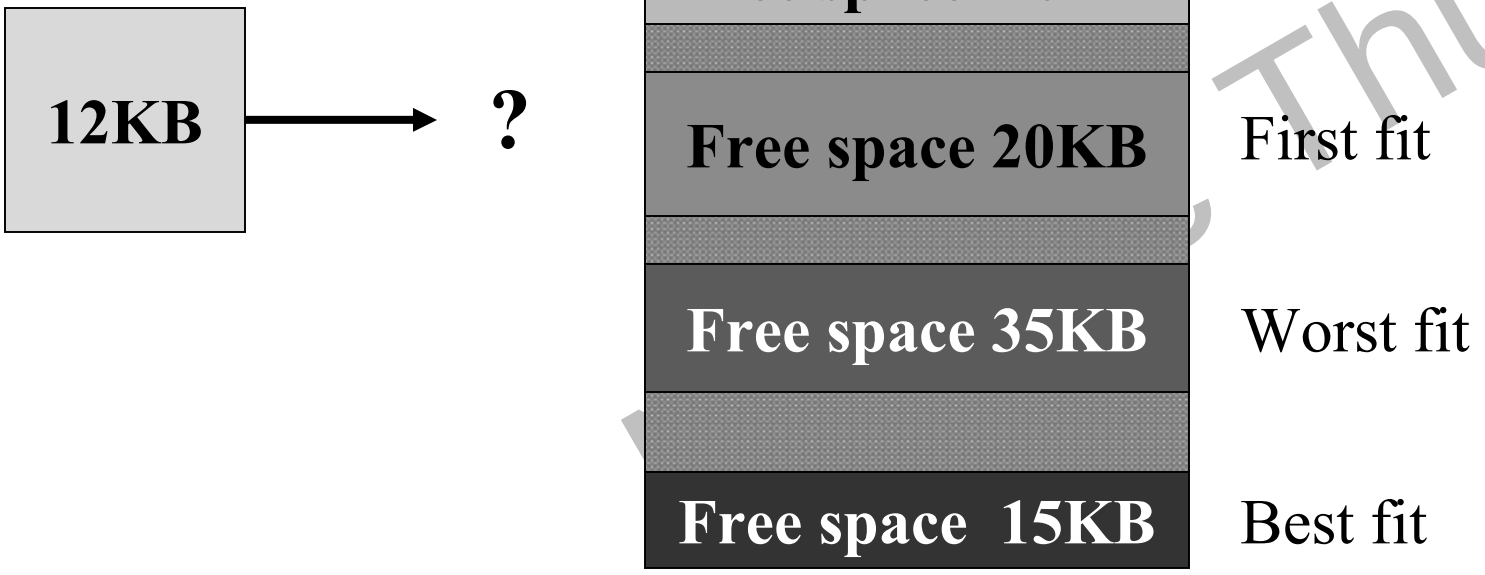
➢ Có 2 phương pháp:

- Quản lý bằng một bảng các bit.
- Quản lý bằng danh sách(First fit, Best fit, Worst fit)

Ng Duc Thuan

HỆ THỐNG ĐA CHƯƠNG PHÂN ĐOẠN THAY ĐỔI

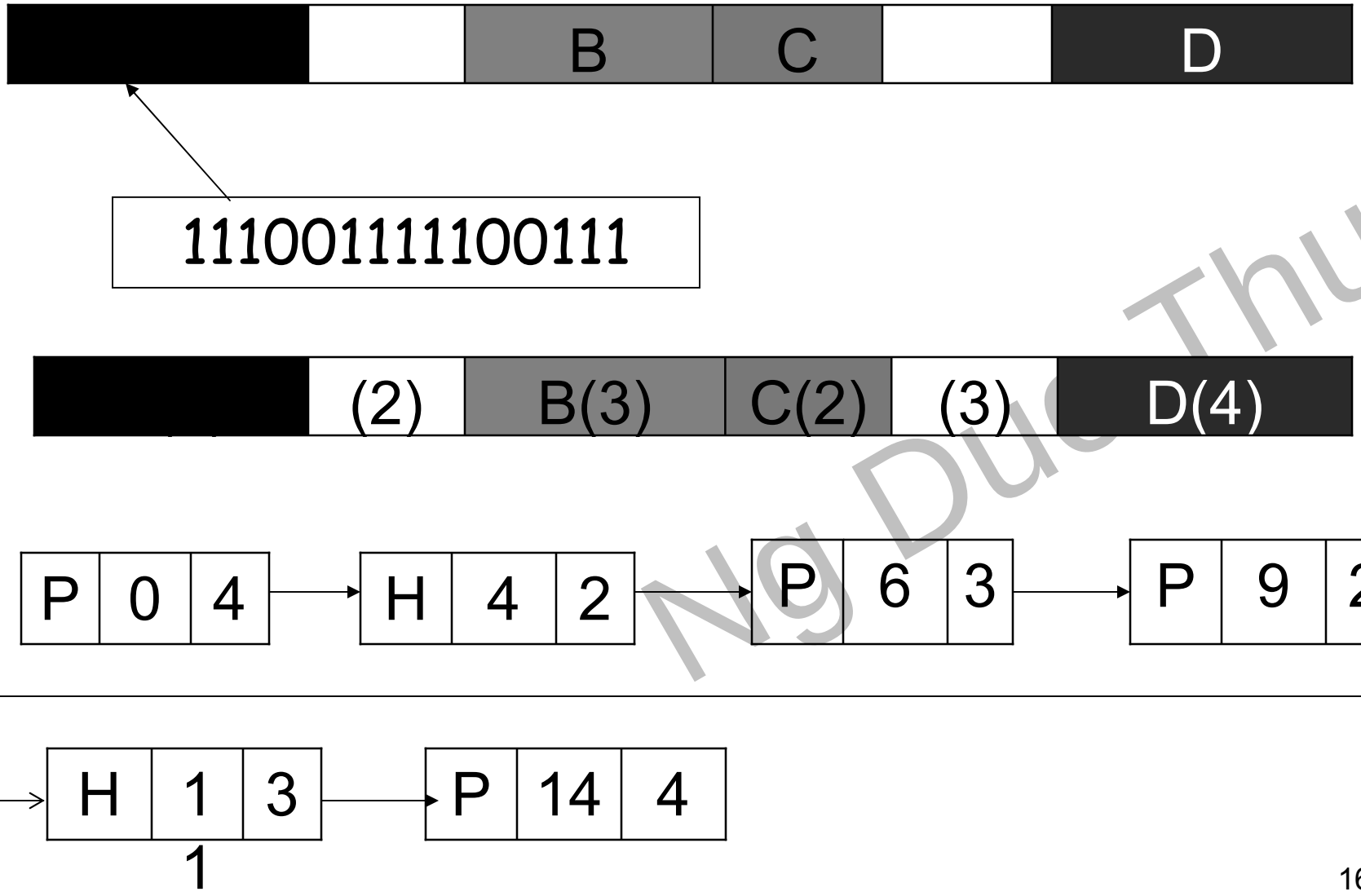
- Chiến lược sắp đặt



- Vấn đề phân mảnh vùng nhớ (*fragmentation*)

Thuan

Chương 5: Quản lý bộ nhớ

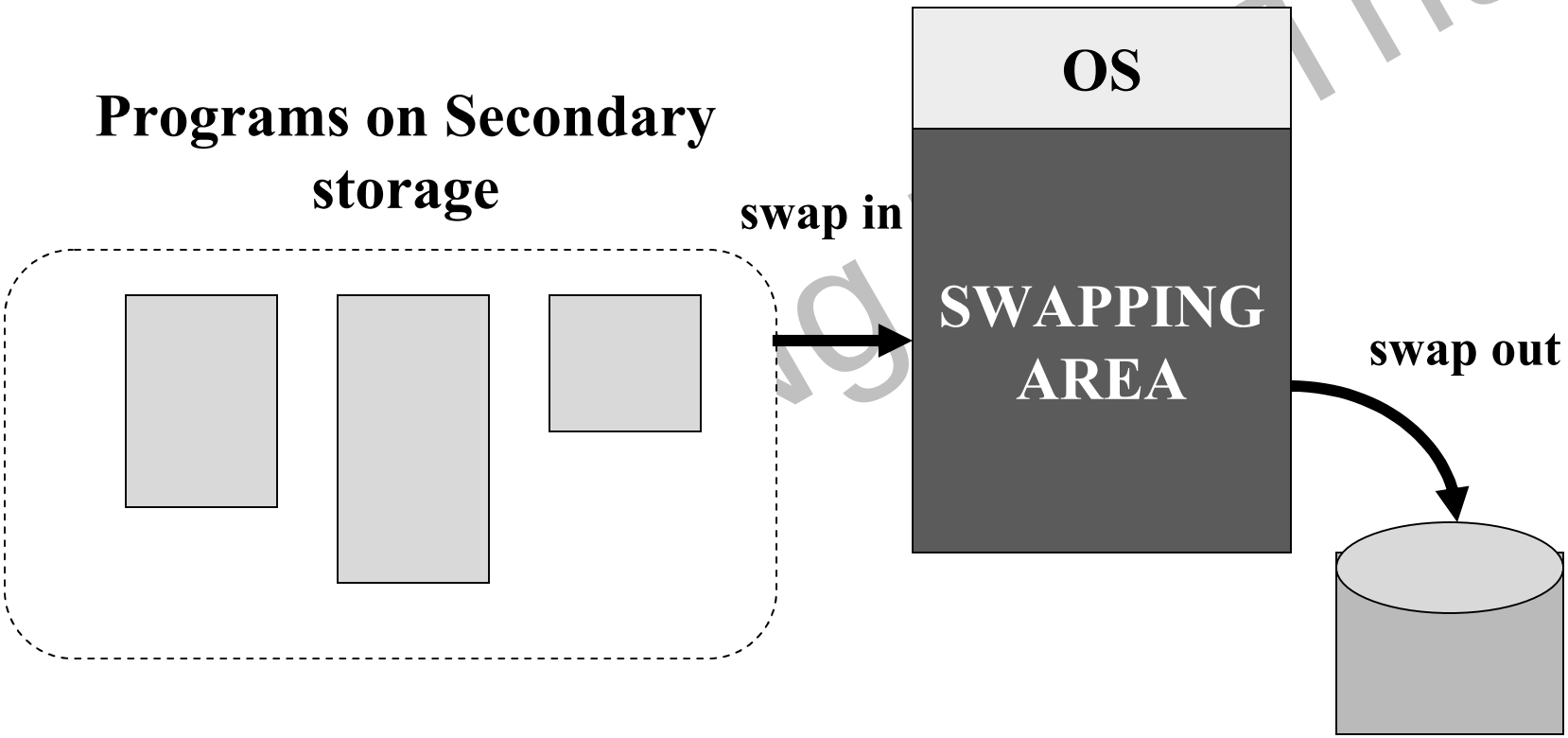


Ng Duc Thuan

Chương 5: Quản lý bộ nhớ

4. Các hệ thống đa chương với kỹ thuật "SWAPPING"

Ý tưởng: Một tiến trình chờ tương đối dài tạm thời chuyển ra bộ nhớ phụ(swap out). Khi kết thúc việc chờ tiến trình trở lại bộ nhớ chính để xử lý (swap in.)



Chương 5: Quản lý bộ nhớ

5.3 Cấp phát không liên tục

1. Phân trang(paging)

Ý tưởng:

1. Phân bộ nhớ thành các khối có kích thước cố định bằng nhau.

- ✓ Một tiến trình kích thước n trang sẽ yêu cầu n khung trang tự do.
- ✓ Cơ chế MMU(Manager Memory Unit) trong kỹ thuật phân trang.
- ✓ Hỗ trợ phần cứng thực hiện chuyển đổi địa chỉ trong cơ chế phân trang là bảng trang(pages table). Mỗi phần tử trong bảng cho biết các đại chỉ bắt đầu của vị trí lưu trữ trang tương ứng trong bộ nhớ vật lý.

Ng Duc Thuan

Chương 5: Quản lý bộ nhớ

5.3 Cấp phát không liên tục

Chuyển đổi địa chỉ:

Mỗi địa chỉ phát bởi CPU bao gồm 2 thành phần:

+ Số hiệu trang (*p*)

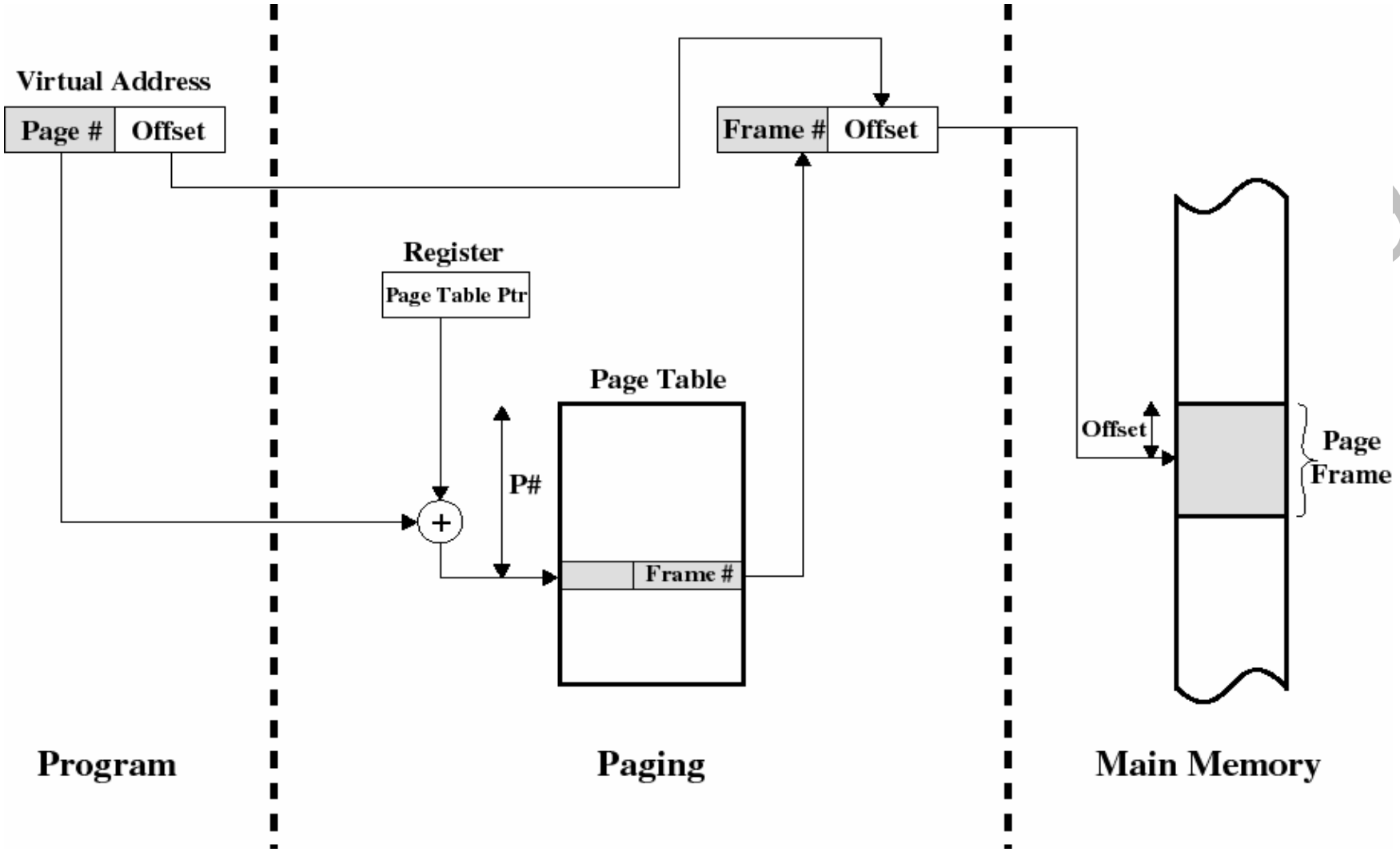
+ Địa chỉ tương đối trong trang (*d*)

Kích thước mỗi trang do phần cứng quy định thường lũy thừa của 2 nằm trong miền trị (512 ->8192).

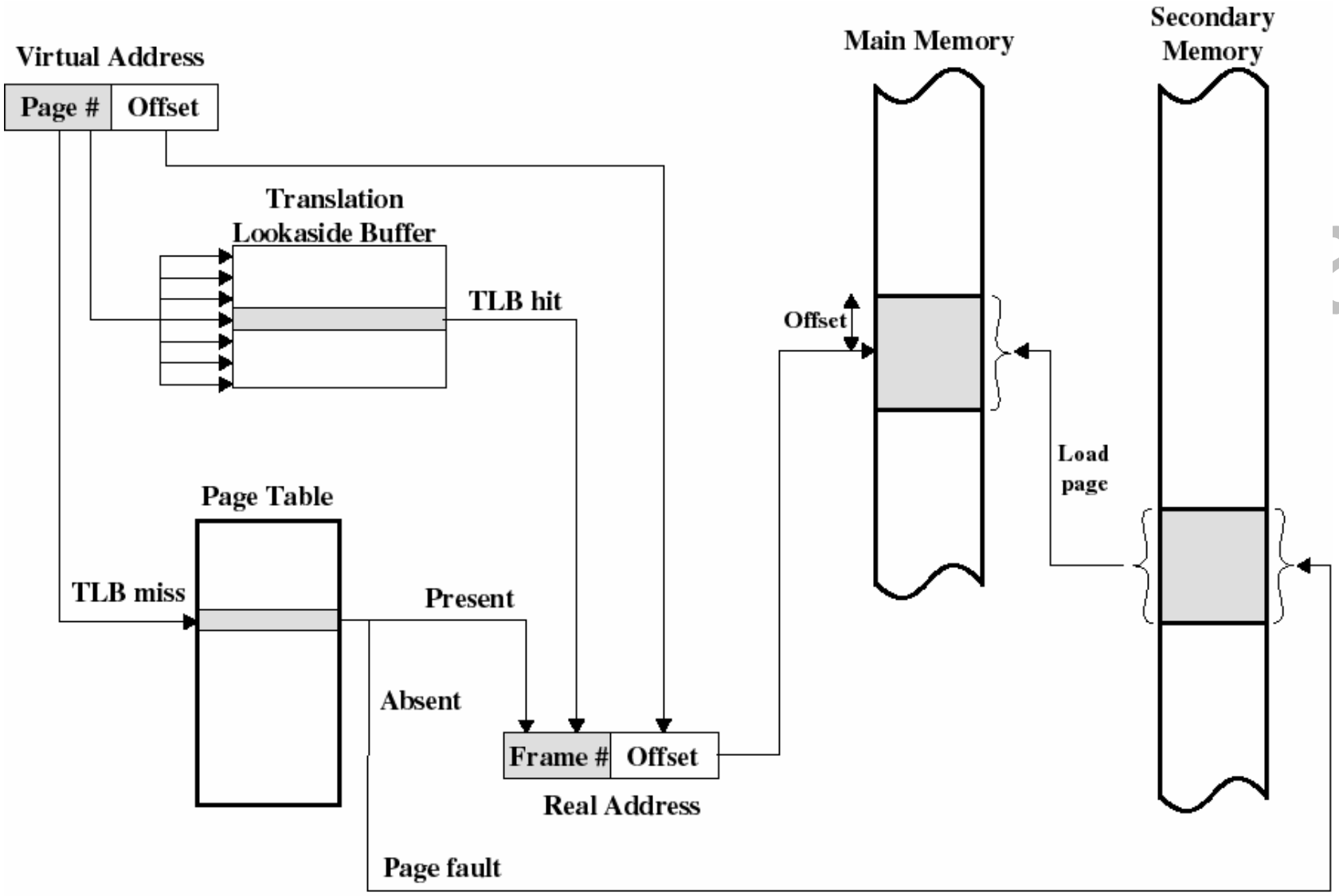
➢Nếu kích thước của không gian địa chỉ là 2^m và kích thước trang 2^n, thì m-n bits cao của địa chỉ ảo sẽ biểu diễn số trang, và n bits thấp cho biết địa chỉ tương đối trong trang.

Ng Duc Thuan

ÁNH XẠ ĐỊA CHỈ TRỰC TIẾP TRONG HỆ THỐNG PHÂN TRANG



ÁNH XẠ TRANG DÙNG BỘ NHỚ KẾT HỢP



Thảo luận

- Kỹ thuật phân trang loại bỏ hiện tượng phân mảnh ngoại vi.
- Vẫn còn xuất hiện tượng phân mảnh nội vi khi kích thước tiến trình không là bội số kích thước của trang.
- Một tiến trình người dùng coi bộ nhớ phân trang như không gian liên tục, đồng nhất và chỉ chứa duy nhất một tiến trình.
- Phần cứng nhiệm vụ đổi địa chỉ logic thành địa chỉ vật lý.
- Để lưu trữ thông tin chi tiết quá trình cấp phát bộ nhớ, HĐH sử dụng một bảng khung trang.

Ng Duc Thuan

Chương 5: Quản lý bộ nhớ

KHÁI NIỆM BỘ NHỚ ẢO

Là hình ảnh của bộ nhớ thực

- Địa chỉ ảo V: tham khảo bởi process
- Địa chỉ thực R : có trong bộ nhớ thực

$$|V| >> |R|$$

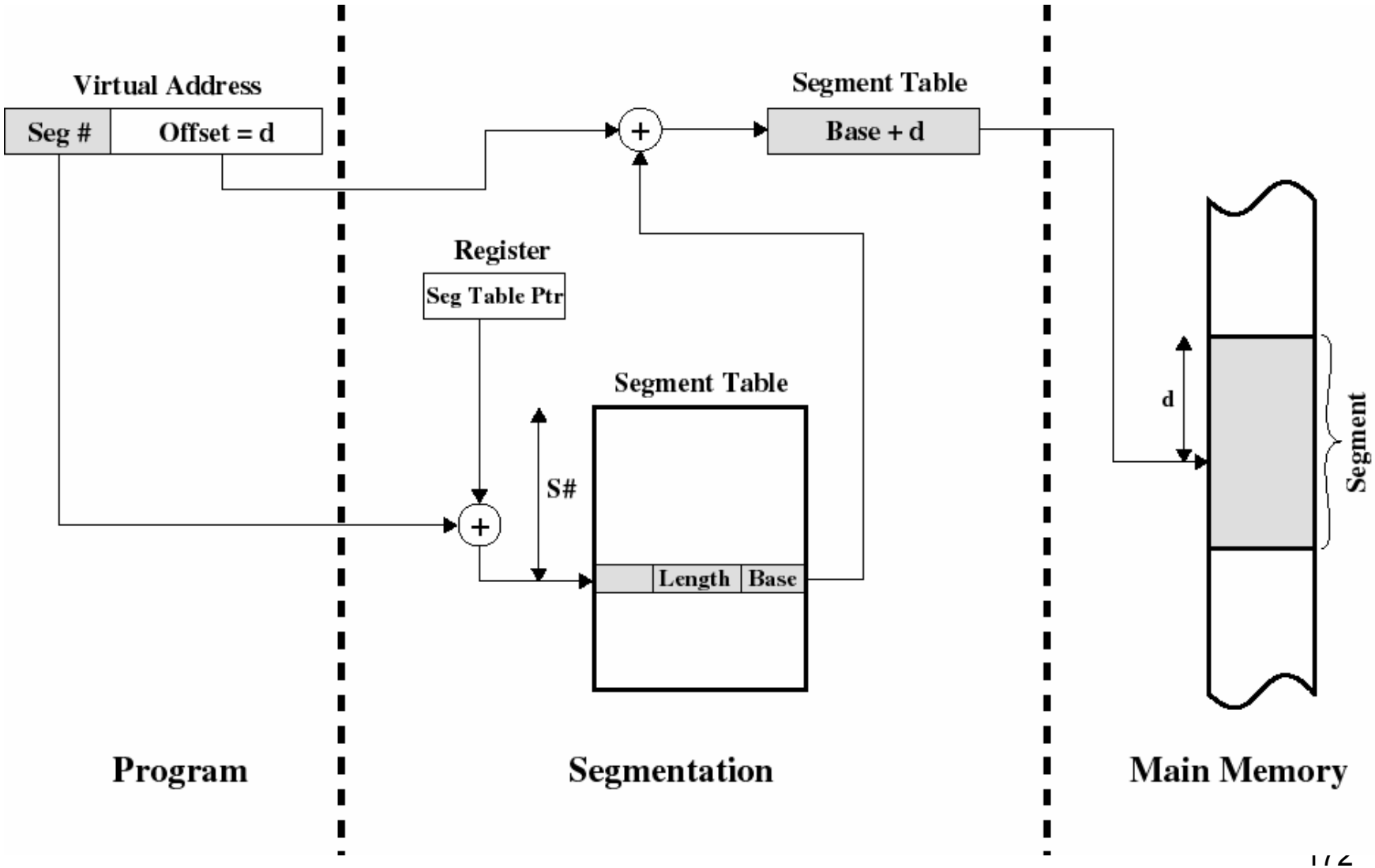
Địa chỉ ảo được ánh xạ thành địa chỉ thực mỗi khi quá trình thực thi → dynamic address translation

Sự cần thiết của bộ nhớ ảo

- Dễ phát triển ứng dụng
- Lưu trữ được nhiều quá trình trong bộ nhớ
- Tái định vị (relocation) các quá trình
- Cho các quá trình chia sẻ vùng nhớ dễ dàng

Ng Duc Thuan

ÁNH XẠ ĐỊA CHỈ TRONG HỆ THỐNG PHÂN ĐOẠN



Chương 5: Quản lý bộ nhớ

PHỐI HỢP PHÂN TRANG & PHÂN ĐOẠN

Địa chỉ ảo V=(s, p, d)

- s: chỉ số đoạn (segment #)
- p : chỉ số trang trong đoạn (page #)
- d : độ dời của ô nhớ trong trang (displacement)

Địa chỉ thực R=(p′, d′)

- p′ : chỉ số trang thực (frame #)
- d′ : độ dời của ô nhớ trong trang thực

Ánh xạ địa chỉ

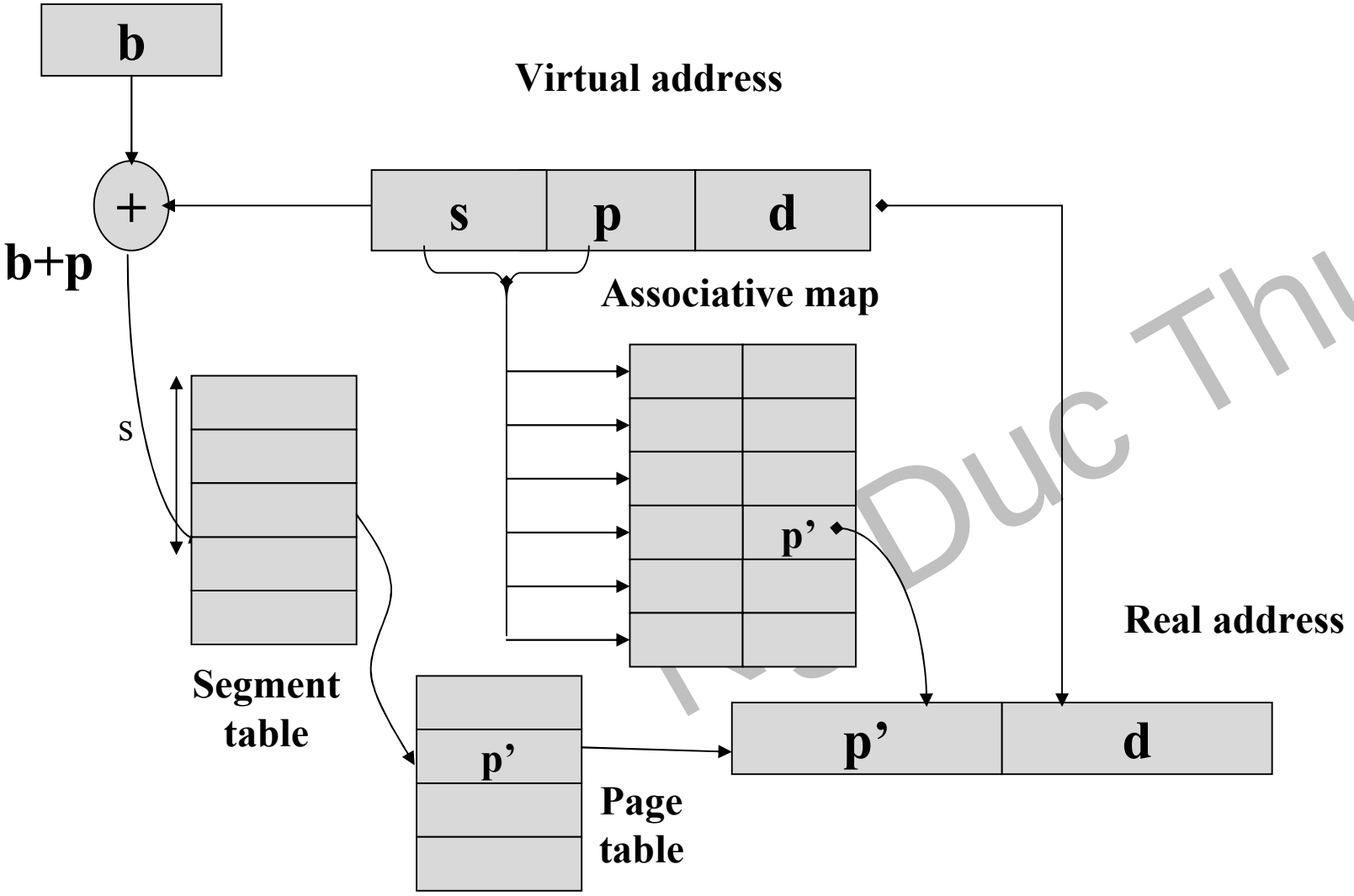
(s, p) → Associate memory → p′

Hoặc s→s′ (s′, p) → p′

(s′ : địa chỉ đầu bảng ánh xạ trang với mỗi đoạn)

Ng Duc Thuan

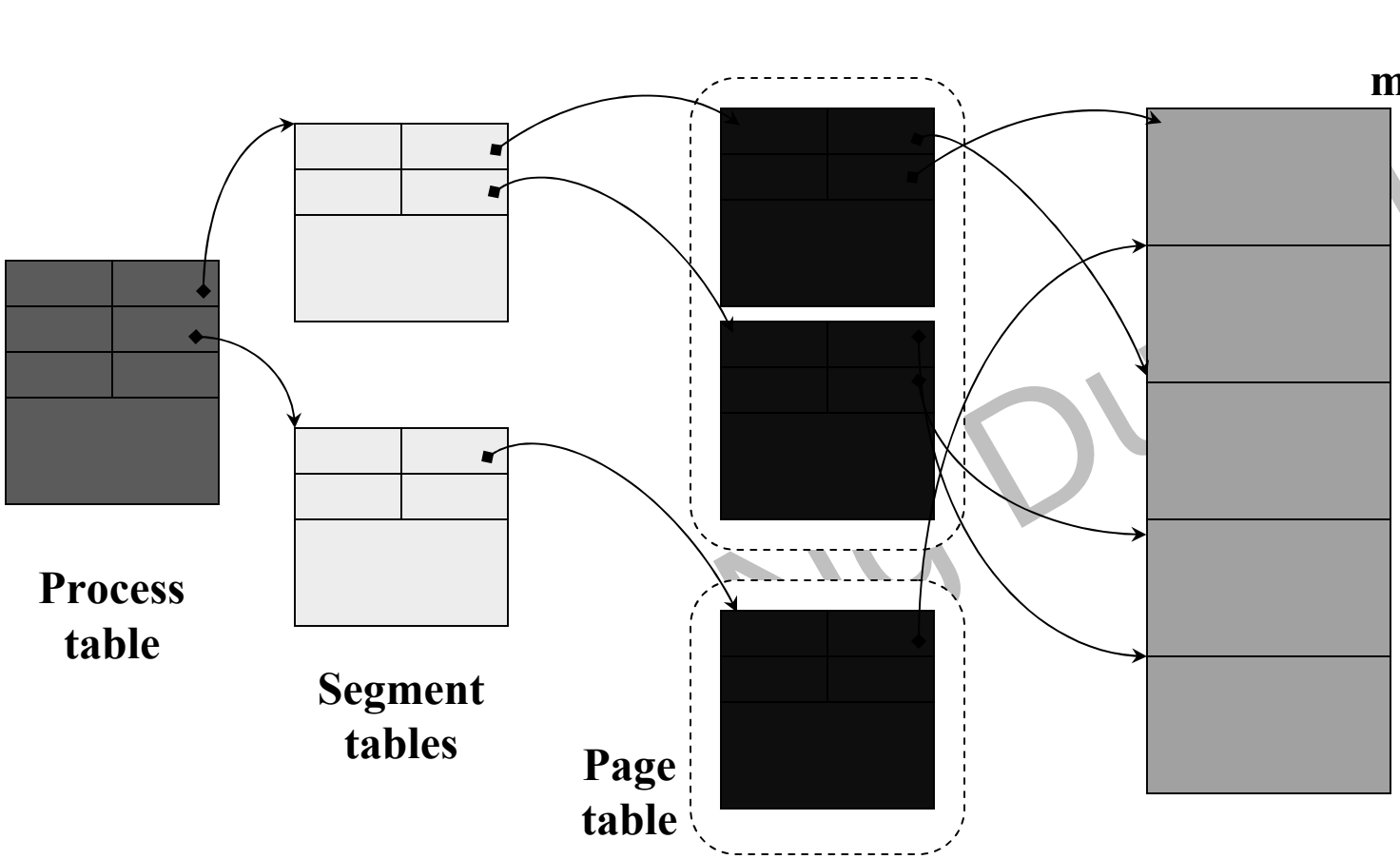
ÁNH XẠ ĐỊA CHỈ TRONG HỆ THỐNG PHÂN ĐOẠN KẾT HỢP PHÂN TRANG



Nguyễn Đức Thuận

CẤU TRÚC ÁNH XẠ BỘ NHỚ

(trong hệ thống phân đoạn kết hợp phân trang)



Du Thuan

Chương 5: Quản lý bộ nhớ

CÁC CHIẾN LƯỢC QUẢN LÝ BỘ NHỚ ẢO

Các chiến lược quản lý

- **Chiến lược nạp (*Fetch strategies*)**
- **Chiến lược sắp đặt (*Placement strategies*)**
- **Chiến lược thay thế(*Replacement strategies*)**

Chiến lược nạp

- Nạp trang theo yêu cầu (*Demand paging*)
- Nạp trang tiên đoán (*Anticipatory paging*)
- Page fault và các bước xử lý page fault

Chiến lược sắp đặt

Chiến lược thay thế

Ng Duc Thuan

CÁC GIẢI THUẬT THAY THẾ TRANG

Yêu cầu : Tối thiểu số page fault

Nguyên tắc tối ưu : Chọn trang thay thế là:

- Trang không còn dùng nữa
- Trang sẽ không dùng lại trong thời gian xa nhất

Các tiêu chuẩn (thực tế) để chọn trang thay thế

- Các trang không bị thay đổi
- Các trang không bị khóa
- Các trang không thuộc quá trình nhiều page fault
- Các trang không thuộc tập làm việc của quá trình

Một số giải thuật thay thế trang

- Thay thế trang ngẫu nhiên
- FIFO, LRU, giải thuật xấp xỉ LRU, LFU, NUR

Ng Duc Thuan

GIẢI THUẬT TỐI ƯU (OPT)

Chọn trang thay thế là trang sẽ không được tham khảo trong thời gian lâu nhất.

Ví dụ:

	1	2	3	4	1	2	5	1	2	3	4	5
Thời điểm t	0	1	2	3	4	5	6	7	8	9	10	11
Bộ nhớ thực có 3 frame	1	1	1	1	1	1	1	1	1	1	1	1
		2	2	2	2	2	2	2	2	3	4	4
			3	4	4	4	5	5	5	5	5	5

7 page fault

Duc Thuan

GIẢI THUẬT FIFO

Chọn trang thay thế là trang ở trong bộ nhớ thực trong khoảng thời gian lâu nhất.

Nghịch lý Belady

	1	2	3	4	1	2	5	1	2	3	4	5
Bộ nhớ thực có 3 frame	1	1	1	4	4	4	5	5	5	5	5	5
		2	2	2	1	1	1	1	1	3	3	3
			3	3	3	2	2	2	2	2	4	4
Bộ nhớ thực có 4 frame	1	1	1	1	1	1	5	5	5	5	4	4
		2	2	2	2	2	2	1	1	1	1	5
			3	3	3	3	3	3	2	2	2	2
				4	4	4	4	4	4	3	3	3

9 page fault

10 page fault

Dr. Duong Thuan

GIẢI THUẬT LRU (*Least Recently Used*)

Chọn trang thay thế là trang đã không được tham khảo trong thời gian lâu nhất.

Thời điểm t	0	1	2	3	4	5	6	7	8	9	10	11
Chuỗi tham khảo	1	2	3	4	1	2	5	1	2	3	4	5
Bộ nhớ thực có 3 frame	**1**	1	1	**4**	4	4	**5**	5	5	**3**	3	**5**
		2	2	2	**1**	1	1	1	1	1	**4**	4
			3	3	3	**2**	2	2	2	2	2	2

Ngọc Thuan

GIẢI THUẬT NUR (Not Used Recently)

Là giải thuật xấp xỉ LRU

Dùng thêm 2 bit cho mỗi trang

- Referenced bit R
- Modified bit M (còn gọi là dirty bit)

Trang sẽ thuộc 1 trong 4 nhóm, thay thế trang sẽ theo độ ưu tiên của nhóm trang

R	M	Ý nghĩa đối với trang nhớ
0	0	Chưa tham chiếu, chưa sửa đổi
0	1	Chưa tham chiếu, đã sửa đổi ?
1	0	Đã tham chiếu, chưa sửa đổi
1	1	Đã tham chiếu, đã sửa đổi

Thứ tự ưu tiên thay thế trang giảm dần

Ng Duc Thuan

Chương 6: Quản lý xuất /nhập

- Các khái niệm cơ bản
  - HĐH phải quản lý tất cả các thiết bị nhập xuất, ra chỉ thị cho các thiết bị, kiểm soát các ngắt và lỗi
  - HĐH phải cung cấp giao tiếp đơn giản, tiện dụng giữa các thiết bị và hệ thống. Giao tiếp phải độc lập với thiết bị
  - Lập trình viên nhìn các thiết bị IO dưới góc độ phần mềm
  - Thiết bị logic: chìa khóa của vấn đề độc lập thiết bị
  - Thiết bị logic được tổ chức thành 4 lớp: kiểm sóat lỗi, điều khiển thiết bị, phần mềm HĐH độc lập thiết bị, phần mềm mức người sử dụng

Ng Duc Thuan

Chương 6: Quản lý xuất /nhập

- Hệ thống quản lý nhập xuất
  - Tổ chức theo từng lớp. Mỗi lớp có chức năng nhất định
  - Các lớp giao tiếp với nhau theo sơ đồ:
    - Xử lý của user: tạo lời gọi nhập xuất, định dạng nhập xuất
    - Phần mềm ĐLTB: đặt tên, bảo vệ, tổ chức khối, bộ đệm định vị
    - Điều khiển thiết bị: thiết lập thanh ghi TB, kiểm tra trạng thái
    - Kiểm soát ngắt: báo cho driver khi nhập xuất hòan tất
    - Phần cứng: thực hiện thao tác nhập xuất

Ng Đức Thuan

Chương 6: Quản lý xuất /nhập

- Mô hình tổ chức và quản lý nhập xuất
  - Mô hình
    - Thiết bị nhập xuất
    - Thiết bị logic
  - Các chức năng
    - Thiết bị nhập xuất
    - Thiết bị logic

Ng Duc Thuan

Chương 6: Quản lý xuất /nhập

- Mô hình thiết bị nhập xuất. Các loại thiết bị IO:
  - Thiết bị khối:
    - Thông tin lưu trữ trong các khối có địa chỉ xác định.
    - Kích thước khối cố định. Thông thường từ 128-1024 byte
    - Dữ liệu truy xuất theo từng khối riêng biệt. VD: đĩa
  - Thiết bị tuần tự:
    - Dữ liệu lưu trên chuỗi các bits tuần tự, không có địa chỉ xác định.
    - Không seek được (VD: màn hình, bàn phím, máy in, card mạng, chuột…)
  - Các thiết bị khác, VD: bộ nhớ màn hình, đồng hồ…

Ng Đức Thuan

Chương 6: Quản lý xuất /nhập

- Điều khiển thiết bị
  - Thiết bị IO có 2 phần: phần cơ là bản thân thiết bị và phần điện tử là mạch (card) điều khiển thiết bị
  - Mỗi bộ phận điều khiển có thể gắn kết /quản lý nhiều loại thiết bị khác nhau
  - Nhà sản xuất thiết bị (và bộ điều khiển) phải tuân theo chuẩn giao tiếp. VD: ANSI, IEEE, ISO…
  - Giao tiếp giữa thiết bị và bộ điều khiển là giao tiếp mức thấp
- DMA (Direct Memory Access)
  - DMA: kênh truyền dữ liệu trực tiếp từ bộ nhớ đến thiết bị IO
  - Đa số các loại thiết bị (đặc biệt dạng khối) đều hỗ trợ DMA

Ng Đức Thuận

Chương 6: Quản lý xuất /nhập

- Thiết bị logic. Kiểm soát ngắt
  - Ngắt là tình huống phức tạp, cần được che khuất trong HĐH
  - Ngắt chỉ được tạo sau khi các tiến trình nhập xuất hoàn tất
  - Khóa tiến trình bằng lệnh WAIT hoặc RECEIVE thông điệp
- Device Drivers
  - Tất cả các đoạn mã độc lập đều chuyển đến device driver
  - Mỗi device driver kiểm soát từng loại/tập hợp thiết bị
  - Device driver phát chỉ thị và kiểm tra thực hiện chính xác

Ng Duc Thuan

Chương 6: Quản lý xuất /nhập

- Phần mềm nhập xuất độc lập thiết bị
  - Độc lập về mặt thiết bị
  - Độc lập về mặt hệ thống
- Phần mềm nhập xuất ở mức người sử dụng
  - Một phần các phần mềm nhập xuất chứa các thư viện liên kết với chương trình của người sử dụng
  - Lời gọi hệ thống nhập xuất được các hàm thư viện thực hiện

Ng Duc Thuan

Chương 6: Quản lý xuất /nhập

- Các chức năng thiết bị nhập xuất. Điều khiển thiết bị:
  - Chức năng của bộ điều khiển là giao tiếp với HĐH (qua bus)
  - Bộ điều khiển chuyển đổi dãy các bit tuần tự trong một khối các byte trong buffer của bộ điều khiển, hiệu chỉnh và chuyển dữ liệu vào bộ nhớ chính
  - Mỗi bộ điều khiển có các thanh ghi để liên lạc với CPU. Các thanh ghi này được ánh xạ thành một phần của bộ nhớ chính tại địa chỉ xác định của từng loại thiết bị
  - HĐH thực hiện nhập xuất bằng cách ghi lệnh lên thanh ghi bộ điều khiển. CPU rời bộ điều khiển để thực hiện công việc khác. Khi thực hiện xong, bộ điều khiển tạo ngắt gọi CPU đến lấy kết quả trong thanh ghi

Ng Đức Thuan

Chương 6: Quản lý xuất /nhập

- Các chức năng thiết bị nhập xuất. DMA
  - DMA: truy nhập bộ nhớ trực tiếp, giúp CPU không lãng phí thời gian
  - CPU gửi cho bộ điều khiển các thông số như địa chỉ trên đĩa (nguồn), địa chỉ trong bộ nhớ (đích), số lượng byte dữ liệu
  - Sau khi đọc tòan bộ dữ liệu từ thiết bị vào buffer, kiểm tra checksum hòan tất, bộ điều khiển chuyển byte đầu tiên vào bộ nhớ chính tại dịa chỉ DMA. Thao tác được thực hiện đến khi chuyển xong dữ liệu

Ng Duc Thuan

Chương 6: Quản lý xuất /nhập

- Thiết bị logic. Kiểm soát ngắt
  - Khi 1 ngắt xảy ra, hàm xử lý ngắt khởi tạo tiến trình xử lý ngắt
  - Chức năng của ngắt là làm cho tiến trình đang bị khóa được thi hành trở lại
- Device Drivers
  - Chức năng của device driver là nhận yêu cầu từ phần mềm nhập xuất độc lập thiết bị ở lớp trên và giám sát việc thực hiện các yêu cầu này
  - Sau khi HĐH hòan tất việc kiểm tra lỗi, driver sẽ chuyển dữ liệu cho phần mềm độc lập thiết bị, trả thông tin về trạng thái cho nơi gọi, kiểm tra hàng đợi để thực hiện tiếp hay khóa lại chờ yêu cầu tiếp theo

Ng Duc Thuan

Chương 6: Quản lý xuất /nhập

- Phần mềm nhập xuất độc lập thiết bị
  - Chức năng cơ bản: cung cấp giao tiếp đồng nhất cho phần mềm phạm vi người sử dụng
  - Chức năng 1: tạo ánh xạ giữa thiết bị và tên gọi hình thức
  - Chức năng 2: bảo vệ thiết bị (vd: quyền truy nhập của user)
  - Chức năng 3: cung cấp khối dữ liệu độc lập thiết bị
  - Chức năng 4: cung cấp buffer đồng bộ hóa hoạt động
  - Chức năng 5: định vị lưu trữ trên thiết bị khối
  - Chức năng 6: cấp phát, giải phóng các thiết bị chuyên dụng
  - Chứcnăng 7: thông báo lỗi cho lớp trên từ các lỗi do device driver báo về

Ng Đức Thuan

Chương 6: Quản lý xuất /nhập

- Phần mềm nhập xuất phạm vi người sử dụng
  - Thư viện nhập xuất chuẩn chứa một số hàm có chức năng nhập xuất, chạy như chương trình người dùng
  - Các hàm thư viện chuyển các tham số thích hợp cho lời gọi hệ thống
  - Chức năng của spooling là tránh trường hợp một tiến trình truy xuất và chiếm giữ thiết bị khiến các tiến trình khác không truy xuất được thiết bị đó
  - Ứng dụng của spooling: printing, sending/receiving email

Ng Duc Thuan

Chương 6: Quản lý xuất /nhập

- Cài đặt hệ thống quản lý nhập xuất
  - Cài đặt hệ thống nhập xuất đĩa
  - Cài đặt hệ thống nhập xuất chuẩn
  - Cài đặt đồng hồ

Ng Duc Thuan

Chương 6: Quản lý xuất /nhập

- Cài đặt hệ thống nhập xuất đĩa
  - Ưu điểm của đĩa: dung lượng, chi phí, bảo tòan thông tin
  - Cấu trúc vật lý: cylinder, track, head, sector
  - Tốc độ đĩa phụ thuộc vào các thao tác:
    - Seek: di chuyển đầu đọc đến track/cylinder (seek time *)
    - Chờ cho khối cần thiết đến dưới đầu đọc (latency time *)
    - Đọc dữ liệu từ đĩa vào bộ nhớ (transfer time)
  - HĐH cần có các thuật toán lập lịch truy xuất đĩa

Ng Duc Thuan

Chương 6: Quản lý xuất /nhập

- Các thuật toán đọc đĩa
  - Lập lịch FCFS
  - Lập lịch SSTF
  - Lập lịch SCAN
  - Lập lịch C-SCAN
  - Lập lịch LOOK

Ng Duc Thuan

Chương 6: Quản lý xuất /nhập

- Lập lịch FCFS (First Come, First Served)
  - Phương pháp đơn giản, dễ lập trình
  - Không cung cấp dịch vụ tốt

  Ví dụ đầu đọc đang ở khối 53, cần đọc các khối theo thứ tự

  98, 183, 37, 122, 14, 124, 65, 67

  Đầu đọc phải lần lượt đi qua các khối

  53, 98, 183, 37, 122, 14, 124, 65, 67

Ng Duc Thuan

Chương 6: Quản lý xuất /nhập

- Lập lịch SSTF (Shortest-Seek-Time-First)
  - Di chuyển đầu đọc đến các khối cần thiết theo vị trí lần lượt gần với vị trí hiện hành của đầu đọc nhất

  Ví dụ đầu đọc đang ở khối 53, cần đọc các khối theo thứ tự

  98, 183, 37, 122, 14, 124, 65, 67

  Đầu đọc lần lượt đi qua các khối

  53, 65, 67, 37, 14, 98, 122, 124, 183
  - Thích hợp cho hệ thống cần truy xuất dữ liệu liên tục

Ng Duc Thuan

Chương 6: Quản lý xuất /nhập

- Lập lịch SCAN
  - Đầu đọc di chuyển về 1 phía của đĩa và từ đó di chuyển qua phía kia.

