Chương I:
TỔNG QUAN VỀ HĐH

NỘI DUNG:

1.1 Khái niệm về HĐH

1.2 Phân Loại HĐH

1.3 Giới thiệu về cấu trúc của HĐH

1.4 Tìm hiểu về lịch sử phát triển của HĐH

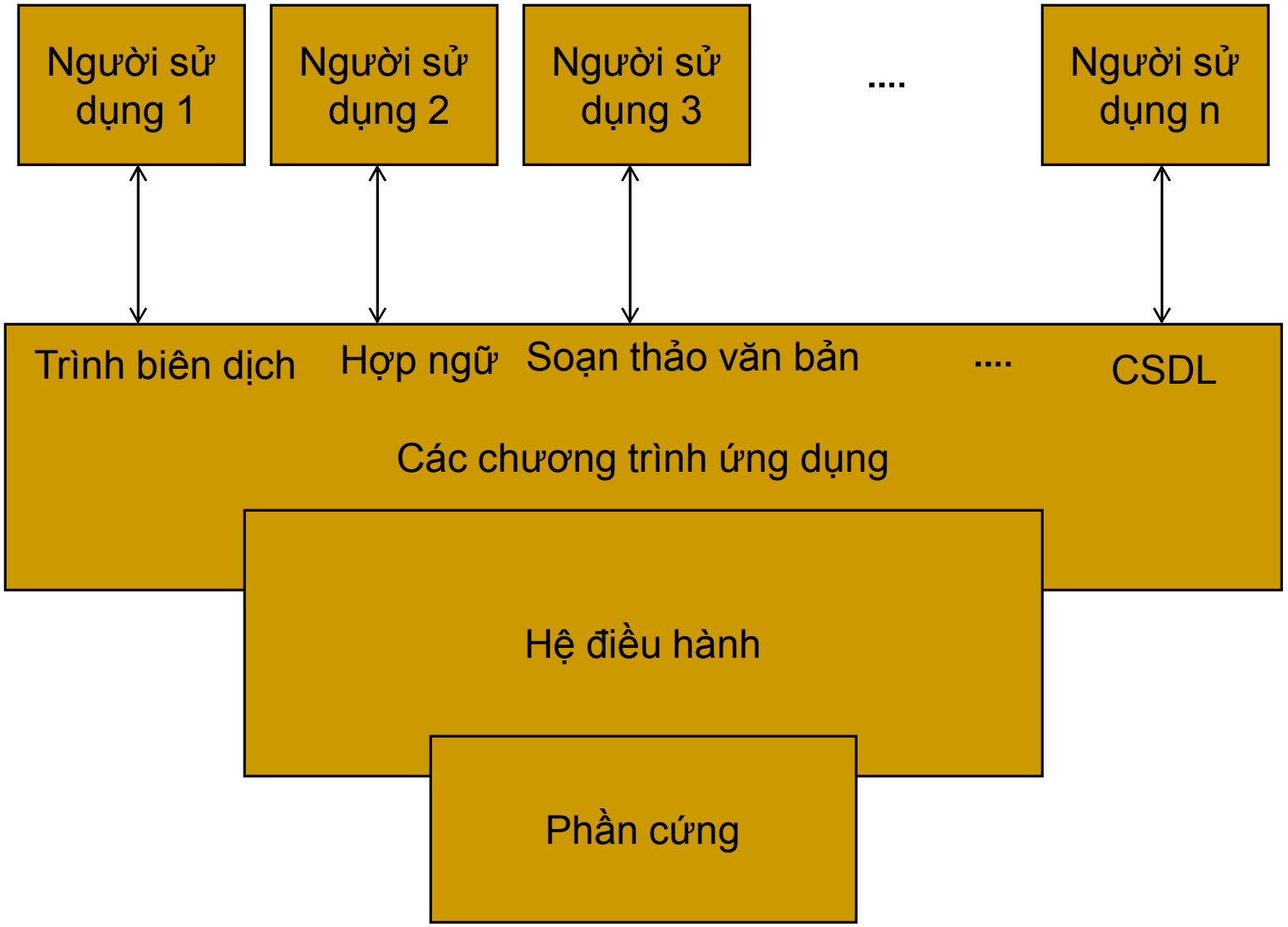
1.5 Giới thiệu một số HĐH phổ biến hiện nay

1.1 KHÁI NIỆM VỀ HĐH

- Hệ điều hành là một chương trình hay một hệ chương trình phần mềm máy tính, hoạt động ở lớp trung gian giữa người sử dụng và phần cứng máy tính
- Mục tiêu của HĐH là cung cấp môi trường để người sử dụng:
  - Thực thi dễ dàng các chương trình
  - Sử dụng máy tính trở nên dễ dàng, khai thác phần cứng máy tính một cách hiệu quả

1.1 KHÁI NiỆM VỀ HĐH(tt)

- HĐH là một bộ phận quan trọng của hệ thống máy tính. Một hệ thống máy tính bao gồm 4 phần:
  - Phần cứng: CPU; Bộ nhớ; Các thiết bị xuất/nhập
  - Các chương trình ứng dụng
  - Hệ điều hành
  - Đối tượng sử dụng: Người, thiết bị hoặc máy tính khác



4 Thành phần của hệ thống máy tính

1.2 PHÂN LOẠI HĐH

- Hệ thống sử lý theo lô đơn giản
- Hệ thống sử lý theo lô đa chương
- Hệ thống chia sẻ thời gian
- Hệ thống song song
- Hệ thống phân tán
- Hệ thống xử lý thời gia thực
- V.v.

HỆ THỐNG XỬ LÝ THEO LÔ ĐƠN GiẢN

- Các tác vụ được đưa vào hàng đợt
- Thực hiện các tác vụ lần lượt theo những chỉ thị đã được xác định trước
- Tác vụ tiếp theo tự động được thực hiện khi tác vụ trước kết thúc 1 cách tự động
- Có bộ giám sát thường trực để giám sát việc thực hiện của các tác vụ trong hệ thống
- → Processor rơi vào trạng thái chờ khi hệ thống truy xuất thiết bị vào ra

HỆ THỐNG XỬ LÝ THEO LÔ ĐA CHƯƠNG

- Thực hiện được nhiều tác vụ đồng thời
- HĐH nạp 1 phần code và data của tác vụ vào bộ nhớ
- Khi có tác vụ đang sử dụng Processor thực hiện truy xuất thiết bị vào ra thì Processor sẽ được chuyển thực hiện tác vụ khác
- Cần có cơ chế lập lịch cho Processor

HỆ THỐNG CHIA SẺ THỜI GIAN

- Các tác vụ, tiến trình được sử dụng Processor luân phiên nhau theo lịch phân chia thời gian sử dụng Processor đã được lập (t rất nhỏ)
- Cung cấp cho mỗi người sử dụng 1 phần nhỏ trong máy tính chia sẻ ->Người sử dụng có thể yêu cầu máy tính thực hiện đồng thời nhiều công việc
- Có cơ chế quản trị vào bảo vệ bộ nhớ, sử dụng bộ nhớ ảo

HỆ THỐNG SONG SONG

- Có nhiều Processor trong cùng một hệ thống máy tính
- Các Processor cùng chia sẻ đường truyền dữ liệu, đồng hồ xung, bộ nhớ và các thiết bị ngoại vi
- Có 2 loại HĐH đa Processor:
  - Đa xử lý đối xứng (Symmetric multiprocessing-SMP)
  - Đa xử lý bất đối xứng (Asymmetric multiprocessing-ASMP)

HỆ THỐNG SONG SONG(tt)

- Đa xử lý đối xứng:
  - Mỗi Processor chạy độc lập trên một bản sao HĐH như nhau
  - Cho phép nhiều tiến trình chạy đồng thời trên một hệ thống
- Đa xử lý bất đối xứng:
  - Mỗi Processor được giao một nhiệm vụ riêng biệt
  - Có một hoặc 2 Processor chủ làm nhiệm vụ lập lịch, xác định công việc cho các Processor thành viên

HỆ THỐNG PHÂN TÁN

- Phân tán sự tính toán trên các bộ xử lý vật lý
- Mỗi bộ xử lý có bộ nhớ cục bộ riêng
- Các bộ xử lý thông tin với nhau thông qua các đường truyền thông tốc độ cao
- Có 2 dạng hệ thống: Client/Server và Peer-to-Peer

HỆ THỐNG XỬ LÝ THỜI GIAN THỰC

- Có khả năng cho kết quả tức thời, chính xác sau mỗi tác vụ
- Tác vụ cần thực hiện không đưa vào hàng đợi mà sử lý tức thời và trả lại ngay kết quả chính xác trong khoảng thời gian bị thúc ép nhanh nhất

1.3 CẤU TRÚC CỦA HĐH

1.3.1 CÁC THÀNH PHẦN CỦA HĐH

- Quản lý tiến trình
- Quản lý bộ nhớ chính
- Quản lý xuất/nhập
- Quản lý bộ nhớ phụ
- Quản lý tập tin
- Thông dịch lệnh
- Bảo vệ hệ thống

NHIỆM VỤ CỦA THÀNH PHẦN QUẢN LÝ TIẾN TRÌNH

- Tạo lập và hủy bỏ tiến trình
- Tạm dừng và kích hoạt lại tiến trình đã bị tạm dừng
- Tạo cơ chế thông tin liên lạc giữa các tiến trình
- Tạo cơ chế đồng bộ hóa giữa các tiến trình

NHIỆM VỤ CỦA THÀNH PHẦN QuẢN LÝ BỘ NHỚ CHÍNH

- Cấp phát, thu hồi vùng nhớ
- Ghi nhận trạng thái bộ nhớ chính
- Bảo về bộ nhớ
- Quyết định tiến trình nào được nạp vào bộ nhớ

NHIỆM VỤ CỦA THÀNH PHẦN QUẢN LÝ XUẤT/NHẬP

- Làm cho các thao tác trao đổi thông tin trên các thiết bị nhập/xuất được trong suốt với người sử dụng
- Một hệ thống nhập/xuất bao gồm:
  - Hệ thống buffer caching.
  - Bộ giao tiếp điều khiển thiết bị.
  - Bộ điều khiển cho các thiết bị đặc thù.

NHIỆM VỤ CỦA THÀNH PHẦN QUẢN LÝ BỘ NHỚ PHỤ

- 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

NHIỆM VỤ CỦA THÀNH PHẦN QuẢN LÝ TẬP TIN

- Tạo/xóa tập tin, thư mục
- Bảo vệ tập tin khi có truy xuất đồng thời
- Cung cấp các thao tác xử lý và bảo vệ tập tin, thư mục
- Tạo cơ chế truy xuất tập tin thông qua tên tập tin,…

NHIỆM VỤ CỦA THÀNH PHẦN THÔNG DỊCH LỆNH

- Đóng vai trò giao tiếp giữa HĐH và người sử dụng
- Một số HĐH thành phần này nằm trong nhân của nó, một số HĐH khác thiết kế dưới dạng 1 chương trình đặc biệt

NHIỆM VỤ CỦA THÀNH PHẦN BẢO VỆ HỆ THỐNG

- Kiểm soát quá trình truy xuất của chương trình, tiến trình, hoặc người sử dụng với tài nguyên của hệ thống

1.3.2 CẤU TRÚC CỦA HĐH

a. HỆ THỐNG ĐƠN KHỐI:

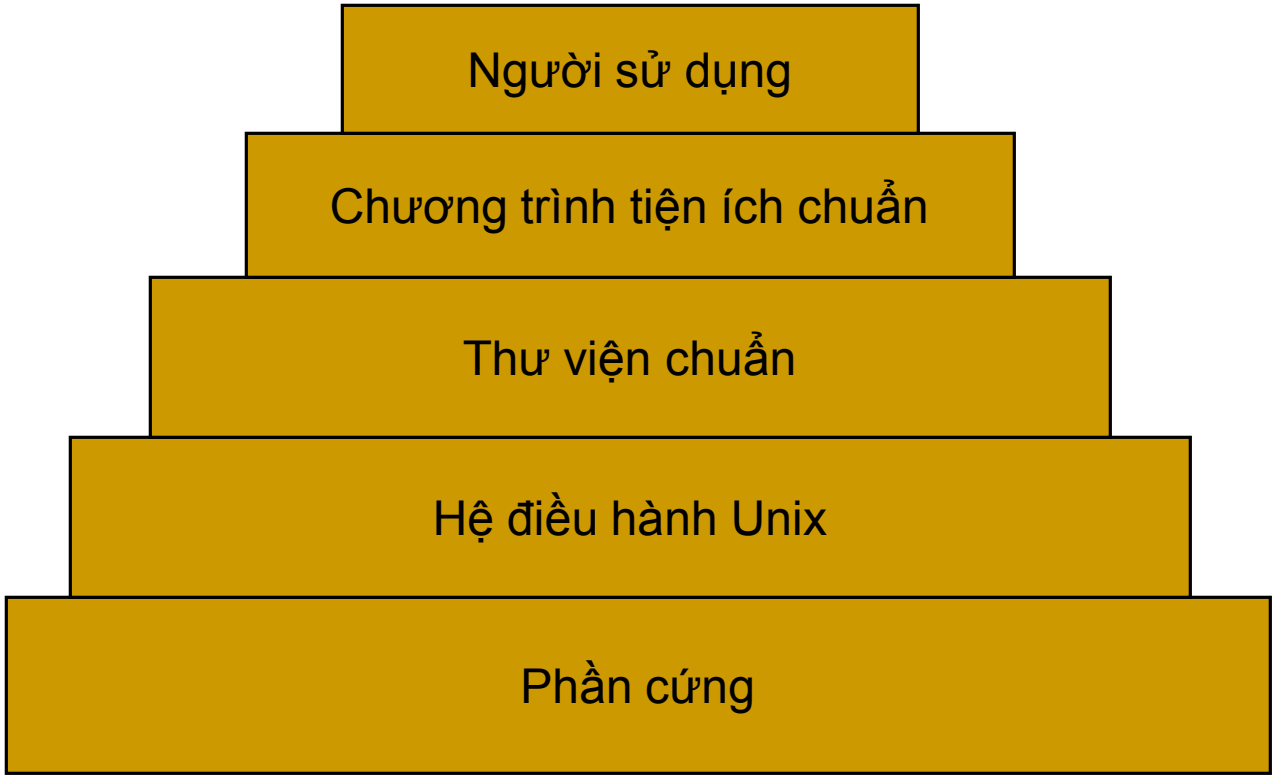
- Là một tập hợp các thủ tục, mỗi thủ tục có thể gọi thực hiện một thủ tục khác bất kỳ lúc nào cần thiết
- MS-DOS là một hệ điều hành có cấu trúc đơn giản, nó cung cấp những chức năng lớn nhất cho hệ thống tối thiểu

1.3.2 CẤU TRÚC CỦA HĐH(tt)

b. HỆ THỐNG PHÂN LỚP:

- Hệ thống được chia thành một số lớp
- Mỗi lớp được xây dựng dựa trên một lớp bên dưới
- Lớp dưới cùng là phần cứng, lớp trên cùng là giao diện với người sử dụng

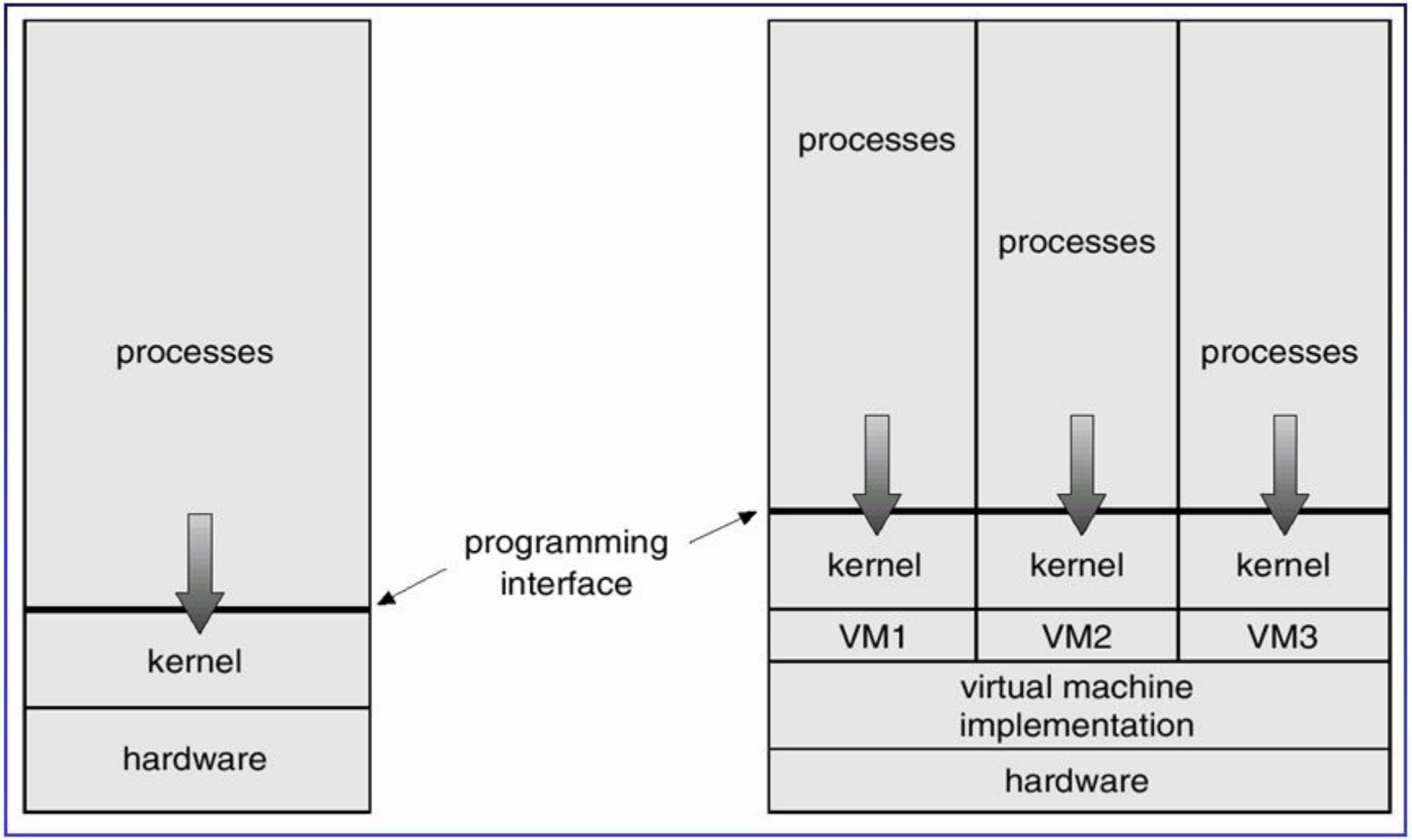
Hệ thống phân lớp của UNIX



1.3.2 CẤU TRÚC CỦA HĐH(tt)

c. MÁY ẢO (Virtual Machine)

- Là bản sao chính xác các đặc tính phần cứng của máy tính thực. Được cung cấp phần cứng và kernel của HĐH như máy thật
- Tài nguyên máy tính vật lý được chia sẻ để tạo ra các máy ảo
- Mỗi tiến trình được thực hiện trên một máy ảo độc lập


processes
kernel
hardware
programming
interface
processes
processes
processes
kernel
kernel
kernel
VM1
VM2
VM3
virtual machine
implementation
hardware

1.3.2 CẤU TRÚC CỦA HĐH(tt)

d. MÔ HÌNH Client/Server:

- Mô hình này các tiến trình được chia thành 2 loại
  - Tiến trình Client: Là các tiến trình bên ngoài hay tiến trình của chương trình người sử dụng
  - Tiến trình Server: Là các tiến trình của HĐH
- Khi cần thực hiện 1 chức năng của hệ thống tiến trình client gửi yêu cầu đến tiến trình server tương ứng, tiến trình server xử lý và trả về cho client

1.4 LỊCH SỬ PHÁT TRIỂN CỦA HĐH

1. Thế hệ 1(1945-1955)

- Máy tính dùng ống chân không ra đời
- Vận hành máy tính cần 1 nhóm người: Thiết kế, xây dựng chương trình, thao tác, quản lý,…
- Chưa có khái niệm về ngôn ngữ lập trình và HĐH

1.4 LỊCH SỬ PHÁT TRIỂN CỦA HĐH(tt)

2. Thế hệ 2(1955-1965)

- Máy tính dùng bán dẫn ra đời
- Bộ phận sử dụng máy tính được phân chia rõ ràng: người thiết kế, người xây dựng, người lập trình, người vận hành,…
- Ngôn ngữ Assembly và Foxtran ra đời
- Chương trình được viết trên phiếu đục lỗ
- Hệ thống xử lý theo lô ra đời, các yêu cầu thực hiện được lưu trên băng từ, hệ thống đọc và thi hành lần lược
- Hệ thống xử lý theo lô hoạt động dưới sự điều khiển của 1 chương trình đặc biệt

1.4 LỊCH SỬ PHÁT TRIỂN CỦA HĐH(tt)

3. Thế hệ 3(1965-1980)

- Máy tính được sử dụng rộng rãi
- Ra đời máy tính IBM 360 sử dụng mạch IC
- Thiết bị ngoại vi dùng cho máy tính xuất hiện ngày càng nhiều
- Các thao tác điều khiển máy tính ngày càng phức tạp
- HĐH ra đời nhằm điều phối, kiểm soát hoạt động của hệ thống và giải quyết các yêu cầu tranh chấp thiết bị
- Bắt đầu có khái niệm đa chương, chia sẻ thời gian thực và kỹ thuật spool
- Xuất hiện HĐH Multics và Unix

1.4 LỊCH SỬ PHÁT TRIỂN CỦA HĐH(tt)

4. Thế hệ 4(1980->)

- Máy tính cá nhân ra đời
- HĐH MS-DOS ra đời gắn liền với máy tính IBM PC
- Ra đời và phát triển nhiều HĐH gắn liền với sự phát triển của phần cứng máy tính

BÀI TẬP

1. Hệ điều hành là :

a. Một chương trình

b. Một chương trình hay hệ chương trình

c. Một thiết bị

d. ROM-BIOS

2. Một hệ điều hành bao gồm:

a. Hệ thống quản lý tập tin, thiết bị I/O

b. Hệ thống quản lý tiến trình, bộ nhớ

c. a và b

CHƯƠNG II:
QUẢN LÝ TIẾN TRÌNH

1. TỔNG QUAN VỀ TiẾN TRÌNH

1.1 Tiến trình(process)?

- Tiến trình là một chương trình đang được thực thi, được sở hữu 1 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.

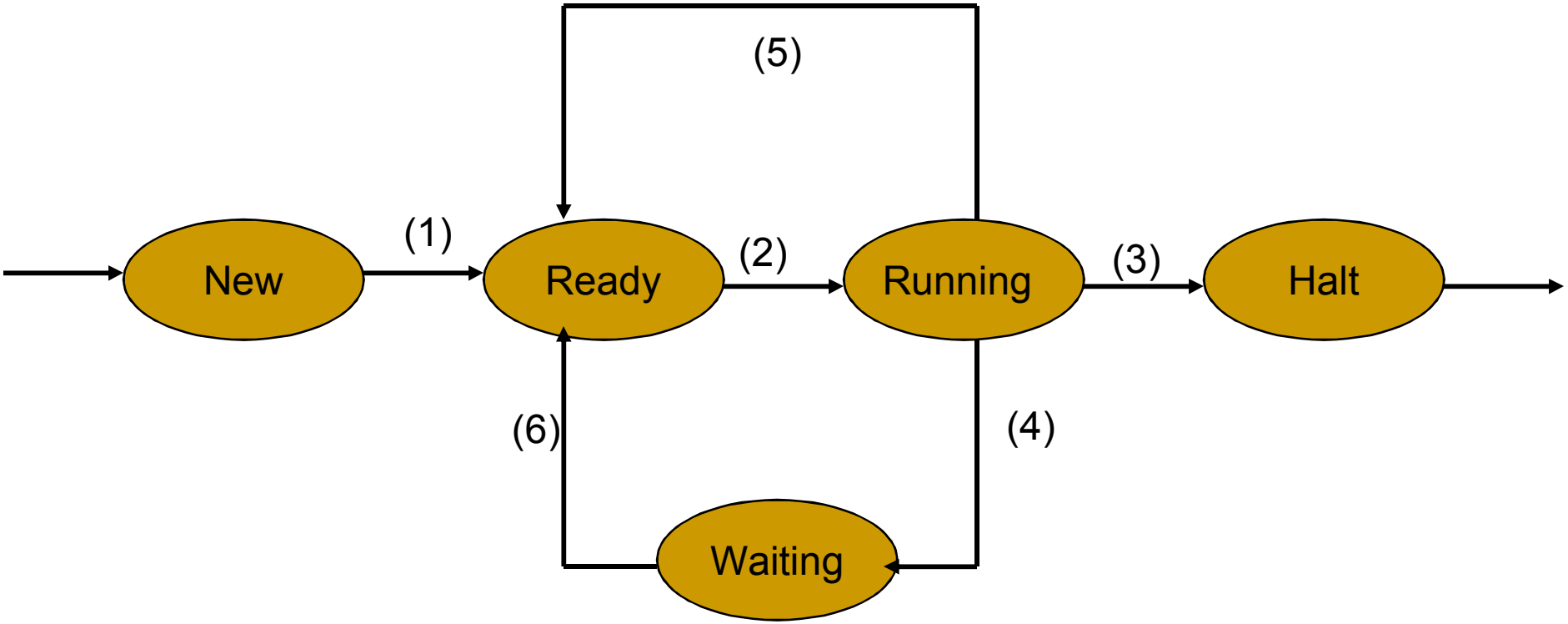
1.1 Tiến trình(process)?(tt)

- Tiến trình bao gồm 3 thành phần: Code, Data, Stack
  - Code: Thành phần câu lệnh thực hiện
  - Data: Thành phần dữ liệu
  - Stack: Thành phần lưu thông tin tạm thời
- Các câu lệnh trong code chỉ dùng data và stack riêng của mình ngoại trừ các vùng dùng chung
- Tiến trình được hệ thống phân biệt bằng số hiệu pid (proccess indentification)

1.2 Các trạng thái của tiến trình

- Trạng thái của tiến trình tại mỗi thời điểm được xác định bởi hoạt động hiện thời của nó:
  - New: tiến trình được tạo lập
  - Ready: tiến trình đã sẵn sàng, đang chờ cấp CPU
  - Running: tiến trình đang được xử lý
  - Waiting: tiến trình tạm dừng và chờ vì thiếu tài nguyên hay chờ 1 sự kiện nào đó
  - Halt: Tiến trình hoàn tất

Mô tả chuyển trạng thái của tiến trình



1.2 Các trạng thái của tiến trình(tt)

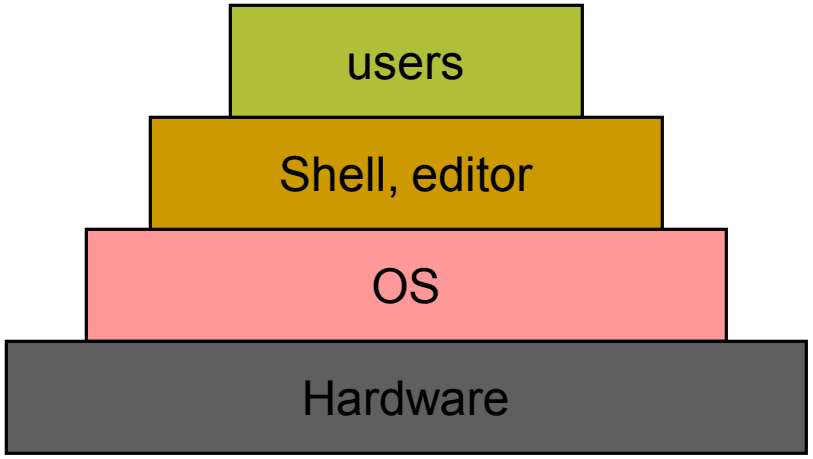
- Tại một thời điểm chỉ có 1 tiến trình ở trạng thái Running trên 1 bộ xử lý bất kỳ và có thể có nhiều tiến trình ở trạng thái Ready và Waiting

1.3 Chế độ xử lý của tiến trình

- Tiến trình của HĐH cần được bảo vệ khỏi sự xâm phạm của tiến trình khác
- Chế độ xử lý được chia thành 2 chế độ nhờ sự hỗ trợ của phần cứng: Đặc quyền và không đặc quyền
- Tiến trình của HĐH hoạt động trong chế độ đặc quyền và của người sử dụng hoạt động trong chế độ không đặc quyền

1.3 Chế độ xử lý của tiến trình(tt)

- Tập lệnh của CPU được chia thành 2 tập



Chế độ không đặc quyền

Chế độ đặc quyền

1.4 Các thao tác điều khển tiến trình

a. Khởi tạo tiến trình

- HĐH gán PID và đưa vào danh sách quản lý của hệ thống
- Cấp phát không gian bộ nhớ
- Khởi tạo các thông tin cần thiết cho khối điều khiển tiến trình: Các PID của p cha (nếu có), thông tin trạng thái, độ ưu tiên, ngữ cảnh của processor
- Cung cấp đầy đủ các tài nguyên (trừ processor)
- Đưa tiến trình vào danh sách p nào đó: ready list, suspend list, waiting list

1.4. Các thao tác điều khển tiến trình

b. Kết thúc tiến trình HĐH thực hiện các thao tác:

- Thu hồi tài nguyên đã cấp phát cho p
- Loại bỏ tiến trình ra khỏi danh sách quản lý của hệ thống
- Hủy bỏ khối điều khiển p

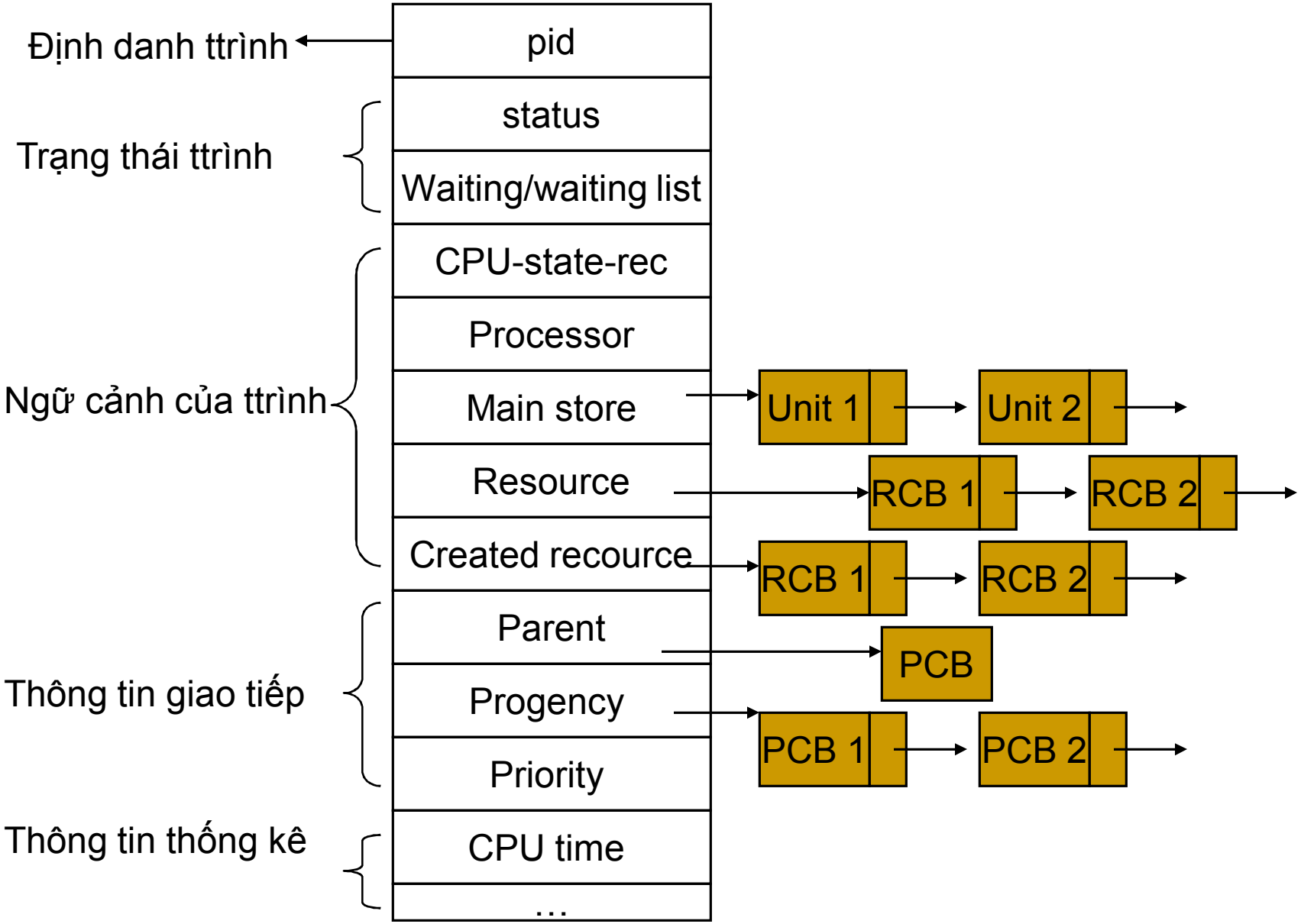
1.4. Các thao tác điều khển tiến trình

c. Thay đổi trạng thái của p HĐH thực hiện:

- Lưu ngữ cảnh của processor
- Cập nhật PCB (process control block) của tiến trình sao cho phù hợp với trạng thái của p
- Di chuyển PCB của p đến 1 hàng đợi thích hợp
- Chọn tiến trình khác để cho phép nó thực hiện
- Cập nhật PCB của p vừa thực hiện
- Cập nhật thông tin liên quan đến quản lý bộ nhớ
- Khôi phục lại ngữ cảnh của processor

1.5 Khối điều khiển tiến trình(process control block -PCB).

- Quản lý mọi hoạt động của tiến trình
- Cấu trúc dữ liệu của khối điều khiển bao gồm:
  - Định danh tiến trình
  - Trạng thái của tiến trình
  - Ngữ cảnh của tiến trình
  - Thông tin giao tiếp
  - Thông tin thống kê


Định danh ttrình
pid
Trạng thái ttrình
status
Waiting/waiting list
Ngữ cảnh của ttrình
CPU-state-rec
Processor
Main store
Unit 1
Unit 2
Resource
RCB 1
RCB 2
Created recource
RCB 1
RCB 2
Thông tin giao tiếp
Parent
PCB
Progency
PCB 1
PCB 2
Priority
Thông tin thống kê
CPU time
…

1.6 Tiểu trình(thread)

- Thông thường mỗi tiến trình có 1 không gian địa chỉ và 1 dòng xử lý
- Mong muốn có nhiều dòng xử lý cùng chia sẻ 1 không gian địa chỉ và các dòng xử lý hoạt động song song như các tiến trình độc lập
- Xuất hiện HĐH có cơ chế thực thi mới gọi là tiểu trình
- Như vậy, tiểu trình là:
  - 1 đơn vị xử lý cơ bản
  - Sở hữu 1 con trỏ lệnh, tập các thanh ghi, 1 vùng nhớ stack riêng
  - Có các trạng thái như tiến trình thật.

2. TÀI NGUYÊN GĂNG VÀ ĐOẠN GĂNG

2.1 Tài nguyên găng(Critical Resource)

- Tài nguyên găng?
  - Những tài nguyên được HĐH chia sẻ cho nhiều tiến trình hoạt động đồng thời dùng chung mà có nguy cơ tranh chấp giữa các tiến trình này khi sử dụng chúng
- 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

2.1 Tài nguyên găng(Critical Resource)

- Ví dụ: bài toán rút tiền ngân hàng từ tài khoản dùng chung

If (tài khoản – tiền rút >=0)

tài khoản:=tài khoản – tiền rút

Else

Thông báo lỗi

endif

2.2 Đoạn găng(Critical Section)

- Các đoạn code trong các chương trình dùng để truy cập đến tài nguyên găng được gọi là đoạn găng
- Để hạn chế lỗi có thể xảy ra do sử dụng tài nguyên găng, tại 1 thời điểm HĐH chỉ cho 1 tiến trình nằm trong đoạn găng
- HĐH có cơ chế điều độ tiến trình qua đoạn găng

2.3 Yêu cầu của công tác điều độ tiến trình qua đoạn găng

- Tại 1 thời điểm chỉ cho phép 1 tiến trình nằm trong đoạn găng, các tiến trình khác có nhu cầu vào đoạn găng phải chờ
- Tiến trình chờ ngoài đoạn găng không được ngăn cản các tiến trình khác vào đoạn găng
- Không có tiến trình nào phải chờ lâu để được vào đoạn găng
- Đánh thức các tiến trình trong hàng đợi để tạo điều kiện cho nó vào đoạn găng khi tài nguyên găng được giải phóng

2.4 Điều độ tiến trình qua đoạn găng

a. Giải pháp phần cứng

- Dùng cặp chỉ thị STI(setting interrupt) và CLI (clean interrupt)

Ví dụ:

```
Procedure P(i: integer)
    begin
        repeat
                CLI;
                        <đoạn găng của p>;
                STI;
                        <Đoạn không găng>;
        until .F.
    end;
```

- Dùng chỉ thị TSL(Test and set)

```
Function TestAnhSetLock(Var i:integer):boolean
Begin
   if i=0 then
        begin
                i:=1;
                TestAnhSetLock:=true
        end;
   else
        TestAnhSetLock:=false
End;
```

```
Procedure P(lock: integer);
    begin
        repeat
                while  (TestAnhSetLock(lock)) do;
                <đoạn găng của p>;
                lock:=0
                <Đoạn không găng>;
        until .F.
    end;
```

b. Giải pháp dùng biến khóa

- Dùng biến khóa chung

```
Procedure P(lock: integer);
    begin
        repeat
                while    lock=1 do;
                Lock=1
                <đoạn găng của p>;
                lock:=0
                <Đoạn không găng>;
        until .F.
    end;
```

- Dùng biến khóa riêng

```
Var lock1, lock2: byte;
begin
lock1:=0; lock2:=1
        p1: repeat
                while    lock2=1 do;
                Lock1:=1
                <đoạn găng của p>;
                lock1:=0
                <Đoạn không găng>;
            until .F.
        p2: repeat
                while    lock1=1 do;
                Lock2:=1
                <đoạn găng của p>;
                lock2:=0
                <Đoạn không găng>;
            until .F.

end
```

C. Giải pháp được hỗ trợ bởi HĐH và ngôn ngữ lập trình

- Dùng Semaphore(đèn báo)
  - Semaphore S là 1 biến nguyên, khởi gán bằng 1 giá trị không âm, là khả năng phục vụ của tài nguyên găng tương ứng với nó
  - Ứng với S có 1 hàng đợi F(s) lưu các tiến trình đang chờ trên S
  - Thao tác Down giảm S 1 đơn vị, Up tăng S 1 đơn vị
  - Mỗi tiến trình trước khi vào đoạn găng cần gọi Down để giảm S và kiểm tra nếu S>=0 thì được vào đoạn găng
  - Mỗi tiến trình khi ra khỏi đoạn găng phải gọi Up để tăng S lên 1 đơn vị và ktra nếu S <=0 thì đưa 1 tiến trình trong F(s) vào đoạn găng

```
Procedure Down(S);
Begin
   S:=S-1;
   If s<0 then
      Begin
         Status(p)=waiting;
         Enter(p,F(s));
      end
End;
```

```
Procedure Up(S);
Begin
   S:=S+1;
   If s<0 then
      Begin
         Exit(Q,F(S));
         Status(Q)=ready;
         Enter(Q,ready-list);
      end
End;
```

3. TẮC NGHẼN VÀ CHỐNG TẮC NGHẼN

3.1 Tắc nghẽn

- Sự xung đột về tài nguyên của các tiến trình hoạt động đồng thời trong hệ thống
- Tắc nghẽn thường xảy ra với xung đột tài nguyên không phân chia được, ít xảy ra với tài nguyên phân chia được

3.2 Điều kiện hình thành tắc nghẽn

- Sử dụng tài nguyên không thể chia sẻ
- Chiếm giữ và yêu cầu tài nguyên
- Không thu hồi tài nguyên từ tiến trình đang chiếm giữ chúng
- Đợi vòng tròn

3.3 Các mức phòng tránh tắc nghẽn

- Ngăn ngừa
- Dự báo và tránh tắc nghẽn
- Nhận biết và khắc phục

4. ĐIỀU PHỐI TIẾN TRÌNH

4.1 Mục tiêu điều phối

- Sự công bằng
- Tính hiệu quả
- Thời gian đáp ứng hợp lý
- Thời gian lưu lại trong hệ thống
- Thông lượng tối đa

4.2 Cơ chế điều phối

- Độc quyền: Tiến trình toàn quyền sử dụng processor cho đến khi kết thúc hoặc tự động trả lại
  - Quyết định điều phối khi tiến trình chuyển từ Running sang Waiting (blocked) hoặc kết thúc
- Không độc quyền: Tiến trình đang xử lý thì bị thu hồi processor để cấp cho tiến trình khác
  - Quyết định điều phối khi tiến trình chuyển từ Running sang Waiting (blocked) hoặc ready hoặc kết thúc hoặc từ Waiting sang ready

4.3 Các đặc điểm của tiến trình

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

4.4 Tổ chức điều phối

- HĐH sử dụng 2 loại danh sách để tổ chức lưu trữ các tiến trình:
  - Danh sách Ready: Chỉ tồn tại 1 danh sách này
  - Danh sách Waiting: Có thể tồn tại nhiều danh sách này

4.5 Các chiến lược điều phối

- Chiến lược FIFO: Tiến trình nào được đưa vào danh sách ready trước sẽ được cấp Processor trước
- Ví dụ

Tiến trình	Thời điểm vào	t/g xử lý
P1	0	24
P2	1	3
P3	2	3

Thời điểm cấp processor		
P1	P2	P3
0	24	27

Thời gian chờ:

P1: 0

P2: 23

P3: 25

4.5 Các chiến lược điều phối

- Chiến lược phân phối xoay vòng:
  - Tiến trình nào vào danh sách Ready trước được cấp processor trước
  - Mỗi tiến trình chỉ được sử dụng processor trong 1 khoản thời gian bằng nhau được gọi là Quantum
- Ví dụ

Tiến trình	Thời điểm vào	t/g xử lý
P1	0	24
P2	1	3
P3	2	3
Quantum=4		

Tiến trình	P1	P2	P3	P1	P1	P1	P1	
Thời điểm	0	4	7	10	14	18	22	

4.5 Các chiến lược điều phối

- Chiến lược theo độ ưu tiên:
  - 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 của tiến trình do HĐH gán và có thể bị thay đổi
  - Giải thuật điều phối với độ ưu tiên có thể theo nguyên tắc độc quyền hay không độc quyền
  - Điều phối với độ ưu tiên và không độc quyền sẽ thu hồi processor từ tiến trình hiện hành để cấp cho tiến trình mới nếu độ ưu tiên của tiến trình này cao hơn
  - Điều phối với độ ưu tiên và độc quyền sẽ chỉ chèn tiến trình mới vào danh sách sẵn sàng tại vị trí phù hợp

4.5 Các chiến lược điều phối

Ví dụ

Tiến trình	Độ ưu tiên	t/g xử lý
P1	3	24
P2	2	3
P3	1	3

Thời điểm cấp processor		
P1	P2	P3
0	24	27

Nhược điểm: Tiến trình có độ ưu tiên thấp dễ rơi vào trạng thái chờ vô hạn

=>Cần giảm độ ưu tiên của tiến trình sau mỗi lần được cấp processor

4.5 Các chiến lược điều phối

- Chiến lược công việc ngắn nhất (shortest job first - SJF):
  - Đây là một trường hợp đặc biệt của giải thuật điều phối với độ ưu tiên
  - độ ưu tiên *p* được gán cho mỗi tiến trình là nghịch đảo của thời gian xử lý *t* mà tiến trình yêu cầu : $p = 1/t$
  - CPU được sẽ được cấp phát cho tiến trình yêu cầu ít thời gian nhất để kết thúc tiến trình
  - Giải thuật này cũng có thể độc quyền hoặc không độc quyền

4.5 Các chiến lược điều phối

- Chiến lược nhiều cấp độ ưu tiên
  - Phân lớp các tiến trình tùy theo độ ưu tiên của chúng để có cách thức điều phối thích hợp cho từng nhóm
  - 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

CHƯƠNG III:
QUẢN LÝ BỘ NHỚ

1. TỔNG QUAN

1.1 Vì sao phải tổ chức, quản lý bộ nhớ?

- CPU chỉ có thể trao đổi thông tin với bộ nhớ chính
- Các chương trình muốn được thực thi cần được nạp vào bộ nhớ chính, tạo lập tiến trình tương ứng để xử lý
- Các hệ thống đa chương trên bộ nhớ chính ngoài HĐH có thể có nhiều tiến trình đang hoạt động
- Kích thước bộ nhớ chính là hữu hạn nhưng yêu cầu bộ nhớ thì vô hạn
- …

1.1 Vì sao phải tổ chức, quản lý bộ nhớ?

- Như vậy, HĐH cần phải tổ chức quản lý bộ nhớ một cách hợp lý để có thể:
  - Đưa bất kỳ một tiến trình nào đó vào bộ nhớ khi có yêu cầu, cho dù khi trên bộ nhớ không còn không gian trống
  - Bảo vệ các tiến trình của hệ điều hành và các tiến trình trên bộ nhớ, tránh các trường hợp truy xuất bất hợp lệ xảy ra.

1.2 Nhiệm vụ của bộ phận quản lý bộ nhớ

- Tái định vị
- Bảo vệ bộ nhớ
- Chia sẻ bộ nhớ
- Tổ chức bộ nhớ logic
- Tổ chức bộ nhớ vật lý

Tái định vị

- Trong các hệ thống đa chương không gian bộ nhớ chính thường được chia sẽ cho nhiều tiến trình và yêu cầu bộ nhớ của các tiến trình luôn lớn hơn không gian bộ nhớ vật lý mà tiến trình mà hệ thống hiện có
- ⇨ Cần thực hiện cơ chế hoán đổi (Swap):
  - ⇨ Một chương trình đang hoạt động trên bộ nhớ sẽ bị đưa ra đĩa (swap-out) và sẽ được đưa vào lại(swap-in) tại thời điểm thích hợp

Tái định vị(tt)

- Khi thực hiện swap-in 1 chương trình vào lại bộ nhớ HĐH phải định vị nó đúng vào vị trí mà trước khi nó bị swap-out
- ⇨ HĐH phải có cơ chế ghi lại tất cả các thông tin liên quan đến 1 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.

Bảo vệ bộ nhớ

- 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.
- Mỗi tiến trình chỉ được phép truy suất đến không gian địa chỉ mà HĐH đã cấp cho nó
- ⇨ Bộ phận Qlý bộ nhớ phải biết không gian địa chỉ của tất cả các tiến trình trên bộ nhớ
- ⇨ Khi tiến trình đưa ra địa chỉ truy xuất bộ phận Qlý bộ nhớ phải kiểm tra tất cả các yêu cầu truy xuất bộ nhớ của mỗi

Chia sẻ bộ nhớ

- Bất kỳ một chiến lược nào được cài đặt đều phải có tính mềm dẻo để cho phép nhiều tiến trình có thể truy cập đến cùng một địa chỉ trên bộ nhớ chính

Tổ chức bộ nhớ logic

- 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
- Cách tổ chức này có sự kết hợp chặt chẻ với phần cứng máy tính nhưng lại không phù hợp với cách xây dựng của chương trình
- Đại đa số các chương trình được tổ chức thành các modul

Tổ chức bộ nhớ vật lý

- Bộ nhớ máy tính được tổ chức theo 2 cấp:
  - Bộ nhớ chính: tốc độ truy xuất nhanh, nhưng giá thành cao và dữ liệu không thể tồn tại lâu dài trên nó.
  - Bộ nhớ phụ: giá rẻ, dung lượng lớn, dữ liệu được lưu trữ lâu dài nhưng tốc độ truy xuất chậm.
- ⇨ Theo giản đồ 2 cấp này, việc tổ chức luồng thông tin giữa bộ nhớ chính và bộ nhớ phụ là nhiệm vụ quan trọng của hệ thống

1.3 Không gian địa chỉ và không gian vật lý

- 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

1.4 Các cấu trúc chương trình

- Cấu trúc chương trình tuyến tính
- Cấu trúc chương trình động
- Cấu trúc chương trình Overlay
- Cấu trúc chương trình phân trang
- Cấu trúc chương trình phân đoạn

Cấu trúc chương trình tuyến tính

- Tất cả các modun, thư viện sử dụng trong chương trình khi biên dịch sẽ được biên dịch thành 1 modun duy nhất
- Khi thực hiện HĐH phải nạp toàn bộ modun này vào bộ nhớ
- Cấu trúc chương trình này có tính độc lập cao và có tốc độ thực thi cao
- Làm lãng phí bộ nhớ vì kích thước chương trình tăng lên khi biên dịch

Cấu trúc chương trình động

- Chương trình được viết dưới dạng các modun riêng rẽ
- Được biên dịch thành các modun riêng rẽ, các thư viện chuẩn của HĐH và của NNlập trình không được tích hợp trong modun chính của chương trình
- Khi thực thi chương trình chỉ 1 modun chính được nạp vào bộ nhớ, các modun khác khi cần sẽ được nạp vào sau
- Cấu trúc này tiết kiệm được không gian nhớ nhưng thực thi chập hơn cấu trúc tuyến tính

Cấu trúc chương trình Overlay

- Chương trình được biên dịch thành các modun riêng rẽ
- Các modun chương trình được chia thành các mức khác nhau:
  - Mức 0: Chứa modul gốc dừng để nạp chương trình
  - Mức 1: Chức các modul được gọi bởi mức 0
  - Mức 2: Chức các modul được gọi bởi mức 1
  - …
  - Mức i: Chức các modul được gọi bởi mức i-1

Cấu trúc chương trình Overlay(tt)

- Các modun trong cùng một mức có thể có kích thước khác nhau, kích thước của modun lớn nhất trong lớp được xem là kích thước của mức
- Bộ nhớ dành cho chương trình cũng được tổ chức thành các mức tương ứng với các chương trình
- Khi thực hiện chương trình HĐH nạp sơ đồ overlay của chương trình vào bộ nhớ sau đó nạp các modun cần thiết ban đầu vào bộ nhớ
- HĐH dựa vào sơ đồ overlay để nạp các modun khác nếu cần

Cấu trúc chương trình phân trang

- Các modun chương trình được biên dịch thành 1 modun duy nhất nhưng sau đó được chia thành các phần có kích thước bằng nhau được gọi là các trang
- Bộ nhớ phải được phân trang, tức chia thành các không gian nhớ bằng nhau gọi là khung trang
- HĐH phải xây dựng bộ điều khiển trang(PCT-page control table)

Cấu trúc chương trình phân đoạn

- Chương trình được biên dịch thành nhiều modun độc lập, được gọi là các đoạn
- Bộ nhớ phải được phân đoạn, tức chia thành các không gian có kích thước có thể không bằng nhau tương ứng với kích thước của các đọan chương trình
- Khi thực hiện chương trình HĐH có thể nạp tất cả các đoạn hoặc 1 vài đoạn cần thiết vào các phân đoạn nhớ liên tiếp hoặc k liên tiếp
- HĐH phải xây dựng bộ điều khiển đoạn(SCT-Segment control table)

2. KỸ THUẬT CẤP PHÁT BỘ NHỚ

2.1 Kỹ thuật phân vùng cố định

- Không gian địa chỉ được chia thành 2 vùng cố định
  - Vùng địa chỉ thấp dùng để chứa HĐH
  - Vùng còn lại (tạm gọi là user program) cấp cho các tiến trình được nạp vào bộ nhớ chính

2.1 Kỹ thuật phân vùng cố định(tt)

- Với hệ thống đơn chương:
  - Việc quản lý bộ nhớ đơn giản vì vùng nhớ user program chỉ cấp cho 1 chương trình
  - HĐH sử dụng 1 thanh ghi giới hạn để ghi địa chỉ ranh giới giữa HĐH và chương trình người sử dụng
  - Khi chương trình người sử dụng đưa ra địa chỉ cần truy xuất, HĐH sẽ so sánh với giá trị giới hạn được ghi trong thanh ghi giới hạn
    - Nếu nhỏ hơn giá trị giới hạn thì HĐH từ chối việc truy suất
    - Ngược lại, nếu lớn hơn sẽ cho phép truy xuất

2.1 Kỹ thuật phân vùng cố định(tt)

- Với hệ thống đa chương:
  - Vùng nhớ user program được chia n phần không nhất thiết phải bằng nhau. Mỗi phần được được gọi là 1 phân vùng
  - Mỗi tiến trình có thể được nạp vào 1 phân vùng bất kỳ nếu kích thước của nó <= kích thước của phân vùng và phân vùng này còn trống
  - Khi có tiến trình cần được nạp vào bộ nhớ mà không còn phân vùng trống thí HĐH sẽ swap-out 1 tiến trình tại 1 phân vùng nào đó có kích thước vừa đủ, không chứa tiến trình đang ở trạng thái ready hoặc running và không có quan hệ với tiến trình đang ở trạng thái running khác để nạp tiến trình vừa có yêu cầu

2.1 Kỹ thuật phân vùng cố định(tt)

Hình 3.1 Ví dụ về phân vùng cố định của bộ nhớ 64MByte

2.1 Kỹ thuật phân vùng cố định(tt)

- Có 2 khó khăn với việc dùng phân vùng cố định có kích thước bằng nhau
  - Thứ 1: Nếu chương trình có kích thước quá lớn so với 1 kích thước của phân vùng, để giải quyết việc này thì:
    - Người lập trình phải thiết kế chương trình theo cấu trúc overlay
    - Chỉ 1 phần cần thiết của chương trình mới được nạp vào bộ nhớ lúc nạp chương trình. Khi cần mudun nào đó mà không sẵn có trong bộ nhớ người sử dụng phải nạp nó vào đúng phân vùng của chương trình và sẽ ghi đè lên bất kỳ chương trình hoặc dữ liệu ở trong đó

2.1 Kỹ thuật phân vùng cố định(tt)

- Thứ 2: Khi kích thước của chương trình nhỏ hơn kích thước của 1 phân vùng hoặc lớn hơn kích thước của phân vùng nhưng không phải là bội số của kích thước phân vùng.

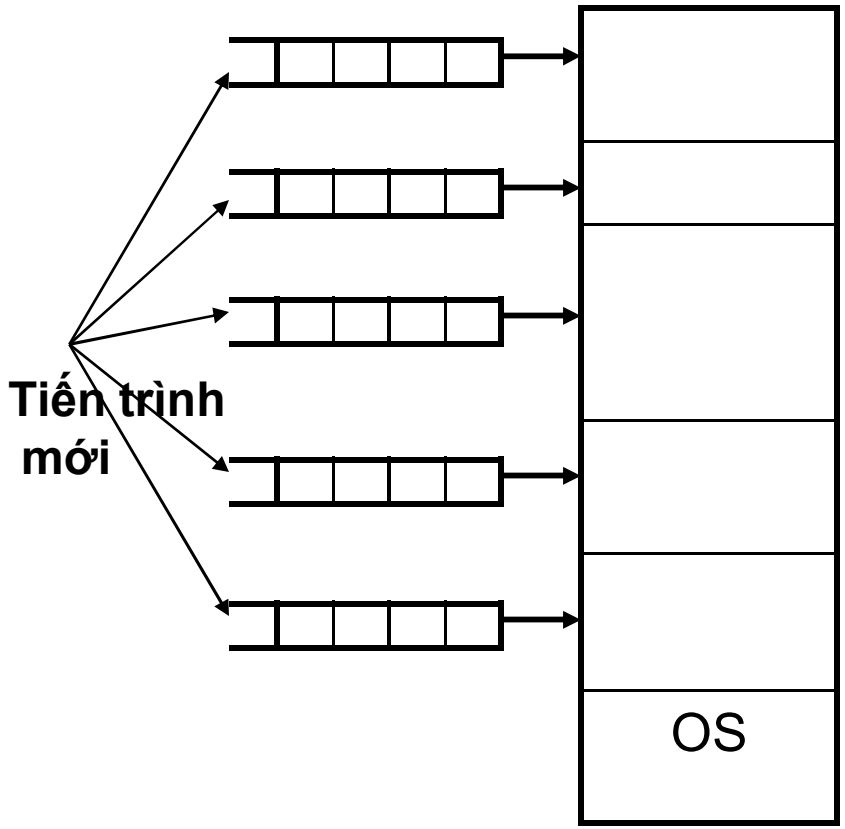
  Điều này gây ra sự ***phân mảnh nội vi***, lãng phí bộ nhớ

2.1 Kỹ thuật phân vùng cố định(tt)

- Để khắc phục nhược điểm này có thể sử dụng phân vùng cố định có kích thước không bằng nhau
- Có 2 lựa chọn để đưa tiến trình vào dạng phân vùng này

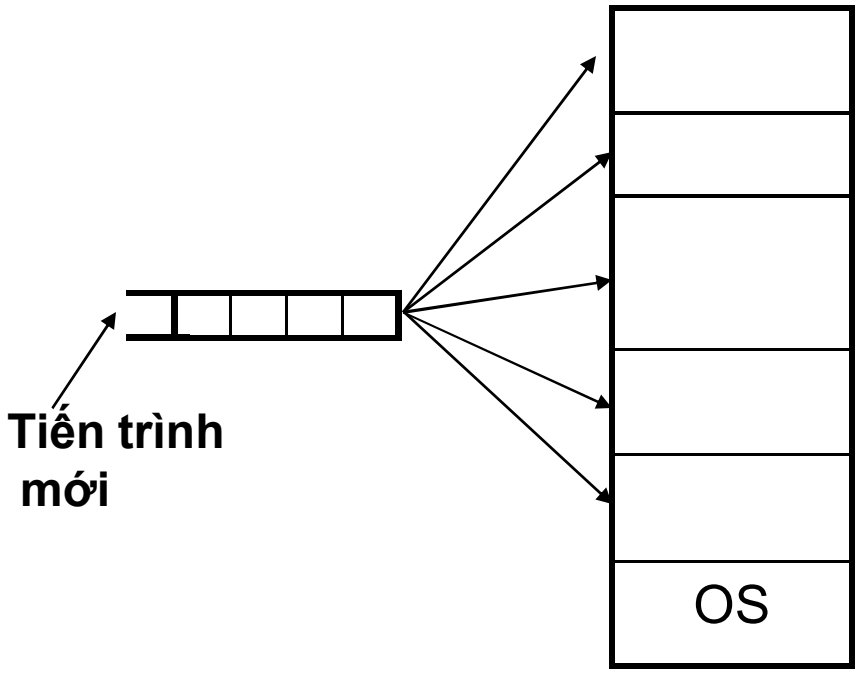
2.1 Kỹ thuật phân vùng cố định(tt)

- Lựa chọn 1:
  - Mỗi phân vùng có một hàng đợi tương ứng
  - Khi 1 tiến trình cần được nạp vào bộ nhớ sẽ đưa vào hàng đợi của phân vùng có kích thước vừa đủ để chứa nó để được đưa vào phân vùng
  - Nhược điểm: Có thể có phân vùng đang trống nhưng lại có nhiều tiến trình đang chờ để vào phân vùng khác



2.1 Kỹ thuật phân vùng cố định(tt)

- Lựa chọn 2:
  - Dùng 1 hàng đời chung cho tất cả các phân vùng
  - Khi có tiến trình muốn nạp vào bộ nhớ nhưng chưa được nạp sẽ được đưa vào hàng đợi
  - Khi có phân vùng trống, HĐH sẽ chọn tiến trình có kích thước vừa đủ để đưa vào phân vùng
  - Phương pháp này gây khó khăn trong việc lựa chọn tiến trình để nạp vào phân vùng

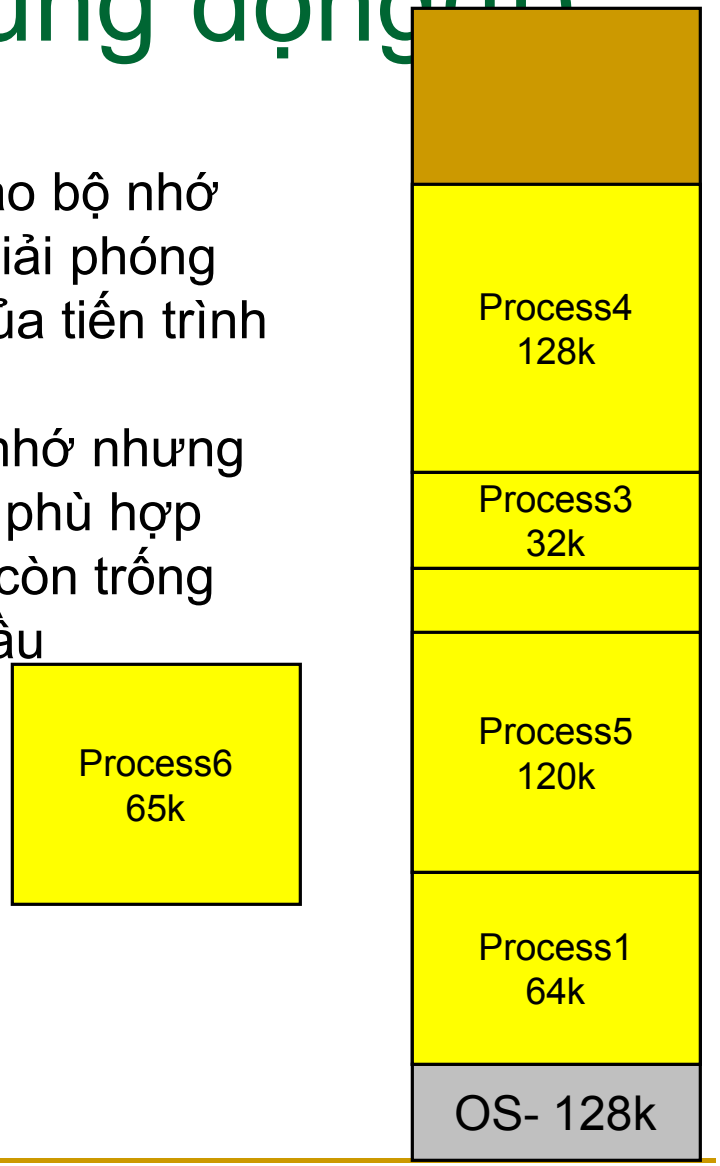


2.2 Kỹ thuật phân vùng động

- Vùng nhớ user program không được phân chia trước
- Khi có tiến trình nạp vào bộ nhớ HĐH cấp cho nó không gian nhớ đúng kích thước của nó
- Khi tiến trình kết thúc vùng nhớ của nó sẽ được thu hồi để HĐH cấp cho tiến trình khác kể cả tiến trình mới có kích thước nhỏ hơn vùng nhớ của tiến trình đã giải phóng đã giải phóng

2.2 Kỹ thuật phân vùng động(tt)

1. Tiến trình 1,2,3,4 lần lượt được nạp vào bộ nhớ
2. Tiến trình 2 kết thúc, vùng nhớ được giải phóng
3. Tiến trình 5 được nạp vào vùng nhớ của tiến trình 2 vừa giải phóng
4. Tiến trình 6 yêu cầu được nạp vào bộ nhớ nhưng không thể vì không có vùng nhớ trống phù hợp để nạp trong khi tổng dung lượng nhớ còn trống lớn hơn kích thước mà tiến trình yêu cầu



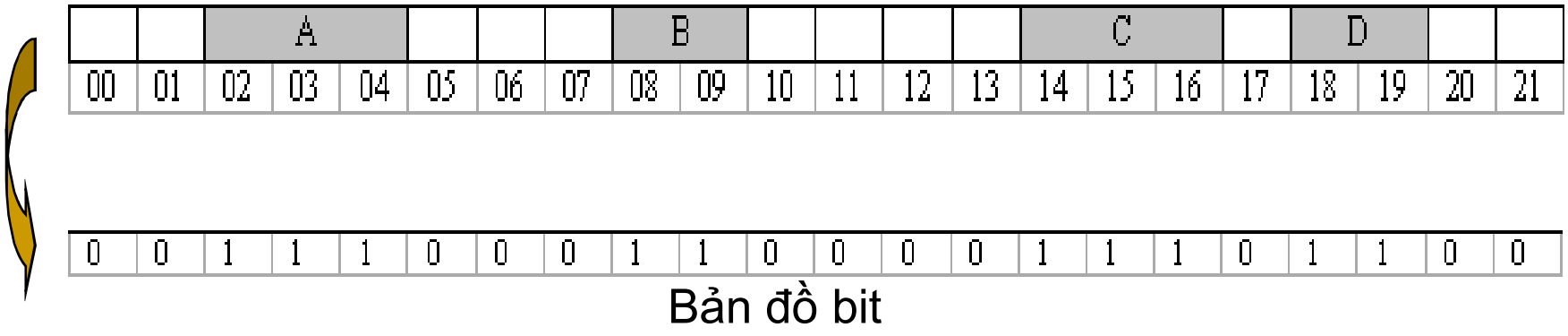
2.2 Kỹ thuật phân vùng động(tt)

- Trong kỹ thuật phân vùng động, HĐH phải đưa ra các cơ chế thích hợp 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.
- Hai cơ chế HĐH đều chia không gian nhớ thành các đơn vị cấp phát có kích thước bằng nhau, các đơn vị cấp phát liên tiếp nhau tạo thành 1 khối nhớ, HĐH cấp phát các khối nhớ này cho các tiến trình

		A						B						C				D			
00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21

2.2 Kỹ thuật phân vùng động(tt)

- Cơ chế bản đồ Bit: Mỗi đơn vị cấp phát được đại diện bởi một Bit trong bản đồ bit. Đơn vị cấp phát còn trống đại diện bằng bit 0, ngược lại đại diện bằng bit 1



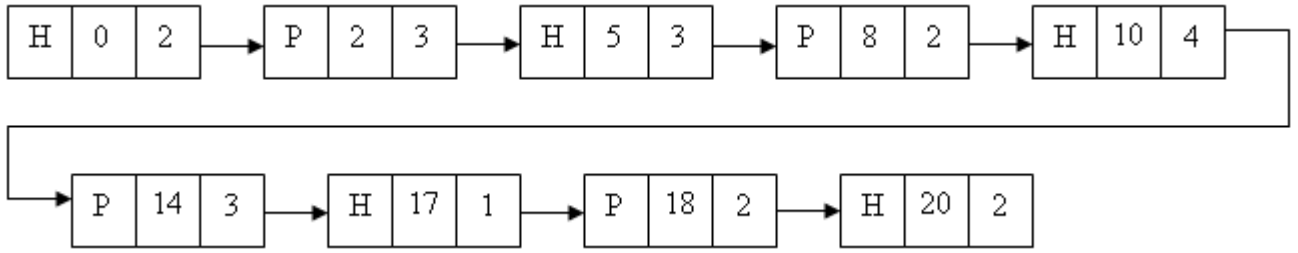
Bản đồ bit

2.2 Kỹ thuật phân vùng động(tt)

- Cơ chế danh sách liên kết:
  - Mỗi khối trên bộ nhớ được đại diện bởi một phần tử trong danh sách liên kết
  - Mỗi phần tử gồm 3 trường chính:
    - Trường đầu tiên: cho biết khối nhớ đã cấp phát (kí hiệu P) hay còn trống (kí hiệu H)
    - Trường thứ 2: cho biết thư tự của đơn vị cấp phát đầu tiên trong khối
    - Trường thứ 3: cho biết đơn vị tổng số đơn vị cấp phát trong khối

2.2 Kỹ thuật phân vùng động(tt)

		A						B						C				D			
00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21



2.2 Kỹ thuật phân vùng động(tt)

- Khi có một tiến trình cần được nạp vào bộ nhớ mà bộ nhớ có nhiều hơn một khối nhớ trống có kích thước lớn hơn kích thước của tiến trình đó, HĐH phải quyết định chọn một khối nhớ phù hợp để nạp tiến trình sao cho việc lựa chọn này dẫn đến việc sử dụng bộ nhớ chính là hiệu quả nhất.
- Có 3 thuật toán mà HĐH sử dụng trong trường hợp này: Best-fit, First-fit, và Next-fit

2.2 Kỹ thuật phân vùng động(tt)

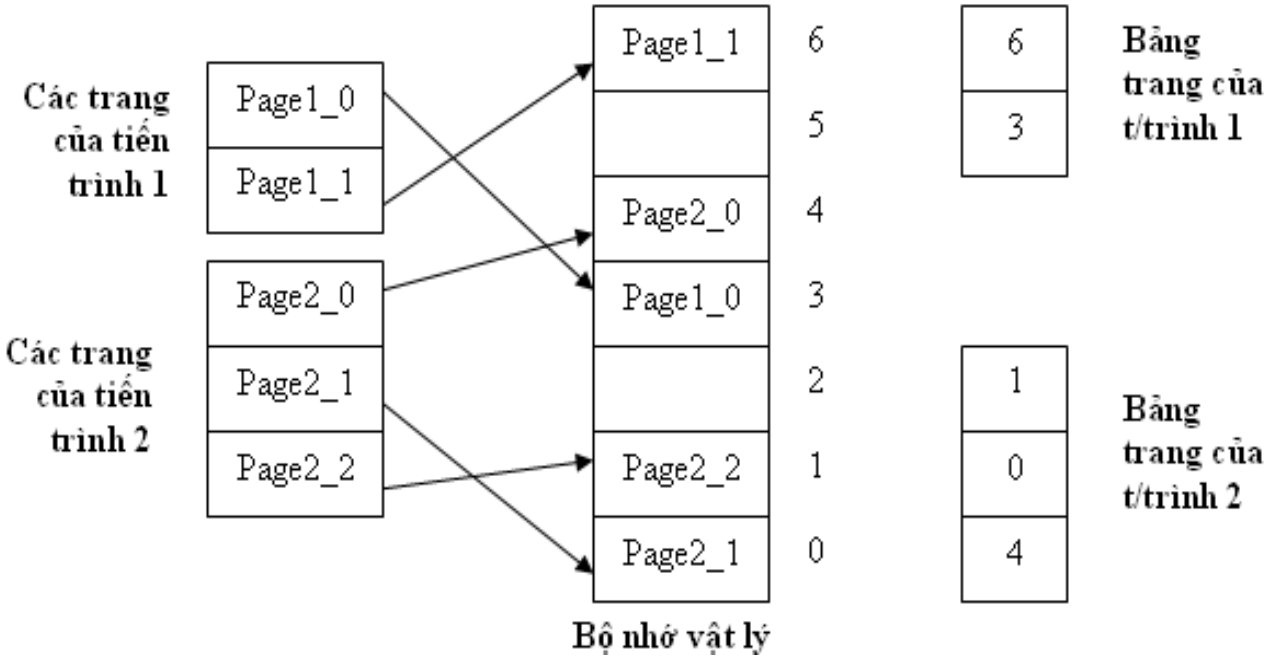
- **Best-fit:** chọn khối nhớ có kích thước vừa đúng bằng kích thước của tiến trình cần được nạp vào bộ nhớ.
- **First-fit:** HĐH sẽ bắt đầu quét qua các khối nhớ trống bắt đầu từ khối nhớ trống đầu tiên trong bộ nhớ, và sẽ chọn khối nhớ trống đầu tiên có kích thước đủ lớn để nạp tiến trình.
- **Next-fit:** tương tự như First-fit nhưng ở đây HĐH bắt đầu quét từ khối nhớ trống kế sau khối nhớ vừa được cấp phát và chọn khối nhớ trống kế tiếp đủ lớn để nạp tiến trình

2.3 Kỹ thuật phân trang đơn

- Bộ nhớ chính được chia thành các phần bằng nhau và cố định, được đánh số bắt đầu từ 0 và được gọi là các khung trang
- Không gian địa chỉ của các tiến trình cũng được chia thành các phần có kích thước bằng kích thước của một khung trang được gọi là các trang
- Khi tiến trình nạp vào bộ nhớ thì các trang được nạp vào các khung trang bất kỳ còn trống có thể không liên tiếp nhau

2.3 Kỹ thuật phân trang đơn(tt)

- HĐH sử dụng các bảng trang(PCT) để theo dõi vị trí các trang của tiến trình trên bộ nhớ. Mỗi tiến trình có bảng trang riêng



2.3 Kỹ thuật phân trang đơn(tt)

- Sự phân mảnh trong cơ chế này?
  - Nếu kích thước của tiến trình không phải là bội số của kích thước 1 khung trang thì sẽ xảy ra hiện tượng phân mãnh nội vi

2.4 Kỹ thuật phân đoạn đơn

- Bộ nhớ chính 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
- Mỗi phân đoạn bao gồm số hiệu phân đoạn và kích thước của nó
- Không gian địa chỉ của các tiến trình kể cả các dữ liệu liên quan cũng được chia thành các đoạn có kích thước không nhất thiết phải bằng nhau

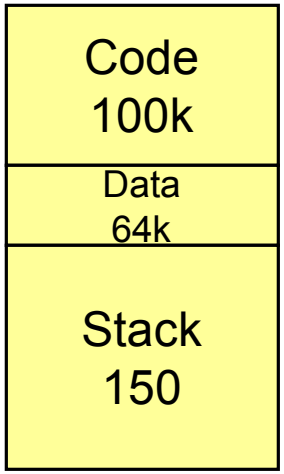
2.4 Kỹ thuật phân đoạn đơn(tt)

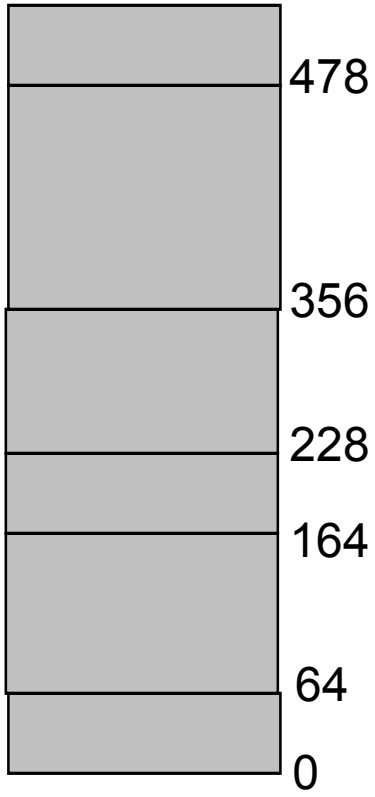
- Khi tiến trình được nạp vào bộ nhớ thì tất cả các đoạn của nó được nạp vào các phân đoạn còn trống trên bộ nhớ, các phân đoạn này có thể không liên tục nhau
- Để theo dõi các đoạn của các tiến trình khác nhau trên bộ nhớ HĐH sử dụng các bảng phân đoạn (SCT), thông thường mỗi tiến trình có 1 bảng phân đoạn riêng

2.4 Kỹ thuật phân đoạn đơn(tt)

- Mỗi phần tử t rong bảng phân đoạn tối thiểu gồm 2 trường
  - Trường thứ nhất: cho biết địa chỉ cơ sở của phân đoạn mà đoạn chương trình tương ứng được nạp
  - Trường thứ 2: cho biết độ dài của phân đoạn

2.4 Kỹ thuật phân đoạn đơn(tt)





base	limit
64	100
164	64
356	150

3. KỸ THUẬT BỘ NHỚ ẢO

3.1 Khái niệm nhớ ảo

- Để thực thi chương trình có kích thước lớn hơn bộ nhớ vật lý cấp phát cho nó
  - cần xây dựng chương trình theo cấu trúc Overlay
  - gây khó khăn cho người lập trình
- Để khắc phục khó khăn cho người lập trình, ý tưởng sử dụng bộ nhớ ảo ra đời
- Kỹ thuật bộ nhớ ảo cho phép xử lý một tiến trình không được nạp toàn bộ vào bộ nhớ vật lý

3.1 Khái niệm nhớ ảo(tt)

- 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, chuyển đổi sang không gian vật lý do hệ điều hành thực hiện với sự trợ giúp của các cơ chế phần cứng

3.2 Cài đặt bộ nhớ ảo

- Có thể cài đặt bộ nhớ ảo theo 2 kỹ thuật
  - Phân trang theo yêu cầu: Sử dụng kỹ thuật phân trang kết hợp với kỹ thuật swap
  - Phân đoạn theo yêu cầu: sử dụng kỹ thuật phân đoạn kết hợp với kỹ thuật swap

3.2.1 Phân trang theo yêu cầu

- Sử dụng kỹ thuật phân trang kết hợp với kỹ thuật swap
- Một chương trình được xem như 1 tập hợp các trang thường trú trên bộ nhớ ngoài
- Khi thực thi hệ thống không nạp toàn bộ chương trình vào bộ nhớ trong mà chỉ nạp những trang cần thiết trong thời điểm hiện tại
  - ⇨Một trang chỉ được nạp vào bộ nhớ trong khi cần thiết

3.2.1 Phân trang theo yêu cầu(tt)

- Cần có cơ chế phần cứng để phân biệt các trang đang ở bộ nhớ trong và các trang đang ở bộ nhớ ngoài
  - ⇨ Tổ chức bảng trang như kỹ thuật phân trang đơn nhưng 1 phần tử trong bảng trang chứa nhiều thông tin phức tạp hơn
  - ⇨ Cần có 1 bit cho biết trang tương ứng của tiến trình có hay không trong bộ nhớ chinh và 1 bit cho biết trang có bị sửa đổi hay không so với lần nập gần nhất

Hiện tượng lỗi trang

Khi hệ thống truy xuất tới 1 trang được đánh dấu là bất hợp lệ sẽ làm phát sinh lỗi trang, HĐH xử lý lỗi trang như sau:

Bước 1: 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 2

Bước 2: Tìm vị trí chứa trang muốn truy xuất trên đĩa.

Bước 3: Tìm một khung trang trống trong bộ nhớ chính

- Nếu tìm thấy: đến bước 4
- Ngược lại, thực hiện cơ chế swap out 1 trang thích hợp trên bộ nhớ chính sau đó cập nhật bảng trang tương ứng rồi đến bước 4

Hiện tượng lỗi trang(tt)

Bước 4:

- Chuyển trang muốn truy xuất từ bộ nhớ phụ vào bộ nhớ chính tại khung trang đã xác định được
- Cập nhật nội dung bảng trang tương ứng.
- Tái kích hoạt tiến trình người sử dụng

Thay thế trang

- Khi các khung đã đầy mà cần nạp thêm trang thì phải thay thế một trang đang có trên khung
- Nếu trang bị thay thế có thay đổi nội dung thì cần phải đưa ra đĩa
- Có các phương pháp chọn phần tử thay thế:
  - Optimal: Thay thế trang sẽ lâu được sử dụng nhất trong tương lai
  - FIFO: trang ở trong bộ nhớ lâu nhất sẽ được chọn thay thế
  - LRU (Least Recently Used): trang được chọn để thay thế sẽ là trang lâu nhất chưa được truy xuất

3.2.2 Phân đoạn đoạn theo yêu cầu

- Bộ nhớ ảo bao gồm các đoạn (segment) có kích thuớc không cố định
- Khi nạp đoạn vào bộ nhớ thì hệ điều hành tìm khoảng trống đủ để nạp đoạn
- Có bảng đoạn quản lý các đoạn

3.2.3 Phân đoạn kết hợp phân trang

- Kết hợp các ưu điểm của phân đoạn và phân trang
- Bộ nhớ ảo bao gồm các đoạn
- Trong mỗi đoạn thực hiện phân trang

Tài liệu tham khảo

- Trần Hạnh Nhi, Giáo trình HĐH nâng cao, ĐH Khoa học Tự nhiên Tp.HCM, 1998
- Nguyễn Gia Định-Nguyễn Kim Tuấn, Nguyên Lý HĐH, NXB Khoa học kỹ thuật, 2005
- William Stallting, Operating Systems, Prentice Hall, 1995

CHƯƠNG IV:
QUẢN LÝ FILE VÀ ĐĨA

1. CÁC KHÁI NIỆM CƠ BẢN

File?

- File hay còn gọi là tập tin, là tập hợp thông tin/dữ liệu được tổ chức theo một cấu trúc nào đó.
- Nội dung của tập tin có thể là chương trình, dữ liệu, văn bản,...
- Mỗi tập tin được lưu trên thiết bị lưu trữ đều được đặt tên.
- Mỗi hệ điều hành có qui ước đặt tên khác nhau, tên tập tin thường có 2 phần: phần tên (name) và phần mở rộng (extension).

Các thuộc tính trên file

- **Tên** (name)
- **Định danh** (identifier)
- **Kiểu** (type)
- **Vị trí** (location)
- **Kích thước** (size)
- **Giờ** (time), **ngày** (date) và **định danh người dùng** (user identification)
- Các thông tin tập tin được lưu trữ trên cấu trúc thư mục và được duy trì trên thiết bị

Các thao tác trên file

- Tạo
- Mở
- Đóng
- Ghi
- Đọc
- Di chuyển
- Xóa
- Tìm
- Lấy thuộc tính
- Đổi tên
- .V.v.

