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.

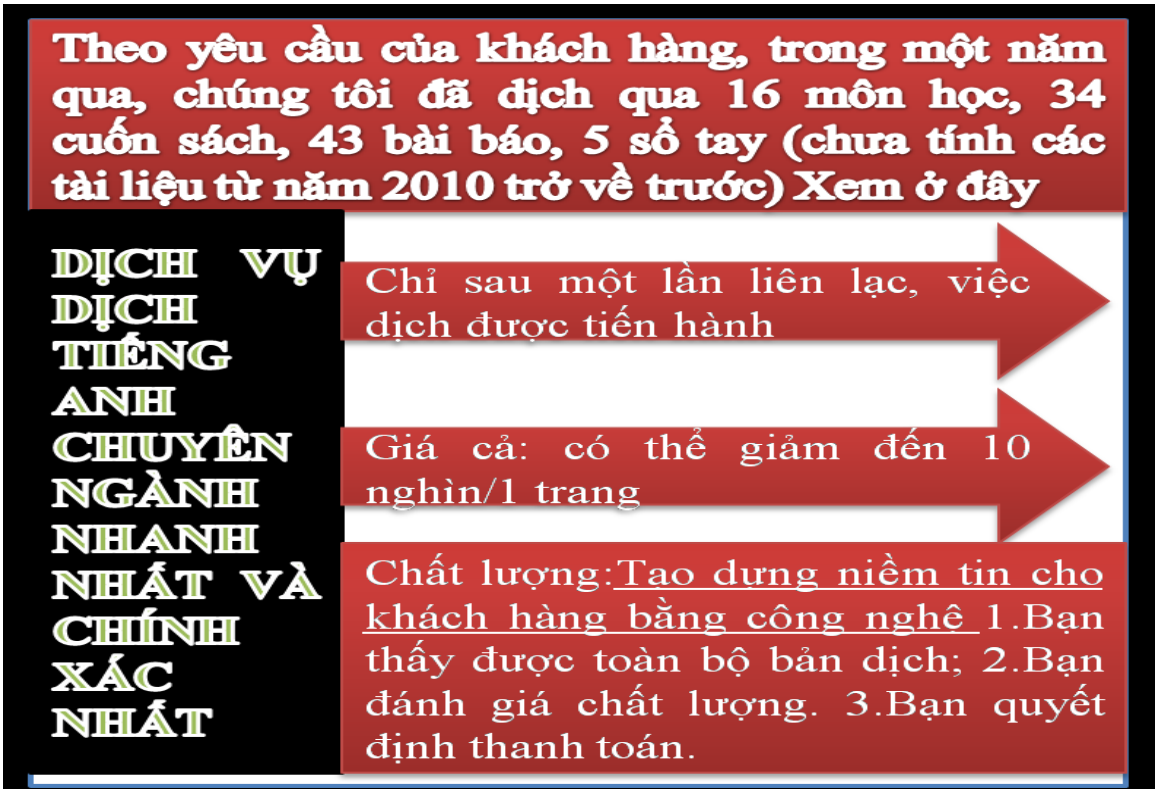
Bạn có thể tham khảo nguồn tài liệu được dịch từ tiếng Anh tại đây:

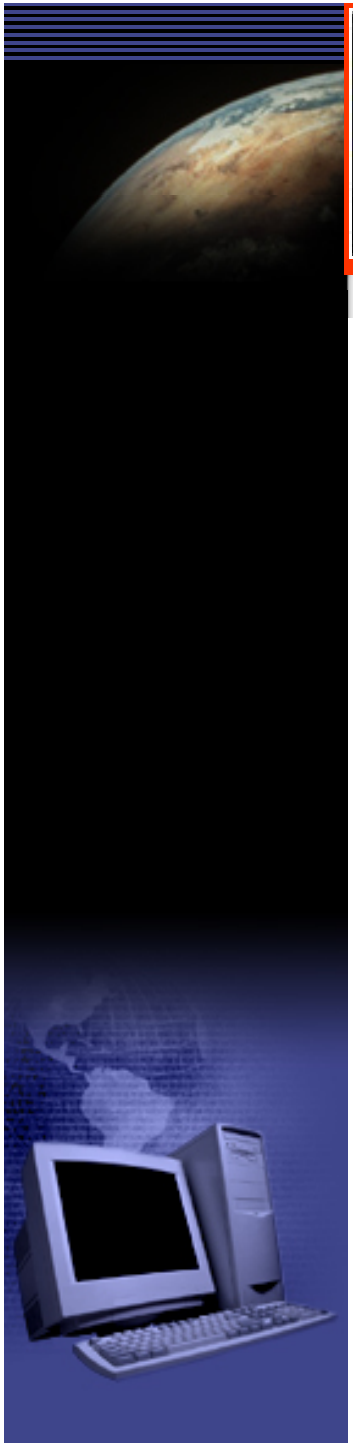
http://mientayvn.com/Tai_lieu_da_dich.html

Thông tin liên hệ:

Yahoo mail: thanhlam1910_2006@yahoo.com

Gmail: frbwrthes@gmail.com



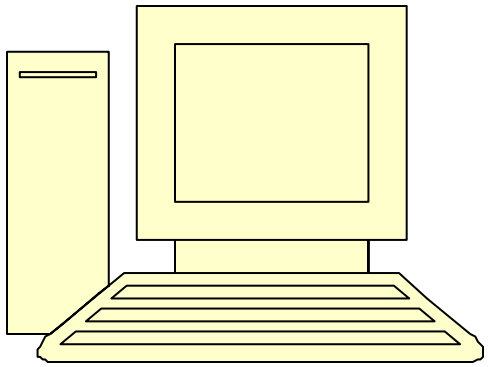



# ĐẠI HỌC ĐÀ NẴNG

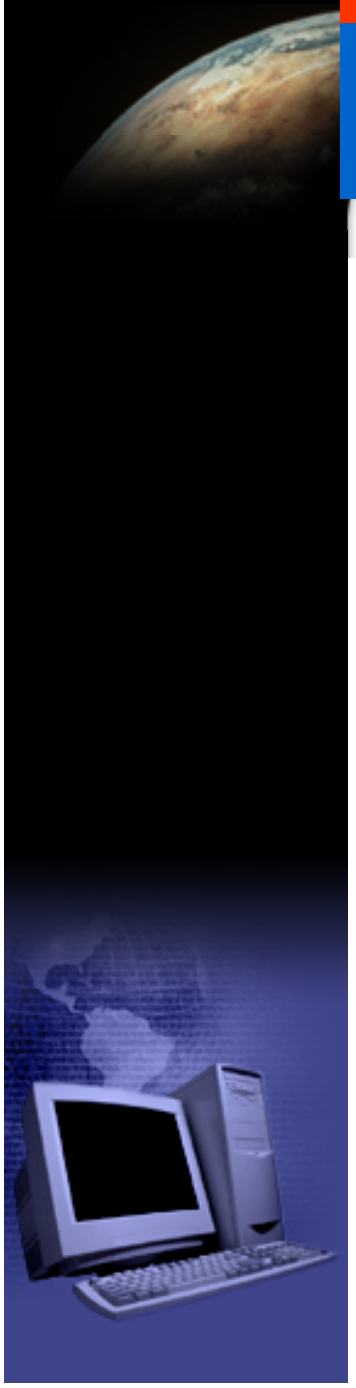
## TRƯỜNG ĐẠI HỌC BÁCH KHOA

## KHOA CÔNG NGHỆ THÔNG TIN

# NGUYÊN LÝ HỆ ĐIỀU HÀNH



# Giới thiệu



## Nội dung giáo trình

**CHƯƠNG 1. MỞ ĐẦU**

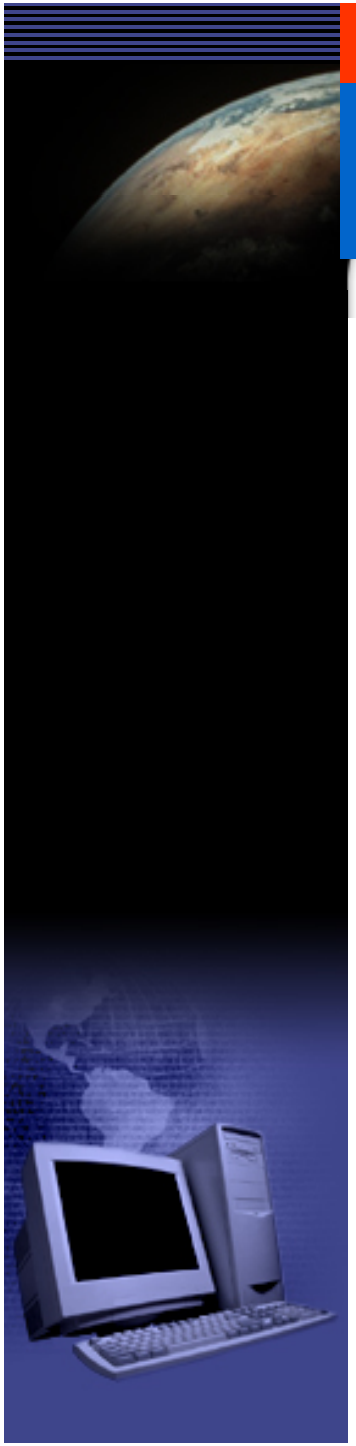
**CHƯƠNG 2. TIẾN TRÌNH**

**CHƯƠNG 3. VÀO/RA**

**CHƯƠNG 4. QUẢN LÝ BỘ NHỚ**

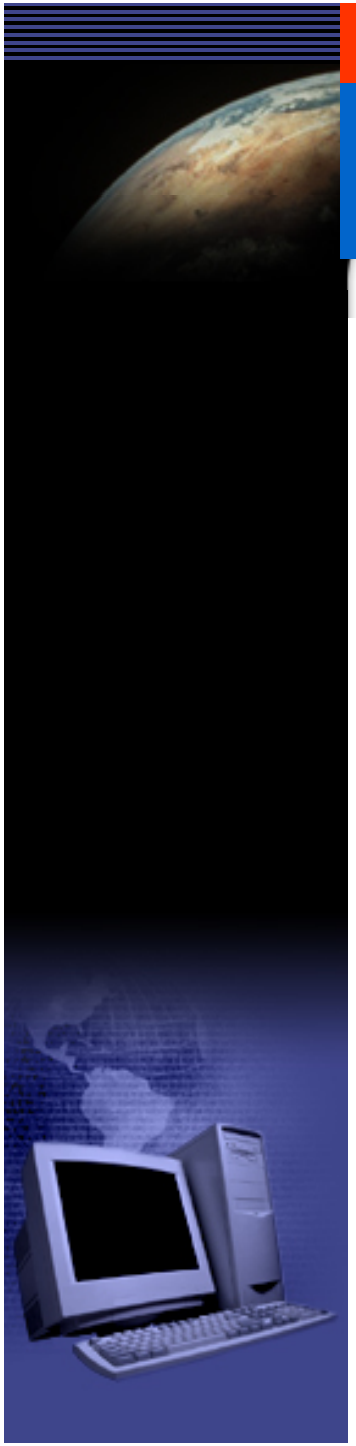
**CHƯƠNG 5. HỆ THỐNG FILE**

## CHƯƠNG 1. MỞ ĐẦU



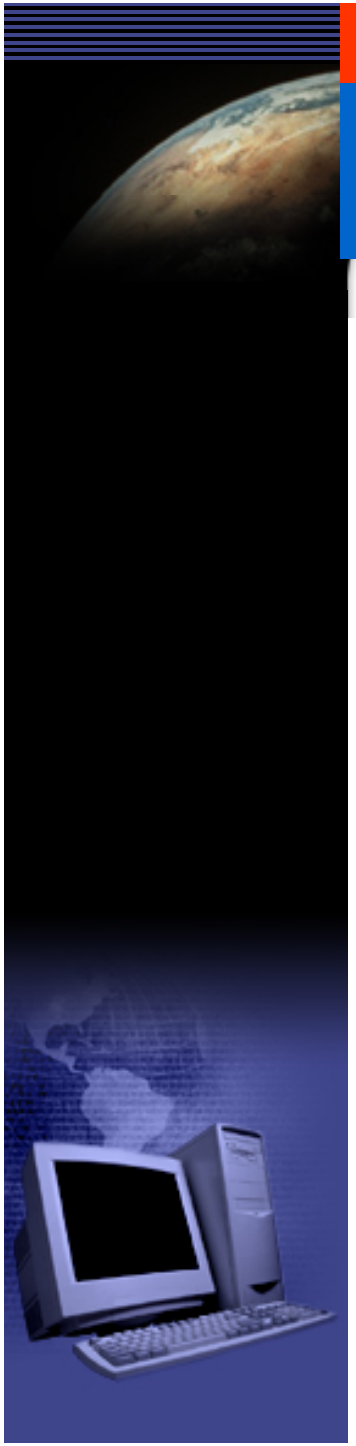
# Các vấn đề

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

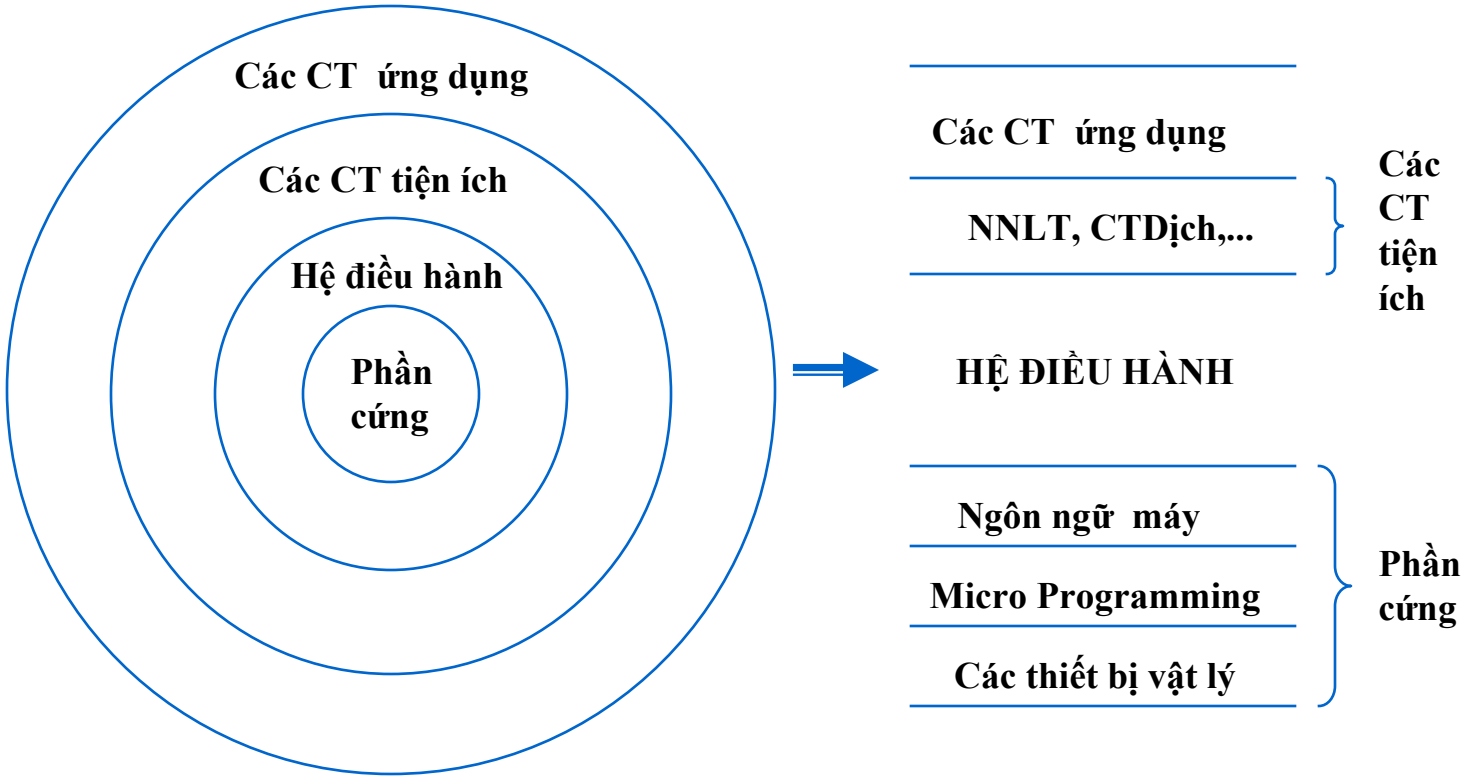
# Tài nguyên

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

**Các file, chương trình dùng chung,...**

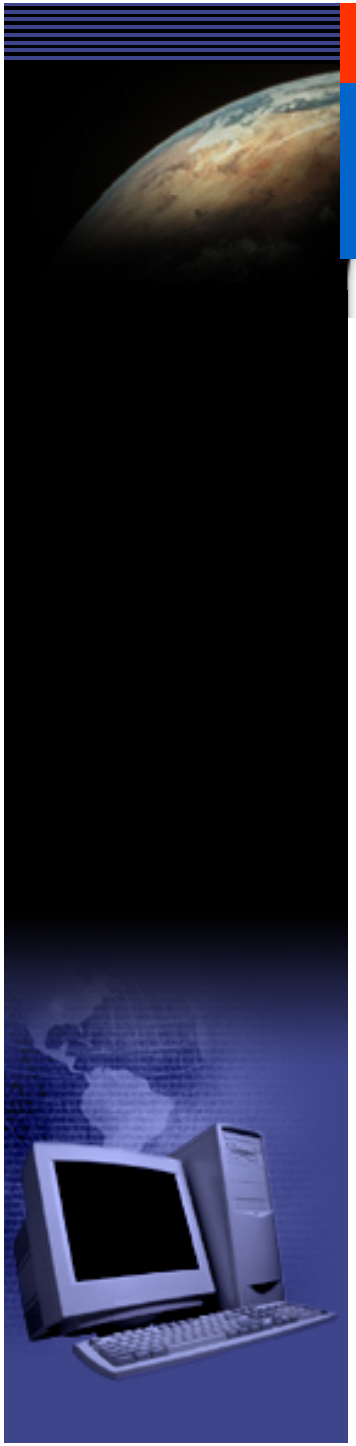
## CHƯƠNG 1. MỞ ĐẦU

# Vị trí của hệ điều hành



# Các thành phần của hệ điều hành

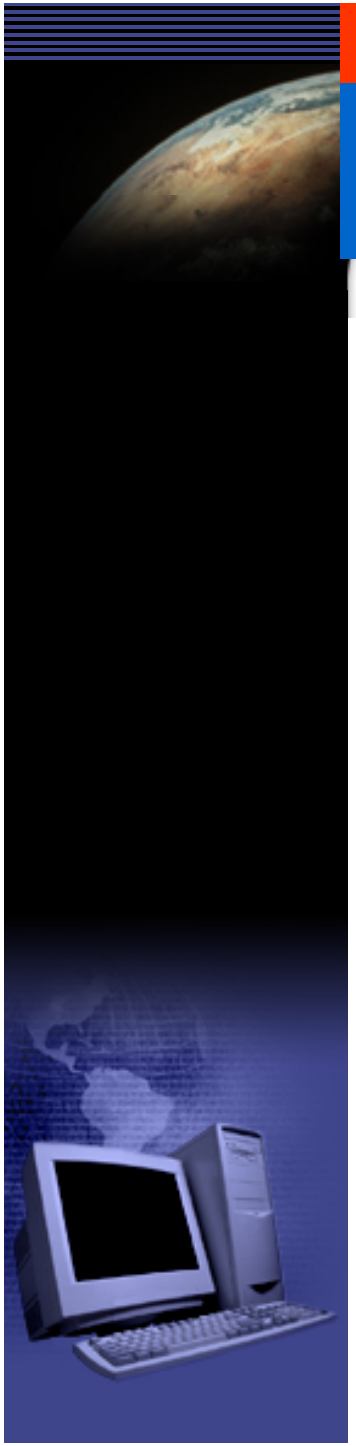
- Quản lý tiến trình
- Quản lý bộ nhớ
- Quản lý nhập xuất
- Quản lý tập tin
- Hệ thống bảo vệ
- Hệ thông dịch lệnh (Shell)
- Quản lý mạng



# 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

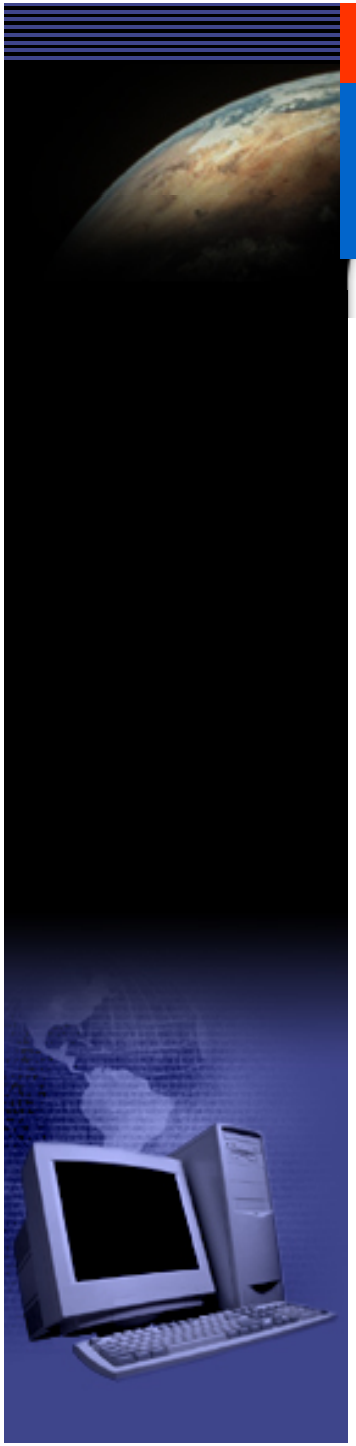
## Quản lý bộ nhớ

- 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

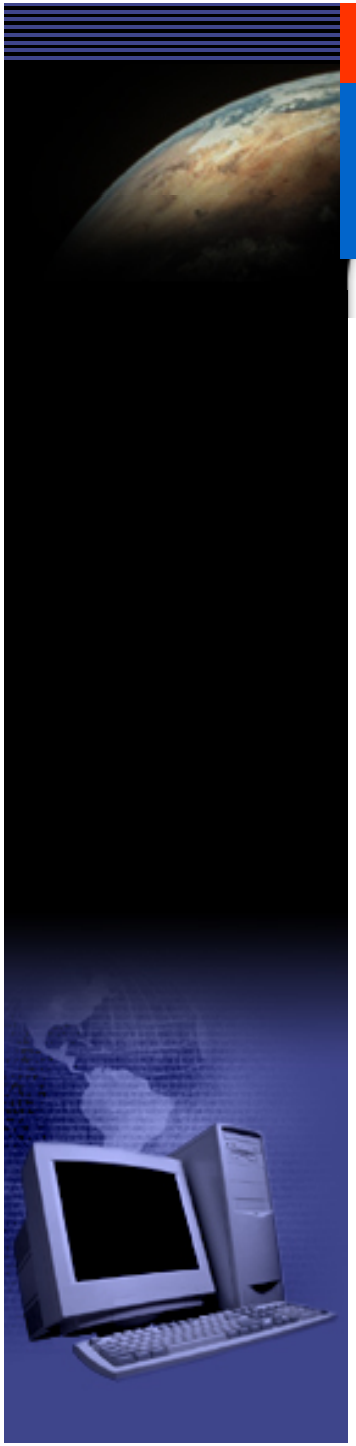
- **Xây dựng cơ chế bảo vệ thích hợp.**

**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

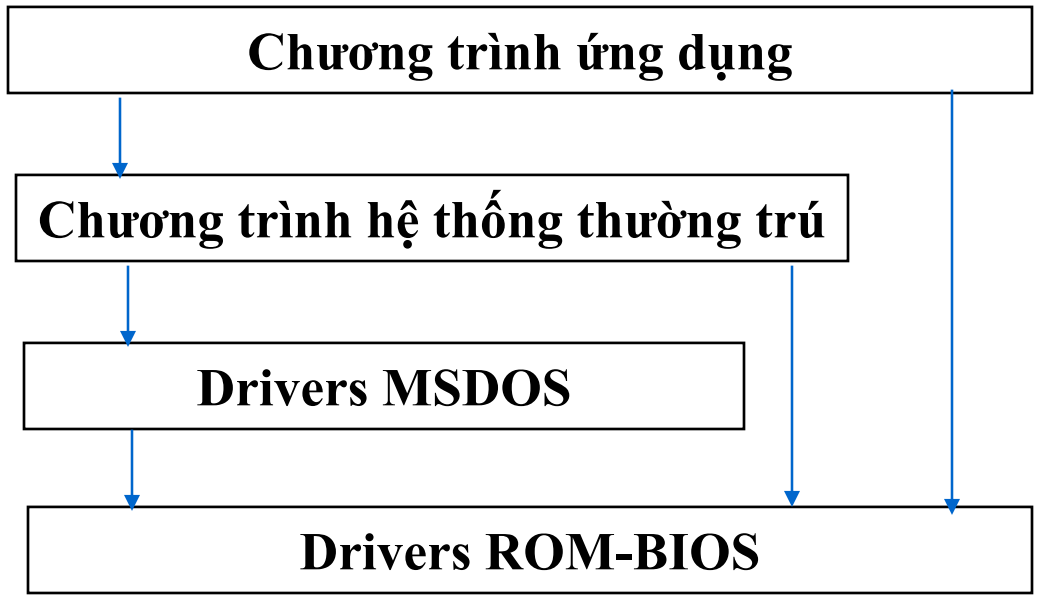
- Hệ thống nguyên khối (Monolithic System)
- Hệ thống phân lớp (Layer System)
- Máy ảo (Virtual Machine)
- Mô hình Client-Server (Client-Server Model)

# 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

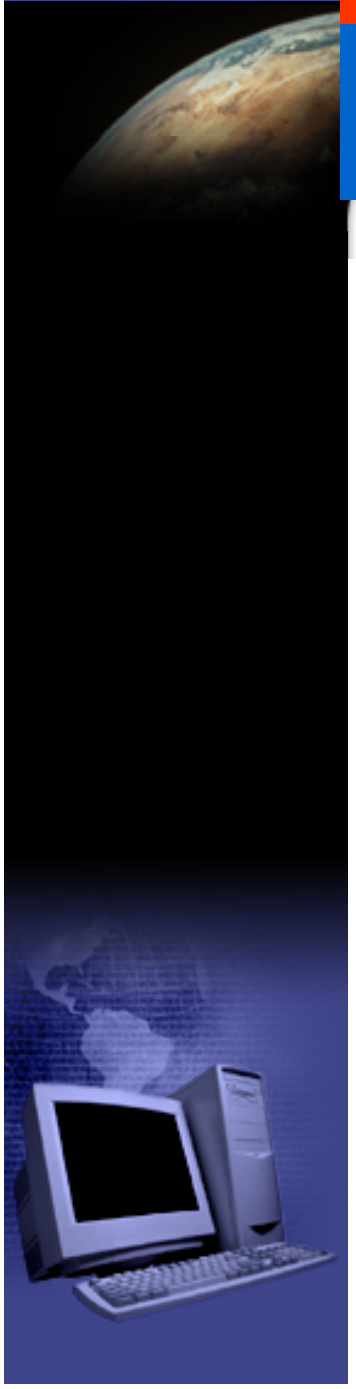
- Hệ thống được xây dựng bởi nhiều lớp.
- Mỗi lớp được xây dựng dựa trên các lớp bên trong
- Lớp trong cùng (lớp 0): phần cứng
- Lớp ngoài cùng (lớp N): giao diện với NSD
- Mỗi lớp là một đối tượng trừu tượng (dữ liệu+thao tác xử lý dữ liệu).
- Mỗi lớp có thể gọi các thủ tục của các lớp bên trong

# 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

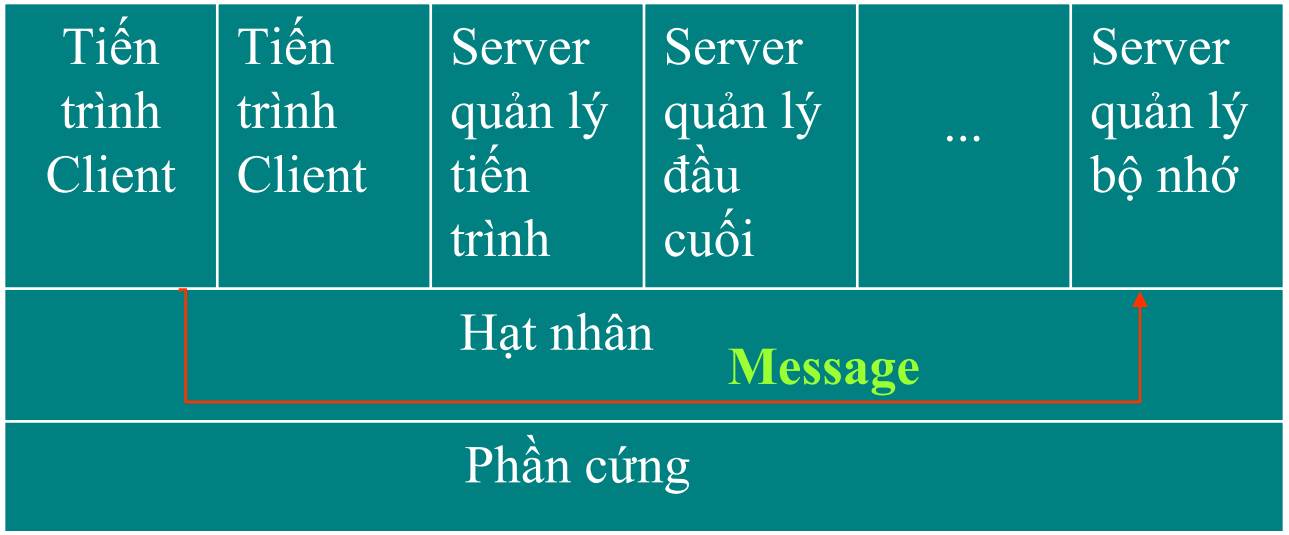


# Máy ảo

# Mô hình Client-Server

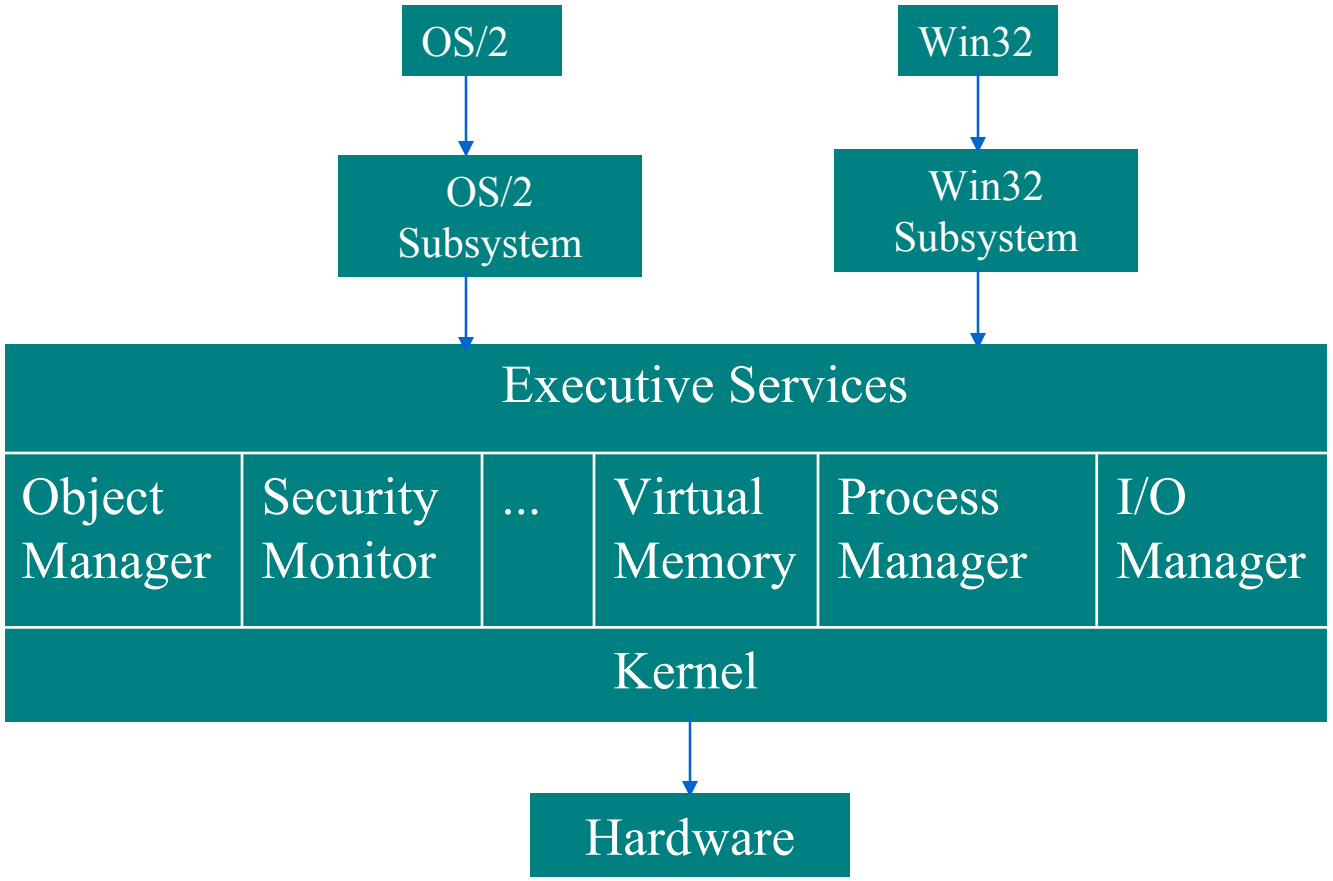
- **HĐH bao gồm nhiều tiến trình đóng vai trò Server với các chức năng chuyên biệt.**
- **Phần hạt nhân HĐH đóng vai trò giao tiếp giữa tiến trình Client và tiến trình Server.**
- **Chỉ có phần hạt nhân cực nhỏ phụ thuộc vào phần cứng.**

# Mô hình Client-Server



# Mô hình Client-Server

## Ví dụ: Cấu trúc Windows NT



## CHƯƠNG 2. TIẾN TRÌNH

# Các vấn đề

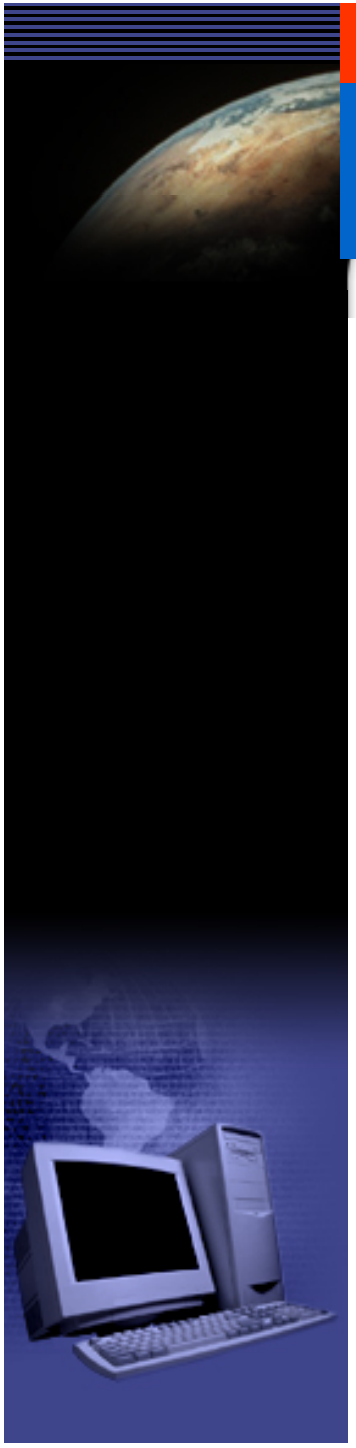
1. Các khái niệm
2. Mô hình trạng thái
3. Thao tác trên tiến trình
4. Điều phối tiến trình
5. Đồng bộ hoá tiến trình

# 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**



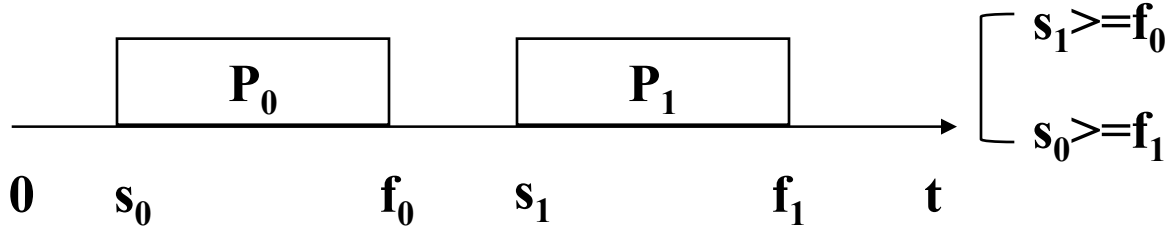
# Các khái niệm

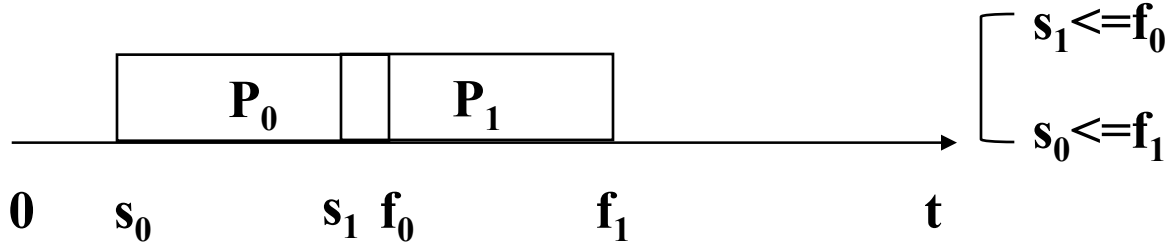
- **Có 2 loại tiến trình:**

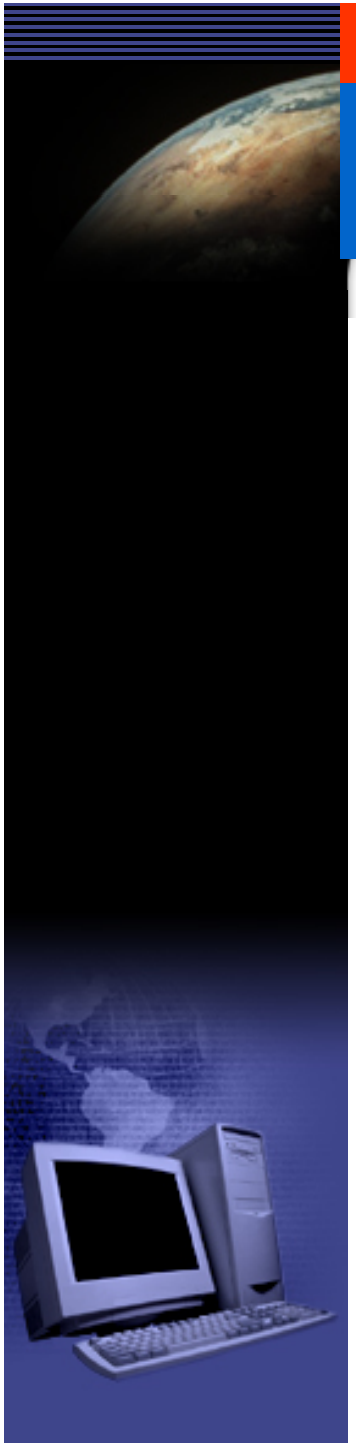
**- Tiến trình kế tiếp: thời điểm bắt đầu của tiến trình này nằm sau thời điểm kết thúc của tiến trình kia**

**- 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

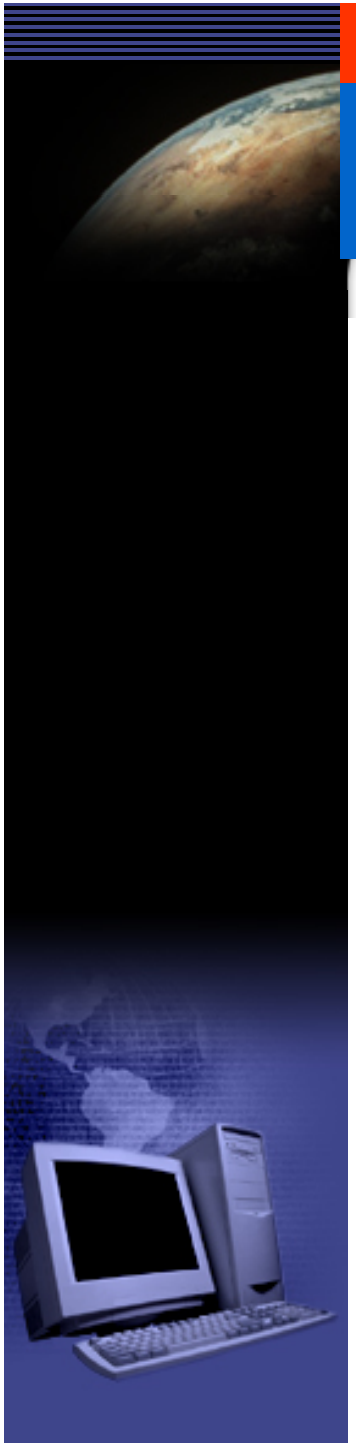




# Các khái niệm

- **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:**

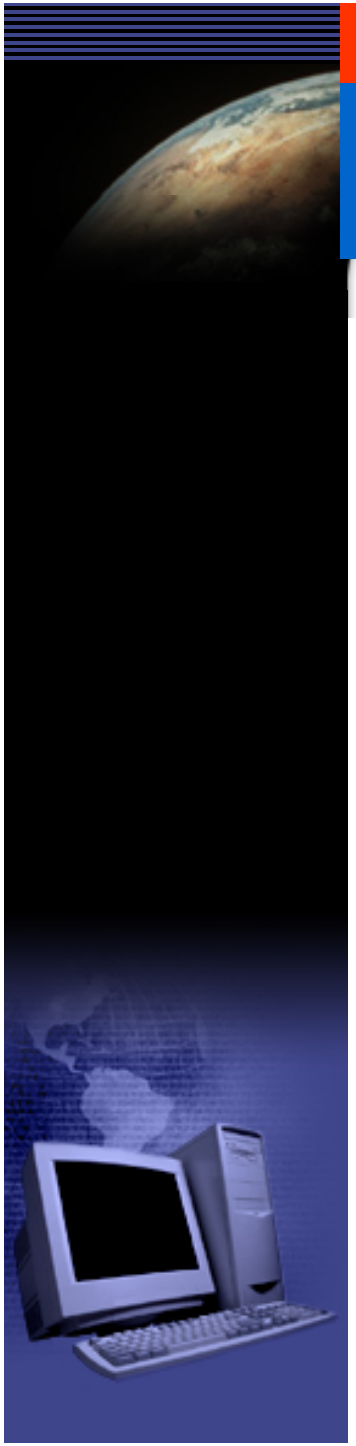
**- Trạng thái CPU: nội dung các thanh ghi (IP). Lưu trữ nội dung thanh ghi khi xảy ra ngắt.**

**- 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**

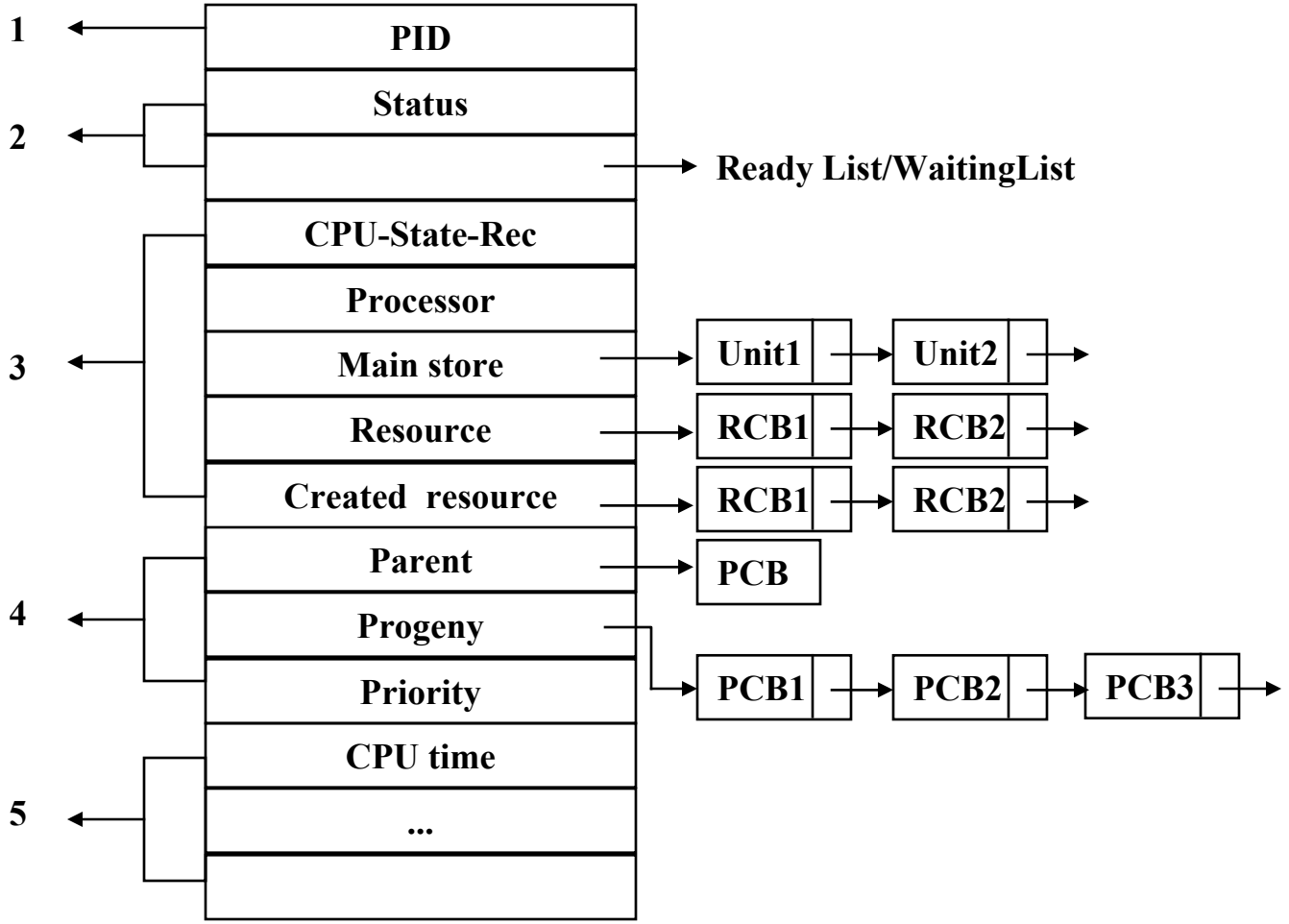
- **Thông tin thống kê về hoạt động của tiến trình:**

**- 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

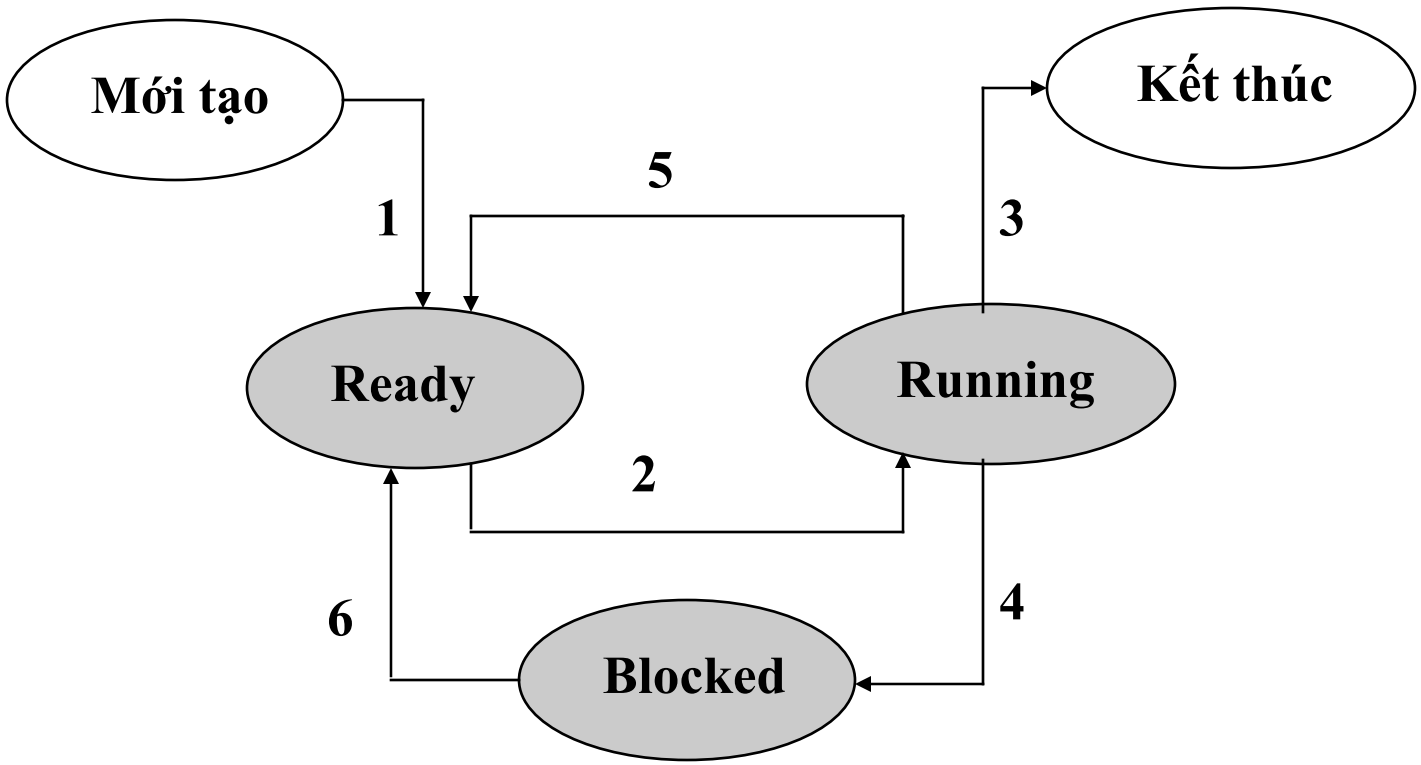
# Mô hình trạng thái

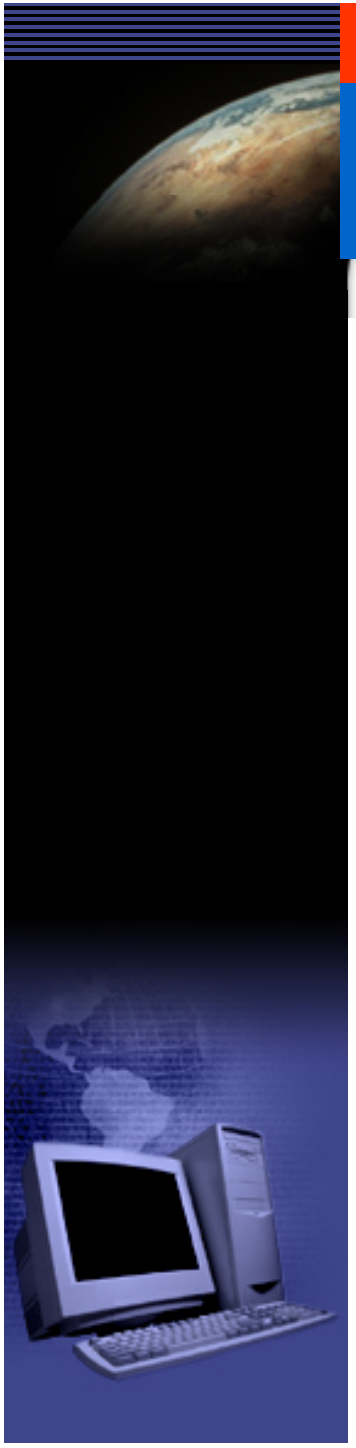
## ❖ Các trạng thái của tiến trình

- **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

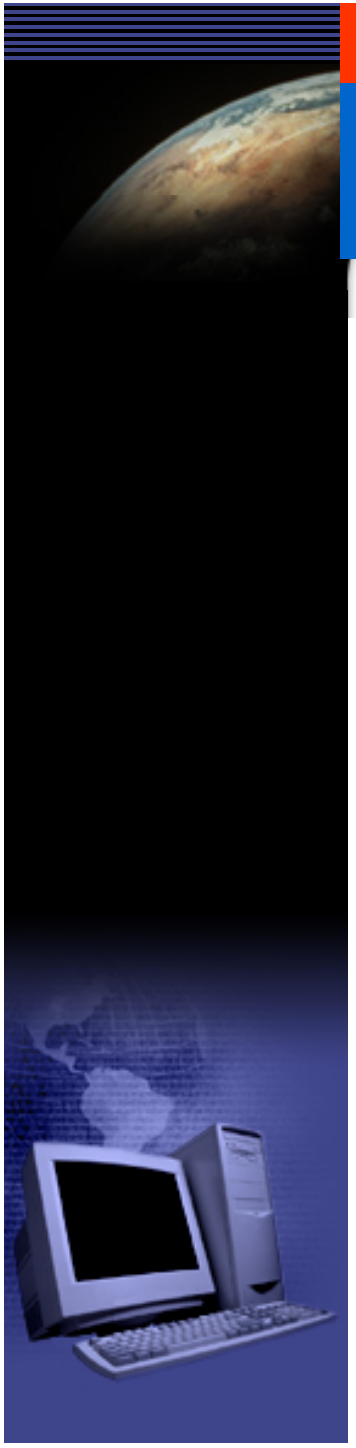
❖ **Sơ đồ chuyển trạng thái của tiến trình**

(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.

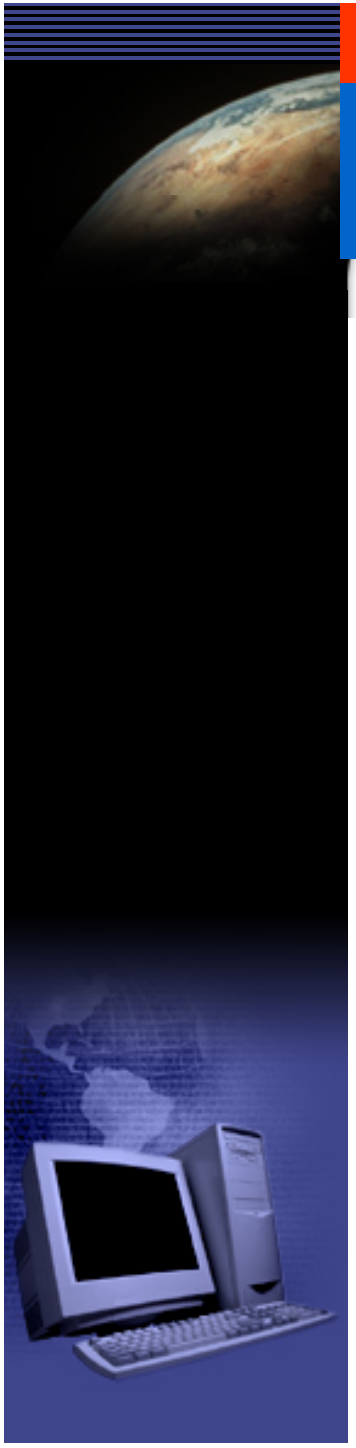


# Mô hình trạng thái

❖ Sơ đồ chuyển trạng thái của tiến trình

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

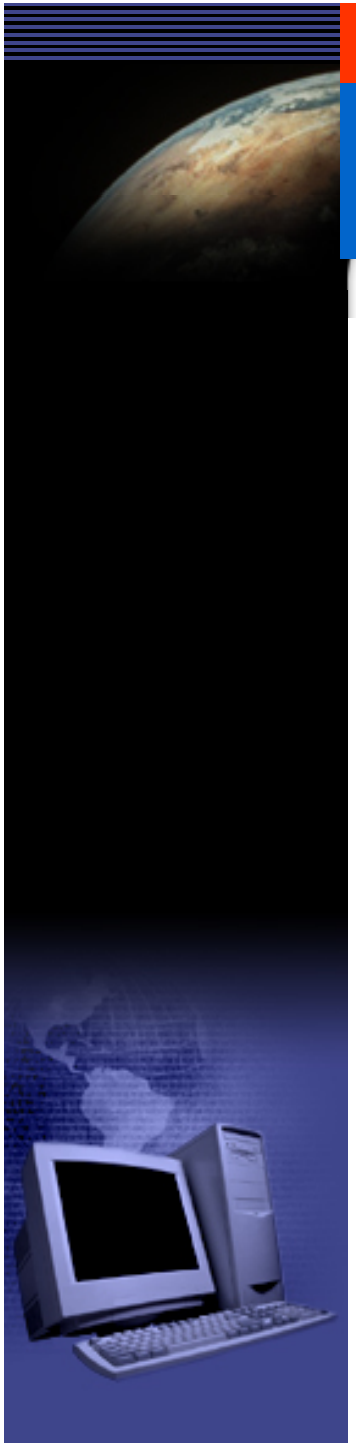
(6) Tài nguyên mà tiến trình yêu cầu đã sẵn sàng để cấp phát, hay sự kiện, thao tác nhập/xuất tiến trình đang đợi hoàn tất.



# 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:

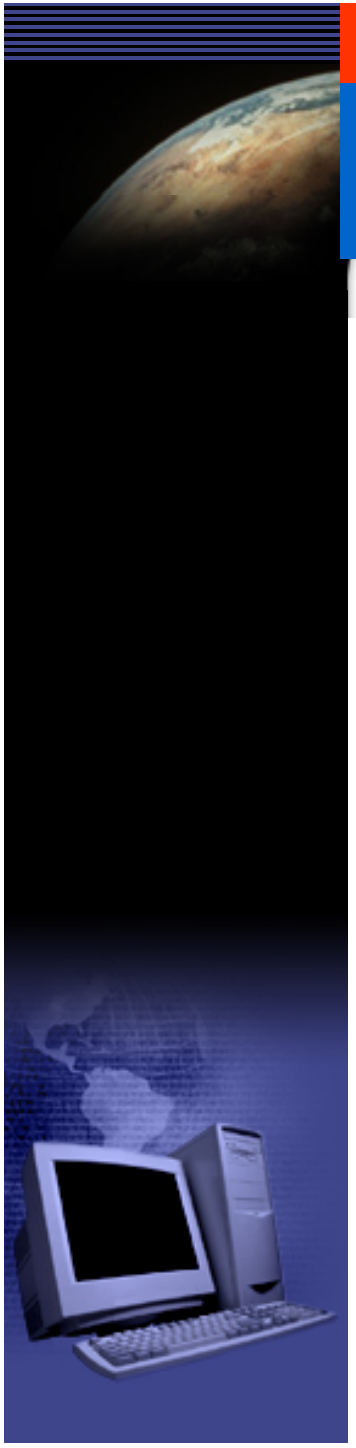
✓ Định danh cho tiến trình (PID)

✓ 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

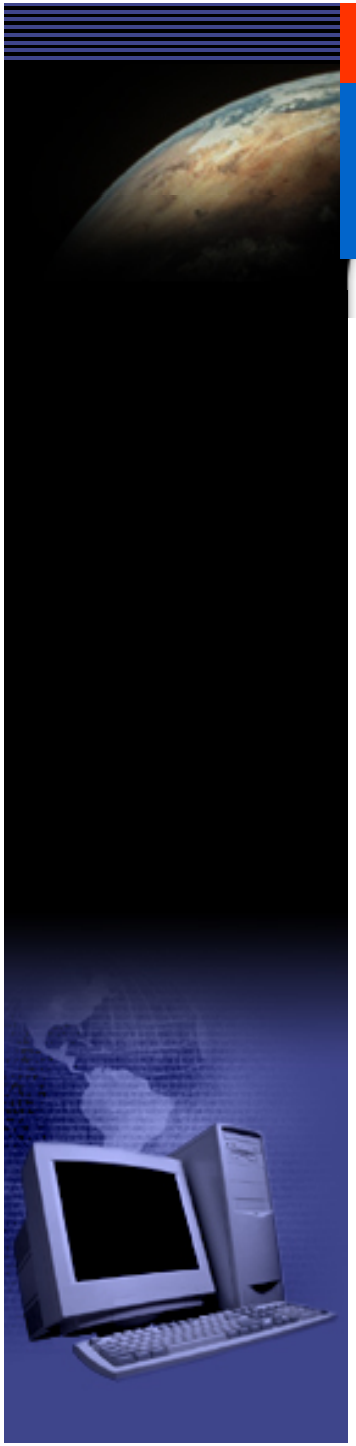


# Thao tác trên tiến trình

## ❖ Kết thúc tiến trình

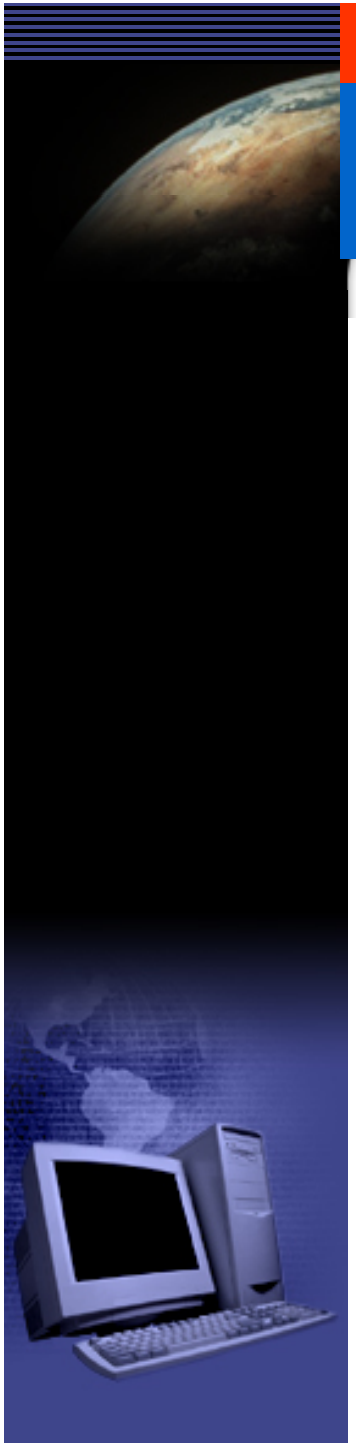
**Khi tiến trình kết thúc, HĐH thực hiện:**

- **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

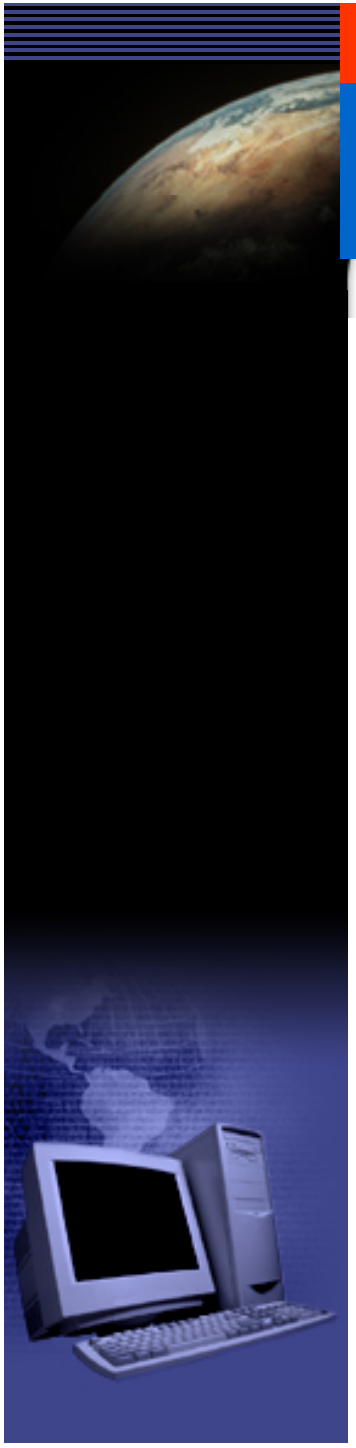
- Mục tiêu điều phối
- Tiêu chuẩn điều phối
- Điều phối không độc quyền, điều phối độc quyền
- Đồng hồ ngắt giờ
- Độ ưu tiên của tiến trình
- Tổ chức điều phối
- Các chiến lược điều phối



# Điều phối tiến trình

## ❖ Mục tiêu điều phối

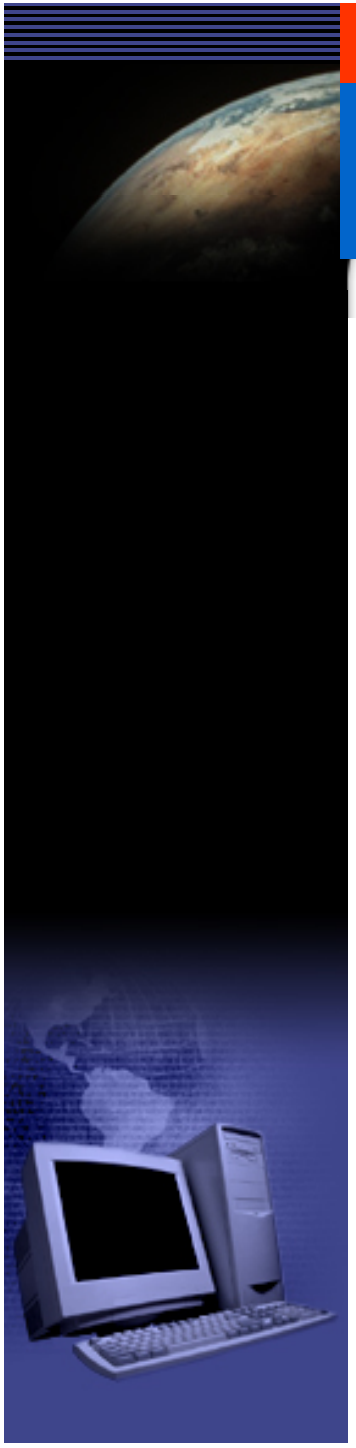
- Sự công bằng giữa các tiến trình
- Tính hiệu quả (tận dụng 100% thời gian sử dụng CPU)
- Cực tiểu hoá thời gian lưu lại trong hệ thống
- Thời gian đáp ứng hợp lý (cực tiểu hoá thời gian hồi đáp cho các tương tác của NSD)
- Thông lượng tối đa (cực đại hoá số công việc được xử lý trong một thời gian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



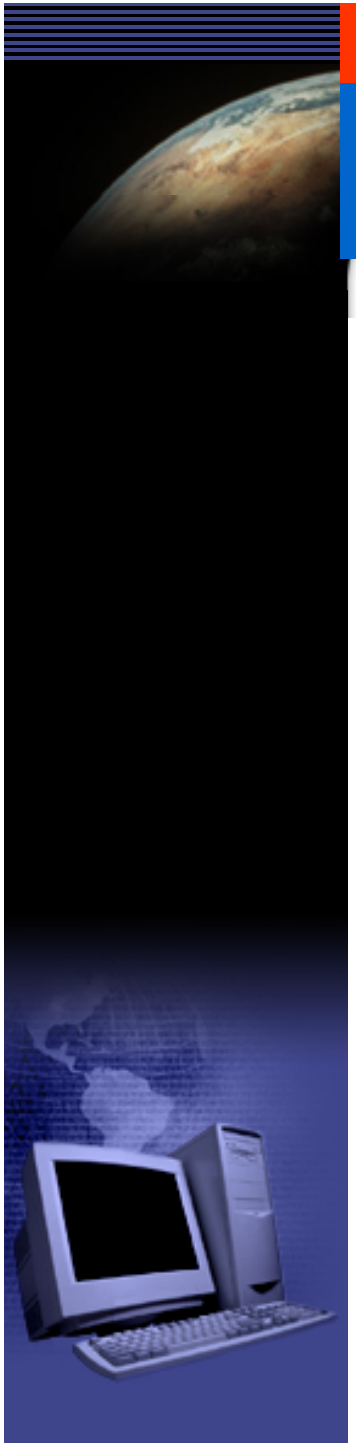
# Điều phối tiến trình

## ❖ 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



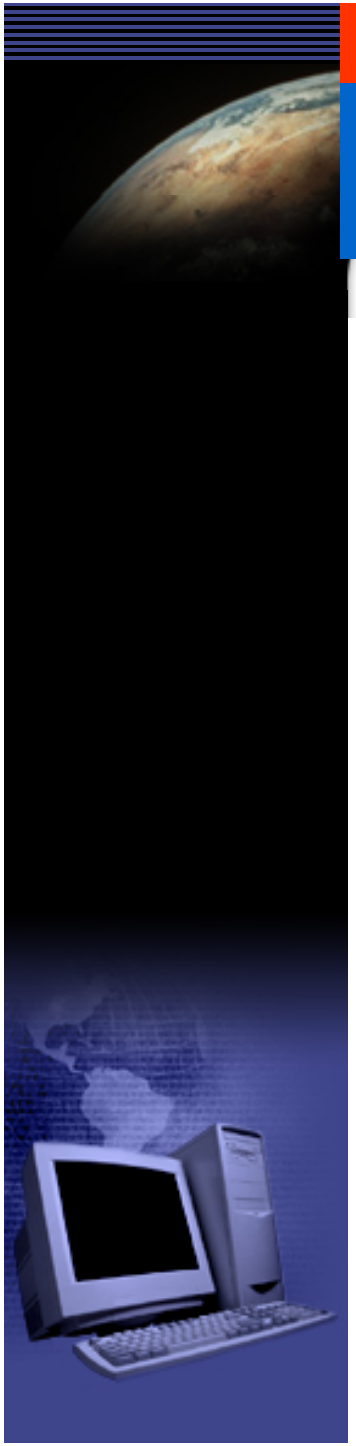
# Điều phối tiến trình

## ❖ 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



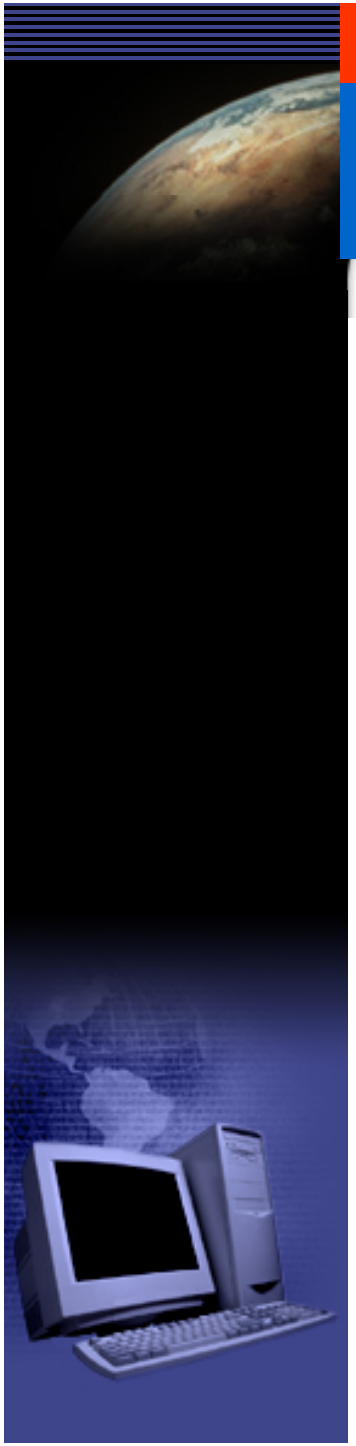
# Điều phối tiến trình

❖ **Điều phối không độc quyền**

**+ Tiến trình chuyển từ trạng thái blocked sang Ready**

**+ Tiến trình kết thúc**

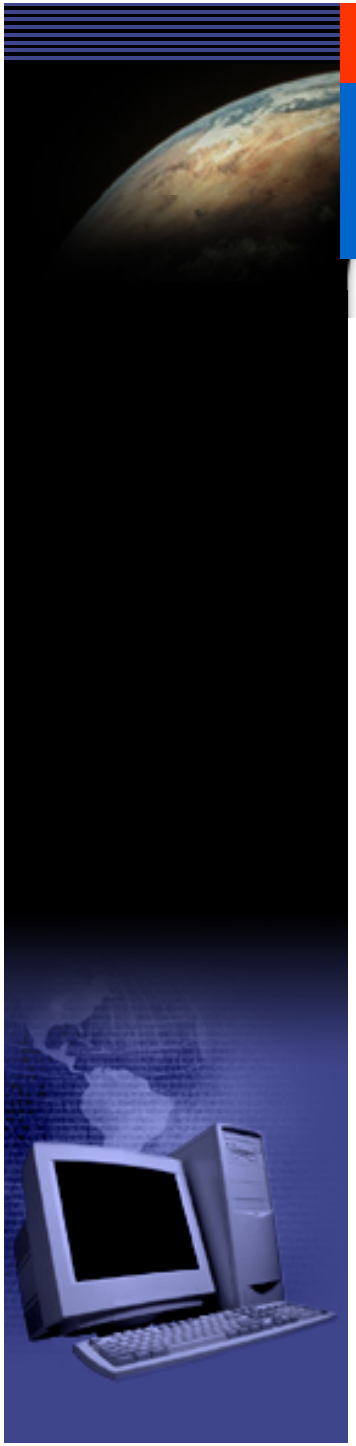
- **Ngăn cản được tình trạng các tiến trình độc chiếm CPU, nhưng việc tam dừng một tiến trình dẫn đến các mâu thuẫn trong truy xuất. Đòi hỏi phương pháp đồng bộ hoá thích hợp**



# 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.



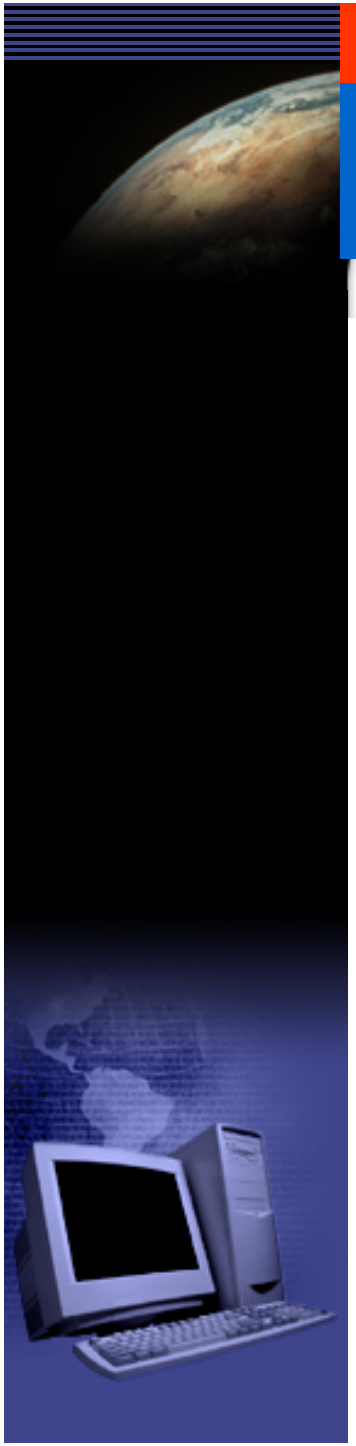
# Điều phối tiến trình

## ❖ 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



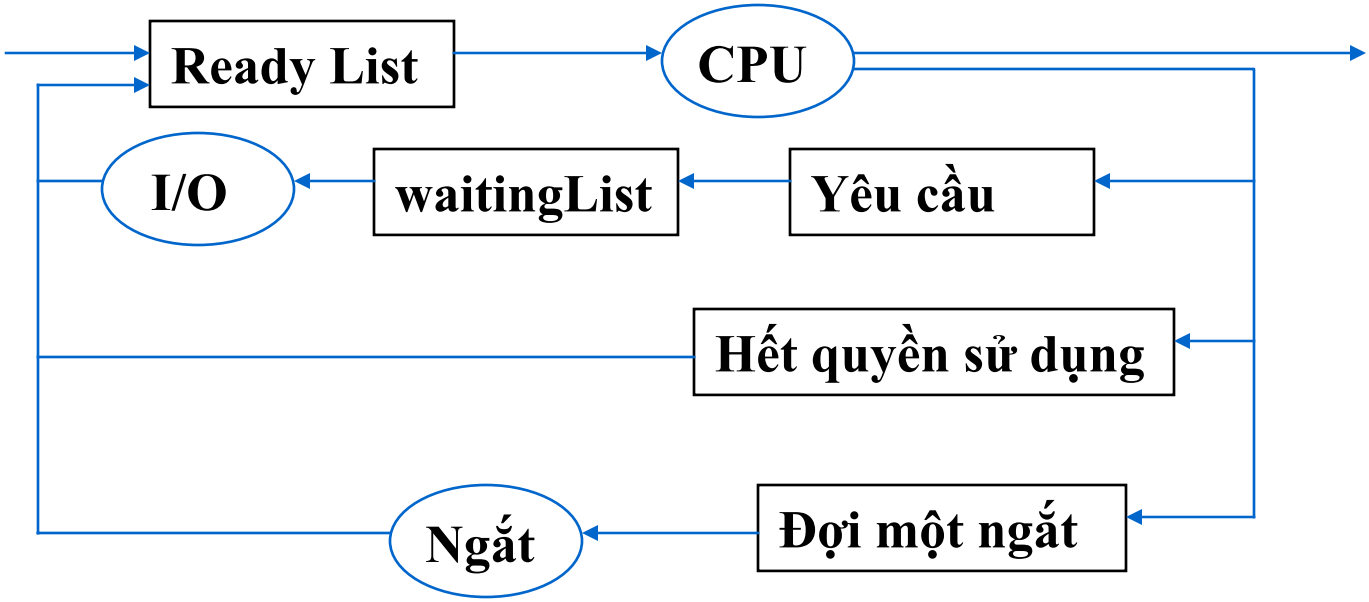
# Điều phối tiến trình

## ❖ Tổ chức điều phối

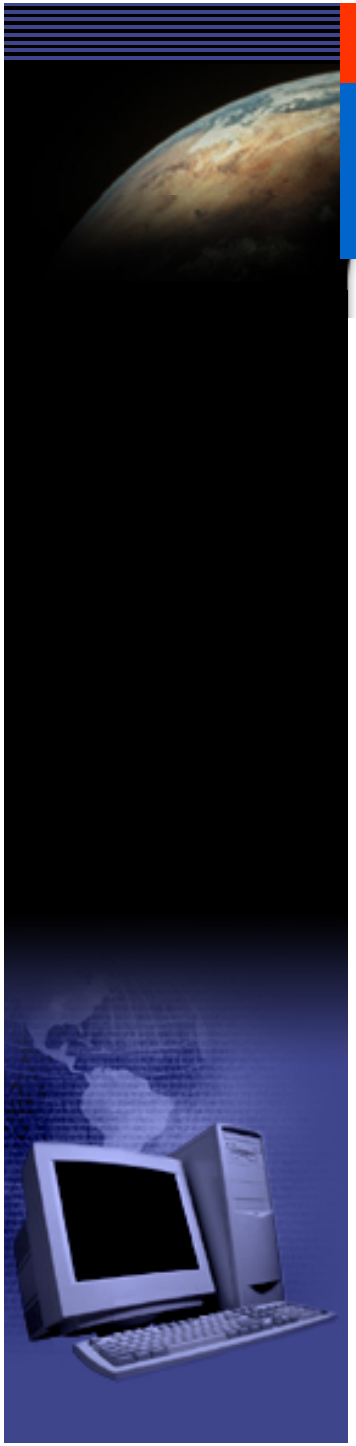
- Danh sách sẵn sàng (Ready List)
- Danh sách chờ đợi (Waiting List)
- Các danh sách chờ đợi riêng cho từng tài nguyên (thiết bị ngoại vi)