  Ví dụ đầu đọc đang ở khối 53, cần đọc các khối theo thứ tự

  98, 183, 37, 122, 14, 124, 65, 67

  Đầu đọc lần lượt đi qua các khối

  53, 37, 14, 0, 65, 67, 98, 122, 124 và 183
  - Thích hợp cho hệ thống truy xuất dữ liệu khối lượng lớn

Ng Duc Thuan

Chương 6: Quản lý xuất /nhập

- Lập lịch C-SCAN
  - Tương tự thuật tóan SCAN, chỉ khác khi di chuyển đến một đầu nào đó của đĩa, nó sẽ lập tức trở về đầu bắt đầu của đĩa

  Ví dụ đầu đọc đang ở khối 53, cần đọc các khối theo thứ tự

  98, 183, 37, 122, 14, 124, 65, 67

  Đầu đọc lần lượt đi qua các khối

  53, 65, 67, 98, 122, 124, 183, 0, 14, 37

  - Thích hợp cho hệ thống truy xuất dữ liệu khối lượng lớn

Ng Duc Thuan

Chương 6: Quản lý xuất /nhập

- Lập lịch LOOK
  - Giống C-SCAN nhưng chỉ chuyển đến khối xa nhất ở mỗi hướng chứ không đến cuối.

  Ví dụ đầu đọc đang ở khối 53, cần đọc các khối theo thứ tự

  98, 183, 37, 122, 14, 124, 65, 67

  Đầu đọc lần lượt đi qua các khối

  53, 65, 67, 98, 122, 124, 183, 14, 37
  - Thích hợp cho hệ thống truy xuất dữ liệu khối lượng lớn

Ng Duc Thuan

Chương 6: Quản lý xuất /nhập

- Quản lý lỗi
  - Lỗi lập trình: tìm không thấy cylinder, sector, head, địa chỉ buffer. Xử lý bằng kiểm tra tham số, thông báo lỗi.
  - Lỗi checksum tạm thời: gây ra bởi bụi trên đầu đọc
  - Lỗi checksum thường trực: đĩa hư vật lý trên các khối
  - Lỗi tìm kiếm: seek đầu đọc sai địa chỉ
  - Lỗi điều khiển: bộ điều khiển từ chối thi hành lệnh

Ng Duc Thuan

Chương 6: Quản lý xuất /nhập

- RAM disk
  - RAM disk dùng một phần bộ nhớ chính để lưu trữ các khối dữ liệu
  - RAM disk được chia làm nhiều khối tùy theo dung lượng yêu cầu cấp phát. Mỗi khối có cùng kích thước với khối trên đĩa
  - Khi nhận được chỉ thị đọc/ghi các khối, driver tìm trong bộ nhớ RAM disk vị trí của khối, thực hiện đọc/ghi ngay trong vùng nhớ
  - RAM disk truy xuất nhanh hơn đĩa vật lý
  - HĐH phải lưu dữ liệu vào đĩa cứng trước khi người dùng shutdown

Ng Duc Thuan

Chương 6: Quản lý xuất /nhập

- Interleave
  - Khoảng cách giữa các sector dùng để đồng bộ chức năng đọc/ghi dữ liệu trên đĩa.
  - Interleave được xác định trong quá trình format đĩa

VD: đĩa có 17 sector/track, interleave=4, sơ đồ sector như sau:

1, 14, 10, 6, 2, 15, 11, 7, 3, 16, 12, 8, 4, 17, 13, 9, 5

Lần 1: **1**, 14, 10, 6, **2**, 15, 11, 7, **3**, 16, 12, 8, **4**, 17, 13, 9, **5**

Lần 2: 1, 14, 10, **6**, 2, 15, 11, **7**, 3, 16, 12, **8**, 4, 17, 13, **9**, 5

Lần 3: 1, 14, **10**, 6, 2, 15, **11**, 7, 3, 16, **12**, 8, 4, 17, **13**, 9, 5

Lần 4: 1, **14**, 10, 6, 2, **15**, 11, 7, 3, **16**, 12, 8, 4, **17**, 13, 9, 5

Ng Duc Thuan

Chương 6: Quản lý xuất /nhập

- Cài đặt hệ thống nhập xuất chuẩn (terminal)
  - Terminal là hệ thống nhập xuất (chỉ có bàn phím, màn hình và bộ điều hợp dữ liệu vào ra)
  - Terminal chuyển dữ liệu Một máy tính có thể liên lạc với nhiều terminal
  - HĐH chia terminal thành 2 loại: RS-232 và ánh xạ bộ nhớ
  - RS-232: hard-copy, glasstty, Intelligent, blit
  - Ánh xạ bộ nhớ: ký tự, bit

Ng Duc Thuan

Chương 6: Quản lý xuất /nhập

- Các terminal RS-232
  - Thiết bị giao tiếp tuần tự theo bit với bàn phím, màn hình...
  - Connector 25 pins: mass, receiver, sender, 22 reserved pins
  - Data package: start bit-data to be tranfered-end bit(s)
  - Tốc độ chuyển:1200, 2400, 4800, 9600 bps (bit per second)
  - Sử dụng bộ chuyển UART gắn trên card giao tiếp
  - Hard-copy/glassttys: nhập ký tự từ bàn phím, chuyển cho máy tính, xuất ra máy in/màn hình
  - Terminal Intelligent: gửi ký tự ASCII ESC sau những ký tự khác nhau để di chuyển con trỏ trên màn hình
  - Blit: bộ xử lý mạnh với màn hình độ phân giải 1024x800

Ng Duc Thuan

Chương 6: Quản lý xuất /nhập

- Các terminal ánh xạ bộ nhớ
  - Giao tiếp với máy tính bằng video RAM (không dùng cổng serial)
  - Chip điều khiển trên video card lấy thông tin từ video RAM, tạo ra tín hiệu video để điều khiển màn hình
  - Màn hình tạo tia điện tử quét từ trên xuống dưới.
  - Bộ điều khiển tín hiệu sẽ xác định mỗi điểm (pixel) là sáng hay tối (đối với màn hình mono), màu gì (màn hình màu)
  - Text mode (mono $B000, màu $B800): 2000 ký tự 2 byte (thuộc tính-chữ), box 9x14, bố trí trên 25 dòng, 80 cột. Graphic mode ($A000): độ phân giải 320x200, 640x480, 800x600, 1027x768... pixels

Ng Đức Thuan

Chương 6: Quản lý xuất /nhập

- Phần mềm nhập cho terminal
  - Dữ liệu nhập từ bàn phím, chuyển cho ứng dụng
  - Khi có phím nhấn, ngắt bàn phím báo cho bộ điều khiển biết có ký tự nhập đang lưu trữ trong buffer của bộ nhớ chính. Bộ điều khiển sẽ ánh xạ lại mã ASCII của ký tự phím nhấn
  - Có 2 dạng buffer bàn phím: pool buffer, structred buffer
  - Một số terminal cần phần mềm echoing điều khiển hiển thị ký tự gõ: mật khẩu, tab, backspace, line feed, carriage return...

Ng Duc Thuan

Chương 6: Quản lý xuất /nhập

- Phần mềm xuất cho terminal
  - Mỗi loại terminal sử dụng phần mềm xuất khác nhau
  - Terminal RS-232 sử dụng pool buffer chứa dữ liệu xuất: các ký tự được xuất tuần tự theo các tín hiệu ngắt
  - Terminal ánh xạ: các ký tự được xuất 1 lần từ video RAM. Các ký tự đặc biệt (backspace, bell, cr, lf...) được cập nhật cho phù hợp.
  - Các chức năng của phần mềm soạn thảo màn hình terminal: di chuyển con trỏ, chèn xóa ký tự/dòng, cuộn màn hình lên xuống, tạo hiệu ứng (tương phản, gạch dưới, nhấp nháy...), tạo/hủy/di chuyển/quản trị cửa sổ...

Ng Duc Thuan

Chương 6: Quản lý xuất /nhập

- Cài đặt đồng hồ (timer)
  - Timer là thiết bị phần cứng đặc biệt, không thuộc thiết bị khối (vd đĩa) hay thiết bị tuần tự (vd bàn phím, màn hình).
  - Chức năng chính của timer
    - Kiểm soát thời gian trong ngày
    - Phân chia thời gian chia xẻ cho các tiến trình sử dụng CPU
  - Phần mềm đồng hồ hoạt động như device driver

Ng Duc Thuan

Chương 6: Quản lý xuất /nhập

- Phần cứng đồng hồ:
  - Dạng 1: sử dụng điện thế 110/220v, tạo ngắt theo mỗi chu kỳ hiệu điện thế 50/60 MHz
  - Dạng 2: bộ dao động thạch anh, bộ đếm và bộ thanh ghi
    - Dưới tác dụng của dòng diện, tinh thể thạch anh tạo ra dao động 5-100 MHz, chuyển cho bộ đếm
    - Bộ đếm giảm dần sau mỗi dao động, tạo ngắt khi Counter=0. Bộ thanh ghi sẽ nạp lại giá trị cho bộ đếm
    - Khi ngắt đồng hồ kích hoạt, HĐH sẽ thực hiện trình xử lý ngắt của nó

Ng Duc Thuan

Chương 6: Quản lý xuất /nhập

Phần mềm đồng hồ

- Phần cứng đồng hồ tạo các ngắt theo từng khoảng thời gian đều đặn
- Phần mềm (driver) đồng hồ có nhiệm vụ:
  - Quản lí thời gian trong ngày
  - Không cho phép tiến trình chạy lâu hơn thời gian cho phép
  - Điều phối kế hoạch sử dụng CPU
  - Cung cấp watchdog timer cho hệ thống

Ng Duc Thuan

Chương 6: Quản lý xuất /nhập

- Quản lý thời gian trong ngày
  - Tăng bộ đếm sau mỗi nhịp đồng hồ.
  - Vấn đề lưu ý : Kích thước bộ đếm
  - Các loại bộ đếm:
    - Bộ đếm nhịp 32 bits: dao động 60MHz bị tràn sau 2 năm
    - Bộ đếm 64 bits: tốn kém
    - Bộ đếm giây 32 bits: 2^{32} lưu được 136 năm
    - Bộ đếm nhịp liên hệ với thời gian khởi động của hệ thống

Ng Duc Thuan

Chương 6: Quản lý xuất /nhập

- Quản lý thời gian chạy của tiến trình
  - Khi một tiến trình bắt đầu, bộ lập lịch sẽ khởi tạo giá trị cho bộ đếm.
  - Giá trị này giảm dần sau mỗi ngắt đồng hồ
  - Counter=0, thời gian chạy của tiến trình kết thúc. Bộ điều khiển đồng hồ sẽ yêu cầu bộ lập lịch thiết lập giá trị cho tiến trình khác
- Điều phối kế hoạch sử dụng CPU
  - Sử dụng timer cục bộ cho từng tiến trình
  - Khởi tạo, kích hoạt timer khi tiến trình bắt đầu
  - Dừng timer khi tiến trình kết thúc

Ng Duc Thuan

Chương 6: Quản lý xuất /nhập

- Cung cấp watchdog timer
  - Một số thiết bị nhập xuất cần đo thời gian đạt đến trạng thái sẵn sàng hoạt động

  VD: Sau 500 ms từ lúc khởi động, ổ đĩa mềm mới đạt được tốc độ cần thiết cho các tác vụ truy xuất
  - Watchdog timer đếm thời gian cho các thiết bị nhập xuất

  VD: Không tắt motor ổ đĩa mềm, chờ các thao tác nhập xuất tiếp theo. Quá thời gian không có yêu cầu truy xuất, tắt motor.

Ng Duc Thuan

VIỆN ĐẠI HỌC MỞ
KHOA CÔNG NGHỆ ĐIỆN TỬ-THÔNG TIN

BÀI GIẢNG

NGUYÊN LÝ HỆ ĐIỀU HÀNH

TRẦN ANH TUẤN

Hà Nội 2012

Chương 1
TỔNG QUAN VỀ HỆ ĐIỀU HÀNH

Vai trò, mối quan hệ của HĐH với người sử dụng và các thành phần trong máy tính.

Các khái niệm, chức năng, thành phần, cấu trúc và các tính chất, nguyên tắc xây dựng HĐH.

Các vấn đề đặt ra khi nghiên cứu HĐH.

1.1. MỘT SỐ KHÁI NIỆM LIÊN QUAN

1.1.1. Một số khái niệm về HĐH

1.1.1.1. HĐH là môi trường trung gian

- Người sử dụng MTĐT:

+ Chỉ quan tâm đến việc hệ thống có đáp ứng được các yêu cầu của họ hay không: bộ nhớ, thiết bị ngoại vi, các chương trình phần mềm…

+ Nhu cầu của người sử dụng rất lớn, đa dạng, luôn thay đổi.

+ Khai thác không triệt để các tính năng của MTĐT.

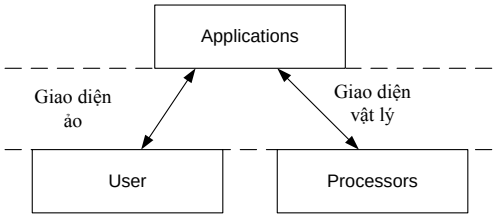
- MTĐT:

+ Khả năng thường khác xa với mong muốn của người sử dụng.

+ Tính năng của MTĐT gần như không thay đổi trong thời gian dài

+ Không phù hợp giữa chi phí phục vụ cho MTĐT và lợi ích mà MTĐT mang lại.

- Hệ điều hành là môi trường trung gian giữa *người sử dụng* với các *ứng dụng* thông qua *giao diện ảo* và giữa các *chương trình ứng dụng* với phần cứng của MTĐT qua *giao diện vật lý*, để sử dụng hiệu quả phần cứng của MTĐT.



- Giao diện ảo là giao diện giữa người sử dụng và MTĐT được thực hiện thông qua hệ thống menu. Các thành phần của hệ thống rất phức tạp và đa dạng về chức năng. Hệ thống menu là một dạng bảng chọn, giới thiệu cho người sử dụng các khả năng phục vụ của MTĐT, để người sử dụng có thể khai thác tốt các công cụ có trong tay.

+ giới thiệu các thành phần của hệ thống.

+ giới thiệu cơ chế ra lệnh - thực hiện: các danh mục công việc có thể yêu cầu.

+ các công việc được phân nhóm theo từng phạm trù, thuận tiện tìm kiếm.

+ sau khi kích hoạt hệ thống, MTĐT hướng dẫn người sử dụng chi tiết các công việc tiếp theo.

- Giao diện vật lý: các câu lệnh của các chương trình ứng dụng được chuyển thành các lệnh điều khiển hoạt động của BXL và các thành phần khác của máy tính, cung cấp chức năng sử dụng theo yêu cầu người dùng.

1.1.1.2. Một số khái niệm

HĐH là một khái niệm của ngành khoa học ứng dụng, liên quan đến nhiều lĩnh vực, nhiều lớp người sử dụng khác nhau. Có nhiều khái niệm về HĐH trên các quan điểm khác nhau.

- Quan điểm người sử dụng: HĐH là tập hợp các chương trình, phục vụ khai thác hệ thống tính toán một cách dễ dàng thuận tiện.

- Quan điểm người làm công tác quản lý: HĐH là một tập các chương trình, phục vụ quản lý chặt chẽ và sử dụng tối ưu các tài nguyên của hệ thống tính toán.

- Quan điểm người làm công tác kỹ thuật: HĐH là hệ thống chương trình, trang bị cho một máy tính cụ thể để tạo ra một máy logic mới, với các tài nguyên và khả năng mới.

- Quan điểm người làm lập trình hệ thống: HĐH là hệ thống mô hình hoá, mô phỏng các hoạt động của máy, của người sử dụng và của thao tác viên (người viết chương trình), hoạt động trong chế độ đối thoại, nhằm tạo môi trường khai thác thuận tiện và quản lý tối ưu các tài nguyên của hệ thống tính toán.

+ quan điểm từ bên trong HĐH (hệ thống)

+ xác định các thành phần của hệ thống, các mối quan hệ giữa các thành phần đó với nhau.

+ HĐH là một hệ chuyên gia: mô phỏng 3 thành phần, trong đó 2 thành phần là con người (hệ chuyên gia sớm nhất và hoàn thiện nhất: hệ thống tự động hoá trợ giúp, nâng cao hiệu quả vận hành và khai thác MTĐT).

HĐH là phần mềm hệ thống, là môi trường thực thi triển khai (thực hiện) các yêu cầu của người sử dụng, các chương trình (phần mềm) ứng dụng và điều khiển phần cứng (các tài nguyên) để đáp ứng ứng các yêu cầu của người sử dụng.

Trong phạm vi môn học, hệ điều hành là hệ thống chương trình với các chức năng điều phối (chọn lựa) và phân phối các công việc cho các thành phần chức năng trong MTĐT để sử dụng hiệu quả nhất hệ thống máy tính và thỏa mãn ở mức cao nhất yêu cầu đa dạng của người dùng.

1.1.2. Các tài nguyên hệ thống (System Resources)

- Tài nguyên là vật chất hoặc phi vật chất, có thể đáp ứng cho một nhu cầu nhất định. Tài nguyên hệ thống là các thành phần (bộ phận) trong hệ thống tính toán (máy tính), được dùng để máy tính thực hiện các chương trình theo yêu cầu. Tài nguyên hệ thống bao gồm tài nguyên phần cứng và tài nguyên phần mềm (các chương trình, các dữ liệu).

- Các thành phần chức năng trong cấu trúc logic của máy tính là những tài nguyên hệ thống, HĐH cần thực hiện quản lý khai thác sử dụng chúng. HĐH cần nắm chắc các thuộc tính của mỗi loại tài nguyên và thuộc tính được sử dụng để quản lý khai thác sử dụng (để chia sẻ) chúng. Các tài nguyên của hệ thống có 2 thuộc tính cơ bản sau:

- Thuộc tính không gian (dung lượng): là kích thước, là mức độ lưu trữ được của tài nguyên. Thuộc tính không gian chỉ có đối với tài nguyên bộ nhớ, như đĩa nhớ, bộ nhớ chính.

- Thuộc tính thời gian: là thời gian sử dụng (hay thời gian chiếm giữ) tài nguyên. Thuộc tính thời gian có đối với hầu hết các tài nguyên hệ thống, như thời gian sử dụng các processor, thời gian truy xuất dữ liệu trên bộ nhớ, sử dụng các kênh, các thiết bị điều khiển...

1.1.2.1. Bộ nhớ trong (chính):

- BN là thiết bị lưu trữ duy nhất mà thông qua đó BXL trực tiếp sử dụng các dữ liệu hoặc trao đổi các thông tin trong BN.

- Các thuộc tính của BN:

+ Thời gian truy nhập trực tiếp (thời gian trực tiếp để truy nhập đến địa chỉ ô nhớ bất kỳ trong BN)

+ Thời gian truy nhập dữ liệu tuần tự khi tổ chức lưu trữ dữ liệu tuần tự (liên tiếp)

+ Kích thước (dung lượng) BN: byte…

+ Đơn giá BN: tuỳ theo tốc độ truy nhập trực tiếp và kế tiếp

- BN được gọi là thực hiện nếu processor có thể thực hiện câu lệnh bất kỳ ghi trong đó. Với BN thực hiện, thời gian truy nhập trực tiếp và thời gian truy nhập tuần tự là gần bằng nhau.

- Các loại mức BN:

+ Mức BN trong: có mức nhớ cao nhất, thuộc quyền quản lý của hệ thống. BN trong - BN cơ sở - bao giờ cũng là BN thực hiện.

+ Mức BN trong thứ hai, kém linh hoạt hơn, nhưng có thời gian truy nhập gần bằng loại thứ nhất: BN mở rộng.

+ Mức BN ngoài - BN phụ: thời gian truy nhập trực tiếp thường lớn hơn thời gian truy nhập tuần tự (vài chục đến vài trăm lần). BN ngoài là đĩa cứng, các thiết bị nhớ, có khối lượng nhớ lớn, an toàn trong lưu trữ.

Để chương trình được thực hiện, dữ liệu (các thông tin đầu vào, các chương trình…) được đưa từ BN ngoài vào BN trong - là BN mà HĐH trực tiếp quản lý - và sau đó được BXL truy nhập đáp ứng theo yêu cầu chương trình. HĐH sẽ quản lý các vùng BN cấp phát cho chương trình và thời điểm điểm thực hiện cấp phát, vừa nhằm sử dụng BN hiệu quả nhất (đáp ứng cho nhiều chương trình đồng thời), đảm bảo yêu cầu của chương trình, vừa đảm bảo đồng bộ với tốc độ hoạt động của BXL.

1.1.2.2. Bộ xử lý (thiết bị xử lý trung tâm)

- Bộ xử lý là một tài nguyên quan trọng, được truy nhập ở mức câu lệnh và là nơi duy nhất thực hiện các câu lệnh. Mỗi processor được quản lý và phân phối riêng biệt như những tài nguyên độc lập, phân phối cho các tiến trình.

- Các thuộc tính (tốc độ xử lý, độ dài từ máy: từ máy là lượng thông tin đồng thời mà BXL xử lý trong một nhịp làm việc; độ dài từ máy là số lượng bit nhị phân của toán hạng đối số trong phép tính cơ bản của BXL). Trong đó, tham số đặc trưng của tài nguyên processor: thời gian thực hiện câu lệnh.

1.1.2.3. Thiết bị điều khiển vào/ra

- Khái niệm: thiết bị kết nối giữa BXL với các thiết bị ngoài (chuyển đổi thông tin giữa môi trường ngoài và khu vực trung tâm; đồng bộ hóa hoạt động của các thành phần và nâng hiệu quả sử dụng BXL

- Kênh (channel – bộ xử lý vào-ra): điều khiển sự trao đổi thông tin giữa BN trong và thiết bị ngoài; phân loại: kênh chậm (đa tuyến – nối với nhiều thiết bị tốc độ chậm, phục vụ lần lượt sau mỗi byte vào-ra; kênh nhanh (kênh chọn – nối vào một thiết bị, phục vụ trọn vẹn một yêu cầu vào-ra).

- Thiết bị điều khiển thiết bị ngoại vi: phân cấp chức năng của hệ thống vào-ra dưới sự điều khiển của kênh là thiết bị điều khiển thiết bị vào-ra.

- Thiết bị vào-ra: trực tiếp thực hiện thao tác đưa thông tin vào-ra.

- Tham số đặc trưng của tài nguyên thiết bị điều khiển: thời gian sử dụng, là thời gian kết nối giữa thiết bị điều khiển I/O với các thiết bị ngoại vi.

1.1.2.4. Nguyên tắc sử dụng (chia sẻ - dùng chung) tài nguyên hệ thống

Việc phân chia tài nguyên để cấp phát cho các tiến trình khi nó có yêu cầu, đặc biệt là các tiến trình hoạt động đồng thời với nhau và giải quyết vấn đề tranh chấp tài nguyên giữa các tiến trình đồng thời khi yêu cầu phục vụ của các tiến trình này vượt quá khả năng cấp phát của một tài nguyên kể cả đó là tài nguyên phân chia được.

Nguyên tắc sử dụng tài nguyên hệ thống là cách thức phân chia tài nguyên hệ thống cho các tiến trình nhằm đáp ứng cho nhiều tiến trình đồng thời và khả năng có hạn của tài nguyên hệ thống trong những điều kiện nhất định.

- Tài nguyên phân chia được (tài nguyên có thuộc tính dung lượng - BN): phân chia tài nguyên cho nhiều đối tượng sử dụng đồng thời (chia sẻ theo không gian).
- Tài nguyên không thể phân chia được (tài nguyên có thuộc tính thời gian sử dụng - phần lớn các tài nguyên trong hệ thống thuộc loại này – được chia sẻ theo thời gian):

+ phân phối lần lượt đáp ứng nhu cầu cho từng tiến trình.

+ chọn tốc độ chuyển giao và khoảng thời gian phục vụ sao cho tạo cảm giác các tiến trình được phục vụ đồng thời. Đó là quá trình *sử dụng song song*: các đĩa từ, các modul thuộc loại sử dụng nhiều lần…

+ 1 số loại tài nguyên chỉ có thể *sử dụng tuần tự*: máy in, máy vẽ… HĐH phải tạo ra tài nguyên ảo.

- Tài nguyên ảo là tài nguyên mà khi cung cấp cho người sử dụng, một hoặc một số thuộc tính của tài nguyên đó đã được biến đổi cho phù hợp yêu cầu người sử dụng. Nó chỉ xuất hiện khi hệ thống cần tới nó hoặc khi hệ thống tạo ra nó và nó sẽ tự động mất đi khi hệ thống kết thúc hay chính xác hơn là khi tiến trình gắn với nó đã kết thúc.

- Cách truy nhập và sử dụng tài nguyên ảo thể hiện trên thiết bị vật lý là khác nhau.

- Phạm vi áp dụng của tài nguyên ảo:

+ Áp dụng rộng rãi cho các tài nguyên chỉ hoạt động theo nguyên tắc tuần tự (máy in...)

+ Đối với các tài nguyên phân phối song song: tách tài nguyên vật lý thành nhiều *tài nguyên logic*, theo cách nhìn của người sử dụng (căn cứ vào dạng tài nguyên vật lý cụ thể, sử dụng kỹ thuật mô phỏng và các chương trình kích hoạt cần thiết để tổ chức các tài nguyên logic).

1.2. CHỨC NĂNG CỦA HỆ ĐIỀU HÀNH

1.2.1. Giả lập một máy tính mở rộng

Giả lập một máy tính mở rộng là tạo một môi trường giao tiếp giữa người sử dụng với máy tính mở rộng, để người sử dụng chỉ thông qua môi trường giao tiếp này tác động đến máy tính mở rộng nhằm thực hiện được các chương trình yêu cầu.

Máy tính mở rộng là một máy tính có đầy đủ các chức năng của một máy tính thực nhưng đơn giản và dễ sử dụng hơn. Khi người sử dụng tác động vào máy tính mở rộng, mọi sự chuyển đổi thông tin điều khiển từ máy tính mở rộng sang máy tính thực hoặc ngược lại đều do hệ điều hành thực hiện. Giả lập một máy tính mở rộng giúp người sử dụng khai thác các chức năng của phần cứng máy tính dễ dàng và hiệu quả hơn.

1.2.2. Quản lý, chia sẻ tài nguyên của hệ thống

Tài nguyên hệ thống, đặc biệt là các tài nguyên phần cứng thường rất giới hạn nhưng cần đáp ứng cho nhiều chương trình đồng thời. Nhằm thỏa mãn nhiều yêu cầu chỉ với số lượng tài nguyên hữu hạn, nâng cao hiệu quả sử dụng tài nguyên, HĐH cần có cơ chế và chiến lược thích hợp để quản lý việc phân phối tài nguyên.

Nhiều chương trình sử dụng có nhu cầu chia sẻ thông tin (tài nguyên phần mềm). HĐH cần đảm bảo phối hợp việc chia sẻ tài nguyên và việc truy suất đến các tài nguyên là hợp lệ.

1.2.3. Một số chức năng khác

Trên đây là hai chức năng tổng quát của một hệ điều hành, đó cũng được xem như là các mục tiêu mà các nhà thiết kế, cài đặt hệ điều hành phải hướng tới. Các hệ điều hành hiện nay có các chức năng cụ thể sau đây:

- Hệ điều hành cho phép thực hiện nhiều chương trình đồng thời trong môi trường đa tác vụ - **Multitasking Environment.** Hệ điều hành phải xác định khi nào thì một ứng dụng được chạy và mỗi ứng dụng được chạy trong khoảng thời gian bao lâu thì phải dừng lại để cho các ứng dụng khác được chạy.
- Hệ điều hành tự nạp nó vào bộ nhớ - **It loads itself into memory**: Quá trình nạp hệ điều hành vào bộ nhớ được gọi là quá trình **Booting**. Chỉ khi nào hệ điều hành đã được nạp vào bộ nhớ thì nó mới cho phép người sử dụng giao tiếp với phần cứng. Trong các hệ thống có nhiều ứng dụng đồng thời hoạt động trên bộ nhớ thì hệ điều hành phải chịu trách nhiệm chia sẻ không gian bộ nhớ RAM và bộ nhớ cache cho các ứng dụng này.
- **Hệ điều hành và API: Application Programming Interface**: API là một tập các hàm/thủ tục được xây dựng sẵn bên trong hệ thống. Hệ điều hành giúp cho chương trình của người sử dụng giao tiếp với API hay thực hiện một lời gọi đến các hàm/thủ tục của API.

- Nạp dữ liệu cần thiết vào bộ nhớ - **It loads the requied data into memory**: Hệ điều hành phải luôn theo dõi bản đồ cấp phát bộ nhớ, gồm địa chỉ dữ liệu và chương trình được lưu trữ tại BN để ghi và đọc các dữ liệu, chương trình khi cần thiết.

- Hệ điều hành biên dịch các chỉ thị chương trình - **It interprets program instructions**: Hệ điều hành cũng chịu trách nhiệm sinh ra thông báo lỗi khi hệ thống gặp lỗi trong khi đang hoạt động.

1.3. THÀNH PHẦN VÀ CẤU TRÚC CỦA HỆ ĐIỀU HÀNH

1.3.1. Thành phần HĐH

1.3.1.1. Thành phần HĐH theo quan điểm modul

Hệ điều hành là một hệ thống chương trình lớn, thực hiện nhiều nhiệm vụ khác nhau, do đó các nhà thiết kế thường chia hệ điều hành thành nhiều thành phần (modul chương trình), mỗi thành phần đảm nhận một nhóm các nhiệm vụ nào đó, các nhiệm vụ này có liên quan với nhau. Cách phân chia nhiệm vụ cho mỗi thành phần, cách kết nối các thành phần lại với nhau để nó thực hiện được một nhiệm vụ lớn hơn khi cần và cách gọi các thành phần này khi cần nó thực hiện một nhiệm vụ nào đó... Tất cả các phương thức trên tạo nên cấu trúc của hệ điều hành.

Modul chương trình điều khiển: quản lý tài nguyên; quản lý nhiệm vụ, tiến trình; quản lý dữ liệu và tổ chức truy nhập; thư ký và điều phối nhiệm vụ.

Modul chương trình phục vụ hệ thống: biên tập; dịch; phục vụ (tạo môi trường mới, biên bản, thống kê…).

Các chương trình hệ thống luôn có mặt trong BN chính để điều khiển máy tính làm việc – chúng thuộc vào nhân hệ thống; các chương trình điều khiển chỉ được nạp vào BN chính khi cần thiết. Các modul được đưa vào nhân là những modul thường xuyên được sử dụng nhiều nhất.

Nhân của HĐH thông thường:

- Modul chương trình tải (Loader): đưa một chương trình vào BN chính bắt đầu từ một địa chỉ nào đó để sau đó cho phép chương trình đã được tải nhận điều khiển để chạy hoặc không;

- Modul chương trình dẫn dắt (monitor): lựa chọn các bước làm việc của toàn hệ thống.

- Modul chương trình lập lịch (scheduler): lựa chọn chương trình tiếp theo để thực hiện;

- Một số modul khác và các thông tin hệ thống là tham số hệ thống.

1.3.1.2. Thành phần của hệ điều hành theo chức năng

a- Thành phần quản lý tiến trình

Tạo lập tiến trình và đưa nó vào danh sách quản lý tiến trình của hệ thống. Khi tiến trình kết thúc, phải loại bỏ tiến trình ra khỏi danh sách quản lý tiến trình của hệ thống.

Cung cấp đầy đủ tài nguyên để tiến trình đi vào hoạt động và phải đảm bảo đủ tài

nguyên để duy trì sự hoạt động của tiến trình cho đến khi tiến trình kết thúc. Khi tiến trình kết thúc, phải thu hồi những tài nguyên mà hệ điều hành đã cấp cho tiến trình.

Khi tiến trình không thể tiếp tục hoạt động được thì HĐH phải tạm dừng tiến trình, thu hồi tài nguyên mà tiến trình đang chiếm giữ, sau đó nếu điều kiện thuận lợi thì HĐH phải tái kích hoạt tiến trình để tiến trình tiếp tục hoạt động cho đến khi kết thúc.

Trong các hệ thống có nhiều tiến trình hoạt động song song hệ điều hành phải giải quyết vấn đề tranh chấp tài nguyên giữa các tiến trình, điều phối processor cho các tiến trình, giúp các tiến trình trao đổi thông tin và hoạt động đồng bộ với nhau, đảm bảo nguyên tắc tất cả các tiến trình đã được khởi tạo phải được thực hiện và kết thúc được.

Tóm lại, bộ phận quản lý tiến trình của HĐH phải thực hiện những nhiệm vụ sau đây:

- Tạo lập, hủy bỏ tiến trình.
- Tạm dừng, tái kích hoạt tiến trình.
- Tạo cơ chế thông tin liên lạc giữa các tiến trình.
- Tạo cơ chế đồng bộ hóa giữa các tiến trình.

b- Thành phần quản lý bộ nhớ chính

Bộ nhớ chính là một trong những tài nguyên quan trọng của hệ thống, đây là thiết bị lưu trữ duy nhất mà BXL có thể truy xuất trực tiếp được.

Các chương trình của người sử dụng muốn thực hiện được bởi BXL thì trước hết nó phải được hệ điều hành nạp vào bộ nhớ chính, chuyển đổi các địa chỉ sử dụng trong chương trình thành những địa chỉ mà BXL có thể truy xuất được.

Khi chương trình, tiến trình có yêu cầu được nạp vào bộ nhớ thì hệ điều hành phải cấp phát không gian nhớ cho nó. Khi chương trình, tiến trình kết thúc thì hệ điều hành phải thu hồi lại không gian nhớ đã cấp phát cho chương trình, tiến trình trước đó.

Trong các hệ thống đa chương hay đa tiến trình, trong bộ nhớ tồn tại nhiều chương trình/ nhiều tiến trình, hệ điều hành phải thực hiện nhiệm vụ bảo vệ các vùng nhớ đã cấp phát cho các chương trình/ tiến trình, tránh sự vi phạm trên các vùng nhớ của nhau.

Tóm lại, bộ phận quản lý bộ nhớ chính của HĐH thực hiện những nhiệm vụ sau:

- Cấp phát, thu hồi vùng nhớ.
- Ghi nhận trạng thái bộ nhớ chính.
- Bảo vệ bộ nhớ.
- Quyết định tiến trình nào được nạp vào bộ nhớ.

c- Thành phần quản lý vào-ra

Một trong những mục tiêu của HĐH là giúp người sử dụng khai thác hệ thống máy tính dễ dàng và hiệu quả, do đó các thao tác trao đổi thông tin trên thiết bị xuất/ nhập phải *trong suốt* đối với người sử dụng.

Hệ điều hành có một bộ phận điều khiển thiết bị I/O, bộ phận này phối hợp cùng BXL để quản lý sự hoạt động và trao đổi thông tin giữa hệ thống, chương trình người sử dụng và người sử dụng với các thiết bị xuất/ nhập. Bộ phận điều khiển thiết bị thực hiện những nhiệm vụ sau:

- Gửi mã lệnh điều khiển đến thiết bị bằng các mã điều khiển trước khi bắt đầu một quá trình trao đổi dữ liệu với thiết bị.

- Tiếp nhận yêu cầu ngắt (Interrupt) từ các thiết bị khi thiết bị cần trao đổi với hệ thống, hệ điều hành tiếp nhận yêu cầu ngắt, xem xét và thực hiện một thủ tục để đáp ứng yêu cầu của các thiết bị.

- Phát hiện và xử lý lỗi: hệ điều hành phải tạo ra các cơ chế thích hợp để phát hiện lỗi sớm nhất và khắc phục các lỗi vừa xảy ra nếu có thể.

d- Thành phần quản lý bộ nhớ phụ (đĩa)

Sau mỗi thao tác cấp phát block tại BN, HĐH phải ghi nhận trạng thái của các block trên đĩa, đặc biệt là các block còn tự do để chuẩn bị cho các quá trình cấp block sau này.

Trong quá trình sử dụng tập tin, nội dung của tập tin có thể thay đổi (tăng, giảm), do đó hệ điều hành phải tổ chức cấp phát động các block cho tập tin.

Lập lịch cho đĩa là lựa chọn thứ tự đọc các block trên đĩa nhằm nâng cao tốc độ đọc dữ liệu trên đĩa.

Tóm lại, bộ phận quản lý bộ nhớ phụ thực hiện những nhiệm vụ sau:

- Quản lý không gian trống trên đĩa.
- Định vị lưu trữ thông tin trên đĩa.
- Lập lịch cho vấn đề ghi/ đọc thông tin trên đĩa của đầu từ.

e- Thành phần quản lý tập tin

Tập tin là đơn vị lưu trữ cơ bản nhất trên các thiết bị lưu trữ vật lý, mỗi tập tin có một tên riêng. Hệ điều hành phải thiết lập mối quan hệ tương ứng giữa tên tập tin và thiết bị lưu trữ chứa tập tin. Việc truy xuất đến thông tin đang lưu trữ trên bất kỳ thiết bị lưu trữ nào được thực hiện thông qua tên của nó, các thao tác còn lại do hệ điều hành thực hiện.

Trong hệ thống có nhiều tiến trình đồng thời truy xuất tập tin hệ điều hành phải tạo ra những cơ chế thích hợp để bảo vệ tập tin tránh việc ghi/ đọc bất hợp lệ trên tập tin.

Tóm lại: bộ phận quản lý tập tin của hệ điều hành thực hiện những nhiệm vụ sau:

- Tạo/ xoá một tập tin/ thư mục.
- Bảo vệ tập tin khi có hiện tượng truy xuất đồng thời.
- Cung cấp các thao tác xử lý và bảo vệ tập tin/ thư mục.
- Tạo mối quan hệ giữa tập tin và bộ nhớ phụ chứa tập tin.
- Tạo cơ chế truy xuất tập tin thông qua tên tập tin.

g- Thành phần thông dịch lệnh

Đây là bộ phận quan trọng nhất của hệ điều hành, nó đóng vai trò giao tiếp giữa hệ điều hành và người sử dụng. Các lệnh được chuyển đến HĐH dưới dạng chỉ thị điều khiển. Chương trình shell – bộ thông dịch lệnh – chỉ làm nhiệm vụ đơn giản là nhận lệnh tiếp theo và thông dịch lênh đó để HĐH có xử lý tương ứng.

h- Thành phần bảo vệ hệ thống

Các tiến trình đồng thời được bảo vệ lẫn nhau (tránh sự xâm phạm lẫn nhau làm sai lệch hệ thống). Do đó, HĐH cung cấp cơ chế để đảm bảo rằng tập tin, bộ nhớ, BXL,

và những tài nguyên khác chỉ được truy xuất bởi những tiến trình có quyền. Ví dụ, bộ nhớ đảm bảo rằng tiến trình chỉ được thi hành trong phạm vi địa chỉ của nó. Bộ thời gian đảm bảo rằng không có tiến trình nào độc chiếm BXL và các thiết bị ngoại vi cũng được bảo vệ.

Hệ thống bảo vệ là một cơ chế kiểm soát quá trình truy xuất của chương trình, tiến trình, hoặc người sử dụng với tài nguyên của hệ thống. Cơ chế này cũng cung cấp cách thức để mô tả lại mức độ kiểm soát.

Hệ thống bảo vệ cũng làm tăng độ an toàn khi kiểm tra lỗi trong giao tiếp giữa những hệ thống nhỏ bên trong.

Ngoài ra các hệ điều hành mạng, các hệ điều hành phân tán hiện nay còn có thêm thành phần kết nối mạng và truyền thông..

Để đáp ứng yêu cầu của người sử dụng và chương trình người sử dụng các nhiệm vụ của hệ điều hành được thiết kế dưới dạng các dịch vụ:

- Thi hành chương trình: nạp chương trình của người sử dụng vào bộ nhớ, chuẩn bị đầy đủ các điều kiện về tài nguyên để chương trình có thể chạy được và kết thúc được, có thể kết thúc bình thường hoặc kết thúc do bị lỗi. Khi chương trình kết thúc hệ điều hành phải thu hồi tài nguyên đã cấp cho chương trình và ghi lại các thông tin mà chương trình đã thay đổi trong quá trình chạy (nếu có).
- Thực hiện các thao tác xuất nhập dữ liệu: hệ điều hành hỗ trợ việc xuất nhập dữ liệu cho chương trình, phải nạp được dữ liệu mà chương trình cần vào bộ nhớ.
- Thực hiện các thao tác trên hệ thống tập tin: cung cấp các công cụ để chương trình dễ dàng thực hiện các thao tác đọc ghi trên các tập tin, các thao tác này phải thực sự an toàn, đặc biệt là trong môi trường đa nhiệm.
- Trao đổi thông tin giữa các tiến trình: cung cấp các dịch vụ cần thiết để các tiến trình có thể trao đổi thông tin với nhau và phối hợp cùng nhau để hoàn thành một tác vụ nào đó.
- Phát hiện và xử lý lỗi: Hệ điều hành phải có các công cụ để chính hệ điều hành và để hệ điều hành giúp chương trình của người sử dụng phát hiện các lỗi do hệ thống phát sinh. Hệ điều hành cũng phải đưa ra các dịch vụ để xử lý các lỗi sao cho hiệu quả nhất.

1.3.2. Các cấu trúc của hệ điều hành

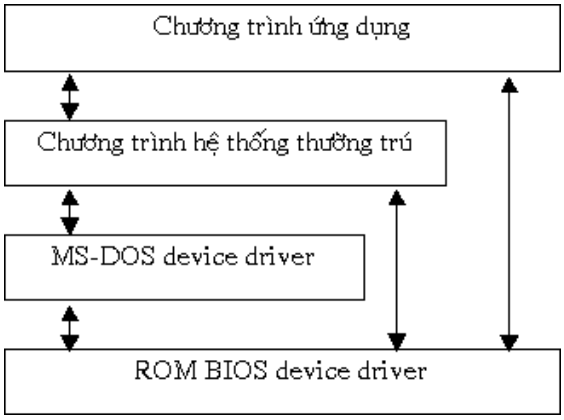
1.3.2.1. Hệ thống đơn khối (monolithic systems)

Trong hệ thống này hệ điều hành là một tập hợp các thủ tục, mỗi thủ tục có thể gọi thực hiện một thủ tục khác bất kỳ lúc nào khi cần thiết.