Các kiểu file

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

Cấu trúc file

Các hệ điều hành thường hỗ trợ ba cấu trúc file thông dụng là:

- **Không có cấu trúc**: file là một dãy tuần tự các byte
- **Có cấu trúc**: File là một dãy các mẫu tin có kích thước cố định
- **Cấu trúc cây**: File gồm một cây của những mẫu tin không cần thiết có cùng chiều dài, mỗi mẫu tin có một trường khoá giúp việc tìm kiếm nhanh hơn

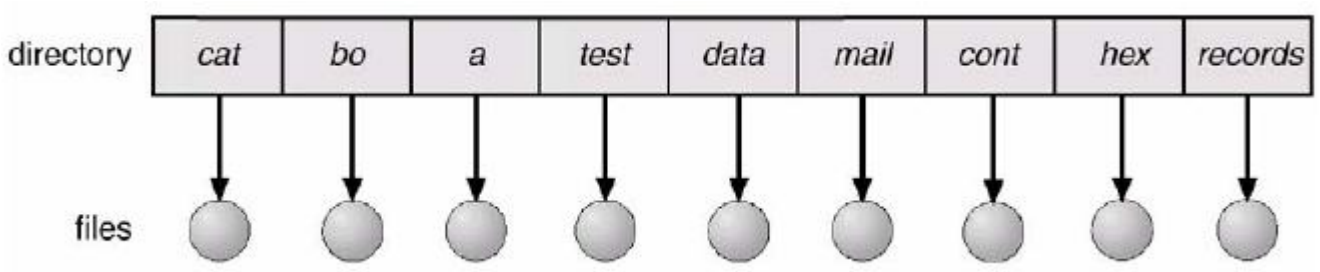
2. CÁC PHƯƠNG PHÁP TRUY XUẤT

- Truy xuất tuần tự
- Truy xuất trực tiếp

3. CẤU TRÚC THƯ MỤC

3.1 Cấu trúc thư mục dạng đơn cấp

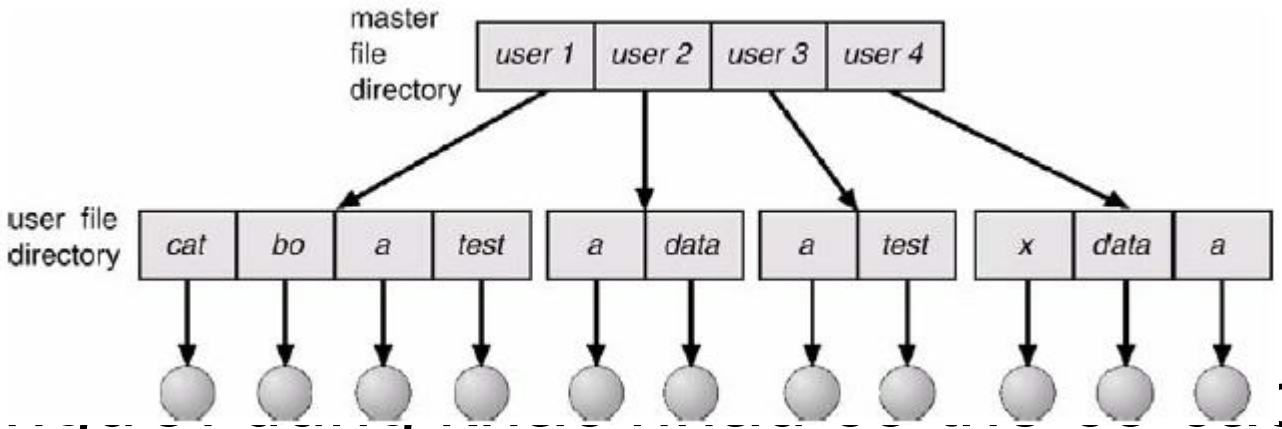
- Một thư mục cho tất cả các tập tin



- Thư mục đơn cấp có nhiều hạn chế khi số lượng tập tin tăng. Vì tất cả tập tin được chứa trong cùng thư mục, chúng phải có tên khác nhau.

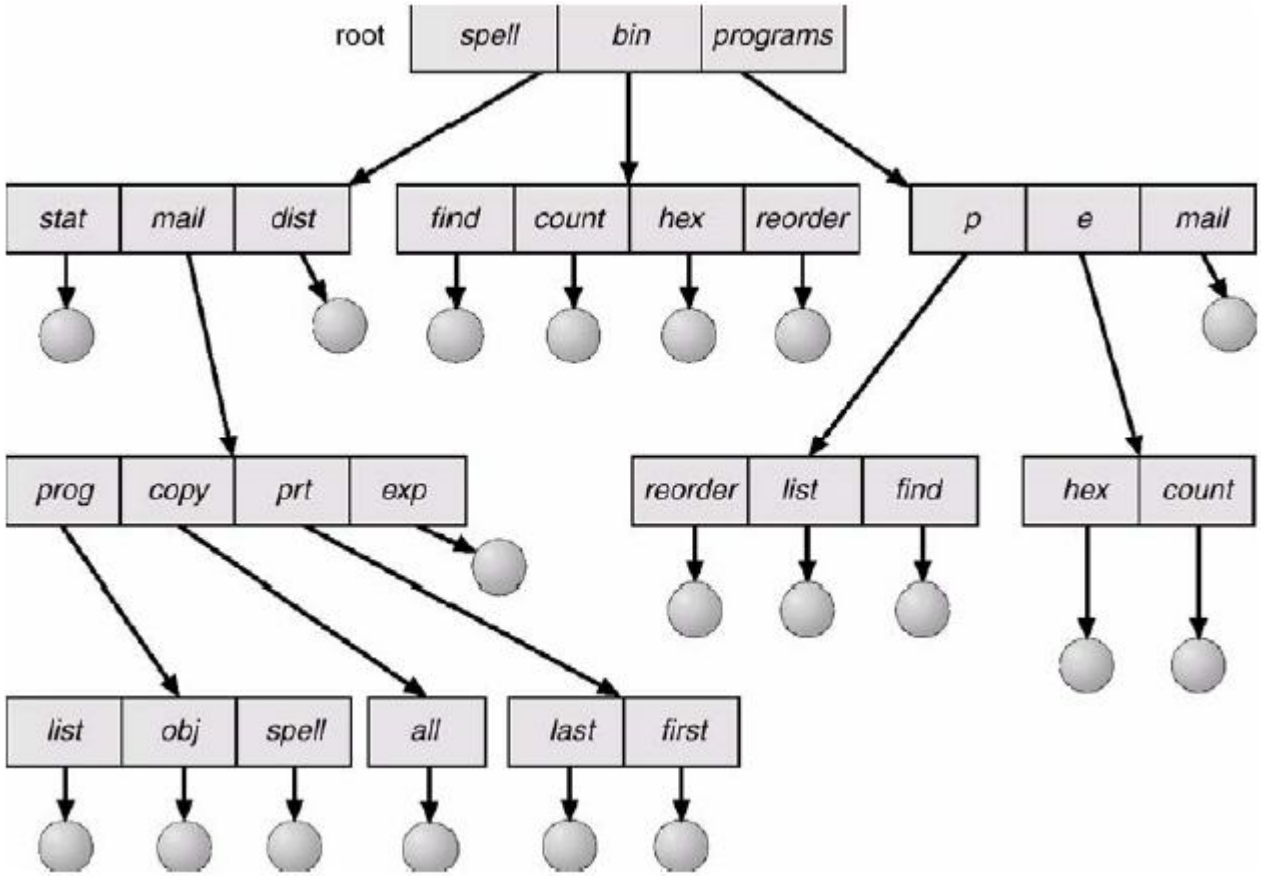
3.2 Cấu trúc thư mục dạng hai cấp

- Mỗi người dùng có 1 thư mục riêng



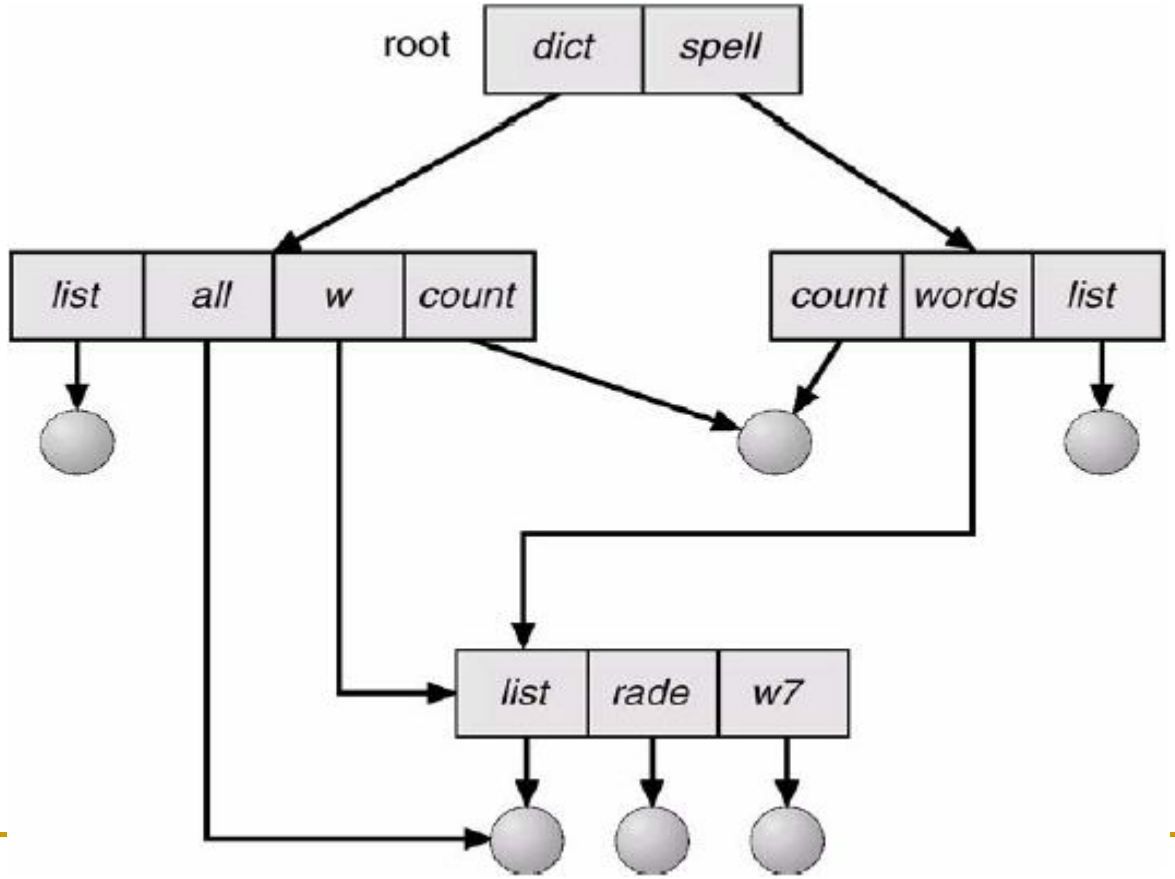
- các [illegible] tập tin với cùng một tên
- Cấu trúc này cô lập một người dùng từ người dùng khác.

3.3 Cấu trúc thư mục dạng cây

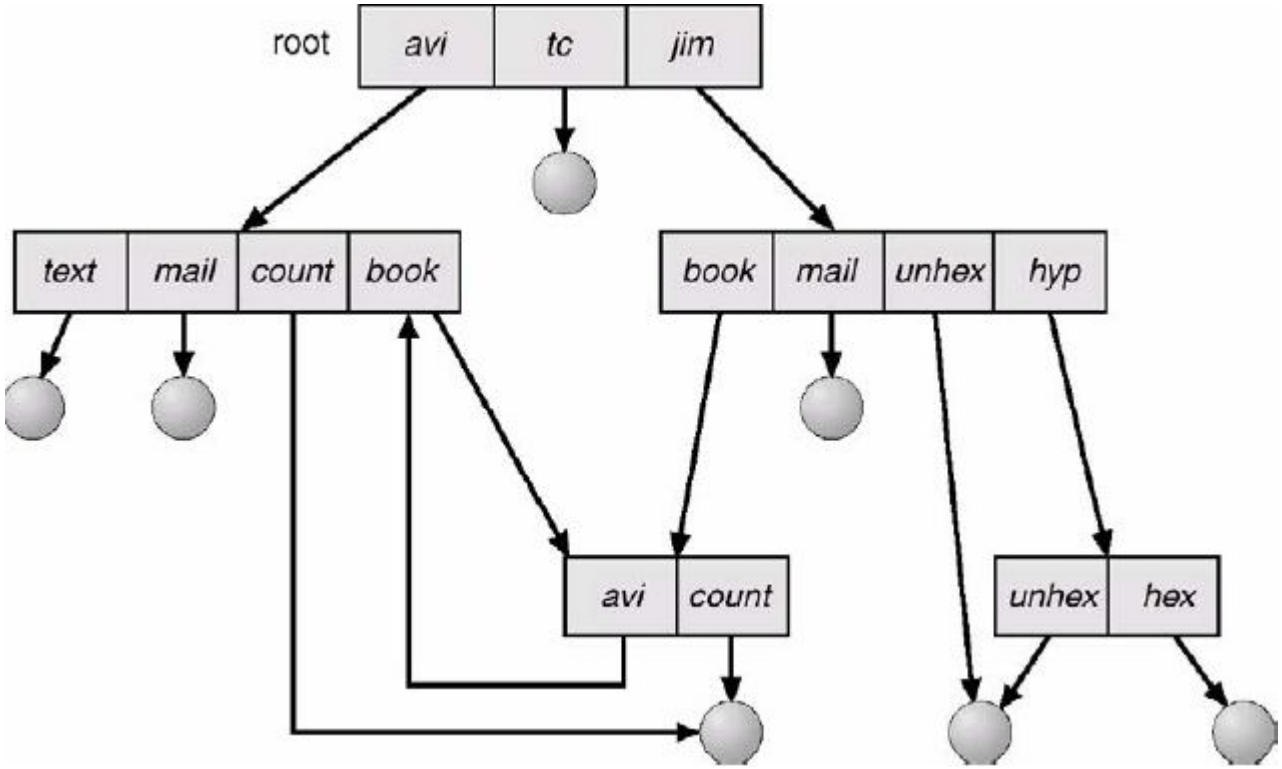


3.4 Cấu trúc thư mục dạng đồ thị không chứa chu trình

- Có chung nhau thư mục con và các file



3.5. Cấu trúc thư mục dạng đồ thị tổng quát



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

4.1 BẢNG DANH MỤC QUẢN LÝ THƯ MỤC, TẬP TIN

- Lưu trữ các thông tin liên quan đến các tập tin và các thư mục đang tồn tại trên đĩa(hoặc thiết bị lưu trữ khác)
- Bảng danh mục gồm nhiều entry, mỗi entry sẽ lưu thông tin về tên, thuộc tính, vị trí lưu trữ,... của một tập tin hay thư mục.
- Khi có tập tin/thư mục được tạo ra, HĐH sẽ dùng một entry trong bảng danh mục để chứa các thông tin của nó
- Khi một tập tin/thư mục xóa khỏi đĩa thì HĐH sẽ giải phóng entry của nó trong bảng danh mục

4.1 BẢNG DANH MỤC QUẢN LÝ THƯ MỤC, TẬP TIN(tt)

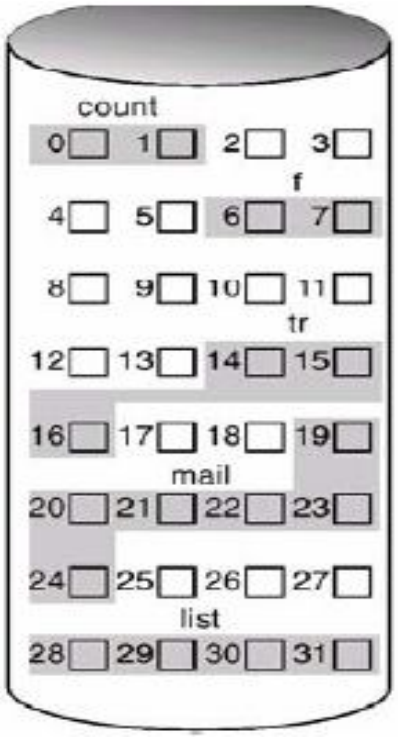
- Số lượng entry trong bảng dnah mục có thể cố định hoặc không cố định
- Bảng danh mục thường được lưu trữ tại một không gian đặc biệt nào đó trên đĩa
- Trong quá trình hoạt động bảng danh mục thường được HĐH nạp từ đĩa vào bộ nhớ để sẵn sàng cho việc truy xuất file của HĐH sau này

4.2 Bảng phân phối vùng nhớ

- HĐH chia không gian đĩa thành các khối (block) có kích thước bằng nhau
- Nội dung file được chia thành các block bằng nhau và bằng kích thước block trên đĩa trừ block cuối cùng
- Khi lưu tập tin trên đĩa HĐH cấp vừa đủ số block để lưu trữ tập tin
  - ⇨ HĐH tổ chức bảng phân phối vùng nhớ để lưu giữ dãy các khối trên đĩa đã cấp phát cho tập tin hay thư mục

4.3 Các phương pháp cấp phát vùng nhớ

- Cấp phát liên tục: lưu trữ tập tin trên dãy các block liên tiếp

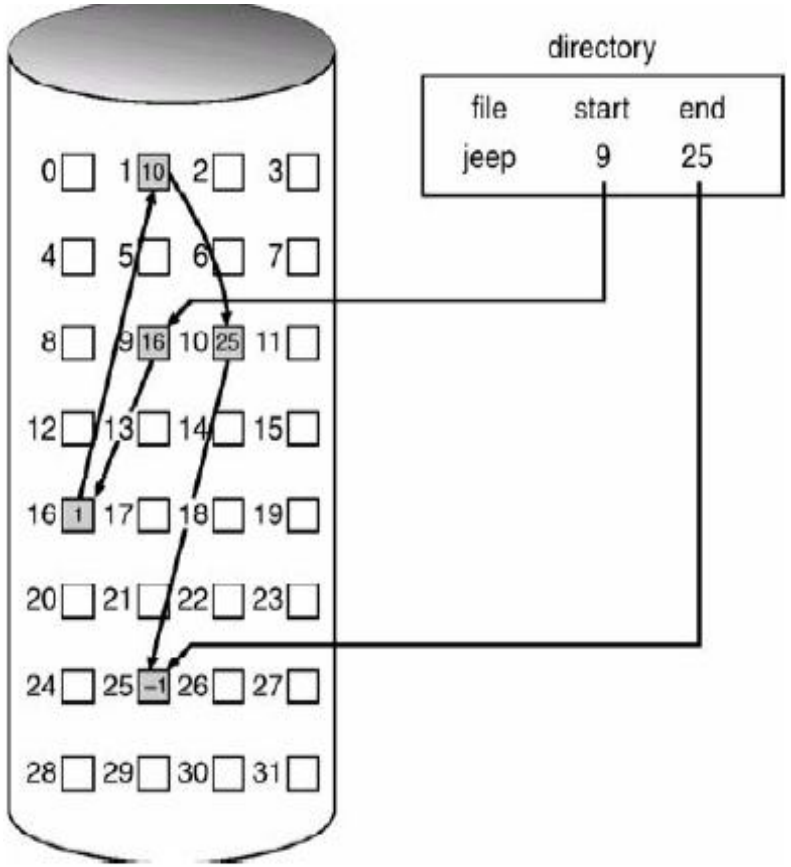


directory

file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

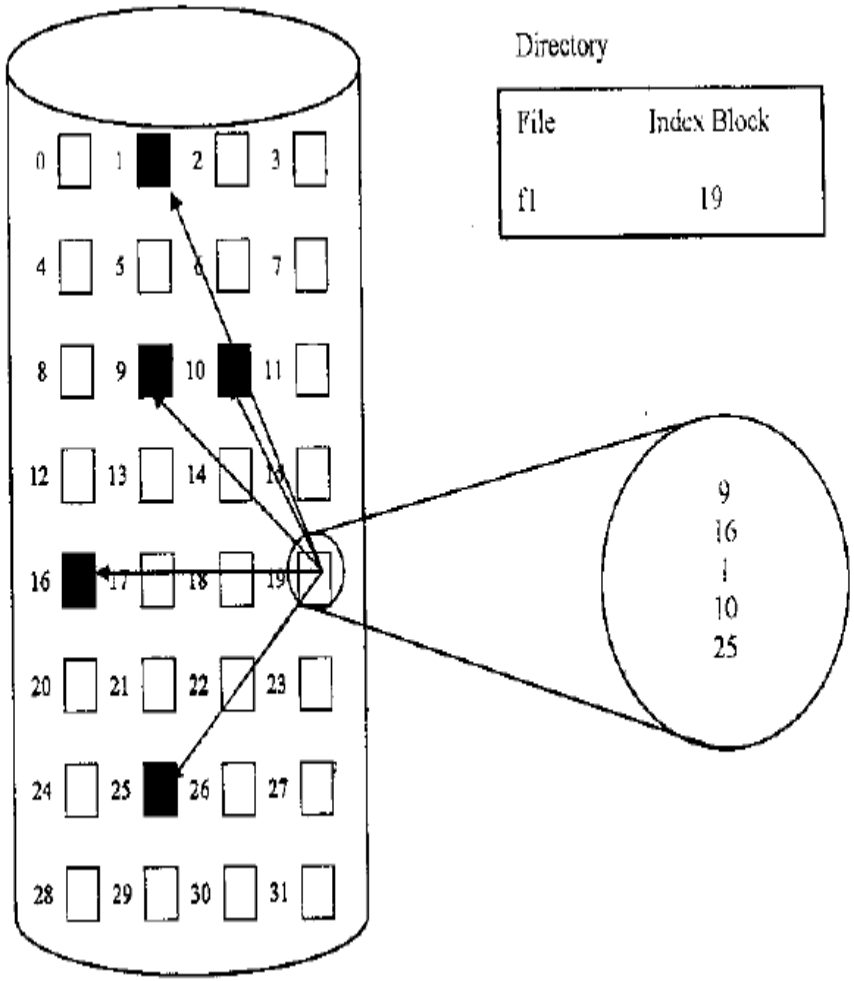
4.3 Các phương pháp cấp phát vùng nhớ(tt)

- Cấp phát theo danh sách liên kết:
  - sử dụng danh sách liên kết các block để quản lý các block chứa file
  - Word đầu tiên của mỗi block đĩa được sử dụng như 1 con trỏ trỏ đến block kế tiếp
  - Kích thước của block đĩa lớn hơn kích thước block file 1 word



4.3 Các phương pháp cấp phát vùng nhớ(tt)

- Cấp phát theo danh sách liên kết sử dung Index:
- Tất cả các con trỏ liên kết các block được lưu vào 1 vị trí gọi là khối chỉ mục
- Mỗi tập tin có khối chỉ mục của chính nó, là 1 mảng địa chỉ block đĩa lưu tập tin

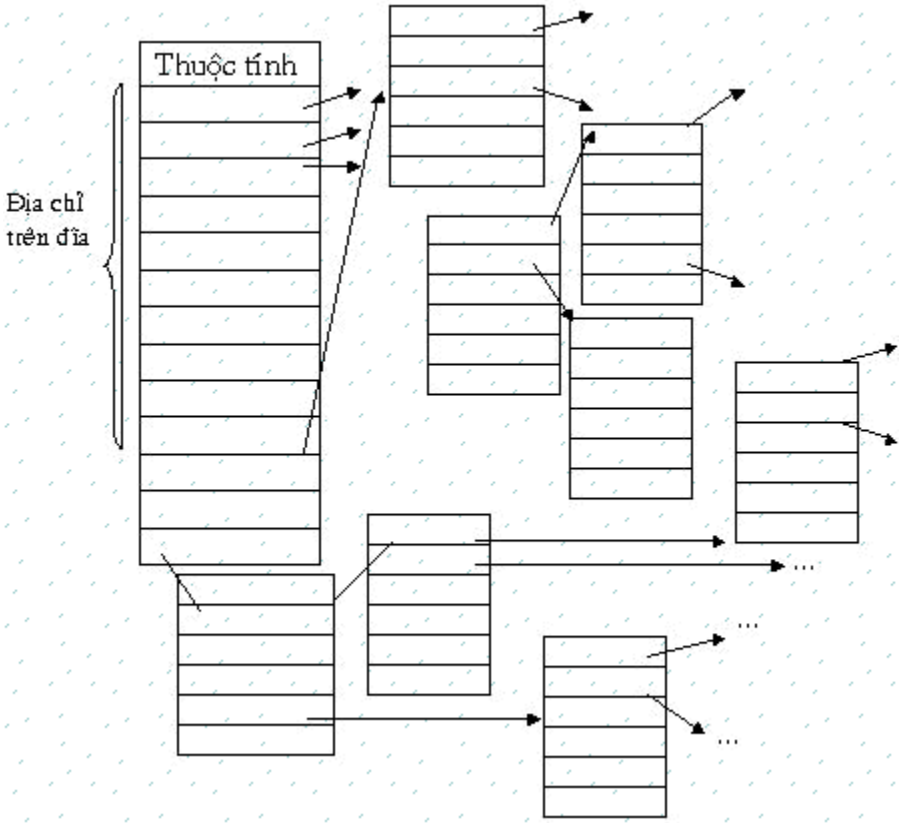


I-NODES

- HĐH thiết kế 1 bảng nhỏ để theo dõi các block của 1 file được gọi là I-nodes
- Một I-nodes gồm 2 phần:
  - Phần 1 chứa các thuộc tính tập tin
  - Phần 2 được chia ra làm 2 phần nhỏ
    - Phần nhỏ thứ nhất 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)
    - Phần tử thứ 12 chứa địa chỉ gián tiếp cấp 2 (double indirect)
    - Phần tử thứ 13 chứa địa chỉ gián tiếp cấp 3 (double indirect)

I-NODES(tt)

- Địa chỉ gián tiếp cấp 1: 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 của tập tin
- Địa chỉ gián tiếp cấp 2: chứa địa chỉ của bảng các khối địa chỉ gián tiếp cấp 1
- Địa chỉ gián tiếp cấp 3: chứa địa chỉ của bảng các khối địa chỉ gián tiếp cấp 2.



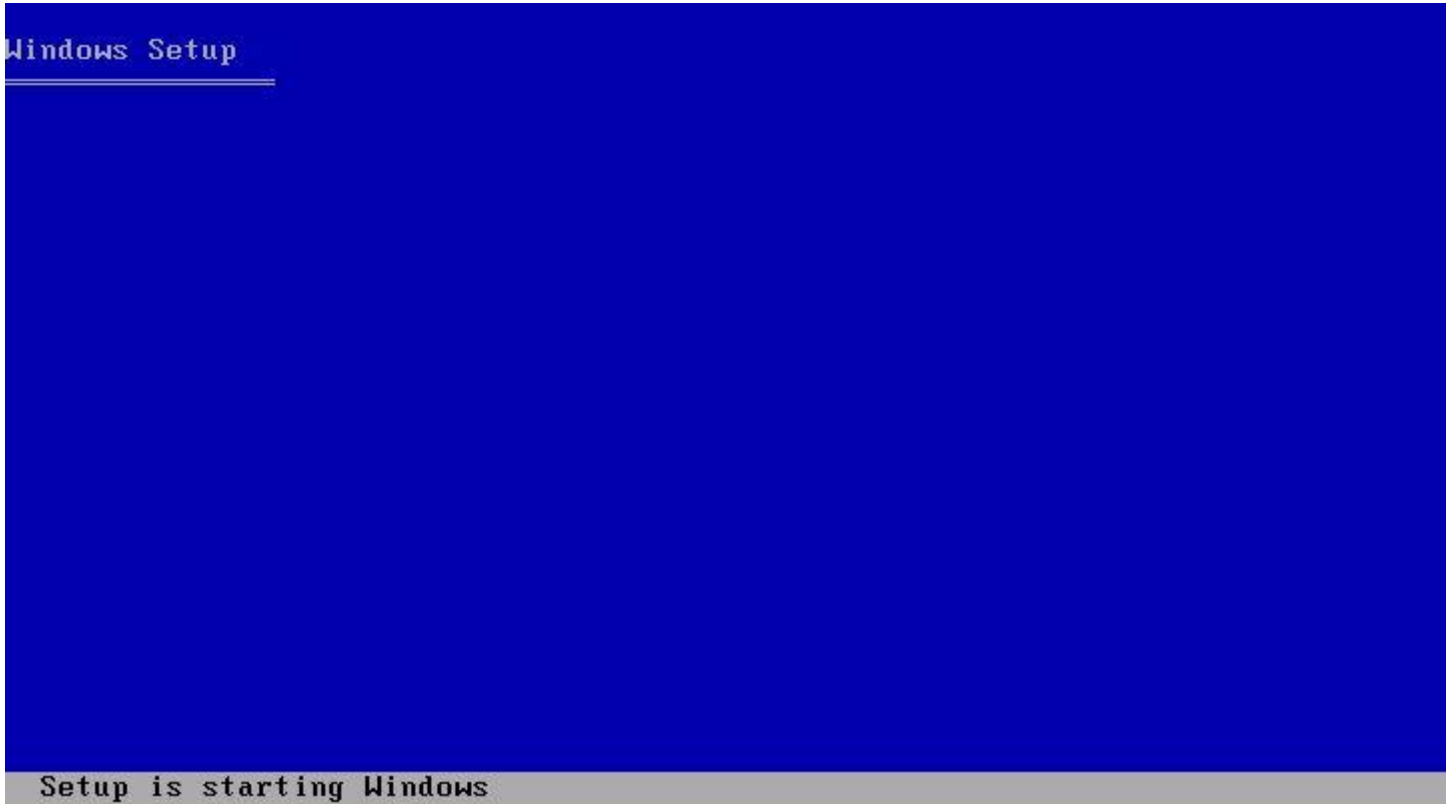
HỆ ĐIỀU HÀNH WINDOWS XP

Bài 1: Cài đặt WindowsXP

- Cần phải có đĩa CD Windows
- Chuẩn bị máy, có đĩa cứng.
- Cho máy khởi động (boot) từ CD.

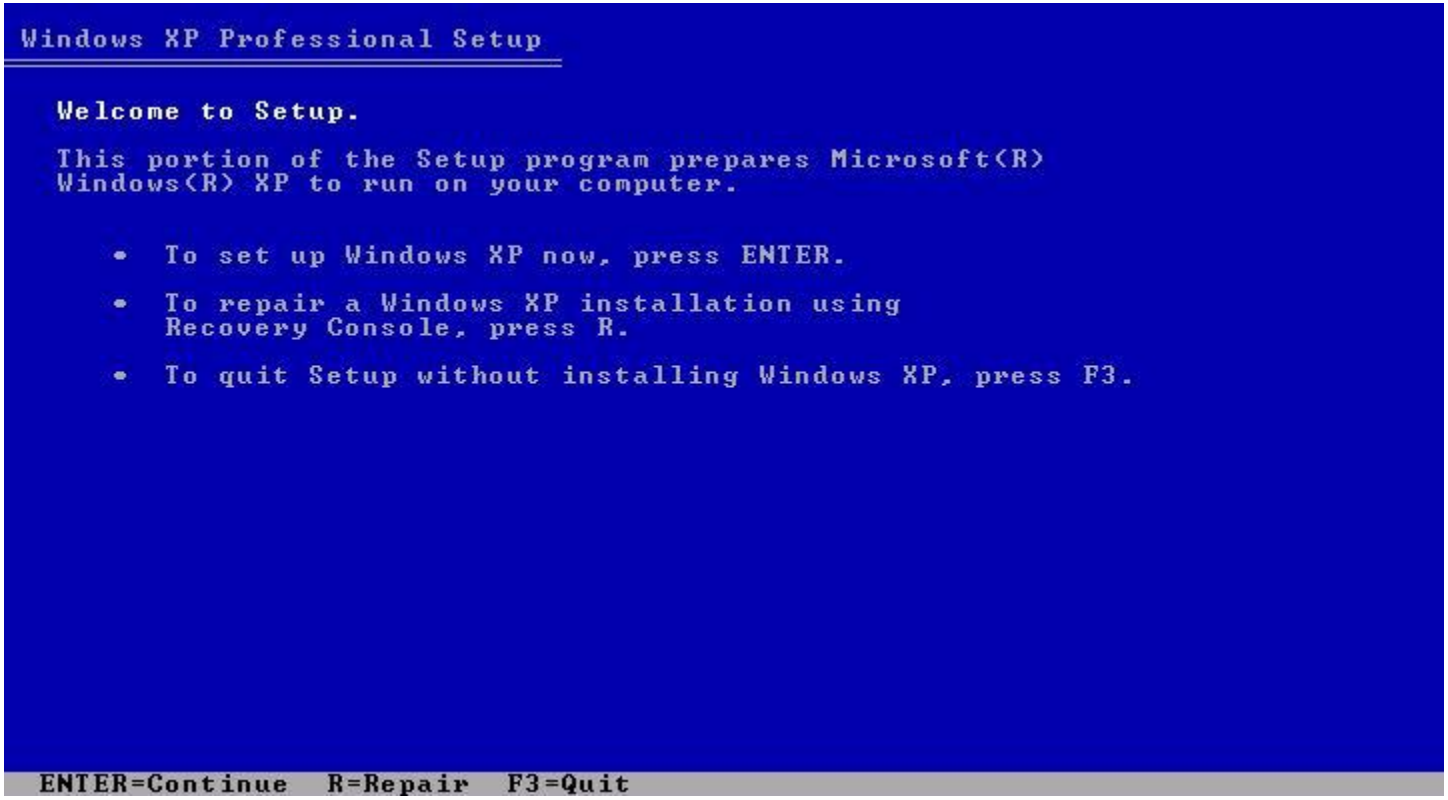
Bắt đầu cài đặt
Đưa đĩa CD Win XP

Chú ý:
Máy phải đặt ở chế độ Boot từ CD



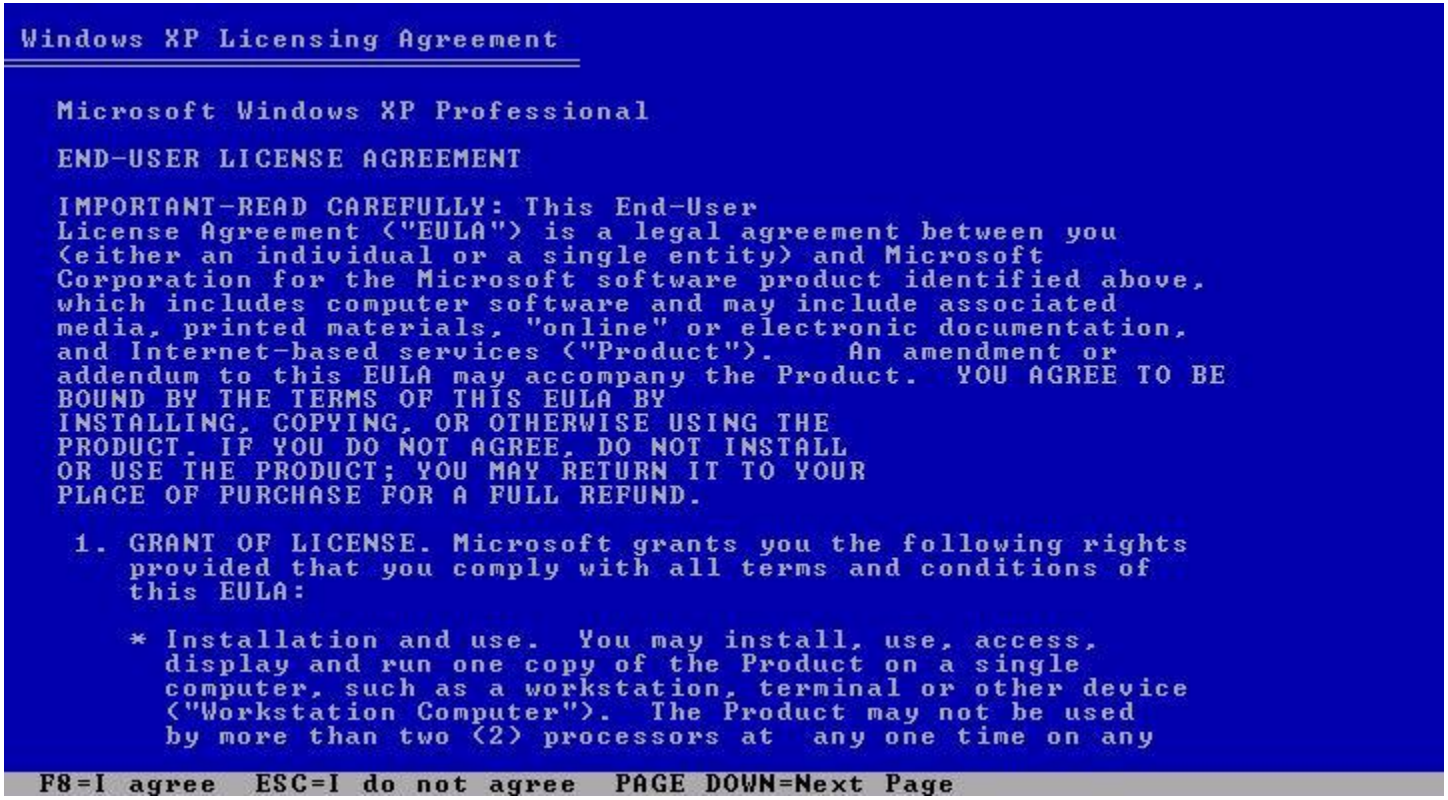
Chọn dạng cài đặt

Enter: tiếp tục
R: Sửa lỗi widows
F3: Thoát



Màn hình thông báo bản quyền

F8: Đồng ý xác nhận bản quyền.
ESC: Không đồng ý

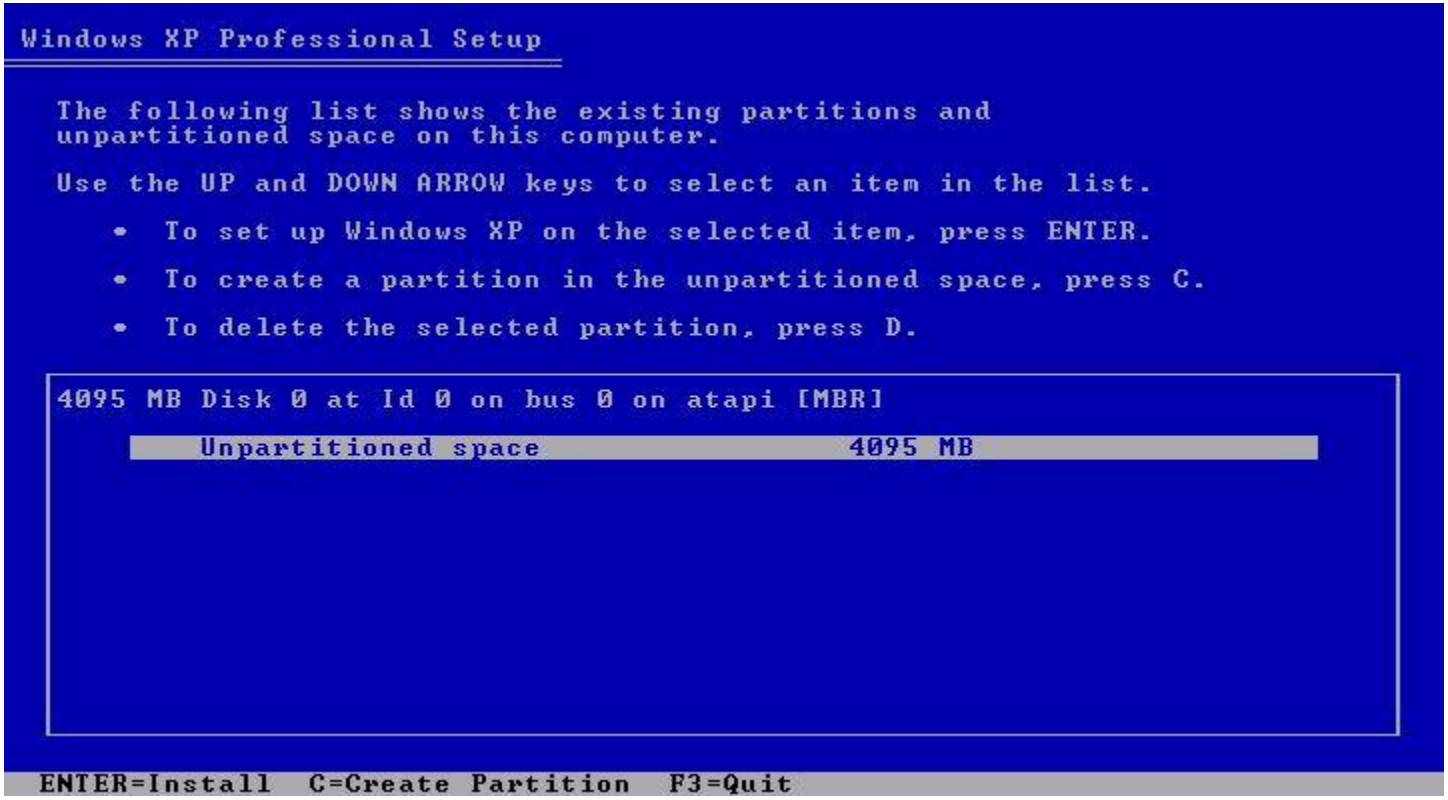


Phân vùng đĩa cứng

ENTER: Để cài WinXP vào phân vùng đã chọn.

C: Để tạo một ổ đĩa mới từ phân vùng trống đã chọn.

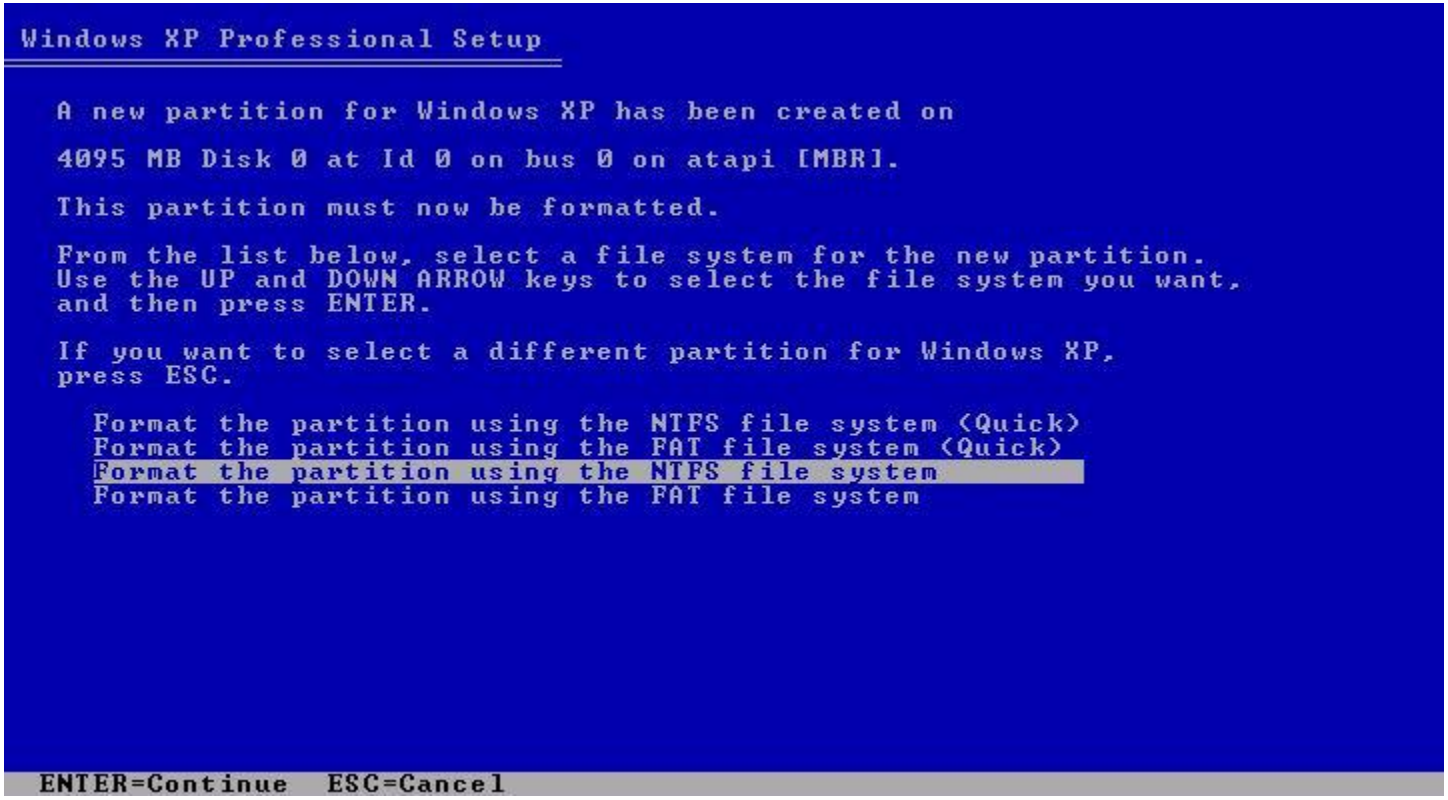
D: Để xoá phân vùng đã chọn và tạo ra một phân vùng trống



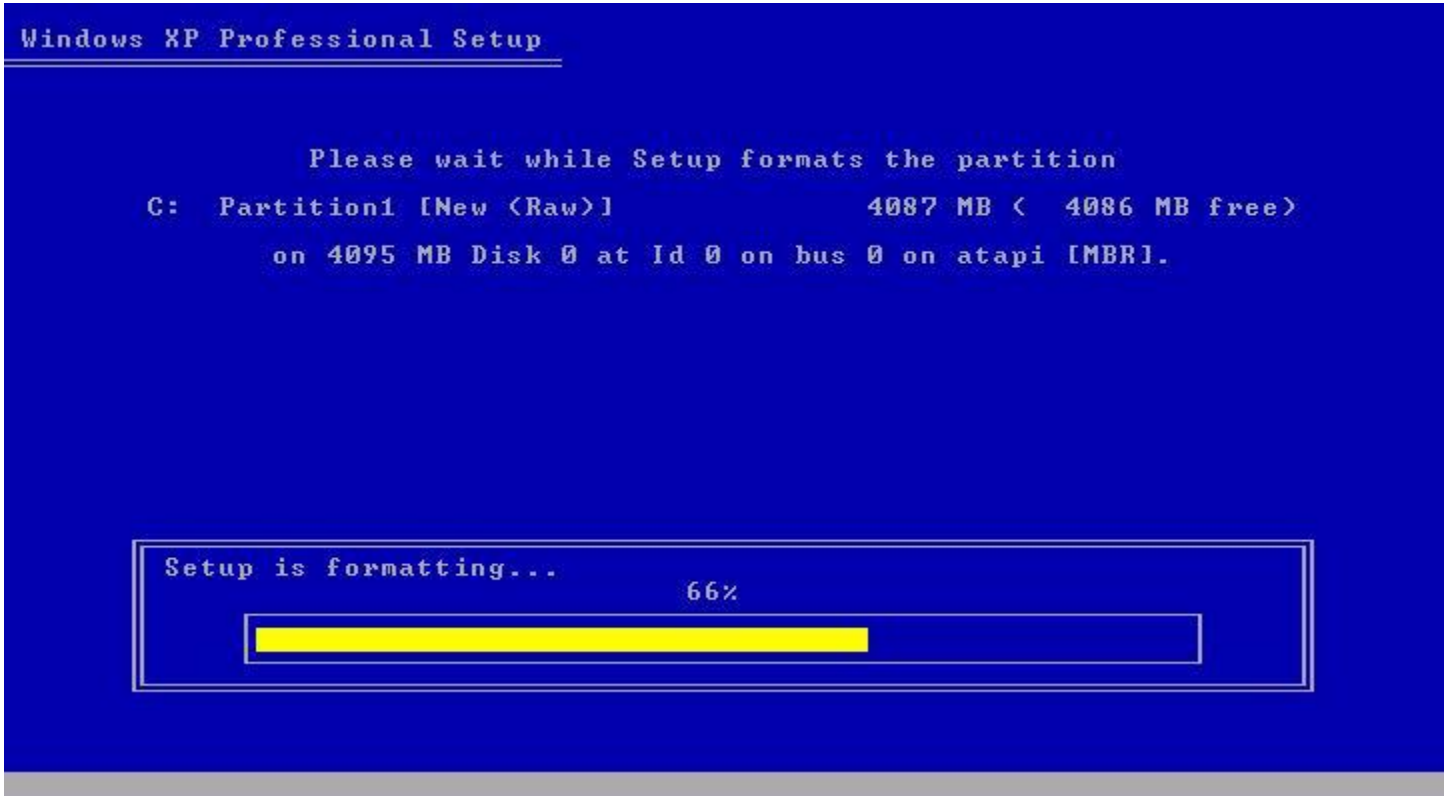
Định dạng đĩa cứng

Chọn loại định dạng đĩa cứng (FAT32, NTFS).

Nên chọn NTFS (vì có tính bảo mật cao,...)

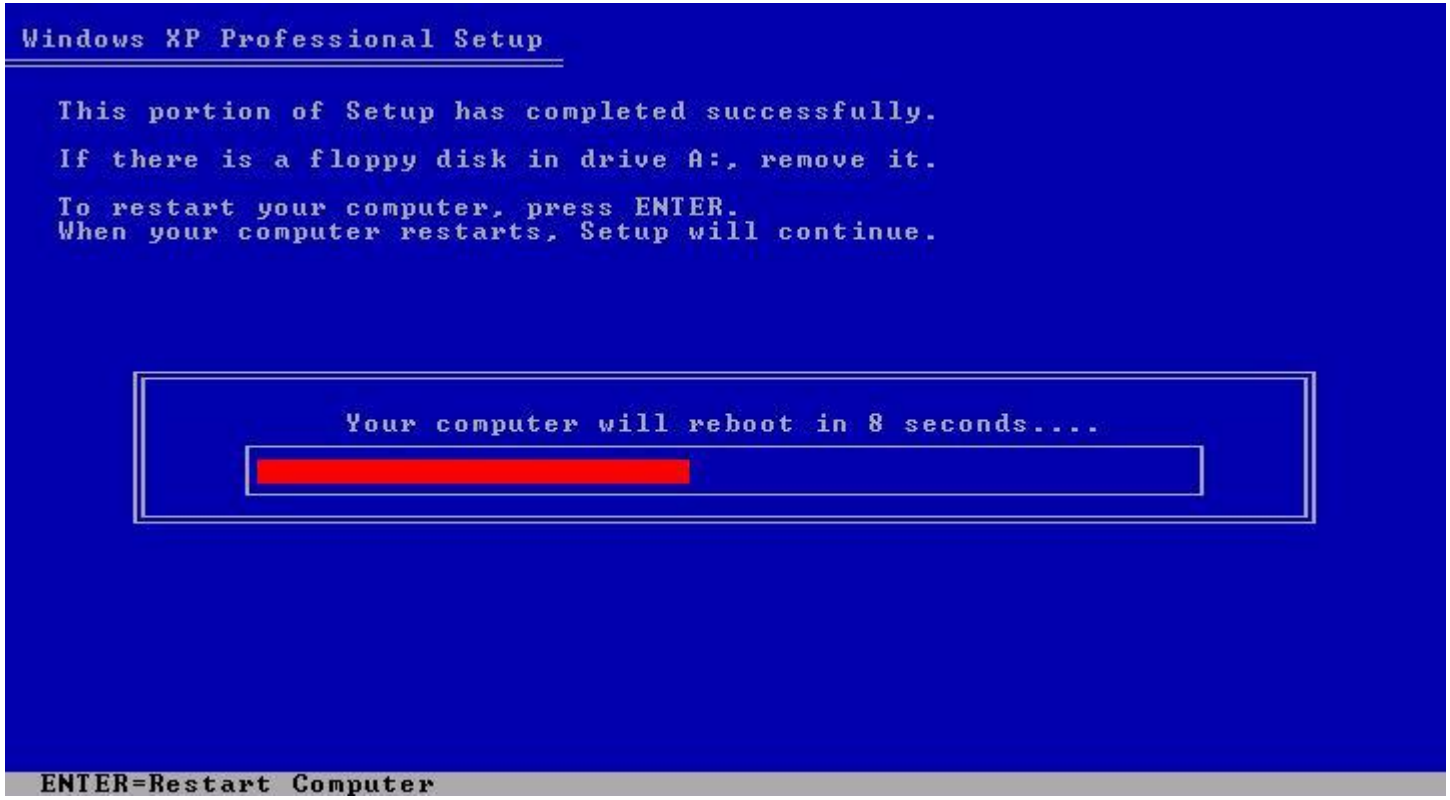


Chờ định dạng đĩa



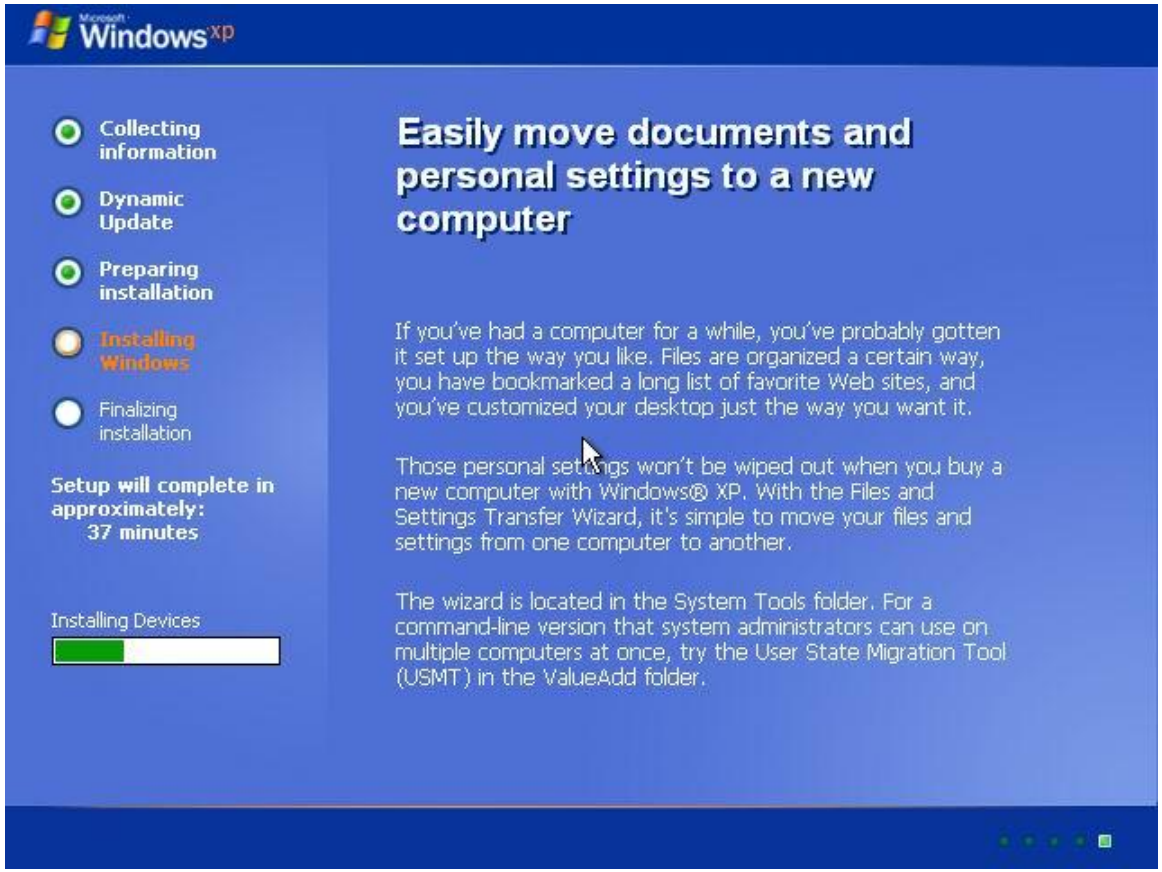
Chờ chương trình cài đặt copy file

Tự động khởi động lại máy

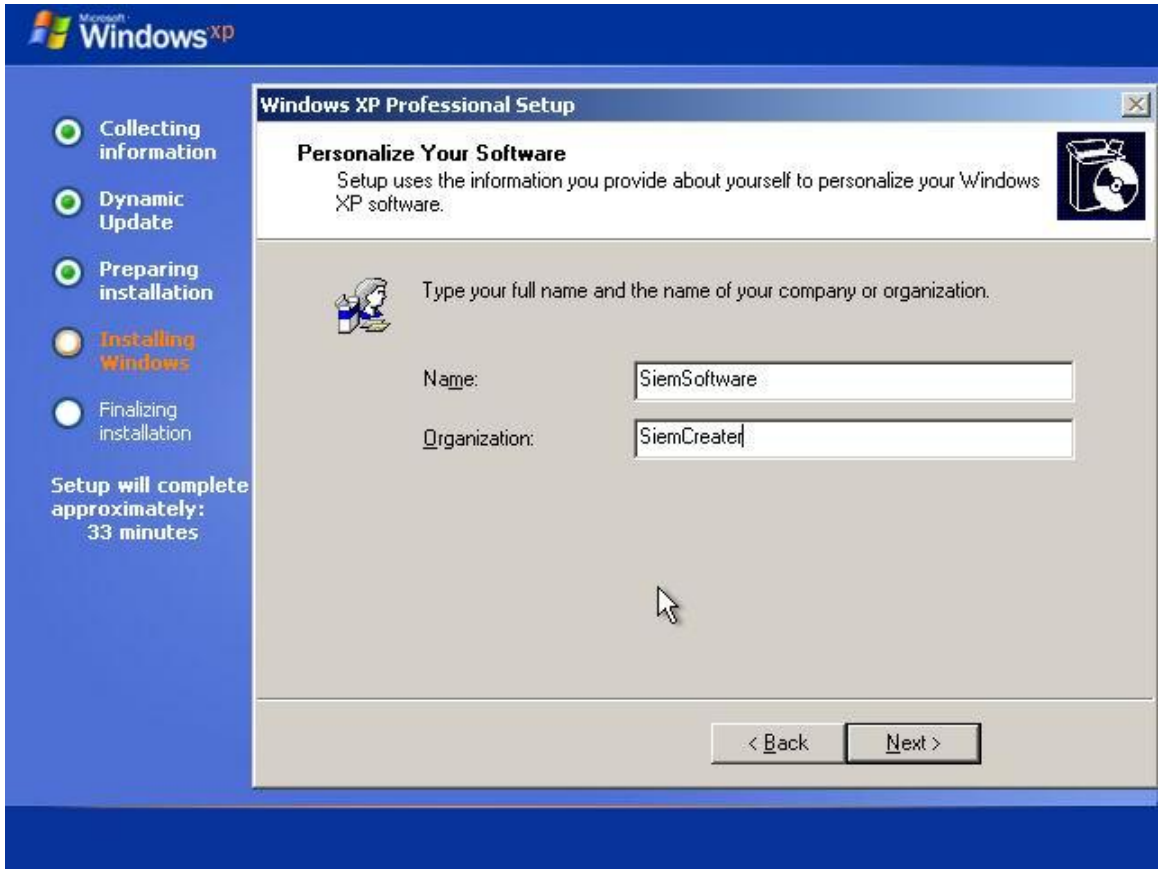


Chờ chương trình bắt đầu cài

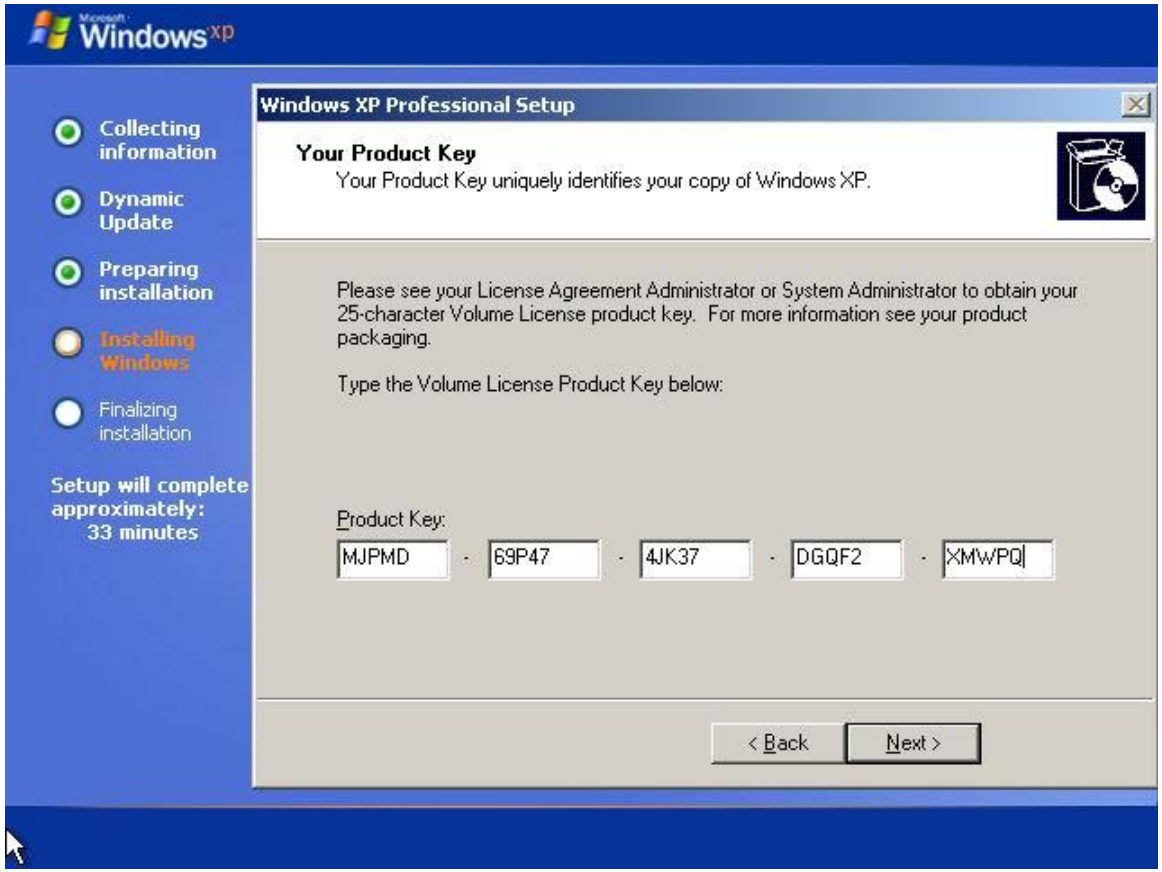
Tiếp tục chờ


Windows xp
Collecting information
Dynamic Update
Preparing installation
Installing Windows
Finalizing installation
Setup will complete in approximately:
37 minutes
Installing Devices
Easily move documents and personal settings to a new computer
If you've had a computer for a while, you've probably gotten it set up the way you like. Files are organized a certain way, you have bookmarked a long list of favorite Web sites, and you've customized your desktop just the way you want it.
Those personal settings won't be wiped out when you buy a new computer with Windows® XP. With the Files and Settings Transfer Wizard, it's simple to move your files and settings from one computer to another.
The wizard is located in the System Tools folder. For a command-line version that system administrators can use on multiple computers at once, try the User State Migration Tool (USMT) in the ValueAdd folder.

Màn hình sau xuất hiện nhập tên(name) và tổ chức



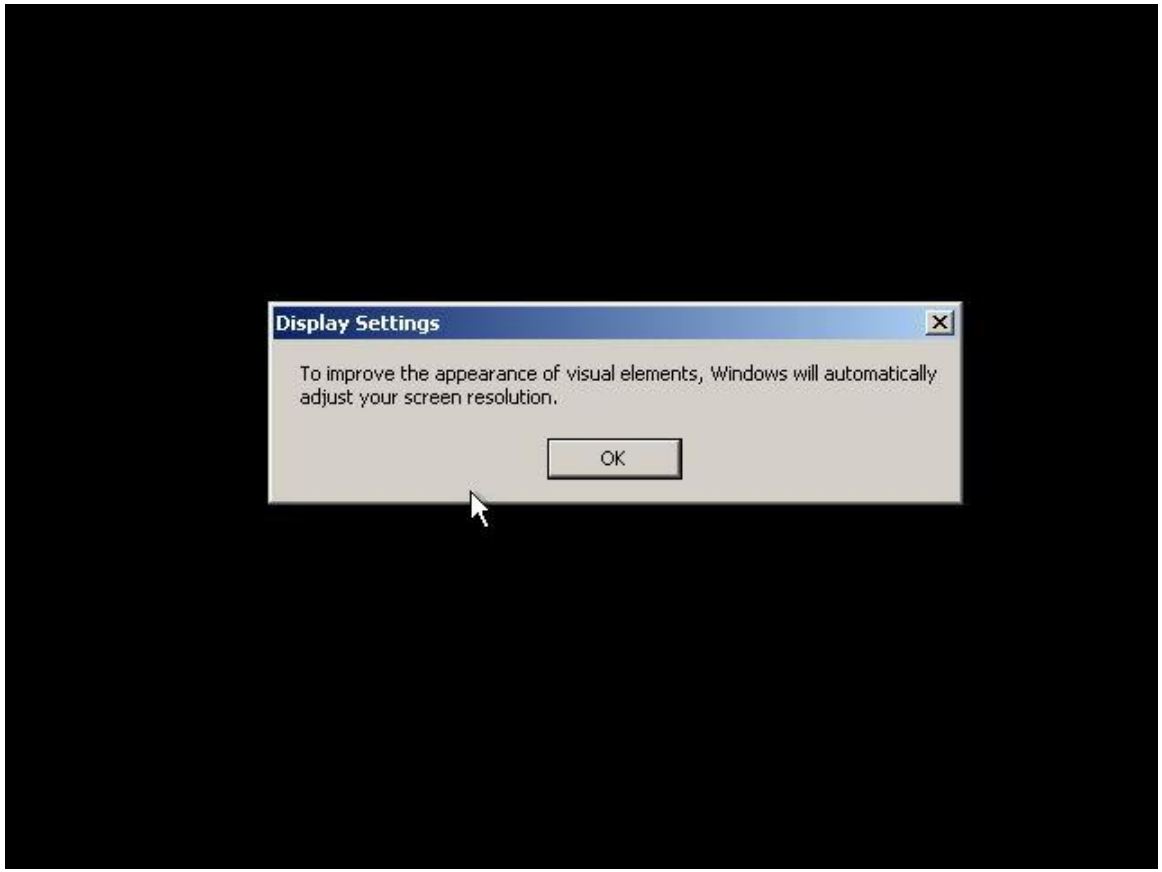
Nhập mã số (CD key)



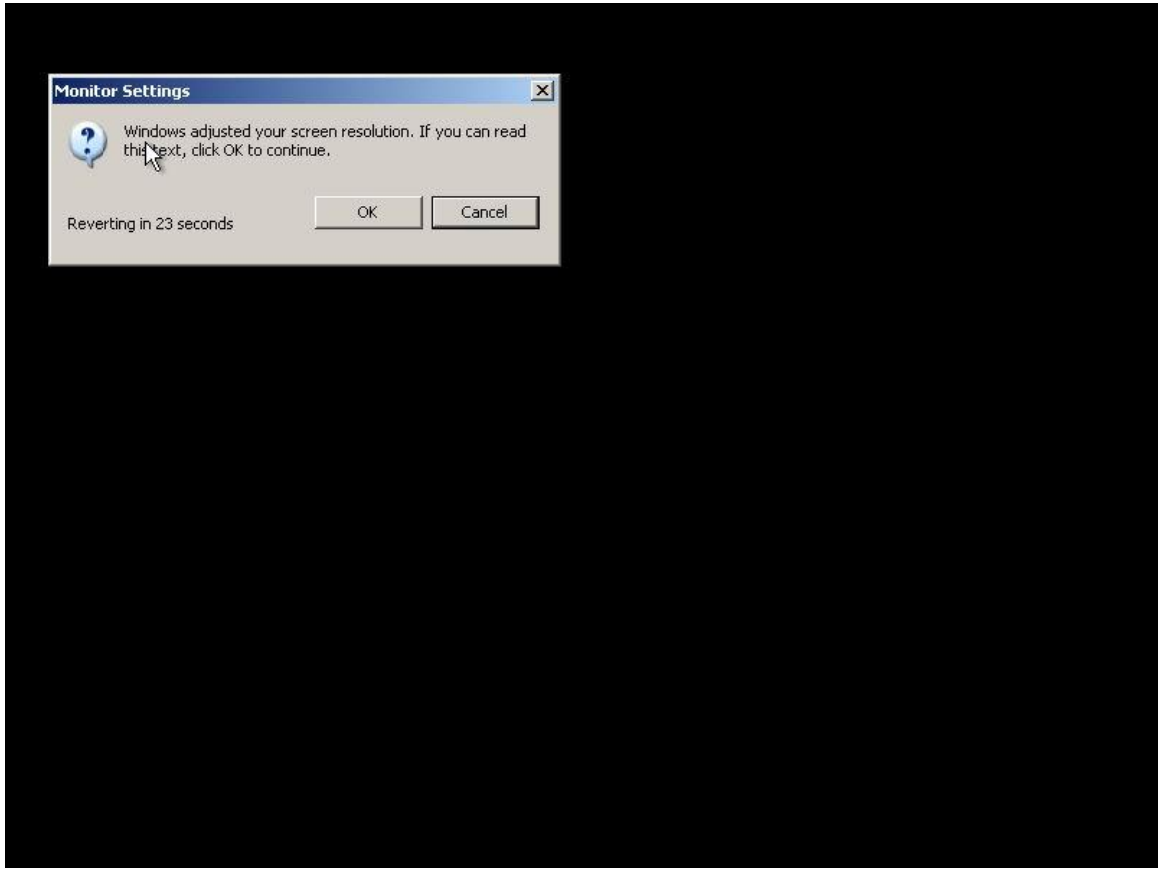
Nhập tên máy tính Password cho quản trị máy (administrator)

Chọn ngày giờ và múi giờ

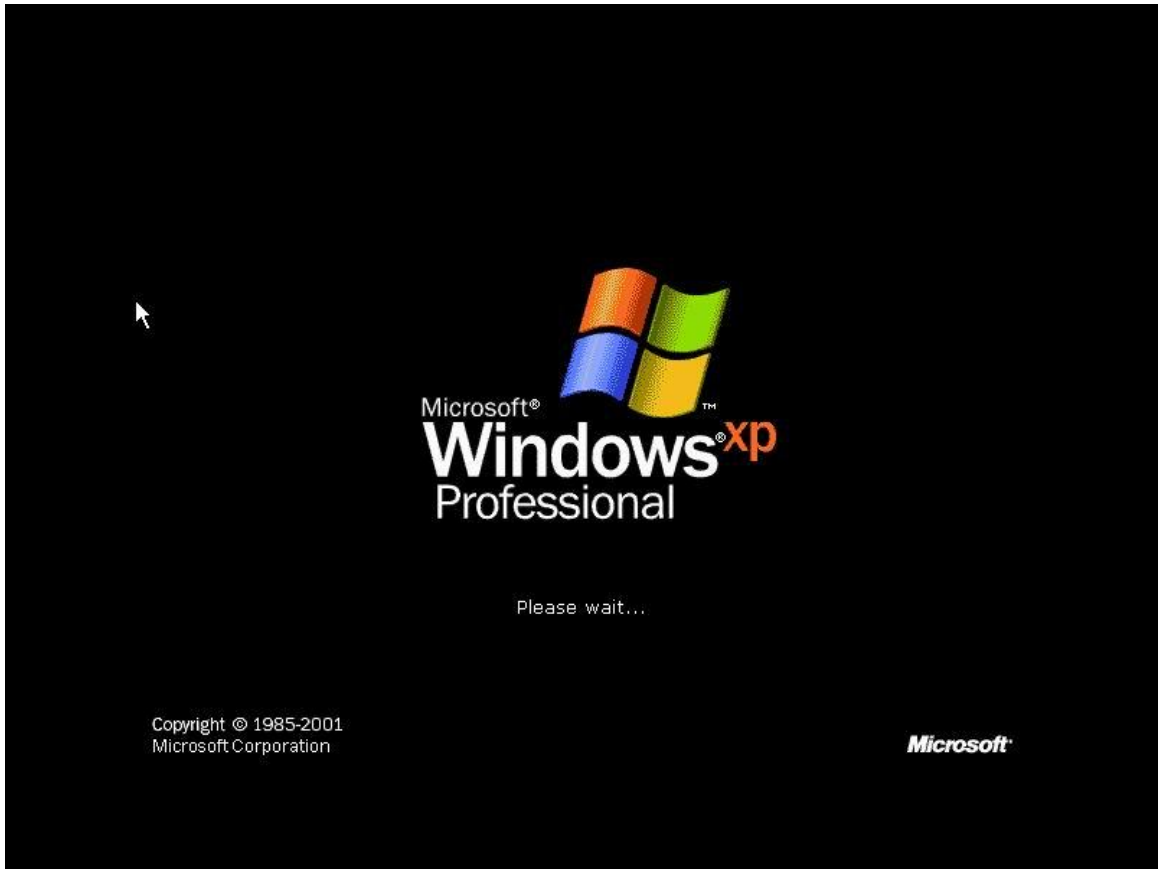
Windows nhận dạng màn hình nhấn OK (nếu có)



Nhận dạng độ phân giải màn hình



Màn hình xác nhận hoàn thành



Nhấn next để tiếp tục

Nhập tên các người dùng

Tên không trùng với tên máy

Hoàn thành cài đặt

Bài 2: LÀM QUEN VỚI WINDOWS

- Khái niệm hệ điều hành:
  - Là tập các chương trình cơ sở có nhiệm vụ điều khiển phần cứng máy tính.
  - Làm nền tảng cho các chương trình ứng dụng.
  - Tạo ra môi trường giao tiếp giữa người và máy.

Các hệ điều hành phổ biến

- DOS (Disk Operating System): Là HĐH đầu tiên của máy tính. Hiện nay không còn dùng phổ biến.
- Microsoft Windows: Là hệ điều hành phổ biến nhất hiện nay vì có ưu điểm dễ sử dụng. HĐH Windows có nhiều phiên bản như Windows 98, Windows 2000, Windows XP,…
- Linux: Là HĐH nguồn mở hoàn toàn miễn phí. Ưu điểm ổn định, nhược điểm khó sử dụng.

Các ưu điểm của HĐH Microsoft Windows

- Giao tiếp với người dùng thông qua **giao diện đồ hoạ.**

- Là **HĐH đa nhiệm**, có thể thực hiện nhiều trình ứng dụng song song cùng một lúc.

- Có thể lưu trữ tên tập tin dài đến **250 ký tự.**

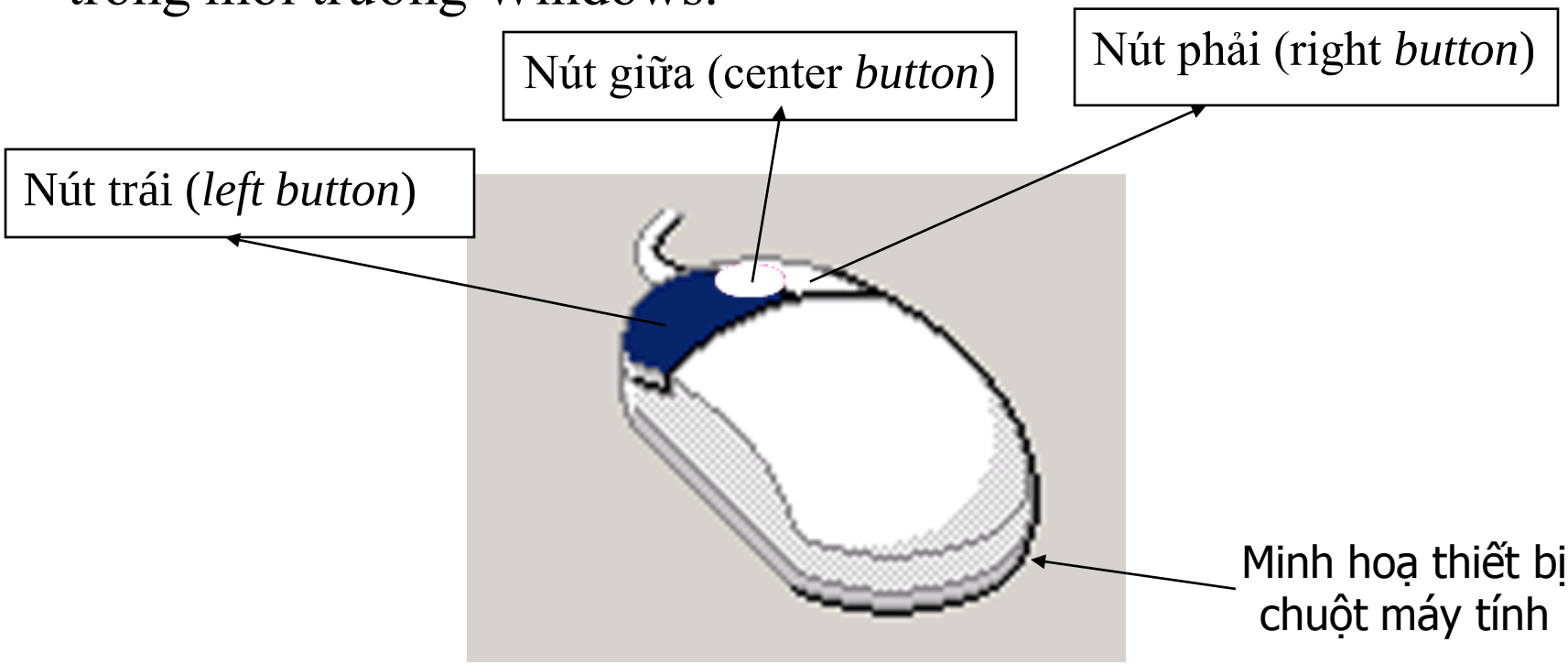
- Sử dụng cơ chế **tự động tìm và nhận thiết bị phần cứng**(**Plug and Play**).

- Cung cấp các khả năng có sẵn để nối mạng giữa các máy tính: chia sẻ tài nguyên, Email, Web, ...

- Hỗ trợ tốt thành phần **đa phương tiện (Multimedia).**

Sử dụng chuột:

Chuột (mouse): Được sử dụng nhiều nhất trong WINDOWS. Không có chuột có lẽ không thể dùng được các chương trình trong môi trường Windows.



Một số động tác hay dùng đối với chuột

Nhấp trái (Click trái): bấm nút trái chuột một lần.

Đng ể chọn ối tượng

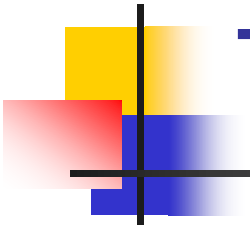
Nhấp phải(Click phải): bấm ỉt phải chuột một lần.

Dùng để chọn menu ngữ cảnh

Nhấp đúp (Double Click): Bấm nút trái chuột 2 lần liên tiếp.

Dùng để khởi động một ứng dụng.

Kéo - Rê (Drop - Drag): Nhấp và giữ nút (trái) chuột, di chuyển chuột đến vị trí nào đó và nhả nút chuột.



Thực hành chọn đối tượng

- Chọn 1 đối tượng
- Chọn nhiều đối tượng không liên tục
- Chọn 1 nhóm đối tượng liên tục

Màn hình Desktop:

Giao diện màn hình chính của Windows được gọi là "**Desktop**" bởi vì Màn hình nền Windows XP là cửa sổ đầu tiên của Hệ điều hành dành cho người sử dụng. Người dùng ra lệnh cho hệ điều hành bằng cách thao tác với biểu tượng.

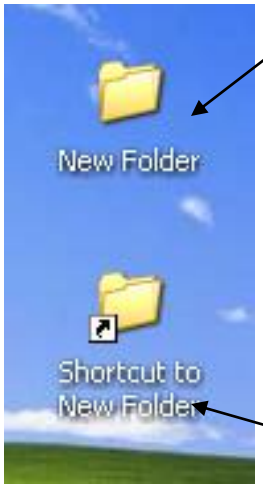

Nơi cất giữ tài liệu do người dùng tạo ra
My Documents
LAC VIET mtd2002-EVA
Nơi quản lý toàn bộ tài nguyên của máy tính như: ổ đĩa, thư mục, tập tin,...
My Computer
Pelles
My Network Places
Mạng máy tính: nơi truy xuất, chia sẻ tài nguyên của các máy khác trong mạng
Recycle Bin
Thùng rác: nơi lưu trữ những tập tin, thư mục bị xóa
Internet Explorer
Khay hệ thống
Nút Start: chứa các chương trình được cài đặt vào máy tính
Thanh tác vụ: chứa các chương trình đang được mở
start
minh
Adobe Acrobat Prof...
Microsoft PowerPoin...
8:45 PM

Nhận biết biểu tượng

Biểu tượng đặc biệt của hệ điều hành

- Gồm biểu tượng My Documents, My Computer, My Network Places, Recycle Bin, Internet Explorer.
- Khi cài đặt xong hệ điều hành thì các biểu tượng này được tạo ra ngay trên màn hình nền.

Nhận biết biểu tượng



Biểu tượng của thư mục

- Một thư mục được hệ điều hành Windows biểu diễn bằng một biểu tượng. Hình ảnh của biểu tượng là túi hồ sơ màu vàng như hình minh hoạ và tên biểu tượng xuất hiện bên dưới.

Biểu tượng lối tắt cho thư mục

- Tiếng Anh gọi là **Shortcut,** có hình ảnh là túi hồ sơ màu vàng có thêm mũi tên ở góc dưới bên trái.

Thực hành chọn một biểu tượng

Để chọn một biểu tượng chúng ta thao tác đơn giản là nhắp chuột lên biểu tượng đó.

Biểu tượng ở trạng thái tự do

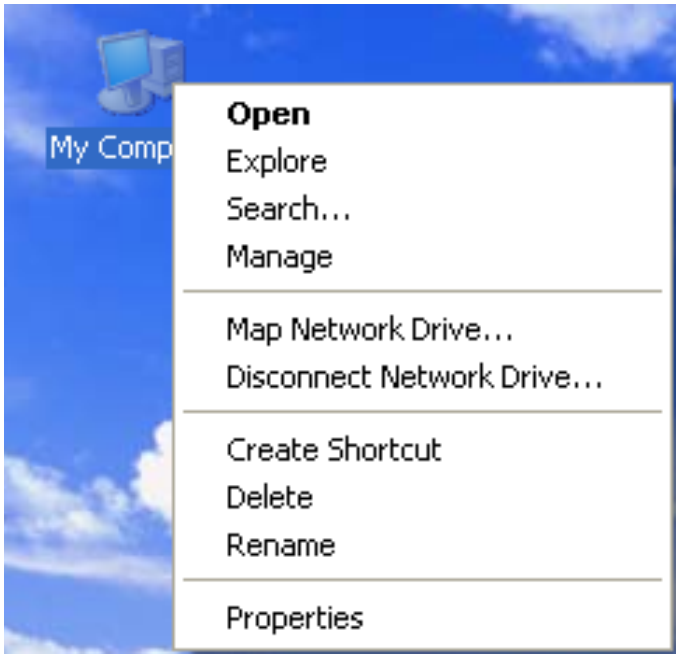
Biểu tượng ở trạng thái được chọn

Thực hành di chuyển biểu tượng

- Đầu tiên là chọn biểu tượng My Computer, kéo di chuột. Một hình ảnh của biểu tượng xuất hiện ở dạng nét mờ thể hiện vị trí di chuyển của biểu tượng.