# Điều phối tiến trình

## ❖ Tổ chức điều phối




**Sơ đồ chuyển đổi giữa các danh sách điều phối**



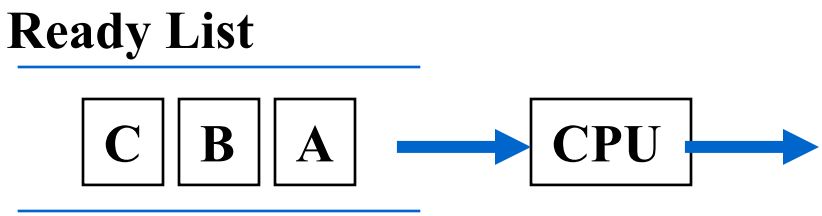
# Điều phối tiến trình

## ❖ 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

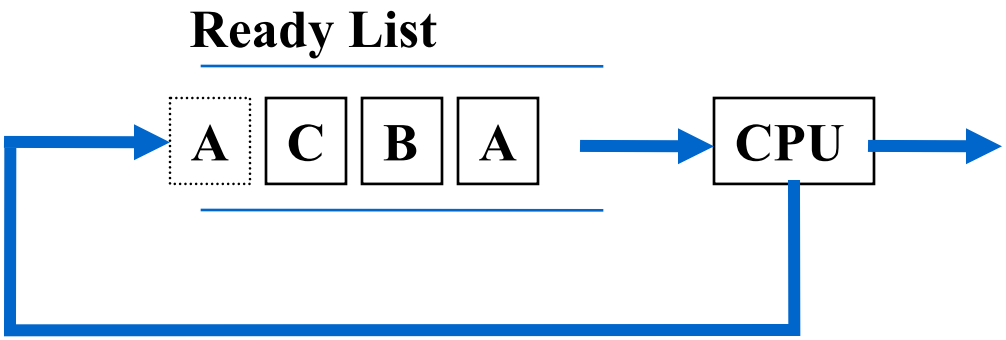
## ❖ Chiến lược điều phối



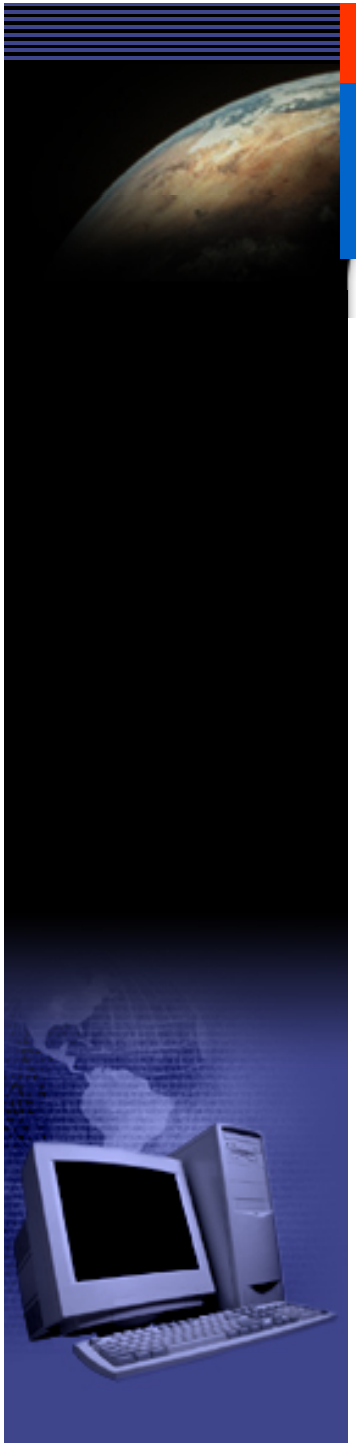

**Điều phối FIFO**

# Điều phối tiến trình

## ❖ Chiến lược điều phối



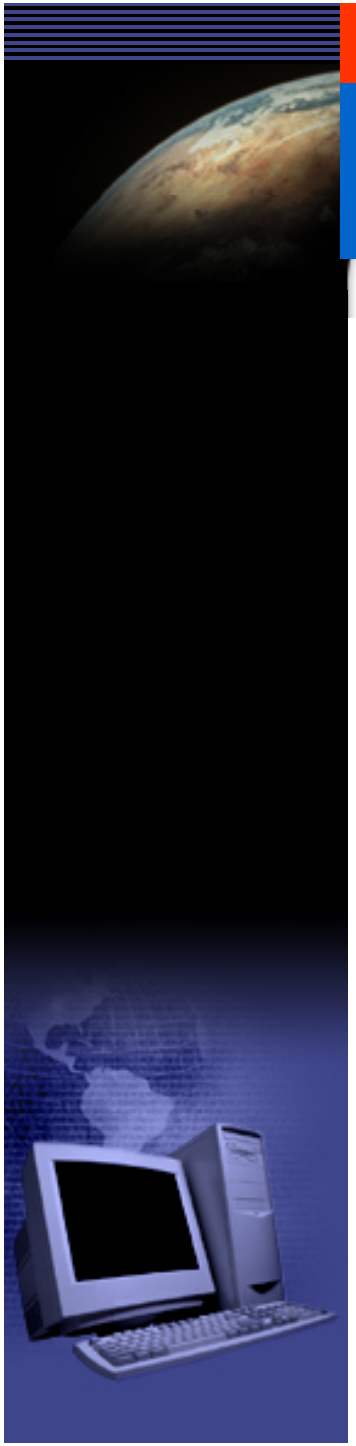

**Điều phối Round Robin**



# 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

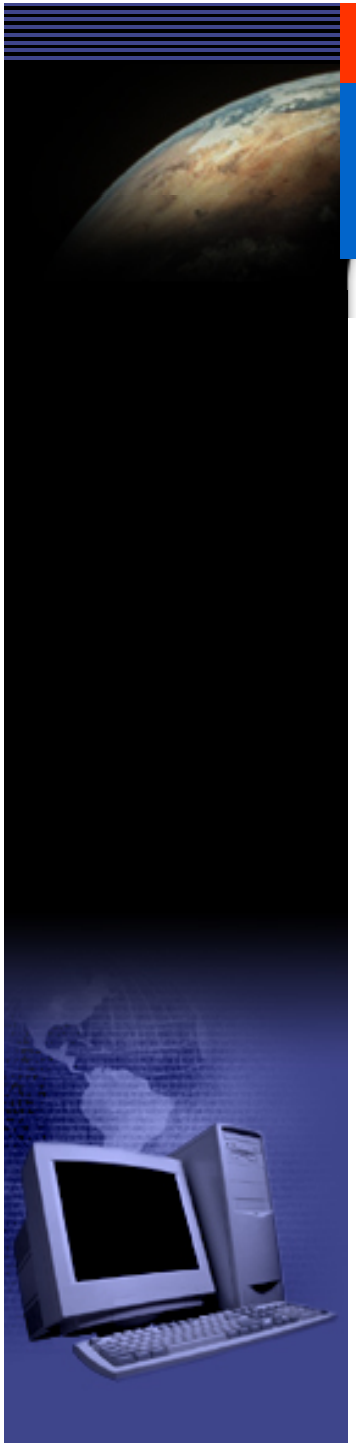
## ❖ Miền găng (Critical Section)

- **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**

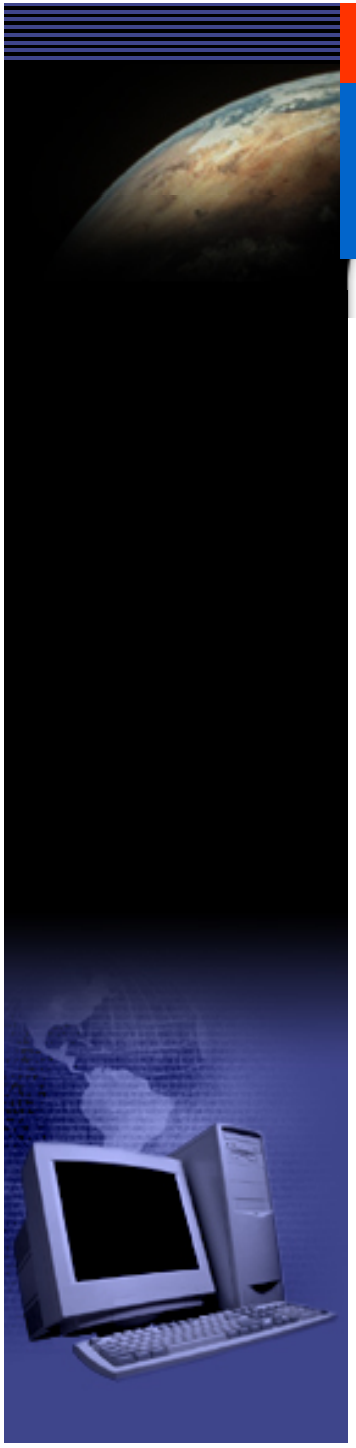


# Đồng bộ hoá tiến trình

## ❖ 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

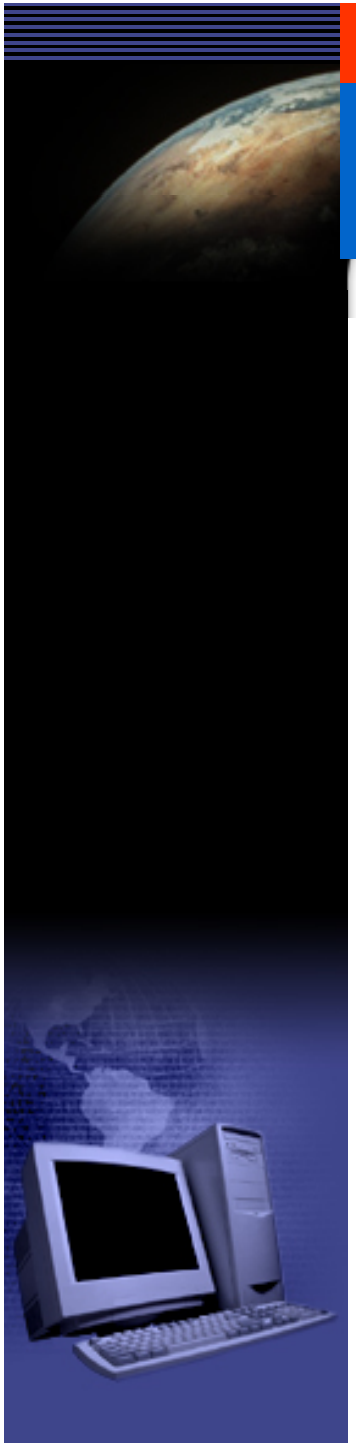
## ❖ Giải pháp

### ➢ 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



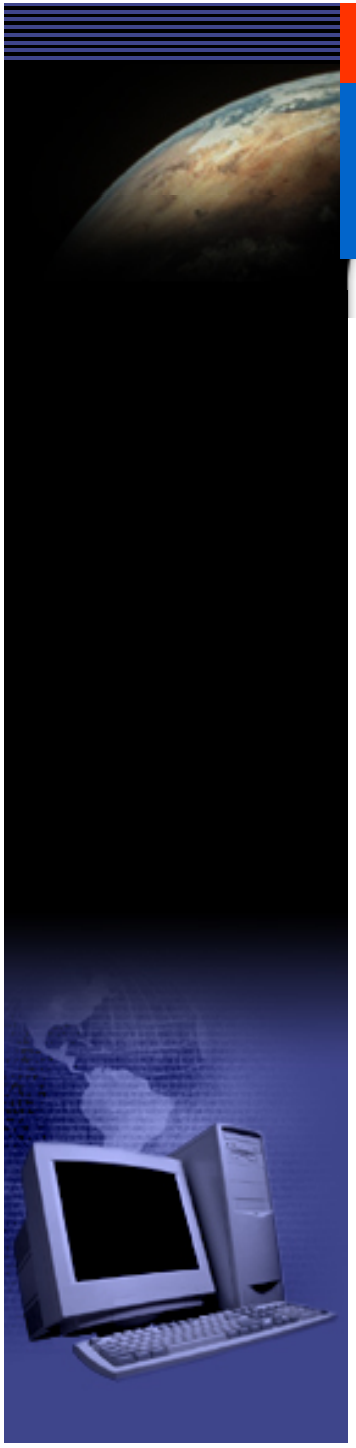
# Đồng bộ hoá tiến trình

## ❖ Giải pháp

### ➢ Sử dụng biến khoá

**- Giải thuật sử dụng biến khoá để đồng bộ**

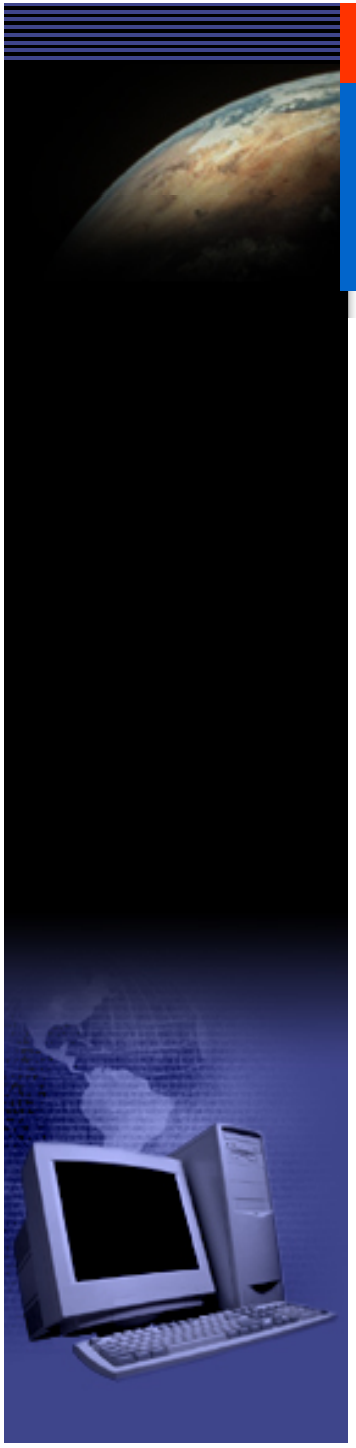
```
while (1) {
    while (lock==1);// wait
    lock=1;
    critical_section();
    lock=0;
    Noncritical_section();
}
```



# Đồng bộ hoá tiến trình

- Giải thuật sử dụng biến khoá để đồng bộ

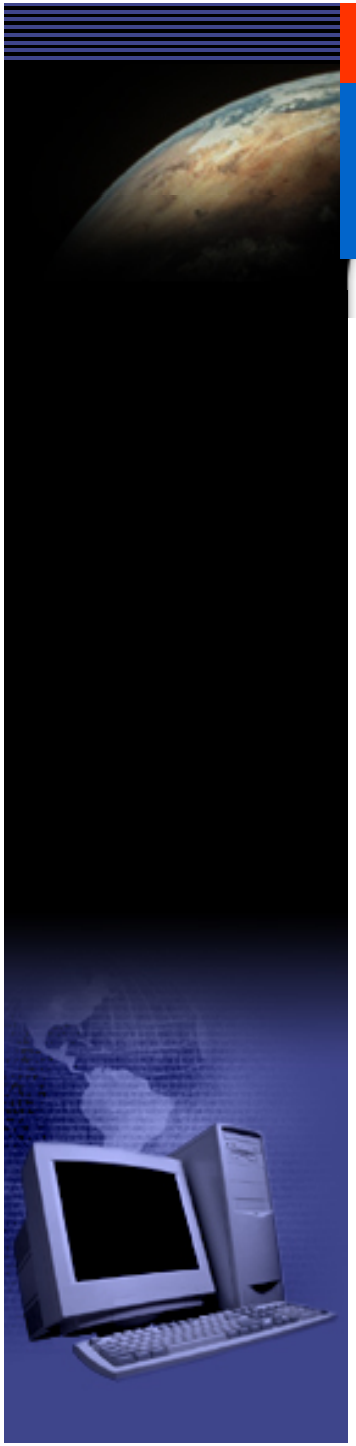
```
while (1) {
    while (lock==1);// wait
    lock=1;
    critical_section();
    lock=0;
    non_critical_section();
    }
```



# 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

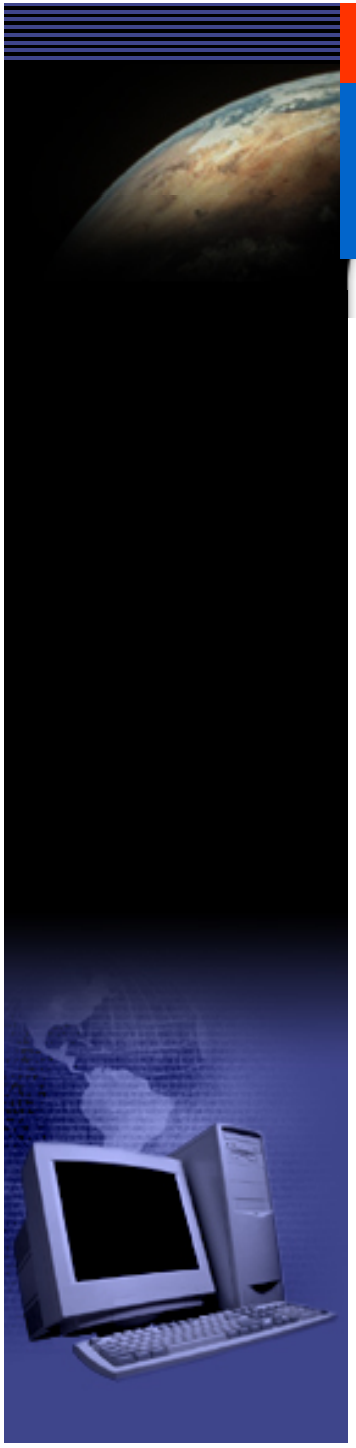
### ➢Kiểm tra luân phiên

- 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

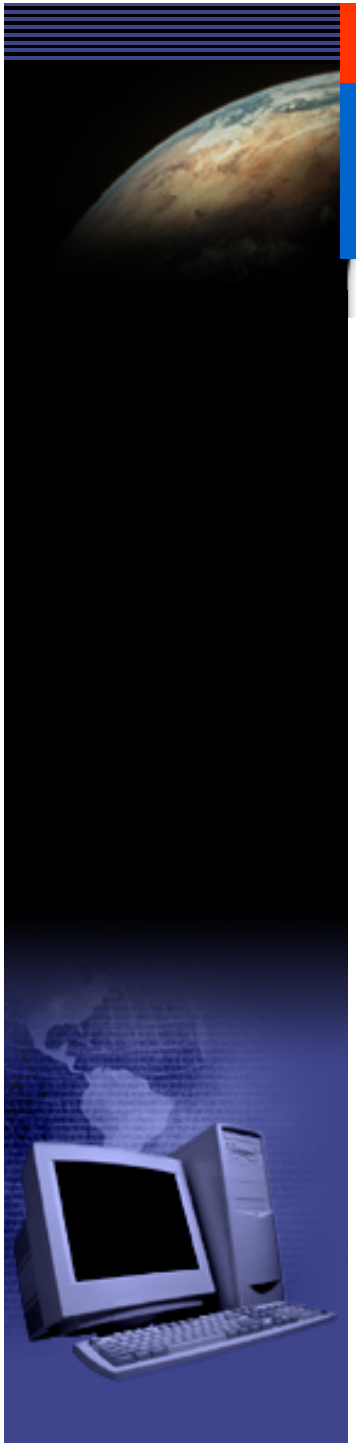
```
#define N  2  // Chỉ 2 tiến trình
int turn=0, interested[N]={0,0};
void enter_region(int process) // Vào ĐG
{  int other=1-process;//other là tiến trình đối của process
   interested[prcess]=1;
   turn=process;
   while ((turn==process)&&interested[other]==1);//chờ
}
```

# 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**

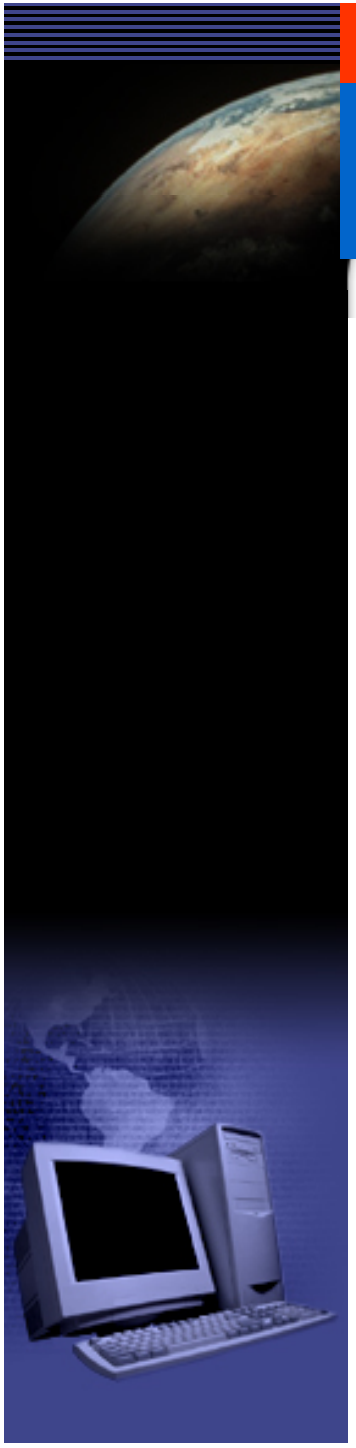
# Đồng bộ hoá tiến trình

## ❖ 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];**

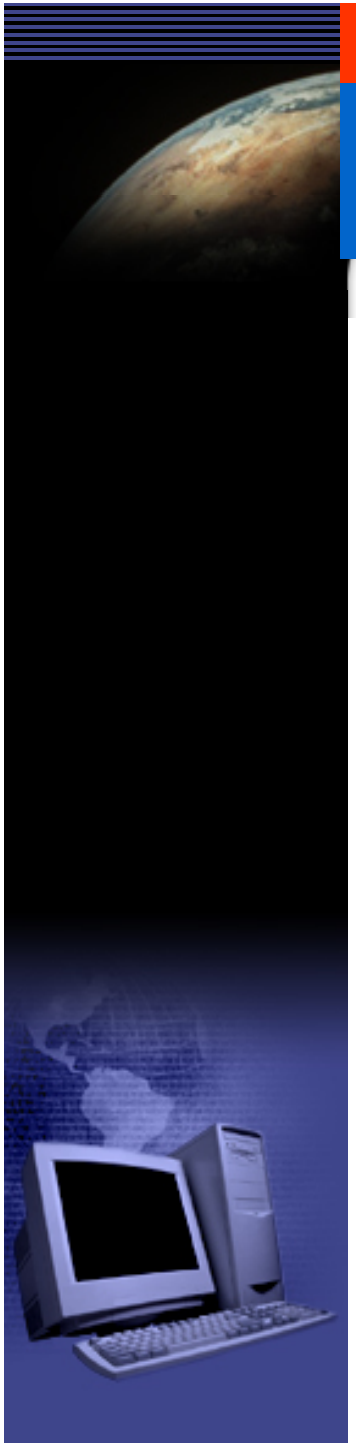
**// max[p,r] nhu cầu tối đa của tiến trình p về tài nguyên r**

**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**

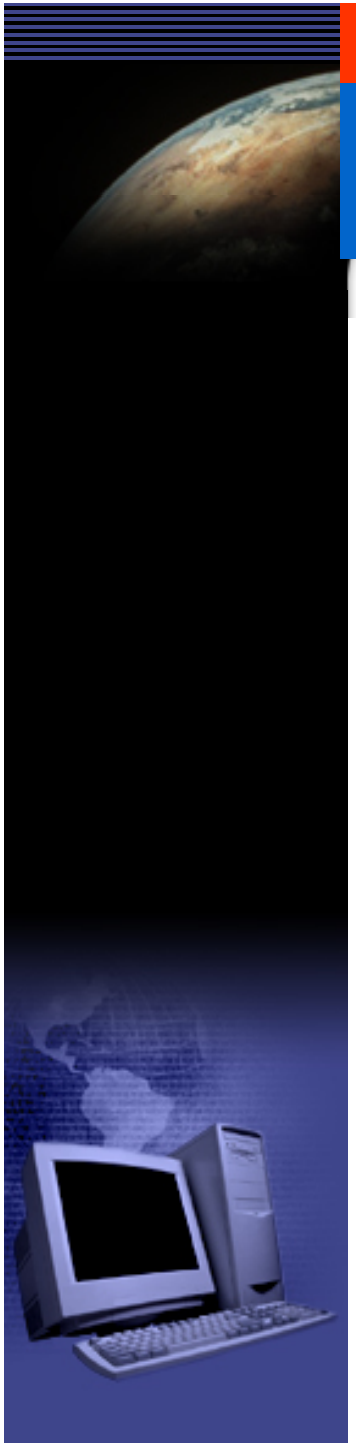
   **nếu không có i như thế, đến bước 3**

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.**

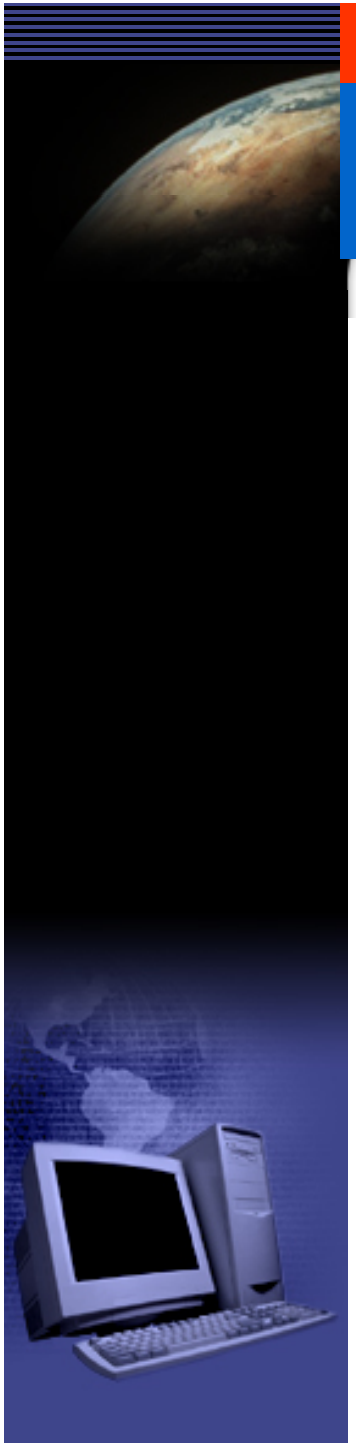
# Xác định trạng thái an toàn

❖ **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 | | | |

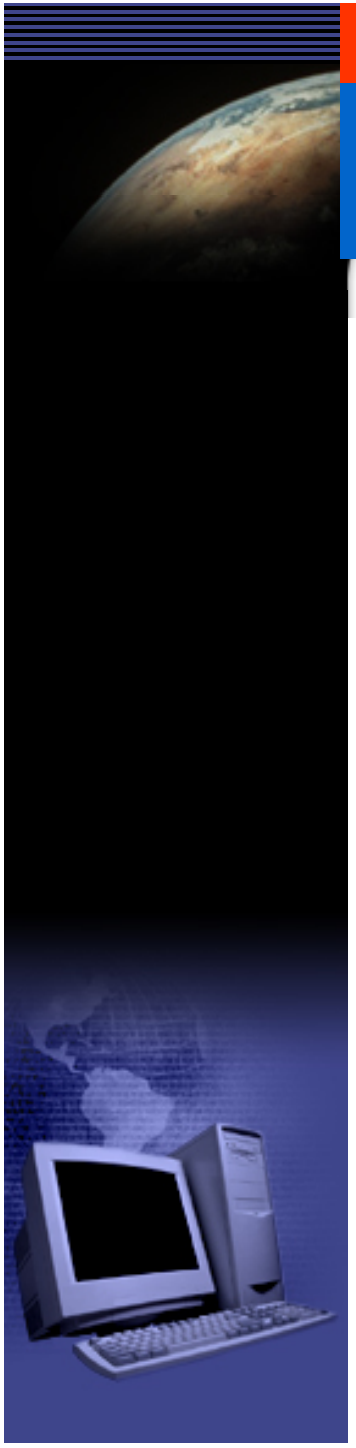
**Tiến trình P2 yêu cầu 4 R1, 1 R3. Hãy cho biết yêu cầu này có thể đáp ứng mà đảm bảo không xảy ra tình trạng tắt nghẽn hay không?**

## CHƯƠNG 4. QUẢN LÝ BỘ NHỚ



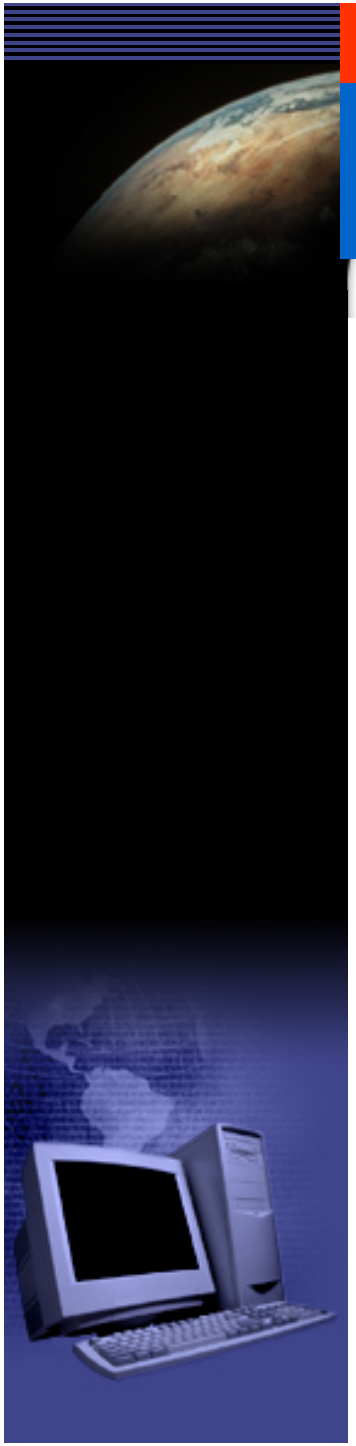
# Các vấn đề

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

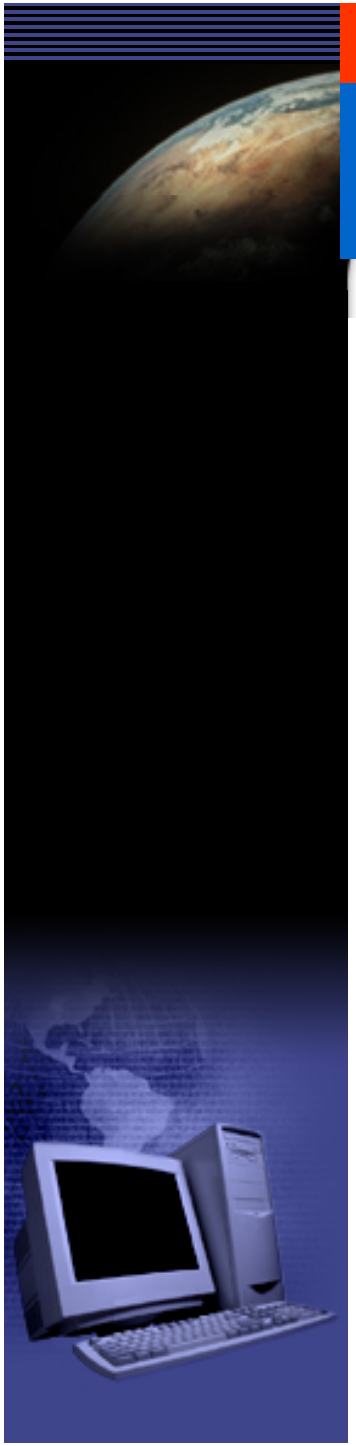
- **Hệ điều hành thực hiện:**
  - **Sự tương ứng giữa địa chỉ logic và địa chỉ vật lý**
  - **Quản lý bộ nhớ vật lý**
  - **Chia sẻ thông tin**
  - **Bảo vệ**

# 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.**

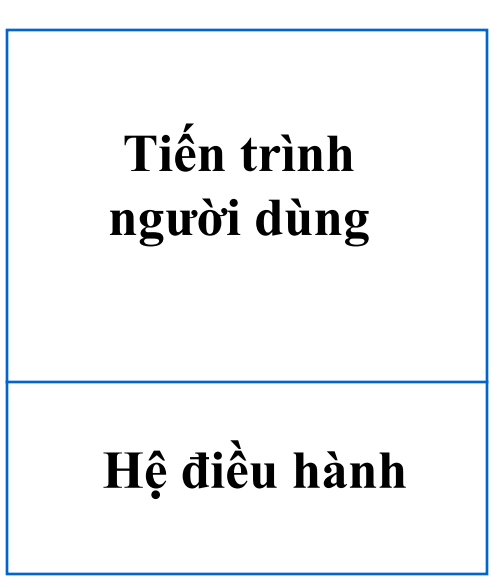


# Cấp phát liên tục

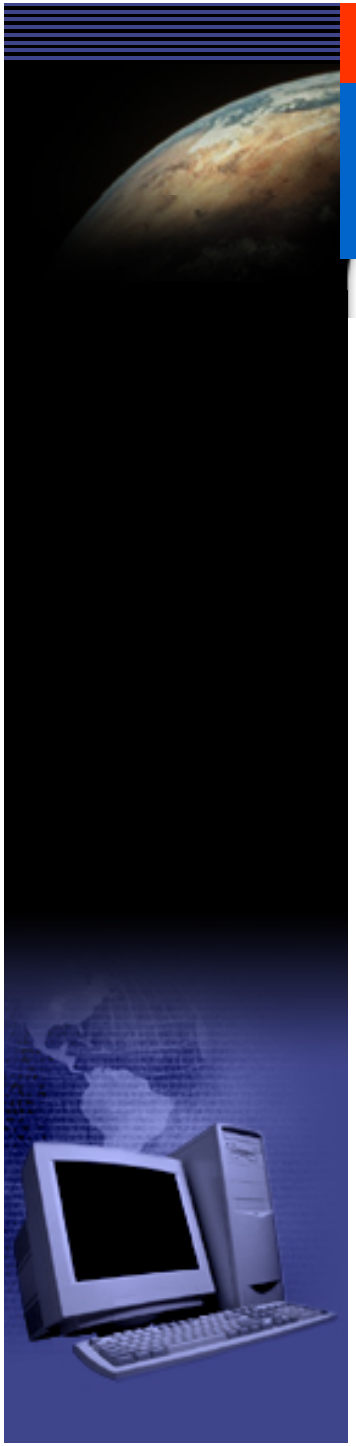
- Các hệ đơn chương
- Các hệ thống đa chương với phân vùng cố định
- Các hệ thống đa chương với phân vùng động
- Các hệ thống đa chương với kỹ thuật “Swapping”

# Cấp phát liên tục

- Các hệ đơn chương




Tổ chức bộ nhớ trong hệ thống đơn chương



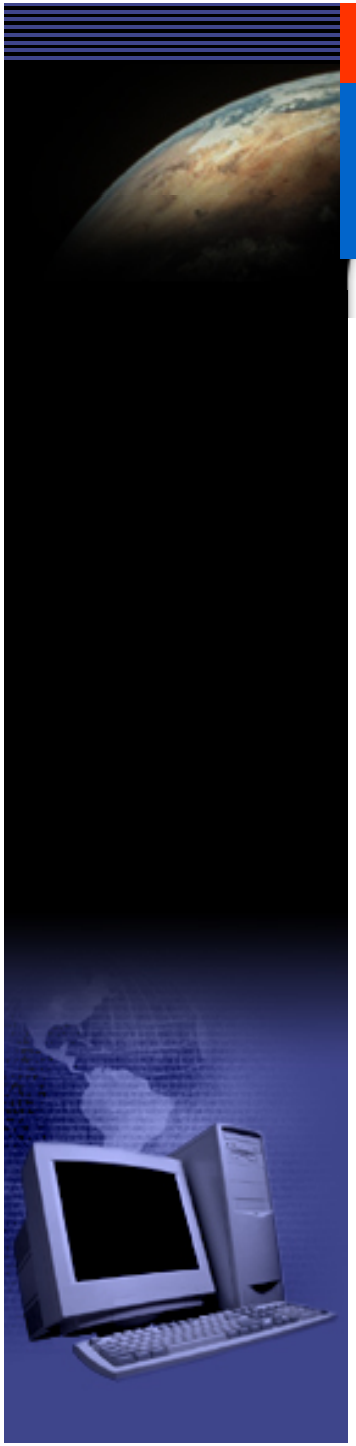
# Cấp phát liên tục

❖ **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.**

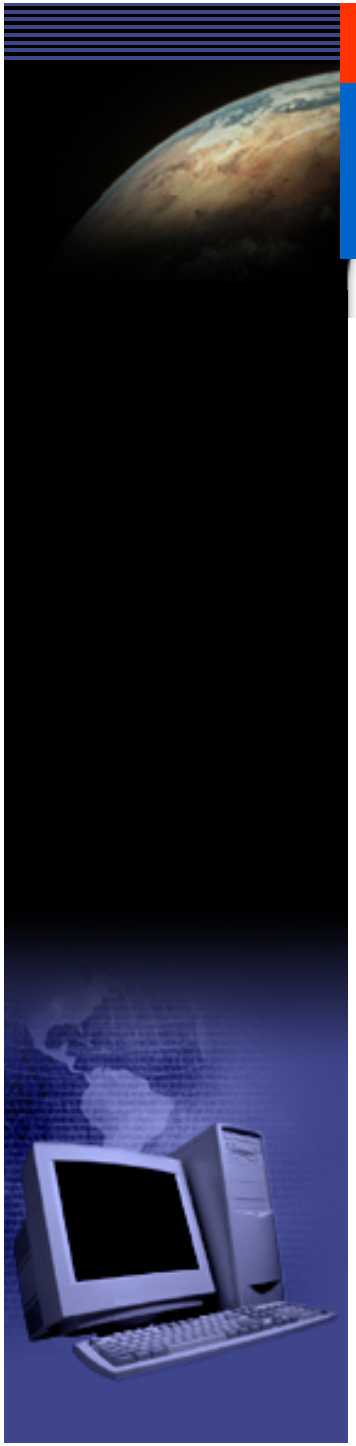
  **+ Ngược lại : một ngắt sẽ được phát sinh báo sự truy xuất bất hợp lý.**
- **Tại một thời điểm chỉ có một chương trình được xử lý.**



# 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.**

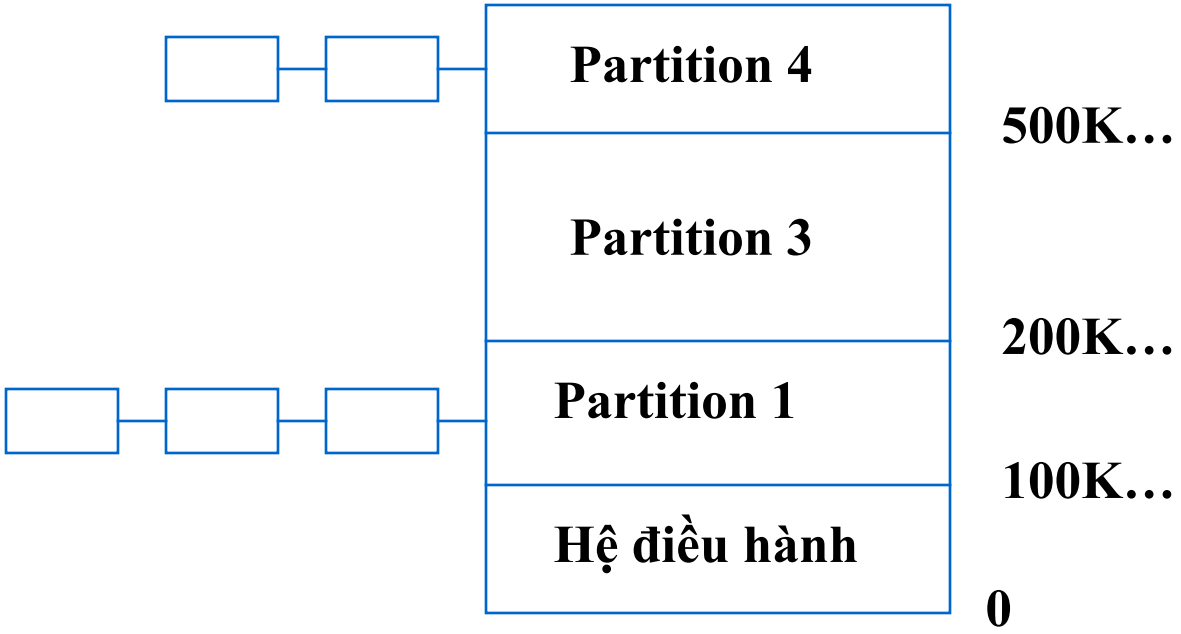


# Cấp phát liên tục

- 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

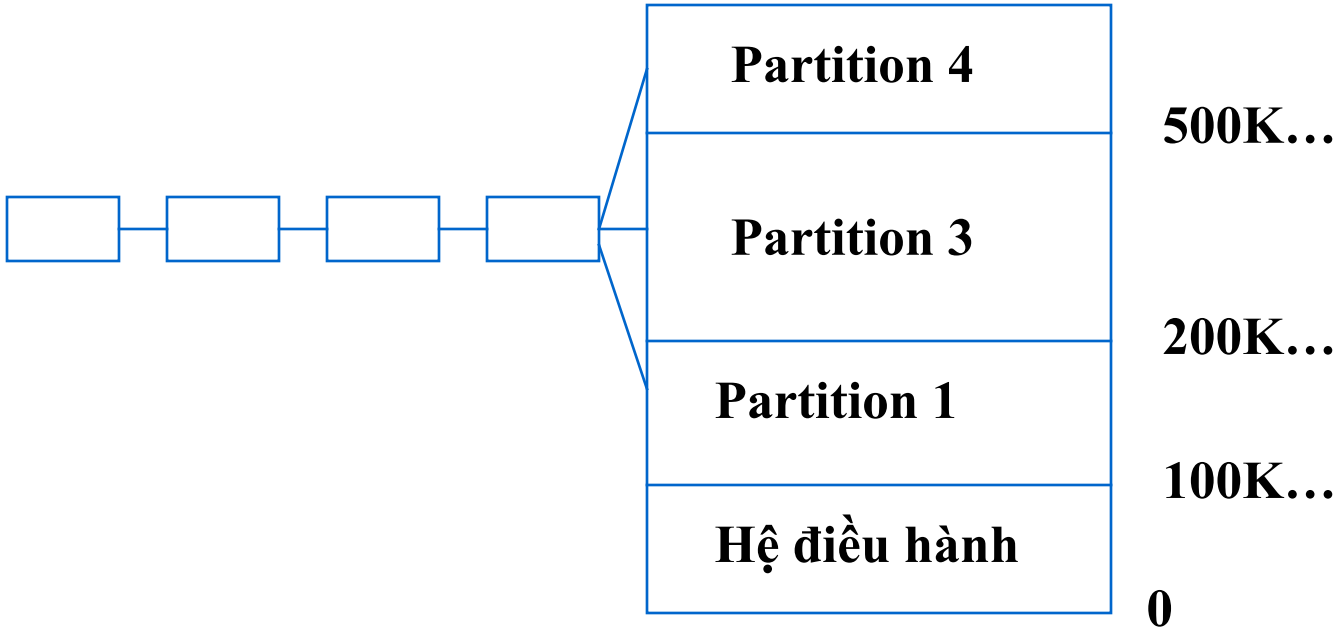
- Các hệ thống đa chương với phân vùng cố định



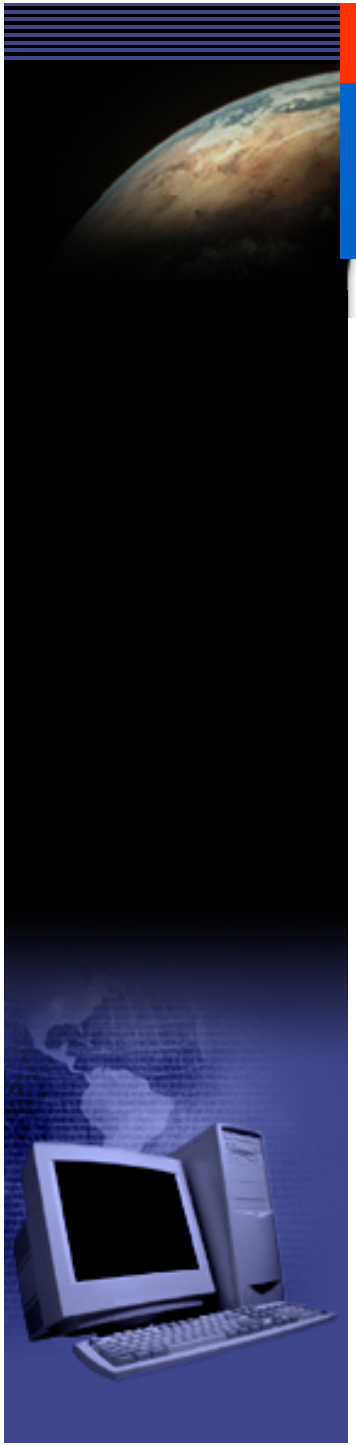

Phân vùng cố định nhiều hàng đợi

# Cấp phát liên tục

- Các hệ thống đa chương với phân vùng cố định

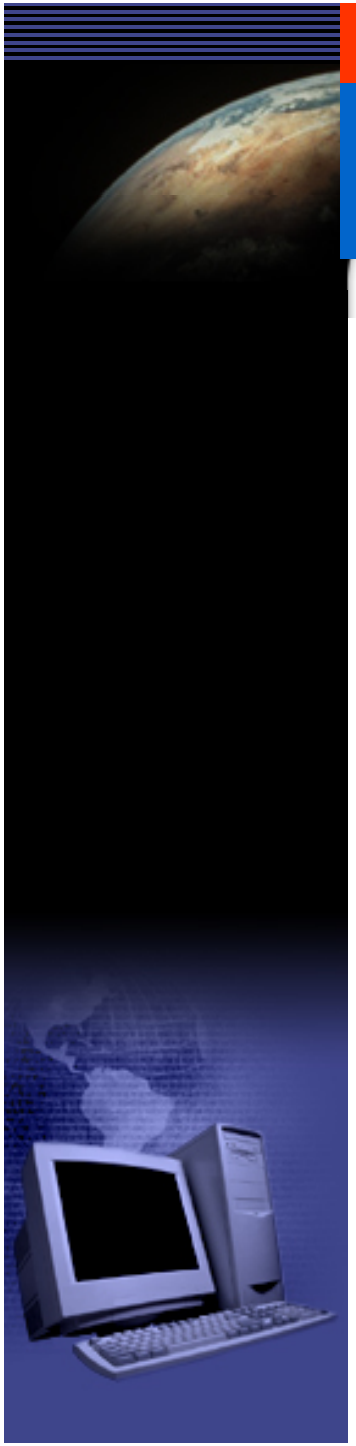



Phân vùng cố định một hàng đợi



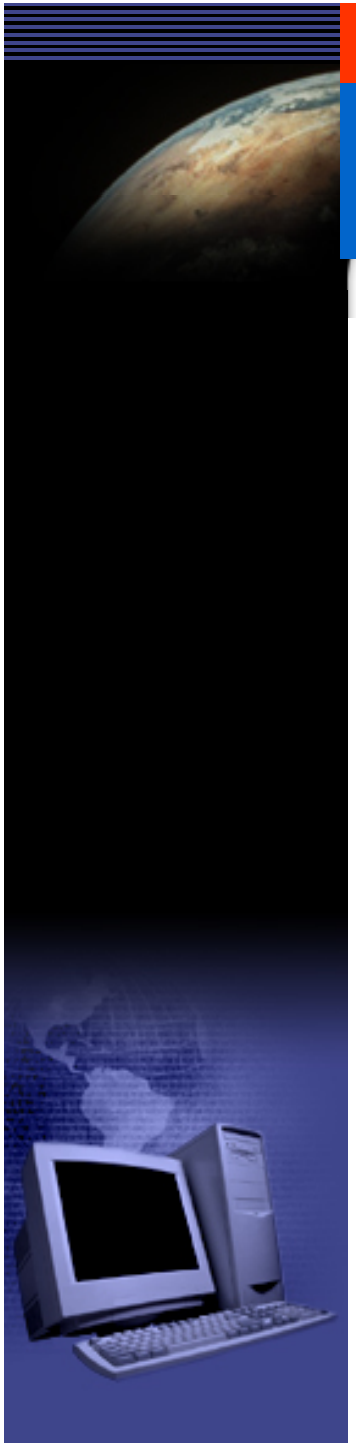
# Cấp phát liên tục

- 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 ⇒ phân mảnh nội vi
  - Mức độ đa chương bị giới hạn bởi số lượng phân vùng

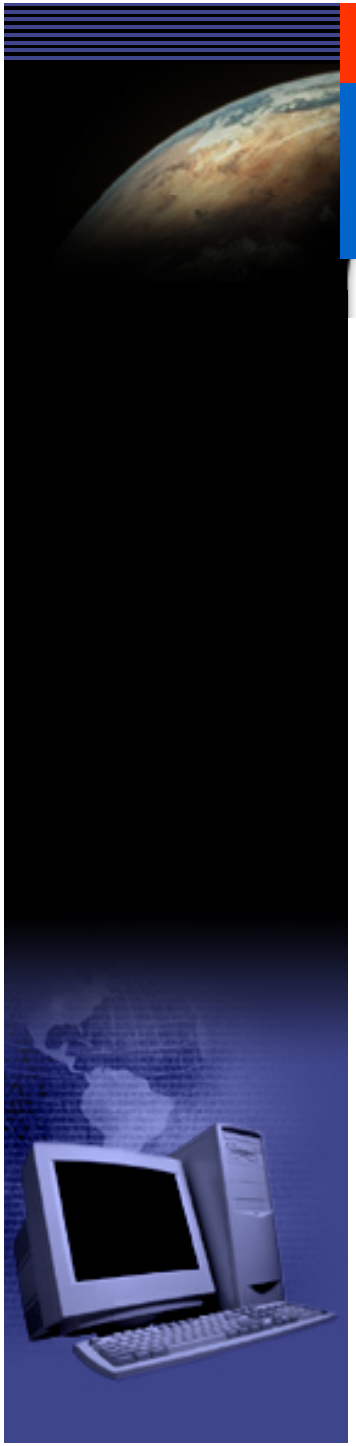


# Cấp phát liên tục

- 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

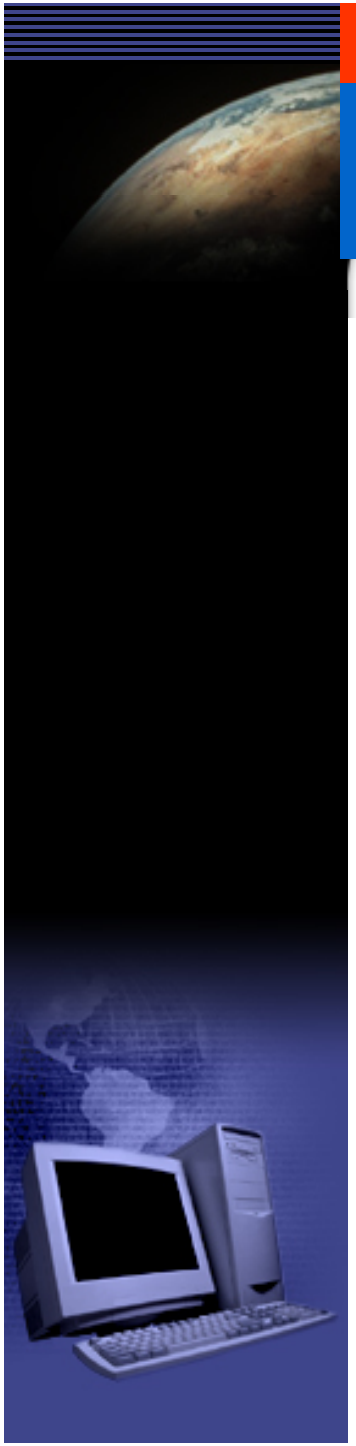
Ví dụ: giả sử chương trình truy xuất đến địa chỉ 100 (địa chỉ tương đối), ct được nạp vào phân vùng 1 địa chỉ bắt đầu 100k, thì địa chỉ truy xuất là (100k+100)

- 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.

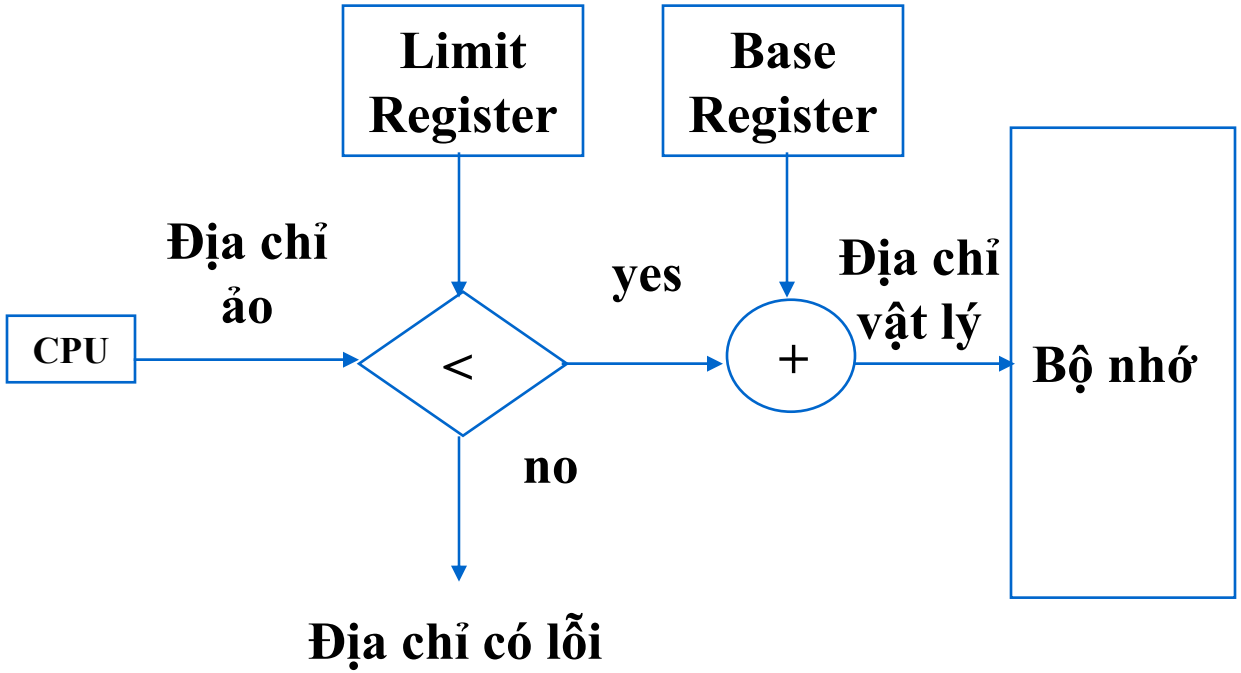


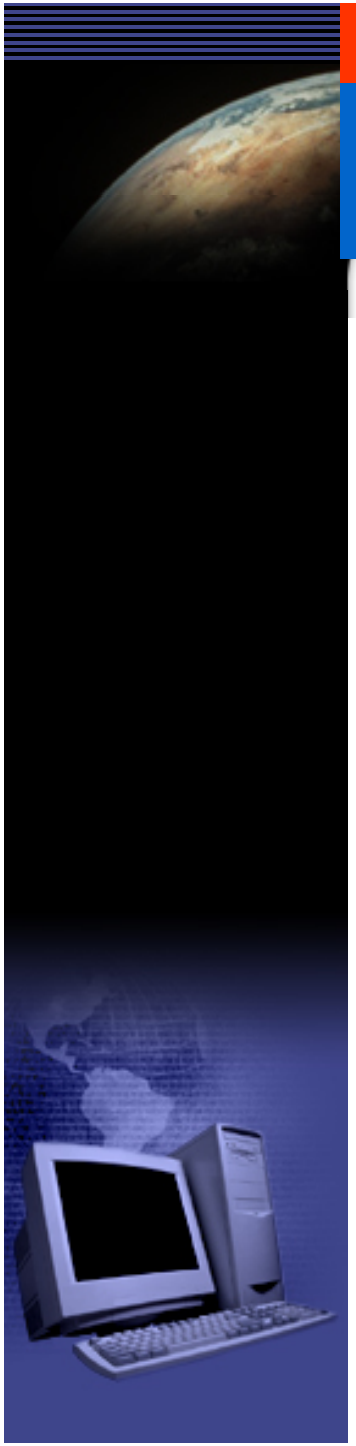
# Cấp phát liên tục

- 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**

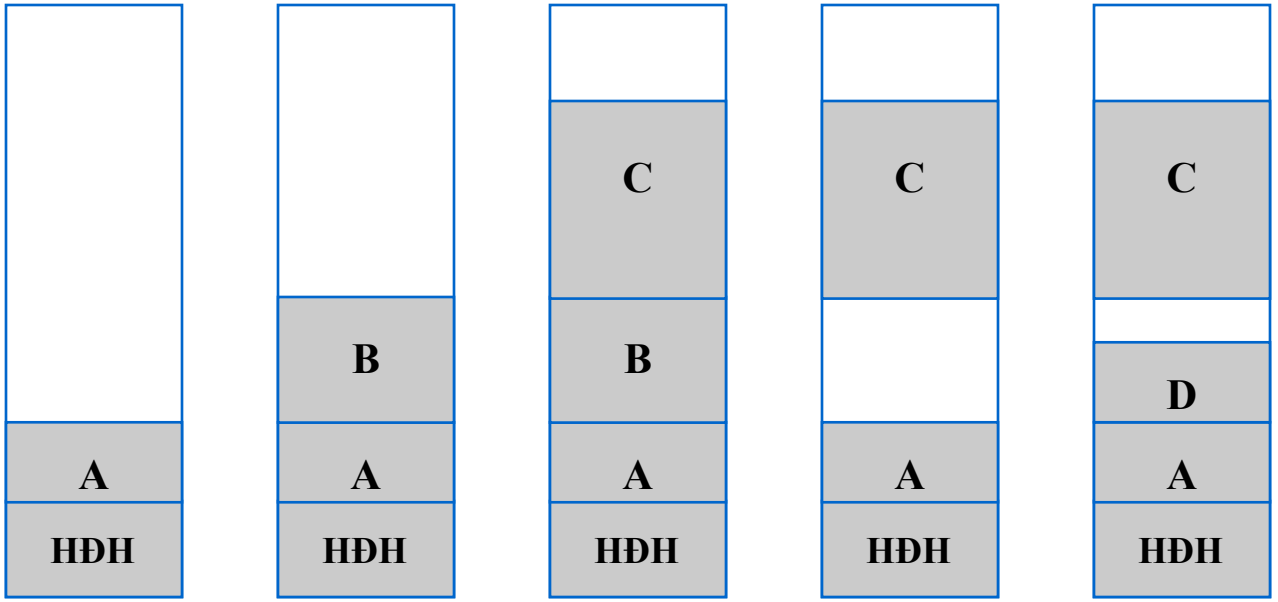
- **Xảy ra hiện tượng phân mảnh ngoại vi**
- **Kỹ thuật "dồn bộ nhớ": kết hợp các mảnh bộ nhớ nhỏ rời rạc thành một vùng nhớ lớn liên tục**

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

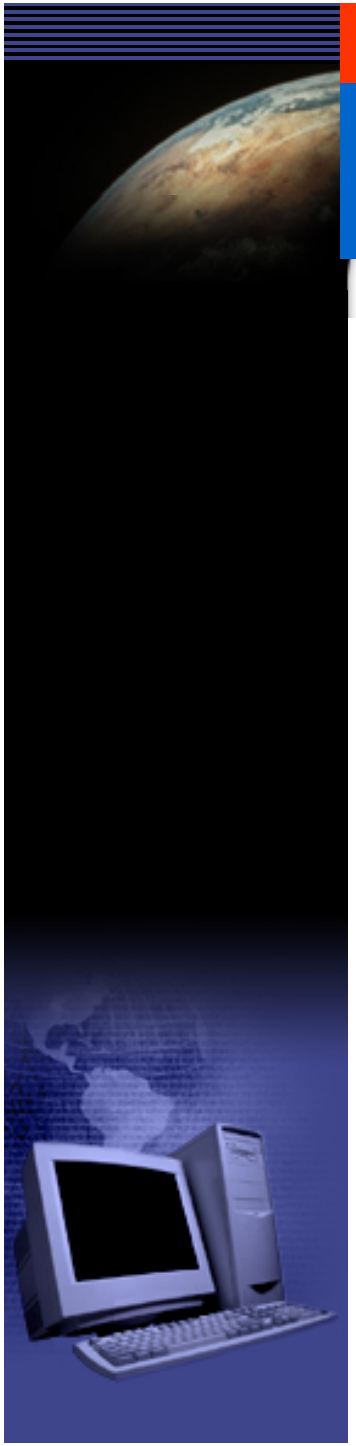
⇒ **Kích thước tiến trình tăng trưởng trong quá trình xử lý mà không còn vùng nhớ trống gần kề (dời chỗ tiến trình, cấp phát dư).**

# Cấp phát liên tục

- Các hệ thống đa chương với phân vùng động




Cấp phát các phân vùng động

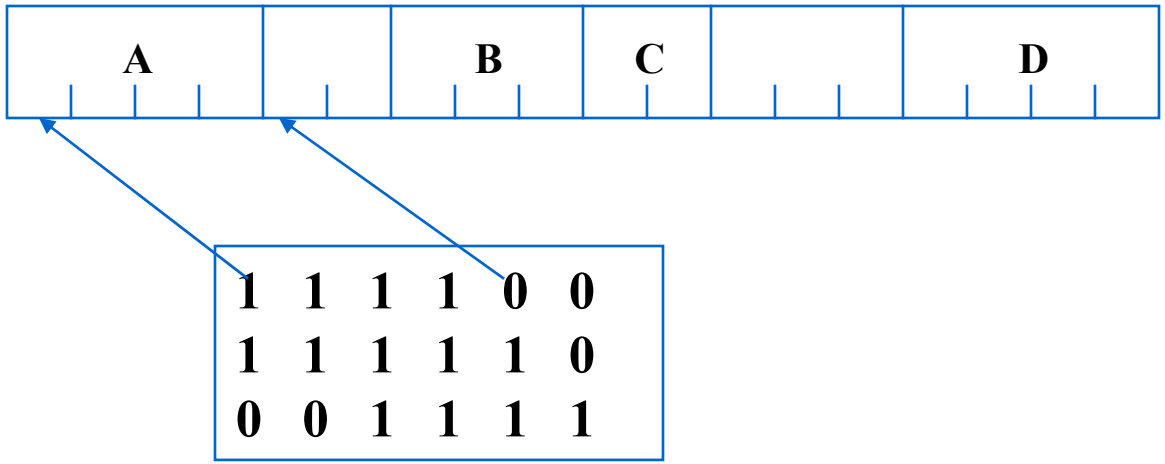


# Cấp phát liên tục

- 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

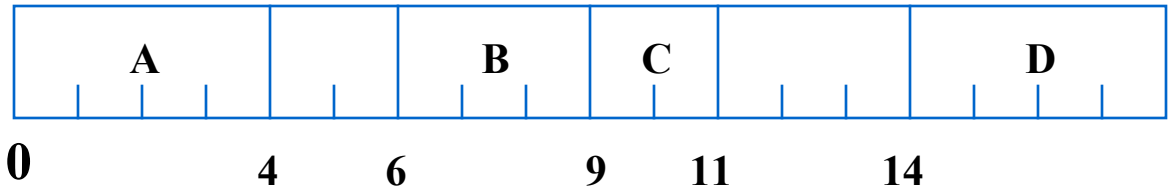
# Cấp phát liên tục

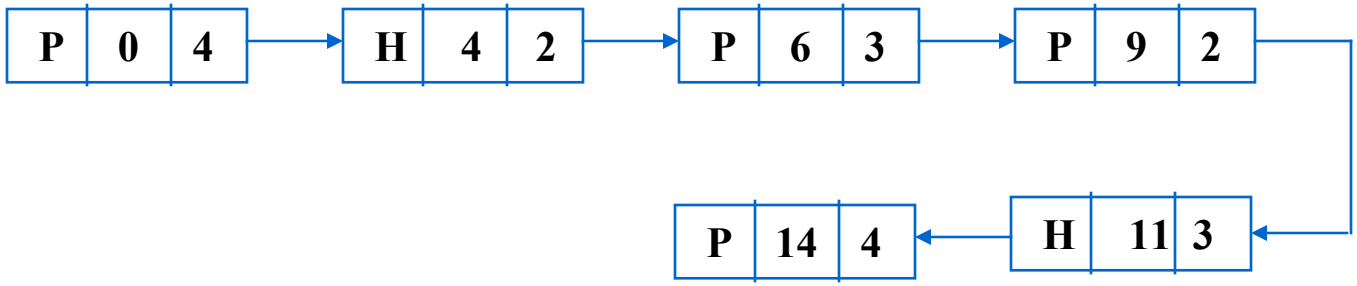
- Các hệ thống đa chương với phân vùng động
- Quản lý bằng một bảng các bit



# Cấp phát liên tục

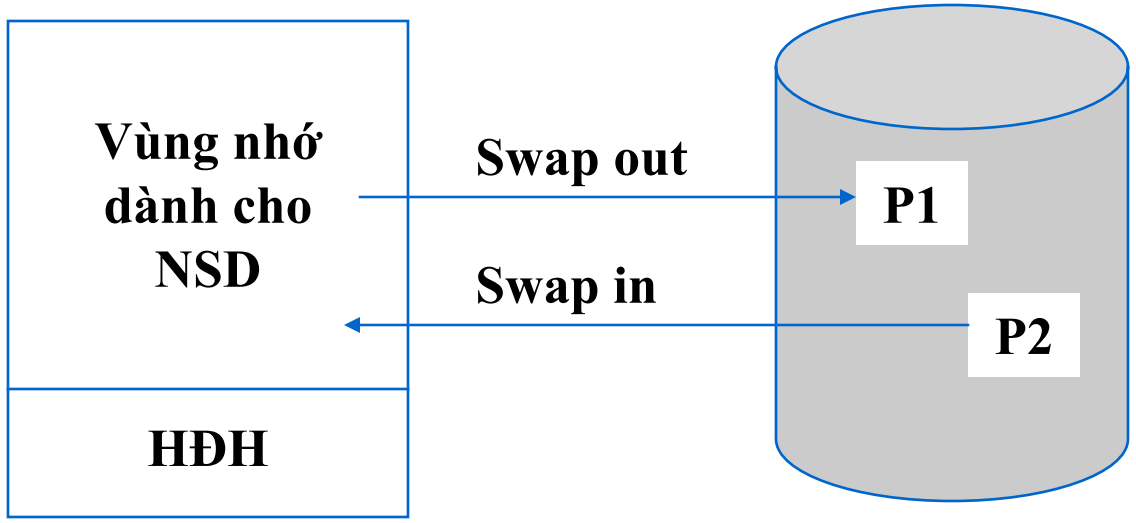
- Các hệ thống đa chương với phân vùng động
- Quản lý bằng danh sách

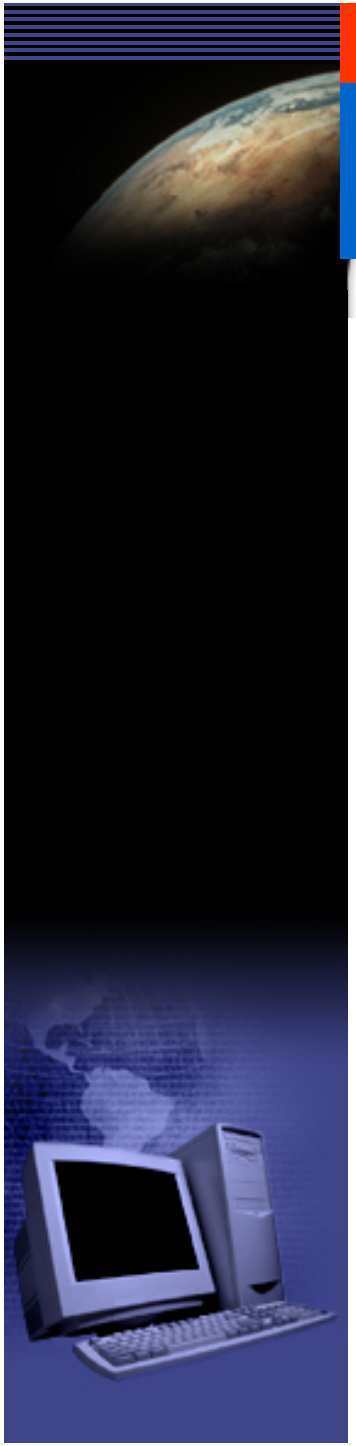


# Cấp phát liên tục

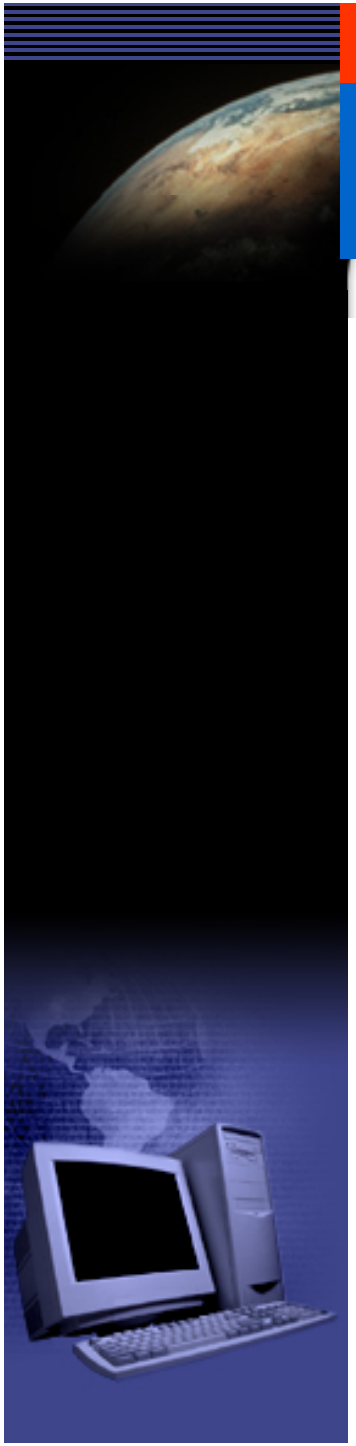
- Các hệ thống đa chương với kỹ thuật "Swapping"





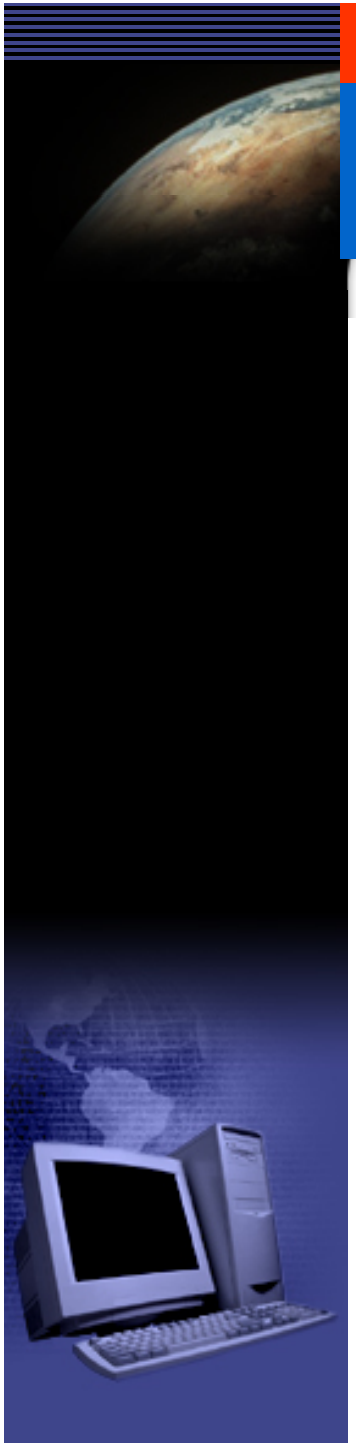
# Cấp phát liên tục

- 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

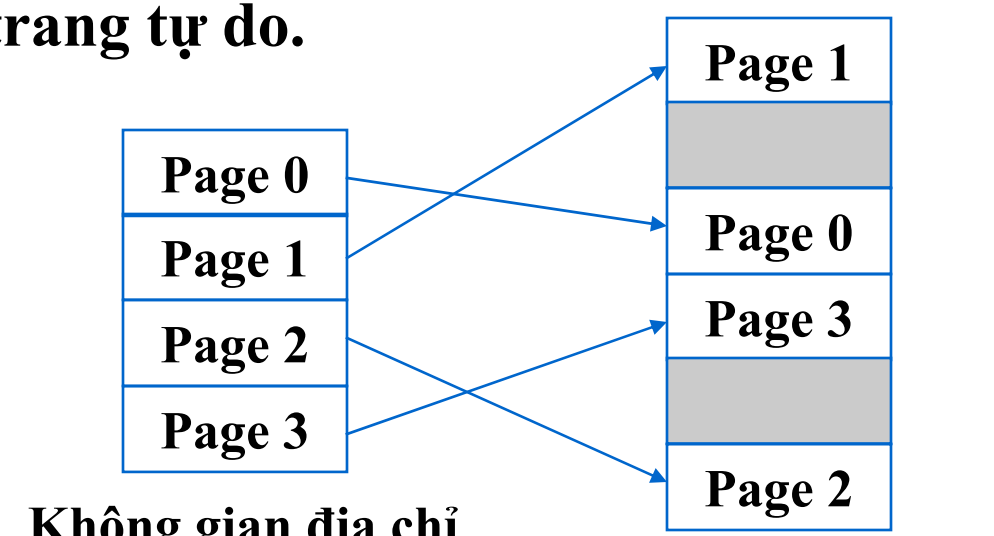
- **Phân trang**
  - **Ý tưởng**
  - **Cơ chế MMU**
  - **Chuyển đổi địa chỉ**
  - **Cài đặt bảng trang**
  - **Tổ chức bảng trang**