Hệ thống đơn khối thường được tổ chức theo nhiều dạng cấu trúc khác nhau:

- Sau khi biên dịch tất cả các thủ tục riêng hoặc các file chứa thủ tục của hệ điều hành được liên kết lại với nhau và được chứa vào một file được gọi là file đối tượng, trong file đối tượng này còn chứa cả các thông tin về sự liên kết của các thủ tục.
- Sau khi biên dịch các thủ tục của hệ điều hành không được liên kết lại, mà hệ

thống chỉ tạo ra file hoặc một bảng chỉ mục để chứa thông tin của các thủ tục hệ điều hành, mỗi phần tử trong bảng chỉ mục chứa một con trỏ trỏ tới thủ tục tương ứng, con trỏ này dùng để gọi thủ tục khi cần thiết. Ta có thể xem cách gọi ngắt (Interrupt) trong ngôn ngữ lập trình cấp thấp và cách thực hiện đáp ứng ngắt dựa vào bảng vector ngắt trong MS_DOS là một ví dụ cho cấu trúc này.



Trong cấu trúc đơn giản của hệ thống đơn khối, HĐH là một tập các thủ tục, có thể gọi lẫn nhau. Các thủ tục được chia thành 3 lớp:

1. Một tập các thủ tục chính (chương trình của người sử dụng) gọi đến một thủ tục dịch vụ của hệ điều hành. Lời gọi này được gọi là lời gọi hệ thống.

2. Một tập các thủ tục dịch vụ (service) để đáp ứng những lời gọi hệ thống từ các chương trình người sử dụng.

3. Một tập các thủ tục tiện ích (utility) hỗ trợ cho các thủ tục dịch trong việc thực hiện cho các lời gọi hệ thống.

Nhận xét:

- Chương trình của người sử dụng có thể truy xuất trực tiếp đến các chi tiết phần cứng bằng cách gọi một thủ tục cấp thấp, gây khó khăn cho hệ điều hành trong việc kiểm soát và bảo vệ hệ thống.

- Các thủ tục dịch vụ mang tính chất tĩnh, nó chỉ hoạt động khi được gọi bởi chương trình của người sử dụng, điều này làm cho hệ điều hành thiếu chủ động trong việc quản lý môi trường.

1.3.2.2. Các hệ thống phân lớp (Layered Systems)

Hệ thống được chia thành một số lớp, mỗi lớp được xây dựng dựa vào lớp bên trong. Lớp trong cùng thường là phần cứng, lớp ngoài cùng là giao diện với người sử dụng.

Mỗi lớp là một đối tượng trừu tượng, chứa dựng bên trong nó các dữ liệu và thao tác xử lý dữ liệu đó. Lớp n chứa dựng một cấu trúc dữ liệu và các thủ tục có thể được gọi bởi lớp n+1 hoặc ngược lại có thể gọi các thủ tục ở lớp n-1.

Ví dụ về một hệ điều hành phân lớp:

Lớp 5: Chương trình ứng dụng

Lớp 4: Quản lý bộ đệm cho các thiết bị xuất nhập

Lớp 3: Trình điều khiển thao tác (console – giao diện điều khiển)

Lớp 2: Quản lý bộ nhớ

Lớp 1: Điều phối processor

Lớp 0: Phần cứng hệ thống

Chú ý: Khái niệm lớp liên quan đến các mức giao tiếp trong hệ thống máy tính:

- Người sử dụng
- Chương trình ứng dụng
- Dịch vụ hệ thống
- Nhân
- Phần cứng máy tính

Nhận xét:

- Các thủ tục (lớp) trong HĐH có tính module giúp đơn giản hóa việc gỡ rối và kiểm tra hệ thống. Việc thiết kế và cài đặt hệ thống được đơn giản hóa khi hệ thống được phân chia thành nhiều lớp.
- Mỗi lớp che giấu sự tồn tại của cấu trúc dữ liệu, thao tác và phần cứng từ các lớp cấp cao hơn.
- Khi xây dựng hệ điều hành, khó xác định được số lượng lớp, thứ tự và chức năng của mỗi lớp.
- Hiệu quả của hệ thống không cao do tại mỗi lớp phải thêm chi phí cho lời gọi hệ thống; thời gian thực sự lời gọi hệ thống lâu hơn khi được thực hiện trên hệ thống không phân tầng.

1.3.2.3. Máy ảo (Virtual Machine)

Các máy ảo là những bản sao ảo chính xác các đặc tính phần cứng của máy tính thực sự và cho phép một hệ điều hành khác hoạt động trên đó như trên phần cứng thực sự. Phần nhân hệ thống thực hiện giám sát máy ảo chịu trách nhiệm giao tiếp với phần cứng và cho phép khả năng đa chương bằng cách cung cấp nhiều máy ảo cho các lớp bên trên.

Mục đích của việc sử dụng máy ảo là xây dựng các hệ thống đa chương với nhiều tiến trình thực hiện đồng thời, mỗi tiến trình được cung cấp một máy ảo với đầy đủ tài nguyên ảo, để nó thực hiện được.

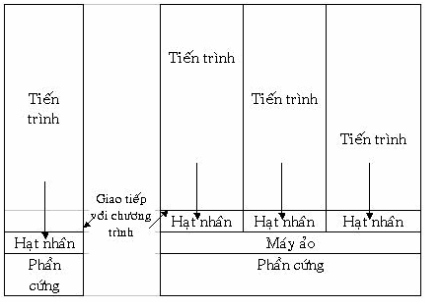
Trong cấu trúc này phần nhân của hệ thống trở thành bộ phận tổ chức giám sát máy ảo, phần này chịu trách nhiệm giao tiếp với phần cứng, chia sẻ tài nguyên hệ thống để tạo ra nhiều máy ảo, hoạt động độc lập với nhau, để cung cấp cho lớp trên.

Tài nguyên của hệ thống được chia xẻ để tạo những máy ảo. Lập lịch BXL chia xẻ BXL cho các người sử dụng. Spooling và hệ thống tập tin được chia thành những card đọc ảo và máy in ảo. Một terminal cung cấp các chức năng tạo các thao tác màn hình ảo. Xây dựng hệ thống đĩa ảo cho các máy ảo.

Trong các máy ảo, các tiến trình phức sẽ thực hiện trên một bộ xử lý và bộ nhớ riêng. Nhưng các tiến trình này có các lời gọi hệ thống và hệ thống tập tin không được cung cấp trực tiếp từ phần cứng.

Ở đây cần phân biệt sự khác nhau giữa máy ảo và máy tính mở rộng, máy ảo là bản sao chính xác các đặc tính phần cứng của máy tính thực sự và cho phép HĐH hoạt động trên nó, sau đó HĐH xây dựng máy tính mở rộng để cung cấp cho người sử dụng.

Hình vẽ trên đây cho chúng ta thấy sự khác nhau trong hệ thống không có máy ảo và hệ thống có máy ảo:



Nhận xét:

- Việc cài đặt các phần mềm giả lập phần cứng để tạo ra máy ảo thường rất khó khăn và phức tạp.
- Vấn đề bảo vệ tài nguyên hệ thống và tài nguyên đã cấp phát cho các tiến trình, sẽ trở nên đơn giản hơn vì mỗi tiến trình thực hiện trên một máy tính (ảo) độc lập với nhau nên việc tranh chấp tài nguyên là không thể xảy ra.
- Nhờ hệ thống máy ảo mà một ứng dụng được xây dựng trên hệ điều hành có thể hoạt động được trên hệ điều hành khác.

1.3.2.4. Mô hình Client/ Server (client/ server model)

Các hệ điều hành hiện đại thường chuyển dần các tác vụ của hệ điều hành ra các lớp bên ngoài nhằm thu nhỏ phần cốt lõi của hệ điều hành thành hạt nhân cực tiểu (kernel) sao cho chỉ phần hạt nhân này phụ thuộc vào phần cứng. Để thực hiện được điều này hệ điều hành xây dựng theo mô hình Client/ Server, theo mô hình này hệ điều hành bao gồm nhiều tiến trình đóng vai trò Server có các chức năng chuyên biệt như quản lý tiến trình, quản lý bộ nhớ, ..., phần hạt nhân cuả hệ điều hành chỉ thực hiện nhiệm vụ tạo cơ chế thông tin liên lạc giữa các tiến trình Client và Server.

Như vậy các tiến trình trong hệ thống được chia thành 2 loại:

- Tiến trình bên ngoài hay tiến trình của chương trình người sử dụng được gọi là các tiến trình Client.
- Tiến trình của hệ điều hành được gọi là tiến trình Server.

Khi cần thực hiện một chức năng hệ thống các tiến trình Client sẽ gửi yêu cầu tới tiến trình server tương ứng, tiến trình server sẽ xử lý và trả lời kết quả cho tiến trình Client.

Nhận xét:

- Hệ thống này dễ thay đổi và dễ mở rộng hệ điều hành. Để thay đổi các chức năng của hệ điều hành chỉ cần thay đổi ở server tương ứng, để mở rộng hệ điều hành chỉ cần thêm các server mới vào hệ thống.
- Các tiến trình Server của hệ điều hành hoạt động trong chế độ không đặc quyền nên không thể truy cập trực tiếp đến phần cứng, điều này giúp hệ thống được bảo vệ tốt hơn.

1.4. PHÂN LOẠI HỆ ĐIỀU HÀNH

Có nhiều cách khác nhau để phân loại hệ điều hành, theo cách thực hiện các công việc, các tác vụ, các tiến trình của người sử dụng để phân loại hệ điều hành.

1.4.2. Hệ điều hành đơn chương trình

- Chỉ phục vụ cho một chương trình
- HĐH chỉ phục vụ việc vào-ra liên quan đến chương trình được thực hiện thông qua đĩa từ.

1.4.3. Hệ điều hành đa chương trình

- Tại mỗi thời điểm tồn tại nhiều chương trình (của người sử dụng và của hệ thống) tại BN trong, đòi hỏi chia sẻ các tài nguyên hệ thống
- Trong đa chương trình, tài nguyên hệ thống quan trọng nhất chia sẻ là BXL.

1.4.3.1. Xử lý theo lô (mẻ) đơn giản

Xử lý theo lô (đơn giản) là các tác vụ trong hàng đợi tác vụ được thực hiện một cách tự động và liên tiếp theo những chỉ thị đã được xác định trước, không cần có sự can thiệp từ bên ngoài. Các chương trình và dữ liệu của mỗi tác vụ được lưu vào hàng đợi tương ứng với thứ tự thực hiện của tác vụ đó. Kết quả xử lý được cho ra liên tiếp theo thứ tự thực hiện

Hệ điều hành có bộ phận thường trú trong bộ nhớ chính để giám sát việc thực hiện của các tác vụ trong hệ thống.

Xử lý theo lô không cho phép thay đổi chương trình và dữ liệu của các tác vụ ngay cả khi chúng còn nằm trong hàng đợi. Mặt khác trong quá trình thực hiện tác vụ nếu tác vụ chuyển sang truy xuất trên thiết bị vào/ra thì processor rơi vào trạng thái chờ, gây lãng phí thời gian xử lý của processor.

1.4.3.2. Hệ điều hành xử lý theo lô đa chương

Xử lý theo lô đa chương là khả năng thực hiện đồng thời nhiều tác vụ, nhiều chương trình. Các tác vụ đều ở trạng thái sẵn sàng (danh sách sẵn sàng). Hệ điều hành chỉ nạp một phần code và data của các tác vụ vào bộ nhớ (các phần còn lại sẽ được nạp sau tại thời điểm thích hợp). Mỗi tác vụ được thực hiện trong một khoảng thời gian nào đó cho đến khi có các yêu cầu vào/ra. Khi tác vụ đang thực hiện cần truy xuất vào/ra thì processor sẽ được chuyển sang phục vụ cho tác vụ khác. Cứ như vậy, HĐH tổ chức chuyển hướng processor để phục vụ hết các phần tác vụ trong bộ nhớ cũng như các tác vụ mà hệ thống yêu cầu.

MFT Multiprogramming with Fixed number of Tasks: quy định sẵn số lượng các bài toán đồng thời tại BN chính.

MVT Multiprogramming with Variable number of Tasks: các bài toán được cấp phát liên tiếp trong BN chính.

Xử lý theo lô đa chương cho phép tiết kiệm bộ nhớ và hạn chế thời gian rỗi của processor. Tuy nhiên cần chi phí cao cho việc lập lịch processor, là lựa chọn tác vụ nào trong số các tác vụ đang ở trạng thái sẵn sàng để cung cấp processor cho nó. Ngoài ra hệ

điều hành còn phải giải quyết việc chia sẻ bộ nhớ chính cho các tác vụ khác nhau.

1.4.3.3. Hệ điều hành phân chia thời gian (TSS - Time Shared System)

Khái niệm chia sẻ thời gian ra đời đã đánh dấu một bước phát triển mới của hệ điều hành trong việc điều khiển các hệ thống đa người dùng. Chia sẻ thời gian là chia sẻ thời gian phục vụ của processor cho các tác vụ, các tiến trình đang ở trong trạng thái sẵn sàng.

Nguyên tắc của hệ điều hành chia sẻ thời gian tương tự như trong hệ điều hành xử lý theo lô đa chương nhưng việc chuyển processor từ tác vụ, tiến trình này sang tác vụ, tiến trình khác không phụ thuộc vào việc tác vụ, tiến trình hiện tại có truy xuất vào/ra hay không mà chỉ phụ thuộc vào sự điều phối processor của hệ điều hành.

Thời gian chuyển đổi processor giữa các tác vụ là rất nhỏ nên coi như các tác vụ được thực hiện đồng thời.

Hệ điều hành chia sẻ thời gian là mở rộng logic của hệ điều hành đa chương và nó thường được gọi là hệ điều hành đa nhiệm (Multitasking).

1.4.3.4. Hệ điều hành xử lý thời gian thực

Xử lý theo thời gian thực là sau mỗi tác vụ, kết quả thực hiện mỗi tác vụ được cho ra tức thời, chính xác.

Trong hệ điều hành này các tác vụ cần thực hiện không được đưa vào hàng đợi mà được xử lý tức thời và trả lại ngay kết quả hoặc thông báo lỗi cho người sử dụng có yêu cầu. Hệ điều hành này hoạt động đòi hỏi sự phối hợp cao giữa phần mềm và phần cứng.

Ngoài ra, còn có thể phân loại HĐH theo cấu trúc, theo phạm vi áp dụng, theo tên các nhà cung cấp, theo mã nguồn viết HĐH…

Hệ điều hành đa vi xử lý: Là các hệ điều hành dùng để điều khiển sự hoạt động của các hệ thống máy tính có nhiều vi xử lý. Các hệ điều hành đa vi xử lý (multiprocessor) gồm có 2 loại:

- **Đa xử lý đối xứng (SMP: symmetric):** vi xử lý bất kỳ cũng có thể chạy một loại tiểu trình bất kỳ, các vi xử lý giao tiếp với nhau thông qua một bộ nhớ dùng chung. Hệ SMP cung cấp một cơ chế chịu lỗi và khả năng cân bằng tải tối ưu hơn, nguy cơ xảy ra tình trạng bế tắcở BXL giảm đi đáng kể. Vấn đề đồng bộ giữa các vi xử lý được đặt lên hàng đầu khi thiết kế hệ điều hành cho hệ thống SMP. Hệ điều hành Windows NT, hệ điều hành Windows 2000 là các hệ điều hành đa xử lý đối xứng.
- **Đa xử lý bất đối xứng (ASMP: asymmetric):** có một hoặc hai vi xử lý để sử dụng riêng, các vi xử lý còn lại dùng để điều khiển các chương trình của người sử dụng. Hệ ASMP đơn giản hơn nhiều so với hệ SMP, nhưng trong hệ này nếu có một vi xử lý trong các vi xử lý dành riêng cho hệ điều hành bị hỏng thì hệ thống có thể ngừng hoạt động.

Hệ điều hành mạng: Là các hệ điều hành dùng để điều khiển sự hoạt động của mạng máy tính. Ngoài các chức năng cơ bản của một hệ điều hành, các hệ điều hành mạng còn phải thực hiện việc chia sẻ và bảo vệ tài nguyên của mạng. Hệ điều hành Windows 9x/NT, Windows 200, Linux, là các hệ điều hành mạng máy tính.

Tóm lại: các HĐH ra đời sau luôn tìm cách khắc phục các hạn chế của HĐH trước đó và phát triển nhiều hơn nữa để đáp ứng yêu cầu ngày càng cao của của người sử dụng và chương trình người sử dụng, cũng như khai thác tối đa các chức năng của phần cứng máy tính để nâng cao hiệu suất của hệ thống. Nhưng chức năng của HĐH càng cao thì chi phí cho nó cũng tăng theo và cấu trúc của HĐH cũng sẽ phức tạp hơn.

1.5. CÁC TÍNH CHẤT CỦA HỆ ĐIỀU HÀNH

Các tính chất cơ bản của HĐH là không phụ thuộc vào máy tính cụ thể, vào đối tượng phục vụ, vào phiên bản thể hiện.

- *Độ tin cậy cao*:

+ Mọi hoạt động, mọi thông báo của HĐH đều phải chuẩn xác tuyệt đối (chỉ được cung cấp thông tin đúng cho người sử dụng, phải có các phương tiện hỗ trợ kiểm tra tính đúng đắn của dữ liệu trong các phép lưu trữ và xử lý).

+ HĐH thông báo lỗi và ngừng xử lý hoặc trao quyền quyết định cho thao tác viên hoặc người sử dụng khi hệ thống bị lỗi hoặc không thể xác định được tính đúng đắn của thông tin.

- *An toàn* là mức độ bảo vệ khác nhau đối với dữ liệu, chương trình và các tài nguyên trong mọi trường hợp và trong mọi chế độ hoạt động. (Chú ý, đặc biệt trong hệ thống đa nhiệm).

- *Hiệu suất (hiệu quả)* là các tài nguyên của hệ thống phải được khai thác triệt để:

+ Trong điều kiện tài nguyên hạn chế, hệ thống vẫn phải giải quyết được các yêu cầu phức tạp.

+ Đảm bảo tính đồng bộ của toàn hệ thống, không để các thiết bị tốc độ chậm trì hoãn hoạt động của toàn hệ thống.

- *Chuẩn và hệ thống mở (kế thừa)* là khả năng đảm bảo những tính năng của các hệ thống trước đó, đồng thời có khả năng thích nghi với những thay đổi có thể có trong tương lai (tích hợp được các ứng dụng, tương thích). Đây là tính chất rất quan trọng và là bắt buộc đối với mọi HĐH, đặc biệt với các HĐH thế hệ mới. Do vậy, việc thiết kế HĐH phải được tuân thủ theo 1 số nguyên tắc nhất định.

- *Thuận tiện* là mức độ đáp ứng các yêu cầu khác nhau đối với nhiều người sử dụng khác nhau:

+ hệ thống phải dễ dàng sử dụng, có nhiều mức hiệu quả khác nhau tuỳ theo kiến thức và kinh nghiệm của người dùng.

+ hệ thống trợ giúp phong phú cho người sử dụng trong quá trình khai thác.

- *Tính thương mại*
- *Khả năng bảo trì*
- *Bảo vệ và an ninh*

→ HĐH phải dung hoà, có ưu tiên hợp lý các tính chất trên.

1.6. NGUYÊN TẮC TỔ CHỨC XÂY DỰNG CÁC THÀNH PHẦN CỦA HĐH

- Nguyên tắc modul:

+ hệ thống có tính chất modul là hệ thống được xây dựng từ những modul độc lập và tồn tại bộ quy tắc liên kết chúng thành hệ thống có tổ chức.

+ các dạng modul: modul chức năng và modul chương trình

+ quan hệ giữa các modul: thông qua dữ liệu vào và ra

+ tính phân cấp của các modul: các modul con khi liên kết sẽ tạo thành một modul lớn hơn (mức cao hơn) để giải quyết những vấn đề lớn hơn, phức tạp hơn.

+ nguyên tắc modul cho phép tổ hợp những modul hiện có theo nhiều cách khác nhau, đảm bảo tính đa dạng chức năng của hệ thống.

- Nguyên tắc tương đối trong định vị:

+ các modul chương trình có địa chỉ tương đối trong BN (địa chỉ tính từ đầu BN), khi thực hiện chúng mới được định vị tại vùng BN cụ thể.

+ nguyên tắc này cho phép sử dụng BN một cách linh hoạt và HĐH không bị phụ thuộc vào cấu hình BN cụ thể.

- Nguyên tắc Macroprocessor:

+ mỗi macroprocessor thực hiện một yêu cầu cụ thể.

+ khi có nhiệm vụ cụ thể, hệ thống sẽ xây dựng các phiếu yêu cầu, liệt kê các bước phải thực hiện

+ xây dựng chương trình trên cơ sở phiếu yêu cầu và tổ chức thực hiện chương trình đó

+ nguyên tắc này làm cho quá trình đối thoại được linh hoạt, không cần chương trình dịch phức tạp (thường dùng trong các hệ quản trị CSDL).

- Nguyên tắc khởi tạo trong cài đặt:

+ chương trình được viết đầy đủ và được viết dưới dạng các modul (nguyên lý macroprocessor) - phiên bản đầy đủ.

+ người sử dụng khi cài đặt sẽ tạo phiên bản mới thích hợp (tối ưu cả về cấu trúc và phương thức hoạt động) bằng cách loại bỏ những modul không cần thiết.

+ nguyên tắc cho phép đảm bảo tính thuận tiện của HĐH.

- Nguyên tắc lặp chức năng:

+ một công việc được thực hiện bằng nhiều cách khác nhau với những tổ hợp modul khác nhau.

+ trong hệ thống tồn tại nhiều modul khác nhau cùng giải quyết một vấn đề (nhiều chương trình dịch cho một ngôn ngữ thuật toán nào đó)

+ người sử dụng chủ động lựa chọn giải thuật tối ưu

+ nguyên tắc bảo đảm độ an toàn cao cho hệ thống (hệ thống hoạt động bình thường ngay cả khi thiếu hoặc hỏng 1 số thành phần của hệ thống); cho phép người sử dụng thuận tiện khai thác hệ thống, khai thác hết các tính năng của hệ thống và lựa chọn được giải thuật tối ưu.

- Nguyên tắc giá trị chuẩn:

+ mỗi modul, câu lệnh… có thể có nhiều giá trị tham số.

+ hệ thống chuẩn bị sẵn bộ các giá trị tham số ứng với trường hợp thường gặp nhất (giá trị chuẩn).

+ khi thực hiện, nếu lời gọi modul hay câu lệnh thiếu tham số nào thì hệ thống sẽ bổ sung bằng giá trị quy định trước.

+ nguyên tắc đảm bảo tính thuận tiện

- Nguyên tắc bảo vệ nhiều mức:

+ tổ chức nhiều mức bảo vệ đối với dữ liệu và chương trình.

+ các mức bảo vệ: bảo vệ tại file, tại thư mực, tại ổ đĩa…; bảo vệ thường xuyên hoặc bảo vệ trong từng chế độ làm việc.

+ áp dụng cho cả các thông tin ghi trong RAM.

+ nguyên tắc cho phép đảm bảo an toàn hệ thống và an toàn dữ liệu

1.7. LỊCH SỬ PHÁT TRIỂN CỦA HỆ ĐIỀU HÀNH

Thế hệ 1 (1945 - 1955)

Vào những năm 1950 máy tính dùng đèn điện tử chân không ra đời. Mỗi máy tính được một nhóm người thực hiện, bao gồm việc thiết kế, xây dựng chương trình, thao tác, quản lý,...

Người lập trình phải dùng ngôn ngữ máy tuyệt đối để lập trình. Khái niệm ngôn ngữ lập trình và hệ điều hành chưa được biết đến trong khoảng thời gian này.

Thế hệ 2 (1955 - 1965)

Máy tính dùng bán dẫn ra đời và được sản xuất để cung cấp cho khách hàng. Bộ phận sử dụng máy tính được phân chia rõ ràng: người thiết kế, người xây dựng, người vận hành, người lập trình, và người bảo trì. Ngôn ngữ lập trình là Assembly và Fortran. Để thực hiện một thao tác, lập trình viên viết chương trình trên phiếu đục lỗ, sau đó đưa phiếu vào máy, máy thực hiện cho kết quả ở máy in.

Hệ thống xử lý theo lô, cơ chế xử lý song song cũng ra đời trong thời kỳ này. Các thao tác cần thực hiện trên máy tính được ghi trước trên băng từ, hệ thống sẽ đọc băng từ, thực hiện lần lượt và cho kết quả ở băng từ xuất. Hệ thống xử lý theo lô hoạt động dưới sự điều khiển của một chương trình đặc biệt chính là hệ điều hành sau này.

Thế hệ 3 (1965 - 1980)

Máy IBM 360 dựa trên các vi mạch, thiết kế cho cả tính toán trong thương mại và tính toán trong khoa học, được sản xuất hàng loạt để tung ra thị trường. Các thiết bị ngoại vi xuất hiện ngày càng nhiều, các thao tác điều khiển máy tính và thiết bị ngoại vi ngày càng phức tạp hơn. Trước tình hình này, hệ điều hành đã ra đời cho phép sử dụng chung trên tất cả các máy tính của nhà sản xuất và người sử dụng.

Hệ điều hành ra đời nhằm điều phối, kiểm soát hoạt động của hệ thống và giải quyết các yêu cầu tranh chấp thiết bị. Hệ điều hành đầu tiên được viết bằng ngôn ngữ Assembly. Hệ điều hành xuất hiện khái niệm đa chương, khái niệm chia sẻ thời gian và kỹ thuật Spool. Trong giai đoạn này cũng xuất hiện các hệ điều hành Multics và Unix.

Thế hệ 4 (từ 1980)

Máy tính cá nhân ra đời. Trong thời kỳ này ra đời các hệ điều hành MS-DOS (gắn liền với máy tính IBM-PC), hệ điều hành mạng và hệ điều hành phân tán.

- Đa số các hệ điều hành đều được xây dựng từ ngôn ngữ lập trình cấp thấp trừ hệ điều hành Unix, nó được xây dựng từ C, một ngôn ngữ lập trình cấp cao.
- Nếu không có hệ điều hành thì việc khai thác và sử dụng máy tính sẽ khó khăn và phức tạp rất nhiều và không phải bất kỳ ai cũng có thể sử dụng máy tính được.
- Sự ra đời và phát triển của hệ điều hành gắn liền với sự phát triển của máy tính,

và ngược lại sự phát triển của máy tính kéo theo sự phát triển của hệ điều hành. Hệ điều hành thực sự phát triển khi máy tính PC xuất hiện trên thị trường.

1.8. GIỚI THIỆU MỘT SỐ HỆ ĐIỀU HÀNH

1.8.2. Hệ điều hành Windows 95

1.8.3. Hệ điều hành Windows 2000

1.8.4. Hệ điều hành Linux

Sinh viên tự tìm hiểu các nội dung trên.

Chương 2
QUẢN LÝ TIẾN TRÌNH

- Các chương trình (người sử dụng, hệ thống) bao gồm nhiều công việc, liên quan đến các tài nguyên (bộ nhớ, bộ xử lý…);

- Nhiều chương trình yêu cầu được thực hiện đồng thời.

- Vấn đề quản lý tiến trình liên quan đến cả quản lý bộ nhớ, bộ xử lý và các tài nguyên khác; vấn đề đồng bộ các tiến trình và các hiện tượng tranh chấp, bế tắc do phân phối tài nguyên gây ra.

- Hệ điều hành phải cho phép thực hiện nhiều tiến trình đồng thời để khai thác tối đa thời gian xử lý của processor nhưng cũng cung cấp được thời gian hồi đáp hợp lý.

- Hệ điều hành phải cấp phát tài nguyên để tiến trình hoạt động một cách hiệu quả với một chính sách hợp lý nhưng không xảy ra tình trạng bế tắctrong hệ thống.

- Hệ điều hành có thể được yêu cầu để hỗ trợ truyền thông liên tiến trình và người sử dụng tạo ra tiến trình.

Hệ điều hành phải có nhiệm vụ tạo ra tiến trình, điều khiển sự hoạt động của tiến trình và kết thúc tiến trình.

Một số hệ điều hành phân biệt hai khái niệm tiến trình và tiểu trình. Tiến trình liên quan đến quyền sở hữu tài nguyên, tiểu trình liên quan đến sự thực hiện chương trình.

Trong các hệ điều hành đa chương, có nhiều tiến trình tồn tại trên bộ nhớ chính, các tiến trình này luân phiên giữa hai trạng thái: sử dụng processor và đợi thực hiện vào/ra hay một vài sự kiện nào đó xảy ra.

2.1. TỔNG QUAN VỀ TIẾN TRÌNH

2.1.1. Tiến trình và phân loại tiến trình

2.1.1.1. Tiến trình (process)

Tiến trình là một bộ phận của một chương trình đang thực hiện, đơn vị thực hiện tiến trình là processer. Tiến trình là một bộ phận của chương trình sở hữu một con trỏ lệnh, một con trỏ stack, một tập các thanh ghi, một không gian địa chỉ trong bộ nhớ chính và tất cả các thông tin cần thiết khác để tiến trình có thể hoạt động được. Tóm lại, tiến trình là tất cả những gì liên quan đến chương trình được thực hiện tại bộ xử lý.

Việc đưa ra tiến trình cho phép máy tính có thể thực hiện nhiều tác vụ đồng thời (chuyển đổi BXL qua lại để duy trì hoạt động của nhiều chương trình cùng lúc). Khi đó, mọi chương trình trong hệ thống được tổ chức thành những tiến trình. Mỗi tiến trình khi hoạt động cần có một số tài nguyên như BXL, BN chính, các thiết bị nhập xuất và một số yếu tố khác.

Trong quá trình hoạt động của tiến trình, quá trình chuyển từ trạng thái này sang trạng thái khác không phải do chính bản thân tiến trình mà là do sự tác động từ bên ngoài, cụ thể ở đây là bộ phận điều phối tiến trình của hệ điều hành.

2.1.1.2. Phân loại tiến trình

Các tiến trình trong hệ thống theo quan hệ về thời gian tồn tại có thể chia thành hai loại: tiến trình tuần tự và tiến trình song song. Tiến trình tuần tự là các tiến trình mà điểm khởi tạo của nó sau điểm kết thúc của tiến trình trước đó. Tiến trình song song (đồng thời) là các tiến trình mà điểm khởi tạo của tiến trình này nằm ở giữa của các tiến trình khác, tức là có thể khởi tạo một tiến trình mới khi các tiến trình trước đó chưa kết thúc. Tiến trình song song được chia thành nhiều loại:

Tiến trình song song độc lập: là các tiến trình hoạt động song song nhưng không có quan hệ thông tin với nhau, trong trường hợp này HĐH phải thiết lập cơ chế bảo vệ dữ liệu của các tiến trình, và cấp phát tài nguyên cho các tiến trình một cách hợp lý.

Những tiến trình song song có quan hệ bao gồm:

Tiến trình song song có quan hệ thông tin: trong quá trình hoạt động các tiến trình thường trao đổi thông tin với nhau, trong một số trường hợp tiến trình gửi thông báo cần phải nhận được tín hiệu từ tiến trình nhận để tiếp tục, điều này dễ dẫn đến bế tắc khi tiến trình nhận tín hiệu không ở trong trạng thái nhận hay tiến trình gửi không ở trong trạng thái nhận thông báo trả lời.

Tiến trình song song phân cấp: trong quá trình hoạt động một tiến trình có thể khởi tạo các tiến trình khác hoạt động song song với nó, tiến trình khởi tạo được gọi là tiến trình cha, tiến trình được tạo gọi là tiến trình con. Trong mô hình này HĐH phải giải quyết vấn đề cấp phát tài nguyên cho các tiến trình con. Tiến trình con nhận tài nguyên ở đâu, từ tiến trình cha hay từ hệ thống. Để giải quyết vấn đề này HĐH đưa ra 2 mô hình quản lý tài nguyên: Thứ nhất, mô hình tập trung, trong mô hình này hệ điều hành chịu trách nhiệm phân phối tài nguyên cho tất cả các tiến trình trong hệ thống. Thứ hai, mô hình phân tán, trong mô hình này HĐH cho phép tiến trình con nhận tài nguyên từ tiến trình cha, tức là tiến trình khởi tạo có nhiệm vụ nhận tài nguyên từ HĐH để cấp phát cho các tiến trình mà nó tạo ra, và nó có nhiệm vụ thu hồi lại tài nguyên đã cấp phát trả về cho HĐH trước khi kết thúc.

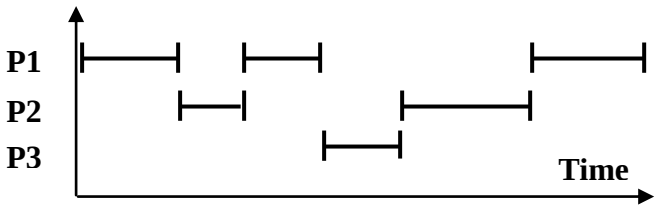
Tiến trình song song đồng mức: là các tiến trình hoạt động song song sử dụng chung tài nguyên theo nguyên tắc lần lượt, mỗi tiến trình sau một khoảng thời gian chiếm giữ tài nguyên phải tự động trả lại tài nguyên cho tiến trình kia.

Các tiến trình tuần tự chỉ xuất hiện trong các HĐH đơn nhiệm đa chương, như HĐH MS_DOS, loại tiến trình này tồn tại nhiều hạn chế, điển hình nhất là không khai thác tối đa thời gian xử lý của processor.

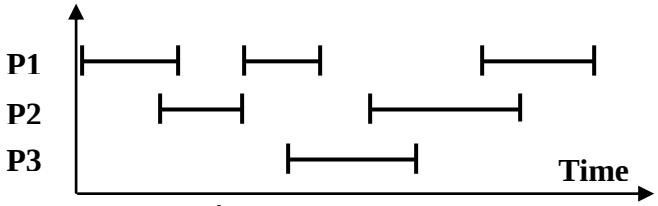
Các tiến trình song song xuất hiện trong các HĐH đa nhiệm đa chương, trên cả hệ thống uniprocessor và multiprocessor. Nhưng sự song song thực, chỉ có ở các hệ thống multiprocessor, trong hệ thống này mỗi processor chịu trách nhiệm thực hiện một tiến trình. Sự song song trên các hệ thống uniprocessor là sự song song giả, các tiến trình song song trên hệ thống này thực chất là các tiến trình thay nhau sử dụng processor, tiến trình này đang chạy thì có thể dừng lại để nhường processor cho tiến trình khác chạy và

sẽ tiếp tục lại sau đó khi có được processor. Đây là trường hợp mà ở trên ta cho rằng: điểm khởi tạo của tiến trình này nằm ở thân của tiến trình khác.

Hình vẽ sau đây minh họa sự khác nhau, về mặt thực hiện, giữa các tiến trình song song/ đồng thời trong hệ thống uniprocessor với các tiến trình song song/ đồng thời trong hệ thống multiprocessor.



a. Trong hệ thống uniprocessor



b. Trong hệ thống Multiprocessor

Hình 2.1: Sự thực hiện đồng thời của các tiến trình trong hệ thống uniprocessor (a) và hệ thống multiprocessor (b).

Trong tài liệu này chúng ta chỉ khảo sát sự hoạt động của các tiến trình song song (hay đồng thời) trên các hệ thống uniprocessor.

Đối với người sử dụng thì trong hệ thống chỉ có hai nhóm tiến trình: các tiến trình của HĐH và các tiến trình của chương trình người sử dụng. Các tiến trình của HĐH hoạt động trong chế độ đặc quyền, nhờ đó mà nó có thể truy xuất vào các vùng dữ liệu được bảo vệ của hệ thống. Trong khi đó các tiến trình của chương trình người sử dụng hoạt động trong chế độ không đặc quyền, nên nó không thể truy xuất vào hệ thống, nhờ đó mà HĐH được bảo vệ. Các tiến trình của chương trình người sử dụng có thể truy xuất vào hệ thống thông qua các tiến trình của HĐH bằng cách thực hiện một lời gọi hệ thống.

2.1.2. Mô hình tiến trình (bộ xử lý vật lý và bộ xử lý logic)

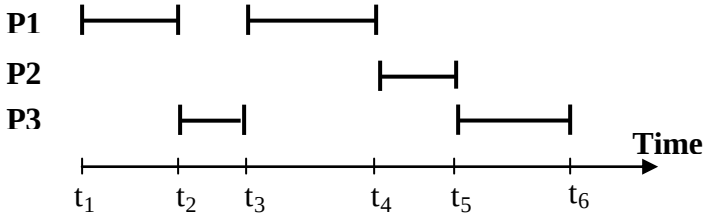
Đa số các hệ thống, có thể có nhiều chương trình hoạt động đồng thời (co-occurrence). Về nguyên tắc, để thực hiện được điều này thì hệ thống phải có nhiều processor, mỗi processor thực hiện một chương trình, nhưng mong muốn của HĐH cũng như người sử dụng là *thực hiện sự đa chương trên các hệ thống chỉ có một processor*.

Để thực hiện được điều này HĐH đã sử dụng mô hình tiến trình để tạo ra sự song song giả hay tạo ra các processor logic từ processor vật lý. Các processor logic có thể hoạt động song song với nhau, mỗi processor logic chịu trách nhiệm thực hiện một tiến trình.

Trong mô hình tiến trình HĐH khởi tạo và đưa vào hệ thống nhiều tiến trình của một chương trình hoặc của nhiều chương trình khác nhau, cấp phát đầy đủ tài nguyên (trừ processor) cho tiến trình và đưa các tiến trình sang trạng thái sẵn sàng. Hệ điều hành bắt

đầu cấp processor cho một tiến trình trong số các tiến trình ở trạng thái sẵn sàng để tiến trình này hoạt động, sau một khoảng thời gian nào đó HĐH thu hồi processor của tiến trình này để cấp cho một tiến trình sẵn sàng khác, sau đó HĐH lại thu hồi processor từ tiến trình mà nó vừa cấp để cấp cho tiến trình khác, có thể là tiến trình mà trước đây bị HĐH thu hồi processor khi nó chưa kết thúc, và cứ như thế cho đến khi tất cả các tiến trình mà HĐH khởi tạo đều hoạt động và kết thúc được. Khoảng thời gian chuyển processor từ tiến trình này sang tiến trình khác hay khoảng thời gian giữa hai lần được cấp phát processor của một tiến trình là rất nhỏ nên các tiến trình có cảm giác luôn được sở hữu processor (logic) hay hệ thống có cảm giác các tiến trình/ chương trình hoạt động song song nhau. Hiện tượng này được gọi là sự song song giả.

Giả sử trong hệ thống có 3 tiến trình sẵn sàng P_1, P_2, P_3 thì quá trình chuyển processor giữa 3 tiến trình này có thể minh họa như sau:



Hình 2.2: Sự hoạt động "song song" của các tiến trình P_1, P_2, P_3

Rõ ràng với mô hình tiến trình hệ thống có được 2 điều lợi:

- Tiết kiệm được bộ nhớ: vì không phải nạp tất cả chương trình vào bộ nhớ mà chỉ nạp các tiến trình cần thiết nhất, sau đó tùy theo yêu cầu mà có thể nạp tiếp các tiến trình khác.

- Cho phép các chương trình hoạt động song song nên tốc độ xử lý của toàn hệ thống tăng lên và khai thác tối đa thời gian xử lý của processor.

Việc chọn thời điểm dừng của tiến trình đang hoạt động (đang chiếm giữ processor) để thu hồi processor chuyển cho tiến trình khác hay việc chọn tiến trình tiếp theo nào trong số các tiến trình đang ở trạng thái sẵn sàng để cấp processor là những vấn đề khá phức tạp đòi hỏi HĐH phải có một cơ chế điều phối thích hợp thì mới có thể tạo ra được hiệu ứng song song giả và sử dụng tối ưu thời gian xử lý của processor. Bộ phận thực hiện chức năng này của HĐH được gọi là bộ điều phối (dispatcher) tiến trình.

2.1.3. Tiểu trình và đa tiểu trình (luồng và đa luồng)

2.1.3.1. Tiểu trình (thread)

Thông thường mỗi tiến trình có một không gian địa chỉ và chỉ có một dòng xử lý. Nhưng trong thực tế có một số ứng dụng cần nhiều dòng xử lý cùng chia sẻ một không gian địa chỉ tiến trình (chung tài nguyên), các dòng xử lý này có thể hoạt động song song với nhau như các tiến trình độc lập trên hệ thống. Để thực hiện được điều này các HĐH hiện nay đưa ra một cơ chế thực thi (các chỉ thị trong chương trình) mới, được gọi là tiểu trình.

Tiểu trình là một đơn vị xử lý cơ bản trong hệ thống. Mỗi tiểu trình xử lý tuần tự đoạn code của nó, sỡ hữu một con trỏ lệnh, tập các thanh ghi và một vùng nhớ stack riêng (ngăn xếp). Các tiểu trình chia sẻ BXL với nhau giống như cách chia sẻ giữa các tiến trình: một tiểu trình xử lý trong khi các tiểu trình khác chờ đến lượt. Một tiểu trình cũng có thể tạo lập các tiến trình con, và nhận các trạng thái khác nhau như một tiến trình thật sự. Một tiến trình có thể sở hữu nhiều tiểu trình.

2.1.3.2. Đa tiểu trình (Multithreaad) trong đơn tiến trình

Đa tiểu trình là có nhiều tiểu trình trong phạm vi một tiến trình đơn và được xem xét theo mô hình tác vụ (task - được định nghĩa như là một đơn vị của sự bảo vệ hay đơn vị cấp phát tài nguyên).

Trong phạm vi một tác vụ, có thể có một hoặc nhiều tiểu trình, mỗi tiểu trình bao gồm: Một trạng thái thực thi tiểu trình (running, ready,…); một lưu trữ về ngữ cảnh của processor khi tiểu trình ở trạng thái not running; các thông tin thống kê về việc sử dụng các biến cục bộ của tiểu trình; một stack thực thi. Việc truy xuất đến bộ nhớ và tài nguyên của tác vụ, được chia sẻ với tất cả các tiểu trình khác trong tác vụ.

Trong mô hình đa tiểu trình, có thể có nhiều tiểu trình được tạo ra và được giải phóng, có thể đồng thời, trong một khoảng thời gian ngắn. Trong hệ thống multiprocessor thì các tiểu trình trong cùng một tác vụ có thể thực hiện đồng thời trên các processor khác nhau, do đó hiệu suất của hệ thống tăng lên. Trong hệ thống uniprocessor, sự hình thành các tiểu trình này cũng thật sự hữu ích, khi một chương trình phải thực hiện nhiều chức năng khác nhau. Hiệu quả của việc sử dụng tiểu trình được thấy rõ trong các ứng dụng cần có sự truyền thông giữa các tiến trình hoặc các chương trình khác nhau.

Các thao tác lập lịch và điều phối tiến trình của HĐH thực hiện trên cơ sở tiểu trình. Nhưng nếu có một thao tác nào đó ảnh hưởng đến tấ cả các tiểu trình trong tác vụ thì HĐH phải tác động vào tác vụ.

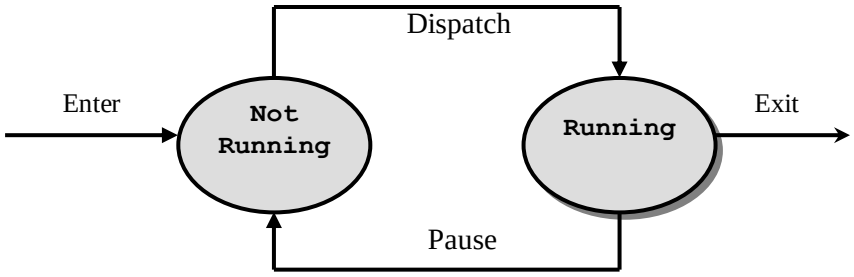
2.1.4. Các trạng thái tiến trình

Từ khi được đưa vào hệ thống cho đến khi kết thúc, tiến trình tồn tại ở các trạng thái khác nhau. Trạng thái của tiến trình tại một thời điểm được xác định bởi hoạt động hiện thời của tiến trình tại thời điểm đó

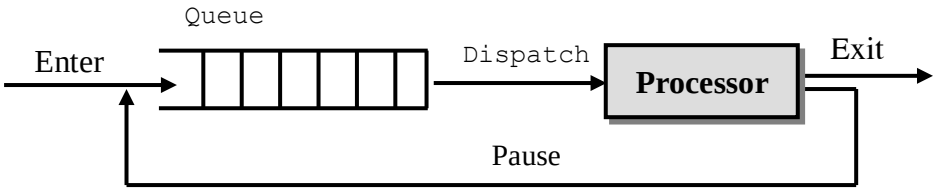
2.1.4.1. Tiến trình hai trạng thái

Một số ít HĐH chỉ cho phép tiến trình tồn tại ở một trong hai trạng thái: Not Running và Running. Khi một tiến trình mới được tạo thành, nó vào hệ thống ở trạng thái Not Running. Tiến trình được cấp BXL để chuyển sang trạng thái Running. Sau đó, nếu tiến trình hoàn thành (kết thúc), nó sẽ được đưa ra khỏi hệ thống (exit). Nếu tiến trình đang thực hiện bị ngắt thì bộ điều phối tiến trình sẽ thu hồi lại BXL và tiến trình này bị tạm dừng (pause) và chuyển về trạng thái Not running.