Minh hoạ biểu tượng di chuyển

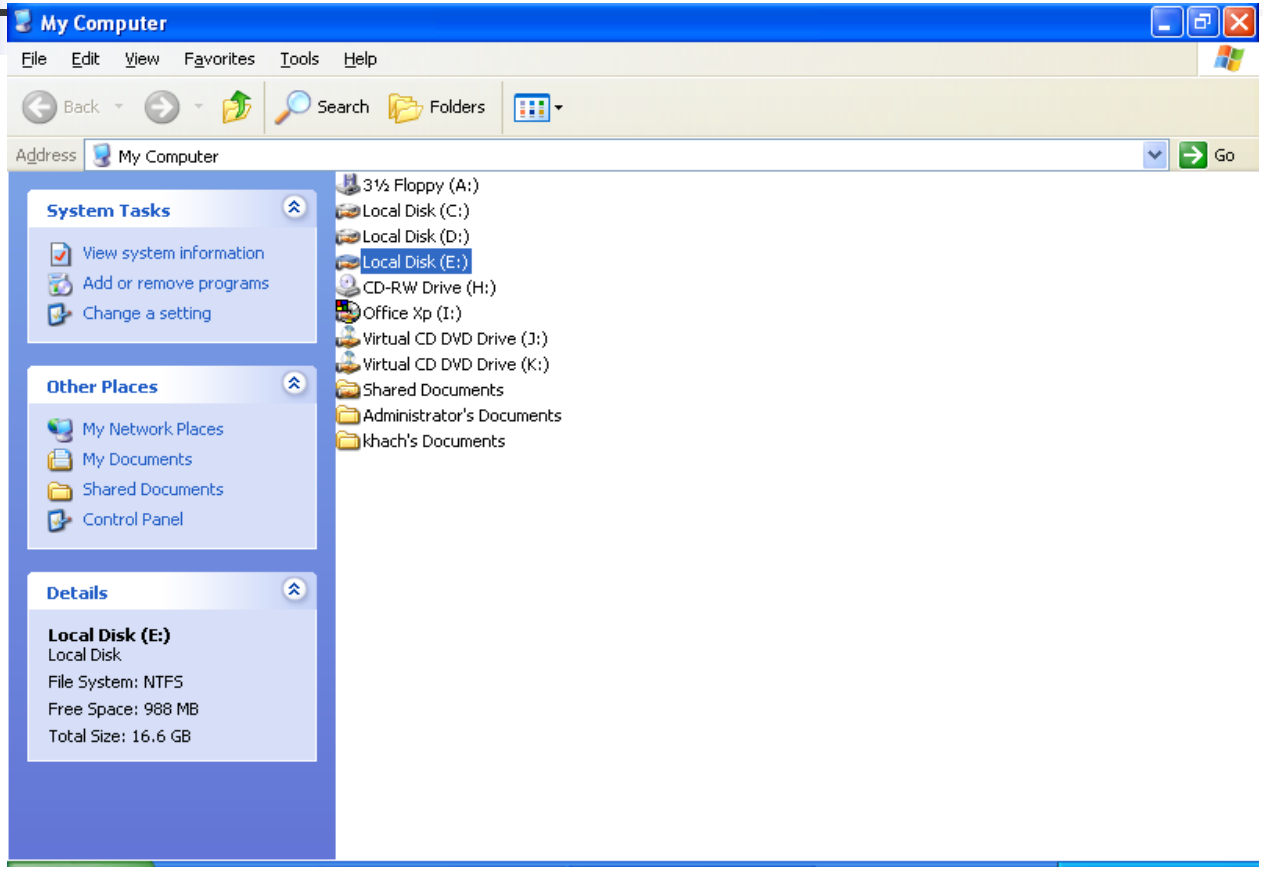
Thao tác với hộp lệnh (menu tắt)



Hộp lệnh xuất hiện sau khi nhắp chuột phải

- Đầu tiên là chọn biểu tượng, sau đó nhắp chuột phải trên vùng chọn sẽ làm xuất hiện hộp lệnh.
- Nhắp chuột trên mục lệnh của hộp lệnh có ý nghĩa là thi hành lệnh.

Thao tác với hộp lệnh



Cửa sổ My Computer xuất hiện sau khi thực hiện lệnh Open trên hộp lệnh

Xem thông tin ổ đĩa

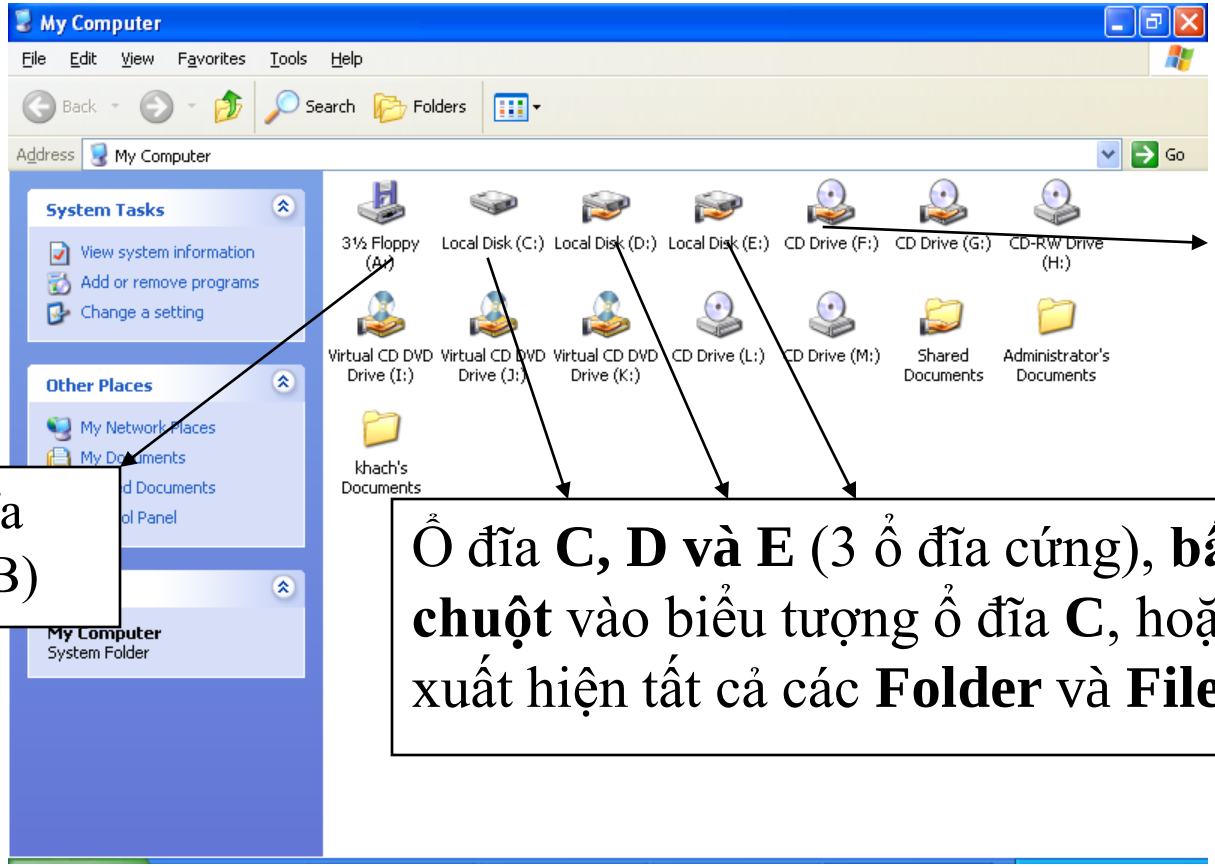
Từ màn hình Desktop, để nhìn thấy các ổ đĩa trên máy

+ Double Click vào **My Computer**: Đĩa mềm, ổ cứng, đĩa CD-ROM.

+ Double Click vào ổ đĩa, có thể duyệt qua các **File và Folder** từ một trong các ổ đĩa này.

Muốn trở về thư mục trước đó thì Click nút **Up**

Xem thông tin ổ đĩa



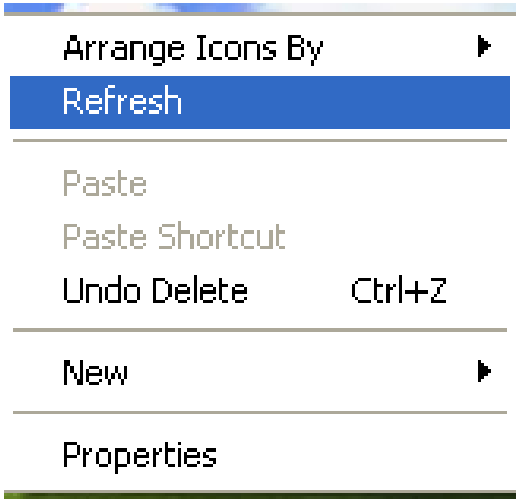
Ổ đĩa CD ROM(Đĩa Compact)

Ổ đĩa A (đĩa mềm 1.4 MB)

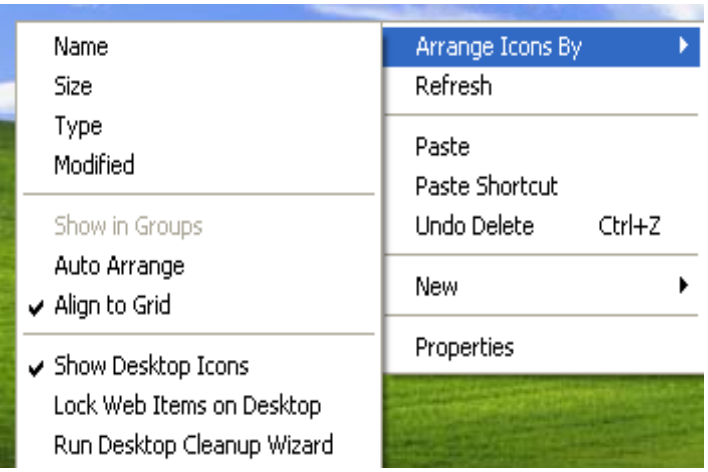
Ổ đĩa **C, D và E** (3 ổ đĩa cứng), **bấm đúp chuột** vào biểu tượng ổ đĩa **C**, hoặc **D** sẽ xuất hiện tất cả các **Folder** và **File**

Hộp lệnh của màn hình nền - Desktop

- Nhắp chuột phải vào chỗ trống trên màn hình nền sẽ làm xuất hiện hộp lệnh cho phép người sử dụng điều khiển màn hình.
- Chúng ta tìm hiểu ý nghĩa và thao tác từng lệnh sau:



Nhóm lệnh Arrange Icons



Mục lệnh Arrange Icons

Cho phép sắp xếp các đối tượng trong cửa sổ theo các mục:

- *By Name: sắp theo tên.*
- *By Type: sắp theo kiểu hay là phần mở rộng của tên tệp.*
- *By Size: sắp theo dung lượng nhớ.*
- *By Date: sắp theo ngày tháng khởi tạo/chỉnh sửa đối tượng.*
- *Auto Arrange: có nghĩa là tự động sắp xếp.*

Các Nhóm lệnh khác

Lệnh Line Up Icons:

- Có ý nghĩa là sắp xếp các biểu tượng trên màn hình nền có hàng có lối

Lệnh Refresh (gọi là làm tươi):

- Có ý nghĩa cập nhật thông tin mới nhất.

Lệnh Paste (gọi là dán):

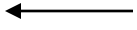
- Có ý nghĩa sao chép nội dung đã được tạo ảnh bằng lệnh Copy (gọi là sao chép) hay lệnh Cut (gọi là cắt) lên màn hình nền.

Các Nhóm lệnh khác

Lệnh Paste Shortcut (gọi là tạo nút bấm nhanh):

- Có ý nghĩa tạo nút bấm nhanh trên màn hình nền cho nội dung đã được tạo ảnh bằng lệnh Copy (gọi là sao chép) hay lệnh Cut.

Nhóm lệnh New:



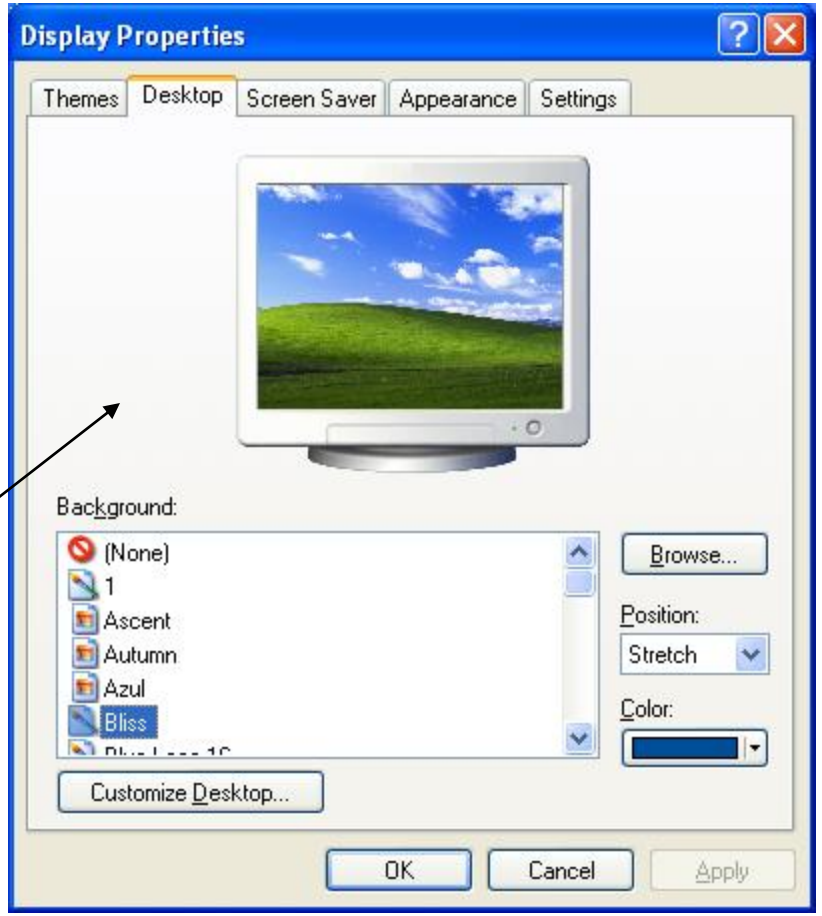
Mục lệnh New

Nhóm lệnh New

- lệnh Folder: cho phép tạo thư mục mới.
- lệnh Shortcut: cho phép tạo nút bấm nhanh trên màn hình nền cho các đối tượng bất kỳ như tệp tin, thư mục,...

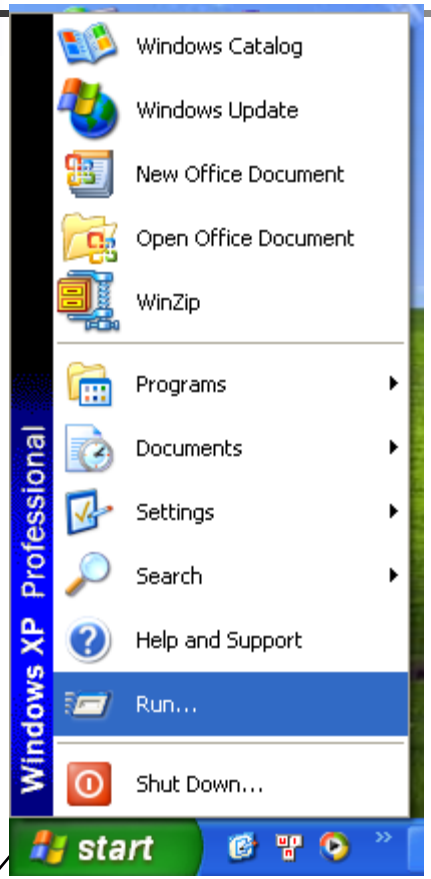
Lệnh Properties

- Cho phép mở hộp thoại để xem và chỉnh sửa các thuộc tính của màn hình nền.



Hộp thoại cho phép chỉnh sửa các thuộc tính của màn hình

Thanh menu Start

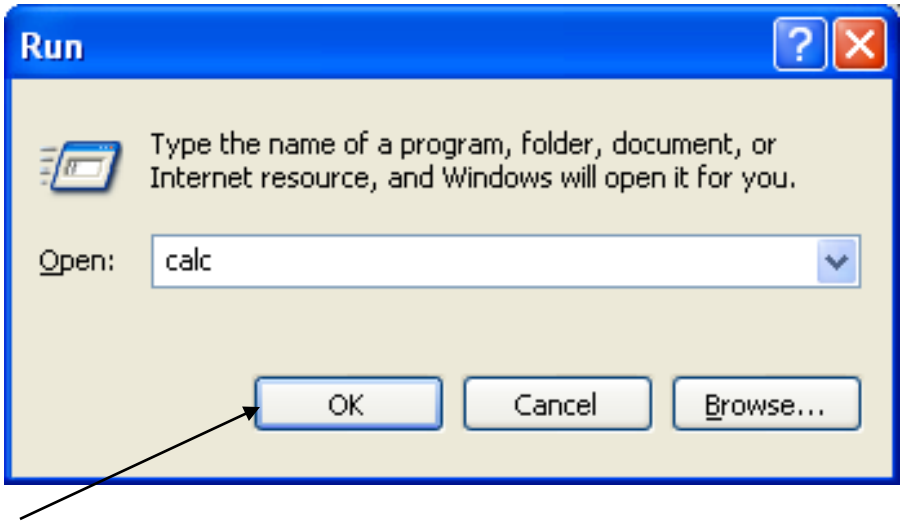


Nhắp chuột vào nút Start làm xuất hiện hộp danh mục chọn

- Nhắp chuột vào nút Start có mặt trên màn hình nền sẽ làm xuất hiện hộp danh mục chọn được phân chia theo chủ đề cho phép người sử dụng dễ dàng ra lệnh cho máy tính.

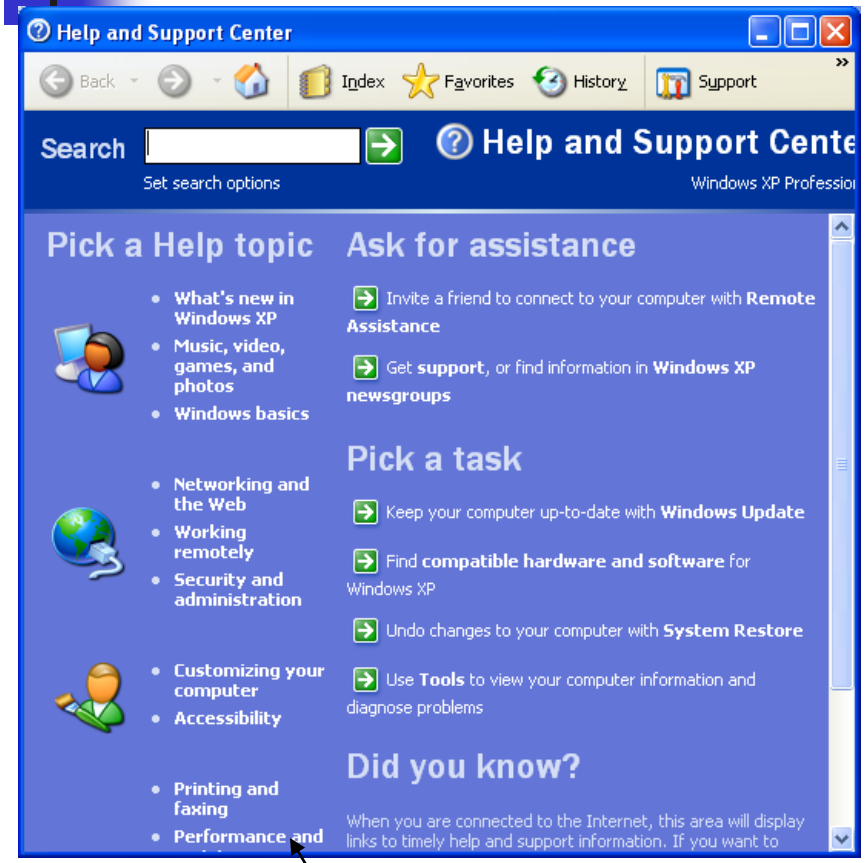
Mục Run - cho phép gõ lệnh

- Nhắp chuột vào nút Start, chọn mục Run làm xuất hiện hộp thoại nhập lệnh. Ví dụ gõ lệnh "calc" để mở bảng tính số học.



Nhập lệnh vào ô nhập Open, sau đó nhấn nút OK

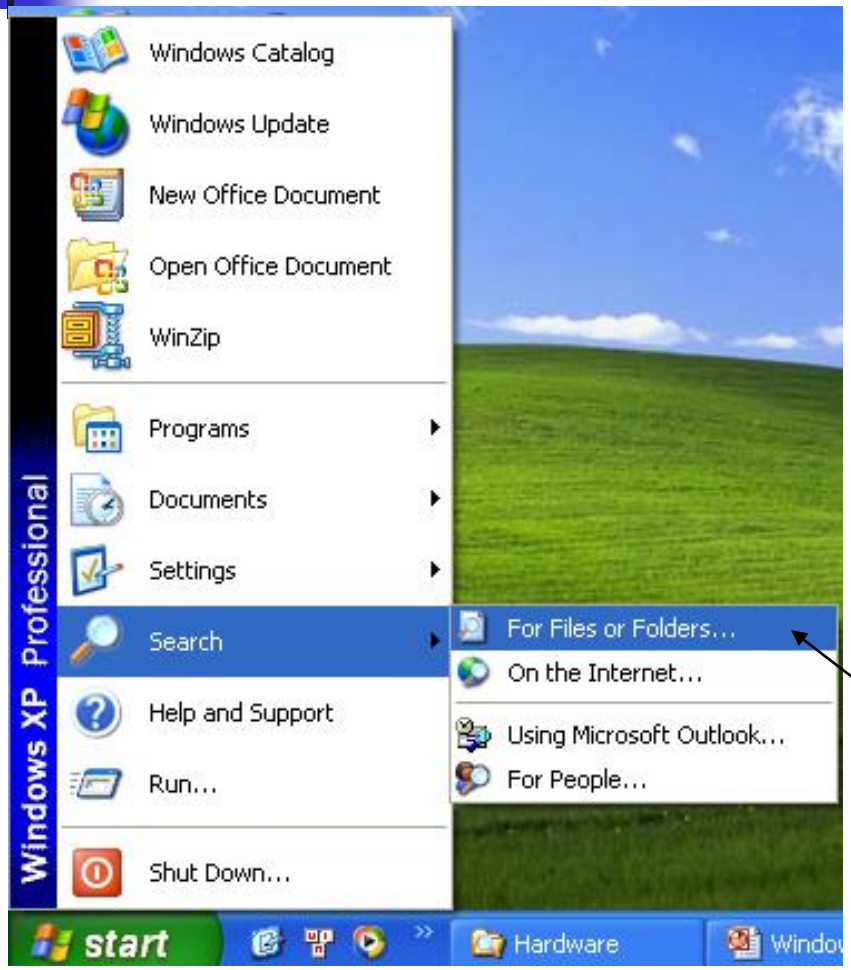
Mục Help - mở phần trợ giúp



Hộp thoại Help

- Phần trợ giúp rất cần thiết khi sử dụng Windows nhưng đòi hỏi người sử dụng phải biết tiếng Anh. Để sử dụng phần Trợ giúp, nhắp chuột vào nút **Start,** sau đó chọn mục **Help and Support** để mở hộp thoại Help.

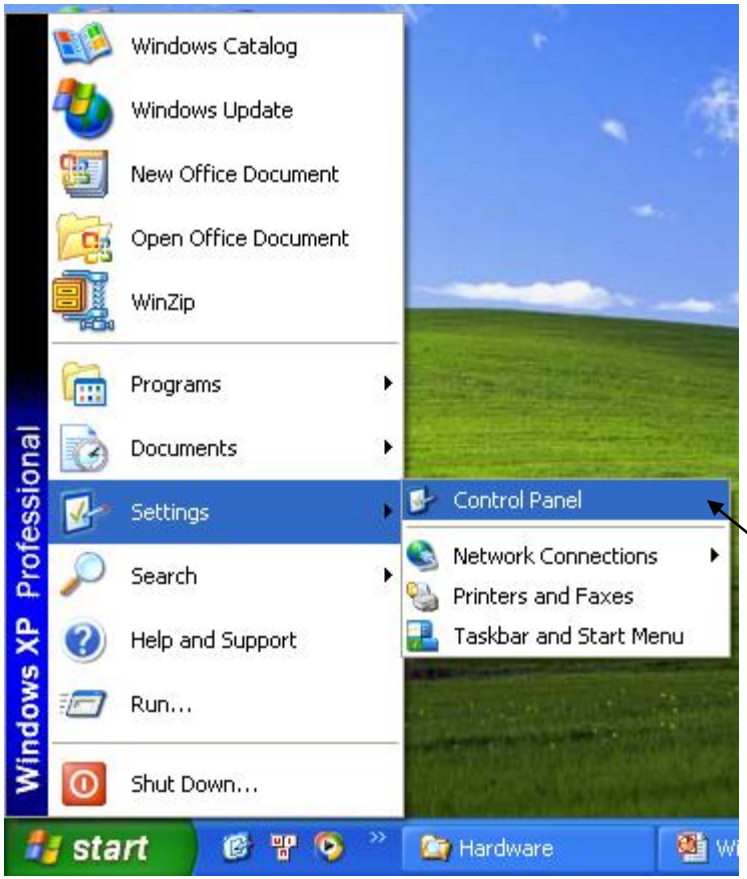
Mục Search - hỗ trợ tìm kiếm thông tin trên máy



- Nhắp chọn nút **Start → Search → For Files or Folders** làm xuất hiện hộp thoại hỗ trợ tìm kiếm.

Chọn Search – For Files or Folders

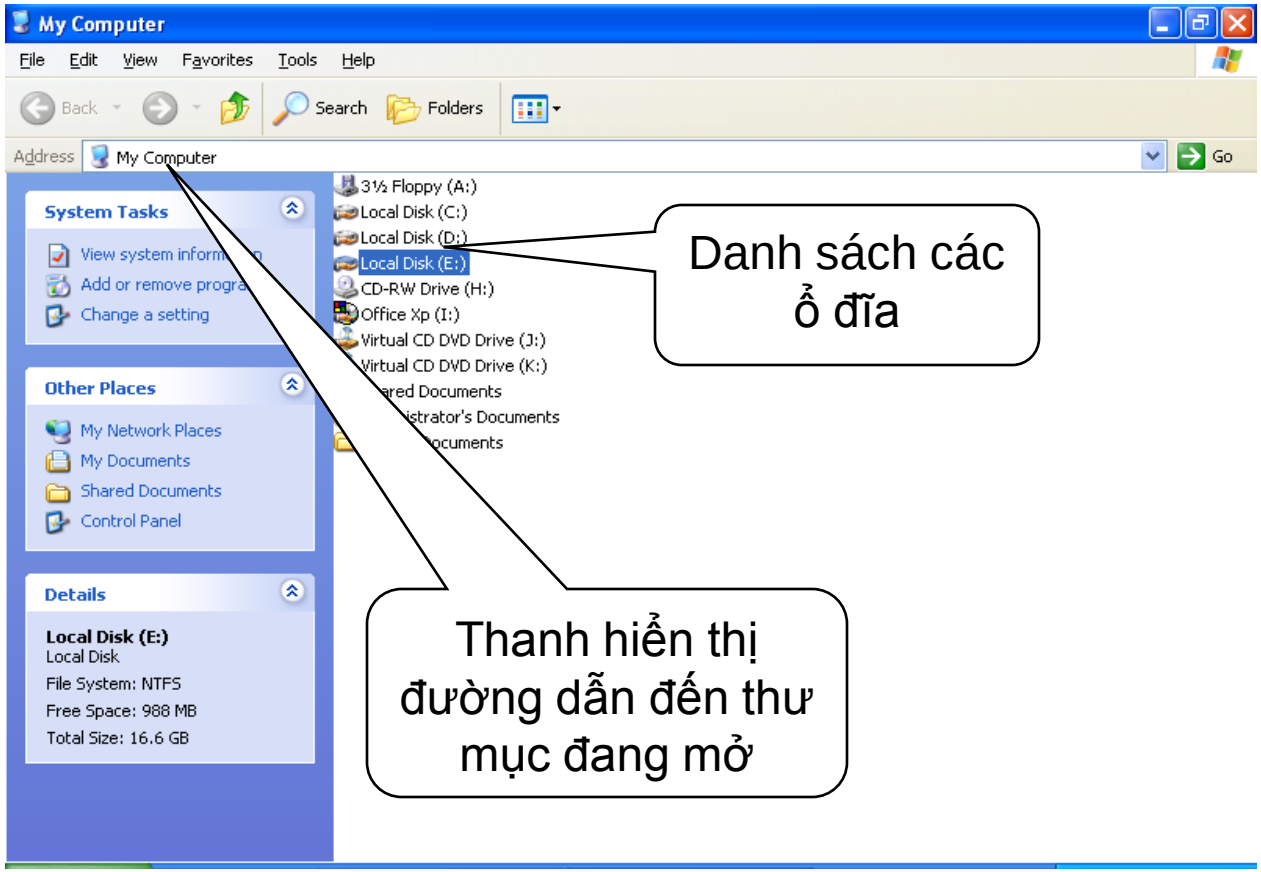
Mục Settings - hỗ trợ quản trị hệ thống



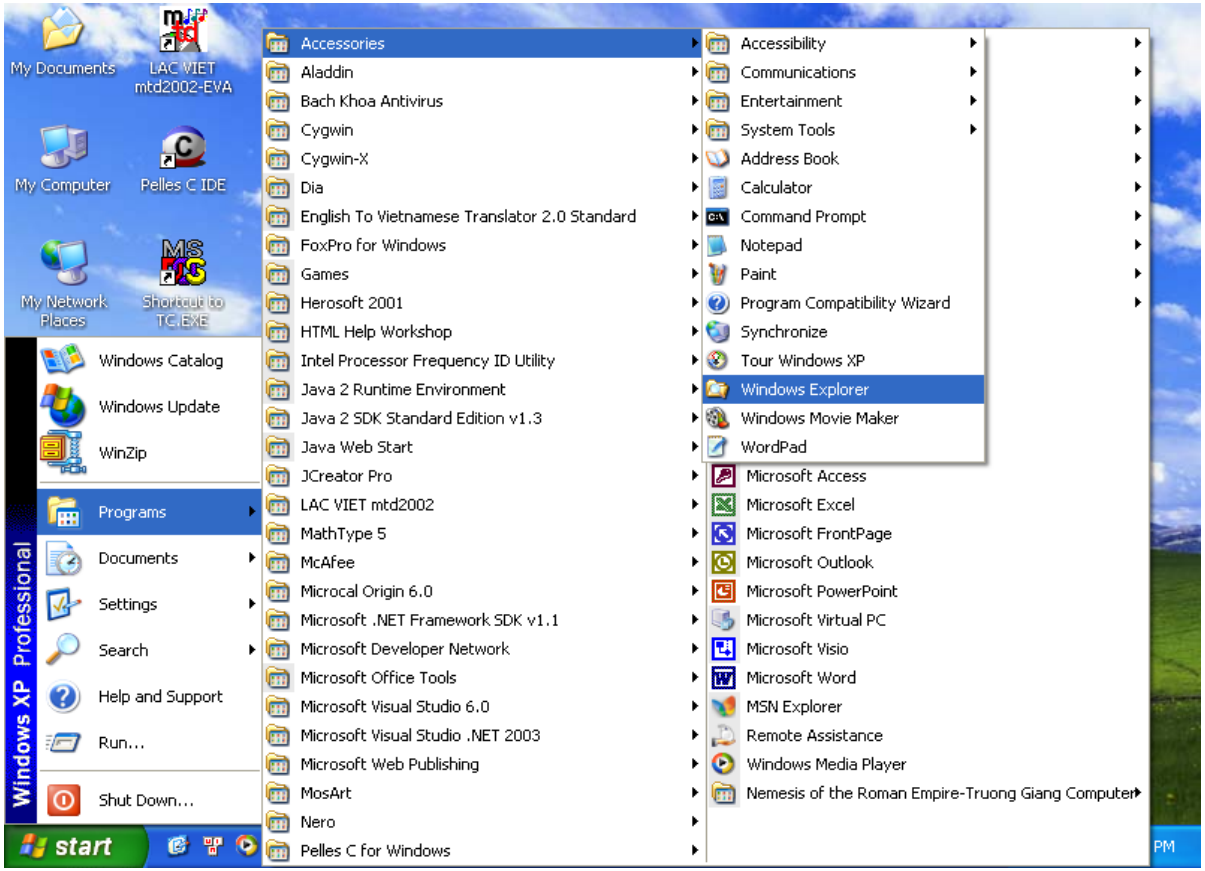
- Nhắp chuột vào nút **Start** →**Settings**, sau đó chọn các mục con như Control Panel, Printers...

Chọn Settings-Control Panel

Cửa sổ **My Computer**

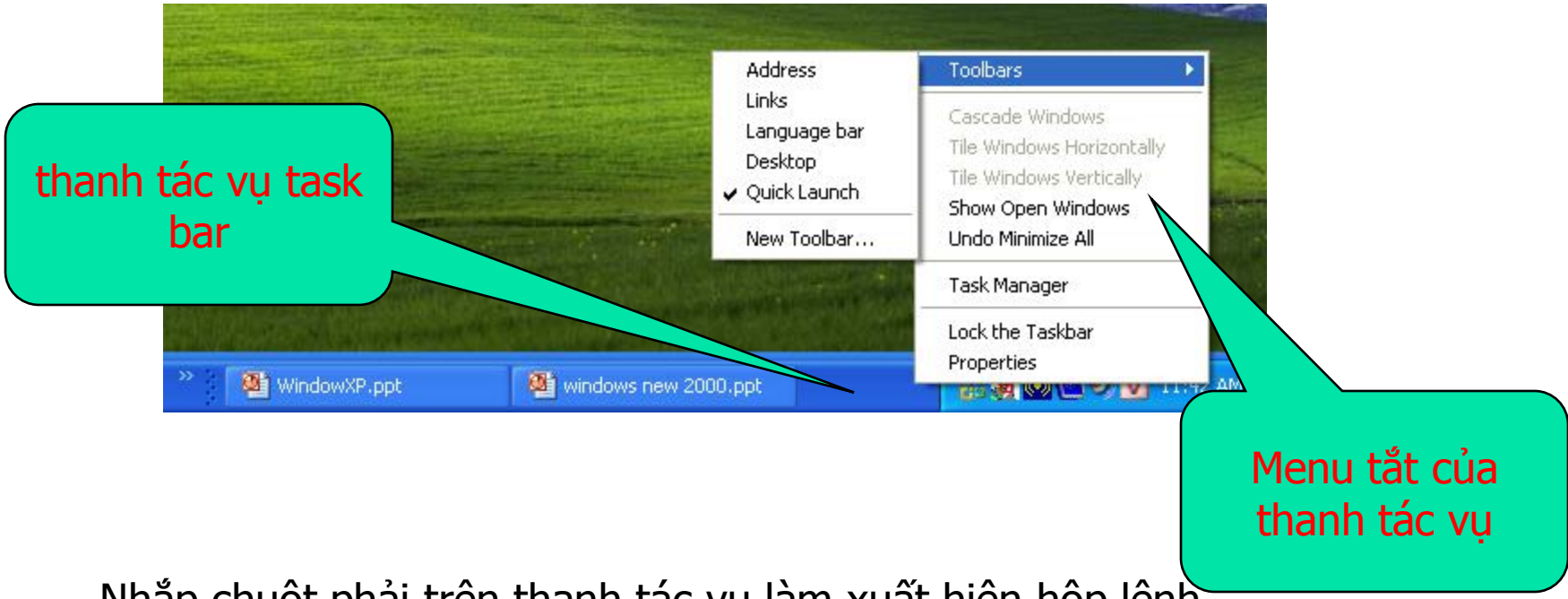


Các chương trình nằm trong mục Programs



Menu tắt của thanh tác vụ

- Nhắp chuột phải trên thanh tác vụ làm xuất hiện hộp lệnh.

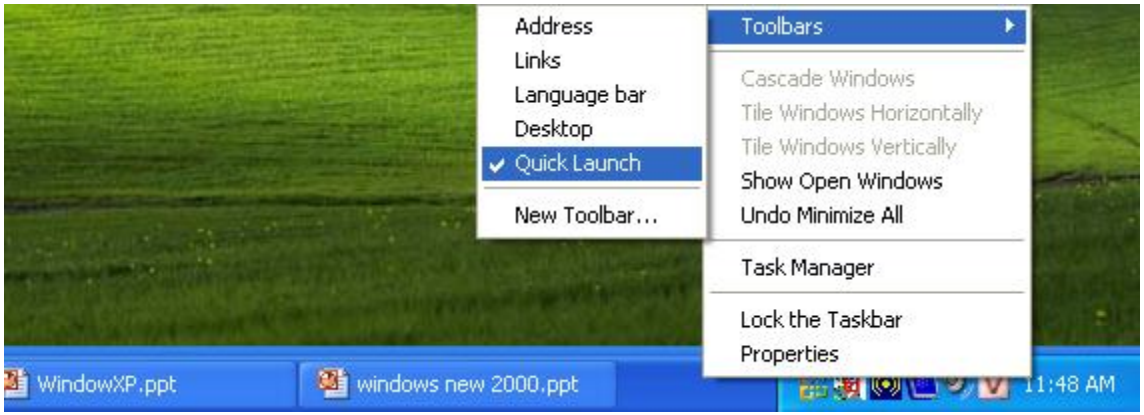


Nhắp chuột phải trên thanh tác vụ làm xuất hiện hộp lệnh

Ý nghĩa các lệnh như sau

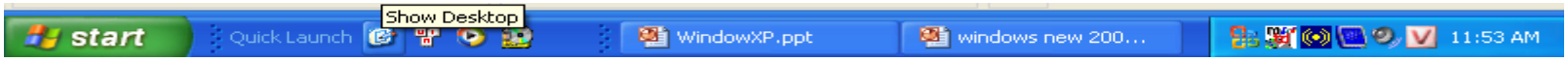
Lệnh Toolbars:

- Trên thanh tác vụ có thể mở nhiều thanh công cụ để làm việc. Có thể mở hoặc đóng bớt các thanh công cụ qua nhóm lệnh Toolbars.

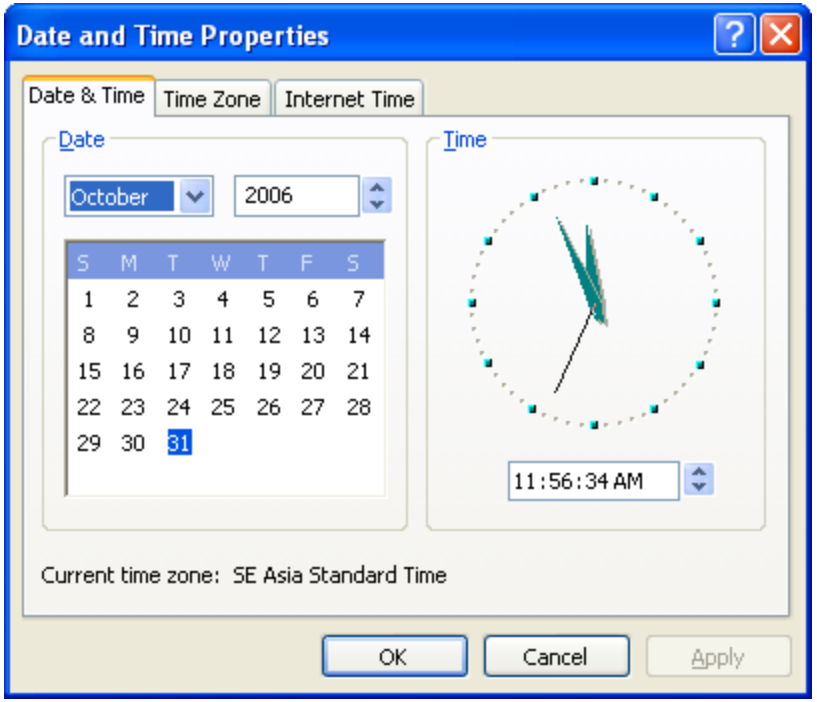


Ý nghĩa các lệnh như sau

- Mặc dù có thể chọn nhiều thanh công cụ đặt trên thanh tác vụ nhưng chúng ta chỉ nên chọn một thanh công cụ đó là thanh Quick Launch.
- Thanh công cụ Quick Launch là một tiện ích không thể bỏ qua đối với người sử dụng. chúng ta nên đặt các nút lối tắt ở đây để tiện sử dụng.
- Trên Quick Launch thường có đặt nút Show Desktop (hình minh hoạ) đây là nút đưa chúng ta nhanh chóng trở về màn hình nền.



Ý nghĩa các lệnh như sau



Hộp thoại cho phép chỉnh sửa thời gian hệ thống

Lệnh Adjust Date/Time:

- Cho phép mở hộp thoại Date/Time Properties để hiệu chỉnh đồng hồ máy tính.
- Thẻ Date&Time cho phép chỉnh sửa ngày/tháng/năm và giờ. Thẻ Time
- Zone cho phép chỉnh múi giờ đúng theo múi giờ của Việt Nam.

Ý nghĩa các lệnh như sau

Lệnh Cascade Windows:

- cho phép sắp xếp các cửa sổ đang mở theo dạng xếp mái ngói.

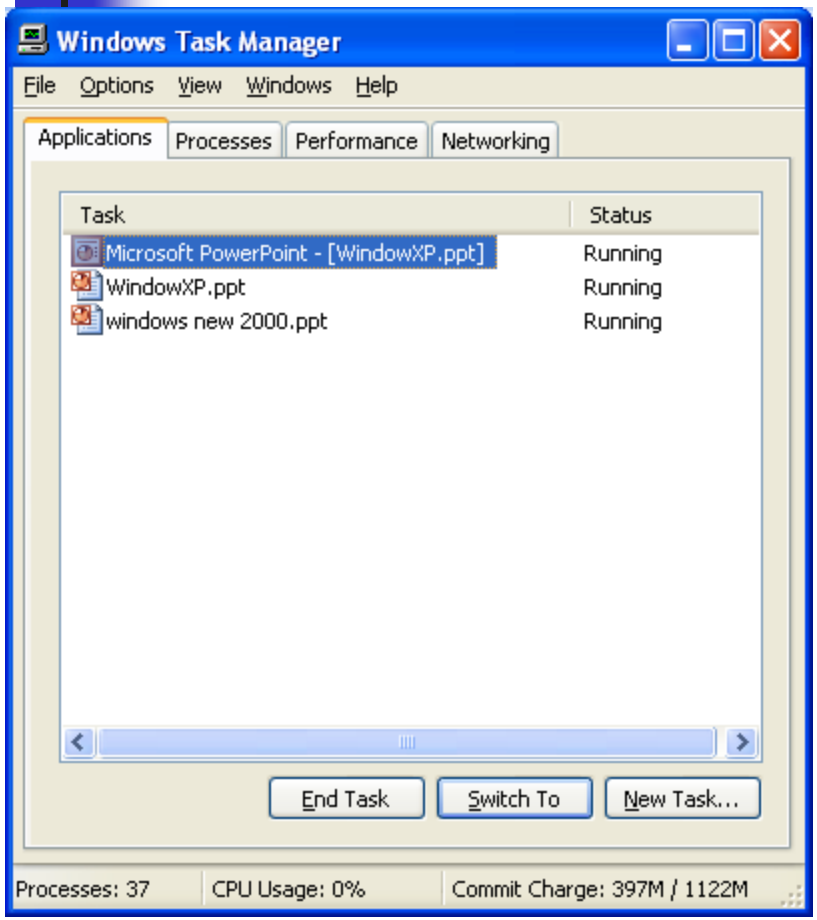
Lệnh Tile Windows Horizontally và Tile Windows Verticaly:

- cho phép xếp các cửa sổ dàn ngang trên màn hình nền, không có cửa sổ bị che lấp.

Lệnh Minimize All Windows:

- cho phép thu nhỏ tất cả các cửa sổ đang mở cùng một lần.

Ý nghĩa các lệnh như sau



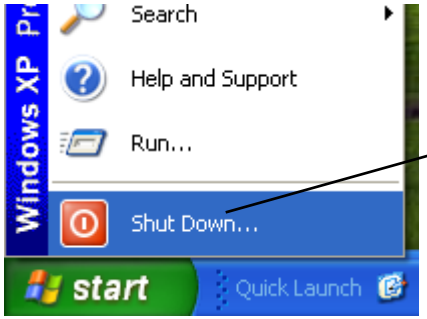
Hộp thoại Windows Task Manager

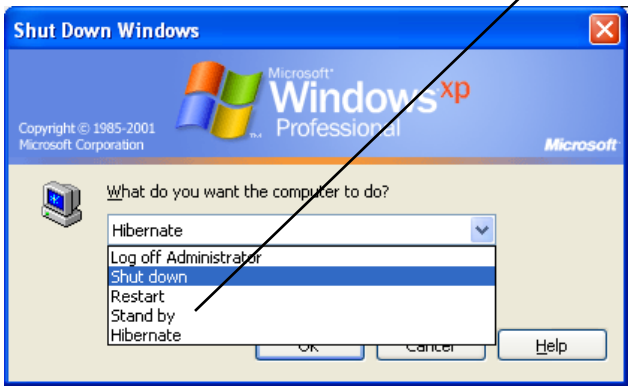
Lệnh Task Manager:

- **cho phép mở cửa sổ quản lý chương trình -Windows Task Manager.**
- **Trong thẻ Applications của cửa sổ này, người sử dụng có thể chọn một chương trình hay nhiều chương trình và ra lệnh đóng chương trình bằng cách nhấn nút End Task. Đây là chức năng rất tiện ích cho việc đóng những chương trình đang gây tắc nghẽn hệ thống.**

Khởi động lại và tắt máy tính

Để tắt máy đúng cách ta làm theo trình tự sau:





1. **Đóng bất kỳ chương trình nào đang mở**, lưu tất cả các tài liệu nếu cần thiết

2. Bấm chuột vào nút **Start** \ chọn **Shut Down...**

3. Chọn **Shut Down** (nếu muốn tắt máy)
Restart (nếu bạn muốn khởi động lại máy)
Log off Username (nếu bạn muốn thoát khỏi phiên hiện hành và để đăng nhập vào hệ thống với một tên Username khác mà không muốn khởi động lại máy)
Stand By (nếu bạn muốn cho máy ở chế độ ngủ đông)

4. Bấm **OK** để chấp nhận

5.. **Tắt nguồn điện** dẫn vào màn hình và máy tính.

Tắt máy tính theo kiểu áp đặt

Các bộ phận chính của một máy PC

- Tắt nguồn điện bằng cách bấm nút **POWER** trên hộp máy (có thể phải giữ tay trên nút khoảng 30 giây) hoặc
- Nhấn nút **Reset** trên hộp máy.

Khởi động lại máy

Trong trường hợp máy tính không còn điều khiển được bằng bàn phím và chuột thì:

- **Nhắp tổ hợp phím Ctrl + Alt + Del, hoặc từ cửa sổ màn hình nền, nhắp chọn nút Start → Shutdown làm xuất hiện hộp thoại Shut Down Windows. Trong ô chọn, chọn mục Restart.**

Tóm tắt : Học viên cần nắm

1. Tắt và khởi động máy đúng cách.
2. Nhận diện và Sử dụng các công cụ trên màn hình Desktop, sắp xếp các đối tượng.
3. Các thao tác khi sử dụng chuột.

Bài 3: Làm quen với tập tin và thư mục

Mục tiêu bài học:

- Học viên nắm vững khái niệm về tệp tin và thư mục, cách tổ chức và quản lý chúng trong Hệ điều hành Windows.
- Thành thạo việc chọn nhóm biểu tượng liền kề và rời rạc.
- Thực hiện các thao tác cơ bản với tệp tin và thư mục trên màn hình nền như: tạo, chỉnh sửa, mở nội dung, di chuyển, và xoá.

Ổ đĩa vật lý và ổ đĩa Logic

- Các ổ đĩa cứng có dung lượng nhớ rất lớn nên hệ điều hành có chức năng chia nhỏ ổ đĩa cứng thành các ổ đĩa gọi là ổ đĩa cứng logic để người sử dụng có thể tiện sử dụng.
- Các ổ đĩa được HĐH gán bằng các chữ cái như sau:
  - A:, B: Đĩa mềm
  - C:, D:, … Đĩa cứng
  - E:, F:, … Đĩa CDROM

Tập tin (File)

- Trong hệ điều hành Windows, tệp tin là đối tượng chứa dữ liệu. Ví dụ các văn bản sau khi nhập vào máy được lưu thành các tệp tin để sau đó có mở ra xem lại, chỉnh sửa/in ấn và có thể xoá đi.
- Cấu trúc của tập tin gồm 2 phần:
- **Tên tập tin.*phần mở rộng***

  VD: baitho.*doc*

Đặc trưng của tập tin

- Độ dài của tên tập tin không quá 8 ký tự (đối với DOS) và không quá 250 ký tự (đối với Windows 98 trở lên).
- Kiểu tập tin phụ thuộc vào phần mở rộng của tập tin (VD: **baitap.doc** – có phần mở rộng là **.doc** cho nên đây là tập tin **word**).

Một số kiểu tập tin thông dụng

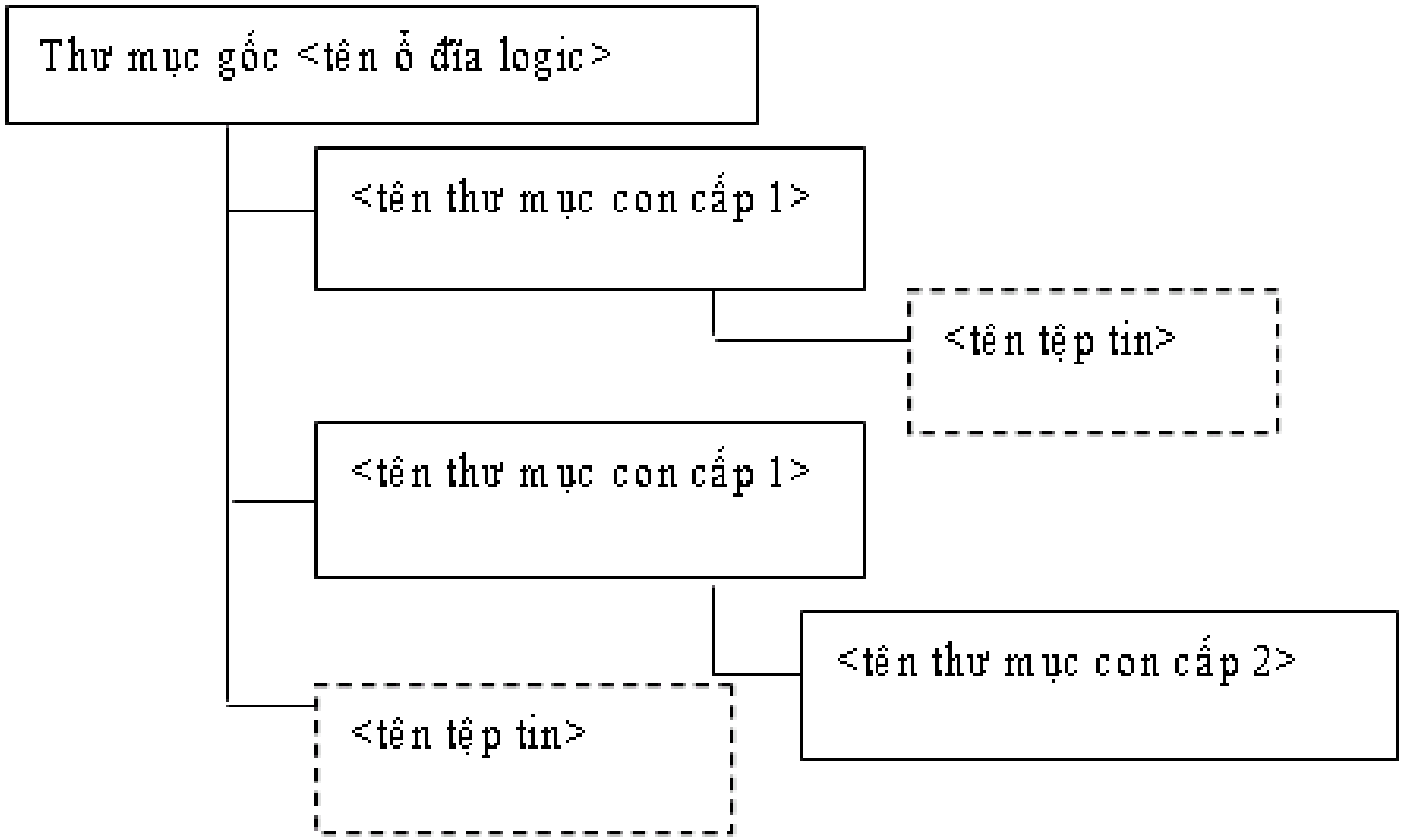
- **.doc, .txt, .rtf**: Các tập tin văn bản Word
- **.xls** : Các tập tin bảng tính Excel
- **.exe, .bat**: Các tập tin chương trình
- **.com** : tập tin lệnh
- **.gif, .jpeg, .bmp**: Các tập tin chứa hình ảnh
- **.mp3, .dat, . Wav**: Các tập tin âm thanh, video
- **html, htm**: Các tập tin siêu văn bản
- **sql, mdb**: Các tập tin chứa cơ sở dữ liệu

……

Thư mục - Folder

- Để lưu giữ, sắp xếp các tập tin thành một hệ thống phân cấp có tính chặt chẽ và tiện dụng khi tìm kiếm, hệ điều hành Windows cho phép người sử dụng xây dựng cây thư mục theo cách thức:
- Các đặc trưng của thư mục:
  - Ổ đĩa logic của máy tính được xác định là thư mục gốc
  - Có thể tạo nhiều thư mục con trong thư mục
  - Các thư mục cùng cấp không được trùng tên
  - Tập tin phải được chứa trong một thư mục

Thư mục - Folder



Hệ thống cây thư mục của HĐH Windows

Đường dẫn cho tập tin

- Để diễn tả vị trí của tệp tin trong hệ thống thư mục chúng ta cần viết đường dẫn theo cách sau:
- [tên qui ước đĩa logic:] [\] [<tên thư mục> \ ...\ <tên thư mục> \ <tên tệp tin>]

VD: Đường dẫn *C:|congvan2004|danhsachCB1.doc*

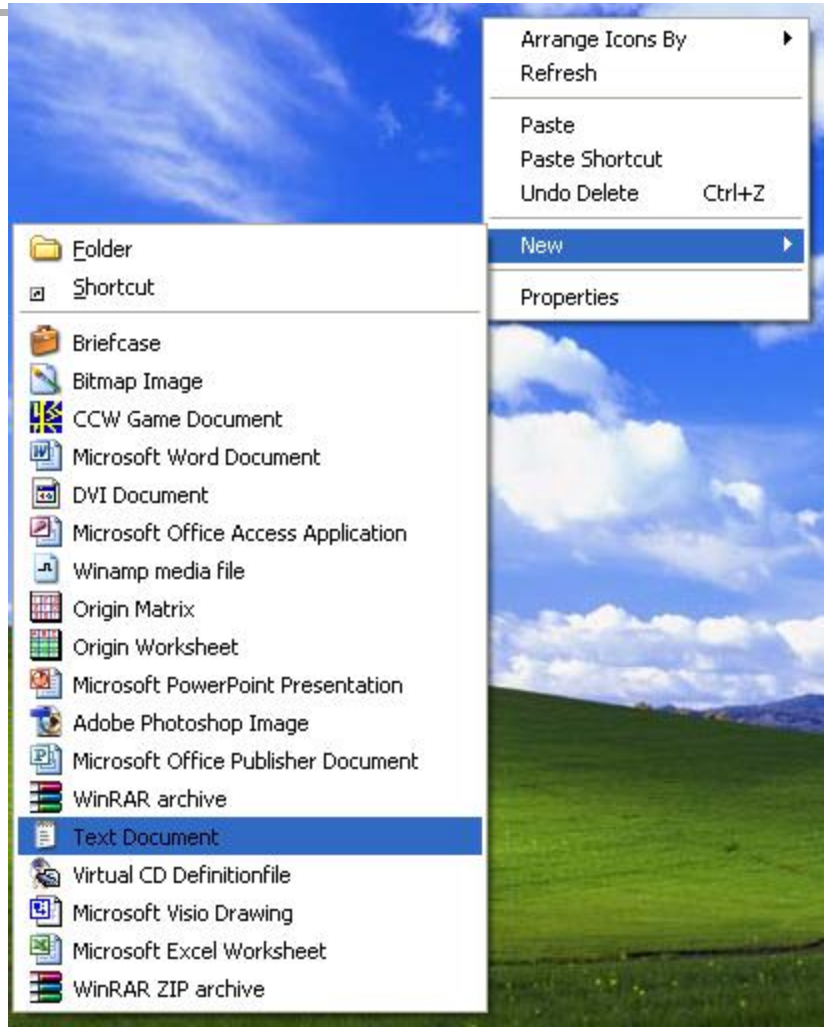
- Chỉ ra tập tin *danhsachCB1.doc* đang được chứa trong thư mục *congvan2004* thuộc đĩa C.

Tạo mới một tập tin

- **Nhắp chuột phải trên màn hình nền làm xuất hiện hộp lệnh.**
- **Lựa chọn loại tập tin muốn tạo** (Vd: Chọn mục **New →Text Document** (Tập tin được tạo ra chỉ là một tập tin rỗng chưa có nội dung gì bên trong).
- hoặc **Microsoft Word Document** - Tập tin văn bản, Microsoft Excel Worksheet - bảng tính Excel,...).

Tạo mới một tập tin

- Minh hoạ các mục chọn trên hộp lệnh để tạo tệp tin kiểu Text



Tạo mới một thư mục

Ba thư mục mới và hai tệp tin được tạo

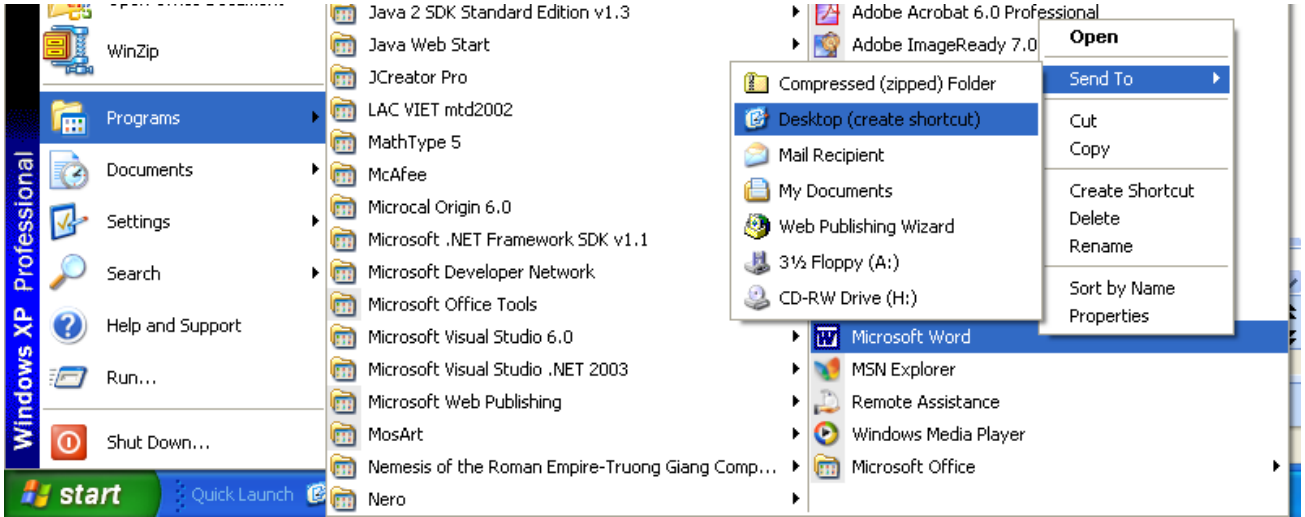
- Nhắp chuột phải trên màn hình nền làm xuất hiện hộp lệnh.
- Chọn mục **New** →**Folder**.
- **Đặt tên thư mục** mà bạn vừa mới tạo (thư mục mới có tên mặc định là **New Folder**)

Tạo biểu tượng lối tắt

Shortcut (lệnh tắt): Là một biểu tượng lối tắt được đặt trên **desktop** cho phép bạn chạy các ứng dụng một cách dễ dàng hơn. (*tạo 1 lần và sử dụng nhiều lần*).

Tạo một Shortcut:

- **Nhắp nút Start, chọn mục Program→Microsoft Word.**
- **Nhắp chuột phải tại mục Calculator để mở hộp lệnh. Chọn mục SendTo→Desktop (create shortcut).**



Các mục đậm màu thể hiện các vị trí cần di chuyển chuột đến

Đổi tên của biểu tượng

- Nhắp chuột phải lên biểu tượng làm xuất hiện hộp lệnh.
- Chọn mục **Rename**. Sau đó gõ tên mới vào ô nhập.
- Hoặc Chỉ cần nhắp chuột một lần vào phần tên của biểu tượng lập tức hệ điều hành cho phép gõ tên mới như hình minh họa.

Chọn lệnh Rename để đổi tên

Biểu tượng ở trạng thái cho phép nhập tên mới

Thi hành một ứng dụng:

• **Bằng biểu tượng : Double click** vào biểu tượng trên màn hình **Desktop.**

•**Bằng Menu Start:** Click chuột vào nút **Start\Program click vào ứng dụng** cần khởi động.

•**Bằng Mycomputer:** Nhấp đúp vào biểu tượng **Mycomputer**, lần theo đường dẫn chứa tập tin muốn thi hành.

•**Bằng lệnh Run:** Click chuột vào nút **Start \ Run**, một hộp thoại xuất hiện, gõ tên chương trình cần chạy (Vd: **Excel**)vào hộp Open rồi nhấn OK.

Thay đổi cách hiển thị của các đối tượng

Click chuột phải vào **menu View** của cửa sổ và lựa chọn các mục như hình, lựa chọn 1 trong bốn mục dưới đây để hiển thị:

Large Icons: Biểu tượng lớn.

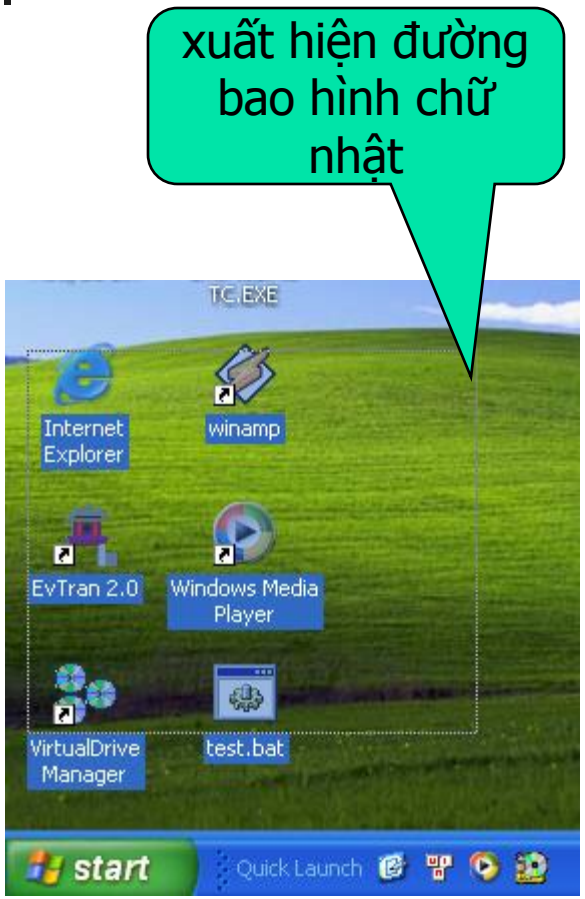
Small Icons: Biểu tượng nhỏ.

List: Kiểu liệt kê.

Detail: Kiểu nhìn chi tiết.

Thumbnails: Hiển thị nhanh các tập tin ảnh.

Chọn nhóm tập tin, thư mục



Chọn nhóm đối tượng liền kề

Thao tác bằng thiết bị chuột

- Kéo di chuột tạo một đường hình chữ nhật bao quanh các biểu tượng muốn chọn. Các biểu tượng được chọn đổi sang màu tối nên thao tác chọn được gọi nôm na là "bôi đen đối tượng".

Chọn nhóm tập tin, thư mục

Chọn nhóm biểu tượng rời rạc

Thao tác kết hợp giữa chuột và bàn phím

- Nhắp chuột chọn một biểu tượng.
- Giữ phím Ctrl và nhắp chuột vào biểu tượng khác.
- (Điểm quan trọng ở đây là giữ phím Ctrl khi chọn các đối tượng. Có thể nhắp chọn lần thứ hai trên một đối tượng để nhanh chóng hủy chọn chỉ riêng cho đối tượng đó).

Chọn nhóm tập tin, thư mục

Chọn các tập tin /thư mục không liên tục: *Bấm phím* ***Ctrl*** *và Click chọn* ***tên thư mục*** **cần chọn.**

Chọn các tập tin /thư mục liên tục: *Bấm phím* **Shift** *và* ***Click chọn chuột*** *vào* **đầu** *và* **cuối** *của* **khối thư mục** *mà mình muốn chọn***.**

Hủy chọn: *Để hủy chọn toàn bộ các đối tượng đã chọn , ta* nhấp chuột tại vị trí trống bất kỳ.

Chuyển về thư mục trên một cấp: *Click nút* **Back** *hoặc nút* **Up.**

Đổi tên tập tin, thư mục:

Có 3 cách:

Chọn **tên tập tin/thư mục** *cần đổi tên.*

Cách 1: *Chọn menu lệnh* **File\ Rename** → *nhập tên mới.*

Cách 2: **Click phải chuột → Rename**

Cách 3: *Nhấn phím* **F2**

Sao chép, di chuyển tập tin, thư mục

Chọn các tập tin / thư mục cần sao chép (di chuyển), *làm theo một trong 3 cách sau:*

Cách 1: *vào menu* **Edit\Copy** *(hoặc* **Cut** *nếu muốn di chuyển)*

Cách 2: *nhấn* **Ctrl +C** *(hoặc* **Ctrl + X** *nếu muốn di chuyển)*

Cách 3: **Ctr + kéo chuột** *(hoặc* **Shift** + **kéo chuột** *nếu muốn di chuyển)*

Mở thư mục cần sao chép đến *(thư mục đích) rồi chọn* **Edit\ Paste** *(hoặc nhấn* **Ctrl + V***)*

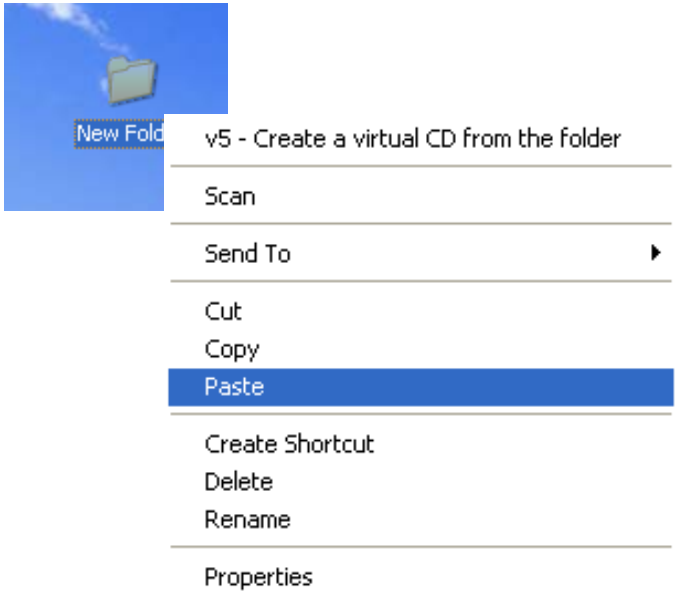
Sao chép, di chuyển tập tin, thư mục

Chuẩn bị: tạo sẵn trên màn hình nền thư mục có tên là ***congvan*** *và tệp tin* ***Image1***.

- Chọn nhóm biểu tượng gồm thư mục **congvan** và tệp tin **Image1**.
- Nhắp chuột phải trên vùng chọn làm xuất hiện hộp lệnh. Sau đó chọn mục Copy.

Nhắp chuột phải làm xuất hiện hộp lệnh, sau đó chọn mục Copy

Sao chép, di chuyển tập tin, thư mục



Chọn mục Paste

- Tạo mới thư mục tên là New Folder. Nhắp chuột phải trên thư mục này làm xuất hiện hộp lệnh và sau đó chọn mục Paste để dán bản sao của **congvan** và **Image1** vào trong thư mục này.

Gợi ý thêm: Sử dụng tổ hợp phím để thao tác nhanh

- Bấm tổ hợp phím Ctrl+C tương đương với việc chọn mục Copy trong hộp lệnh.
- Bấm tổ hợp phím Ctrl+V tương đương với việc chọn mục Paste trong hộp lệnh.

Di chuyển tệp tin đến thư mục khác

Chọn biểu tượng tệp tin.

- Nhắp chuột phải lên vùng chọn để làm xuất hiện hộp lệnh. Sau đó chọn mục Cut.
- Nhắp nút phải lên thư mục, nơi sẽ cất giữ tệp tin, để làm xuất hiện hộp lệnh. Tiếp theo là chọn mục Paste.

Gợi ý thêm: Sử dụng tổ hợp phím để thao tác nhanh

- Bấm tổ hợp phím Ctrl+X tương đương với chọn mục Cut trong hộp lệnh.

Hiển thị thông tin của tập tin, thư mục

- Chọn nhóm biểu tượng.
- Nhắp chuột phải trên vùng chọn làm xuất hiện hộp lệnh, chọn mục Properties.

Chọn mục thuộc tính

Thông tin của thư mục xuất hiện

Hiển thị thông tin của tập tin, thư mục

Thông tin của tệp tin, thư mục xuất hiện trong hộp Properties cho người dùng biết số lượng tệp tin, thư mục và quan trọng nhất là tổng dung lượng nhớ (ví dụ theo hình minh họa là 35.2MB)

Đặt thuộc tính chỉ đọc (read-only)

Trong hộp thoại Properties, nhắp chọn ô Read-only để đặt thuộc tính chỉ đọc cho tệp tin hay thư mục và có thể nhắp chọn ô Hidden để đặt thuộc tính che dấu cho tệp tin hay thư mục.

chọn cho phép đặt thuộc tính chỉ đọc

Xoá các tập tin, thư mục

Chọn các tập tin / thư mục cần xoá, *làm theo các cách sau:*

Cách 1: *Vào menu* **File/Delete**

Cách 2: *Ấn phím* **Delete**

Cách 3: **Click phải** → *chọn* **Delete**

→ *Nhấn* **Yes** *để chấp nhận xóa (nếu không muốn xóa nhấn* **No***)*

Lưu ý: *Sau khi xóa* **tập tin/ thư mục** *sẽ được Windows bỏ vào thùng rác. Nếu muốn xóa vĩnh viễn, ta cũng làm như thao tác trên nhưng nhấn giữ thêm phím* **Shift** *trong khi chọn* **Delete.(*tổ hợp phím Shift-Del*)**

Xoá các tập tin, thư mục

- Biểu tượng của thùng rác trên màn hình nền

thùng rác rỗng

thùng rác chứa tệp tin hoặc thư mục đã bị xóa

Lưu ý: Nếu dùng tổ hợp phím Shift-Del để thực hiện xóa tệp tin, thư mục thì đối tượng bị xoá sẽ mất hẳn không lưu lại trong thùng rác.

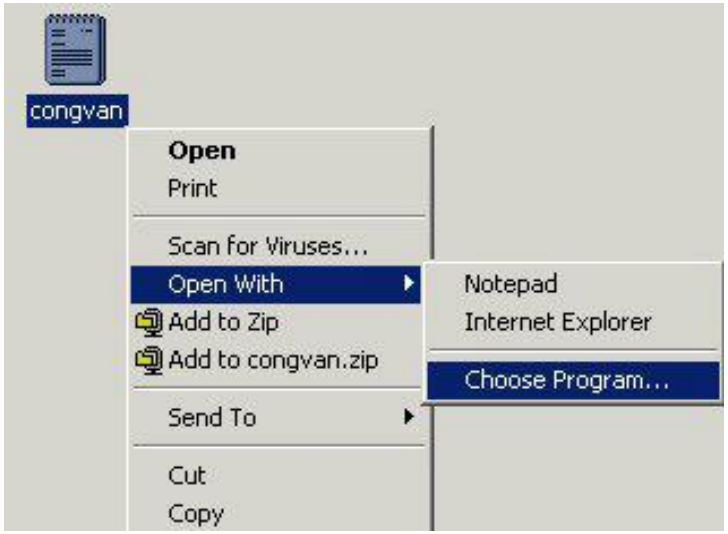
Khôi phục tập tin, thư mục đã xoá

Double Click vào biểu tượng **Recycle Bin** ở trong **desktop** → Chọn **tập tin/ thư mục** cần khôi phục:

- **Cách 1:** Vào menu **File \ Restore**
- **Cách 2: Click phải \ Restore**

Mở tệp tin

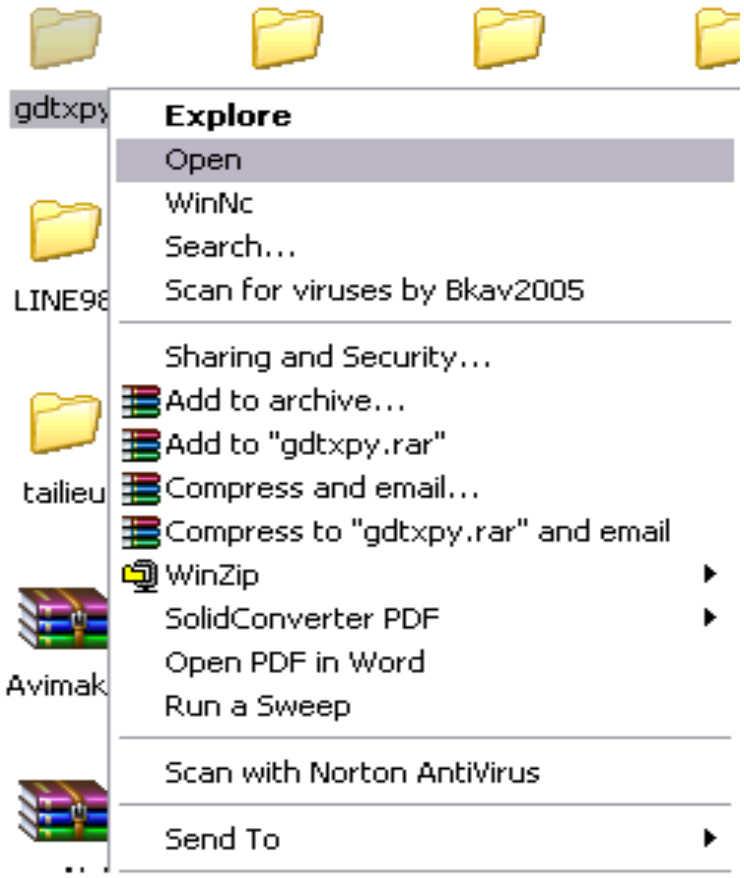
- Nhắp đúp lên biểu tượng tệp tin hoặc
- Nhắp chuột phải trên biểu tượng tệp tin làm xuất hiện hộp lệnh, khi đó có hai lệnh để chọn: lệnh Open hoặc lệnh Open with
- Chọn một chương trình mở tệp và nhắp nút OK.



chọn mục Open With để mở tệp bằng chương trình tự chọn

Chọn lựa chương trình mở tệp là Microsoft Word

Mở thư mục



chọn mục Open để mở thư mục

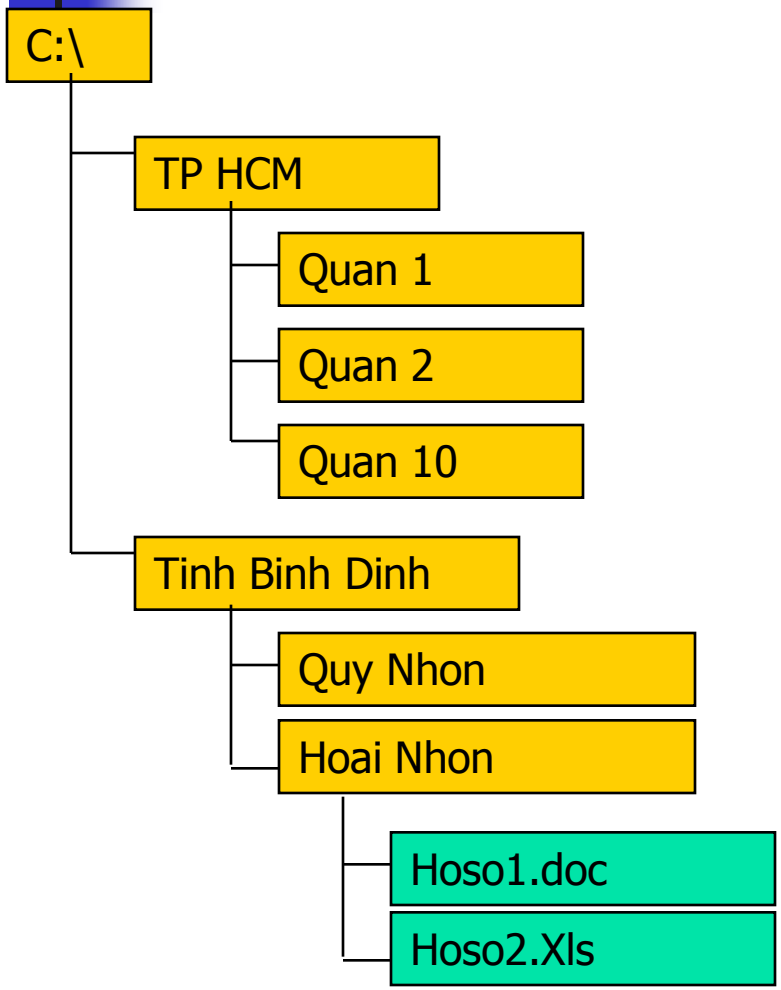
Có hai cách để mở thư mục:

- nhắp đúp lên biểu tượng thư mục hoặc
- nhắp chuột phải trên biểu tượng thư mục làm xuất hiện hộp lệnh, sau đó chọn mục Open.

Tóm tắt bài 3

- ***Phân biệt được tập tin, thư mục, các kiểu tập tin, hiểu khái niệm đường dẫn.***
- ***Tạo mới tập tin, thư mục.***
- ***Đổi tên tập tin, thư mục.***
- ***Copy, di chuyển tập tin, thư mục.***
- ***Hiển thị thông tin, thay đổi cách hiển thị của tập tin, thư mục.***
- ***Xóa, phục hồi tập tin, thư mục.***

BÀI TẬP

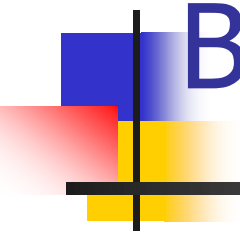


Câu 1: Tạo cây thư mục như hình bên.

Câu 2: Sao chép 2 tập tin hoso1.doc và hoso2.xls vào thư mục Quan 2.

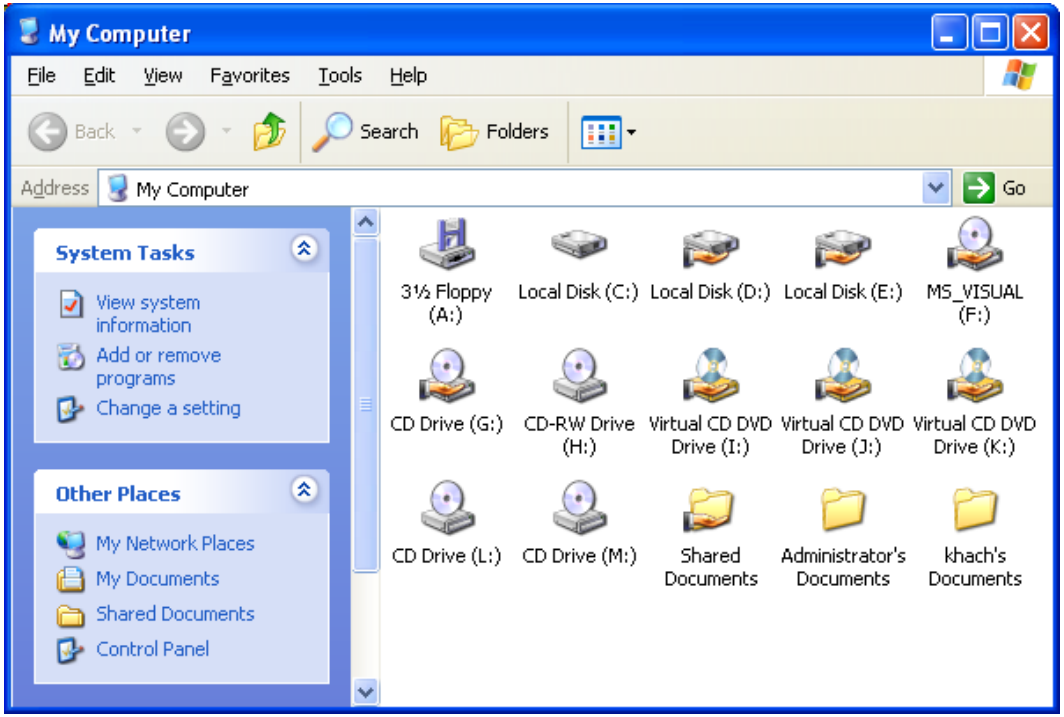
Câu 3: Xoá thư mục Hoai Nhon

Câu 4: Khôi phục lại thư mục vừa xoá



Bài 4: LÀM VIỆC VỚI CỬA SỔ

Các thao tác cơ bản

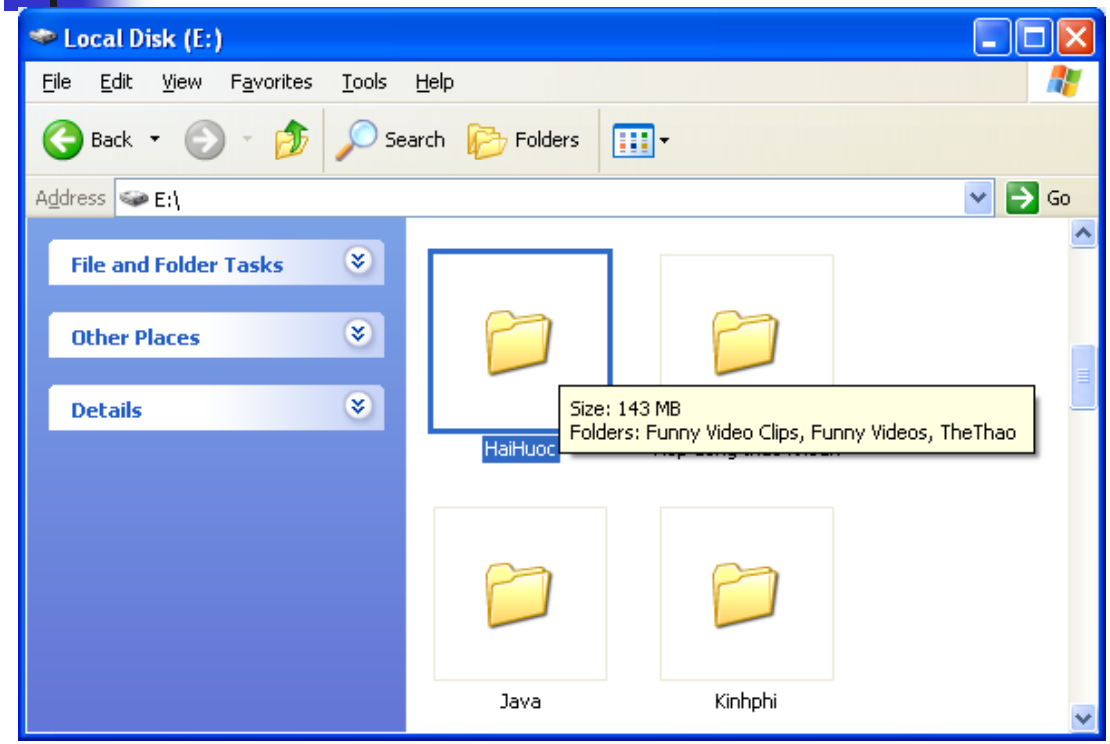


Cửa sổ My Computer

Mở cửa sổ :

- **Bằng My computer:** Nhấp đúp chuột lên biểu tượng My computer.
- **Bằng biểu tượng thư mục, tập tin:** Nhấp đúp lên biểu tượng thư mục, tập tin cần mở.