# 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.**

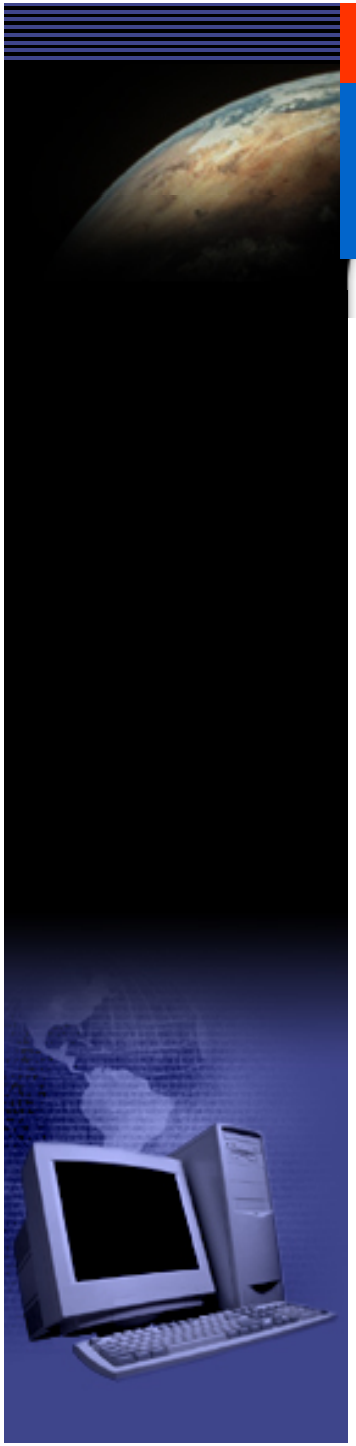
# Cấp phát không liên tục

- **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**

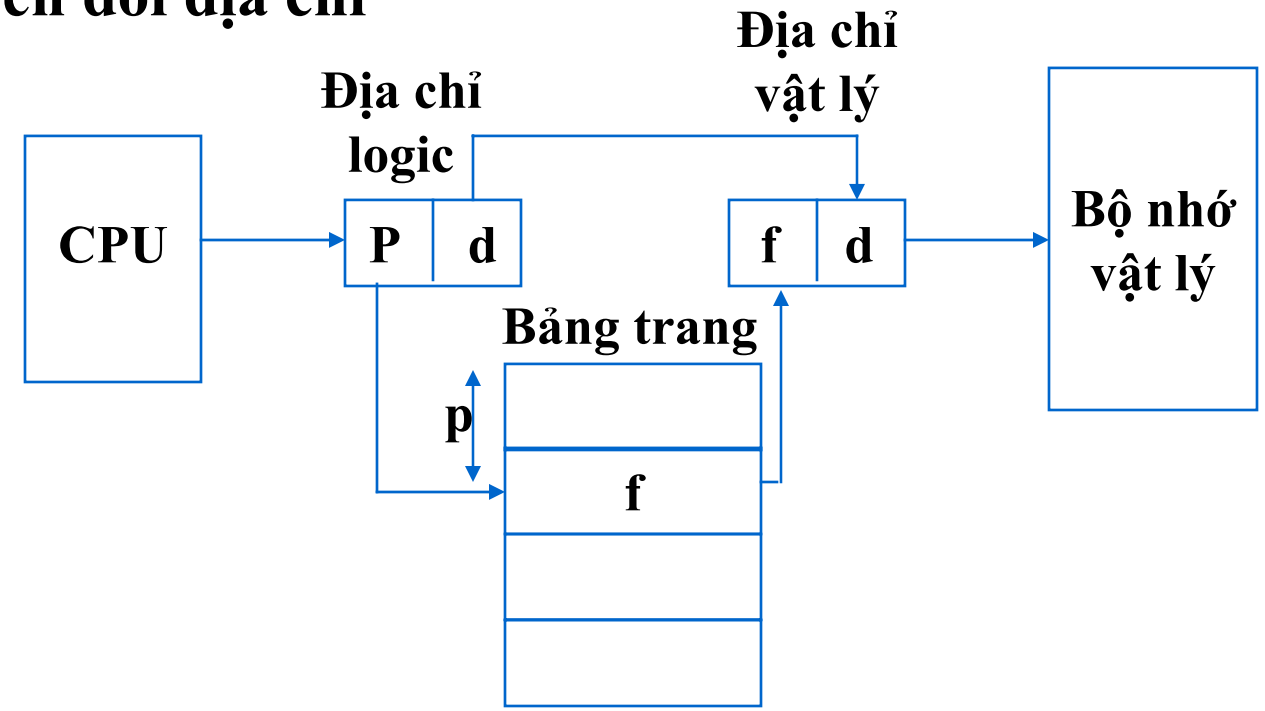
**+ p: số hiệu trang**

**+ d: địa chỉ tương đối**

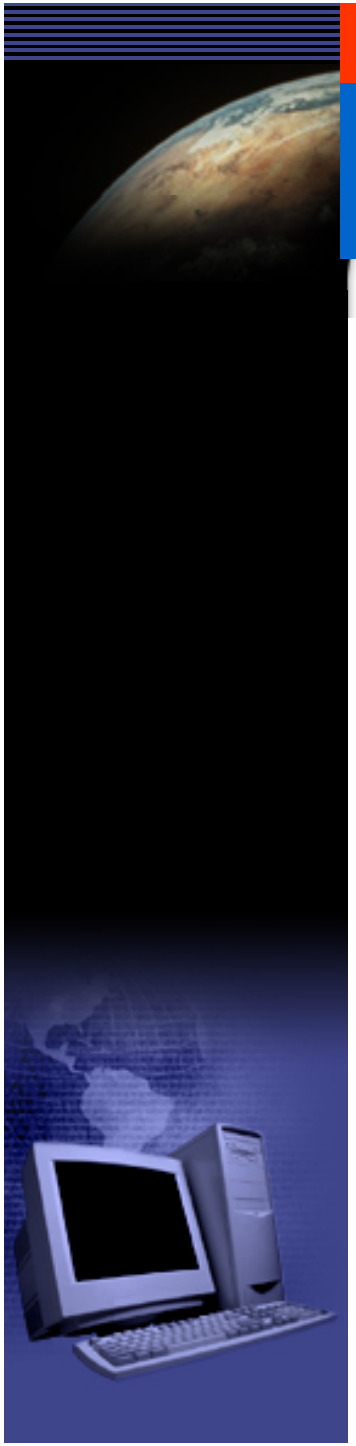
**- Địa chỉ vật lý=địa chỉ bắt đầu của trang + d.**

# Cấp phát không liên tục

- Phân trang
  - Chuyển đổi địa chỉ




Cơ chế phần cứng hỗ trợ phân trang

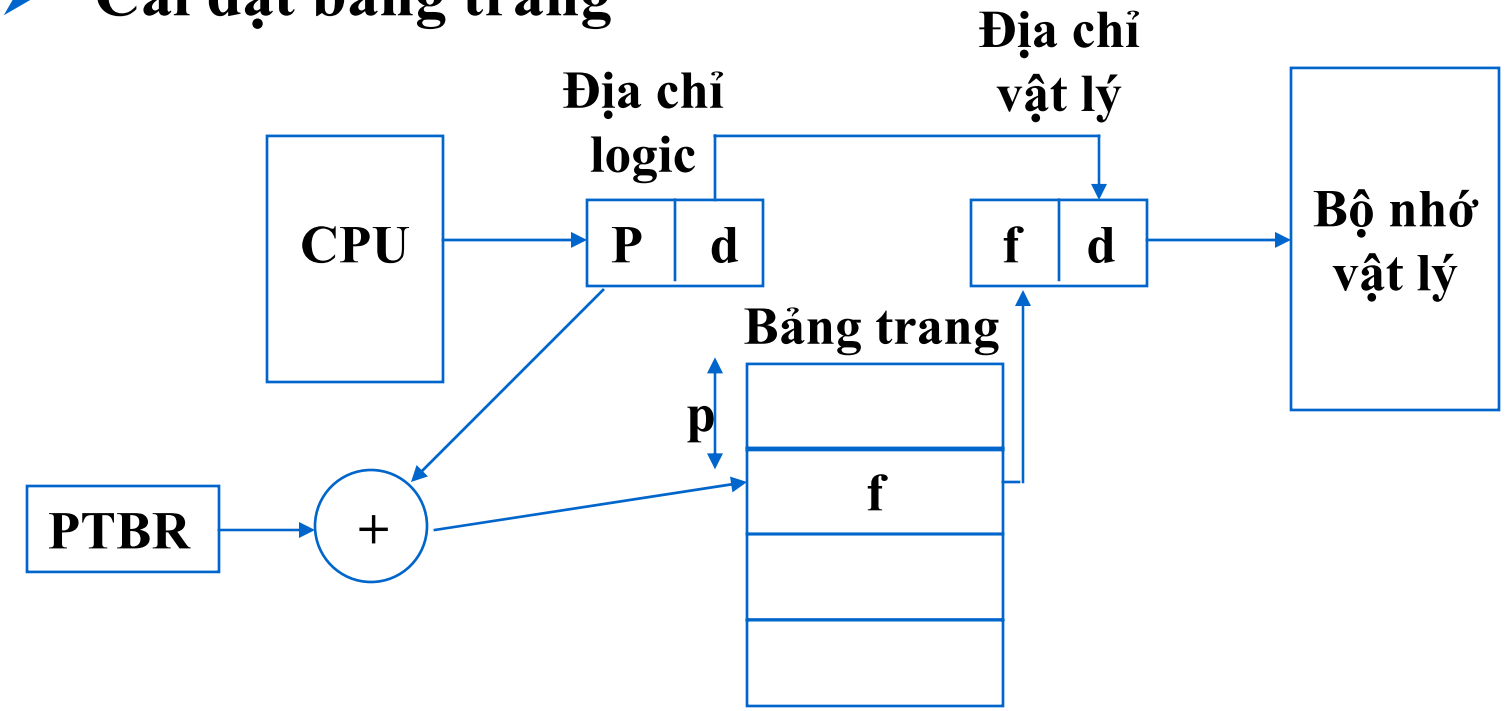


# Cấp phát không liên tục

- **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

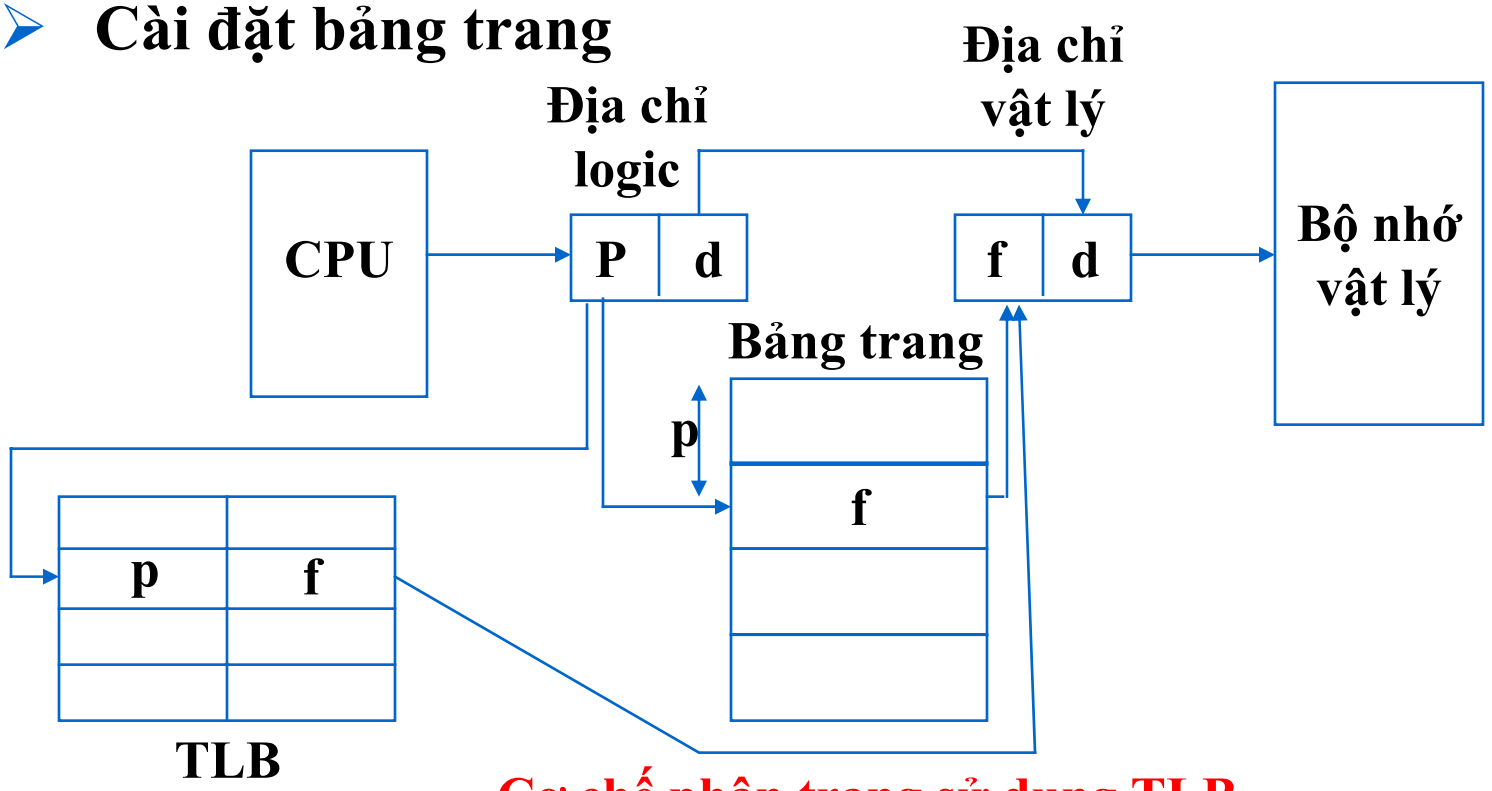
- Phân trang
  - Cài đặt bảng trang



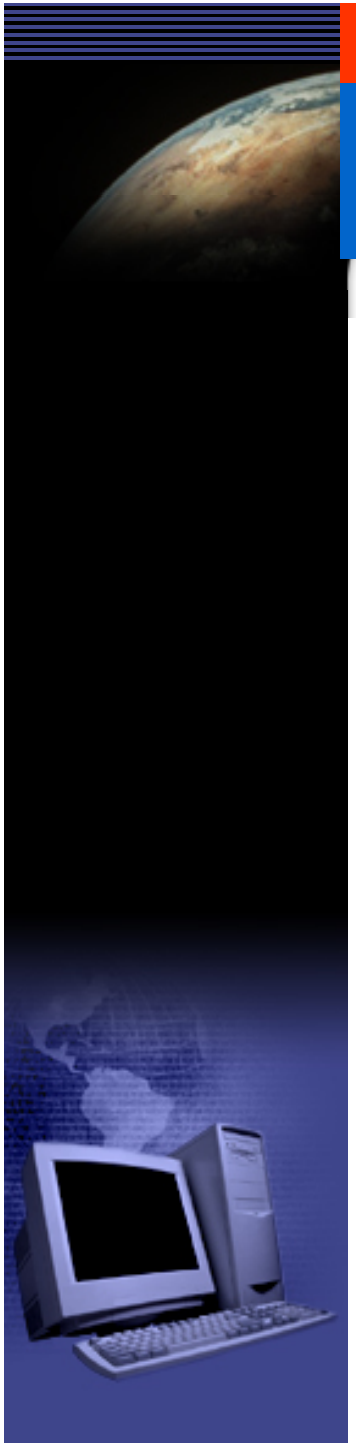

Cơ chế phân trang sử dụng PTBR

# 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.**

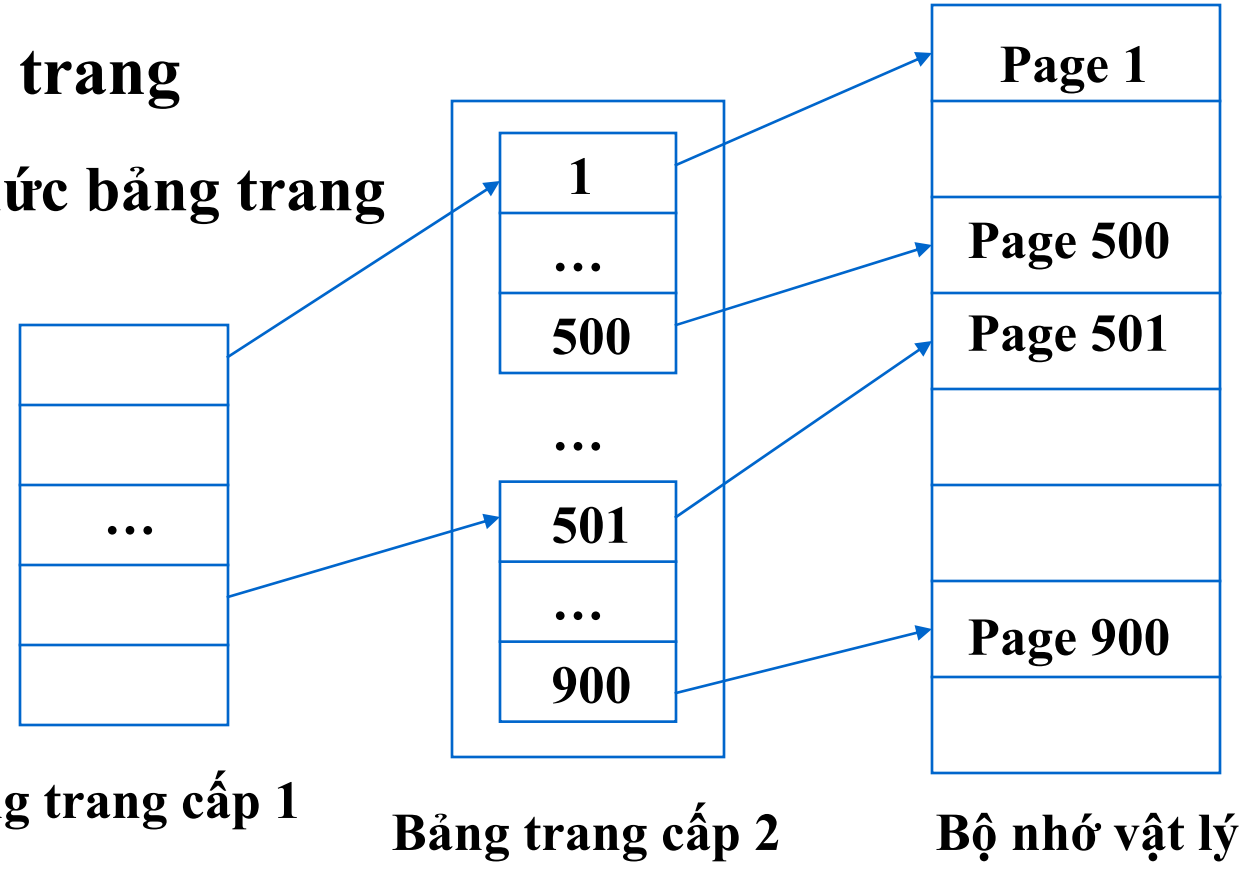
# Cấp phát không liên tục

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

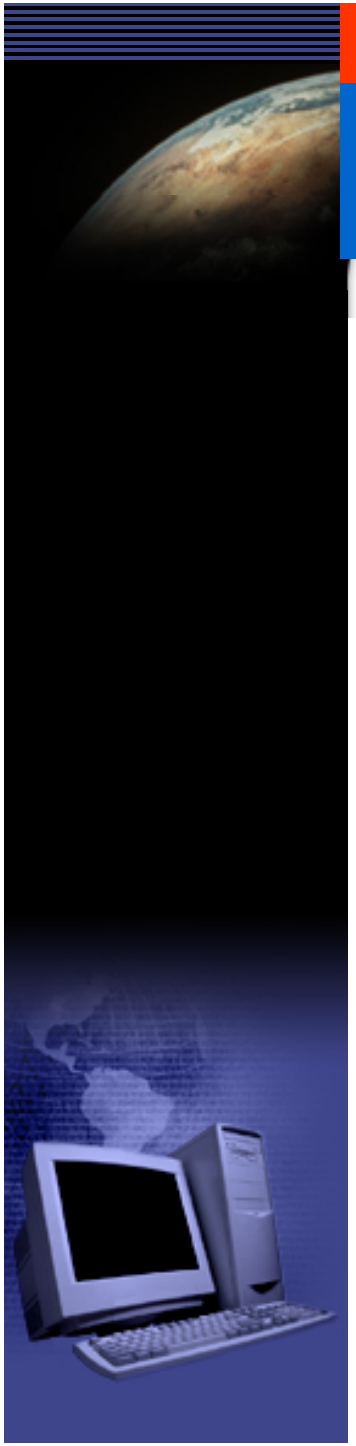
  **Phân chia bảng trang thành các phần nhỏ, bản thân bảng trang cũng sẽ được phân trang**

# Cấp phát không liên tục

- Phân trang
  - Tổ chức bảng trang

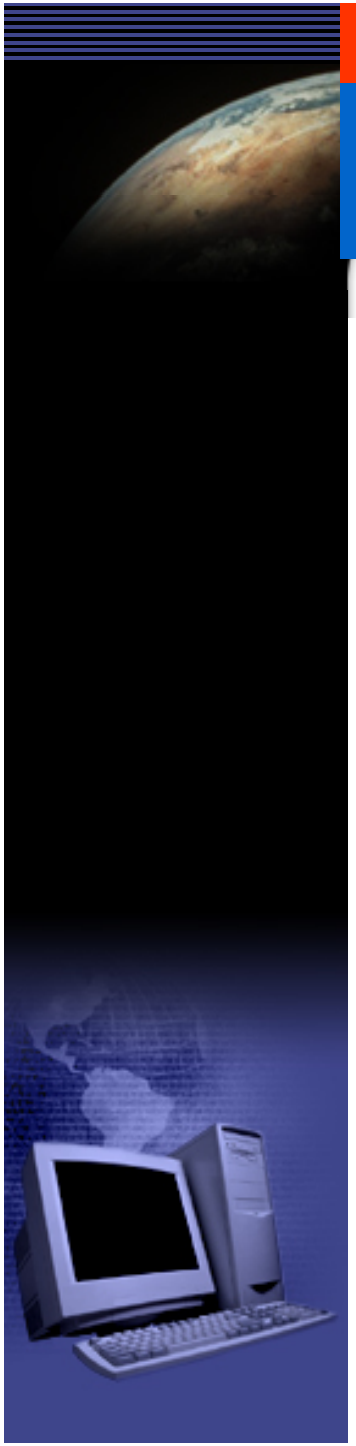



Bảng trang nhị cấp



# 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**



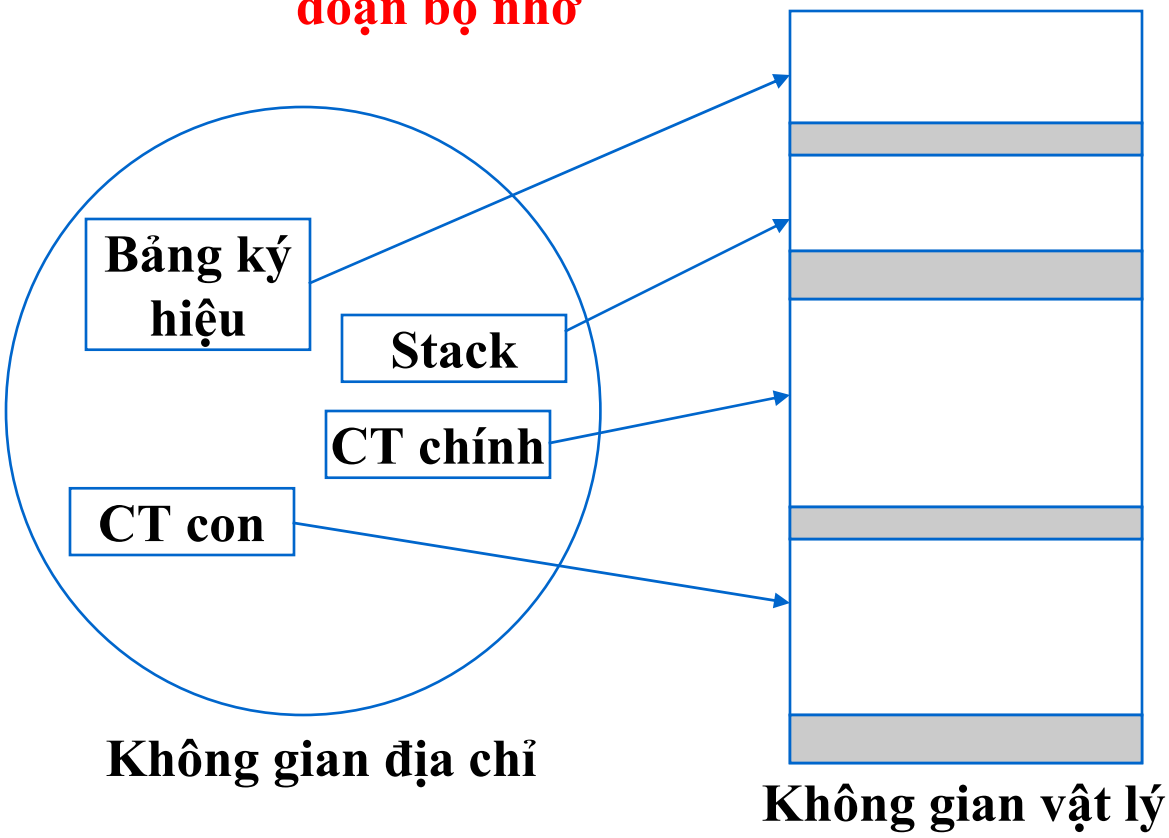
# Cấp phát không liên tục

- **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

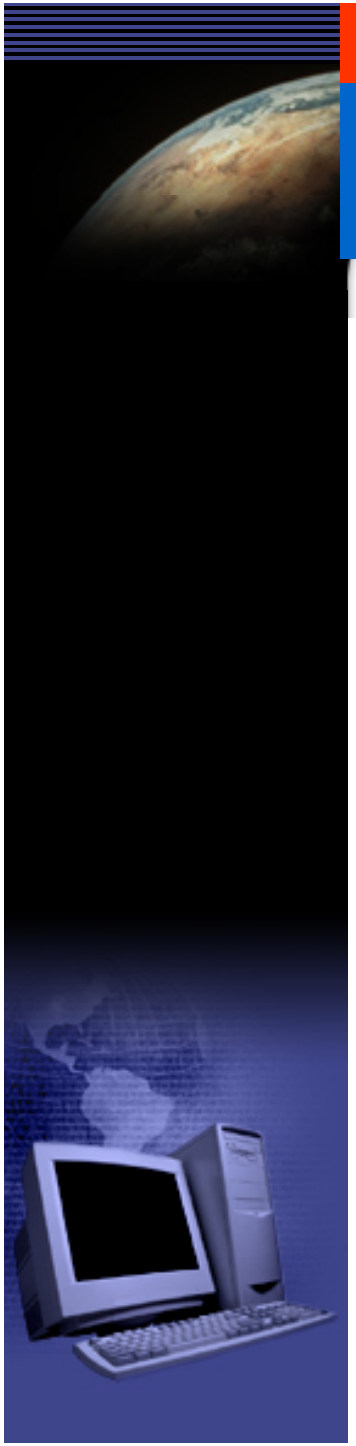
- Phân đoạn
  - Ý tưởng

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**

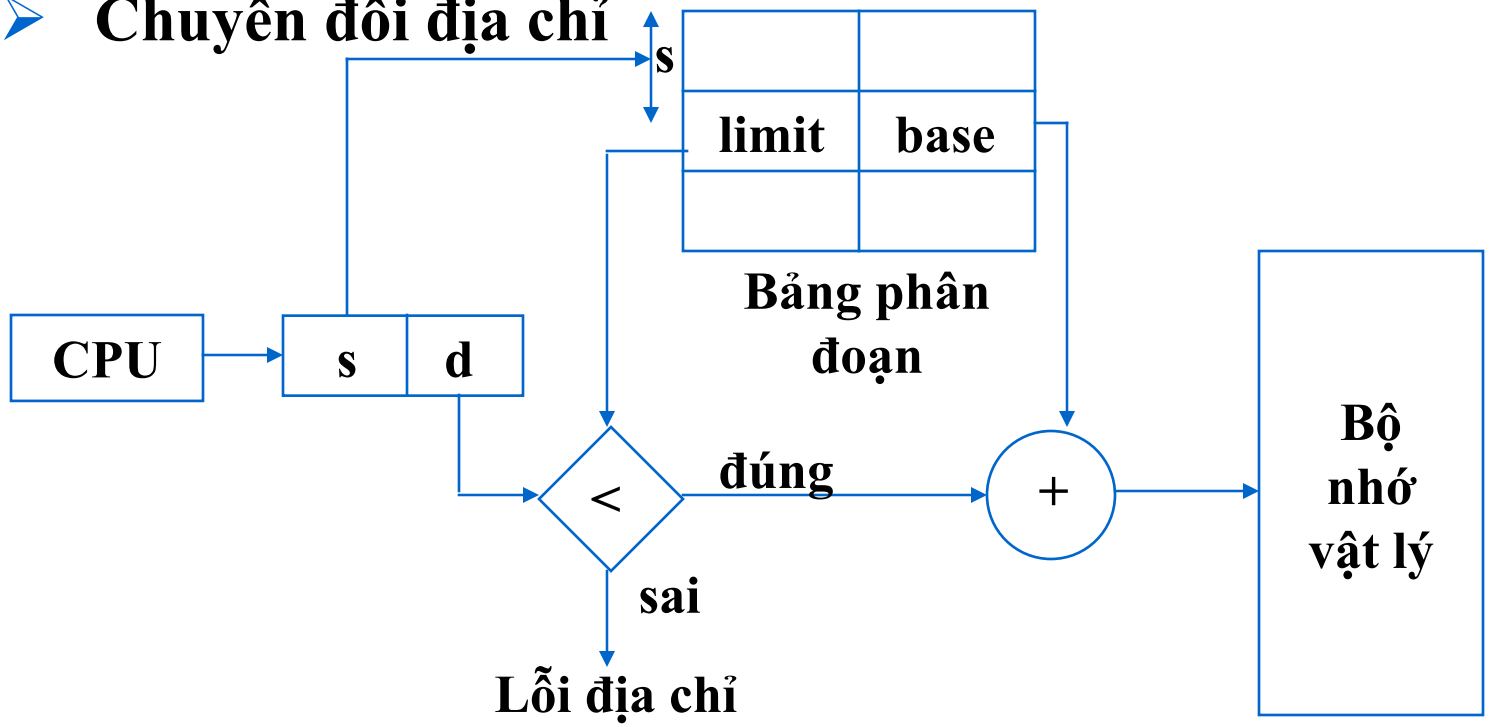


# Cấp phát không liên tục

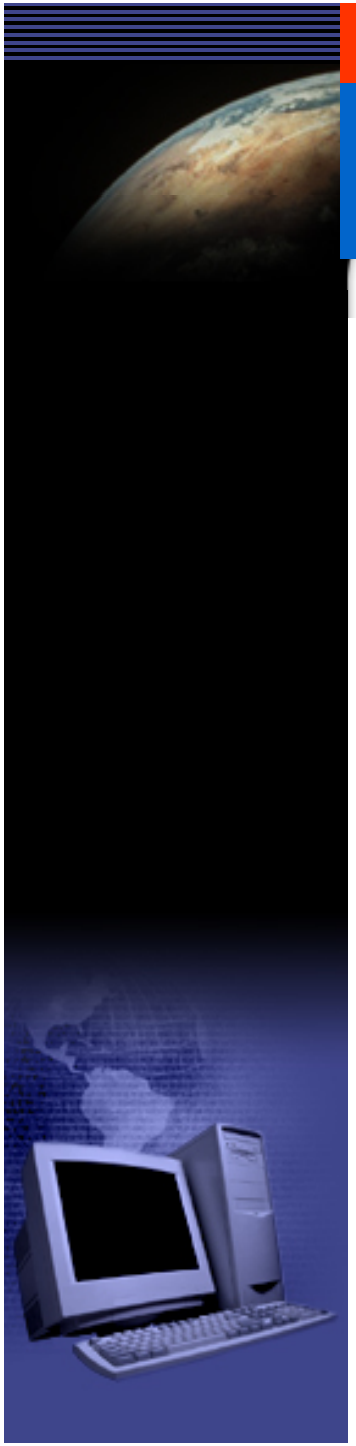
- **Phân đoạn**
  - **Chuyển đổi địa chỉ**
    - **Mỗi địa chỉ logic: <s,d>**
      - **s: số hiệu phân đoạn**
      - **d: địa chỉ tương đối offset, có giá trị từ 0 đến độ dài phân đoạn.**
    - **Địa chỉ vật lý=d+ giá trị chứa trong thanh ghi nền**

# Cấp phát không liên tục

- Phân đoạn
  - Chuyển đổi địa chỉ




Cơ chế phần cứng hỗ trợ kỹ thuật phân đoạn



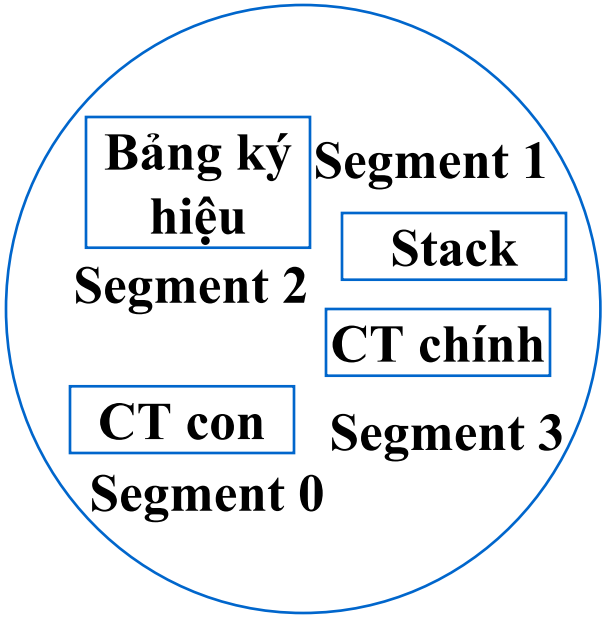
# Cấp phát không liên tục

- **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

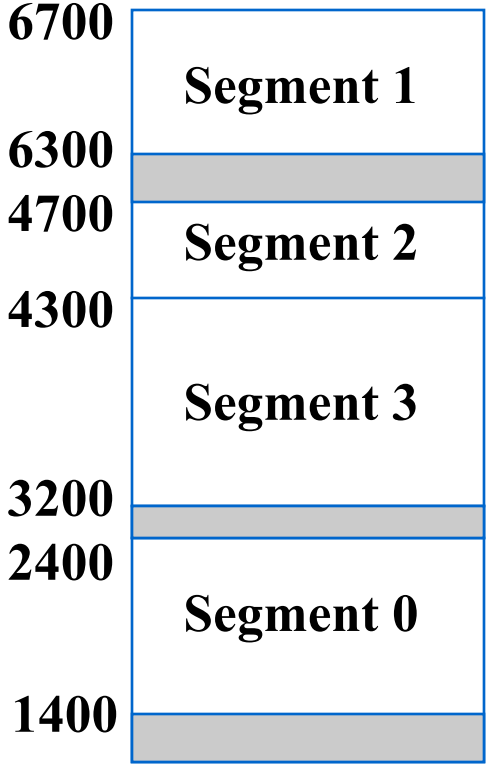
## Phân đoạn

### Cài đặt bảng phân đoạn




| limit | base |
|---|---|
| 1000 | 1400 |
| 400 | 6300 |
| 400 | 4300 |
| 1100 | 3200 |

Hệ thống phân đoạn



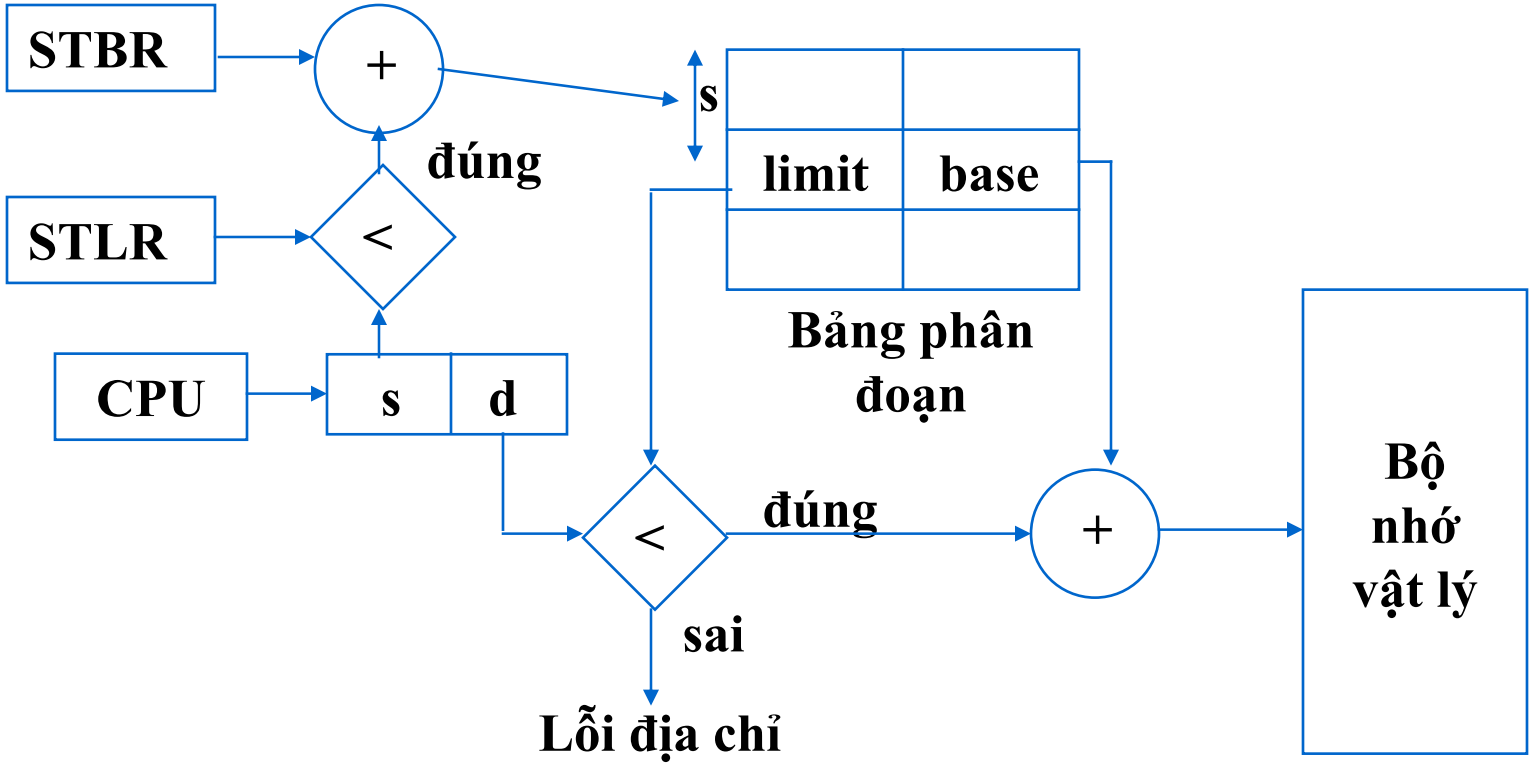

Không gian địa chỉ

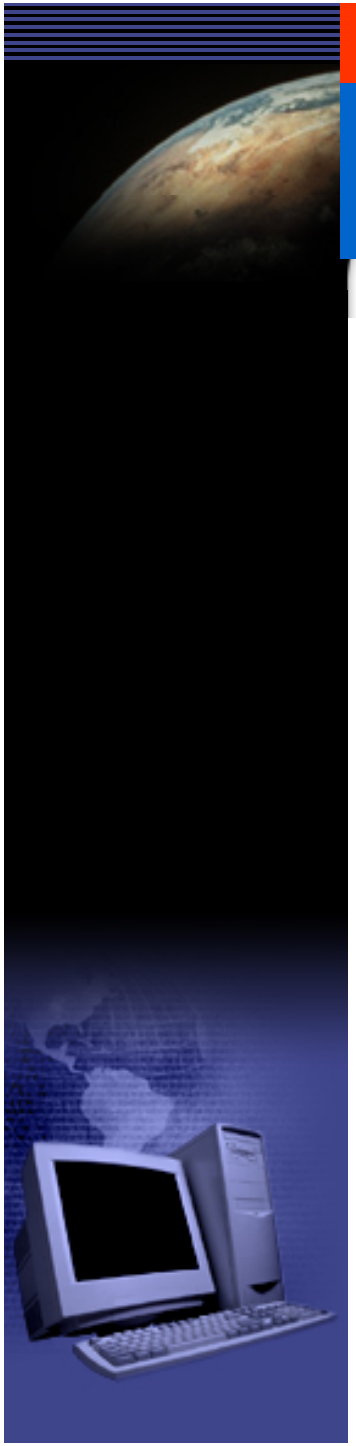
Không gian vật lý

# Cấp phát không liên tục

## Phân đoạn

### Cài đặt bảng phân đoạn





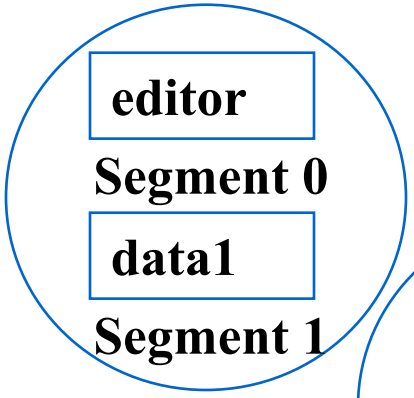
# Cấp phát không liên tục

- **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

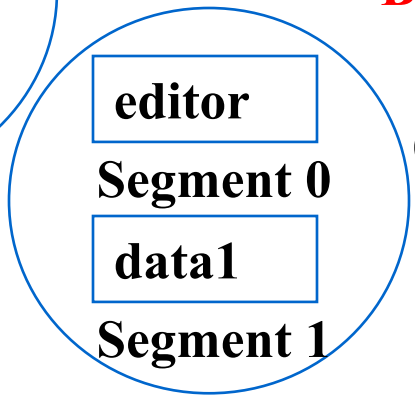
Không gian địa chỉ p1




Bảng phân đoạn p1

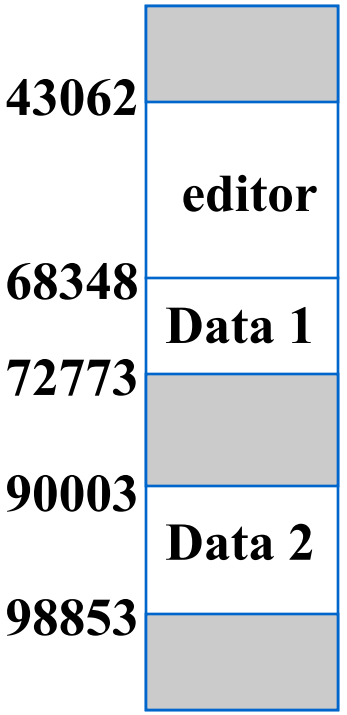
| | limit | base |
|---|---|---|
| 0 | 25286 | 43062 |
| 1 | 4425 | 68348 |

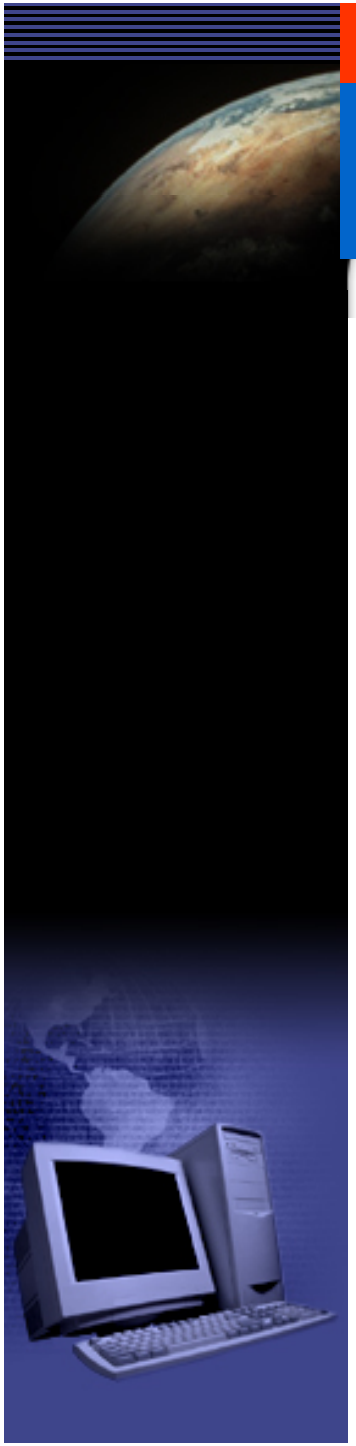
Không gian địa chỉ p2




Bảng phân đoạn p2

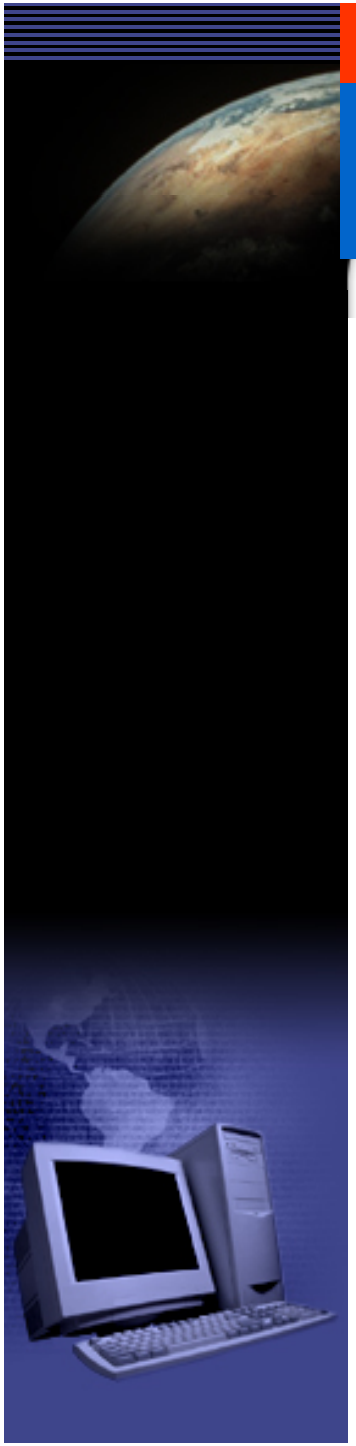
| | limit | base |
|---|---|---|
| 0 | 25286 | 43062 |
| 1 | 8850 | 90003 |





# Cấp phát không liên tục

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

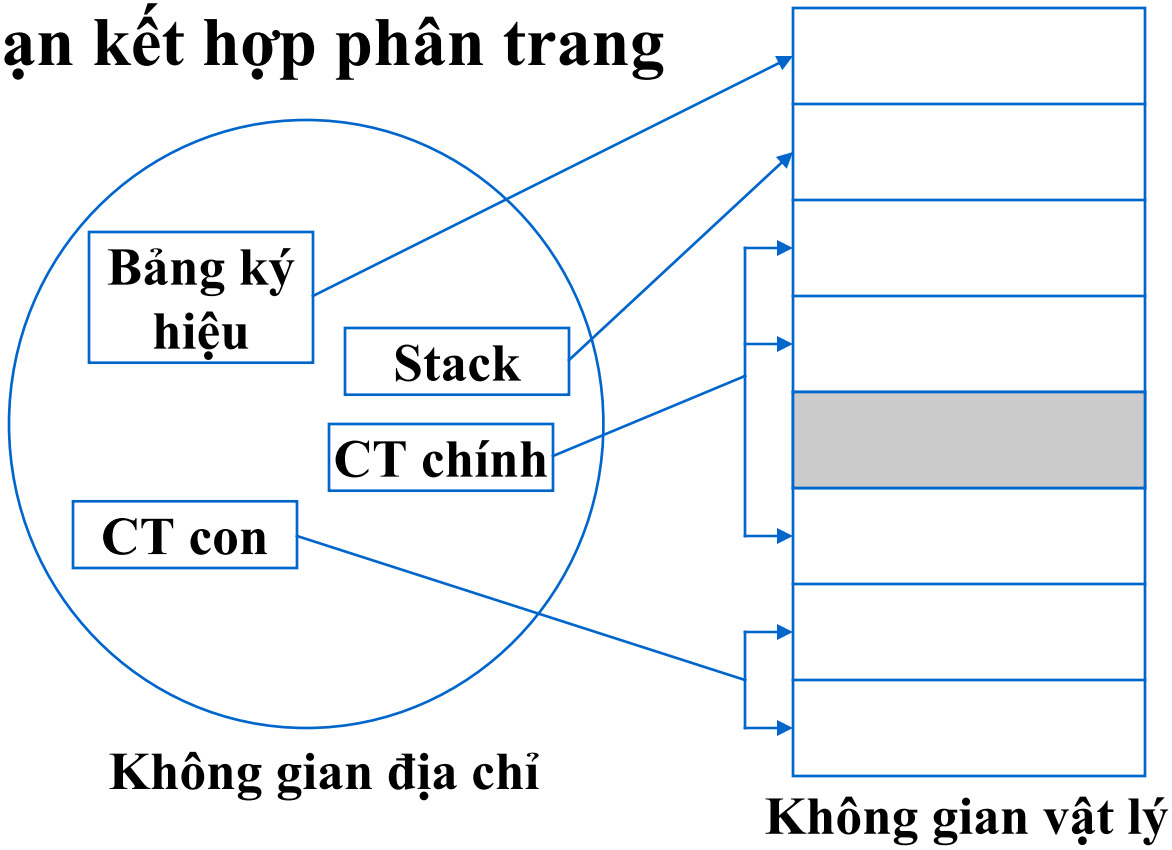


# Cấp phát không liên tục

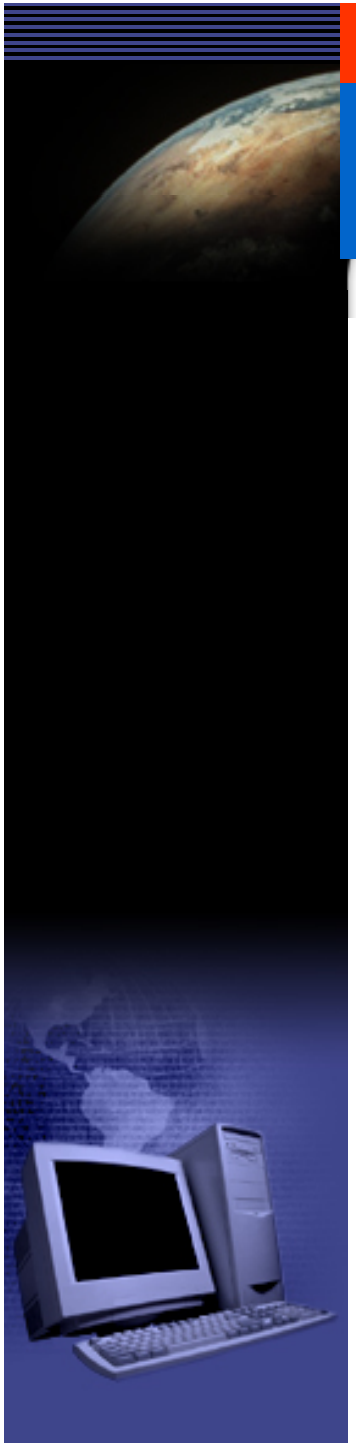
- **Phân đoạn kết hợp phân trang**
  - **Ý tưởng**
    - **Không gian địa chỉ: tập hợp các phân đoạn.**
    - **Mỗi phân đoạn: chia thành nhiều**
    - **Tiến trình được đưa vào hệ thống, HĐH sẽ cấp phát cho tiến trình các trang cần thiết để chứa đủ các phân đoạn của tiến trình**

# Cấp phát không liên tục

- Phân đoạn kết hợp phân trang
  - Ý tưởng




Mô hình phân đoạn kết hợp phân trang

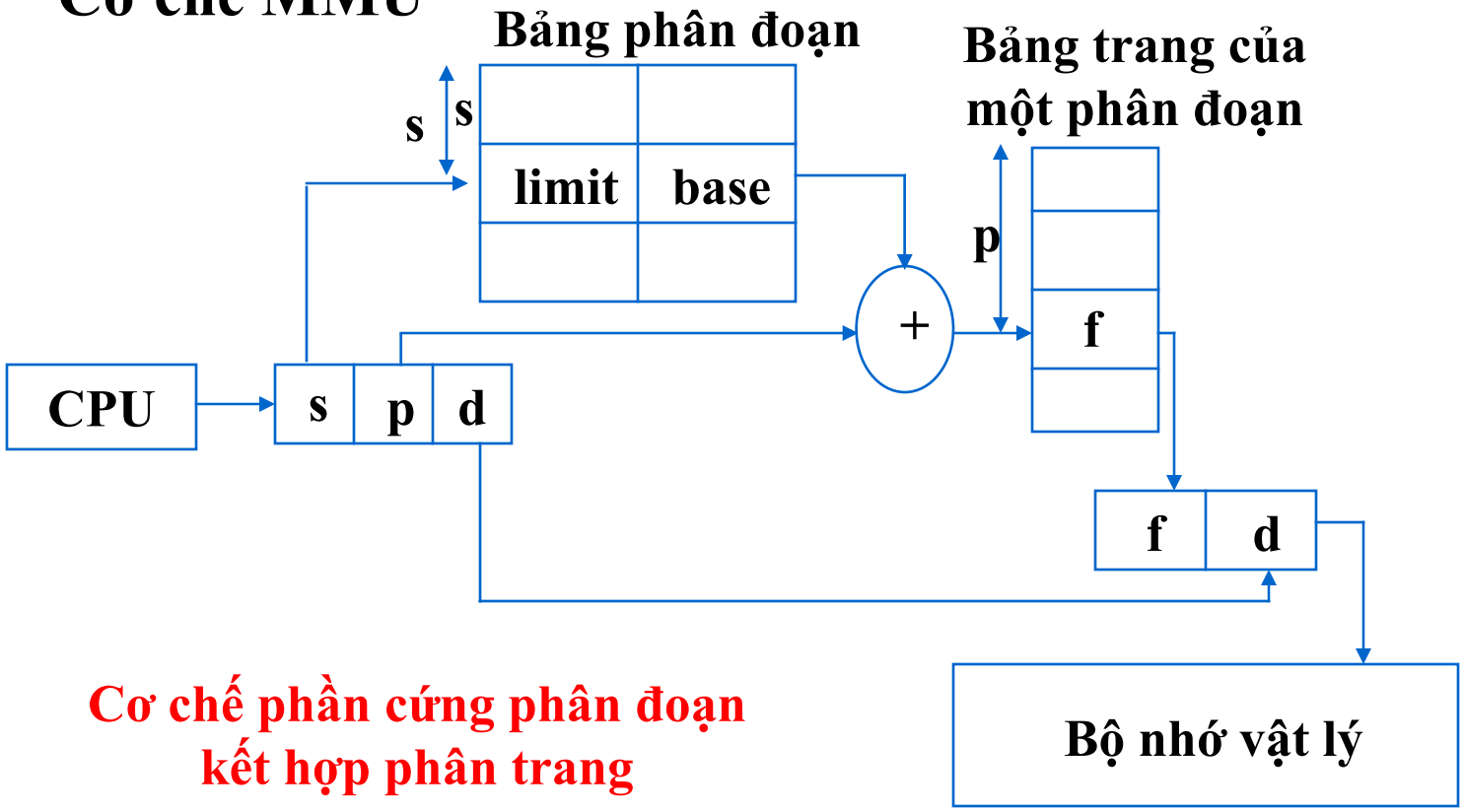


# Cấp phát không liên tục

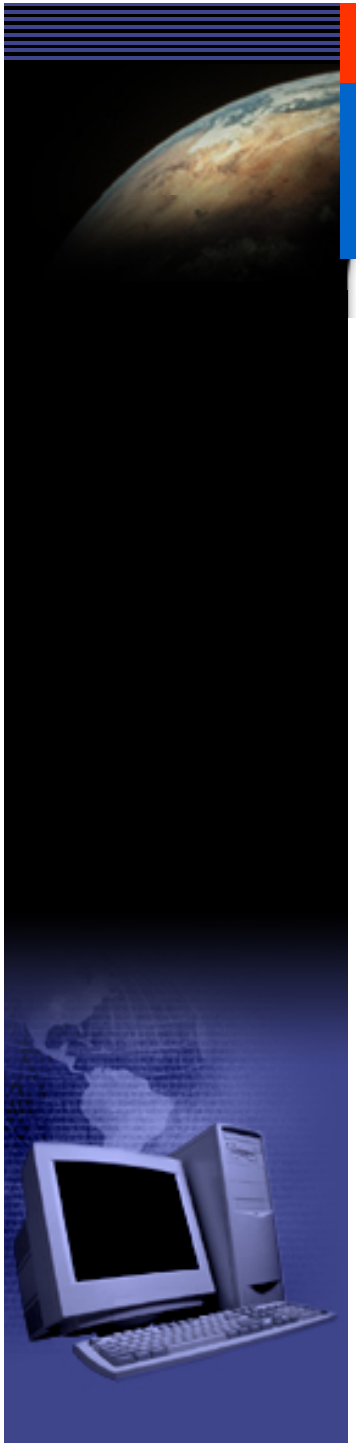
- **Phân đoạn kết hợp phân trang**
- **Chuyển đổi địa chỉ**
  - **Mỗi địa chỉ: <s, p, d>**
    - **S: số hiệu phân đoạn**
    - **P: số hiệu trang**
    - **D: địa chỉ tương đối**

# Cấp phát không liên tục

- Phân đoạn kết hợp phân trang
  - Cơ chế MMU

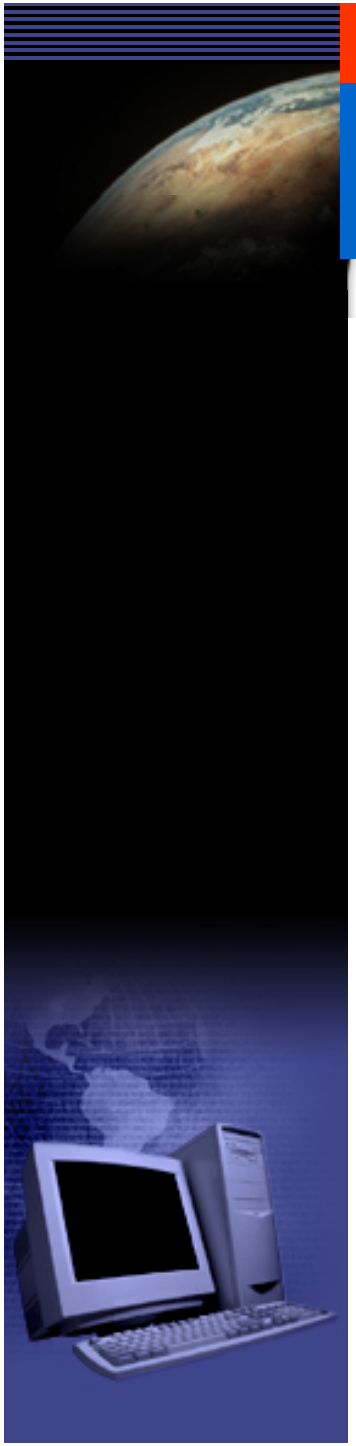



Cơ chế phần cứng phân đoạn kết hợp phân trang



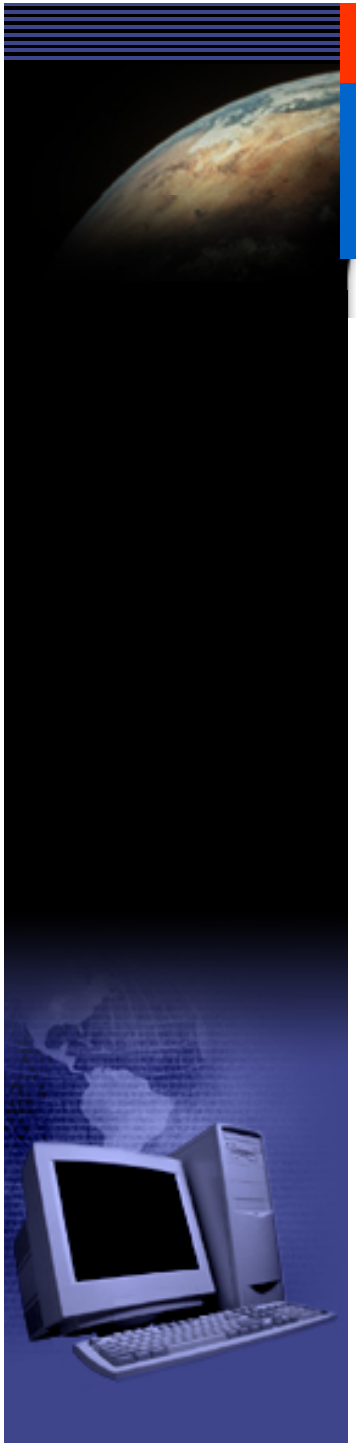
# Bộ nhớ ảo

- **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.**



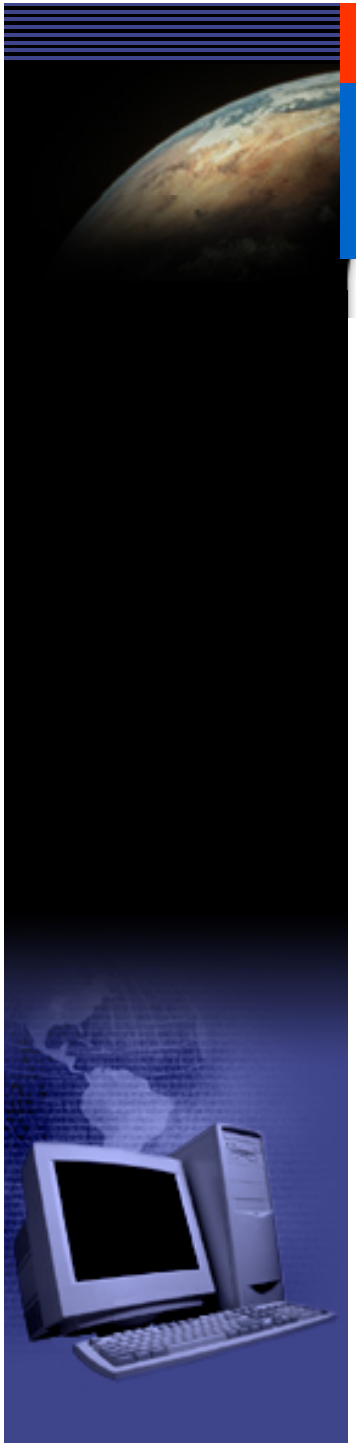
# Bộ nhớ ảo

- **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)

# Bài giảng Hệ Điều Hành Linux

- Contact : ThS. Bùi Trung Úy
  - Email : btrunguy@gmail.com
  - Website :

- Scheduler:
  - Theory : 2 Credits (36 hours)
  - Practise :

# Nội dung môn học

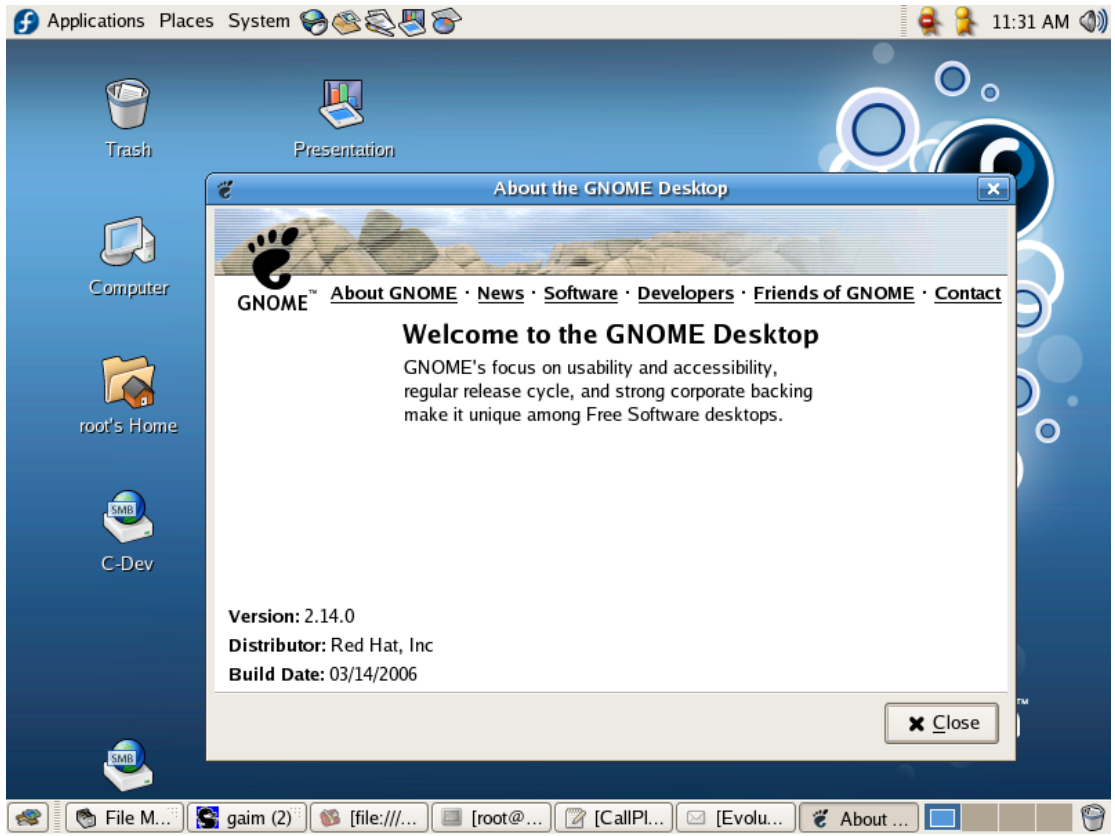
- Tổng quan về Unix/Linux
- Cài đặt và điều quản thiết bị
- Quản trị hệ thống và người dùng
- Thao tác trên hệ thống tập tin
- Xử lý văn bản và các bộ lọc
- Lập trình Shell trên Linux

# Tài liệu tham khảo

- Bài giảng Linux – Gv.Bùi Trung Úy - DTU.
- Bảo mật và tối ưu trong Redhat Linux – Trần Thạch Tùng – NXB LĐXH.
- Cẩm nang Linux – Nguyễn Tiến – Nxb GD.
- Lập trình trên Linux – Nguyễn Phương Lan – Tập 1

- Silberschatz Galvin- Operating System Concepts
- Scott Mann, Ellen L. Mitchel- Linux System Security

# Tổng quan về Linux



# Linux là gì?

- Linux là một HĐH dạng UNIX (Unix-like Operating System) chạy trên máy PC với bộ điều khiển trung tâm (CPU) Intel 80386 trở lên, hay các bộ vi xử lý trung tâm tương thích AMD, Cyrix.
- Linux ngày nay còn có thể chạy trên các máy Macintosh hoặc SUN Sparc.
- Một đặc điểm nỗi bật của Linux là một hệ điều hành miễn phí và mở nguồn mở.

# Lịch sử ra đời của Unix

- Giữa năm 1960, AT&T Bell Laboratories và một số trung tâm khác tham gia tạo ra một HĐH mới được đặt tên là Multics (Multiplexed Information and Computing Service)
- Đến năm 1969, chương trình Multics bị bãi bỏ vì đó là một dự án quá nhiều tham vọng.
- Ken Thompson, Dennis Ritchie và một số đồng nghiệp của Bell Labs đã không bỏ cuộc. Thay vì xây dựng một HĐH làm nhiều việc một lúc, họ phát triển một HĐH đơn giản - chỉ làm tốt một việc là chạy chương trình.
- Peter Neumann đặt tên cho HĐH đơn giản này là Unix.

# Lịch sử ra đời của Unix

- Năm 1973, sử dụng ngôn ngữ C của Ritchie, Thompson đã viết lại toàn bộ hệ điều hành Unix và đây là một thay đổi quan trọng của Unix.
- Nhờ đó Unix từ chỗ là hệ điều hành cho một máy PDP-xx trở thành hệ điều hành có thể chạy trên nhiều loại máy tính khác nhau.
- Khoảng 1977 bản quyền của UNIX được giải phóng và hệ điều hành UNIX trở thành một thương phẩm

# Lịch sử ra đời của Linux

- Năm 1991, Linus Torvalds, sinh viên của đại học tổng hợp Helsinki, Phần lan, bắt đầu xem xét Minix với mục đích nghiên cứu cách tạo ra một HĐH Unix chạy trên máy PC với bộ vi xử lý Intel 80386
- Ngày 25/8/1991, Linus cho ra version 0.01 và thông báo trên comp.os.minix của Internet về dự án của mình.
- Ngày 1/1992, Linus cho ra version 0.12 với shell và C compiler. Linus đặt tên HĐH của mình là Linux.
- Năm 1994, phiên bản chính thức 1.0 được phát hành

# Lịch sử ra đời của Linux

- Linux được viết lại toàn bộ từ con số không, tức là không sử dụng một dòng lệnh nào của Unix, để tránh vấn đề bản quyền của Unix.
- Tuy nhiên hoạt động của Linux hoàn toàn dựa trên nguyên tắc của hệ điều hành Unix. Vì vậy nếu một người nắm được Linux, thì sẽ nắm được UNIX.
- Quá trình phát triển của Linux được tăng tốc bởi sự hỗ trợ của chương trình GNU (GNU's Not Unix)

# Lịch sử ra đời của Linux

- Linux có một linh vật chính thức –Linux penguin, gọi là Tux.
- Hình vẽ sau cho thấy linh vật của Linux

*The Linux Penguin Tux*

# Vấn đề bản quyền GNU

- Các chương trình tuân theo GNU Copyleft or GPL (General Public License) có bản quyền như sau:
  - Tác giả vẫn là sở hữu của chương trình của mình.
  - Ai cũng được quyền bán copy của chương trình với giá bất kỳ mà không phải trả cho tác giả ban đầu.
  - Người sở hữu chương trình tạo điều kiện cho người khác sao chép chương trình nguồn để phát triển tiếp chương trình

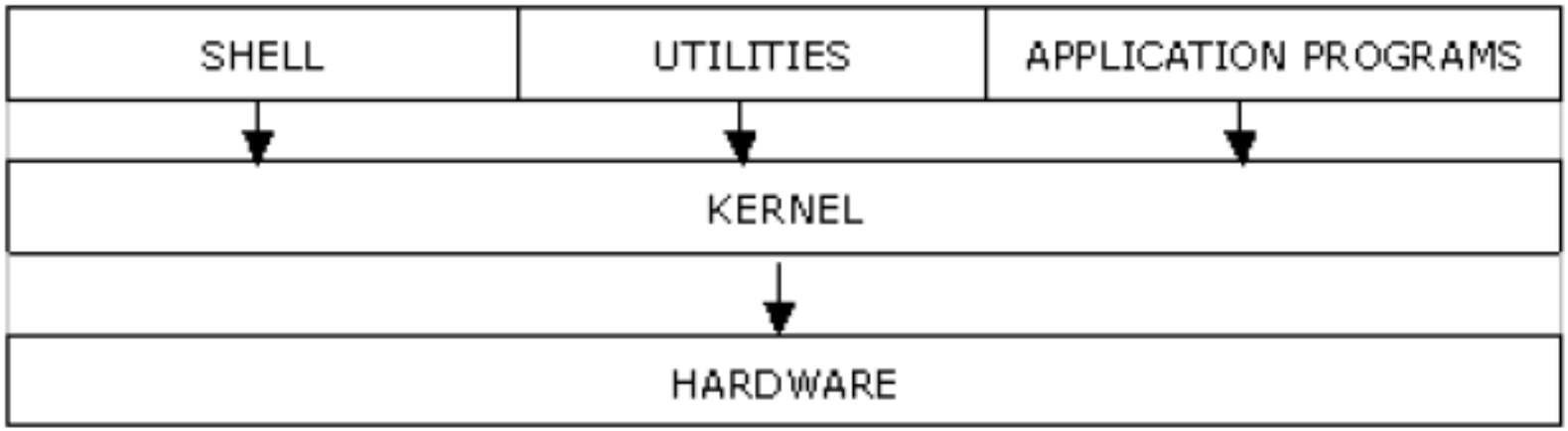
# Các đặc trưng của Linux (1)

- Linux là miễn phí (free) và Open Source: Mã nguồn mở, bao gồm cả kernel, drivers, các công cụ phát triển,…
- Linux rất ổn định: Ngay cả server Linux phục vụ những mạng lớn (hàng trăm máy trạm) cũng hoạt động rất ổn định.
- Multi-Tasking, Multi-Threading: là khả năng mà HĐH gán cho từng tiến trình hoặc tuyến quyền sử dụng CPU trong một khoảng thời gian nhất định
- Multi-User: là khả năng cho phép nhiều người dùng đồng thời truy cập cùng một CPU.

# Các đặc trưng của Linux (2)

- Multi-platform: Chạy trên nhiều nền tảng phần cứng khác nhau.
- Multi-standard Compliant: Tương thích với hầu hết các hệ POSIX, System V, và BSD (ở mức source).
- Hỗ trợ nhiều hệ thống File: Minix-1, MS-DOS, VFAT, FAT-32, ISO 9660 (CD-ROMs),…hai hệ thống tập tin chính của Linux là ext2fs và ext3fs.
- Multiple Networking Protocols: Các giao thức nền tảng được hỗ trợ bởi Kernel như: TCP, IPv4, IPv6, AX.25, X.25, IPX, Appletalk, Netrom, v.v…

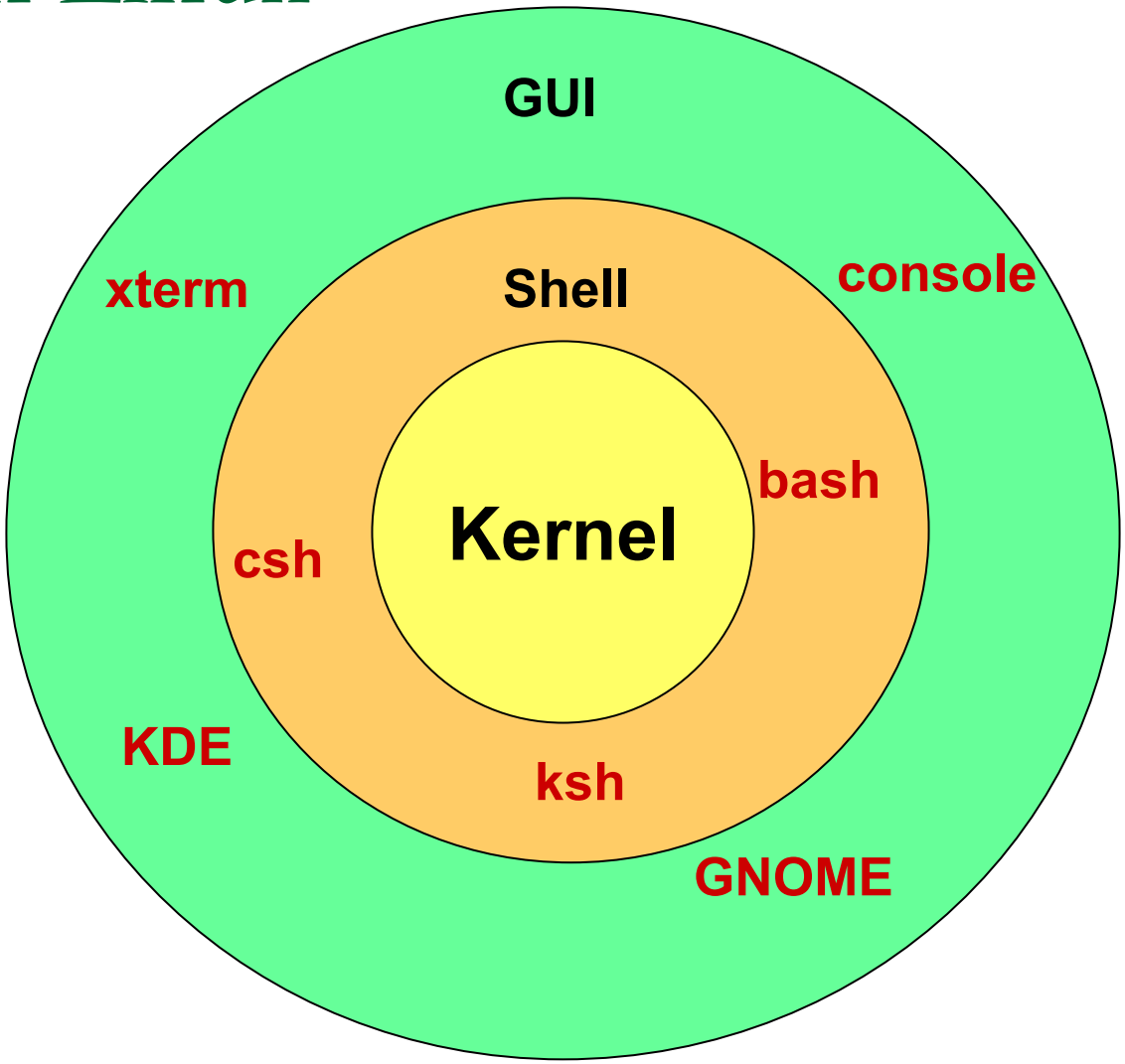
# Các thành phần chính của Linux



# Các thành phần chính của Linux

- Hệ lõi (kernel-nhân): xác lập nhiều thường trình cấp thấp và tương tác trực tiếp với CPU, điều khiển thiết bị phần cứng và điều khiển việc thực hiện chương trình.
- Cấu trúc hệ thống tập tin: là hệ thống lưu trữ các thông tin trên thiết bị lưu trữ.
- Hệ võ (shell): là cách người dùng tương tác gián tiếp với phần cứng thông qua kernel. Hệ võ ngầm định là bash. Các hệ võ khác như tcsh, ksh, zsh...
- Các tiện ích: có chức năng chính là thực hiện các công việc dịch vụ của hệ điều hành.