Tại mỗi thời điểm chỉ có duy nhất một tiến trình ở trạng thái Running, nhưng có thể có nhiều tiến trình ở trạng thái Not running và được chứa trong một hàng đợi (Queue). Tiến trình đang ở trạng thái Running bị chuyển sang trạng thái Not running sẽ được đưa vào hàng đợi.



Hình 2.3.a: Sơ đồ chuyển trạng thái tiến trình



Hình 2.3.b: Sơ đồ chuyển tiến trình vào hàng đợi

2.1.4.2. Tiến trình 3 trạng thái

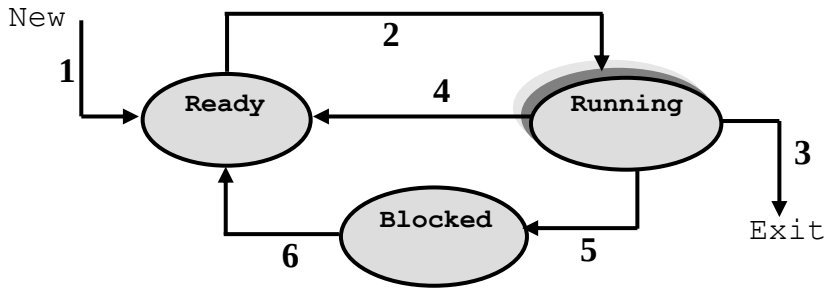
Đa số hệ điều hành đều cho phép tiến trình tồn tại ở một trong 3 trạng thái, đó là: ready, running, blocked:

- Trạng thái Ready (sẵn sàng): Các tiến trình, sau khi khởi tạo sẽ được cấp phát đầy đủ tài nguyên (trừ processor), tiến trình có trạng thái ready. Trạng thái ready là trạng thái của một tiến trình trong hệ thống đang chờ được cấp processor để bắt đầu thực hiện.

- Trạng thái Running (thực hiện): Là trạng thái mà tiến trình đang được sở hữu processor để hoạt động, hay nói cách khác là các chỉ thị của tiến trình đang được thực hiện/ xử lý bởi processor.

- Trạng thái Blocked (bị khoá-bị phong tỏa): Là trạng thái mà tiến trình chưa kết thúc và đang chờ được cấp phát thêm tài nguyên, chờ một sự kiện nào đó xảy ra, hay chờ một quá trình vào/ra kết thúc.

Quá trình chuyển trạng thái của các tiến trình trong được mô tả bởi sơ đồ sau:



Hình 2.4.a: Sơ đồ chuyển trạng thái tiến trình

1. (Admit) Tiến trình được khởi tạo, được đưa vào hệ thống, được cấp phát đầy đủ tài nguyên chỉ thiếu processor.

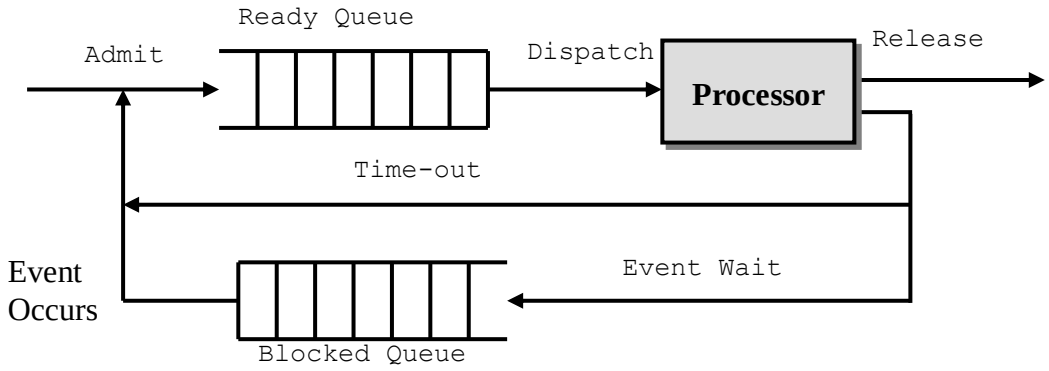
2. (Dispatch) Tiến trình được cấp processor để bắt đầu thực hiện/ xử lý.

3. (Release) Tiến trình hoàn thành xử lý và kết thúc.

4. (Time_out) Tiến trình bị bộ điều phối tiến trình thu hồi processor, do hết thời gian được quyền sử dụng processor, để cấp phát cho tiến trình khác.

5. (Event wait) Tiến trình đang chờ một sự kiện nào đó xảy ra hay đang chờ một thao vào/ra kết thúc hay tài nguyên mà tiến trình yêu cầu chưa được HĐH đáp ứng.

6. (Event Occurs) Sự kiện mà tiến trình chờ đã xảy ra, thao tác vào/ra mà tiến trình đợi đã kết thúc, hay tài nguyên mà tiến trình yêu cầu đã được HĐH đáp ứng,



Hình 2.4.b: Sơ đồ chuyển tiến trình vào các hàng đợi

Bộ phận điều phối tiến trình thu hồi processor từ một tiến trình đang thực hiện trong các trường hợp sau:

- Tiến trình đang thực hiện hết thời gian (time-out) được quyền sử dụng processor mà bộ phận điều phối dành cho nó.
- Có một tiến trình mới phát sinh và tiến trình mới này có độ ưu tiên cao hơn tiến trình hiện tại.
- Có một tiến trình mới phát sinh và tiến trình này mới cần một khoảng thời gian của processor nhỏ hơn nhiều so với khoảng thời gian còn lại mà tiến trình hiện tại cần processor.

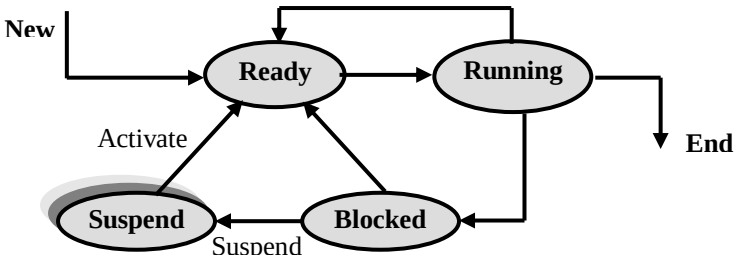
Tại một thời điểm xác định trong hệ thống có thể có nhiều tiến trình đang ở trạng thái Ready hoặc Blocked nhưng chỉ có một tiến trình ở trạng thái Running. Các tiến trình ở trạng thái Ready và Blocked được chứa trong các hàng đợi (Queue) riêng.

Có nhiều lý do để một tiến trình đang ở trạng thái running chuyển sang trạng thái blocked, do đó đa số các hệ điều hành đều thiết kế một hệ thống hàng đợi gồm nhiều hàng đợi, mỗi hàng đợi dùng để chứa những tiến trình đang đợi cùng một sự kiện nào đó.

2.1.4.3. Tiến trình bốn trạng thái

Trong môi trường HĐH đa nhiệm thì việc tổ chức các Queue để lưu các tiến trình chưa thể hoạt động là cần thiết, nhưng nếu tồn tại quá nhiều tiến trình trong Queue (trong bộ nhớ chính), sẽ dẫn đến trình trạng lãng phí bộ nhớ, không còn đủ bộ nhớ để nạp các tiến trình khác khi cần thiết. Mặt khác nếu các tiến trình trong Queue đang chiếm giữ tài nguyên của hệ thống, mà những tài nguyên này lại là những tài nguyên các tiến trình khác đang cần, điều này dẫn đến tình trạng sử dụng tài nguyên không hợp lý, làm cho hệ thống thiếu tài nguyên (thực chất là thừa) trầm trọng và có thể làm cho hệ thống tắc nghẽn.

Một trạng thái tiến trình mới là trạng thái Suspend (tạm dừng), rất cần thiết cho các hệ thống sử dụng kỹ thuật Swap trong việc cấp phát bộ nhớ cho các tiến trình.



Hình 2.5.a: Sơ đồ chuyển trạng thái tiến trình có suspend

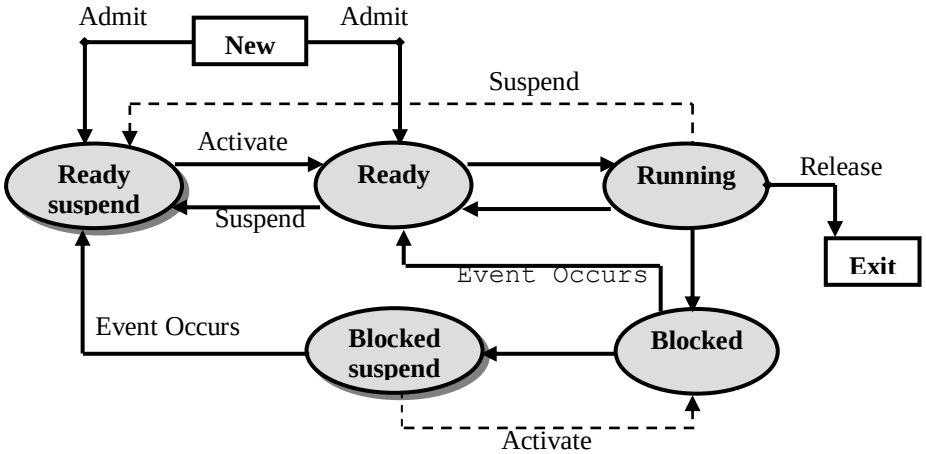
Trạng thái Suspend là trạng thái của một tiến trình khi nó đang được lưu trữ trên bộ nhớ phụ. Đó là các tiến trình đang ở trong trạng thái blocked và/hoặc ready bị HĐH chuyển ra đĩa để thu hồi lại không gian nhớ đã cấp cho tiến trình hoặc thu hồi lại tài nguyên đã cấp cho tiến trình để cấp cho một tiến trình khác đang rất cần được nạp vào bộ nhớ tại thời điểm hiện tại.

2.1.4.4. Tiến trình 5 trạng thái

Có 2 trạng thái suspend, một trạng thái suspend dành cho các tiến trình từ blocked chuyển đến, trạng thái này được gọi là blocked-suspend và một trạng thái suspend dành cho các tiến trình từ ready chuyển đến, trạng thái này được gọi là ready-suspend.

Tóm lại, có các trạng thái tiến trình sau:

- Ready: tiến trình được định vị trong bộ nhớ chính và đang chờ được cấp processor để thực hiện.
- Blocked: tiến trình được định vị trong bộ nhớ chính và đang đợi một sự kiện hay một quá trình I/O nào đó.
- Blocked-suspend: tiến trình đang bị chứa trên bộ nhớ phụ (đĩa) và đang đợi một sự kiện nào đó.
- Ready-suspend: tiến trình đang bị chứa trên bộ nhớ phụ nhưng sẵn sàng thực hiện ngay sau khi được nạp vào bộ nhớ chính.



Hình 2.5.b: Sơ đồ chuyển trạng thái tiến trình với 2 suspend

Sau đây chúng ta xem xét sự chuyển trạng thái tiến trình trong sơ đồ trên:

1. Blocked sang Blocked-suspend: nếu không còn tiến trình ready trong bộ nhớ chính và bộ nhớ chính không còn không gian nhớ trống thì phải có ít nhất một tiến trình blocked bị chuyển ra ngoài, blocked-suspend, để dành bộ nhớ cho một tiến trình không bị khoá (not blocked) khác.

2. Blocked-suspend sang Ready-suspend: một tiến trình đang ở trạng thái blocked-suspend được chuyển sang trạng thái ready-suspend khi sự kiện mà nó đợi đã xảy ra.

3. Ready-suspend sang Ready:

- Không còn tiến trình ready trong bộ nhớ chính, hệ điều hành phải nạp một tiến trình mới vào để nó tiếp tục thực hiện
- Nếu có tiến trình ready-suspend có độ ưu tiên cao hơn so với các tiến trình ready hiện tại thì hệ điều hành có thể chuyển nó sang trạng thái ready để nó nhiều cơ hội để được thực hiện hơn.

4. Ready sang Ready suspend: Hệ điều hành thường chuyển các tiến trình sang suspend dựa vào 2 điều kiện: chiếm ít không gian bộ nhớ hơn và có độ ưu tiên thấp hơn thì HĐH có thể chuyển một tiến trình ready sang trạng thái suspend.

Như vậy với việc chuyển tiến trình sang trạng thái suspend HĐH sẽ chủ động hơn trong việc cấp phát bộ nhớ và ngăn chặn các tình huống bế tắccó thể xảy ra do sự tranh chấp về tài nguyên, nhờ vậy mà HĐH tiết kiệm được bộ nhớ, chia sẻ được tài nguyên cho nhiều tiến trình và tăng được mức độ đa chương của hệ thống. Tuy nhiên, để có được những lợi ích trên HĐH đã phải chi phí rất nhiều cho việc tạm dừng tiến trình. Hệ điều hành phải xem xét tiến trình nào được chọn để suspend, khi suspend một tiến trình HĐH phải lưu lại tất cả các thông tin liên quan đến tiến trình đó (con trỏ lệnh, tài nguyên mà tiến trình đã được cấp, ...), HĐH phải lựa chọn thời điểm thích hợp để đưa tiến trình ra bộ nhớ ngoài, ... những thao tác đó sẽ làm chậm tốc độ thực hiện của toàn bộ hệ thống. Nhưng dầu sao đi nữa thì HĐH vẫn phải sử dụng trạng thái suspend vì tăng mức độ đa chương của hệ thống là một trong những mục tiêu lớn của HĐH.

2.1.5. Khối quản lý tiến trình (mô tả tiến trình - process descriptor)

Khối quản lý tiến trình (PCB – process control block) là 1 cấu trúc dữ liệu tương ứng với 1 tiến trình và cho phép xác định đơn vị tiến trình đó. PCB được lưu trữ trong BN và là một trong những cấu trúc dữ liệu trung tâm và quan trọng của HĐH. Mỗi PCB chứa tất cả các thông tin về tiến trình mà nó rất cần cho HĐH.

- Định danh tiến trình (PID: process identification): mỗi tiến trình được gán một định danh duy nhất để phân biệt với các tiến trình khác trong hệ thống. Định danh của tiến trình cần thiết trong quá trình thực hiện tiến trình, khi truyền thông giữa các tiến trình, khi xác định quan hệ giữa các tiến trình (tiến trình cha và tiến trình con).
- Trạng thái tiến trình xác định hoạt động hiện hành của tiến trình.
- Ngữ cảnh của tiến trình: mô tả các tài nguyên tiến trình đang trong quá trình, hoặc để phục vụ cho hoạt động hiện tại, hoặc để làm cơ sở phục hồi hoạt động cho tiến trình, bao gồm các thông tin về:

+ Trạng thái BXL: bao gồm nội dung các thanh ghi, quan trọng nhất là con trỏ lệnh IP (lưu trữ địa chỉ câu lệnh kế tiếp tiến trình sẽ xử lý). Các thông tin này cần được lưu trữ để phục hồi hoạt động của tiến trình khi xảy ra ngắt.

+ Số hiệu BXL mà tiến trình đang sử dụng.

+ Định vị của tiến trình (process location): là danh sách các không gian nhớ được cấp cho tiến trình trong quá trình thực hiện tiến trình (phần nằm trên đĩa phụ, phần nằm tại BN chính). Định vị của tiến trình phụ thuộc vào chiến lược quản lý bộ nhớ đang sử dụng.

+ Tài nguyên sử dụng: danh sách các tài nguyên mà tiến trình đang sử dụng.

+ Tài nguyên tạo lập: danh sách các tài nguyên được tiến trình tạo lập.

- Thông tin giao tiếp: là các thông tin về quan hệ của tiến trình với các tiến trình khác trong hệ thống:

+ Tiến trình cha: là tiến trình tạo lập ra tiến trình này.

+ Tiến trình con: các tiến trình do tiến trình này tạo ra.

+ Độ ưu tiên tiến trình giúp bộ điều phối có thông tin để lựa chọn tiến trình.

- Thông tin thống kê: là những thông tin thống kê về hoạt động tiến trình (thời gian đã sử dụng BXL, thời gian chờ…) giúp đánh giá tình hình hệ thống và dự đoán các tfhgs liên quan.

Các thông tin mô tả tiến trình có thể biểu diễn dưới dạng bảng mô tả và khung mô tả. Bảng mô tả tiến trình là một cấu trúc dữ liệu, chứa đầy đủ các thông tin trên. Khung tiến trình là khung dành chỉ có cấu trúc thông tin cho tiến trình, khi có tiến trình các thông tin cụ thể mới được gán vào khung để thực hiện tiến trình.

Khối quản lý tiến trình PCB được xây dựng mỗi khi có nhu cầu và huỷ sau khi công việc hoàn thành. Có nhiều modul thành phần trong HĐH có thể read và/hoặc modified PCB như: lập lịch tiến trình, cấp phát tài nguyên cho tiến trình, ngắt tiến trình… Có thể nói các thiết lập trong PCB định nghĩa trạng thái của HĐH.

2.1.6. Các thao tác điều khiển tiến trình

2.1.6.1. Khi khởi tạo tiến trình

Trong quá trình xử lý, một tiến trình có thể được khởi tạo; hoặc một tiến trình khi đang hoạt động có thể khởi tạo một hoặc một số tiến trình mới bằng cách sử dụng lời gọi hệ thống tương ứng.

Hệ điều hành *gán PID* – định danh tiến trình cho tiến trình mới và *đưa tiến trình vào danh sách quản lý* của hệ thống. Danh sách tiến trình có thể là ready list, suspend list, waiting list…, sao cho phù hợp với chiến lược điều phối tiến trình hiện tại của bộ phận điều phối tiến trình.

Hệ điều hành xác định độ ưu tiên cho tiến trình.

Hệ điều hành tạo PCB cho tiến trình chứa các thông tin cần thiết như các PID của tiến trình cha (nếu có), thông tin trạng thái tiến trình, thông tin ngữ cảnh của processor (bộ đếm chương trình và các thanh ghi khác),..

Cung cấp đầy đủ các tài nguyên cần thiết nhất, trừ processor, để tiến trình có thể vào trạng thái ready được hoặc bắt đầu hoạt động được. Cấp phát không gian bộ nhớ cho tiến trình. Ở đây HĐH cần phải xác định được kích thước của tiến trình, bao gồm code, data và stack. Giá trị kích thước này có thể được gán mặc định dựa theo loại của tiến trình hoặc được gán theo yêu cầu của người sử dụng khi có một công việc (job) được tạo. Nếu một tiến trình được sinh ra bởi một tiến trình khác, thì tiến trình cha có thể chuyển kích thước của nó đến hệ điều hành trong yêu cầu tạo tiến trình.

2.1.6.2. Khi kết thúc tiến trình

Một tiến trình kết thúc khi: hoàn tất công việc; sử dụng BXL vượt quá thời hạn; sử dụng quá tài nguyên quy định; khi bộ nhớ không đủ; khi vi phạm một số quy định; khi mắc một số lỗi về phép toán; khi thiết bị ngoại vi bị lỗi; khi các lệnh bị sai; khi có tiến trình khác có quyền ưu tiên cao hơn; khi có dữ liệu sai và khi HĐH dừng một số tiến trình.

Khi kết thúc tiến trình, HĐH thực hiện các công việc sau:

- Thu hồi các tài nguyên hệ thống đã cấp phát cho tiến trình.
- Hủy bỏ tiến trình ra khỏi danh sách quản lý của hệ thống.
- Huỷ bỏ khối điều khiển tiến trình.

Hầu hết các HĐH đều không cho phép tiến trình con hoạt động khi tiến trình cha đã kết thúc. Trong những trường hợp như thế HĐH sẽ chủ động việc kết thúc tiến trình con khi tiến trình cha vừa kết thúc.

2.1.6.3. Khi thay đổi trạng thái tiến trình

Khi một tiến trình đang ở trạng thái running bị chuyển sang trạng thái khác (ready, blocked, …) thì HĐH phải tạo ra sự thay đổi trong môi trường làm việc của nó.

Lưu (save) ngữ cảnh của processor, bao gồm thanh ghi bộ đếm chương trình (PC: program counter) và các thanh ghi khác.

Cập nhật PCB của tiến trình, sao cho phù hợp với trạng thái mới của tiến trình, bao gồm trạng thái mới của tiến trình, các thông tin tính toán,…

Di chuyển PCB của tiến trình đến một hàng đợi thích hợp, để đáp ứng được các yêu cầu của công tác điều phối tiến trình.

Chọn một tiến trình khác để cho phép nó thực hiện.

Cập nhật PCB của tiến trình vừa được chọn thực hiện ở trên, chủ yếu là thay đổi trạng thái của tiến trình đến trạng thái running.

Cập nhật các thông tin liên quan đến quản lý bộ nhớ. Bước này phụ thuộc vào các yêu cầu chuyển đổi địa chỉ bộ nhớ đang được sử dụng.

Khôi phục (Restore) lại ngữ cảnh của processor và thay đổi giá trị của bộ đếm chương trình và các thanh ghi khác sao cho phù hợp với tiến trình được chọn ở trên, để tiến trình này có thể bắt đầu hoạt động được.

Như vậy, khi hệ điều hành chuyển một tiến trình từ trạng thái running (đang chạy) sang một trạng thái nào đó (tạm dừng) thì HĐH phải lưu trữ các thông tin cần thiết, nhất

là Program Count, để sau này HĐH có thể cho tiến trình tiếp tục hoạt động trở (tái kích hoạt) lại được. Đồng thời HĐH phải chọn một tiến trình nào đó đang ở trạng thái ready để cho tiến trình này chạy (chuyển tiến trình sang trạng thái running). Tại đây, trong các thao tác phải thực hiện, HĐH phải thực hiện việc thay đổi giá trị của PC, thay đổi ngữ cảnh processor, để PC chỉ đến địa chỉ của chỉ thị đầu tiên của tiến trình running mới này trong bộ nhớ. Đây cũng chính là bản chất của việc thực hiện các tiến trình trong các hệ thống uniprocessor.

2.1.7. Cấp phát tài nguyên cho tiến trình

Hệ điều hành cần cấp phát tài nguyên để đáp ứng được yêu cầu của người sử dụng trong điều kiện tài nguyên hệ thống rất giới hạn và đôi khi tài nguyên là không thể chia sẻ. Hệ điều hành quản lý nhiều loại tài nguyên, và mỗi loại tài nguyên có một cơ chế cấp phát và một chiến lược cấp phát hiệu quả. Việc cấp phát tài nguyên phải bảo đảm được các mục tiêu sau:

- Bảo đảm được số lượng nhất định các tiến trình được thực hiện.
- Bảo đảm yêu cầu thời gian cho phép được cấp phát tài nguyên đối với mỗi tiến trình.
- Tối ưu hóa sử dụng tài nguyên.

Để thực hiện được mục tiêu trên, hệ điều hành quản lý cấp phát dựa trên các thông tin sau:

- Định danh tài nguyên.
- Trạng thái tài nguyên (phần tài nguyên đã được cấp phát, phần có thể sử dụng)
- Hàng đợi trên một tài nguyên là danh sách các tiến trình đang chờ được cấp phát tài nguyên tương ứng.

2.2. ĐIỀU PHỐI TIẾN TRÌNH

2.2.1. Khái quát về điều phối tiến trình

2.2.1.1. Khái niệm

Trong môi trường hệ điều hành đa nhiệm, bộ phận điều phối tiến trình có nhiệm vụ xem xét và quyết định khi nào thì dừng tiến trình hiện tại để thu hồi processor và chuyển processor cho tiến trình khác, và khi đã có được processor thì chọn tiến trình nào trong số các tiến trình ở trạng thái ready để cấp processor cho nó.

Điều phối tiến trình là lựa chọn tiến trình đã sẵn sàng và phân phối processor cho nó để tiến trình được thực hiện. Điều phối tiến trình được thực hiện bởi bộ điều phối (dispatcher), nó không đưa ra các cơ chế mà trực tiếp đưa ra các quyết định lựa chọn tiến trình.

Có 3 mức điều phối tiến trình: điều phối mức cao (tác vụ), mức giữa (tiến trình) và mức thấp (điều độ tiến trình). Điều phối tác vụ được phải thực hiện trước điều phối tiến trình. Ở mức này HĐH thực hiện việc chọn tác vụ để đưa vào hệ thống (BN chính). Khi có một tiến trình được tạo lập hoặc khi có một tiến trình kết thúc xử lý thì bộ phận điều phối tác vụ được kích hoạt. Điều phối tác vụ quyết định sự đa chương của hệ thống và hiệu quả cũng như mục tiêu của điều phối của bộ phận điều phối tiến trình. Ví dụ, để khi thác tối đa thời gian xử lý của processor thì bộ phận điều phối tác vụ phải đưa vào hệ

thống số lượng các tiến trình tính hướng Vào/Ra cân đối với số lượng các tiến trình tính hướng xử lý, các tiến trình này thuộc những tác vụ nào. Nếu trong hệ thống có quá nhiều tiến trình tính hướng Vào/Ra thì sẽ lãng phí thời gian xử lý của processor. Nếu trong hệ thống có quá nhiều tiến trình tính hướng xử lý thì processor không thể đáp ứng và có thể các tiến trình phải đợi lâu trong hệ thống, dẫn đến hiệu quả tương tác sẽ thấp.

Có thể xem các mức lập lịch ứng với các bộ điều phối dài hạn và bộ điều phối ngắn hạn. Bộ điều phối dài hạn lựa chọn các tiến trình từ bên ngoài tải vào BN. Bộ điều phối ngắn hạn lựa chọn tiến trình nào đó đã sẵn sàng để cấp phát BXL cho tiến trình đó. Tần suất hoạt động của bộ điều phối ngắn hạn rất cao hơn so với tần suất hoạt động của bộ điều phối dài hạn.

2.2.1.2. Căn cứ điều phối tiến trình

Khi tổ chức điều phối tiến trình, bộ phận điều phối tiến trình căn cứ vào độ (mức, chỉ số) ưu tiên của tiến trình và thời gian lượng tử:

- Độ ưu tiên của tiến trình là mức độ ưu tiên quyết định thứ tự sử dụng BXL để tiến trình được thực hiện, do HĐH gán cho tiến trình trong suốt thời gian tồn tại tiến trình. Mỗi tiến trình được gán một độ ưu tiên nhất định, độ ưu tiên này có thể được phát sinh tự động bởi hệ thống hoặc được gán tường minh trong chương trình của người sử dụng. Tính chất độ ưu tiên của tiến trình có hai loại: độ ưu tiên tĩnh: là độ ưu tiên gán trước cho tiến trình và không thay đổi trong suốt thời gian tồn tại của tiến trình và độ ưu tiên động: là độ ưu tiên được gán cho tiến trình trong quá trình hoạt động của nó, hệ điều hành sẽ gán lại độ ưu tiên cho tiến trình khi môi trường xử lý của tiến trình bị thay đổi. Lúc này, hệ điều hành phải thay đổi độ ưu tiên của tiến trình cho phù hợp với tình trạng hiện tại của hệ thống và công tác điều phối tiến trình của HĐH.

Độ ưu tiên (hay mức ưu tiên) của tiến trình được xác định căn cứ vào:

- Các thuộc tính của tiến trình bao gồm: Thời điểm khởi tạo, thời điểm yêu cầu kết thúc tiến trình; độ dài tiến trình là thời gian sử dụng processor của tiến trình để hoàn thành xử lý; thời gian tiến trình đã được xử lý là thời gian tiến trình đã sử dụng processor; thời gian còn lại tiến trình cần processor là khoảng thời gian cần processor để hoàn thành xử lý.
- Tính chất của tiến trình: Tiến trình thiên hướng Vào/Ra (I/O-boundedness): là các tiến trình cần nhiều thời gian hơn cho việc thực hiện các thao tác xuất/nhập dữ liệu, so với thời gian mà tiến trình cần để thực hiện các chỉ thị trong nó. Tiến trình thiên hướng xử lý (CPU-boundedness): là các tiến trình cần nhiều thời gian hơn cho việc thực hiện các chỉ thị trong nó, so với thời gian mà tiến trình để thực hiện các thao tác Vào/Ra.
- Tính chất xử lý tiến trình: Xử lý thời gian thực (tiến trình tương tác) tiến trình cần phải trả lại kết quả tức thời (như trong hệ điều hành tương tác); xử lý theo lô: xử lý một loạt tiến trình (như trong hệ điều hành xử lý theo lô); xử lý theo chế độ phân chia thời gian.

Thời gian lượng tử (quan tum) là thời gian gán processor cho tiến trình để tiến

trình được thực hiện. Đối với mỗi tiến trình hay cho mọi tiến trình thời gian lượng tử có thể không đổi, có thể thay đổi (theo yêu cầu xử lý tiến trình hoặc theo độ dài tiến trình sau mỗi lần xử lý. Bộ quản lý tiến trình sử dụng bộ định thời (timer) điều khiển việc chuyển quyền sử dụng BXL, thông qua cơ chế ngắt.

2.2.1.3. Mục tiêu điều phối tiến trình

Bộ phận điều phối tiến trình của hệ điều hành phải đạt được các mục tiêu sau đây (đáp ứng cho tiến trình và đáp ứng việc sử dụng processor) trong công tác điều phối của nó.

- Sự công bằng (Fairness): Các tiến trình đều công bằng với nhau trong việc chia sẻ thời gian xử lý của processor, không có tiến trình nào phải chờ đợi vô hạn để được cấp processor.

- Thời gian đáp ứng hợp lý (Response time): Đối với các tiến trình tương tác, đây là khoảng thời gian từ khi tiến trình đưa ra yêu cầu cho đến khi nhận được sự hồi đáp. Một tiến trình đáp ứng yêu cầu của người sử dụng, phải nhận được thông tin hồi đáp từ yêu cầu của nó thì nó mới có thể trả lời người sử dụng. Do đó, theo người sử dụng thì bộ phận điều phối phải cực tiểu hoá thời gian hồi đáp của các tiến trình, có như vậy thì tính tương tác của tiến trình mới tăng lên.

- Thời gian lưu lại trong hệ thống (T_{TRnd} - Turnaround time): Đây là khoảng thời gian từ khi tiến trình được đưa ra đến khi được hoàn thành. Bao gồm thời gian thực hiện thực tế cộng với thời gian đợi tài nguyên (bao gồm cả đợi processor). Đại lượng này dùng trong các hệ điều hành xử lý theo lô. Do đó, bộ phận điều phối phải cực tiểu thời gian hoàn thành (lưu lại trong hệ thống) của các tác vụ xử lý theo lô.

- Tính hiệu quả (Efficiency): Tận dụng được 100% thời gian xử lý của processor. Trong công tác điều phối, khi processor rỗi bộ phận điều phối sẽ chuyển ngay nó cho tiến trình khác, nếu trong hệ thống có tiến trình đang ở trạng thái chờ processor, nên mục tiêu này dễ đạt được. Tuy nhiên, nếu hệ điều hành đưa vào hệ thống quá nhiều tiến trình thiên hướng vào/ra, thì nguy cơ processor bị rỗi là có thể. Do đó, để đạt được mục tiêu này hệ điều hành phải tính toán và quyết định nên đưa vào hệ thống bao nhiêu tiến trình thiên hướng vào/ra, bao nhiêu tiến trình thiên hướng xử lý, là thích hợp.

- Thông lượng tối đa (Throughtput): Chính sách điều phối phải cố gắng để cực đại được số lượng tiến trình hoàn thành trên một đơn vị thời gian. Mục tiêu này ít phụ thuộc vào chính sách điều phối mà phụ thuộc nhiều vào thời gian thực hiện trung bình của các tiến trình.

Tải vào hệ thống được mô tả bằng tốc độ tiến trình đến hàng đợi và thời gian phục vụ $\tau(p_i)$. Nếu λ là số lượng trung bình các tiến trình đến hàng đợi, μ là tốc độ phục vụ trung bình thì hệ số đánh giá khả năng đáp ứng tải của hệ thống là ρ và được xác định:

$$\rho = \frac{\lambda}{\mu};$$

nếu $\rho>1$, hệ thống sẽ bị quá tải; hệ thống chỉ tiến tới trạng thái ổn định khi $\rho<1$.

Công tác điều phối của hệ điều hành khó có thể thỏa mãn đồng thời tất cả các mục tiêu trên vì bản thân các mục tiêu này đã có sự mâu thuẫn với nhau. Các hệ điều hành chỉ có thể dung hòa các mục tiêu này ở một mức độ nào đó.

2.2.1.4. Các cơ chế điều phối tiến trình

Trong công tác điều phối tiến trình bộ điều phối sử dụng hai cơ chế điều phối: Điều phối độc quyền (chia sẻ tự nguyện BXL) và điều phối không độc quyền (chia sẻ không tự nguyện BXL).

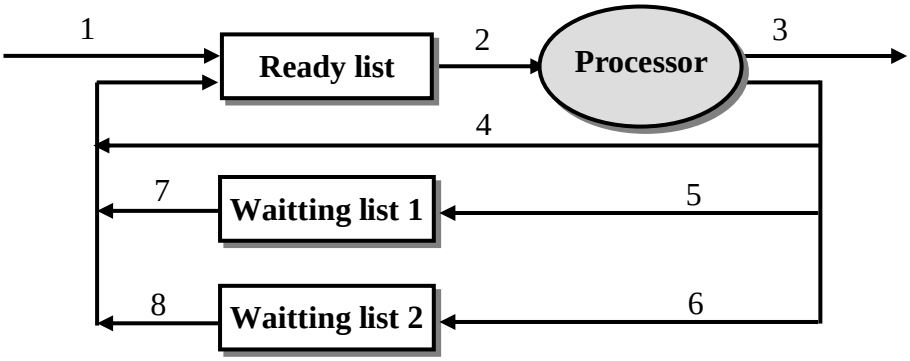
- Điều phối độc quyền: Khi có được processor tiến trình toàn quyền sử dụng processor cho đến khi tiến trình kết thúc xử lý hoặc tiến trình tự động trả lại processor cho hệ thống. Các quyết định điều phối xảy ra khi: Tiến trình chuyển trạng thái từ Running sang Blocked hoặc khi tiến trình kết thúc.
- Điều phối không độc quyền: Bộ phận điều phối tiến trình có thể bị tạm dừng (ngắt) tiến trình đang xử lý để thu hồi processor của nó, để cấp cho tiến trình khác, sao cho phù hợp với công tác điều phối hiện tại. Các quyết định điều phối xảy ra khi: Tiến trình chuyển trạng thái hoặc khi tiến trình kết thúc.

2.4.2.5. Các danh sách sử dụng trong điều phối tiến trình

Hệ điều hành sử dụng hai loại danh sách để thực hiện điều phối các tiến trình là *danh sách sẵn sàng (ready list)* và *danh sách chờ đợi (waiting list)*.

Khi một tiến trình bắt đầu đi vào hệ thống, nó được chèn vào *danh sách các tác vụ* (job list). Danh sách này bao gồm tất cả các tiến trình của hệ thống. Nhưng chỉ các tiến trình đang thường trú trong bộ nhớ chính và ở trạng thái sẵn sàng tiếp nhận BXL để hoạt động mới được đưa vào *danh sách sẵn sàng*.

Chỉ có những tiến trình trong ready list mới được chọn để cấp processor. Các tiến trình bị chuyển về trạng thái blocked sẽ được bổ sung vào waiting list. Hệ thống chỉ có duy nhất một ready list, nhưng có thể tồn tại nhiều waiting list. Thông thường hệ điều hành thiết kế nhiều waitting list, mỗi waitting list dùng để chứa các tiến trình đang đợi được cấp phát một tài nguyên hay một sự kiện riêng biệt nào đó. Hình sau đây minh hoạ cho việc chuyển tiến trình giữa các danh sách:



Hình 2.7: Sơ đồ chuyển tiến trình vào các danh sách

1. Tiến trình trong hệ thống được cấp đầy đủ tài nguyên chỉ thiếu processor.
2. Tiến trình được bộ điều phối chọn ra để cấp processor để bắt đầu xử lý.

3. Tiến trình kết thúc xử lý và trả lại processor cho hệ điều hành.

4. Tiến trình hết thời gian được quyền sử dụng processor (time-out), bị bộ điều phối tiến trình thu hồi lại processor.

5. Tiến trình bị khóa (blocked) do yêu cầu tài nguyên nhưng chưa được hệ điều hành cấp phát. Khi đó tiến trình được đưa vào danh sách các tiến trình đợi tài nguyên (waiting list 1).

6. Tiến trình bị khóa (blocked) do đang đợi một sự kiện nào đó xảy ra. Khi đó tiến trình được bộ điều phối đưa vào danh sách các tiến trình đợi tài nguyên (waiting list 2).

7. Tài nguyên mà tiến trình yêu cầu đã được hệ điều hành cấp phát. Khi đó tiến trình được bộ điều phối chuyển sang danh sách các tiến trình ở trạng thái sẵn sang (ready list) để chờ được cấp processor để được hoạt động.

8. Sự kiện mà tiến trình chờ đã xảy ra. Khi đó tiến trình được bộ điều phối chuyển sang danh sách các tiến trình ở trạng thái sẵn sàng (ready list) để chờ được cấp processor.

2.2.2. Tổ chức điều phối tiến trình

2.2.2.1. Các chiến lược điều phối độc quyền

a- Chiến lược phục vụ bình đẳng FCFS (First Come First Served)

Trong chiến lược này, khi processor rỗi thì hệ điều hành sẽ cấp nó cho tiến trình đầu tiên trong ready list, đây là tiến trình được chuyển sang trạng thái ready sớm nhất, có thể là tiến trình được đưa vào hệ thống sớm nhất. FCFS được sử dụng trong điều phối độc quyền nên khi tiến trình được cấp processor nó sẽ sở hữu processor cho đến khi kết thúc xử lý hay phải đợi một thao tác vào/ra hoàn thành, khi đó tiến trình chủ động trả lại processor cho hệ thống.

Ví dụ: Nếu hệ điều hành cần cấp processor cho 5 tiến trình P1, P2, P3, P4,P5 với thời điểm vào ready list và khoảng thời gian mỗi tiến trình cần processor được mô tả trong bảng sau:

Tiến trình	Thời điểm vào RL (t_i)	Độ ưu tiên	Thời gian xử lý $\tau(P_i)$
P1	0	3	24
P2	1	5	3
P3	2	2	7
P4	3	1	18
P5	4	4	12

Thứ tự cấp phát BXL cho các tiến trình là :

P1	P2	P3	P4	P5
0	24	27	34	52 64

Nếu coi tất cả các tiến trình cùng được khởi tạo trong RL vào một thời điểm, thì thời gian lưu lại hệ thống T_{TRnd} của mỗi tiến trình được tính như sau:

$T_{TRnd}(P1) = \tau(P1) = 24;$

$T_{TRnd}(P2) = \tau(P2) + T_{TRnd}(P1) = 3 + 24 = 27;$

$T_{TRnd}(P3) = \tau(P3) + T_{TRnd}(P2) = 7 + 27 = 34;$

$T_{TRnd}(P4) = \tau(P4) + T_{TRnd}(P2) = 18 + 34 = 52;$

$T_{TRnd}(P5) = \tau(P5) + T_{TRnd}(P2) = 12 + 52 = 64;$

Thời gian chờ đợi được xử lý là:

W(P1) =0, W(P2) =24, W(P3)=27, W(P4) =34, W(P5)=52. Thời gian chờ trung bình W_{tb} = (0+24+27+34+52)/5 = 27,4 đơn vị thời gian.

Như vậy FCFS tồn tại một số hạn chế: Thứ nhất, có thời gian chờ đợi trung bình lớn nên không phù hợp với các hệ thống chia sẻ thời gian. Thứ hai, khả năng tương tác kém khi nó được áp dụng trên các hệ thống uniprocessor. Thứ ba, nếu các tiến trình ở đầu ready list cần nhiều thời gian của processor thì các tiến trình ở cuối ready list sẽ phải chờ lâu mới được cấp processor.

b- Chiến lược ưu tiên cho tiến trình ngắn nhất SJF (Shortest Job Fist)

Đây là trường hợp đặc biệt của chiến lược theo độ ưu tiên. Trong chiến lược này độ ưu tiên của mỗi tiến trình tương ứng với 1/t, với t là khoảng thời gian mà tiến trình cần processor. Bộ điều phối sẽ chọn tiến trình có độ ưu tiên lớn để cấp processor, tức là ưu tiên cho những tiến trình có thời gian xử lý (thời gian cần processor) nhỏ.

Chiến lược này có thể có thời gian chờ đợi trung bình đạt cực tiểu. Nhưng hệ điều hành khó có thể đoán được thời gian xử lý mà tiến trình yêu cầu.

Ví dụ với 5 tiến trình như bảng trên và coi các tiến trình cùng xuất hiện trong RL. Thứ tự cấp phát BXL như sau:

P2	P3	P5	P4	P1
0	3	10	22	40 64

Thời gian lưu lại hệ thống T_{TRnd} của mỗi tiến trình được tính như sau:

$T_{TRnd}(P1) = \tau(P2) + \tau(P3) + \tau(P5) + \tau(P4) + \tau(P1) = 3 + 7 + 12 + 18 + 24 = 64$;

$T_{TRnd}(P2) = \tau(P2) = 3$;

$T_{TRnd}(P3) = \tau(P2) + \tau(P3) = 3 + 7 = 10$;

$T_{TRnd}(P4) = \tau(P2) + \tau(P3) + \tau(P5) + \tau(P4) = 3 + 7 + 12 + 18 = 40$;

$T_{TRnd}(P5) = \tau(P2) + \tau(P3) + \tau(P5) = 3 + 7 + 12 = 22$;

Thời gian chờ đợi được xử lý là:

W(P1) =40, W(P2) =0, W(P3)=3, W(P4) =22, W(P5)=10. Thời gian chờ trung bình W_{tb} = (40+0+3+22+10)/5 = 15 đơn vị thời gian.

c- Chiến lược điều phối theo độ ưu tiên

Trong chiến lược này, bộ phận điều phối tiến trình dựa vào độ ưu tiên của các tiến trình để tổ chức cấp processor cho tiến trình. Tiến trình được chọn để cấp processor là tiến trình có độ ưu tiên cao nhất, tại thời điểm hiện tại.

Khi hệ thống phát sinh một tiến trình ready mới, thì bộ phận điều phối sẽ so sánh độ ưu tiên của tiến trình mới phát sinh với độ ưu tiên của tiến trình đang sở hữu processor (tạm gọi là tiến trình hiện tại) để chèn tiến trình mới vào ready list tại vị trí thích hợp.

Chiến lược này cũng phải sử dụng ready list, và ready list luôn được xếp theo thứ tự giảm dần của độ ưu tiên kể từ đầu danh sách. Điều này có nghĩa là tiến trình được chọn để cấp processor là tiến trình ở đầu ready list.

Chiến lược này có thể dẫn đến hậu quả: các tiến trình có độ ưu tiên thấp sẽ rơi vào tình trạng chờ đợi vô hạn. Để khắc phục điều này HĐH thường hạ độ ưu tiên của các tiến trình có độ ưu tiên cao sau mỗi lần nó được cấp processor.

Ví dụ với 5 tiến trình có độ ưu tiên như bảng trên và coi các tiến trình cùng xuất hiện trong RL. Thứ tự cấp phát BXL như sau:

P4	P3	P1	P5	P2
0	18	25	49	61 64

Thời gian lưu lại hệ thống T_{TRnd} của mỗi tiến trình được tính như sau:

$T_{TRnd}(P1) = \tau(P4) + \tau(P3) + \tau(P1) = 18 + 7 + 24 = 49$;

$T_{TRnd}(P2) = \tau(P4) + \tau(P3) + \tau(P1) + \tau(P5) + \tau(P2) = 18 + 7 + 24 + 12 + 3 = 64$;

$T_{TRnd}(P3) = \tau(P4) + \tau(P3) = 18 + 7 = 25$;

$T_{TRnd}(P4) = \tau(P4) = 18$;

$T_{TRnd}(P5) = \tau(P4) + \tau(P3) + \tau(P1) + \tau(P5) = 18 + 7 + 24 + 12 = 61$;

Thời gian chờ đợi được xử lý là:

W(P1) =25, W(P2) =61, W(P3)=18, W(P4) =0, W(P5)=49. Thời gian chờ trung bình W_{tb} = (25+61+18+0+49)/5 = 30,6 đơn vị thời gian.

d- Chiến lược điều phối có thời hạn (Deadline Scheduling)

Điều phối có thời hạn là chiến lược điều phối để đảm bảo tiến trình không những được hoàn thành mà còn phải hoàn thành (kết thúc) đúng thời hạn cho trước. Những tiến trình loại này thường có trong các hệ thống thời gian thực.

Để thực hiện điều phối tiến trình, việc điều phối phải căn cứ vào thời hạn yêu cầu phải kết thúc tiến trình, thời gian yêu cầu BXL của tiến trình đó và các yêu cầu thời gian của các tiến trình khác trong hàng đợi để thực hiện lập lịch. Trong trường hợp này, các tiêu chí như thời gian chờ, thời gian lưu lại trong hệ thống không được coi là tiêu chí quan trọng để đánh giá hiệu suất của hệ thống.