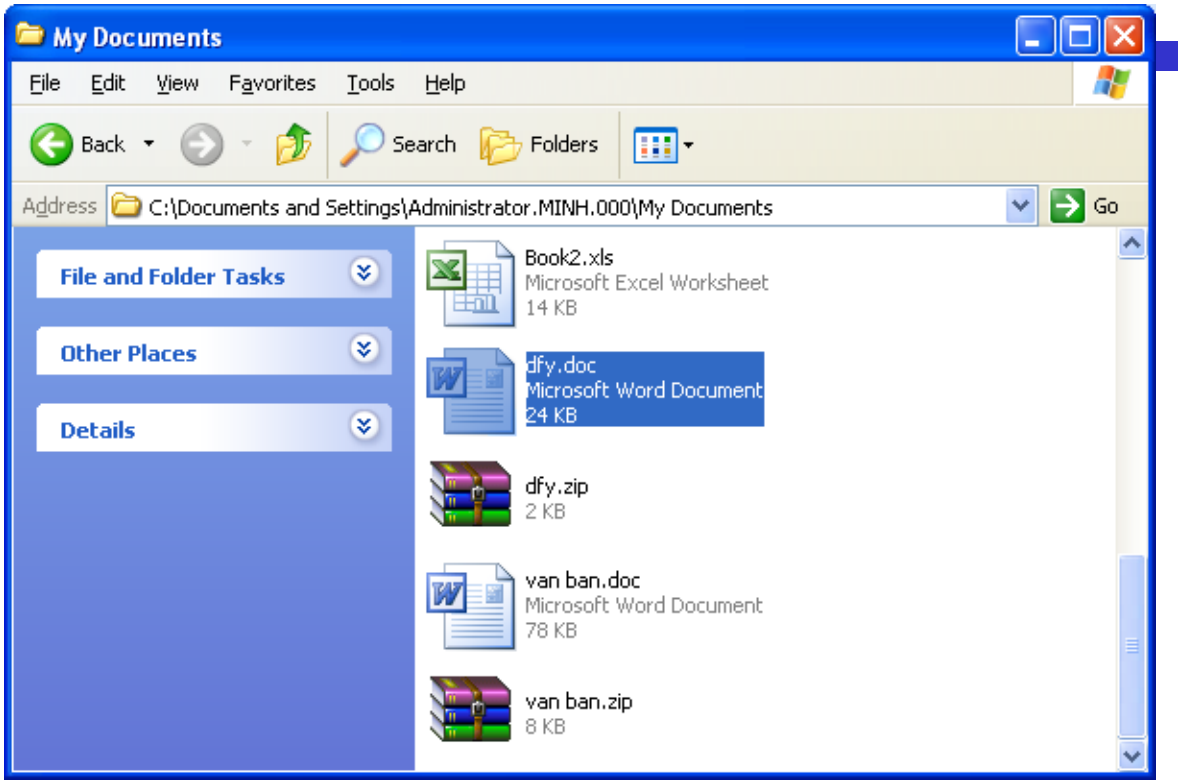
Cửa sổ làm việc với thư mục



Mở cửa sổ thư mục

- Thao tác nhắp đúp chuột lên biểu tượng thư mục bất kỳ có trên màn hình để mở cửa sổ làm việc với thư mục.

Cửa sổ làm việc với tệp tin

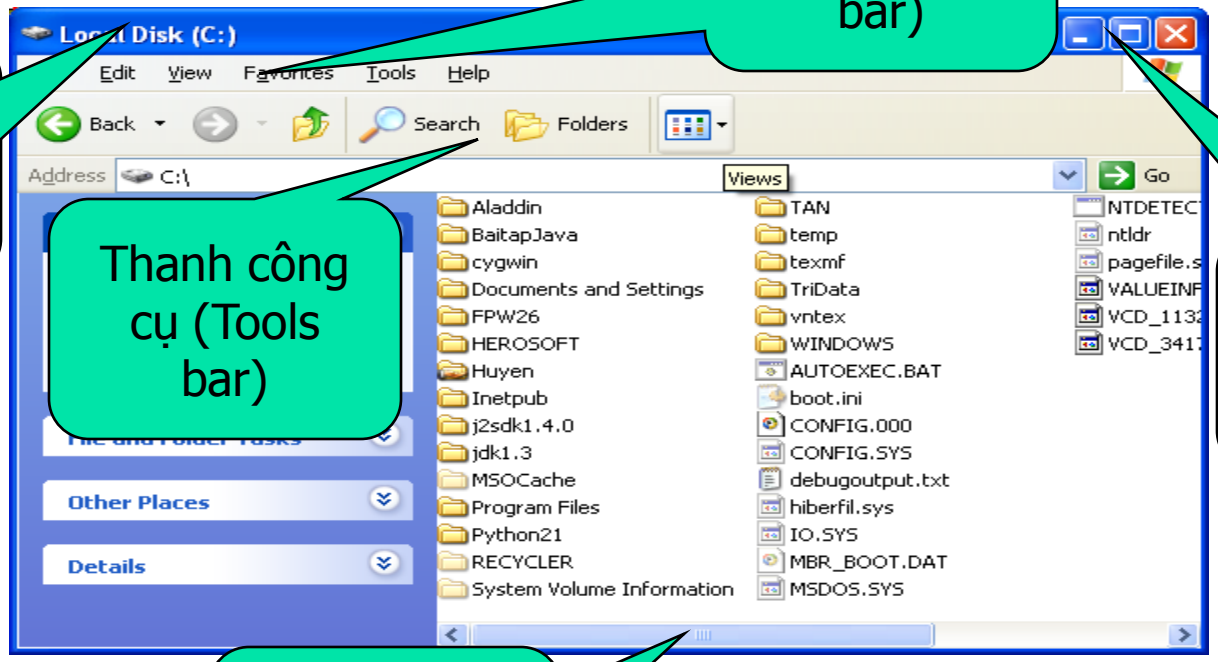


Thao tác nhắp đúp chuột lên biểu tượng tệp tin bất kỳ có trên màn hình nền để mở cửa sổ làm việc với tệp tin.

Các thành phần của cửa sổ thư mục

- Thanh tiêu đề nằm phía trên cửa sổ. Khi nhắp đúp chuột trên thanh tiêu đề làm phóng to hoặc thu nhỏ cửa sổ.
- Dưới thanh tiêu đề là thanh thực đơn lệnh gồm thực đơn lệnh FILE, EDIT,... Tất cả các lệnh để điều khiển cửa sổ và để điều khiển các đối tượng khác trong cửa sổ đều có mặt trong thanh thực đơn lệnh.
- Thanh công cụ chứa các nút gắn sẵn chức năng (ví dụ nút  để xoá tệp tin hay thư mục đã chọn) giúp cho người sử dụng thao tác dễ dàng hơn, chính xác hơn và nhanh hơn so với việc chọn các lệnh có trên thanh thực đơn.

Các thành phần của cửa sổ thư mục



Thanh thực đơn (Menu bar)

Thanh tiêu đề (Title bar)

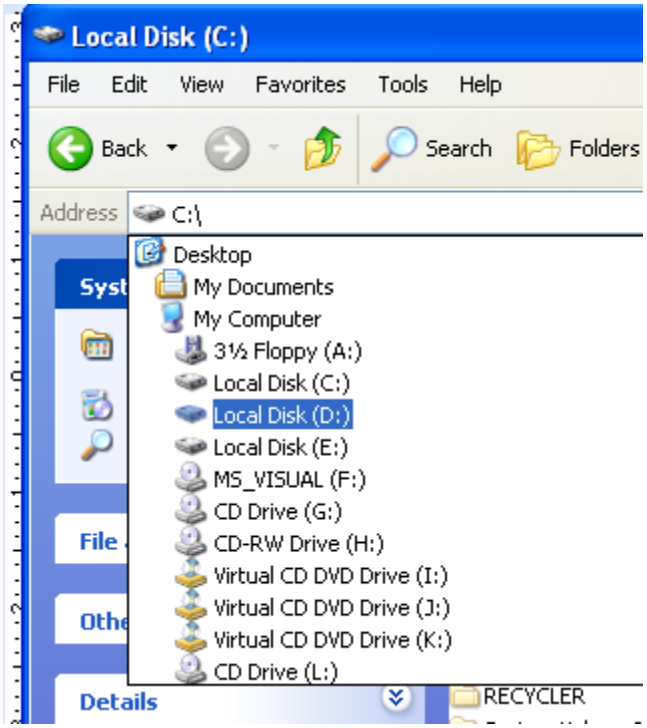
Thanh công cụ (Tools bar)

Các nút phóng to thu nhỏ, đóng

Thanh cuộn

Cửa sổ ổ đĩa C:

Thanh công cụ địa chỉ Address



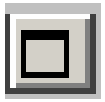
Hộp chọn hỗ trợ khả năng hiển thị sơ đồ thư mục dạng cây

- Thanh công cụ địa chỉ (tiếng Anh là Address) có hộp chọn hỗ trợ khả năng hiển thị thư mục theo sơ đồ dạng cây giúp người dùng dễ hình dung cách tổ chức thư mục trên bộ nhớ ngoài mỗi khi làm thao tác chọn và mở các cửa sổ tiếp theo.

Thu nhỏ, phóng to, đóng

Tại góc trên, bên phải cửa sổ có các nút:

- là nút thu nhỏ cửa sổ. Chỉ còn một nút bấm nhanh trên thanh tác vụ cho phép mở lại cửa sổ;



- là nút chức năng phóng to cửa sổ chiếm toàn bộ màn hình;

- là nút biến đổi cửa sổ về trạng thái có thể co giãn được;

- là nút đóng cửa sổ.

Cửa sổ hiện tại

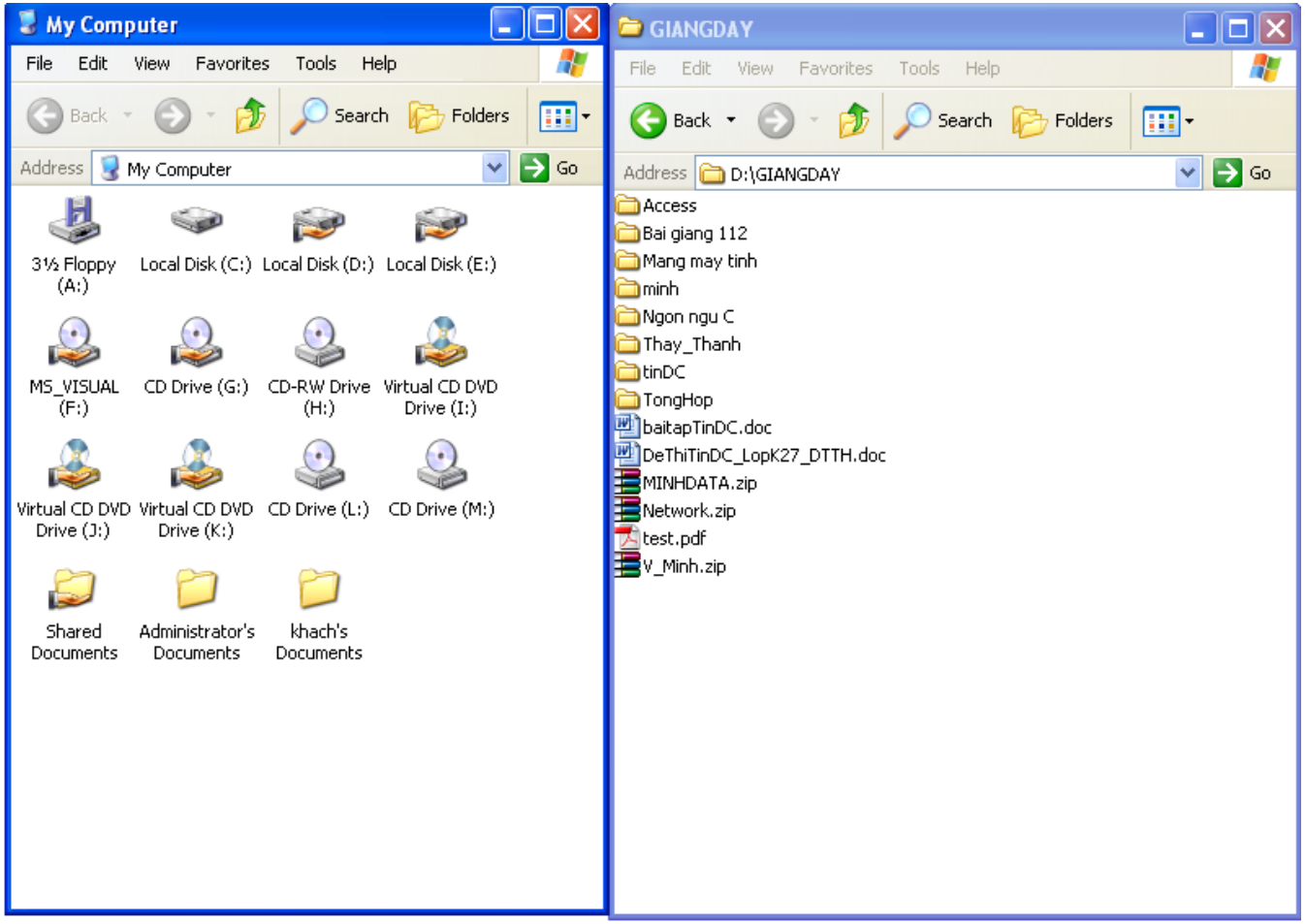
- Hệ điều hành MS-Windows cho phép mở nhiều thư mục trong cùng một cửa sổ.

  Theo thứ tự mở thư mục chúng ta có các khái niệm sau:
- Cửa sổ hiện tại, là cửa sổ thư mục đang mở.
- cửa sổ ngay trước là cửa sổ thư mục xuất hiện ngay trước cửa sổ hiện tại.
- cửa sổ liền sau là cửa sổ đã được mở sau cửa sổ hiện tại.

Cửa sổ hoạt động

- Hệ điều hành MS-Windows cho phép mở nhiều cửa sổ khác nhau.
- Hệ điều hành cho phép mở nhiều cửa sổ nhưng tại một thời điểm nhất định chỉ có một cửa sổ thư mục cho phép thao tác, và được gọi là cửa sổ hoạt động. Theo hình minh hoạ dưới đây, cửa sổ hoạt động là cửa sổ My Computer. Thanh tiêu đề cửa sổ này hiển thị sáng màu.

Cửa sổ hoạt động



Hai cửa sổ đang mở, nhưng chỉ có cửa sổ My Computer là cửa sổ hoạt động

Sử dụng thanh công cụ cơ bản - Standard Buttons

- Nút được sử dụng để quay trở lại cửa sổ ngay trước cửa sổ hiện tại.
- Nút được sử dụng khi chúng ta đã có sử dụng nút Back. Nút này cho phép quay trở về cửa sổ đã có trước khi bấm nút Back.
- Nhắp nút để chuyển đến cửa sổ thư mục mẹ của cửa sổ thư mục hiện tại.
- Nhắp nút làm xuất hiện vùng tìm kiếm thông tin (Search). Nhắp lần thứ hai để đóng vùng tìm kiếm.
- Nhắp nút làm xuất hiện vùng hiển thị thư mục dạng cây. Nhắp lần thứ hai để đóng vùng hiển thị thư mục dạng cây.

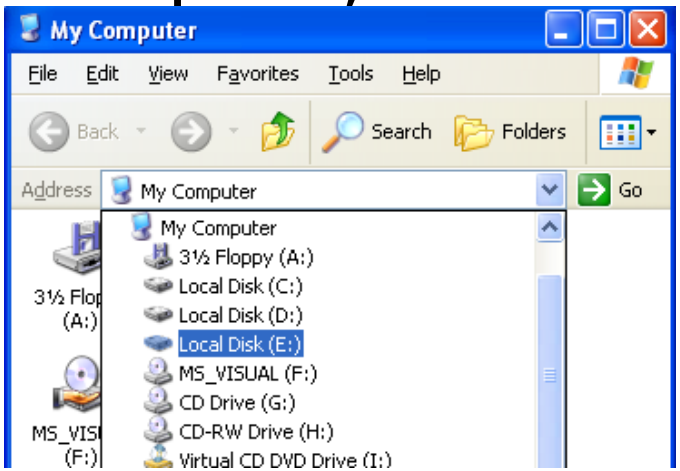
Ý nghĩa các mục trong hộp lệnh View

Thực hiện nhắp chuột lên dấu mũi tên làm xuất hiện nút Views bị che dấu. Trong hộp lệnh View có thể chọn các cách hiển thị tệp tin, thư mục trong cửa sổ:

- Mục Large Icons: khi chọn mục này, biểu tượng được hiển thị ở kích thước lớn.
- Mục Small Icons: khi chọn mục này, biểu tượng được hiển thị ở kích thước bé.
- Mục List: để hiển thị biểu tượng theo danh sách.
- Mục Details: để hiển thị biểu tượng với đầy đủ thông tin gồm: tên, kích thước,..
- Mục Thumbnails: cho phép hiển thị các tệp ảnh.

Thanh địa chỉ - Address

Trên thanh Address, bấm chọn mũi tên làm xuất hiện danh sách biểu tượng để chọn lựa và mở các cửa sổ khác như: Desktop, My Documents, My Computer,...



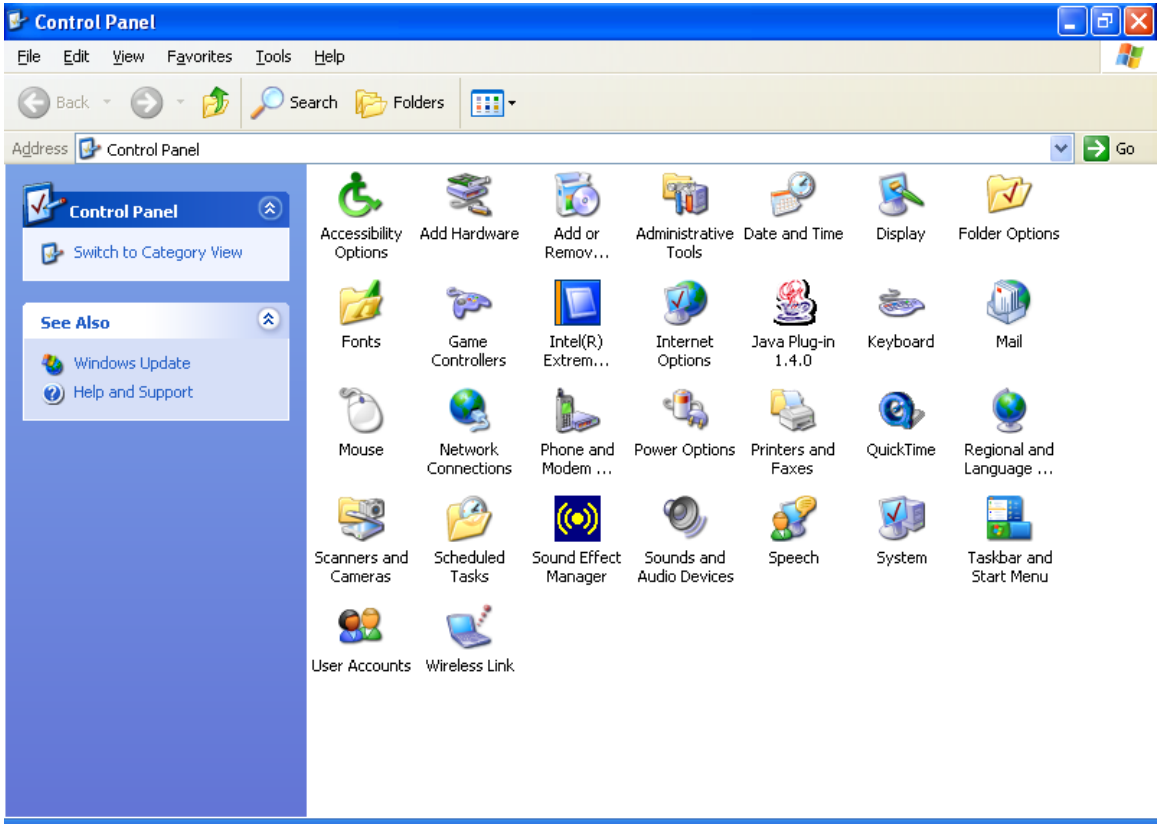
Trên thanh địa chỉ cho phép chọn nhanh các ổ đĩa

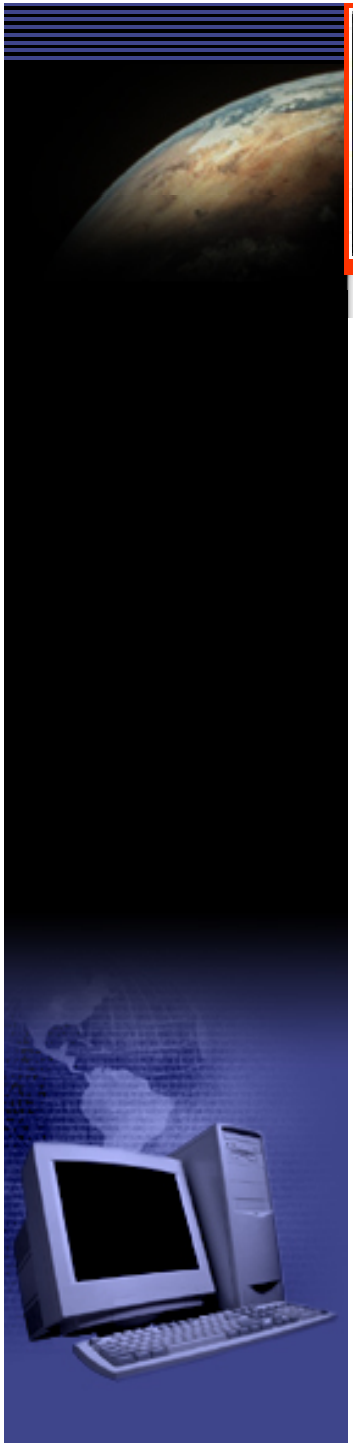
Di chuyển (hoán đổi) nhiều cửa sổ

- Mở ít nhất 3 cửa sổ: My computer, Recycle Bin, Winword.
- Để duy chuyển cửa sổ ta làm 3 cách sau:
  - **Cách 1:** Sử dụng các nút Minimize để thu nhỏ cửa sổ
  - **Cách 2:** Bấm chuột vào cửa sổ nằm trên thanh tác vụ
  - **Cách 3:** Sử dụng phím Alt + Tab

Cửa sổ Control Panel

- Nhắp chuột lên nút **Start → Settings → Control Panel** để mở cửa sổ Control Panel.



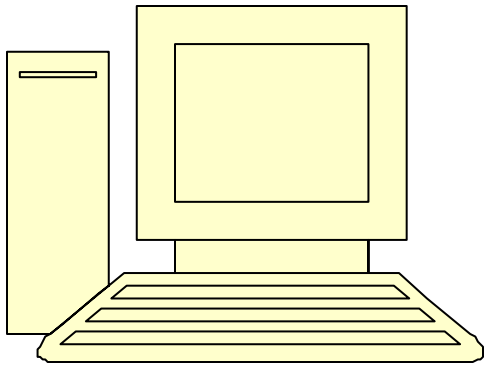


ĐẠI HỌC ĐÀ NẴNG

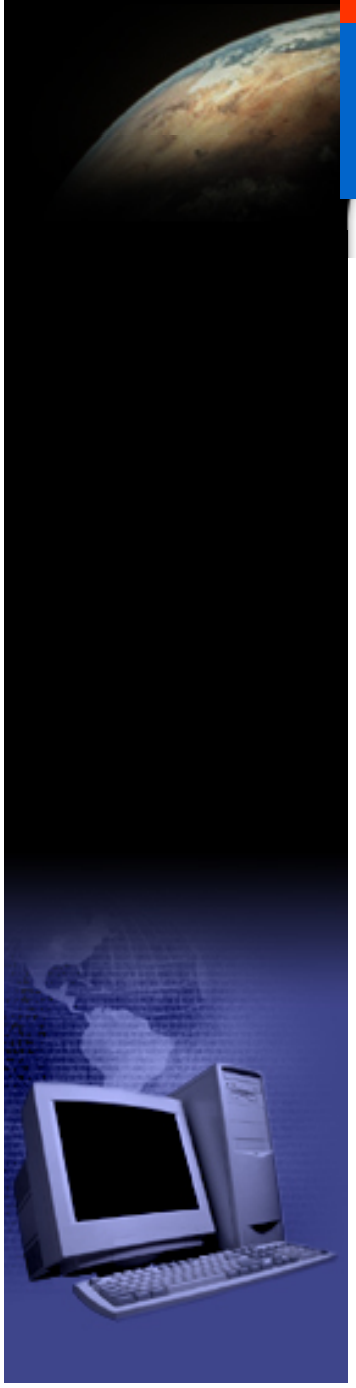
TRƯỜNG ĐẠI HỌC BÁCH KHOA

KHOA CÔNG NGHỆ THÔNG TIN

NGUYÊN LÝ HỆ ĐIỀU HÀNH



Giới thiệu



Nội dung giáo trình

CHƯƠNG 1. MỞ ĐẦU

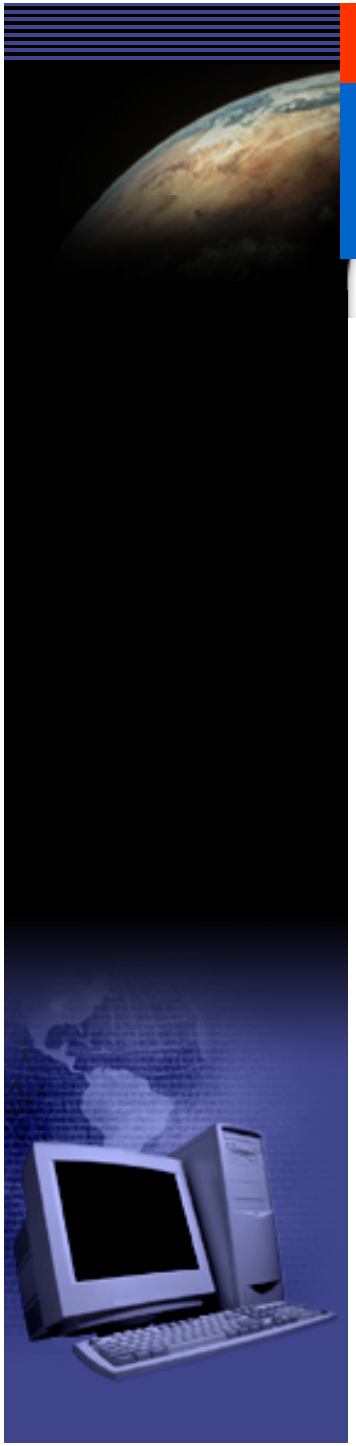
CHƯƠNG 2. TIẾN TRÌNH

CHƯƠNG 3. VÀO/RA

CHƯƠNG 4. QUẢN LÝ BỘ NHỚ

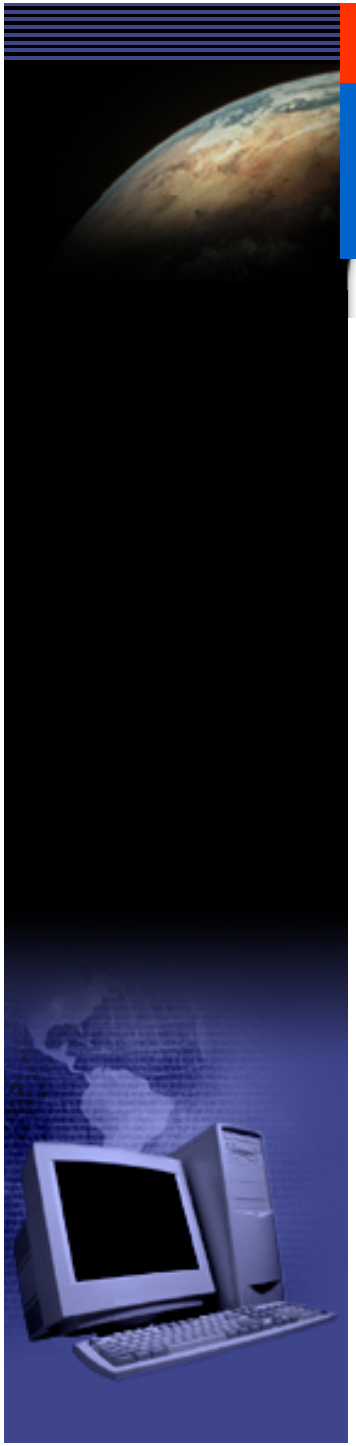
CHƯƠNG 5. HỆ THỐNG FILE

CHƯƠNG 1. MỞ ĐẦU



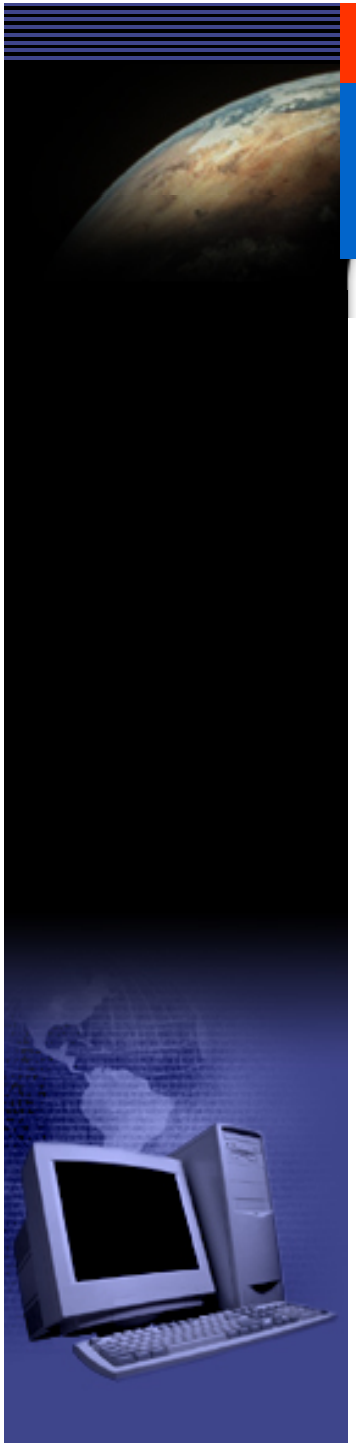
Các vấn đề

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

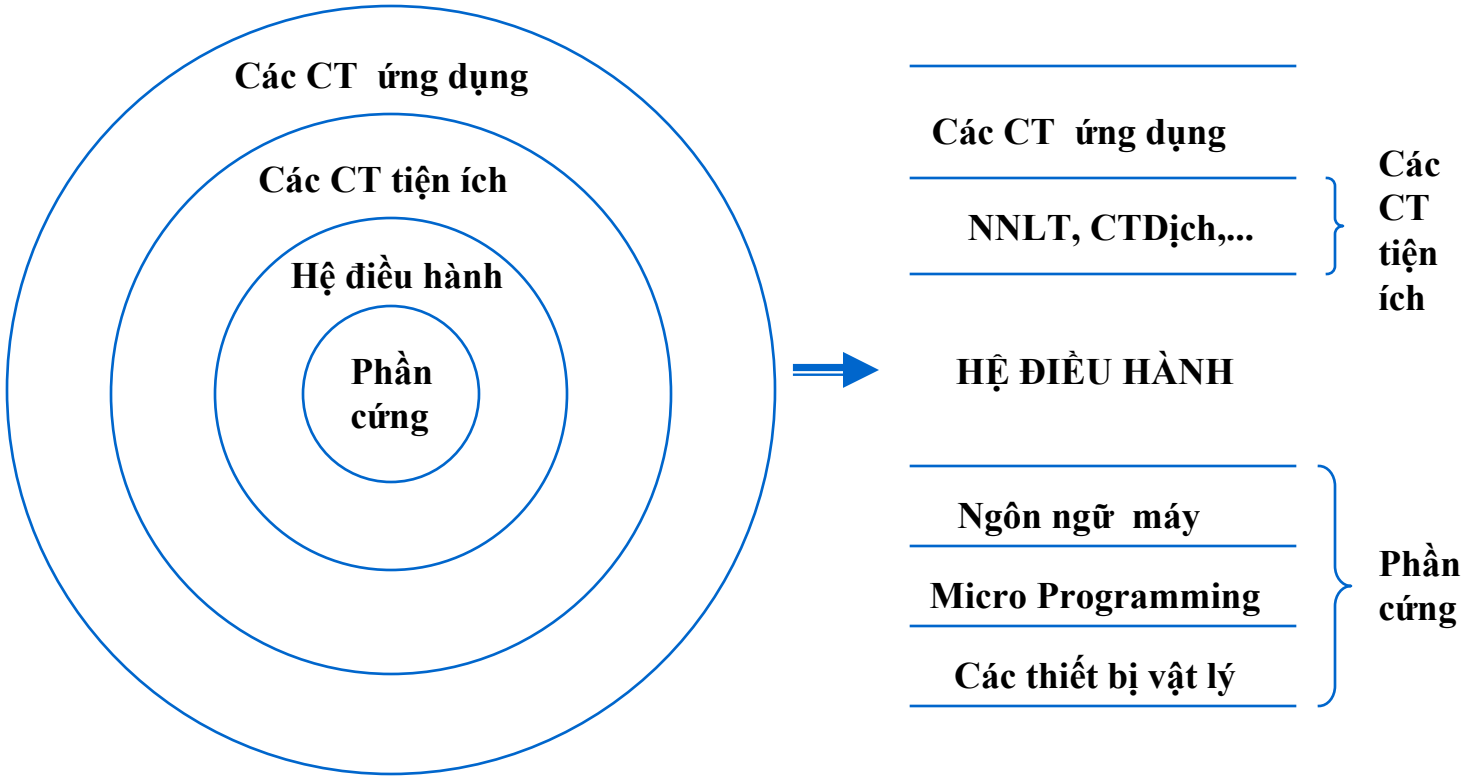
Tài nguyên

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

Các file, chương trình dùng chung,...

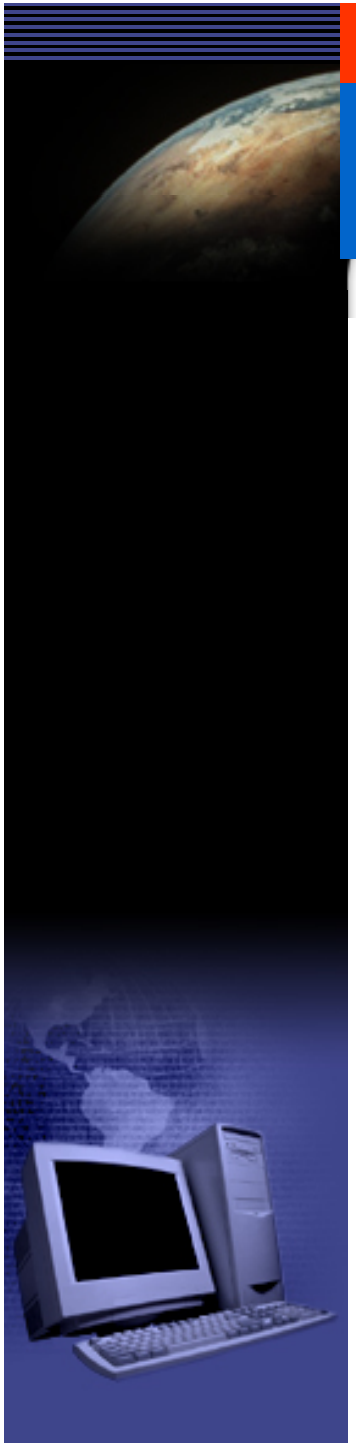
CHƯƠNG 1. MỞ ĐẦU

Vị trí của hệ điều hành



Các thành phần của hệ điều hành

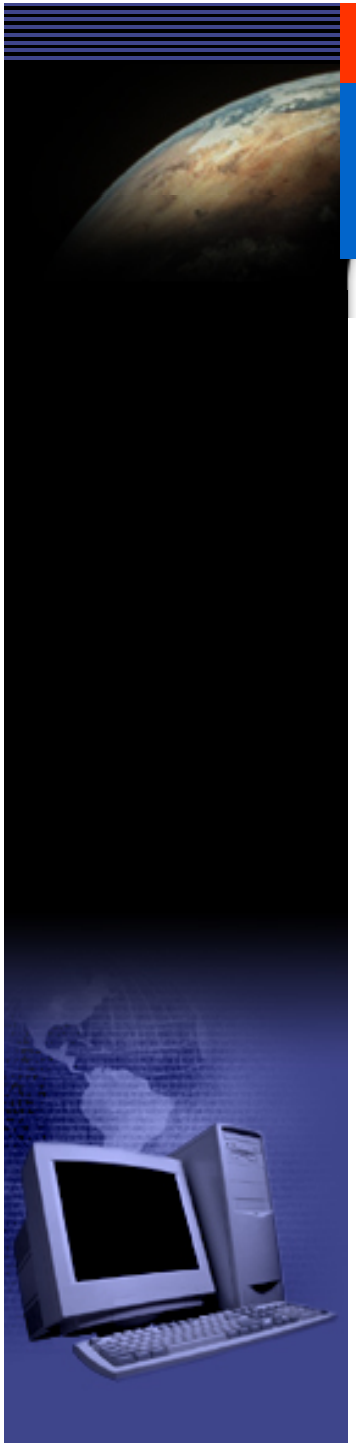
- Quản lý tiến trình
- Quản lý bộ nhớ
- Quản lý nhập xuất
- Quản lý tập tin
- Hệ thống bảo vệ
- Hệ thông dịch lệnh (Shell)
- Quản lý mạng



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

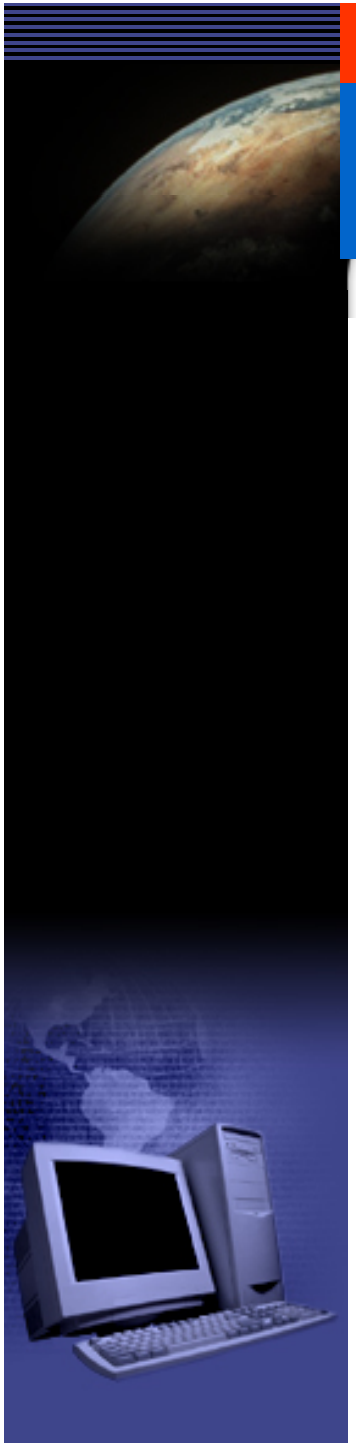
Quản lý bộ nhớ

- 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

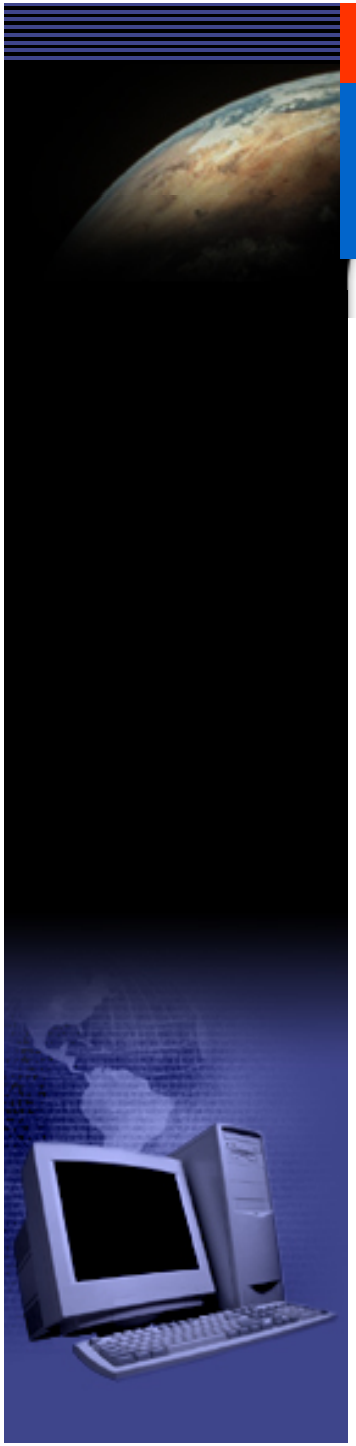
- **Xây dựng cơ chế bảo vệ thích hợp.**

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

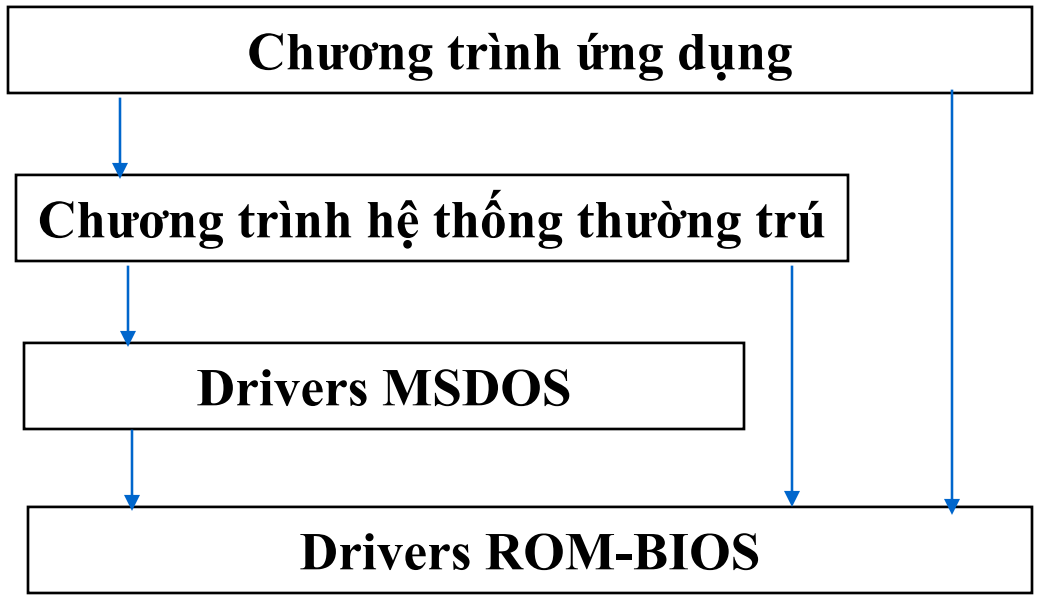
- Hệ thống nguyên khối (Monolithic System)
- Hệ thống phân lớp (Layer System)
- Máy ảo (Virtual Machine)
- Mô hình Client-Server (Client-Server Model)

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

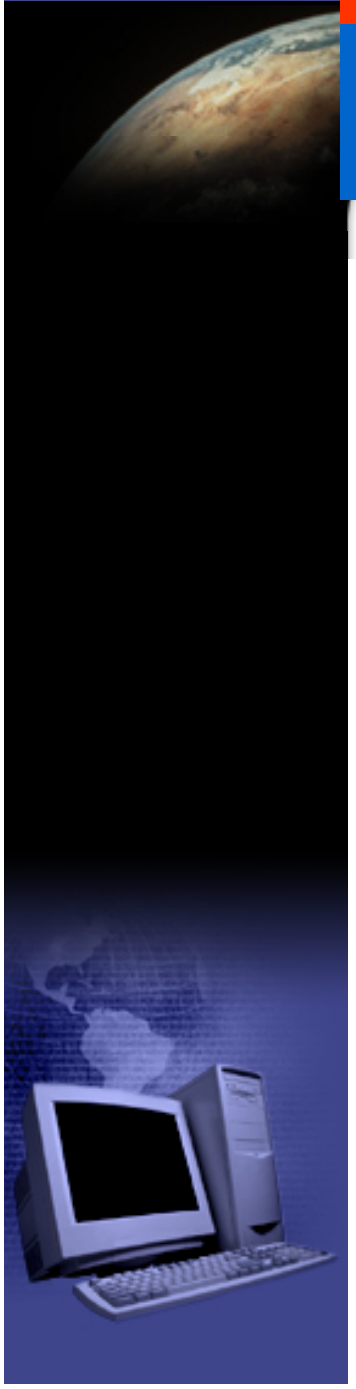
- Hệ thống được xây dựng bởi nhiều lớp.
- Mỗi lớp được xây dựng dựa trên các lớp bên trong
- Lớp trong cùng (lớp 0): phần cứng
- Lớp ngoài cùng (lớp N): giao diện với NSD
- Mỗi lớp là một đối tượng trừu tượng (dữ liệu+thao tác xử lý dữ liệu).
- Mỗi lớp có thể gọi các thủ tục của các lớp bên trong

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

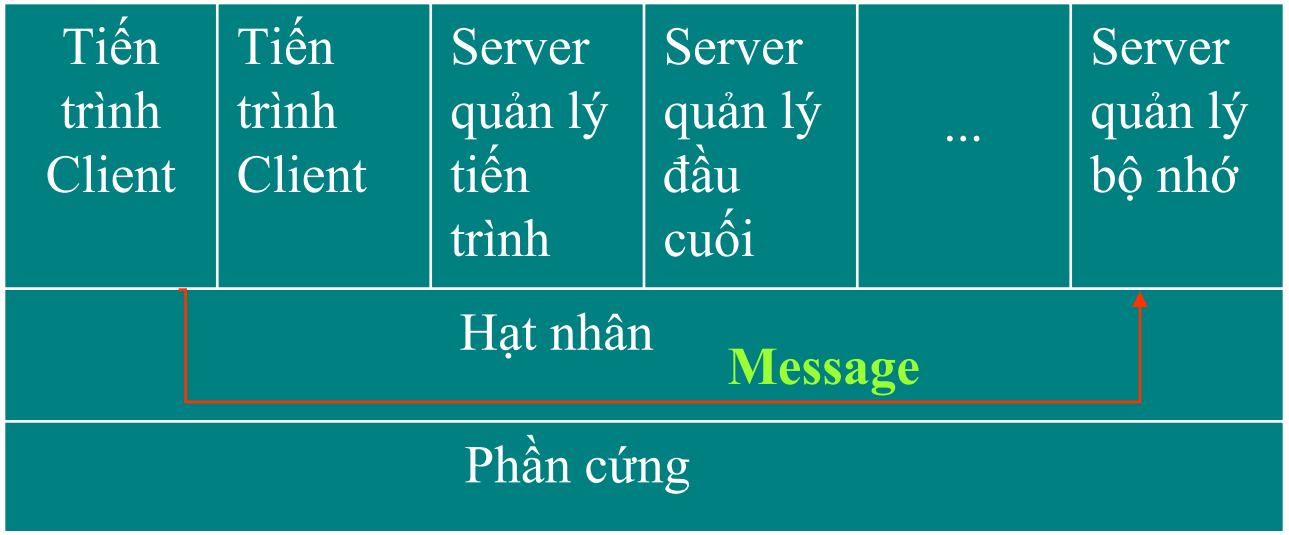


Máy ảo

Mô hình Client-Server

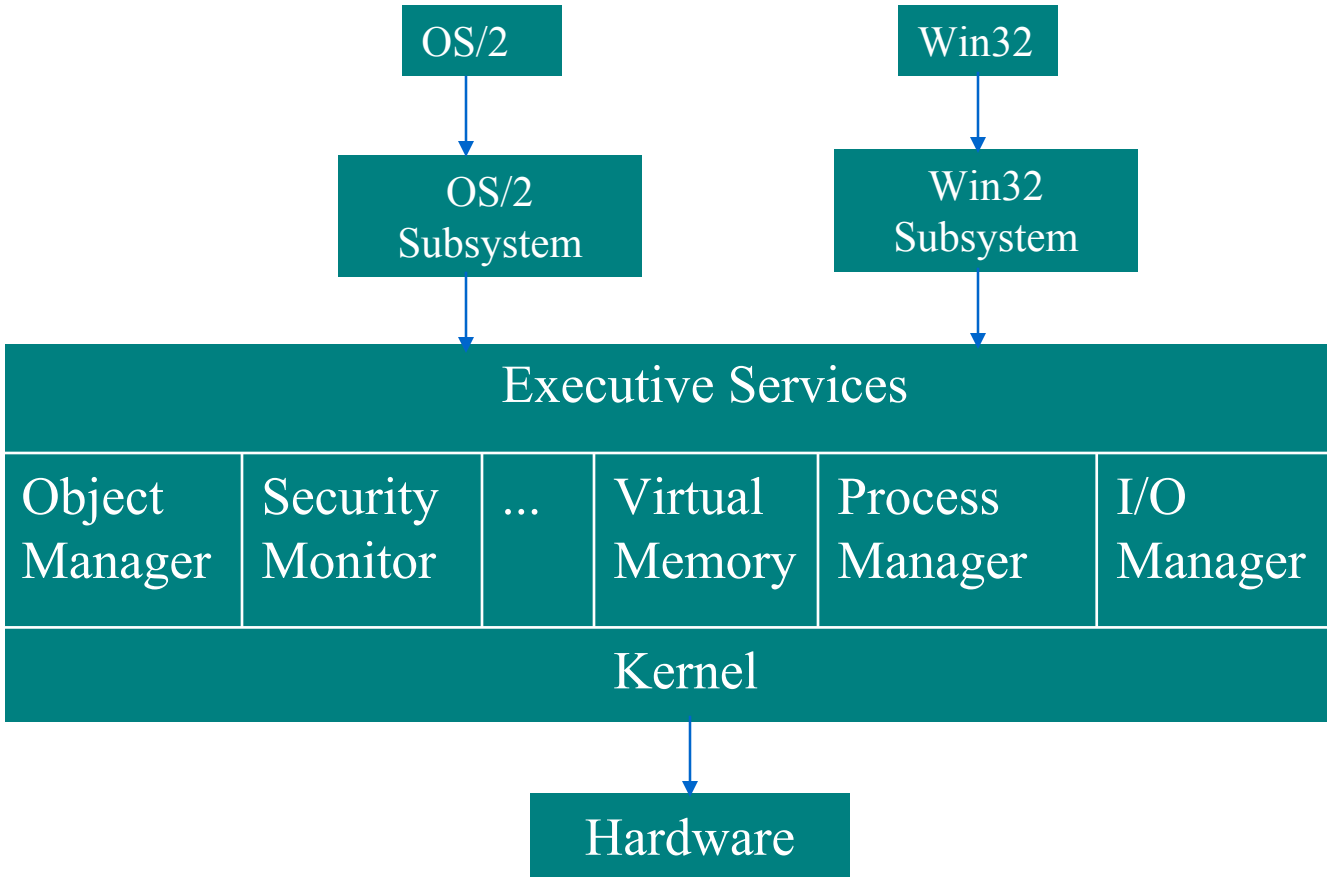
- **HĐH bao gồm nhiều tiến trình đóng vai trò Server với các chức năng chuyên biệt.**
- **Phần hạt nhân HĐH đóng vai trò giao tiếp giữa tiến trình Client và tiến trình Server.**
- **Chỉ có phần hạt nhân cực nhỏ phụ thuộc vào phần cứng.**

Mô hình Client-Server



Mô hình Client-Server

Ví dụ: Cấu trúc Windows NT



CHƯƠNG 2. TIẾN TRÌNH

Các vấn đề

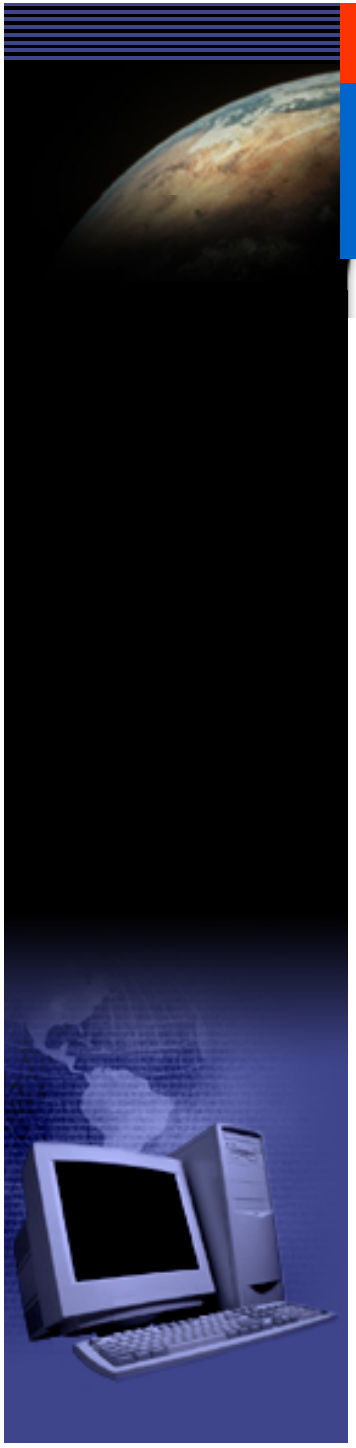
1. Các khái niệm
2. Mô hình trạng thái
3. Thao tác trên tiến trình
4. Điều phối tiến trình
5. Đồng bộ hoá tiến trình

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**



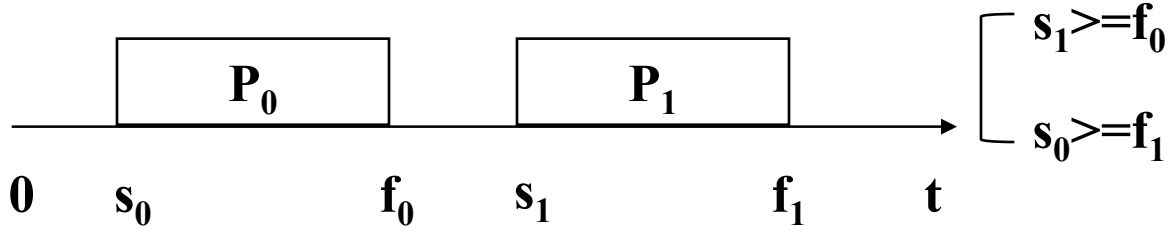
Các khái niệm

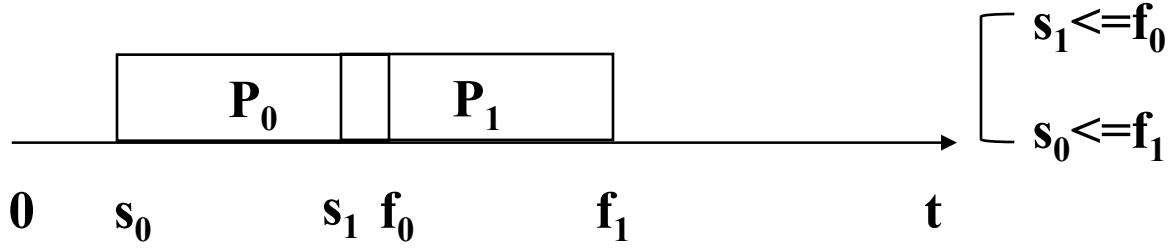
- **Có 2 loại tiến trình:**

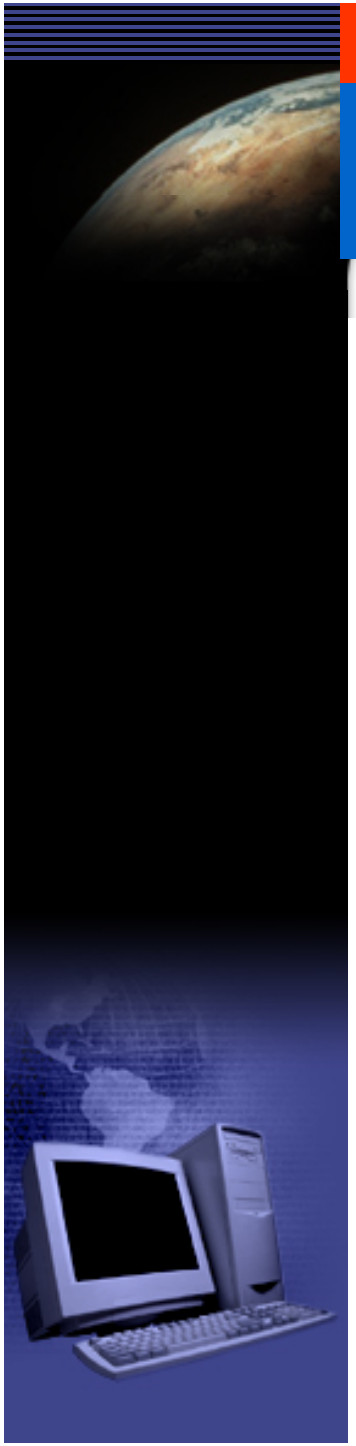
- Tiến trình kế tiếp: thời điểm bắt đầu của tiến trình này nằm sau thời điểm kết thúc của tiến trình kia

- 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

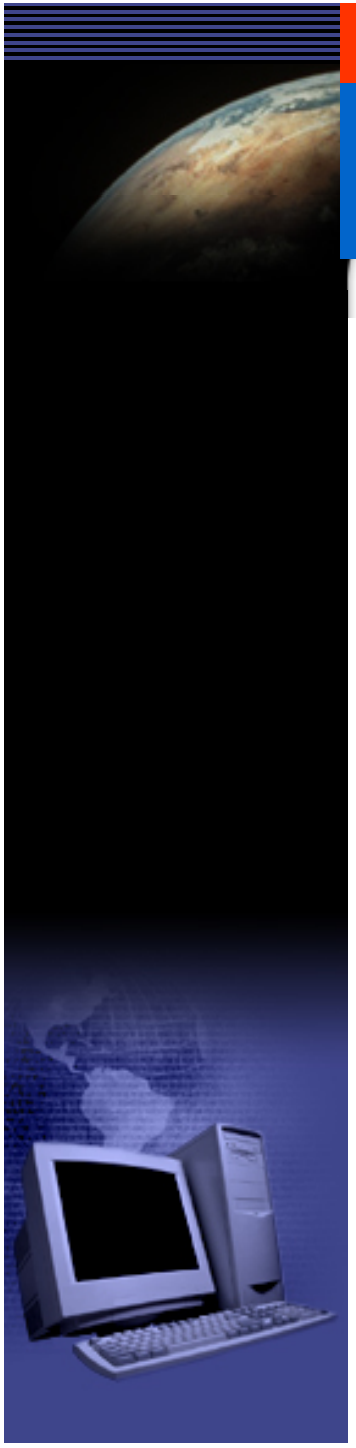






Các khái niệm

- **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:**

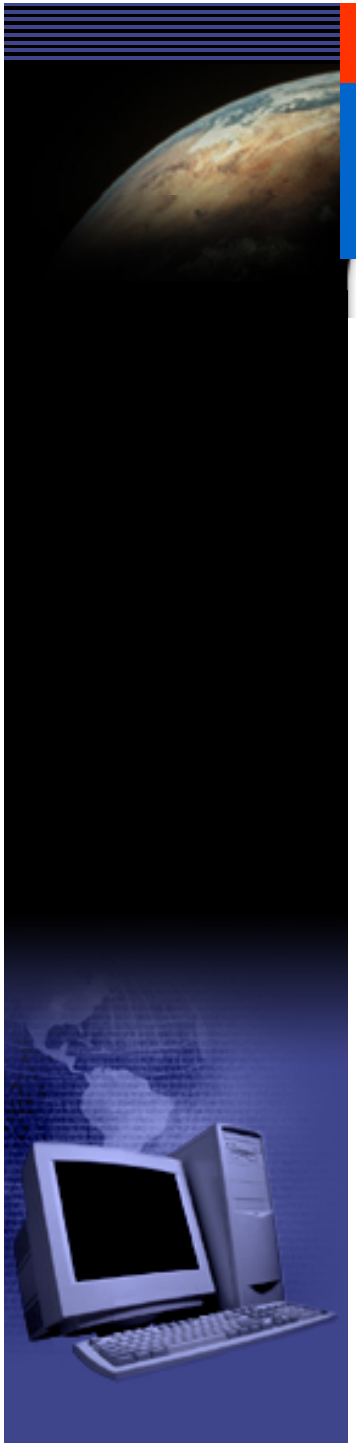
- Trạng thái CPU: nội dung các thanh ghi (IP). Lưu trữ nội dung thanh ghi khi xảy ra ngắt.

- 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

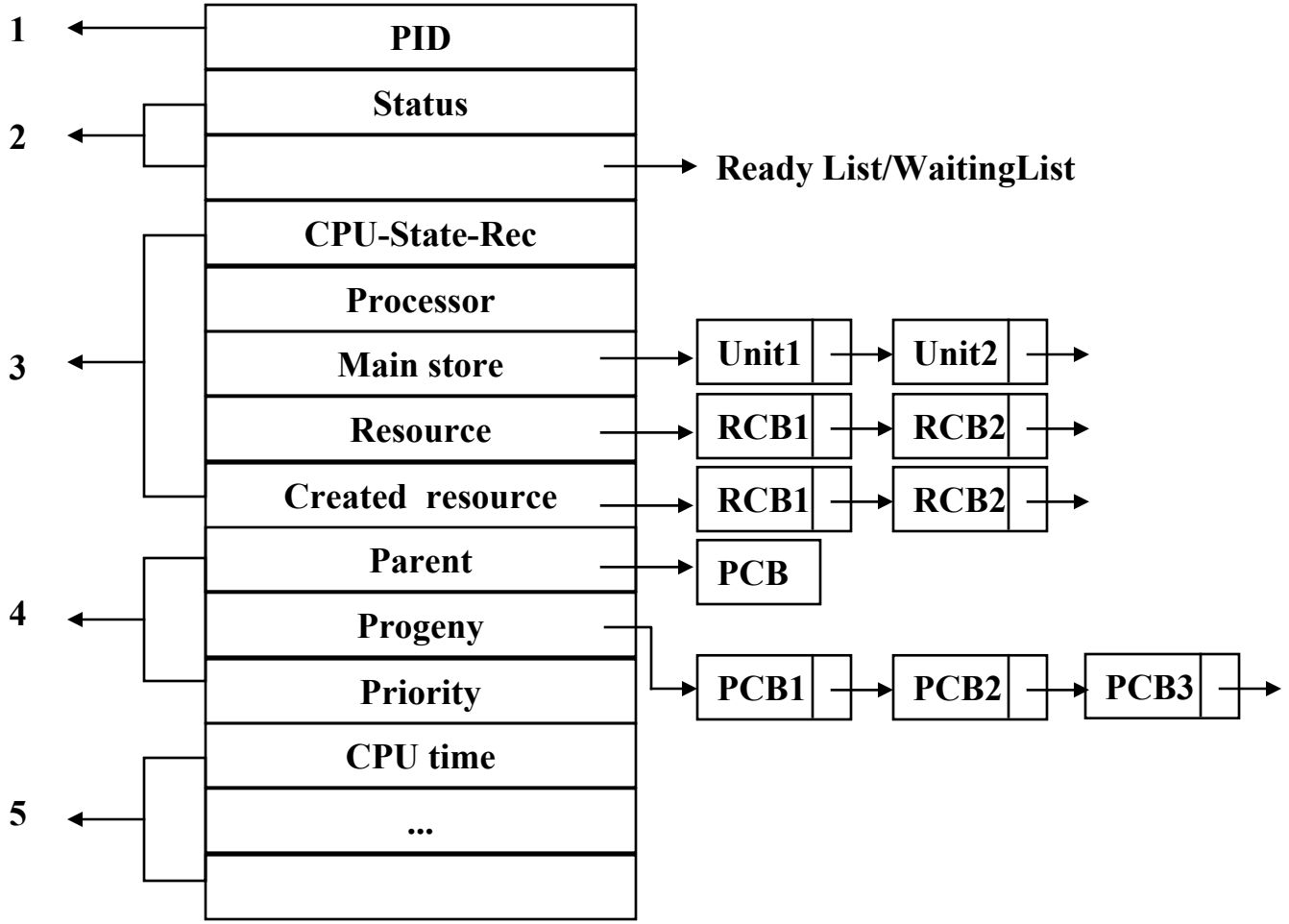
- **Thông tin thống kê về hoạt động của tiến trình:**

- 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

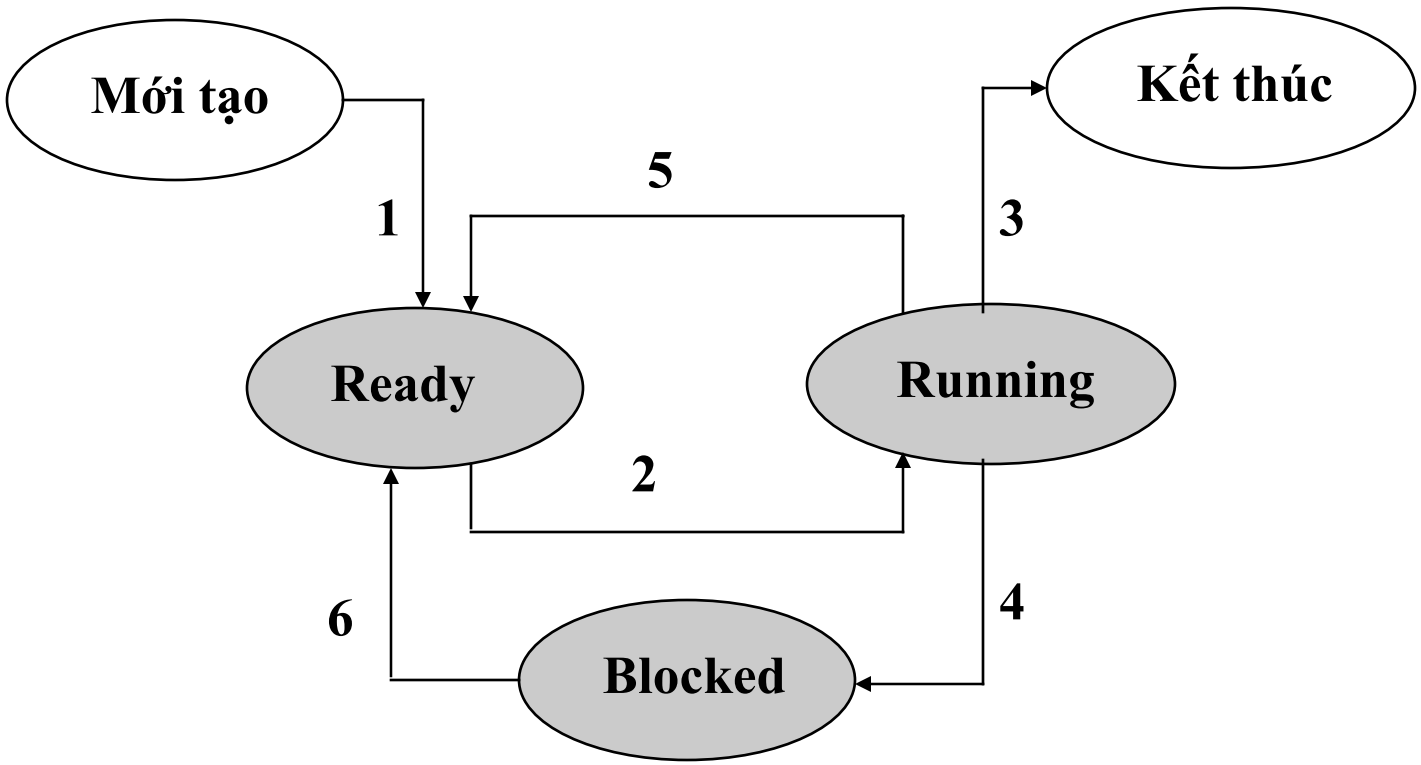
Mô hình trạng thái

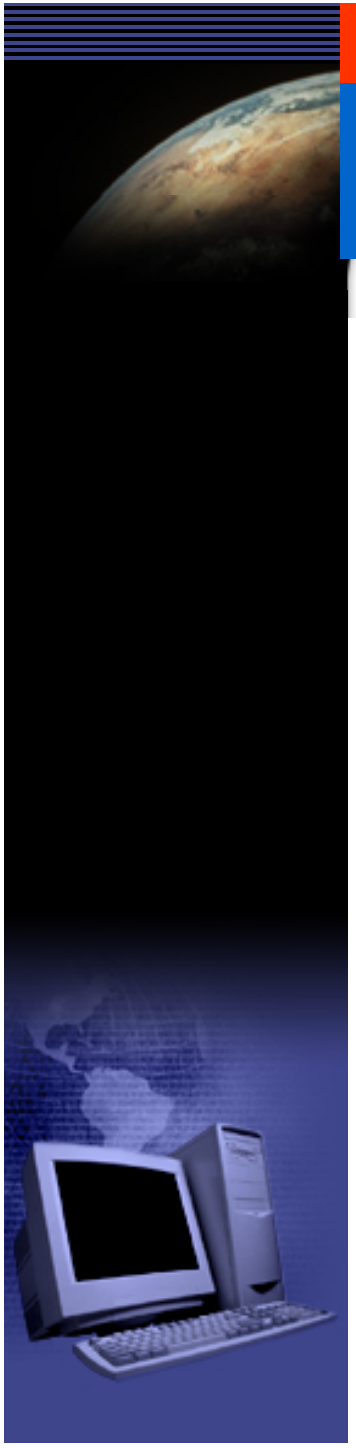
❖ Các trạng thái của tiến trình

- **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

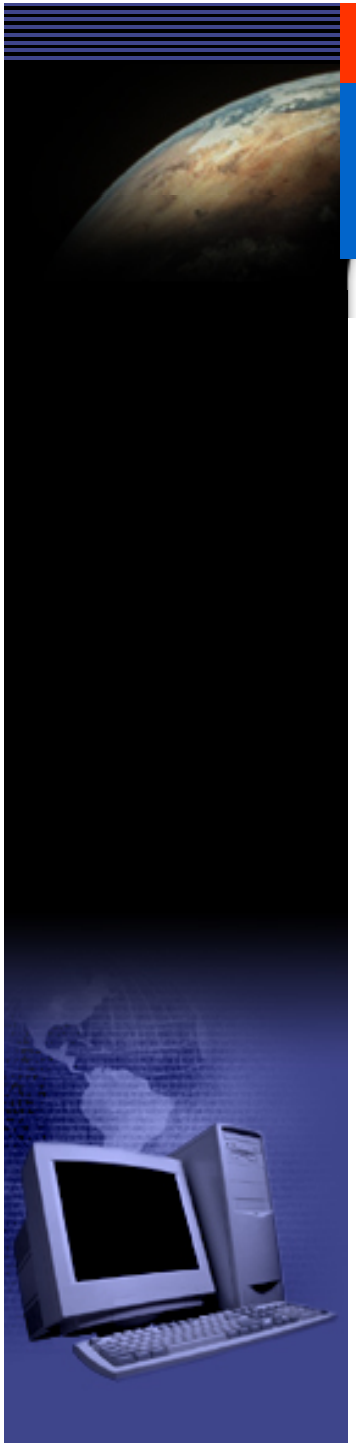
❖ **Sơ đồ chuyển trạng thái của tiến trình**

(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.

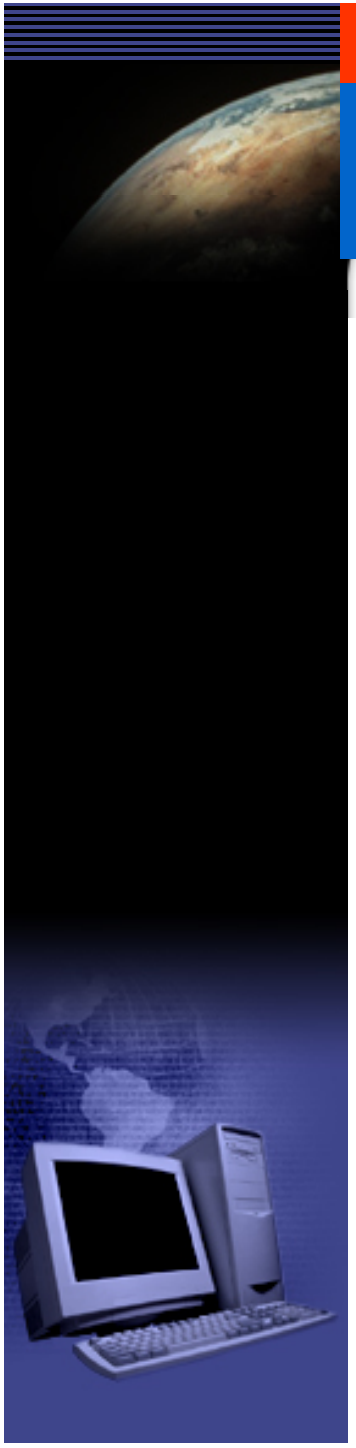


Mô hình trạng thái

❖ Sơ đồ chuyển trạng thái của tiến trình

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

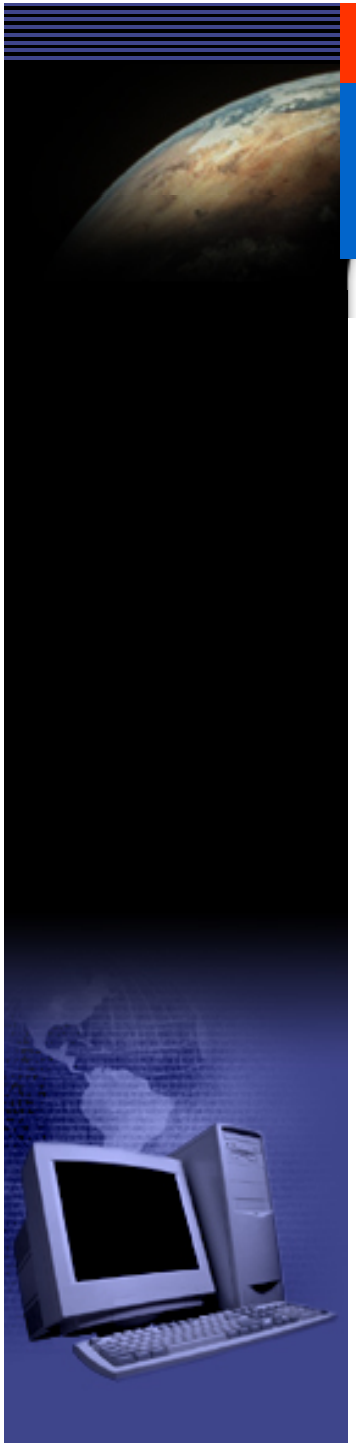
(6) Tài nguyên mà tiến trình yêu cầu đã sẵn sàng để cấp phát, hay sự kiện, thao tác nhập/xuất tiến trình đang đợi hoàn tất.