# Nhân Linux



# Nhân Linux

- Dự án đước khởi xướng vào năm 1991 bởi Linus Tovard bằng một bài viết nổi tiếng trong nhóm tin Usenet comp.os.minix, trong đó có đoạn viết:
  - *"I'm doing a (free) operating system (just a hobby, won't be big and professional like gnu) for 386(486) AT clones..." [1]*
- Phần hạt nhân (lõi hay kernel) của Linux có thể hiểu đơn giản là một tập hợp các chương trình thường trú trong bộ nhớ.
- Kernel là phần chính của hệ điều hành, phụ trách hầu hết các chức năng chính của hệ điều hành như quản lý bộ nhớ, thực thi nhiệm vụ và truy nhập phần cứng...

# Phiên bản nhân

- Các phiên bản của nhân Linux được xác định bởi hệ thống số dạng: X.YY.ZZ
  - Nếu YY là số chẵn => phiên bản ổn định.
  - Nếu YY là số lẻ => phiên bản thử nghiệm (không dùng để phát triển các bản phân phối)
- Ví dụ: Kernel 2.4.20
  - 2 là Số chính
  - 4 là số phụ, phiên bản ổn định
  - 20 là cấp vá đắp (patch level), phiên bản ổn định (nếu số lẻ là phiên bản đang thử nghiệm)

# Bản phân phối Linux

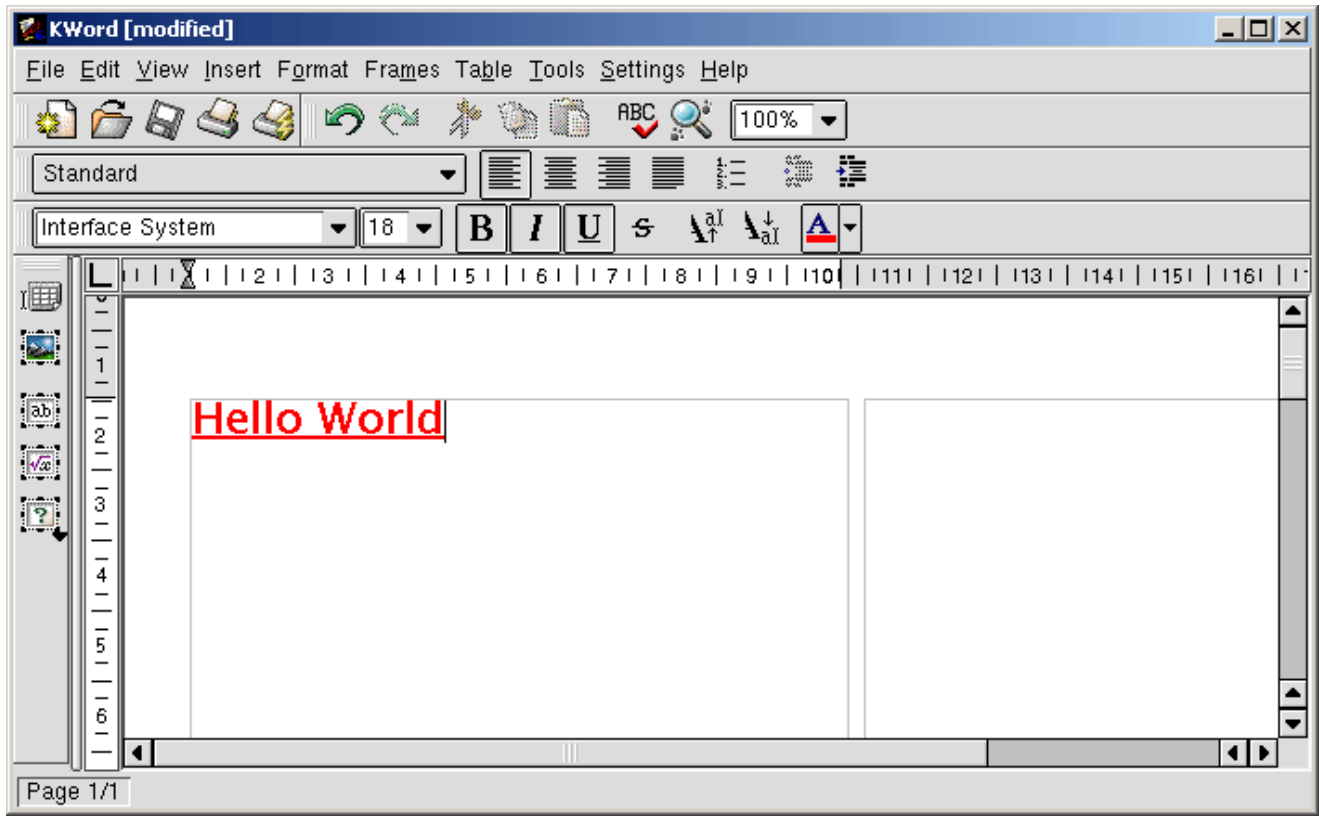
- Bản phân phối Linux là bộ các chương trình ứng dụng bao gồm cả 4 phần chính của một hệ điều hành (shell, kernel, file system, utility) và các chương trình phục vụ người dùng,…
- Tất cả các chương trình trong bản phân phối đều theo bản quyền GPL.
- Hiện nay có rất nhiều công ty cung cấp các bản phân phối khác nhau (tham khảo ở http://www.linuxhq.com).

# Cách đánh số phiên bản

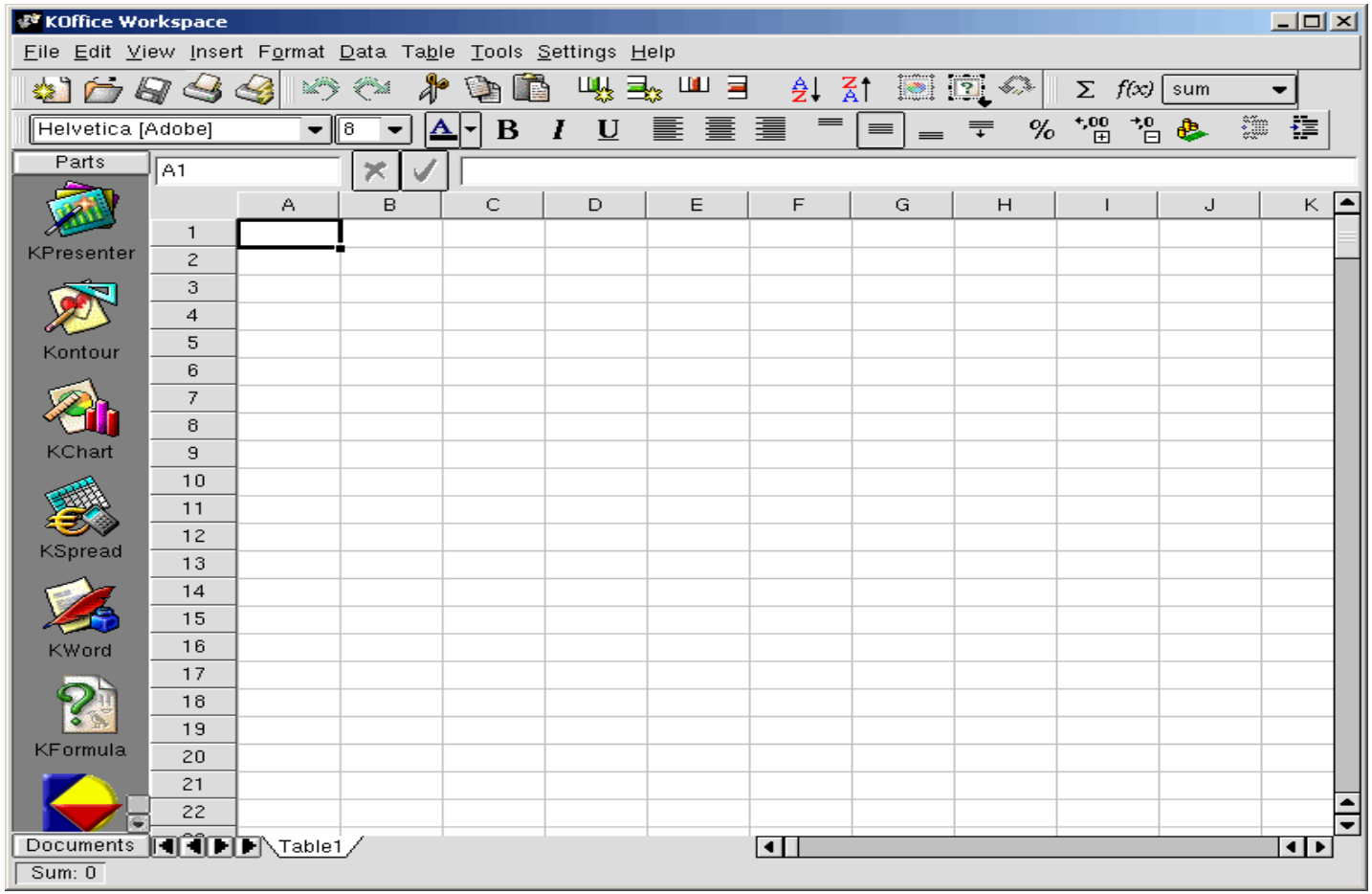
- Cần phân biệt số phiên bản của bản phân phối với số phiên bản của nhân.
- Nhân Linux hiện đang được điều hành và phát triển bởi Linus Torvalds, nên phiên bản của nhân tăng theo thứ tự, chứ không phân nhánh và nhân lên như các bản phân phối.
- Ví dụ:
  - Bản phân phối openSuSE Linux 10.1 (kernel 2.6.16.13)
  - Bản phân phối Fedora 5 (kernel 2.6.16.13)

# Một số ứng dụng trên Linux (1)

- Các ứng dụng cho văn phòng: OpenOffice, KOffice, StartOffice,…



# Một số ứng dụng trên Linux (2)



# Một số ứng dụng trên Linux (1)

- Các ứng dụng mạng và Internet:
  - WebServer, FTP Server, Mail Server: Apache, VsFTP, SendMail,…
  - WebBrowser, Instant Messenger: Firefox, GIMP,…
- Các công cụ phát triển:
  - C/C++
  - Java: Eclipse
  - Python
  - Perl, CGI
  - …

# Một số bản phân phối

- Redhat

http://www.redhat.com

- Fedora Core

http://fedora.redhat.com

- Debian


http://www.debian.org

- Mandrake


http://www.linux-mandrake.com

- TurboLinux


http://www.turbolinux.com

- SuSE


http://www.suse.com

# Redhat và Fedora Core

- Redhat và Fedora Core là hai bản Linux có lẽ là thịnh hành nhất trên thế giới, phát hành bởi công ty Redhat.
  - Từ 2003, Công ty Redhat phát triển Redhat Enterprise Linux (RHEL) với mục đích thương mại, nhằm vào các công ty, xí nghiệp.
  - Redhat cũng đầu tư mở ra dự án Fedora nhằm phát triển phiên bản Fedora Core cho người dùng bình thường.
  - Bản Linux của RedHat cuối cùng dừng ở phiên bản 9.0. Phiên bản của Fedora được bắt đầu từ 1. Hiện nay đã có Fedora Core 6.

# SuSE Linux

- Made in Germany (Đức)
- Bản Linux cực kỳ thịnh hành ở châu Âu và Bắc Mỹ.
- Năm 2003, công ty SuSE bị Novell mua. Novell đang dốc sức đầu tư cho SuSE để nhắm vào enterprise users cạnh tranh với Redhat.
- Bản SuSE mới nhất hiện nay là 9.1

- Web site: http://www.suse.com

# Debian Linux

- Là bản phân phối Linux cũng rất phổ biến.
- Nhiều người có ý kiến cho rằng:
  - Người không chuyên nên dùng Fedora Core để có thể làm quen với những kỹ thuật mới.
  - Người chuyên nghiệp nên dùng Debian vì sự ổn định tuyệt vời của nó.
- Bản mới nhất 3.0R2

- Web site: http://www.debian.org

# Mandrake Linux

- Made in France
- Cũng rất thịnh hành ở châu Âu, Mỹ và Việt Nam. Đây là bản được ưu ái nhất trong vấn đề Việt hóa.
- Bản mới nhất hiện nay là 10.0

- Web site: http://www.mandrakelinux.com

# Turbo Linux

- Nổi tiếng tại Nhật, Trung Quốc.
- Công ty Turbo đang đầu tư mạnh để thống trị thị trường Trung Quốc
- Bản Turbo mới nhất hiện nay là 10F

# Knoppix Linux

- Made in Germany
- Bản live Linux cũng được ưa chuộng hiện nay.
- Cho phép khởi động trực tiếp từ CD mà không cần cài đặt vào ổ cứng.
- Phiên bản mới nhất là 3.4

# Vietkey Linux

- Made in Vietnam
- Hoàn toàn không nổi bật sau khi đạt giải cuộc thi TTVN 2003.
- Phát triển bởi nhóm Vietkey trên nền Redhat 7.2

# Tóm lại

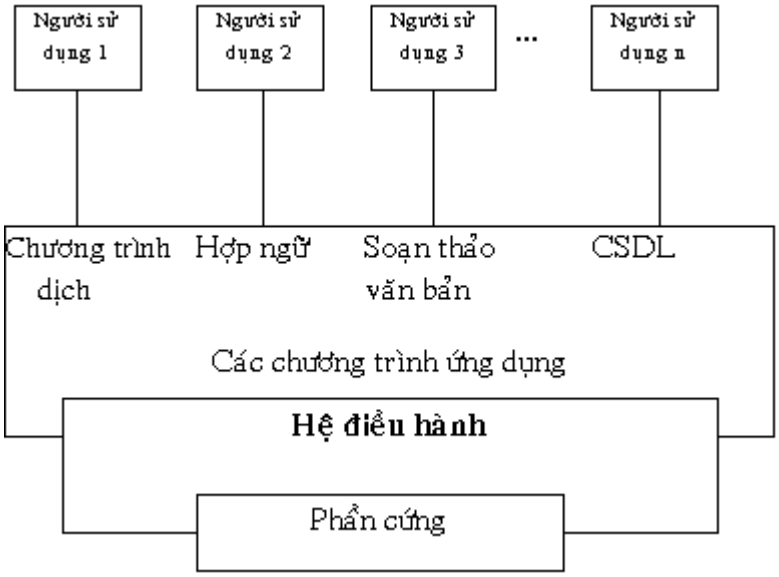
# Mục lục

# 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)

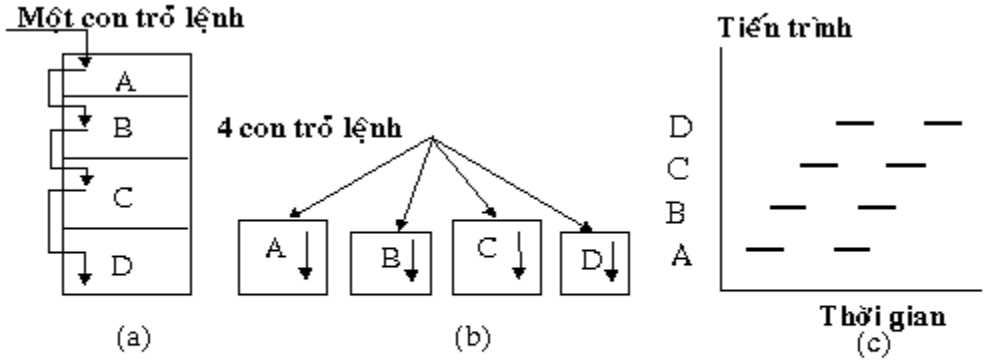
Để hỗ trợ sự đa chương, máy tính phải có khả năng thực hiện nhiều tác vụ đồng thời. Nhưng việc điều khiển nhiều hoạt động song song ở cấp độ phần cứng là rất khó

khăn. Vì thế các nhà thiết kế hệ điều hành đề xuất một mô hình *song song gỉa lặp* 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)

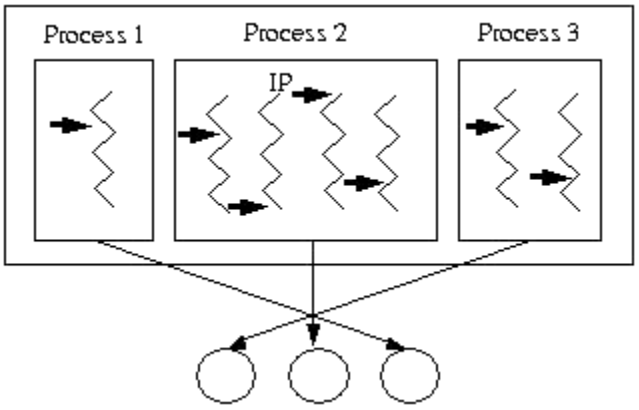
Trong hầu hết các hệ điều hành, **mỗi tiến trình có một không gian địa chỉ và chỉ có một dòng xử lý**. Tuy nhiên, có nhiều tình huống người sử dụng mong muốn có nhiều dòng xử lý cùng chia sẻ một không gian địa chỉ, và các dòng xử lý này hoạt động song song tương tự như các tiến trình phân biệt (ngoại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

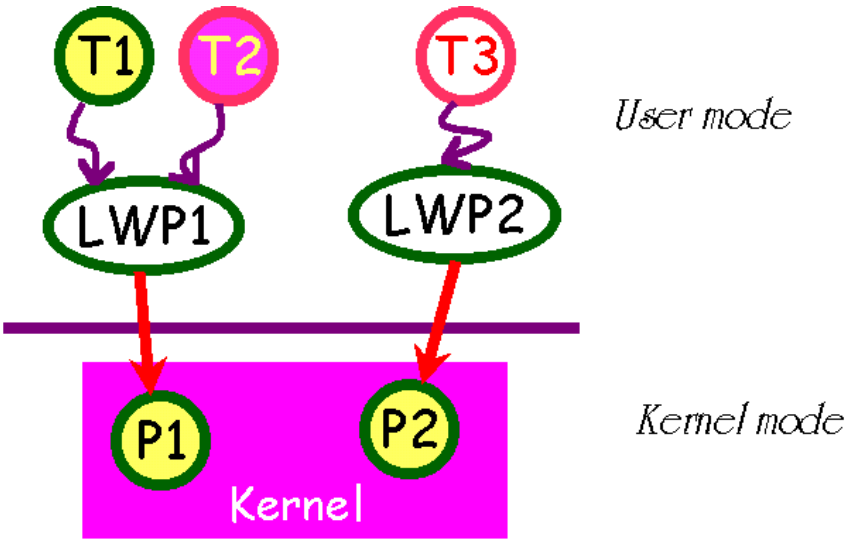
| Tiến trình |
| --- |
| Không gian địa chỉ |
| Tài nguyên toàn cục |
| Các thông tin thống kê |

| 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

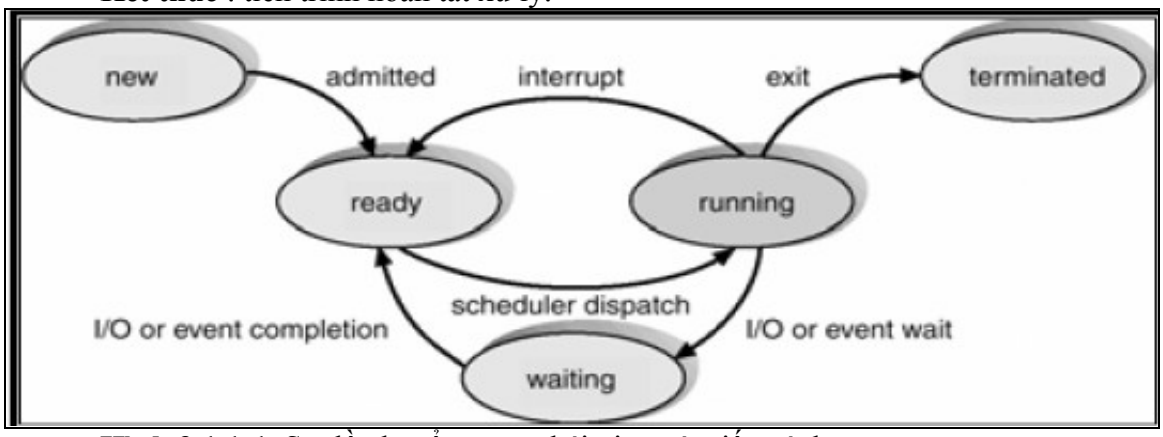
## 3.1. Tổ chức quản lý tiến trình

### 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.**

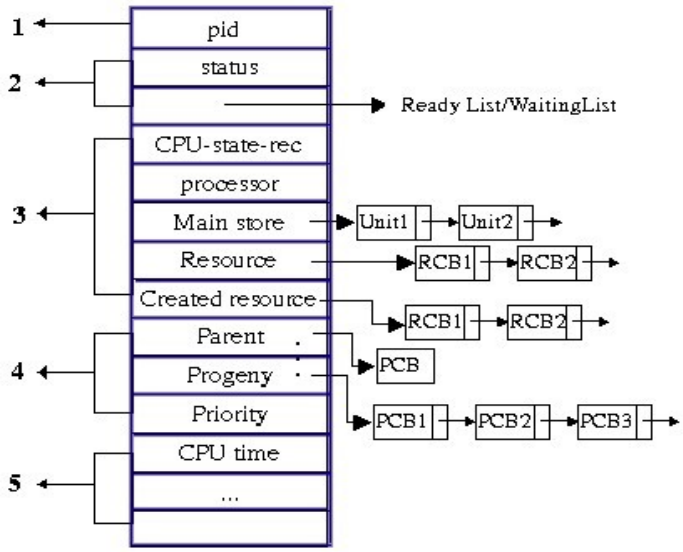

**Hình 3.1.2-1.** Hai chế độ xử lý

### 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.



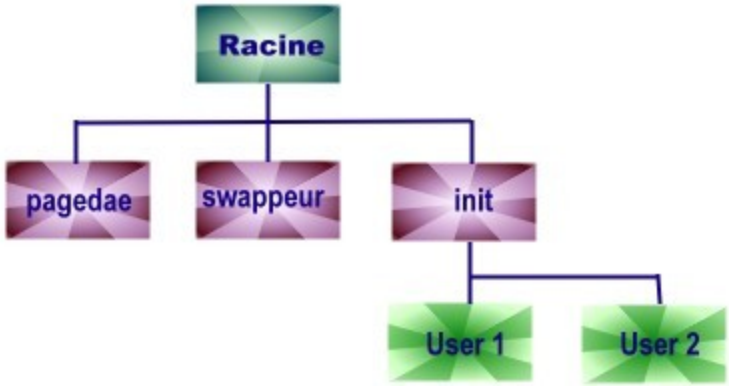

**Hình 3.1.3-1.** Khối điều khiển tiến trình

## 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.

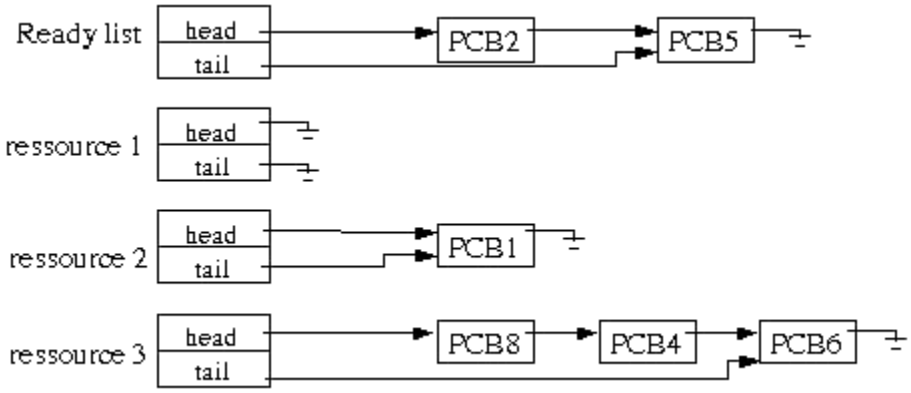
Đối với các hệ thống tương tác(time sharing), các hệ thời gian thực (real time),cần phải sử dụng nguyên lý điều phối không độc quyền để các tiến trình quan trọng có cơ hội hồi đáp kịp thời. Tuy nhiên thực hiện điều phối theo nguyên lý không độc quyền đòi hỏi những cơ chế phức tạp trong việc phân định 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**.

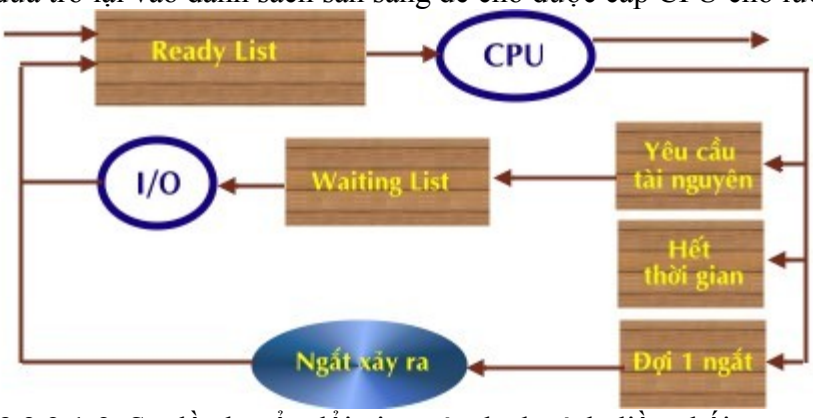



**Hình 3.2.2.1-1.** Các danh sách điều phối

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**.

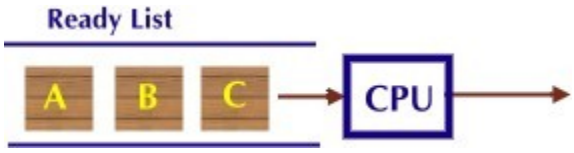
Trong nhiều hệ điều hành, có thể không có bộ điều phối tác vụ hoặc tách biệt rất ít đối với bộ điều phối tiến trình. Một vài hệ điều hành lại đưa ra một cấp độ điều phối trung gian kết hợp cả hai cấp độ điều phối tác vụ và tiến trình


**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.

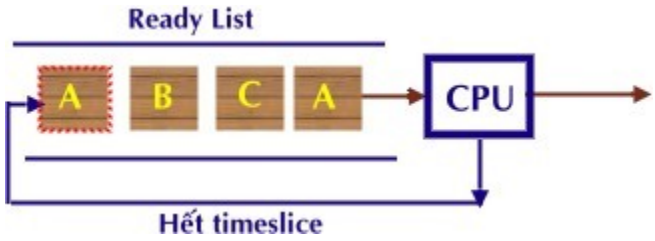
- **Thảo luận** : *Thời gian chờ trung bình không đạt cực tiểu, và biến đổi đáng kể đối với các giá trị về thời gian yêu cầu xử lý và thứ tự khác nhau của các tiến trình trong danh sách sẵn sàng. Có thể xảy ra hiện tượng tích lũy thời gian chờ, khi các tất cả các tiến trình (có thể có yêu cầu thời gian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

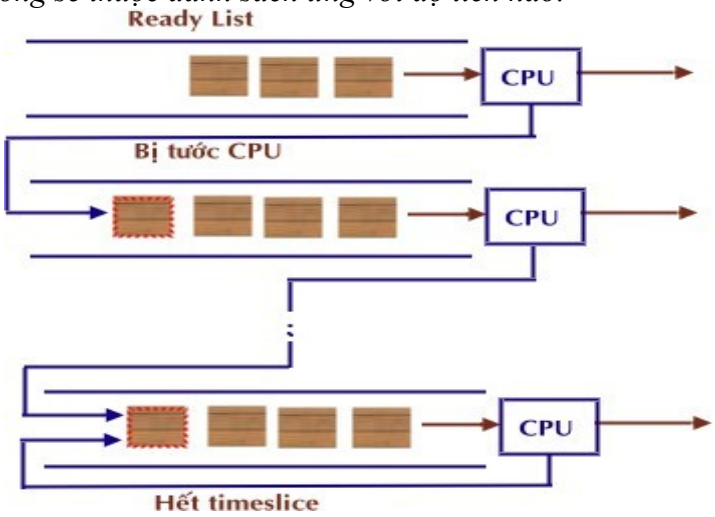
### *3.2.3.5. Chiến lược điều phối với nhiều mức độ ưu tiên*

- **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 |

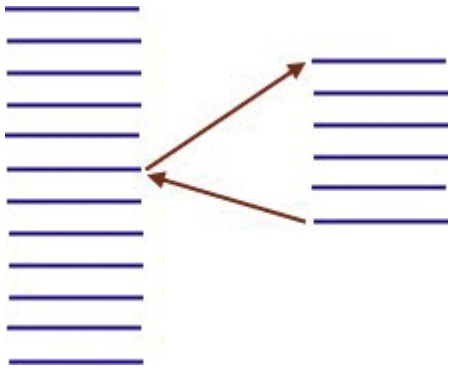
Mỗi tiến trình sỡ hữu một bảng biểu diễn các tín hiệu khác nhau. Với mỗi tín hiệu sẽ có tương ứng một trình xử lý tín hiệu *(signal handler)* qui định các xử lý của tiến trình khi nhận được tín hiệu tương ứng.

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.



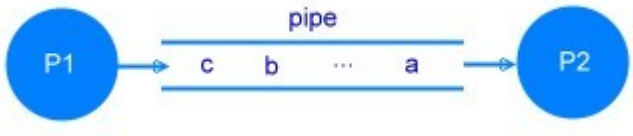
**Hình 4.2.1-1.** Liên lạc bằng tín hiệu

**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 !

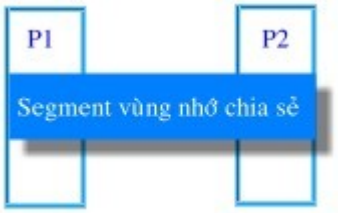
Cơ chế này cho phép truyền dữ liệu với cách thức không cấu trúc.

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*****:**

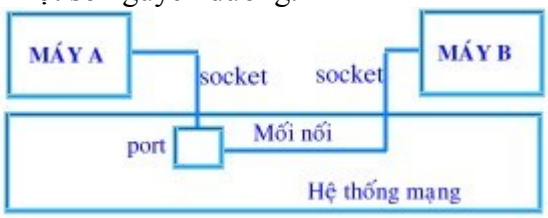
Một liên kết được thành lập giữa hai tiến trình. Trước khi mối liên kết này được thiết lập, một trong hai tiến trình phải đợi có một tiến trình khác yêu cầu kết nối.Có thể sử dụng socket để liên lạc theo mô hình client-serveur. Trong mô hình này, server sử dụng

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.

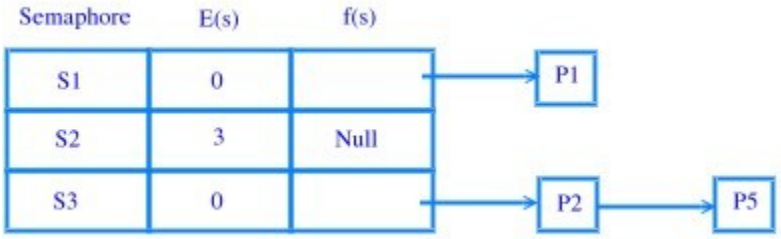
### *4.4.2.1. Semaphore*

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 !

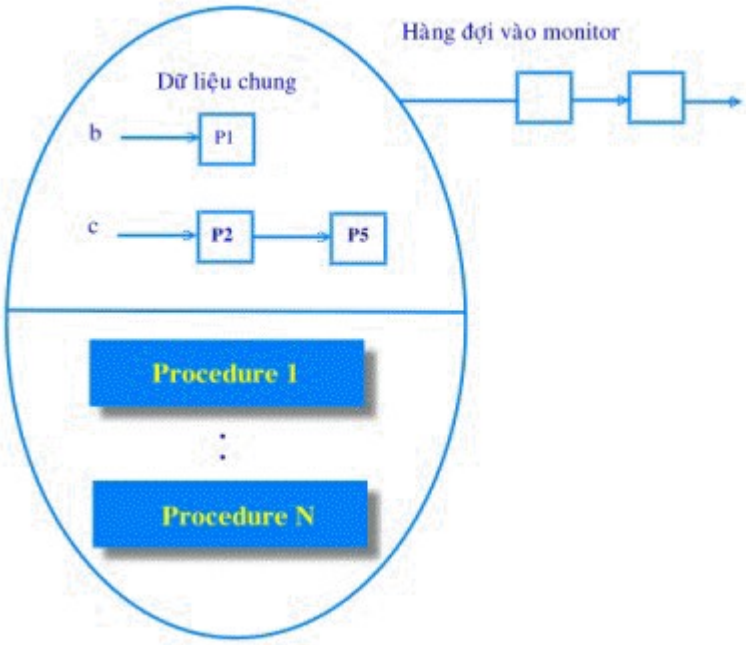
Vì thế việc sử dụng đúng cách semaphore để đồng bộ hóa phụ thuộc hoàn toàn vào lập trình viên và đòi hỏi lập trình viên phải hết sức thận trọng.

### *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

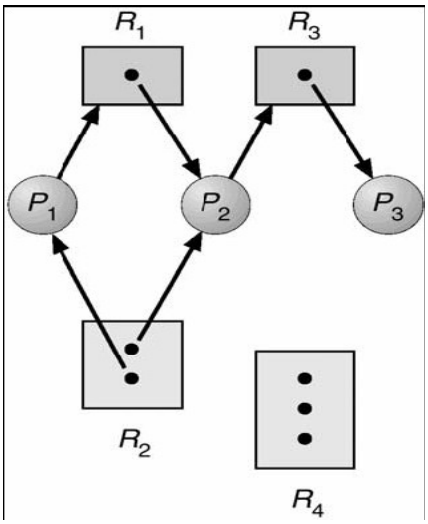
E. Một tập các đỉnh V được chia làm hai loại nút P = {P1, P2,…,Pn} là tập hợp các quá trình hoạt động trong hệ thống, và R = {R1, R2, ..., Rm} là tập hợp chứa tất cả các loại tài nguyên trong hệ thống.

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 $R_3$

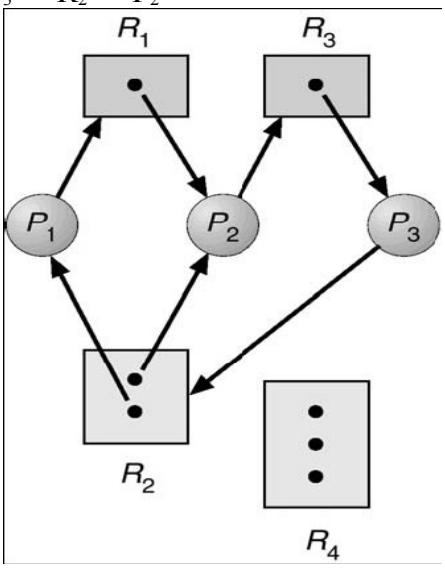
Đồ thị cấp phát tài nguyên hiển thị rằng, nếu đồ thị không chứa chu trình, thì không có quá trình nào trong hệ thống bị deadlock. Nếu đồ thị có chứa chu trình, thì deadlock có thể tồn tại. Nếu mỗi loại tài nguyên có chính xác một thể hiện, thì một chu trình ngụ ý rằng một deadlock xảy ra. Nếu một chu 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:

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

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

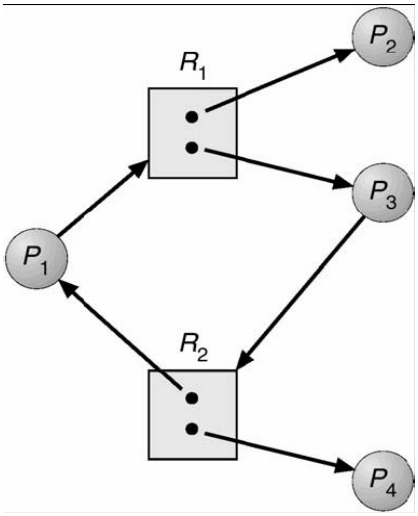



**Đồ thị cấp phát tài nguyên với deadlock**

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.

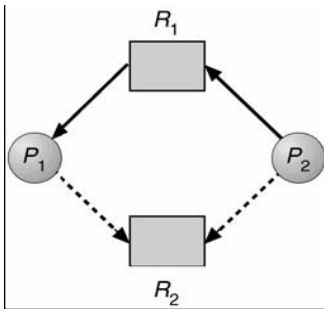
### 5.5.2. Giải thuật đồ thị cấp phát tài nguyên

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.

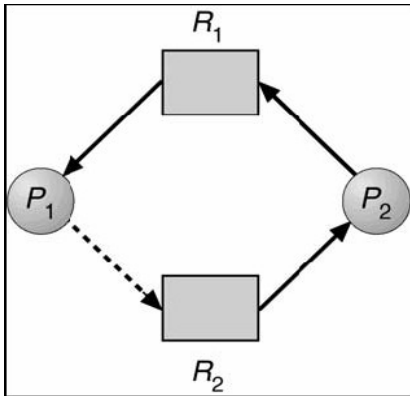
Giả sử rằng Pi yêu cầu tài nguyên Rj. Yêu cầu có thể được gán chỉ nếu chuyển cạnh yêu cầu Pi → Rj tới cạnh gán Rj→Pi không dẫn đến việc hình thành chu trình trong đồ thị cấp phát tài nguyên. Chú ý rằng chúng ta kiểm tra tính an toàn bằng cách dùng giải thuật phát hiện chu trình. Một giải thuật để phát 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.

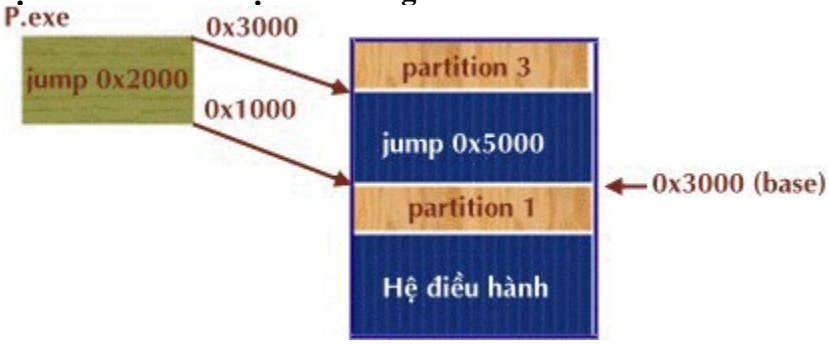
Chương trình của người sử dụng chỉ thao tác trên các địa chỉ ảo, không bao giờ nhìn thấy các địa chỉ vật lý . Địa chỉ thật sự ứng với vị trí của dữ liệu trong bô nhớ chỉ được xác định khi thực hiện truy xuất đến dữ liệu.

## 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**.

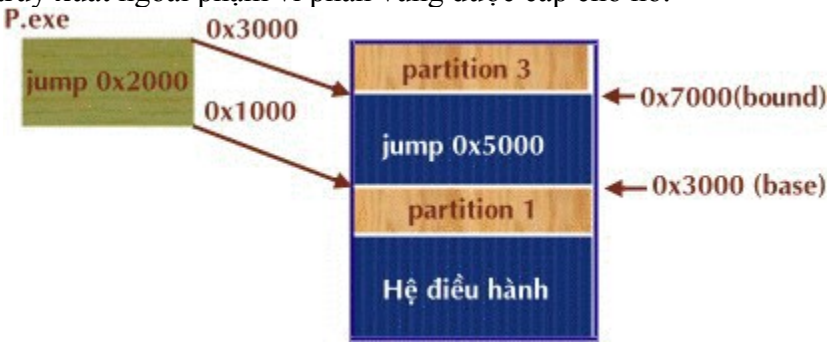


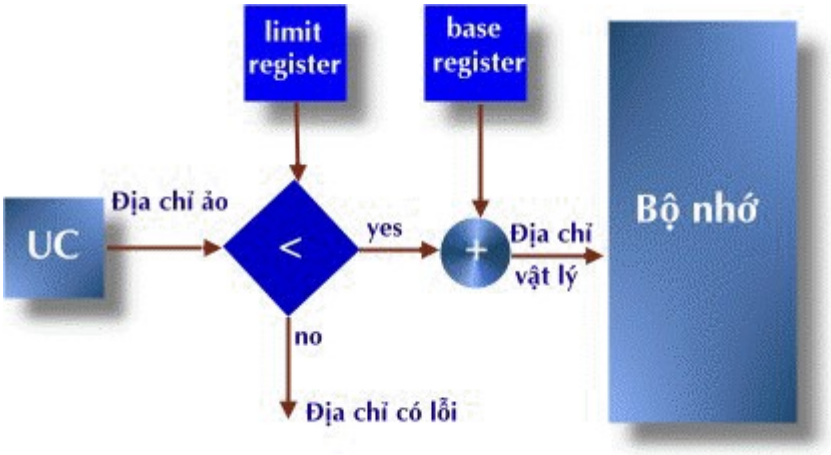

**Thảo luận:**

- 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.

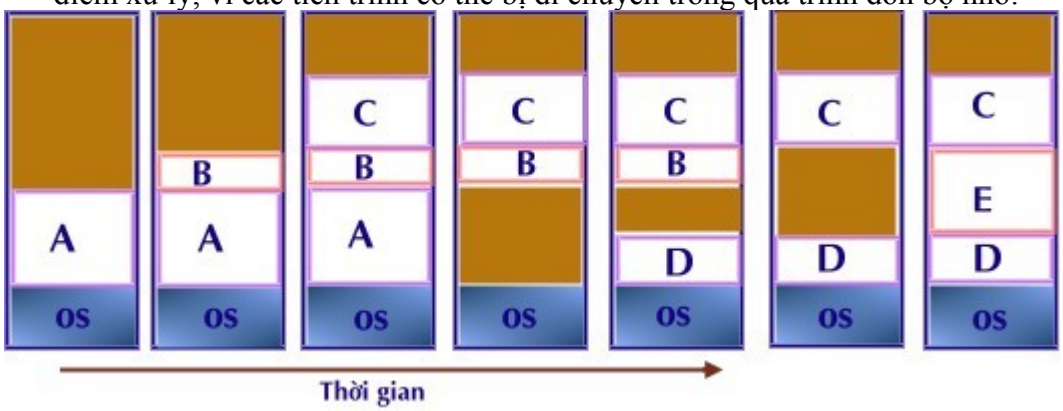





**Hai thanh ghi hỗ trợ chuyển đổi địa chỉ**

**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:

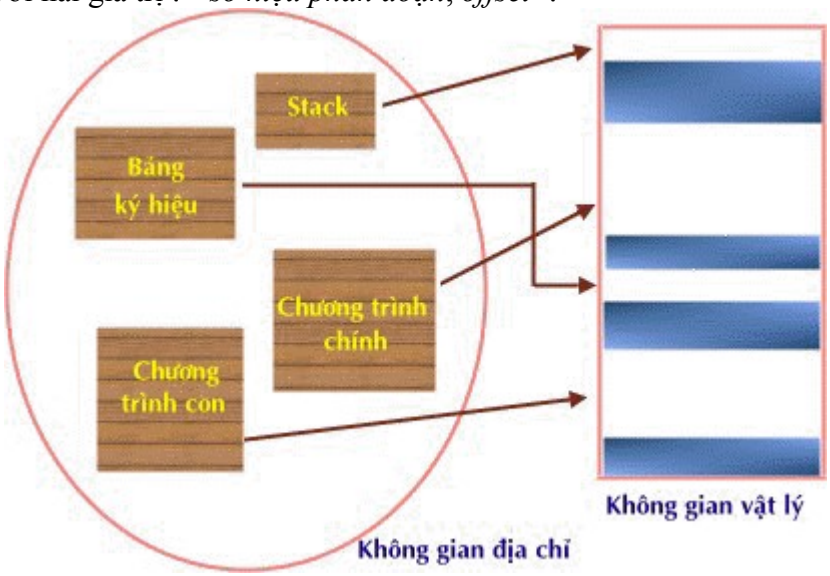
- Dời chỗ tiến trình: di chuyển tiến trình đến một vùng nhớ khác đủ lớn để thỏa mãn nhu cầu tăng trưởng của tiến trình.

- 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:**

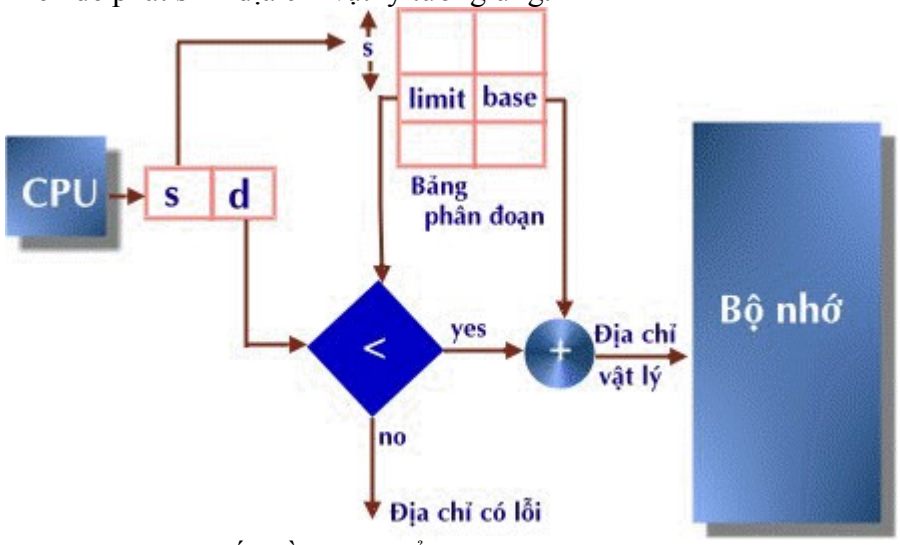
Cần phải xây dựng một ánh xạ để chuyển đổi các địa chỉ 2 chiều được người dùng định nghĩa thành địa chỉ vật lý một chiều. Sự chuyển đổi này được thực hiện qua một *bảng phân đoạn*. Mỗi thành phần trong bảng phân đoạn bao gồm một *thanh ghi nền* và

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.

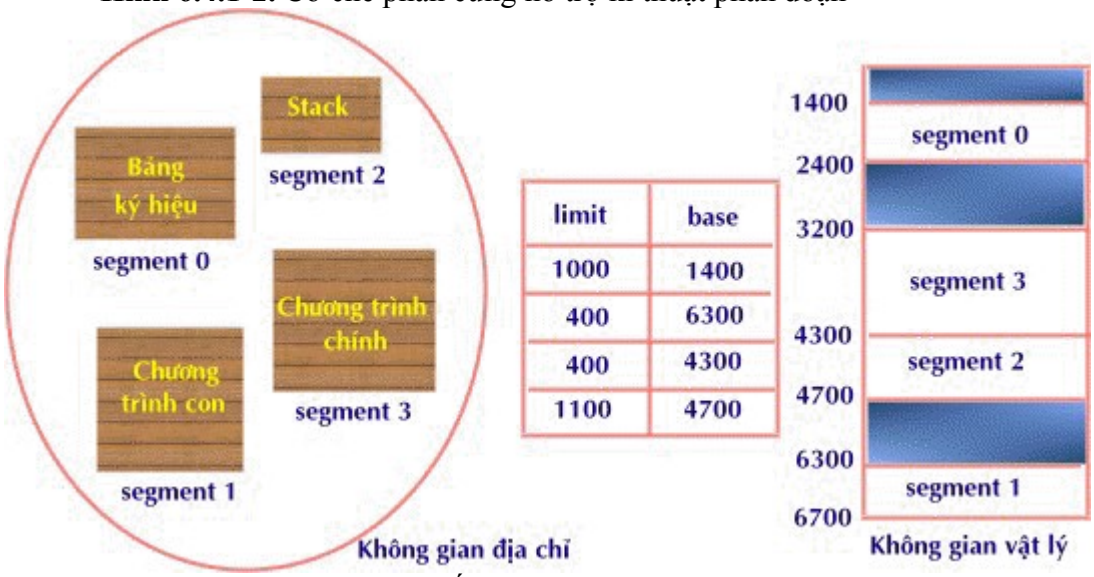
**Chuyển đổi địa chỉ:**

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

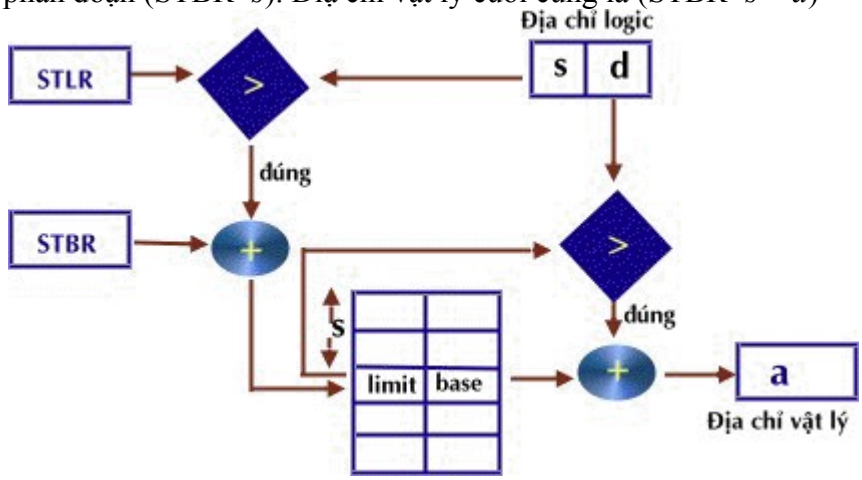



**Hình 6.4.1-3.** Hệ thống phân đoạn

**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*)

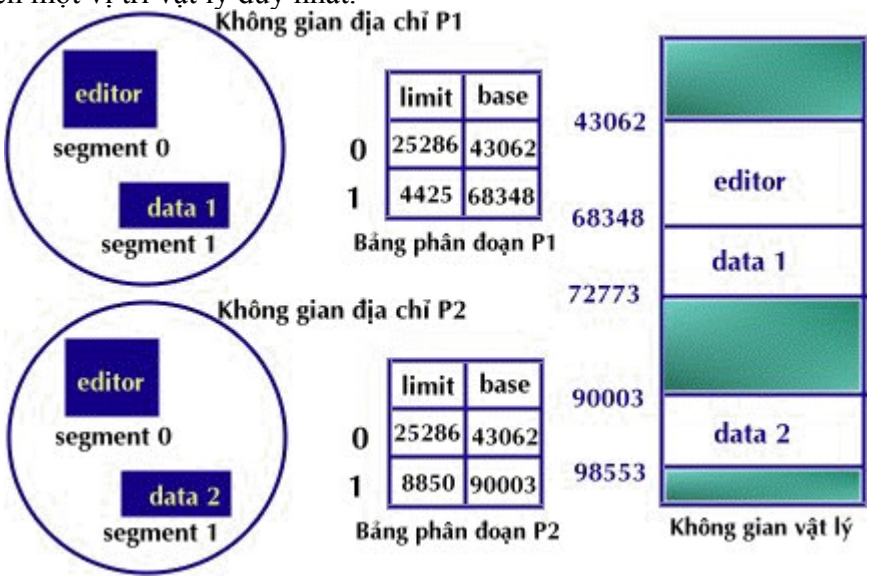



**Hình 6.4.1-4.** Sử dụng STBR, STLR và bảng phân đoạn

**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.

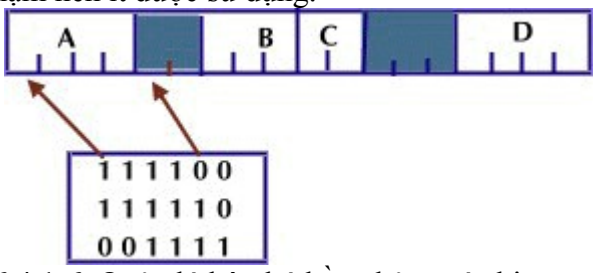
**Chia sẻ phân đoạn**: Một ưu điểm khác của kỹ thuật phân đoạn là khả năng chia sẻ ở mức độ phân đoạn. Nhờ khả năng này, các tiến trình có thể chia sẻ với nhau từng phần chương trình ( ví dụ các thủ tục, hàm), không nhất thiết phải chia sẻ toàn bộ chương trình như trường hợp phân trang. Mỗi tiến trình 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

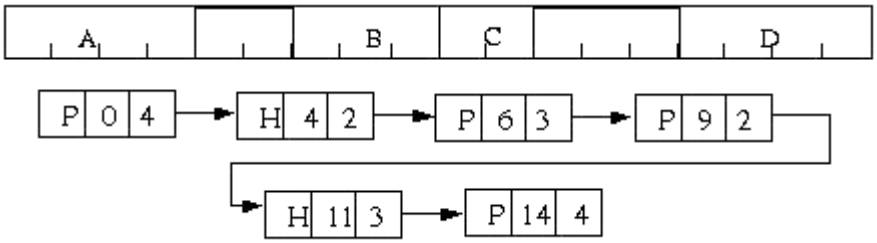
**Thảo luận**:

- 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.

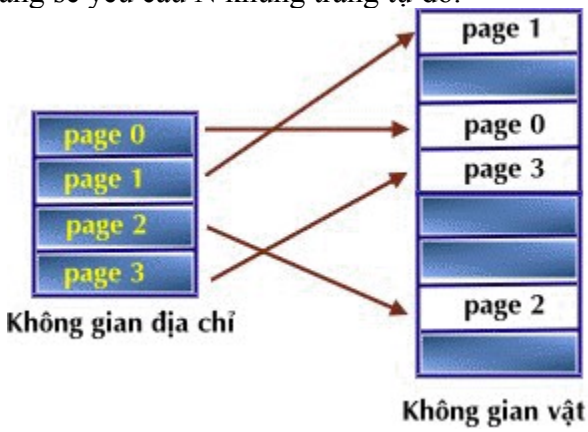
Trong hệ thống sử dụng kỹ thuật phân đoạn , hiện tượng phân mảnh ngoại vi lại xuất hiện khi các khối nhớ tự do đều quá nhỏ, không đủ để chứa một phân đoạn.

## 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:**

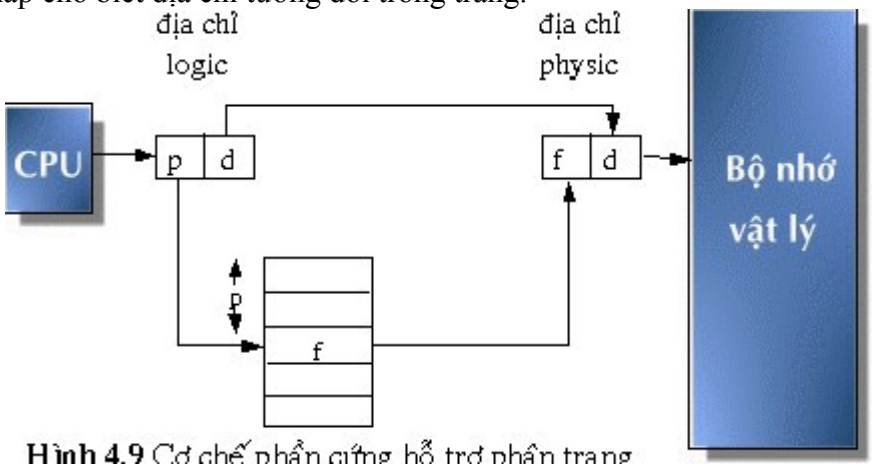
Cơ chế phần cứng hỗ trợ thực hiện chuyển đổi địa chỉ trong cơ chế phân trang là bảng trang (*pages table*). Mỗi phần tử trong bảng trang cho biết các địa chỉ bắt đầu của vị trí lưu trữ trang tương ứng trong bộ nhớ vật lý ( số hiệu khung trang trong bộ nhớ vật lý đang chứa trang ).

**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.



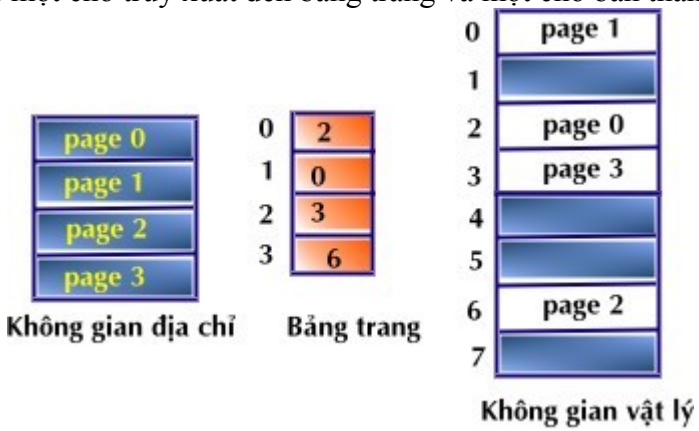

**Hình 6.4.2-2.** Cơ chế phần cứng hỗ trợ phân trang

**Cài đặt bảng trang**:

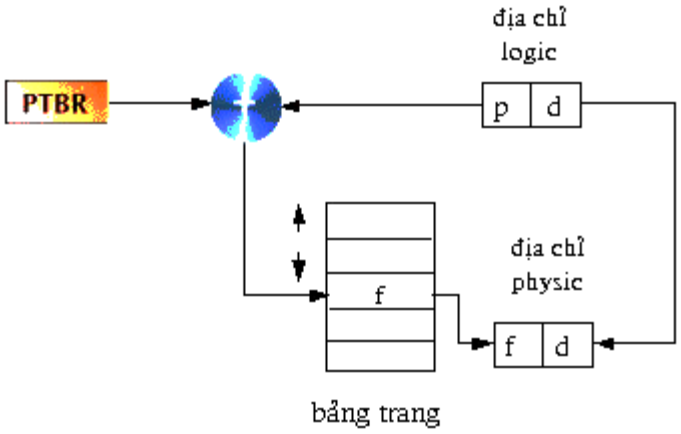
Trong trường hợp đơn giản nhất, bảng trang một tập các thanh ghi được sử dụng để cài đặt bảng trang. Tuy nhiên việc sử dụng thanh ghi chỉ phù hợp với các bảng trang

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!



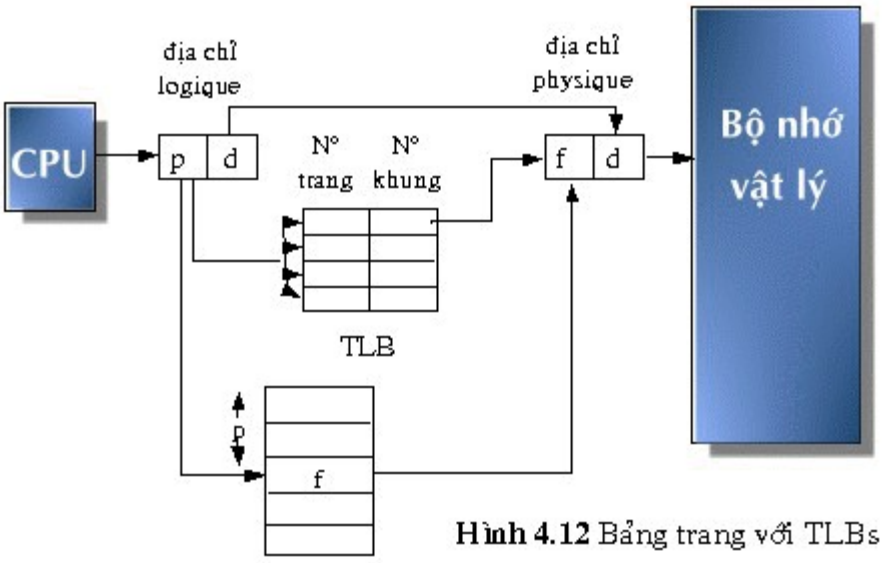

**Hình 6.4.2-3.** Mô hình bộ nhớ phân trang




**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.



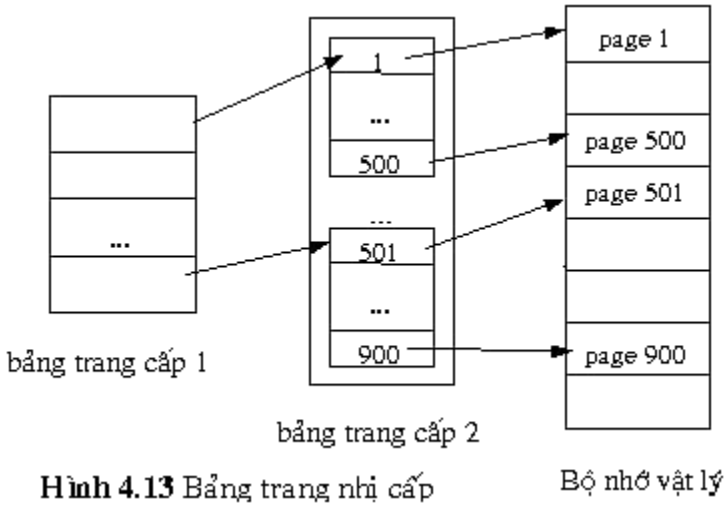

**Hình 4.12** Bảng trang với TLBs

**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

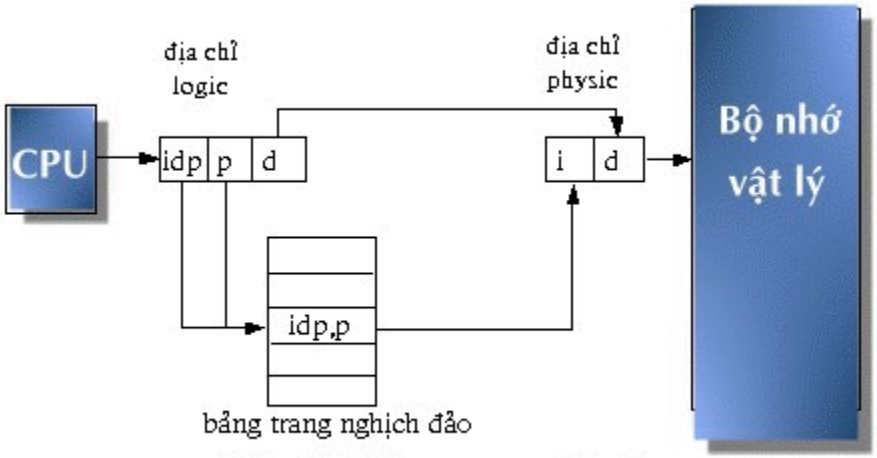
**Hình 6.4.2-6**. Bảng trang cấp 2

*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).

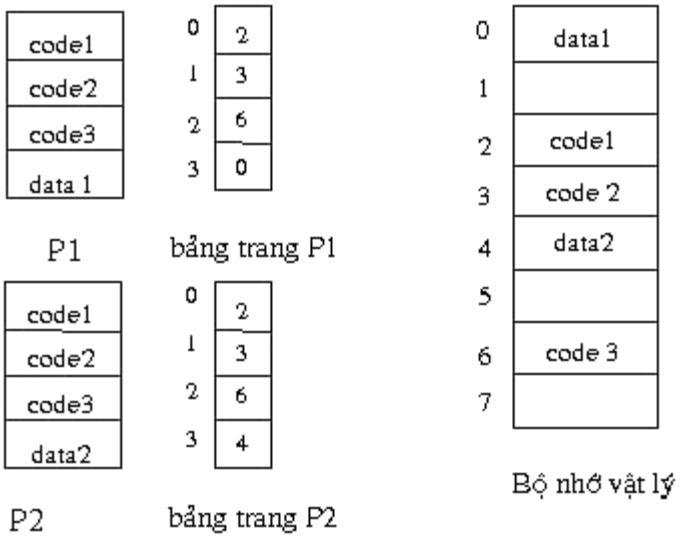
- *Hợp lệ* : trang tương ứng thuộc về không gian địa chỉ của tiến trình.
- *Bất hợp lệ* : trang tương ứng không nằm trong không gian địa chỉ của tiến trình, điều này có nghĩa tiến trình đã truy xuất đến một địa chỉ không được phép.

| 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

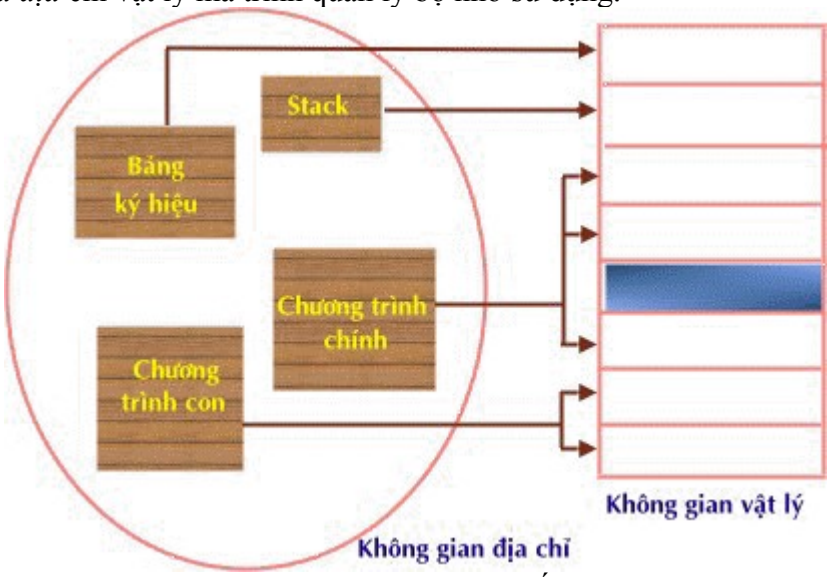
Kỹ thuật phân đoạn thõa mãn được nhu cầu thể hiện cấu trúc logic của chương trình nhưng nó dẫn đến tình huống phải cấp phát các khối nhớ có kích thước khác nhau cho các phân đoạn trong bộ nhớ vật lý. Điều này làm rắc rối vấn đề hơn rất nhiều so với việc

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*.

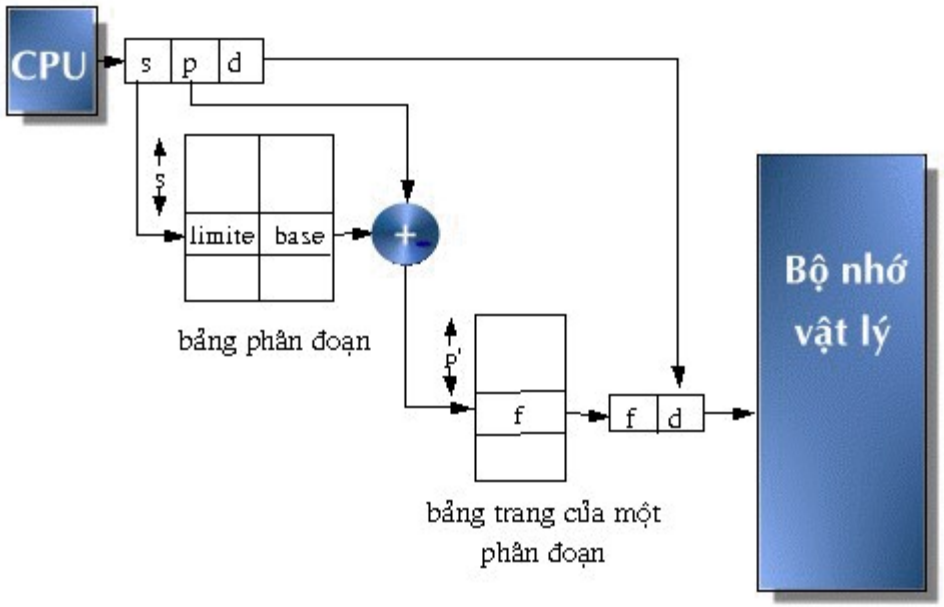
### 6.4.3. Phân đoạn kết hợp phân trang (Paged segmentation)

**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.

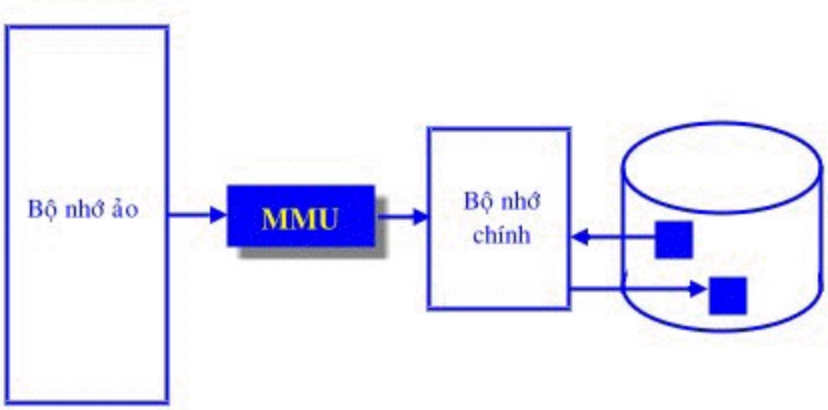
Để giải phóng lập trình viên khỏi các suy tư về giới hạn của bộ nhớ, mà cũng không tăng thêm khó khăn cho công việc lập trình của họ, người ta nghĩ đến các kỹ thuật tự động, cho phép xử lý một chương trình có kích thước lớn chỉ với một vùng nhớ có kích thước nhỏ . Giải pháp được tìm thấy với khái niệm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

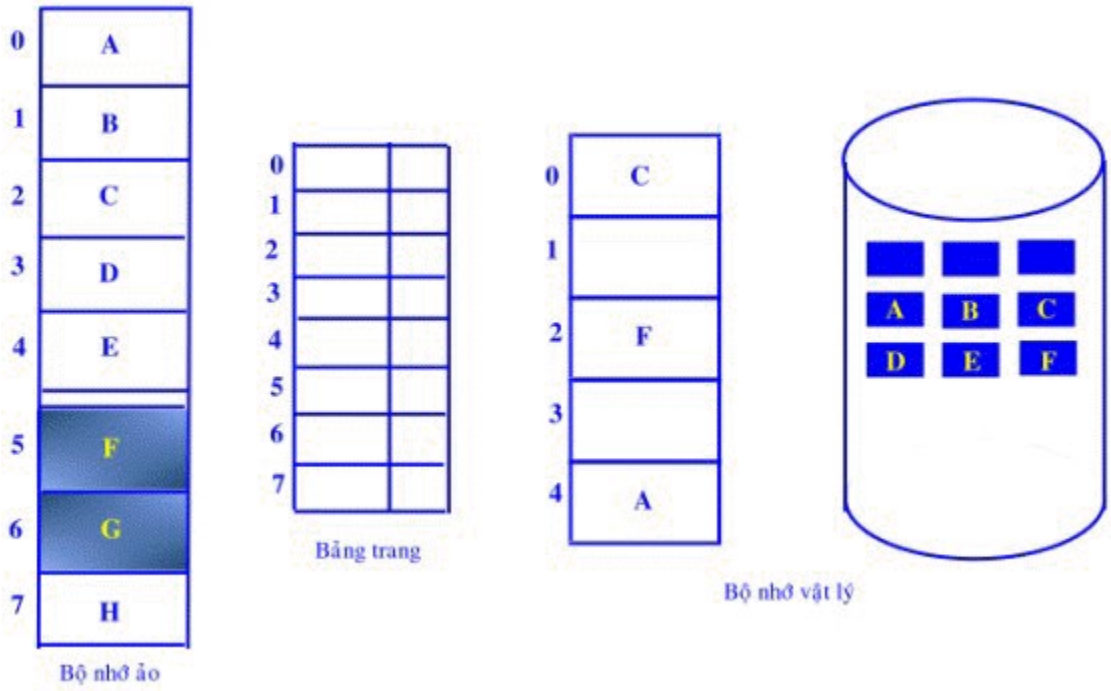
- *valid* : trang tương ứng là hợp lệ và đang ở trong bộ nhớ chính .
- *invalid* : hoặc trang bất hợp lệ (không thuộc về không gian địa chỉ của tiến trình) hoặc trang hợp lệ nhưng đang được lưu trên bộ nhớ phụ.

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.*

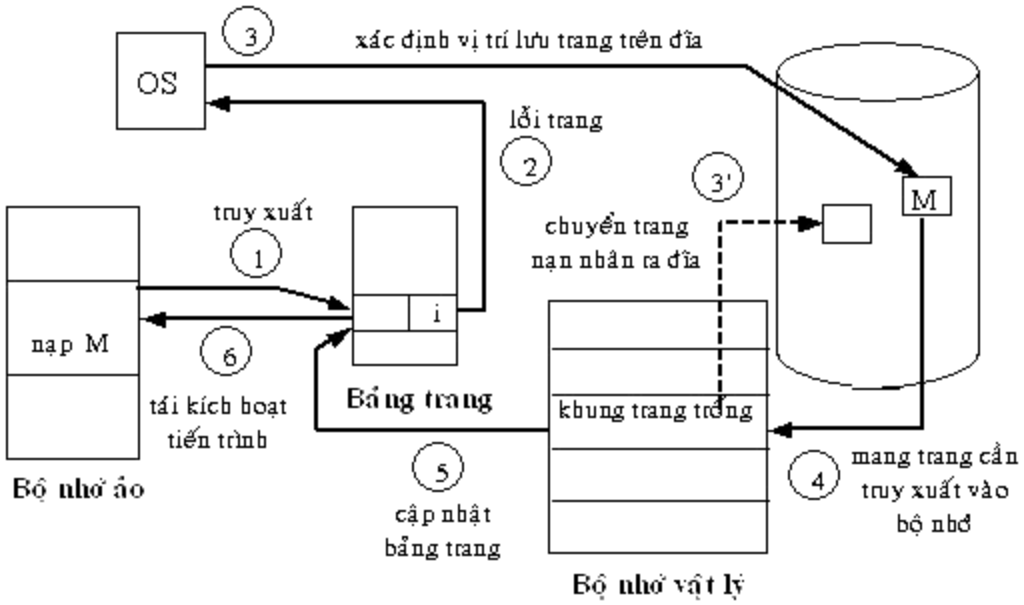



**Hình 6.5.1.2-1.** Bảng trang với một số trang trên bộ nhớ phụ

**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 »**

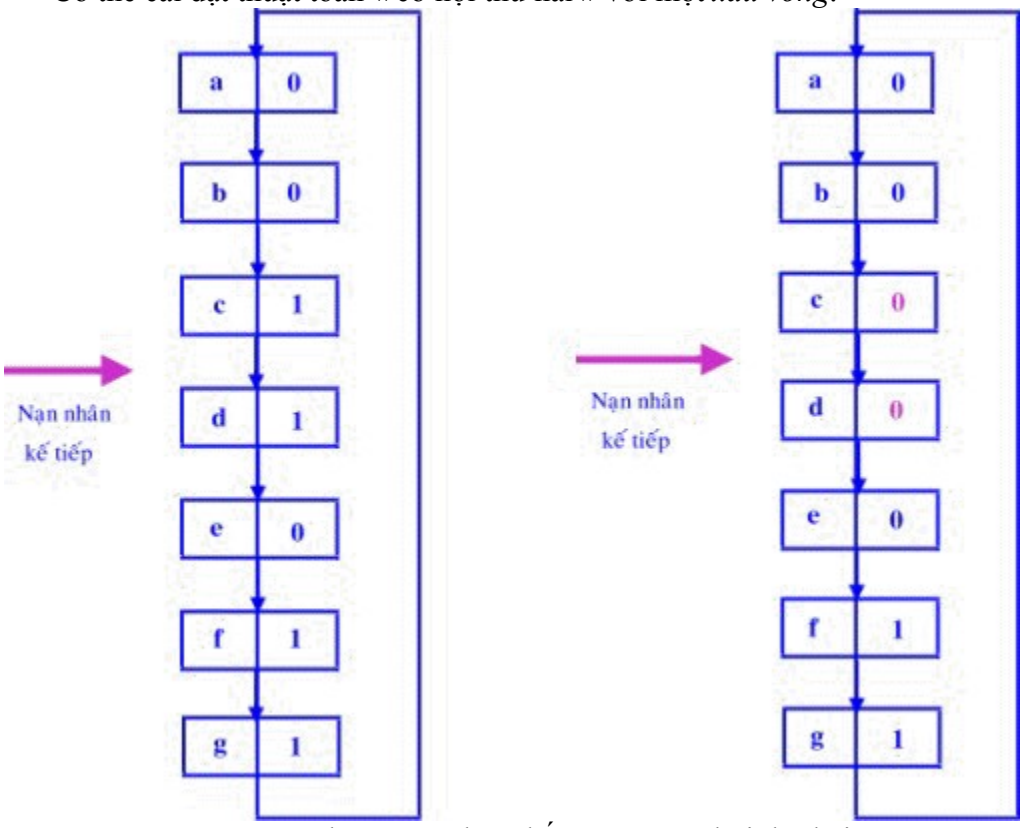
Tiếp cận: Sử dụng một bit reference duy nhất. Thuật toán cơ sở vẫn là FIFO, tuy nhiên khi chọn được một trang theo tiêu chuẩn FIFO, kiểm tra bit reference của trang đó :

- 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.