2.2.2.2. Các chiến lược điều phối không độc quyền

a- Chiến lược phân phối xoay vòng (RR - Round Robin)

Trong chiến lược này, ready list được thiết kết theo dạng danh sách nối vòng. Tiến trình được bộ điều phối chọn để cấp processor cũng là tiến trình ở đầu ready list, nhưng sau một khoảng thời gian nhất định nào đó thì bộ điều phối thu hồi lại processor của tiến trình vừa được cấp processor và chuyển processor cho tiến trình kế tiếp (bây giờ đã trở thành tiến trình đầu tiên) trong ready list, tiến trình vừa bị thu hồi processor được đưa vào lại cuối ready list. Rõ ràng đây là chiến lược điều phối không độc quyền.

Khoảng thời gian mà mỗi tiến trình được sở hữu processor để hoạt động là bằng nhau, và thường được gọi là Quantum.

Ví dụ với 5 tiến trình như bảng trên và coi các tiến trình cùng xuất hiện trong RL. Thứ tự cấp phát BXL như sau:

P1	P2	P3	P4	P5	P1	P3	P4	P5
0	4 - 7	7	11	15	19	23- 26	26	30

P1	P4	P5	P1	P4	P1	P4	P1
34	38	42-46	46	50	54	58- 60	60-64

Thời gian lưu lại hệ thống T_{TRnd} của mỗi tiến trình được tính như sau:

$T_{TRnd}(P1) = 64$; $T_{TRnd}(P2) = 7$; $T_{TRnd}(P3) = 26$; $T_{TRnd}(P4) = 60$; $T_{TRnd}(P5) = 46$;

Thời gian chờ đợi được xử lý là:

W(P1) =0, W(P2) =4, W(P3)=7, W(P4) =11, W(P5)=15. Thời gian chờ trung bình W_{tb} = (0+4+7+11+15)/5 = 7,4 đơn vị thời gian.

Như vậy RR có thời gian chờ đợi trung bình nhỏ hơn so với FCFS.

Nếu có *n* tiến trình trong danh sách sẵn sàng và sử dụng quantum *q*, thì mỗi tiến trình sẽ được cấp phát BXL *1/n* trong từng khoảng thời gian *q*. Mỗi tiến trình sẽ không phải đợi quá *(n-1)q* đơn vị thời gian trước khi nhận được BXL cho lượt kế tiếp.

Trong chiến lược này, vấn đề đặt ra đối với công tác thiết kế là: nên chọn quantum bằng bao nhiêu là thích hợp, nếu quantum nhỏ thì hệ thống phải tốn nhiều thời gian cho việc cập nhật ready list và chuyển trạng thái tiến trình, dẫn đến vi phạm mục tiêu: khai thác tối đa thời gian xử lý của processor. Nếu quantum lớn thì thời gian chờ đợi trung bình và thời gian hồi đáp sẽ tăng lên, dẫn đến tính tương tác của hệ thống bị giảm xuống.

b- Chiến lược điều phối nhiều hàng đợi: Hệ điều hành phân lớp các tiến trình theo độ ưu tiên của chúng để có cách thức điều phối thích hợp cho từng hàng đợi tiến trình. Mỗi cấp độ ưu tiên có một realy list riêng. Bộ điều phối dùng chiến lược điều phối thích hợp cho từng realy list. Hệ điều hành cũng phải thiết kế một cơ chế thích hợp để điều phối tiến trình giữa các lớp.

Ở đây HĐH thường tổ chức gán độ ưu tiên cho tiến trình theo nguyên tắc kết hợp giữa gán tĩnh và gán động. Khi khởi tạo tiến trình được gán độ ưu tiên tĩnh, sau đó phụ thuộc vào môi trường hoạt động của tiến trình và công tác điều phối tiến trình của bộ phận điều phối mà HĐH có thể thay đổi độ ưu tiên của tiến trình.

Các tiến trình ở ready list có độ ưu tiên thấp sẽ phải chờ đợi processor trong một khoảng thời gian dài, có thể là vô hạn. Để khắc phục điều này HĐH xây dựng chiến lược điều phối nhiều mức độ ưu tiên xoay vòng. Hệ điều hành chuyển dần một tiến trình ở ready list có độ ưu tiên cao xuống ready list có độ ưu tiên thấp hơn sau mỗi lần sử dụng procesor, và ngược lại một tiến trình ở lâu trong ready list có độ ưu tiên thấp thì sẽ được chuyển dần lên ready list có độ ưu tiên cao hơn.

Khi xây dựng chiến lược nhiều mức độ ưu tiên xoay vòng HĐH cần xác định các thông tin sau: Số lượng các lớp ưu tiên. Chiến lược điều phối riêng cho từng read list trong mỗi lớp ưu tiên. Một tiến trình ready mới sẽ được đưa vào ready list nào. Khi nào thì thực hiện việc di chuyển một tiến trình từ ready list này sang ready list khác.

Trong hệ thống, người ta thường chia tiến trình thành 2 dạng: tiến trình có nhiều tương tác I/O – tiến trình tiền cảnh (Foreground) và tiến trình không có nhiều yêu cầu này – tiến trình hậu cảnh (background). Các tiến trình tiền cảnh luôn có độ ưu tiên cao hơn các tiến trình hậu cảnh. Ví dụ: tiến trình xử lý ngắt có mức ưu tiên 1; tiến trình điều khiển thiết bị có độ ưu tiên 2; tiến trình tương tác với người dùng ưu tiên 3; tiến trình soạn thảo ưu tiên 4; tiến trình xử lý theo lô ưu tiên 5; tiến trình cần nhiều thời gian thực thi ưu tiên 6…

c- Một số chiến lược điều phối khác

Điều phối có đảm bảo (Guaranteed Scheduling) là chiến lược điều phối có đảm bảo chất lượng thông qua việc kiểm soát thời gian sử dụng BXL của tiến trình. Tiến trình nào có tỷ lệ giữa thời gian hệ thống giành cho tiến trình với số lượng tiến trình có trong hàng đợi mà thấp sẽ được ưu tiên phục vụ trước.

Điều phối quay sổ số (Lottery) là dạng cụ thể của chiến lược điều phối có đảm bảo. Việc tiến trình được cấp phát tài nguyên tương ứng với việc tiến trình có được vé số của mình trúng thưởng. Mức độ ưu tiên của tiến trình được đáp ứng bằng cách được cung cấp nhiều vé số hơn, làm tăng khả năng trúng thưởng hơn.

Điều phối công bằng là chiến lược nhằm đảm bảo cho mỗi người sử dụng có được lượng thời gian sử dụng BXL không tương ứng với số lượng các tiến trình mà người sử dụng đó tạo ra. Ví dụ, người sử dụng 1 có 4 tiến trình A,B,C,D và người sử dụng 2 có tiến trình E. Theo tỷ lệ số tiến trình tạo ra, người sử dụng 1 chiếm 80% thời gian BXL và người sử dụng 2 chiếm 20%. Nếu tỷ lệ này là 1:1, kết quả điều phối là A,E,B,E,C,E,D,E,A,E… Nếu tỷ lệ này là 2:1, kết quả điều phối là A,B,E,C,D,E,A,B,E,…

2.3. TƯƠNG TRANH VÀ ĐỒNG BỘ

2.3.1. Tương tranh

2.3.1.1. Nhu cầu liên lạc giữa các tiến trình

Trong môi trường đa chương, hầu hết các tiến trình là các tiến trình cộng tác, tức là chúng có thể làm ảnh hưởng hoặc bị ảnh hưởng đến các tiến trình khác. Sự tác động lẫn nhau này do các nguyên nhân sau:

Chia sẻ thông tin: nhiều tiến trình có cùng quan tâm đến những dữ liệu nào đó và cần có môi trường cho phép chúng truy cập đồng thời đến dữ liệu chung.

Hợp tác để hoàn thành tác vụ: khi tác vụ được chia thành nhiều tiến trình để tiến hành song song, các tiến trình này cần hợp tác với nhau để hoàn thành tác vụ ban đầu. Khi đó, chúng cũng cần HĐH cung cấp một cơ chế để có thể trao đổi thông tin với nhau.

Ngoài ra, khi chia hệ thống thành các module theo chức năng, khi kết hợp chúng cũng yêu cầu thiết lập quan hệ để trao đổi thông tin giữa chúng.

2.3.1.2. Các cơ chế thông tin liên lạc

a- Tín hiệu (Signal)

Tín hiệu là một cơ chế phần mềm tương tự như các ngắt cứng tác động đến các tiến trình. Một tín hiệu được sử dụng để thông báo cho tiến trình về một sự kiện nào đó xảy ra. Có nhiều tín hiệu được định nghĩa, mỗi một tín hiệu có một ý nghĩa tương ứng với một sự kiện đặc trưng.

Ví dụ : Một số tín hiệu của UNIX

Tín hiệu	Mô tả
SIGINT	Người dùng nhấn phím DEL để ngắt xử lý tiến trình
SIGQUIT	Yêu cầu thoát xử lý
SIGILL	Tiến trình xử lý một chỉ thị bất hợp lệ
SIGKILL	Yêu cầu kết thúc một tiến trình
SIGFPT	Lỗi floating – point xảy ra (chia cho 0)
SIGPIPE	Tiến trình ghi dữ liệu vào pipe mà không có reader
SIGSEGV	Tiến trình truy xuất đến một địa chỉ bất hợp lệ
SIGCLD	Tiến trình con kết thúc

Mỗi tiến trình sở hữu một bảng biểu diễn các tín hiệu khác nhau. Với mỗi tín hiệu sẽ có tương ứng một trình xử lý tín hiệu *(signal handler)* qui định các xử lý của tiến trình khi nhận được tín hiệu tương ứng.

Các tín hiệu được gửi đi bởi:

- Phần cứng (ví dụ lỗi do các phép tính số học)
- Hạt nhân HĐH gửi đến một tiến trình (ví dụ lưu ý tiến trình khi có một thiết bị nhập/xuất tự do).
- Một tiến trình gửi đến một tiến trình khác (ví dụ tiến trình cha yêu cầu một tiến trình con kết thúc)
- Người dùng (ví dụ nhấn phím Ctrl-C để ngắt xử lý của tiến trình)

Khi một tiến trình nhận một tín hiệu, nó có thể xử sự theo một trong các cách sau:

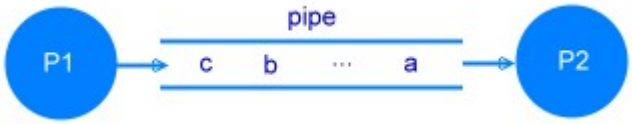
- Bỏ qua tín hiệu
- Xử lý tín hiệu theo kiểu mặc định
- Tiếp nhận tín hiệu và xử lý theo cách đặc biệt của tiến trình.

Liên lạc bằng tín hiệu mang tính chất *không đồng bộ*, nghĩa là một tiến trình nhận tín hiệu không thể xác định trước thời điểm nhận tính hiệu. Hơn nữa các tiến trình không thể kiểm tra được sự kiện tương ứng với tín hiệu có thật sự xảy ra? Cuối cùng, các tiến trình chỉ có thể thông báo cho nhau về một biến cố nào đó, mà không trao đổi dữ liệu theo cơ chế này được.

b- Pipe

Một pipe là một kênh liên lạc trực tiếp giữa hai tiến trình: dữ liệu xuất của tiến trình này được chuyển đến làm dữ liệu nhập cho tiến trình kia dưới dạng một dòng các byte.

Khi một pipe được thiết lập giữa hai tiến trình, một trong chúng sẽ ghi dữ liệu vào pipe và tiến trình kia sẽ đọc dữ liệu từ pipe. Thứ tự dữ liệu truyền qua pipe được bảo toàn theo nguyên tắc FCFS. Một pipe có kích thước giới hạn (thường là 4096 ký tự).



Một tiến trình chỉ có thể sử dụng một pipe do nó tạo ra hay kế thừa từ tiến trình cha. Hệ điều hành cung cấp các lời gọi hệ thống read/write cho các tiến trình thực hiện thao tác đọc/ghi dữ liệu trong pipe. Hệ điều hành cũng chịu trách nhiệm đồng bộ hóa việc truy xuất pipe trong các tình huống:

Tiến trình đọc pipe sẽ bị khóa nếu pipe trống, nó sẽ phải đợi đến khi pipe có dữ liệu để truy xuất.

Tiến trình ghi pipe sẽ bị khóa nếu pipe đầy, nó sẽ phải đợi đến khi pipe có chỗ trống để chứa dữ liệu.

Liên lạc bằng pipe là một cơ chế liên lạc *một chiều (unidirectional)*, nghĩa là một tiến trình kết nối với một pipe chỉ có thể thực hiện một trong hai thao tác đọc hoặc ghi, nhưng không thể thực hiện cả hai. Một số HĐH cho phép thiết lập hai pipe giữa một cặp

tiến trình để tạo liên lạc hai chiều. Trong những hệ thống đó, có nguy cơ xảy ra tình trạng *bế tắc*(deadlock): một pipe bị giới hạn về kích thước, do vậy nếu cả hai pipe nối kết hai tiến trình đều đầy (hoặc đều trống) và cả hai tiến trình đều muốn ghi (hay đọc) dữ liệu vào pipe (mỗi tiến trình ghi dữ liệu vào một pipe), chúng sẽ cùng bị khóa và chờ lẫn nhau mãi mãi!

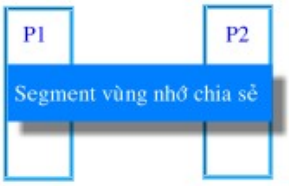
Cơ chế này cho phép truyền dữ liệu với cách thức không cấu trúc.

Ngoài ra, một giới hạn của hình thức liên lạc này là chỉ cho phép kết nối hai tiến trình có quan hệ cha-con, và trên cùng một máy tính.

c- Vùng nhớ chia sẻ

Cách tiếp cận của cơ chế này là cho nhiều tiến trình cùng truy xuất đến một vùng nhớ chung gọi là *vùng nhớ chia sẻ (shared memory)*. Không có bất kỳ hành vi truyền dữ liệu nào cần phải thực hiện ở đây, dữ liệu chỉ đơn giản được đặt vào một vùng nhớ mà nhiều tiến trình có thể cùng truy cập được.

Với phương thức này, các tiến trình chia sẻ một vùng nhớ vật lý thông qua trung gian không gian địa chỉ của chúng. Một vùng nhớ chia sẻ tồn tại độc lập với các tiến trình, và khi một tiến trình muốn truy xuất đến vùng nhớ này, tiến trình phải kết gắn vùng nhớ chung đó vào không gian địa chỉ riêng của từng tiến trình, và thao tác trên đó như một vùng nhớ riêng của mình.



Hình. Liên lạc qua vùng nhớ chia sẻ

Đây là phương pháp nhanh nhất để trao đổi dữ liệu giữa các tiến trình. Nhưng phương thức này cũng làm phát sinh các khó khăn trong việc bảo đảm sự toàn vẹn dữ liệu (*coherence*), ví dụ: làm sao biết được dữ liệu mà một tiến trình truy xuất là dữ liệu mới nhất mà tiến trình khác đã ghi? Làm thế nào ngăn cản hai tiến trình cùng đồng thời ghi dữ liệu vào vùng nhớ chung?… Rõ ràng vùng nhớ chia sẻ cần được bảo vệ bằng những cơ chế đồng bộ hóa thích hợp..

Một khuyết điểm của phương pháp liên lạc này là không thể áp dụng hiệu quả trong các hệ phân tán, để trao đổi thông tin giữa các máy tính khác nhau.

d- Trao đổi thông điệp (Message)

Hệ điều hành còn cung cấp một cơ chế liên lạc giữa các tiến trình không thông qua việc chia sẻ một tài nguyên chung, mà thông qua việc gửi thông điệp. Để hỗ trợ cơ chế liên lạc bằng thông điệp, hệ điều hành cung cấp các hàm IPC chuẩn (Interprocess communication), cơ bản là hai hàm:

Send (message): gửi một thông điệp

Receive (message): nhận một thông điệp

Nếu hai tiến trình P và Q muốn liên lạc với nhau, cần phải thiết lập một mối liên kết giữa hai tiến trình, sau đó P, Q sử dụng các hàm IPC thích hợp để trao đổi thông điệp, cuối cùng khi sự liên lạc chấm dứt mối liên kết giữa hai tiến trình sẽ bị hủy. Có nhiều cách thức để thực hiện sự liên kết giữa hai tiến trình và cài đặt các theo tác send /receive tương ứng: liên lạc trực tiếp hay gián tiếp, liên lạc đồng bộ hoặc không đồng bộ, kích thước thông điệp là cố định hay không… Nếu các tiến trình liên lạc theo kiểu liên kết tường minh, các hàm Send và Receive sẽ được cài đặt với tham số:

Send (destination, message): gửi một thông điệp đến *destination*

Receive (source,message): nhận một thông điệp từ *source*

Đơn vị truyền thông tin trong cơ chế trao đổi thông điệp là một thông điệp, do đó các tiến trình có thể trao đổi dữ liệu ở dạng có cấu trúc.

Định dạng thông điệp

Loại thông điệp
Địa chỉ đích
Địa chỉ nguồn
Độ dài thông điệp
Các thông tin điều kiển
Nội dung thông điệp

e- Sockets

Một socket là một thiết bị truyền thông hai chiều tương tự như tập tin, chúng ta có thể đọc hay ghi lên nó, tuy nhiên mỗi socket là một thành phần trong một mối nối nào đó giữa các máy trên mạng máy tính và các thao tác đọc/ghi chính là sự trao đổi dữ liệu giữa các ứng dụng trên nhiều máy khác nhau.

Sử dụng socket có thể mô phỏng hai phương thức liên lạc trong thực tế: liên lạc thư tín (socket đóng vai trò bưu cục) và liên lạc điện thoại (socket đóng vai trò tổng đài)

Các thuộc tính của socket:

- Domaine: định nghĩa dạng thức địa chỉ và các nghi thức sử dụng. Có nhiều domaines, ví dụ UNIX, INTERNET, XEROX_NS,...
- Type: định nghĩa các đặc điểm liên lạc:

Sự tin cậy

Sự bảo toàn thứ tự dữ liệu

Lặp lại dữ liệu

Chế độ nối kết

Bảo toàn giới hạn thông điệp

Khả năng gửi thông điệp khẩn

Để thực hiện liên lạc bằng socket, cần tiến hành các thao tác:

Tạo lập hay mở một socket

Gắn kết một socket với một địa chỉ

Liên lạc: có hai kiểu liên lạc tùy thuộc vào chế độ nối kết:

Liên lạc trong chế độ không liên kết : liên lạc theo hình thức hộp thư:

Hai tiến trình liên lạc với nhau không kết nối trực tiếp

Mỗi thông điệp phải kèm theo địa chỉ người nhận.

Hình thức liên lạc này có đặc điểm được:

Người gửi không chắc chắn thông điệp của học được gửi đến người nhận,

Một thông điệp có thể được gửi nhiều lần,

Hai thông điệp được gửi theo một thứ tự nào đó có thể đến tay người nhận theo một thứ tự khác.

Một tiến trình sau khi đã mở một socket có thể sử dụng nó để liên lạc với nhiều tiến trình khác nhau nhờ sử hai primitive *send* và *receive*.

Liên lạc trong chế độ nối kết

Một liên kết được thành lập giữa hai tiến trình. Trước khi mối liên kết này được thiết lập, một trong hai tiến trình phải đợi có một tiến trình khác yêu cầu kết nối. Có thể sử dụng socket để liên lạc theo mô hình client-server. Trong mô hình này, server sử dụng lời gọi hệ thống listen và accept để nối kết với client, sau đó, client và server có thể trao đổi thông tin bằng cách sử dụng các primitive send và receive.

Hủy một socket

Cơ chế socket có thể sử dụng để chuẩn hoá mối liên lạc giữa các tiến trình vốn không liên hệ với nhau, và có thể hoạt động trong những hệ thống khác nhau.

2.3.2. Đồng bộ hóa tiến trình

Trong một hệ thống cho phép các tiến trình liên lạc với nhau, bao giờ hệ điều hành cũng cần cung cấp kèm theo những cơ chế đồng bộ hóa để bảo đảm hoạt động của các tiến trình đồng hành không tác động sai lệch đến nhau. Truy xuất đồng hành dữ liệu được chia sẻ có thể dẫn tới việc không đồng nhất dữ liệu. Để đồng bộ hóa tiến trình, cần có các cơ chế đảm bảo việc thực thi có thứ tự của các tiến trình hợp tác chia sẻ không gian địa chỉ để tính đúng đắn của dữ liệu luôn được duy trì.

2.3.2.1. Nhu cầu đồng bộ hóa tiến trình

a- Yêu cầu độc quyền truy xuất (Mutual exclusion)

Các tài nguyên trong hệ thống được phân thành hai loại: tài nguyên có thể chia sẻ cho phép nhiều tiến trình đồng thời truy xuất, và tài nguyên không thể chia sẻ chỉ chấp nhận một (hay một số lượng hạn chế) tiến trình sử dụng tại một thời điểm. Tính không thể chia sẻ của tài nguyên thường có nguồn gốc từ một trong hai nguyên nhân sau đây:

Đặc tính cấu tạo phần cứng của tài nguyên không cho phép chia sẻ.

Nếu nhiều tiến trình sử dụng tài nguyên đồng thời, có nguy cơ xảy ra các kết quả không dự đoán được do hoạt động của các tiến trình trên tài nguyên ảnh hưởng lẫn nhau.

Để giải quyết vấn đề, cần bảo đảm tiến trình độc quyền truy xuất tài nguyên, nghĩa là hệ thống phải kiểm soát sao cho tại một thời điểm, chỉ có một tiến trình được quyền truy xuất một tài nguyên không thể chia sẻ.

b- Yêu cầu phối hợp (Synchronization)

Nhìn chung, mối tương quan về tốc độ thực hiện của hai tiến trình trong hệ thống là không thể biết trước, vì điều này phụ thuộc vào nhiều yếu tố động như tần suất xảy ra các ngắt của từng tiến trình, thời gian tiến trình được cấp phát bộ xử lý… Có thể nói rằng các tiến trình hoạt động không đồng bộ với nhau. Nhưng có những tình huống các tiến trình cần hợp tác trong việc hoàn thành tác vụ, khi đó cần phải đồng bộ hóa hoạt động của các tiến trình, ví dụ một tiến trình chỉ có thể xử lý nếu một tiến trình khác đã kết thúc một công việc nào đó…

2.3.2.2. Tài nguyên găng và miền găng

a- Tài nguyên găng (Critical Resource)

Tài nguyên găng là tài nguyên mà tại mỗi thời điểm, chỉ có thể phục vụ được một tiến trình; khi có 2 hay nhiều tiến trình cùng có nhu cầu đối với tài nguyên thì tài nguyên đó trở thành tài nguyên găng.

Những tài nguyên găng thường phải chia sẻ cho nhiều tiến trình hoạt động đồng thời dùng chung, và có nguy cơ dễ dẫn đến sự tranh chấp giữa các tiến trình này khi sử dụng chúng. Tài nguyên găng có thể là tài nguyên phần cứng hoặc tài nguyên phần mềm, có thể là tài nguyên phân chia được hoặc không phân chia được, nhưng đa số thường là tài nguyên phân chia được như là: các biến chung, các file chia sẻ.

Trong môi trường HĐH đa nhiệm - đa chương – đa người sử dụng, việc chia sẻ tài nguyên cho các tiến trình của người sử dụng dùng chung là cần thiết, nhưng nếu HĐH không tổ chức tốt việc sử dụng tài nguyên dùng chung của các tiến trình hoạt động đồng thời, thì không những không mang lại hiệu quả khai thác tài nguyên của hệ thống mà còn làm hỏng dữ liệu của các ứng dụng. Nguy hiểm hơn, việc hỏng dữ liệu này có thể HĐH và ứng dụng không thể phát hiện được. Việc hỏng dữ liệu của ứng dụng có thể làm sai lệch ý nghĩa thiết kế của nó. Đây là điều mà cả HĐH và người lập trình đều không mong muốn.

Các tiến trình hoạt động đồng thời thường cạnh tranh với nhau trong việc sử dụng tài nguyên dùng chung. Hai tiến trình hoạt động đồng thời cùng ghi vào một không gian nhớ chung (một biến chung) trên bộ nhớ hay hai tiến trình đồng thời cùng ghi dữ liệu vào một file chia sẻ, đó là những biểu hiện của sự cạnh tranh về việc sử dụng tài nguyên dùng chung của các tiến trình. Để các tiến trình hoạt động đồng thời không cạnh tranh hay xung đột với nhau khi sử dụng tài nguyên dùng chung HĐH phải tổ chức cho các tiến trình này được độc quyền truy xuất/ sử dụng trên các tài nguyên dùng chung này.

b- Đoạn găng (Critical Section)

Đoạn chương trình (code) trong các tiến trình đồng thời, có chứa các tài nguyên găng, được gọi là đoạn găng hay miền găng.

Để hạn chế các lỗi có thể xảy ra do sử dụng tài nguyên găng, HĐH phải điều khiển các tiến trình sao cho, tại một thời điểm chỉ có một tiến trình nằm trong đoạn găng, nếu có nhiều tiến trình cùng muốn vào (thực hiện) đoạn găng thì chỉ có một tiến trình được vào, các tiến trình khác phải chờ, một tiến trình khi ra khỏi (kết thúc) đoạn găng phải báo cho HĐH và/hoặc các tiến trình khác biết để các tiến trình này vào đoạn găng,… Công tác của HĐH điều khiển tiến trình thực hiện đoạn găng được gọi là đồng bộ hóa tiến trình hay *điều độ tiến trình qua đoạn găng*.

Việc điều độ tiến trình phải đạt được các yêu cầu sau:

1. Tại một thời điểm chỉ có một tiến trình được nằm trong đoạn găng.

2. Nếu có nhiều tiến trình đồng thời cùng yêu cầu vào đoạn găng thì chỉ có một tiến trình được phép vào đoạn găng, các tiến trình khác phải xếp hàng chờ trong hàng đợi.

3. Không cho phép một tiến trình nằm vô hạn trong đoạn găng và không cho phép một tiến trình chờ vô hạn trước đoạn găng (chờ trong hàng đợi).

4. Nếu tài nguyên găng được giải phóng thì HĐH có nhiệm vụ đánh thức các tiến trình trong hàng đợi ra để tạo điều kiện cho nó vào đoạn găng.

Nguyên lý cơ bản của điều độ là tổ chức truy xuất độc quyền trên tài nguyên găng, nhưng sự bắt buộc độc quyền này còn tồn tại hai hạn chế lớn là có thể dẫn đến bế tắc(Deadlock) trong hệ thống và có thể các tiến trình bị đói tài nguyên (Starvation).

2.3.2.3. Các giải pháp

a- Các giải pháp phần cứng

** Dùng cặp chỉ thị STI & CLI*

Một số vi xử lý cung cấp cặp chỉ thị CLI và STI để người lập trình thực hiện các thao tác mở ngắt (STI: Setting Interrupt) và cấm ngắt (CLI: Clean Interrupt) của hệ thống trong lập trình.

Trước khi vào đoạn găng tiến trình thực hiện chỉ thị CLI, để yêu cầu cấm các ngắt trong hệ thống, khi đó ngắt đồng hồ không thể phát sinh, nghĩa là không có một tiến trình nào khác có thể phát sinh, nhờ đó mà tiến trình trong đoạn găng toàn quyền sử dụng tài nguyên găng cho đến hết thời gian xử lý của nó.

Khi kết thúc truy xuất tài nguyên găng, tiến trình ra khỏi đoạn găng, tiến trình thực hiện chỉ thị STI để cho phép ngắt trở lại. Khi đó các tiến trình khác có thể tiếp tục hoạt động và có thể vào đoạn găng.

Sơ đồ điều độ này đơn giản, dễ cài đặt. Tuy nhiên, cần phải có sự hỗ trợ của vi xử lý và dễ gây ra hiện tượng treo toàn bộ hệ thống, khi tiến trình trong đoạn găng không có khả năng ra khỏi đoạn găng. Tiến trình không ra khỏi đoạn găng nên nó không thể thực hiện chỉ thị STI để mở ngắt cho hệ thống, nên hệ thống bị treo hoàn toàn.

Giải pháp này không thể sử dụng trên các hệ thống multiprocessor, vì CLI chỉ cấm ngắt trên vi xử lý hiện tại chứ không thể cấm ngắt của các vi xử lý khác. Tức là, sau khi đã cấm ngắt, tiến trình trong đoạn găng vẫn có thể bị tranh chấp tài nguyên găng bởi các tiến trình trên các vi xử lý khác trong hệ thống.

** Dùng chỉ thị TSL (Test and Set)*

Để tổ chức điều độ tiến trình với TSL chương trình phải sử dụng biến chia sẻ Lock, khởi gán bằng 0. Theo đó, mỗi tiến trình trước khi vào đoạn găng phải kiểm tra giá trị của Lock. Nếu Lock = 0 thì vào đoạn găng. Nếu Lock = 1 thì phải đợi cho đến khi Lock = 0. Như vậy, trước khi vào đoạn găng tiến trình phải gọi hàm TestAndSetLock, để kiểm tra giá trị trả về của hàm này:

- Nếu bằng False, là đang có một tiến trình trong đoạn găng, thì phải chờ cho đến khi hàm trả về True, có một tiến trình vừa ra khỏi đoạn găng.

- Nếu bằng True, thì tiến trình sẽ vào đoạn găng để sử dụng tài nguyên găng. Khi kết thúc sử dụng tài nguyên găng ra khỏi đoạn găng thì tiến trình phải đặt lại giá trị của Lock, Lock = 0, để các tiến trình khác có thể vào đoạn găng.

Sơ đồ này đơn giản, dễ cài đặt nhưng cần phải có sự hỗ trợ của vi xử lý. Ngoài ra nó còn một hạn chế lớn là gây lãng phí thời gian xử lý của processor do tồn tại hiện tượng chờ đợi tích cực trong sơ đồ (While (TestAndSetlock(lock)) DO;). Hiện tượng chờ đợi tích cực là hiện tượng processor chỉ chờ một sự kiện nào đó xảy ra mà không làm gì cả.

Tóm lại: Việc sử dụng các chỉ thị phần cứng đặc biệt để tổ chức điều độ tiến trình qua đoạn găng, hay còn gọi là tổ chức truy xuất độc quyền trên tài nguyên găng, có những ưu và nhược điểm sau đây:

Ưu điểm:

- Thích hợp với một số lượng bất kỳ các tiến trình cả trên hệ thống Uniprocessor và hệ thống Multiprocessor.
- Khá đơn giản cho nên dễ xác định độ chính xác.
- Có thể được sử dụng để hỗ trợ cho nhiều đoạn găng; mỗi đoạn găng có thể định nghĩa cho nó một biến riêng.

Nhược điểm:

- Trong khi một tiến trình đang chờ đợi được vào đoạn găng thì nó tiếp tục làm tốn thời gian xử lý của processor, mà ta gọi là chờ đợi tích cực.
- Sự đói tài nguyên có thể xảy ra. Khi một tiến trình rời khỏi một đoạn găng, bộ phận điều độ tiến trình phải chọn một tiến trình trong số nhiều tiến trình ngoài đoạn găng để cho nó vào đoạn găng. Việc chọn này có thể dẫn đến hiện tượng có một tiến trình đợi mãi mà không thể vào đoạn găng được.
- Sự bế tắccó thể xảy ra. Hãy xét một tình huống trên một hệ thống uniprocessor. Tiến trình P1 thực thi chỉ thị đặc biệt (TesAndSetLock, Exchange) và vào đoạn găng của nó. P1 sau đó bị ngắt để nhường processor cho P2, P2 là tiến trình có độ ưu tiên cao hơn. Nếu như P2 cũng định sử dụng tài nguyên như P1, P2 sẽ bị từ chối truy xuất bởi vì cơ chế độc quyền. Do đó P2 sẽ đi vào vòng lặp busy-waitting. Tuy nhiên, P1 sẽ không bao giờ được cấp processor để tiếp tục vì nó có độ ưu tiên thấp hơn so với P2.

b- Các giải pháp dùng biến khoá

** Dùng biến khóa chung*

Xuất phát từ nguyên tắc cơ bản của tổ chức độc quyền là, tại mỗi thời điểm chỉ có duy nhất một tiến trình có thể truy xuất đến một vùng nhớ chia sẻ, các HĐH sử dụng biến khoá chung để tổ chức truy xuất độc quyền trên tài nguyên găng. Phương pháp này còn gọi là phương pháp Busy and Waitting (bận và đợi), nó được nhà toán học người Hà Lan tên là Dekker đề xuất.

Với mỗi tài nguyên găng, HĐH dùng một biến chung để điều khiển việc sử dụng tài nguyên này của các tiến trình đồng thời. Tạm gọi là biến chung này là Lock, Lock được chia sẻ cho nhiều tiến trình và được khởi gán = 0.

Theo đó, mỗi tiến trình trước khi vào đoạn găng phải kiểm tra giá trị của Lock:

- Nếu Lock = 1, tức là đã có tiến trình nào đó trong đoạn găng, thì tiến trình phải chờ cho đến khi Lock = 0 (có thể chuyển sang trạng thái blocked để chờ).
- Nếu Lock = 0, tức là không có tiến trình nào trong đoạn găng, thì tiến trình thiết lập quyền vào đoạn găng, đặt Lock = 1, và vào đoạn găng. Tiến trình vừa ra khỏi đoạn găng phải đặt Lock = 0, để các tiến trình khác có thể vào đoạn găng.

Sơ đồ điều độ dùng biến khoá chung này đơn giản, dễ xây dựng nhưng vẫn xuất hiện hiện tượng chờ đợi tích cực, khi chờ cho đến khi Lock = 0 (While Lock = 1 DO;). Hiện tượng chờ đợi tích cực gây lãng phí thời gian của processor.

Nếu một tiến trình trong đoạn găng không thể ra khỏi đoạn găng, thì các tiến trình chờ ngoài đoạn găng có thể chờ đợi vô hạn (vì Lock không được đặt lại = 0).

** Dùng biến khóa riêng*

Mỗi tiến trình sử dụng một biến khoá Lock riêng, tương ứng với một tài nguyên găng trong hệ thống. Biến khoá riêng của tất cả các tiến trình đều được khởi gán bằng 0, tức là chưa vào đoạn găng.

Theo đó, mỗi tiến trình trước khi vào đoạn găng ứng với một tài nguyên găng nào đó thì trước hết phải kiểm tra biến khoá riêng, tương ứng với tài nguyên găng mà tiến trình muốn truy xuất, của tất cả các tiến trình còn lại:

- Nếu tồn tại một biến khoá riêng của một tiến trình nào đó bằng 1, Lock = 1, tức là đã có một tiến trình nào đó ở trong đoạn găng, thì tiến trình phải chờ ngoài đoạn găng cho đến khi tất cả biến khoá riêng = 0.
- Nếu tất cả các biến khóa riêng của các tiến trình đều = 0, Lock = 0, tức là không có tiến trình nào trong đoạn găng, thì tiến trình thiết lập quyền vào đoạn găng, đặt Lock = 1, và vào đoạn găng. Tiến trình vừa ra khỏi đoạn găng phải đặt Lock = 0, để các tiến trình khác có thể vào đoạn găng.

Sơ đồ này đơn giản dễ cài đặt. Một tiến trình nào đó ở ngoài đoạn găng bị blocked sẽ không ngăn cản được các tiến trình khác vào đoạn găng, nhưng nếu tiến trình trong đoạn găng bị lỗi không thể ra khỏi đoạn găng, Lock luôn luôn = 0, thì các tiến trình khác sẽ không được quyền vào đoạn găng.

Phương pháp này vẫn còn tồn tại hiện tượng chờ đợi tích cực và sơ đồ điều độ sẽ trở nên phức tạp khi có nhiều hơn hai tiến trình muốn vào đoạn găng.

c- Các giải pháp được hỗ trợ bởi hệ điều hành và ngôn ngữ lập trình

Các giải pháp trên tồn tại hiện tượng chờ đợi tích cực, gây lãng phí thời gian xử lý của processor. Điều này có thể khắc phục bằng một nguyên tắc rất cơ bản: nếu một tiến trình khi chưa đủ điều kiện vào đoạn găng thì được chuyển ngay sang trạng thái blocked để nó trả lại processor cho hệ thống, để hệ thống cấp cho tiến trình khác. Để thực hiện được điều này cần phải có sự hỗ trợ của HĐH và các ngôn ngữ lập trình để các tiến trình có thể chuyển trạng thái của nó. Hai thủ tục Sleep và Wakeup được HĐH cung cấp để sử dụng cho mục đích này:

- Khi tiến trình chưa đủ điều kiện vào đoạn găng nó sẽ thực hiện một lời gọi hệ thống để gọi Sleep để chuyển nó sang trạng thái blocked, và tiến trình được gọi này đưa vào hàng đợi để đợi cho đến khi có một tiến trình khác gọi thủ tục Wakeup để giải phóng nó ra khỏi hàng đợi và có thể đưa nó vào đoạn găng.
- Một tiến trình khi ra khỏi đoạn găng phải gọi Wakeup để đánh thức một tiến trình trong hàng đợi blocked ra để tạo điều kiện cho tiến trình này vào đoạn găng.

Như vậy giải pháp này được áp dụng trên nhóm các tiến trình hoạt động đồng thời có trao đổi thông tin với nhau, và các tiến trình phải hợp tác với nhau để hoàn thành nhiệm vụ. Các tiến trình này liên lạc với nhau bằng cách gửi tín hiệu cho nhau. Một tiến trình trong hệ thống này có thể bị buộc phải dừng (bị blocked) cho đến khi nhận được một tín hiệu nào đó từ tiến trình bên kia, đó là tiến trình hợp tác với nó.

Thực tế đã chỉ ra được rằng, nếu chỉ dùng hai thủ tục trên thì sơ đồ điều độ sẽ không đáp ứng được các yêu cầu của công tác điều độ, do đó khi cài đặt các HĐH chỉ sử dụng ý tưởng của Sleep và Wakeup. Sau đây là các giải pháp sử dụng ý tưởng của Sleep và Wakeup.

** Giải pháp dùng Semaphore (đèn báo)*

Giải pháp này được Dijkstra đề xuất vào năm 1965. Semaphore được định nghĩa để sử dụng trong các sơ đồ điều độ như sau:

- Semaphore S là một biến nguyên, khởi gán bằng một giá trị không âm, đó là khả năng phục vụ của tài nguyên găng tương ứng với nó.
- Ứng với S có một hàng đợi F(s) để lưu các tiến trình đang bị blocked trên S.
- Chỉ có hai thao tác Down và Up được tác động đến semaphore S. Down giảm S xuống một đơn vị, Up tăng S lên một đơn vị.
- Mỗi tiến trình trước khi vào đoạn găng thì phải gọi Down để kiểm tra và xác lập quyền vào đoạn găng. Khi tiến trình gọi Down(S) thì hệ thống sẽ thực hiện như sau: S:= S-1, nếu S > = 0 thì tiến trình tiếp tục xử lý và vào đoạn găng, nếu S < 0 thì tiến trình phải vào hàng đợi để chờ cho đến khi S > = 0. Down được cài đặt như sau:

```
Procedure  Down(s);
Begin
        S := S -1;
        If  S < 0 Then                          {S >= 0 thì tiếp tục}
          Begin
                Status(p)= blocked;             {chuyển tiến trình sang blocked}
                Enter(p, F(s));         {đưa tiến trình vào hàng đợi F(S)}
          end;
End;
```

- Mỗi tiến trình ngay sau khi ra khỏi đoạn găng phải gọi Up để kiểm tra xem có tiến trình nào đang đợi trong hàng đợi hay không, nếu có thì đưa tiến trình trong hàng đợi vào đoạn găng. Khi tiến trình gọi Up thì hệ thống sẽ thực hiện như sau: S:= S + 1, nếu S <= 0 đưa một tiến trình trong F(s) vào đoạn găng. Up được cài đặt như sau:

```
Procedure  Up(s);
Begin
        S := S + 1;
```

```
        If S <= 0 Then
            Begin
                Exit(Q,F(s) );              {đưa tiến trình ra khỏi F(S)}
                Status(Q) = ready ;      {chuyển tiến trình sang ready}
                Enter(Q, ready-list );     {đưa tiến trình vào ready list}
             End;
    End;
```

Ở đây chúng ta cần lưu ý rằng: Down và Up là các thủ tục của HĐH, nên HĐH đã cài đặt cơ chế độc quyền cho nó, tức là các lệnh bên trong nó không thể tách rời nhau. Hai thủ tục Down(S) và Up(S) đưa ra ở trên chỉ để minh họa cho nguyên lý hoạt động của Down và Up.

Sử dụng semaphore để điều độ tiến trình, mang lại những thuận lợi sau:

- Mỗi tiến trình chỉ kiểm tra quyền vào đoạn găng một lần, khi chờ nó không làm gì cả, tiến trình ra khỏi đoạn găng phải đánh thức nó.
- Không xuất hiện hiện tượng chờ đợi tích cực, nên khai thác tối đa thời gian xử lý của processor.
- Nhờ cơ chế hàng đợi mà HĐH có thể thực hiện gán độ ưu tiên cho các tiến trình khi chúng ở trong hành đợi.
- Trị tuyệt đối của S cho biết số lượng các tiến trình đang đợi trên F(S).

Nên nhớ rằng, Down và Up là các thủ tục của HĐH nên sơ đồ điều độ sẽ bị thay đổi khi thay đổi HĐH. Đây là một trở ngại của việc sử dụng semaphore để tổ chức điều độ tiến trình.

Ví dụ 1: Sự thực hiện của hai tiến trình A và B trong sơ đồ điều độ trên

TT	P thực hiện	Down/ Up	S	P hoạt động	P trong hàng đợi
0	-	-	**1**	-	-
1	A	Down	0	A	-
2	B	Down	-1	A	B
3	A	Up	0	B	A
4	A	Down	-1	B	A
5	B	Up	0	A	-
6	B	Down	1	-	-

Ví dụ 2: Sơ đồ điều độ hệ thống có 6 tiến trình hoạt động đồng thời, cùng sử dụng tài nguyên găng, cơ chế truy xuất độc quyền, được mô tả như sau:

- Có 6 tiến trình yêu cầu sử dụng tài nguyên găng tương ứng với S lần lượt là:

  A B C D E F
- Độ ưu tiên của các tiến trình là (5 là độ ưu tiên cao nhất):

  1 1 2 4 2 5
- Thời gian các tiến trình cần sử dụng tài nguyên găng là:

  4 2 2 2 1 1

TT	P thực hiện	Down/ Up	S	P hoạt động	P trong hàng đợi
0	-	-	**1**	-	-
1	A	Down	0	A	-
2	B	Down	-1	A	B
3	C	Down	-2	A	C B

4	D	Down	-3	A	D	C	B
5	A	Up	-2	D	C	B	
6	E	Down	-3	D	C	E	B
7	D	Up	-2	C	E	B	
8	F	Down	-3	C	F	E	B
9	C	Up	-2	F	E	B	
10	F	Up	-1	E	B		
11	E	Up	0	B	-		
12	B	Up	**1**	-	-		

Chú ý: Thứ nhất, trị tuyệt đối của S cho biết số lượng các tiến trình trong hành đợi F(S). Thứ hai, tiến trình chưa được vào đoạn găng thì được đưa vào hàng đợi và tiến trình ra khỏi đoạn găng sẽ đánh thức tiến trình có độ ưu tiên cao nhất trong hàng đợi để đưa nó vào đoạn găng. Tiến trình được đưa vào hàng đợi sau nhưng có độ ưu tiên cao hơn sẽ được đưa vào đoạn găng trước các tiến trình được đưa vào hàng đợi trước nó.

** Giải pháp dùng Monitors*

Giải pháp này được Hoar đề xuất năm 1974 sau đó vào năm 1975 được Brinch & Hansnen đề xuất lại. Monitor là cấu trúc phần mềm đặc biệt được cung cấp bởi ngôn ngữ lập trình, nó bao gồm các thủ tục, các biến và các cấu trúc dữ liệu được định nghĩa bởi Monitor. Chương trình Monitor có các tính chất sau đây:

- Các biến và cấu trúc dữ liệu bên trong monitor chỉ có thể được thao tác bởi các thủ tục được định nghĩa bên trong monitor đó.
- Một tiến trình muốn vào monitor phải gọi một thủ tục của monitor đó.
- Tại một thời điểm, chỉ có một tiến trình duy nhất được hoạt động bên trong monitor. Các tiến trình khác đã gọi monitor phải hoãn lại để chờ monitor rảnh.

Hai tính chất 1 và 2 tương tự như các tính chất của các đối tượng trong lập trình hướng đối tượng. Như vậy một HĐH hoặc một ngôn ngữ lập trình hướng đối tượng có thể cài đặt monitor như là một đối tượng có các tính chất đặc biệt.

Với tính chất thứ 3 monitor có khả năng thực hiện các cơ chế độc quyền, các biến trong monitor có thể được truy xuất chỉ bởi một tiến trình tại một thời điểm. Như vậy các cấu trúc dữ liệu dùng chung có thể được bảo vệ bằng cách đặt chúng bên trong monitor. Nếu dữ liệu bên trong monitor là tài nguyên găng thì monitor cung cấp sự độc quyền trong việc truy xuất đến tài nguyên găng đó.