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:

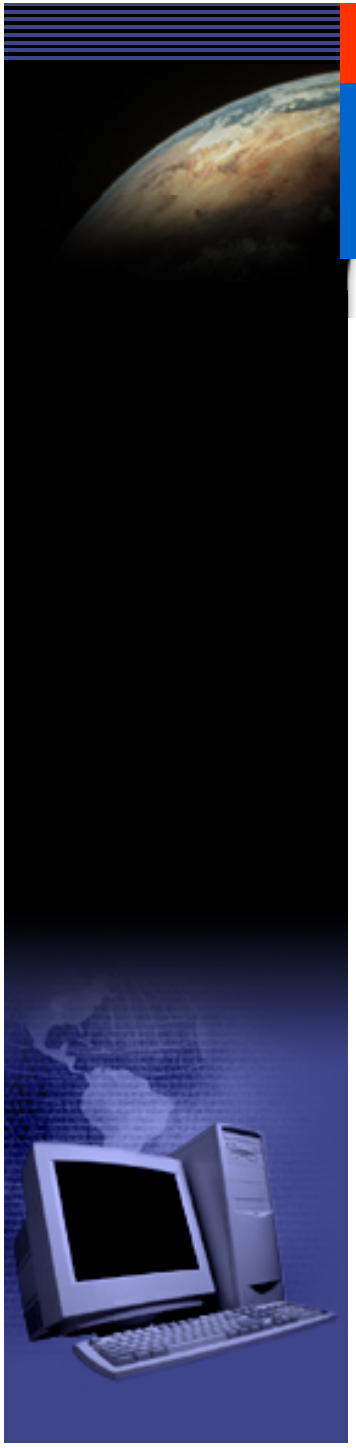
✓ Định danh cho tiến trình (PID)

✓ 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

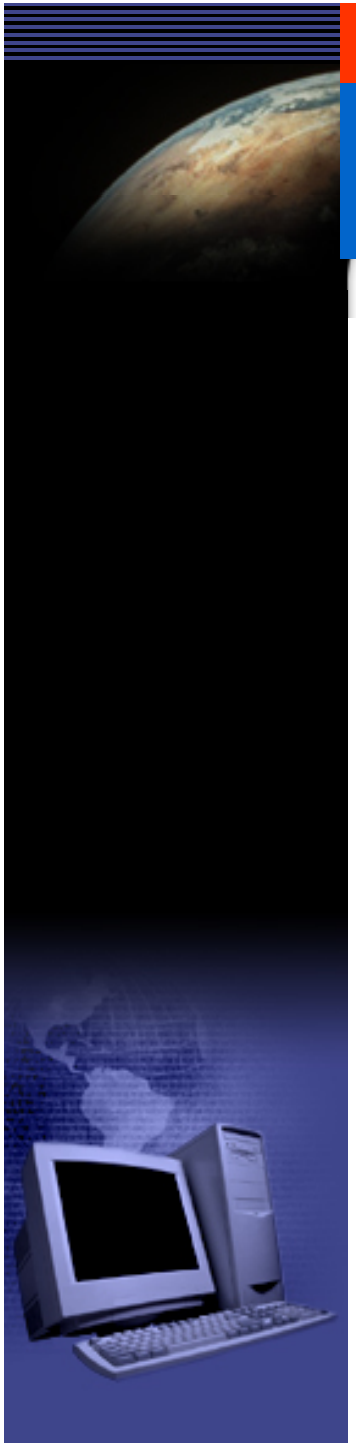


Thao tác trên tiến trình

❖ Kết thúc tiến trình

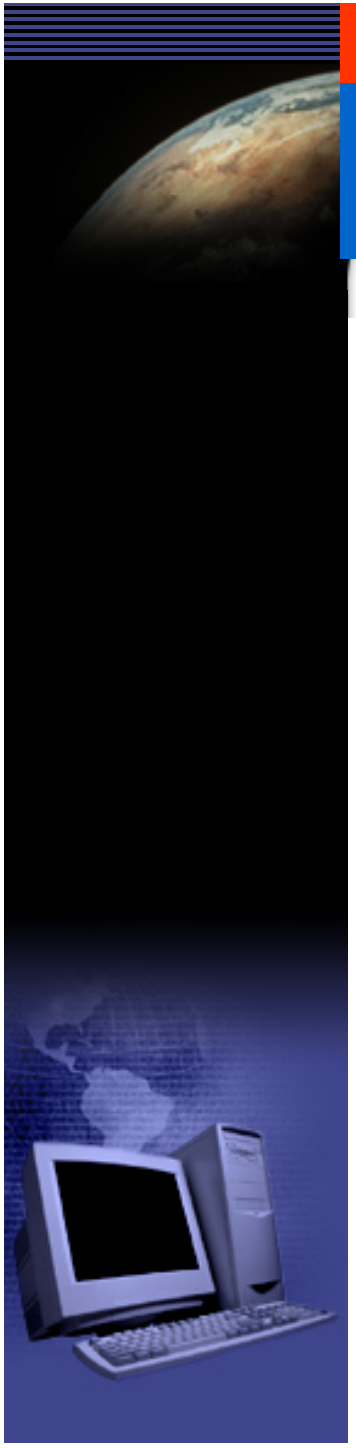
Khi tiến trình kết thúc, HĐH thực hiện:

- **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

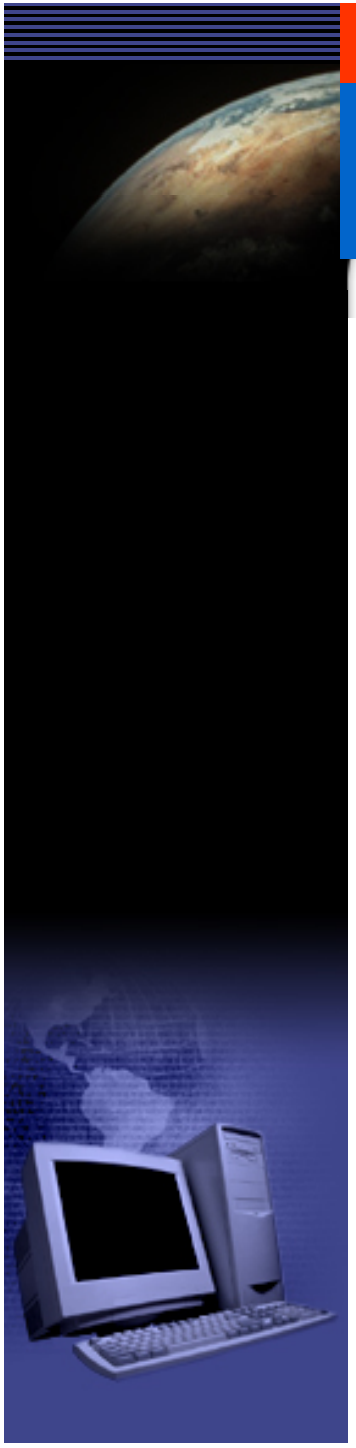
- Mục tiêu điều phối
- Tiêu chuẩn điều phối
- Điều phối không độc quyền, điều phối độc quyền
- Đồng hồ ngắt giờ
- Độ ưu tiên của tiến trình
- Tổ chức điều phối
- Các chiến lược điều phối



Điều phối tiến trình

❖ Mục tiêu điều phối

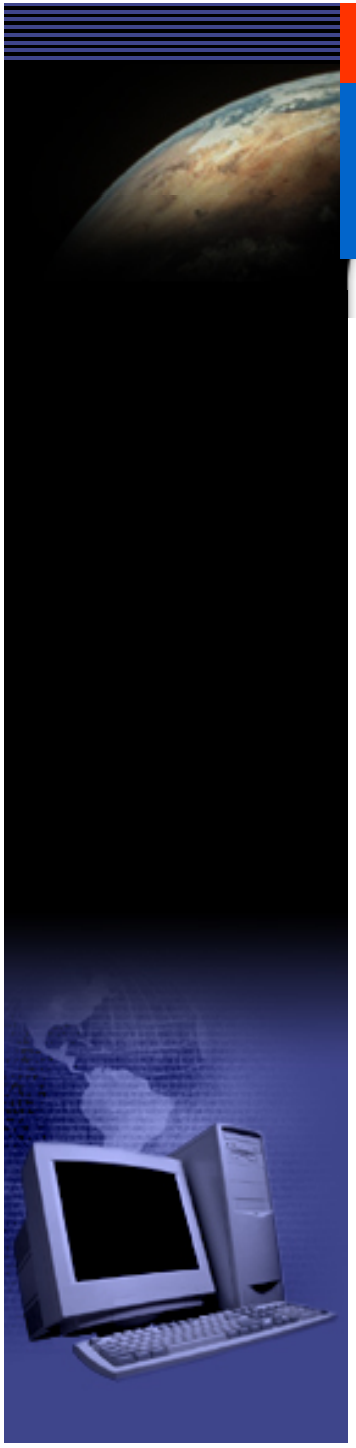
- Sự công bằng giữa các tiến trình
- Tính hiệu quả (tận dụng 100% thời gian sử dụng CPU)
- Cực tiểu hoá thời gian lưu lại trong hệ thống
- Thời gian đáp ứng hợp lý (cực tiểu hoá thời gian hồi đáp cho các tương tác của NSD)
- Thông lượng tối đa (cực đại hoá số công việc được xử lý trong một thời gian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



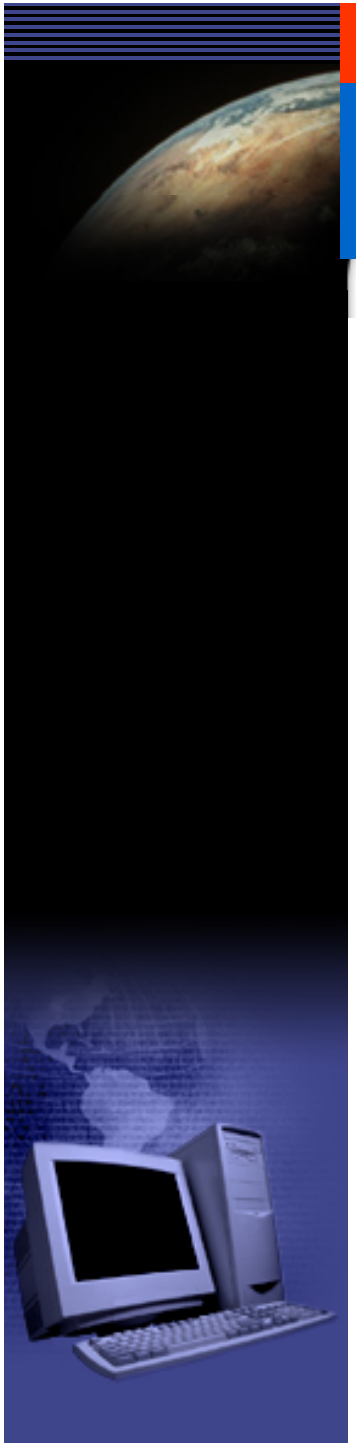
Điều phối tiến trình

❖ 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



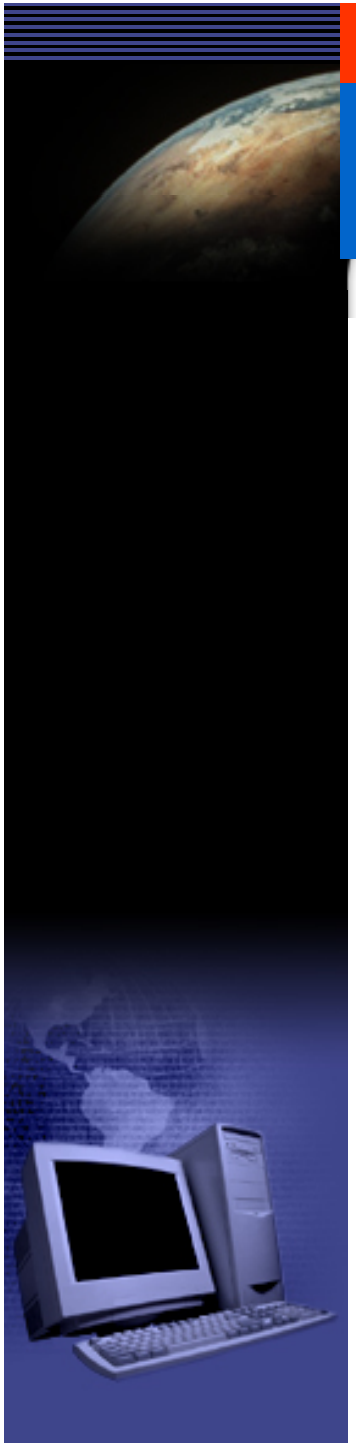
Điều phối tiến trình

❖ 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



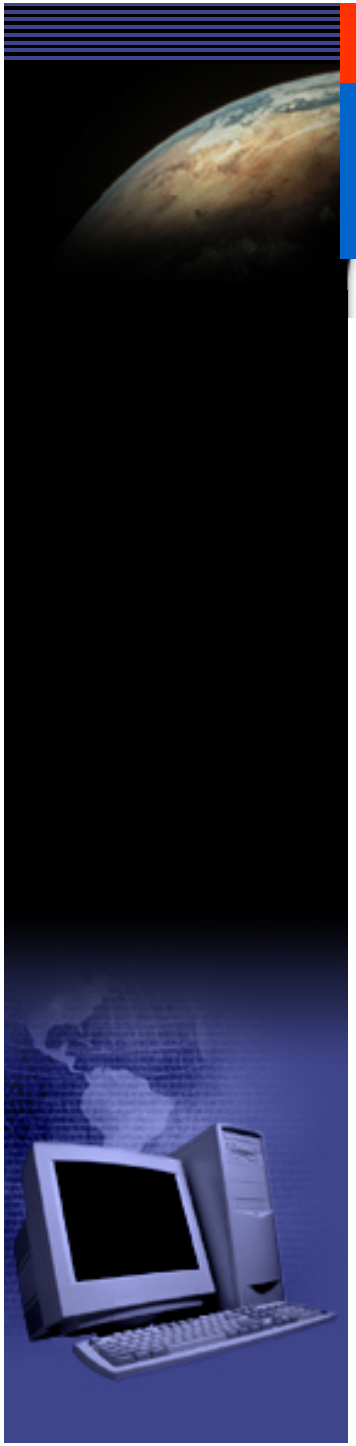
Điều phối tiến trình

❖ **Điều phối không độc quyền**

+ Tiến trình chuyển từ trạng thái blocked sang Ready

+ Tiến trình kết thúc

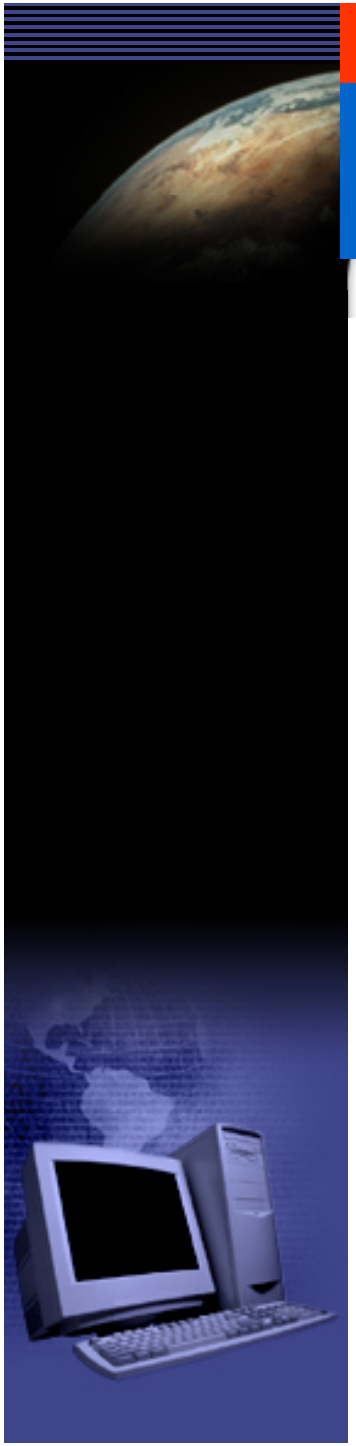
- **Ngăn cản được tình trạng các tiến trình độc chiếm CPU, nhưng việc tam dừng một tiến trình dẫn đến các mâu thuẫn trong truy xuất. Đòi hỏi phương pháp đồng bộ hoá thích hợp**



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.



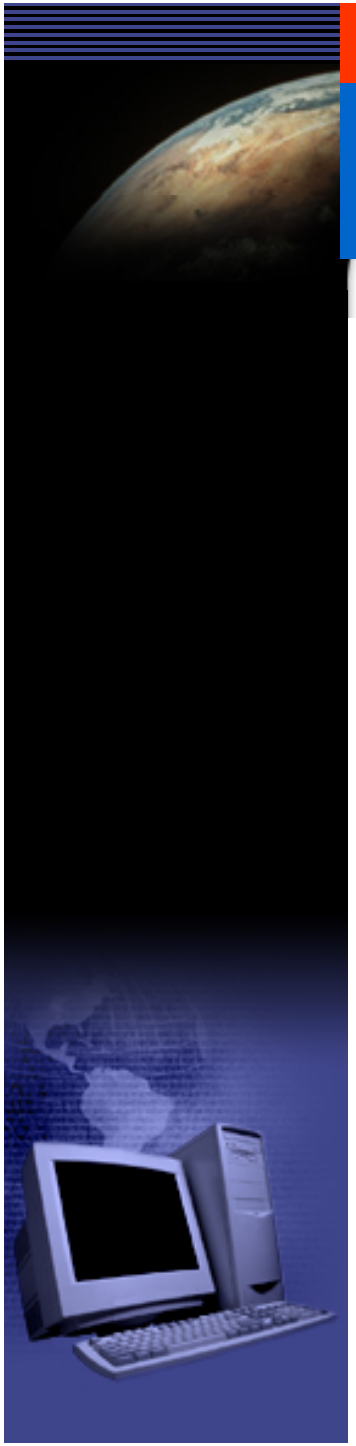
Điều phối tiến trình

❖ 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



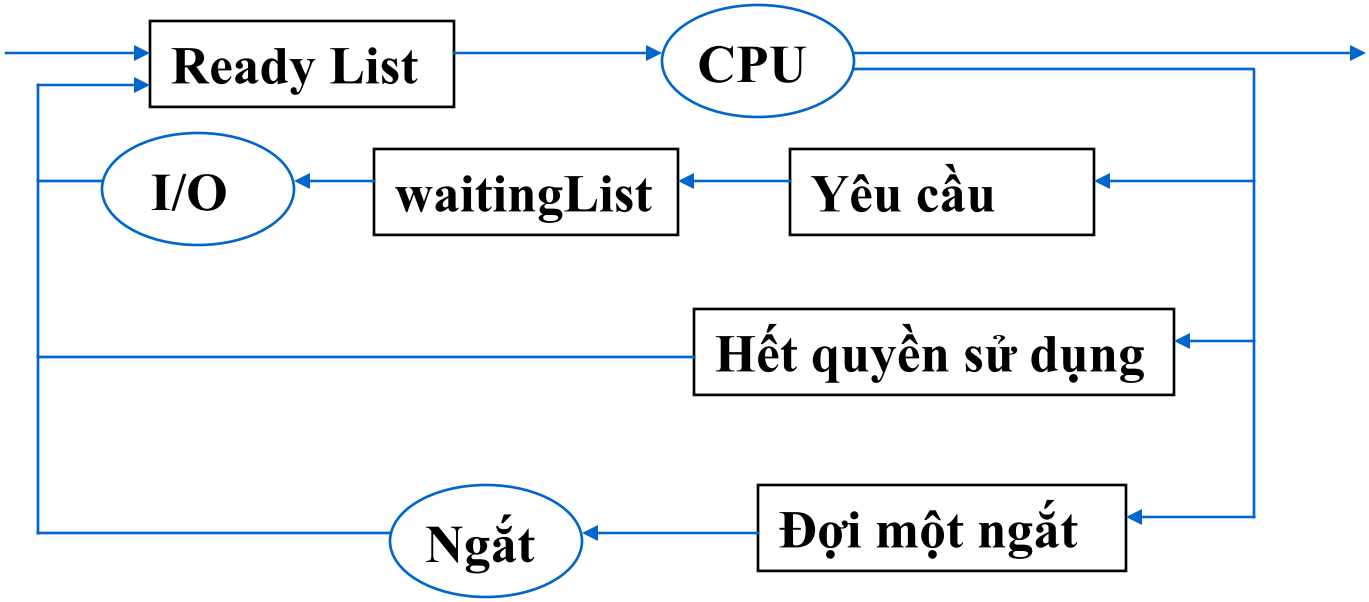
Điều phối tiến trình

❖ Tổ chức điều phối

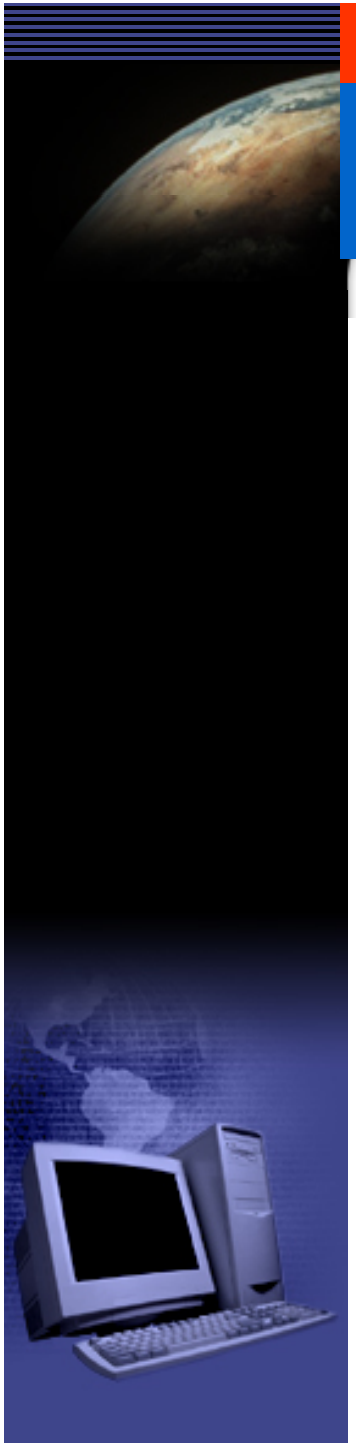
- Danh sách sẵn sàng (Ready List)
- Danh sách chờ đợi (Waiting List)
- Các danh sách chờ đợi riêng cho từng tài nguyên (thiết bị ngoại vi)

Điều phối tiến trình

❖ Tổ chức điều phối



Sơ đồ chuyển đổi giữa các danh sách điều phối



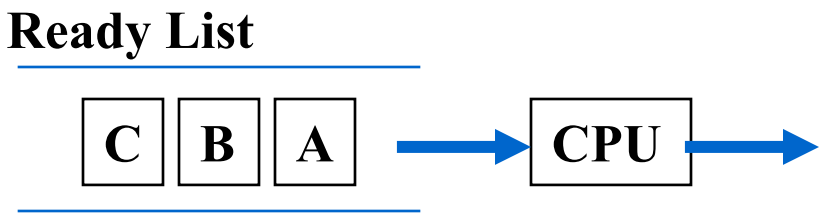
Điều phối tiến trình

❖ 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

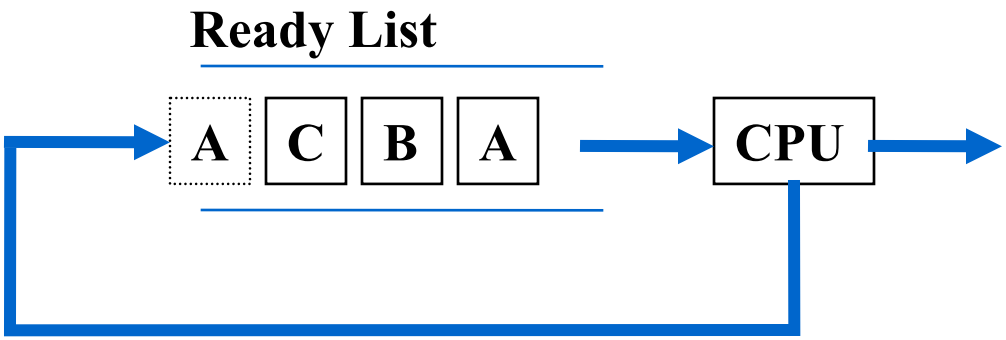
❖ Chiến lược điều phối



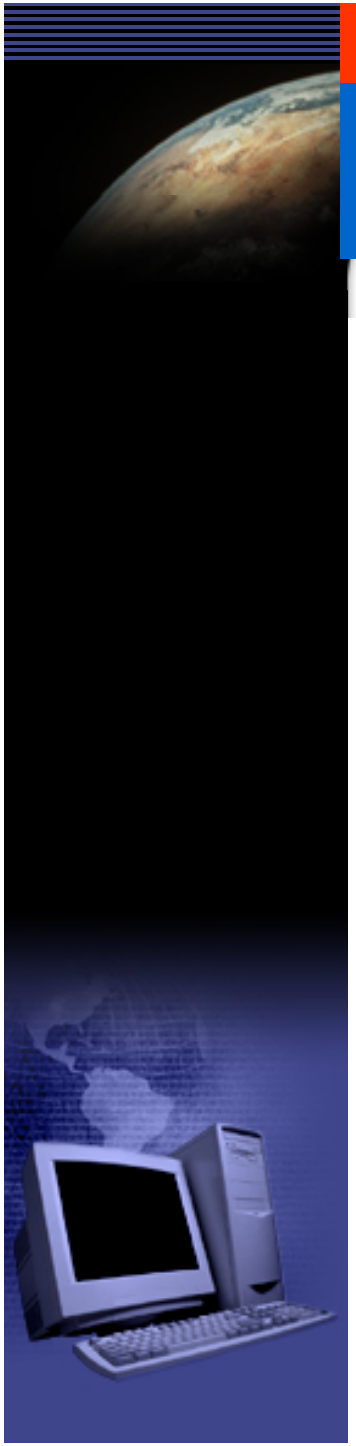
Điều phối FIFO

Điều phối tiến trình

❖ Chiến lược điều phối



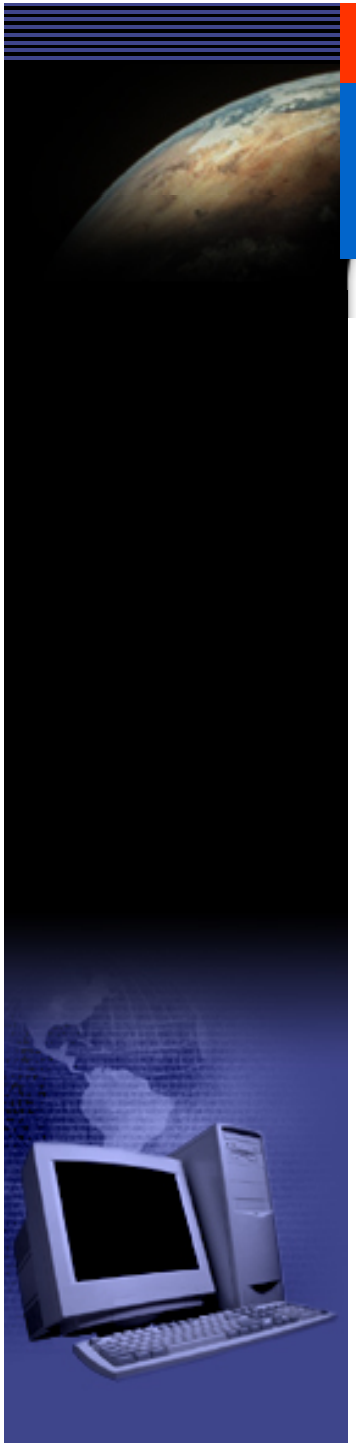
Điều phối Round Robin



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

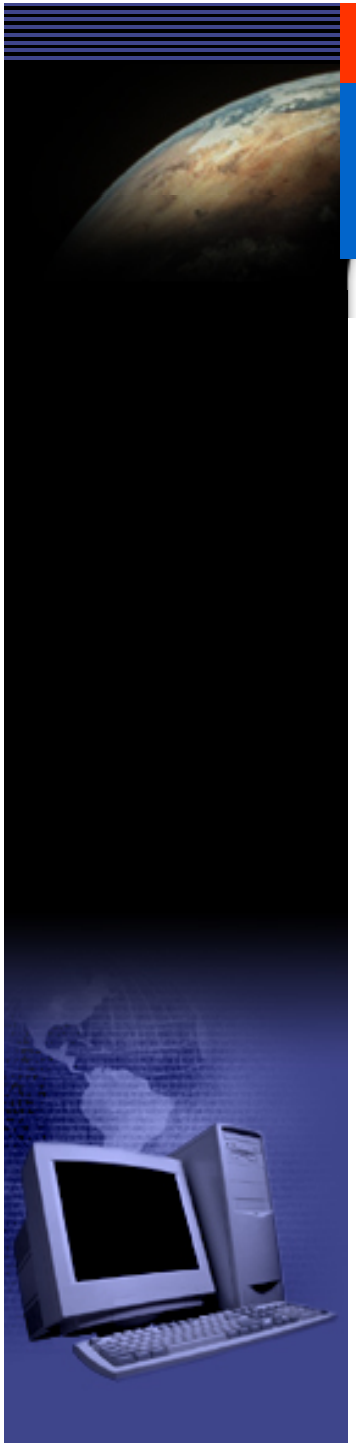
❖ Miền găng (Critical Section)

- **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

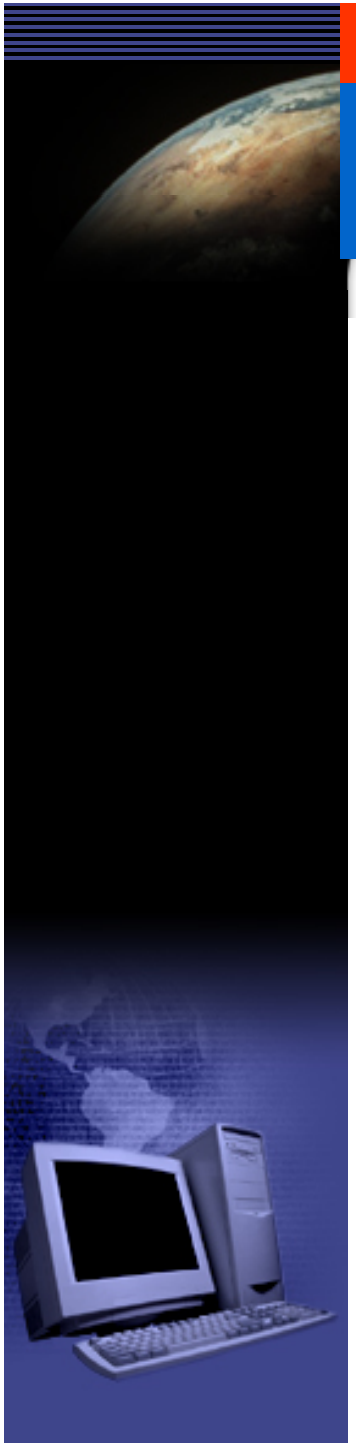


Đồng bộ hoá tiến trình

❖ 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

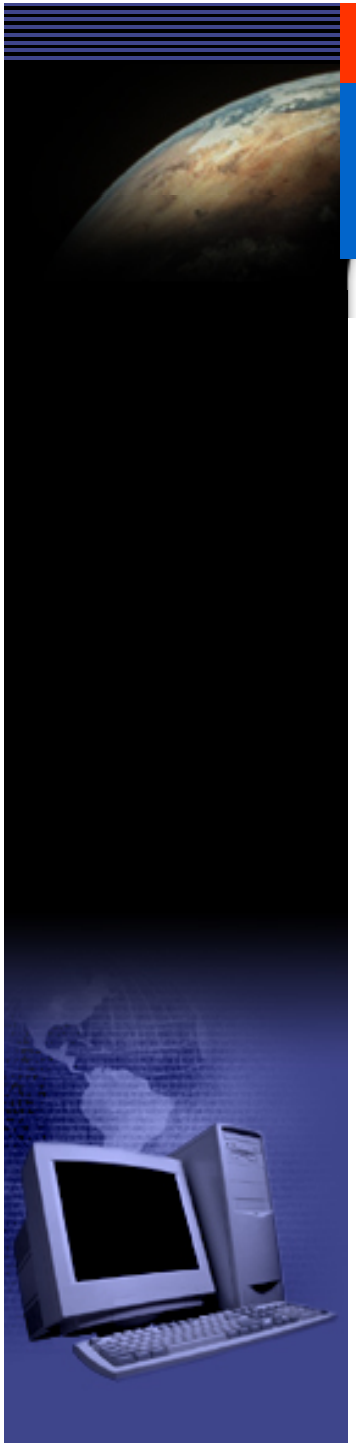
❖ Giải pháp

➢ 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



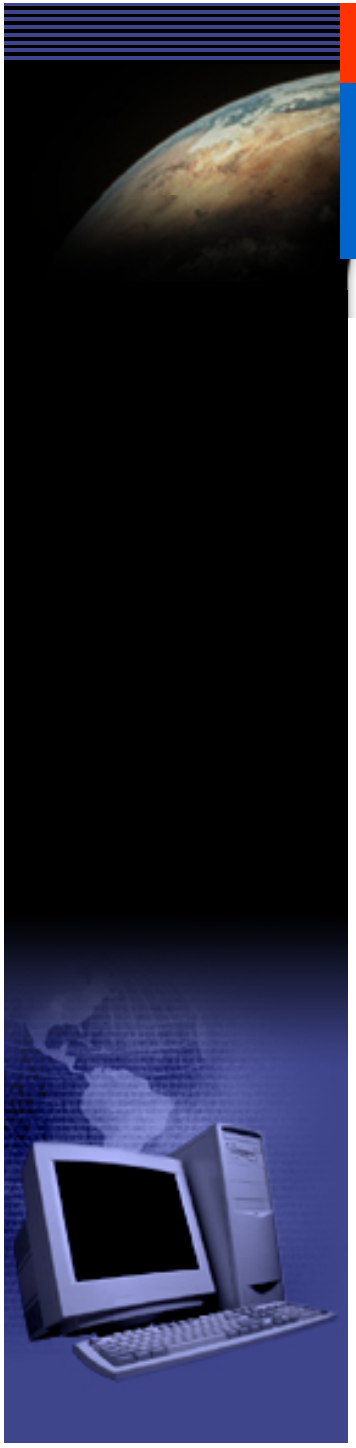
Đồng bộ hoá tiến trình

❖ Giải pháp

➢ Sử dụng biến khoá

- Giải thuật sử dụng biến khoá để đồng bộ

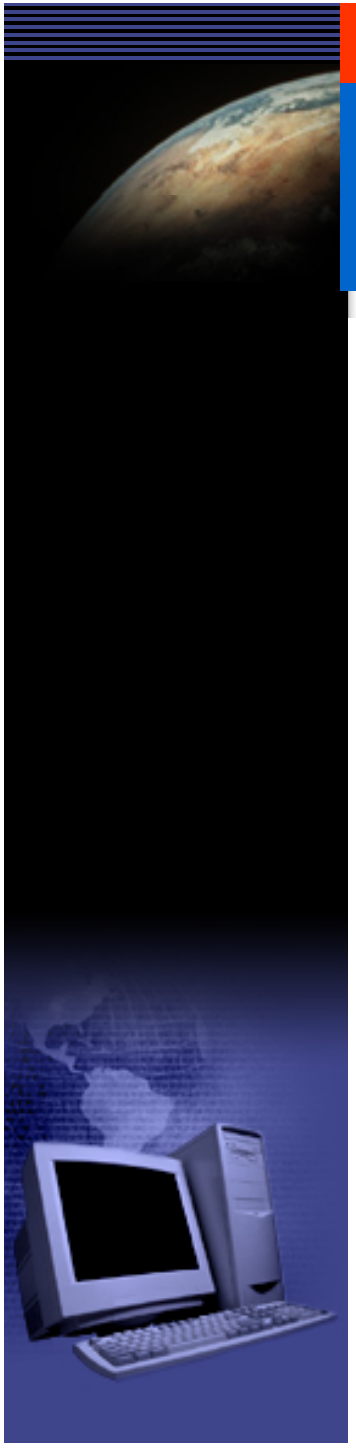
```
while (1) {
    while (lock==1);// wait
    lock=1;
    critical_section();
    lock=0;
    Noncritical_section();
}
```



Đồng bộ hoá tiến trình

- Giải thuật sử dụng biến khoá để đồng bộ

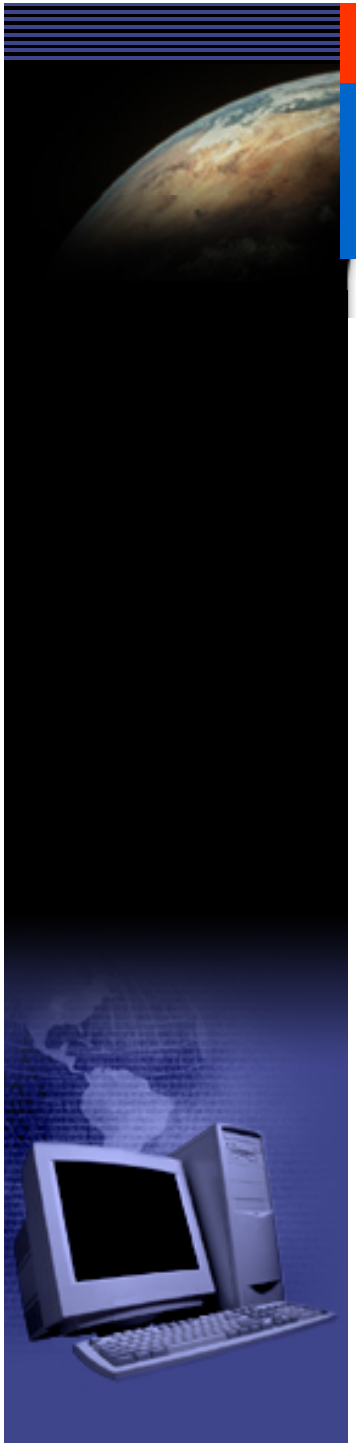
```
while (1) {
    while (lock==1);// wait
    lock=1;
    critical_section();
    lock=0;
    non_critical_section();
    }
```



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

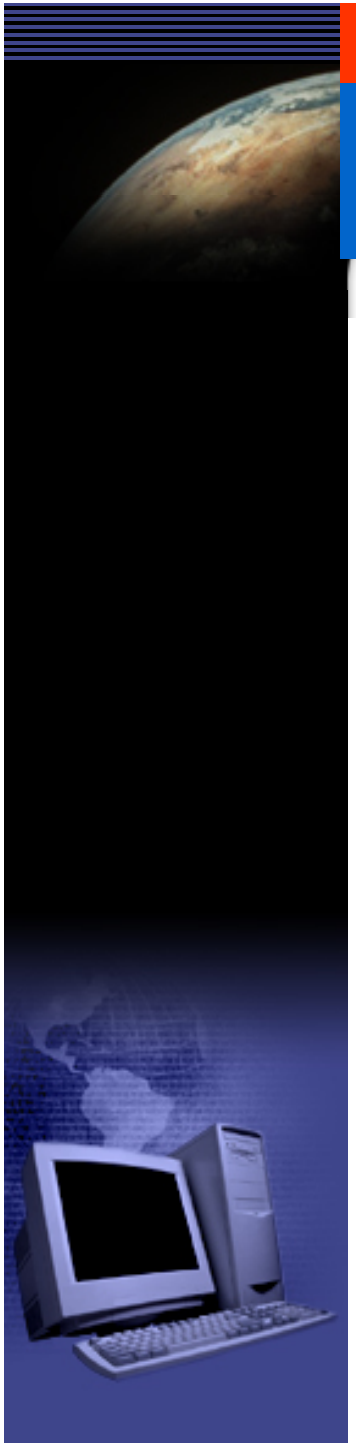
➢Kiểm tra luân phiên

- 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

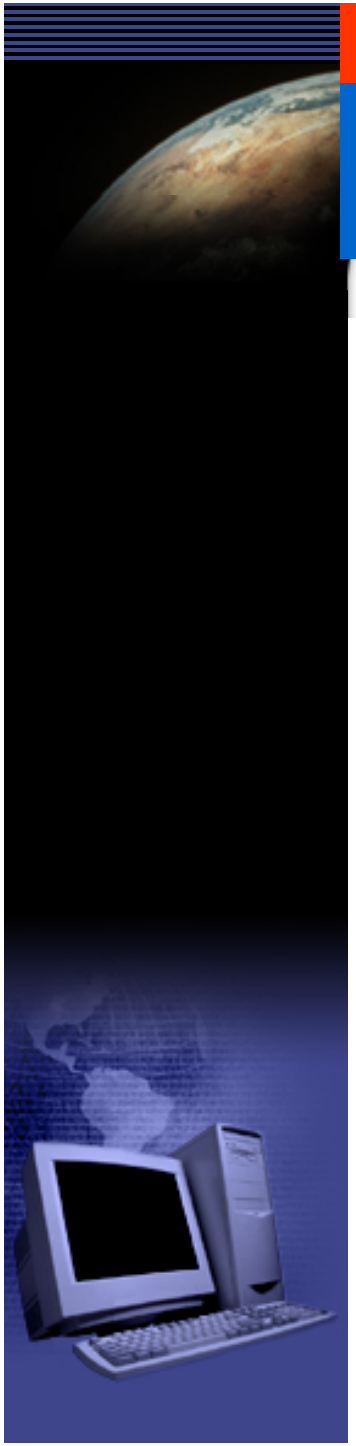
```
#define N  2  // Chỉ 2 tiến trình
int turn=0, interested[N]={0,0};
void enter_region(int process) // Vào ĐG
{  int other=1-process;//other là tiến trình đối của process
   interested[prcess]=1;
   turn=process;
   while ((turn==process)&&interested[other]==1);//chờ
}
```

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

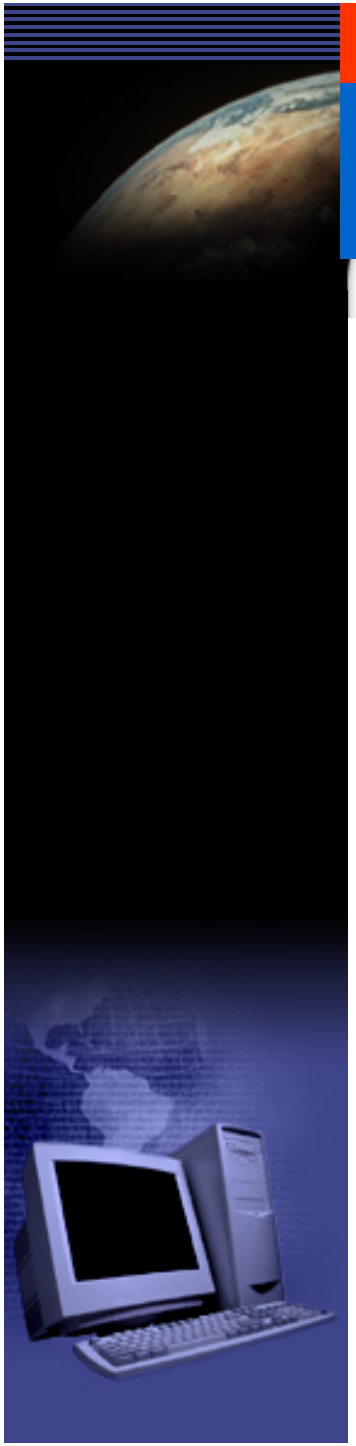
Đồng bộ hoá tiến trình

❖ 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];

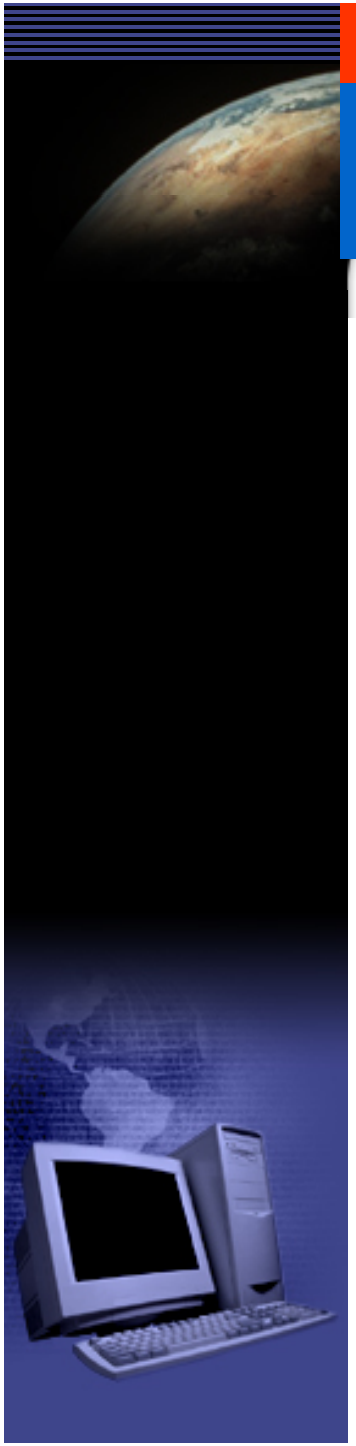
// max[p,r] nhu cầu tối đa của tiến trình p về tài nguyên r

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

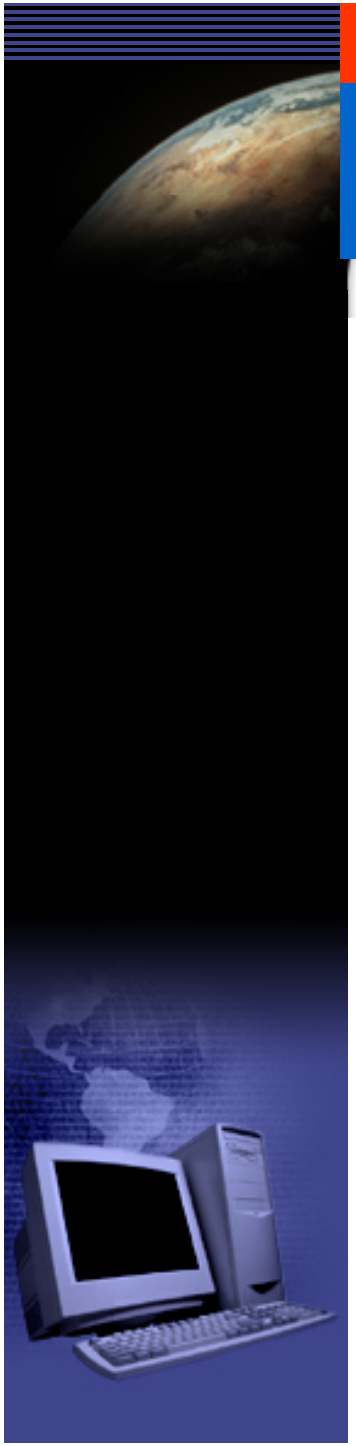
   nếu không có i như thế, đến bước 3

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.

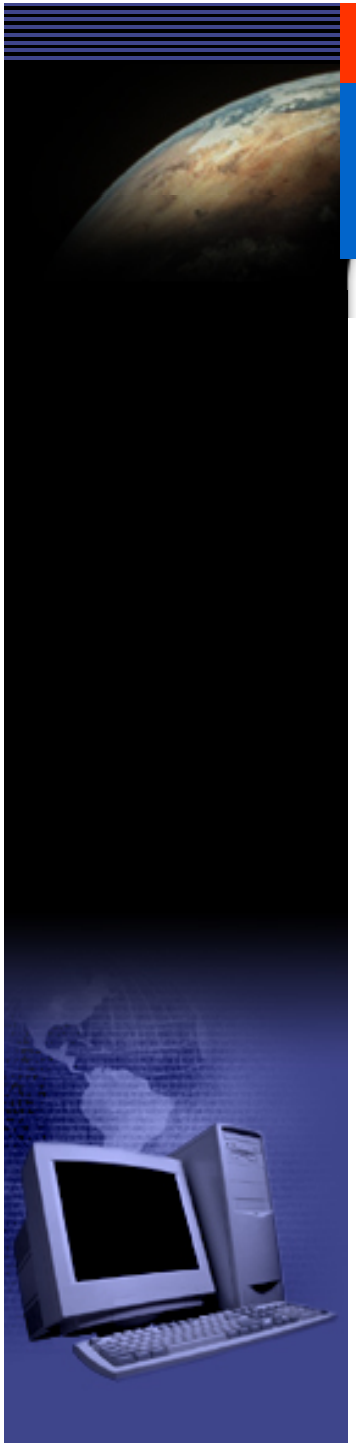
Xác định trạng thái an toàn

❖ **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			

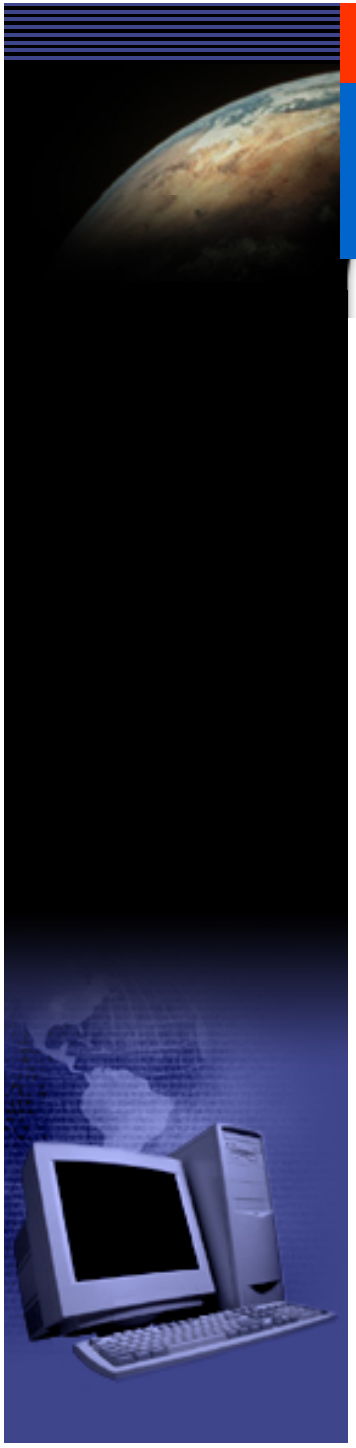
Tiến trình P2 yêu cầu 4 R1, 1 R3. Hãy cho biết yêu cầu này có thể đáp ứng mà đảm bảo không xảy ra tình trạng tắt nghẽn hay không?

CHƯƠNG 4. QUẢN LÝ BỘ NHỚ



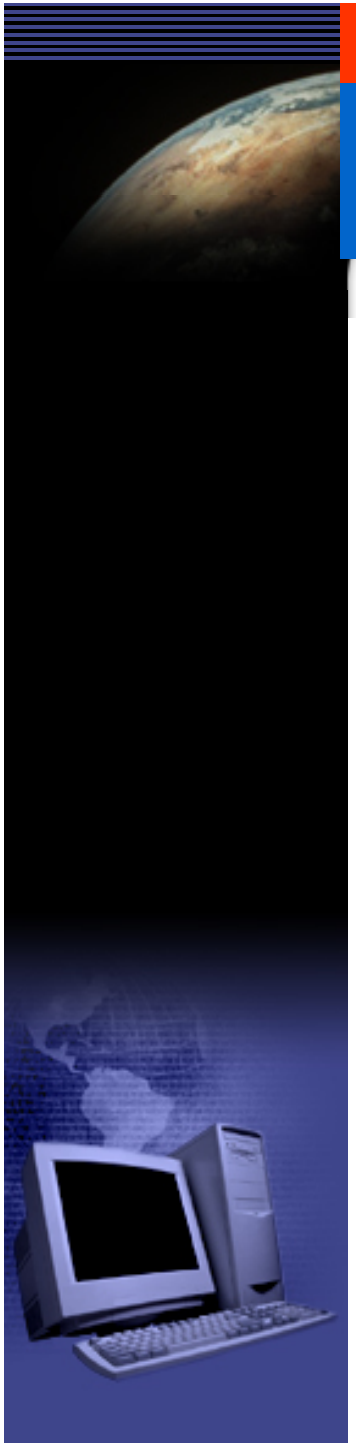
Các vấn đề

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

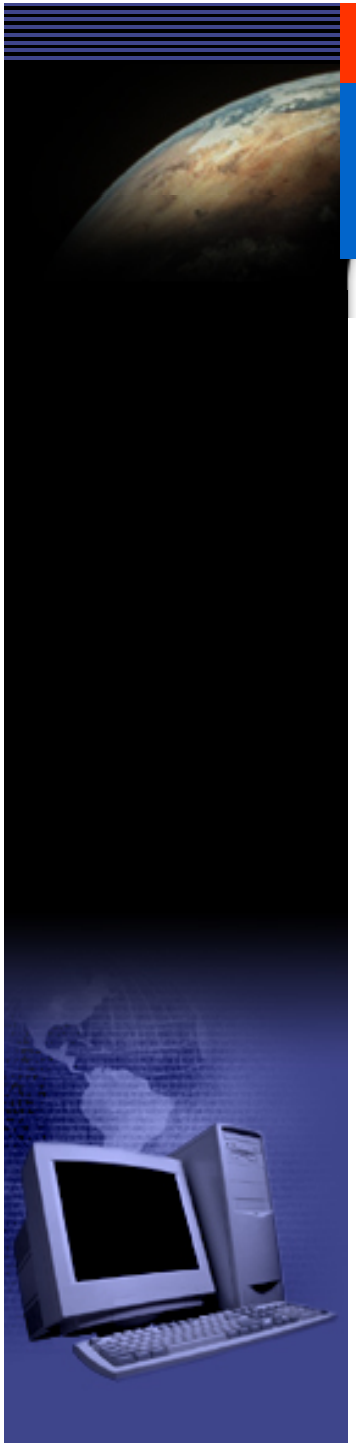
- **Hệ điều hành thực hiện:**
  - **Sự tương ứng giữa địa chỉ logic và địa chỉ vật lý**
  - **Quản lý bộ nhớ vật lý**
  - **Chia sẻ thông tin**
  - **Bảo vệ**

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.**

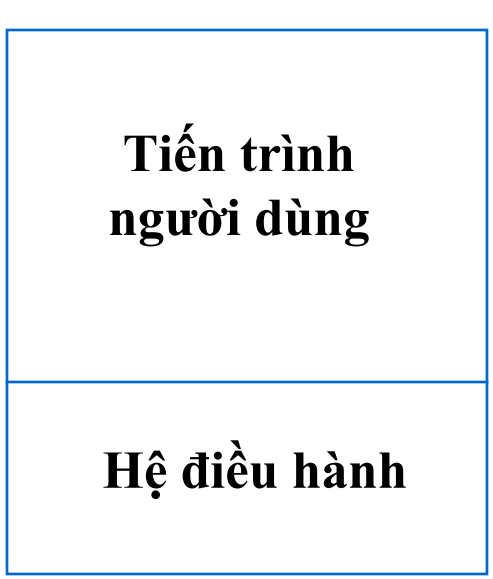


Cấp phát liên tục

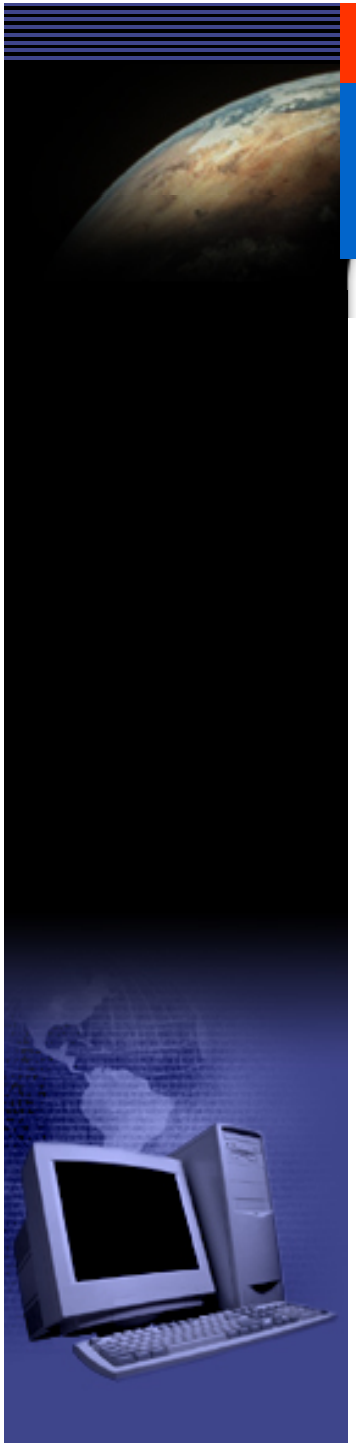
- Các hệ đơn chương
- Các hệ thống đa chương với phân vùng cố định
- Các hệ thống đa chương với phân vùng động
- Các hệ thống đa chương với kỹ thuật “Swapping”

Cấp phát liên tục

- Các hệ đơn chương



Tổ chức bộ nhớ trong hệ thống đơn chương



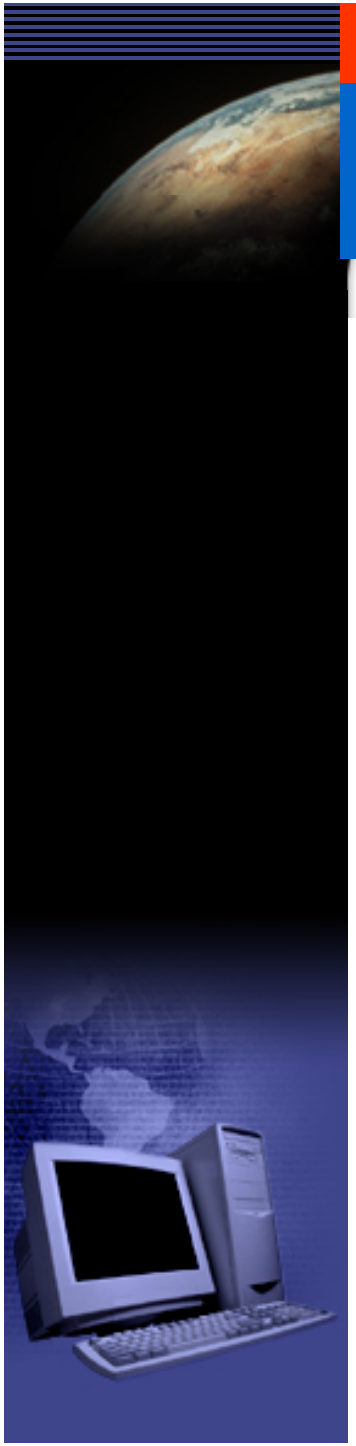
Cấp phát liên tục

❖ **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.

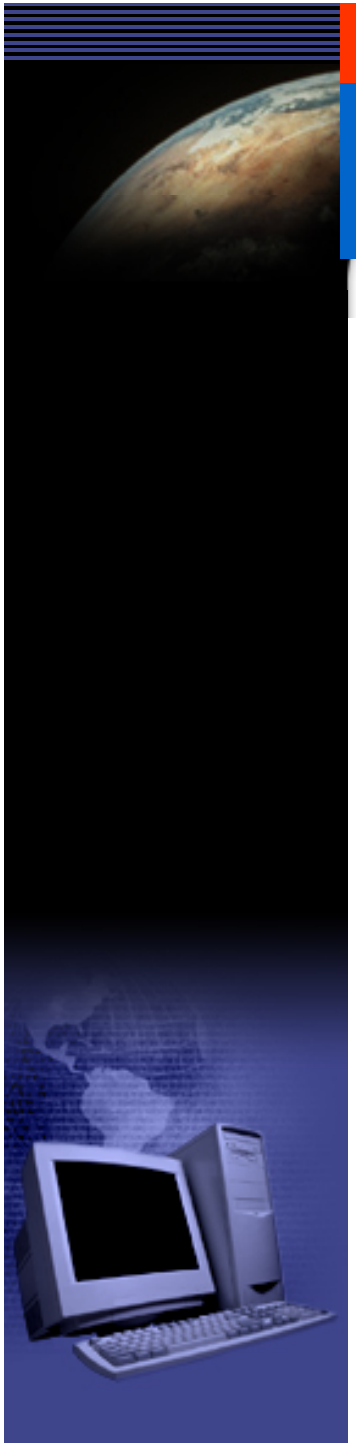
  + Ngược lại : một ngắt sẽ được phát sinh báo sự truy xuất bất hợp lý.
- **Tại một thời điểm chỉ có một chương trình được xử lý.**



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.

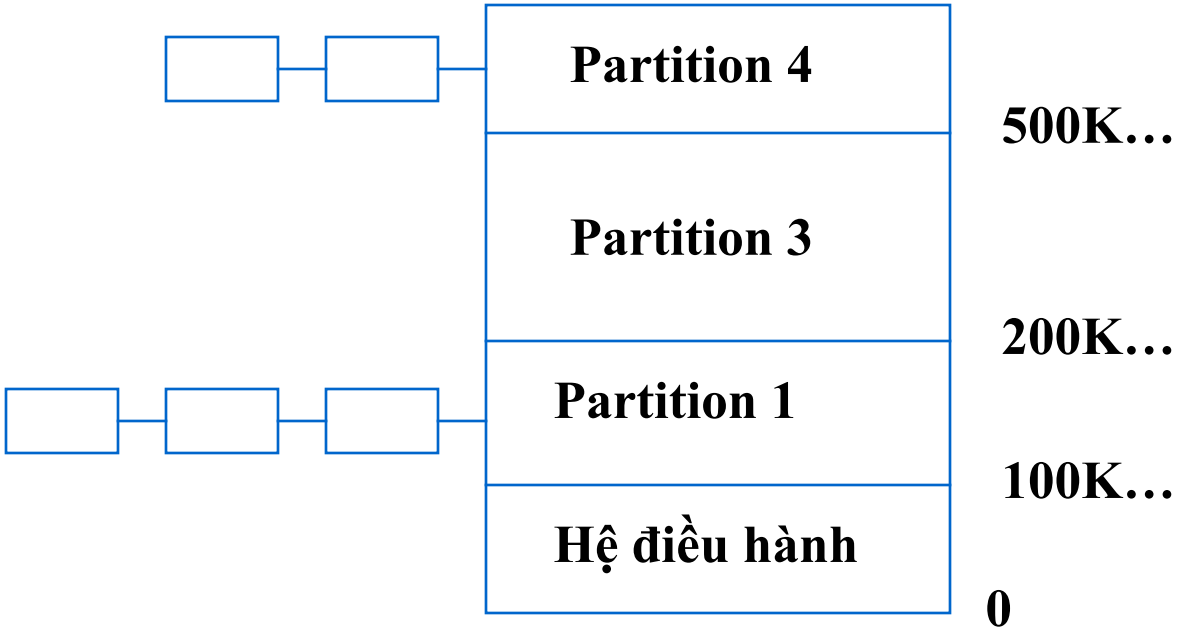


Cấp phát liên tục

- 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

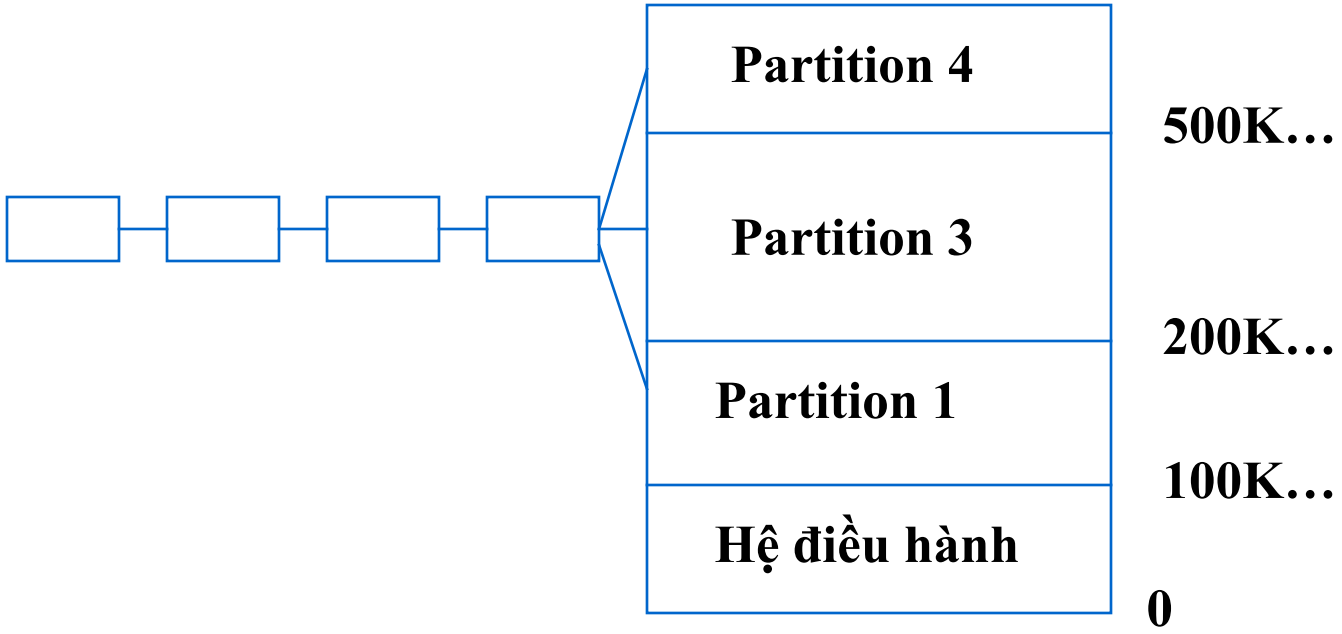
- Các hệ thống đa chương với phân vùng cố định



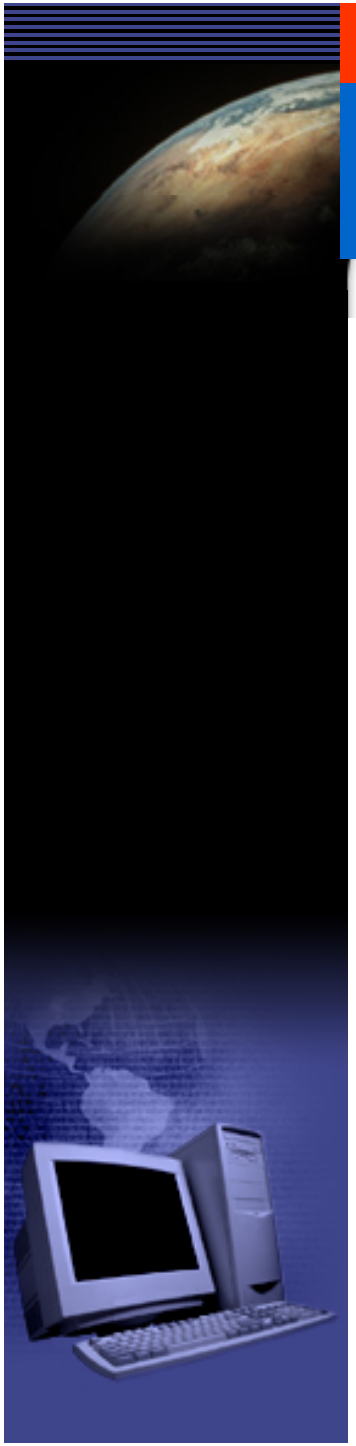
Phân vùng cố định nhiều hàng đợi

Cấp phát liên tục

- Các hệ thống đa chương với phân vùng cố định

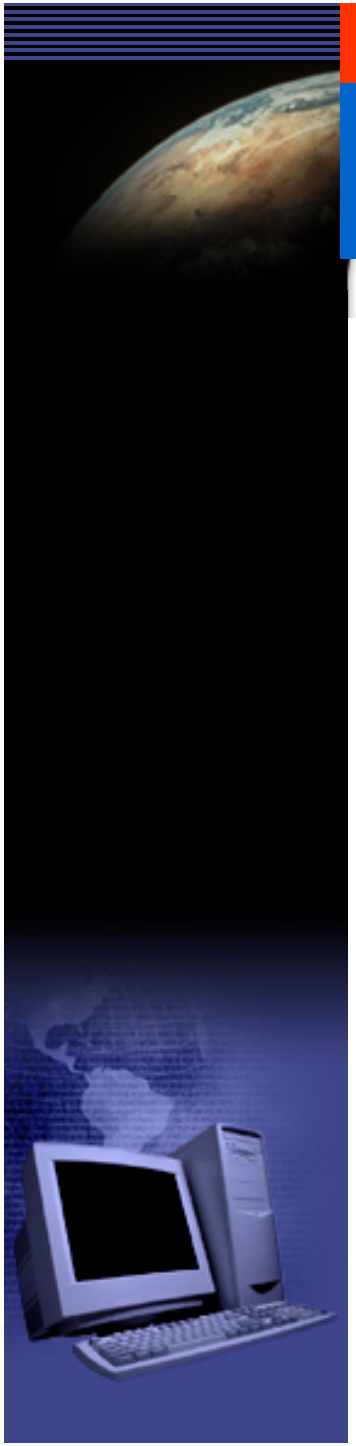


Phân vùng cố định một hàng đợi



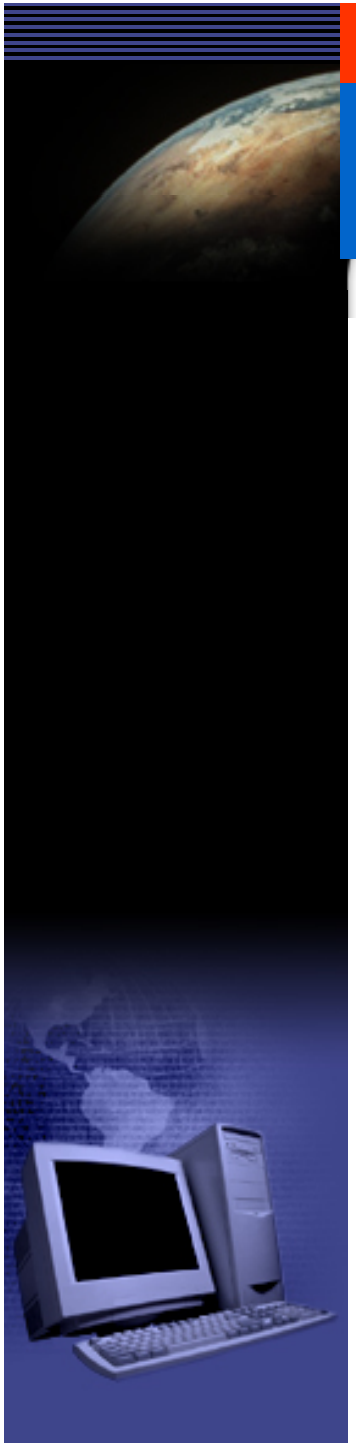
Cấp phát liên tục

- 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

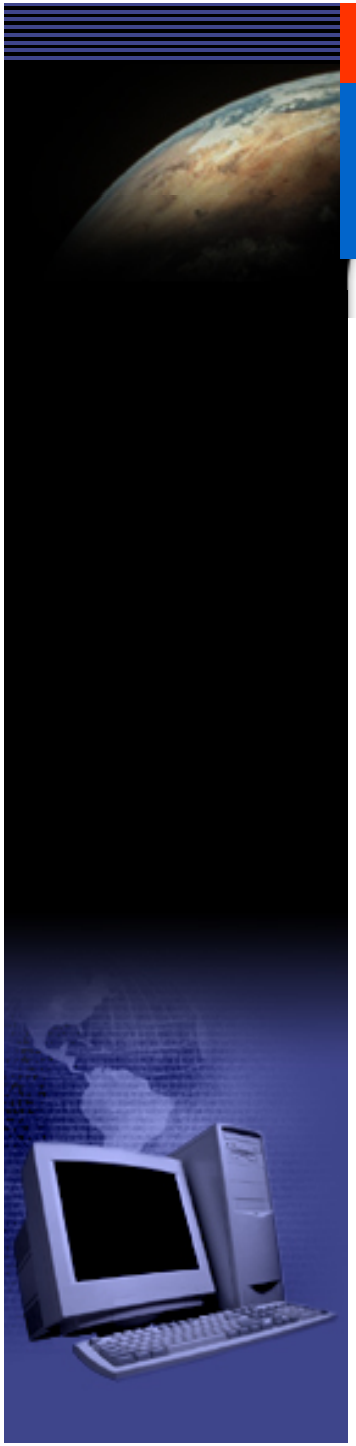


Cấp phát liên tục

- 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

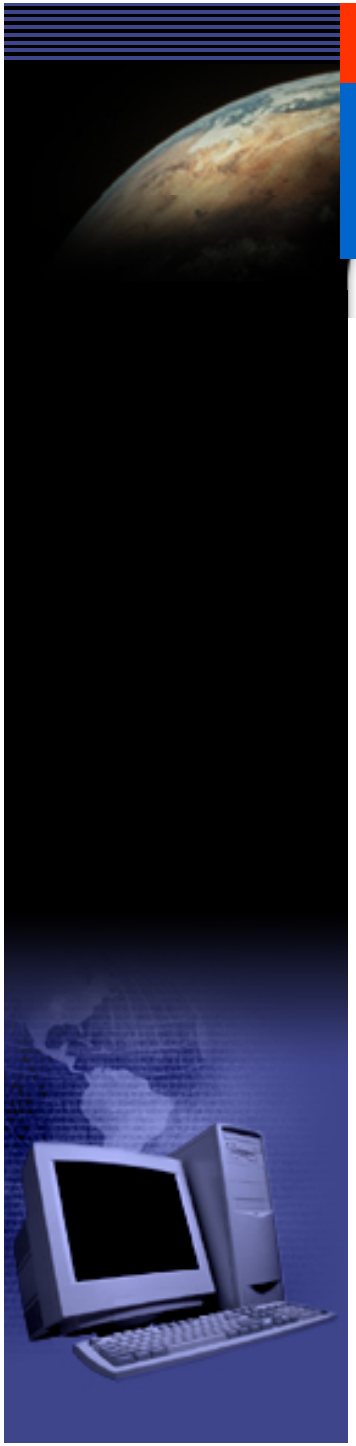
Ví dụ: giả sử chương trình truy xuất đến địa chỉ 100 (địa chỉ tương đối), ct được nạp vào phân vùng 1 địa chỉ bắt đầu 100k, thì địa chỉ truy xuất là (100k+100)

- 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.

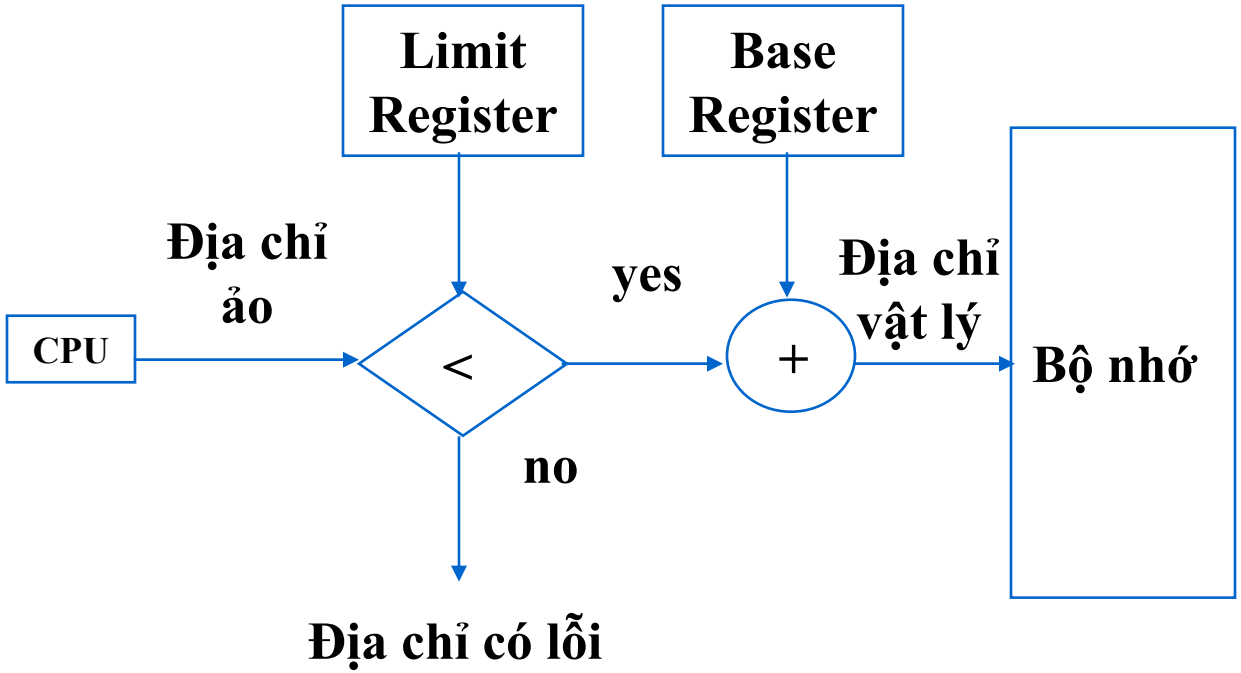


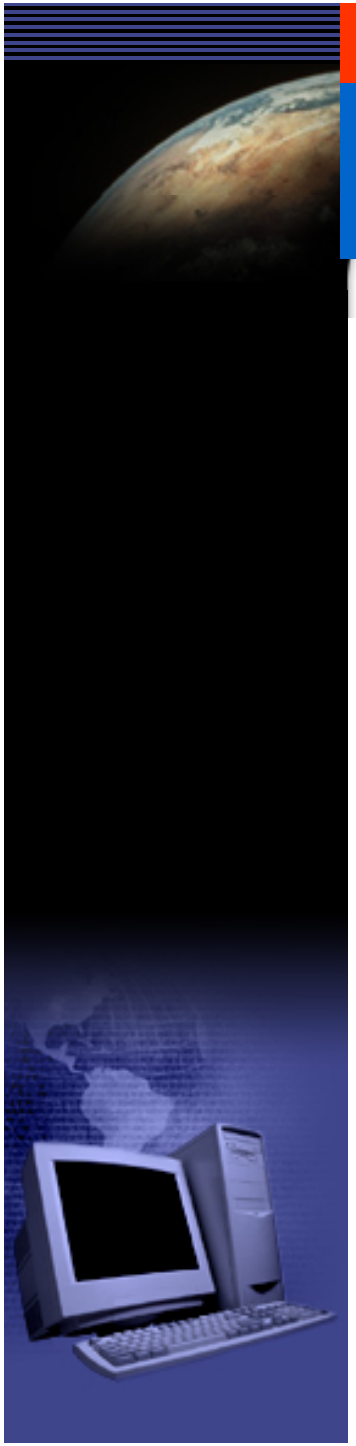
Cấp phát liên tục

- 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**

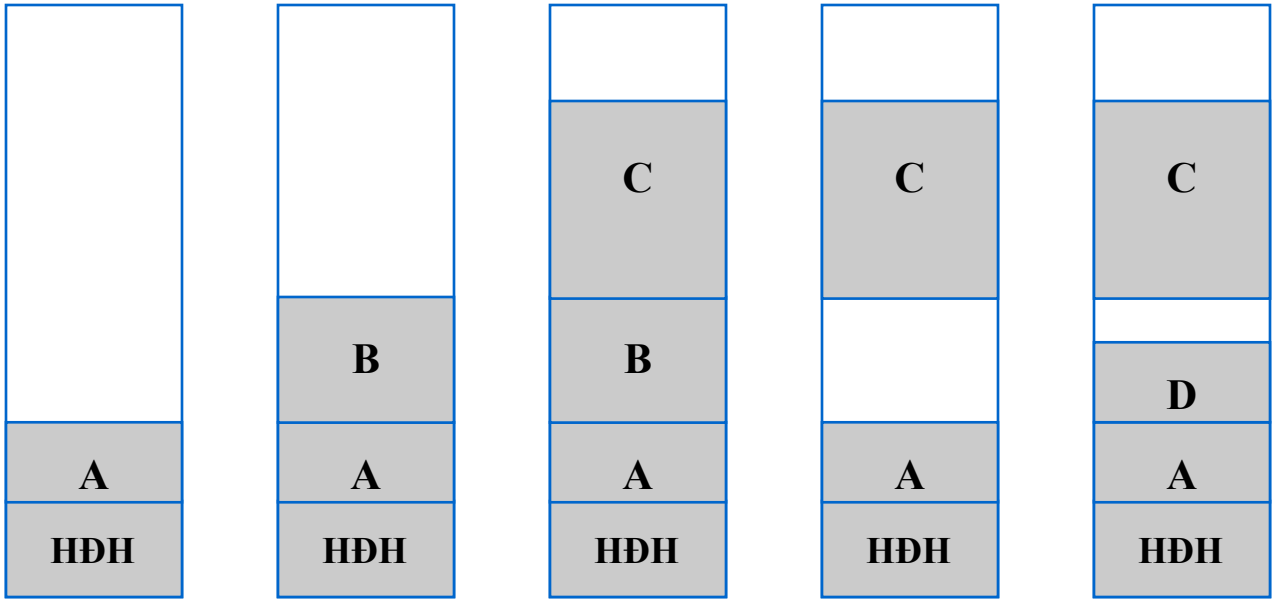
- **Xảy ra hiện tượng phân mảnh ngoại vi**
- **Kỹ thuật "dồn bộ nhớ": kết hợp các mảnh bộ nhớ nhỏ rời rạc thành một vùng nhớ lớn liên tục**

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

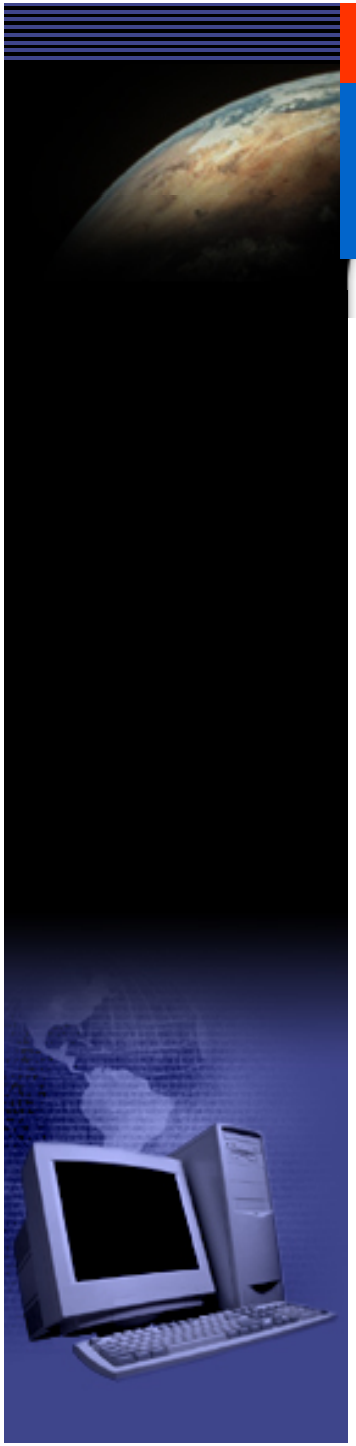
⇒ **Kích thước tiến trình tăng trưởng trong quá trình xử lý mà không còn vùng nhớ trống gần kề (dời chỗ tiến trình, cấp phát dư).**

Cấp phát liên tục

- Các hệ thống đa chương với phân vùng động



Cấp phát các phân vùng động

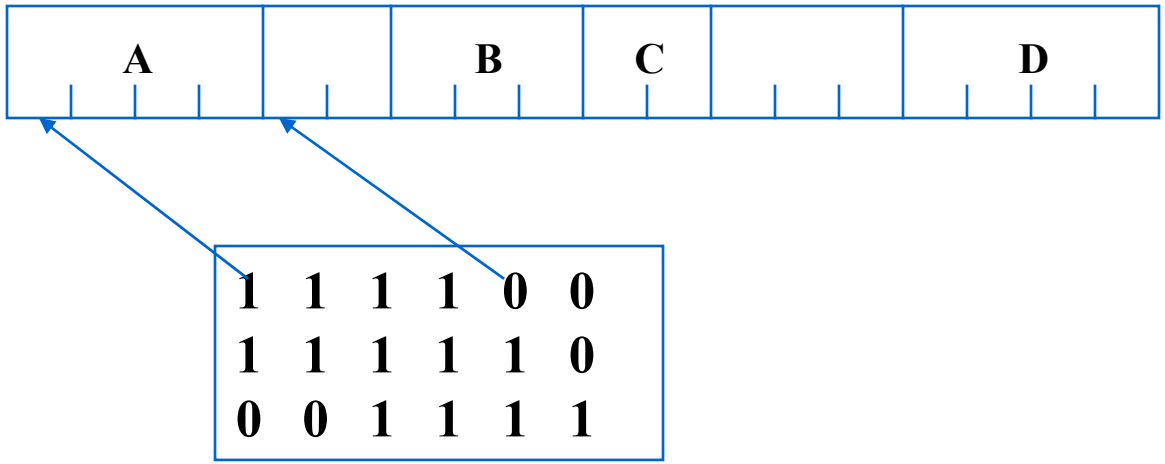


Cấp phát liên tục

- 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

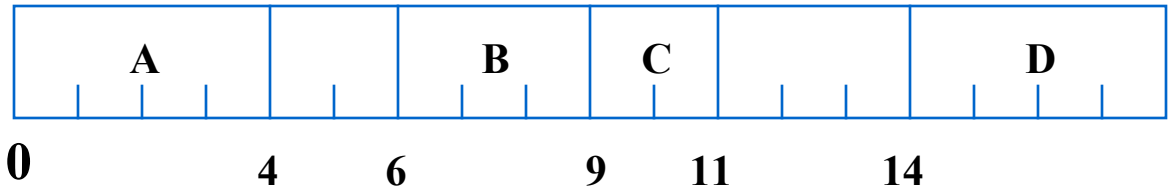
Cấp phát liên tục

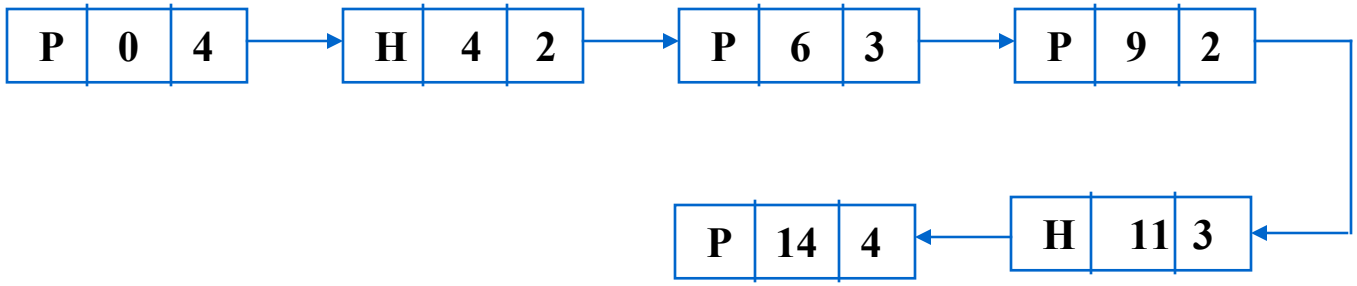
- Các hệ thống đa chương với phân vùng động
- Quản lý bằng một bảng các bit