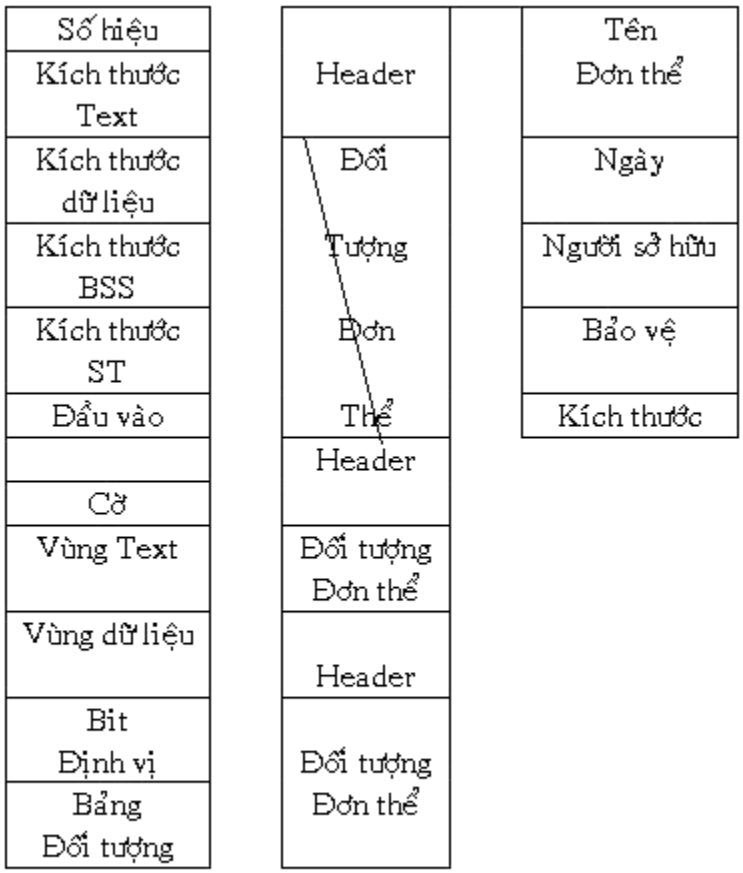
- *Tập tin thường* : là tập tin text hay tập tin nhị phân chứa thông tin của người sử dụng.
- *Thư mục* : là những tập tin hệ thống dùng để lưu giữ cấu trúc của hệ thống tập tin.
- *Tập tin có ký tự đặc biệt* : liên quan đến nhập xuất thông qua các thiết bị nhập xuất tuần tự như màn hình, máy in, mạng.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

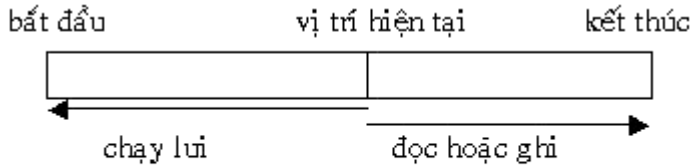
***Truy xuất tập tin :***

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 :***

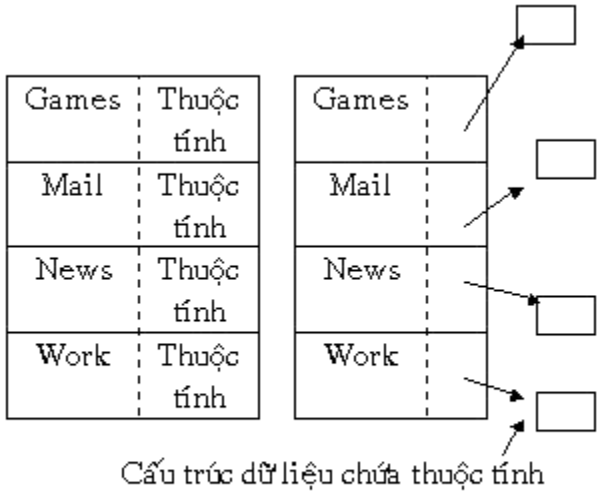
Một thư mục thường thường chứa một số *entry*, mỗi entry cho một tập tin. Mỗi entry chứa tên tập tin, thuộc tính và địa chỉ trên đĩa lưu dữ liệu hoặc một entry chỉ chứa tên tập tin và một con trỏ, trỏ tới một cấu trúc, trên đó có thuộc tính và vị trí lưu trữ của tập tin.

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.

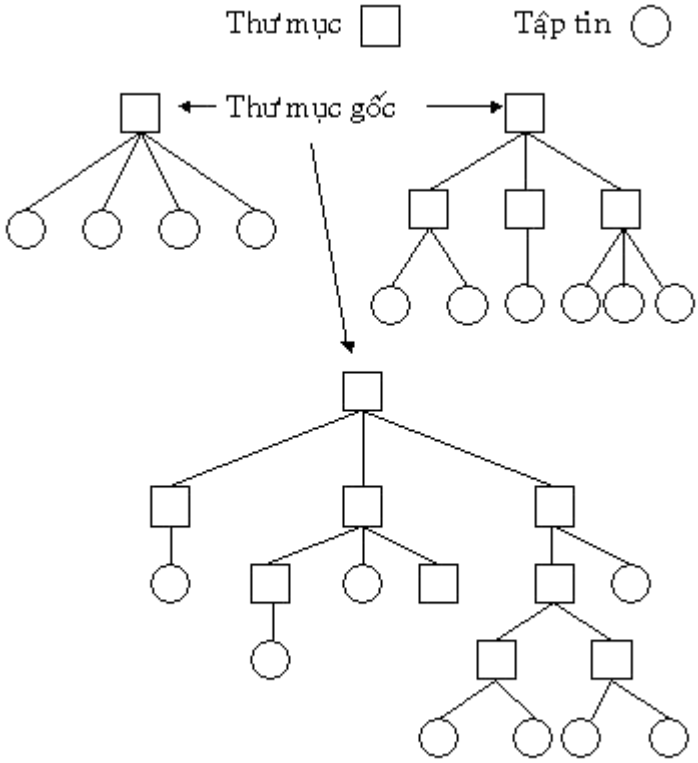
Số lượng thư mục trên mỗi hệ thống là khác nhau. Thiết kế đơn giản nhất là hệ thống chỉ có thư mục đơn(còn gọi là thư mục một cấp), chứa tất cả các tập tin của tất cả người dùng, cách này dễ tổ chức và khai thác nhưng cũng dễ gây ra khó khăn khi có nhiều người sử dụng vì sẽ có nhiều tập tin trùng tên.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

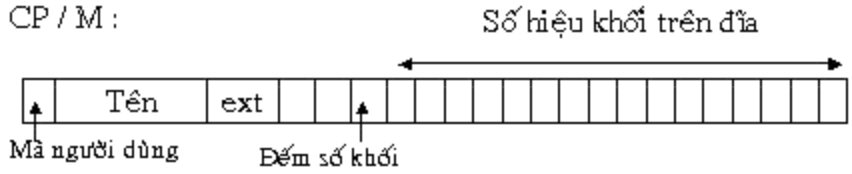
## 7.3.1. Bảng quản lý tệp tin, thư mục

### *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*

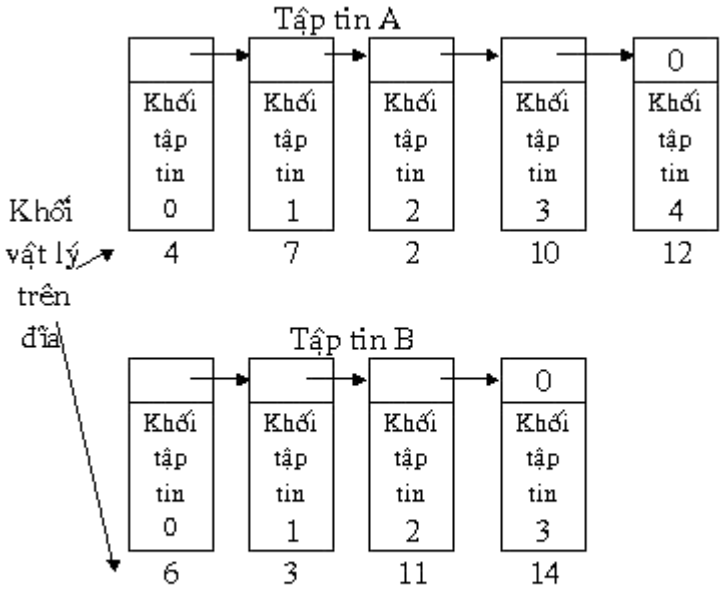
**Định vị liên tiếp:**

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:**



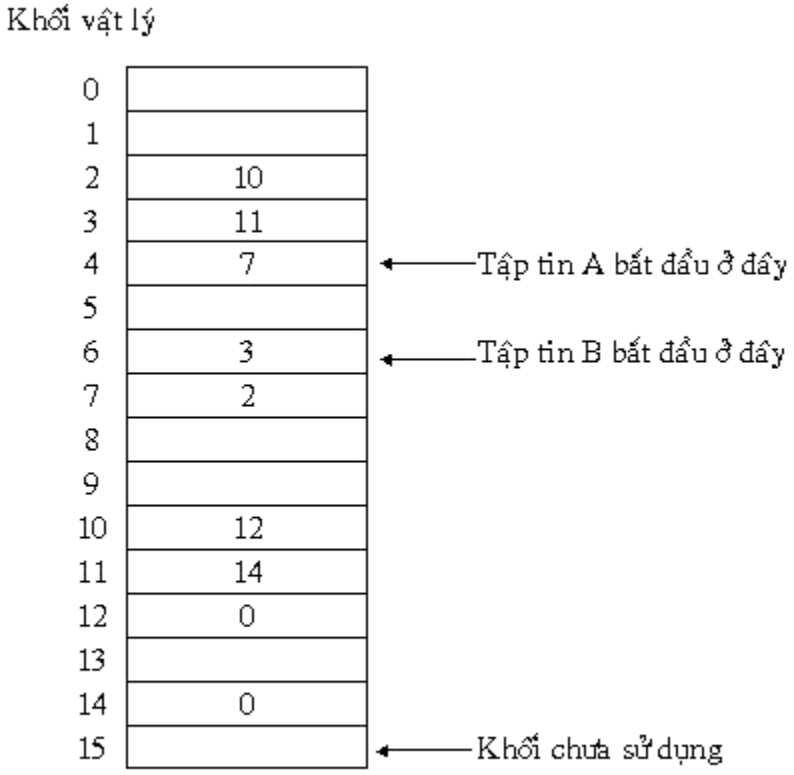

**Hình 9.2** Định vị bằng danh sách liên kết

**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

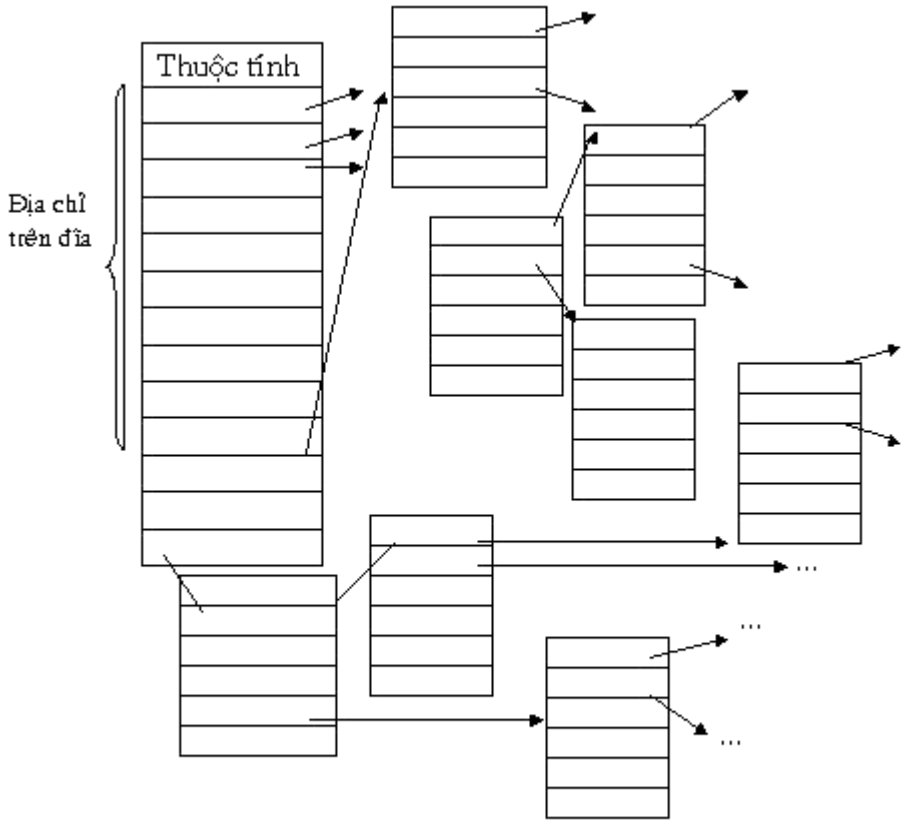
**Hình 7.3.2.2-2.** Bảng chỉ mục của danh sách liên kết

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

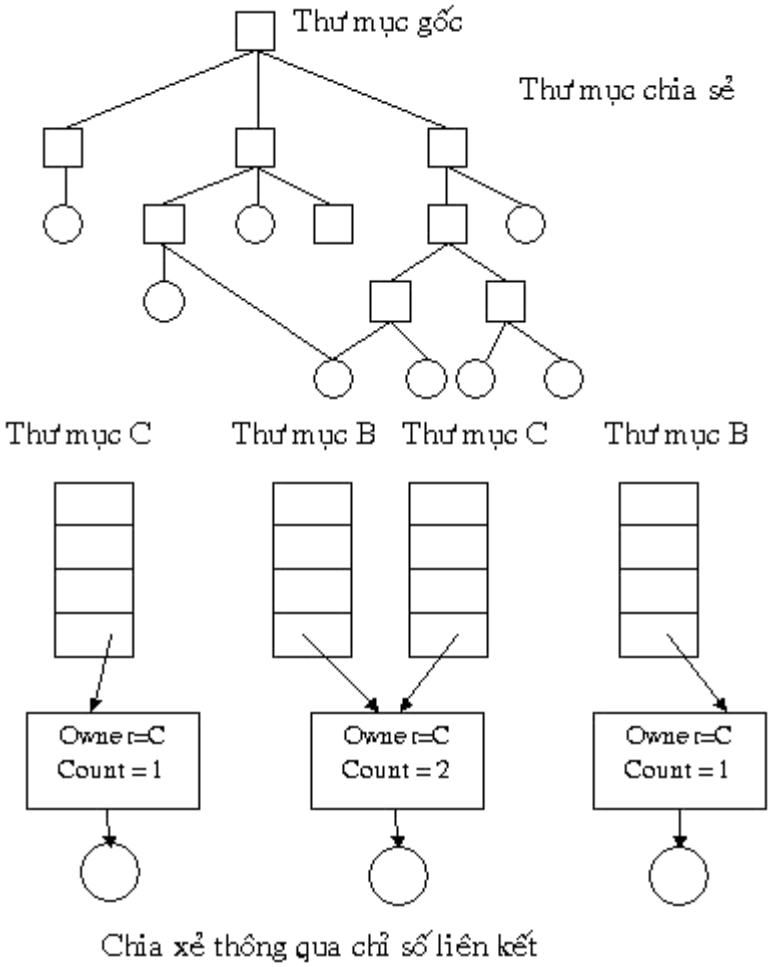
**Hình 7.3.2.2-3.** Cấu trúc I-node

## 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.

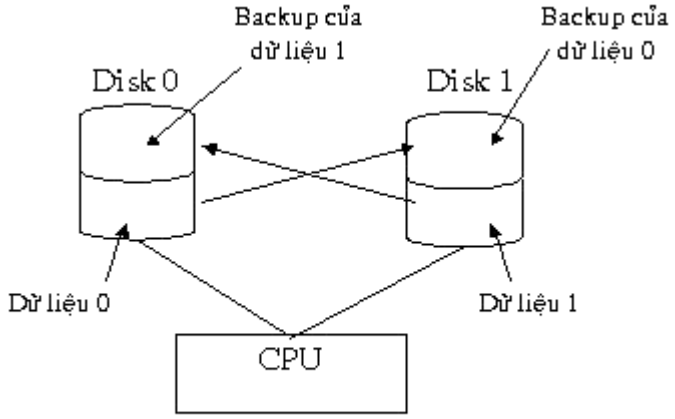
### *7.3.4.2. Sao lưu*

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.

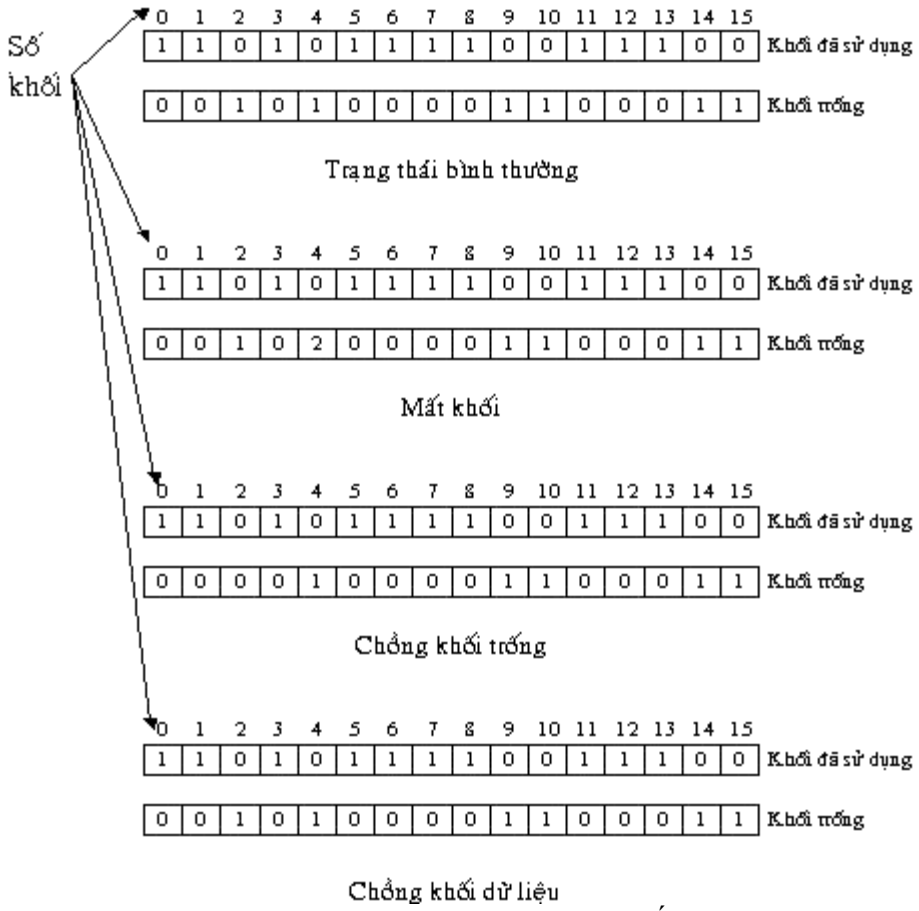



**Hình 9.7** Backup

**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 :

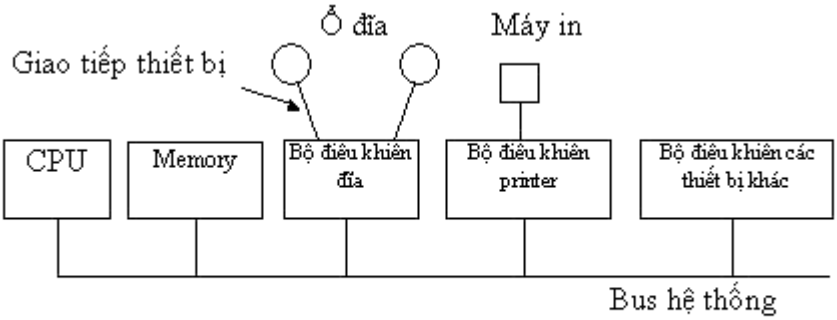
- Bộ xử lý kiểm soát trực tiếp các thiết bị ngoại vi.
- Hệ thống có thêm bộ điều khiển thiết bị. Bộ xử lý sử dụng cách thực hiện nhập xuất thứ nhất. Theo cách này bộ xử lý được tách rời khỏi các mô tả chi tiết của các thiết bị ngoại vi.
- Bộ xử lý sử dụng thêm cơ chế ngắt.
- Sử dụng cơ chế DMA, bộ xử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

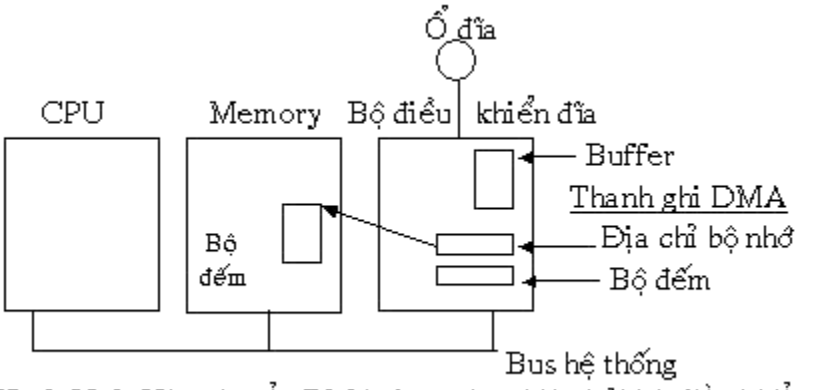
Hệ điều hành thực hiện nhập xuất bằng cách ghi lệnh lên các thanh ghi của bộ điều khiển. Ví dụ : bộ điều khiển đĩa mềm của IBMPC chấp nhận 15 lệnh khác nhau như : READ, WRITE, SEEK, FORMAT, RECALIBRATE, một số lệnh có tham số và các tham số cũng được nạp vào thanh ghi. Khi một lệnh đã được chấp nhận, 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 dể 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

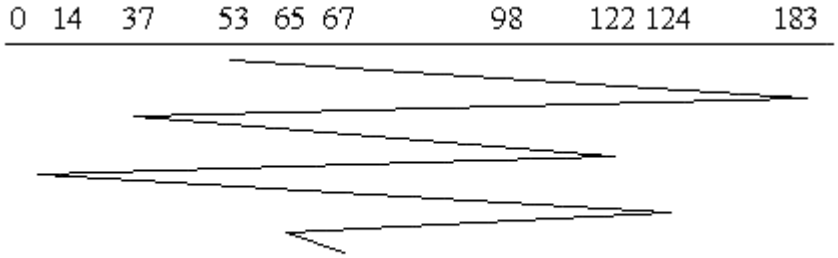
Một đĩa bao gồm nhiều cylinder, mỗi cylinder chứa nhiều track trên các head. Mỗi track được chia làm nhiều sector (từ 8 đến 32). Mỗi sector có số byte là như nhau dù vị trí của nó ở gần tâm hay ở ngoài rìa đĩa, những khoảng trống thừa không dùng đến.
Một đặc điểm thiết bị cài đặt quan trọng cho driver 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 :

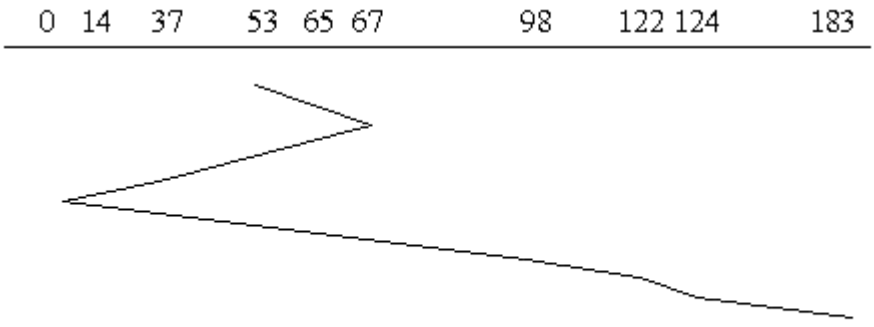



**Hình 12.1** Phương pháp FCFS

**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

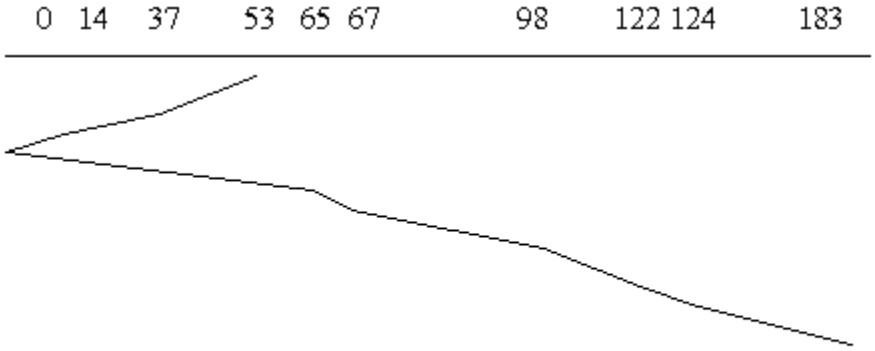
**Hình 9.1.2.2-1.** Lập lịch SSTF

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 :



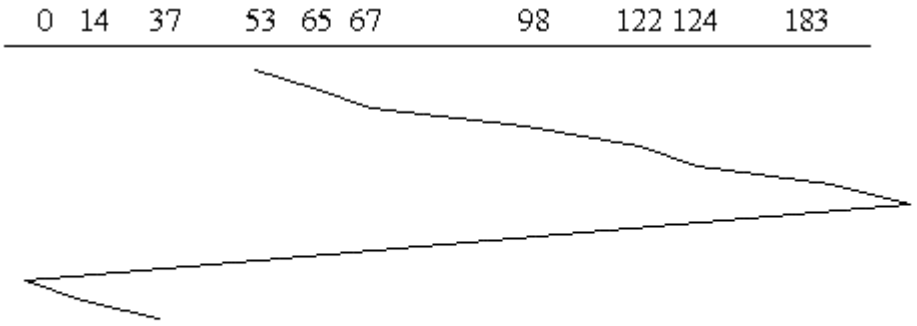

**Hình 12.3** Phương pháp SCAN

**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 :

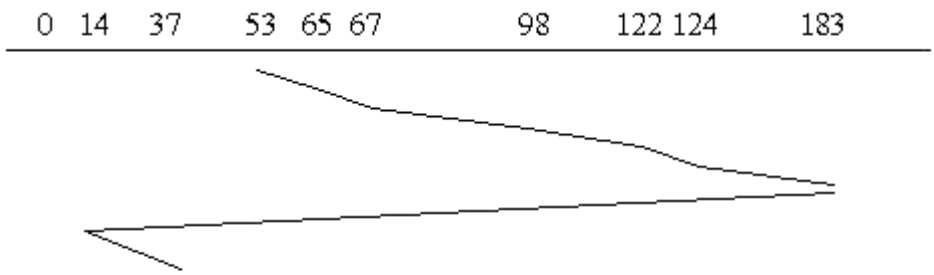



**Hình 12.4** Phương pháp C-SCAN

**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.

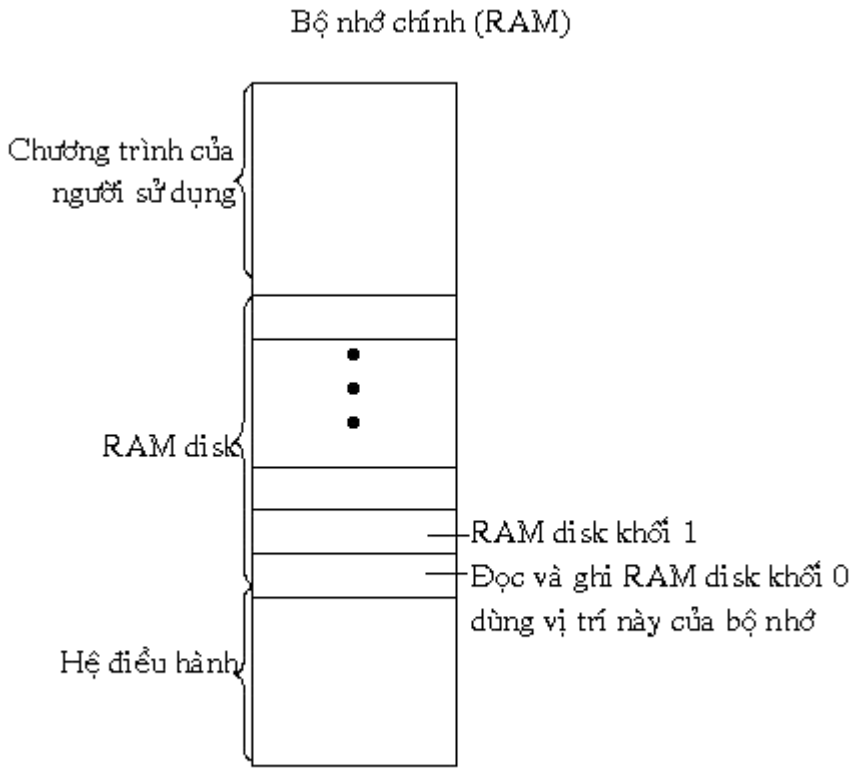
## 9.1.3. Quản lý lỗi

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

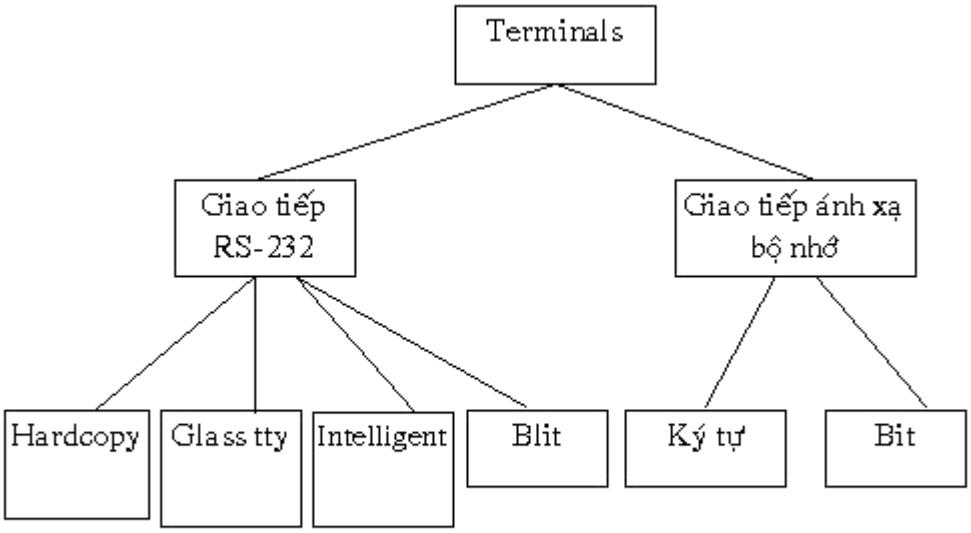
Như vậy sau bốn lần thứ tự các sector đọc được vẫn là từ 1 đến 17

# 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

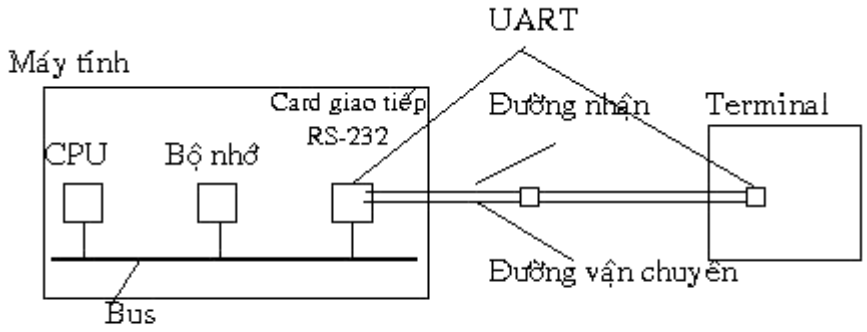
Dưới quan điểm của hệ điều hành, terminal được chia làm hai loại lớn dựa vào cách liên lạc với hệ điều hành. Loại thứ nhất bao gồm những loại terminal giao tiếp theo chuẩn RS-232. Loại thứ hai là những terminal dùng ánh xạ bộ nhớ. Mỗi loại được chia làm nhiều loại nhỏ như hình sau :




**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

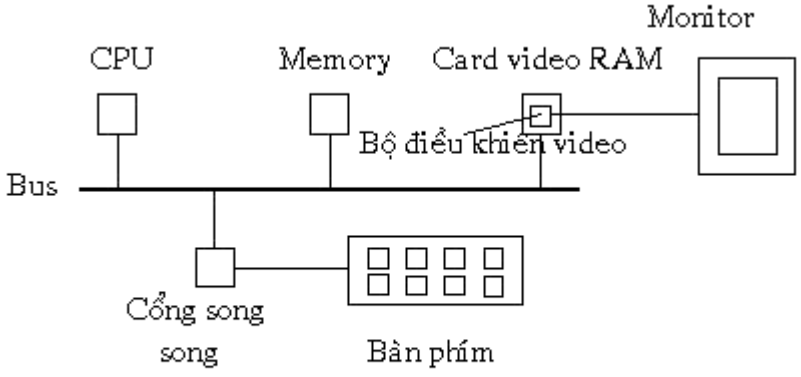
**Hình 9.2.1-2**. RS-232

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

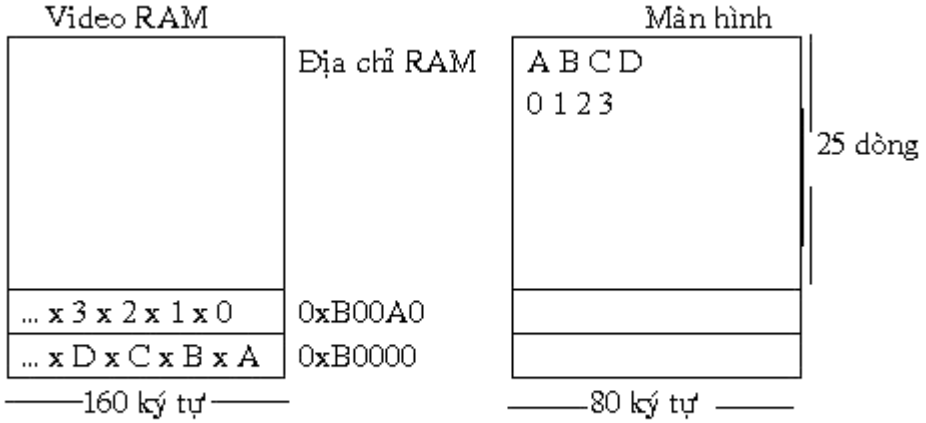
**Hình 9.2.2-1**. Terminal ánh xạ bộ nhớ

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

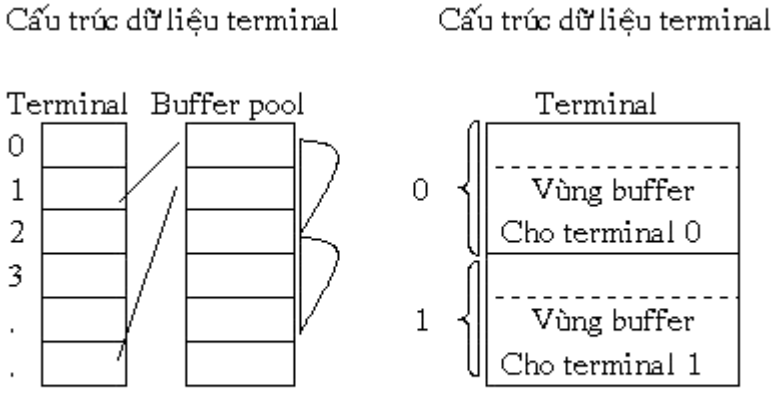
Khi CPU ghi một ký tự vào video RAM, nó xuất hiện trên màn hình theo mỗi lần hiển thị (1/50 giây cho mono, 1/60 cho màu ). CPU có thể nạp 4K ảnh màn hình đã được tính trước vào video RAM trong vài phần triệu giây. Với tốc độ 9600 bps, ghi 2000 ký tự vào terminal RS-232 mất khoảng 2083 phần triệu giây. 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.

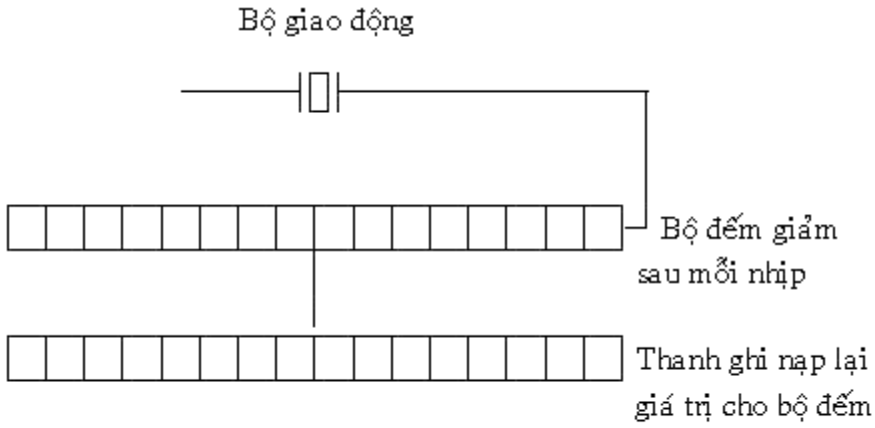
# 9.3. CÀI ĐẶT ĐỒNG HỒ

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.

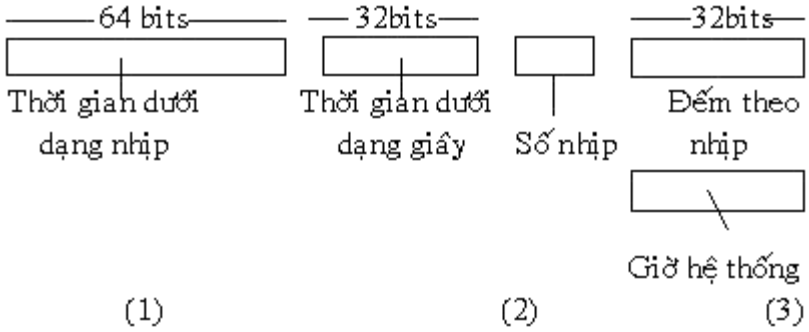
Ưu điểm của đồng hồ có thể lập trình là ngắt định kỳ được điều khiển bởi phần mềm. Nếu sử dụng tin thể thạch anh có tần số 1 MHz, bộ đếm sẽ có nhịp là mỗi micro giây. Với thanh ghi 16 bit, ngắt có thể được lập trình để xảy ra trong khoảng từ 1 đến 65535 msec.

## 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.

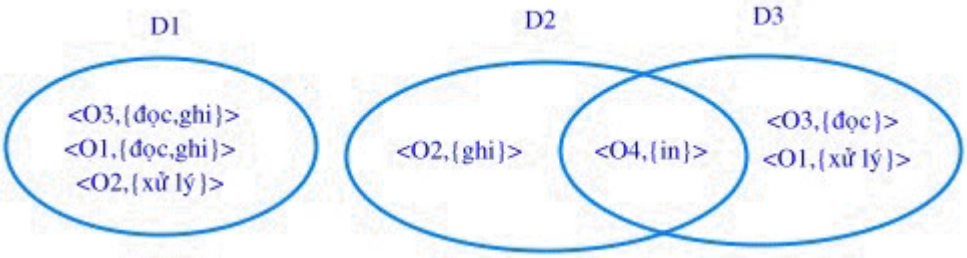
Mỗi tiến trình trong hệ thống đều hoạt động trong một miền bảo vệ *(protection domain)* nào đó. Một miền bảo vệ sẽ xác định các tài nguyên ( đối tượng) mà những tiến trình hoạt động trong miền bảo vệ này có thể sử dụng, và các thao tác hợp lệ các tiến trình này có thể thực hiện trên những tài nguyên 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<br>ghi | | đọc<br>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<br>ghi | | đọc<br>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<br>xử lý | | Ghi |
| **$D_2$** | | đọc*<br>owner | đọc*<br>owner<br>ghi* |

| | | | |
|---|---|---|---|
| **$D_3$** | xử lý | | |

(a)

| **object domain** | **$F_1$** | **$F_2$** | **$F_3$** |
|---|---|---|---|
| **$D_1$** | owner<br>xử lý | | |
| **$D_2$** | | owner<br>đọc*<br>ghi* | đọc*<br>owner<br>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<br>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

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

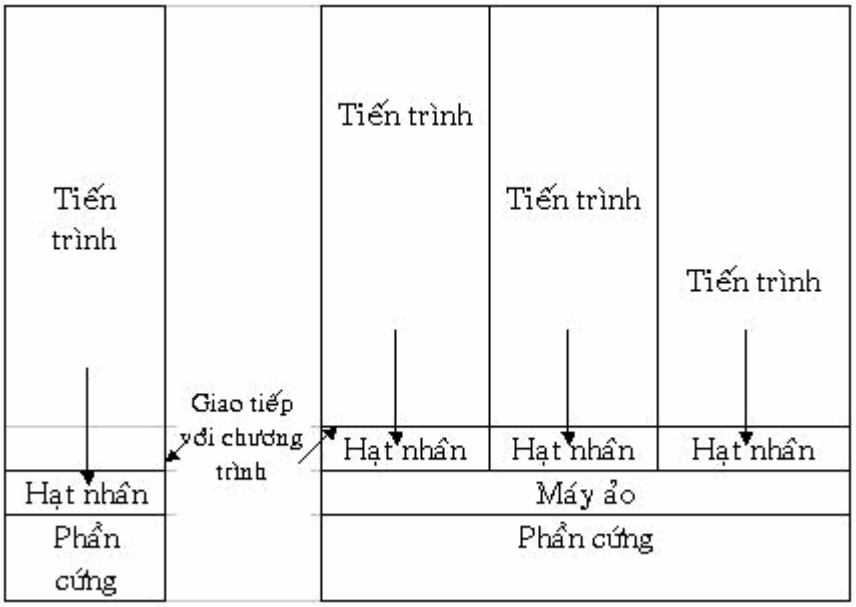
**c) Máy ảo (Virtual Machine)**

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.

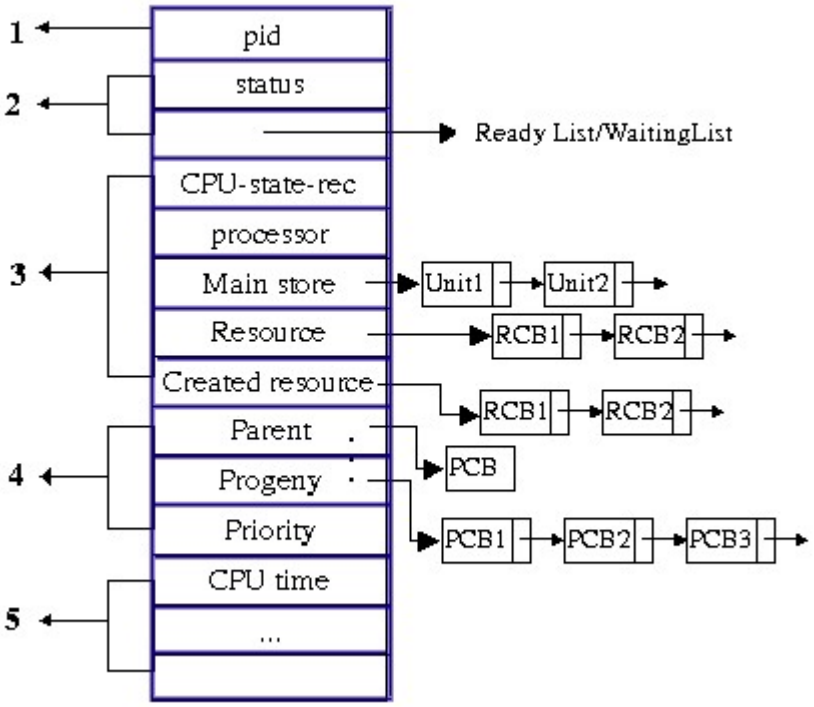
*Tài nguyên sử dụng*: danh sách các tài mguyên hệ thống mà tiến trình đang sử dụng.

*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.

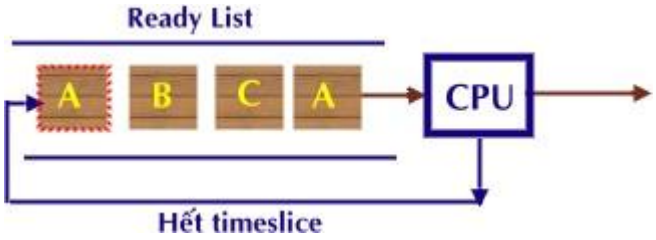
Một số hạn chế:

- 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

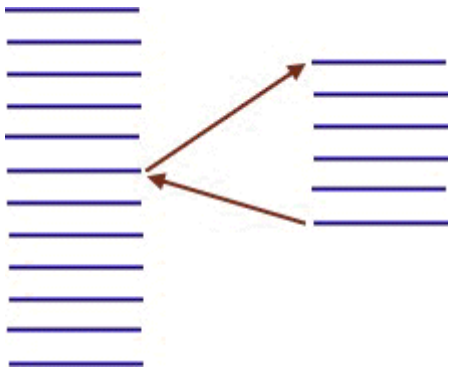
Giải thuật điều phối cho từng danh sách ứng với một cấp ưu tiên.

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

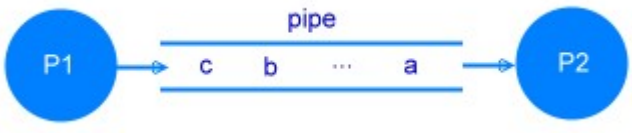
Tiếp nhận tín hiệu và xử lý theo cách đặc biệt của tiến trình.

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 !

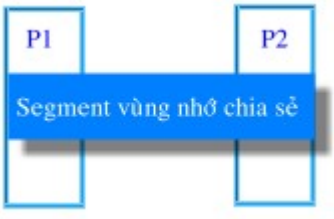
Cơ chế này cho phép truyền dữ liệu với cách thức không cấu trúc.

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.

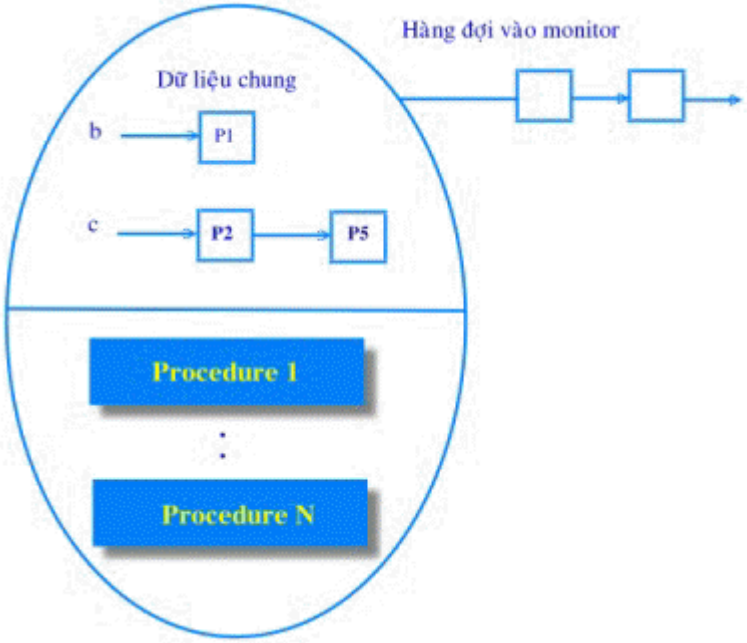
Hai tính chất 1 và 2 tương tự như các tính chất của các đối tượng trong lập trình hướng đối tượng. Như vậy một hệ điều hành hoặc một ngôn ngữ lập trình hướng đối tượng có thể cài đặt monitor như là một đối tượng có các tính chất đặc biệt. Với tính chất 3 monitor có khả năng thực hiện các cơ chế độc quyền, 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++)

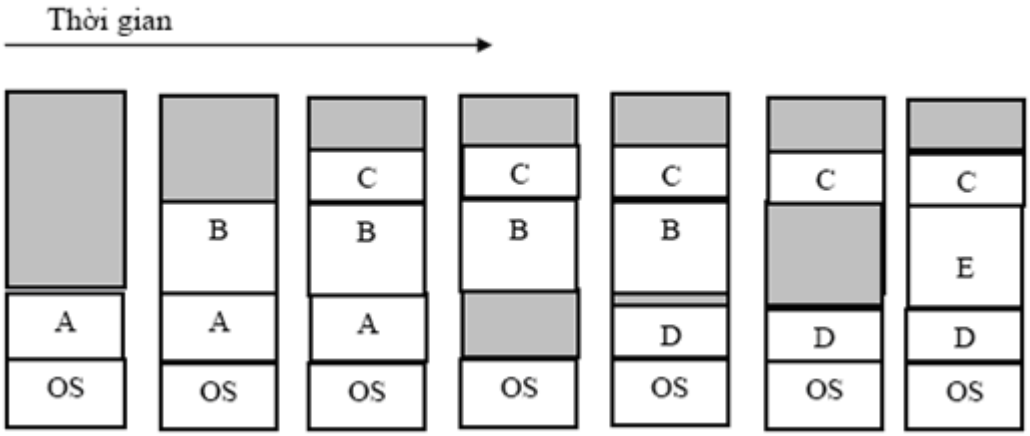
Finish[i] = (Allocation[i] == 0);

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

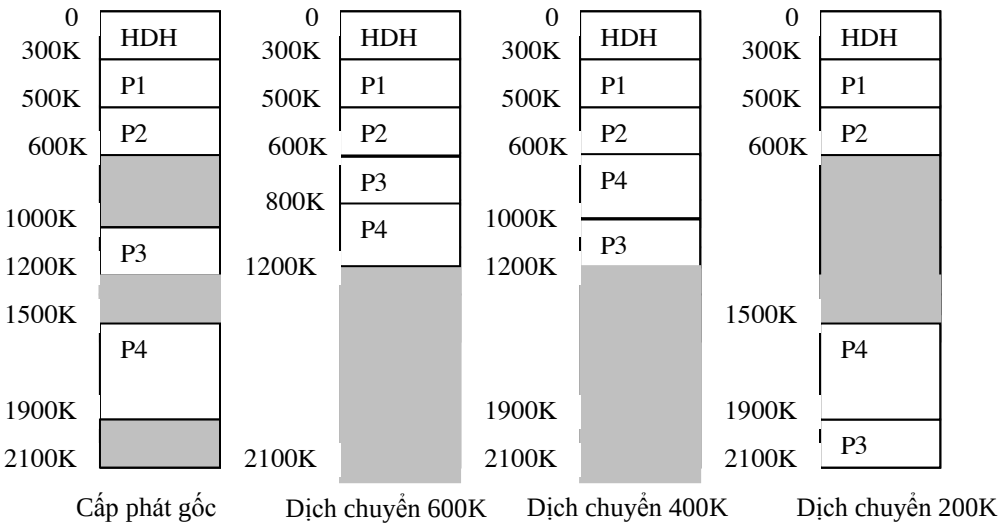
Trong kỹ thuật phân vùng động, số lượng các vùng trên bộ nhớ và kích thước mỗi phân vùng là có thể thay đổi. Tức là phần user program trên bộ nhớ không dược phân chia trước mà nó chỉ được ấn định sau khi dã có một tiến trình được nạp vào bộ nhớ chính. Khi có một tiến trình được nạp vào bộ nhớ nó được 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**

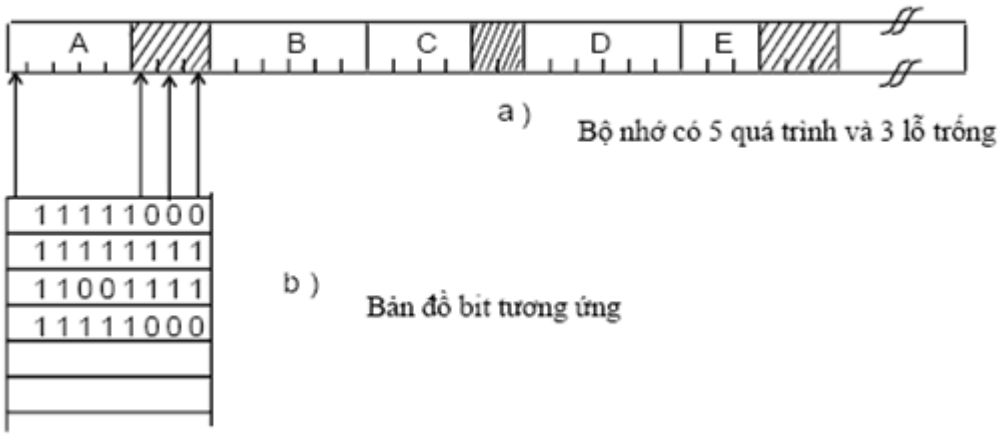
Bộ nhớ được chia thành các đơn vị cấp phát, có kích thước bằng nhau.

- **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.

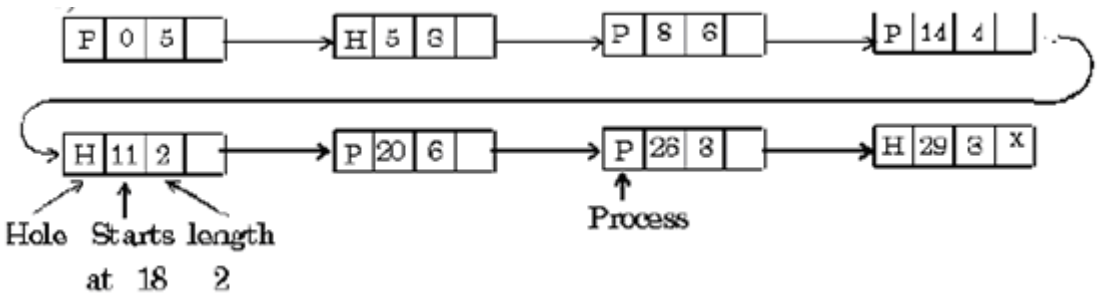



Hình 3.6 Quản lý bộ nhớ bằng bản đồ bit

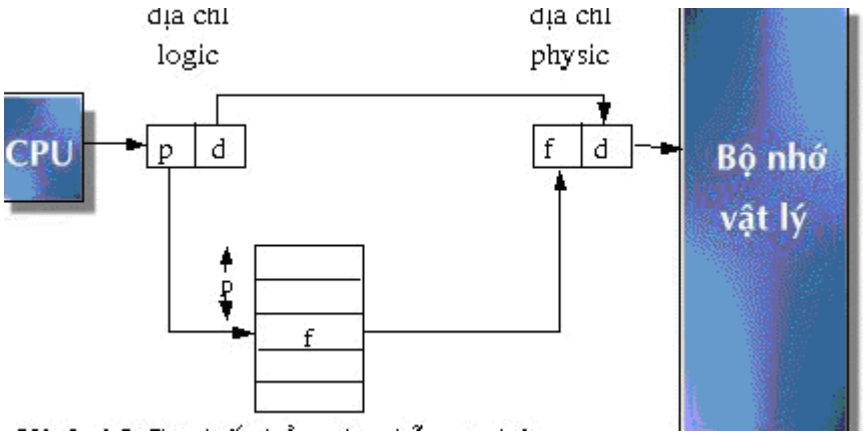
**- Quản lý bằng danh sách liên kết:**

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.

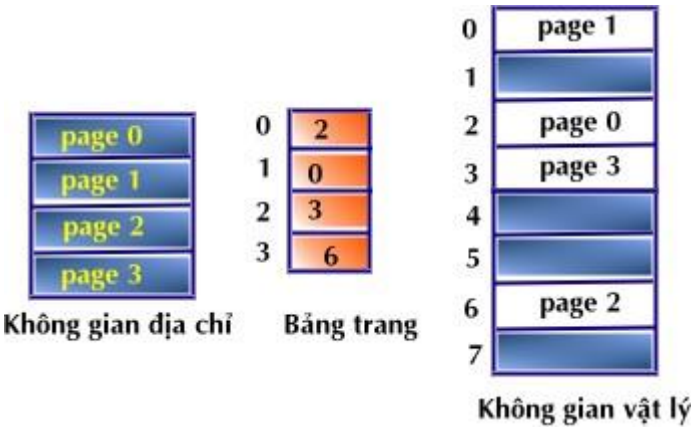
Cơ chế phần cứng hỗ trợ thực hiện chuyển đổi địa chỉ trong cơ chế phân trang là bảng trang (*pages table*). Bảng trang để theo dõi vị trí các trang tiến trình trên bộ nhớ.

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.

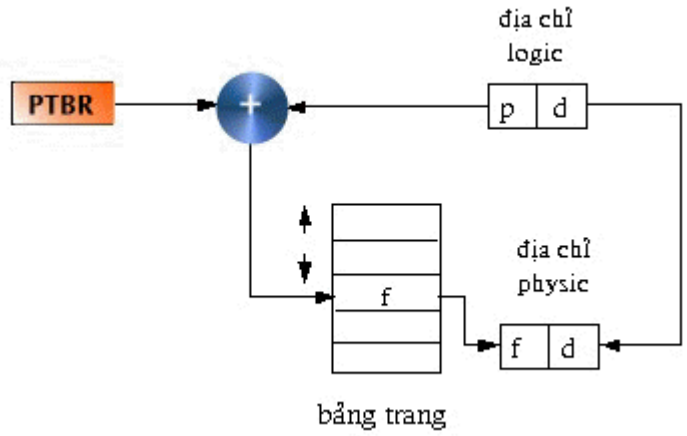
| Số trang | Độ dời trang |
|---|---|
| P | D |
| m –n | N |

**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).

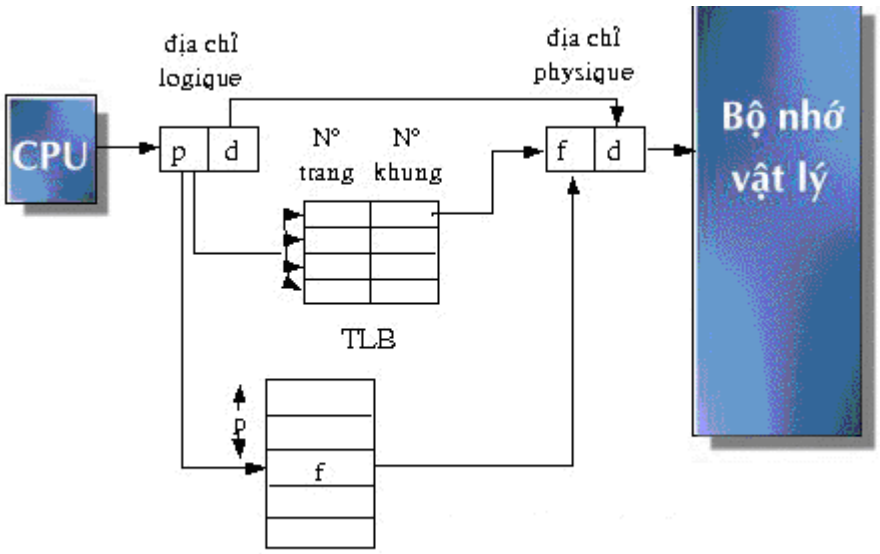
Theo cách tổ chức này, mỗi truy xuất đến dữ liệu hay chỉ thị đều đòi hỏi hai lần truy xuất bộ nhớ : một cho truy xuất đến bảng trang và một cho bản thân dữ liệu, do vậy truy cập chậm.




Hình 3.11 Sử dụng thanh ghi nền trỏ đến bảng trang

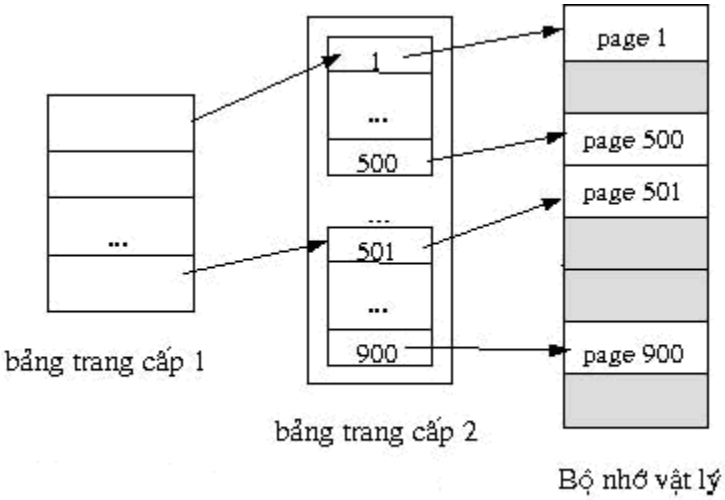
- Để nâng cao tốc độ truy xuất, sử dụng thêm một vùng nhớ đặc biệt , với tốc độ truy xuất nhanh và cho phép tìm kiếm song song, vùng nhớ cache nhỏ này thường được gọi là bộ nhớ kết hợp (translation look-aside buffer TLBs). Mỗi thanh ghi trong bộ nhớ kết hợp chứa số hiệu trang và số hiệu khung trang tương 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

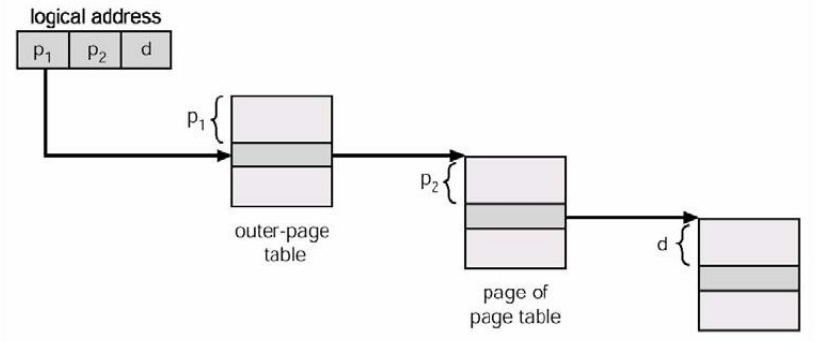
Mỗi hệ điều hành có một phương pháp riêng để tổ chức lưu trữ bảng trang. Đa số các hệ điều hành cấp cho mỗi tiến trình một bảng trang. Tuy nhiên phương pháp này không thể chấp nhận được nếu hệ điều hành cho phép quản lý một không gian địa chỉ có dung lượng quá ($2^{31}$, $2^{9}$): trong các hệ thống 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 |

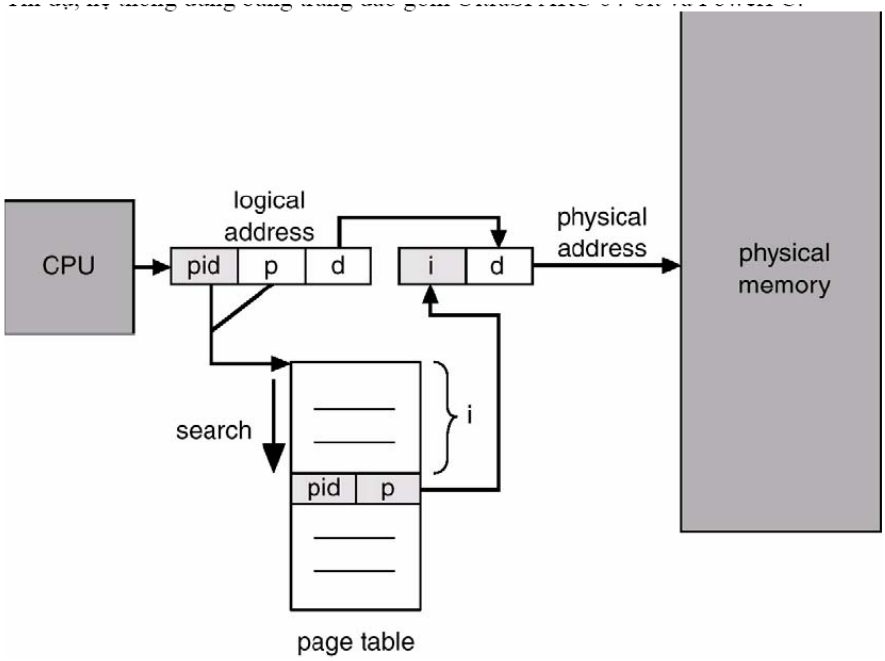
***Bảng trang nghịch đảo*** (inverted page table).

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.

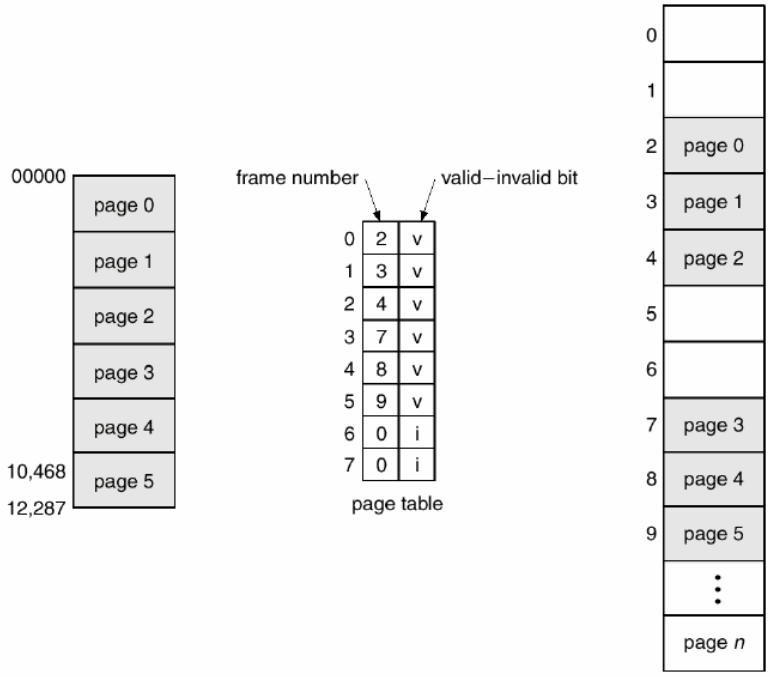
Ngoài ra, một bit phụ trội được thêm vào trong cấu trúc một phần tử của bảng trang : bit hợp lệ-bất hợp lệ (valid-invalid).

*Hợp lệ* : trang tương ứng thuộc về không gian địa chỉ của tiến trình.

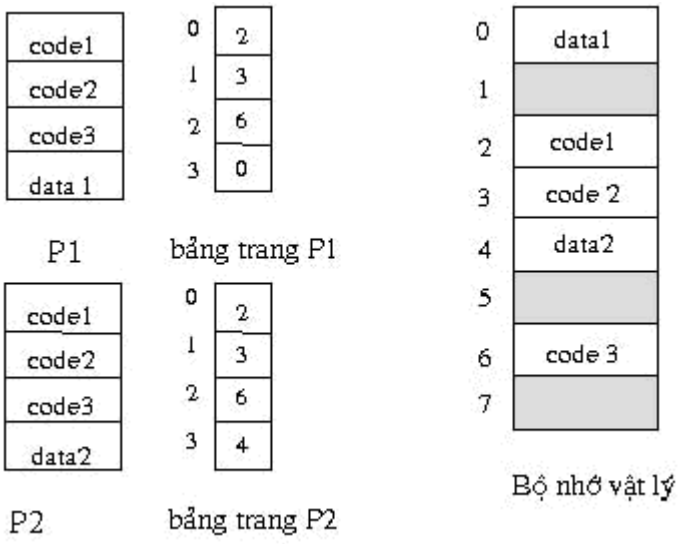
| số hiệu khung trang | bit valid-invalid |
|---|---|

*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**:



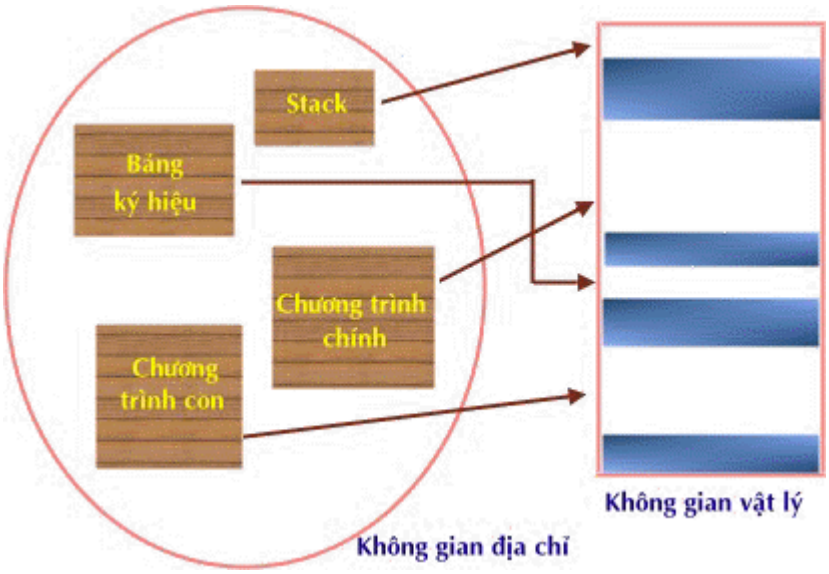

***Hình 3.17*** Chia sẻ các trang trong hệ phân trang

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 hỗ trợ kĩ thuật phân đoạn

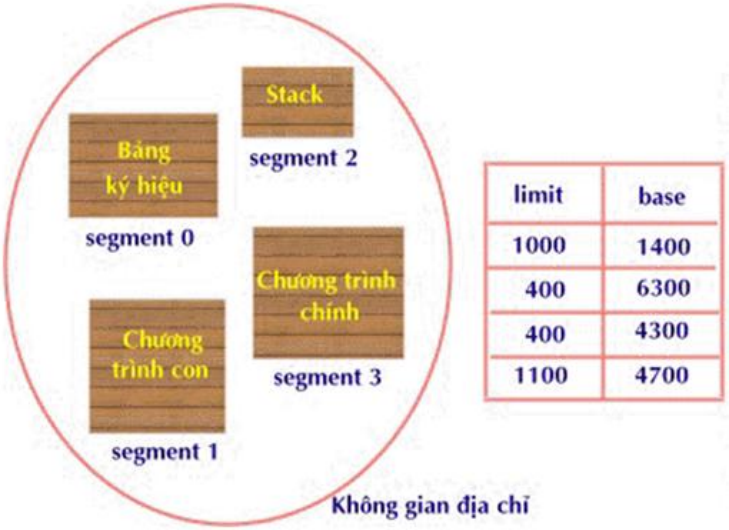
Để theo dõi các đoạn của các tiến trình khác nhau trên bộ nhớ, hệ điều hành sử dụng các bảng phân đoạn tiến trình, thông thường một tiến trình có một bảng phân đoạn riêng. Mỗi phần tử trong bảng phân đoạn gồm tối thiểu 2 trường: trường thứ nhất cho biết địa chỉ cơ sở(base) của phân vùng mà đoạn chương 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.



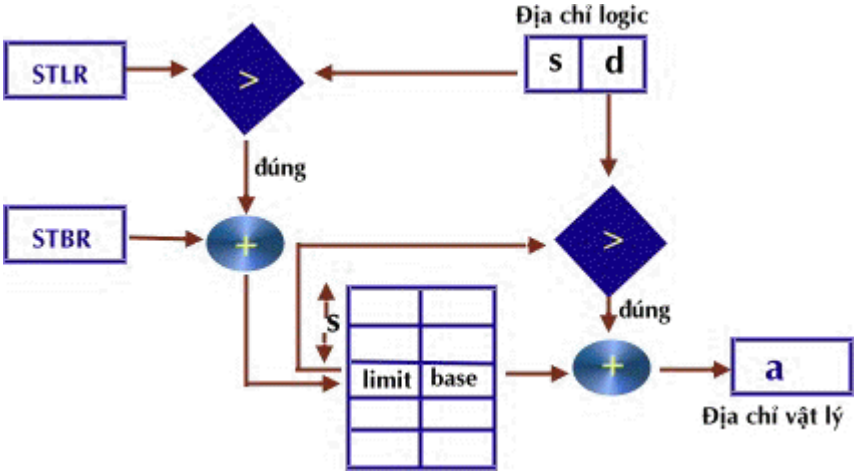

**Cài đặt bảng phân đoạn:**

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*)

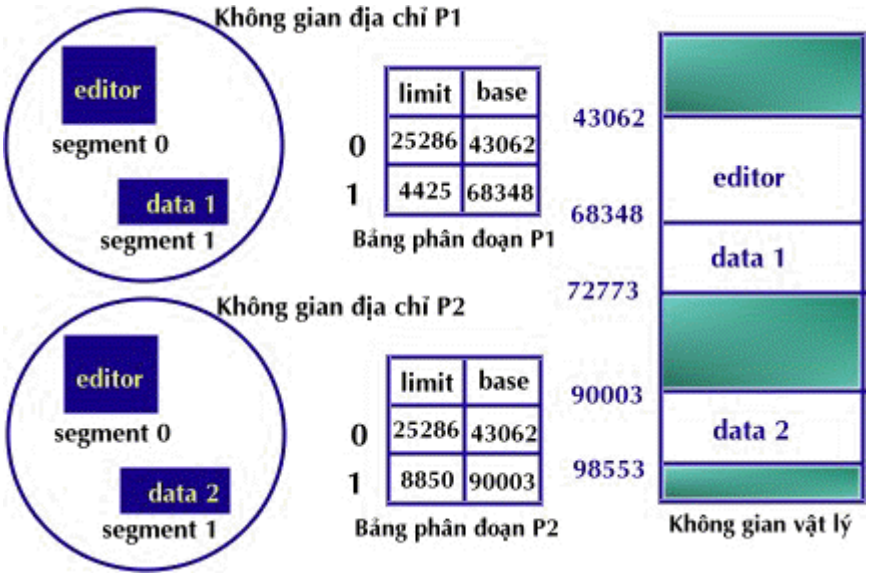
Hình 3.21 Sử dụng STBR, STLR và bảng phân đoạn

**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.

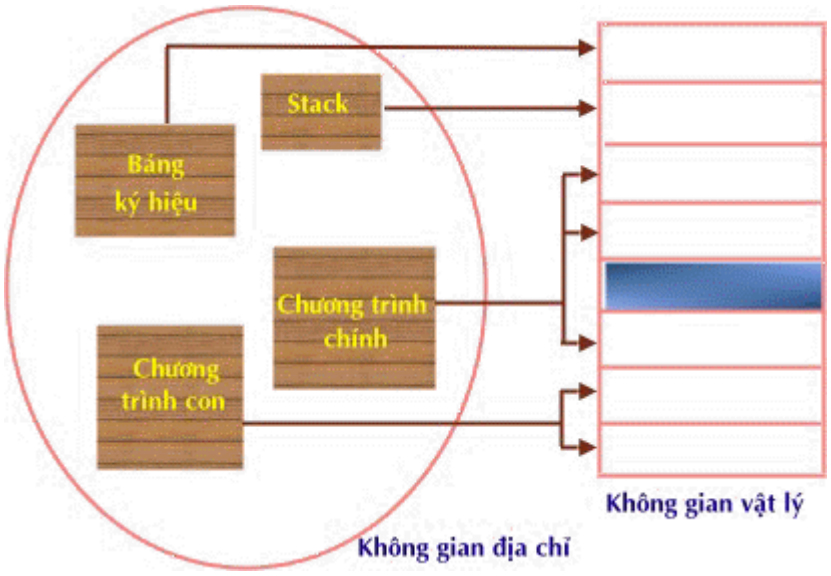
Kỹ thuật phân đoạn dùng chính nó để cài đặt các chính sách bảo vệ và chia sẻ bộ nhớ. Bởi vì mỗi phần tử trong bảng đoạn bao gồm một trường độ dài đoạn và trường chỉ nơi bắt đầu của đoạn, nên một tiến trình trong đoạn không thể truy cập đến một vị trí trong bộ nhớ chính mà vị trí này vượt qua giới hạn 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*

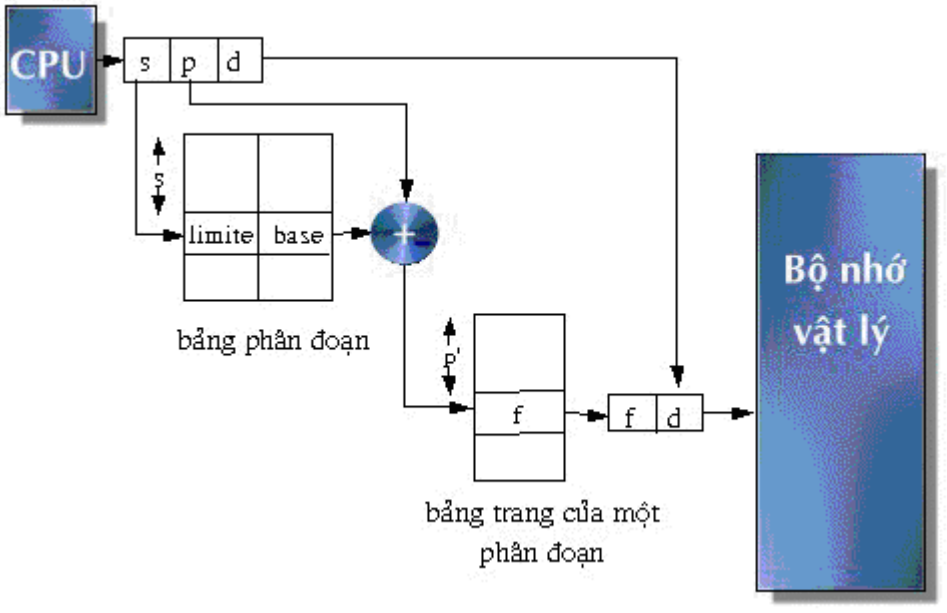
Một ưu điểm khác của kỹ thuật phân đoạn là khả năng chia sẻ ở mức độ phân đoạn. Nhờ khả năng này, các tiến trình có thể chia sẻ với nhau từng phần chương trình ( ví dụ các thủ tục, hàm), không nhất thiết phải chia sẻ toàn bộ




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:

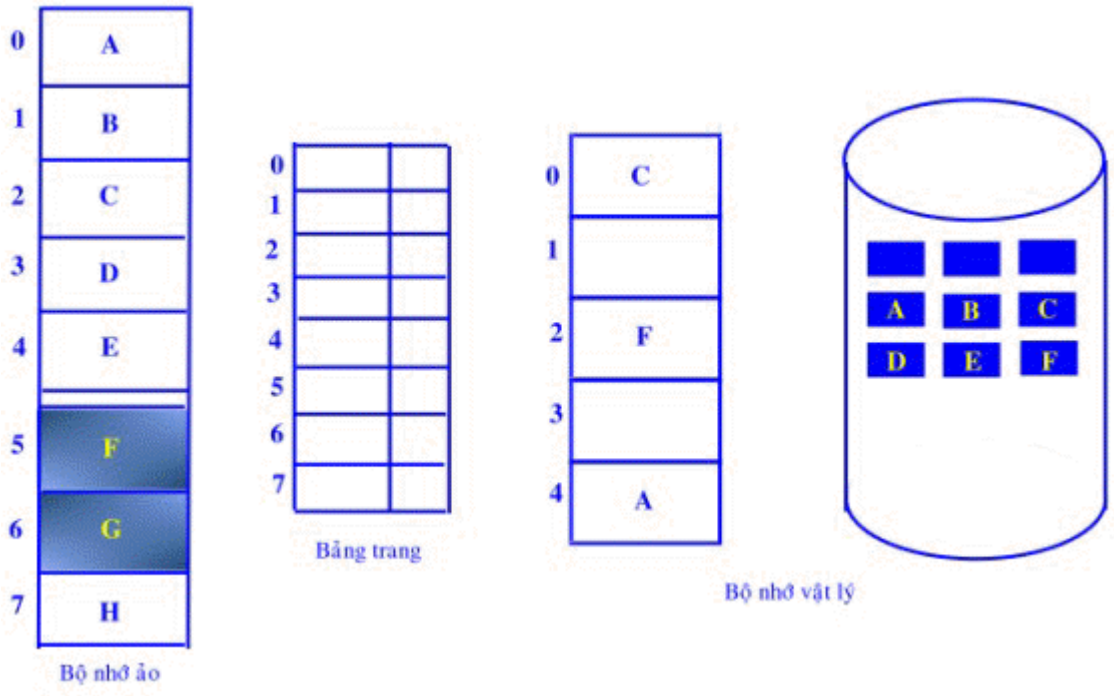
*valid* : trang tương ứng là hợp lệ và đang ở trong bộ nhớ chính .