Monitor cung cấp các công cụ đồng bộ hoá để người lập trình sử dụng trong các sơ đồ điều độ. Công cụ đồng bộ hoá được định nghĩa để sử dụng trong các sơ đồ điều độ như sau: Trong một monitor, có thể định nghĩa các *biến điều kiện* và hai thao tác kèm theo là Wait và Signal, chỉ có wait và signal được tác động đến các biến điều kiện.

** Giải pháp trao đổi Message (thông điệp)*

Khi các tiến trình có sự tương tác với tiến trình khác, hai yêu cầu cơ bản cần phải được thỏa mãn đó là: sự đồng bộ hoá (synchronization) và sự truyền thông (communication). Các tiến trình phải được đồng bộ để thực hiện độc quyền. Các tiến trình hợp tác có thể cần phải trao đổi thông tin. Một hướng tiếp cận để cung cấp cả hai chức năng đó là sự truyền thông điệp (message passing). Truyền thông điệp có ưu điểm là có thể thực hiện được trên cả hai hệ thống uniprocessor và multiprocessor, khi các hệ thống này hoạt động trên mô hình bộ nhớ chia sẻ

Các hệ thống truyền thông điệp có thể có nhiều dạng. Dạng chung nhất mà trong đó đề cập đến các đặc trưng có trong nhiều hệ thống khác nhau. Các hàm của truyền thông điệp trên thực tế có dạng tương tự như hai hàm sau:

- Send(destination, message): gửi thông điệp đến tiến trình đích.
- Receive(source, message): nhận thông điệp từ tiến trình nguồn.

Một tiến trình gửi thông tin dưới dạng một thông điệp (message) đến một tiến trình khác, bằng hàm Send, được nhận biết bởi tham số destination. Một tiến trình nhận thông điệp (message), bằng hàm Receive, từ một tiến trình được nhận biết bởi tham số source. Tiến trình gọi Receive phải chờ cho đến khi nhận được message từ tiến trình source thì mới có thể tiếp tục được.

Việc sử dụng Send và Receive để tổ chức điều độ được thực hiện như sau:

- Có một tiến trình kiểm soát việc sử dụng tài nguyên găng.
- Tiến trình có yêu cầu tài nguyên găng sẽ gửi một thông điệp đến tiến trình kiểm soát và sau đó chuyển sang trạng thái blocked cho đến khi nhận được một thông điệp chấp nhận cho truy xuất từ tiến trình kiểm soát tài nguyên găng.
- Khi sử dụng xong tài nguyên găng, tiến trình vừa sử dụng tài nguyên găng gửi một thông điệp khác đến tiến trình kiểm soát để báo kết thúc truy xuất.
- Tiến trình kiểm soát, khi nhận được thông điệp yêu cầu tài nguyên găng, nó sẽ chờ cho đến khi tài nguyên găng sẵn sàng để cấp phát thì gửi một thông điệp đến tiến trình đang bị khoá trên tài ngyên đó để đánh thức tiến trình này.

Giải pháp này thường được cài đặt trên các hệ thống mạng máy tính, đặc biệt là trên các hệ thống mạng phân tán. Đây là lợi thế mà semaphore và monitor không có được.

2.4. BẾ TẮC VÀ PHÒNG CHỐNG BẾ TẮC

2.4.1. Bế tắc (deadlock)

Bế tắc (bế tắc) là hiện tượng trong hệ thống có hai hay nhiều tiến trình cùng chờ đợi một sự kiện, mà nếu sự kiện đó không xảy ra thì sự chờ đợi ấy là chờ vô hạn. Sự kiện thường là chờ được cấp tài nguyên, và sự đợi này có thể kéo dài vô hạn nếu không có sự tác động từ bên ngoài.

Năm 1971, Coffman đã đưa ra và chứng tỏ được rằng, nếu hệ thống tồn tại đồng thời bốn điều kiện sau đây thì hệ thống sẽ xảy ra tắc nghẽn:

1. Tồn tại tài nguyên găng (điều kiện độc quyền): sử dụng tài nguyên này là loại trừ lẫn nhau (mutual exclusion) hay độc quyền sử dụng: Đối với các tài nguyên không phân chia được thì tại mỗi thời điểm chỉ có một tiến trình sử dụng được tài nguyên.

2. Giữ và đợi (hold and wait): Một tiến trình hiện tại đang chiếm giữ tài nguyên, lại xin cấp phát thêm tài nguyên mới.

Trong nhiều trường hợp các điều kiện trên là rất cần thiết đối với hệ thống. Sự thực hiện độc quyền là cần thiết để bảo đảm tính đúng đắn của kết quả và tính toàn vẹn của dữ liệu (chúng ta đã thấy điều này ở phần tài nguyên găng trên đây). Tương tự, sự ưu

tiên không thể thực hiện một cách tuỳ tiện, đặt biệt đối với các tài nguyên có liên quan với nhau, việc giải phóng từ một tiến trình này có thể ảnh hưởng đên kết quả xử lý của các tiến trình khác.

3. Đợi vòng tròn (Circular wait): Đây là trường hợp của ví dụ 1 mà chúng ta đã nêu ở trên. Tức là, mỗi tiến trình đang chiếm giữ tài nguyên mà tiến trình khác đang cần.

Sự bế tắccó thể tồn tại với ba điều kiện trên, nhưng cũng có thể không xảy ra chỉ với 3 điều kiện đó. Để chắc chắn bế tắc xảy ra cần phải có điều kiện thứ tư

4. Không có sự phân phối lại tài nguyên, không có tài nguyên nào có thể được giải phóng từ một tiến trình đang chiếm giữ nó.

Ba điều kiện đầu là điều kiện cần chứ không phải là điều kiện đủ để xảy ra tắc nghẽn. Điều kiện thứ tư là kết quả tất yếu từ ba điều kiện đầu.

2.4.2. Phòng chống bế tắc (Deadlock Prevention)

Có 3 quan điểm phòng chống bế tắc, tùy thuộc vào mức độ xảy ra bế tắc và các hậu quả mà nó gây ra. Đó là phòng ngừa, dự báo và phòng tránh, nhận biết và khắc phục.

2.4.2.1. Phòng ngừa

Phòng chống bế tắc là thiết kế một hệ thống sao cho không để hiện tượng bế tắc xảy ra. Các phương thức phòng chống bế tắc đều tập trung giải quyết bốn điều kiện gây ra tắc nghẽn, sao cho hệ thống không thể xảy ra đồng thời bốn điều kiện trên.

- Đối với điều kiện tồn tại tài nguyên găng (điều kiện độc quyền): Điều kiện này gần như không tránh khỏi, vì sự độc quyền là cần thiết đối với tài nguyên thuộc loại phân chia được như các biến chung, các tập tin chia sẻ, HĐH cần phải hỗ trợ sự độc quyền trên các tài nguyên này. Tuy nhiên, với những tài nguyên thuộc loại không phân chia được hệ điều hành có thể sử dụng kỹ thuật SPOOL (Simultaneous Peripheral Operation Online) để tạo ra nhiều tài nguyên ảo cung cấp cho các tiến trình đồng thời.
- Đối với điều kiện giữ và đợi: Điều kiện này có thể ngăn chặn bằng cách buộc tiến trình yêu cầu tất cả tài nguyên mà nó cần tại một thời điểm và tiến trình sẽ bị khoá (blocked) cho đến khi yêu cầu tài nguyên của nó được HĐH đáp ứng. Phương pháp này không hiệu quả. Thứ nhất, tiến trình phải đợi trong một khoảng thời gian dài để có đủ tài nguyên mới có thể chuyển sang hoạt động được, trong khi tiến trình chỉ cần một số ít tài nguyên trong số đó là có thể hoạt động được, sau đó yêu cầu tiếp. Thứ hai, lãng phí tài nguyên, vì có thể tiến trình giữ nhiều tài nguyên mà chỉ đến khi sắp kết thúc tiến trình mới sử dụng, và có thể đây là những tài nguyên mà các tiến trình khác đang rất cần. Ở đây HĐH có thể tổ chức phân lớp tài nguyên hệ thống. Theo đó tiến trình phải trả tài nguyên ở mức thấp mới được cấp phát tài nguyên ở cấp cao hơn.
- Đối với điều kiện chờ đợi vòng tròn: Điều kiện này có thể ngăn chặn bằng cách phân lớp tài nguyên của hệ thống. Theo đó, nếu một tiến trình được cấp phát tài nguyên ở lớp L, thì sau đó nó chỉ có thể yêu cầu các tài nguyên ở lớp thấp hơn lớp L.
- Đối với điều kiện phân phối lại tài nguyên: Điều kiện này có thể ngăn chặn bằng cách, khi tiến trình bị rơi vào trạng thái khoá, HĐH có thể thu hồi tài nguyên của

tiến trình bị khoá để cấp phát cho tiến trình khác và cấp lại đầy đủ tài nguyên cho tiến trình khi tiến trình được đưa ra khỏi trạng thái khoá.

2.4.2.2. Nhận biết bế tắc (Deadlock Detection)

Các phương thức ngăn chặn bế tắc ở trên đều tập trung vào việc hạn chế quyền truy xuất đến tài nguyên và áp đặt các ràng buộc lên các tiến trình. Điều này có thể ảnh hưởng đến mục tiêu khai thác hiệu quả tài nguyên của HĐH, ngăn chặn độc quyền trên tài nguyên là một ví dụ, HĐH phải cài đặt các cơ chế độc quyền để bảo vệ các tài nguyên chia sẻ. Và như đã phân tích ở trên việc cấp phát tài nguyên một lần cho các tiến trình để ngăn chặn hiện tượng hold and wait cũng tồn tại một vài hạn chế.

Các HĐH có thể giải quyết vấn đề bế tắc theo hướng phát hiện bế tắc để tìm cách thoát khỏi tắc nghẽn. Phát hiện bế tắc không giới hạn truy xuất tài nguyên và không áp đặt các ràng buộc lên tiến trình. Với phương thức phát hiện bế tắc, các yêu cầu cấp phát tài nguyên được đáp ứng ngay nếu có thể. Để phát hiện bế tắc HĐH thường cài đặt một thuật toán để phát hiện hệ thống có tồn tại hiện tượng chờ đợi vòng tròn hay không.

Việc kiểm tra, để xem thử hệ thống có khả năng xảy ra bế tắc hay không có thể được thực hiện liên tục mỗi khi có một yêu cầu tài nguyên, hoặc chỉ thực hiện thỉnh thoảng theo chu kỳ, phụ thuộc vào sự bế tắc xảy ra như thế nào. Việc kiểm tra bế tắc mỗi khi có yêu cầu tài nguyên sẽ nhận biết được khả năng xảy ra bế tắc nhanh hơn, thuật toán được áp dụng đơn giản hơn vì chỉ dự vào sự thay đổi trạng thái của hệ thống. Tuy nhiên, hệ thống phải tốn nhiều thời gian cho mỗi lần kiểm tra bế tắc.

Mỗi khi bế tắc được phát hiện, HĐH thực hiện một vài giải pháp để thoát khỏi bế tắc. Sau đây là một vài giải pháp có thể:

1. Thoát tất cả các tiến trình bị bế tắc. Đây là một giải pháp đơn giản nhất, thường được các HĐH sử dụng nhất.

2. Sao lưu lại mỗi tiến trình bị bế tắc tại một vài điểm kiểm tra được định nghĩa trước, sau đó khởi động lại tất cả các tiến trình. Giải pháp này yêu cầu HĐH phải lưu lại các thông tin cần thiết tại điểm dừng của tiến trình, đặc biệt là con trỏ lệnh và các tài nguyên tiến trình đang sử dụng, để có thể khởi động lại tiến trình được. Giải pháp này có nguy cơ xuất hiện bế tắc trở lại là rất cao, vì khi tất cả các tiến trình đều được reset trở lại thì việc tranh chấp tài nguyên là khó tránh khỏi. Ngoài ra HĐH phải chi phí rất cao cho việc tạm dừng và tái kích hoạt tiến trình.

3. Chỉ kết thúc một tiến trình trong tập tiến trình bị bế tắc, thu hồi tài nguyên của tiến trình này, để cấp phát cho một tiến trình nào đó trong tập tiến trình bế tắc để giúp tiến trình này ra khỏi bế tắc, rồi gọi lại thuật toán kiểm tra bế tắc để xem hệ thống đã ra khỏi bế tắc hay chưa, nếu rồi thì dừng, nếu chưa thì tiếp tục giải phóng thêm tiến trình khác. Và lần lượt như thế cho đến khi tất cả các tiến trình trong tập tiến trình bế tắc đều ra khỏi tình trạng bế tắc. Trong giải pháp này vấn đề đặt ra đối với HĐH là nên chọn tiến trình nào để giải phóng đầu tiên và dựa vào tiêu chuẩn nào để chọn lựa sao cho chi phí để giải phóng bế tắc là thấp nhất.

4. Tập trung toàn bộ quyền ưu tiên sử dụng tài nguyên cho một tiến trình, để tiến trình này ra khỏi bế tắc, và rồi kiểm tra xem hệ thống đã ra khỏi bế tắc hay chưa, nếu rồi

thì dừng lại, nếu chưa thì tiếp tục. Lần lượt như thế cho đến khi hệ thống ra khỏi bế tắc. Trong giải pháp này HĐH phải tính đến chuyện tái kích hoạt lại tiến trình sau khi hệ thống ra khỏi bế tắc.

Đối với các giải pháp 3 và 4, tiêu chuẩn để giải phóng tiến trình hay ưu tiên tài nguyên là: Thời gian xử lý ít nhất; thời gian cần processor còn lại ít nhất; tài nguyên cần cấp phát là ít nhất; quyền ưu tiên là thấp nhất.

2.4.2.3. Dự báo và phòng tránh

Dự báo và phòng tránh là mỗi lần cấp phát tài nguyên cho tiến trình, HĐH kiểm tra xem việc cấp phát tài nguyên có khả năng dẫn đến bế tắc hay không, nếu có khả năng dẫn đến bế tắc thì dừng việc cấp phát tài nguyên lại.

Trạng thái an toàn: trạng thái A là an toàn nếu hệ thống có thể thỏa mãn các nhu cầu tài nguyên (cho đến tối đa) của mỗi tiến trình theo một thứ tự nào đó mà vẫn ngăn chặn được tắc nghẽn. Một chuỗi cấp phát an toàn: một thứ tự của các tiến trình <P_1, P_2,...,Pn> là an toàn đối với tình trạng cấp phát hiện hành nếu với mỗi tiến trình P_i nhu cầu tài nguyên của P_i có thể được thỏa mãn với các tài nguyên còn tự do của hệ thống, cộng với các tài nguyên đang bị chiếm giữ bởi các tiến trình P_j khác, với j<i.

Một trạng thái an toàn không thể là trạng thái bế tắc. Ngược lại một trạng thái không an toàn có thể dẫn đến tình trạng bế tắc. Chiến lược cấp phát: chỉ thỏa mãn yêu cầu tài nguyên của tiến trình khi trạng thái kết quả là an toàn!

Như vậy, trong giải pháp này, HĐH không thực hiện bất kỳ biện pháp phòng ngừa nào, chỉ thực hiện kiểm tra trước cấp phát tài nguyên cho tiến trình.

Phương pháp này tốn ít chi phí và hệ thống không có khả năng xày ra bế tắc. Tuy nhiên phải liên tục kiểm tra trước khi cấp phát tài nguyên.

Áp dụng khi hệ thống ít khi xảy ra bế tắc nhưng nếu xảy ra bế tắc sẽ có hậu quả lớn.

Ví dụ: Giả sử tình trạng hiện hành của hệ thống được mô tả như sau:

	Max			Allocation			Available		
	R1	R2	R3	R1	R2	R3	R1	R2	R3
P1	3	2	2	1	0	0	4	1	2
P2	6	1	3	2	1	1			
P3	3	1	4	2	1	1			
P4	4	2	2	0	0	2			

Need = Max - Allocation

	Need			Allocation			Available		
	R1	R2	R3	R1	R2	R3	R1	R2	R3
P1	2	2	2	1	0	0			
P2	4	0	2	2	1	1	4	1	2
P3	1	0	3	2	1	1			
P4	4	2	0	0	0	2			

Giả sử tiến trình P2 yêu cầu 4 cho R1, 1 cho R3. Hãy cho biết yêu cầu này có thể đáp ứng mà bảo đảm không xảy ra tình trạng deadlock hay không?

Nhận thấy Available[1] =4, Available[3] =2 đủ để thõa mãn yêu cầu của P2, ta có

	Need			Allocation			Available		
	R1	R2	R3	R1	R2	R3	R1	R2	R3
P1	2	2	2	1	0	0			
P2	0	0	0	6	1	3	0	1	0
P3	1	0	3	2	1	1			
P4	4	2	0	0	0	2			

Tiến trình P2 hoàn thành, các tài nguyên 6R1, 1R2 và 3R3 được thu hồi. Sau đó tiếp tục cấp phát cho tiến trình P1.

	Need			Allocation			Available		
	R1	R2	R3	R1	R2	R3	R1	R2	R3
P1	0	0	0	3	2	2	4	0	1
P2	0	0	0	0	0	0			
P3	1	0	3	2	1	1			
P4	4	2	0	0	0	2			

Tương tự, P2 hoàn thành và cấp phát cho tiến trình P3.

	Need			Allocation			Available		
	R1	R2	R3	R1	R2	R3	R1	R2	R3
P1	0	0	0	0	0	0	6	2	0
P2	0	0	0	0	0	0			
P3	0	0	0	3	1	4			
P4	4	2	0	0	0	2			

Tương tự, P3 hoàn thành và cấp phát cho tiến trình P4.

	Need			Allocation			Available		
	R1	R2	R3	R1	R2	R3	R1	R2	R3
P1	0	0	0	0	0	0	5	1	4
P2	0	0	0	0	0	0			
P3	0	0	0	0	0	0			
P4	0	0	0	4	2	2			

P4 hoàn thành và hoàn trả lại tài nguyên.

	Need			Allocation			Available		
	R1	R2	R3	R1	R2	R3	R1	R2	R3
P1	0	0	0	0	0	0	9	3	6
P2	0	0	0	0	0	0			
P3	0	0	0	0	0	0			
P4	0	0	0	0	0	0			

Trạng thái kết quả là an toàn, có thể cấp phát.

Chương 3
QUẢN LÝ BỘ NHỚ

Mục tiêu

- Quản lý bộ nhớ (chính) là một trong những nhiệm vụ quan trọng và phức tạp nhất của HĐH, cho phép cấp phát và chia sẻ cho nhiều tiến trình đang ở trong trạng thái active; đồng thời còn phải đảm bảo được một số yêu cầu khác.
- Đưa một chương trình từ bộ nhớ ngoài vào bộ nhớ chính để chương trình được thực hiện, HĐH phải thực hiện các phép ánh xạ, chuẩn bị định vị và quản lý cấp phát. Trong đó, chuẩn bị định vị có vai trò quan trọng nhất vì thực chất đó là cách chia bộ nhớ chính để cấp phát cho các chương trình, vừa đáp ứng cho chương trình, vừa đảm bảo hiệu quả sử dụng bộ nhớ trong là cao nhất.

Yêu cầu

- Nắm được bản chất của việc quản lý bộ nhớ chính.
- Với mỗi chiến lược quản lý bộ nhớ chính, nắm được các nguyên tắc của chiến lược, cách chuyển đổi địa chỉ và những ưu nhược điểm nổi bật đối với mỗi chiến lược đó.

3.1. KHÁI QUÁT VỀ QUẢN LÝ BỘ NHỚ

3.1.1. Quá trình chuyển đổi chương trình từ BN ngoài vào BN chính

Chương trình từ bộ nhớ ngoài được chuyển vào bộ nhớ chính để được thực hiện cần phải qua các bước thực hiện sau:

- Ánh xạ (hay chuyển đổi) là việc chuyển đổi tên của chương trình, các biến, địa chỉ các ô nhớ, các câu lệnh từ chương trình tại BN ngoài thành tên chương trình, tên biến, địa chỉ các ô nhớ, các lệnh được thực hiện tại BN chính… Việc thực hiện thông qua các hàm chuyển đổi. Các tên chương trình, tên biến, địa chỉ… của chương trình tại BN ngoài có thể giống, khác nhau; nhưng tên, địa chỉ… tại BN chính là duy nhất (phân biệt).
- Chuẩn bị định vị: là chuẩn bị vùng bộ nhớ tại BN chính để sẵn sàng cấp phát cho chương trình để chương trình được thực hiện. Chuẩn bị định vị, thực chất là xác định cách chia BN chính thành các vùng nhớ để cấp phát cho nhiều chương trình. Mỗi cách chuẩn bị định vị liên quan đến các chiến lược khác nhau cấp phát vùng bộ nhớ cho chương trình, hay là các chiến lược quản lý BN khác nhau.
- Thời điểm định vị: là thời điểm thực hiện cấp phát vùng BN chính cho chương trình theo các chiến lược cấp phát vùng BN đã xác định, tức là chương trình từ BN ngoài được nạp vào BN chính để được thực hiện. Xác định thời điểm định vị là nhiệm vụ quản lý cấp phát của HĐH.

3.1.2. Cấu trúc chương trình

Cấu trúc chương trình là việc biên tập, sắp xếp lại chương trình để chương trình được thực hiện tại BXL. Cách biên tập, sắp xếp lại chương trình khi cấu trúc chương trình cho biết cách thức chương trình được thực hiện tại BXL như thế nào.

Cấu trúc tuần tự.

Cấu trúc động.

Cấu trúc Overlay

Cấu trúc phân đoạn

Cấu trúc phân trang.

Các cấu trúc chương trình cho thấy nhu cầu về vùng BN cần cấp phát và cách quản lý cấp phát như thế nào. Cấu trúc chương trình có liên quan mật thiết đến các chiến lược quản lý BN.

3.1.3. Không gian địa chỉ và không gian vật lý

Việc tìm kiếm, chuyển đổi các đơn vị dữ liệu trong quá trình tính toán được thực hiện thông qua địa chỉ của đơn vị dữ liệu đó trong các BN. Để quản lý việc cấp phát chúng, một cách thuận tiện nhất là quản lý các địa chỉ đơn vị nhớ chứa dữ liệu đó.

Trong quản lý BN, người ta thường dùng 2 khái niệm: địa chỉ luận lý và địa chỉ vật lý.

Địa chỉ luận lý (logical address) – địa chỉ ảo là địa chỉ được tạo ra bởi BXL. Địa chỉ này được sử dụng khi thực hiện biên dịch, cấu trúc chương trình và chuyển đổi các đơn vị dữ liệu. Các địa chỉ luận lý được tạo ra trong nhiều chương trình và tại BN logic.

Địa chỉ vật lý (phisycal address) là địa chỉ mà chỉ có đơn vị quản lý BN nhìn thấy và thao tác trên nó. Địa chỉ vật lý gắn liền với BN chính cụ thể trong máy tính và mỗi đơn vị nhớ trong BN chính chỉ có một địa chỉ vật lý duy nhất tương ứng với nó (còn gọi là địa chỉ tuyệt đối). Địa chỉ vật lý được nạp vào thanh ghi địa chỉ.

Không gian địa chỉ là tập hợp tất cả các địa chỉ luận lý được phát sinh bởi một chương trình.

Không gian vật lý là tập hợp các địa chỉ vật lý tương ứng các địa chỉ luận lý của chương trình. Không gian vật lý thường gắn liền với BN vật lý – BN chính.

Các địa chỉ nguồn trong chương trình là địa chỉ tượng trưng. Khi thực hiện chương trình, phải có nhiều giai đoạn chuyển đổi địa chỉ để trở thành các địa chỉ tuyệt đối trong BN chính. Việc kết buộc các chỉ thị và dữ liệu với các địa chỉ BN vào một trong những giai đoạn sau:

- Thời điểm biên dịch: phát sinh địa chỉ vật lý nếu biết vị trí tiến trình thường trú trong BN. Sau thời điểm này nếu có sự thay đổi so với vị trí tiến trình thường trú ban đầu trong BN thì cần phải biên dịch lại chương trình.
- Thời điểm nạp (tải): nếu tại thời điểm biên dịch chưa thể biết vị trí tiến trình thường trú trong BN, trình biên dịch phải phát sinh mã địa chỉ tương đối (translatable) và tại thời điểm nạp địa chỉ này được chuyển thành địa chỉ tuyệt đối. Khi có sự thay đổi vị trí lưu trữ, chỉ cần nạp lai chương trình để tính toán lại địa chỉ tuyệt đối mà không cần biên dịch lại.
- Thời điểm xử lý: nếu có nhu cầu di chuyển tiến trình từ vùng nhớ này sang vùng nhớ khác trong quá trình tiến trình xử lý, thì việc kết buộc địa chỉ được thực hiện tại thời điểm xử lý và việc chuyển đổi này do phần cứng thực hiện.

Cơ chế phần cứng được sử dụng để chuyển đổi địa chỉ loogic thành địa chỉ vật lý tại thời điểm xử lý là MMU – memory management unit.

3.1.4. Khái niệm về quản lý bộ nhớ

Quản lý BN là việc chia BN chính thành các vùng BN và quản lý cấp phát để đáp ứng cho nhiều chương trình được thực hiện đồng thời.

Quản lý BN liên quan đến cách chia BN chính thành các vùng nhớ và xác định kích thước các vùng nhớ khi chia. Cách chia BN chính và cách xác định kích thước vùng nhớ khi chia khác nhau sẽ hình thành các kỹ thuật cấp phát (quản lý) BN khác nhau.

Quản lý BN nhằm 2 mục đích cơ bản:

- Đáp ứng nhu cầu BN cho các chương trình khi thực hiện
- Đảm bảo sử dụng BN chính một cách hiệu quả nhất.

3.1.5. Nhiệm vụ của quản lý bộ nhớ

3.1.5.1. Sự tái định vị (Relocation)

Trong các hệ thống đa chương, không gian bộ nhớ chính thường được chia sẻ cho nhiều tiến trình khác nhau và yêu cầu bộ nhớ của các tiến trình luôn lớn hơn không gian bộ nhớ vật lý mà hệ thống có được. Do đó, một chương trình đang hoạt động trên bộ nhớ cũng có thể bị đưa ra đĩa (swap-out) và nó sẽ được đưa vào lại (swap-in) bộ nhớ tại một thời điểm thích hợp nào đó sau này.

Để đảm bảo khi đưa một chương trình vào lại bộ nhớ được định vị nó vào đúng vị trí mà nó đã được nạp trước đó, HĐH phải có các cơ chế để ghi lại tất cả các thông tin liên quan đến một chương trình bị swap-out, các thông tin này là cơ sở để hệ điều hành swap-in chương trình vào lại bộ nhớ chính và cho nó tiếp tục hoạt động. Sau khi swap-out một chương trình, HĐH phải tổ chức lại bộ nhớ để chuẩn bị nạp tiến trình vừa có yêu cầu.

Ngoài ra trong nhiệm vụ này hệ điều hành phải có khả năng chuyển đổi các địa chỉ bộ nhớ được ghi trong code của chương trình thành các địa chỉ vật lý thực tế trên bộ nhớ chính khi chương trình thực hiện các thao tác truy xuất trên bộ nhớ.

Trong một số trường hợp khác các chương trình bị swap-out có thể được swap-in vào lại bộ nhớ tại vị trí khác với vị trí mà nó được nạp trước đó.

3.1.5.2. Bảo vệ bộ nhớ (Protection)

Mỗi tiến trình phải được bảo vệ để chống lại sự truy xuất bất hợp lệ vô tình hay có chủ ý của các tiến trình khác. Vì thế các tiến trình trong các chương trình khác không thể tham chiếu đến các vùng nhớ đã dành cho một tiến trình khác để thực hiện các thao tác đọc/ghi mà không được phép (permission), mà nó chỉ có thể truy xuất đến không gian địa chỉ bộ nhớ mà hệ điều hành đã cấp cho tiến trình đó.

Để thực hiện điều này hệ thống quản lý bộ nhớ phải biết được không gian địa chỉ của các tiến trình khác trên bộ nhớ và phải kiểm tra tất cả các yêu cầu truy xuất bộ nhớ của mỗi tiến trình khi tiến trình đưa ra địa chỉ truy xuất. Hệ điều hành có nhiều chiến lược khác nhau để thực hiện điều này.

Điều quan trọng nhất mà hệ thống quản lý bộ nhớ phải thực hiện là không cho phép các tiến trình của người sử dụng truy cập đến bất kỳ một vị trí nào của chính HĐH, ngoại trừ vùng dữ liệu và các rountine mà HĐH cung cấp cho chương trình người sử dụng.

3.1.5.3. Chia sẻ bộ nhớ (Sharing)

Chia sẻ BN để cho phép nhiều tiến trình có thể truy cập đến cùng một địa chỉ trên bộ nhớ chính khi có nhiều tiến trình cùng thực hiện một chương trình (cho phép mỗi tiến trình cùng truy cập đến một bản copy của chương trình) hoặc khi các tiến trình đồng thời (co-operating) trên một vài tác vụ có thể cần để chia sẻ truy cập đến cùng một cấu trúc dữ liệu. Hệ thống quản lý bộ nhớ phải điều khiển việc truy cập đến không gian bộ nhớ được chia sẻ mà không vi phạm đến các yêu cầu bảo vệ bộ nhớ.

Trong môi trường HĐH đa nhiệm hệ điều hành phải chia sẻ không gian nhớ cho các tiến trình để HĐH có thể nạp được nhiều tiến trình vào bộ nhớ để các tiến trình này có thể hoạt động đồng thời với nhau.

3.1.5.4. Tổ chức bộ nhớ logic (Logical organization)

Bộ nhớ chính của hệ thống máy tính được tổ chức như là một dòng hoặc một mảng, không gian địa chỉ bao gồm một dãy có thứ tự các byte hoặc các word. Bộ nhớ phụ cũng được tổ chức tương tự. Mặc dù việc tổ chức này có sự kết hợp chặt chẽ với phần cứng thực tế của máy nhưng nó không phù hợp với các chương trình. Đa số các chương trình đều được chia thành các modun, một vài trong số đó là không thể thay đổi (read only, execute only) và một vài trong số đó chứa dữ liệu là có thể thay đổi. Nếu HĐH và phần cứng máy tính có thể giao dịch một cách hiệu quả với các chương trình của người sử dụng và dữ liệu trong các modun thì một số thuận lợi có thể thấy rõ sau đây:

- Các modun có thể được viết và biên dịch độc lập, với tất cả các tham chiếu từ một modun đến modun khác được giải quyết bởi hệ thống tại thời điểm chạy.
- Các mức độ khác nhau của sự bảo vệ, read-only, execute-only, có thể cho ra các modun khác nhau.
- Nó có thể đưa ra các cơ chế để các modun có thể được chia sẻ giữa các tiến trình.

Công cụ đáp ứng cho yêu cầu này là sự phân đoạn (segmentation), đây là một trong những kỹ thuật quản lý bộ nhớ được trình bày trong chương này.

3.1.5.5. Tổ chức bộ nhớ vật lý (Physical organization)

Như chúng ta đã biết bộ nhớ máy tính được tổ chức theo 2 cấp: bộ nhớ chính và bộ nhớ phụ. Bộ nhớ chính cung cấp một tốc độ truy cập dữ liệu cao, nhưng dữ liệu trên nó phải được làm tươi thường xuyên và không thể tồn tại lâu dài trên nó. Bộ nhớ phụ có tốc độ truy xuất chậm hơn so với bộ nhớ chính và không cần làm tươi thường xuyên. Vì thế bộ nhớ phụ có khả năng lưu trữ lớn và cho phép lưu trữ dữ liệu và chương trình trong một khoảng thời gian dài, trong khi đó bộ nhớ chính chỉ để giữ (hold) một khối lượng nhỏ các chương trình và dữ liệu đang được sử dụng tại thời điểm hiện tại.

Trong giản đồ 2 cấp này, việc tổ chức luồng thông tin giữa bộ nhớ chính và bộ nhớ phụ là một nhiệm vụ quan trọng của hệ thống. Sự chịu trách nhiệm cho luồng này có

thể được gán cho từng người lập trình riêng, nhưng điều này là không hợp lý và có thể gây rắc rối, là do hai nguyên nhân:

- Không gian bộ nhớ chính dành cho các chương trình cùng với dữ liệu của nó thường là không đủ, trong trường hợp này, người lập trình phải tiến hành một thao tác được hiểu như là Overlaying (phủ), theo đó chương trình và dữ liệu được tổ chức thành các modun khác nhau có thể được gán trong cùng một vùng của bộ nhớ, trong đó có một chương trình chính chịu trách nhiệm chuyển các modun vào và ra khi cần.
- Trong môi trường đa chương trình, người lập trình không thể biết tại một thời điểm xác định có bao nhiêu không gian nhớ còn trống hoặc khi nào thì không gian nhớ sẽ trống.

Như vậy nhiệm vụ di chuyển thông tin giữa 2 cấp bộ nhớ phải do hệ thống thực hiện. Đây là nhiệm vụ cơ bản mà thành phần quản lý bộ nhớ phải thực hiện.

3.2. KỸ THUẬT CẤP PHÁT BỘ NHỚ

Kỹ thuật cấp phát BN hay còn được gọi là chiến lược quản lý BN là nội dung quan trọng của HĐH. Trong các hệ thống đa nhiệm, các kỹ thuật cấp phát BN ảnh hưởng trực tiếp đến việc thực hiện các tiến trình và việc đồng bộ hóa hoạt động các thành phần trong máy tính, đảm bảo hiệu quả sử dụng hệ thống.

Có thể phân chia kỹ thuật cấp phát BN theo nhiều cách khác nhau: theo cách chia BN (cố định, không cố định); theo kích thước vùng BN cấp phát (phân chương, phân đoạn, phân trang); theo cách cấp phát (liên tục, không liên tục)… Mỗi kỹ thuật cấp phát BN sẽ cho những đặc điểm khác nhau và mức độ đáp ứng chương trình cũng như mức độ hiệu quả sử dụng BN chính là khác nhau.

3.2.1. Kỹ thuật phân vùng cố định (Fixed Partitioning)

3.2.1.1. Hệ thống đơn chương (cấp phát 1 vùng BN liên tục)

Không gian địa chỉ của bộ nhớ chính được chia thành 2 phần cố định, phần nằm ở vùng địa chỉ thấp dùng để chứa chính HĐH, phần còn lại, tạm gọi là phần user program, dùng cho người sử dụng.

Phần user program được cấp cho một chương trình duy nhất. Hệ điều hành chỉ kiểm soát sự truy xuất bộ nhớ của chương trình người sử dụng, không cho nó truy xuất lên vùng nhớ của HĐH. Để thực hiện việc này HĐH sử dụng một thanh ghi giới hạn. Địa chỉ cao nhất của vùng nhớ cấp cho HĐH được nạp vào thanh ghi giới hạn (chặn dưới của địa chỉ vùng nhớ cấp cho người dùng). Khi chương trình người sử dụng cần truy xuất một địa chỉ nào đó thì HĐH sẽ so sánh địa chỉ này với giá trị địa chỉ được ghi trong thanh ghi giới hạn, nếu nhỏ hơn thì từ chối không cho truy xuất, ngược lại thì cho phép truy xuất. Việc so sánh địa chỉ này cần phải có sự hỗ trợ của phần cứng và có thể làm giảm tốc độ truy xuất bộ nhớ của hệ thống nhưng bảo vệ được HĐH tránh việc chương trình của người sử dụng làm hỏng HĐH dẫn đến làm hỏng hệ thống.

3.2.1.2. Hệ thống đa chương

Trong các hệ thống đa chương, phần user program lại được phân ra thành nhiều phân vùng (partition) với các cố định có kích thước ngẫu nhiên tùy ý. Một tiến trình có

thể được nạp vào bất kỳ partition nào nếu kích thước của nó nhỏ hơn hoặc bằng kích thước của partition và partition này còn trống. Khi có một tiến trình cần được nạp vào bộ nhớ nhưng tất cả các partition đều đã chứa các tiến trình khác thì HĐH có thể chuyển một tiến trình nào đó, mà HĐH cho là hợp lệ (kích thước vừa đủ, không đang ở trạng thái ready hoặc running, không có quan hệ với các tiến trình running khác,...), ra ngoài (swap out), để lấy partition trống đó nạp tiến trình vừa có yêu cầu. Đây là nhiệm vụ phức tạp của HĐH, HĐH phải chi phí cao cho công việc này.

Khi phân vùng cố định với kích thước bằng nhau, phát sinh tồn tại sau:

- Khi kích thước của một chương trình là quá lớn so với kích thước của một partition thì người lập trình phải thiết kế chương trình theo cấu trúc overlay, theo đó chỉ những phần cần thiết của chương trình mới được nạp vào bộ nhớ chính khi khởi tạo chương trình, sau đó người lập trình phải nạp tiếp các modun cần thiết khác vào đúng partition của chương trình và sẽ ghi đè lên bất kỳ chương trình hoặc dữ liệu ở trong đó. Cấu trúc chương trình overlay tiết kiệm được bộ nhớ nhưng yêu cầu cao ở người lập trình.
- Khi kích thước của một chương trình nhỏ hơn kích thước của một partition hoặc quá lớn so với kích thước của một partition nhưng không phải là bội số của kích thước một partition thì dễ xảy ra hiện tượng phân mảnh bên trong[1] (internal fragmentation) bộ nhớ, gây lãng phí bộ nhớ.

Khi phân vùng có kích thước không bằng nhau, việc đưa một tiến trình vào partition sẽ phức tạp hơn nhiều so với trường hợp các phân vùng có kích thước bằng nhau. Khi đó có hai cách để lựa chọn khi đưa một tiến trình vào partition:

- Mỗi phân vùng có một hàng đợi tương ứng, theo đó mỗi tiến trình được đưa đến hàng đợi của phân vùng có kích thước nhỏ nhất đủ thỏa mãn nhu cầu chứa nó. Cách tiếp cận này sẽ đơn giản trong việc đưa một tiến trình từ hàng đợi vào phân vùng vì không có sự lựa chọn nào khác ở đây, khi phân vùng mà tiến trình đợi trống nó sẽ được đưa vào phân vùng đó. Tuy nhiên các tiếp cận này kém linh động vì có thể có một phân vùng đang trống, trong khi đó có nhiều tiến trình đang phải phải đợi để được nạp vào các phân vùng khác, điều này gây lãng phí trong việc sử dụng bộ nhớ.
- Hệ thống dùng một hàng đợi chung cho tất cả các phân vùng, theo đó tất cả các tiến trình được đưa vào hàng đợi chung này. Sau đó nếu có một phân vùng trống thì hệ thống sẽ xem xét để đưa một tiến trình có kích thước vừa đủ vào phân vùng trống đó. Cách tiếp cận này linh động hơn so với việc sử dụng nhiều hàng đợi như ở trên, nhưng việc chọn một tiến trình trong hàng đợi để đưa vào phân vùng là một việc làm khá phức tạp của HĐH vì nó phải dựa vào nhiều yếu tố khác nhau như: độ ưu tiên của tiến trình, trạng thái hiện tại của tiến trình, các mối quan hệ của tiến trình,...

Cấp phát phân vùng cố định còn một số hạn chế sau đây:

- Số lượng các tiến trình có thể hoạt động trong hệ thống tại một thời điểm phụ thuộc vào số lượng các phân vùng cố định trên bộ nhớ.

[1] Phân mảnh trong: hiện tượng xuất hiện những vùng nhớ còn trống khi cấp phát khối nhớ cho các tiến trình. Chi phí quản lý các vùng nhớ trống trong phân mảnh trong lớn hơn rất nhiều so với chính giá trị của các khỏng trống đó.

- Nếu kích thước của tiến trình nhỏ hơn kích thước của một phân vùng thì có thể dẫn đến hiện tượng phân mảnh nội vi gây lãng phí trong việc sử dụng bộ nhớ.

Sự phân vùng cố định ít được sử dụng trong các hệ điều hành hiện nay.

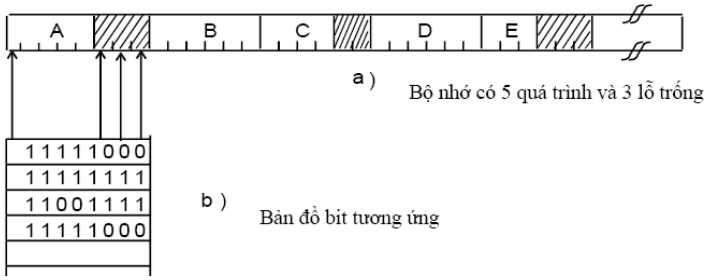
3.2.2. Kỹ thuật phân vùng động (Dynamic Partitioning)

Phần user program trên bộ nhớ không được phân chia trước mà nó chỉ được ấn định sau khi đã có một tiến trình được nạp vào bộ nhớ chính. Khi có một tiến trình được nạp vào bộ nhớ nó được HĐH cấp cho nó vùng nhớ vừa đủ để chứa tiến trình, phần còn lại để sẵn sàng cấp cho tiến trình khác sau này. Khi một tiến trình kết thúc nó được đưa ra ngoài và phần không gian bộ nhớ mà tiến trình này trả lại cho HĐH sẽ được HĐH cấp cho tiến trình khác, cả khi tiến trình này có kích thước nhỏ hơn kích thước của không gian nhớ trống đó.

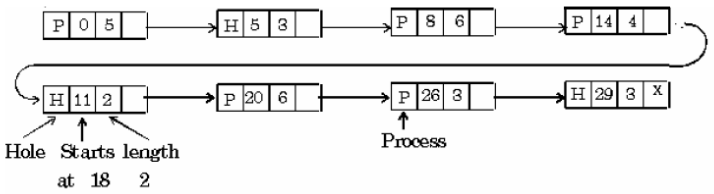
Khi phân vùng động, dần dần trong bộ nhớ hình thành nhiều không gian nhớ có kích thước nhỏ không đủ chứa các tiến trình nằm rải rác trên bộ nhớ chính, hiện tượng này được gọi là hiện tượng phân mảnh bên ngoài (external fragmentation). Để chống lại sự lãng phí bộ nhớ do phân mảnh, HĐH phải thực hiện việc sắp xếp lại bộ nhớ, để các không gian nhớ nhỏ rời rác nằm liền kề lại với nhau tạo thành một khối nhớ có kích thước đủ lớn để chứa được một tiến trình nào đó. Việc làm này làm chậm tốc độ của hệ thống, HĐH phải chi phí cao cho việc này, đặc biệt là việc tái định vị các tiến trình khi một tiến trình bị đưa ra khỏi bộ nhớ và được nạp vào lại bộ nhớ để tiếp tục hoạt động.

Trong kỹ thuật phân vùng động, để quản lý các khối nhớ đã cấp phát hay còn trống trên bộ nhớ, HĐH sử dụng 2 cơ chế: Bản đồ bít và Danh sách liên kết. Trong cả 2 cơ chế này HĐH đều chia không gian nhớ thành các đơn vị cấp phát có kích thước bằng nhau.

- Quản lý bằng bản đồ bít: mỗi đơn vị cấp phát được phản ánh bằng một bít trong bản đồ bít. Bít 0 nếu đơn vị cấp phát BN còn trống, ngược lại bit 1 nếu đơn vị cấp phát đã được cấp phát cho một tiến trình. Khi cần nạp một tiến trình có kích thước k đơn vị, cần phải tìm trong bảng một dãy có k bit nhận giá trị 0.



- Quản lý bằng danh sách liên kết: Tổ chức một danh sách các phân đoạn đã cấp phát và phân đoạn tự do. Mỗi phân đoạn trong danh sách được thể hiện bằng 3 trường chính: trường thứ nhất cho biết phân đoạn đã cấp phát (P: process) hay đang còn trống (H: Hole), trường thứ hai cho biết thứ tự của đơn vị cấp phát đầu tiên trong phân đoạn, trường thứ ba cho biết phân đoạn gồm bao nhiêu đơn vị cấp phát.



Như vậy khi cần nạp một tiến trình vào bộ nhớ thì HĐH phải dựa vào bản đồ bit hoặc danh sách liên kết để tìm ra một phân đoạn có kích thước đủ để nạp tiến trình. Sau khi thực hiện một thao tác cấp phát hoặc sau khi đưa một tiến trình ra khỏi bộ nhớ thì HĐH phải cập nhật lại bản đồ bít hoặc danh sách liên kết, điều này có thể làm giảm tốc độ thực hiện của hệ thống.

Chọn kích thước của một đơn vị cấp phát là một vấn đề quan trọng trong thiết kế, nếu kích thước đơn vị cấp phát nhỏ thì bản đồ bít sẽ lớn, hệ thống phải tốn bộ nhớ để chứa nó. Nếu kích thước của một đơn vị cấp phát lớn thì bản đồ bít sẽ nhỏ, nhưng sự lãng phí bộ nhớ ở đơn vị cấp phát cuối cùng của một tiến trình sẽ lớn khi kích thước của tiến trình không phải là bội số của một đơn vị cấp phát. Điều vừa trình bày cũng đúng trong trường hợp danh sách liên kết.