Cấp phát liên tục

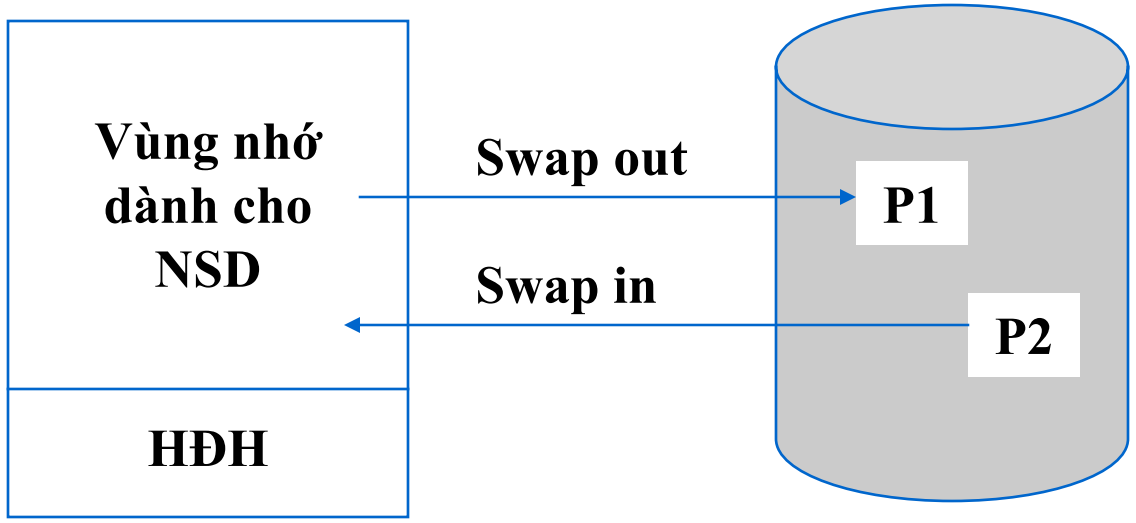
- Các hệ thống đa chương với phân vùng động
- Quản lý bằng danh sách

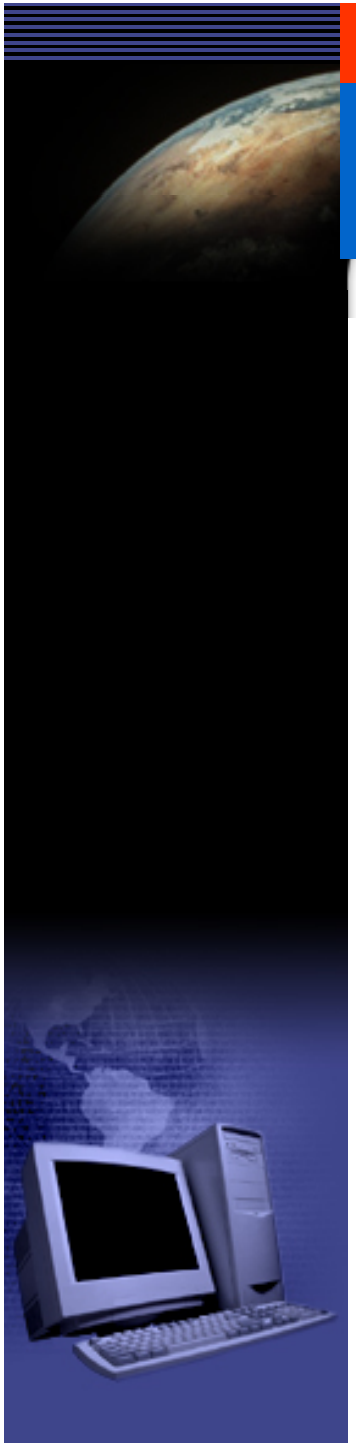




Cấp phát liên tục

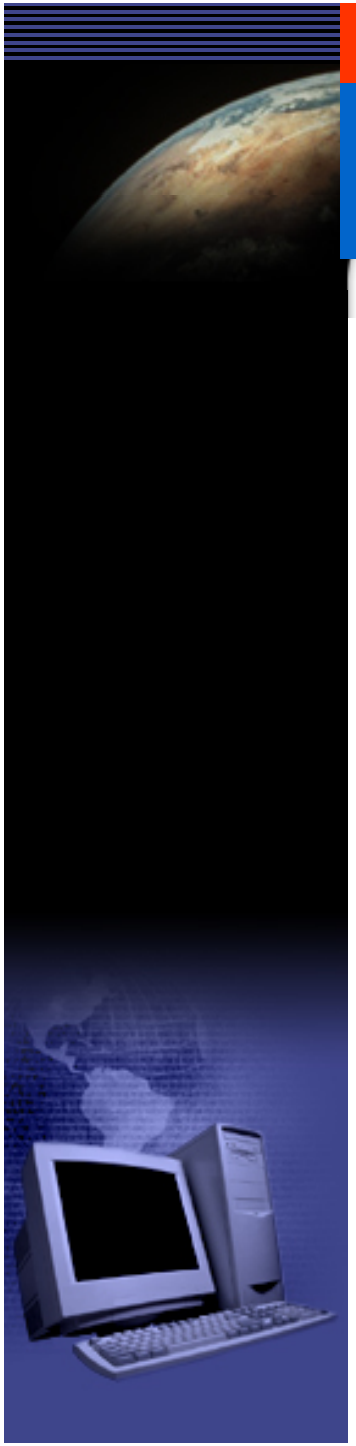
- Các hệ thống đa chương với kỹ thuật "Swapping"





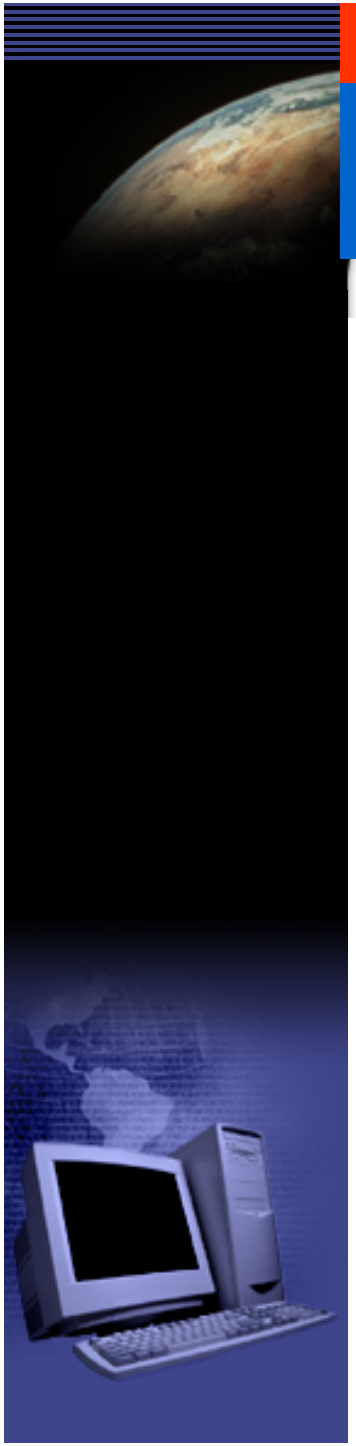
Cấp phát liên tục

- 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

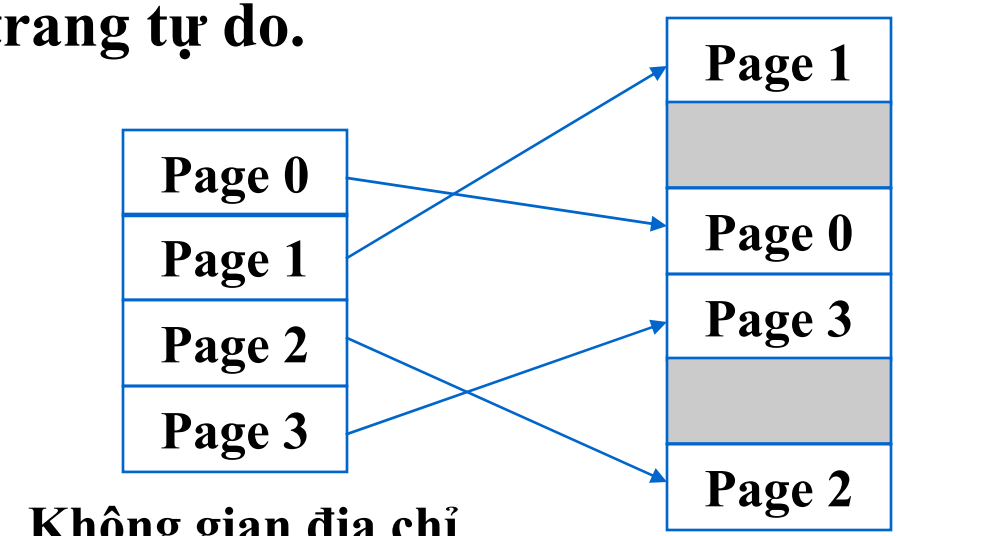
- **Phân trang**
  - **Ý tưởng**
  - **Cơ chế MMU**
  - **Chuyển đổi địa chỉ**
  - **Cài đặt bảng trang**
  - **Tổ chức bảng trang**

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.**

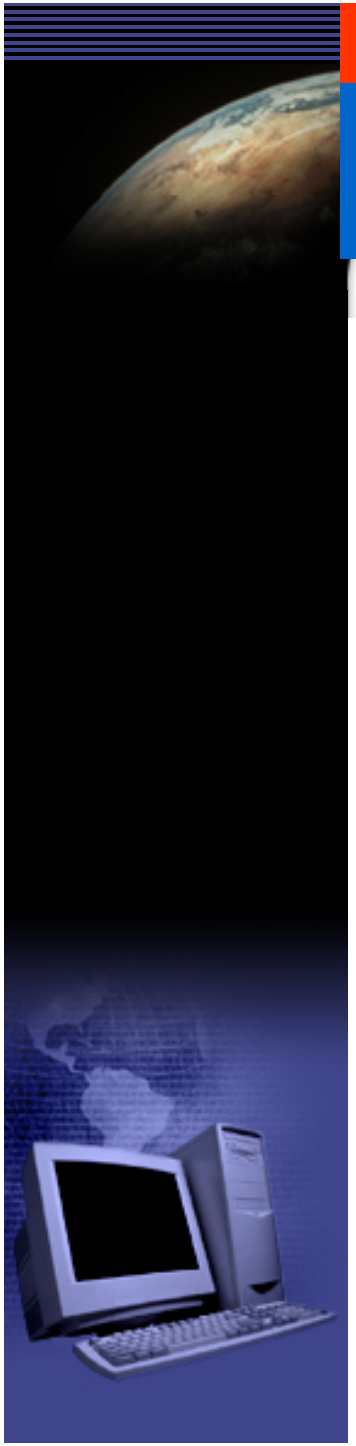
Cấp phát không liên tục

- **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

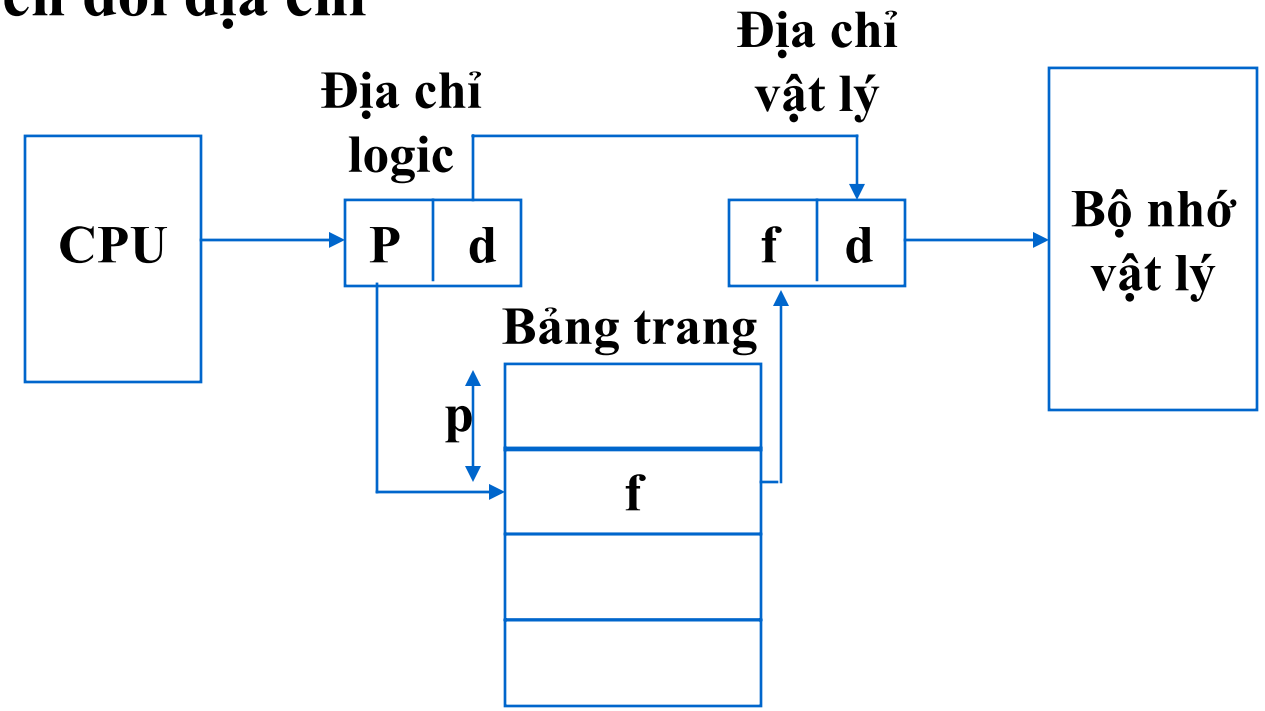
+ p: số hiệu trang

+ d: địa chỉ tương đối

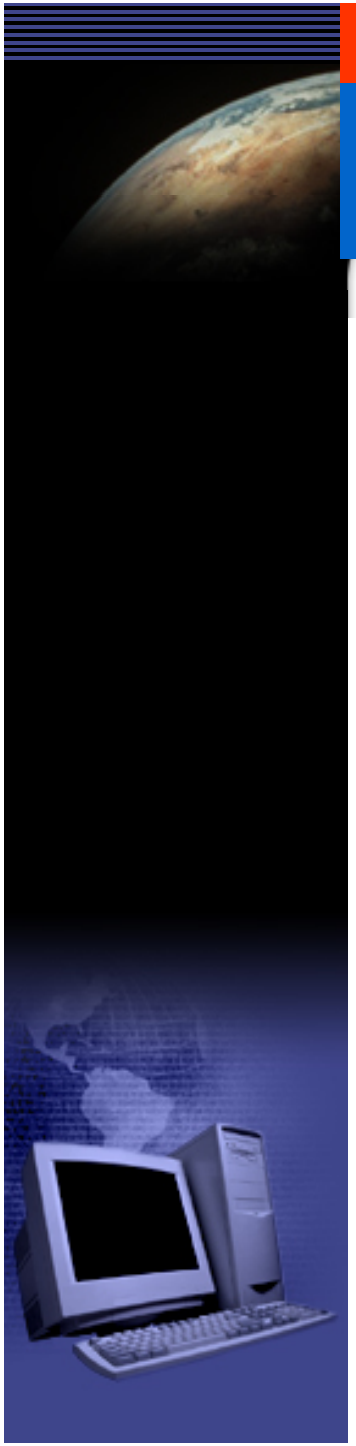
- Địa chỉ vật lý=địa chỉ bắt đầu của trang + d.

Cấp phát không liên tục

- Phân trang
  - Chuyển đổi địa chỉ



Cơ chế phần cứng hỗ trợ phân trang

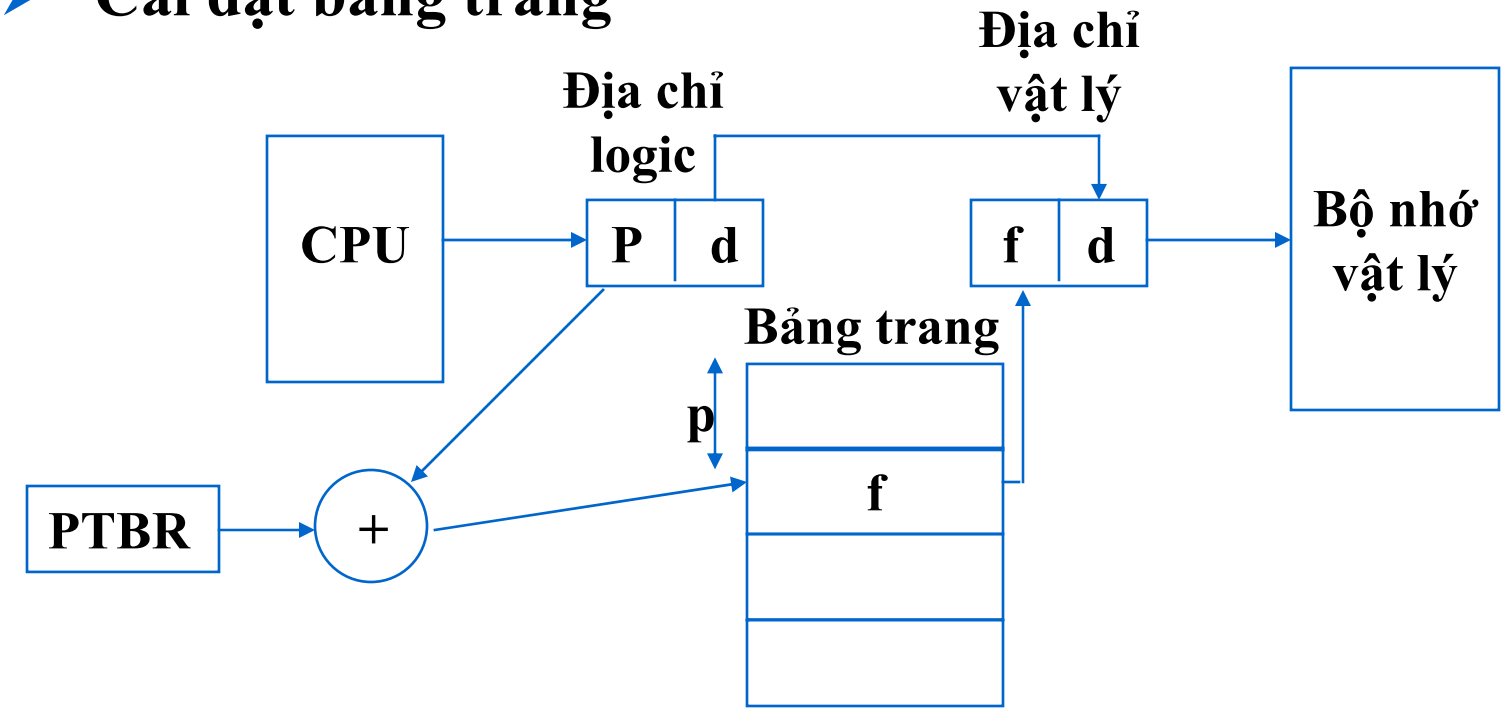


Cấp phát không liên tục

- **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

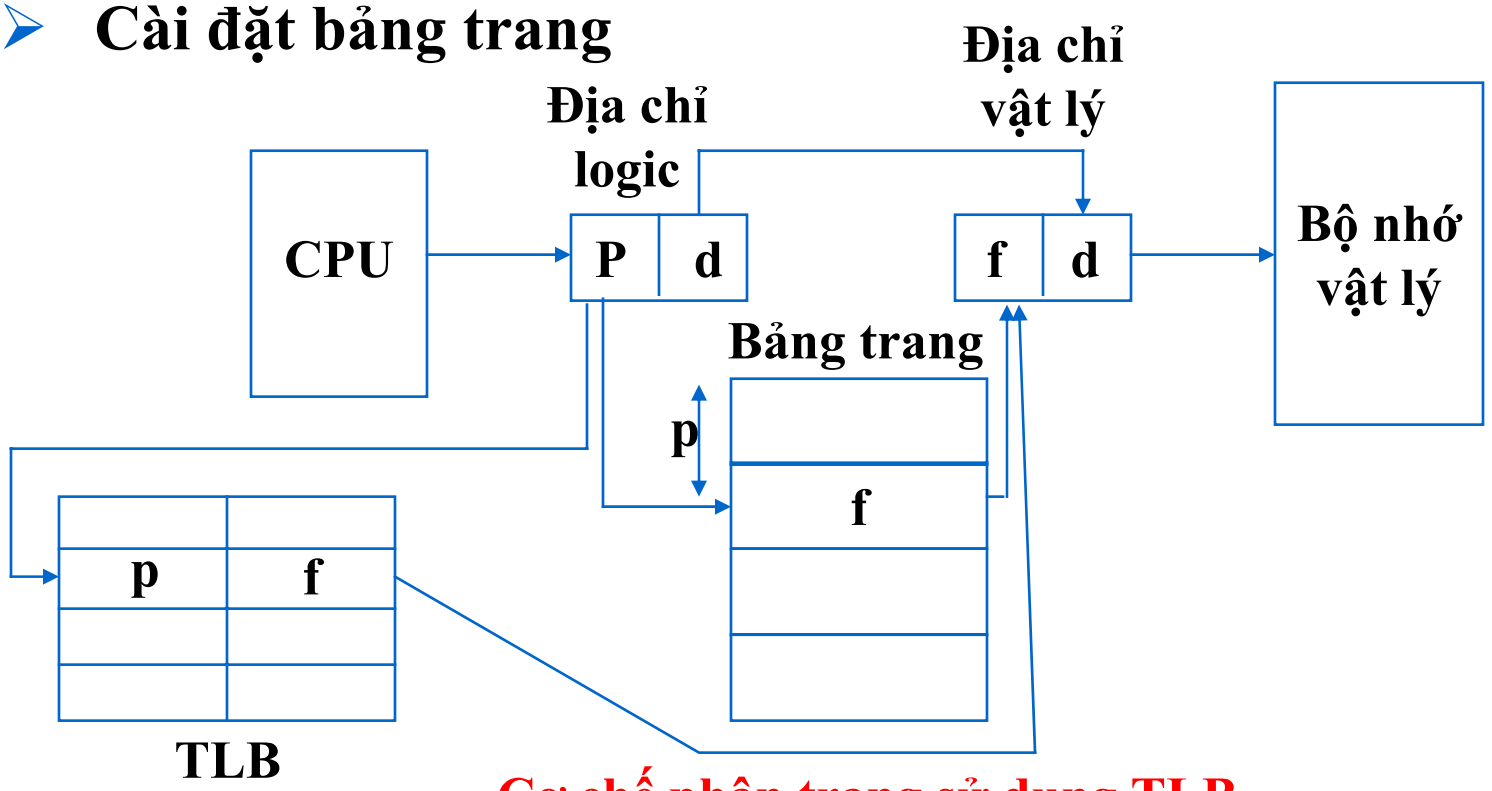
- Phân trang
  - Cài đặt bảng trang



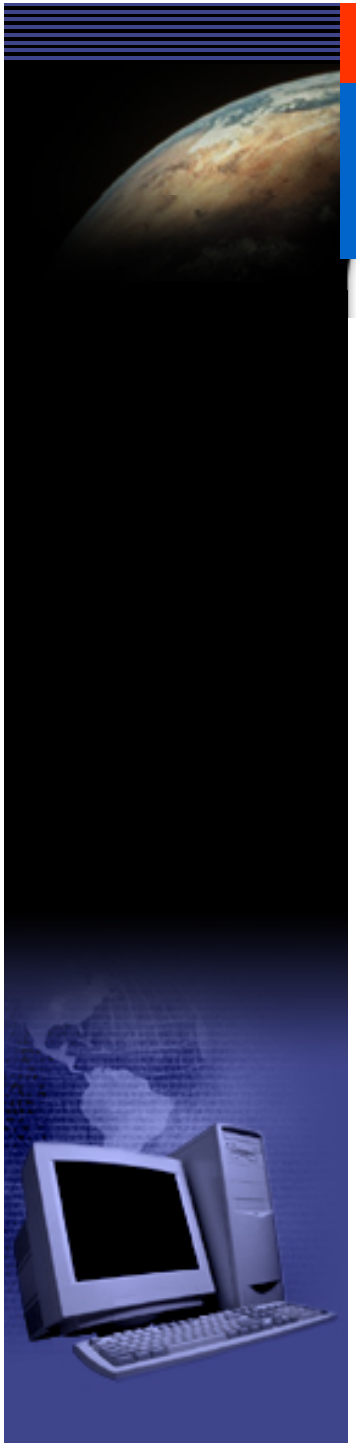
Cơ chế phân trang sử dụng PTBR

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.**

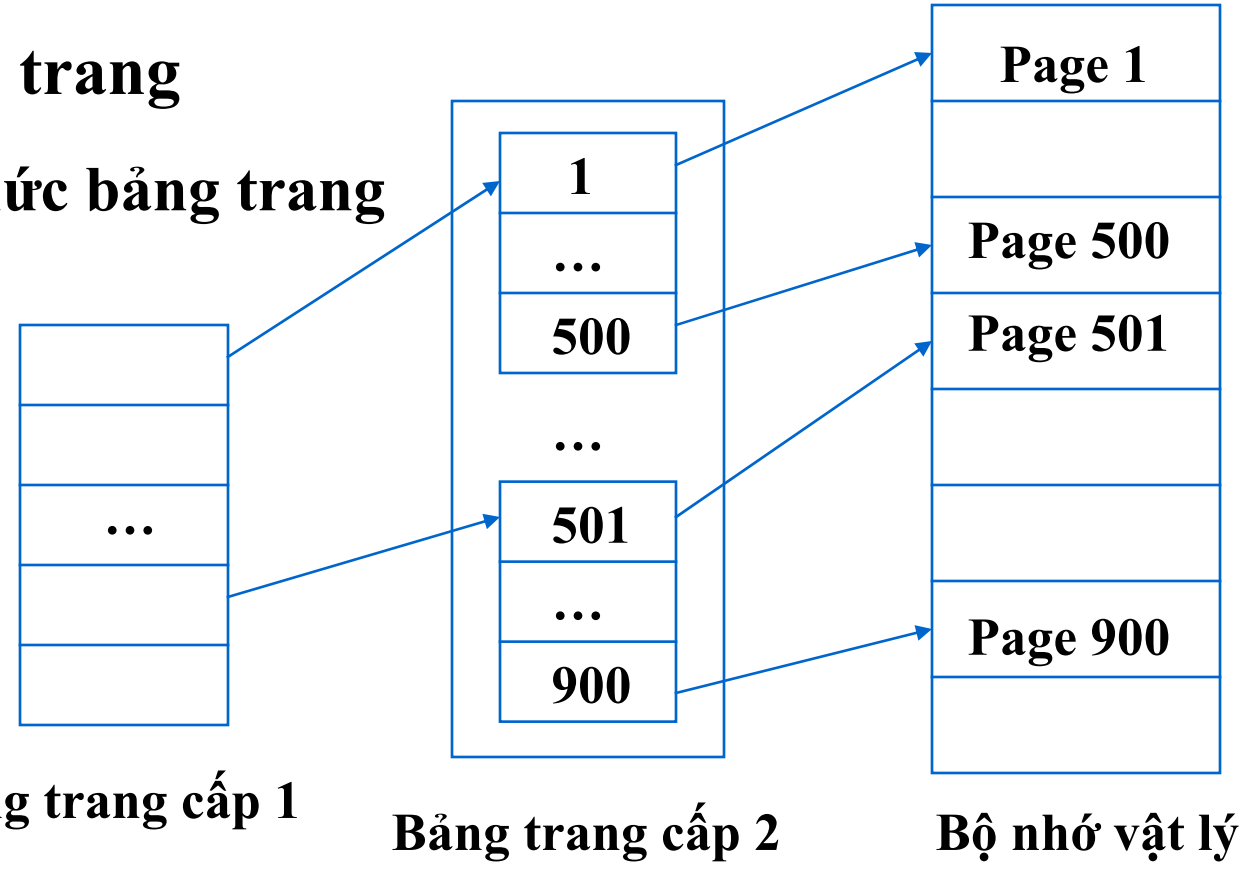
Cấp phát không liên tục

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

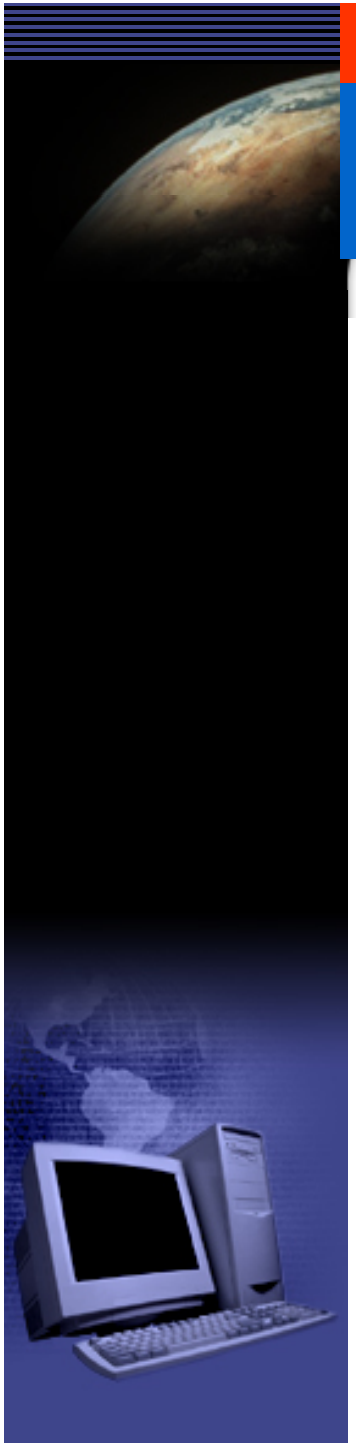
  Phân chia bảng trang thành các phần nhỏ, bản thân bảng trang cũng sẽ được phân trang

Cấp phát không liên tục

- Phân trang
  - Tổ chức bảng trang

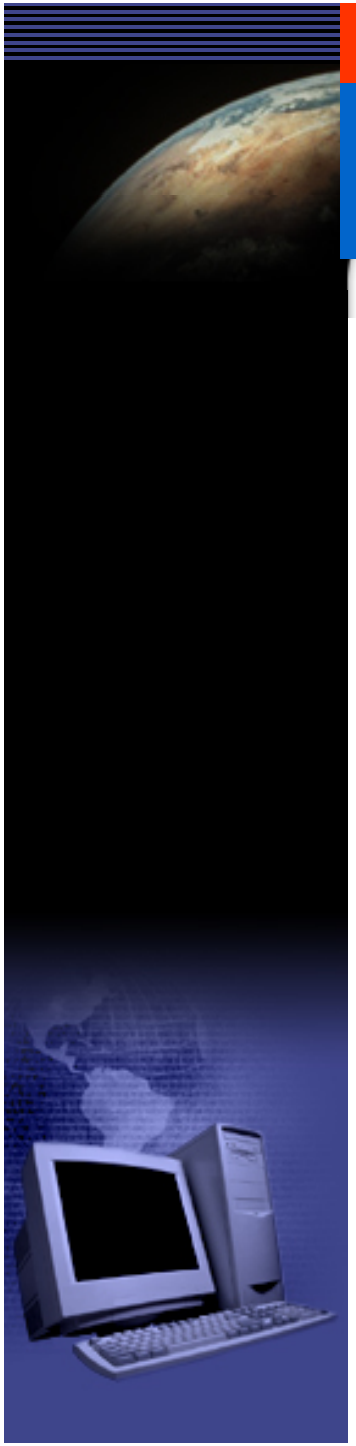


Bảng trang nhị cấp



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**



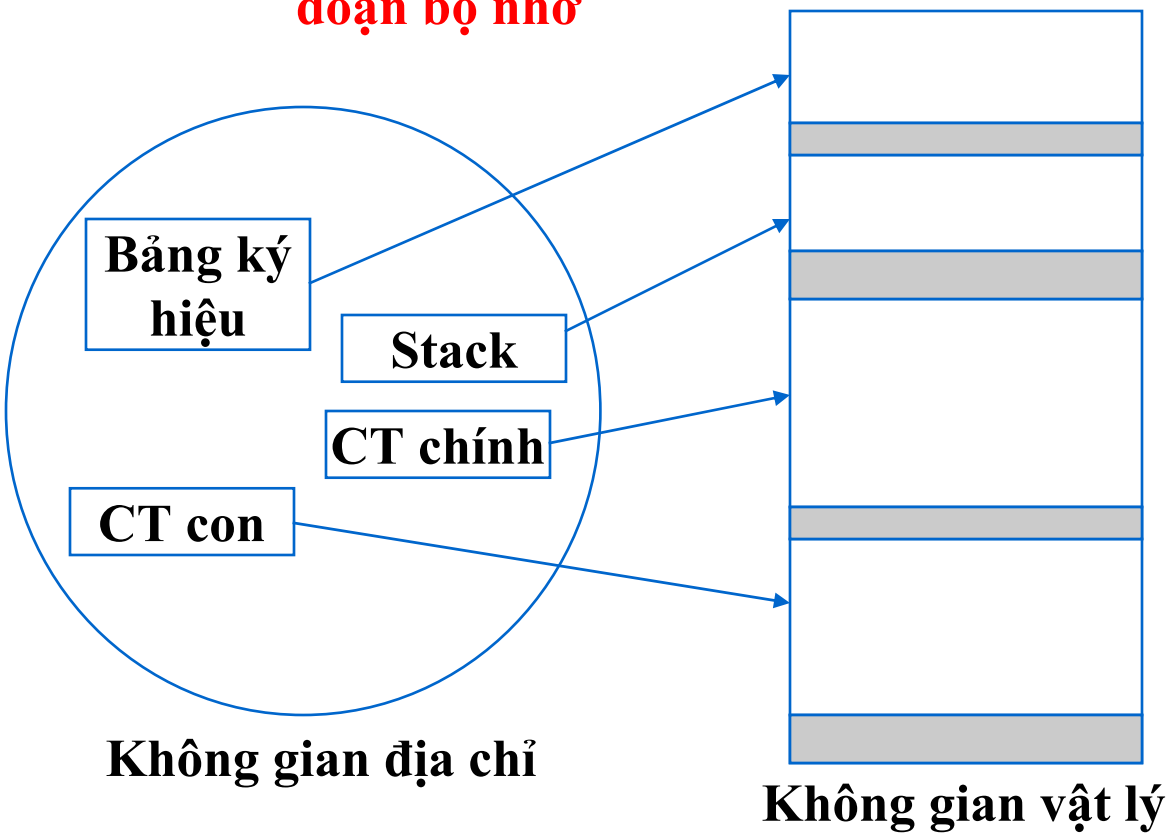
Cấp phát không liên tục

- **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

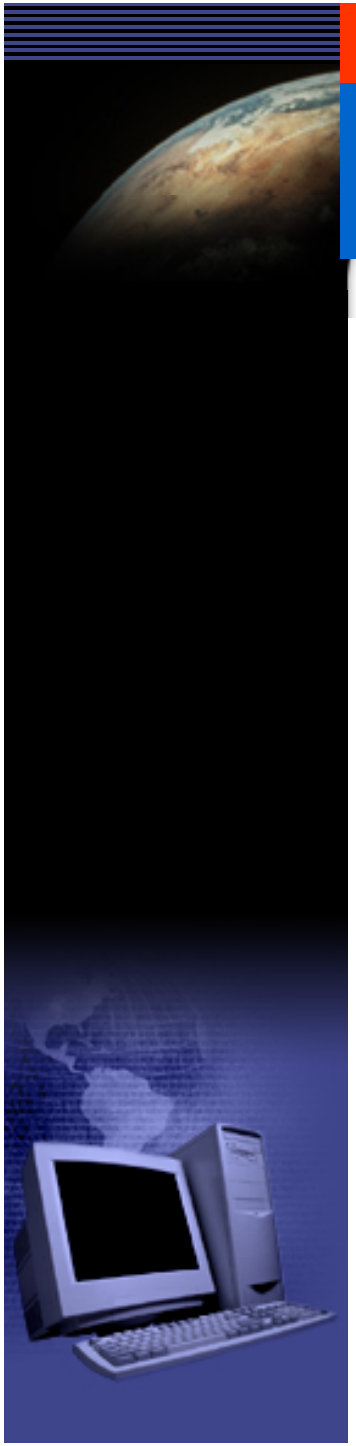
- Phân đoạn
  - Ý tưởng

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**

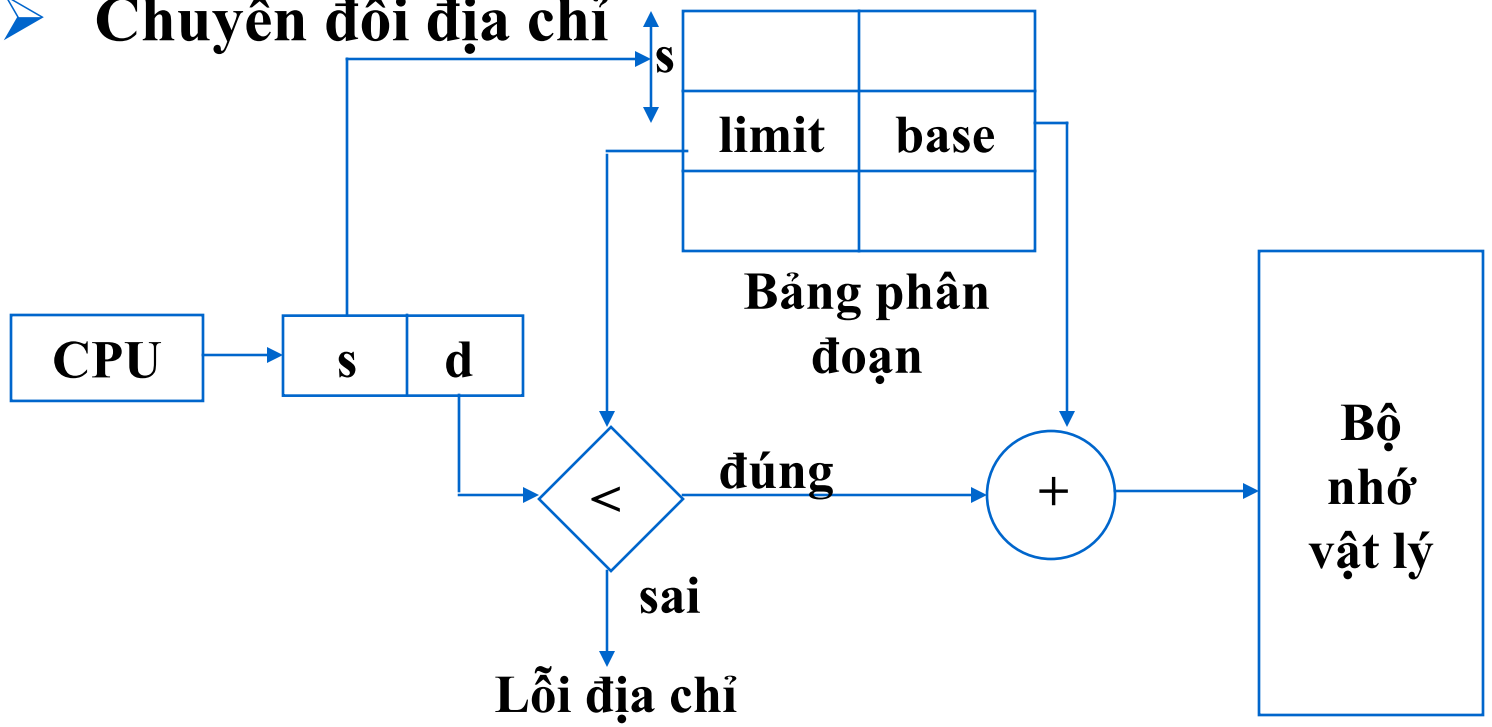


Cấp phát không liên tục

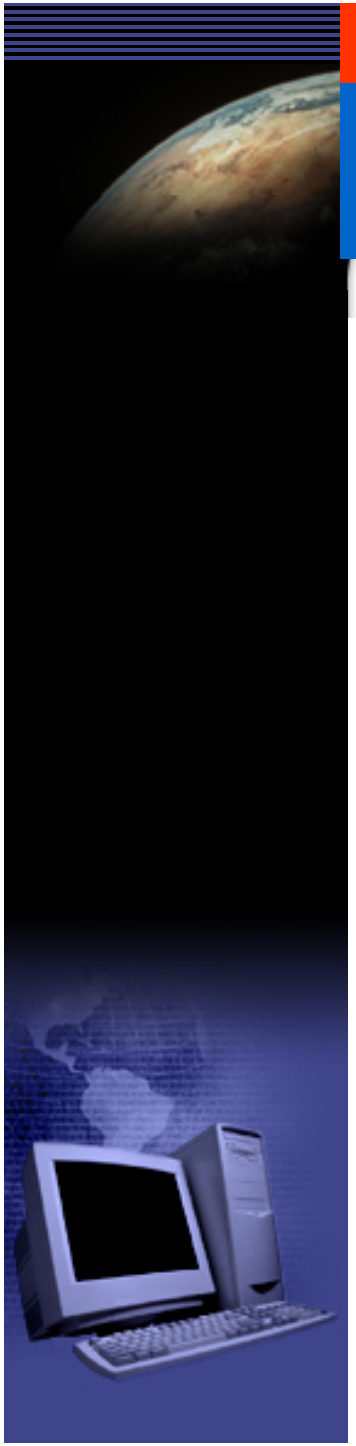
- **Phân đoạn**
  - **Chuyển đổi địa chỉ**
    - **Mỗi địa chỉ logic: <s,d>**
      - **s: số hiệu phân đoạn**
      - **d: địa chỉ tương đối offset, có giá trị từ 0 đến độ dài phân đoạn.**
    - **Địa chỉ vật lý=d+ giá trị chứa trong thanh ghi nền**

Cấp phát không liên tục

- Phân đoạn
  - Chuyển đổi địa chỉ



Cơ chế phần cứng hỗ trợ kỹ thuật phân đoạn



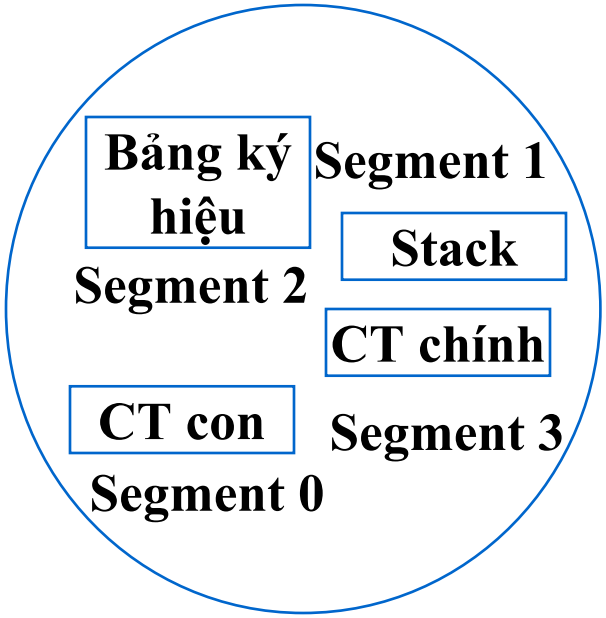
Cấp phát không liên tục

- **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

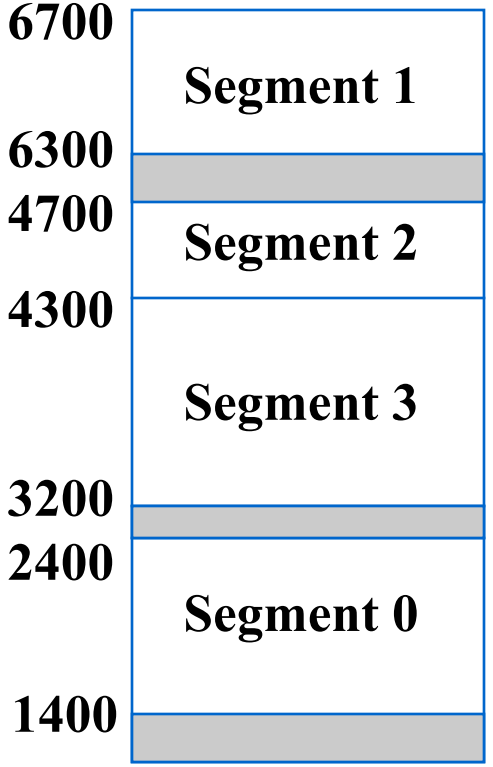
Phân đoạn

Cài đặt bảng phân đoạn



limit	base
1000	1400
400	6300
400	4300
1100	3200

Hệ thống phân đoạn



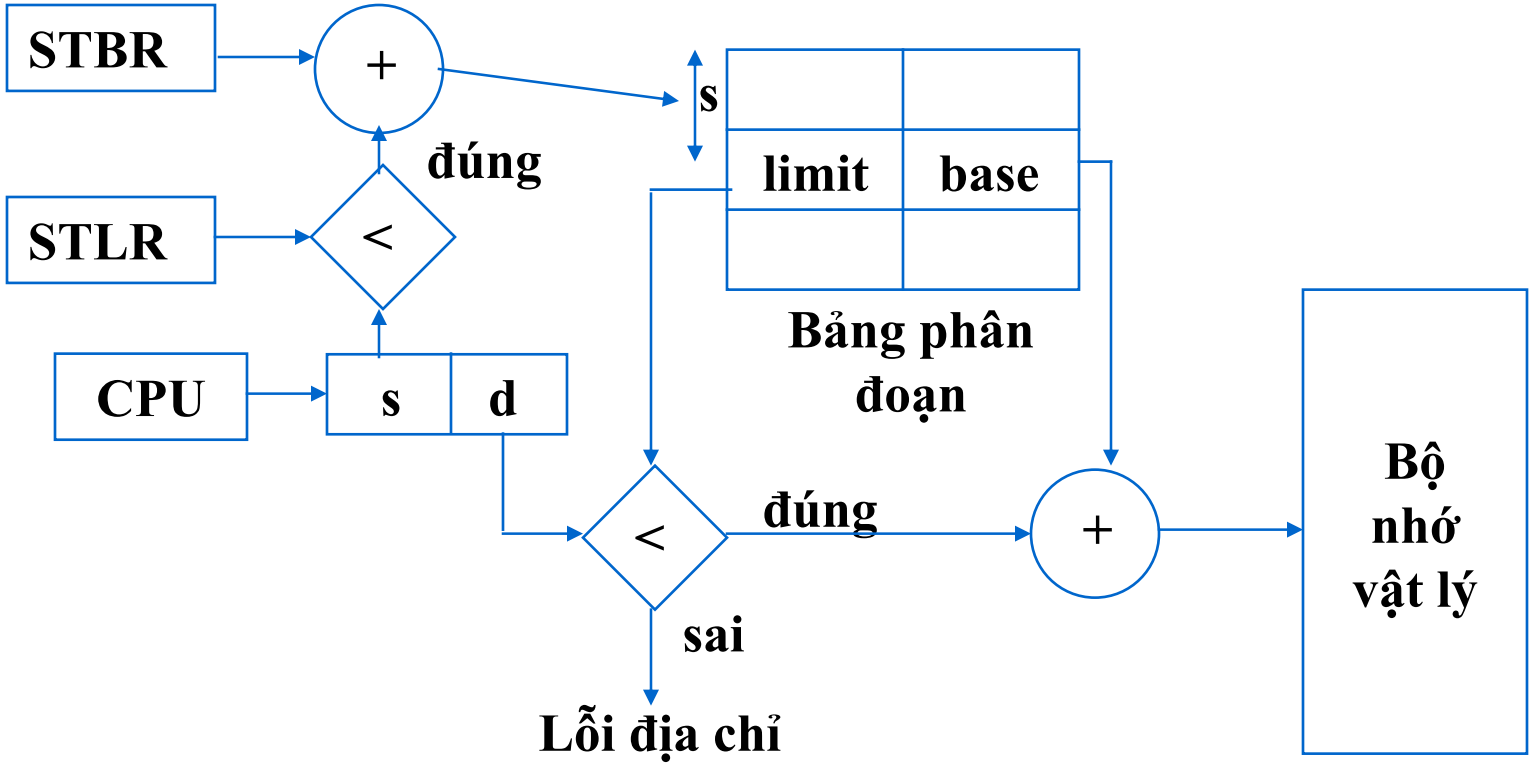
Không gian địa chỉ

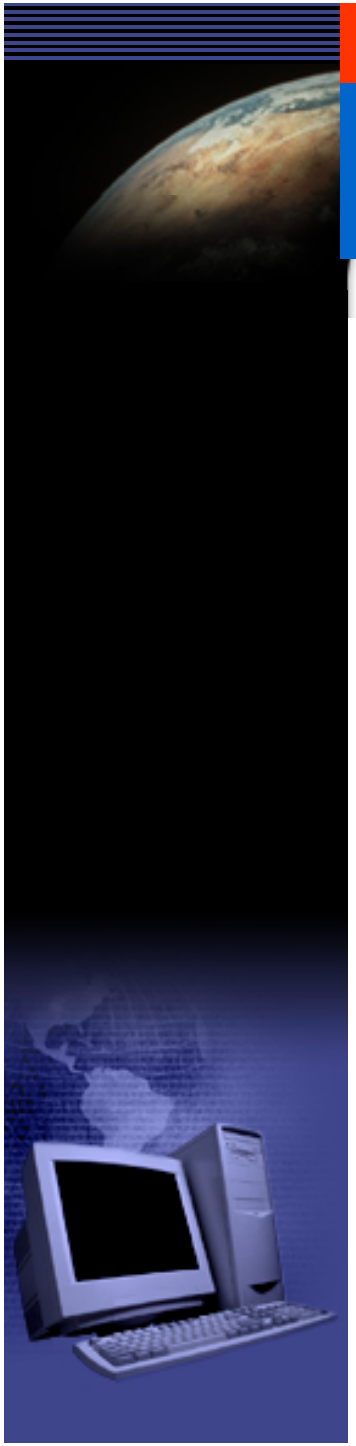
Không gian vật lý

Cấp phát không liên tục

Phân đoạn

Cài đặt bảng phân đoạn





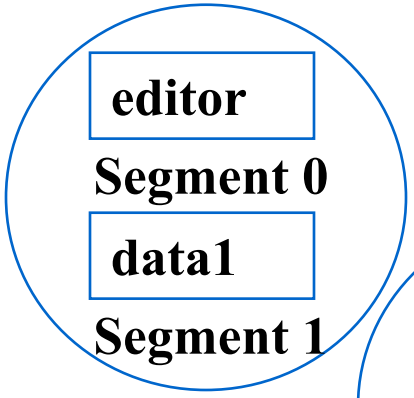
Cấp phát không liên tục

- **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

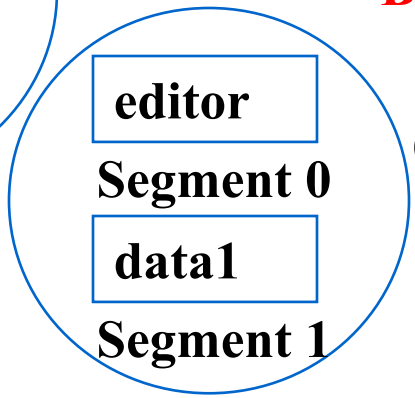
Không gian địa chỉ p1



Bảng phân đoạn p1

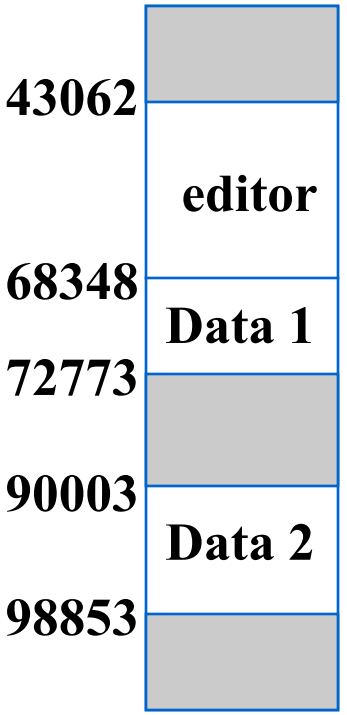
	limit	base
0	25286	43062
1	4425	68348

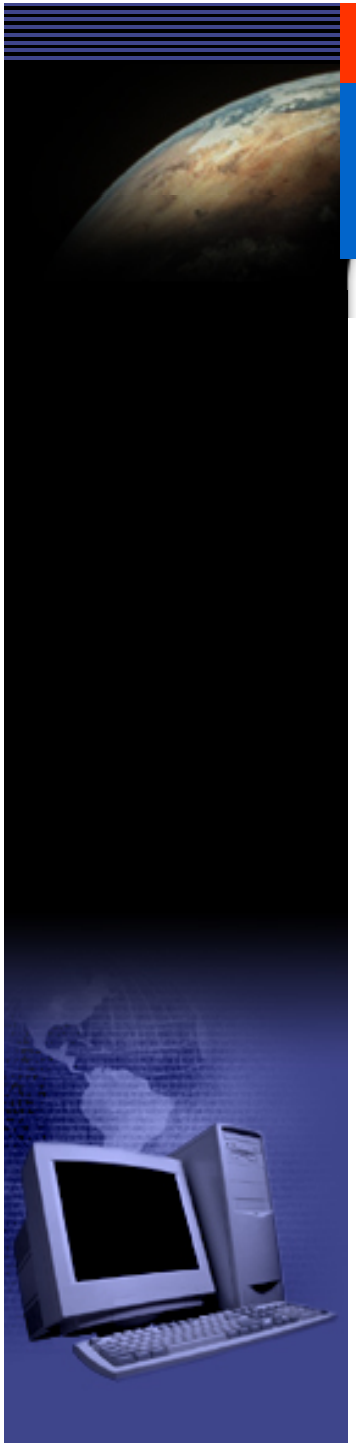
Không gian địa chỉ p2



Bảng phân đoạn p2

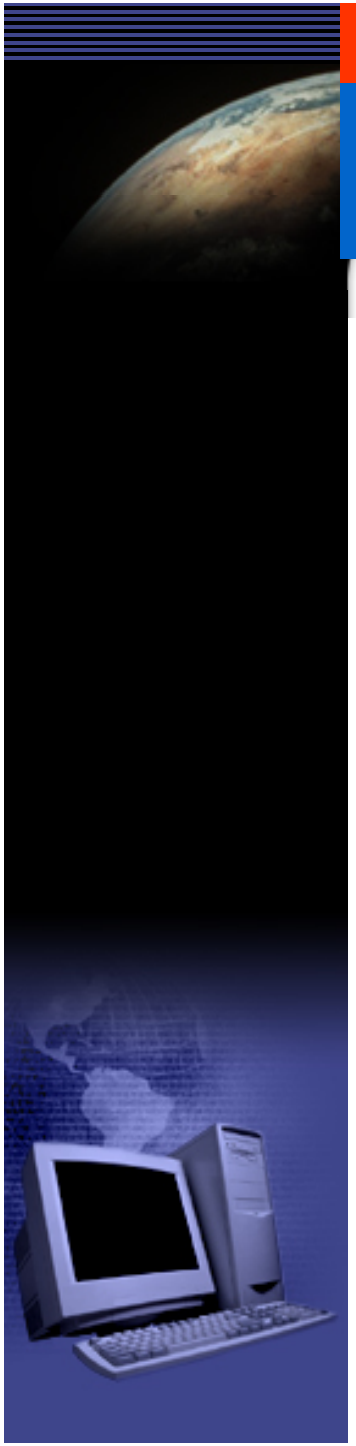
	limit	base
0	25286	43062
1	8850	90003





Cấp phát không liên tục

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

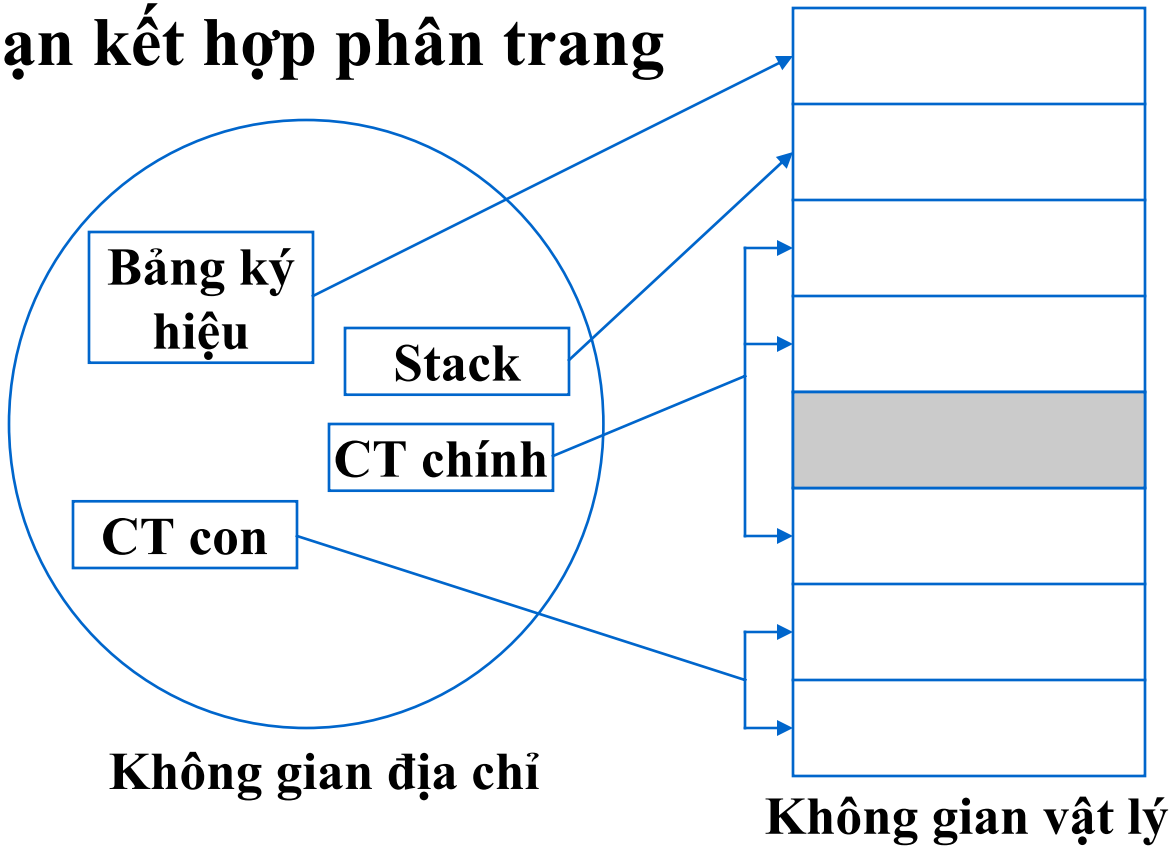


Cấp phát không liên tục

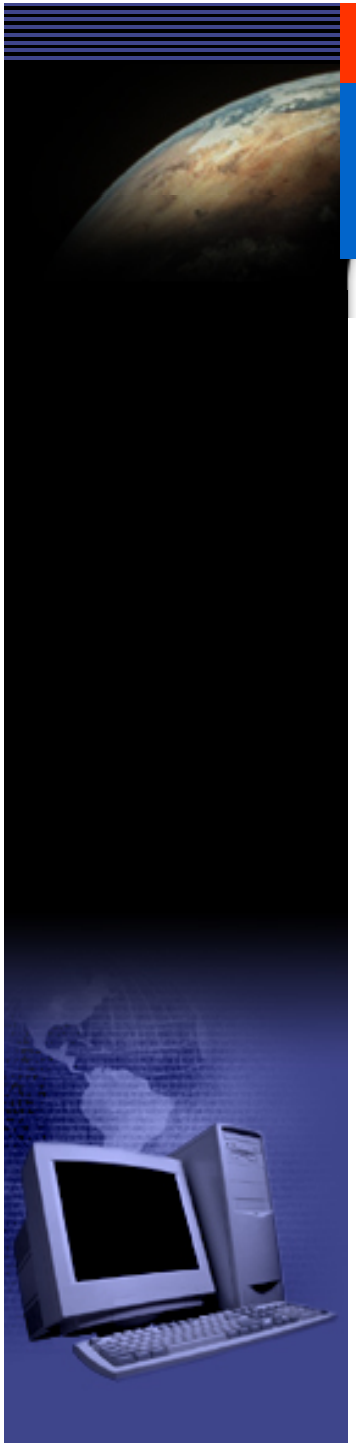
- **Phân đoạn kết hợp phân trang**
  - **Ý tưởng**
    - **Không gian địa chỉ: tập hợp các phân đoạn.**
    - **Mỗi phân đoạn: chia thành nhiều**
    - **Tiến trình được đưa vào hệ thống, HĐH sẽ cấp phát cho tiến trình các trang cần thiết để chứa đủ các phân đoạn của tiến trình**

Cấp phát không liên tục

- Phân đoạn kết hợp phân trang
  - Ý tưởng



Mô hình phân đoạn kết hợp phân trang

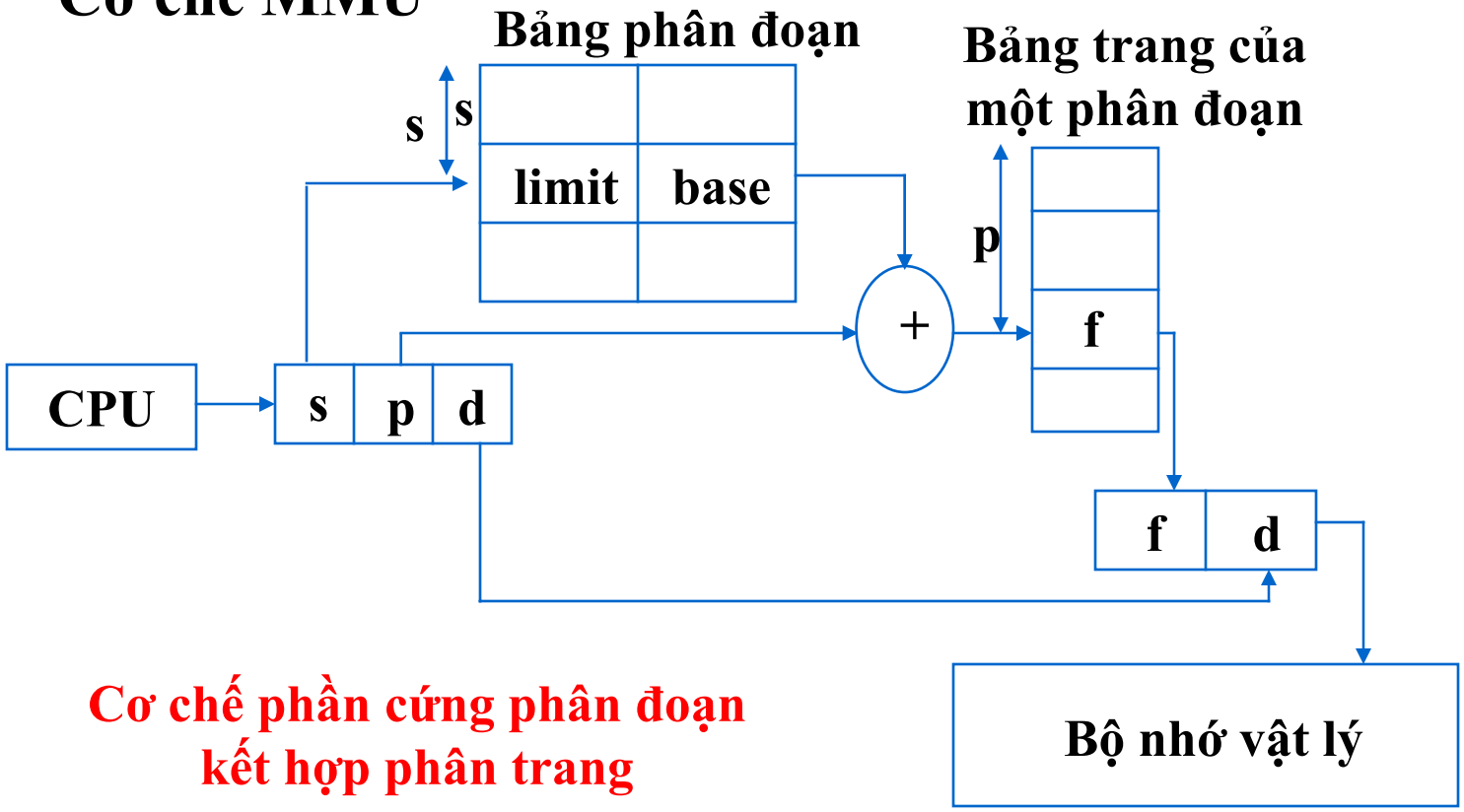


Cấp phát không liên tục

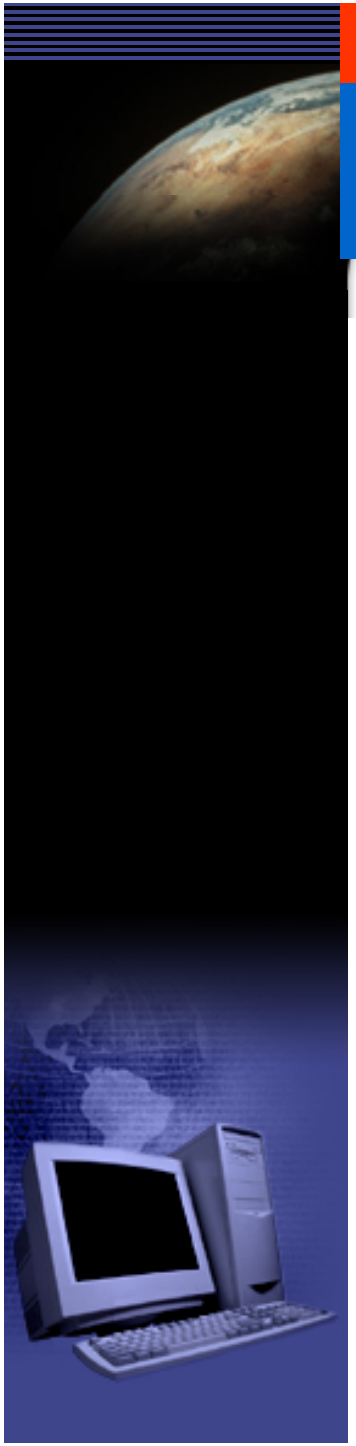
- **Phân đoạn kết hợp phân trang**
- **Chuyển đổi địa chỉ**
  - **Mỗi địa chỉ: <s, p, d>**
    - **S: số hiệu phân đoạn**
    - **P: số hiệu trang**
    - **D: địa chỉ tương đối**

Cấp phát không liên tục

- Phân đoạn kết hợp phân trang
  - Cơ chế MMU

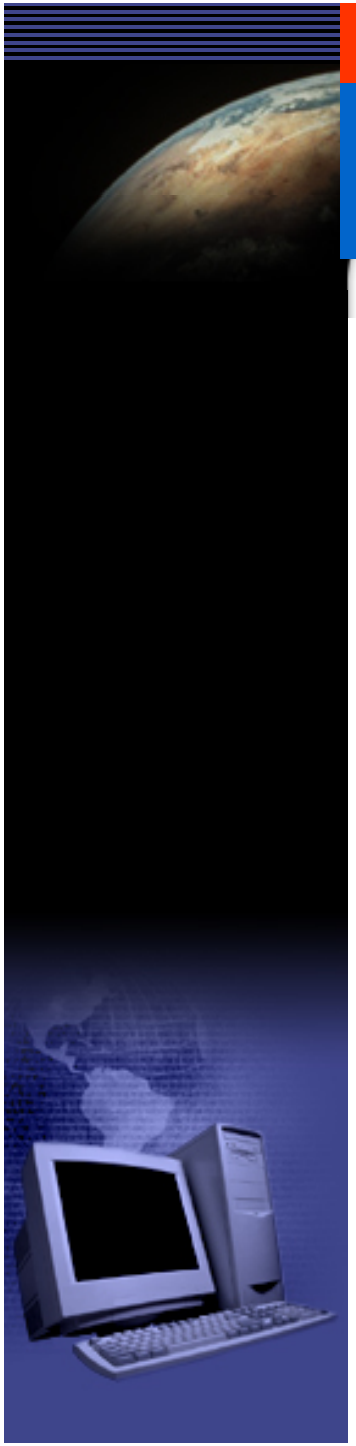


Cơ chế phần cứng phân đoạn kết hợp phân trang



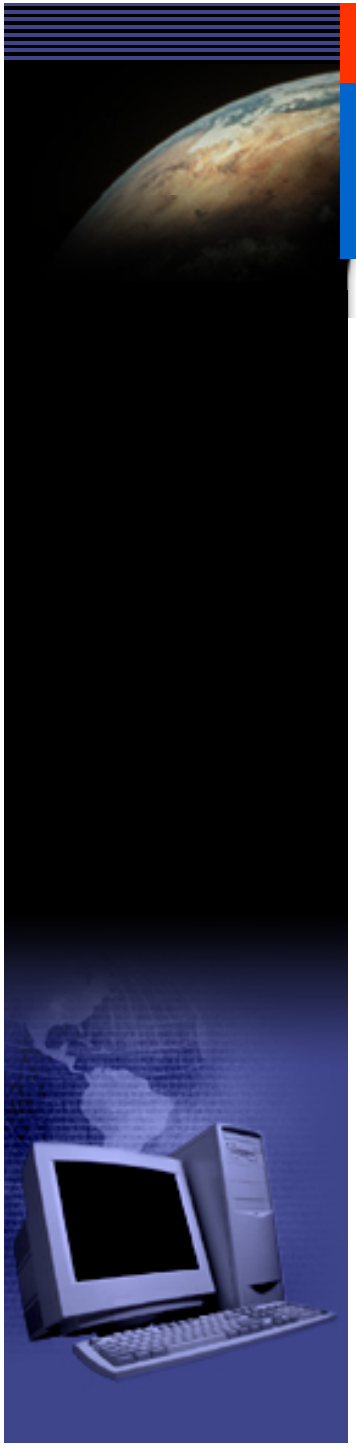
Bộ nhớ ảo

- **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.**



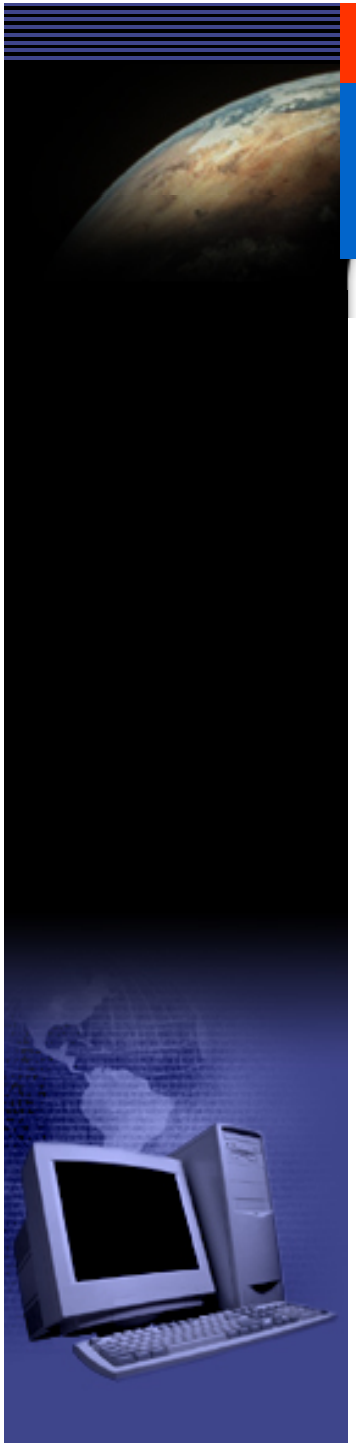
Bộ nhớ ảo

- **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)

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

BÀI GIẢNG

HỆ ĐIỀU HÀNH MÃ NGUỒN MỞ

TÊN HỌC PHẦN : **HỆ ĐIỀU HÀNH MÃ NGUỒN MỞ**
MÃ HỌC PHẦN : **17308**
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: **Hệ điều hành mã nguồn mở** *Loại học phần:* **2**
Bộ môn phụ trách giảng dạy: **Kỹ thuật máy tính** *Khoa phụ trách:* **CNTT**
Mã học phần: **17303** *Tổng số TC:* **3**

TS tiết	Lý thuyết	Thực hành/Xemina	Tự học	Bài tập lớn	Đồ án môn học
60	30	30	0	0	0

Điều kiện tiên quyết:

Sinh viên phải học xong các học phần sau mới được đăng ký học phần này:
Kiến trúc máy tính, Nguyên lý hệ điều hành

Mục tiêu của học phần:

- Nắm bắt được về hệ điều hành mã nguồn mở.

Nội dung chủ yếu

- Các kiến thức cơ bản về hệ điều hành Linux.
- Các dịch vụ trên Linux

Nội dung chi tiết của học phần:

TÊN CHƯƠNG MỤC	PHÂN PHỐI SỐ TIẾT				
	TS	**LT**	**BT**	**TH**	**KT**
Chương 1: Giới thiệu Unix – Linux	**2**	**2**			
1.1. Giới thiệu chung		0,5			
1.2. Các thành phần cơ bản của Linux		0,5			
1.3. Sử dụng lệnh trong Linux		1			
Chương 2. Thao tác với hệ thống	**10**	**3**		**6**	**1**
2.1. Tiến trình khởi động Linux		0.5			
2.2. Thủ tục đăng nhập và các lệnh thoát khỏi hệ thống		1			
2.3. Một số liên quan đến hệ thống		1			1
Chương 3. Quản trị hệ thống và người dùng	**8**	**4**		**4**	
3.1 Quản lý người dùng		1			
3.2 Quản lý nhóm các vấn đề liên quan		1			
3.3 Quản trị hệ thống		2			
Chương 4. Hệ thống file	**12**	**7**		**4**	**1**
4.1. Tổng quan về hệ thống file		0,5			
4.2. Quyền truy nhập thư mục và file		0,5			
4.3. Thao tác với thư mục		1			
4.4. Các lệnh làm việc với file		1			
4.5 Nén và sao lưu các file					1
Chương 5. Truyền thông và mạng	**8**	**4**		**4**	

5.1. Lệnh truyền thông		1			
5.2 Cấu hình Card giao tiếp mạng		1			
5.3. Các dịch vụ mạng		1			
5.4 Hệ thống file trên mạng		1			
Chương 6: Lập trình shell và lập trình C trên Linux	**20**	**7**		**12**	**1**
6.1. Cách thức pipes và các yếu tố cơ bản lập trình trên shell		2			
6.2. Một số lệnh lập trình trên shell		2			1
6.3. Lập trình C trên Linux		1			

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, hoàn thành bài tập lớn theo yêu cầu.

Tài liệu học tập :

- Richard Petersen - Linux: The Complete Reference, Sixth Edition – Nhà xuất bản McGraw-Hill Osborne Media ,2007.
- Michael Rash - Linux Firewalls: Attack Detection and Response with iptables, psad, and fwsnort – Nhà xuất bản No Starch Press ,2007
- Christopher Negus - Linux Bible – Nhà xuất bản Wiley, 2007
- Andrew Hudson và Paul Hudson – Fedora 7 UNLEASHED, 2007

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.

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

Điểm đánh giá học phần Z = 0.3X + 0.7Y.

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 1. GIỚI THIỆU CHUNG VỀ LINUX

1.1. Giới thiệu chung

1.1.1. Tổng quan về Linux

Linus Tovalds (một sinh viên Phần lan) đưa ra nhân (phiên bản đầu tiên) cho hệ điều hành Linux vào tháng 8 năm 1991 trên cơ sở cải tiến một phiên bản UNIX có tên Minix do Giáo sư Andrew S. Tanenbaum xây dựng và phổ biến. Nhân Linux tuy nhỏ song là tự đóng gói. Kết hợp với các thành phần trong hệ thống GNU, hệ điều hành Linux đã được hình thành. Và cũng từ thời điểm đó, theo tư tưởng GNU, hàng nghìn, hàng vạn chuyên gia trên toàn thế giới (những người này hình thành nên cộng đồng Linux) đã tham gia vào tiến trình phát triển Linux và vì vậy Linux ngày càng đáp ứng nhu cầu của người dùng.

Năm 1991, Linus Torvald viêt thêm phiên bản nhân v0.01 (kernel) đầu tiên của Linux đưa lên các BBS, nhóm người dùng để mọi người cùng sử dụng và phát triển.

Năm 1994, hệ điều hành Linux phiên bản 1.0 được chính thức phát hành và ngày càng nhận được sự quan tâm của người dùng.

Năm 1995, nhân 1.2 được phổ biến. Phiên bản này đã hỗ trợ một phạm vi rộng và phong phú phần cứng, bao gồm cả kiến trúc tuyến phần cứng PCI mới

Năm 1996, nhân Linux 2.0 được phổ biến. Phân bản này đã hỗ trợ kiến trúc phức hợp, bao gồm cả cổng Alpha 64-bit đầy đủ, và hỗ trợ kiến trúc đa bộ xử lý. Phân phối nhân Linux 2.0 cũng thi hành được trên bộ xử lý Motorola 68000 và kiến trúc SPARC của SUN. Các thi hành của Linux dựa trên vi nhân GNU Mach cũng chạy trên PC và PowerMac.

Năm 1999, phiên bản nhân v2.2 mang nhiều đặc tính ưu việt và giúp cho Linux bắt đầu trở thành đối thủ cạnh tranh đáng kể của MS Windows trên môi trường server.

Năm 2000 phiên bản nhân v2.4 hỗ trợ nhiều thiết bị mới (đa xử lý tới 32 chip, USB, RAM trên 2GB...) bắt đầu đặt chân vào thị trường máy chủ cao cấp.

Các phiên bản của Linux được xác định bởi hệ thống chỉ số theo một số mức (hai hoặc ba mức). Trong đó đã quy ước rằng với các chỉ số từ mức thứ hai trở đi, nếu là số chẵn thì dòng nhân đó đã khá ổn định và tương đối hoàn thiện, còn nếu là số lẻ thì dòng nhân đó vẫn đang được phát triển tiếp.

1.1.2. Vấn đề bản quyền

Về lý thuyết, mọi người có thể khởi tạo một hệ thống Linux bằng cách tiếp nhận bản mới nhất các thành phần cần thiết từ các site ftp và biên dịch chúng. Trong thời kỳ đầu tiên, người dùng Linux phải tiến hành toàn bộ các thao tác này và vì vậy công việc là khá vất vả. Tuy nhiên, do có sự tham gia đông đảo của các cá nhân và nhóm phát triển Linux, đã tiến hành thực hiện nhiều giải pháp nhằm làm cho công việc khởi tạo hệ thống đỡ vất vả. Một trong những giải pháp điển hình nhất là cung cấp tập các gói chương trình đã tiền dịch, chuẩn hóa.

Những tập hợp như vậy hay những bản phân phối là lớn hơn nhiều so với hệ thống Linux cơ sở. Chúng thường bao gồm các tiện ích bổ sung cho khởi tạo hệ thống, các thư viện quản lý, cũng như nhiều gói đã được tiền dịch, sẵn sàng khởi tạo của nhiều bộ công cụ UNIX dùng chung, chẳng hạn như phục vụ tin, trình duyệt web, công cụ xử lý, soạn thảo văn bản và thậm chí các trò chơi.

Cách thức phân phối ban đầu rất đơn giản song ngày càng được nâng cấp và hoàn thiện bằng phương tiện quản lý gói tiên tiến. Các bản phân phối ngày nay bao gồm các cơ sở dữ liệu tiến hóa gói, cho phép các gói dễ dàng được khởi tạo, nâng cấp và loại bỏ.

Nhà phân phối đầu tiên thực hiện theo phương châm này là Slakware, và chính họ là những chuyển biến mạnh mẽ trong cộng đồng Linux đối với công việc quản lý gói khởi tạo

Linux. Tiện ích quản lý gói RPM (RedHat Package Manager) của công ty RedHat là một trong những phương tiện điển hình.

Nhân Linux là phần mềm tự do được phân phối theo Giấy phép sở hữu công cộng phần mềm GNU GPL.

1.1.3. Các thành phần tích hợp Hệ điều hành Linux

Linux sử dụng rất nhiều thành phần từ Dự án phần mềm tự do GNU, từ hệ điều hành BSDcủa Đại học Berkeley và từ hệ thống X-Window của MIT.

Thư viện hệ thống chính của Linux được bắt nguồn từ Dự án GNU, sau đó được rất nhiều người trong cộng đồng Linux phát triển tiếp, những phát triển tiếp theo như vậy chủ yếu liên quan tới việc giải quyết các vấn đề như thiếu vắng địa chỉ (lỗi trang), thiếu hiệu quả và gỡ rối. Một số thành phần khác của Dự án GNU, chẳng hạn như trình biên dịch GNU C (gcc), vốn là chất lượng cao nên được sử dụng nguyên xy trong Linux.

Các tool quản lý mạng được bắt nguồn từ mã 4.3BSD song sau đó đã được cộng đồng Linux phát triển, chẳng hạn như thư viện toán học đồng xử lý dấu chấm động Intel và các trình điều khiển thiết bị phần cứng âm thanh PC. Các tool quản lý mạng này sau đó lại được bổ sung vào hệ thống BSD.

Hệ thống Linux được duy trì gần như bởi một mạng lưới không chặt chẽ các nhà phát triển phần mềm cộng tác với nhau qua Internet, mạng lưới này gồm các nhóm nhỏ và cá nhân chịu trách nhiệm duy trì tính toàn vẹn của từng thành phần. Một lượng nhỏ các site phân cấp ftp Internat công cộng đã đóng vai trò nhà kho theo chuẩn de facto để chứa các thành phần này. Tài liệu Chuẩn phân cấp hệ thống file (File System Hierarchy Standard) được cộng đồng Linux duy trì nhằm giữ tính tương thích khắc phục được sự khác biệt rất lớn giữa các thành phần hệ thống.

1.1.4. Một số đặc điểm chính của Linux

Dưới đây trình bày một số đặc điểm chính của của hệ điều hành Linux hiện tại:

- Linux tương thích với nhiều hệ điều hành như DOS, MicroSoft Windows...:
- Cho phép cài đặt Linux cùng với các hệ điều hành khác trên cùng một ổ cứng. Linux có thể truy nhập đến các file của các hệ điều hành cùng một ổ đĩa. Linux cho phép chạy mô phỏng các chương trình thuộc các hệ điều hành khác.
- Do giữ được chuẩn của UNIX nên sự chuyển đổi giữa Linux và các hệ UNIX khác là dễ dàng.
- Linux là một hệ điều hành UNIX tiêu biểu với các đặc trưng là đa người dùng, đa chương trình và đa xử lý.
- Linux có giao diện đồ hoạ (GUI) thừa hưởng từ hệ thống X-Window. Linux hỗ trợ nhiều giao thức mạng, bắt nguồn và phát triển từ dòng BSD. Thêm vào đó, Linux còn hỗ trợ tính toán thời gian thực.
- Linux khá mạnh và chạy rất nhanh ngay cả khi nhiều tiến trình hoặc nhiều cửa sổ.
- Linux được cài đặt trên nhiều chủng loại máy tính khác nhau như PC, Mini và việc cài đặt khá thuận lợi. Tuy nhiên, hiện nay chưa xuất hiện Linux trên máy tính lớn (mainframe).
- Linux ngày càng được hỗ trợ bởi các phần mềm ứng dụng bổ sung như soạn thảo, quản lý mạng, quản trị cơ sở dữ liệu, bảng tính...
- Linux hỗ trợ tốt cho tính toán song song và máy tính cụm (PC-cluster) là một hướng nghiên cứu triển khai ứng dụng nhiều triển vọng hiện nay.
- Là một hệ điều hành với mã nguồn mở, được phát triển qua cộng đồng nguồn mở (bao gồm cả Free Software Foundation) nên Linux phát triển nhanh. Linux là một trong một số ít các hệ điều hành được quan tâm nhiều nhất trên thế giới hiện nay.