*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

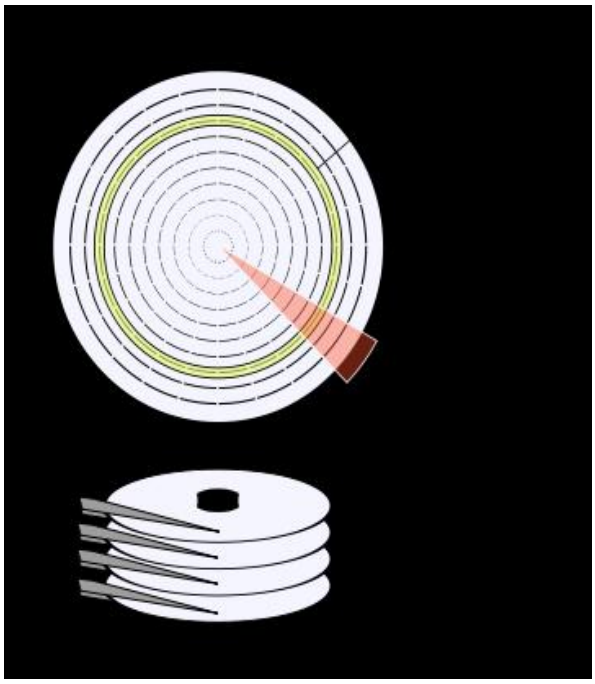
| Mặt đĩa | Track | Sector | Sector logic | Thông tin lưu trữ |
|---|---|---|---|---|
| 0 | 0 | 1 | 0 | Boot record |
| 0 | 0 | 2-5 | 1-4 | FAT |
| 0 | 0 | 6-9 | 5-8 | Thư mục gốc |
| 1 | 0 | 1-3 | 9-11 | Thư mục gốc |
| 1 | 0 | 4-9 | 12-17 | Dữ liệu |
| 0 | 1 | 1-9 | 18-26 | Dữ liệu |

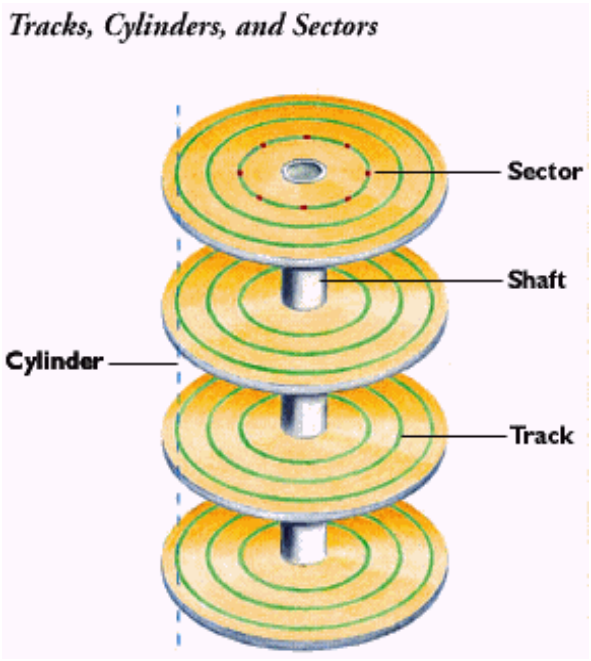
Bảng: Tương ứng giữa sector vật lý và sector logic trên đĩa mềm

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

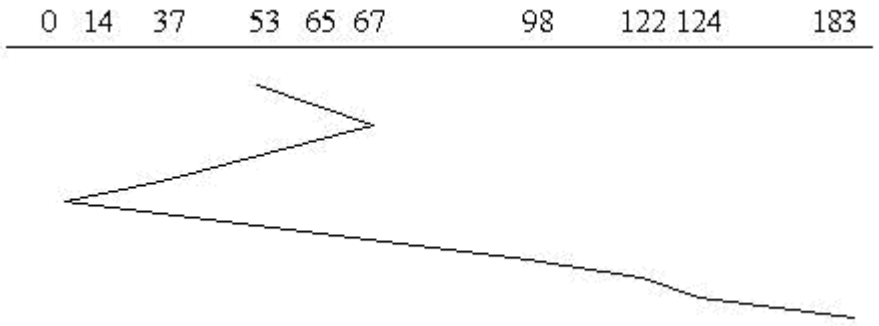
**b)Lập lịch SSTF (shortest-seek-time-first)**

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

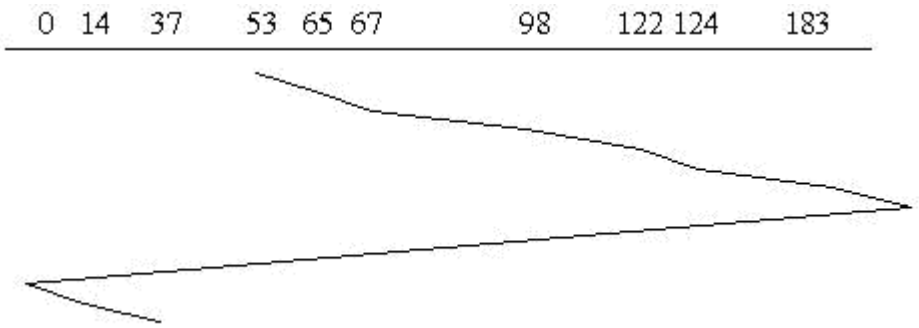
Giả sử hiện tại đầu đọc đang ở vị trí 53. Như vậy đầu đọc lần lượt đi qua các khối

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**

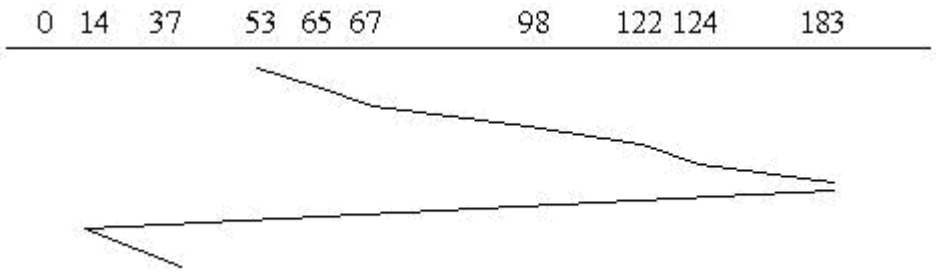
Thuật toán này tương tự như thuật toán SCAN, chỉ khác là khi nó di chuyển đến một đầu nào đó của đĩa, nó sẽ lập tức trở về đầu bắt đầu của đĩa. Lấy lại ví dụ trên, khi đó thứ tự truy xuất các khối sẽ là : 53, 65, 67, 98, 122, 124, 183, 199, 0, 14, 37 như hình sau :




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.

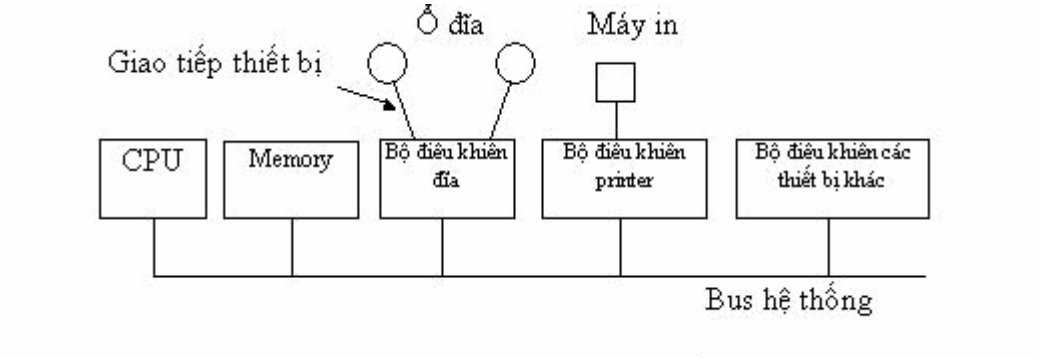
Đơn vị truyền dữ liệu (khối hoặc ký tự).

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. |

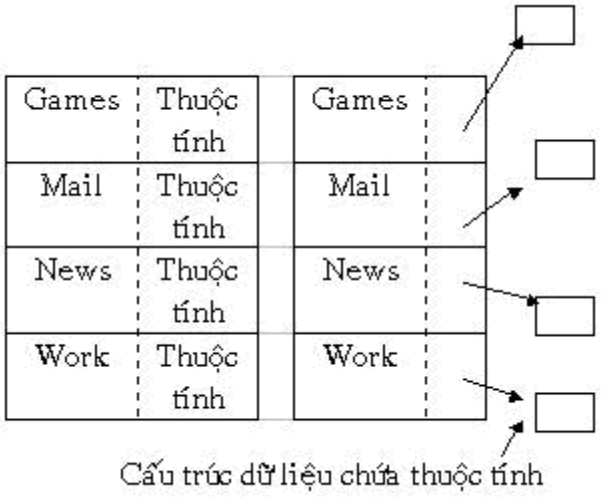
Ngoài tên và dữ liệu, hệ điều hành cung cấp thêm một số thông tin cho tập tin gọi là thuộc tính.

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

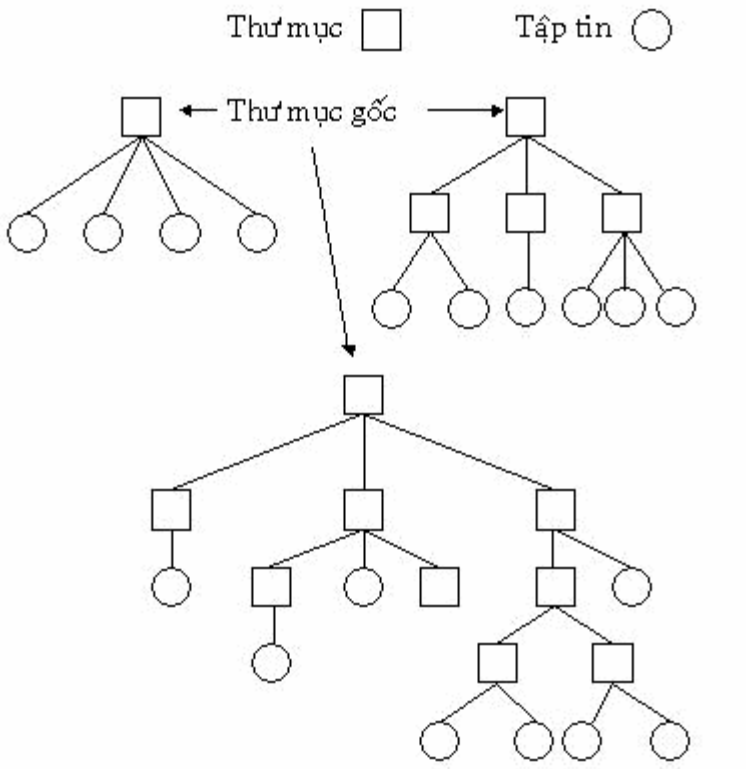
| Tên thuộc tính | Ý nghĩa |
|---|---|
| Bảo vệ | Ai có thể truy xuất được và bằng cách nào |
| Mật khẩu | Mật khẩu cần thiết để truy xuất tập tin |
| Người tạo | Id của người tạo tập tin |
| Người sở hữu | Người sở hữu hiện tại |
| Chỉ đọc | 0 là đọc ghi, 1 là chỉ đọc |
| ẩn | 0 là bình thường, 1 là 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.

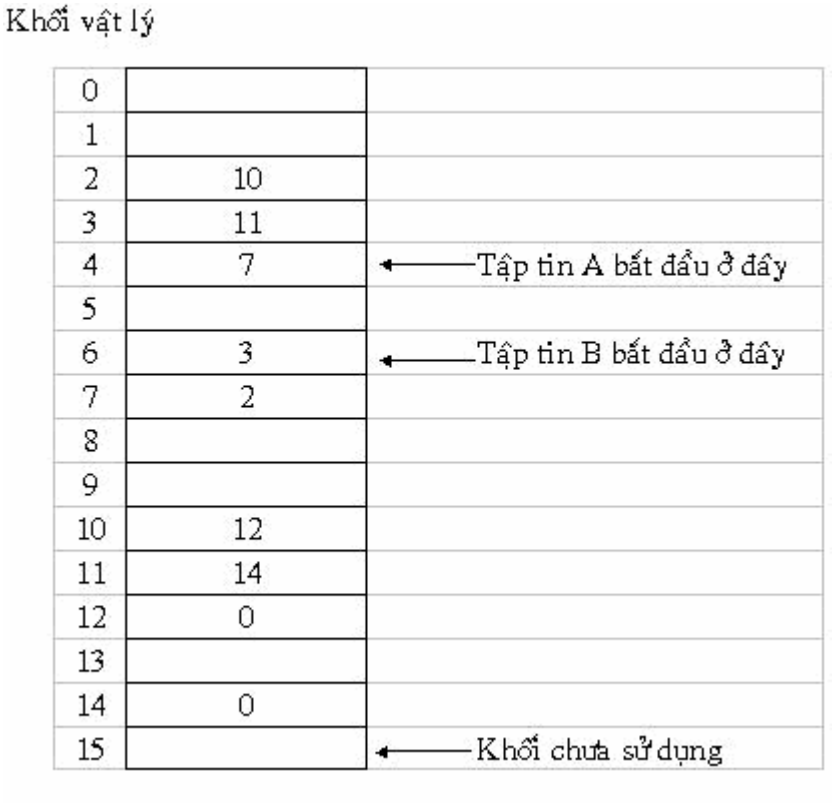
*Đọc thư mục* :Lệnh này trả về entry tiếp theo trong thư mục đã mở. Thông thường có thể đọc thư mục bằng lời gọi hệ thống READ, lệnh đọc thư mục luôn luôn trả về một entry dưới dạng chuẩn .

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.

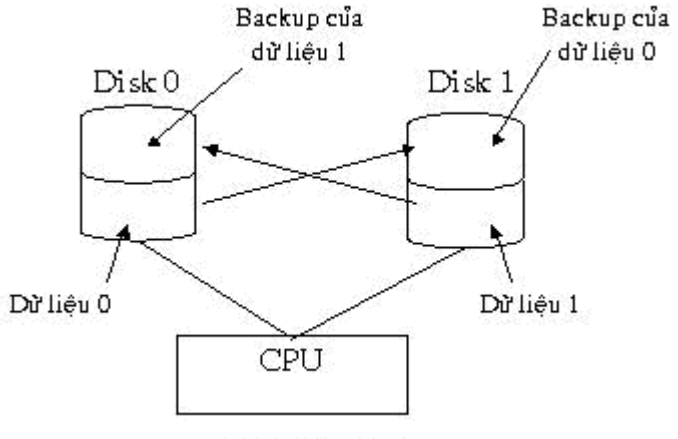
Phần mềm là hệ thống tập tin xây dựng một tập tin chứa các khối hỏng. Kỹ thuật này loại trừ chúng ra khỏi danh sách các khối trống, do đó nó sẽ không được cấp phát cho tập tin.

**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.

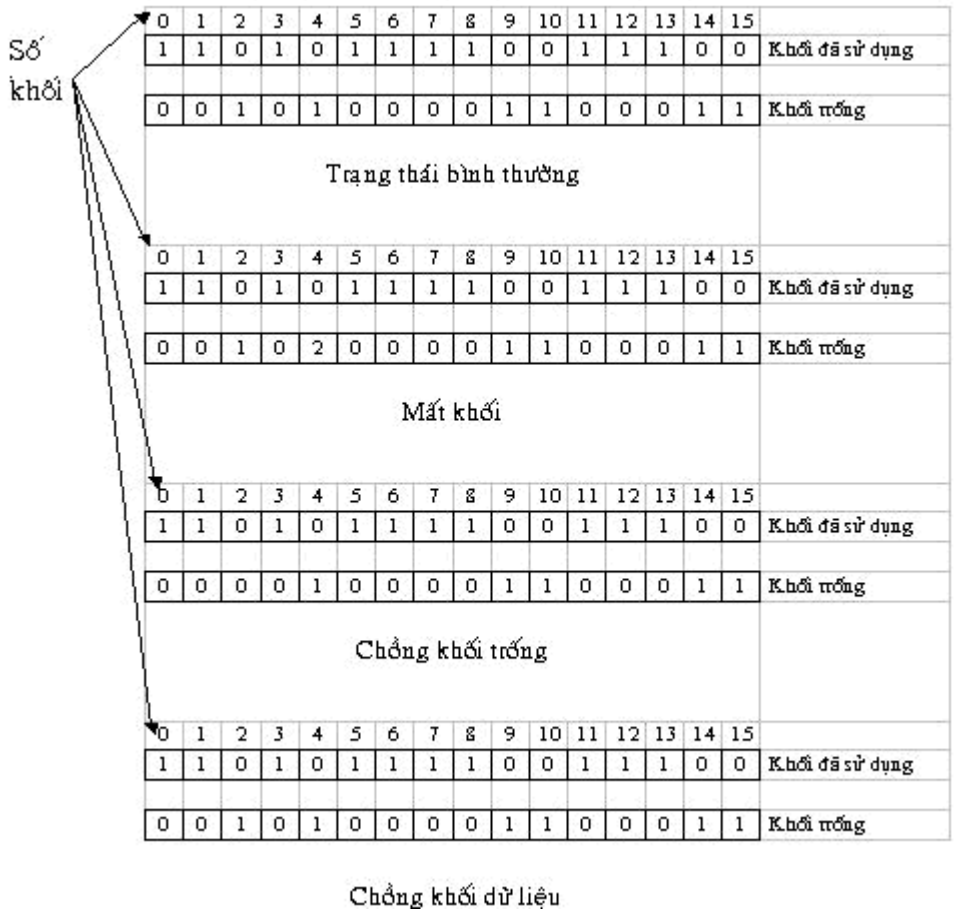
Đối với các đĩa cứng lớn, việc backup thường được tiến hành ngay trên nó. Một chiến lược dể cài đặt nhưng lãng phí một nữa đĩa là chia đĩa cứng làm hai phần một phần dữ liệu và một phần là backup. Mỗi tối, dữ liệu từ phần dữ liệu sẽ được chép sang phần backup.




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.

BỘ GIAO THÔNG VẬN TẢI
**TRƯỜNG ĐẠI HỌC HÀNG HẢI**
**BỘ MÔN: KỸ THUẬT MÁY TÍNH**
**KHOA: CÔNG NGHỆ THÔNG TIN**

# BÀI GIẢNG

# NGUYÊN LÝ HỆ ĐIỀU HÀNH

TÊN HỌC PHẦN : **NGUYÊN LÝ HỆ ĐIỀU HÀNH**
MÃ HỌC PHẦN : **17303**
TRÌNH ĐỘ ĐÀO TẠO : **ĐẠI HỌC CHÍNH QUY**
DÙNG CHO SV NGÀNH : **CÔNG NGHỆ THÔNG TIN**

**HẢI PHÒNG - 2010**

# MỤC LỤC

# YÊU CẦU VÀ NỘI DUNG CHI TIẾT

*Tên học phần:* **Nguyên lý Hệ điều hành** *Loại học phần:* **1**
*Bộ môn phụ trách giảng dạy:* **Kỹ thuật máy tính** *Khoa phụ trách:* **CNTT**
*Mã học phần:* **17303** *Tổng số TC:* **2**

| TS tiết | Lý thuyết | Thực hành/Xemina | Tự học | Bài tập lớn | Đồ án môn học |
|---|---|---|---|---|---|
| 45 | 45 | 0 | 0 | 0 | 0 |

**Điều kiện tiên quyết:**

Sinh viên phải học xong các học phần sau mới được đăng ký học phần này:

Tin học đại cương, Kiến trúc máy tính, Kỹ thuật lập trình, Cấu trúc dữ liệu, Điện tử số, Mạch và tín hiệu

**Mục tiêu của học phần:**

- Cung cấp cho sinh viên những khái niệm tổng quan về Hệ điều hành, các phương pháp tiếp cận giải quyết các bài toán điều khiển hoạt động của hệ thống máy.
- Cung cấp kiến thức chung về nguyên lý hoạt động theo chương trình trên hệ thống đơn, đa bộ xử lý của máy tính.

**Nội dung chủ yếu**

Chương I: Những khái niệm cơ bản
Chương II: Quản lý thiết bị
Chương III : Quản lý bộ nhớ
Chương IV : Quản lý tiến trình
Chương V : Hệ điều hành nhiều vi xử lý - hệ điều hành phân tán

**Nội dung chi tiết của học phần:**

| TÊN CHƯƠNG MỤC | PHÂN PHỐI SỐ TIẾT | | | | |
|---|---|---|---|---|---|
| | **TS** | **LT** | **BT** | **TH** | **KT** |
| **Chương I: Những khái niệm cơ bản** | **6** | **6** | | | |
| 1.1. Cấu trúc phân lớp và hệ thống tính toán | | 1 | | | |
| 1.2. Tài nguyên hệ thống | | 1 | | | |
| 1.3. Định nghĩa hệ điều hành | | 1 | | | |
| 1.4. Tính chất chung của hệ điều hành | | 1 | | | |
| 1.5. Thành phần hệ điều hành | | 1 | | | |
| 1.6. Các nguyên tắc xây dựng hệ điều hành | | 1 | | | |
| **Chương II: Quản lý thiết bị** | **9** | **8** | | | **1** |
| 2.1. Quan hệ phân cấp trong tổ chức và quản lý thiết bị ngoại vi | | 2 | | | |
| 2.2. Cơ chế phòng đệm | | 2 | | | 1 |
| 2.3. Cơ chế SPOOL | | 1 | | | |
| 2.4. Quản lý File | | 1 | | | |
| 2.5. Quản lý file trong hệ điều hành MSDOS | | 2 | | | |
| **Chương III : Quản lý bộ nhớ** | **10** | **9** | | | **1** |

| TÊN CHƯƠNG MỤC | PHÂN PHỐI SỐ TIẾT | | | | |
|---|---|---|---|---|---|
| | TS | LT | BT | TH | KT |
| 3.1. Các giai đoạn xử lý chương trình | | 1 | | | |
| 3.2. Quản lý bộ nhớ logic- cấu trúc một chương trình | | 3 | | | |
| 3.3. Quản lý bộ nhớ vật lý | | 3 | | | |
| 3.4. Quản lý bộ nhớ IBM PC của MSDOS | | 2 | | | 1 |
| **Chương IV : Quản lý tiến trình** | **13** | **12** | | | **1** |
| 4.1. Quản lý tiến trình | | 7 | | | |
| 4.2. Quản lý Processor | | 5 | | | |
| **Chương V : Hệ điều hành nhiều vi xử lý - hệ điều hành phân tán** | **7** | **7** | | | |
| 5.2. Hệ điều hành nhiều vi xử lý | | 1 | | | |
| 5.3. Hệ điều hành phân tán | | 2 | | | |
| 5.4. Quản lý tài nguyên trong hệ điều hành phân tán | | 4 | | | |

**Nhiệm vụ của sinh viên :**

Tham dự các buổi thuyết trình của giáo viên, tự học, tự làm bài tập do giáo viên giao, tham dự các buổi thực hành, các bài kiểm tra định kỳ và cuối kỳ.

**Tài liệu học tập :**

1. Văn Nguyễn Thanh Tùng, *Giáo trình Hệ điều hành*, ĐH Bách Khoa HN
2. Milan Milenkovic, *Operating systems concept and design*.
3. Mc Graw Prin, *Operating system.*
4. Prentice Hall, *Modern Operating system*
5. Hà Quang Thụy, *Giáo trình Nguyên lý các hệ điều hành*, NXB KHKT Hà Nội, 2004.
6. Hoàng Kiếm, *Giáo trình Nguyên lý hệ điều hành*, Đại học Quốc gia TP HCM
7. Nguyễn Kim Tuấn, Nguyễn Gia Định, *Nguyên* lý *hệ điều hành*, NXB KHKT Hà Nội, 2005.

**Hình thức và tiêu chuẩn đánh giá sinh viên:**

- Đánh giá dựa trên tình hình tham dự buổi học trên lớp, các buổi thực hành, điểm kiểm tra thường xuyên và điểm kết thúc học phần.
- Hình thức thi cuối kỳ : thi viết rọc phách, thời gian làm bài: 75 phút

**Thang điểm: Thang điểm chữ A, B, C, D, F**

**Điểm đánh giá học phần Z = 0.2X + 0.8Y.**

Bài giảng này là tài liệu **chính thức và thống nhất** của Bộ môn Kỹ thuật máy tính, Khoa Công nghệ Thông tin và được dùng để giảng dạy cho sinh viên.

**Ngày phê duyệt: 15 / 6 / 2010**

**Trưởng Bộ môn: ThS. Ngô Quốc Vinh**

# Chương I: NHỮNG KHÁI NIỆM CƠ BẢN

***Quan tâm của người dùng***

- Các hệ thống chương trình có cấu trúc như thế nào?
- Các hệ thống có đặc trưng gì?
- Hệ thống cung cấp cho người dùng những tài nguyên gì?

## 1.1. Cấu trúc phân lớp và hệ thống tính toán

Khi người dùng thực hiện một chương trình, hệ thống có đáp ứng được các yêu cầu hay không

*Bao gồm:*

- Hệ thống có chương trình cần thực hiện hay không
- Có đủ bộ nhớ để làm việc hay không
- Có các thiết bị ngoại vi theo yêu cầu hay không

Tuy nhiên yêu cầu của người dùng là đa dạng, khả năng của hệ thống có hạn nên đôi khi chi phí cho hệ thống khá cao song lợi ích mà hệ thống mang lại nhỏ.

Để khắc phục đưa ra giải **pháp tăng tính vạn năng của hệ thống qua processor**:

### *1.1.1. Cơ sở hoá hệ lệnh*

Trước đây trong máy tính đã lắp ráp nhiều vi mạch thực hiện các chức năng chuyên dụng tính căn, sin, e_mũ, loga.. vì vậy khi sử dụng rất khó có thể sửa chữa, thay đổi được.

Hiện nay các chức năng này đã được thay thế bằng phần mềm do đó máy tính vạn năng hơn, tốc độ cao hơn, độ ổn định và giá thành hạ.

Các Chương trình bao quanh phần kỹ thuật tạo thành một môi trường tính toán. Mỗi Chương trình muốn được thực hiện phải gắn với môi trường và thừa hưởng ở môi trường mọi khả năng của hệ thống. Làm cho thông tin lưu chuyển dễ dàng giữa các thành phần của hệ thống. Thông tin đầu ra của một module này có thể làm đầu vào cho một module khác. Mọi biến đổi trung gian đều do hệ thống đảm nhiệm và trong suốt với người sử dụng.

### *1.1.2. Tách thiết bị ngoại vi ra khỏi processor (micro hoá procesor)*

- Chuyển giao một số công việc cho thiết bị ngoại vi đảm nhiệm
- Processor tập trung xử lý bit
- Đề suất các thuật toán giải quyết các tác vụ trên bằng các phép xử lý bit, byte, hoàn thiện phương pháp xử lý trên máy tính điện tử
- Xây dựng sẵn các Modul chương trình cung cấp cho người dùng dưới dạng các chương trình chuẩn - thư viện các chương trình

Tuy nhiên trong thực tế khi các yêu cầu gia tăng thì các chương trình dưới dạng thư viện ngày càng tăng nên số lượng, nội dung của các thư viện tăng.

Giải pháp:

- Cung cấp cho người dùng các công cụ cho phép họ mô tả các giải thuật cần thiết, đồng thời cơ sở hoá các thư viện do đó ngôn ngữ thuật toán và chương trình dịch ra đời
- Người dùng có thể tác động lên máy tính điện tử thông qua các chương trình mẫu hoặc chương trình dịch

### *1.1.3. Chuyển nguyên tắc Lệnh thành Menu*

Cơ chế ra lệnh

- Người dùng phải tự nắm bắt trước các công việc mà hệ thống có thể làm được, qua đó chỉ thị cho hệ thống làm việc.

Cơ chế Menu

- Hệ thống giới thiệu cho người dùng các khả năng phục vụ của mình dưới dạng các bảng chọn, người dùng chỉ chờ cho hệ thống trình bày danh mục các công việc và lựa chọn công việc có thể yêu cầu
- Các công việc được phân nhóm theo từng phạm trù để dễ tìm kiếm
- Hệ thống mang tính chất tự đào tạo: càng làm việc càng hiểu sâu hơn

Nguyên tắc xây dựng Menu

*Bằng lời:*

- ✓ Dùng lời chỉ chính xác công việc sẽ thực hiện, tổ chức độ phân giải tốt
- ✓ Dễ thực hiện
- ✓ Chịu hàng rào ngôn ngữ

*Bằng biểu tượng:*

- ✓ Mỗi công việc được miêu tả bằng một hình ảnh
- ✓ Hấp dẫn, dễ hiểu với mọi loại đối tượng
- ✓ Chống được hàng rào ngôn ngữ
- ✓ Khó tổ chức và độ phân giải thấp

*Khắc phục nhược điểm của hai hình thức tổ chức trên: tổ chức cả hai hình thức:*

- ✓ Khi đưa hộp sáng hay khung tích cực tới một biểu tượng thì dòng chú thích xuất hiện
- ✓ Khi đưa hộp sáng hay khung tích cực áp vào một mục nào đó bằng lời thì biểu tượng xuất hiện

Ngoài ra còn tồn tại cơ chế phím nóng, lệnh chuẩn

***Tóm lại***: Hệ thống phải có trách nhiệm đảm bảo các điều kiện vật chất về các chương trình có thể thực hiện được đồng thời phải duy trì hệ thống ở trạng thái đồng bộ (có nghĩa là hệ thống phải có chức năng quản lý tài nguyên)

## 1.2. Tài nguyên hệ thống

Bao gồm:

- Không gian: Không gian nhớ
- Thời gian: Thời gian thực hiện lệnh
- Thiết bị ngoại vi

### *1.2.1. Bộ nhớ*

- Bộ nhớ là nơi lưu trữ thông tin.
- Đặc trưng bộ nhớ
  - ✓ Thời gian truy nhập
  - ✓ Phân cấp
  - ✓ Phân loại
- Thời gian truy nhập
  - ✓ Thời gian truy nhập trực tiếp: thời gian trực tiếp để truy nhập tới địa chỉ bất kỳ trong bộ nhớ.
  - ✓ Thời gian truy nhập tuần tự: Khi tồn tại một cách tổ chức lưu trữ kế tiếp.
- Phân cấp bộ nhớ
  - ✓ Bộ nhớ thường được phân cấp theo tốc độ truy nhập trực tiếp hay kế tiếp. Bộ nhớ được gọi là thực hiện nếu processor có thể thực hiện câu lệnh bất kỳ ghi trong đó. Đặc điểm của bộ nhớ này là thời gian truy nhập thực hiện và truy nhập tuần tự là bằng nhau. Bộ nhớ trong bao giờ cũng là bộ nhớ thực hiện.
  - ✓ Không gian bộ nhớ
  - ✓ Giá thành

- Phân loại bộ nhớ
  - ✓ Bộ nhớ trong: Có tốc độ truy nhập cao nhưng không gian bộ nhớ nhỏ
  - ✓ Bộ nhớ ngoài: Có không gian bộ nhớ lớn nhưng tốc độ truy nhập thấp.

  Thời gian truy nhập trực tiếp thường lớn hơn thời gian truy tuần tự. Loại bộ nhớ phổ biến là bộ nhớ đĩa cứng, đĩa mềm, băng từ, đĩa quang.

### *1.2.2. Thời gian thực hiện lệnh*

- Processor là một tài nguyên quan trọng của hệ thống, được truy nhập ở mức câu lệnh và chỉ có nó mới làm cho câu lệnh được thực hiện.
- Processor được dùng cho nhiều tiến trình khác nhau do đó việc phân chia thời gian sử dụng processor của mỗi tiến trình phải được tối ưu hoá, đặc biệt là khi chúng còn dùng chung tài nguyên khác: Chương trình, dữ liệu, thiết bị vào ra...
- Thời gian: thời gian thực hiện một câu lệnh
- Trong hệ thống có nhiều processor thì thời gian của mỗi processor được quản lý và phân phối riêng biệt như những tài nguyên độc lập

### *1.2.3. Thiết bị ngoại vi*

- Số lượng nhiều
- Chủng loại đa dạng
- Tốc độ xử lý << tốc độ processor
- Các thiết bị tiếp nhận, lưu trữ thông tin ở bộ nhớ ngoài trong thời gian dài được gọi là thiết bị ngoại vi (Máy in, bàn phím, màn hình, chuột, modem,… ). Chúng còn được gọi là thiết bị vào ra. Chúng thường được gắn với MTDT thông qua các thiết bị trung gian (các thiết bị quản lý, thiết bị điều khiển).
- Tài nguyên có hai loại: Phân chia được và không phân chia được.
  - ✓ Phân chia được: Cho phép nhiều người hay Chương trình sử dụng nó một cách đồng thời. Điển hình là bộ nhớ(trong và ngoài): có thể nạp nhiều Chương trình vào bộ nhớ trong, hay 1 Chương trình sử dụng nhiều tệp trên đĩa cứng.
  - ✓ Không phân chia được: phần lớn các tài nguyên còn lại. Tuy nhiên có thể phân phối việc sử dụng chúng sao cho người sử dụng cảm giác như được phục vụ đồng thời.

## 1.3. Định nghĩa hệ điều hành

Hệ điều hành là một phần quan trọng của mọi hệ thống thông tin. Một hệ thống thông tin gồm 4 thành phần: phần cứng, hệ điều hành, Chương trình ứng dụng, người sử dụng

Phần cứng: CPU, bộ nhớ, thiết bị vào ra cung cấp các tài nguyên thông tin cơ sở.

Các Chương trình ứng dụng: Chương trình dịch, hệ thống cơ sở dữ liệu, trình soạn thảo văn bản . qui định cách sử dụng các tài nguyên đó để giải quyết những vấn đề của người sử dụng.

Hệ điều hành điều khiển và đồng bộ việc sử dụng phần cứng của các Chương trình ứng dụng phục vụ các người sử dụng khác nhau với các mục đích sử dụng phong phú đa dạng.

Ta có thể hiểu Hệ điều hành là Hệ thống các Chương trình đảm bảo các chức năng **giao tiếp người máy** và **quản lý tài nguyên hệ thống tính toán**.

Tuy nhiên đứng dưới các góc độ khác nhau nên có nhiều cách tiếp cận khác nhau khi định nghĩa về hệ điều hành:

### *1.3.1. Với người dùng*

Hệ điều hành là hệ thống chương trình tạo điều kiện để khai thác tài nguyên hệ thống tính toán một cách dễ dàng, thuận tiện

Người sử dụng khi thực hiện một Chương trình nào đó trên máy tính điện tử thì chỉ quan tâm đến việc hệ thống có đáp ứng được nhu cầu của họ hay không? Có Chương trình

cần thực hiện, có đủ bộ nhớ để chạy Họ không quan tâm đến việc hệ điều hành làm gì nhằm mục đích gì, có cấu trúc như thế nào?

### *1.3.2. Với người quản lý*

Hệ điều hành là tập các chương trình phục vụ quản lý chặt chẽ và sử dụng tối ưu các tài nguyên hệ thống

### *1.3.3. Với cán bộ kỹ thuật*

Hệ điều hành là hệ thống chương trình trang bị cho một máy tính cụ thể mức vật lý để tạo ra một máy logic mới với các tài nguyên và khả năng mới.

### *1.3.4. Với cán bộ lập trình hệ thống*

Hệ điều hành là một hệ thống mô hình hoá mô phỏng các hoạt động của máy, của người dùng và của thao tác viên 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

Đối với các cán bộ lập trình hệ thống, vị trí của họ là ở bên trong hệ điều hành. Họ quan sát các module, các thành phần của hệ thống, quan sát mối quan hệ giữa chúng. Đây là quan điểm của chúng ta trong suốt quá trình khảo sát nghiên cứu hệ điều hành.

***Tóm lại:***

Hệ điều hành là một hệ chuyên gia ra đời sớm nhất và hoàn thiện nhất vì hai yếu tố:

- ✓ Vấn đề mà hệ điều hành giải quyết nảy sinh từ những người làm tin học do đó bài toán chính xác và rõ ràng.
- ✓ Người tham gia thiết kế chương trình là các cán bộ lập trình có tay nghề cao.

## 1.4. Phân loại hệ điều hành

Bao gồm:

- ✓ Hệ điều hành đơn nhiệm và hệ điều hành đa nhiệm
- ✓ Hệ điều hành đơn Chương và hệ điều hành đa Chương (MultiUsers)
- ✓ Hệ điều hành tập trung và hệ điều hành phân tán
- ✓ Hệ điều hành phân chia thời gian và hệ điều hành thời gian thực

### *1.4.1. Hệ điều hành đơn nhiệm và hệ điều hành đa nhiệm*

Dựa vào cách thức đưa Chương trình vào bộ nhớ, chọn Chương trình có sẵn trong bộ nhớ để processor thực hiện, người ta phân thành: hệ điều hành đơn nhiệm, đa nhiệm.

*Hệ điều hành đơn nhiệm*

- Tại một thời điểm xác định, khi một Chương trình được đưa vào bộ nhớ thì nó chiếm giữ mọi tài nguyên của hệ thống, và vì vậy Chương trình khác không thể được đưa vào bộ nhớ trong khi nó chưa kết thúc.
- Nhưng do các thiết bị vào ra thường làm việc với tốc độ chậm, người ta dùng kỹ thuật SPOOLING (simultanous peripheral Operation on line): cho phép tạo ra hiệu ứng song song các thiết bị chỉ cho phép vào ra tuần tự (sẽ đề cập chi tiết ở Chương sau).

*Hệ điều hành đa nhiệm*

- Hệ điều hành cho phép tại một thời điểm có nhiều Chương trình ở trong bộ nhớ trong. Chúng có nhu cầu được phân phối thời gian phục vụ CPU, bộ nhớ và thiết bị ngoại vi. Như vậy CPU, bộ nhớ, thiết bị ngoại vi v.v.. là các tài nguyên được chia sẻ cho các Chương trình đó. Vấn đề là làm sao đảm bảo tốt nhất tính bình đẳng khi giải quyết vấn đề phân phối tài nguyên.

### *1.4.2. Hệ điều hành đơn Chương và hệ điều hành đa Chương (MultiUsers)*

*Hệ điều hành đơn chương*

- Tại một thời điểm xác định hệ điều hành chỉ cho phép một người sử dụng thao tác

mà thôi.

*Hệ điều hành đa chương*

- Hệ điều hành cho phép tại một thời điểm có thể phục vụ nhiều người sử dụng.

### *1.4.3. Hệ điều hành tập trung và hệ điều hành phân tán*

*Hệ điều hành tập trung*

- Trên một hệ thống máy tính chỉ có một HĐH duy nhất cài ở máy chủ. Các máy trạm được khởi động nhờ máy chủ và nó chỉ làm chức năng nhập/xuất dữ liệu. Mọi xử lý đều tập trung ở máy chủ.

*Hệ điều hành phân tán*

- Trên mỗi máy có 1 hệ điều hành khác nhau, máy chủ chịu trách nhiệm cung ứng các dịch vụ để truy nhập đến các tài nguyên chung và điều hành toàn
- hệ thống, các phép xử lý có thể tiến hành ở máy trạm.

### *1.4.4. Hệ điều hành phân chia thời gian và hệ điều hành thời gian thực*

*Hệ điều hành phân chia thời gian (Share time)*

- Một CPU luôn phiên phục vụ các tiến trình và 1 tiến trình có thể rơi vào trạng thái chờ đợi khi chưa được phân phối CPU.

*Hệ điều hành thời gian thực (Real time)*

- Một tiến trình khi đã xâm nhập vào hệ thống thì ở bất kỳ lúc nào đều được phân phối CPU.

## 1.5. Tính chất chung của hệ điều hành

### *1.5.1. Độ tin cậy cao*

Mọi hoạt động thông báo của hệ điều hành chuẩn xác tuyệt đối

Khi chắc chắn đúng thì máy mới cung cấp thông tin cho người dùng

Mọi công việc bao giờ cũng được kiểm tra, đánh giá

***Ví dụ: C:\>COPY A:\ F1.TXT B:***

Kiểm tra lệnh COPY
Kiểm tra các điều khiển
Tồn tại hay không các ổ đĩa
Động cơ có quay không
Đĩa có truy nhập được không
Tồn tại hay không tệp tin f1.txt
Chất lượng thông tin trên đĩa như thế nào?
Đọc một phần thông tin trong F1.TXT hay toàn bộ
…

### *1.5.2. Độ an toàn*

Tổ chức cho dữ liệu và chương trình không bị xoá hoặc thay đổi ngoài ý muốn.

Chức năng bảo vệ thông tin được chia thành nhiều mức:

- *Các mức do hệ thống đảm nhiệm:* Ví dụ: trong các hệ thống UNIX, khi muốn xoá hay sửa đổi nội dung một tệp, người sử dụng phải có quyền xoá sửa đối với file đó.
- *Các mức do người sử dụng đảm nhiệm:* Ví dụ: Lệnh DEL *.* của MSDOS, hệ thống hỏi lại người sử dụng một lần nữa để tránh sai sót vô ý.

### *1.5.3. Hiệu quả*

Các tài nguyên phải được khai thác triệt để ngay cả khi điều kiện tài nguyên hạn chế song vẫn có thể giải quyết các yêu cầu phức tạp.

Tính đồng bộ cao (duy trì đồng độ trong toàn bộ hệ thống)

### *1.5.4. Tổng quát*

Tính kế thừa các phiên bản trước đây

Thích nghi với những thay đổi có thể có trong tương lai

### *1.5.5. Thuận tiện*

- Dễ sử dụng
- Có nhiều mức hiệu quả khác nhau tuỳ kinh nghiệm và kiến thức người dùng:
  - ✓ Giao tiếp dạng dòng lệnh
  - ✓ Giao tiếp dạng thực đơn (Menu)
  - ✓ Giao tiếp dạng biểu tượng

## 1.6. Nguyên tắc xây dựng hệ điều hành

### *1.6.1. Modul*

Xây dựng từ các Modul độc lập quan hệ với nhau thông qua dữ liệu Vào/ra

Tồn tại cơ chế liên kết các Modul độc lập thành hệ thống có tổ chức

### *1.6.2. Nguyên tắc tương đối trong định vị*

Các Modul được viết theo địa chỉ tương đối kể từ đầu bộ nhớ, khi thực hiện chúng được định vị tại vùng nhớ cụ thể như vậy hệ thống sử dụng bộ nhớ linh hoạt hơn và hệ điều hành không phụ thuộc vào cấu hình bộ nhớ

### *1.6.3. Macroprocessor*

Khi có một công việc cụ thể, hệ thống sẽ:

- ✓ Xây dựng các phiếu yêu cầu
- ✓ Liệt kê các bước phải thực hiện
- ✓ Xây dựng chương trình tương ứng
- ✓ Thực hiện chương trình

Ví dụ: Trong MSDOS ta có các tệp config.sys và autoexec.bat

### *1.6.4. Phủ chức năng*

Một công việc của hệ điều hành có thể được thực hiện bằng nhiều phương tiện khác nhau cho phép người dùng chọn giải pháp tối ưu với bài toán của mình

Ví dụ: Khi in tệp F1.TXT có các giải pháp:

```
C:\>COPY  F1.TXT  PRN
C:\>TYPE  F1.TXT  >PRN
C:\>PRINT  F1.TXT
```

### *1.6.5. Giá trị chuẩn (ngầm định):*

Hệ thống chuẩn bị sẵn các bảng giá trị cho các tham số điều khiển

Nếu trong các câu lệnh của người dùng còn thiếu những tham số giá trị thì hệ thống sẽ tự động lấy giá trị tương ứng ở bảng giá trị chuẩn ra để thực hiện

Ví dụ: C:\BT> DIR

| | |
|---|---|
| Xem ổ đĩa nào: | C |
| Thư mục nào: | BT |
| Cái gì: | Mọi thư mục con, tệp trong thư mục này và không bị che |

| | |
|---|---|
| Như thế nào: | Đầy đủ thông tin, liên tục theo dữ liệu |
| Ra đâu: | Thiết bị chuẩn |
| Tham số: | Mọi tham số |

### *1.6.6. Tham số*

- Tham số vị trí: Là loại tham số mà ý nghĩa của nó xác định bởi vị trí xuất hiện trong bảng tham số. Đứng đầu dòng tham số

- Tham số khoá: Là loại tham số mà ý nghĩa xác định bằng từ khóa

Ví dụ: C:\>DIR D: /W/A/P

C:\>DIR D: /A/P/W

Trong đó:

| | |
|---|---|
| D: | là tham số vị trí |
| /W, /A hay /P | là tham số khoá |

### *1.6.7. Nguyên lý bảo vệ*

- Chương trình và dữ liệu phải được bảo vệ nhiều mức, bằng nhiều khoá.
- Ví dụ trong Linux
  + Mức 1: Người sử dụng phải có tài khoản mới được sử dụng máy tính.
  + Mức 2: Chỉ những người sử dụng thuộc nhóm A mới được truy nhập và tệp chung của nhóm A.

## 1.7. Thành phần hệ điều hành

### *1.7.1. Thành phần của hệ điều hành*

- Ngôn ngữ làm việc và giao tiếp: Hệ điều hành có quan hệ với ba đối tượng nên tồn tại ba ngôn ngữ làm việc và giao tiếp
  - ✓ *Ngôn ngữ máy (Ngôn ngữ thực hiện)*:
    Là ngôn ngữ thực hiện duy nhất của hệ thống. Mọi ngôn ngữ khác đều phải được ánh xạ sang ngôn ngữ thực hiện
  - ✓ *Ngôn ngữ vận hành (hệ điều hành):*
    Thao tác viên giao tiếp với hệ thống
  - ✓ *Ngôn ngữ thuật toán:*
    Người dùng giao tiếp với hệ thống: Pascal, C... (Cần phải có chương trình dịch).
- Các Modul chương trình của hệ thống có thể chia thành hai lớp:
  - ✓ *Chương trình điều khiển:*
    + Quản lý tài nguyên
    + Quản lý tiến trình
    + Quản lý, tổ chức dữ liệu
    + Chương trình thư ký, điều phối nhiệm vụ
  - ✓ *Chương trình phục vụ:*
    + Chương trình biên tập
    + Chương trình dịch

### *1.7.2. Thành phần của MSDOS*

Những năm 1980, khi hãng Intel cho ra đời bộ vi xử lý 16 bít 8086, Jim Paterson xây dựng hệ điều hành trang bị cho loại máy tính sử dụng bộ vi xử lý này đó là 86-DOS.

Hãng Microsoft đã mua lại hệ điều hành của Jim Paterson và phát triển thành hệ điều hành PC-DOS hay MSDOS. Phiên bản đầu tiên của MSDOS thế hệ 1.0 ra đời vào 8/1981.

- Các cải tiến cơ bản của MSDOS 1.0
  - ✓ Có thêm loại Chương trình chạy EXE bên cạnh các Chương trình COM.

- ✓ Hệ điều hành đã tách bộ xử lý lệnh thành một phần nội trú và một phần ngoại trú.
- ✓ Để tiện lợi cho việc quản lý đĩa người ta đưa ra bảng File Allocation Table viết tắt là FAT để quản lý đĩa. Mỗi phần tử của bảng FAT tương ứng với 521 byte trên đĩa gọi là sector, chỉ ra sector này đã có dữ liệu hay còn tự do.
- ✓ MSDOS 1.0 cho phép xử lý lô (batch) một số lệnh của MSDOS bằng cách tạo một tệp batch.
- ✓ Ngày tháng tạo hay cập nhật tệp cũng được lưu trữ cùng với thông tin của tệp.

- Cùng với thời gian, hãng Microsoft đã nâng cấp hệ điều hành này lên các phiên bản mới 2.0, 3.0, 4.0…

- Các thành phần của MSDOS
  - ✓ BIOS: Chứa các Chương trình của supervisor và quản lý tệp nhưng chưa kết nối thành hệ thống. Do đó cần Chương trình kích hoạt.
  - ✓ Chương trình mồi Boot Strap Loader: nằm ở sector đầu tiên của đĩa từ dùng để kích hoạt toàn bộ Chương trình hệ thống.
  - ✓ IO.SYS: Dưới sự hỗ trợ của BSL bao lấy BIOS, cung cấp các dịch vụ cơ bản nhất như chia sẻ tài nguyên, quản lý bộ nhớ.
  - ✓ MSDOS.SYS: mở rộng IO.SYS lần nữa
  - ✓ COMMAND.COM: liên lạc giữa người sử dụng và hệ thống, chứa các lệnh nội trú.
  - ✓ Các lệnh ngoài: là thành phần mở rộng theo từng lĩnh vực.
  - ✓ Các tiện ích khác: Chương trình nén đĩa (DBLSPACE)…

## CÂU HỎI VÀ BÀI TẬP

1.1. Hãy liệt kê sơ bộ về một số đặc trưng của các hệ điều hành đã sử dụng.

1.2. Trình bày các đặc trưng của CPU, bộ nhớ, kênh dẫn

1.3. Những đại lượng nào liên quan đến tốc độ xử lý của CPU

1.4. Anh, chị hãy lấy ví dụ minh họa về các tính chất của hệ điều hành đang sử dụng cụ thể

1.5. Anh, chị hãy trình bày về các nguyên tắc xây dựng hệ điều hành. Lấy ví dụ minh họa cụ thể.

1.6. Anh, chị hãy lấy ví dụ minh họa về các thành phần cơ bản của hệ điều hành đang sử dụng cụ thể. Nêu ý nghĩa, tác dụng của các thành phần đó.

# Chương II: QUẢN LÝ THIẾT BỊ

***Đặt vấn đề***

- Thiết bị ngoại vi đã trở thành đối tượng làm việc của hệ điều hành khi hệ thống phức tạp
- Các thiết bị ngoại vi đảm nhiệm việc truyền thông tin qua lại giữa các bộ phận của hệ thống

⇨ Vì vậy vấn đề tổ chức thông tin, phương pháp truy nhập tới chúng như thế nào

***Đề cập:***

- Tổ chức thiết bị ngoại vi
- Chiến lược điều khiển
- Phương pháp phát hiện và xử lý lỗi

## 2.1. Quan hệ phân cấp trong tổ chức và quản lý thiết bị ngoại vi

### *2.1.1. Sự đa dạng của các thiết bị ngoại vi:*

- Chuẩn: bắt buộc
- Phụ: bổ sung

### *2.1.2. Quan hệ giữa vi xử lý với thiết bị ngoại vi*

- Vi xử lý không thể làm việc trực tiếp với các thiết bị ngoại vi
- Vi xử lý cùng với thiết bị ngoại vi thực hiện các thao tác vào/ra

⇨ Tồn tại cách tổ chức sao cho vi xử lý không phụ thuộc vào các biến động của thiết bị ngoại vi

***Nguyên tắc:***

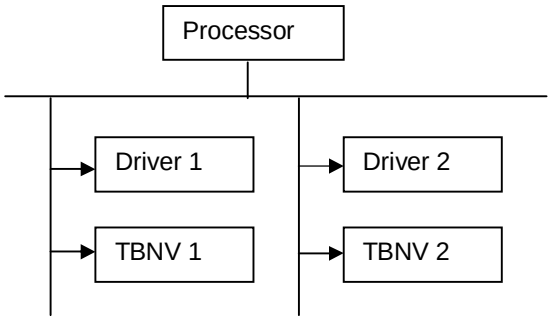
- Vi xử lý chỉ điều khiển các thao tác vào/ra chứ không trực tiếp thực hiện
- Các thiết bị ngoại vi không trực tiếp gắn vào vi xử lý mà gắn với thiết bị quản lý
- Một thiết bị điều khiển và các thiết bị ngoại vi phụ thuộc nó tạo thành một kênh (channel)

***Như vậy***: thiết bị quản lý đóng vai trò như một máy tính chuyên dụng:

- Nhiệm vụ điều khiển thiết bị ngoại vi
- Có ngôn ngữ riêng, lệnh riêng
- Thiết bị ngoại vi và thiết bị điều khiển hoạt động độc lập với nhau và độc lập với vi xử lý
- Chương trình viết trên ngôn ngữ thiết bị điều khiển và thiết bị ngoại vi gọi là chương trình kênh (channel program)

### *2.1.3. Thực hiện các phép vào/ra*

Vi xử lý tạo ra một chương trình tương ứng với công việc cần thực hiện, sau đó chuyển giao chương trình kênh và dữ liệu tương ứng cho thiết bị điều khiển và tiếp tục thực hiện chương trình của mình



Các phép vào/ra được điều khiển theo nguyên lý Macroprocessor cho phép trong lúc các phép vào/ra được thực hiện ở thiết bị ngoại vi thì vi xử lý vẫn hoạt động song song (thực hiện các tính toán và điều khiển khác khi chưa cần đến kết quả vào/ra)

Khi công việc được hoàn thành báo cho vi xử lý biết bằng tín hiệu ngắt. Tuỳ theo tín hiệu ngắt:

- ✓ Vi xử lý ngắt ngay
- ✓ Lưu trữ để chờ xử lý sau đó
- ✓ Huỷ bỏ

Để hệ thống có thể làm việc với các kênh vi xử lý phải biết ngôn ngữ kênh (ngôn ngữ được đưa vào hệ thống khi nạp hệ điều hành)

***Ví dụ: MSDOS***

Trong CONFIG.SYS

***DEVICE =...***

Đảm bảo tương tác chặt chẽ giữa thiết bị ngoại vi và vi xử lý thì kênh phát tín hiệu ngắt vào/ra, nó luôn luôn bảo vệ hệ thống một trị số qua đó có thể đánh giá chất lượng thực hiện phép vào/ra: mã trở về (return code). vi xử lý tạm dừng công việc của mình và chuyển sang phân tích mã trở về để đánh giá kết quả, chất lượng công việc

### *2.1.4. Kết thúc chương trình kênh*

Các lệnh trong chương trình kênh kết thúc khác nhau nên một phép vào/ra có thể thúc ở nhiều mức vì vậy kênh báo cho hệ thống biết kết quả phép vào/ra càng sớm càng tốt

Các chương trình ứng dụng, chương trình ngắt vào/ra, chương trình kênh tạo thành các tiến trình độc lập, hoạt động song song và chịu sự điều độ chung của hệ thống.

## 2.2. Cơ chế phòng đệm (Buffer)

Đặc điểm của thiết bị ngoại vi là tốc độ chậm (nhỏ hơn rất nhiều so với tốc độ của vi xử lý) do đó khi một thiết bị ngoại vi làm việc hệ thống cần:

- ✓ Kích hoạt thiết bị ngoại vi
- ✓ Chờ thiết bị ngoại vi đạt trạng thái thích hợp

Để đảm bảo hiệu suất sử dụng, hệ thống cần phải:

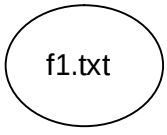
- ✓ Giảm số lượng các phép vào/ra vật lý
- ✓ Thực hiện song song các phép vào/ra và xử lý thông tin khác
- ✓ Thực hiện trước các phép nhập dữ liệu

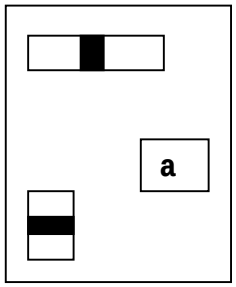
Như vậy tồn tại một số vùng nhớ trung gian làm nơi lưu trữ thông tin trong các phép vào/ra gọi là phòng đệm

- ✓ Cơ chế phòng đệm cho phép khắc phục:
- ✓ Thực hiện trước các phép nhập dữ liệu
- ✓ Tích luỹ kết quả ra
- ✓ Đảm bảo xử lý song song giữa các phép trao đổi vào/ra và xử lý
- ✓ Giảm số lần truy nhập vật lý
- ✓ Đảm bảo biến đổi topo thực hiện trước hoặc sau khi xử lý thông tin mà không làm mất tính liên tục của thông tin

Với vi xử lý thì phòng đệm chính là các thanh ghi

***Ví dụ:***



***Phân loại:***

- ✓ Phòng đệm trung gian
- ✓ Phòng đệm xử lý
- ✓ Phòng đệm vòng

### *2.2.1. Phòng đệm trung gian:*

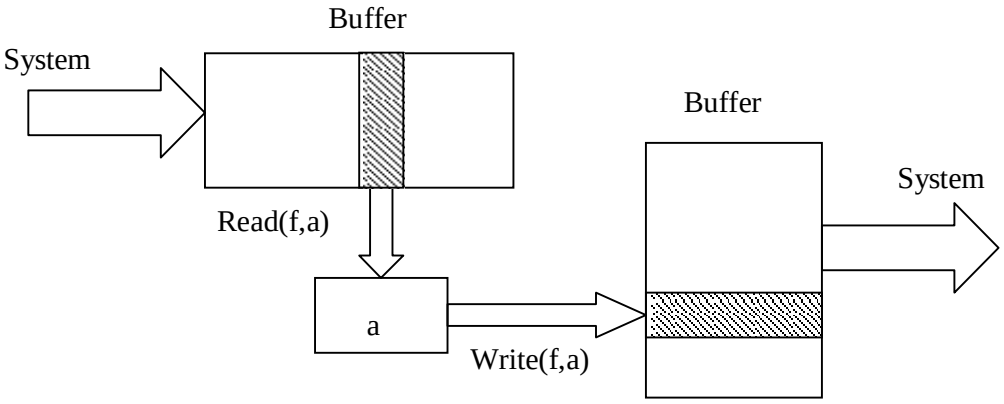
Lưu trữ tạm thời kết quả vào/ra như vậy tồn tại hai cơ chế phòng đệm:

Phòng đệm vào

- Phòng đệm chỉ dùng để nhập thông tin. Trong hệ thốngsẽ có lệnh để đưa thông tin vào phòng đệm (đọc vật lý).
- Khi gặp chỉ thị đọc (READ), thông tin sẽ được tách và chuyển từ phòng đệm vào các địa chỉ tương ứng trong Chương trình ứng dụng. Như vậy, mỗi giá trị được lưu trữ ở hai nơi trong bộ nhớ (một ở phòng đệm và một ở vùng bộ nhớ trong Chương trình ứng dụng). Khi giá trị cuối cùng của phòng đệm vào được lấy ra thì phòng đệm được giải phóng (rỗng) và hệ thống đưa thông tin mới vào phòng đệm trong thời gian ngắn nhất có thể.
- Để giảm thời gian chờ đợi, hệ thống có thể tổ chức nhiều phòng đệm vào, khi hết thông tin ở một phòng đệm, hệ thống sẽ chuyển sang phòng đệm khác.

Phòng đệm ra

- Khi có chỉ thị ghi (WRITE), thông tin được đưa vào phòng đệm. Khi phòng đệm ra đầy, hệ thống sẽ đưa thông tin ra thiết bị ngoại vi.
- Hệ thống cũng có thể tổ chức nhiều phòng đệm ra.




Ưu điểm:

- Đơn giản
- Hệ số song song cao (do tốc độ giải phóng vùng đệm lớn)
- Vạn năng, áp dụng cho mọi phép vào/ra

Nhược

- Tốn bộ nhớ
- Thời gian trao đổi
- Nhiều lỗi xử lý

### *2.2.2. Phòng đệm xử lý:*

Thông tin được xử lý ngay trong phòng đệm

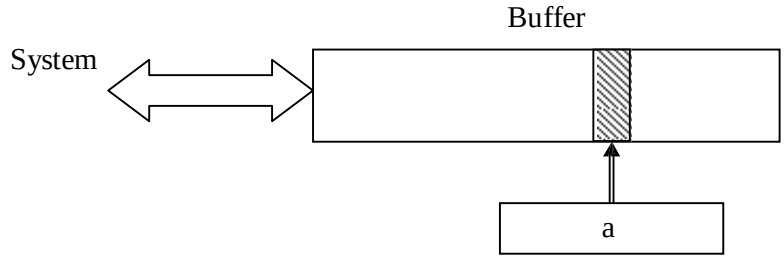
Truy nhập thông tin theo địa chỉ (tính địa chỉ của thông tin trong phòng đệm và cung cấp cho chương trình)

Ưu điểm:

- Tiết kiệm bộ nhớ
- Không mất thời gian chuyển thông tin ỏ bộ nhớ trong

Nhược:

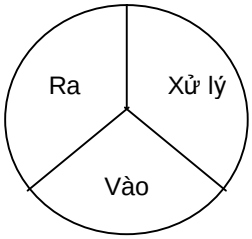
- Hệ số song song thấp
- Tốc độ giải phóng phòng đệm chậm
- Tính vạn năng không cao




### *2.2.3. Phòng đệm vòng*

Kết hợp cả 2 loại phòng đệm trên

Tổ chức 3 phòng đệm




Sau một khoảng thời gian ba phòng đệm quay vòng tròn

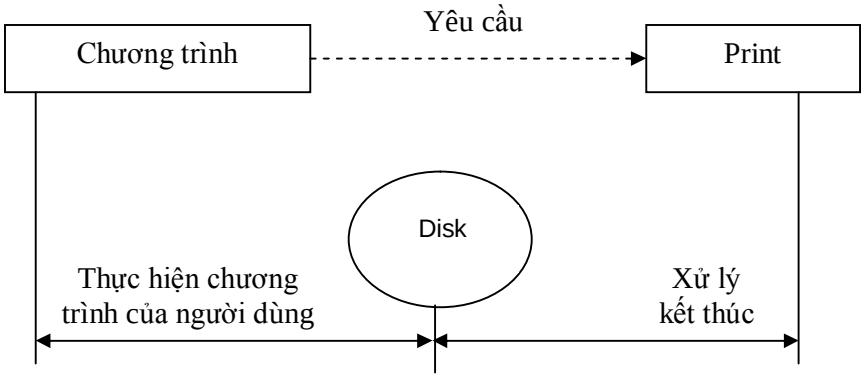
Tổ chức:

- Phòng đệm có thể gắn với từng file cụ thể: chỉ được xây dựng khi mở file hoặc đóng file
- Phòng đệm gắn với hệ thống: khi xây dựng hệ thống thì xây dựng ngay cơ chế phòng đệm và chỉ gắn vào một file cụ thể nào đó

## 2.3. Cơ chế SPOOL (Simultaneous Peripheral Operation On_Line - Hệ thống mô phỏng các phép trao đổi thiết bị ngoại vi trong chế độ trực tiếp)

Vai trò của thiết bị ngoại vi: trạm nhận chương trình kênh và dữ liệu, gửi các mã trạng thái cho hệ thống phân tích

Tuy nhiên: mọi chương trình và dữ liệu của thiết bị ngoại vi hoạt động tương tự như thiết bị ngoại vi có thực vì vậy có thể dùng phần mềm để mô phỏng hoạt động của thiết bị ngoại vi và coi nó như một thiết bị ngoại vi ảo.




Ứng dụng:

- Mô phỏng quá trình điều khiển, quản lý thiết bị ngoại vi
- Tạo ra các SPOOL, mô phỏng các phép trao đổi ngoại vi ngay trong lúc thực hiện

SPOOL: kỹ thuật xử lý mà thiết bị cuối trong chương trình của người dùng được tạm thời thay thế bởi thiết bị trung gian

- Sau khi kết thúc chương trình vào thời điểm thuận tiện thông tin sẽ được đưa ra thiết bị cuối theo yêu cầu của người dùng
- Không can thiệp vào chương trình của người dùng
- Tiến hành ngay trong lúc thực hiện phép trao đổi vào/ra

Tác dụng:

- Làm cho chương trình của người dùng thực hiện nhanh hơn
- Giảm giá thành chi phí
- Khai thác thiết bị ngoại vi tốt hơn
- Giảm yêu cầu về số lượng thiết bị
- Tạo ra kỹ thuật lập trình tương ứng

Tổ chức SPOOL: cơ chế thực hiện:

- Lưu kết quả đưa ra ở thiết bị trung gian, chuyển giao kết quả này ra phần xử lý kết thúc
- Lưu giữ chương trình kênh

## 2.4. Quản lý file

***Lý do:***

- Người dùng phải lưu trữ thông tin ở bộ nhớ ngoài vì vậy hệ điều hành phải có vai trò sao cho người dùng truy nhập thuận tiện
- Nhu cầu dùng chung các file dữ liệu

***Hệ quản lý file phải có các tính chất:***

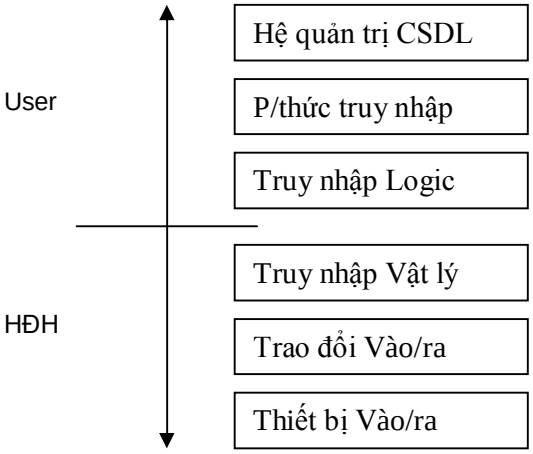
- Tính độc lập của file với vi xử lý và với thiết bị ngoại vi vì vậy hệ thống khi làm việc phải quản lý file theo tên
- Bảo vệ dữ liệu: không để mất thông tin khi có sự cố kỹ thuật hoặc chương trình thậm chí truy nhập bất hợp lệ
- Tổ chức có hiệu quả đảm bảo tiết kiệm bộ nhớ ngoài và dễ truy nhập
  - ✓ Tổ chức tuần tự theo byte: dữ liệu được tổ chức lưu trữ, đọc và ghi một cách tuần tự từng byte. Cách tổ chức này có tính vạn năng, mọi ứng dụng đều có thể sử dụng tệp.
  - ✓ Tổ chức tuần tự theo bản ghi: dữ liệu được tổ chức lưu trữ, đọc và ghi một cách tuần tự từng bản ghi với kích thước cố định.
  - ✓ Tổ chức cây các bản ghi: dữ liệu được tổ chức lưu trữ, đọc và ghi theo cây các bản ghi theo trường khoá.
- Mọi thao tác phức tạp phải "trong suốt " với người dùng đảm bảo công cụ truy nhập tới tay người dùng ở dạng đơn giản nhất

***Như vậy:***

- Tồn tại các câu lệnh: đọc, ghi, tạo, đổi tên, đóng, mở file...
- Tổ chức thông tin trên phương tiện mang tin và tự động ghi nhận sơ đồ
- Bố trí file để đáp ứng yêu cầu truy nhập và tìm kiếm
  - ✓ Cấu trúc lưu trữ tuần tự, tồn tại bản ghi đặc biệt lưu trữ các tham số file
  - ✓ Tồn tại cơ chế thư mục, bộ phận hoá tên file ở phạm vi nhất định, các Thông tin liên hệ với nhau bằng danh sách móc nối
- Có cơ chế bảo vệ file:
  - ✓ Tĩnh: liên quan tới toàn bộ file và cố định theo thời gian
  - ✓ Động: xác lập khi mở file đọc, ghi thông tin
- Xoá dữ liệu trong file:
  - ✓ Mức vật lý: toàn bộ nội dung file

- ✓ Mức logic: ngắt các móc nối liên hệ với file

***Phân lớp:***




Ở mức người dùng:

- Giao diện tốt
- Mang tính đặc thù của hệ thống

Mức hệ điều hành:

- Mang tính vạn năng
- Tồn tại nhiều thành phần, phụ thuộc vào thiết bị vì nó phải liên hệ với hệ thống

## 2.5. Quản lý file trong hệ điều hành MSDOS

Bộ nhớ ngoài (đĩa từ) có hai tham số chính:

- Tham số về thiết bị đọc đĩa từ
- Tham số về bản thân đĩa

### *2.5.1. Thiết bị đọc, ghi:*

Nguyên tắc hoạt động theo nam châm điện

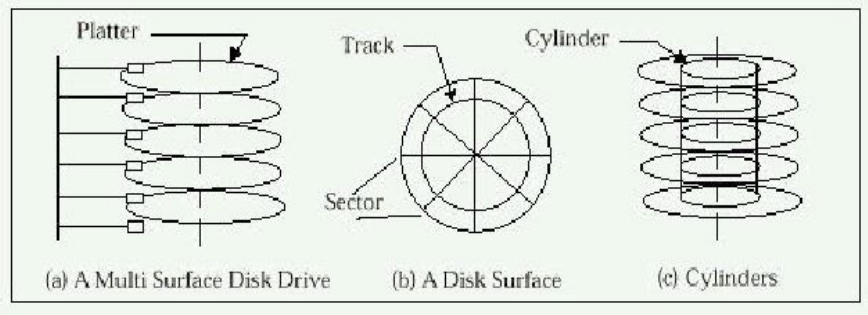
### *2.5.2. Tham số đĩa từ:*

Lưu trữ cố định trên đĩa.

Đĩa có thể có 1, 2 hay nhiều mặt (side), chúng được đánh số thứ tự từ 0. Mỗi mặt được truy nhập bằng một đầu từ (head).

Trên các mặt, thông tin được ghi theo các đường tròn đồng tâm (Track - rãnh từ) có địa điểm đầu thẳng hàng nhau. Chúng được đánh số thứ tự từ 0 và từ ngoài vào trong tâm đĩa.