Danh sách liên kết có thể được sắp xếp theo thứ tự tăng dần hoặc giảm dần của kích thước hoặc địa chỉ, điều này giúp cho việc tìm khối nhớ trống có kích thước vừa đủ để nạp các tiến trình theo các thuật toán dưới đây sẽ đạt tốc độ nhanh hơn và hiệu quả cao hơn. Một số hệ điều hành tổ chức 2 danh sách liên kết riêng để theo dõi các đơn vị cấp phát trên bộ nhớ, một danh sách để theo dõi các block đã cấp phát và một danh dách để theo dõi các block còn trống. Cách này giúp việc tìm các khối nhớ trống nhanh hơn, chỉ tìm trên danh sách các khối nhớ trống, nhưng tốn thời gian nhiều hơn cho việc cập nhật danh sách sau mỗi thao tác cấp phát, vì phải thực hiện trên cả hai danh sách.

Khi có một tiến trình cần được nạp vào bộ nhớ mà trong bộ nhớ có nhiều hơn một khối nhớ trống (Free Block) có kích thước lớn hơn kích thước của tiến trình đó, thì HĐH phải quyết định chọn một khối nhớ trống phù hợp nào để nạp tiến trình sao cho việc lựa chọn này dẫn đến việc sử dụng bộ nhớ chính là hiệu quả nhất.

- **First-fit:** chọn khối nhớ trống đầu tiên có kích thước đủ lớn để nạp tiến trình.
- **Best-fit:** chọn khối nhớ trống có kích thước nhỏ nhất nhưng đáp ứng kích thước của tiến trình cần được nạp vào bộ nhớ.
- **Worst-fit:** chọn khối nhớ trống có kích thước lớn nhất để cấp phát.

Các HĐH không cài đặt cố định trước một thuật toán nào, tuỳ vào trường hợp cụ thể mà nó chọn cấp phát theo một thuật toán nào đó, sao cho chi phí về việc cấp phát là thấp nhất và hạn chế được sự phân mảnh bộ nhớ sau này. Việc chọn thuật toán này thường phụ thuộc vào thứ tự swap và kích thước của tiến trình.

Thuật toán First-fit đơn giản, dễ cài đặt, hiệu quả cao nhất về tốc độ cấp phát.

Thuật toán Best-fit, là một thuật toán có hiệu suất thấp nhất vì HĐH phải duyệt qua tất các các khối nhớ trống để tìm ra một khối nhớ có kích thước vừa đủ để chứa tiến

trình vừa yêu cầu, điều này làm giảm tốc độ cấp phát của hệ điều hành. Mặt khác với việc chọn kích thước vừa đủ có thể dẫn đến sự phân mảnh lớn trên bộ nhớ, tức là có quá nhiều khối nhớ có kích thước quá nhỏ trên bộ nhớ, nhưng nếu xét về mặt lãng phí bộ nhớ tại thời điểm cấp phát thì thuật toán này làm lãng phí ít nhất.

Thuật toán Worst-fit cho hiệu quả không bằng First-fit, nhưng nó thường xuyên sử dụng được các khối nhớ trống lớn nhất, nên có thể hạn chế được sự phân mảnh.

Tóm lại, khó có thể đánh giá về hiệu quả sử dụng của các thuật toán này, vì hiệu quả của nó được xét trong "tương lai" và trên nhiều khía cạnh khác nhau chứ không phải chỉ xét tại thời điểm cấp phát. Và hơn nữa trong bản thân các thuật toán này đã có các mâu thuẫn với nhau về hiệu quả sử dụng của nó.

Trong hệ thống đa chương, một tiến trình đang ở trên bộ nhớ có thể bị đưa ra ngoài (swap-out) để dành chỗ nạp một tiến trình mới có yêu cầu, và tiến trình này sẽ được nạp vào lại (swap-in) bộ nhớ tại một thời điểm thích hợp sau này.

- Cho phép nâng cao mức độ đa chương vì tăng được số lượng tiến trình đồng thời.
- Khi tiến trình được mang lại vào BN chính, nếu sự kết buộc địa chỉ được thực hiện vào thời điểm nạp, tiến trình cần được nạp vào đúng vị trí nó đã chiếm giữ trước đó. Nếu kết buộc địa chỉ vào thời điểm xử lý, thì có thể nạp tiến trình vào vị trí vùng nhớ bất kỳ.
- Cần BN phụ có kích thước đủ lớn để lưu trữ các tiến trình bị swap-out và phải cho phép truy xuất trực tiếp đến các tiến trình này.
- Trong các hệ swapping, thời gian chuyển đổi giữa các tác vụ cần được quan tâm. Mỗi tiến trình cần được phân chia một khoảng thời gian sử dụng BXL đủ lớn để không thấy quá rõ sự chậm trễ do các thao tác swap gây ra. Nếu không, hệ thống sẽ dùng phần lớn thời gian để dời chuyển các tiến trình ra BN chính và sử dụng BXL sẽ không hiệu quả.

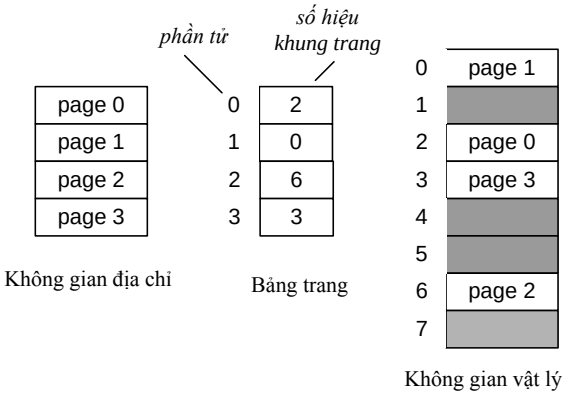
Chú ý: Một nhược điểm lớn của các kỹ thuật ở trên là dẫn đến hiện tượng phân mảnh bộ nhớ gây lãng phí bộ nhớ nên hiệu quả sử dụng bộ nhớ kém. Để khắc phục HĐH sử dụng các kỹ thuật phân trang hoặc phân đoạn bộ nhớ.

3.2.3. Kỹ thuật phân trang đơn (Simple Paging)

Trong kỹ thuật này không gian địa chỉ bộ nhớ vật lý được chia thành các phần có kích thước cố định và bằng nhau, được đánh số địa chỉ bắt đầu từ 0 và được gọi là các khung trang (page frame). Không gian địa chỉ của các tiến trình cũng được chia thành các phần có kích thước bằng nhau và bằng kích thước của một khung trang, được gọi là các trang (page) của tiến trình.

Khi một tiến trình được nạp vào bộ nhớ thì các trang của tiến trình được nạp vào các khung trang còn trống bất kỳ, có thể không liên tiếp nhau của bộ nhớ. Khi HĐH cần nạp một tiến trình có n trang vào bộ nhớ thì nó phải tìm đủ n khung trang trống để nạp tiến trình này. Nếu kích thước của tiến trình không phải là bội số của kích thước một khung trang thì sẽ xảy ra hiện tượng phân mảnh nội vi ở khung trang chứa trang cuối cùng của tiến trình. Ở đây không xảy ra hiện tượng phân mảnh ngoại vi. Trên bộ nhớ có

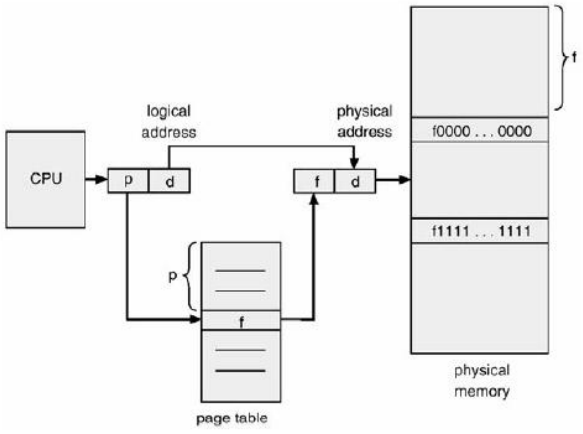
thể tồn tại các trang của nhiều tiến trình khác nhau. Khi một tiến trình bị swap-out thì các khung trang mà tiến trình này chiếm giữ sẽ được giải phóng để HĐH có thể nạp các trang tiến trình khác.



Để theo dõi trạng thái của các khung trang (còn trống hay đã cấp phát) trên bộ nhớ, HĐH sử dụng một danh sách để ghi số hiệu của các khung trang còn trống trên bộ nhớ. Dựa vào danh sách này, HĐH tìm ra các khung trang trống trước khi quyết định nạp một tiến trình vào bộ nhớ. Danh sách này được cập nhật ngay sau khi HĐH nạp một tiến trình vào bộ nhớ, được kết thúc hoặc bị swap out ra bên ngoài.

Hệ điều hành sử dụng các bảng trang (PCT: page control table) để theo dõi vị trí các trang tiến trình trên bộ nhớ, mỗi tiến trình có một bảng trang riêng. Bảng trang bao gồm nhiều phần tử, thường là bằng số lượng trang của một tiến trình mà bảng trang này theo dõi, các phần tử được đánh số bắt đầu từ 0. Phần tử 0 chứa số hiệu của khung trang đang chứa trang 0 của tiến trình, phần tử 1 chứa số hiệu của khung trang đang chứa trang 1 của tiến trình,… Các bảng trang có thể được chứa trong các thanh ghi nếu có kích thước nhỏ, nếu kích thước bảng trang lớn thì nó được chứa trong bộ nhớ chính, khi đó HĐH sẽ dùng một thanh ghi để lưu trữ địa chỉ bắt đầu nơi lưu trữ bảng trang, thanh ghi này được gọi là thanh ghi PTBR: page table base register.

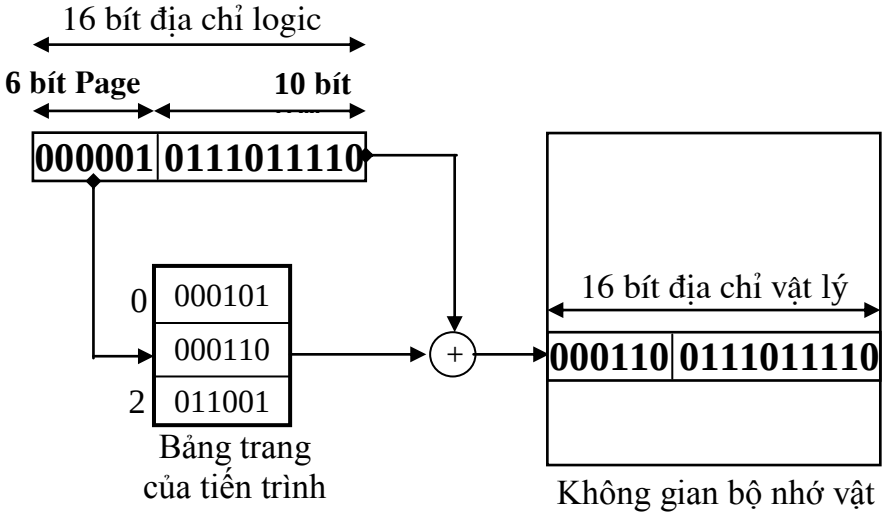
Trong kỹ thuật phân trang này khi cần truy xuất bộ nhớ BXL phải phát ra một địa chỉ logic gồm 2 thành phần: Số hiệu trang (Page): cho biết số hiệu trang tương ứng cần truy xuất. Địa chỉ tương đối trong trang (Offset): giá trị này sẽ được kết hợp với địa chỉ bắt đầu của trang để xác định địa chỉ vật lý của ô nhớ cần truy xuất. Việc chuyển đổi từ địa chỉ logic sang địa chỉ vật lý do processor thực hiện.



Kích thước của mỗi trang hay khung trang do phần cứng quy định, thường biến đổi từ 512 byte đến 8192 byte 16MB). Nếu kích thước của không gian địa chỉ là 2^m và kích thước của trang là 2^n thì m-n bít cao của địa chỉ logic là số hiệu trang (page) và n bít còn lại là địa chỉ tương đối trong trang (offset).

Việc chuyển từ địa chỉ logic sang địa chỉ vật lý được thực hiện theo các bước sau:

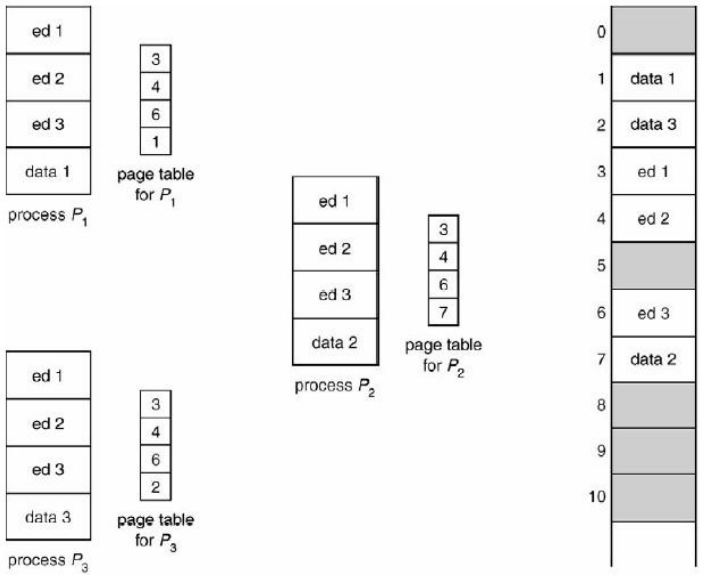
- Trích ra m-n bít trái nhất (thấp nhất) của địa chỉ logic để xác định số hiệu trang cần truy xuất.
- Sử dụng số hiệu trang ở trên để chỉ đến phần tử tương ứng trong bảng trang của tiến trình, để xác định khung trang tương ứng, ví dụ là k.
- Địa chỉ vật lý bắt đầu của khung trang là k x 2^n , và địa chỉ vật lý của byte cần truy xuất là số hiệu trang cộng với giá trị offset. Địa chỉ vật lý không cần tính toán, nó dễ dàng có được bằng cách nối số hiệu khung trang với giá trị offset.



Hình Sơ đồ chuyển đổi địa chỉ logic (page) – vật lý

Trong sơ đồ ví dụ ở trên, chúng ta có địa chỉ logic là: 0000010111011110, với số hiệu trang là 1, offset là 478, giả định rằng trang này thường trú trong bộ nhớ chính tại khung trang 6 = 000110. Thì địa chỉ vật lý là khung trang số 6 và offset là 478 = 000110111011110.

Chia sẻ BN trong kỹ thuật phân trang được sử dụng nhằm sử dụng hiệu quả BN chính và hỗ trợ được cho nhiều người dùng (trình biên dịch, hệ thống cửa sổ, thư viện, CSDL…). BN được chia sẻ được cài đặt cho nhiều địa chỉ ảo (mỗi địa chỉ ảo cho mỗi quá trình chia sẻ BN) mà chúng cùng được ánh xạ đến 1 địa chỉ vật lý.



Nhận xét về kỹ thuật phân trang

- Có thể thấy sự phân trang tương tự như sự phân vùng cố định. Sự khác nhau là với phân trang các phân vùng có kích thước nhỏ hơn, một chương trình có thể chiếm giữ nhiều hơn một phân vùng, và các phân vùng này có thể không liền kề với nhau.
- Kỹ thuật phân trang loại bỏ được phân mảnh ngoại vi, nhưng vẫn có thể xảy ra phân mảnh nội vi khi kích thước của tiến trình không đúng bằng bội số kích thước của một trang, khi đó khung trang cuối cùng sẽ không được sử dụng hết.
- Khi cần truy xuất đến dữ liệu hay chỉ thị trên BN thì hệ thống phải cần một lần truy xuất đến bảng trang, điều này có thể làm giảm tốc độ truy xuất BN. Để khắc phục HĐH sử dụng thêm một bảng trang cache, để lưu trữ các trang BN vừa được truy cập gần đây nhất. Bảng trang cache này sẽ được sử dụng mỗi khi BXL phát ra một địa chỉ cần truy xuất.
- Mỗi HĐH có một cơ chế tổ chức bảng trang riêng, đa số các HĐH đều tạo cho mỗi tiến trình một bảng trang riêng khi nó được nạp vào BN chính. Bảng trang lớn sẽ tốn BN để chứa nó.
- Để bảo vệ các khung trang HĐH đưa thêm một bít bảo vệ vào bảng trang. Theo đó mỗi khi tham khảo vào bảng trang để truy xuất BN hệ thống sẽ kiểm tra các thao tác truy xuất trên khung trang tương ứng có hợp lệ với thuộc tính bảo vệ của nó hay không.
- Sự phân trang không phản ánh được cách mà người sử dụng nhìn nhận về BN. Với người sử dụng, BN là một tập các đối tượng chương trình và dữ liệu như các segment, các thư viện,... và các biến, các vùng nhớ chia sẻ, stack... Vấn đề đặt ra là tìm một cách thức biểu diễn BN sao cho nó gần với cách nhìn nhận của người sử dụng hơn. Kỹ thuật phân đoạn BN có thể thực hiện được mục tiêu này.

3.2.4. Kỹ thuật phân đoạn đơn (Simple Segmentation)

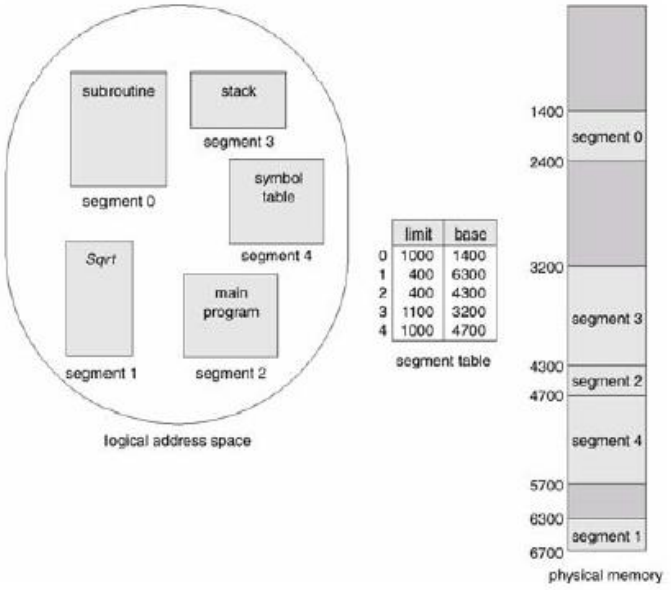
Không gian địa chỉ logic của các tiến trình kể cả các dữ liệu liên quan là tập hợp các đoạn khác nhau và không nhất thiết có kích thước bằng nhau, thông thường mỗi thành phần của một chương trình/tiến trình như: code, data, stack, subprogram,.. là một đoạn.

Không gian địa chỉ bộ nhớ vật lý cũng được chia thành các phần cố định có kích thước không bằng nhau, được đánh số bắt đầu từ 0, được gọi là các phân đoạn (segment). Mỗi phân đoạn bao gồm số hiệu phân đoạn và kích thước của nó.

Khi một tiến trình được nạp vào bộ nhớ thì tất cả các đoạn của nó sẽ được nạp vào các phân đoạn còn trống khác nhau trên bộ nhớ. Các phân đoạn này có thể không liên tiếp nhau.

Để theo dõi các đoạn của các tiến trình khác nhau trên bộ nhớ, HĐH sử dụng các bảng phân đoạn (SCT: Segment control Table) cho mỗi tiến trình. Mỗi phần tử trong bảng phân đoạn gồm tối thiểu 2 trường: địa chỉ cơ sở (base) của phân đoạn và độ dài/giới hạn (length/limit) của phân đoạn. Địa chỉ cơ sở (base) đoạn xác định địa chỉ vật lý của byte đầu tiên trong đoạn. Giới hạn đoạn (limit) xác định kích thước đoạn, trường này còn có tác dụng dùng để kiểm soát sự truy xuất bất hợp lệ của các tiến trình. Các bảng phân

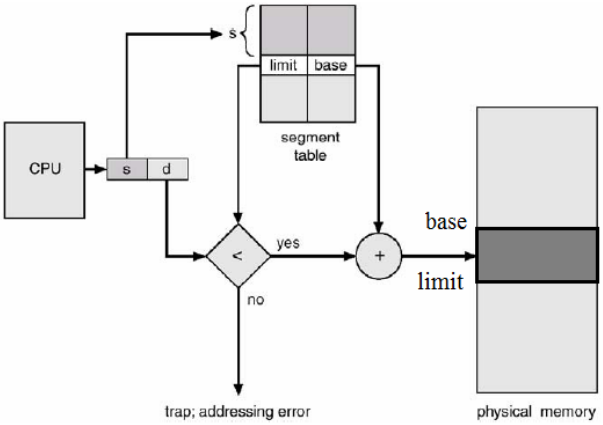
đoạn có thể được chứa trong các thanh ghi nếu có kích thước nhỏ, nếu kích thước bảng phân đoạn lớn thì nó được chứa trong BN chính, khi đó HĐH sẽ dùng một thanh ghi để lưu trữ địa chỉ bắt đầu nơi lưu trữ bảng phân đoạn, thanh ghi này được gọi là thanh ghi STBR: Segment table base register. Ngoài ra vì số lượng các đoạn của một chương trình/tiến trình có thể thay đổi nên hệ điều hành dùng thêm thanh ghi STLR: Segment table length register, để ghi kích thước hiện tại của bảng phân đoạn. Hệ điều hành cũng tổ chức một danh sách riêng để theo dõi các segment còn trống trên bộ nhớ.



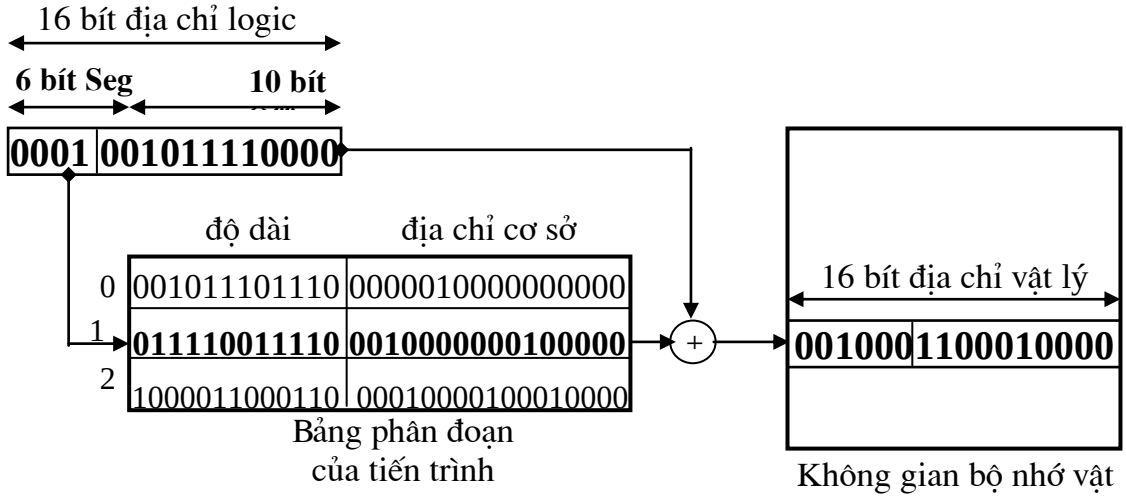
Trong kỹ thuật này địa chỉ logic mà BXL sử dụng gồm 2 thành phần: Số hiệu đoạn (segment): cho biết số hiệu đoạn tương ứng cần truy xuất và địa chỉ tương đối trong đoạn (Offset): giá trị này sẽ được kết hợp với địa chỉ bắt đầu của đoạn để xác định địa chỉ vật lý của ô nhớ cần truy xuất. Việc chuyển đổi từ địa chỉ logic sang địa chỉ vật lý do processor thực hiện.

Nếu có một địa chỉ logic gồm $n + m$ bít, thì n bít bên trái là số hiệu segment, m bít bên phải còn lại là offset. Trong ví dụ minh hoạ sau đây thì $n = 4$ và $m = 12$, như vậy kích thước tối đa của một segment là $2^{12} = 4096$ byte. Sau đây là các bước cần thiết của việc chuyển đổi địa chỉ:

- Trích ra n bít bên trái của địa chỉ logic để xác định số hiệu của phân đoạn cần truy xuất.
- Sử dụng số hiệu phân đoạn ở trên để chỉ đến phần tử trong bảng phân đoạn của tiến trình, để tìm địa chỉ vật lý bắt đầu của phân đoạn.
- So sánh thành phần offset của địa chỉ logic, được trích ra từ m bít bên phải của địa chỉ logic, với thành phần length của phân đoạn. Nếu offset > length thì địa chỉ truy xuất là không hợp lệ.
- Địa chỉ vật lý mong muốn là địa chỉ vật lý bắt đầu của phân đoạn cộng với giá trị offset.

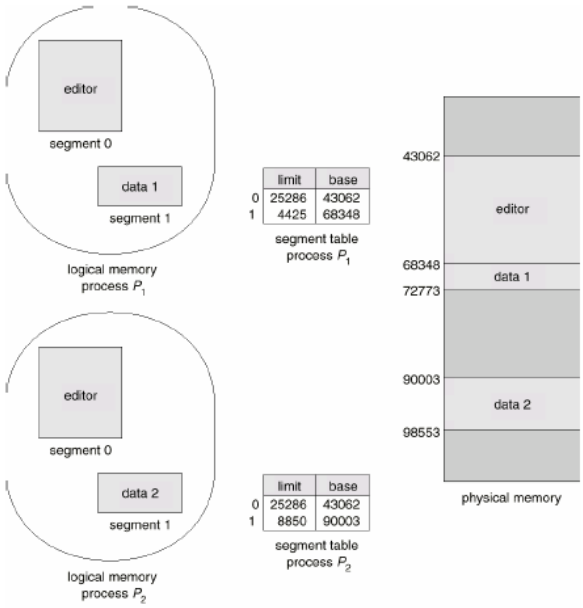


Trong ví dụ sau đây, ta có địa chỉ logic là: 0001001011110000, với số hiệu segment là 1, offset là 752, giả định segment này thường trú trong bộ nhớ chính tại địa chỉ vật lý là 0010000000100000, thì địa chỉ vật lý tương ứng với địa chỉ logic ở trên là: 0010000000100000 + 001011110000 = 0010001100010000.



Hình Sơ đồ chuyển đổi địa chỉ logic (segment) – vật lý

Trong kỹ thuật phân đoạn, có thể chia sẻ các phân đoạn chương trình hoặc chia sẻ các gói chương trình con, dữ liệu.



Nhận xét về kỹ thuật phân đoạn

- Vì các segment có kích thước không bằng nhau nên sự phân đoạn tương tự như sự phân vùng động. Sự khác nhau là với sự phân đoạn một chương trình có thể chiếm giữ hơn một phân vùng, và các phân vùng này có thể không liền kề với nhau. Sự phân vùng loại trừ được sự phân mảnh nội vi, nhưng như sự phân vùng động nó vẫn xuất hiện hiện tượng phân mảnh ngoại vi.

- Sự phân đoạn là tường minh đối với người lập trình, và nó cung cấp một sự thuận lợi để người lập trình tổ chức chương trình và dữ liệu. Người lập trình hoặc trình biên dịch có thể gán các chương trình và dữ liệu đến các đoạn nhớ khác nhau.

- Tương tự như trong kỹ thuật phân vùng động, kỹ thuật này cũng phải giải quyết vấn đề cấp phát động, ở đây hệ điều hành thường dùng thuật toán best-fit hay first-fit.

- Kỹ thuật phân đoạn thể hiện được cấu trúc logic của chương trình, nhưng nó phải cấp phát các khối nhớ có kích thước khác nhau cho các phân đoạn của chương trình trên bộ nhớ vật lý, điều này phức tạp hơn nhiều so với việc cấp phát các khung trang. Để dung hòa vấn đề này các hệ điều hành có thể kết hợp cả phân trang và phân đoạn.

* Kỹ thuật phân đoạn kết hợp phân trang

Không gian địa chỉ là một tập các phân đoạn, mỗi phân đoạn được chia thành các trang. Khi một tiến trình được đưa vào hệ thống, HĐH sẽ cấp phát cho tiến trình các trang cần thiết để chứa đủ các phân đoạn của tiến trình.

Cơ chế MMU cần có một bảng phân đoạn và mỗi phân đoạn có một bảng trang phân biệt.

Địa chỉ logic mỗi đơn vị nhớ có (s, p, d); trong đó s – là số hiệu phân đoạn, để chỉ đến phần tử tương ứng trong bảng phân đoạn; p – số hiệu trang, chỉ đến phần tử tương ứng trong bảng trang của phân đoạn; d – địa chỉ tương đối trong trang, kết hợp với địa chỉ đầu của trang để tạo ra địa chỉ vật lý mà trình quản lý BN sử dụng.

Đặc điểm chung

Các kỹ thuật trên đều có khuynh hướng cấp phát vùng nhớ cho tiến trình toàn bộ các trang yêu cầu trước khi thực sự xử lý.

Do kích thước BN chính bị giới hạn, nên kích thước tiến trình cũng bị giới hạn.

Khó có thể duy trì đồng thời nhiều tiến trình trong BN, và như vậy khó nâng cao mức độ đa chương của hệ thống.

3.3. KỸ THUẬT BỘ NHỚ ẢO (Virtual Memory)

3.3.1. Bộ nhớ ảo

Nếu đặt toàn bộ không gian địa chỉ vào BN vật lý thì kích thước của chương trình bị giới hạn bởi kích thước BN vật lý. Thực tế, trong nhiều trường hợp, chúng ta không cần nạp toàn bộ chương trình vào BN vật lý, vì tại một thời điểm chỉ có một chỉ thị của tiến trình được xử lý.

Để thực hiện được việc nạp từng phần của chương trình vào BN vật lý, tại mỗi thời điểm chỉ lưu trữ trong BN vật lý các chỉ thị và dữ liệu của chương trình cần thiết cho

việc thi hành tại thời điểm đó. Khi cần đến các chỉ thị khác, những chỉ thị mới sẽ được nạp vào BN tại vị trí trước đó bị chiếm giữ bởi các chỉ thị nay không còn cần đến nữa. Khi đó, một chương trình có thể lớn hơn kích thước của vùng nhớ cấp phát cho nó. Kỹ thuật Overlay rất thuận tiện cho HĐH, nhưng đòi hỏi người lập trình viên phải lập trình theo cấu trúc Overlay, mà điều này là rất phức tạp. Bộ nhớ ảo cho phép khắc phục được khó khăn trên.

Bộ nhớ ảo là một kỹ thuật cho phép xử lý một tiến trình không được nạp toàn bộ vào BN chính. Bộ nhớ ảo mô hình hóa BN như một bảng lưu trữ rất lớn và đồng nhất, tách biệt hẳn khái niệm không gian địa chỉ và không gian vật lý. Người sử dụng chỉ nhìn thấy và làm việc trong không gian địa chỉ ảo, việc chuyển đổi sang không gian vật lý do HĐH và sự trợ giúp của cơ chế phần cứng thực hiện.

Việc sử dụng bộ nhớ ảo mang lại các lợi ích sau đây:

- Hệ điều hành có thể nạp được nhiều tiến trình hơn vào bộ nhớ, trên bộ nhớ tồn tại các trang/đoạn của nhiều tiến trình khác nhau.
- Với kỹ thuật bộ nhớ ảo người lập trình không cần quan tâm đến kích thước của chương trình và kích thước của bộ nhớ tại thời điểm nạp chương trình, tất cả mọi việc này đều do hệ điều hành và phần cứng thực hiện.

Để cài đặt được bộ nhớ ảo hệ HĐH cần phải có:

- Một lượng không gian bộ nhớ phụ (đĩa) cần thiết đủ để chứa các trang/đoạn bị swap out, không gian đĩa này được gọi là không gian swap.
- Có cơ chế để theo dõi các trang/đoạn của một tiến trình, của tất cả các tiến trình đang hoạt động trên bộ nhớ chính, là đang ở trên bộ nhớ chính hay ở trên bộ nhớ phụ. Trong trường hợp này HĐH thường đưa thêm một bít trạng thái (bit present) vào các phần tử trong PCT hoặc SCT.
- Dựa vào các tiêu chuẩn cụ thể để chọn một trang nào đó trong số các trang đang ở trên BN chính để swap out trong trường hợp cần thiết. Các HĐH đã đưa ra các thuật toán cụ thể để phục vụ cho mục đích này.

3.3.2. Kỹ thuật bộ nhớ ảo

Hệ điều hành có thể cài đặt bộ nhớ ảo theo 2 kỹ thuật:

- Phân trang theo yêu cầu: phân trang kết hợp với swap.
- Phân đoạn theo yêu cầu: phân đoạn kết hợp với swap.

Cả hai kỹ thuật trên đều phải có sự hỗ trợ của phần cứng máy tính, cụ thể là processor. Đa số các HĐH đều chọn kỹ thuật phân trang theo yêu cầu, vì nó đơn giản, dễ cài đặt và chi phí thấp hơn.

3.3.2.1. Sự phân trang theo yêu cầu (Demand paging)

Trong kỹ thuật phân trang đơn, mỗi tiến trình sở hữu một bảng trang riêng, khi tất cả các trang của tiến trình được nạp vào BN chính thì bảng trang của tiến trình được tạo ra và cũng được nạp vào BN (nếu lớn), mỗi phần tử trong bảng trang chỉ chứa số hiệu của khung trang mà trang tương ứng được nạp vào. Trong kỹ thuật BN ảo cũng vậy, nhưng

một phần tử trong bảng trang sẽ chứa nhiều thông tin phức tạp hơn. Bởi vì trong kỹ thuật BN ảo chỉ những trang cần thiết trong thời điểm hiện tại mới được nạp vào BN chính. Như vậy, một trang chỉ được nạp khi BN chính có yêu cầu. Do đó cần phải có một bít để cho biết một page tương ứng của tiến trình là có hay không trên BN chính và một bít cho biết page có bị thay đổi hay không so với lần nạp gần đây nhất. Cụ thể là nó phải có thêm các bít điều khiển:

Virtual Address

Page Number	Offset

		Các bít điều khiển khác	Frame Number

Hình Một phần tử trong bảng trang

- Bít P (Present): Cho biết trang tương ứng đang ở trên BN chính (1) hay ở trên bộ nhớ phụ (0).

- Bít M (Modify): Cho biết nội dung của trang tương ứng có bị thay đổi hay không so với lần nạp gần đây nhất. Nếu nó không bị thay đổi thì việc phải ghi lại nội dung của một trang khi cần phải đưa một trang ra lại BN ngoài là không cần thiết, điều này giúp tăng tốc độ trong các thao tác thay thế trang trong khung trang.

- Các bít điều khiển khác: Các bít này phục vụ cho các mục đích bảo vệ trang và chia sẻ các khung trang.

Chuyển đổi địa chỉ trong hệ thống phân trang

Chương trình của người sử dụng sử dụng địa chỉ logic hoặc virtual gồm: page number và offset để truy xuất dữ liệu trên BN chính. Bộ phận quản lý BN phải chuyển địa chỉ virtual này thành địa chỉ vật lý tương ứng bao gồm: page number và offset. Để thực hiện việc này bộ phận quản lý BN phải dựa vào bảng trang (PCT).

Cấu trúc bảng trang cho phép phản ánh tình trạng của một trang là đang nằm tại BN chính hay BN phụ. BN phụ lưu trữ các trang không được nạp vào BN chính, thường là các đĩa nhớ. Vùng không gian đĩa dùng để lưu trữ tạm các trang trong kỹ thuật swapping được gọi là không gian swapping.

Vì kích thước của PCT có thể lớn và thay đổi theo kích thước của tiến trình do đó trong kỹ thuật BN ảo HĐH thường chứa PCT trong bộ nhớ chính và dùng một thanh ghi để ghi địa chỉ bắt đầu của bộ nhớ nơi lưu trữ PCT của tiến trình khi tiến trình được nạp vào bộ nhớ chính để chạy.

Đa số các HĐH đều thiết kế một bảng trang riêng cho mỗi tiến trình. Nhưng mỗi tiến trình có thể chiếm giữ một không gian lớn BN ảo, trong trường hợp này bảng trang rất lớn và hệ thống phải tốn không gian BN để chứa nó. Ví dụ, nếu một tiến trình có đến 2^{31} = 2GB bộ nhớ ảo, mỗi trang có kích thước 2^9 = 512 byte, thì tiến trình này phải cần đến 2^{22} phần tử trong bảng trang. Để khắc phục vấn đề này, trong các kỹ thuật BN ảo hệ thống lưu trữ bảng trang trong BN ảo chứ không lưu trữ trong BN thực, và bản thân bảng trang cũng phải được phân trang. Khi tiến trình thực hiện, chỉ có một phần của bảng trang được nạp vào BN chính, đây là phần chứa các phần tử của các trang đang thực hiện tại thời điểm hiện tại.

Kích thước của trang:

Kích thước của một trang do phần cứng quy định, đây là một trong những quyết định quan trọng trong việc thiết kế processor. Nếu kích thước của trang nhỏ thì sự phân mảnh bên trong sẽ nhỏ hơn, việc sử dụng BN chính sẽ được hiệu quả hơn. Nhưng nếu kích thước trang nhỏ thì số lượng trang trên một tiến trình sẽ lớn hơn, bảng trang của tiến trình sẽ lớn, sẽ chiếm nhiều BN hơn, và như thế việc sử dụng BN chính sẽ kém hiệu quả hơn. Các vi xử lý họ Intel 486 và họ Motorola 68040 chọn kích thước của một trang là 4096 byte.

Ngoài ra kích thước của trang còn ảnh hưởng đến tỉ lệ xảy ra lỗi trang. Ví dụ: khi kích thước của trang là rất nhỏ thì sẽ có một lượng lớn các trang của tiến trình trên BNchính, sau một thời gian thì tất cả các trang của BN sẽ chứa các tiến trình được tham chiếu gần đây, vì thế tốc độ xảy ra lỗi trang được giảm xuống.

Lỗi trang (page fault)

Trong mô hình BN ảo khi cần truy xuất đến một page của tiến trình thì trước hết hệ thống phải kiểm tra bít present tại phần tử tương ứng với page cần truy xuất trong PCT, để biết được page cần truy xuất đã được nạp vào BN hay chưa. Trường hợp hệ thống cần truy xuất đến một page của tiến trình mà page đó đã được nạp vào BN chính, được gọi là truy xuất hợp lệ (v: valid). Trường hợp hệ thống cần truy xuất đến một page của tiến trình mà page đó chưa được nạp vào BN chính, được gọi là truy xuất bất hợp lệ (i: invalid). Khi hệ thống truy xuất đến một trang của tiến trình mà trang đó không thuộc phạm vi không gian địa chỉ của tiến trình cũng được gọi là truy xuất bất hợp lệ.

Khi hệ thống truy xuất đến một page được đánh dấu là bất hợp lệ thì sẽ phát sinh một lỗi trang. Như vậy lỗi trang là hiện tượng hệ thống cần truy xuất đến một page của tiến trình mà trang này chưa được nạp vào BN, hay không thuộc không gian địa chỉ của tiến trình. Ở đây ta chỉ xét lỗi trang của trường hợp: Page cần truy xuất chưa được nạp vào BN chính.

Khi nhận được tín hiệu lỗi trang, HĐH phải tạm dừng tiến trình hiện tại để tiến hành việc xử lý lỗi trang. Khi xử lý lỗi trang HĐH có thể gặp một trong hai tình huống sau:

a- Hệ thống còn frame trống: Hệ điều hành sẽ thực hiện các bước sau:

1. Tìm vị trí của page cần truy xuất trên đĩa.
2. Nạp page vừa tìm thấy vào BN chính.
3. Cập nhật lại bảng trang (PCT) tiến trình.
4. Tái kích hoạt tiến trình để tiến trình tiếp tục hoạt động.

b- Hệ thống không còn frame trống:

1. Tìm vị trí của page cần truy xuất trên đĩa.
2. Tìm một page không hoạt động hoặc không thực sự cần thiết tại thời điểm hiện tại để swap out nó ra đĩa, lấy frame trống đó để nạp page mà hệ thống vừa cần truy xuất. Page bị swap out sẽ được HĐH swap in trở lại BN tại một thời điểm thích hợp sau này.
3. Cập nhật PCT của tiến trình có page vừa bị swap out.

4. Nạp trang vừa tìm thấy ở trên (bước 1) vào frame trống ở trên (bước 2).
5. Cập nhật lại bảng trang (PCT) của tiến trình.
6. Tái kích hoạt tiến trình để tiến trình tiếp tục hoạt động.

Xử lý lỗi trang là một trong những nhiệm vụ quan trọng và phức tạp của hệ thống và HĐH. Để xử lý lỗi trang hệ thống phải tạm dừng các thao tác hiện tại, trong trường hợp này hệ thống phải lưu lại các thông tin cần thiết như: con trỏ lệnh, nội dung của các thanh ghi, các không gian địa chỉ bộ nhớ…, các thông tin này là cơ sở để hệ thống tái kích hoạt tiến trình bị tạm dừng trước đó khi nó đã hoàn thành việc xử lý lỗi trang.

Khi xử lý lỗi trang, trong trường hợp hệ thống không còn frame trống HĐH phải chú ý đến các vấn đề sau:

Nên chọn page nào trong số các page trên bộ nhớ chính để swap out: có thể chọn page của tiến trình xảy ra lỗi trang để thay thế (*thay thế cục bộ*), hoặc chọn page của tiến trình khác để thay thế (*thay thế toàn cục*). Nếu chọn page của tiến trình xảy ra lỗi trang thì sẽ đơn giản hơn với HĐH và không ảnh hưởng đến các tiến trình khác, nhưng cách này có thể làm cho tiến trình hiện tại lại tiếp tục xảy ra lỗi trang ngay sau khi HĐH vừa xử lý lỗi trang cho nó, vì page mà HĐH vừa chọn để đưa ra (swap out) lại là page cần truy xuất ở thời điểm tiếp theo. Nếu chọn page của tiến trình khác thì tiến trình hiện tại sẽ ít có nguy cơ xảy ra lỗi trang ngay sau đó hơn, nhưng cách này sẽ phức tạp hơn cho HĐH, vì HĐH phải kiểm soát lỗi trang của nhiều tiến trình khác trong hệ thống, và HĐH khó có thể dự đoán được nguy cơ xảy ra lỗi trang của các tiến trình trong hệ thống. Trong trường hợp này có thể lỗi trang sẽ lan truyền đến nhiều tiến trình khác trong hệ thống, khi đó việc xử lý lỗi trang của HĐH sẽ phức tạp hơn rất nhiều. Đa số các HĐH đều chọn cách thứ nhất vì nó đơn giản và không ảnh hưởng đến các tiến trình khác trong hệ thống.

"Neo" một số page: Trên BN chính tồn tại các page của các tiến trình đặc biệt quan trọng đối với hệ thống, nếu các tiến trình này bị tạm dừng thì sẽ ảnh hưởng rất lớn đến hệ thống và có thể làm cho hệ thống ngừng hoạt động, nên HĐH không được đưa các page này ra đĩa trong bất kỳ trường hợp nào. Để tránh các thuật toán thay trang chọn các page này HĐH tổ chức đánh dấu các page này, bằng cách đưa thêm một bít mới vào các phần tử trong các PCT, bít này được gọi là bít neo. Như vậy các thuật toán thay trang sẽ không xem xét đến các page được đánh dấu neo khi cần phải đưa một trang nào đó ra đĩa.

Phải tránh được trường hợp hệ thống xảy ra hiện tượng "trì trệ hệ thống": Trì trệ hệ thống là hiện tượng mà hệ thống luôn ở trong tình trạng xử lý lỗi trang, tức là đa phần thời gian xử lý của processor đều dành cho việc xử lý lỗi trang của HĐH. Hiện tượng này có thể được mô tả như sau: khi xử lý lỗi trang trong trường hợp trên BN chính không còn frame trống, HĐH phải chọn một page nào đó, ví dụ P_3, để swap out nó, để lấy frame trống đó, để nạp page vừa có yêu cầu nạp, để khắc phục lỗi trang. Nhưng khi vừa khắc phục lỗi trang này thì hệ thống lại xảy ra lỗi trang mới do hệ thống cần truy xuất dữ liệu ở trang P_3, HĐH lại phải khắc phục lỗi trang này, và HĐH phải swap out một page nào đó, ví dụ P_5. Nhưng ngay sau đó hệ thống lại xảy ra lỗi trang mới do không tìm thấy

page P_5 trên BN chính và HĐH lại phải xử lý lỗi trang, và cứ như thế có thể HĐH phải kéo dài việc xử lý lỗi trang mà không thể kết thúc được. Trong trường hợp này ta nói rằng: hệ thống đã rơi vào tình trạng "trì trệ hệ thống". Như vậy hệ thống có thể xảy ra hiện tượng "trì trệ hệ thống" khi: trên BN không còn frame trống, page mà thuật toán thay trang chọn để swap out là một page không được "tốt", xét về khía cạnh dự báo lỗi trang của hệ điều hành.

Đánh dấu các trang bị thay đổi: Khi xử lý lỗi trang, HĐH thường phải thực hiện thao tác swap out, mang một page của một tiến trình tại một khung trang nào đó ra lưu tạm trên đĩa cứng, tại không gian swap. Hệ điều hành phải tốn thời gian cho thao tác swap out, điều này sẽ làm giảm tốc độ của hệ thống và có thể gây lãng phí thời gian xử lý của processor. Hệ điều hành có thể hạn chế được điều này bằng cách: *không phải lúc nào hệ điều hành cũng thực hiện swap out một page để lấy khung trang trống mà hệ điều hành chỉ thực sự swap out một page khi page đó đã bị thay đổi kể từ lần nó được nạp vào bộ nhớ gần đây nhất.* Khi đã quyết định swap out một page để lấy khung trang trống để nạp một page mới vào BN, mà page cần swap này không bị thay đổi kể từ lần nạp gần đây nhất, HĐH sẽ không swap out nó mà HĐH chỉ nạp page mới vào BN và ghi đè lên nó, điều này có nghĩa là HĐH đã tiết kiện được thời gian swap out một page tiến trình ra đĩa. Để làm được điều này HĐH phải giải quyết hai vấn đề sau: Thứ nhất, làm thế nào để xác định được một page là đã bị thay đổi hay chưa kể từ lần nạp vào BN gần đây nhất. Thứ hai, nếu không swap out một page thì khi cần HĐH sẽ swap in nó từ đâu.