- Linux là một hệ điều hành hỗ trợ đa ngôn ngữ một cách toàn diện nhất. Do Linux cho phép hỗ trợ các bộ mã chuẩn từ 16 bit trở lên (trong đó có các bộ mã ISO10646, Unicode) cho nên việc bản địa hóa trên Linux là triệt để nhất trong các hệ điều hành.

Tuy nhiên cũng tồn tại một số khó khăn làm cho Linux chưa thực sự trở thành một hệ điều hành phổ dụng, dưới đây là một số khó khăn điển hình:

- Tuy đã có công cụ hỗ trợ cài đặt, tuy nhiên, việc cài đặt Linux còn tương đối phức tạp và khó khăn. Khả năng tương thích của Linux với một số loại thiết bị phần cứng còn thấp do chưa có các trình điều khiển cho nhiều thiết bị,
- Phần mềm ứng dụng chạy trên nền Linux tuy đã phong phú song so với một số hệ điều hành khác, đặc biệt là khi so sánh với MS Windows, thì vẫn còn có khoảng cách.

Với sự hỗ trợ của nhiều công ty tin học hàng đầu thế giới (IBM, SUN, HP...) và sự tham gia phát triển của hàng vạn chuyên gia trên toàn thế giới thuộc cộng đồng Linux, các khó khăn của Linux chắc chắn sẽ nhanh chóng được khắc phục.

1.2. Các thành phần cơ bản của Linux

Hệ thống Linux, được thi hành như một hệ điều hành UNIX truyền thống, gồm shell và ba thành phần (đã dạng mã chương trình) sau đây:

- Nhân hệ điều hành chịu trách nhiệm duy trì các đối tượng trừu tượng quan trọng của hệ điều hành, bao gồm bộ nhớ ảo và tiến trình. Các mô đun chương trình trong nhân được đặc quyền trong hệ thống, bao gồm đặc quyền thường trực ở bộ nhớ trong.
- Thư viện hệ thống xác định một tập chuẩn các hàm để các ứng dụng tương tác với nhân, và thi hành nhiều chức năng của hệ thống nhưng không cần có các đặc quyền của mô đun thuộc nhân. Một hệ thống con điển hình được thi hành dựa trên thư viên hệ thống là hệ thống file Linux.
- Tiện ích hệ thống là các chương trình thi hành các nhiệm vụ quản lý riêng rẽ, chuyên biệt. Một số tiện ích hệ thống được gọi ra chỉ một lần để khởi động và cấu hình phương tiện hệ thống, một số tiện ích khác, theo thuật ngữ UNIX được gọi là trình chạy ngầm (daemon), có thể chạy một cách thường xuyên (thường theo chu kỳ), điều khiển các bài toán như hưởng ứng các kết nối mạng mới đến, tiếp nhận yêu cầu logon, hoặc cập nhật các file log.

Tiện ích (hay lệnh) có sẵn trong hệ điều hành (dưới đây tiện ích được coi là lệnh thường trực). Nội dung chính yếu của tài liệu này giới thiệu chi tiết về một số lệnh thông dụng nhất của Linux. Hệ thống file sẽ được giới thiệu trong chương 3. Trong các chương sau có đề cập tới nhiều nội dung liên quan đến nhân và shell, song dưới đây là một số nét sơ bộ về chúng.

1.2.1. Nhân hệ thống (kernel)

Nhân (còn được gọi là hệ lõi) của Linux, là một bộ các môdun chương trình có vai trò điều khiển các thành phần của máy tính, phân phối các tài nguyên cho người dùng (các tiến trình người dùng). Nhân chính là cầu nối giữa chương trình ứng dụng với phần cứng. Người dùng sử dụng bàn phím gõ nội dung yêu cầu của mình và yêu cầu đó được nhân gửi tới shell: Shell phân tích lệnh và gọi các chương trình tương ứng với lệnh để thực hiện.

Một trong những chức năng quan trọng nhất của nhân là giải quyết bài toán lập lịch, tức là hệ thống cần phân chia CPU cho nhiều tiến trình hiện thời cùng tồn tại. Đối với Linux, số lượng tiến trình có thể lên tới con số hàng nghìn. Với số lượng tiến trình đồng thời nhiều như vậy, các thuật toán lập lịch cần phải đủ hiệu quả: Linux thường lập lịch theo chế độ Round Robin (RR) thực hiện việc luân chuyển CPU theo lượng tử thời gian.

Thành phần quan trọng thứ hai trong nhân là hệ thống các môđun chương trình (được gọi là lời gọi hệ thống) làm việc với hệ thống file. Linux có hai cách thức làm việc với các file: làm việc theo byte (ký tự) và làm việc theo khối. Một đặc điểm đáng chú ý là file trong Linux có thể được nhiều người cùng truy nhập tới nên các lời gọi hệ thống làm việc với file cần đảm bảo việc file được truy nhập theo quyền và được chia xẻ cho người dùng.

1.2.2. Hệ vỏ (shell)

Người dùng mong muốn máy tính thực hiện một công việc nào đó thì cần gõ lệnh thể hiện yêu cầu của mình để hệ thống đáp ứng yêu cầu đó. Shell là bộ dịch lệnh và hoạt động như một kết nối trung gian giữa nhân với người dùng: Shell nhận dòng lệnh do người dùng đưa vào; và từ dòng lệnh nói trên, nhân tách ra các bộ phận để nhận được một hay một số lệnh tương ứng với các đoạn văn bản có trong dòng lệnh. Một lệnh bao gồm tên lệnh và tham số: từ đầu tiên là tên lệnh, các từ tiếp theo (nếu có) là các tham số. Tiếp theo, shell sử dụng nhân để khởi sinh một tiến trình mới (khởi tạo tiến trình) và sau đó, shell chờ đợi tiến trình con này tiến hành, hoàn thiện và kết thúc. Khi shell sẵn sàng tiếp nhận dòng lệnh của người dùng, một dấu nhắc shell (còn gọi là dấu nhắc nhập lệnh) xuất hiện trên màn hình.

Linux có hai loại shell phổ biến là: C-shell (dấu nhắc %), Bourne-shell (dấu nhắc $) và một số shell phát triển từ các shell nói trên (chẳng hạn, TCshell - tcsh với dấu nhắc ngầm định > phát triển từ C-shell và GNU Bourne - bash với dấu nhắc bash # phát triển từ Bourne-shell). Dấu mời phân biệt shell nói trên không phải hoàn toàn rõ ràng do Linux cho phép người dùng thay đổi lại dấu nhắc shell nhờ việc thay giá trị các biến môi trường **PS1** và **PS2.** Trong tài liệu này, chúng ta sử dụng ký hiệu "hàng rào #" để biểu thị dấu nhắc shell.

C-shell có tên gọi như vậy là do cách viết lệnh và chương trình lệnh Linux tựa như ngôn ngữ C. Bourne-shell mang tên tác giả của nó là Steven Bourne. Một số lệnh trong C- shell (chẳng hạn lệnh **alias**) không còn có trong Bourne-shell và vì vậy để nhận biết hệ thống đang làm việc với shell nào, chúng ta gõ lệnh:

 # alias

Nếu một danh sách xuất hiện thì shell đang sử dụng là C-shell; ngược lại, nếu xuất hiện thông báo "Command not found" thì shell đó là Bourne-shell.

Lệnh được chia thành 3 loại lệnh:

- Lệnh thường trực (có sẵn của Linux). Tuyệt đại đa số lệnh được giới thiệu trong tài liệu này là lệnh thường trực. Chúng bao gồm các lệnh được chứa sẵn trong shell và các lệnh thường trực khác.
- File chương trình ngôn ngữ máy: chẳng hạn, người dùng viết trình trên ngôn ngữ C qua bộ dịch gcc (bao gồm cả trình kết nối link) để tạo ra một chương trình trên ngôn ngữ máy.
- File chương trình shell (Shell Scrip).

Khi kết thúc một dòng lệnh cần gõ phím ENTER để shell phân tích và thực hiện lệnh.

1.3. Sử dụng lệnh trong Linux

Như đã giới thiệu ở phần trên, Linux là một hệ điều hành đa người dùng, đa nhiệm, được phát triển bởi hàng nghìn chuyên gia tin học trên toàn thế giới nên hệ thống lệnh cũng ngày càng phong phú; đến thời điểm hiện nay Linux có khoảng hơn một nghìn lệnh. Tuy nhiên chỉ có khoảng vài chục lệnh là thông dụng nhất đối với người dùng.

Cũng như đã nói ở trên, người dùng làm việc với máy tính thông qua việc sử dụng trạm cuối: người dùng đưa yêu cầu của mình bằng cách gõ "lệnh" từ bàn phím và giao cho hệ điều hành xử lý.

Khi cài đặt Linux lên máy tính cá nhân thì máy tính cá nhân vừa đóng vai trò trạm cuối, vừa đóng vai trò máy tính xử lý.

1.3.1. Dạng tổng quát của lệnh Linux

Cú pháp lệnh: ***# <Tên lệnh> [<các tham số>]***

Trong đó:

- Tên lệnh là một dãy ký tự, không có dấu cách, biểu thị cho một lệnh của Linux hay một chương trình. Người dùng cần hệ điều hành đáp ứng yêu cầu gì của mình thì phải chọn đúng tên lệnh. Tên lệnh là bắt buộc phải có khi gõ lệnh.
- Các tham số có thể có hoặc không có, được viết theo quy định của lệnh mà chúng ta sử dụng, nhằm cung cấp thông tin về các đối tượng mà lệnh tác động tới. Ý nghĩa của các dấu [, <, >,] được giải thích ở phần quy tắc viết lệnh.

Các tham số được phân ra thành hai loại: tham số khóa (sau đây gọi là "***tùy chọn***") và tham số vị trí.

- Tham số vị trí thường là tên file, thư mục và thường là các đối tượng chịu sự tác động của lệnh. Khi gõ lệnh, tham số vị trí được thay bằng những đối tượng mà người dùng cần hướng tác động tới.
- Tham số khóa chính là những tham số điều khiển hoạt động của lệnh theo các trường hợp riêng. Trong Linux, tham số khóa thường bắt đầu bởi dấu trừ "-" hoặc hai dấu trừ liên tiếp "--". Một lệnh có thể có một số hoặc rất nhiều tham số khóa.

Ví dụ, khi người dùng gõ lệnh xem thông tin về các file:

 # ls -l

Trong lệnh này:

- **ls :** là tên lệnh thực hiện việc đưa danh sách các tên file/ thư mục con trong một thư mục,
- **-l** : là tham số khóa, cho biết yêu cầu xem đầy đủ thông tin về các đối tượng hiện ra. Chú ý, trong tham số khóa chữ cái (chữ "l") phải đi ngay sau dấu trừ "-".

Chú ý:

- Linux (và UNIX nói chung) được xây dựng trên ngôn ngữ lập trình C, vì vậy khi gõ lệnh phải phân biệt chữ thường với chữ hoa. Ngoại trừ một số ngoại lệ, trong Linux chúng ta thấy phổ biến là:
  - Các tên lệnh là chữ thường,
  - Một số tham số khi biểu diễn bởi chữ thường hoặc chữ hoa sẽ có ý nghĩa hoàn toàn khác nhau).
  - Tên các biến môi trường cũng thường dùng chữ hoa.
- Linux phân biệt siêu người dùng (superuser hoặc root) với người dùng thông thường. Trong tập hợp lệnh của Linux, có một số lệnh cũng như một số tham số khóa mà chỉ siêu người dùng mới được phép sử dụng.
- Một dòng lệnh có thể có nhiều hơn một lệnh, trong đó lệnh sau được ngăn cách bởi với lệnh đi ngay trước bằng dấu ";" hoặc dấu "|".
- Khi gõ lệnh, nếu dòng lệnh quá dài, Linux cho phép ngắt dòng lệnh xuống dòng dưới bằng cách thêm ký tự báo hiệu chuyển dòng "\" tại cuối dòng.
- Sau khi người dùng gõ xong dòng lệnh, shell tiếp nhận dòng lệnh này và phân tích nội dung văn bản của lệnh. Nếu lệnh được gõ đúng thì nó được thực hiện; ngược lại, trong trường hợp có sai sót khi gõ lệnh thì shell thông báo về sai sót vàdấu nhắc shell lại hiện ra để chờ lệnh tiếp theo của người dùng. Về phổ biến, nếu như sau khi người dùng gõ lệnh, không thấy thông báo sai sót hiện ra thì có nghĩa lệnh đã được thực hiện một cách bình thường.

1.3.2. Các ký hiệu đại diện

Khi chúng ta sử dụng các câu lệnh về file và thư mục, chúng ta có thể sử dụng các ký tự đặc biệt được gọi là các ký tự đại diện để xác định tên file, tên thư mục.:

Ký tự	Ý nghĩa
*	Tương ứng với thứ tự bất kỳ của một hay nhiều ký tự
?	Tương ứng với một ký tự bất kỳ
[]	Tương ứng với một trong những ký tự trong ngoặc hoặc giới hạn

Ví dụ:

Jo* : Các file bắt đầu với Jo

Jo*y : Các file bắt đầu với Jo và kết thúc với y

Ut*l*s.c : Các file bắt đầu với Ut, chứa một ký tự l và kết thúc với s.c

?.h : Các file bắt đầu với một ký tự đơn, theo sau bởi .h

Doc[0-9].txt : Các file có tên Doc0.txt, Doc1.txt ….Doc9.txt

Doc0[A-Z].txt : Các file có tên Doc0A.txt, Doc0B.txt …Doc0Z.txt

Các ký hiệu liên quan đến cú pháp câu lệnh được sử dụng bởi phần lớn các câu lệnh. Chúng cung cấp một cách thuận tiện và đồng nhất để xác định các mẫu phù hợp. Chúng tương tự với các ký tự đại diện, nhưng chúng mạnh hơn rất nhiều. Chúng cung cấp một phạm vi rộng các mẫu lựa chọn.

Ký tự	Ý nghĩa
.	Tương ứng với một ký tự đơn bất kỳ ngoại trừ dòng mới
*	Tương ứng với không hoặc nhiều hơn các ký tự đứng trước
^	Tương ứng với bắt đầu của một dòng
$	Tương ứng với kết thúc một dòng
\<	Tương ứng với bắt đầu một từ
\>	Tương ứng với kết thúc một từ
[]	Tương ứng với một trong các ký tự bên trong hoặc một dãy các ký tự
[^]	Tương ứng với các ký tự bất kỳ không nằm trong ngoặc
\	Lấy ký hiệu theo sau dấu gạch ngược

1.3.3. Trợ giúp lệnh

Do Linux là một hệ điều hành rất phức tạp với hàng nghìn lệnh và mỗi lệnh lại có thể có tới vài hoặc vài chục tình huống sử dụng do chúng cho phép có nhiều tùy chọn lệnh. Để trợ giúp cách sử dụng các câu lệnh, Linux cho phép người dùng sử dụng cách thức gọi trang Man để có được các thông tin đầy đủ giới thiệu nội dung các lệnh.

Cú pháp lệnh: ***# man <tên-lệnh>***

CÂU HỎI VÀ BÀI TẬP

1. Tìm hiểu về các phiên bản phát triển của Linux.
2. Trình bày nguyên tắc thực hiện lệnh trên Linux
3. Thực hiện cài đặt hệ điều hành Linux cụ thể trên máy tính
4. Nghiên cứu các thao tác giao tiếp với Linux; so sánh các thao tác với hệ điều hành Windows.

Chương 2. THAO TÁC VỚI HỆ THỐNG

2.1. Tiến trình khởi động Linux

Một trong những cách thức khởi động Linux phổ biến nhất là cách thức do chương trình LILO (LInux LOader) thực hiện. Chương trình LILO được nạp lên đĩa của máy tính khi cài đặt hệ điều hành Linux. LILO được nạp vào Master Boot Record của đĩa cứng hoặc vào Boot Sector tại phân vùng khởi động (trên đĩa cứng hoặc đĩa mềm). Giả sử máy tính của chúng ta đã cài đặt Linux và sử dụng LILO để khởi động hệ điều hành. LILO thích hợp với việc trên máy tính được cài đặt một số hệ điều hành khác nhau và theo đó, LILO còn cho phép người dùng chọn lựa hệ điều hành để khởi động.

Giai đoạn khởi động Linux tùy thuộc vào cấu hình LILO đã được lựa chọn trong tiến trình cài đặt Linux. Trong tình huống đơn giản nhất, Linux được khởi động từ đĩa cứng hay đĩa mềm khởi động.

Tiến trình khởi động Linux có thể được mô tả theo sơ đồ sau:

Theo sơ đồ này, LILO được tải vào máy để thực hiện mà việc đầu tiên là đưa nhân vào bộ nhớ trong và sau đó tải chương trình init để thực hiện việc khởi động Linux.

Nếu cài đặt nhiều phiên bản Linux hay cài Linux cùng các hệ điều hành khác (trong các trường hợp như thế, mỗi phiên bản Linux hoặc hệ điều hành khác được gán ***nhãn*** - label để phân biệt). Khi đó ta nhập nhãn của một trong những hệ điều hành hiện có trên máy trên dòng thông báo ***LILO boot:***

Ví dụ:

LILO boot: linux

Sau khi Linux đã được chọn để khởi động, trình ***init*** thực hiện, chúng ta sẽ thấy một khoảng vài chục dòng thông báo cho biết hệ thống phần cứng được Linux nhận diện và thiết lập cấu hình cùng với tất cả trình điều khiển phần mềm được nạp khi khởi động. Tại thời điểm khởi động hệ thống ***init*** thực hiện vai trò đầu tiên của mình là chạy chương trình shell trong file ***/etc/inittab*** và các dòng thông báo trên đây chính là kết quả của việc chạy chương trình shell đó. Sau khi chương trình shell trên được thực hiện xong, bắt đầu quá trình người dùng đăng nhập (login) vào hệ thống.

2.2. Thủ tục đăng nhập và các lệnh thoát khỏi hệ thống

2.2.1. Đăng nhập

Sau khi hệ thống Linux khởi động xong, trên màn hình xuất hiện dấu nhắc đăng nhập.

Tại dấu nhắc đăng nhập, ta nhập tên đăng nhập, kèm theo một mật khẩu đăng nhập.

May1 login: root
Password:

Sau khi đăng nhập thành công, dấu nhắc *shell* xuất hiện (#) mời người dùng thực hiện các thao tác tiếp theo.

Last login: Fri Oct 27 14:16:09 on tty2
Root[may1 /root]#

2.2.2. Ra khỏi hệ thống

Có rất nhiều cách cho phép thoát khỏi hệ thống, ở đây chúng ta xem xét một số cách thông dụng nhất.

Dùng tổ hợp phím Ctrl + Alt + Del:

Đây là cách đơn giản nhất để đảm bảo thoát khỏi hệ điều hành Linux. Nếu đang làm việc trong môi trường X Window, cần nhấn tổ hợp phím **Ctrl**+**Alt**+**BackSpace** trước.

Dùng lệnh shutdown:

shutdown [tùy-chọn] <time> [cảnh-báo]

Lệnh này cho phép dừng tất cả các dịch vụ đang chạy trên hệ thống.

Các tùy chọn:

- **-k** : Không thực sự shutdown mà chỉ cảnh báo.
- **-r** : Khởi động lại ngay sau khi shutdown.
- **-h** : Tắt máy thực sự sau khi shutdown.
- **-f** : Khởi động lại nhanh và bỏ qua việc kiểm tra đĩa.
- **-F** : Khởi động lại và thực hiện việc kiểm tra đĩa.
- **-c** : Bỏ qua không chạy lệnh shutdown.
- **-t *s -giây*** : Chờ khoảng thời gian số-giây

Hai tham số vị trí còn lại:

- **time** : Đặt thời điểm shutdown.
- **cảnh-báo** : Cảnh báo đến tất cả người dùng trên hệ thống.

Dùng lệnh halt:

halt [tùy-chọn]

Lệnh này tắt hẳn máy.

Các tuỳ chọn:

- **-w** : không thực sự tắt máy nhưng vẫn ghi các thông tin lên file */var/log/wtmp*
- **-d** : không ghi thông tin lên file */var/log/wtmp.*
- **-n**: có ý nghĩa tương tự như **–d** song không tiến hành việc đồng bộ hóa.
- **-f** : thực hiện tắt máy ngay mà không thực hiện lần lượt việc dừng các dịch vụ có trên hệ thống.
- **-i** : chỉ thực hiện dừng tất cả các dịch vụ mạng trước khi tắt máy.

Chú ý:

- Trước khi thực hiện tắt máy, cần phải lưu lại dữ liệu trước để tránh bị mất
- Có thể sử dụng lệnh **exit** để trở về dấu nhắc đăng nhập hoặc kết thúc phiên làm việc bằng lệnh **logout**.

2.2.3. Khởi động lại hệ thống

Ngoài việc thoát khỏi hệ thống nhờ các cách thức trên đây, khi cần thiết có thể khởi động lại hệ thống nhờ lệnh **reboot**.

Cú pháp lệnh: ***reboot [tùy-chọn]***

Lệnh này cho phép khởi động lại hệ thống. Nói chung thì chỉ siêu người dùng mới được phép sử dụng lệnh **reboot**, tuy nhiên, nếu hệ thống chỉ có duy nhất một người dùng đang làm việc thì lệnh **reboot** vẫn được thực hiện song hệ thống đòi hỏi việc xác nhận mật khẩu.

Các tùy chọn của lệnh **reboot (**là **-w**, **-d**, **-n**, **-f**, **-i)** có ý nghĩa tương tự như trong lệnh **halt**.

2.2.4. Khởi động vào chế độ đồ hoạ

Linux cho phép nhiều chế độ khởi động, những chế độ này được liệt kê trong file */etc/inittab*. Dưới đây là nội dung của file này:

 # inittab This file describes how the INIT process should set up
 # the system in a certain run-level.
#

 # Author: Miquel van Smoorenburg, <miquelsùdrinkel.nl.mugnet.org>
 # Modified for RHS Linux by Marc Ewing and Donnie Barnes
#
 # Default runlevel. The runlevels used by RHS are:
 # 0 - halt (Do NOT set initdefault to this)
 # 1 - Single user mode
 # 2 - Multiuser, without NFS (The same as 3, if you do not have networking)
 # 3 - Full multiuser mode
 # 4 – unused
 # 5 - X11
 # 6 - reboot (Do NOT set initdefault to this)
#
id:3:initdefault:
 # System initialization.
si::sysinit:/etc/rc.d/rc.sysinit
l0:0:wait:/etc/rc.d/rc 0
l0:0:wait:/etc/rc.d/rc 0
l1:1:wait:/etc/rc.d/rc 1
l2:2:wait:/etc/rc.d/rc 2
l3:3:wait:/etc/rc.d/rc 3
l4:4:wait:/etc/rc.d/rc 4
l5:5:wait:/etc/rc.d/rc 5
l6:6:wait:/etc/rc.d/rc 6
 # Things to run in every runlevel.
ud::once:/sbin/update
 # Trap CTRL-ALT-DELETE
ca::ctrlaltdel:/sbin/shutdown -t3 -r now
#ca::ctrlaltdel:/bin/echo "You can't do that"
 # When our UPS tells us power has failed, assume we have a few minutes
# of power left. Schedule a shutdown for 2 minutes from now.
 # This does, of course, assume you have powerd installed and your
 # UPS connected and working correctly.
pf::powerfail:/sbin/shutdown -f -h +2 "Power Failure; System Shutting Down"
 # If power was restored before the shutdown kicked in, cancel it.
pr:12345:powerokwait:/sbin/shutdown -c "Power Restored; Shutdown Cancelled"
 # Run gettys in standard runlevels
1:2345:respawn:/sbin/mingetty tty1
2:2345:respawn:/sbin/mingetty tty2
#3:2345:respawn:/sbin/mingetty tty3
#4:2345:respawn:/sbin/mingetty tty4
#5:2345:respawn:/sbin/mingetty tty5
#6:2345:respawn:/sbin/mingetty tty6
 # Run xdm in runlevel 5
 # xdm is now a separate service
x:5:respawn:/etc/X11/prefdm –nodaemon

Trong đó chế độ khởi động số 3 là chế độ khởi động vào chế độ Text, và chế độ 5 là khởi động vào chế độ đồ hoạ. Như vậy để cho máy tính khởi động vào chế độ đồ hoạ ta sửa lại dòng cấu hình

id:3:initdefault:

thành

id:5:initdefault:

2.3. Một số liên quan đến hệ thống

2.3.1. Lệnh thay đổi mật khẩu

Cú pháp lệnh: ***passwd [tùy-chọn] [tên-người-dùng]***

Các tùy chọn:

-k : Đòi hỏi phải gõ lại mật khẩu cũ trước khi thay đổi mật khẩu mới.
-f : Không cần kiểm tra mật khẩu cũ. (Chỉ supervisor mới có quyền)
-l : Khóa một tài khoản người dùng. (Chỉ supervisor mới có quyền)
-stdin : việc nhập mật khẩu người dùng chỉ được tiến hành từ thiết bị vào chuẩn không thể tiến hành từ đường dẫn (pipe). Nếu không có tham số này cho phép nhập mật khẩu cả từ thiết bị vào chuẩn hoặc từ đường dẫn.
-u : Mở khóa một tài khoản. (Chỉ supervisor mới có quyền)
-d : Xóa bỏ mật khẩu của người dùng. (Chỉ supervisor mới có quyền)
-S : hiển thị thông tin ngắn gọn về trạng thái mật khẩu của người dùng được đưa ra. (Chỉ supervisor mới có quyền)

Nếu tên-người-dùng không có trong lệnh thì ngầm định là chính người dùng đã gõ lệnh này.

2.3.2. Lệnh xem, thiết lập ngày, giờ

Lệnh xem, thiết lập ngày

Cú pháp lệnh: ***date [-tùy_chọn] [ngày giờ]***

Lệnh xem, thiết lập giờ

Cú pháp lệnh: ***time [-tùy_chọn] [+định-dạng]***

Lệnh xem lịch

Cú pháp lệnh: ***cal [tùy-chọn] [<tháng> [<năm>]]***

2.3.3. Lệnh kiểm tra những ai đang sử sụng hệ thống

Cú pháp lệnh: ***who***

Để kiểm tra định danh của người đang sử dụng hiện thời, dùng lệnh: ***who am i***

2.3.4. Thay đổi nội dung dấu nhắc shell

Trong Linux có hai loại dấu nhắc: dấu nhắc cấp một (dấu nhắc shell) xuất hiện khi nhập lệnh và dấu nhắc cấp hai (dấu nhắc nhập liệu) xuất hiện khi lệnh cần có dữ liệu được nhập từ bàn phím và tương ứng với hai biến nhắc tên là **PS1** và **PS2**. **PS1** là biến hệ thống tương ứng với dấu nhắc cấp 1: Giá trị của **PS1** chính là nội dung hiển thị của dấu nhắc shell. Để nhận biết thông tin hệ thống hiện tại, một nhu cầu đặt ra là cần thay đổi giá trị của các biến hệ thống **PS1** và **PS2**.

Linux cho phép thay đổi giá trị của biến hệ thống **PS1** bằng lệnh gán trị mới cho nó. Lệnh này có dạng:

 # PS1='<dãy ký tự>'

Năm (5) ký tự đầu tiên của lệnh gán phải được viết liên tiếp nhau. Dãy ký tự nằm giữa cặp hai dấu nháy đơn (có thể sử dụng cặp hai dấu kép) và không được phép chứa dấu nháy. Dãy ký tự này bao gồm các cặp ký tự điều khiển và các ký tự khác, cho phép có thể có dấu cách. Cặp ký tự điều khiển gồm hai ký tự, ký tự đầu tiên là dấu sổ xuôi "\" còn ký tự thứ hai nhận một trong các trường hợp liệt kê trong bảng dưới đây.

Cặp ký tự điều khiển	Ý nghĩa
\!	Hiển thị thứ tự của lệnh trong lịch sử
\#	Hiển thị thứ tự của lệnh
\$	Hiển thị dấu $. Đối với superuser thì hiển thị dấu #

Cặp ký tự điều khiển	Ý nghĩa
\\	Hiển thị dấu sổ (\)
\d	Hiển thị ngày hiện tại
\h	Hiển thị tên máy (hostname)
\n	Ký hiệu xuống dòng
\s	Hiển thị tên hệ shell
\t	Hiển thị giờ hiện tại
\u	Hiển thị tên người dùng
\W	Hiển thị tên thực sự của thư mục hiện thời
\w	Hiển thị tên đầy đủ của thư mục hiện thời

2.3.5. Lệnh gọi ngôn ngữ tính toán số học

Linux cung cấp một ngôn ngữ tính toán với độ chính xác tùy ý thông qua lệnh **bc**. Khi yêu cầu lệnh này, người dùng được cung cấp một ngôn ngữ tính toán (và cho phép lập trình tính toán có dạng ngôn ngữ lập trình C) hoạt động theo thông dịch. Trong ngôn ngữ lập trình được cung cấp (tạm thời gọi là ngôn ngữ ***bc***), tồn tại rất nhiều công cụ hỗ trợ tính toán và lập trình tính toán: kiểu phép toán số học phong phú, phép toán so sánh, một số hàm chuẩn, biến chuẩn, cấu trúc điều khiển, cách thức định nghĩa hàm, cách thức thay đổi độ chính xác, đặt lời chú thích... Chỉ cần sử dụng một phần nhỏ tác động của lệnh **bc**, chúng ta đã có một "máy tính số bấm tay" hiệu quả.

Cú pháp lệnh: ***bc [tùy-chọn] [file...]***

Các tuỳ chọn:

-l, --mathlib : phép tính theo chuẩn thư viện toán học
-w, --warn : thực hiện phép tính không tuân theo chuẩn POSIX
-s, --standard : phép tính chính xác theo chuẩn của ngôn ngữ POSIX
-q, --quiet : không hiện ra lời giới thiệu về phần mềm GNU

CÂU HỎI VÀ BÀI TẬP

1. Thực hiện các thao tác đăng nhập, thoát khỏi hệ thống trên hệ điều hành Linux
2. Thực hiện các thao tác liên quan đến hệ thống Linux
3. Thực hiện các thao tác sử dụng lệnh bc để tính toán một bài toán số học đơn giản.

Chương 3. HỆ THỐNG FILE

3.1 Tổng quan về hệ thống file

3.1.1. Một số khái niệm

File là một tập hợp dữ liệu có tổ chức được hệ điều hành quản lý theo yêu cầu của người dùng. Cách tổ chức dữ liệu trong file thuộc về chủ của nó là người đã tạo ra file. Hệ điều hành đảm bảo các chức năng liên quan đến file nên người dùng không cần biết file của mình lưu ở vùng nào trên đĩa từ, bằng cách nào đọc/ghi lên các vùng của đĩa từ mà vẫn thực hiện được yêu cầu tìm kiếm, xử lý lên các file.

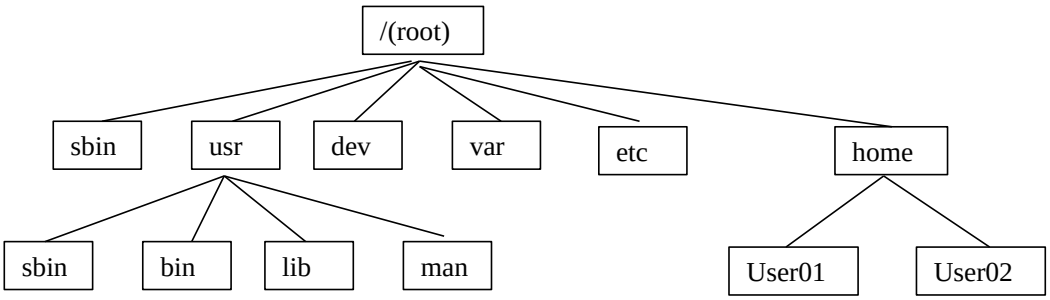
Hệ điều hành quản lý các file theo tên gọi của file (tên file) và một số thuộc tính liên quan đến file.

Để làm việc được với các file, hệ điều hành không chỉ quản lý nội dung file mà còn phải quản lý các thông tin liên quan đến các file. Thư mục (directory) là đối tượng được dùng để chứa thông tin về các file (thư mục chứa các file). Các thư mục cũng được hệ điều hành quản lý trên thiết bị lưu trữ ngoài và vì vậy, theo nghĩa này, thư mục cũng được coi là file song trong một số trường hợp để phân biệt với "file" thư mục, chúng ta dùng thuật ngữ ***file thông thường***. Khác với file thông thường, hệ điều hành lại quan tâm đến nội dung của thư mục.

Một số nội dung sau đây liên quan đến tên file (bao gồm cả tên thư mục):

- Tên file trong Linux có thể dài tới 256 ký tự, bao gồm các chữ cái, chữ số, dấu gạch nối, gạch chân, dấu chấm. Nếu trong tên file có nhiều dấu chấm "." thì xâu con của tên file từ dấu chấm cuối cùng được gọi là phần mở rộng của tên file
- Có phân biệt chữ hoa và chữ thường đối với tên thư mục/file
  Nếu trong tên thư mục/file có chứa khoảng trống, sẽ phải đặt tên thư mục/file vào trong cặp dấu nháy kép để sử dụng thư mục/file đó.
- Một số ký tự sau không được sử dụng trong tên thư mục/file: !, *, $, &, #...
  Khi sử dụng chương trình **mc** (Midnight Commander), việc hiển thị tên file sẽ bổ sung một ký tự theo nghĩa: dấu "*" cho file khả thi trong Linux, dấu "~" cho file sao lưu, dấu "." cho file ẩn, dấu "@" cho file liên kết...

Tập hợp tất cả các file có trong hệ điều hành được gọi là ***hệ thống file*** là một hệ thống thống nhất. Hệ thống file có cấu trúc hình cây, được xuất phát từ một thư mục gốc (ký hiệu là "/") và cho phép tạo ra thư mục con trong một thư mục bất kỳ. Thông thường, khi khởi tạo Linux đã có ngay hệ thống file của nó.



Để chỉ một file hay một thư mục, chúng ta cần đưa ra một đường dẫn. Ví dụ để xác định file **user01**, chúng ta viết như sau: **/home/user01**

Đường dẫn file xác định từ thư mục gốc được biết với tên gọi là **đường dẫn tuyệt đối** sẽ rất khó khăn cho người dùng khi thực hiện gõ lệnh. Vì vậy, Linux (cũng như nhiều hệ điều hành khác) sử dụng khái niệm **thư mục hiện thời** (là một thư mục trong hệ thống file mà hiện thời "người dùng đang ở đó") của mỗi người dùng làm việc trong hệ thống. Qua thư mục hiện thời, Linux cho phép người dùng chỉ một file trong lệnh ngắn gọn hơn nhiều.

Đường dẫn được xác định qua thư mục hiện thời được gọi là **đường dẫn tương đối**.

Khi một người dùng đăng nhập vào hệ thống, Linux luôn chuyển người dùng vào thư mục riêng, và tại thời điểm đó thư mục riêng là thư mục hiện thời của người dùng. Thư mục riêng của siêu người dùng là **/root**, thư mục riêng của người dùng có tên là ***user01*** là **/home/user1...** Linux cho phép dùng lệnh **cd** để chuyển sang thư mục khác (lấy thư mục khác làm thư mục hiện thời). Hai dấu chấm ".." được dùng để chỉ thư mục ngay trên thư mục hiện thời (cha của thư mục hiện thời).

Linux còn cho phép ghép một hệ thống file trên một thiết bị nhớ (đĩa mềm, vùng đĩa cứng chưa được đưa vào hệ thống file) thành một thư mục con trong hệ thống file của hệ thống bằng lệnh ***mount***. Các hệ thống file được ghép thuộc vào các kiểu khác nhau.

3.1.2. Sơ bộ kiến trúc nội tại của hệ thống file

Trên đĩa từ, hệ thống file được coi là dãy tuần tự các khối lôgic mỗi khối chứa hoặc 512B hoặc 1024B hoặc bội của 512B là cố định trong một hệ thống file. Trong hệ thống file, các khối dữ liệu được địa chỉ hóa bằng cách đánh chỉ số liên tiếp, mỗi địa chỉ được chứa trong 4 byte (32 bit).

Cấu trúc nội tại của hệ thống file bao gồm 4 thành phần kế tiếp nhau: Boot block (dùng để khởi động hệ thống), Siêu khối (Super block), Danh sách inode và Vùng dữ liệu.

Siêu khối

Siêu khối chứa nhiều thông tin liên quan đến trạng thái của hệ thống file. Trong siêu khối có các trường sau đây:

- Kích thước của danh sách inode (định kích cỡ vùng không gian trên Hệ thống file quản lý các inode).
- Kích thước của hệ thống file.

  Hai kích thước trên đây tính theo đơn vị dung lượng bộ nhớ ngoài,
- Một danh sách chỉ số các khối rỗi (thường trực trên siêu khối) trong hệ thống file. Chỉ số các khối rỗi thường trực trên siêu khối được dùng để đáp ứng nhu cầu phân phối mới.
- Chỉ số của khối rỗi tiếp theo trong danh sách các khối rỗi. Chỉ số khối rỗi tiếp theo dùng để hỗ trợ việc tìm kiếm tiếp các khối rỗi: bắt đầu tìm từ khối có chỉ số này trở đi.
- Một danh sách các inode rỗi (thường trực trên siêu khối) trong hệ thống file. Danh sách này chứa chỉ số các inode rỗi được dùng để phân phối ngay được cho một file mới được khởi tạo.
- Chỉ số inode rỗi tiếp theo trong danh sách các inode rỗi. Chỉ số inode rỗi tiếp theo định vị việc tìm kiếm tiếp thêm inode rỗi: bắt đầu tìm từ inode có chỉ số này trở đi.

  Hai tham số trên đây tạo thành cặp xác định được danh sách các inode rỗi trên hệ thống file các thao tác tạo file mới, xoá file cập nhật thông tin này.
- Các trường khóa (lock) danh sách các khối rỗi và danh sách inode rỗi: Trong một số trường hợp, chẳng hạn khi hệ thống đang làm việc thực sự với đĩa từ để cập nhật các danh sách này, hệ thống không cho phép cập nhật tới hai danh sách nói trên.
- Cờ chỉ dẫn về việc siêu khối đã được biến đổi: Định kỳ thời gian siêu khối ở bộ nhớ trong được cập nhật lại vào siêu khối ở đĩa từ và vì vậy cần có thông tin về việc siêu khối ở bộ nhớ trong khác với nội dung ở bộ nhớ ngoài: nếu hai bản không giống nhau thì cần phải biến đổi để chúng được đồng nhất.
- Cờ chỉ dẫn rằng hệ thống file chỉ có thể đọc (cấm ghi): Trong một số trường hợp, hệ thống đang cập nhật thông tin từ bộ nhớ ngoài thì chỉ cho phép đọc đối với hệ thống file,
- Số lượng tổng cộng các khối rỗi trong hệ thống file,
- Số lượng tổng cộng các inode rỗi trong hệ thống file,

- Thông tin về thiết bị,
- Kích thước khối (đơn vị phân phối dữ liệu) của hệ thống file. Hiện tại kích thước phổ biến của khối là 1KB.

Trong thời gian máy hoạt động, theo từng giai đoạn, nhân sẽ đưa siêu khối lên đĩa nếu nó đã được biến đổi để phù hợp với dữ liệu trên hệ thống file.

Inode

Mỗi khi một tiến trình khởi tạo một file mới, nhân hệ thống sẽ gán cho nó một inode chưa sử dụng. Để hiểu rõ hơn về inode, chúng ta xem xét sơ lược mối quan hệ liên quan giữa file dữ liệu và việc lưu trữ trên vật dẫn ngoài đối với Linux.

Nội dung của file được chứa trong vùng dữ liệu của hệ thống file và được phân chia các khối dữ liệu (chứa nội dung file) và hình ảnh phân bố nội dung file có trong một inode tương ứng. Liên kết đến tập hợp các khối dữ liệu này là một inode, chỉ thông qua inode mới có thể làm việc với dữ liệu tại các khối dữ liệu: Inode chứa dựng thông tin về tập hợp các khối dữ liệu nội dung file. Có thể quan niệm rằng, tổ hợp gồm inode và tập các khối dữ liệu như vậy là một file vật lý: inode có thông tin về file vật lý, trong đó có địa chỉ của các khối nhớ chứa nội dung của file vật lý. Thuật ngữ inode là sự kết hợp của hai từ index với node và được sử dụng phổ dụng trong Linux.

Các inode được phân biệt nhau theo chỉ số của inode: đó chính là số thứ tự của inode trong danh sách inode trên hệ thống file. Thông thường, hệ thống dùng 2 bytes để lưu trữ chỉ số của inode. Với cách lưu trữ chỉ số như thế, không có nhiều hơn 65535 inode trong một hệ thống file.

Như vậy, một file chỉ có một inode song một file lại có một hoặc một số tên file. Người dùng tác động thông qua tên file và tên file lại tham chiếu đến inode (tên file và chỉ số inode là hai trường của một phần tử của một thư mục). Một inode có thể tương ứng với một hoặc nhiều tên file, mỗi tương ứng như vậy được gọi là một liên kết. Inode được lưu trữ tại vùng danh sách các inode.

Trong tiến trình làm việc, Linux dùng một vùng bộ nhớ, được gọi là bảng inode (trong một số trường hợp, nó còn được gọi tường minh là bảng sao in-core inode) với chức năng tương ứng với vùng danh sách các inode có trong hệ thống file, hỗ trợ cho tiến trình truy nhập dữ liệu trong hệ thống file. Nội dung của một in-core inode không chỉ chứa các thông tin trong inode tương ứng mà còn được bổ sung các thông tin mới giúp cho tiến trình xử lý inode.

Inode bao gồm các trường thông tin sau đây:

- Kiểu file. Trong Linux phân loại các kiểu file: file thông thường (regular), thư mục, đặc tả ký tự, đặc tả khối và ống dẫn FIFO (pipes). Linux quy định trường kiểu file có giá trị 0 tương ứng đó là inode chưa được sử dụng.
- Quyền truy nhập file. Trong Linux, file là một tài nguyên chung của hệ thống vì vậy quyền truy nhập file được đặc biệt quan tâm để tránh những trường hợp truy nhập không hợp lệ. Đối với một inode, có 3 mức quyền truy nhập liên quan đến các đối tượng:
  - Mức chủ của file (ký hiệu là ***u***),
  - Mức nhóm người dùng của chủ nhân của file (ký hiệu là ***g***),
  - Mức người dùng khác (ký hiệu là ***a***).
- Số lượng liên kết đối với inode: Đây chính là số lượng các tên file trên các thư mục được liên kết với inode này,
- Định danh chủ nhân của inode,
- Định danh nhóm chủ nhân: xác định tên nhóm người dùng mà chủ file là một thành viên của nhóm này,
- Độ dài của file tính theo byte,
- Thời gian truy nhập file:
  - Thời gian file được sửa đổi muộn nhất,

- Thời gian file được truy nhập muộn nhất,
- Thời gian file được khởi tạo,

Bảng chứa địa chỉ khối dữ liệu của File trong UNIX

Bảng chứa địa chỉ khối dữ liệu của file gồm 13 phần tử với 10 phần tử trực tiếp và 3 phần tử gián tiếp: Mỗi phần tử có độ dài 4 bytes, chứa một số hiệu của một khối nhớ trên đĩa. Mỗi phần tử trực tiếp trỏ tới 1 khối dữ liệu thực sự chứa nội dung file. Phần tử gián tiếp bậc 1 (single indirect) trỏ tới 1 khối nhớ ngoài. Khác với phần tử trực tiếp, khối nhớ ngoài này không dùng để chứa dữ liệu của file mà lại chứa danh sách chỉ số các khối nhớ ngoài và chính các khối nhớ ngoài này mới thực sự chứa nội dung file. Như vậy, nếu khối có độ dài 1KB và một chỉ số khối ngoài có độ dài 4 bytes thì địa chỉ gián tiếp cho phép định vị không gian trên đĩa lưu trữ dữ liệu của file tới 256KB (Không gian bộ nhớ ngoài trong vùng dữ liệu phải dùng tới là 257KB). Tương tự đối với các phần tử gián tiếp mức cao hơn.

Cơ chế quản lý địa chỉ file như trên cho thấy có sự phân biệt giữa file nhỏ với file lớn. File nhỏ có độ dài bé hơn và theo cách tổ chức như trên, phương pháp truy nhập sẽ cho phép tốc độ nhanh hơn, đơn giản hơn do chỉ phải làm việc với các phần tử trực tiếp. Khi xử lý, thuật toán đọc File tiến hành theo các cách khác nhau đối với các phần tử trực tiếp và gián tiếp.

Vùng dữ liệu

Bao gồm các khối dữ liệu, mỗi khối dữ liệu được đánh chỉ số để phân biệt. Khối trên vùng dữ liệu được dùng để chứa nội dung các file, nội dung các thư mục và nội dung các khối định vị địa chỉ của các file. Chú ý rằng, chỉ số của khối dữ liệu được chứa trong 32 bit và thông tin này xác định dung lượng lớn nhất của hệ thống file.

3.1.3. Hỗ trợ nhiều hệ thống File

Các phiên bản đầu tiên của Linux chỉ hỗ trợ một hệ thống file duy nhất đó là hệ thống file minix. Sau đó, với sự mở rộng nhân, cộng đồng Linux đã thêm vào nó rất nhiều kiểu hệ thống file khác nhau và Linux trở thành một hệ điều hành hỗ trợ rất nhiều hệ thống file như:

- Hệ thống file CODA: Là một hệ thống file mạng cho phép người dùng có thể kết gán các hệ thống file từ xa và truy cập chúng như các hệ thống file cục bộ.
- Hệ thống file EFS: Là một dạng hệ thống file sử dụng cho CDROM.
- Hệ thống file EXT2: (The second extended file system) là hệ thống được dùng chủ yếu trên các phiên bản của hệ điều hành Linux.
- Hệ thống file HFS: Là hệ thống file chạy trên các máy Apple Macintosh.
- Hệ thống file HPFS: Là hệ thống file được sử dụng trong hệ điều hành OS/2. Linux hỗ trợ hệ thống file này ở mức chỉ đọc.
- Hệ thống file ISOFS: Là hệ thống file được sử dụng cho các đĩa CD. Hệ thống thông dụng nhất cho các đĩa CD hiện nay là ISO 9660.
- Hệ thống file MSDOS: Với sự hỗ trợ này, hệ thống Linux có thể truy cập được các phân vùng của hệ điều hành MSDOS. Linux cũng có thể sử dụng kiểu MSDOS để truy cập các phân vùng của Window 95/98 tuy nhiên khi đó, các ưu điểm của hệ điều hành Window sẽ không còn giá trị ví dụ như tên file chỉ tối đa 13 ký tự (kể cả mở rộng).
- Hệ thống file NFS: (Network File System) là một hệ thống file trên mạng hỗ trợ việc truy cập dữ liệu từ xa giống như hệ thống file CODA. Với NFS, các máy chạy Linux có thể chia sẻ các phân vùng đĩa trên mạng để sử dụng như là các phân vùng cục bộ của chính máy mình.
- Hệ thống file NTFS: Với sự hỗ trợ này, hệ thống Linux có thể truy cập vào các phân vùng của hệ điều hành Microsoft Window NT.
- Hệ thống file ROMFS: Là các hệ thống file chỉ đọc (read only) được sử dụng chủ yếu cho việc khởi tạo đĩa ảo (ramdisk) trong tiến trình khởi động đĩa cài đặt.

- Hệ thống file SMB: (Server Mesage Block) là một giao thức của Windows dùng để chia sẻ file giữa các hệ điều hành Windows 95/98, Windows NT và OS/2 Lan Manager. Với sự hỗ trợ SMB, hệ điều hành Linux có thể chia sẻ cũng như truy cập các file nằm trên các phân vùng của một máy chạy các hệ điều hành kể trên.
- Hệ thống file VFAT: Là hệ thống file mở rộng của hệ thống FAT. Hệ thống file này được sử dụng trong các hệ điều hành Windows 95/98.
- …

Như vậy, ngoài khả năng hỗ trợ nhiều loại thiết bị, Linux còn có khả năng hỗ trợ nhiều kiểu hệ thống file. Bằng cách hỗ trợ nhiều kiểu hệ thống file, Linux có thể truy cập và xử lý các file của nhiều hệ điều hành khác nhau. Mặc dù có khả năng truy cập nhiều hệ thống file khác nhau, hệ thống file của Linux vẫn phải đảm bảo cung cấp cho người dùng một giao diện nhất quán đối với các file, bảo vệ các file trên các hệ thống khác nhau, tối ưu các thao tác truy cập vào thiết bị... Để thực hiện được điều này, Linux sử dụng một hệ thống file đặc biệt gọi là hệ thống file ảo VFS (Virtual File System).

Hệ thống file ảo VFS được thiết kế để cung cấp một giao diện thống nhất về các file được lưu trữ trên các thiết bị. VFS có trách nhiệm cung cấp cho chương trình người dùng một giao diện nhất quán về hệ thống file thông qua các lệnh gọi hệ thống (system call). Mỗi khi có một yêu cầu truy cập file, VFS sẽ dựa vào các hệ thống file thực để tìm kiếm file yêu cầu trên các thiết bị vật lý. Với mỗi file tìm được, nó thực hiện thao tác mở file đó và cho tương ứng file với một cấu trúc dữ liệu gọi là i-node. VFS cung cấp rất nhiều lệnh gọi để thao tác với hệ thống file nhưng chủ yếu thuộc vào các loại sau:

- Các thao tác liên quan tới hệ thống file.
- Các thao tác liên quan tới i-node.
- Các thao tác với file đang mở.
- Các thao tác với vùng đệm dữ liệu.

3.1.4. Liên kết tượng trưng (lệnh ln)

Trong Linux có hai kiểu liên kết đó là liên kết tượng trưng (liên kết mềm) và liên kết cứng.

- "Liên kết cứng" là một cách gọi khác đối với một file đang tồn tại (không có sự phân biệt giữa file gốc và file liên kết). Theo cách nói kỹ thuật, chúng cùng chia sẻ một inode và inode này chứa đựng tất cả các thông tin về file. Không thể tạo một liên kết cứng tới một thư mục.

- "Liên kết tượng trưng" là một kiểu file đặc biệt, trong đó, một file liên kết thực sự tham chiếu theo tên đến một file khác. Có thể hiểu kiểu file này như là một con trỏ chỉ dẫn tới một file hoặc một thư mục, và được sử dụng để thay thế cho file hoặc thư mục được trỏ tới. Hầu hết các thao tác (như mở, đọc, ghi...) được thực hiện trên các file liên kết, sau đó, nhân hệ thống sẽ tự động "tham chiếu" và thực hiện trên file đích của liên kết. Tuy nhiên, có một số các thao tác như xóa file, file liên kết sẽ bị xóa bỏ chứ không phải file đích của nó.

Cú pháp lệnh: ***ln [tùy-chọn] <đích> [tên-nối]***

Lệnh này sẽ tạo một liên kết đến thư mục/file đ**ích** với tên file liên kết là **tên-nối**. Nếu **tên-nối** không có, một liên kết với tên file liên kết giống như tên file đ**ích** sẽ được tạo ra trong thư mục hiện thời.

Các tuỳ chọn:

- **-b, --backup[=CONTROL]** : tạo liên kết quay trở lại cho mỗi file đích đang tồn tại.
- **-f, --force** : xóa bỏ các file đích đang tồn tại.
- **-d, -F, --directory** : tạo liên kết cứng đến các thư mục (chỉ dành cho người dùng có quyền quản trị hệ thống; Một số phiên bản không có tùy chọn này).

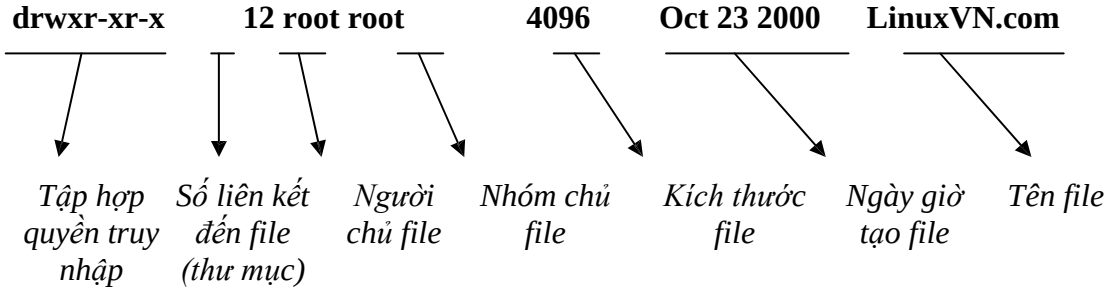
- **-n, --no-dereference** : một file bình thường được xem là đích liên kết từ một thư mục.
- **-i, interactive** : vẫn tạo liên kết dù file đích đã bị xóa bỏ.
- **-s, --symbolic** : tạo các liên kết tượng trưng.
- **--target-directory=<tên-thư-mục>** : xác định thư mục tên-thư-mục là thư mục có chứa các liên kết.
- **-v, --verbose** : hiển thị tên các file trước khi tạo liên kết.
- **--help** : hiển thị trang trợ giúp và thoát.

3.2 Quyền truy nhập thư mục và file

3.2.1 Quyền truy nhập

Mỗi file và thư mục trong Linux đều có một chủ sở hữu và một nhóm sở hữu, cũng như một tập hợp các quyền truy nhập. Cho phép thay đổi các quyền truy nhập và quyền sở hữu file và thư mục nhằm cung cấp truy nhập nhiều hơn hay ít hơn.

Thông tin về một file có dạng sau (được hiện ra theo lệnh hiện danh sách file **ls -l**):



Trong đó, dãy 10 ký tự đầu tiên mô tả kiểu file và quyền truy nhập đối với tập tin đó, chúng được chia ra làm 4 phần: kiểu file, các quyền truy nhập đến file của chủ sở hữu, của nhóm sở hữu và người dùng khác.

Theo mặc định, người dùng tạo một file chính là người chủ (sở hữu) của file đó và là người có quyền sở hữu nó. Người chủ của file có đặc quyền thay đổi quyền truy nhập hay quyền sở hữu đối với file đó. Tất nhiên, một khi đã chuyển quyền sở hữu của mình cho người dùng khác thì người chủ cũ không được phép chuyển quyền sở hữu và quyền truy nhập được nữa.

Có một số kiểu file trong Linux. Ký tự đầu tiên trong tập hợp 10 ký tự mô tả kiểu file và quyền truy nhập sẽ cho biết file thuộc kiểu nào (chữ cái đó được gọi là chữ cái biểu diễn).

Chữ cái biểu diễn	Kiểu file
d	Thư mục (directory)
b	File kiểu khối (block-type special file)
c	File kiểu ký tự (character-type special file)
l	Liên kết tượng trưng (symbolic link)
p	File đường ống (pipe)
s	Socket
-	File bình thường (regular file)

Chín ký tự tiếp theo trong chuỗi là quyền truy nhập được chia ra làm 3 nhóm tương ứng với quyền truy nhập của người sử hữu, nhóm sở hữu và người dùng khác.

Có ba loại quyền truy nhập chính đối với thư mục/file, đó là: đọc (read -r), ghi (write -w) và thực hiện (execute -x).

Chữ cái biểu diễn	Kiểu file
---	Không cho phép một quyền truy nhập nào
r--	Chỉ được quyền đọc
r-x	Quyền đọc và thực hiện (cho chương trình và shell script)
rw-	Quyền đọc và ghi
rwx	Cho phép tất cả các quyền truy nhập (cho chương trình)

Tuy nhiên, đối với thư mục thì chỉ có ba loại ký hiệu của các quyền truy nhập là: ---, **r-x** và **rwx**, vì nội dung của thư mục là danh sách của các file và các thư mục con có bên trong thư mục đó. Quyền đọc một thư mục là được xem nội dung của thư mục đó và quyền thực hiện đối với một thư mục là quyền tìm được file và thư mục con có trong thư mục.

3.2.2. Các lệnh cơ bản

3.2.2.1. Thay đổi quyền sở hữu file

Cú pháp lệnh: ***chown [tùy-chọn] [chủ][.nhóm] <file...>***

- Nếu chỉ có tham số về ***chủ***, thì người dùng ***chủ*** sẽ có quyền sở hữu file và nhóm sở hữu không thay đổi.
- Nếu theo sau tên người ***chủ*** là dấu "**.**" và tên của một ***nhóm*** thì nhóm đó sẽ nhóm sở hữu file.
- Nếu chỉ có dấu "**.**" và ***nhóm*** mà không có tên người chủ thì chỉ có quyền sở hữu nhóm của file thay đổi, lúc này, lệnh **chown** có tác dụng giống như lệnh **chgrp**

Các tùy chọn:

- -c, --changes : hiển thị dòng thông báo chỉ với các file mà lệnh làm thay đổi sở hữu (số thông báo hiện ra có thể ít hơn trường hợp **-v, -verbosr**).
- -f, --silent, --quiet : bỏ qua hầu hết các thông báo lỗi.
- -R, --recursive : thực hiện đổi quyền sở hữu đối với thư mục và file theo đệ quy.
- -v, --verbose : hiển thị dòng thông báo với mọi file liên quan mà chown tác động tới (có hoặc không thay đổi sở hữu).
- --help : đưa ra trang trợ giúp và thoát.

3.2.2.2. Thay đổi quyền sở hữu nhóm

Các file (và người dùng) còn thuộc vào các nhóm, đây là phương thức truy nhập file thuận tiện cho nhiều người dùng nhưng không phải tất cả người dùng trên hệ thống. Khi đăng nhập, mặc định sẽ là thành viên của một nhóm được thiết lập khi siêu người dùng root tạo tài khoản người dùng. Cho phép một người dùng thuộc nhiều nhóm khác nhau, nhưng mỗi lần đăng nhập chỉ là thành viên của một nhóm.

Cú pháp lệnh: ***chgrp [tùy-chọn] {nhóm|--reference=nhómR} <file...>***

Lệnh này cho phép thay thuộc tính nhóm sở hữu của file theo tên nhóm được chỉ ra trực tiếp theo tham số **nhóm** hoặc gián tiếp qua thuộc tính nhóm của file có tên là **nhómR**.

Các tùy chọn: (một số tương tự như ở lệnh **chown**):

- -c, --changes : hiển thị dòng thông báo chỉ với các file mà lệnh làm thay đổi sở hữu (số thông báo hiện ra có thể ít hơn trường hợp **-v, -verbosr**).
- -f, --silent, --quiet : bỏ qua hầu hết các thông báo lỗi.
- -R, --recursive : thực hiện đổi quyền sở hữu đối với thư mục và file theo đệ quy.
- -v, --verbose : hiển thị dòng thông báo với mọi file liên quan mà chgrp tác động tới (có hoặc không thay đổi sở hữu).
- --help : hiển thị trang trợ giúp và thoát

Tham số *--reference=nhómR* cho thấy cách gián tiếp thay nhóm chủ của file theo nhóm chủ của một file khác (tên là nhómR) là cách thức được ưa chuộng hơn. Tham số này là xung khắc với tham số nhóm của lệnh.

3.2.2.3. Thay đổi quyền truy cập file

Cú pháp lệnh **chmod** có ba dạng:

chmod [tùy-chọn] <mod [,mod]...> <file...>

chmod [tùy-chọn] <mod-hệ-8> <file...>

chmod [tùy-chọn] --reference=nhómR <file...>

Lệnh **chmod** cho phép xác lập quyền truy nhập theo kiểu (**mode)** trên **file**. Dạng đầu tiên là dạng xác lập tương đối, dạng thứ hai là dạng xác lập tuyệt đối và dạng cuối cùng là dạng gián tiếp chỉ dẫn theo quyền truy nhập của file **nhómR**.

Các tùy chọn (*-c,--changes; -f, --silent, --quiet; -v, --verbose; -R, --recursive; --help*) có ý nghĩa tương tự các tuỳ chọn tương ứng của các lệnh **chown**, **chgrp**:

Tham số **--reference=RFILE** cũng ý nghĩa gián tiếp như trong lệnh **chgrp**.

Giải thích về hai cách xác lập quyền truy nhập file trong lệnh **chmod** như sau: xác lập tuyệt đối (dùng hệ thống mã số viết theo hệ cơ số 8 biểu diễn cho các quyền truy nhập) và xác lập tương đối (dùng các chữ cái để biểu diễn quyền truy nhập).

Cách xác lập tương đối

Cách xác lập tương đối là dễ nhớ theo ý nghĩa của nội dung các **mod** và chỉ những thay đổi thực sự mới được biểu diễn trong lệnh.

Quyền truy cập	Thao tác thay đổi	Kiểu truy cập
u=user (người sở hữu) **g=group** (nhóm sở hữu) **o=other** (người khác) **a=all** (tất cả người dùng)	+ (thêm quyền) - (gỡ bỏ quyền) = (xác nhận quyền)	**r=read** (đọc) **w=write** (ghi) **x=execute** (thực hiện)

Ví dụ: ***chmod g+w test***

Cách xác lập tuyệt đối

Đối với người dùng hiểu sơ bộ về biểu diễn số trong hệ cơ số 8 thì cách xác lập tuyệt đối lại được ưa chuộng hơn.

Ta đã biết quyền truy nhập file xác định thông qua dãy gồm 9 vị trí dưới dạng **rwxrwxrwx,** Như vậy thuộc tính quyền truy nhập của một file có thể biểu diễn thành 9 bít nhị phân trong đó bít có giá trị 1 thì quyền đó được xác định, ngược lại thì quyền đó bị tháo bỏ. Như vậy, chủ sở hữu tương ứng với 3 bít đầu tiên, nhóm sở hữu tương ứng với 3 bít giữa, người dùng khác tương ứng với 3 bít cuối. Mỗi cụm 3 bít như vậy cho một chữ số hệ 8 (nhận giá trị từ 0 đến 7) và thuộc tính quyền truy nhập tương ứng với 3 chữ số hệ 8.

Quyền	Chữ số hệ 8	Quyền	Chữ số hệ 8
Chỉ đọc	4	Chỉ đọc và ghi	6
Chỉ ghi	2	Chỉ đọc và thực hiện	5
Chỉ thực hiện	1	Chỉ ghi và thực hiện	3
Không có quyền nào	0	Đọc, ghi và thực hiện	7

3.2.2.4. Đăng nhập vào một nhóm người dùng mới

Linux cho phép một người dùng có thể là thành viên của một hoặc nhiều nhóm người dùng khác nhau, trong đó có một nhóm được gọi là nhóm khởi động. Điều này được đảm bảo khi thực hiện lệnh **adduser** hoặc **usersdd**. Tuy nhiên, tại một thời điểm, một người dùng thuộc vào chỉ một nhóm. Khi một người dùng đăng nhập, hệ thống ngầm định người dùng đó là thành viên của nhóm khởi động, và có quyền truy nhập đối với những file thuộc quyền sở hữu của nhóm khởi động đó. Nếu muốn sử dụng quyền sở hữu theo các nhóm khác đối với

những file thì người dùng phải chuyển đổi thành thành viên của một nhóm những nhóm đã được gắn với người dùng.

Lệnh **newgr** cho phép người dùng chuyển sang nhóm người dùng khác đã gắn với mình với Cú pháp lệnh:

newgrp [nhóm]

Trong đó nhóm là một tên nhóm người dùng tồn tại trong hệ thống.

3.3 Thao tác với thư mục

3.3.1 Một số thư mục đặc biệt

* Thư mục gốc **/**

Đây là thư mục gốc chứa đựng tất cả các thư mục con có trong hệ thống.

* Thư mục ***/root***

Thư mục **/root** có thể được coi là "thư mục riêng" của siêu người dùng. Thư mục này được sử dụng để lưu trữ các file tạm thời, nhân Linux và ảnh khởi động, các file nhị phân quan trọng (những file được sử dụng đến trước khi Linux có thể gắn kết đến phân vùng **/user**), các file đăng nhập quan trọng, bộ đệm in cho việc in ấn, hay vùng lưu tạm cho việc nhận và gửi email. Nó cũng được sử dụng cho các vùng trống tạm thời khi thực hiện các thao tác quan trọng, ví dụ như khi xây dựng (**build**) một gói RPM từ các file RPM nguồn.

* Thư mục ***/bin***

Trong Linux, chương trình được coi là khả thi nếu nó có thể thực hiện được. Khi một chương trình được biên dịch, nó sẽ có dạng là file nhị phân. Như vậy, chương trình ứng dụng trong Linux là một file nhị phân khả thi.Chính vì lẽ đó, những nhà phát triển Linux đã quyết định phải tổ chức một thư mục "binaries" để lưu trữ các chương trình khả thi có trên hệ thống, đó chính là thư mục **/bin.**

Ban đầu, thư mục **/bin** (bin là viết tắt của từ **binary**) là nơi lưu trữ các file nhị phân khả thi. Nhưng theo thời gian, ngày càng có nhiều hơn các file khả thi có trong Linux, do đó, có thêm các thư mục như **/sbin, /usr/bin** được sử dụng để lưu trữ các file đó.

* Thư mục ***/dev***

Một phần không thể thiếu trong bất kỳ máy tính nào đó là các trình điều khiển thiết bị. Không có chúng, sẽ không thể có được bất kỳ thông tin nào trên màn hình của, cũng không thể nhập được thông tin, và cũng không thể sử dụng đĩa mềm của.

Tất cả các trình điều khiển thiết bị đều được lưu trữ trong thư mục **/dev.**

* Thư mục ***/etc***

Quản trị hệ thống trong Linux không phải là đơn giản, chẳng hạn như việc quản lý tài khoản người dùng, vấn đề bảo mật, trình điều khiển thiết bị, cấu hình phần cứng,... Để giảm bớt độ phức tạp, thư mục **/etc** đã được thiết kế để lưu trữ tất cả các thông tin hay các file cấu hình hệ thống.

* Thư mục ***/lib***

Linux có một trung tâm lưu trữ các thư viện hàm và thủ tục, đó là thư mục **/lib.**

* Thư mục ***/lost+found***

Một file được khôi phục sau khi có bất kỳ một vấn đề hoặc gặp một lỗi về ghi đĩa trên hệ thống đều được lưu vào thư mục này.

* Thư mục ***/mnt***

Thư mục **/mnt** là nơi để kết nối các thiết bị (ví dụ đĩa cứng, đĩa mềm...) vào hệ thống file chính nhờ lệnh **mount**. Thông thường các thư mục con của **/mnt** chính là gốc của các hệ thống file được kết nối: **/mnt/floppy**: đĩa mềm, **/mnt/hda1**: vùng đầu tiên của đĩa cứng thứ nhất (**hda**), **/mnt/hdb3**: vùng thứ ba của đĩa cứng thứ 2 (**hdb**)...

* Thư mục ***/tmp***

Thư mục **/tmp** được rất nhiều chương trình trong Linux sử dụng như một nơi để lưu trữ các file tạm thời.

* Thư mục ***/usr***

Thông thường thì thư mục **/usr** là trung tâm lưu trữ tất cả các lệnh hướng đến người dùng (user-related commands). Tuy nhiên, ngày nay thật khó xác định trong thư mục này có những thứ gì, bởi vì hầu hết các file nhị phân cần cho Linux đều được lưu trữ ở đây, trong đó đáng chú ý là thư mục con **/usr/src** bao gồm các thư mục con chứa các chương trình nguồn của nhân Linux.

* Thư mục ***/home***

Thư mục này chứa các thư mục cá nhân của người dùng: mỗi người dùng tương ứng với một thư mục con ở đây, tên người dùng được lấy làm tên của thư mục con.

* Thư mục ***/var***

Thư mục **/var** được sử dụng để lưu trữ các file chứa các thông tin luôn luôn thay đổi, bao gồm bộ đệm in, vùng lưu tạm thời cho việc nhận và gửi thư (mail), các khóa tiến trình,...

* Thư mục ***/boot***

Là thư mục chứa nhân của hệ thống (**Linux-*.*.**), **System.map** (file ánh xạ đến các ***driver*** để nạp các hệ thống file khác), ảnh (image) của hệ thống file dùng cho **initrd** (**ramdisk**), trình điều khiển cho các thiết bị RAID (một thiết bị gồm một mảng các ổ đĩa cứng để tăng tốc độ và độ an toàn khi ghi dữ liệu), các bản sao lưu **boot record** của các phân vùng đĩa khác. Thư mục này cho phép khởi động và nạp lại bất kỳ trình điều khiển nào được yêu cầu để đọc các hệ thống file khác.

* Thư mục ***/proc***

Đây là thư mục dành cho nhân (**kernel**) của hệ điều hành và thực tế đây là một hệ thống file độc lập do nhân khởi tạo.

* Thư mục ***/misc*** và thư mục ***/opt***

Cho phép lưu trữ mọi đối tượng vào hai thư mục này.

* Thư mục ***/sbin***

Thư mục lưu giữ các file hệ thống thường tự động chạy.

3.3.2 Các lệnh cơ bản về thư mục

* Xác định thư mục hiện thời

Cú pháp lệnh: ***pwd***

Lệnh này cho biết hiện người dùng đang ở trong thư mục nào và hiện ra theo dạng một đường dẫn tuyệt đối.

* Xem thông tin về thư mục

Cú pháp lệnh: ***ls [tùy-chọn] [file]...***

Lệnh này đưa ra danh sách các file liên quan đến tham số ***file*** trong lệnh. Trường hợp phổ biến tham số file là một thư mục, tuy nhiên trong một số trường hợp khác, tham số ***file*** xác định nhóm (khi sử dụng các mô tả nhóm *, ? và cặp [và])**;** nếu không có tham số ***file,*** mặc định danh sách các file có trong thư mục hiện thời sẽ được hiển thị.

Các tùy chọn:

-a : liệt kê tất cả các file, bao gồm cả file ẩn.
-l : đưa ra thông tin đầy đủ nhất về các file và thư mục.
-s : chỉ ra kích thước của file, tính theo khối (1 khối = 1204 byte).
-F : xác định kiểu file (/ = thư mục, * = chương trình khả thi).
-m : liệt kê các file được ngăn cách nhau bởi dấu ",".

- -C : đưa ra danh sách các file và thư mục theo dạng cột (hai thư mục gần nhau được xếp vào một cột).
- -1 : hiển thị mỗi file hoặc thư mục trên một dòng.
- -t : sắp xếp các file và thư mục trong danh sách theo thứ tự về thời gian được sửa đổi gần đây nhất.
- -x : đưa ra danh sách các file và thư mục theo dạng cột (hai thư mục gần nhau được xếp trên hai dòng đầu của hai cột kề nhau).
- -r : sắp xếp danh sách hiển thị theo thứ tự ngược lại.
- -R : liệt kê lần lượt các thư mục và nội dung của các thư mục.

Ví dụ, lệnh: ***# ls -l***

sẽ hiển thị danh sách đầy đủ nhất về các file và thư mục có trong thư mục hiện thời.

```
total 108
drwxr-xr-x 12 thu root 4096 Oct 23 2000 LinuxVN.com
drwxr-xr-x 2 root root 4096 Oct 31 2000 bin
drwxr-xr-x 2 root root 4096 Dec 11 16:54 boot
drwxr-xr-x 7 root root 36864 Dec 11 16:54 dev
drwxr-xr-x 43 root root 4096 Dec 11 16:55 etc
drwxr-xr-x 5 root root 4096 Dec 11 16:57 home
drwxr-xr-x 4 root root 4096 Oct 31 2000 lib
drwxr-xr-x 2 root root 16384 Oct 31 2000 lost+found
drwxr-xr-x 2 root root 0       Dec 11 16:54 misc
drwxr-xr-x 5 root root 4096 Oct 31 2000 mnt
drwxr-xr-x 2 root root 4096 Aug 23 12:03 opt
dr-xr-xr-x 56 root root 0       Dec 11 11:54 proc
drwxr-x--- 12 root root 4096 Dec 11 16:55 root
drwxr-xr-x 3 root root 4096 Oct 31 2000 sbin
drwxr-xr-x 3 root root 4096 Oct 31 2000 tftpboot
drwxrwxrwx 8 root root 4096 Dec 11 16:58 tmp
drwxr-xr-x 22 root root 4096 Oct 31 2000 usr
drwxr-xr-x 22 root root 4096 Oct 31 2000 var
```

Dòng đầu tiên "total 108" cho biết tổng số khối (1024 byte) trên đĩa lưu trữ các file trong danh sách (14*4+36+16=108).

Mỗi dòng tiếp theo trình bày thông tin về mỗi file hay thư mục con.

Khi gõ lệnh: ***# ls [is]****

Cho danh sách các file và thư mục con có tên bắt đầu bằng hoặc chữ cái **i** hoặc chữ cái **s** có trong thư mục hiện thời:

info-dir	**initlog.conf**	**inittab**	**services**
shadow	**shadow-**	**shells**	**smb.conf**
sysctl.conf	**syslog.conf**		

* Lệnh tạo thư mục

Cú pháp lệnh: ***mkdir [tùy-chọn] <thư-mục>***

Lệnh này cho phép tạo một thư mục mới nếu thư mục đó chưa thực sự tồn tại. Để tạo một thư mục, cần đặc tả tên và vị trí của nó trên hệ thống file (vị trí mặc định là thư mục hiện thời). Nếu thư mục đã tồn tại, hệ thống sẽ thông báo cho biết.

Các tùy chọn:

- -m, --mode=Mod : thiết lập quyền truy nhập **Mod** như trong lệnh **chmod** nhưng không cho quyền **rwxrwxrwx.**
- -p, --parents : tạo các thư mục cần thiết mà không thông báo lỗi khi nó đã tồn tại.

- --verbose : hiển thị các thông báo cho mỗi thư mục được tạo.
- --help : đưa ra trang trợ giúp và thoát.

Ví dụ: ***# mkdir /home/test***

* Lệnh xóa bỏ thư mục

Như đã biết, lệnh **mkdir** để tạo ra một thư mục mới, và về đối ngẫu thì lệnh **rmdir** được dùng để xóa bỏ một thư mục.

Cú pháp lệnh: ***rmdir [tùy-chọn[<thư-mục>***

Có thể xóa bỏ bất kỳ thư mục nào nếu có quyền đó. Lưu ý rằng, thư mục chỉ bị xóa khi nó "rỗng", tức là không tồn tại file hay thư mục con nào trong đó.

Không có cách gì khôi phục lại các thư mục đã bị xóa, vì thế hãy chú ý trước khi quyết định xóa một thư mục.

Các tùy chọn:

- --ignore-fail-on-non-empty : bỏ qua các lỗi nếu xóa một thư mục không rỗng.
- -p, --parents : xóa bỏ một thư mục, sau đó lần lượt xóa bỏ tiếp các thư mục có trên đường dẫn chứa thư mục vừa xóa. Ví dụ, dòng lệnh **rmdir -p /a/b/c** sẽ tương đương với ba dòng lệnh **rmdir /a/b/c**, **rmdir /a/b**, **rmdir /a** (với điều kiện các thư mục là rỗng).
- --verbose : đưa ra thông báo khi xóa một thư mục.
- --help : hiển thị trang trợ giúp và thoát.

Ví dụ: ***# rmdir -p /test/test1/test2***

* Lệnh đổi tên thư mục

Cú pháp lệnh: ***mv <tên-cũ> <tên-mới>***

Lệnh này cho phép đổi tên một thư mục từ **tên-cũ** thành **tên-mới.**

Nếu sử dụng lệnh **mv** để đổi tên một thư mục với một cái tên đã được đặt cho một file thì lệnh sẽ gặp lỗi.

Nếu tên mới trùng với tên một thư mục đang tồn tại thì nội dung của thư mục được đổi tên sẽ ghi đè lên nội dung của thư mục trùng tên.

3.4. Các lệnh làm việc với file

3.4.1 Các kiểu file có trong Linux

Trong Linux có rất nhiều file khác nhau, nhưng bao giờ cũng tồn tại một số kiểu file cần thiết cho hệ điều hành và người dùng, dưới đây giới thiệu lại một số các kiểu file cơ bản.

- ***File người dùng (user data file):*** là các file tạo ra do hoạt động của người dùng khi kích hoạt các chương trình ứng dụng tương ứng. Ví dụ như các file thuần văn bản, các file cơ sở dữ liệu hay các file bảng tính.
- ***File hệ thống (system data file):*** là các file lưu trữ thông tin của hệ thống như: cấu hình cho khởi động, tài khoản của người dùng, thông tin thiết bị... thường được cất trong các tệp dạng văn bản để người dùng có thể can thiệp, sửa đổi theo ý mình.
- ***File thực hiện (executable file):*** là các file chứa mã lệnh hay chỉ thị cho máy tính thực hiện. File thực hiện lưu trữ dưới dạng mã máy mà ta khó có thể tìm hiểu được ý nghĩa của nó, nhưng tồn tại một số công cụ để "hiểu" được các file đó. Khi dùng trình ứng dụng **mc** (Midnight Commander, chương 8), file thực hiện được bắt đầu bởi dấu (*) và thường có màu xanh lục.
- ***Thư mục hay còn gọi là file bao hàm (directory):*** là file bao hàm các file khác và có cấu tạo hoàn toàn tương tự như file thông thường khác nên có thể gọi là file. Trong **mc**, file bao hàm thường có màu trắng và bắt đầu bằng dấu ngã (~) hoặc dấu chia (/). Ví dụ: **/, /home, /bin, /usr, /usr/man, /dev...**

- ***File thiết bị (device file):*** là file mô tả thiết bị, dùng như là định danh để chỉ ra thiết bị cần thao tác. Theo quy ước, file thiết bị được lưu trữ trong thư mục **/dev**. Các file thiết bị hay gặp trong thư mục này là **tty** (teletype - thiết bị truyền thông), **ttyS** (teletype serial - thiết bị truyền thông nối tiếp), **fd0**, **fd1**,... (floppy disk- thiết bị ổ đĩa mềm), **hda1**, **hda2**,... **hdb1**, **hdb2**,... (hardisk - thiết bị ổ cứng theo chuẩn IDE; a, b,... đánh số ổ đĩa vật lý; 1, 2, 3... đánh số ổ logic). Trong **mc**, file thiết bị có màu tím và bắt đầu bằng dấu cộng (+).
- ***File liên kết (linked file):*** là những file chứa tham chiếu đến các file khác trong hệ thống tệp tin của Linux. Tham chiếu này cho phép người dùng tìm nhanh tới file thay vì tới vị trí nguyên thủy của nó. Hơn nữa, người ta có thể gắn vào đó các thông tin phụ trợ làm cho file này có tính năng trội hơn so với tính năng nguyên thủy của nó. Ta thấy loại file này giống như khái niệm ***shortcut*** trong MS-Windows98.

Không giống một số hệ điều hành khác (như MS-DOS chẳng hạn), Linux quản lý thời gian của tệp tin qua các thông số thời gian truy nhập (accesed time), thời gian kiến tạo (created time) và thời gian sửa đổi (modified time).

3.4.2. Các lệnh tạo file

Trong Linux có rất nhiều cách để tạo file, sau đây là các cách hay được dùng.

* Tạo file với lệnh ***touch***

Lệnh **touch** có nhiều chức năng, trong đó một chức năng là giúp tạo file mới trên hệ thống: **touch** rất hữu ích cho việc tổ chức một tập hợp các file mới.

Cú pháp lệnh: ***touch <file>***

Thực chất lệnh này có tác dụng dùng để cập nhật thời gian truy nhập và sửa chữa lần cuối của một file. Vì lý do này, các file được tạo bằng lệnh **touch** đều được sắp xếp theo thời gian sửa đổi. Nếu sử dụng lệnh **touch** đối với một file chưa tồn tại, chương trình sẽ tạo ra file đó. Sử dụng bất kỳ trình soạn thảo nào để soạn thảo file mới.

* Tạo file bằng cách đổi hướng đầu ra của lệnh (>)

Cách này rất hữu ích nếu muốn lưu kết quả của một lệnh đã thực hiện.

Để gửi kết quả của một lệnh vào một file, dùng dấu ">" theo nghĩa chuyển hướng lối ra chuẩn.

Ví dụ: ***# ls -l /bin > /home/thu/lenhls***

Linux tự động tạo nếu file **lenhls** chưa có, trong trường hợp ngược lại, nội dung file cũ sẽ bị thế chỗ bởi kết quả của lệnh.

Nếu muốn bổ sung kết quả vào cuối file thay vì thay thế nội dung file, ử dụng dấu ">>".

* Tạo file với lệnh ***cat***

Lệnh **cat** tuy đơn giản nhưng rất hữu dụng trong Linux. Chúng ta có thể sử dụng lệnh này để lấy thông tin từ đầu vào (bàn phím...) rồi kết xuất ra file hoặc các nguồn khác (màn hình...), hay để xem nội dung của một file... Phần này trình bày tác dụng của lệnh **cat** đối với việc tạo file.

Cú pháp lệnh: ***cat > <file>***

Theo ngầm định, lệnh này cho phép lấy thông tin đầu vào từ bàn phím rồi xuất ra màn hình. Soạn thảo nội dung của một file bằng lệnh cat tức là đã đổi hướng đầu ra của lệnh từ màn hình vào một file. Người dùng gõ nội dung của file ngay tại dấu nhắc màn hình và gõ **CTRL+d** để kết thúc việc soạn thảo. Nhược điểm của cách tạo file này là nó không cho phép sửa lỗi, ví dụ nếu muốn sửa một lỗi chính tả trên một dòng, chỉ có cách là xóa đến vị trí của lỗi và gõ lại nội dung vừa bị xóa.

Ví dụ: **# cat > /home/vd/newfile**

Sau khi soạn thảo xong, gõ **Enter** và **CTRL+d** để trở về dấu nhắc lệnh, nếu không gõ **Enter** thì phải gõ **CTRL+d** hai lần.

Có thể sử dụng luôn lệnh **cat** để xem nội dung của file vừa soạn thảo:

3.4.3 Các lệnh thao tác trên file

* Sao chép file

Lệnh **cp** có hai dạng như sau:

cp [tùy-chọn] <file-nguồn>... <file-đích>

cp [tùy-chọn] --target-directory=<thư-mục> <file-nguồn>...