Tập hợp các rãnh có bán kính bằng nhau trên các mặt tạo thành từ trụ (Cylinder)




Trên các rãnh, thông tin ghi theo từng phần một gọi là cung từ (sector) có độ dài bằng nhau, một sector có thể là 128, 256, 512, 1024… byte. Các sector được đánh số bắt đầu từ 1 (Hiện nay để nâng cao dung lượng lưu trữ trên đĩa từ, kỹ thuật LBA được sử dụng…)

Sector 1 không nằm cạnh sector 2 mà cách một khoảng nào đó gọi là hệ số đan xen (interleave). Interleave là số nguyên tố cùng nhau với số sector trên track

***Ví dụ:*** Đĩa mềm: Interleave=7

Đĩa cứng hệ số này từ 3 đến 4

Địa chỉ vật lý của 1 sector được xác định bởi:

- ✓ Số hiệu của Side/Head
- ✓ Số hiệu của Track/Cylinder
- ✓ Số hiệu của Sector

Trong thực tế còn sử dụng khái niệm liên cung (Cluster): Là số các sector liên tiếp nhau về mặt logic và là đơn vị phân phối bộ nhớ cho người dùng (1 cluster có thể là 2, 4, 8, 16, 32... sector). Địa chỉ logic

Địa chỉ logic của 1 sector còn được xác định bởi:

- ✓ Số hiệu của Cluster (tính từ Cylinder 0, Head 1)
- ✓ Số Sector/1 Cluster
- ✓ Số hiệu của Sector (tính từ đầu Cylinder)

***Đọc/ghi thông tin trên 1 sector của đĩa***

Sử dụng ngắt 13H của BIOS để đọc/ghi đĩa, với kiểu dữ liệu thanh ghi (Registers)

Giá trị các thanh ghi:

AH: 01h: Ghi Sector; 02h: Đọc Sector

AL: Số Sector cần đọc/ghi

CH: Số hiệu Track/Cylinder

CL: Số hiệu Sector

DH: Số hiệu đầu từ

DL: Số hiệu đĩa (F0h = A...; 80H = HD0; 81H = HD1)

ES:BX => địa chỉ vùng nhớ

Chú ý: Giá trị Sector gồm 6 bit và Cylinder là 10bit:

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Cylinder | | | | | | | | | | Sector | | | | | |

Thủ tục mã hoá Cylinder và Sector

```
Function CylSecEncode(Cylinder, Sector : Word) : Word;
Begin
    CylSecEncode := (Lo(Cylinder) shl 8) or (Hi(Cylinder) shl 6) or Sector;
End;
```

Output:

Nếu có lỗi: Carry Flag=CY=1 và mã lỗi trong AH

Nếu không lỗi: AH = 0 và ES:BX => địa chỉ vùng nhớ

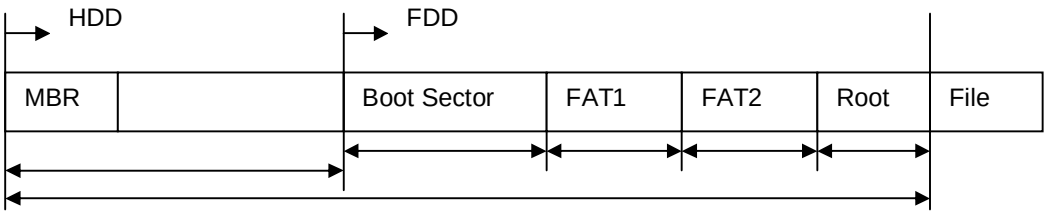
***Một đĩa cứng bao gồm:***

- Phần hệ thống
- Phần dữ liệu

Phần hệ thống bao gồm:

- Master boot record
- Boot sector
- FAT
- ROOT

Hình ảnh cấu trúc:




Vùng hệ thống

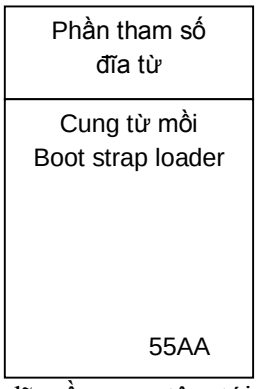
- MBR: Master Boot Boot chính của đĩa từ cứng
- MBR trỏ tới Boot Sector và những Boot Sector còn lại (nếu có)
- Boot Sector trỏ tới Root và từ Root truy nhập vào FAT1, từ FAT1 truy nhập tới File

Boot sector: luôn tồn tại ở mỗi đĩa từ

Nó bao gồm hai phần:

- Xác định tham số tổ chức của đĩa: đặc thù cho mỗi đĩa
- Chương trình mồi phục vụ cho việc nạp hệ điều hành: vùng này là bắt buộc với đĩa hệ thống, với đĩa ghi dữ liệu tì có thể bỏ trống

Nạp hệ thống: Thực chất là đọc Boot Sector và ghi vào vùng địa chỉ 7C00h




Để truy nhập thông tin trên đĩa cần quan tâm tới các thông số:

- Số byte cho một sector
- Số sector trước FAT
- Số bảng FAT
- Số mục vào (entry) cho root (32 byte cho một entry)
- Tổng số sector trên đĩa
- Số lượng sector cho một bảng FAT
- Số sector trên một track
- Số đầu đọc, ghi

Truy nhập Boot Sector

- Xác định vị trí của nó trên đĩa
- Đọc trực tiếp sector thông qua ngắt 13h hoặc 25h

Vị trí Boot Sector:

- Đĩa mềm: sec1, đầu đọc 0, cylinder 0
- Đĩa cứng: sec1, đầu đọc 1, cylinder 0

*A/ Bảng tham số BR:*

| Số hiệu | Địa chỉ offset | Chiều dài (byte) | Ý nghĩa |
|---|---|---|---|
| 1 | 0 | 3 | EBxx90 (số hiệu đặc biệt) |
| 2 | 3 | 8 | Tên hệ thống format đĩa |
| 3 | B | 2 | Số byte/sector (byte thấp được lưu trữ trước 1234→ 34\|12) |
| 4 | D | 1 | Sec/clus kích thước trong bảng phân phối cho người dùng |
| 5 | E | 2 | K/c từ đầu logic đĩa từ tới bảng FAT1 |
| 6 | 10 | 1 | Số bảng FAT |
| 7 | 11 | 2 | Số phần tử thư mục gốc root |
| 8 | 13 | 2 | Số sec trên đĩa nếu dung lượng đĩa nhỏ hơn 32MB |
| 9 | 15 | 1 | Loại đĩa: F8: HD; F9: FD(1.2M); F10: FD (1.44M) |
| 10 | 16 | 2 | Số sec/FAT |
| 11 | 18 | 2 | Số sec/track |
| 12 | 1A | 2 | Số đầu từ |
| 13 | 1C | 4 | Đ/c tuyệt đối boot sector |
| 14 | 20 | 4 | Số sec trên đĩa nếu dung lượng đĩa lớn hơn 32MB |
| 15 | 24 | 1 | Đ/c vật lý đĩa từ: 80: C, 81: D, 00: FD |
| 16 | 25 | 1 | Dự trữ |
| 17 | 26 | 1 | Dấu hiệu 29h |
| 18 | 27 | 4 | Serial number |
| 19 | 2B | 11 | Volume name |
| 20 | 36 | 8 | FAT |
| Còn lại 482 byte chứa Chương trình mồi | | | |

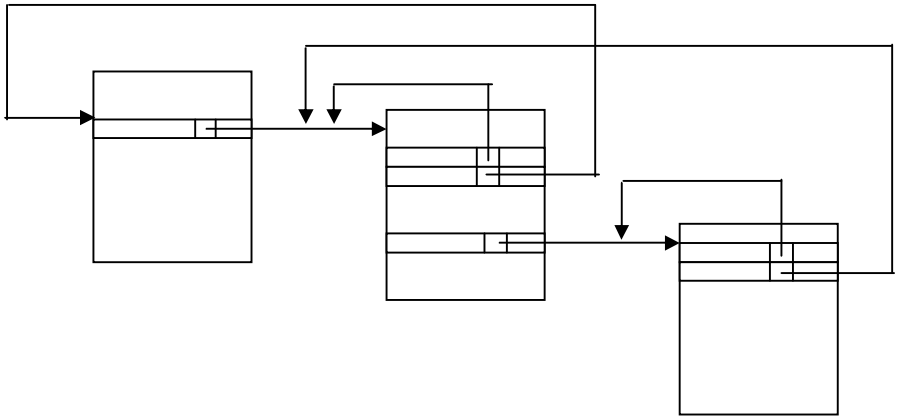
Ví dụ:

```
EB BC 90     4D 53 44 4F 53 35 2E 30    00 02    20
01 00     02     00 02     00 00     F8     80 00     3D 00
12 00     BC 19 00 00     0E 00 10 00     50     00     29
D2 15 BE 18     4E 4F 20 4E 41 4D 45 20 20 20 20
46 41 54 31 36 20 20 20 F1 33
```

*B/ Thư mục gốc (Root Directory)*

Dãy các mục vào, mỗi mục vào 32 byte chia thành:

| Số hiệu | Địa chỉ | Độ dài | ý nghĩa |
|---|---|---|---|
| 1 | 0 | 8 | Tên t/m, tệp (nếu thiếu bổ sung dấu cách 20h) |
| 2 | 8 | 3 | Phần mở rộng |
| 3 | B | 1 | Thuộc tính |
| 4 | C | 10 | Chưa dùng tới (với MSDOS6.22) |
| 5 | 16 | 2 | Giờ tạo lập |
| 6 | 18 | 2 | Ngày tạo lập |
| 7 | 1A | 2 | Chứa liên cung khởi động |
| 8 | 1C | 4 | Kích thước tệp |



***Byte số 0 trong tên thư mục, tệp:***

- Nếu là 00h thì phần tử này chưa sử dụng bao giờ
- Nếu là E5h thì phần tử này đã được sử dụng nhưng bị xoá
- Nếu là 2E 20h (.) phần tử đầu tiên của thư mục con. Liên cung khởi động (Starting cluster) của phần tử 1 chỉ chính nó
- Nếu là 2E 2Eh (..) phần tử thứ hai của thư mục con. Liên cung khởi động (Starting cluster) của phần tử 2 chỉ thư mục mẹ
- Liên cung khởi động (Starting cluster) của thư mục gốc với số hiệu: 00h

***Byte thuộc tính:***

| | A | D | V | S | H | R |
|---|---|---|---|---|---|---|
| Trọng số: | 32 | 16 | 8 | 4 | 2 | 1 |

A: thuộc tính lưu trữ

D: thư mục

V: Nhãn đĩa

S: hệ thống: các chương trình có đặc quyền hệ thống mới có thể truy nhập

H: ẩn đánh dấu một tệp bị che

R: chỉ đọc

Ví dụ:

Tệp IO.SYS với các thuộc tính: A S H R

Trọng số : 32 4 2 1 Giá trị: $(39)_{10}$:27h

Tệp COMMAND.COM với các thuộc tính: A

Trọng số : 32 Giá trị: $(32)_{10}$:20h

Thư mục TP : D

Trọng số : 16 Giá trị: $(16)_{10}$: 10h

<u>Kiểm tra thuộc tính một tệp:</u>

Đọc giá trị trong byte Attribute (At)

Thực hiện phép AND tương ứng với trọng số của các thuộc tính đó

Ví dụ:

Thuộc tính H: At AND 2<>0

Thư mục con: At AND 16<>0

<u>Gán thuộc tính một tệp:</u>

Thực hiện phép OR tương ứng với trọng số của các thuộc tính đó

Ví dụ:

Thuộc tính H: At OR 2

Xoá thuộc tính một tệp:

Thực hiện phép AND tương ứng mã bù1 tương ứng với thuộc tính đó

Ví dụ:

Thuộc tính R (thực hiện phép AND với 1111 1110): At AND FEh

Thuộc tính H (thực hiện phép AND với 1111 1101): At AND FDh

***Ngày, giờ tạo lập hệ thống:***

Byte Time: xxxxx xxxxxx xxxxx

giờ phút giây

Byte Date: xxxxxxx xxxx xxxxx

số năm tháng ngày (Giá trị năm tính từ năm 1980)

*C/ Bảng FAT*

Chức năng:

- Tạo danh sách móc nối các Cluster của cùng một tệp (quản lý bộ nhớ đã sử dụng)
- Quản lý bộ nhớ tự do (vùng bộ nhớ chưa dành cho tệp tin hay thư mục nào)
- Đánh dấu các Bad Cluster (nâng cao độ tin cậy đĩa)

Bao gồm:

- Dãy các phần tử, mỗi phần tử có thể là: 12,16, 32bit tương ứng cho FAT12, FAT16, FAT32.
- Các phần tử được đánh số 0,1,2..
- Từ phần tử thứ 2, mỗi phần tử trong FAT tương ứng với một Cluster và ngược lại

  FAT12: Dung lượng: $2^{12}$= 4096KB = 4MB

  FAT16: Dung lượng: $2^{16}$= 64MB với 2Sector/1Cluster

  $2^{16}$= 128MB với 4Sector/1Cluster

  $2^{16}$= 1024MB với 32Sector/1Cluster

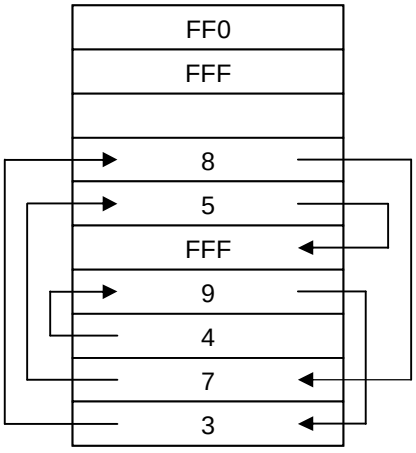
  FAT32: Dung lượng: $2^{32}$= 8GB với 2Sector/1Cluster
- Phần tử thứ nhất: tất cả các bit là 1 do đó

  Với đĩa cứng: HD: số hiệu FF8h

  Với đĩa mềm: FD: số hiệu FF0h
- Dấu hiệu kết thúc 1 chuỗi Cluster là FFFh hoặc FFFFh

Ví dụ: Đĩa mềm 1.44MB với FAT12

Starting Cluster: 6:




Để tăng tốc độ truy nhập, ngay lần truy nhập đầu tiên làm việc với đĩa từ, hệ thống sẽ

đọc luôn FAT và ROOT vào RAM, như vậy hệ thống chỉ cần truy nhập vào bộ nhớ để lấy thông tin, không cần phải truy nhập lại đĩa từ do vậy tăng được tốc độ và giảm được di chuyển cơ khí của đầu từ.

Đọc FAT

```
uses crt,MSDOS;
 const S16:string[16]='0123456789abcdef';
 var B:array[0..511]of byte;
 i,j:integer;start:word;
 drv,cyl,head,sec,numsec,drive:byte;
 Function R_sector (drive,cyl,head,sec,numsec:byte):integer;
 var reg:registers;
 begin
 with reg do
 begin
 dl:=drive;dh:=head;ch:=cyl;
 cl:=sec;al:=numsec;ah:=2;
 es:=seg (b);bx:=ofs (b);
 end;
 intr ($13,reg);
 end;
 Function R_fat (var start:word):word;
 var k,k1,k2,k3,tg,l:integer;ch:char;
 begin
 for i:=0 to 511 do b[i]:=0;
 if (drv=0)or (drv=1)then
 begin
 drive:=0;head:=0;
 end else
 begin
 drive:=$80;head:=1;
 end;
 i:=start;
 if (drv=0)or (drv=1)then
 begin
 j:= (i*3)div 2;k:=j div 512;
 tg:=R_sector (drive,0,head,2+k,1);
 j:=j mod 512;
 l:=memw[seg (b[j]):ofs (b[j])];
 if odd (i) then l:=l shr 4
 else l:=l and $0fff;
 end
 else
 begin
 j:=i*2;k:=j div 512;
 tg:=R_sector (drive,0,head,2+k,1);
 j:=j mod 512;
 l:=memw[seg (b[j]):ofs (b[j])];
 l:=l and $0fff;
 end;
 k1:=l shr 8+1;k2:= (l shr 4)and $0f+1;k3:=l and $0f+1;
 write (s16[k1],s16[k2],s16[k3],' ');
 if (s16[k1]+s16[k2]+s16[k3]<>'fff')then R_fat:=l else
 begin
 writeln;writeln ('End of file press any case:');
 ch:=readkey;
 if (ch='q')then halt (1);
 end
 end;
 Begin clrscr;
 write ('Ten odia:');readln (drv);
 for start:=7 to 150 do i:=r_fat (start);
 End.
```

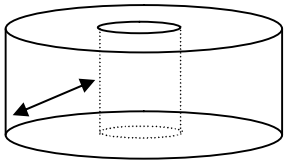
*D/ Partition*

Bao gồm: 4 phần tử, mỗi phần tử 16byte chia thành 4 trường, mỗi trường 4byte.

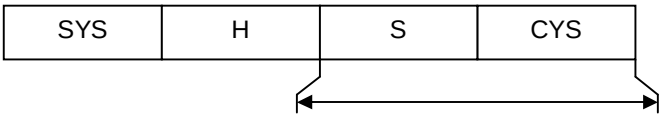
Mỗi phần tử khác không xác định 1 phần tử được sử dụng như 1 đĩa từ độc lập

Nếu biết địa chỉ vật lý đầu có thể tính được địa chỉ logic đầu, các tham số còn lại

| Đ/c V/lý đầu | Đ/c V/lý cuối | Đ/c Logic cuối | Tổng số Sector |
|---|---|---|---|



*Địa chỉ vật lý đầu:* 4byte




SYS: byte hệ thống

Bằng 00h nếu đĩa là đĩa làm việc

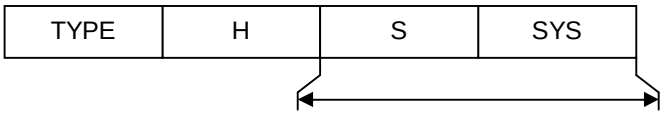
Bằng 80h nếu là đĩa hệ thống (phần chứa hệ thống được đặt tên là ổ đĩa C)

H: chứa số đầu đọc

S: chứa số sector

CYL: số cylinder

*Địa chỉ vật lý cuối:* 4byte




TYPE:

00h: cấm đọc ghi (không cho phép truy nhập)

01h: áp dụng cho đĩa có dung lượng nhỏ hơn 4MB (FAT 12)

04h: áp dụng cho đĩa có dung lượng nhỏ hơn 32MB (FAT 16)

06h: áp dụng cho đĩa có dung lượng lớn hơn 32MB (FAT 16)

0Ch: áp dụng cho đĩa có dung lượng lớn hơn FAT32

SYS:

Bằng 80h nếu là đĩa hệ thống TYPE = 51 (DM: disk manager)

Bằng 05h: loại mở rộng (extended) cấu trúc logic như một đĩa cứng vật lý vì vậy tồn tại master boot riêng, có partition riêng.

Ví dụ:

Với FAT16: địa chỉ bắt đầu của Partition là 1BEh

80 01 01 00 06 3F FF 4D 3F 00 00 00 41 00 34 00

00 00 00 00...

00 00

00 00

55 AA

Đọc MBR

```
uses crt,MSDOS;
 const s16:string[16]='0123456789abcdef';
 var reg:registers;
 B:array[0..511]of byte;
 i:integer; j,k:byte;ch:char;
 begin clrscr;
 with reg do
 begin
 dl:=$80;dh:=0;
 cl:=1;ch:=0;
 al:=1;ah:=2;
 bx:=ofs (b);es:=seg (b);
 end;
 intr ($13,reg);
 for i:=$1be to 511 do
 begin
 j:=b[i]shr 4+1;
 k:=b[i]and$0f+1;
 write (s16[j]:2,s16[k]);
 if (i+1)mod 16 =0 then
 begin
 write (' ':5);
 for j:=i-15 to i do
 if (b[j]<32)or (b[j]=255)then write ('.')
 else write (chr (b[j]));
 if (i=255)then ch:=readkey;
 writeln;
 end end;
 readln
 end.
```

## CÂU HỎI VÀ BÀI TẬP

2.1. Trình bày việc phân cấp trong tổ chức và quản lý thiết bị ngoại vi.

2.2. Trình bày vai trò của bộ đệm? Nêu các cách điều khiển bộ đệm vào ra dữ liệu.

2.3. Trình bày các phương pháp tổ chức dữ liệu trên hệ thống máy tính

2.4. Anh, chị hãy trình bày các phương pháp truy nhập dữ liệu đã được dùng phổ biến hiện nay.

2.5. Vẽ sơ đồ thuật toán việc đọc và hiển thị giá trị của 05 chỉ mục đầu tiên trong bảng FAT32

2.6. Xây dựng chương trình đọc thông tin trên 1 đĩa cứng và hiển thị ra màn hình cho biết đĩa đó có bao nhiêu mặt, trên mỗi mặt có bao nhiêu rãnh, trên mỗi rãnh có bao nhiêu sector, số sector trên một liên cung và tổng số liên cung.

2.7. Xây dựng chương trình liệt kê và lưu vào đĩa bảng phân vùng của đĩa cứng cụ thể

2.8. Xây dựng chương trình liệt kê và lưu các thông số hệ thống được lưu trữ trong Boot Record của 01 đĩa logic.

2.9. Xây dựng chương trình liệt kê các mục vào của Root trên 01 đĩa cứng cụ thể FAT16 hoặc FAT32)

2.10. Xây dựng chương trình liệt kê 01 sector bất kỳ trên đĩa.

2.11. Xây dựng chương trình liệt kê thông tin của 01 file text đã lưu trữ trên đĩa

2.12. Xây dựng chương trình liệt kê thông tin của bảng MFT trong đĩa cứng sử dụng kỹ thuật quản lý file NTFS

# Chương III: QUẢN LÝ BỘ NHỚ

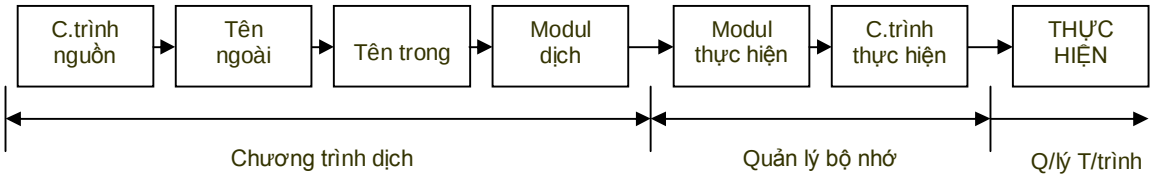
## 3.1. Đặt vấn đề

Bộ nhớ là 1 tài nguyên không thể thiếu được, đóng vai trò lưu trữ thông tin để xử lý vì vậy nó có liên quan tới tốc độ xử lý.

Một phần bộ nhớ trong dùng lưu trữ nhân (kernel) của hệ thống- tập các chương trình điều khiển thường xuyên có mặt ở bộ nhớ trong để thực hiện khi cần.

Chức năng khác của hệ điều hành là bảo vệ chương trình và dữ liệu khỏi bị hư hỏng, truy nhập một cách không hợp thức khi các chương trình khác hoạt động.

Các bước xử lý chương trình:




Chương trình nguồn: Các chương trình được viết dưới dạng ngôn ngữ thuật toán qua chương trình dịch dịch sang ngôn ngữ máy.

Các phép ánh xạ: hệ thống phải chuyển đổi các tên ngoài thành tên trong

Tên ngoài: do người dùng đặt

Tên trong: tên do hệ thống đặt trong quá trình dịch, dùng phân phối bộ nhớ và xác lập mối quan hệ đơn trị tên địa chỉ (do hàm địa chỉ thực hiện)

Hàm địa chỉ xác lập quan hệ giữa không gian tên và không gian bộ nhớ (bộ nhớ logic)

Modul dịch của chương trình là chương trình viết trên ngôn ngữ máy, nhưng nó mới được xét độc lập không nhúng vào quan hệ chung của toàn hệ thống vì vậy cần phải chuyển thành chương trình thực hiện.

Việc tạp hợp các chương trình modul dịch thành các chương trình dạng thực hiện do chương trình biên tập (LINK) đảm nhiệm.

Định vị chương trình: nạp chương trình vào bộ nhớ trong cụ thể, đặt vào vị trí xác định và sửa địa chỉ cho thích hợp với môi trường khai thác cụ thể.

Với chương trình .COM: dung lượng nhỏ hơn 64KB nên có thể đặt ở một nơi nào đó và thực hiện ngay không cần sửa đổi

Với chương trình .EXE: chương trình được chuẩn bị gần dạng thực hiện nhưng chưa lắp ráp vì vậy khi đưa vào bộ nhớ phải láp ráp theo chương trình điều khiển (biên tập lại)

Chế độ lập trình:

- ✓ $V_{LG}$: dung lượng bộ nhớ logic
- ✓ $V_{PH}$: dung lượng bộ nhớ vật lý

Chế độ bộ nhớ thực:

Yêu cầu $V_{LG} < V_{PH}$ như vậy bộ nhớ sử dụng nhỏ hơn bộ nhớ ta có

Chế độ bộ nhớ ảo:

Không có ràng buộc giữa $V_{LG}$ và $V_{PH}$ như vậy khi quản lý không gian bộ nhớ:

- ✓ Logic: quan tâm tới chương trình được bố trí như thế nào\
- ✓ Vật lý: quan tâm tới chương trình và dữ liệu

Tóm lại: quan tâm tới việc xác lập quan hệ giữa hai bộ nhớ

### 3.2. Quản lý bộ nhớ logic - cấu trúc một chương trình

Một chương trình có thể bao gồm nhiều modul, các modul có thể có cùng một dạng cấu trúc hoặc có những cấu trúc khác nhau

Chương trình có thể có các dạng cấu trúc:

- Tuyến tính
- Động
- Overlay
- Phân đoạn
- Phân trang

#### *3.2.1. Cấu trúc tuyến tính*

Sau khi biên tập các modul được tập hợp thành 1 chương trình hoàn thiện chứa đầy đủ thông tin để có thể thực hiện

Thực hiện: định vị 1 lần vào bộ nhớ

Ưu điểm:

- Đơn giản, chỉ việc tìm các móc nối
- Không có sự gò bó về thời gian
- Tính lưu động cao: có thể chuyển từ nơi này tới nơi khác

Nhược:

- Lãng phí bộ nhớ vì phải sử dụng vùng bộ nhớ lớn hơn mức cần thiết

#### *3.2.2. Cấu trúc động*

Từng modul được biên tập riêng biệt

Khi thực hiện chỉ việc nạp modul đầu tiên vào bộ nhớ

Khi cần modul khác người sử dụng phải sử dụng lệnh macro hệ thống để nạp định vị modul hoặc xoá modul ra khỏi bộ nhớ

Ví dụ: Lệnh Macro

Attach: nạp, gắn vào

Load: nạp modul vào nhưng chưa thực hiện

Delete: xoá modul khỏi bộ nhớ

Người dùng có thể tham gia trực tiếp voà quá trình định vị

Ưu điểm:

- Tiết kiệm bộ nhớ

Nhược:

- Yêu cầu người dùng phải biết kích thước hệ thống
- Thời gian thực hiện lớn, vừa thực hiện vừa định vị
- Kém linh động

#### *3.2.3. Cấu trúc Overlay*

Các modul chương trình được chia thành từng lớp

- Lớp 0: modul gốc- modul đầu tiên được gọi
- Lớp 1: modul được modul lớp 0 gọi (không cần được gọi đồng thời)
- Lớp 2: modul được modul lớp 1 gọi
- …

Bộ nhớ dành cho chương trình được chia thành các phần, mức bộ nhớ và mức chương trình

Để biết modul nào thuộc mức nào người dùng phải cung cấp thông tin cho biết:

- Số mức, modul tương ứng với mức (gọi là sơ đồ overlay hay file Overlay - OVL)

- Modul mức 0 được để ở 1 file chương trình riêng, khi cần nạp modul nào thì hệ thống tìm kiếm trong overlay và nạp vào bộ nhớ ở mức overlay tương ứng
- Duy trì hoạt động chương trình theo sơ đồ overlay gọi là supervisor overlay

Khi nạp vào mức đã dùng rồi thì modul cũ bị xoá

Ví dụ:

Ưu điểm:

- Tiết kiệm bộ nhớ
- người dùng không phải can thiệp vào chương trình nguồn
- Các modul không phải lưu trữ nhiều lần

Nhược:

- Người dùng phải cung cấp sơ đồ overlay
- Hiệu quả sử dụng bộ nhớ tăng dần tới 1 mức nào đó thì dừng lại
- Hạn chế 1 số cách gọi chương trình con

### *3.2.4. Cấu trúc phân đoạn*

Khi chương trình của người dùng được biên tập tạo thành các modul riêng biệt, tập hợp các chương trình là 1 bảng điều khiển cho biết chương trình có thể sử dụng những modul nào thông qua SCB (segment control block)

SCB chứa 1 số thông tin trợ giúp định vị chương trình, dựa vào SCB nạp modul vào trong bộ nhớ.

Khi thực hiện chương trình dựa vào SCB kiểm tra xem modul có trong bộ nhớ hay không, nếu chưa có trong bộ nhớ thì chương trình được nạp vào bất kỳ vùng nhớ nào .

Ưu điểm:

- Các modul không cần phải naptucj và không cần có vị trí cố định
- Người dùng không cần phải khai báo bất kỳ thông tin phụ nào
- Thực hiện nhanh hơn so với sơ đồ overlay
- Hiệu quả tăng dần theo kích thước bộ nhớ

Nhược:

- Phụ thuộc cấu trúc ban đầu của chương trình nguồn

### *3.2.5. Cấu trúc phân trang*

Chương trình của người dùng được chia thành từng trang có kích thước giống nhau được quản lý bởi bảng quản lý trang

Khi thực hiện sẽ nạp dần từng trang theo nhu cầu vì vậy hạn chế lãng phí bộ nhớ

Có sự hỗ trợ của phần cứng

Đẩy hệ số tích trữ bộ nhớ lên cao

Phân cấp bộ nhớ:

***Phân Trang:***

ROM (384B):

- Ghi đọc 1 lần
- Không cần nguồn để giữ
- Tốc độ truy nhập cao

Sơ cấp: 640KB đầu tiên

Expended:

- Phục vụ cho Vào/ra
- Processor cần làm việc trực tiếp với bộ nhớ này
- Extended: đòi hỏi chế độ mở rộng

***Ngoài:***

Disk: khối lượng lớn, thời gian lưu trữ lâu dài

***Chuyên dụng:***

- ✓ CMOS (64KB): lưu trữ thông tin cấu hình
- ✓ R: truy nhập nhanh, phải phối hợp với CPU
- ✓ Buffer: hoạt động như 1 máy tính chuyên dụng
- ✓ Cache: phục vụ Vào/ra

### 3.3. Quản lý bộ nhớ vật lý

Bộ nhớ có lích thước cụ thể vì vậy nhạy cảm với các kiểu sử dụng cụ thể

#### *3.3.1. Phân chương cố định*

Bộ nhớ được chia thành n phần không nhất thiết phải bằng nhau, mỗi phần sử dụng như 1 bộ nhớ độc lập gọi là ***Chương***

Bao nhiêu Chương thì có bấy nhiêu chương trình

Mỗi chương trình có 1 danh sách quản lý bộ nhớ tự do chưa sử dụng riêng

Chương trình được nạp vào chương nào sẽ tồn tại ở đó cho tới khi kết thúc

Ưu điểm:

- Đơn giản
- Dễ bảo vệ
- Tồn tại công cụ bên trong bộ nhớ có thể phân chia lại hệ thống
- Có thể phân loại các chương trình trước khi thực hiện vì vậy có thể tổ chức phục vụ gần tối ưu

Nhược:

- Bộ nhớ bị phân đoạn nên khi phân chia lại sẽ thay đổi đường biên vì vậy thông tin bị xoá

#### *3.3.2. Chế độ phân chương động*

Chỉ tồn tại 1 danh sách quản lý bộ nhớ tự do cho toàn bộ hệ thống nhớ.

Mỗi chương trình khi xuất hiện được phân phối 1 vùng nhớ riêng liên tục được sử dụng như 1 bộ nhớ độc lập

Ưu điểm:

- Hệ số song song cao, không cố định
- Số chương trình thực hiện có thể thay đổi
- Không bị phân đoạn nên có thể thực hiện 1 chương trình bất kỳ miễn là có đủ bộ nhớ

- Cơ chế đợi chờ (đủ bộ nhớ thì làm việc)
- Hệ thống điều khiển không bị sao chép đi nơi khác

Nhược:

- Hiệu quả sử dụng bộ nhớ không cao
- Nếu có sự cố kỹ thuật thì chương trình sẽ bị phá huỷ
- Sơ đồ phức tạp
- Xuất hiện hiện tượng phân đoạn ngoài

Khắc phục:

- Bố trí lại bộ nhớ tìm thời điểm thích hợp làn lượt dừng các chương trình đang được thực hiện
- Đưa 1 số chương trình từ vùng nhớ trang sang nhớ ngoài

### *3.3.3. Chế độ phân đoạn*

Chương trình có cấu trúc phân đoạn. (Được biên tập thành các modul riêng biệt → Tồn tạo bảng SCB)

Người dùng hoàn toàn không quan tâm tới SCB và chương trình của họ được bố trí như thế nào trong bộ nhớ

SCB bao gồm các phần tử, mỗi phần tử tương ứng với 1 modul độc lập.

Mỗi phần tử bao gồm 3 trường:

| D | A | L |
|---|---|---|

Trường D:

0: chưa nạp vào bộ nhớ

1: đã nạp

Trường A: địa chỉ nơi nạp modul vào bộ nhớ

Trường L: độ dài modul

Ban đầu D và L có giá trị, L chỉ kích thước modul và D=0 (chưa nạp), SCB được xây dựng ngay khi biên tập

Khi thực hiện SCB được nạp vào trong bộ nhớ, địa chỉ của nó được đưa vào thanh ghi quản lý đoạn $R_S$ (register segment)

Địa chỉ truy nhập dữ liệu được biểu diễn dưới dạng cặp (s,d)

s: số hiệu segment (modul) cần truy nhập

d: địa chỉ tương đối tính từ segment

Truy nhập: 2 lần hướng tới bộ nhớ

- Lần 1: Lấy nội dung của thanh ghi ($R_S$) ghép với s để truy nhập tới phần tử thứ s trong bảng SCB
- Lần 2: dựa vào đó (khi D=1) lấy a+d để truy nhập tới dữ liệu

Ưu điểm:

- Áp dụng trên máy bất kỳ
- Cho phép sử dụng chung các modul trong bộ nhớ

Nhược:

- Hiệu quả phụ thuộc cấu trúc ban đầu của chương trình nguồn
- Phân đoạn ngoài: bố trí lại bộ nhớ

Nếu xuất hiện nhu cầu bố trí lại:

- Đưa ra modul tồn tại duy nhất trong bộ nhớ
- Đưa ra modul có lần sử dụng cách đay lâu nhất
- Đưa ra modul có tần suất sử dụng thấp nhất

### *3.3.4. Chế độ phân trang (ánh xạ bộ nhớ logic thành vật lý)*

Bộ nhớ vật lý được chia thành từng phần bằng nhau gọi là ***Trang***, các trang được đánh số thứ tự 0,1,2..

Chương trình phải có cấu trúc trang

Trang trong chương trình phải có cùng kích thước trang vật lý

1 trang vật lý: 256byte-4KB

Khi làm việc chương trình được biên tập theo từng trang tạo ra các PCB

PCB: là tập hợp các phần tử mỗi phần tử ứng với 1 trang của chương trình

Bao gồm 2 trường:

| D | $A_P$ |
|---|---|

Trường D: dấu hiệu cho biết trang được nạp vào bộ nhớ hay chưa

0: chưa

1: đã nạp

Trường $A_P$: địa chỉ trang

Khi thực hiện :

PCB được nạp vào bộ nhớ

Địa chỉ đầu được đưa vào thanh ghi $R_P$

Địa chỉ dữ liệu được biểu diễn dưới dạng: (p,d)

p: số hiệu trang

d: offset tính từ đầu trang

Truy nhập dữ liệu: 2 hướng tới bộ nhớ

- Lần 1: lấy $R_P$+p để truy nhập tới trang p trong PCB
- Lần 2: đợi d=1 lấy Ap ghép với d truy nhập dữ liệu

Ưu:

- Không có hiện tượng phân đoạn ngoài
- Hạn chế việc thiếu bộ nhớ
- Khi thiếu bộ nhớ có thể giải phóng bằng cách đưa 1 trang ra ngoài
    - Trang tồn tại lâu nhất trong bộ nhớ
    - Trang có số lần sử dụng cách đay lâu nhất
    - Trang có tần suất sử dụng thấp nhất

Nhược:

- Bảng PCB có thể có kích thước lớn

### *3.3.5. Chế độ kết hợp phân trang và phân đoạn*

Bộ nhớ được tổ chức theo kiểu phân trang

Chương trình được tổ chức theo kiểu phân đoạn (Tồn tại SCB)

Mỗi modul được biên tập theo chế độ phân trang vì vậy mỗi modul có 1 PSB riêng

Mỗi phần tử của SCB sẽ quản lý các PCB tương ứng của modul

D: xác định PCB vào bộ nhớ hay chưa

A: địa chỉ đầu PCB

L: độ dài Modul

Khi thực hiện một chương trình SCB được nạp vào trong bộ nhớ. Địa chỉ đầu của nó được đưa vào thanh ghi Rs

Bộ nhớ được chia thành 3 phần:

Phần1: chứa SCB

Phần2: chứa các PCB

Phần3: chứa các trang chương trình và dữ liệu

Truy nhập:

Địa chỉ biểu diễn: (s,p,d)

Trong đó:

s: modul cần truy nhập

p: trang cần truy nhập

d: địa chỉ offset tính từ đầu trang

Truy xuất ô nhớ: Mỗi lần truy nhập cần 3 lần hướng tới bộ nhớ

- Lần1: lấy nội dung $R_S$+s: truy nhập phần tử s của SCB
- Lần2: d=1: lấy A+p truy nhập phần tử thứ p của PCB thứ s
- Lần3: Dp=1: lấy $A_P$ ghép với d truy nhập dữ liệu

Ưu:

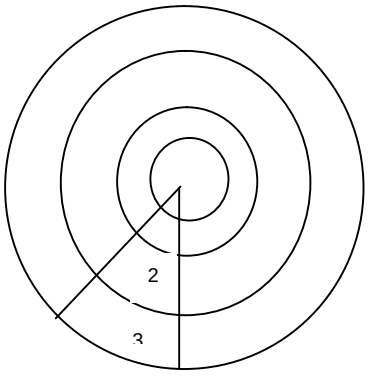
- Kết hợp ưu của phân đoạn và phân trang
- Chống hiện tượng phân đoạn ngoài
- Đảm bảo sử dụng hiệu quả bộ nhớ

### 3.4. Quản lý bộ nhớ IBM PC của MSDOS

Hệ thống MSDOS được chia thành 4 mức 0,1,2,3

- Mức 0: Nhân hệ điều hành (Kennel)
- Mức 1: Quản lý thiết bị, File
- Mức 2: Chương trình phục vụ hệ thống
- Mức 3: Chương trình ứng dụng

Mức ưu tiên 0-3




Một chương trình chỉ được quyền truy nhập tới chương trình và dữ liệu cùng mức ưu tiên hoặc kém mức ưu tiên hơn

Bộ nhớ phân phối cho 1 chương trình chia làm 2 loại:

Bộ nhớ chung:

- Vùng nhớ mà mọi chương trình đều được biết và được quyền truy nhập
- Có bảng tham số điều khiển GDT (Global Descreption Table)

Bộ nhớ riêng:

- Phân phối cho chương trình nào thì chỉ có chương trình đó được biết và được quyền truy nhập

- Có bảng tham số LDT (Local Descreption Table)

Với máy PC có 2 chế độ làm việc là:

- Chế độ thực (Real mode)
- Chế độ bảo vệ (protect mode)

Nguyên tắc:

Bộ nhớ được chia thành từng khối

Real mode:

- Dung lượng khối <64KB
- Các khối được đánh số 0,1,2... gọi là Index
- Khối 0-9 dành cho người dùng sấp xỉ 640KB: bộ nhớ cơ sở
- Khối A,B cho các phương pháp tổ chức truy nhập
- Khối C-F: ROM
- Khối F,E: ROMBASIC

Protect mode:

- 0-3FFF: 16K khối
- Mỗi khối tương ứng với 1 vùng bộ nhớ thực RAM
- Từ bộ nhớ khối sang bộ nhớ logic
- Khi 1 khối được nạp trong bộ nhớ thì phần tử tương ứng sẽ thuộc 1 vùng nhớ vật lý

**CÂU HỎI VÀ BÀI TẬP**

3.1. Anh chị hãy trình bày tổng quan về kỹ thuật Swapping bộ nhớ

3.2. Anh chị hãy trình bày tổng quan về các chiến lược điều khiển trang bộ nhớ.

3.2 Anh chị hãy trình bày tổng quan về kỹ thuật quản lý bộ nhớ ảo trong Windows NT

# Chương IV: QUẢN LÝ TIẾN TRÌNH

## 4.1. Quản lý tiến trình

### *4.1.1. Khái niệm*

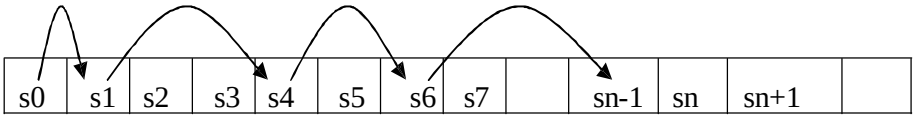
Phương pháp tiếp cận:

Coi tiến trình là nhóm các byte có nội dung thay đổi theo 1 luật nào đó, luật hướng dẫn Processor thực hiện.

- Saltzer: Tiến trình là chương trình do 1 processor logic thực hiện
- Dijkstra: Tiến trình là những gì liên quan đến hệ thống tính toán xuất hiện khi thực hiện 1 chương trình
- Đinh nghĩa của Horning & Randell: Tiến trình như 1 quá trình chuyển từ trạng thái này sang trạng thái khác dưới tác động của hàm hành động và xuất phát từ trạng thái ban đầu nào đó

  Hàm hành động : ánh xạ trạng thái sang hành động, hành động dựa vào trạng thái ban đầu

  Từ chuỗi các trạng thái đến công việc




- Quan điểm của người dùng: Tiến trình là một quá trình thực hiện chương trình

### *4.1.2. Tổ chức tiến trình*

Tổ chức

Tiến trình tương đương cấu trúc thông tin cho phép xác định đơn trị tiến trình (cấu trúc thông tin này gọi là khối mô tả thông tin bao gồm):

- Biến trạng thái thông tin : Trạng thái hiện tại của tiến trình
- Vùng bộ nhớ lưu trữ giá trị của các thanh ghi tiến trình sử dụng
- Thông tin về tài nguyên tiến trình đang sử dụng hoặc có quyền sử dụng.

Hình thành tiến trình

- Khung chương trình gán cho các giá trị và tài nguyên cụ thể
- Thông tin được xây dựng khi có yêu cầu và huỷ bỏ khi công việc đã hoàn thành

Phân loại tiến trình

- TT tuần tự : một tiến trình chỉ bắt đầu sau khi tiến trình kia kết thúc
- TT song song: Thời điểm bắt đầu của tiến trình này nằm giữa thời điểm bắt đầu và kết thúc của một tiến trình khác.

Quan hệ:

*Độc lập*: 2 tiến trình không có quan hệ trực tiếp gì với nhau

Yêu cầu : bảo vệ thông tin sao cho một tiến trình không làm hỏng dữ liệu và chương trình của tiến trình khác, như vậy phải phân phối tài nguyên hợp lý

*Tiến trình trao đổi thông tin với nhau*: một tiến trình có thể gửi thông báo cho tiến trình khác, tổ chức các vùng nhớ làm hòm thư.

*Phân lớp*: Trong quá trình hoạt động của một tiến trình có thể khởi tạo một tiến trình khác hoạt động song song: (chương trình chính, chương trình con)

Cơ chế cấp phát tài nguyên:

- Phân tán: Phân phối tài nguyên cho cả chương trình chính và chương trình con
- Tập chung: Tài nguyên chỉ được phân phối cho tiến trình chính

*Tiến trình đồng mức*: Những tiến trình có một số tài nguyên sử dụng chung theo nguyên tắc lần lượt.

### *4.3.3. Điều độ tiến trình - Tài nguyên Găng*

Tài nguyên Găng: Tài nguyên phân phối cho một người phục vụ, như vậy tại một thời điểm nếu đồng thời có nhiều tiến trình muốn sử dụng tài nguyên Găng: điều độ tiến trình để không có khi nào có một tiến trình chiếm dụng tài nguyên

Đoạn chương trình có sử dụng tài nguyên Găng gọi là đoạn Găng

Ví dụ:

*TTA ghi nội dung biến Dem vào TgA (biến cục bộ)*

*TTB ghi nội dung biến Dem vào TgB*

*TTA tăng TgA*

*TTB tăng TgB*

Nếu không để ý kỹ, có thể hiểu lầm là biến Dem tăng 2 đơn vị. Song thực chất cả 2 tiến trình A và B đều tăng nội dung Dem, song nội dung này chỉ tăng 1 đơn vị. Cần phải có cách giải quyết cụ thể.

- Dem : Tài nguyên Găng
- Đoạn chương trình xử lý biến Dem : Chương trình găng : Đoạn găng.

Khắc phục đụng độ :

- Tại một thời điểm có không quá một tiến trình nằm trong đoạn Găng
- Không một tiến trình nào được phép ở lâu vô hạn trong đoạn Găng
- Không một tiến trình nào phải chờ vô hạn ngoài đoạn Găng

Công cụ điều độ tiến trình qua đoạn găng :

- Cấp thấp: nằm ngoài tiến trình được điều độ
- Cấp cao: nằm trong tiến trình

Công cụ điều độ cấp thấp *:*

- Phương pháp khoá trong
- Phương pháp kiểm tra và xác lập
- Kĩ thuật đèn báo

*a, Phương pháp khoá trong (Kiểm tra luân phiên)*

Nguyên tắc: hai hay nhiều tiến trình cùng định ghi vào một địa chỉ nào đó của bộ nhớ trong thì sơ đồ kĩ thuật chỉ cho phép một tiến trình làm việc còn tiến trình khác phải chờ

Mỗi tiến trình: sử dụng một byte trong vùng bộ nhớ chung làm khoá, khi vào được đoạn Găng, gán giá trị là 1, thông báo cho các tiến trình khác biết đã có tiến trình sử dụng tài nguyên găng

Giải thuật Delker

```
Begin
     k1 := 0; k2:= 0; tg:=1;
     kt1:=1; kt2:=1;
          begin
               repeat
                    k1:=1;
                    While k2=1 do Ct2
                    if Tg=2 then  begin
                         k1:=0;
                         While tg=2 do Ct2
                         k1:=1;
                         end;
                    k1:=0; tg:=2;
               until kt1=0;
               repeat
```

```
                k2:=1;
                While k1=1 do Ct2
                if Tg=2 then  begin
                     k2:=0;
                     While tg=1 do Ct2
                     k2:=1;
                     end;
                k2:=0; tg:=1;
           until kt2=0;
```

Ưu điểm

- Dễ tổ chức thực hiện
- Có tính chất vạn năng áp dụng cho mọi công cụ và mọi hệ thống.

Nhược:

- Độ phức tạp tỷ lệ với số lượng tiến trình và số tài nguyên găng
- Một tiến trình có thể bị ngăn chặn bởi tiến trình thứ 3
- Khi tốc độ hai tiến trình khá chênh lệch, một trong hai tiến trình phải chờ

*b. Phương pháp kiểm tra và xác lập (Phương pháp Perterson)*

Tương đương với phương pháp khoá trong sử dụng các giá trị kiểm tra là các biến trạng thái: tham số (cục bộ, toàn cục).

Giải thuật

PAR là một lệnh gồm hai tham số:

- ✓ L: cục bộ (Local)
- ✓ G: toàn cục (Global)

Chức năng PAR

Gán L = G và gán G = 1;

- ✓ Hai lệnh trên phải được thực hiện liên tục không bị chia rẽ.
- ✓ Mỗi tiến trình sẽ sử dụng hai biến là biến local của mình và biến global của toàn Chương trình.

Giải thuật

```
Var L1, L2, G: byte;
Begin
G:=0;
begin
TT:=1;
repeat
   L1:=1;
   while L1=1 do PAR(L1);
   {đoạn giữa tiến trình 1}
   G:=0;
   {phần còn lại của tiến trình 1}
until false
TT:=2;
repeat
   L2:=1;
   while L2=1 do PAR(L2);
   {đoạn giữa tiến trình 2}
   G:=0;
   {phần còn lại của tiến trình 2}
until false
end;
End;
```

Ưu điểm:

- Khắc phục được độ phức tạp của thuật toán, độ phức tạp thuật toán không phụ thuộc vào số lượng tiến trình.

Nhược điểm:

- Vẫn còn hiện tượng chờ đợi tích cực.

*c. KT đèn báo (Semaphore - Dijkstra)*

Hệ thống sử dụng biến đèn báo nguyên đặc biệt (Semaphore) s. Ban đầu s nhận một giá trị bằng khả năng phục vụ của tài nguyên găng. Hệ thống có hai phép để thao tác trên s là P(s) và V(s).

P (s): Proberen (tiếng Hà Lan) có nghĩa là giảm

Giảm S đi 1 đơn vị

Nếu s ≥ 0 tiếp tục thực hiện tiến trình

Ngược lại đưa tiến trình vào dòng xếp hàng

V (s): Verhogen có nghĩa là kiểm tra

Tăng S lên 1

Nếu s ≤ 0 kích hoạt một tiến trình ra hoạt động

Giải thuật:

```
Var s: byte;
Begin
s:=1;
begin
tt:=1;
repeat
P(s)
{đoạn giữa tiến trình 1}
V(s);
{phần còn lại của tiến trình 1}
until false
tt:=2;
repeat
   P(s)
{đoạn giữa tiến trình 2}
   V(s);
   {phần còn lại của tiến trình 2}
until false
end;
End;
```

- Đặc điểm quan trọng là 2 phép P và V là liên tục, trong quá trình thực hiện P hoặc V thì processor không bị ngắt để chuyển sang công việc khác.

- Tuy nhiên các phép xử lý này có thể không tồn tại trên các máy vì P và V phải làm việc với dòng xếp hàng và thông tin lưu trữ khá lớn. Để khắc phục điều này người ta xây dựng các thủ tục procedure để thực hiện các phép xử lý này.

+ Đầu của thân thủ tục bao giờ cũng ra lệnh cấm ngắt tức là chặn mọi tín hiệu vào processor CLI, trừ những tín hiệu bắt buộc (ngắt không che

được).

+ Cuối thân thủ tục có lệnh giải phóng ngắt (STI).

*d. Công cụ điều độ cấp cao – chương trình thư kí (Monitor)*

Đặc điểm:

- Nằm ngoài tiến trình của người sử dụng
- Người sử dụng không biết tài nguyên gì và khi nào thuộc đoạn găng

Chương trình thư ký (Monitor): cấu trúc đặc biệt bao gồm các thủ tục, các biến và cấu trúc dữ liệu hoạt động trong chế độ phân chia thoì gian , hỗ trợ việc thực hiện tiến trình, với các thuộc tính:

- Các biến và cấu trúc dữ liệu trong Monitor chỉ có thể được thao tác bởi các thủ tục định nghĩa bên trong Monitor
- Tại một thời điểm, một tiến trình duy nhất được làm việc với chương trình thư ký
- Mỗi lần sử dụng tài nguyên mới, hệ thống gắn chương trình thư ký với tiến trình

Trong một Monitor có thể định nghĩa các biến điều kiện C và hai thao tác là Wait () và Signal ():

- Wait (C): chuyển trạng thái tiến trình sang trạng thái khoá và đặt tiến trình vào hàng đợi trên biến điều kiện C
- Signal (C): nếu có một tiến trình đang bị khoá trong hàng đợi của C thì tái kích hoạt tiến trình đó và tiến trình sẽ ời khỏi Monitor

Thuật toán

```
Wait (C)
begin
    status (p)=khoá
    enter (p, f (C)) { đưa p vào hàng đợi}
end;
Signal (C)
begin
    if f (C)<>nil then
    exit (q, f (C)) { đưa q ra khỏi hàng đợi}
end;
```

### *4.1.4. Tình trạng tắc nghẽn*

Tắc nghẽn: Khi có nhiều tài nguyên găng trong một tiến trình, các tiến trình sẽ rơi vào tình trạng chờ đợi lẫn nhau

Tình trạng tắc nghẽn: hai hay nhiều tiến trình cùng chờ đợi một sự kiện và nếu không có tác động đặc biệt từ ngoài thì sự chờ đợi ấy là vô hạn

- Phòng chống:
- Phòng ngừa : tránh không để tiến trình rơi vào tình trạng tắc nghẽn
- Dự báo và tránh : Kiểm tra xem tiến trình có rơi vào tình trạng tắc nghẽn hay không, thông báo kịp thời trước khi tắc nghẽn sảy ra
- Nhận biết và khắc phục : Phát hiện các tiến trình bị tắc nghẽn và giải quyết

*a. Phòng ngừa*

Xem xét các điều kiện tắc nghẽn:

- Thiếu tài nguyên Găng
- Chờ vô hạn khi chưa được vào đoạn Găng
- Không có hệ thống phân phối lại tài nguyên
- Tồn tại chờ đợi vòng

Điều kiện 1: Dùng kĩ thuật SPOOL: Khi kết thúc tiến trình thì kết quả được chuyển ngược lại tài nguyên vật lý mà sever yêu cầu, việc chuyển ngược này theo nguyên tắc lần lượt và do chương trình hệ thống đảm nhận như vậy không xảy ra xung đột

Điều kiện 2: Phân phối trước tài nguyên, tiến trình chỉ được bắt đầu khi nhận đủ tài nguyên trong một số lần phân phối

Điều kiện 3: Tạo các điểm gác: Hệ thống sẽ lưu lại toàn bộ thông tin trạng thái tiến trình, nếu cần thiết có thể huỷ tiến trình, giải phóng tài nguyên, sau đó nếu cho phép sẽ tiếp tục công việc bằng cách khôi phục trạng thái cuối.

Điều kiện 4: Chờ đợi vòng: Phân lớp tài nguyên, tiến trình chỉ nhận được tài nguyên mức cao hơn sau khi đã trả lại tài nguyên mức thấp.

*b. Dự báo và phòng tránh*

Không phòng ngừa nhưng mỗi lần phân phối tài nguyên thì kiểm tra xem việc phân phối đó có khả năng đẩy hệ thống vào tình trạng tắc nghẽn không? Nếu xuất hiện nguy cơ trên thì tìm cách giải quyết cụ thể trước khi tắc nghẽn có thể xảy ra

Thuật toán:

- Có n tiến trình
- Hệ thống có k thiết bị
- Tiến trình i yêu cầu tối đa một lúc max (i) đơn vị thiết bị để có thể thực hiện, nhưng hiện chỉ nhận được f (i) đơn vị thiết bị
- Tiến trình i kết thúc kt (i)=true

Thuật toán :

```
t:=k;
for i:=1 to n do
begin
        t:=t-f (i);
        cl[i]:=max[i];
        kt[i]:=false;
   end;
Flag:=True;
While Flag do
   begin
        flag:=false
        For i:=1 to n do
        if not kt[i] and (cl[i] <=t) then
             begin
             kt[i]:=true;
             t:=t+cl[i]
             Flag:=true;
             end;
end;
if t=k then “An toàn”
else “không an toàn”
```

*c. Nhận biết và khắc phục*

Quan sát trạng thái các tiến trình đang chờ, xem những tiến trình bị rơi vào tắc nghẽn, tuỳ tình hình cụ thể áp dụng các biện pháp cần thiết

Khi phát hiện tắc nghẽn:

- Đình chỉ hoạt động của tiến trình liên quan đưa tiến trình về trạng thái ngắt
- Thu hồi tài nguyên

Đưa tiến trình về trạng thái ngắt:

- Đưa tất cả các tiến trình trong tình trạng tắc nghẽn về ngắt.
- Đưa từng tiến trình khi không còn chu trình gây tắc nghẽn theo các tiêu chí:
  - Độ ưu tiên
  - Thời gian xử lý
  - Số lượng tài nguyên tiến trình đang chiếm dụng
  - Số lượng tài nguyên tiến trình yêu cầu

Thu hồi tài nguyên: thu hồi tài nguyên của một số tiến trình và cấp phát các tài nguyên này cho tới khi loại bỏ được chu trình tắc nghẽn

- Lựa chọn tiến trình thu hồi, những tài nguyên nào bị thu hồi
- Phục hồi trạng thái tiến trình ở trạng thái gần nhất trước đó mà không xảy ra tắc nghẽn
- Tránh cho một tiến trình nào đó luôn bị thu hồi tài nguyên

Ví dụ:

Các tiến trình: P1, P2, P3, P4

Các tài nguyên: R1, R2, R3

Tổng các tài nguyên của hệ thống k = 9R1 + 3R2 + 6R3

Trạng thái hiện thời các tiến trình:

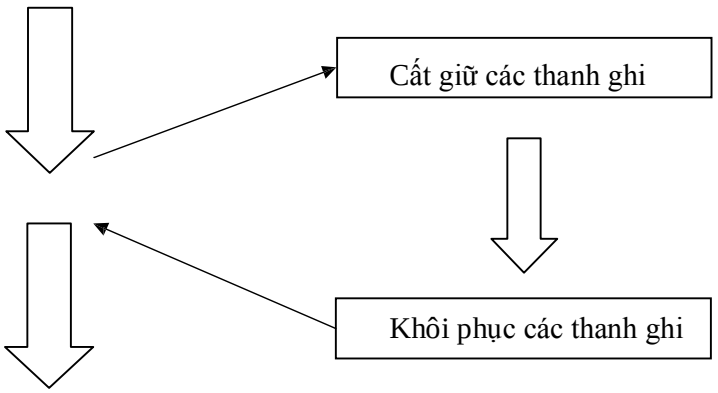
| Tiến trình | Max (i) | | | f (i) | | | t | | |
|---|---|---|---|---|---|---|---|---|---|
| | R1 | R2 | R3 | R1 | R2 | R3 | R1 | R2 | R3 |
| P1 | 3 | 2 | 2 | 1 | 0 | 0 | 4 | 1 | 2 |
| P2 | 6 | 1 | 3 | 2 | 1 | 1 | | | |
| P3 | 3 | 1 | 4 | 2 | 1 | 1 | | | |
| P4 | 4 | 2 | 2 | 0 | 0 | 2 | | | |

Giả sử P2 có yêu cầu 4R1 và 1R3, khi đó việc thoả mãn P2 có đẩy hệ thống tới tình trạng tắc nghẽn hay không?

### *4.1.5. Ngắt (Interupt)*

Phương tiện để các thiết bị trong hệ thống báo cho Processor biết việc thay đổi trạng thái của mình - công cụ chuyển điều khiển tới một tiến trình khác

- Ngắt là hiện tượng tạm ngừng thực hiện một tiến trình để chuyển sang thực hiện một tiến trình khác khi có một sự kiện xảy ra trong hệ thống tính toán.




- Có thể hiểu tạm nghĩa "thực hiện một tiến trình" là thực hiện một Chương trình, tiến trình bị ngắt có thể coi là Chương trình chính, còn tiến trình xử lý ngắt có thể coi là Chương trình con.

- Chương trình con xử lý ngắt là một Chương trình ngôn ngữ máy hoàn toàn bình thường. Chương trình này địa chỉ kết thúc bằng lệnh IRET (Interupt RETurn), nó ra lệnh cho bộ xử lý quay về thực hiện tiếp Chương trình chính đúng từ chỗ mà nó bị ngắt.

- Đối với các hệ thống tính toán việc gọi ngắt dùng cho việc các bộ phận khác nhau của hệ thống tính toán báo cho processor biết về kết quả thực hiện công việc của mình.

***Phân loại ngắt:***

- Ngắt trong: ngắt do các tín hiệu của procesor báo cho processor
- Ngắt ngoài: ngắt do các tính hiệu bên ngoài báo cho processor
- Ngắt cứng: ngắt được gọi bởi các Chương trình được cứng hoá trong các mạch điện tử.
    - Ngắt che được: (Maskable Interupt):
      Là ngắt có thể dùng mặt nạ để ngăn cho không ngắt hoạt động. Ta có thể đặt các bít trong mặt lạ bằng lệnh CLI (CLear Interupt flag).
      Ví dụ: Ngắt chuột là ngắt cứng có thể bị che
    - Ngắt không che được (Non Maskable Interupt):

Là ngắt không thể dùng mặt nạ che được (có độ ưu tiên cao nhất)
Ví dụ: Ngắt 2 báo hiệu có lỗi trong bộ nhớ.

- Ngắt mềm: ngắt được gọi bằng một lệnh ở trong Chương trình. Lệnh gọi ngắt từ Chương trình ngôn ngữ máy là lệnh INT (INTerupt), các lệnh gọi ngắt từ Chương trình ngôn ngữ bậc cao sẽ được dịch thành lệnh INT.
- Các ngắt khác

***Xử lý ngắt***

- ✓ Lưu đặc trưng sự kiện gây ngắt vào nơi quy định
- ✓ Lưu trạng thái của tiến trình bị ngắt vào nơi quy định
- ✓ Chuyển điều khiển tới Chương trình xử lý ngắt
- ✓ Thực hiện Chương trình xử lý ngắt, tức là xử lý sự kiện
- ✓ Khôi phục tiến trình bị ngắt

***Véc tơ ngắt:***

- Khi ngắt được tạo ra, nơi phát sinh nó không cần biết địa chỉ của Chương trình xử lý ngắt tương ứng mà chỉ cần biết số hiệu ngắt. Số hiệu này chỉ đến một phần tử trong một bảng gọi là bảng các vector ngắt nằm ở vùng có địa chỉ thấp nhất trong bộ nhớ và chứa địa chỉ của Chương trình con xử lý ngắt. Địa chỉ bắt đầu của mỗi Chương trình con được xác định bởi địa chỉ đoạn và địa chỉ offset được đặt trước đoạn.
- Hai địa chỉ này đều là 16 bit (2 byte), như vậy mỗi địa chỉ ngắt chiếm 4 byte trong bộ nhớ. Máy tính PC có 256 ngắt khác nhau được đánh số từ 0 đến 255 do vậy độ dài của cả bảng do vậy sẽ là 256*4 = 1024. Bảng vector ngắt chiếm các ô nhớ từ địa chỉ 0 đến 3FFh. Số thứ tự của ngắt bằng số thứ tự của vector ngắt. Địa chỉ của Chương trình xử lý số i được chứa trong bảng véc tơ ngắt từ địa chỉ offset 4*(i-1) đến 4*(i-1) + 3.

Một số ngắt thường dùng

| STT | Số hiệu ngắt | Chức năng | STT | Số hiệu ngắt | Chức năng |
|---|---|---|---|---|---|
| 1 | 00 | Ngắt chia cho 0 | 7 | 20H | Kết thúc Chương trình |
| 2 | 04 | Ngắt tràn số | 8 | 21H | Gọi các hàm của DOS |
| 3 | 08 | Ngắt thời gian | 9 | 25H/26H | Đọc/ghi đĩa |
| 4 | 09 | Ngắt bàn phím | 10 | 27H | Kết thúc nhưng thường trú |
| 5 | 10H | Ngắt phục vụ màn hình | 11 | 33H | Ngắt phục vụ chuột |
| 6 | 19H | Ngắt khởi động hệ thống | 12 | 67H | Quản lý bộ nhớ mở rộng |

(Tham khảo thêm Vi xử lý)

## 4.2. Quản lý Processor

***Đặt vấn đề***

Chương trình không thể thực hiện được nếu nó không được nạp vào bộ nhớ, song ngay cả khi đã được nạp vào bộ nhớ nếu nó không có quyền sử dụng Processor thì vẫn không thể thực hiện được.

- Processor: Tài nguyên phục vụ cho việc thực hiện chương trình. Đơn vị công việc giao cho processor phục vụ là tiến trình, nhiều tiến trình có thể sản sinh từ chương trình.
- Tiến trình: đối tượng mà ta có thể phân phối Processor cho nó.

### *4.2.1. Processor vật lý và Processor logic*

*Processor vật lý*: tất cả các hệ điều hành thực hiện song song đều do một Processor của hệ thống – Processor vật lý điều khiển.

*Processor logic:* người sử dụng đánh giá hoạt động của Processor trên cơ sở quan sát và đánh giá chương trình của mình được thực hiện như thế nào. Processor mà người sử dụng quan sát và đánh giá được gọi là Processor logic - liên quan tới việc thực hiện tiến trình.

Với chế độ xử lý kế tiếp đơn chương trình (Tiến trình tuần tự): $P_{VL} \approx P_{LG}$

Với các tiến trình hoạt động song song quan tâm các chiến lược điều độ Processor ( điều độ tiến trình mức Processor).

*Vấn đề cần quan tâm:*

Nên tạo ra bao nhiêu Processor logic là thích hợp

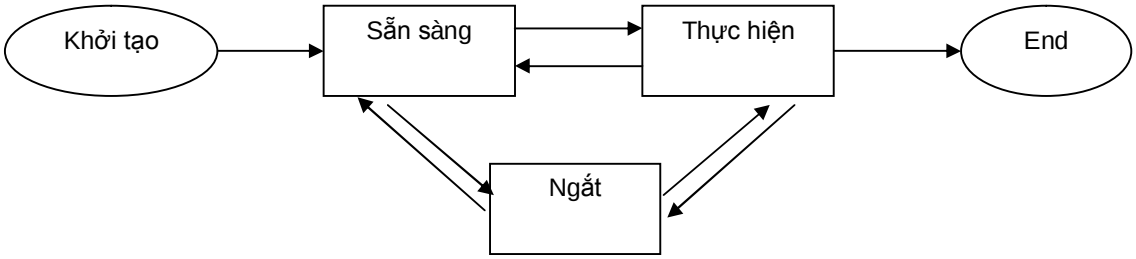
Độ dài khoảng thời gian gắn liên tục Processor vật lý cho Processor logic là bao nhiêu thì hợp lý

Sau khi một Processor logic hết quyền sử dụng Processor vật lý thì cần chọn tiến trình nào để phân phối Processor vật lý.

### *4.2.2. Phân phối Processor*

Trong chế độ đa nhiệm, mỗi tiến trình có thể thuộc một trong ba trạng thái:

- ✓ Sẵn sàng
- ✓ Thực hiện
- ✓ Ngắt




*Trạng thái Thực hiện:* Nếu hệ thống chỉ có một Processor thì mỗi thời điểm chỉ có một tiến trình dành được Processor để thực hiện lệnh của mình. Tiến trình này nằm trong trạng thái thực hiện.

*Trạng thái Ngắt:* Nếu tiến trình không thể thực hiện tiếp được vì bị thiếu một vài điều kiện nào đó tiến trình sẽ nằm trong trạng thái ngắt. Tiến trình gọi tới một môđun nhưng môđun chưa được nạp và định vị trong bộ nhớ. Khi đó tiến trình có thể được lưu trữ tại bộ nhớ ngoài.

*Trạng thái Sẵn sàng:* Tiến trình được phân phối đầy đủ tài nguyên (trừ Processor): tiến trình nằm trong trạng thái sẵn sàng, khi processor rỗi tiến trình sẽ được thực hiện.

Tiến trình có thể rời bỏ trạng thái Thực hiện bởi một trong ba lý do:

- ✓ Tiến trình đã hoàn thành mọi việc cần thiết, khi đó nó trả lại processor và chuyển sang chờ xử lý kết quả.
- ✓ Tự ngắt: Tiến trình chuyển sang trạng thái ngắt khi nó chờ mội sự kiện nào đó.
- ✓ Tiến trình đã sử dụng hết thời gian processor vật lý dành cho nó và được chương trình điều độ chuyển nó từ trạng thái thực hiện sang trạng thái sẵn sàng (phân phối lại tài nguyên hệ thống).

### *4.3.3. Điều độ tiến trình*

Một trong những chức năng của chương trình điều độ là chọn tiến trình để thực hiện (chọn tiến trình đã sẵn sàng và phân phối processor vật lý cho nó).

Mỗi tiến trình sẵn sàng được gắn một thứ tự ưu tiên, thứ tự này được xác định dựa vào các yếu tố:

- ✓ Thời điểm hình thành

- ✓ Tổng thời gian tiến trình được thực hiện
- ✓ Thời gian người sử dụng dự báo kết thúc tiến trình.
- ✓ Tiêu chuẩn đánh giá chất lượng điều độ: Thời gian chờ đợi xử lý – thời gian một tiến trình ở trạng thái sẵn sàng chờ được phân phối Processor vật lý.
- ✓ Các chiến lược thường gặp và cơ chế tổ chức của các chiến lược đó

***A. Chế độ một dòng xếp hàng***

Nguyên tắc: đảm bảo cho mọi tiến trình được phục vụ như nhau, không có một tiến trình nào phải chờ đợi lâu hơn tiến trình khác.

- ✓ Để đánh giá chất lượng điều độ ta có thể dựa vào thời gian chờ đợi trung bình của các tiến trình.
- ✓ Thời gian chờ đợi của các tiến trình được tính từ khi tiến trình ở trạng thái sẵn sàng tới khi tiến trình chuyển sang trạng thái thực hiện.
- ✓ Với mỗi tiến trình ta đo khoảng thời gian này nhiều lần, khi đó có thể tính được thời gian trung bình.
- ✓ Kết hợp việc đo thực nghiệm và phân tích giải thuật điều độ để đánh giá chất lượng điều độ để có được thời gian chờ đợi trung bình chính xác cho các tiến trình.
- ✓ Quan sát và thống kê thời gian của từng tiến trình rút ra thời gian chờ đợi trung bình của hệ thống.

*a. Chiến lược phục vụ bình đẳng FCFS (First Come First Served)*

Đảm bảo mọi tiến trình đều có một thời gian chờ đợi trung bình như nhau, các tiến trình được phục vụ đến khi nó kết thúc hoặc khi phải chuyển sang trạng thái ngắt.

ưu điểm:

- ✓ Processor không bị phân phối lại
- ✓ Chi phí thấp: không phải thay đổi thứ tự ưu tiên điều độ

Nhược điểm:

- ✓ Tiến trình ngắn cũng phải chờ như tiến trình dài
- ✓ Thời gian chờ đợi trung bình tăng vô hạn khi hệ thống tiệm cận tới khả năng phục vụ của mình
- ✓ Khi gặp tiến trình bị ngắt, các tiến trình khác sẽ bị xếp hàng lâu.

*b.Chiến lược ưu tiên những tiến trình có thời gian thực hiện ngắn nhất SJN (Shortest Job Next)*

Xác định thứ tự ưu tiên điều độ trong quá trình thực hiện tiến trình chứ không phải ở lúc khởi tạo.

Đặc điểm:

- ✓ Không phân phối lại Processor
- ✓ Thời gian chờ đợi của các tiến trình ngắn nhỏ hơn so với phương pháp FCFS
- ✓ Thời gian chờ đợi của các tiến trình dài lớn hơn so với phương pháp FCFS
- ✓ Không dự đoán được khi nào tiến trình dài được thực hiện.

*c. Chiến lược ưu tiên các tiến trình có thời gian còn lại ít nhất SRT (Shortest Remaining Time)*

Nhược điểm của FCFS là các tiến trình ngắn phải chờ đợi như tiến trình dài, với SJN thì không dự đoán được khi nào tiến trình dài được thực hiện. Khắc phục các nhược điểm này: so sánh thời gian thực hiện của tiến trình dài đang được thực hiện với thời gian thực hiện tiến trình ngắn được dự báo trước để xem xét độ ưu tiên

Nếu thời gian thực hiện của tiến trình dài đang thực hiện còn lại là nhỏ hơn thì tiếp tục thực hiện tiến trình dài, ngược lại đưa tiến trình về trạng thái ngắt và thực hiện tiến trình ngắn.

*d. Chiến lược xếp hàng lần lượt RR (Round Robin) – phân phối lại Processor*

Nguyên tắc: mỗi một tiến trình trong dòng xếp hàng lần lượt được phân phối một lượng tử thời gian để thực hiện. Sau khoảng thời gian đó, nếu tiến trình chưa kết thúc hoặc không rơi vào trạng thái ngắt thì nó được chuyển về cuối dòng xếp hàng: tiến trình xếp hàng vòng tròn.

Khi có một tiến trình mới, nó sẽ được đưa vào dòng xếp hàng vòng tròn và được đặt ở vị trí được phục vụ ngay lập tức.

Với các tiến trình dài: phân thành m lớp, lớp thứ i tiến trình được phục vụ với khoảng thời gian $T_i$, sau khi đã được thực hiện, tiến trình chưa kết thúc hoặc không bị ngắt nó được chuyển sang lớp thứ i+1 với thời gian phục vụ $T_{i+1} > T_i$.

***B. Chiến lược nhiều dòng xếp hàng***

Dựa vào thông tin do người sử dụng cung cấp và kết quả phân tích của hệ thống, phân lớp các tiến trình và đưa ra chiến lược phục vụ tương ứng.

Các tiến trình có thể được phân thành các lớp:

- ✓ Tiến trình thời gian thực
- ✓ Tiến trình của chế độ sử dụng tập thể phân chia thời gian
- ✓ Tiến trình xử lý lô

## CÂU HỎI VÀ BÀI TẬP

4.1. Anh chị hãy cho biết trên hệ điều hành đang dùng hiện đã sử dụng chiến lược điều khiển tiến trình nào? Cho ví dụ minh họa

4.2. So sánh nguyên tắc, ưu nhược điểm của các chiến lược điều độ tiến trình trong chế độ một dòng xếp hàng.

4.3. Xây dựng chương trình nhận 1 ký tự chữ thường từ bàn phím và chuyển thành ký tự chữ hoa.

4.4. Xây dựng chương trình thường trú để giám sát các ứng dụng thực hiện trên hệ điều hành Windows

# Chương V: HỆ ĐIỀU HÀNH NHIỀU PROCESSOR

## 5.1. Hệ điều hành nhiều Processor

Sự kết hợp của các Processor trong một hệ thống tính toán, sự kết hợp của các hệ thống tính toán đơn Processor.

Mục đích:

Sự chuyên môn hoá các Processor làm giảm gánh nặng xử lý

- ✓ Hoạt động ổn định và năng suất cao
- ✓ Độ tin cậy cao
- ✓ Làm cho các tài nguyên có giá trị cao, mang tính khả dụng đối với bất kỳ người dùng người dùng nào trên mạng.
- ✓ Tăng độ tin cậy của hệ thống nhờ khả năng thay thế khi xảy ra sự cố đối với một máy tính nào đó

### *5.1.1. Cấu hình nhiều Processor*

Tồn tại nhiều phương thực kết nối hai hay nhiều Processor.

Sự kết hợp của máy tính với các hệ thống truyền thông, đặc biệt là mạng viễn thông đã tạo lên mô hình tập trung các máy tính đơn lẻ được kết nối với nhau để cùng thực hiện công việc. Môi trường làm việc nhiều người dùng, cho phép nâng cao hiệu quả khai thác tài nguyên chung từ những vị trí địa lý khác nhau (bộ nhớ, chương trình, nhiệm vụ...)

***Cấu hình phân cấp: Client/ Server:*** một Processor ngoại vi và có thể hoạt động độc lập trong khi giải quyết nhiệm vụ của mình.

Đặc điểm:

- Chương trình dễ tổ chức
- Chương trình điều khiển không phải sao chép nhiều lần.
- Không phải tổ chức kiểu module vào/ra nhiều lần
- Thực hiện ngắt tăng.

***Sơ đồ liên kết mềm linh hoạt:*** Các processor có quan hệ bán phụ thuộc

- Mỗi processor xử lý tiến trình của mình từ khi hình thành tới khi kết thúc.
- Các processor có thể liên hệ, trao đổi thông tin và chuyển giao tiến trình trước khi nó được bắt đầu thực hiện.

Đặc điểm:

- Giảm gánh nặng xử lý tại một processor
- Các processor có thể trao đổi tiến trình, cơ chế điều độ đơn giản

***Sơ đồ liên kết bình quyền:*** Các processor được coi như tập các tài nguyên cùng loại

Thay cho việc thực hiện từng chương trình trên từng processor, phân chia công việc điều khiển cho tất cả các processor. Như vậy một tiến trình có thể bắt đầu ở processor này nhưng có thể kết thúc ở processor khác.

Đặc điểm:

- Giảm gánh nặng xử lý tại một processor
- Các processor có thể trao đổi tiến trình, cơ chế điều độ đơn giản
- Khó đánh giá kết quả thực hiện tiến trình

### *5.1.2. Hệ điều hành nhiều processor:*

Tồn tại một hệ điều hành có chức năng quản ly dữ liệu, tính toán và xử lý một cách thống nhất: Hệ thống như vậy gọi là hệ điều hành nhiều processor.

Với các tiếp cận:

- Tập trung: Tôn trọng hệ điều hành cục bộ đã có trên các hệ thống tính toán, hệ điều hành nhiều processor được cài đặt như một tập các chương trình tiện ích chạy trên hệ thống.
- Phân tán: Bỏ qua hệ điều hành cục bộ đã có trên các hệ thống, cài đặt một hệ điều hành thuần nhất trên toàn mạng

Với mô hình tập trung:

- Cung cấp cho mỗi người dùng một tiến trình đồng nhất làm nhiệm vụ cung cấp một giao diện đồng nhất với tất cả các hệ thống cục bộ đã có
- Tiến trình này quản lý cơ sở dữ liệu chứa thông tin về hệ thống cục bộ và về các chương trình và dữ liệu của người dùng thuần tuý:
  - ✓ Bộ xử lệnh
  - ✓ Dựng các lệnh của người dùng → ngôn ngữ lệnh của hệ thống → gửi tới P để thực hiện
- Đặc điểm:
  - ✓ Đơn giản, không làm ảnh hưởng tới các hệ thông cục bộ đã có
  - ✓ Khó thực hiện I/O

Với mô hình Phân tán:

- Mô hình tiến trình: Mỗi tài nguyên được quản lý bởi một tiến trình nào đó và hệ điều hành điều khiển sự tương tác giữa các tiến trình đó
- Mô hình đối tượng: Coi các tiến trình và các đối tượng, mỗi đối tượng có một kiểu, một biểu diễn và một tập các thao tác có thể thực hiện trên nó
  - o Như vậy:
  - o Tiến trình của uses phải được phép thao tác trên đối tượng
  - o Hệ điều hành quản lý việc thao tác của tiến trình trên đối tượng.

## 5.2. Hệ điều hành phân tán (Distribute Operating System)

### *5.2.1. Khái niệm:*

Tập các chương trình phục vụ tập trung như một giao diện quá trình ứng dụng và hệ thống tính toán nhằm đạt được tính hiệu quả an toàn, dễ sử dụng hệ thống tính toán.

Chức năng của hệ điều hành :

- Điều độ Processor
- Đồng bộ giữa các quá trình tương tác
- Quản lý tài nguyên hệ thống
- Đảm bảo điều khiển truy nhập và bảo vệ tính toàn vẹn hệ thống, phục hồi và cung cấp giao diện người dùng

Quan niệm về hệ điều hành:

- Máy ảo: Trừu tượng hoá hệ thống máy tính (mục tiêu thiết kế cơ bản)
- Quản trị tài nguyên : Phương tiện để đạt được mục đích

Như vậy:

- Hệ điều hành tập trung: Quan tâm tới việc quản trị tài nguyên hệ thống
- Hệ điều hành phân tán: Trừu tượng hoá máy tính

Vào thời điểm mới ra đời: Các hệ điều hành được thiết kế tập trung để chạy trên các hệ thống có một hay nhiều bộ xử lý (Processor)

Với tiếp cận mạng máy tính ngày nay: Các hệ thống làm trên phạm vi rộng, phân tán ở nhiều địa điểm khác nhau đòi hỏi cơ chế quản lý phân tán.

### *5.2.2. Đặc trưng của hệ điều hành phân tán*

#### *a. So với PC*

Khả năng dùng chung dữ liệu:

- Nhiều PC dùng trên nhiều bản sao của dữ liệu tại nhiều nơi, vì vậy chi phí cho việc đồng bộ quản lý truy nhập và bảo mật tốn kém
- MSDOS: Dùng trên một số ít các bản dữ liệu, chi phí giảm

Khả năng dùng chung thiết bị:

- Mỗi PC phải trang bị đầy đủ các thiết bị ngoại vi song nếu được kết nối trong môi trường MSDOS các thiết bị ngoại vi có thể được sử dụng chung bởi nhiều người dùng trong hệ thống, nhờ vậy tiết kiệm và hiệu quả

Khả năng truyền thông:

- Kết nối của PC nhờ dịch vụ mạng viễn thông, thời gian chờ đợi để được phục vụ là không an toàn.
- Với MSDOS: môi trường phân tán

Tính linh hoạt:

- Việc phân chia lại tài nguyên gây ra chi phí tốn kém: lưu chuyển tài liệu, thiết bị, dữ liệu...
- MSDOS: Sử dụng các chức năng chuyên biệt của hệ thống

#### *b. So với hệ điều hành tập trung*

Tốc độ: Năng lực kế toán cao khi tập trung một số bộ vi xử lý trên một máy tính

Tính kinh tế: Tỷ suất giá cả hiệu nâng cao

Tính phân bố: Liên kết các ứng dụng trên các máy riêng biệt

Tính ổn định và tin cậy: Hệ thống vẫn làm việc khi một máy gặp sự cố

Tính mở: Có thể từng bước mở rộng quy mô hệ thống

#### *c. Hạn chế của hệ điều hành phân tán*

- Phần mềm: đòi hỏi hệ điều hành, các ngôn ngữ hình thức, các chương trình ứng dụng phù hợp: thiết kế, cài đặt khó, phức tạp
- Vấn đề mạng: Thay thế toàn bộ hệ thống cũ
- Vấn đề truyền thông: an toàn dữ liệu, lưu lượng đường truyền, quá trình thay thế khi có sự cố.
- Vấn đề bảo mật: Giá thành cao, khó sử dụng chung dữ liệu, chương trình

#### *d. Yêu cầu thiết kế hệ điều hành phân tán*

Tính trong suốt

- Tính trong suốt với người dùng: người dùng nghĩ rằng hệ thống phân tán chỉ là một tập máy tính hoạt động ở chế độ phân chia thời gian
- Tính trong suốt hệ thống: hệ thống trong suốt đối với chương trình, lời gọi hệ thống phải được thiết kế sao cho sự có mặt của nhiều processor là không thể thấy được từ chương trình

Thể hiện:

- *Trong suốt về định vị:* người dùng không thể nói chính xác các tài nguyên nằm ở đâu (tài nguyên được mã hoá vị trí)
- *Trong suốt về ánh xạ:* tên tài nguyên không thay đổi khi di chuyển từ máy này sang máy khác
    - *Trong suốt về lặp lại:* hệ thống có thể (và cần thiết) lưu một số bản sao của cùng một tài nguyên mà người dùng không biết

- *Trong suốt đồng thời:* nhiều tiến trình có thể cùng truy nhập một tài nguyên, các tiến trình có thể không cần biết tới sự có mặt của tiến trình khác
- *Trong suốt song song:* nhiều hoạt động song song được che đối với người dùng
- *Trong suốt lỗi:* cơ chế phục hồi lỗi trong hệ thống được che đối với người dùng
- *Trong suốt kích thước:* cho phép hệ hống mở rộng qui mô dần dần ma không tác động tới người dùng
- *Trong suốt về quan sát*: điểm nhìn phần mềm không thể thấy được đối với người dùng.

Tính modul hoá: Hệ thống được phân chia làm nhiều modul như cho phép bổ sung, thay đổi dễ dàng

Tính khả mở: qui mô hệ thống thường xuyên thay đổi do các yêu cầu nâng cấp

Tính độc lập, quy mô: Mở rộng quy mô mà năng lực hệ thống không thay đổi

Tính chịu lỗi: Thường xuyên sao lưu để phục hồi lỗi

## 5.3. Quản lý tài nguyên trong hệ điều hành phân tán

### *5.3.1. Quản lý thiết bị, quản lý File*

Khái niệm File: đơn vị thông tin nhỏ nhất của người dùng, được quản lý thông qua tên file.