Đối với vấn đề thứ nhất: HĐH chỉ cần thêm một bít, bít modify chẳng hạn, vào phần tử trong bảng trang. Khi một page vừa được nạp vào BN thì bit modify bằng 0, nếu sau đó nội dung của page bị thay đổi thì bít modify được đổi thành 1. Hệ điều hành sẽ dựa vào bít modify này để biết được một page có bị thay đổi hay không kể từ lần nạp vào bộ nhớ gần đây nhất.

Đối với vấn đề thứ hai: HĐH có thể swap in một page tại ví trí ban đầu của nó trên đĩa, hoặc tại không gian swap của nó. Trong một số HĐH khi một tiến trình được tạo thì lập tức HĐH sẽ cấp cho nó một không gian swap trên đĩa, bất kỳ khi nào tiến trình bị swap out nó đều được swap đến không gian swap của nó, khi tiến trình kết thúc thì không gian swap của nó sẽ được giải phóng. Như vậy để chuẩn bị cho việc swap in sau này, khi nạp một page của tiến trình vào bộ nhớ HĐH sẽ ghi nội dung của page này vào không gian swap của nó.

Các thuật toán thay trang

Như đã biết, để xử lý lỗi trang, trong trường hợp trên bộ nhớ không còn frame trống, HĐH phải tìm một page nào đó trên BN chính để đưa ra đĩa, để lấy frame trống đó để phục vụ cho việc xử lý lỗi trang. Khi quyết định chọn một page nào đó để đưa ra đĩa thì HĐH phải đảm bảo rằng việc chọn này là: không ảnh hưởng đến các tiến trình khác, ít có nguy cơ xảy ra lỗi trang ngay sau đó nhất và đặc biệt hệ thống khó có thể rơi vào tình trạng "trì trệ hệ thống" nhất. Trong trường hợp này HĐH đã đưa vào sử dụng các thuật

toán thay trang cụ thể như: Optimal, LRU, FIFO, Clock.

Các thuật toán thay trang khác nhau có các tiêu chí để chọn trang swap out khác nhau, nhưng tất cả đều hướng tới mục tiêu là: đơn giản và ít xảy ra lỗi trang nhất. Nó không quan tâm đến việc page được chọn để swap out là trang của tiến trình gây ra lỗi trang hay trang của một tiến trình nào đó trong hệ thống. Các thuật toán thay trang không xem xét đến các trang bị đánh dấu "neo".

Để so sánh hiệu suất xử lý lỗi trang của các thuật toán thay trang, chúng ta phải áp dụng các thuật toán này trong cùng một điều kiện: có cùng số lượng frame còn trống ban đầu và cần phải nạp một danh sách các trang như nhau vào BN. Thuật toán được gọi là có hiệu suất cao hơn khi nó xảy ra ít lỗi trang hơn.

Trong các thuật toán sau đây chúng xem xét trong trường hợp: ban đầu hệ thống có 3 frame còn trống và HĐH cần phải nạp một danh sách các trang sau đây vào BN: **2, 3, 2, 1, 5, 2, 4, 5, 3, 2, 5, 2**.

Trong các thuật toán sau đây chúng ta chỉ xét đến trường hợp b của lỗi trang, đó là HĐH phải xử lý lỗi trang khi trên BN chính không còn khung trang trống.

Thuật toán FIFO (First In First Out)

Thuật toán FIFO là thuật toán đơn giản và dễ cài đặt nhất. Với thuật toán này thì trang mà HĐH chọn để swap out là trang được đưa vào BN sớm nhất, hay ở trong BN lâu nhất.

	2	**3**	**2**	**1**	**5**	**2**	**4**	**5**	**3**	**2**	**5**	**2**
Frame 1	2	2	2	2	5	5	5	5	3	3	3	3
Frame 2		3	3	3	3	2	2	2	2	2	5	5
Frame 3				1	1	1	4	4	4	4	4	2
					F	F	F		F		F	F

Theo bảng trên thì trong trường hợp này đã xảy ra 6 lỗi trang.

Thuật toán này không phải lúc nào cũng mang lại hiệu quả tốt. Thứ nhất, có thể trang được đưa vào bộ nhớ lâu nhất lại là trang cần được sử dụng ngay sau đó, tức là HĐH vừa swap out nó thì phải swap in nó trở lại BN ngay và rõ ràng trong trường hợp này HĐH lại phải tiếp tục việc xử lý lỗi trang, trường hợp của trang 2 ở trên là một ví dụ. Thứ hai, có thể lỗi trang sẽ tăng lên khi số lượng khung trang được sử dụng tăng lên, trường hợp này được gọi là nghịch lý Belady. Khi HĐH cần nạp các trang sau đây theo thứ tự vào BN: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5 thì sẽ xảy ra 9 lỗi trang nếu sử dụng 3 khung trang, và sẽ xảy ra 10 lỗi trang nếu sử dụng 4 khung trang.

Với thuật toán này HĐH cần phải có cơ chế thích hợp để ghi nhận thời điểm một trang được nạp vào BN để làm cơ sở thay thế sau này. Trong trường hợp này HĐH thường sử dụng một danh sách liên kết để ghi nhận các trang được nạp vào BN, trang được nạp vào sớm nhất được ghi nhận ở đầu danh sách, trang được nạp vào muộn nhất được ghi nhận ở cuối danh sách và trang được chọn để thay thế là trang ở đầu danh sách. Hệ điều hành sẽ xóa phần tử ở đầu danh sách này ngay sau khi một trang đã được thay thế.

Thuật toán LRU (Least Recenty Used)

Theo thuật toán này thì trang được HĐH chọn để thay thế là trang có khoảng thời gian từ lúc nó được truy xuất gần đây nhất đến thời điểm hiện tại là dài nhất, so với các trang đang ở trên BN chính. Như vậy trong trường hợp này HĐH phải ghi nhận thời điểm cuối cùng trang được truy xuất. Bảng sau đây minh hoạ cho việc chọn trang để swap out và thay thế của thuật toán LRU:

	2	**3**	**2**	**1**	**5**	**2**	**4**	**5**	**3**	**2**	**5**	**2**
Frame 1	2	2	2	2	2	2	2	2	3	3	3	3
Frame 2		3	3	3	5	5	5	5	5	5	5	5
Frame 3				1	1	1	4	4	4	2	2	2
					F		F		F	F		

Theo bảng trên thì trong trường hợp này xảy ra 4 lỗi trang.

Thuật toán này cần phải có sự hỗ trợ của phần cứng để xác định thời điểm gần đây nhất trang được truy xuất của tất cả các trang trên BN. Có hai cách được áp dụng:

- Sử dụng bộ đếm: trong cách này, các processor thêm vào cấu trúc của các phần tử bảng trang một trường mới, tạm gọi là trường LRU, trường này ghi nhận thời điểm trang tương ứng được truy xuất gần đây nhất. Và thêm vào cấu trúc của BXL một bộ đếm (Counter). Mỗi khi có sự truy xuất BN thì Counter tăng lên một đơn vị. Mỗi khi một trang trên BN được truy xuất thì giá trị của Counter sẽ được ghi vào trường LRU tại phần tử trong bảng trang tương ứng với trang này. Như vậy trang được chọn để thay thế là trang có LRU là nhỏ nhất.
- Sử dụng Stack: trong cách này HĐH sử dụng một Stack để lưu trữ số hiệu của các trang đã được nạp vào BN chính. Khi một trang được truy xuất thì số hiệu của trang này sẽ được xóa khỏi Stack tại vị trí hiện tại và được đưa lên lại đỉnh Stack. Như vậy trang có số hiệu nằm ở đỉnh stack là trang được sử dụng gần đây nhất, trang có số hiệu nằm ở đáy stack là trang lâu nay ít được sử dụng nhất. Và trang được chọn để thay thế là các trang có số hiệu nằm ở đáy stack.

Thuật toán Optimal (tối ưu)

Theo thuật toán này thì trang được HĐH chọn để thay thế là trang sẽ lâu được sử dụng nhất trong tương lai. Bảng sau đây minh hoạ cho việc chọn trang để swap out và thay thế của thuật toán Optinal:

	2	**3**	**2**	**1**	**5**	**2**	**4**	**5**	**3**	**2**	**5**	**2**
Frame 1	2	2	2	2	2	2	4	4	4	2	2	2
Frame 2		3	3	3	3	3	3	3	3	3	3	3
Frame 3			1	1	5	5	5	5	5	5	5	5
				F			F			F		

Theo bảng trên thì trong trường hợp này chỉ xảy ra 3 lỗi trang.

Mặc dầu thuật toán này ít xảy ra lỗi trang hơn, nhưng trong thực tế khó có thể cài đặt được vì HĐH khó có thể đoán trước được khi nào thì một trang được truy xuất trở lại.

Thuật toán này không chiu tác động của nghịch lý Belady.

Cấp phát khung trang

Với kỹ thuật BN ảo phân trang thì HĐH không cần và cũng không thể mang tất cả các page của một tiến trình nạp vào BN chính để chuẩn bị thực hiện. Vì vậy HĐH cần phải quyết định nạp bao nhiêu page, bao nhiêu tiến trình vào BN. Hay chính xác hơn là nạp bao nhiêu tiến trình và mỗi tiến trình được nạp bao nhiêu page vào BN (được cấp bao nhiêu khung trang). Hệ điều hành có thể quyết định vấn đề này theo các chọn lựa sau đây:

- Chỉ có một lượng nhỏ, có thể là tối thiểu, các page của tiến trình được nạp vào BN. Như vậy HĐH sẽ nạp được nhiều tiến trình vào BN tại bất kỳ thời điểm nào. Điều này làm tăng khả năng đa chương của HĐH và khả năng tìm thấy một tiến trình Ready của HĐH là rất lớn nhờ vậy mà hiệu quả điều phối của HĐH tăng lên. Nhưng trong trường hợp này HĐH phải luôn chú ý đến việc nạp thêm các page của tiến trình vào BN và HĐH khó có thể xác định được số lượng khung trang tối thiểu mà mỗi tiến trình cần khi khởi tạo.
- Nếu có một lượng vừa phải các page của một tiến trình trong BN chính thì có ít hơn số tiến trình được nạp vào bộ nhớ. Như vậy sự đa chương sẽ giảm xuống nhưng tốc độ thực hiện của tiến trình có thể được cải thiện vì khi một chỉ thị của các page trong BN chính cần truy xuất đến một page khác thì nhiều khả năng page này đã có trên BN chính. Nhưng lý thuyết HĐH đã chứng minh được rằng trong trường hợp này tỉ lệ xảy ra lỗi trang là rất lớn.
- Nếu có một lượng lớn các page của một tiến trình trong BN chính, thì sự đa chương sẽ giảm xuống đáng kể. Nhưng lý thuyết HĐH đã chứng minh được rằng trong trường hợp này tỉ lệ xảy ra lỗi trang là rất thấp. Mặt khác điều này có thể gây lãng phí BN vì có thể có các page của một tiến trình rất ít được sử dụng khi nó ở trên BN chính.

Theo trên thì mỗi chọn lựa đều có những điểm thuận lợi và những điểm chưa thuận lợi riêng, do đó tùy trường hợp cụ thể mà HĐH thực hiện cấp phát khung trang cho tiến trình theo một chọn lựa nào đó, để đảm bảo có nhiều tiến trình được nạp vào BN chính, nhưng khả năng và tỉ lệ lỗi trang là thấp nhất và sự lãng phí BN là thấp nhất. Để đáp ứng điều này các HĐH thường thực hiện việc cấp phát khung trang cho các tiến trình theo hai chính sách: *Cấp phát tĩnh và Cấp phát động*

- Chính sách cấp phát tĩnh *(fixed – allocation)*: Với chính sách này HĐH sẽ cấp cho mỗi tiến trình một số lượng khung trang cố định, để nạp đủ các page của tiến trình vào bộ nhớ để nó có thể hoạt động được. Số lượng khung trang này được quyết định tại thời điểm khởi tạo tiến trình. Hệ điều hành cũng có thể quyết định số lượng khung trang tối thiểu cho tiến trình dựa vào loại của tiến trình, đó là tiến trình tương tác, tiến trình xử lý theo lô hay tiến trình theo hướng ứng dụng. Với cấp phát tĩnh, khi có lỗi trang xảy ra trong quá trình thực hiện tiến trình thì hệ điều hành phải swap out một page của tiến trình đó để thực hiện việc xử lý lỗi trang.
- Chính sách cấp phát động *(variable - allocation):* Với chính sách này HĐH chỉ

cấp một lượng vừa đủ khung trang, để nạp đủ các trang cần thiết nhất của tiến trình, để tiến trình có thể khởi tạo và hoạt động được, sau đó tuỳ theo yêu cầu của tiến trình mà HĐH có thể cấp phát thêm khung trang cho nó, để nạp thêm các trang cần thiết khác. Hệ điều hành thường cấp thêm khung trang cho tiến trình khi tiến trình bị rơi vào tình trạng lỗi trang, với các tiến trình có tần suất xảy ra lỗi trang lớn thì HĐH phải cung cấp một lượng khung trang lớn, một cách vượt bật, đủ để tiến trình thoát ra khỏi lỗi trang và nguy cơ lỗi trang tiếp theo là thấp nhất.

Hiệu suất phân trang theo yêu cầu

Hiệu suất phân trang theo yêu cầu là thông số trực tiếp tác động đến hiệu suất của hệ thống. Hiệu suất phân trang theo yêu cầu được đánh giá thông qua thời gian truy cập có ích. Khi phân trang theo yêu cầu, khi không có lỗi trang, thời gian truy cập có ích bằng thời gian truy cập BN trong, t_{BN} bằng khoảng 10 đến 200ns. Khi có lỗi trang, hệ thống phải tải trang từ ổ đĩa cứng, sau đó mới truy nhập đến trang nhớ mong muốn.

Gọi t_{ci} là thời gian truy cập có ích;

t_{xl} là thời gian xử lý lỗi trang;

t_{BN} là thời gian truy cập BN trong;

p là xác suất lỗi trang.

Khi đó, thời gian truy cập có ích được xác định:

$$t_{ci} = (1 - p).\ t_{BN} + p.t_{xl}$$

Khi xử lý lỗi trang, HĐH thường phải thực hiện các công việc sau đây: chuyển quyền điều khiển cho HĐ; lưu lại ngữ cảnh tiến trình; xác nhận ngắt hiện tại do lỗi trang gây ra; kiểm tra việc tham chiếu đến trang hợp lệ và xác định vị trí của trang trên ổ đĩa; ra lệnh tải trang từ ổ đĩa cứng vào frame trống (đợi trong hàng đợi của thiết bị, chờ đầu đọc dịch chuyển trên ổ đĩa, bắt đầu chuyển trang vào frame trống); trong lúc chờ đợi, chuyển CPU cho tiến trình khác (khôn bắt buộc); xuất hiện ngắt từ ổ đĩa cứng (hoàn tất thao tác đọc); lưu ngữ cảnh của tiến trình đang chiếm dụng CPU (nếu có bước 6); xác nhận ngắt là ngắt từ ổ đĩa cứng; cập nhật lại bảng phân trang để chỉ ra trang cần truy cập hiện đã nằm trong BN; chờ CPU được cấp phát lại cho tiến trình này; khôi phục ngữ cảnh tiến trình bị phong tỏa ban đầu, sau đó tiếp tục thi hành chỉ thị đã bị ngắt.

Thời gian thực hiện các công việc trên từ 1 đến 100ms. Thời gian hoán chuyển trang khoảng 24ms (độ trễ ổ đĩa cứng 8ms, dịch chuyển đầu đọc 15ms, truyền dữ liệu 1ms). Như vậy, tổng thời gian phân trang khoảng 25ms (chỉ xét thời gian phục vụ của thiết bị).

Khi xét thời gian xử lý lỗi trang trung bình là 25ms, thời gian truy cập BN là 100ns, thì thời gian truy cập có ích (tính theo ns) sẽ là:

$$t_{ci} = (1 - p).\ 100 + p.25000000 = 100 + 24999900.p$$

Nếu xác suất lỗi trang $p = 10^{-3}$, t_{ci} khoảng 25ms và tốc độ máy giảm 250 lần.

Với $p = 4.10^{-7}$, t_{ci} khoảng 110ns và hiệu suất máy tính chỉ giảm 10%.

3.3.2.2. Phân đoạn theo yêu cầu

Phân trang theo yêu cầu cho phép BN ảo có hiệu quả nhất, nhưng cần nhiều phần cứng hỗ trợ. Phân đoạn theo yêu cầu sẽ khắc phục được nhược điểm này.

Phân đoạn theo yêu cầu hoàn toàn tương tự như phân trang theo yêu cầu. Trong các bộ mô tả đoạn, ngoài các thông tin về kích thước, mức bảo vệ, vị trí của đoạn, còn có trường bít hợp lệ, nhằm xác định đoạn hiện có có nằm trong BN hay không. Quá trình truy cập vào đoạn BN, trong cả 2 trường hợp giống như phân trang theo yêu cầu.

NGUYÊN LÝ HỆ ĐIỀU HÀNH

Giảng viên: Nguyễn Vũ Quốc Hưng
Khoa Công nghệ Thông tin – ĐHSP Hà nội
Email: hungnvq@hnue.edu.vn

Những mục tiêu chính:

- Cung cấp cho sinh viên:
  - Khái niệm cơ bản và nguyên lý của các Hệ điều hành
  - Các phương pháp giải quyết các vấn đề nảy sinh
  - Xem xét một số Hệ điều hành nổi tiếng: Linux, Windows,…

Các kiến thức cần thiết

- Kiến trúc máy tính và Cấu trúc máy tính
- Cấu trúc dữ liệu và Giải thuật
- Ngôn ngữ lập trình: Pascal, C,…
- …

Tài liệu tham khảo

- A. Silberschatz, G. Gagne, and P.B. Galvin, "*Operating System Concepts", Addison Wesley*
- A.S. Tanenbaum, *"Modern Operating Systems"*, *Prentice Hall, 1992*
- Bill McCarty, *"Learning Redhat Linux 3rd Edition"*, *O'Reilly, 2003*
- D.P. Bovet, M. Cesati, *"Understanding Linux Kernel", O'Reilly,2000*
- J.S. Gray, *"Interprocess Communications in Linux: The Nooks and Crannies"*, *Prentice Hall, 2003*
- A. Rubini, J. Corbet, *"Linux Device Drivers 2nd Edition"*, *O'Reilly*
- N.T.N. Hương, *"Hệ điều hành", NXB Bưu điện, 2004*
- ...

Phân phối, Bài tập và Kiểm tra

- Phân phối thời gian
  - 75% dành cho giảng bài
  - 25% dành cho các bài tập lớn
- n
  - p
- Điểm đánh giá.
  - Bài tập lớn: lấy điểm kiểm tra giữa kì
  - Kiểm tra: theo quy chế

Nội dung

- Chương 1: Dẫn nhập
- Chương 2: Quản lý tiến trình
- Chương 3: Quản lý bộ nhớ
- Chương 4: Quản lý hệ thống file
- n DeadLock
- n găng

Chương 1

P

- Khái niệm và Định nghĩa
- Lịch sử OS
- Tổ chức hệ thống máy tính
- Các thành phần của OS
- Các dịch vụ của OS
- Cấu trúc của OS

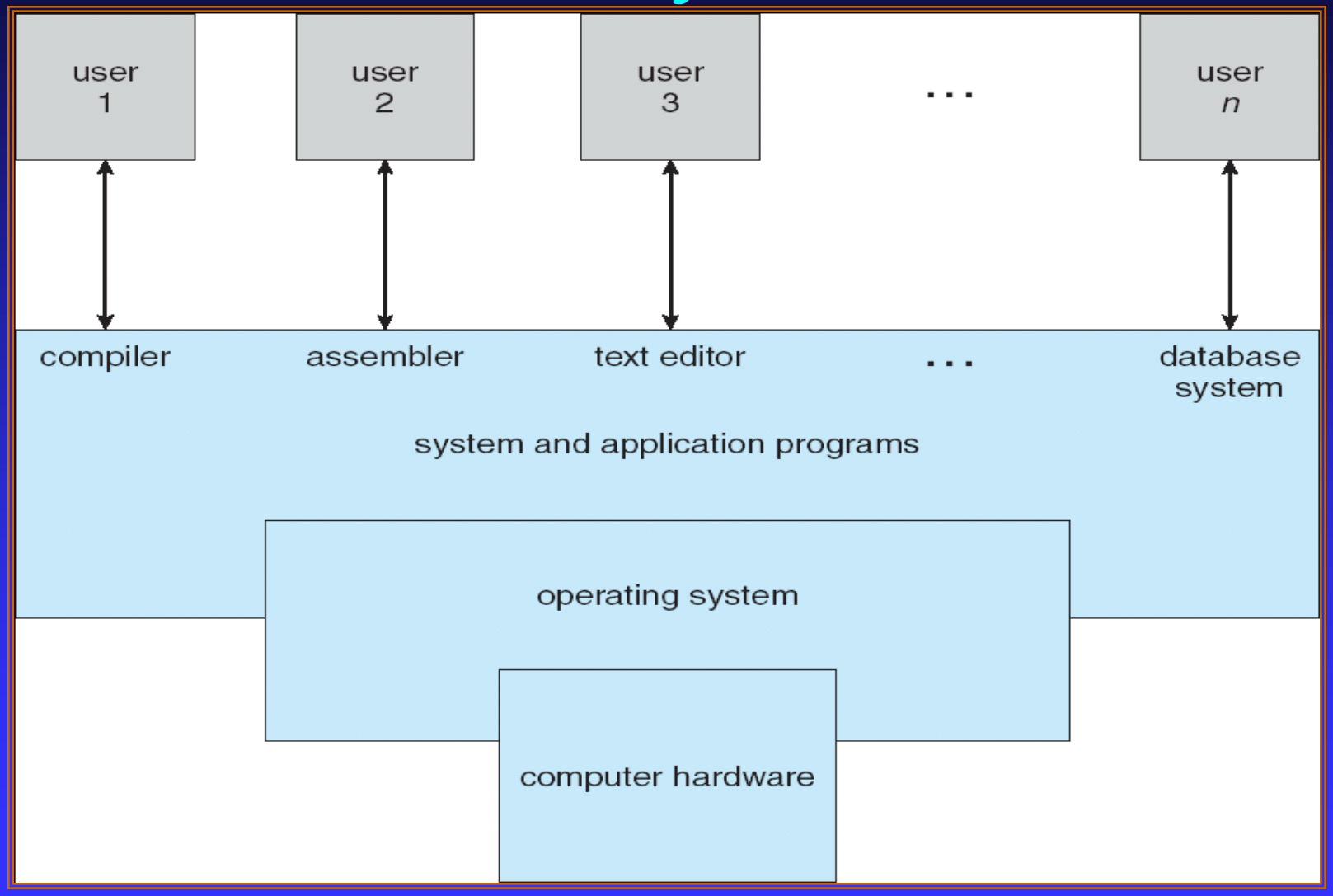
m

- nh (Operating System)
  - OS là phần mềm hoạt động ở lớp trung gian giữa người sử dụng máy tính và phần cứng.
  - Mục tiêu của OS là làm cho người sử dụng:
    - Thực thi dễ dàng các ứng dụng của mình
    - Thao tác điều khiển máy tính trở nên thuận tiện.
    - Khai thác phần cứng máy tính một cách có hiệu quả

m 4 thành phần:

- Phần cứng (Hardware) – cung cấp các tài nguyên cơ bản
  - CPU, memory, I/O devices
- Hệ điều hành (OS - Operating system)
  - Trung gian điều khiển và bố trí việc sử dụng phần cứng cho các ứng dụng và đối tượng sử dụng
- Các chương trình ứng dụng (Application programs) – Các phần mềm phục vụ tác nghiệp của người sử dụng.
  - Word processors, compilers, web browsers, database systems, video games
- Đối tượng sử dụng (Users):
  - Người, thiết bị hoặc máy tính khác

a máy tính


user 1
user 2
user 3
...
user n
compiler
assembler
text editor
...
database system
system and application programs
operating system
computer hardware

C

- OS là tài nguyên:
  - Quản lý tất cả các tài nguyên khác
  - Làm trung gian giữa những mâu thuẫn về hiệu năng của tài nguyên và sự sử dụng chúng một cách thuận tiện
- OS là chương trình điều khiển
  - Điều khiển sự thực hiện các chương trình, ngăn ngừa lỗi và sự sử dụng không thích hợp
- "Chương trình chạy trong toàn bộ thời gian hoạt động của máy tính" đó chính là **kernel.** Mọi chương trình khác là chương trình hệ thống hoặc chương trình ứng dụng

n

- Hệ thống khung lớn (Mainframe Systems)
- Hệ thống để bàn (Desktop Systems)
- Hệ thống đa bộ xử lý (Multiprocessor Systems)
- Hệ thống phân tán (Distributed Systems)
- Hệ thống bó (Clustered Systems)
- Hệ thống thời gian thực (Real – Time Systems)
- Hệ thống cầm tay (Handheld Systems)
- Môi trường tính toán (Computing Environments)

2.1 Mainframe Systems

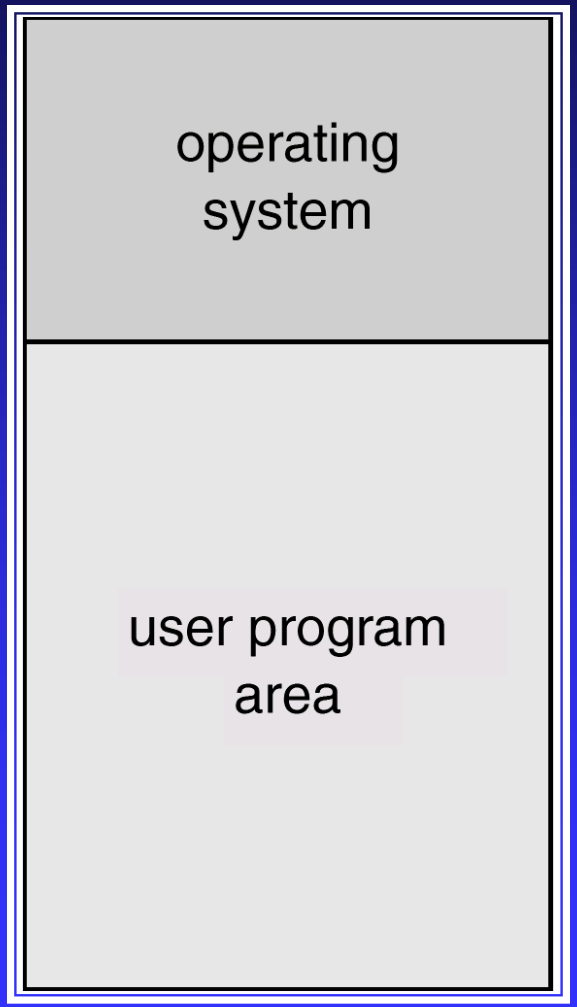
n

n.

Hàng đợi công việc tự động chuyển điều khiển từ công việc này đến công việc khác.

- Hạt nhân là bộ giám sát
  - Khởi động công việc
  - Điều khiển chuyển đổi công việc
  - Khi công việc được hoàn thành việc điều khiển chuyển lại cho bộ giám sát

Sơ đồ bộ nhớ của một hệ thống theo lô đơn giản (Simple Batch System)



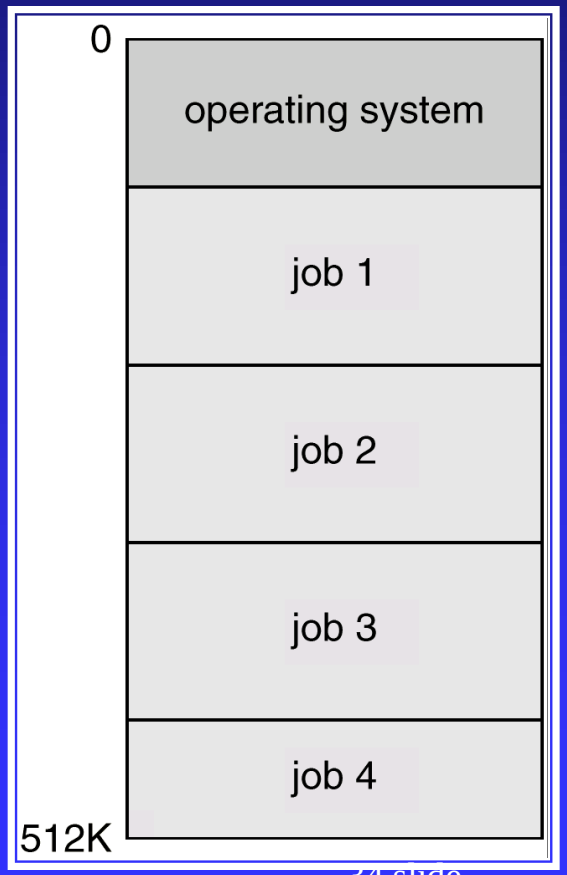
b) Hệ thống lô đa chương (Multiprogramming Systems)

Multiprogramming có các khả năng:

- Một user đơn lẻ không được sử dụng CPU và thiết bị I/O toàn thời gian.
- Khi các công việc được tổ chức đa chương (code and data) thì CPU luôn luôn thực thi một trong nó.
- Một tập con của tất cả các công việc được lưu trữ trong bộ nhớ
- Một trong những công việc đó được chọn và hoạt động theo **job scheduling**
- Khi một công việc rơi vào trạng thái chờ (ví dụ chờ I/O), OS sẽ chuyển sang công việc khác

Hệ thống theo lô đa chương (Multiprogrammed Batch Systems)

Một vài công việc được lưu trữ trong bộ nhớ chính tại cùng môt thời điểm, CPU phục vụ một trong những công việc đó.



Những đặc tính cần thiết để Multiprogramming

- Hệ thống cung cấp các chương trình điều khiển I/O.
- Quản trị bộ nhớ – hệ thống phải có khả năng đinh vị trong bộ nhớ một vài chương trình.
- Lập lịch CPU – hệ thống phải lựa chọn một vài công việc sẵn sàng để chạy.
- Định vị các thiết bị.

c) Hệ thống chia sẻ thời gian (Timesharing Systems)

- **Timesharing (multitasking – đa nhiệm)** là khái niệm mở rộng về mặt logic chỉ việc chuyển điều khiển giữa các công việc thường xuyên đến mức các users tương tác được đến mỗi công việc khi nó đang chạy, tạo ra máy tính tương tác.
  - **Thời gian điều chuyển phù hợp** < 1 second
  - Mỗi user có ít nhất một chương trình trong bộ nhớ ⇨tiến trình **(process)**
  - Nếu một vài công việc sẵn sàng chạy tại cùng một thời điểm ⇨ **CPU scheduling**
  - Nếu các tiến trình không đặt vừa trong bộ nhớ, **swapping** sẽ được thực hiện.
  - **Virtual memory** cho phép thi hành các tiến trình không đặt trọn vẹn trong bộ nhớ.

2.2 Desktop Systems

- *Personal computers* – hệ thống máy tính cho phép một người sử dụng.
- n – bàn phím, chuột, màn hình, máy in.
- Người sử dụng dễ dàng và thuân lợi.
- Có thể sử dụng công nghệ của các hệ thống lớn đã được cá nhân hóa cho người sử dụng nhưng không cần thiết các tính năng bảo vệ cao cấp.
- Có thể chạy với một vài OS (Windows, MacOS, UNIX, Linux)

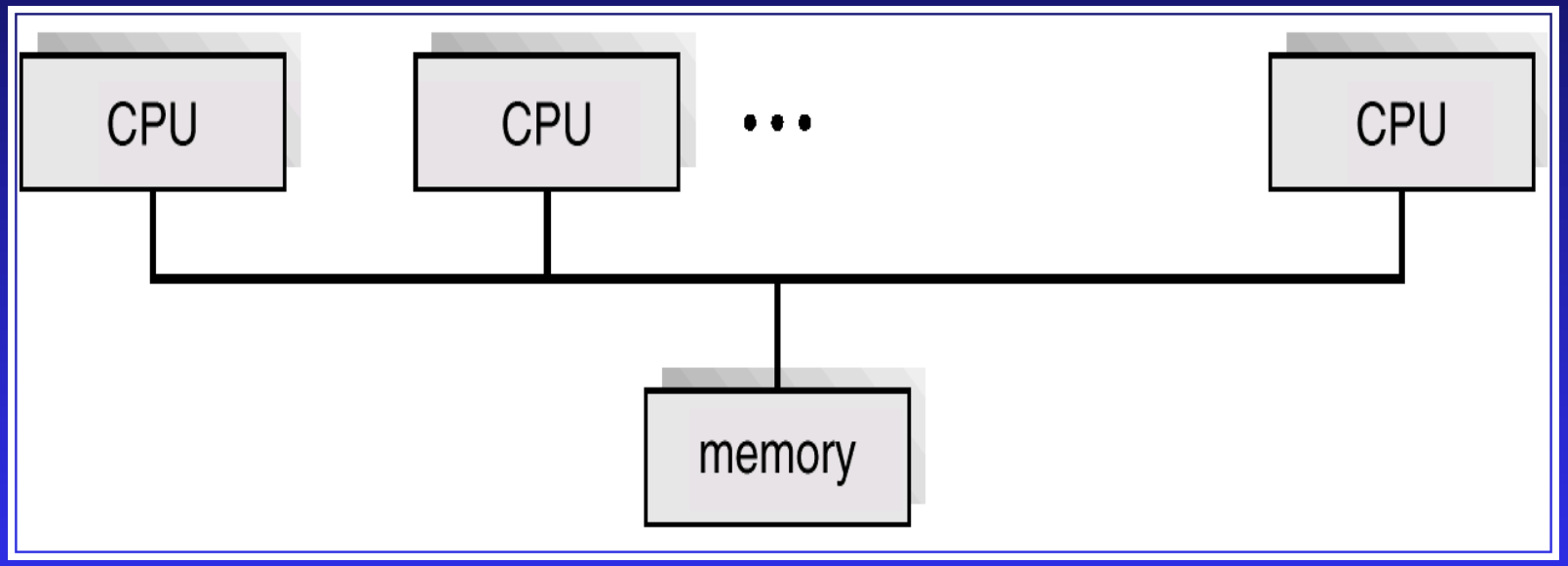
2.3 Parallel Systems

- Multiprocessor systems có nhiều CPU trong một hệ thống liên lạc khép kín.
- *Tightly coupled system* – các processor chia sẻ bộ nhớ và đồng hồ xung; truyền tin luôn luôn thông qua bộ nhớ được chia sẻ.
- Các lợi ích của parallel system:
  - Gia tăng thông lượng truyền tin
  - Kinh tế
  - Gia tăng độ tin cậy

Parallel Systems (Cont.)

- *Symmetric multiprocessing (SMP)*
  - Mỗi processor chạy độc lập trên một bản sao hệ điều hành như nhau.
  - Cho phép nhiều tiến trình cùng chạy đồng thời trên một hệ thống.
  - Nhiều hệ điều hành hiện đại hỗ trợ SMP
- *Asymmetric multiprocessing*
  - Mỗi bộ xử lý được giao một nhiệm vụ riêng biệt; một bộ xử lý chủ (master) lập lịch và xác định công việc cho các bộ xử lý thành viên (slave)
  - Phổ biến trên những hệ thống lớn

Symmetric Multiprocessing Architecture



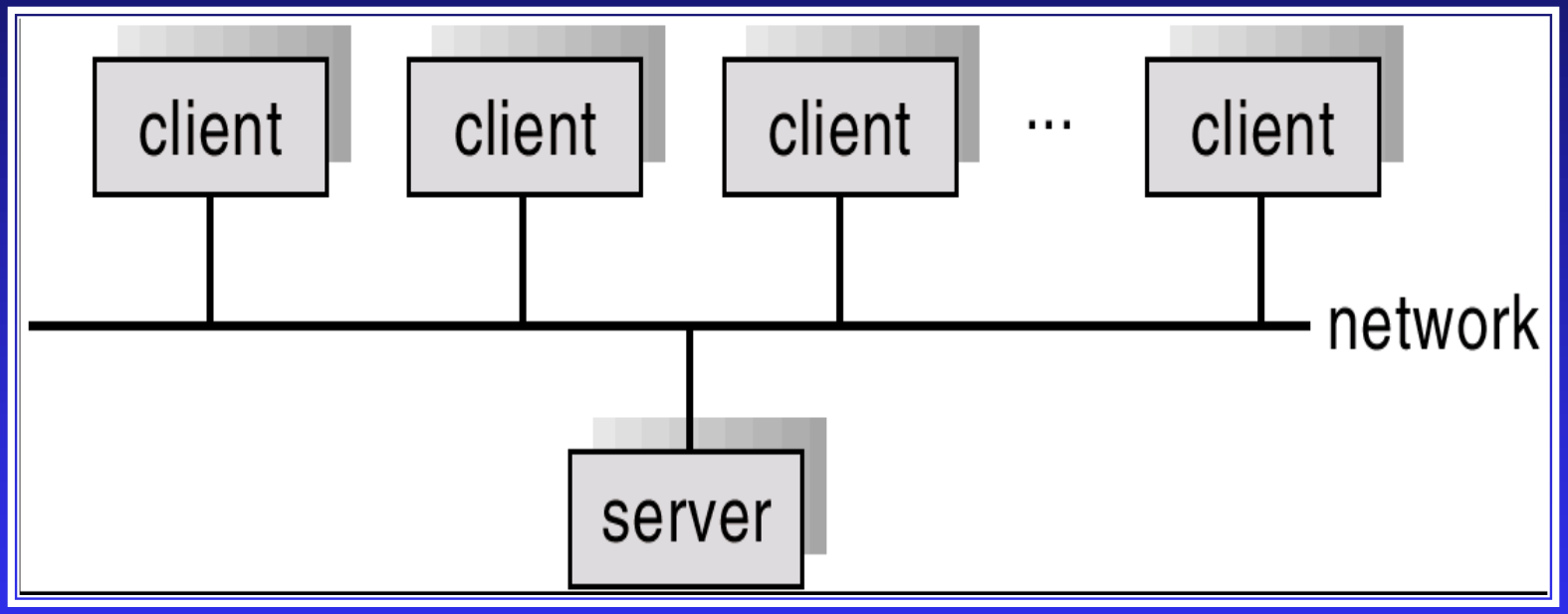
2.4 Distributed Systems

- Phân tán sự tính toán trên một vài bộ xử lý vật lý.
- *Loosely coupled system* – mỗi bộ xử lý có một bộ nhớ riêng của nó. Các bộ xử lý truyền thông tin cho nhau trên đường truyền tốc độ cao.
- Các lợi ích của distributed systems.
  - Chia xẻ tài nguyên
  - Tăng tốc độ tính toán
  - Đáng tin cậy
  - Communications

Distributed Systems (tiếp)

- Đòi hỏi cơ sở hạ tầng mạng máy tính.
- Local area networks (LAN) hoặc Wide area networks (WAN)
- Có hai dạng hệ thống client-server hoặc hệ thống peer-to-peer.

Cấu trúc chung của Client-Server



2.5 Clustered Systems

- Tập hợp hai hoặc nhiều hệ thống có bộ lưu trữ chia sẻ.
- Cung cấp độ an toàn cao.
- *Asymmetric clustering*: Một server chạy các ứng dụng trong khi các server khác dự phòng.
- *Symmetric clustering*: Tất cả N host đều chạy ứng dụng.

2.6 Real-Time Systems

- Luôn được sử dụng như những thiết bị điều khiển trong những ứng dụng chuyên biệt: Thí nghiệm khoa học, hệ mô phỏng y học, điều khiển công nghiệp, ...
- Phải hoàn thành công việc trong một thời gian xác định cưỡng bức.
- Hệ thống Real-Time có thể là *hard* hoặc *soft* real-time.

Real-Time Systems (tiếp)

- Hard real-time:
  - Bộ lưu trữ thứ cấp có giới hạn hoặc không có, dữ liệu được lưu trữ trong bộ nhớ nhỏ hoặc trong ROM.
  - Trái ngược với hệ thống time-sharing và không hỗ trợ các hệ điều hành đa năng.
- Soft real-time
  - Sử dụng điều khiển các robot công nghiệp.
  - Sử dụng trong các ứng dụng multimedia, thực tại ảo (virtual reality) đòi hỏi các tính năng cao cấp của hệ điều hành.

2.7 Handheld Systems

- Personal Digital Assistants (PDAs)
- Cellular telephones (điện thoại di động)
- Một số đặc điểm:
  - Bộ nhớ giới hạn
  - Bộ xử lý tốc đô chậm
  - Màn hình nhỏ.

2.8 Computing Environments

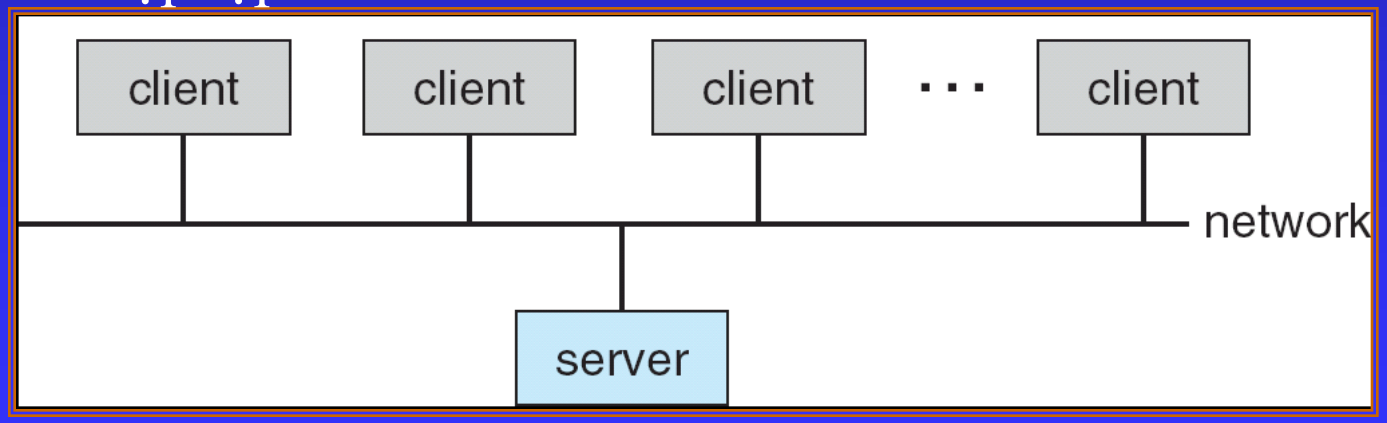
- Môi trường tính toán truyền thống (Traditional computing)
- Môi trường tính toán dựa trên công nghệ Web (Web-Based Computing)
- Môi trường tính toán nhúng (Embedded Computing)

Computing Environments

- Traditional computer
  - Thời gian tính không đòi hỏi chặt chẽ
  - Môi trường cộng tác
    - PCs liên kết tới một mạng, terminal gắn với mainframe hoặc minicomputer xử lý theo khối và chia sẻ thời gian
    - Thông qua các cổng và hệ thống từ xa truy nhập đền nguồn tài nguyên như nhau.
  - Mạng gia đình
    - Sử dụng các máy tính đơn, kết nối modem
    - Không firewall, không kết nối mạng.

Computing Environments (tiếp)

- Client-Server Computing
  - Dumb terminals được thay thế bởi PC thông minh.
  - Có nhiều loại **server**, phù hợp với các đòi hỏi của **clients**
    - **Compute-server** cung cấp giao diện cho client yêu cầu dịch vụ (chẳng hạn database)
    - **File-server** cung cấp giao diện cho client lưu trữ và nạp tập tin



Peer-to-Peer Computing

- Một mô hình khác của hệ thống phân tán.
- P2P không phân chia các máy thành client và server.
  - Coi tất cả các nút đều có vai trò như nhau.
  - Mỗi nút hoạt động như client, server hoặc đóng vai trò cả hai.
  - Mỗi nút phải kết nối bởi mạng P2P
    - Đăng kí dịch vụ với một trung tâm tra cứu dịch vụ trên mạng,
    - Quảng bá yêu cầu dịch vụ và đáp ứng các yêu cầu dịch vụ theo giao thức *discovery protocol*
  - Ví dụ: *Napster và Gnutella*

Web-Based Computing

- Web đã trở thành phổ biến
- Máy tính PC trở thành thiết bị thông dụng
- Nhiều thiết bị được kết nối mạng và liên kết theo kiểu web.
- Nhiều thiết bị loại mới quản lý truyền tải web tương đương server: **load balancers.**
- Sử dụng các hệ điều hành Windows 95 kiểu client-side và phát triển thành Linux và Windows XP có thể phân chia thành clients và servers