- Người dùng phải lưu trữ thông tin ở bộ nhớ ngoài vì vậy hệ điều hành phải có vai trò sao cho người dùng truy nhập thuận tiện
- Nhu cầu dùng chung (chia sẻ) các file dữ liệu.
- Vấn đề đặt ra đối với hệ thống quản lý file: ngoài các tính chất và yêu như đối với hệ quản lý file trong hệ điều hành tập trung, hệ quản lý file trong hệ điều hành phân tán phải đảm bảo:
- Tính trong suốt của hệ thống
- Dịch vụ thư mục
- Hiệu năng hệ thống, độ tin cậy
- Độ an toàn.

***Tính trong suốt***

- Tính trong suốt đăng nhập: người dùng có thể đăng nhập vào các trạm trong hệ thống với cùng một thủ tục đăng nhập
- Trong suốt truy cập: Các tiến trình chạy trên hệ thống có cùng cơ chế truy nhập vào các tệp tin mà không cần để ý xem tệp đó là cục bộ hay từ xa
- Sự độc lập về định vị tệp tin: Các tệp tin có thể được chuyển từ vị trí này tới vị trí khác mà không làm thay đổi tên: trong suốt đối với người dùng
- Tính trong suốt tương tranh: các file được chia sẻ bởi nhiều người dùng, việc truy cập tới một tệp từ một tiến trình không ảnh hưởng tới sự thành lập của tiến trình khác
- Trong suốt lặp: Các tệp được sao lưu để dự phòng cho phép truy nhập đồng bộ (người dùng không biết các bản sao).

***Thiết kế và thực hiện hệ thống tệp tin phân tán***

Đối với người dùng, một tệp tin bao gồm ba thành phần logic:

- Tên tệp và hệ thống tệp
- Các thuộc tính
- Các đơn vị dữ liệu

Các tệp và hệ thống tệp

Các tệp được tạo ra bởi người dùng đi kèm với tên, khi truy nhập tệp, tên tệp sẽ xác định giá trị ID của tệp và giá trị này cũng là giá trị duy nhất xác định vị trí vật lý của tệp

Các thuộc tính: Các thông tin về quyền sở hữu, quyền truy nhập, dạng tệp, kích thước, dấu hiệu thời gian

Các đơn vị dữ liệu

Đơn vị dữ liệu : Byte, khối

Cơ chế truy nhập:

- Tuần tự: Con trỏ định vị tệp được duy trì bởi hệ thống cho phép xác định vị trí đơn vị dữ liệu kế tiếp được truy nhập giữa các tiến trình
- Trực tiếp (truyền thông không liên kết): Vị trí đơn vị dữ liệu cho việc đọc, ghi là rõ ràng. Cơ chế này liên quan tới kích thước của đơn vị dữ liệu, các thao tác đọc ghi phải bao hàm các thông tin điều khiển.
- Chỉ số: Đơn vị dữ liệu được địa chỉ hoá bởi chỉ số hay khoá đi kèm mỗi khối dữ liệu.

***Vấn đề bảo mật***

Bảo vệ dữ liệu: không để mất thông tin khi có sự cố kỹ thuật hoặc chương trình thậm chí truy nhập bất hợp lệ

Kĩ thuật bảo vệ dữ liệu cho hệ phân tán:

Phương pháp mã hoá dữ liêu với thuật toán DBS

- Khoá bí mật: Thuật toán giải mã
- Khoá công khai: sinh mã:

Kerberos: Sự xác nhận là đúng của các thành phần dựa trên cơ sở tin tưởng vào thành phần thứ 3 (mật khẩu)

Chữ ký điện tử: Xác nhận tính nguyên bản mà các văn bản (Digital Signature)

### *5.2.2. Quản lý bộ nhớ*

Ngoài các phương pháp quản lý bộ nhớ như trong hệ điều hành tập trung, vấn đề quan tâm trong việc quản lý bộ nhớ ở hệ điều hành phân tán đó là việc đảm bảo tính chia sẻ bộ nhớ.

Chia sẻ bộ nhớ: Truy nhập bộ nhớ từ xa:

Việc truy nhập được thực hiện tại một nút xa

- Khối dữ liệu xa được di chuyển tới nút cục bộ: truy nhập cục bộ
- Khối dữ liệu xa được sao lưu lại tại nút cục bộ: truy nhập đồng bộ

***Các phương thức***

- Đọc từ xa (Read remote): Khối dữ liệu dùng chung không được di chuyển hay sao lưu, máy trạm gửi yêu cầu tới máy chủ, máy chủ gửi trả lời về dữ liệu cho việc đọc, và báo nhận cho việc ghi.
- Đọc/ghi ánh xạ (Read/write migrate): Nhờ việc truy cập tới một khối dữ liệu từ xa mà khối dữ liệu được di chuyển tới tiến trình yêu cầu. Tiến trình sẽ cập nhập tới bảng ánh xạ khối vật lý - trang ảo của dữ liệu

### *5.2.3. Quản lý tiến trình*

Khái niệm về tiến trình: Đơn vị thực hiện được nhỏ nhất thấy bởi người dùng

Luồng (Thread): đơn vị thực hiện được nhỏ nhất thấy bởi hệ điều hành, được hệ điều hành cấp phát thời gian Processor.

***Quan hệ giữa tiến trình và luồng:***

Tiến trình là không gian địa chỉ trong đó luồng được thực hiện. Hai tiến trình cùng không gian địa chỉ - hai luồng thuộc một tiến trình.

***Quản lý tiến trình:*** việc quản lý các tiến trình thông qua các khối điều khiển tiến trình.

Khối điều khiển: Bản ghi chứa các khối điều khiển các luồng, các cổng thông tin , các tài nguyên hệ thống mà tiến trình đang sử dụng, các thông tin trạng thái tiến trình: Sẵn sàng, thực hiện và ngắt

***Quản lý luồng:*** Khối điều khiển luồng:

Bộ đếm lệnh: PC

- Con trỏ ngăn xếp: SP
- Tập các thanh ghi: Rs
- Trạng thái: Flag

Các chức năng quản lý tiến trình và luồng thông tin chia làm ba loại

- Truyền thông: Đảm bảo sự liên kết giữa các tiến trình
- Đồng bộ: Đảm bảo thực hiện các tiến trình tối ưu
- Điều độ : Đảm bảo các tiến trình sử dụng tài nguyên chia sẻ đúng đắn

***Cài đặt luồng:***

Trong không gian người dùng: Khi luồng gọi một thủ tục hệ thống nó thực hiện liên kết vào thư viện động. Thủ tục thư viện động kiểm tra xem có cần treo luồng đó không, nếu cần nó treo luồng này và chuyển điều khiển cho luồng khác.

Trong nhân hệ thống: Khi luồng gọi một thủ tục hệ thống, nó sẽ được gắn vào nhân hệ thống.

***Truyền thông giữa các tiến trình***

Mô hình truyền thống OSI: mỗi tầng có một chức năng riêng, thông điệp truyền giữa hai ứng dụng dựa trên giao thức, khi qua mỗi tầng nó được gắn thêm vùng header.

Mô hình Cilent/ Server

- Client truyền thông điệp cho server yêu cầu dịch vụ
- Server thực hiện dịch vụ tương ứng và gửi thông điệp trả lời

Các vấn đề:

- Định vị yêu cầu từ Clinet nào:
- Gắn cho mỗi Client một địa chỉ ID
- Client chọn địa chỉ ngẫu nhiên, thông báo được gửi cho taat cả các Server

Đưa tên Server vào Client khi chạy chương trình.

- Chế độ chuyển thông điệp: khoá, không khoá:
  - Khoá: khi có một thông điệp được chuyển, tiến trình của Client bị treo và chờ cho tới khi có trả lời hoặc báo lỗi
  - Không khoá: tiến trình vẫn tiếp tục thực hiện các công việc khác
- Chế độ có bảo đảm và không bảo đảm:
  - Có bảo đảm: Server nhận được thông điệp từ Client nó sẽ phúc đáp lại để Client biết.
  - Không bảo đảm: khi tiến trình gửi thông điệp, nó không được bảo đảm là thông điệp đã đến đích

Mô hình truyền thông nhóm:

Nhóm: tập các tiến trình, vì vậy khi một thành viên nhận được thông điệp tất cả các tiến trình trong nhóm đều có thể được chia sẻ.

***Đồng bộ các tiến trình***

Đồng bộ: Đảm bảo thứ tự thực hiện đúng đắn của các luồng, các tiến trình

Đồng bô đồng hồ thời gian thực:

Giả sử có tệp .OBJ trên một máy được biên dịch từ tệp A.ASM trên một máy khác. Từ một máy thứ ba, người dùng gọi trình LINK để tạo A.EXE từ A.OBJ. Trình liên kết so sánh thời gian cập nhập cuối cùng của A.OBJ và A.ASM để quyết định có biên dịch lại A.ASM hay không. Nếu đồng hồ của máy chứa tệp .OBJ nhanh hơn đồng hồ của máy chứa .ASM thì có thể .ASM đã cập nhật mà .OBJ vẫn mới hơn, kết quả là LINK không liên kết lại .ASM và dùng .OBJ cũ dẫn đến sai mà không biết vì sao.

Khắc phục: đồng bộ thời gian thực:

- Mô hình chuẩn tập trung: các máy trạm đều đặn gửi thông điệp hỏi giờ tới máy chủ chuẩn để thường xuyên hiệu chỉnh gìơ của mình.
- Mô hình chuẩn trung bình: máy chủ đều đặn hỏi các máy trạm giwof của chúng, tính trung bình rồi gửi phản hồi lại thời gian chung.
- Mô hình trung bình phân tán: chia thời gian thành các khoảng đồng bộ lại $t_i = T_0 + i*R$, cứ mỗi thời điểm $t_i$ mỗi máy gửi thời gian trỏ bởi đồng hồ của mình cho mọi máy khác và cũng nhận thời gian từ mọi máy khác gửi tới, tính trung bình và hiệu chỉnh đồng hồ của mình.

Đồng bộ thời gian logic

Giả sử tiến trình A gửi thông điệp cho B, thời điểm thông điệp xuất phát là $t_1$, thời điểm nhận thông điệp là $t_2$. Vì xung nhịp của hai máy khác nhau nên có thể $t_2 < t_1$ khi đó B sẽ huỷ thông điệp

Khắc phục: Trong thông điệp bao hàm cả thời gian xuất phát.

Đồng bộ thứ tự sử dụng đoạn Găng

Thuật toán tập trung: Tiến trình định sử dụng tài nguyên Găng nó sẽ gửi thông điệp tới server xem có quyến sử dụng không?

Thuật toán phân tán: Tiến trình sử dụng tài nguyên Găng nó sẽ gửi thông điệp tới các tiến trình khác. Các tiến trình khi nhận được thông điệp xin phép:

- Nếu nó không ở trong đoạn găng, không có nhu cầu sử dụng tài nguyên găng, nó sẽ cho phép.
- Nếu nó đang trong đoạn găng, nó không trả lời và xếp hàng thông điệp mới đến để trả lời sau.

Nếu nó đang định sử dụng đoạn găng, so sánh thời gian gửi thông điệp của nó trước đây với thời gian gửi của thông điệp mới đến, nếu thấy thông điệp này xuất phát trước nó sẽ tự động đi vào trạng thái chờ và trả lời cho phép.

## CÂU HỎI VÀ BÀI TẬP

5.1. Xây dựng chương trình truyền 1 ký tự giữa 2 máy tính

5.2. Xây dựng chương trình khởi động 1 chương trình từ xa trên hệ thống mạng máy tính

**ĐỀ THI THAM KHẢO (Thời gian làm bài 90 phút)**

**Đề 1:**

1. Trình bày hiểu biết của anh, chị về cấu trúc phân lớp và hệ thống tính toán. Cho ví dụ minh họa cụ thể
2. Trình bày hiểu biết của anh, chị về cơ chế, phân loại phòng đệm. Lấy ví dụ minh họa cụ thể.
3. Trình bày hiểu biết của anh, chị về quản lý bộ nhớ logic theo cấu trúc động, cấu trúc Overlay.
4. Trình bày hiểu biết của anh, chị về điều độ tiến trình theo phương pháp kiểm tra và xác lập
5. Trình bày hiểu biết của anh, chị về các đặc trưng của hệ điều hành phân tán

**Đề 2:**

1. Trình bày hiểu biết của anh, chị về nguyên tắc chung xây dựng hệ điều hành. Cho ví dụ minh họa cụ thể
2. Trình bày hiểu biết của anh, chị về quan hệ phân cấp trong tổ chức và quản lý thiết bị ngoại vi.
3. Trình bày hiểu biết của anh, chị về quản lý bộ nhớ vật lý theo chế độ phân chương động, chế độ phân đoạn
4. Trình bày hiểu biết của anh, chị về tắc nghẽn, các kỹ thuật khắc phục tắc nghẽn
5. Trình bày hiểu biết của anh, chị về hệ điều hành nhiều Processor

**Đề 3:**

1. Trình bày hiểu biết của anh, chị về các tính chất chung của hệ điều hành. Cho ví dụ minh họa cụ thể
2. Trình bày hiểu biết của anh, chị về cơ chế SPOOL. Cho ví dụ minh họa cụ thể
3. Trình bày hiểu biết của anh, chị về quản lý bộ nhớ vật lý theo cấu trúc kết hợp phân đoạn và phân trang.
4. Trình bày hiểu biết của anh, chị về điều độ tiến trình theo kỹ thuật đèn báo.
5. Trình bày hiểu biết của anh, chị về quản lý tài nguyên trong hệ điều hành phân tán

**Đề 4:**

1. Trình bày hiểu biết của anh, chị về định nghĩa về hệ điều hành
2. Trình bày hiểu biết của anh, chị về hệ thống quản lý file
3. Trình bày hiểu biết của anh, chị về các giai đoạn xử lý chương trình trên hệ thống máy tính
4. Trình bày hiểu biết của anh, chị về tiến trình: Khái niệm, tổ chức, trạng thái, phân loại tiến trình
5. Trình bày hiểu biết của anh, chị về quản lý bộ nhớ trong hệ điều hành phân tán.

**Đề 5:**

1. Trình bày hiểu biết của anh, chị về
2. Trình bày hiểu biết của anh, chị về
3. Trình bày hiểu biết của anh, chị về quản lý bộ nhớ vật lý theo chế độ phân chương tĩnh và chế độ phân đoạn.
4. Trình bày hiểu biết của anh, chị về tài nguyên găng, các chiến lược điều độ tiến trình qua đoạn găng.
5. Trình bày hiểu biết của anh, chị về quản lý tiến trình trong hệ điều hành phân tán.

# Hệ điều hành

**Nguyễn Thanh Bình**
Khoa Công nghệ thông tin
Trường Đại học Bách khoa
Đại học Đà Nẵng

## Nội dung

- Các khái niệm
- Hệ điều hành Windows

# Hệ điều hành (HĐH)

- Hệ điều hành (Operating System)
  - là tập hợp các chương trình nhằm điều khiển, quản lý, phân phối sử dụng tài nguyên (CPU, bộ nhớ, các thiết bị ngoại vi, …)
  - là hệ chương trình hoạt động như lớp trung gian giữa người sử dụng và máy tính
- HĐH là thành phần không thể thiếu
- Có nhiều HĐH đang được sử dụng
  - MSDOS, Windows, Unix, Linux, Macintosh, …
- Phổ biến ở Việt Nam: MSDOS, Windows và gần đây là Linux



# Hệ điều hành

- Các chức năng cơ bản của HĐH
  - Quản lý bộ nhớ
  - Quản lý các thiết bị ngoại vi
  - Quản lý nhập/xuất
  - Điều khiển/thực hiện chương trình
- Mục đính là sử dụng phần cứng một cách hiệu quả

# Các khái niệm

- Bộ kí tự ASCII (American Standard Code for Information Interchange): bảng mã chuẩn dùng để trao đổi thông tin
  - Các kí tự: a, b, c, …
  - Các chữ số: 0, 1, … 9
  - Các kí hiệu toán học: >, <, +, -, …
  - Các kí hiệu đặc biệt: @, $, #, …



# Các khái niệm

- Tệp tin (file)
  - Các dữ liệu liên quan với nhau được lưu trữ trên đĩa thành đơn vị, gọi là tệp tin
  - HĐH quản lý thông tin trên đĩa theo **đơn vị tệp**
  - Tệp tin phải có tên, tên tệp tin gồm hai phần:
    - Phần tên
    - Phần mở rộng
    - Hai phần cách nhau dấu '.'
  - Ví dụ
    - Các tên tệp tin: baitap.txt, tinhtong.c, …

# Các khái niệm

- Tệp tin
  - Phần tên là bắt buộc phải có
    - MSDOS chỉ cho phép độ dài tối đa là 8 kí tự
    - Windows có giới hạn lớn hơn (255 kí tự)
  - Phần mở rộng là không bắt buộc
    - MSDOS chỉ cho phép độ dài tối đa là 3 kí tự
    - Windows không giới hạn
    - Phần mở rộng nêu lên đặc trưng loại tệp tin
      - Tệp chương trình: COM, EXE, …
      - Tệp văn bản: C, PAS, TXT, …



# Các khái niệm

- Tệp tin
  - Tên tệp tin có thể sử dụng các kí tự:
    - a..z, A..Z, 0..9, một số kí tự đặc biệt _, $, @, …
    - MSDOS không cho phép sử dụng kí tự trống
    - Windows cho phép sử dụng kí tự trống
    - Không sử dụng một số kí tự đặc biệt: , ; * > …
  - MSDOS và Windows không phân biệt chữ in hoa hay in thường
    - Ví dụ: test.c và TEST.C là như nhau

# Các khái niệm

- Thư mục (directory)
  - Các tệp tin có nội dung liên quan đến nhau được đặt vào trong cùng một thư mục
  - Dễ quản lý các tệp tin
  - Thư mục có thể chứa các tệp tin và các thư mục khác,
  - Thư mục chứa trong thư mục khác gọi là **thư mục con** (sub-directory)
  - MSDOS quy ước đối với mỗi ổ đĩa, có một thư mục chứa tất cả các thư mục và tệp tin khác, đó là **thư mục gốc** (root directory)
    - Thư mục gốc có tên quy ước là "\"
    - Thư mục gốc không thể bị đổi tên hay bị xóa

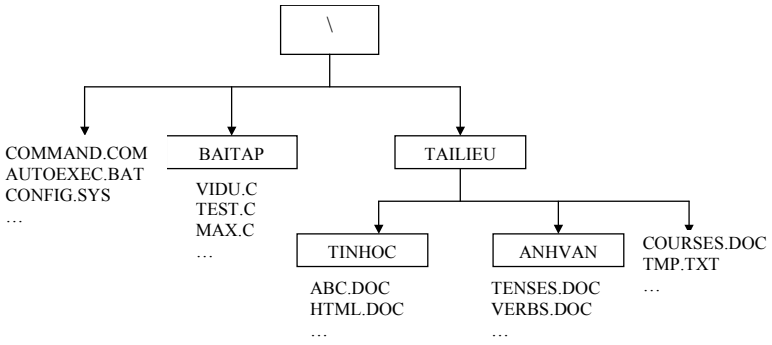


# Các khái niệm

- Thư mục (directory)
  - Thư mục đang làm việc được gọi là **thư mục hiện hành** (current directory)
    - Tên viết tắt của thư mục hiện hành là "."
    - HĐH chỉ thị thư mục hiện hành qua dấu đợi lệnh
      - Ví dụ: C:\ABC>_
  - Thư mục trên thư mục hiện hành một cấp được gọi là **thư mục cha** (parent directory)
    - Tên tắt của thư mục cha là ".."
  - Tổ chức của các thư mục trên một ổ đĩa được gọi là **cây thư mục** (directory tree)

# Các khái niệm

- Ví dụ: cây thư mục






# Các khái niệm

- Ổ đĩa
  - Ổ đĩa mềm, ổ đĩa cứng, CD, DVD, USB, …
  - Ổ đĩa mềm thường có tên là: A, B
  - Ổ đĩa cứng, CD, USB, … thường có tên là: C, D, …
  - Khi sử dụng các lệnh MSDOS, tên ổ đĩa phải kèm theo dấu ":"
    - Ví dụ: A:, C:

# Các khái niệm

- Đường dẫn (path)
  - Đường dẫn dùng để chỉ ra **vị trí** của tệp tin hoặc thư mục trong một cây thư mục
    - Ví dụ: C:\TAILIEU\TINHOC\ABC.DOC
  - **Đường dẫn tuyệt đối**: được bắt đầu bằng tên của ổ đĩa hoặc thư mục gốc
    - Ví dụ: C:\BAITAP\VIDU.C
  - **Đường dẫn tương đối** (so với thư mục hiện hành): được bắt đầu bằng tên của một thư mục con
    - Ví dụ: thư mục hiện hành là TAILIEU
      TINHOC\ABC.DOC



# Nội dung

- Các khái niệm
- Hệ điều hành Windows

# HĐH Windows

- Giới thiệu
  - Sản phẩm của công ty Microsoft
  - HĐH được sử dụng phổ biến nhất
  - Dễ sử dụng
    - Chủ yếu sử dụng chuột
  - Lịch sử phát triển: nhiều phiên bản khác nhau đã ra đời
    - Windows 3.1, Windows 95, Windows 98, Windows 2000, Windows XP (dành cho máy tính cá nhân)
    - Windows NT, Windows Server (dành cho máy chủ)

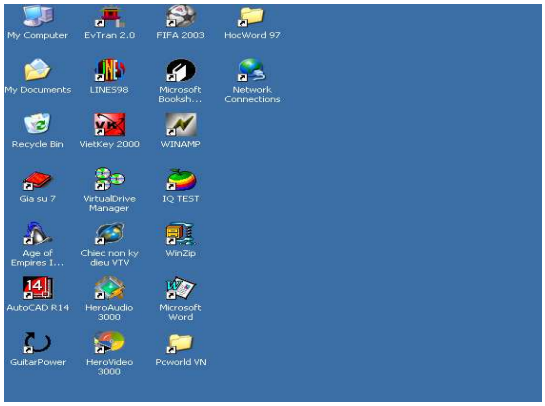


# HĐH Windows

- Các thành phần cơ bản
  - Màn hình giao tiếp Desktop
  - Thanh ứng dụng TaskBar
  - Nút Start
  - Thao tác với chuột
  - Cửa sổ

# HĐH Windows

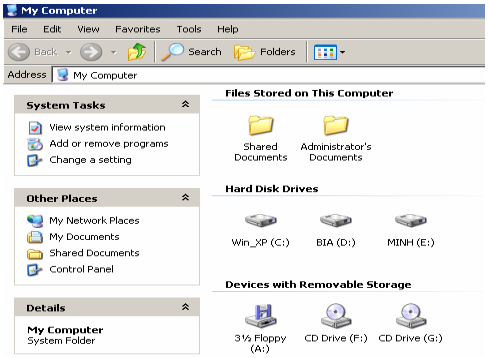
- Màn hình giao tiếp Desktop






# HĐH Windows

- Một số biểu tượng cơ bản trên
  - My computer: hiển thị toàn bộ thông tin trên máy tính



# HĐH Windows

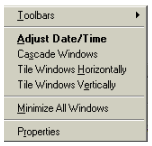
- Một số biểu tượng cơ bản trên
  - Network places: thiết lập các thuộc tính kết nối mạng của máy tính
  - Recycle Bin: thùng rác, chứa dữ liệu bị xóa (có thể phục hồi lại được)
  - Các biểu tượng ShortCut khác: các liên kết đến các ứng dụng



# HĐH Windows

- Thanh ứng dụng TaskBar
  - Chứa nút Start và danh sách các ứng dụng đã được kích hoạt


  - Di chuyển TaskBar: theo 4 cạnh của desktop
  - Thao tác trên TaskBar
    - Nhắp nút chuột phải trên vùng trống của TaskBar



# HĐH Windows

- Nút Start
  - Nằm ở góc trái bên dưới desktop, nhấp chuột vào nút này cho menu (trình tự chọn) nhiều cấp



# HĐH Windows

- Nút Start
  - Programs: chứa danh sách các ứng dụngđược cài đặt trên máy tính
  - Settings: dùng để thiết lập một số tham số cho máy tính
  - Search: tìm kiếm dữ liệu trên máy hoặc trên mạng
  - Run: chạy một chương trình bằng dòng lệnh
  - Log off: thoát ra khỏi một user
  - Turn off computer: tắt máy

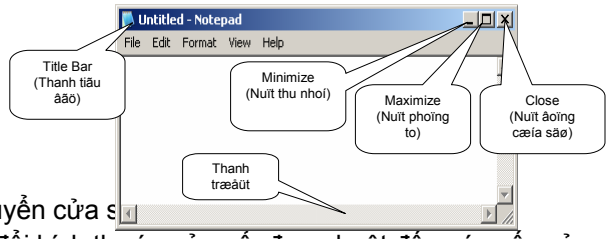
# HĐH Windows

- Thao tác với chuột (Mouse)
  - Dùng để nhập thông tin
  - Các thao tác với chuột:
    - Click (nhắp đơn): nhắp tay vào chuột trái một lần
    - Double click (nhắp đôi): nhắp vào chuột trái hai lần liên tục, tương đương với Click + Enter
    - Drag and Drop (kéo và thả): nhắp chuột vào một biểu tượng, giữ tay và kéo đến nơi cần thả
    - Right click (nhắp chuột phải): nhắp chuột phải để hiển thi menu tắt (nhanh) hoặc thực hiện lệnh tắt nào đó



# HĐH Windows

- Cửa sổ (Windows)
  - Mỗi ứng dụng khi chạy đều được thể hiện trên một cửa số




  - Di chuyển cửa s
  - Thay đổi kích thước cửa số: đưa chuột đến các góc cửa sổ và kéo rê

# HĐH Windows

- Thao tác trên Desktop
  - Mở một biểu tượng
    - Nhắp đôi trên biểu tượng
    - Nhắp chuột phải, chọn **Open**
  - Sắp xếp các biểu tượng trên desktop
    - Thực hiện kéo và thả các biểu tượng
    - Nhắp chuột phải trên vùng trống của desktop và chọn **Arrange Icons By** theo các tiêu chuẩn: name (tên), type (kiểu dữ liệu), size (kích thước), modified (ngày sửa đổi), auto arrange (máy tự động sắp xếp)

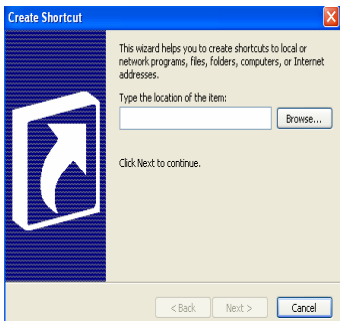


# HĐH Windows

- Thao tác trên Desktop
  - Tạo một mới một biểu tượng Shortcut
    - Tạo bằng wizard: kích chuột phải trên vùng trống desktop, chọn **New ShortCut**

# HĐH Windows

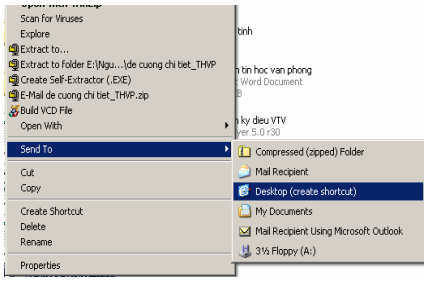
- Thao tác trên Desktop
  - Tạo một mới một biểu tượng Shortcut
    - Tạo bằng wizard






# HĐH Windows

- Thao tác trên Desktop
  - Tạo một mới một biểu tượng Shortcut
    - Tạo bằng **Send To**: trong My Computer hoặc Windows Explorer nhắp chuột phải rồi chọn **Send To \ desktop (create shortcut)** sẽ xuất hiện:



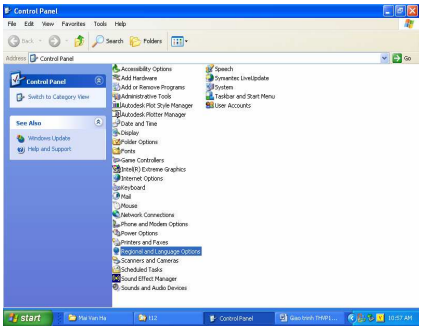
# HĐH Windows

- Thao tác trên Desktop
  - Đổi tên một biểu tượng
    - Nhắp chuột phải trên biểu tượng, chọn **Rename** sau đó gõ tên mới và kết thúc bằng nhấn **Enter**
  - Xóa biểu tượng
    - Chọn biểu tượng (nhắp chuột phải), nhấn phím **Delete**
    - Nhắp chuột phải trên biểu tượng và chọn **Delete**



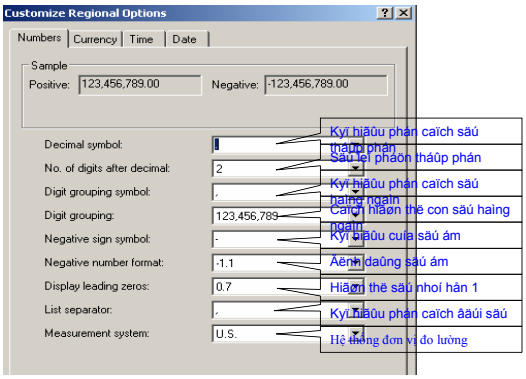
# HĐH Windows

- Thiết lập các thuộc tính của Windows
  - Thay đổi kiểu số, kiểu ngày và giờ hệ thống
    - Chọn nút **Start \ Settings \ Control** Panel màn hình sau sẽ được hiển thị



# HĐH Windows

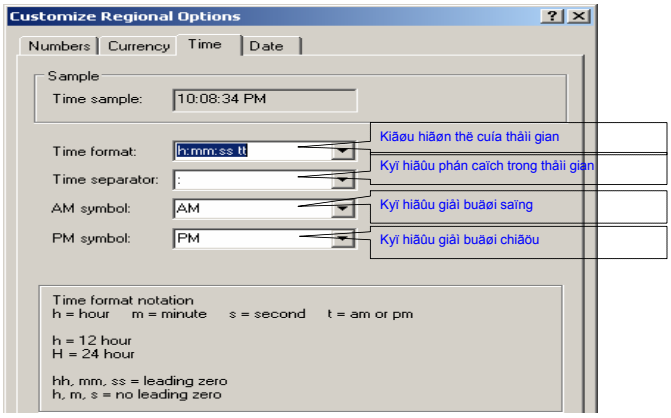
- Thay đổi kiểu số, kiểu ngày và giờ hệ thống
  - Chọn **Regional and Language Options**
    - Thay đổi kiểu số (cách hiển thị)





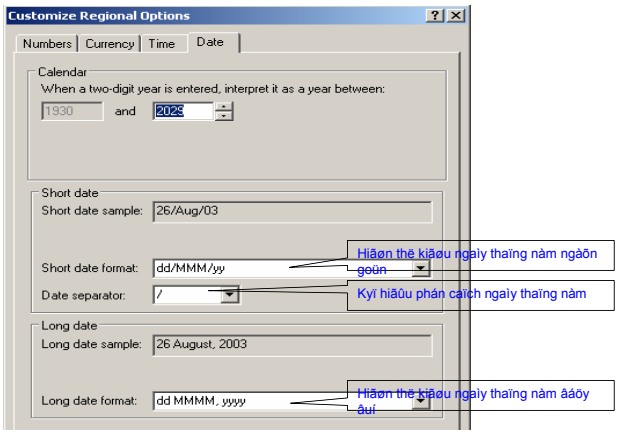

# HĐH Windows

- Thay đổi kiểu hiển thị giờ



# HĐH Windows

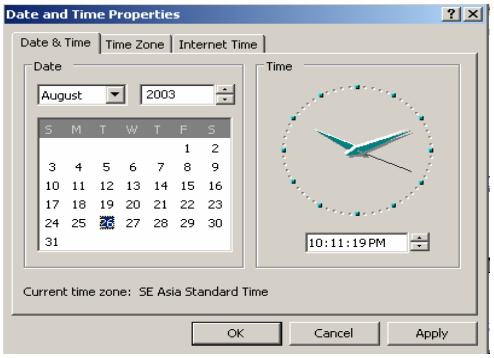
- Thay đổi kiểu hiển thị ngày






# HĐH Windows

- Hiệu chỉnh thời gian hệ thống
  - Nhấp đôi chuột trái vào đồng hồ gốc phải bên dưới trên thanh TaskBar



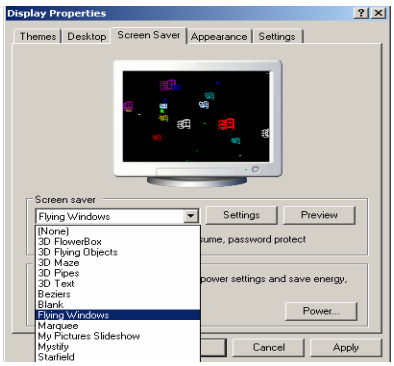
# HĐH Windows

- Thay đổi thiết lập màn hình
  - Nhắp chuột phải trên vùng trống desktop và chọn **properties**
  - Hoặc chọn **Start \ Settings \ Control Panel \ Display**




# HĐH Windows

- Thay đổi thiết lập màn hình
  - Chọn **desktop** để thay đổi màn hình nền
  - Chọn **Screen Saver** để thay đổi các thuộc tính của màn hình bảo vệ



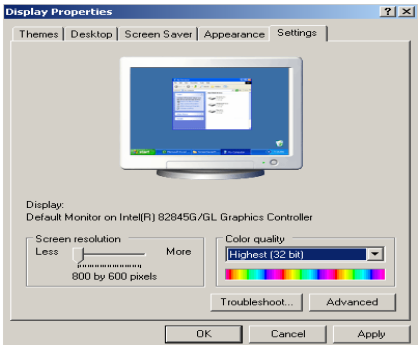
# HĐH Windows

- Thay đổi thiết lập màn hình
  - Chọn **Apparence** để thay đổi chế độ màu sắc, font chữ, … cho desktop




# HĐH Windows

- Thay đổi thiết lập màn hình
  - Chọn **Settings** để thay đổi độ phân giải và chế độ màu của màn hình



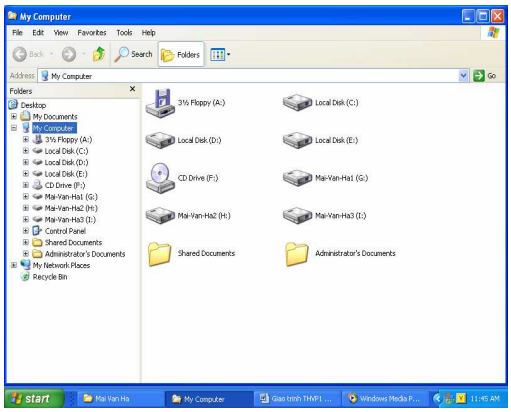
# HĐH Windows

- Làm việc với Windows Explore
  - Công cụ hỗ trợ tìm kiếm, sao chép, … dữ liệu trên máy tính
  - Các cách khởi động
    - Chọn **Start \ Programs \ Windows Explore**
    - Chọn biểu tượng **My Computer** trên desktop
    - Nhắp chuột phải trên nút **Start** và chọn **Explore**
  - **Thoát Explore**
    - Chọn **File \ Close**
    - Nhấn tổ hợp phím **Alt + F4**
    - Nhắp chuột nút (X) gốc phải trên cao cửa sổ



# HĐH Windows

- Màn hình Explore xuất hiện



# HĐH Windows

- Các thành phần của Explore
  - Thanh tiêu đề (Title Bar): chứa tên ổ đĩa, hoặc thư mục
  - Thanh trình tự chọn (Menu Bar): chứa toàn bộ các lệnh của Explore
  - Thanh công cụ (Tool Bar): chứa một số các nút biểu tượng cho phép chọn nhanh các lệnh của Explore
  - Thanh trạng thái (Status Bar): cho biết chương trình nào đang được mở
  - Khung trái cửa sổ (Tree View): hiển thị cây thư mục tất cả các dữ liệu trên máy tính
  - Khung phải cửa sổ (List View): hiển thị nội dung của đối tượng hiện hành trên khung trái



# HĐH Windows

- Các thành phần của Explore
  - Khung phải cửa sổ (List View)
    - Nội dung trên khung trái có thể được hiển thị nhiều dạng khác nhau, bằng cách nhắp chuột phải trên vùng trống khung phải và chọn **View \ ...**
      - **Titles** hiển thị các tệp tin và thư mục dưới dạng biểu tượng lớn
      - **Icons** hiển thị các tệp tin và thư mục dưới dạng biểu tượng nhỏ
      - **List** hiển thị các tệp tin và thư mục dưới dạng danh sách theo thứ tự alphabet
      - **Details** hiển thị các tệp tin và thư mục dưới dạng danh sách với các chi tiết (kích thước, loại đối tượng, ngày tạo ra)

# HĐH Windows

- Thao tác với tệp tin và thư mục trên Explore
  - Chọn đối tượng
    - Chọn một đối tượng: nhấp chuột vào đối tượng đó
    - Chọn một nhóm các đối tượng liên tiếp nhau: nhấp chuột vào đối tượng đầu tiên muốn chọn, sau đó nhấn và giữ phím **Shift** đồng thời nhấp chuột vào đối tượng cuối muốn chọn
    - Chọn nhóm các đối tượng không liên tiếp: ấn phím **Ctrl** và đồn thời nhấp chuột vào các đối tượng cần chọn
    - Chọn tất cả các đối tượng: Chọn **Edit \ Select All** hoặc nhấn tổ hợp phím **Ctrl + A**
  - Hủy việc lựa chọn
    - Nhấn phím **Ctrl** và đồng thời nhấp chuột vào đối tượng muốn hủy lựa chọn



# HĐH Windows

- Thao tác với tệp tin và thư mục trên Explore
  - Di chuyển tệp tin hoặc thư mục: 2 cách
    - Dùng lệnh
      - Chọn tệp tin hoặc thư mục cần di chuyển
      - Chọn trên trình đơn **Edit \ Cut**
      - Chọn vị trí đích
      - Chọn trên trình đơn **Edit \ Paste**
    - Dùng chuột
      - Chọn tệp tin hoặc thư mục cần di chuyển
      - Kéo rê đối tượng đã được chọn từ vị trí nguồn đến vị trí đích (muốn chuyển từ ổ đĩa này sang ổ đĩa khác khi kéo rê phải nhấn thêm phím **Shift**)

# HĐH Windows

- Thao tác với tệp tin và thư mục trên Explore
  - Sao chép tệp tin hoặc thư mục: 2 cách
    - Dùng lệnh
      - Chọn tệp tin hoặc thư mục cần sao chép
      - Chọn trên trình đơn **Edit \ Copy**
      - Chọn vị trí đích
      - Chọn trên trình đơn **Edit \ Paste**
    - Dùng chuột
      - Chọn tệp tin hoặc thư mục cần sao chép
      - Nhấn phím **Ctrl** đồng thời kéo rê đối tượng đã được chọn từ vị trí nguồn đến vị trí đích (muốn chuyển từ ổ đĩa này sang ổ đĩa khác thì khi kéo rê không cần phải nhấn phím **Ctrl**)



# HĐH Windows

- Thao tác với tệp tin và thư mục trên Explore
  - Đổi tên tệp tin hoặc thư mục
    - Chọn tệp tin hoặc thư mục cần đổi tên
    - Nhấp chuột phải trên đối tượng, sau đó chọn **Rename**
    - Gõ vào tên mới và kết thúc bởi phím **Enter**
  - Xóa tệp tin hoặc thư mục
    - Chọn tệp tin hoặc thư mục cần xóa
    - Nhấp chuột phải trên đối tượng, sau đó chọn **Delete**

**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.

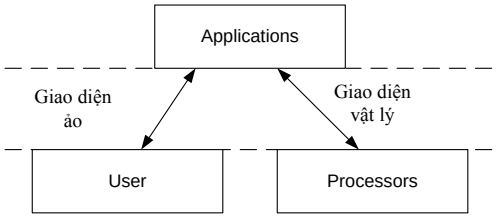
- MTĐT:

+ 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

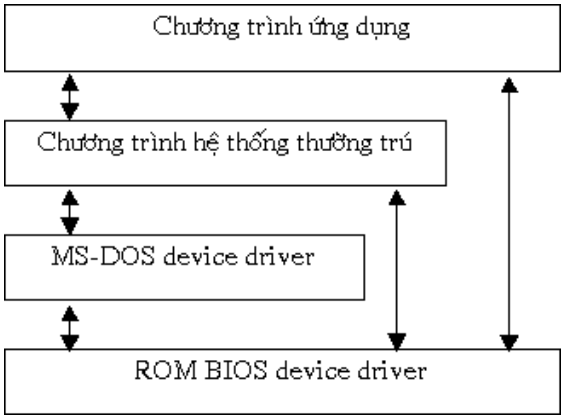
#### *1.3.2.1. Hệ thống đơn khối (monolithic systems)*

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.

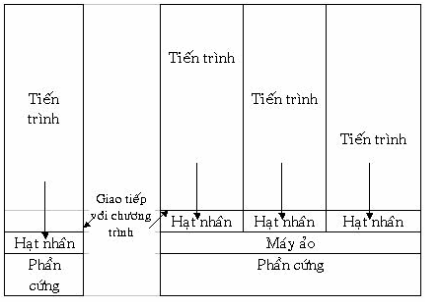
Trong cấu trúc này phần nhân của hệ thống trở thành bộ phận tổ chức giám sát máy ảo, phần này chịu trách nhiệm giao tiếp với phần cứng, chia sẻ tài nguyên hệ thống để tạo ra nhiều máy ảo, hoạt động độc lập với nhau, để cung cấp cho lớp trên.

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.

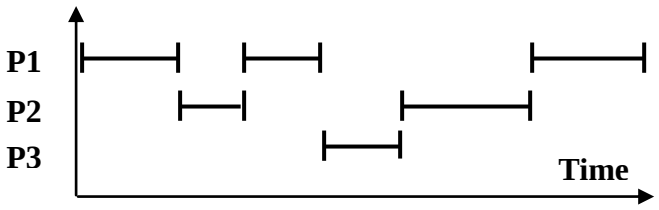
Tiến trình song song đồng mức: là các tiến trình hoạt động song song sử dụng chung tài nguyên theo nguyên tắc lần lượt, mỗi tiến trình sau một khoảng thời gian chiếm giữ tài nguyên phải tự động trả lại tài nguyên cho tiến trình kia.

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.

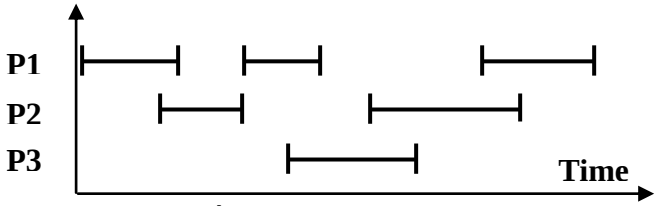
Các tiến trình song song xuất hiện trong các HĐH đa nhiệm đa chương, trên cả hệ thống uniprocessor và multiprocessor. Nhưng sự song song thực, chỉ có ở các hệ thống multiprocessor, trong hệ thống này mỗi processor chịu trách nhiệm thực hiện một tiến trình. Sự song song trên các hệ thống uniprocessor 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)

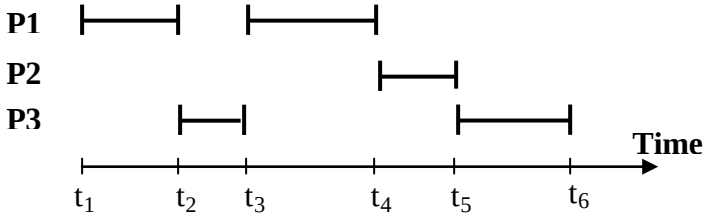
Đa số các hệ thống, có thể có nhiều chương trình hoạt động đồng thời (co-occurrence). Về nguyên tắc, để thực hiện được điều này thì hệ thống phải có nhiều processor, mỗi processor thực hiện một chương trình, nhưng mong muốn của HĐH cũng như người sử dụng là *thực hiện sự đa chương trên các hệ thống chỉ 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.

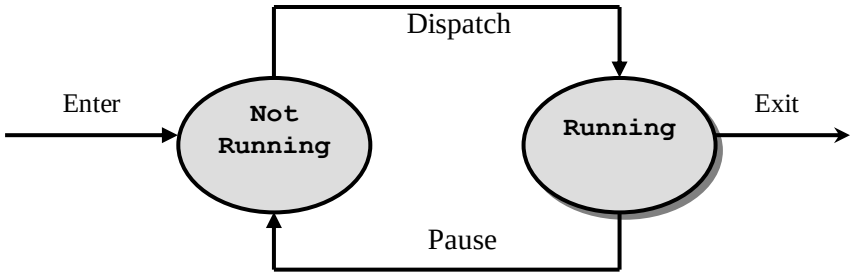
### 2.1.4. Các trạng thái tiến trình

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

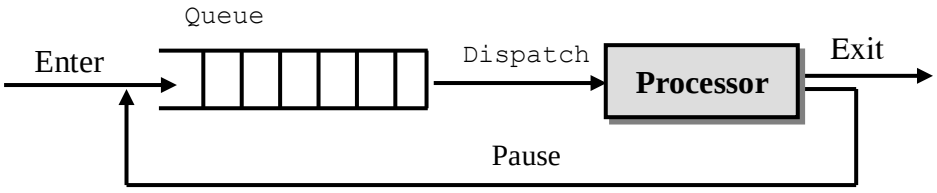
*2.1.4.1. Tiến trình hai trạng thái*

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*

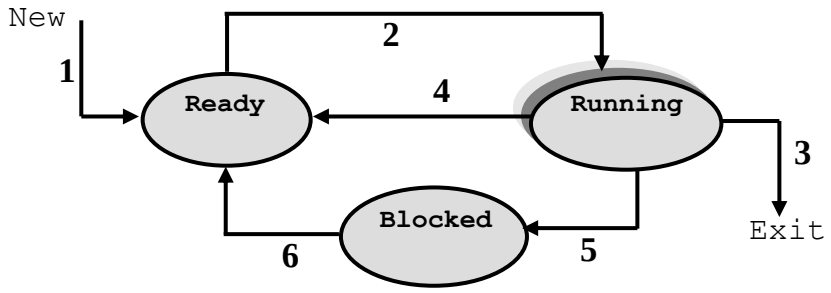
Đa số hệ điều hành đều cho phép tiến trình tồn tại ở một trong 3 trạng thái, đó là: ready, running, blocked:

- 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.

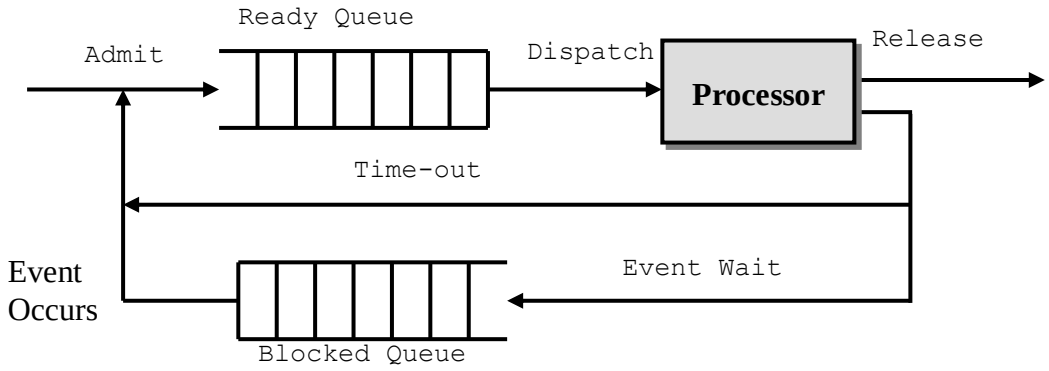
2. (Dispatch) Tiến trình được cấp processor để bắt đầu thực hiện/ xử lý.

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.

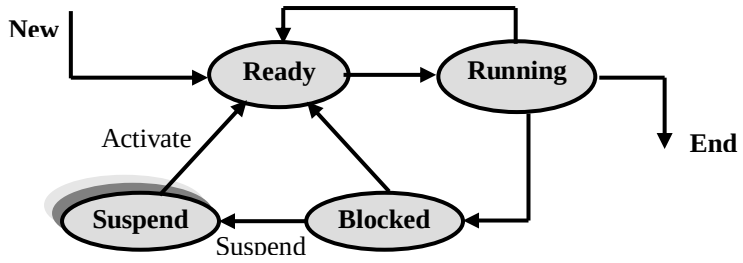
Tại một thời điểm xác định trong hệ thống có thể có nhiều tiến trình đang ở trạng thái Ready hoặc Blocked nhưng chỉ có một tiến trình ở trạng thái Running. Các tiến trình ở trạng thái Ready và Blocked được chứa trong các hàng đợi (Queue) riêng.

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

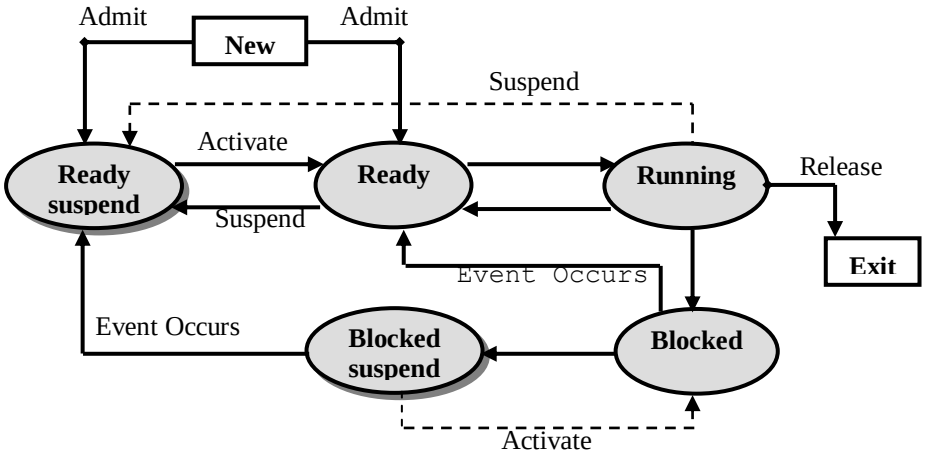
Trạng thái Suspend là trạng thái của một tiến trình khi nó đang được lưu trữ trên bộ nhớ phụ. Đó là các tiến trình đang ở trong trạng thái blocked và/hoặc ready bị HĐH chuyển ra đĩa để thu hồi lại không gian nhớ đã cấp cho tiến trình hoặc thu hồi lại tài nguyên đã cấp cho tiến trình để cấp cho một tiến 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, đế 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 $\lambda$ là số lượng trung bình các tiến trình đến hàng đợi, $\mu$ là tốc độ phục vụ trung bình thì hệ số đánh giá khả năng đáp ứng tải của hệ thống là $\rho$ 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).

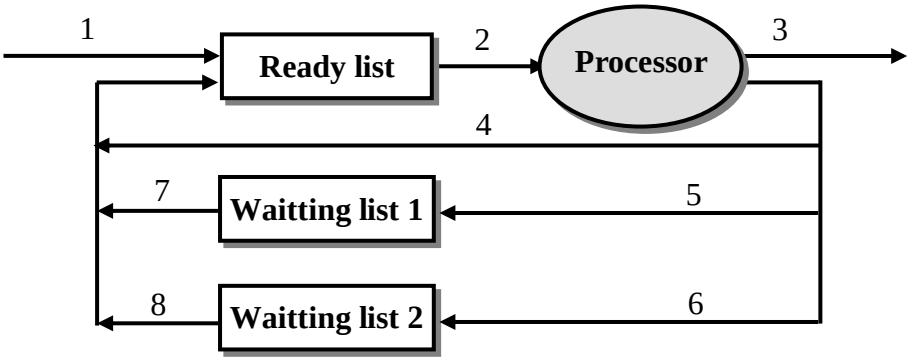
- Điều phối độc quyền: Khi có được processor tiến trình toàn quyền sử dụng processor cho đến khi tiến trình kết thúc xử lý hoặc tiến trình tự động trả lại processor cho hệ thống. Các quyết định điều phối xảy ra khi: Tiến trình chuyển trạng thái từ Running sang Blocked hoặc khi tiến trình kết thúc.
- 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:

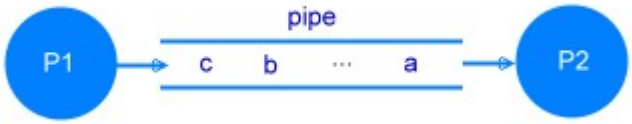
- Bỏ qua tín hiệu
- Xử lý tín hiệu theo kiểu mặc định
- Tiếp nhận tín hiệu và xử lý theo cách đặc biệt của tiến trình.

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!

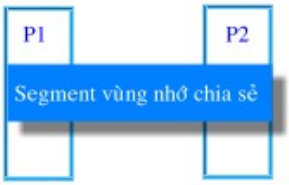
Cơ chế này cho phép truyền dữ liệu với cách thức không cấu trúc.

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.

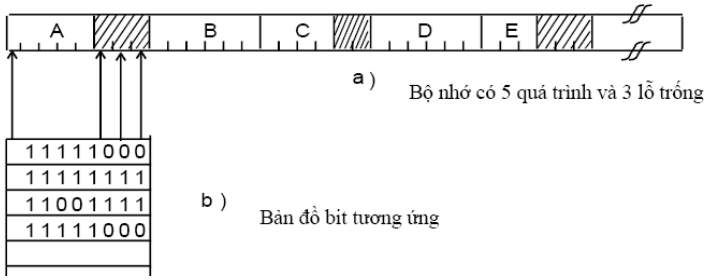
### 3.2.2. Kỹ thuật phân vùng động (Dynamic Partitioning)

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.

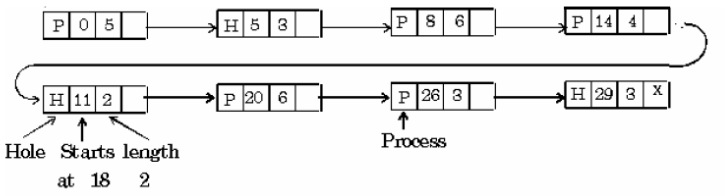
Khi phân vùng động, dần dần trong bộ nhớ hình thành nhiều không gian nhớ có kích thước nhỏ không đủ chứa các tiến trình nằm rải rác trên bộ nhớ chính, hiện tượng này được gọi là hiện tượng phân mảnh bên ngoài (external fragmentation). Để chống lại sự lãng phí bộ nhớ do phân mảnh, HĐH phải thực hiện việc 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.

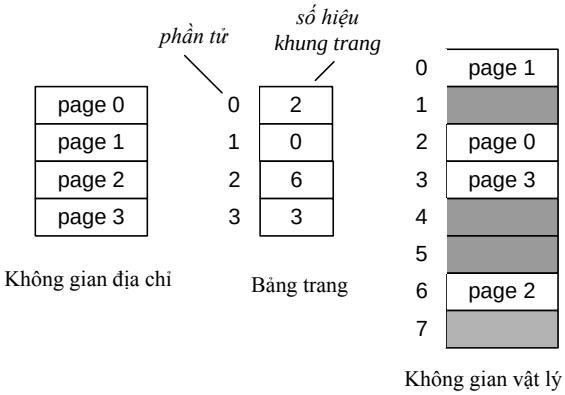
**Chú ý:** Một nhược điểm lớn của các kỹ thuật ở trên là dẫn đến hiện tượng phân mảnh bộ nhớ gây lãng phí bộ nhớ nên hiệu quả sử dụng bộ nhớ kém. Để khắc phục HĐH sử dụng các kỹ thuật phân trang hoặc phân đoạn bộ nhớ.

### 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

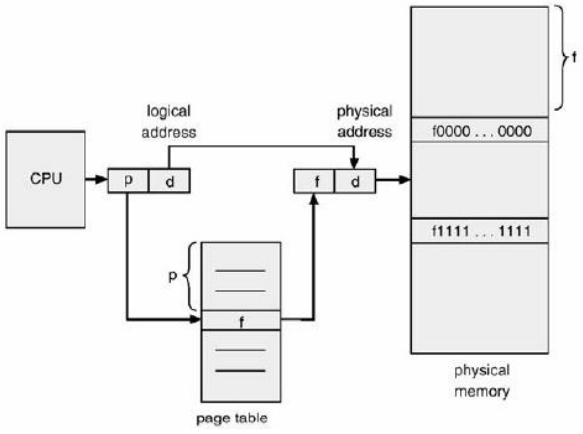
thể tồn tại các trang của nhiều tiến trình khác nhau. Khi một tiến trình bị swap-out thì các khung trang mà tiến trình này chiếm giữ sẽ được giải phóng để HĐH có thể nạp các trang tiến trình khác.




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.

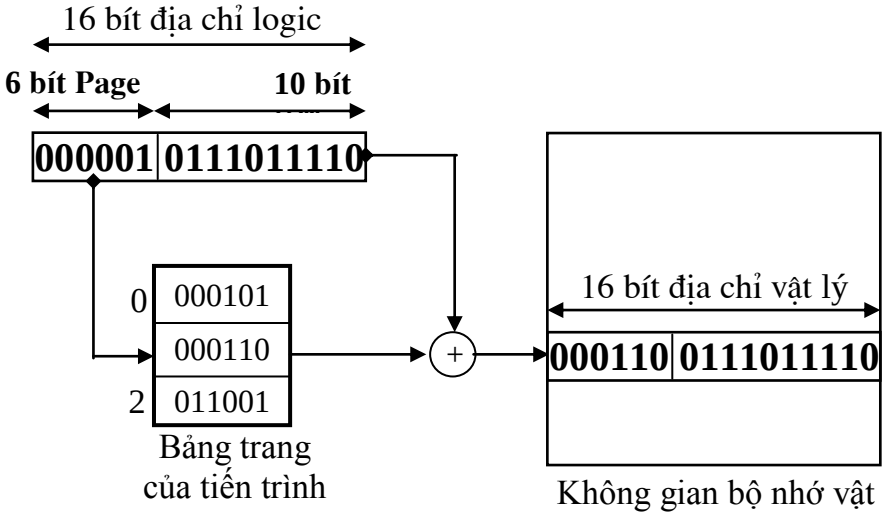
Trong kỹ thuật phân trang này khi cần truy xuất bộ nhớ BXL phải phát ra một địa chỉ logic gồm 2 thành phần: Số hiệu trang (Page): cho biết số hiệu trang tương ứng cần truy xuất. Địa chỉ tương đối trong trang (Offset): giá trị này sẽ được kết hợp với địa chỉ bắt đầu của trang để xác định địa chỉ vật lý 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:

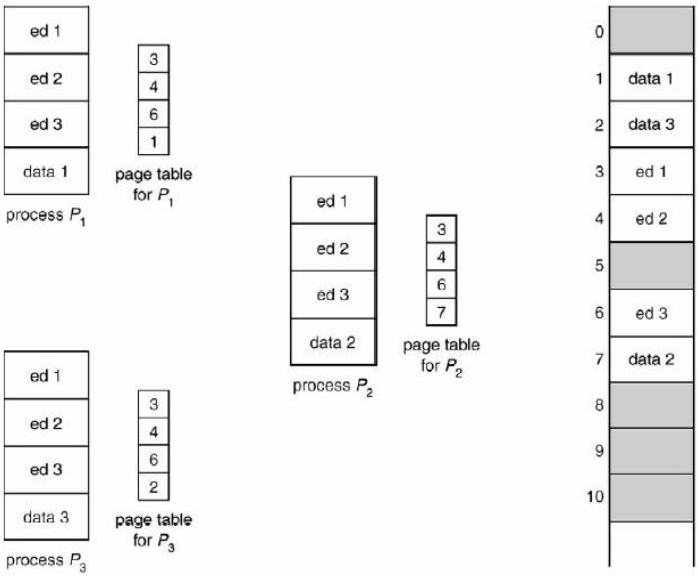
- Trích ra m-n bít trái nhất (thấp nhất) của địa chỉ logic để xác định số hiệu trang cần truy xuất.
- Sử dụng số hiệu trang ở trên để chỉ đến phần tử tương ứng trong bảng trang của tiến trình, để xác định khung trang tương ứng, ví dụ là k.
- Địa chỉ vật lý bắt đầu của khung trang là k x $2^n$ , và địa 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 = 0001100111011110.

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)

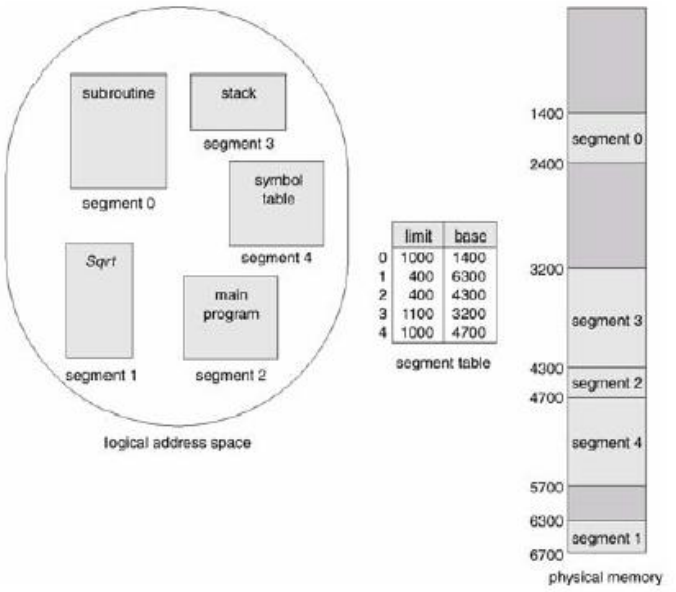
Không gian địa chỉ logic của các tiến trình kể cả các dữ liệu liên quan là tập hợp các đoạn khác nhau và không nhất thiết có kích thước bằng nhau, thông thường mỗi thành phần của một chương trình/tiến trình như: code, data, stack, subprogram,.. là một đoạn.

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

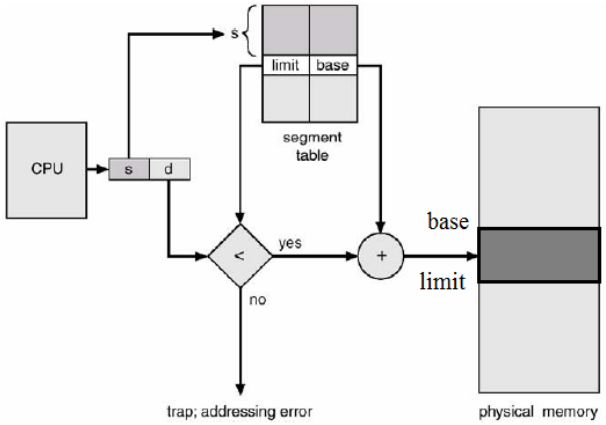
đoạn có thể được chứa trong các thanh ghi nếu có kích thước nhỏ, nếu kích thước bảng phân đoạn lớn thì nó được chứa trong BN chính, khi đó HĐH sẽ dùng một thanh ghi để lưu trữ địa chỉ bắt đầu nơi lưu trữ bảng phân đoạn, thanh ghi này được gọi là thanh ghi STBR: Segment table base register. Ngoài ra vì 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.



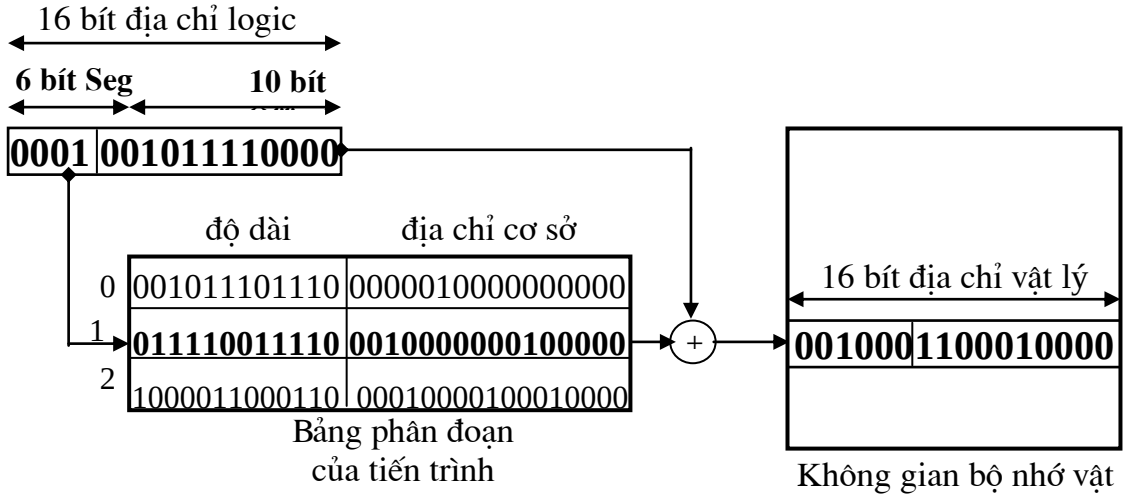

Trong kỹ thuật này địa chỉ logic mà BXL sử dụng gồm 2 thành phần: Số hiệu đoạn (segment): cho biết số hiệu đoạn tương ứng cần truy xuất và địa chỉ tương đối trong đoạn (Offset): giá trị này sẽ được kết hợp với địa chỉ bắt đầu của đoạn để xác định địa chỉ vật lý của ô nhớ cần truy xuất. Việc chuyển đổi 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.

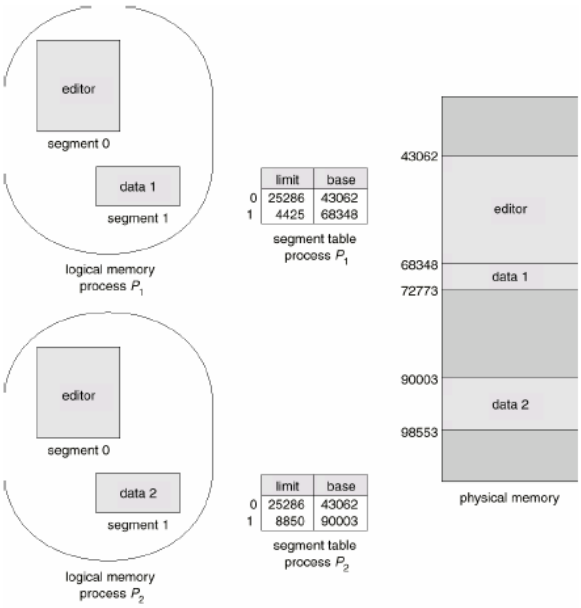



Trong ví dụ sau đây, ta có địa chỉ logic là: 0001001011110000, với số hiệu segment là 1, offset là 752, giả định segment này thường trú trong bộ nhớ chính tại địa chỉ vật lý là 0010000000100000, thì địa chỉ vật lý tương ứng với địa chỉ logic ở trên là: 0010000000100000 + 001011110000 = 0010001100010000.




**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:

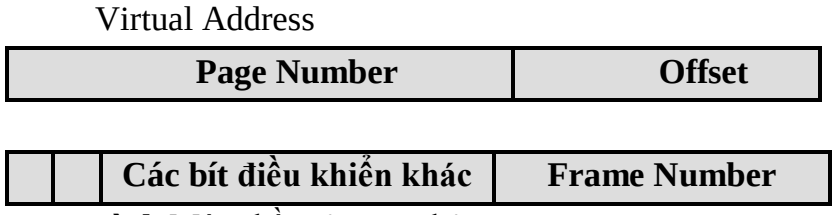
- Phân trang theo yêu cầu: phân trang kết hợp với swap.
- Phân đoạn theo yêu cầu: phân đoạn kết hợp với swap.

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:




**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

Nguyễn Hải Châu
Khoa Công nghệ thông tin
Trường Đại học Công nghệ

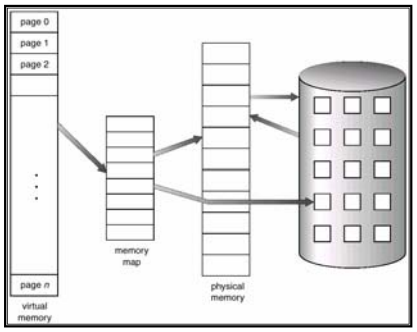
# Bộ nhớ ảo (Virtual Memory)

Yêu cầu phân trang
Tạo tiến trình
Thay thế trang
Cấp phát frame
Thrashing

## 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)

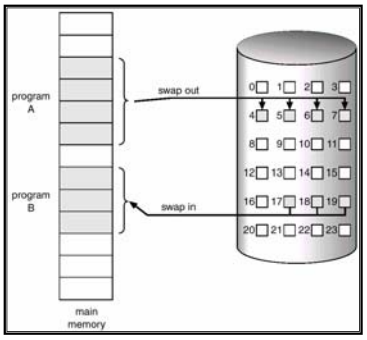
## Minh họa bộ nhớ ảo có dung lượng lớn hơn bộ nhớ vật lý




## Yêu cầu trang (demand paging)

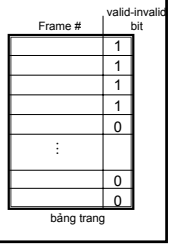
- Chỉ đưa một trang vào bộ nhớ khi cần thiết
  - Giảm thao tác các vào ra
  - Tiết kiệm bộ nhớ
  - Đáp ứng nhanh
  - Tăng được số người sử dụng (tiến trình)
- Khi cần một trang ⇒ tham chiếu đến nó
  - Tham chiếu lỗi ⇒ Hủy bỏ
  - Không nằm trong bộ nhớ ⇒ Đưa trang vào bộ nhớ

## Chuyển một trang (trong bộ nhớ) ra vùng đĩa liên tục

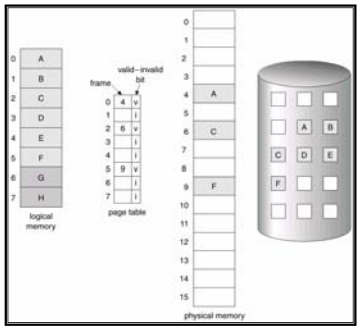


## Valid-Invalid Bit

- 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)



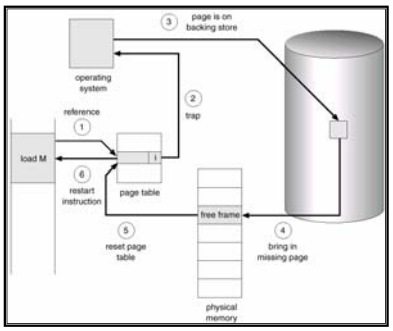

## Bảng trang với một số trang không nằm trong bộ nhớ




## 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)?

## Các bước xử lý page-fault




## 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

## 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)

## 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)

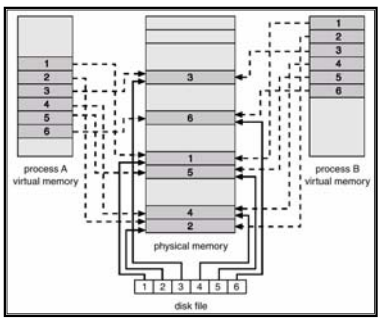
## Copy-on-Write

- 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

## 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

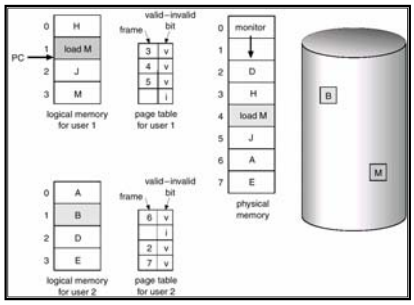
## Ví dụ file ánh xạ bộ nhớ




## 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

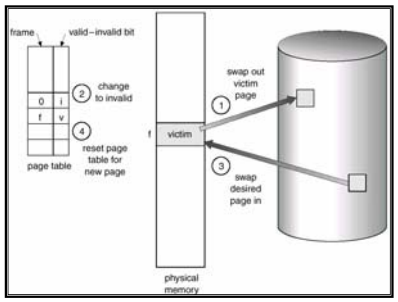
## Ví dụ: Yêu cầu thay thế trang



## 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

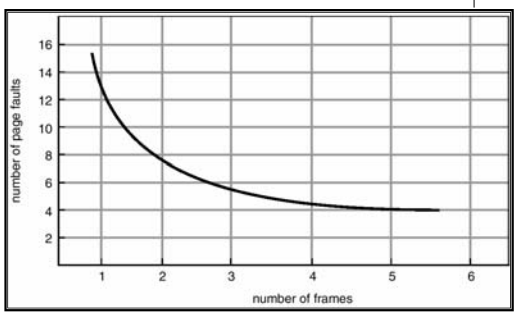
## Thay thế trang




## 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.

## Đồ thị page-fault




## 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

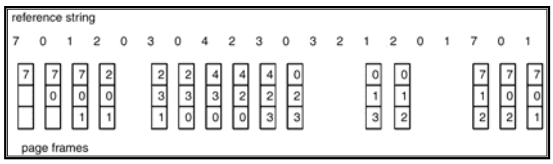
| | | | | |
|---|---|---|---|---|
| 1 | 1 | | 4 | 5 |
| 2 | 2 | | 1 | 3 |
| 3 | 3 | | 2 | 4 |

- Thay thế FIFO – Belady's anomaly
  - Có nhiều frame ⇒ ít page-fault

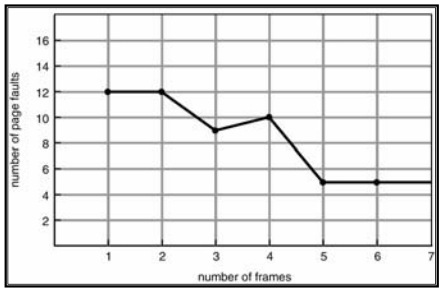
10 page faults

| | | | | |
|---|---|---|---|---|
| 1 | 1 | | 5 | 4 |
| 2 | 2 | | 1 | 5 |
| 3 | 3 | | 2 | |
| 4 | 4 | | 3 | |

## Thay thế trang FIFO

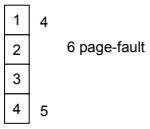


## FIFO Illustrating Belady's Anamoly

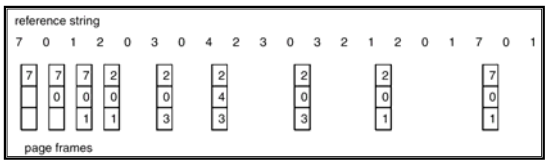



## Thuật toán tối ưu

- 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




## Thay thế trang tối ưu

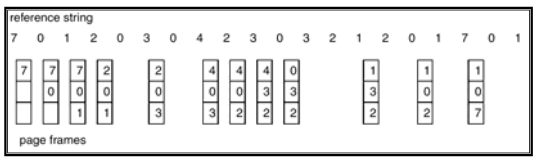



## Thuật toán LRU (Least Recently Used )

- 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

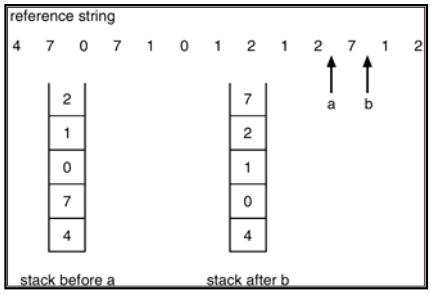
## Thay thế trang LRU




## 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

## Sử dụng ngăn xếp để ghi lại trang vừa mới được sử dụng



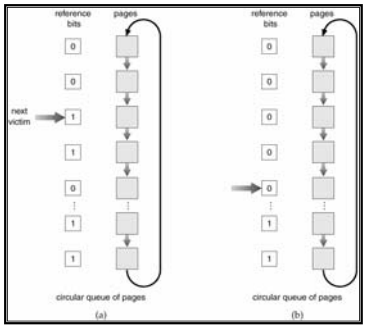

## Thuật toán LRU xấp xỉ

- 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

## 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

## Thuật toán second-chance




## 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

## 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

## 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 $p_i = \frac{s_i}{S} \times m$

$m = 64$
$s_i = 10$
$s_2 = 127$
$a_1 = \frac{10}{137} \times 64 \approx 5$
$a_2 = \frac{127}{137} \times 64 \approx 59$

## 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

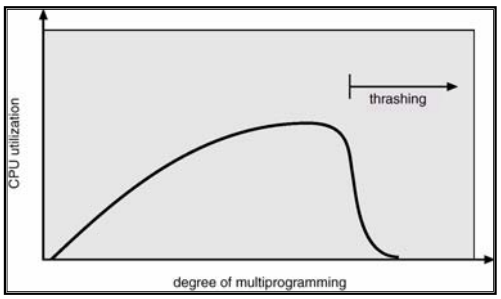
## Cấp phát tổng thể và cục bộ

- 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

## 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

## Minh họa thrashing




## 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

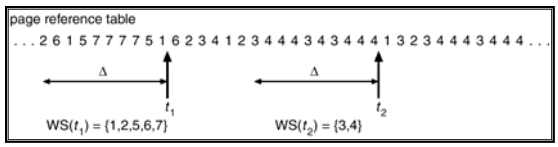
## Mô hình working-set

- Sử dụng tham số $\Delta$ là tổng số lần tham chiếu trang gần nhất
- Tập các trang sử dụng trong $\Delta$ 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 $\Delta$
  - Nếu $\Delta$ quá nhỏ: Không phản ánh đúng tính cục bộ
  - Nếu $\Delta$ quá lớn: vượt qua tính cục bộ
  - Nếu $\Delta = \infty$: $WSS_i$ tập toàn bộ các trang trong quá trình thực hiện tiến trình

## 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

## Mô hình working-set




## Các tệp ánh xạ bộ nhớ

- Sinh viên tự tìm hiểu trong giáo trình từ trang 348 đến trang 353

## 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