Lệnh này cho phép sao **file-nguồn** thành **file-đích** hoặc sao chép từ nhiều file-nguồn vào một thư mục đích (tham số <file-đích> hay <thư-mục>). Dạng thứ hai là một cách viết khác đổi thứ tự hai tham số vị trí.

Các tùy chọn:

-a, --archive : giống như **-dpR** (tổ hợp ba tham số **-d**, **-p**, **-R**, như dưới đây).

-b, --backup[=CONTROL] : tạo file lưu cho mỗi file đích nếu như nó đang tồn tại.

-d, --no-dereference : duy trì các liên kết.

-f, --force : ghi đè file đích đang tồn tại mà không nhắc nhở.

-i, --interactive : có thông báo nhắc nhở trước khi ghi đè.

-l, --link : chỉ tạo liên kết giữa file-đích từ file-nguồn mà không sao chép.

-p, --preserve : duy trì các thuộc tính của file-nguồn sang file-đích.

-r : cho phép sao chép một cách đệ quy file thông thường.

-R : cho phép sao chép một cách đệ quy thư mục.

-s, --symbolic-link : tạo liên kết tượng trưng thay cho việc sao chép các file.

-S, --suffix=<hậu-tố> : bỏ qua các hậu tố thông thường (hoặc được chỉ ra).

-u, --update : chỉ sao chép khi file nguồn mới hơn file đích hoặc khi file đích chưa có.

-v, --verbose : đưa ra thông báo về tiến trình sao chép.

--help : hiển thị trang trợ giúp và thoát.

File đích được tạo ra có cùng kích thước và các quyền truy nhập như file nguồn, tuy nhiên file đích có thời gian tạo lập là thời điểm thực hiện lệnh nên các thuộc tính thời gian sẽ khác.

Nếu ở vị trí đích, mô tả đầy đủ tên file đích thì nội dung file nguồn sẽ được sao chép sang file đích. Trong trường hợp chỉ đưa ra vị trí file đích được đặt trong thư mục nào thì tên của file nguồn sẽ là tên của file đích.

Nếu sử dụng lệnh này để sao một thư mục, sẽ có một thông báo được đưa ra cho biết nguồn là một thư mục và vì vậy không thể dùng lệnh **cp** để sao chép.

Ví dụ sao chép nhiều file cùng một lúc vào một thư mục.

 # cp vd vd1 newdir

Lưu ý: Đối với nhiều lệnh làm việc với file, khi gõ lệnh có thể sử dụng ký hiệu mô tả nhóm để xác định một nhóm file làm cho tăng hiệu lực của các lệnh đó.

* Đổi tên file

Cú pháp lệnh: ***mv <tên-cũ> <tên-mới>***

Lệnh này cho phép đổi tên file từ **tên cũ** thành **tên mới**.

Ví dụ: ***# mv vd newfile***

Trong trường hợp file **newfile** đã tồn tại, nội dung của file **vd** sẽ ghi đè lên nội dung của file **newfile**.

* Xóa file

Lệnh **rm** là lệnh rất "nguy hiểm" vì trong Linux không có lệnh khôi phục lại những gì đã xóa, vì thế hãy cẩn trọng khi sử dụng lệnh này.

Cú pháp lệnh: ***rm [tùy-chọn] <file>...***

Lệnh **rm** cho phép xóa bỏ một file hoặc nhiều file.

Các tùy chọn:

- -d, --directory : loại bỏ liên kết của thư mục, kể cả thư mục không rỗng. Chỉ có siêu người dùng mới được phép dùng tùy chọn này.
- -f, --force : bỏ qua các file (xác định qua tham số **file**) không tồn tại mà không cần nhắc nhở.
- -i, --interactive : nhắc nhở trước khi xóa bỏ một file.
- -r, -R, --recursive : xóa bỏ nội dung của thư mục một cách đệ quy.
- -v, --verbose : đưa ra các thông báo về tiến trình xóa file.
- --help : hiển thị trang trợ giúp và thoát.

Lệnh **rm** cho phép xóa nhiều file cùng một lúc bằng cách chỉ ra tên của các file cần xóa trong dòng lệnh (hoặc dùng ký kiệu mô tả nhóm).

* Lệnh đếm từ và dòng trong file

Cú pháp lệnh: ***wc [tùy-chọn] [file]...***

Lệnh hiện ra số lượng dòng, số lượng từ, số lượng ký tự có trong mỗi file, và một dòng tính tổng nếu có nhiều hơn một file được chỉ ra. Nếu không có tùy chọn nào thì mặc định đưa ra cả số dòng, số từ và số ký tự. Ngầm định khi không có tên file trong lệnh thì sẽ đọc và đếm trên thiết bị vào chuẩn.

Các tuỳ chọn:

- -c, --byte, --chars : đưa ra số ký tự trong file.
- -l, --lines : đưa ra số dòng trong file.
- -L, --max-line-length : đưa ra chiều dài của dòng dài nhất trong file.
- -w, --words : đưa ra số từ trong file.
- --help : hiển thị trang trợ giúp và thoát.

* Lệnh loại bỏ những dòng không quan trọng

Trong một số trường hợp khi xem nội dung một file, chúng ta thấy có một số các thông tin bị trùng lặp, ví dụ các dòng trống hoặc các dòng chứa nội dung giống nhau. Để đồng thời làm gọn và thu nhỏ kích thước của file, có thể sử dụng lệnh **uniq** để liệt kê ra nội dung file sau khi đã loại bỏ các dòng trùng lặp.

Cú pháp lệnh: ***uniq [tùy-chọn] [input] [output]***

Lệnh **uniq** sẽ loại bỏ các dòng trùng lặp kề nhau từ **input** (thiết bị vào chuẩn) và chỉ giữ lại một dòng duy nhất trong số các dòng trùng lặp rồi đưa ra **output** (thiết bị ra chuẩn).

Các tuỳ chọn:

- -c, --count : đếm và hiển thị số lần xuất hiện của các dòng trong file.
- -d : hiển thị lên màn hình dòng bị trùng lặp.
- -u : hiển thị nội dung file sau khi xóa bỏ toàn bộ các dòng bị trùng lặp không giữ lại một dòng nào.
- -i : hiển thị nội dung file sau khi xóa bỏ các dòng trùng lặp và chỉ giữ lại duy nhất một dòng có nội dung bị trùng lặp.
- -D : hiển thị tất cả các dòng trùng lặp trên màn hình.

Nếu sử dụng lệnh **uniq** trên một file không có các dòng trùng lặp thì lệnh không có tác dụng.

* Sắp xếp nội dung file

sort là lệnh đọc các thông tin và sắp xếp chúng theo thứ tự trong bảng chữ cái hoặc theo thứ tự được quy định theo các tùy chọn của lệnh.

Cú pháp lệnh: ***sort [tùy-chọn] [file]...***

Hiển thị nội dung sau khi sắp xếp của một hoặc nhiều file ra thiết bị ra chuẩn là tác dụng của lệnh **sort**. Ngầm định sắp xếp theo thứ tự từ điển của các dòng có trong các file (từng chữ cái theo bảng chữ hệ thống (chẳng hạn ASCII) và kể từ vị trí đầu tiên trong các dòng).

Các tùy chọn:

- **+<số1> [-<số2>]** : Hai giá trị **số1** và số**2** xác định "khóa" sắp xếp của các dòng, thực chất lấy xâu con từ vị trí **số1** tới vị trí **số2** của các dòng để so sánh lấy thứ tự sắp xếp các dòng. Nếu **số2** không có thì coi là hết các dòng; nếu **số2** nhỏ hơn **số1** thì bỏ qua lựa chọn này. Chú ý, nếu có **số2** thì phải cách **số1** ít nhất một dấu cách.
- **-b** : bỏ qua các dấu cách đứng trước trong phạm vi sắp xếp.
- **-c** : kiểm tra nếu file đã sắp xếp thì thôi không sắp xếp nữa.
- **-d** : xem như chỉ có các ký tự [a-zA-Z0-9] trong khóa sắp xếp, các dòng có các ký tự đặc biệt
- (dấu cách, ?...) được đưa lên đầu.
- **-f** : sắp xếp không phân biệt chữ hoa chữ thường.
- **-n** : sắp xếp theo kích thước của file.
- **-r** : chuyển đổi thứ tự sắp xếp hiện thời. Ví dụ, muốn sắp xếp file **vdsort**

3.4.4 Các lệnh thao tác theo nội dung file

* Xác định kiểu file

Cú pháp lệnh: ***file [tùy-chọn] [-f file] [-m <file-ảnh>...] <file>...***

Lệnh file cho phép xác định và in ra kiểu thông tin chứa trong file. Lệnh file sẽ lần lượt kiểm tra từ kiểu file hệ thống, kiểu file magic (ví dụ file mô tả thiết bị) rồi đến kiểu file văn bản thông thường. Nếu file được kiểm tra thỏa mãn một trong ba kiểu file trên thì kiểu file sẽ được in ra theo các dạng cơ bản sau:

- text: dạng file văn bản thông thường, chỉ chứa các mã ký tự ASCII.
- executable: dạng file nhị phân khả thi.
- data: thường là dạng file chứa mã nhị phân và không thể in ra được.

Các tuỳ chọn:

- -b : cho phép chỉ đưa ra kiểu file mà không đưa kèm theo tên file.
- -f tên-file : cho phép hiển thị kiểu của các file có tên trùng với nội dung trên mỗi dòng trong file tên-file. Để kiểm tra trên thiết bị vào chuẩn, sử dụng dấu "-".
- -z : xem kiểu của file nén.

* Xem nội dung file

Ở phần trước, chúng ta đã có dịp làm quen với lệnh **cat** thông qua tác dụng tạo file của lệnh. Phần này giới thiệu tác dụng chủ yếu của lệnh **cat**: đó là tác dụng xem nội dung của một file.

Cú pháp lệnh: ***cat [tùy-chọn] <tên file>***

Các tùy chọn:

- -A, --show-all : giống như tùy chọn **-vET**.
- -b, --number-nonblank : hiển thị thêm số thứ tự trên mỗi dòng (bỏ qua dòng trống).
- -e : giống như tùy chọn **-vE.**
- -E, --show-ends : hiển thị dấu "$" tại cuối mỗi dòng.
- -n, --number : hiển thị số thứ tự của mỗi dòng (kể cả dòng trống).

- -s : nếu trong nội dung file có nhiều dòng trống thì sẽ loại bỏ bớt để chỉ hiển thị một dòng trống.
- -t : giống như **-vT.**
- -T, --show-tabs : hiển thị dấu TAB dưới dạng ^I.
- -v, --show-nonprinting : hiển thị các ký tự không in ra được ngoại trừ LFD và TAB.
- --help : hiển thị trang trợ giúp và thoát.

* Xem nội dung các file lớn

Lệnh **cat** cho phép xem nội dung của một file, nhưng nếu file quá lớn, nội dung file sẽ trôi trên màn hình và chỉ có thể nhìn thấy phần cuối của file. Linux có một lệnh cho phép có thể xem nội dung của một file lớn, đó là lệnh **more.**

Cú pháp lệnh: ***more [-dlfpcsu] [-số] [+/xâumẫu] [+dòng-số] [file...]***

Lệnh **more** hiển thị nội dung của file theo từng trang màn hình.

Các lựa chọn:

- **-s**ố **:** xác định số dòng nội dung của file được hiển thị (**s**ố).
- -d : trên màn hình sẽ hiển thị các thông báo giúp người dùng cách sử dụng đối với lệnh more,
- ví như [Press space to continue, "q" to quit .**]**, hay hiển thị [Press "h" for instructions .**]** thay thế cho tiếng chuông cảnh báo khi bấm sai một phím.
- -l **: more** thường xem ^L là một ký tự đặc biệt, nếu không có tùy chọn này, lệnh sẽ dừng tại dòng đầu tiên có chứa ^L và hiển thị % nội dung đã xem được (^L không bị mất), nhấn phím **space** (hoặc **enter**) để tiếp tục. Nếu có tùy chọn **-l**, nội dung của file sẽ được hiển thị như bình thường nhưng ở một khuôn dạng khác, tức là dấu ^L sẽ mất và trước dòng có chứa ^L sẽ có thêm một dòng trống.
- -p : không cuộn màn hình, thay vào đó là xóa những gì có trên màn hình và hiển thị tiếp nội dung file.
- -c : không cuộn màn hình, thay vào đó xóa màn hình và hiển thị nội dung file bắt đầu từ đỉnh màn hình.
- -s : xóa bớt các dòng trống liền nhau trong nội dung file chỉ giữ lại một dòng.
- -u : bỏ qua dấu gạch chân.
- **+/xâumẫu** : tùy chọn **+/xâumẫu** chỉ ra một chuỗi sẽ được tìm kiếm trước khi hiển thị mỗi file.
- **+dòng-s**ố : bắt đầu hiển thị từ dòng thứ **dòng-số.**

* Xem qua nội dung file

Các đoạn trước cho biết cách thức xem nội dung của một file nhờ lệnh **cat** hay **more.** Trong Linux cũng có các lệnh khác cho nhiều cách thức để xem nội dung của một file. Trước hết, chúng ta hãy làm quen với lệnh **head.**

Cú pháp lệnh: ***head [tùy-chọn] [file]...***

Lệnh này mặc định sẽ đưa ra màn hình 10 dòng đầu tiên của mỗi file. Nếu có nhiều hơn một file, thì lần lượt tên của file và 10 dòng nội dung đầu tiên sẽ được hiển thị. Nếu không có tham số file, hoặc file là dấu "-", thì ngầm định sẽ đọc từ thiết bị vào chuẩn.

Các tuỳ chọn:

- **-c, --bytes=cỡ** : hiển thị cỡ (số nguyên) ký tự đầu tiên trong nội dung file (cỡ có thể nhận giá trị là **b** cho 512, **k** cho 1K, **m** cho 1 Meg)
- **-n, --lines=n** : hiển thị **n** (số nguyên) dòng thay cho 10 dòng ngầm định.
- **-q, --quiet, --silent** : không đưa ra tên file ở dòng đầu.
- **-v, --verbose** : luôn đưa ra tên file ở dòng đầu.
- **--help** : hiển thị trang trợ giúp và thoát.

* Xem qua nội dung file

Cú pháp lệnh: ***tail [tùy-chọn] [file]...***

Lệnh **tail** ngầm định đưa ra màn hình 10 dòng cuối trong nội dung của các file. Nếu có nhiều hơn một file, thì lần lượt tên của file và 10 dòng cuối sẽ được hiển thị. Nếu không có tham số file**,** hoặc file là dấu "-" thì ngầm định sẽ đọc từ thiết bị vào chuẩn.

Các tùy chọn:

- **--retry** : cố gắng mở một file khó truy nhập khi bắt đầu thực hiện lệnh **tail.**
- **-c, --bytes=n** : hiển thị **n** (số) ký tự sau cùng.
- **-f, --follow[={name | descritptor}]** : sau khi hiện nội dung file sẽ hiện thông tin về file: **-f**, **-- follow**, và **--follow=descriptor** là như nhau.
- **-n, --lines=n** : hiển thị **n** (số) dòng cuối cùng của file thay cho 10 dòng ngầm định.
- **--max-unchanged-stats=n** : hiển thị tài liệu về file (ngầm định **n** là 5).
- **--max-consecutive-size-changes=n** : hiển thị tài liệu về file (ngầm định **n** là 200).
- **--pid=PID** : kết hợp với tùy chọn **-f**, chấm dứt sau khi tiến trình có chỉ số = **PID** lỗi.
- **-q, --quiet, --silent** : không đưa ra tên file ở dòng đầu trong nội dung được hiển thị.
- **-s, --sleep-interval=k** : kết hợp với tùy chọn -f, dừng **k** giây giữa các hoạt động.
- **-v, --verbose** : luôn hiển thị tên của file.
- **--help** : hiển thị trang trợ giúp và thoát.

* Thêm số thứ tự của các dòng trong file

Cú pháp lệnh: ***nl [tùy-chọn] <file>***

Lệnh này sẽ đưa nội dung file ra thiết bị ra chuẩn, với số thứ tự của dòng được thêm vào. Nếu không có file (**tên file)**, hoặc khi file là dấu "**-**", thì đọc nội dung từ thiết bị vào chuẩn.

Các tuỳ chọn:

- -b, --body-numbering=STYLE : sử dụng kiểu STYLE cho việc đánh thứ tự các dòng trong nội dung file. Có các kiểu STYLE sau:
  - **a** : đánh số tất cả các dòng kể cả dòng trống;
  - **t** : chỉ đánh số các dòng không trống;
  - **n** : không đánh số dòng.
- -d, --section-delimiter=CC : sử dụng CC để đánh số trang logic (CC là hai ký tự xác định phạm vi cho việc phân trang logic).
- -f, --footer-numbering=STYLE : sử dụng kiểu STYLE để đánh số các dòng trong nội dung file (một câu có thể có hai dòng...).
- -h, --header-numbering=STYLE : sử dụng kiểu STYLE để đánh số các dòng trong nội dung file.
- **-i, --page-increment=s**ố : đánh số thứ tự của dòng theo cấp số cộng có công sai là **s**ố.
- **-l, --join-blank-lines=s**ố :nhóm **s**ố dòng trống vào thành một dòng trống.
- **-n, --number-format=khuôn** : chèn số dòng theo **khuôn** (khuôn: **ln** - căn trái, không có số 0 ở đầu; **rn** - căn phải, không có số 0 ở đầu; **rz** - căn phải và có số 0 ở đầu)
- -p, --no-renumber : không thiết lập lại số dòng tại mỗi trang logic.
- **-s, --number-separator=xâu** : thêm chuỗi **xâu** vào sau số thứ tự của dòng.
- **-v, --first-page=s**ố : số dòng đầu tiên trên mỗi trang logic.
- **-w, --number-width=s**ố : hiển thị số thứ tự của dòng trên cột thứ **s**ố.

- **--help** : hiển thị trang trợ giúp và thoát.

* Tìm sự khác nhau giữa hai file

Cú pháp lệnh: ***diff [tuỳ-chọn] <file1> <file2>***

Trong trường hợp đơn giản, lệnh **diff** sẽ so sánh nội dung của hai file. Nếu file1 là một thư mục còn file2 là một file bình thường, **diff** sẽ so sánh file có tên trùng với file2 trong thư mục file1 với file2.

Nếu cả file1 và file2 đều là thư mục, **diff** sẽ thực hiện sự so sánh lần lượt các file trong cả hai thư mục theo thứ tự từ a-z (sự so sánh này sẽ không đệ qui nếu tuỳ chọn **-r** hoặc **--recursive** không được đưa ra). Tất nhiên so sánh giữa hai thư mục không thể chính xác như khi so sánh hai file.

Các tuỳ chọn:

- **-a**: xem tất cả các file ở dạng văn bản và so sánh theo từng dòng.
- **-b**: bỏ qua sự thay đổi về số lượng của ký tự trống.
- **-B**: bỏ qua mọi sự thay đổi mà chỉ chèn hoặc xoá các dòng trống.
- **--brief**: chỉ thông báo khi có sự khác nhau mà không đưa ra chi tiết nội dung khác nhau.
- **-d**: tìm ra sự khác biệt nhỏ (tuỳ chọn này có thể làm chậm tốc độ làm việc của lệnh **diff**).
- **--exclude-from=file**: khi so sánh thư mục, bỏ qua các file và các thư mục con có tên phù hợp với mẫu có trong **file.**
- **-i**: so sánh không biệt chữ hoa chữ thường.
- **-r**: thực hiện so sánh đệ qui trên thư mục.
- **-s**: thông báo khi hai file là giống nhau.
- **-y**: hiển thị hai file cạnh nhau để dễ phân biệt sự khác nhau.

3.4.5 Các lệnh tìm file

* Tìm theo nội dung file

Lệnh **grep** cũng như lệnh **ls** là hai lệnh rất quan trọng trong Linux. Lệnh này có hai tác dụng cơ bản như sau:

- Lọc đầu ra của một lệnh:

  <lệnh> | grep <mẫu lọc>

- Tìm dòng chứa mẫu đã định trong file được chỉ ra:

  grep [tùy-chọn] <mẫu-lọc> [file]

Lệnh **grep** hiển thị tất cả các dòng có chứa **mẫu-lọc** trong file được chỉ ra (hoặc từ thiết bị vào chuẩn nếu không có file hoặc file có dạng là dấu "-")

Các tùy chọn:

- **-G, --basic-regexp** : xem mẫu lọc như một biểu thức thông thường. Điều này là ngầm định.
- **-E, --extended-regexp** : xem mẫu lọc như một biểu thức mở rộng.
- **-F, --fixed-strings** : xem mẫu như một danh sách các xâu cố định, được phân ra bởi các dòng mới. Ngoài lệnh grep còn có hai lệnh là egrep và fgrep. egrep tương tự như lệnh grep -E, fgrep tương tự với lệnh grep -F .

Lệnh **grep** còn có các tùy chọn sau:

- **-A NUM, --after-context=NUM** : đưa ra NUM dòng nội dung tiếp theo sau dòng có chứa mẫu.
- **-B NUM, --before-context=NUM** : đưa ra NUM dòng nội dung trước dòng có chứa mẫu.

- **-C [NUM], --context[=NUM]** : hiển thị NUM dòng (mặc định là 2 dòng) nội dung.
- **-NUM** : giống **--context=NUM** đưa ra các dòng nội dung trước và sau dòng có chứa mẫu. Tuy nhiên, **grep** sẽ không đưa ra dòng nào nhiều hơn một lần.
- **-b, --byte-offset** : hiển thị địa chỉ tương đối trong file đầu vào trước mỗi dòng được đưa ra
- **-c, --count** : đếm số dòng tương ứng chứa mẫu trong file đầu vào thay cho việc hiển thị các dòng chứa mẫu.
- **-d ACTION, --directories=ACTION** : nếu đầu vào là một thư mục, sử dụng ACTION để xử lý nó. Mặc định, ACTION là **read**, tức là sẽ đọc nội dung thư mục như một file thông thường. Nếu ACTION là **skip**, thư mục sẽ bị bỏ qua. Nếu ACTION là **recurse**, **grep** sẽ đọc nội dung của tất cả các file bên trong thư mục (đệ quy); tùy chọn này tương đương với tùy chọn **-r.**
- **-f file, --file=file** : lấy các mẫu từ **file**, một mẫu trên một dòng. File trống chứa đựng các mẫu rỗng, và các dòng đưa ra cũng là các dòng trống.
- **-H, --with-file** : đưa ra tên file trên mỗi dòng chứa mẫu tương ứng.
- **-h, --no-filename** : không hiển thị tên file kèm theo dòng chứa mẫu trong trường hợp tìm nhiều file.
- **-i** : hiển thị các dòng chứa mẫu không phân biệt chữ hoa chữ thường.
- **-l** : đưa ra tên các file trùng với mẫu lọc.
- **-n, --line-number** : thêm số thứ tự của dòng chứa mẫu trong file.
- **-r, --recursive** : đọc tất cả các file có trong thư mục (đệ quy).
- **-s, --no-messages** : bỏ qua các thông báo lỗi file không đọc được hoặc không tồn tại.
- **-v, --invert-match** : hiển thị các dòng không chứa mẫu.
- **-w, --word-regexp** : chỉ hiển thị những dòng có chứa mẫu lọc là một từ trọn vẹn.
- **-x, --line-regexp** : chỉ hiển thị những dòng mà nội dung trùng hoàn toàn với mẫu lọc.

* Tìm theo các đặc tính của file

Các đoạn trên đây đã giới thiệu cách thức tìm file theo nội dung với các lệnh **grep**, **egrep** và **fgrep**. Linux còn cho phép người dùng sử dụng một cách thức khác đầy năng lực, đó là sử dụng lệnh **find**, lệnh tìm file theo các thuộc tính của file. Lệnh này có một sự khác biệt so với các lệnh khác, đó là các tùy chọn của lệnh là một từ chứ không phải một ký tự. Điều kiện cần đối với lệnh này là chỉ ra được điểm bắt đầu của việc tìm kiếm trong hệ thống file và những quy tắc cần tuân theo của việc tìm kiếm.

Cú pháp lệnh: ***find [đường-dẫn] [biểu-thức]***

Lệnh **find** thực hiện việc tìm kiếm file trên cây thư mục theo **biểu thức** được đưa ra. Mặc định đường dẫn là thư mục hiện thời, **biểu thức** là **-print**.

Biểu thức có thể có những dạng sau:

- Các toán tử: **(EXPR); ! EXPR hoặc -not EXPR; EXPR1 -a EXPR2 hoặc EXPR1 –and EXPR2; EXPR1 -o EXPR2 hoặc EXPR1 -or EXPR2; và EXPR1, EXPR2**
- Các tùy chọn lệnh: tất cả các tùy chọn này luôn trả về giá trị **true** và được đặt ở đầu biểu thức
  - **-daystart** : đo thời gian (-amin, -atime, -cmin, -ctime, -mmin, -mtime).
  - **-depth** : thực hiện tìm kiếm từ nội dung bên trong thư mục trước (mặc định việc tìm kiếm được thực hiện bắt đầu tại gốc cây thư mục có chứa file cần tìm).

- **-follow** : (tùy chọn này chỉ áp dụng cho thư mục) nếu có tùy chọn này thì các liên kết tượng trưng có trong một thư mục liên kết sẽ được chỉ ra.
- **-help**, **--help** : hiển thị kết quả của lệnh find và thoát.

Các **test**

- **-amin n** : tìm file được truy nhập n phút trước.
- **-atime n** : tìm file được truy nhập n*24 giờ trước.
- **-cmin n** : trạng thái của file được thay đổi n phút trước đây.
- **-ctime n** : trạng thái của file được thay đổi n*24 giờ trước đây.
- **-empty** : file rỗng và hoặc là thư mục hoặc là file bình thường.
- **-fstype kiểu** : file thuộc hệ thống file với kiểu.
- **-gid n** : chỉ số nhóm của file là n.
- **-group nhóm** : file thuộc quyền sở hữu của nhóm.
- **-links n** : file có n liên kết.
- **-mmin n** : dữ liệu của file được sửa lần cuối vào n phút trước đây.
- **-mtime n** : dữ liệu của file được sửa vào n*24 giờ trước đây.
- **-name mẫu** : tìm kiếm file có tên là mẫu. Trong tên file có thể chứa cả các ký tự đại diện như
- dấu "*", "?"...
- **-type kiểu** : tìm các file thuộc kiểu với kiểu nhận các giá trị:
  - b: đặc biệt theo khối
  - c: đặc biệt theo ký tự
  - d: thư mục
  - p: pipe
  - f: file bình thường
  - l: liên kết tượng trưng
  - s: socket
- **-uid n**: chỉ số người sở hữu file là n.
- **-user tên-ngư**ờ**i**: file được sở hữu bởi người dùng tên-người.

Các hành động

- **-exec lệnh :** tùy chọn này cho phép kết hợp lệnh **find** với một lệnh khác để có được thông tin nhiều hơn về các thư mục có chứa file cần tìm. Tùy chọn **exec** phải sử dụng dấu {} - nó sẽ thay
- thế cho tên file tương ứng, và dấu '\' tại cuối dòng lệnh, (phải có khoảng trống giữa {} và '\'). Kết thúc lệnh là dấu '**;**'
- **-fprint file :** hiển thị đầy đủ tên file vào trong **file**. Nếu **file** không tồn tại thì sẽ được tạo ra, nếu
- đã tồn tại thì sẽ bị thay thế nội dung.
- **-print :** hiển thị đầy đủ tên file trên thiết bị ra chuẩn.
- **-ls :** hiển thị file hiện thời theo khuôn dạng: liệt kê danh sách đầy đủ kèm cả số thư mục, chỉ số của mỗi file, với kích thước file được tính theo khối (block).

3.5 Nén và sao lưu các file

3.5.1 Sao lưu các file (lệnh tar)

Một vấn đề rất quan trọng trong việc sao lưu đó là lựa chọn phương tiện sao lưu. cần phải quan tâm đến giá cả, độ tin cậy, tốc độ, ích lợi cũng như tính khả dụng của các phương tiện sao lưu.

Có rất nhiều các công cụ có thể được sử dụng để sao lưu. Các công cụ truyền thống là **tar**, **cpio** và **dump** (công cụ trong tài liệu này là **tar**). Ngoài ra còn rất nhiều các công cụ khác có thể lựa chọn tùy theo phương tiện sao lưu có trong hệ thống.

Có hai kiểu sao lưu là sao lưu theo kiểu toàn bộ (***full backup***) và sao lưu theo kiểu tăng dần (***incremental backup***). Sao lưu toàn bộ thực hiện việc sao mọi thứ trên hệ thống file, bao gồm tất cả các file. Sao lưu tăng dần chỉ sao lưu những file được thay đổi hoặc được tạo ra kể từ đợt sao lưu cuối cùng.

Cú pháp lệnh: ***tar [tùy-chọn] [<file>,...] [<thư-mục>,...]***

Lệnh (chương trình) **tar** được thiết kế để tạo lập một file lưu trữ duy nhất. Với **tar**, có thể kết hợp nhiều file thành một file duy nhất có kích thước lớn hơn, điều này sẽ giúp cho việc di chuyển file hoặc sao lưu băng từ trở nên dễ dàng hơn nhiều.

Các tùy chọn:

- **-c, --create** : tạo file lưu trữ mới.
- **-d, --diff, --compare** : tìm ra sự khác nhau giữa file lưu trữ và file hệ thống được lưu trữ.
- **--delete :** xóa từ file lưu trữ (không sử dụng cho băng từ).
- **-r, --append** : chèn thêm file vào cuối file lưu trữ.
- **-t, --list** : liệt kê nội dung của một file lưu trữ.
- **-u, --update** : chỉ thêm vào file lưu trữ các file mới hơn các file đã có.
- **-x, --extract, --get :** tách các file ra khỏi file lưu trữ.
- **-C, --directory tên-thư-mục** : thay đổi đến thư mục có tên là **tên-thư-mục.**
- **--checkpoint** : đưa ra tên thư mục khi đọc file lưu trữ.
- **-f, --file [HOSTNAME:]file** : tùy chọn này xác định tên file lưu trữ hoặc thiết bị lưu trữ là **file** (nếu không có tùy chọn này, mặc định nơi lưu trữ là **/dev/rmt0**).
- **-h, --dereference :** không hiện các file liên kết mà hiện các file mà chúng trỏ tới.
- **-k, --keep-old-files** : giữ nguyên các file lưu trữ đang tồn tại mà không ghi đè file lưu trữ mới lên chúng.
- **-K, --starting-file file** : bắt đầu tại **file** trong file lưu trữ.
- **-l, --one-file-system** : tạo file lưu trữ trên hệ thống file cục bộ.
- **-M, --multi-volume** : tùy chọn này được sử dụng khi dung lượng của file cần sao lưu là lớn và không chứa hết trong một đơn vị lưu trữ vật lý.
- **-N, --after-date DATE, --newer DATE** : chỉ lưu trữ các file mới hơn các file được lưu trữ trong ngày DATE.
- **--remove-files** : xóa file gốc sau khi đã sao lưu chúng vào trong file lưu trữ.
- **--totals** : đưa ra tổng số byte được tạo bởi tùy chọn **--create**.
- **-v, --verbose** : hiển thị danh sách các file đã được xử lý.

3.5.2 Nén dữ liệu

Việc sao lưu rất có ích nhưng đồng thời nó cũng chiếm rất nhiều không gian cần thiết để sao lưu. Để giảm không gian lưu trữ cần thiết, có thể thực hiện việc nén dữ liệu trước khi sao lưu, sau đó thực hiện việc giải nén để nhận lại nội dung trước khi nén.

Trong Linux có khá nhiều cách để nén dữ liệu, tài liệu này giới thiệu hai phương cách phổ biến là **gzip** và **compress**.

* Nén, giải nén và xem nội dung các file với lệnh *gzip, gunzip* và *zcat*

Cú pháp các lệnh:

gzip [tùy-chọn] [-S suffix] [< file>]

gunzip [tùy-chọn] [-S suffix] [<file>]

zcat [tùy-chọn] [<file>]

Lệnh **gzip** sẽ làm giảm kích thước của file và khi sử dụng lệnh này, file gốc sẽ bị thay thế bởi file nén với phần mở rộng là .gz, các thông tin khác liên quan đến file không thay đổi. Nếu không có tên file nào được chỉ ra thì thông tin từ thiết bị vào chuẩn sẽ được nén và gửi ra thiết bị ra chuẩn. Trong một vài trường hợp, lệnh này sẽ bỏ qua liên kết tượng trưng.

Nếu tên file nén quá dài so với tên file gốc, **gzip** sẽ cắt bỏ bớt. **gzip** sẽ chỉ cắt phần tên file vượt quá 3 ký tự (các phần được ngăn cách với nhau bởi dấu chấm). Nếu tên file gồm nhiều phần nhỏ thì phần dài nhất sẽ bị cắt bỏ. Ví dụ, tên file là **gzip.msdos.exe**, khi được nén sẽ có tên là **gzip.msd.exe.gz**.

File được nén có thể được khôi phục trở lại dạng nguyên thể với lệnh **gzip -d** hoặc **gunzip**.

Với lệnh **gzip** có thể giải nén một hoặc nhiều file có phần mở rộng là **.gz**, **-gz**, **.z**, **-z**, **_z** hoặc **.Z... gunzip** dùng để giải nén các file nén bằng lệnh **gzip**, **zip**, **compress**, **compress -H**.

Lệnh **zcat** được sử dụng khi muốn xem nội dung một file nén trên thiết bị ra chuẩn.

Các tùy chọn:

- **-c, --stdout --to-stdout** : đưa ra trên thiết bị ra chuẩn; giữ nguyên file gốc không có sự thay đổi. Nếu có nhiều hơn một file đầu vào, đầu ra sẽ tuần tự là các file được nén một cách độc lập.
- **-d, --decompress --uncompress** : giải nén.
- **-f, --force** : thực hiện nén hoặc giải nén thậm chí file có nhiều liên kết hoặc file tương ứng thực sự đã tồn tại, hay dữ liệu nén được đọc hoặc ghi trên thiết bị đầu cuối.
- **-h, --help** : hiển thị màn hình trợ giúp và thoát.
- **-l, --list** : hiển thị những thông tin sau đối với một file được nén:
  - **compressed size:** kích thước của file nén
  - **uncompressed size**: kích thước của file được giải nén
  - **ratio**: tỷ lệ nén (0.0% nếu không biết)
  - **uncompressed_name**: tên của file được giải nén

  Nếu kết hợp với tùy chọn **--verbose**, các thông tin sau sẽ được hiển thị:
  - **method**: phương thức nén
  - **crc**: CRC 32-bit cho dữ liệu được giải nén
  - **date & time**: thời gian các file được giải nén

  Nếu kết hợp với tùy chọn **--name**, tên file được giải nén, thời gian giải nén được lưu trữ trong file nén

  Nếu kết hợp với tùy chọn **--verbose**, tổng kích thước và tỷ lệ nén của tất cả các file sẽ được hiển thị

  Nếu kết hợp với tùy chọn **--quiet**, tiêu đề và tổng số dòng của các file nén không được hiển thị.
- **-n, --no-name** : khi nén, tùy chọn này sẽ không lưu trữ tên file gốc và thời gian nén, (tên file gốc sẽ luôn được lưu nếu khi nén tên của nó bị cắt bỏ). Khi giải nén, tùy chọn này sẽ không khôi phục lại tên file gốc cũng như thời gian thực hiện việc nén. Tùy chọn này được ngầm định.
- **-N, --name** : tùy chọn này ngược với tùy chọn trên (**-n**), nó hữu ích trên hệ thống có sự giới hạn về độ dài tên file hay khi thời điểm nén bị mất sau khi chuyển đổi file.
- **-q, --quiet** : bỏ qua mọi cảnh báo.
- **-r, --recursive** : nén thư mục.
- **-S .suf, --suffix .suf** : sử dụng phần mở rộng **.suf** thay cho **.gz**. Bất kỳ phần mở rộng nào cũng có thể được đưa ra, nhưng các phần mở rộng khác **.z** và **.gz** sẽ bị ngăn chặn để tránh sự lộn xộn khi các file được chuyển đến hệ thống khác.
- **-t, --test** : tùy chọn này được sử dụng để kiểm tra tính toàn vẹn của file được nén
- **-v, --verbose** : hiển thị phần trăm thu gọn đối với mỗi file được nén hoặc giải nén
- **-#, --fast, --best** : điều chỉnh tốc độ của việc nén bằng cách sử dụng dấu #,

- Nếu -# là **-1** hoặc **--fast** thì sử dụng phương thức nén nhanh nhất (less compression),
- Nếu là **-9** hoặc **--best** thì sẽ dùng phương thức nén chậm nhất (best compression).
- Ngầm định mức nén là **-6** (đây là phương thức nén theo tốc độ nén cao).

* Nén, giải nén và xem file với các lệnh *compress, uncompress, zcat*

Cú pháp các lệnh:

compress [tùy-chọn] [<file>]

uncompress [tùy-chọn] [<file>]

zcat [tùy-chọn] [<file>]

Lệnh **compress** làm giảm kích thước của file và khi sử dụng lệnh này, file gốc sẽ bị thay thế bởi file nén với phần mở rộng là **.Z**, các thông tin khác liên quan đến file không thay đổi. Nếu không có tên file nào được chỉ ra, thông tin từ thiết bị vào chuẩn sẽ được nén và gửi ra thiết bị ra chuẩn. Lệnh **compress** chỉ sử dụng cho các file thông thường. Trong một vài trường hợp, nó sẽ bỏ qua liên kết tượng trưng. Nếu một file có nhiều liên kết cứng, **compress** bỏ qua việc nén file đó trừ khi có tùy chọn **-f**. Các tùy chọn:

- **-f** : nếu tùy chọn này không được đưa ra và **compress** chạy trong chế độ nền trước, người dùng sẽ được nhắc khi các file đã thực sự tồn tại và có thể bị ghi đè. Các file được nén có thể được khôi phục lại nhờ việc sử dụng lệnh **uncompress.**
- **-c** : tùy chọn này sẽ thực hiện việc nén hoặc giải nén rồi đưa ra thiết bị ra chuẩn, không có file nào bị thay đổi.

Lệnh **zcat** tương đương với **uncompress -c. zcat** thực hiện việc giải nén hoặc là các file được liệt kê trong dòng lệnh hoặc từ thiết bị vào chuẩn để đưa ra dữ liệu được giải nén trên thiết bị ra chuẩn.

- **-r** : nếu tùy chọn này được đưa ra, **compress** sẽ thực hiện việc nén các thư mục.
- **-v** : hiển thị tỷ lệ giảm kích thước cho mỗi file được nén.

CÂU HỎI VÀ BÀI tẬP

1. Trình bày đặc trưng của hệ quản lý file.
2. Trình bày khái niệm và cấu trúc siêu khối
3. Trình bày khái niệm và cấu trúc inode
4. Trình bày tên và tác dụng của các thư mục đặc biệt trong Linux
5. Thực hành các lệnh liên quan đến hệ thống file

CHƯƠNG 4. QUẢN TRỊ HỆ THỐNG VÀ NGƯỜI DÙNG

4.1. Quản trị người dùng

4.1.1. Tài khoản người dùng

Như đã biết, trong hệ điều hành đa người dùng, cần phân biệt người dùng khác nhau do quyền sở hữu các tài nguyên trong hệ thống, chẳng hạn như, mỗi người dùng có quyền hạn với file, quá trình của riêng họ. Điều này vẫn rất quan trọng thậm chí cả khi máy tính chỉ có một người sử dụng tại một thời điểm. Mọi truy cập hệ thống Linux đều thông qua tài khoản người dùng. Vì thế, mỗi người sử dụng được gắn với tên duy nhất (đã được đăng ký) và tên đó được sử dụng để đăng nhập. Tuy nhiên một người dùng thực sự có thể có nhiều tên đăng nhập khác nhau. Tài khoản người dùng có thể hiểu là tất cả các file, các tài nguyên, và các thông tin thuộc về người dùng đó.

Khi cài đặt hệ điều hành Linux, đăng nhập ***root*** sẽ được tự động tạo ra. Đăng nhập này được xem là thuộc về siêu người dùng (người dùng cấp cao, người quản trị), vì khi đăng nhập với tư cách người dùng root, có thể làm bất cứ điều gì muốn trên hệ thống. Tốt nhất chỉ nên đăng nhập **root** khi thực sự cần thiết, và hãy đăng nhập vào hệ thống với tư cách là một người dùng bình thường.

Nội dung chương này giới thiệu các lệnh để tạo một người dùng mới, thay đổi thuộc tính của một người dùng cũng như xóa bỏ một người dùng. Lưu ý, chỉ có thể thực hiện được các lệnh trên nếu có quyền của một siêu người dùng.

4.1.2. Các lệnh cơ bản quản lý người dùng

Người dùng được quản lý thông qua tên người dùng (thực ra là chỉ số người dùng). Nhân hệ thống quản lý người dùng theo chỉ số, vì việc quản lý theo chỉ số sẽ dễ dàng và nhanh thông qua một cơ sở dữ liệu lưu trữ các thông tin về người dùng. Việc thêm một người dùng mới chỉ có thể thực hiện được nếu đăng nhập với tư cách là siêu người dùng.

Để tạo một người dùng mới, cần phải thêm thông tin về người dùng đó vào trong cơ sở dữ liệu người dùng, và tạo một thư mục cá nhân cho riêng người dùng đó. Điều này rất cần thiết để thiết lập các biến môi trường phù hợp cho người dùng.

Lệnh chính để thêm người dùng trong hệ thống Linux là **useradd** (hoặc **adduser**).

4.1.2.1. File /etc/passwd

Danh sách người dùng cũng như các thông tin tương ứng được lưu trữ trong file **/etc/passwd**.

Ví dụ dưới đây là nội dung của file **/etc/passwd**:

mail:x:8:12:mail:/var/spool/mail:
games:x:12:100:games:/usr/games:
gopher:x:13:30:gopher:/usr/lib/gopher-data:
bien:x:500:0:Nguyen Thanh Bien:/home/bien:/bin/bash
sangnm:x:17:100:Nguyen Minh Sang:/home/sangnm:/bin/bash
lan:x:501:0:Lan GNU:/home/lan:/bin/bash

Mỗi dòng trong file tương ứng với bảy trường thông tin của một người dùng, và các trường này được ngăn cách nhau bởi dấu '**:**'. ý nghĩa của các trường thông tin đó lần lượt như sau:

Tên người dùng (**username**)
Mật khẩu người dùng (**passwd** - được mã hóa)
Chỉ số người dùng (**user id**)
Các chỉ số nhóm người dùng (**group id**)
Tên đầy đủ hoặc các thông tin khác về tài khoản người dùng (**comment**)

- Thư mục để người dùng đăng nhập
- Shell đăng nhập (chương trình chạy lúc đăng nhập)

Bất kỳ người dùng nào trên hệ thống đều có thể đọc được nội dung file **/etc/passwd**, và có thể đăng nhập với tư cách người dùng khác nếu họ biết được mật khẩu, đây chính là lý do vì sao mật khẩu đăng nhập của người dùng không hiển thị trong nội dung file.

4.1.2.2. Thêm người dùng với lệnh useradd

Siêu người dùng sử dụng lệnh **useradd** để tạo một người dùng mới hoặc cập nhật ngầm định các thông tin về người dùng.

Cú pháp lệnh:

useradd [tùy-chọn] <tên-người-dùng>

useradd -D [tùy-chọn]

Nếu không có tùy chọn **-D**, lệnh **useradd** sẽ tạo một tài khoản người dùng mới sử dụng các giá trị được chỉ ra trên dòng lệnh và các giá trị mặc định của hệ thống. Tài khoản người dùng mới sẽ được nhập vào trong các file hệ thống, thư mục cá nhân sẽ được tạo, hay các file khởi tạo được sao chép, điều này tùy thuộc vào các tùy chọn được đưa ra.

Các tùy chọn:

- **-c, comment** : soạn thảo trường thông tin về người dùng.
- **-d, home_dir** : tạo thư mục đăng nhập cho người dùng.
- **-e, expire_date** : thiết đặt thời gian (YYYY-MM-DD) tài khoản người dùng sẽ bị hủy bỏ.
- **-f, inactive_days** : tùy chọn này xác định số ngày trước khi mật khẩu của người dùng hết hiệu lực khi tài khoản bị hủy bỏ. Nếu =0 thì hủy bỏ tài khoản người dùng ngay sau khi mật khẩu hết hiệu lực, =-1 thì ngược lại (mặc định là -1).
- **-g, initial_group** : tùy chọn này xác định tên hoặc số khởi tạo đăng nhập nhóm người dùng. Tên nhóm phải tồn tại, và số của nhóm phải tham chiếu đến một nhóm đã tồn tại. Số nhóm ngầm định là 1.
- **-G, group** : danh sách các nhóm phụ mà người dùng cũng là thành viên thuộc các nhóm đó. Mỗi nhóm sẽ được ngăn cách với nhóm khác bởi dấu '**,**', mặc định người dùng sẽ thuộc vào nhóm khởi tạo.
- **-m** : với tùy chọn này, thư mục cá nhân của người dùng sẽ được tạo nếu nó chưa tồn tại.
- **-M** : không tạo thư mục người dùng.
- **-n** : ngầm định khi thêm người dùng, một nhóm cùng tên với người dùng sẽ được tạo. Tùy chọn này sẽ loại bỏ sự ngầm định trên.//
- **-p, passwd** : tạo mật khẩu đăng nhập cho người dùng.//
- **-s, shell** : thiết lập shell đăng nhập cho người dùng.
- **-u, uid** : thiết đặt chỉ số người dùng, giá trị này phải là duy nhất.

Thay đổi các giá trị ngầm định

Khi tùy chọn **-D** được sử dụng, lệnh **useradd** sẽ bỏ qua các giá trị ngầm định và cập nhật các giá trị mới.

- **-b, default_home** : thêm tên người dùng vào cuối thư mục cá nhân để tạo tên thư mục cá nhân mới.
- **-e, default_expire_date** : thay đổi thời hạn hết giá trị của tài khoản người dùng.
- **-f, default_inactive** : xác định thời điểm hết hiệu lực của mật khẩu đăng nhập khi tài khoản người dùng bị xóa bỏ.
- **-g, default_group** : thay đổi chỉ số nhóm người dùng.
- **-s, default_shell** : thay đổi shell đăng nhập.

Ngoài lệnh **useradd**, có thể tạo người dùng mới bằng cách sau:

Soạn thảo file **/etc/passwd** bằng **vipw**. Lệnh **vipw** mở trình soạn thảo trên hệ thống và hiệu chỉnh bản sao tạm của file **/etc/passwd**. Việc sử dụng file tạm và khóa file sẽ có tác dụng như một cơ chế khóa để ngăn việc hai người dùng cùng soạn thảo file một lúc. Lúc đó sẽ thêm dòng thông tin mới về người dùng cần tạo. Hãy cẩn thận trong việc soạn thảo tránh nhầm lẫn. Riêng trường mật khẩu nên để trống và tạo mật khẩu sau. Khi file này được lưu, **vipw** sẽ kiểm tra sự đồng nhất trên file bị thay đổi. Nếu tất cả mọi thứ dường như thích hợp thì có nghĩa là file **/etc/passwd** đã được cập nhật.

Ví dụ: thêm người dùng có tên là **new**, chỉ số người dùng **503**, chỉ số nhóm là **100**, thư mục cá nhân là **/home/new** và shell đăng nhập là shell bash:

 # vipw
mail:x:8:12:mail:/var/spool/mail:
games:x:12:100:games:/usr/games:
gopher:x:13:30:gopher:/usr/lib/gopher-data:
bien:x:500:0:Nguyen Thanh Bien:/home/bien:/bin/bash
sang:x:17:100:Nguyen Minh Sang:/home/sangnm:/bin/bash
lan:x:501:0:Lan GNU:/home/lan:/bin/bash
new::503:100:them mot nguoi moi:/home/new:/bin/bash

Tạo thư mục cá nhân của người dùng mới

 # mkdir /home/new

- Sao chép các file từ thư mục **/etc/skel/** (đây là thư mục lưu trữ các file cần thiết cho người dùng) vào file cá nhân vừa tạo
- Thay đổi quyền sở hữu và các quyền truy nhập file **/home/new** với các lệnh **chown** và **chmod**

 # chown new /home/new
 # chmod go=u,go-w /home/new

Thiết lập mật khẩu của người dùng:

 # passwd

new passwd:

Sau khi thiết lập mật khẩu cho người dùng ở bước cuối cùng, tài khoản người dùng sẽ làm việc. Nên thiết lập mật khẩu người dùng ở bước cuối cùng, nếu không họ có thể vô tình đăng nhập trong khi đang sao chép các file.

4.1.2.3. Thay đổi thuộc tính người dùng

Trong Linux có rất nhiều lệnh cho phép thay đổi một số các thuộc tính của tài khoản người dùng như:

- **chfn**: thay đổi thông tin cá nhân của người dùng.
- **chsh**: thay đổi shell đăng nhập.
- **passwd**: thay đổi mật khẩu.

Một số các thuộc tính khác sẽ phải thay đổi bằng tay. Ví dụ, để thay đổi tên người dùng, cần soạn thảo lại trực tiếp trên file **/etc/passwd** (với lệnh **vipw**).

Nhưng có một lệnh tổng quát cho phép có thể thay đổi bất kỳ thông tin nào về tài khoản người dùng, đó là lệnh **usermod.**

Cú pháp lệnh: ***usermod [tùy-chọn] <tên-đăng-nhập>***

Lệnh **usermod** sửa đổi các file tài khoản hệ thống theo các thuộc tính được xác định trên dòng lệnh.

Các tùy chọn:

- ✓ **-c, comment** : soạn thảo trường thông tin về người dùng.
- ✓ **-d, home_dir** : tạo thư mục đăng nhập cho người dùng.

- ✓ **-e, expire_date** : thiết đặt thời gian (YYYY-MM-DD) tài khoản người dùng sẽ bị hủy bỏ.
- ✓ **-f, inactive_days** : tùy chọn này xác định số ngày trước khi mật khẩu của người dùng hết hiệu lực khi tài khoản bị hủy bỏ. Nếu =0 thì hủy bỏ tài khoản người dùng ngay sau khi mật khẩu hết hiệu lực, =-1 thì ngược lại (mặc định là -1).
- ✓ **-g, initial_group** : tùy chọn này xác định tên hoặc số khởi tạo đăng nhập nhóm người dùng. Tên nhóm phải tồn tại, và số của nhóm phải tham chiếu đến một nhóm đã tồn tại. Số nhóm ngầm định là 1.
- ✓ **-G, group** : danh sách các nhóm phụ mà người dùng cũng là thành viên thuộc các nhóm đó. Mỗi nhóm sẽ được ngăn cách với nhóm khác bởi dấu '**,**', mặc định người dùng sẽ thuộc vào nhóm khởi tạo.
- ✓ **-m** : với tùy chọn này, thư mục cá nhân của người dùng sẽ được tạo nếu nó chưa tồn tại.
- ✓ **-M** : không tạo thư mục người dùng.
- ✓ **-n** : ngầm định khi thêm người dùng, một nhóm cùng tên với người dùng sẽ được tạo. Tùy chọn này sẽ loại bỏ sự ngầm định trên.//
- ✓ **-p, passwd** : tạo mật khẩu đăng nhập cho người dùng.//
- ✓ **-s, shell** : thiết lập shell đăng nhập cho người dùng.
- ✓ **-u, uid** : thiết đặt chỉ số người dùng, giá trị này phải là duy nhất.

Lệnh **usermod** không cho phép thay đổi tên của người dùng đang đăng nhập. Phải đảm bảo rằng người dùng đó không thực hiện bất kỳ quá trình nào trong khi lệnh **usermod** đang thực hiện thay đổi các thuộc tính của người dùng đó.

Ví dụ muốn thay đổi tên người dùng **new** thành tên mới là **newuser**, hãy gõ lệnh sau:

 # usermod -l new newuser

4.1.2.4 Xóa bỏ một người dùng

Để xóa bỏ một (tài khoản) người dùng, trước hết phải xóa bỏ mọi thứ có liên quan đến người dùng.

Cú pháp lệnh: ***userdel [-r] <tên-người-dùng>***

Lệnh này sẽ thay đổi nội dung của các file tài khoản hệ thống bằng cách xóa bỏ các thông tin về người dùng được đưa ra trên dòng lệnh. Người dùng này phải thực sự tồn tại. Tuỳ chọn **-r** có ý nghĩa các file tồn tại trong thư mục riêng của người dùng cũng như các file nằm trong các thư mục khác có liên quan đến người dùng bị xóa bỏ cùng lúc với thư mục người dùng.

Lệnh **userdel** sẽ không cho phép xóa bỏ người dùng khi họ đang đăng nhập vào hệ thống. Phải hủy bỏ mọi quá trình có liên quan đến người dùng trước khi xoá bỏ người dùng đó.

Ngoài ra cũng có thể xóa bỏ tài khoản của một người dùng bằng cách hiệu chỉnh lại file ***/etc/passwd.***

4.2. Các lệnh cơ bản liên quan đến nhóm người dùng

Mỗi người dùng trong hệ thống Linux đều thuộc vào một nhóm người dùng cụ thể. Tất cả những người dùng trong cùng một nhóm có thể cùng truy nhập một trình tiện ích, hoặc đều cần truy cập một thiết bị nào đó như máy in chẳng hạn.

Một người dùng cùng lúc có thể là thành viên của nhiều nhóm khác nhau, tuy nhiên tại một thời điểm, người dùng chỉ thuộc vào một nhóm cụ thể.

Nhóm có thể thiết lập các quyền truy nhập để các thành viên của nhóm đó có thể truy cập thiết bị, file, hệ thống file hoặc toàn bộ máy tính mà những người dùng khác không thuộc nhóm đó không thể truy cập được.

4.2.1. Nhóm người dùng và file /etc/group

Thông tin về nhóm người dùng được lưu trong file **/etc/group**, file này có cách bố trí tương tự như file **/etc/passwd**. Ví dụ nội dung của file ***/etc/group*** có thể như sau:

root:x:0:root
bin:x:1:root,bin,daemon
daemon:x:2:root,bin,daemon
sys:x:3:root,bin,adm
adm:x:4:root,adm,daemon
disk:x:6:root
lp:x:7:daemon,lp
mail:x:12:mail
huyen:x:500:
langnu:x:501:

Mỗi dòng trong file có bốn trường được phân cách bởi dấu '**:**'. ý nghĩa của các trường theo thứ tự xuất hiện như sau:

- Tên nhóm người dùng (groupname)
- Mật khẩu nhóm người dùng (**passwd** - được mã hóa), nếu trường này rỗng, tức là nhóm không yêu cầu mật khẩu
- Chỉ số nhóm người dùng (group id)
- Danh sách các người dùng thuộc nhóm đó (users)

4.2.1.2. Thêm nhóm người dùng

Cho phép hiệu chỉnh thông tin trong file **/etc/group** bằng bất kỳ trình soạn thảo văn bản nào có trên hệ thống của để thêm nhóm người dùng, nhưng cách nhanh nhất là sử dụng lệnh **groupadd**.

Cú pháp lệnh : ***groupadd [tùy-chọn] <tên-nhóm>***

Các tuỳ chọn:

- **g, gid** : tùy chọn này xác định chỉ số nhóm người dùng, chỉ số này phải là duy nhất. Chỉ số mới phải có giá trị lớn hơn 500 và lớn hơn các chỉ số nhóm đã có trên hệ thống. Giá trị từ 0 đến 499 chỉ dùng cho các nhóm hệ thống.
- **-r** : tùy chọn này được dùng khi muốn thêm một tài khoản hệ thống.
- **-f** : tùy chọn này sẽ bỏ qua việc nhắc nhở, nếu nhóm người dùng đó đã tồn tại, nó sẽ bị ghi đè.

Ví dụ: Thêm nhóm người dùng bằng cách soạn thảo file **/etc/group**:

installer:x:102:hieu, huy, sang
tiengviet:x:103:minh, long, dung

Hai dòng trên sẽ bổ sung hai nhóm người dùng mới cùng danh sách các thành viên trong nhóm: nhóm **installer** với chỉ số nhóm là **102** và các thành viên là các người dùng có tên **hieu**, **huy**, **sang**. Tương tự là nhóm **tiengviet** với chỉ số nhóm là **103** và danh sách các thành viên là **minh**, **long**, **dung**. Đây là hai nhóm (**102**, **103**) người dùng hệ thống.

Thêm nhóm người dùng mới với lệnh **groupadd**:

 # groupadd -r installer

Lệnh trên sẽ cho phép tạo một nhóm người dùng mới có tên là **installer**, tuy nhiên các thành viên trong nhóm sẽ phải bổ sung bằng cách soạn thảo file **/etc/group.**

4.2.1.2. Sửa đổi các thuộc tính của một nhóm người dùng

Cú pháp lệnh: ***groupmod [tùy-chọn] <tên-nhóm>***

Thông tin về các nhóm xác định qua tham số **tên-nhóm** được điều chỉnh.

Các tùy chọn:

- **-g, gid** : thay đổi giá trị chỉ số của nhóm người dùng.

- **-n, group_name** : thay đổi tên nhóm người dùng.

4.2.1.3. Xóa một nhóm người dùng

Nếu không muốn một nhóm nào đó tồn tại nữa thì chỉ việc xóa tên nhóm đó trong file **/etc/group**. Nhưng phải lưu ý rằng, chỉ xóa được một nhóm khi không có người dùng nào thuộc nhóm đó nữa.

Cú pháp lệnh: ***groupdel <tên-nhóm>***

Lệnh này sẽ sửa đổi các file tài khoản hệ thống, xóa tất cả các thực thể liên quan đến nhóm. Tên nhóm phải thực sự tồn tại.

4.2.2. Các lệnh cơ bản khác có liên quan đến người dùng

Ngoài các lệnh như thêm người dùng, xóa người dùng..., còn có một số lệnh khác có thể giúp ích rất nhiều nếu đang làm việc trên một hệ thống đa người dùng.

4.2.2.1. Đăng nhập với tư cách một người dùng khác khi dùng lệnh su

Đôi lúc muốn thực hiện lệnh như một người dùng khác và sử dụng các file hay thiết bị thuộc quyền sở hữu của người dùng đó. Lệnh **su** cho phép thay đổi tên người dùng một cách hiệu quả và cấp cho các quyền truy nhập của người dùng đó.

Cú pháp lệnh: ***su <người-dùng>***

Nếu đăng nhập với tư cách người dùng bình thường và muốn trở thành siêu người dùng (root) dùng lệnh sau:

 # su root

Khi đó hệ thống sẽ yêu cầu nhập mật khẩu của siêu người dùng. Nếu cung cấp đúng mật mã, thì sẽ là người dùng **root** cho tới khi dùng lệnh **exit** hoặc **CTRL+d** để đăng xuất ra khỏi tài khoản này và trở về đăng nhập ban đầu. Tương tự, nếu đăng nhập với tư cách **root** và muốn trở thành người dùng bình thường có tên là **newer** thì hãy gõ lệnh sau:

 # su newer

sẽ không bị hỏi về mật khẩu khi thay đổi từ siêu người dùng sang một người dùng khác. Tuy nhiên nếu đăng nhập với tư cách người dùng bình thường và muốn chuyển đổi sang một đăng nhập người dùng khác thì phải cung cấp mật khẩu của người dùng đó.

4.2.2.2. Xác định người dùng đang đăng nhập

** Xác định hiện tại có những ai đang đăng nhập trên hệ thống.*

Cú pháp lệnh: ***who [tùy-chọn]***

Các tuỳ chọn:

- **-H, --heading** : hiển thị tiêu đề của các cột trong nội dung lệnh.
- **-m** : hiển thị tên máy và tên người dùng với thiết bị vào chuẩn.
- **-q, --count** : hiển thị tên các người dùng đăng nhập và số người dùng đăng nhập.

Ví dụ:

```
# who
root tty1 Nov 15 03:54
lan pts/0 Nov 15 06:07
#
```

Lệnh **who** hiển thị ba cột thông tin cho từng người dùng trên hệ thống. Cột đầu là tên của người dùng, cột thứ hai là tên thiết bị đầu cuối mà người dùng đó đang sử dụng, cột thứ ba hiển thị ngày giờ người dùng đăng nhập.

Ngoài **who**, có thể sử dụng thêm lệnh **users** để xác định được những người đăng nhập trên hệ thống.

Ví dụ: **# *users***

* Trong trường hợp người dùng không nhớ nổi tên đăng nhập trong một phiên làm việc (điều này nghe có vẻ như hơi vô lý nhưng là tình huống đôi lúc gặp phải), hãy sử dụng lệnh **whoami** và **who am i**.

Cú pháp lệnh: ***whoami***

hoặc ***who am i***

Lệnh **who am i** sẽ hiện kết quả đầy đủ hơn với tên máy đăng nhập, tên người dùng đang đăng nhập, tên thiết bị và ngày giờ đăng nhập.

** Xác định thông tin người dùng*

Cú pháp lệnh: ***id [tùy-chọn] [người-dùng]***

Lệnh này sẽ đưa ra thông tin về người dùng được xác định trên dòng lệnh hoặc thông tin về người dùng hiện thời.

Các tuỳ chọn:

- **-g, --group** : chỉ hiển thị chỉ số nhóm người dùng.
- **-u, --user** : chỉ hiển thị chỉ số của người dùng.
- **--help** : hiển thị trang trợ giúp và thoát.

4.2.2.3. Xác định các quá trình đang được tiến hành

Lệnh **w** cho phép xác định được thông tin về các quá trình đang được thực hiện trên hệ thống và những người dùng tiến hành quá trình đó.

Cú pháp lệnh: ***w [người-dùng]***

Lệnh **w** đưa ra thông tin về người dùng hiện thời trên hệ thống và quá trình họ đang thực hiện. Nếu chỉ ra người dùng trong lệnh thì chỉ hiện ra các quá trình liên quan đến người dùng đó.

4.3. Quản trị hệ thống

4.3.1. Quản lý tiến trình

4.3.1.1. Giới thiệu chung

Linux là hệ điều hành đa người sử dụng, đa tiến trình. Việc điều khiển tiến trình đang hoạt động rất quan trọng trong quản trị hệ thống Linux.

Tiến trình là đoạn chương trình đơn chạy trên không gian địa chỉ ảo của nó.

Cần phân biệt tiến trình với lệnh vì một dòng shell có thể sinh ra nhiều tiến trình.

Có ba loại tiến trình chính trên Linux:

- Tiến trình đối thoại (interarctive process): Là tiến trình khởi động và quản lý bởi shell, kể cả tiến trình foreground hoặc background.
- Tiến trình batch (batch process): Tiến trình không gắn liền đến terminal và được nằm trong hàng đợi để sẵn sàng thực hiện.
- Tiến trình ẩn trên bộ nhớ (Daemon process): Là các tiến trình chạy dưới nền (background). Các tiến trình này thường được khởi động từ đầu. Đây là các chương trình sau khi được gọi lên bộ nhớ, đợi thụ động các yêu cầu của chương trình khách (client).

4.3.1.2. Điều khiển và giám sát các tiến trình

Như để cập trước đây, các tiến trình thường trực thường được bắt đầu bằng tiến trình init khi khởi động. Chúng ta có thể điều khiển tiến trình nào chạy ngay khi khởi động bằng cách cấu hình lại các file cấu hình và kịch bản của init. Ngoại trừ các tiến trình thường trực, các loại tiến trình khác mà chúng ta sẽ chạy được gọi là các tiến trình của người sử dụng hay các tiến trình tương tác. Chúng ta phải chạy một tiến trình tương tác thông qua một shell. Mỗi một shell chuẩn cung cấp một dòng lệnh khi người sử dụng vào tên của một chương trình. Khi người sử dụng vào tên chương trình hợp lệ trên dòng lệnh, shell sẽ tự tạo một bản copy như

một tiến trình mới và thay thế tiến trình mới với chương trình được đặt tên trên dòng lệnh. Nói một cách khác shell sẽ chạy chương trình được đặt tên như một tiến trình khác. Để lấy thông tin về tất cả các tiến trình đang chạy trên hệ thống của chúng ta, chúng ta cần chạy tiện ích có tên là ps

4.3.1.3. Các lệnh xử lý tiến trình

** Lệnh fg và lệnh bg*

Trong phần trước, cách thức gõ phím **CTRL+z** để tạm dừng một tiến trình đã được giới thiệu. Linux còn người dùng cách thức để chạy một chương trình dưới chế độ nền (***background***) - sử dụng lệnh **bg** - trong khi các chương trình khác đang chạy, và để chuyển một chương trình vào trong chế độ nền - dùng ký hiệu **&.**

Nếu một tiến trình hoạt động mà không đưa ra thông tin nào trên màn hình và không cần nhận bất kỳ thông tin đầu vào nào, thì có thể sử dụng lệnh **bg** để đưa nó vào trong chế độ nền (ở chế độ này nó sẽ tiếp tục chạy cho đến khi kết thúc). Khi chương trình cần đưa thông tin ra màn hình hoặc nhận thông tin từ bàn phím, hệ thống sẽ tự động dừng chương trình và thông báo cho người dùng. Cũng có thể sử dụng chỉ số điều khiển công việc (***job control***) để làm việc với chương trình nào muốn. Khi chạy một chương trình trong chế độ nền, chương trình đó được đánh số thứ tự (được bao bởi dấu ngoặc vuông []), theo sau là chỉ số của tiến trình.

Sau đó có thể sử dụng lệnh **fg + số thứ tự của chương trình** để đưa chương trình trở lại chế độ nổi và tiếp tục chạy.

Để có một chương trình (hoặc một lệnh ống) tự động chạy trong chế độ nền, chỉ cần thêm ký hiệu '**&**' vào cuối lệnh.

Trong một số hệ thống, khi tiến trình nền kết thúc thì hệ thống sẽ gửi thông báo tới người dùng, nhưng trên hầu hết các hệ thống, khi tiến trình trên nền hoàn thành thì hệ thống sẽ chờ cho đến khi người dùng gõ phím **Enter** thì mới hiển thị dấu nhắc lệnh mới kèm theo thông báo hoàn thành tiến trình (thường thì một tiến trình hoàn thành sau khoảng 20 giây).

** Hiển thị các tiến trình đang chạy*

Linux cung cấp cho người dùng hai cách thức nhận biết có những chương trình nào đang chạy trong hệ thống. Cách dễ hơn, đó là lệnh **jobs** sẽ cho biết các tiến trình nào đã dừng hoặc là được chạy trong chế độ nền.

Cách phức tạp hơn là sử dụng lệnh **ps**. Lệnh này cho biết thông tin đầy đủ nhất về các tiến trình đang chạy trên hệ thống.

Ví dụ:

```
# ps
PID        TTY          TIME         CMD
7813       pts/0        00:00:00     bash
7908       pts/0        00:00:00     ps
#
```

Trong đó:

- PID - chỉ số của tiến trình,
- TTY - tên thiết bị đầu cuối trên đó tiến trình được thực hiện,
- TIME - thời gian để chạy tiến trình,
- CMD - lệnh khởi tạo tiến trình

Cú pháp lệnh: ***ps [tùy-chọn]***

Lệnh **ps** có một lượng quá phong phú các tùy chọn được chia ra làm nhiều loại. Dưới đây là một số các tùy chọn hay dùng.

Các tùy chọn đơn giản:

- **-A, -e** : chọn để hiển thị tất cả các tiến trình.
- **-T** : chọn để hiển thị các tiến trình trên trạm cuối đang chạy.
- **-a** : chọn để hiển thị tất cả các tiến trình trên một trạm cuối, bao gồm cả các tiến trình của những người dùng khác.
- **-r** : chỉ hiển thị tiến trình đang được chạy.

Chọn theo danh sách

- **-C** : chọn hiển thị các tiến trình theo tên lệnh.
- **-G** : hiển thị các tiến trình theo chỉ số nhóm người dùng.
- **-U** : hiển thị các tiến trình theo tên hoặc chỉ số của người dùng thực sự (người dùng khởi động tiến trình).
- **-p** : hiển thị các tiến trình theo chỉ số của tiến trình.
- **-s** : hiển thị các tiến trình thuộc về một phiên làm việc.
- **-t** : hiển thị các tiến trình thuộc một trạm cuối.
- **-u** : hiển thị các tiến trình theo tên và chỉ số của người dùng hiệu quả.

Thiết đặt khuôn dạng được đưa ra của các tiến trình

- **-f** : hiển thị thông tin về tiến trình với các trường sau UID - chỉ số người dùng, PID - chỉ số tiến trình, PPID - chỉ số tiến trình khởi tạo ra tiến trình, C - , STIME - thời gian khởi tạo tiến trình, TTY - tên thiết bị đầu cuối trên đó tiến trình được chạy, TIME - thời gian để thực hiện tiến trình, CMD - lệnh khởi tạo tiến trình
- **-l** : hiển thị đầy đủ các thông tin về tiến trình với các trường F, S, UID, PID, PPID, C, PRI, NI, ADDR, SZ, WCHAN, TTY, TIME, CMD
- **-o xâu-chọn** : hiển thị các thông tin về tiến trình theo dạng do người dùng tự chọn thông qua xâu-chọn các ký hiệu điều khiển hiển thị có các dạng như sau:

%C, %cpu	% CPU được sử dụng cho tiến trình
%mem	% bộ nhớ được sử dụng để chạy tiến trình
%G	tên nhóm người dùng
%P	chỉ số của tiến trình cha khởi động ra tiến trình con
%U	định danh người dùng
%c	lệnh tạo ra tiến trình
%p	chỉ số của tiến trình
%x	thời gian để chạy tiến trình
%y	thiết bị đầu cuối trên đó tiến trình được thực hiện

** Hủy tiến trình*

Trong một số trường hợp, sử dụng lệnh **kill** để hủy bỏ một tiến trình. Điều quan trọng nhất khi sử dụng lệnh **kill** là phải xác định được chỉ số của tiến trình mà chúng ta muốn hủy.

Cú pháp lệnh:

kill [tùy-chọn] <chỉ-số-của-tiến-trình>

kill -l [tín hiệu]

Lệnh **kill** sẽ gửi một **tín hiệu** đến tiến trình được chỉ ra. Nếu không chỉ ra một tín hiệu nào thì ngầm định là tín hiệu **TERM** sẽ được gửi.

- **-s** : xác định tín hiệu được gửi. Tín hiệu có thể là số hoặc tên của tín hiệu. Dưới đây là một số tín hiệu hay dùng:
  - SIGHUP **(hang up**) đây là tín hiệu được gửi đến tất cả các quá trình đang chạy trước khi **logout** khỏi hệ thống
  - SIGINT (**interrupt**) đây là tín hiệu được gửi khi nhấn **Ctrl+C**
  - SIGK (**kill**) tín hiệu này sẽ dừng tiến trình ngay lập
  - SIGT tín hiệu này yêu cầu dừng tiến trình ngay lập tức, nhưng cho phép chương trình xóa các file tạm
- **-p** : lệnh **kill** sẽ chỉ đưa ra chỉ số của tiến trình mà không gửi một tín hiệu nào.

- **-l** : hiển thị danh sách các tín hiệu mà lệnh **kill** có thể gửi đến các tiến trình (các tín hiệu này có trong file **/usr/include/Linux/signal.h**)

** Cho máy ngừng hoạt động một thời gian*

Nếu muốn cho máy nghỉ một thời gian mà không muốn tắt vì ngại khởi động lại thì cần dùng lệnh **sleep**.

Cú pháp lệnh: ***sleep [tùy-chọn] NUMBER[SUFFIX]***

NUMBER: số giây(s) ngừng hoạt động.
SUFFIX : có thể là giây(s) hoặc phút(m) hoặc giờ hoặc ngày(d)

Các tùy chọn:

- **--help** : hiện thị trợ giúp và thoát
- **--version** : hiển thị thông tin về phiên bản và thoát

** Xem cây tiến trình*

Đã biết lệnh để xem các tiến trình đang chạy trên hệ thống, tuy nhiên trong Linux còn có một lệnh cho phép có thể nhìn thấy mức độ phân cấp của các quá trình, đó là lệnh **pstree**.

Cú pháp lệnh: ***pstree [tùy-chọn] [pid | người-dùng]***

Lệnh **pstree** sẽ hiển thị các tiến trình đang chạy dưới dạng cây tiến trình. Gốc của cây tiến trình thường là init. Nếu đưa ra tên của một người dùng thì cây của các tiến trình do người dùng đó sở hữu sẽ được đưa ra. **pstree** thường gộp các nhánh tiến trình trùng nhau vào trong dấu ngoặc vuông,

ví dụ:

nit -+-getty
|-getty
|-getty
|-getty

thành

init ---4*[getty]

Các tùy chọn:

- **-a** : chỉ ra tham số dòng lệnh. Nếu dòng lệnh của một tiến trình được tráo đổi ra bên ngoài, nó được đưa vào trong dấu ngoặc đơn.
- **-c** : không thể thu gọn các cây con đồng nhất. Mặc định, các cây con sẽ được thu gọn khi có thể
- **-h :** hiển thị tiến trình hiện thời và "tổ tiên" của nó với màu sáng trắng
- **-H** : giống như tùy chọn -h, nhưng tiến trình con của tiến trình hiện thời không có màu sáng trắng
- **-l** : hiển thị dòng dài.
- **-n** : sắp xếp các tiến trình cùng một tổ tiên theo chỉ số tiến trình thay cho sắp xếp theo tên

** Lệnh thiết đặt lại độ ưu tiên của tiến trình*

Ngoài các lệnh xem và hủy bỏ tiến trình, trong Linux còn có hai lệnh liên quan đến độ ưu tiên của tiến trình, đó là lệnh **nice** và lệnh **renice.**

- Để chạy một chương trình với độ ưu tiên định trước, hãy sử dụng lệnh **nice.**

Cú pháp lệnh: ***nice [tùy-chọn] [lệnh [tham-số]...]***

Lệnh **nice** sẽ chạy một chương trình (lệnh) theo độ ưu tiên đã sắp xếp. Nếu không có **lệnh**, mức độ ưu tiên hiện tại sẽ hiển thị. Độ ưu tiên được sắp xếp từ -20 (mức ưu tiên cao nhất) đến 19 (mức ưu tiên thấp nhất).

- **-ADJUST** : tăng độ ưu tiên theo ADJUST đầu tiên

- **--help** : hiển thị trang trợ giúp và thoát

- Để thay đổi độ ưu tiên của một tiến trình đang chạy, hãy sử dụng lệnh **renice.**

Cú pháp lệnh: ***renice <độ-ưu-tiên> [tùy-chọn]***

Lệnh **renice** sẽ thay đổi mức độ ưu tiên của một hoặc nhiều tiến trình đang chạy.

- **-g** : thay đổi quyền ưu tiên theo nhóm người dùng
- **-p** : thay đổi quyền ưu tiên theo chỉ số của tiến trình
- **-u** : thay đổi quyền ưu tiên theo tên người dùng

4.3.2 Quản trị phần mềm

Linux đã phát triển hệ thống quản trị phần mềm có giao diện thân thiện và hiệu quả là RPM (Redhad Package Manager). RPM là chương trình quản lý các package, nó tự động làm các quá trình như cài đặt, nâng cấp, xóa và bảo trì phần mềm trong Linux.

Lệnh rpm có rất nhiều tham số. Để kiểm tra một cách nhanh chóng danh sách các phần mềm có trong hệ thống, ta dùng:

rpm –qa

qa có nghĩa là query all packet.

Để cài đặt một phần mềm, ta dùng lệnh

rpm –ivh <phần_mềm>

Để xóa một phần mềm, ta dùng lệnh

rpm --erase <phần_mềm>

Ngoài ra Linux còn dùng tiện ích kpacket trong môi trường KDE giúp ta quản lý các kpacket một cách có hiệu quả trên giao diện đồ họa.

4.3.3. Quản trị hệ thống Linux

** Hệ thống /proc*

Thư mục /proc rất quan trọng, đóng vai trò sống còn đối với Linux. Đây là thư mục ảo, nó chứa thông tin của hệ thống, các tiến trình, các tham số hệ thống theo thời gian thực. Các thông tin chưa trong nó được tạo ra một cách động dựa trên các quá trình startup và shutdown của hệ thống.

** Các lệnh bảo trì khác:*

Lệnh free: Hiển thị tổng dung lượng bộ nhớ chính và swap đang được dùng và còn trống cũng như share memory và buffers được dùng bởi kernel.

Lệnh df: Hiển thị dung lượng đĩa còn trống trên hệ thống file. Đơn vị là 1K block, với 512B cho 1 blick.

Lệnh sudo: Cho phép quản trị hệ thống nâng cấp quyền truy xuất đến một tập lệnh quản trị hệ thống cho một vài user thường.

** Hệ thống log file:*

var/log/message: Cho biết các sự kiện diễn ra trong hệ thống bao gồm các hành động start, stop các tiến trình, user login logout, các lỗi hệ thống…

var/log/sercure: Lưu giữ thông tin thống kê login, logout và các ipadress truy cập vào hệ thống.

var/log/boot: Lưu các thông tin khi hệ thống mới khởi động.

dmesg: Hiển thị các thông tin của phần cứng khi hệ thống boot lên

dmesg |more

CÂU HỎI VÀ BÀI TẬP

1. Trình bày cấu trúc thông tin lưu trữ về tài khoản người của một dùng trong trong file **/etc/passwd**
2. Trình bày cấu trúc thông tin lưu trữ về nhóm tài khoản người dùng trong trong file **/etc/group**
3. Thực hành các lệnh liên quan đến quản lý nhóm, quản lý tài khoản người dùng.

Chương 5. TRUYỀN THÔNG VÀ MẠNG UNIX-LINUX

5.1. Lệnh truyền thông

5.1.1. Lệnh write

Lệnh write được dùng để trao đổi giữa những người hiện đang cùng làm việc trong hệ thống.

Cú pháp lệnh:

$write <tên người dùng> [<tên trạm cuối>]

Ví dụ cần gửi thông báo đến người dùng có tên user2 sẽ gõ:

$write user2 tty43

- Nếu người dùng user2 hiện không làm việc thì trên màn hình người dùng user1 sẽ hiện ra: "user2 is not logged in" và hiện lại dấu mời shell.
- Nếu người dùng user2 đang làm việc, máy người dùng user2 sẽ phát ra tiếng chuông và trên màn hình hiện ra:
  Message from user1 on tty17 at <giờ, phút>

Cùng lúc đó, tại máy của user1 màn hình trắng để hiện những thông tin gửi tới người dùng user2. Người gửi gõ thông báo của mình theo quy tắc:

- Kết thúc một dòng bằng cụm -o,
- Kết thúc dòng cuối cùng (hết thông báo) bằng cụm -oo.

Để kết thúc kết nối với người dùng user2, người dùng user1 gõ ctrl-d.

Để từ chối mọi việc nhận thông báo từ người khác, sử dụng lệnh không nhận thông báo:

$mesg n (n - no)

Một người khác gửi thông báo đến người này sẽ nhận được việc truy nhập không cho phép permission denied.

Để tiếp tục cho phép người khác gửi thông báo đến, sử dụng lệnh:

$mesg y (y - yes)

5.1.2. Lệnh mail

Lệnh mail cho phép gửi thư điện tử giữa các người dùng, song hoạt động theo chế độ off-line (gián tiếp). Khi dùng lệnh write để truyền thông cho nhau thì đòi hỏi hai người gửi và nhận đồng thời đang làm việc và cùng chấp nhận cuộc trao đổi đó. Cách thức sử dụng mail là khác hẳn: một trong hai người gửi hoặc nhận có thể không đăng nhập vào hệ thống.

Để đảm bảo cách thức truyền thông gián tiếp (còn gọi là off-line) như vậy, hệ thống tạo ra cho mỗi người dùng một hộp thư riêng. Khi một người dùng lệnh mail gửi thư đến một người khác thì thư được tự động cho vào hộp thư của người nhận và người nhận sau đó cũng dùng lệnh mail để xem trong hộp thư có thư mới hay không. Không những thế mail còn cho phép sử dụng trên mạng internet (địa chỉ mail thường dưới dạng ***tên- login@máy.mạng.lĩnh-vực.quốc-gia***).

- Lệnh mail chỉ yêu cầu người gửi (hoặc người nhận) login trong hệ thống. Việc nhận và gửi thư được tiến hành từ một người dùng. Thư gửi đi cho người dùng khác, được lưu tại hộp thư của hệ thống.

- Tại thời điểm login hệ thống, người dùng có thể thấy được có thư mới khi trên màn hình xuất hiện dòng thông báo "you have mail".

Lệnh mail trong UNIX gồm 2 chức năng: gửi thư và quản lý thư. Tương ứng, có hai chế độ làm việc với lệnh mail: mode lệnh (command mode) quản trị thư và mode soạn (compose mode) cho phép tạo thư.

a/ Mode soạn

Mode soạn làm việc trực tiếp với một thư và gửi ngay cho người khác. Mode soạn thực chất là sử dụng lệnh mail có tham số:

$mail < tên_người_nhận

Lệnh này cho phép soạn và gửi thư cho người nhận có tên được chỉ.

Sau khi gõ lệnh, màn hình bị xóa và con trỏ soạn thảo nhấp nháy ở góc trên, trái để người dùng gõ nội dung thư.

Để kết thúc soạn thư, hãy gõ ctrl-d, màn hình của mail biến mất và dấu mời của shell lại xuất hiện.

Chú ý: Dạng sau đây được dùng để gửi thư đã soạn trong nội dung một file nào đó (chú ý dấu "<" chỉ dẫn thiết bị vào chuẩn là nội dung file thay vì cho bàn phím):

$mail tên_người_nhận < tên_file_nội_dung_thư

Cách làm trên đây hay được sử dụng trong gửi / nhận thư điện tử hoặc liên kết truyền thông vì cho phép tiết kiệm được thời gian kết nối vào hệ thống, đặc biệt chi phí phải trả khi kết nối là đáng kể.

b/ Mode lệnh

Như đã nói sử dụng mode lệnh của mail để quản lý hộp thư. Vào mail theo mode lệnh khi dùng lệnh mail không tham số:

$mail

Sau khi gõ lệnh, màn hình mail ở mode lệnh được hiện ra với dấu mời của mode lệnh. (phổ biến là dấu chấm hỏi "?") Tại đây người dùng sử dụng các lệnh của mail quản lý hệ thống thư của mình.

Cần trợ giúp gõ dấu chấm hỏi (màn hình có hai dấu ??): ? màn hình hiện ra dạng sau:

<số>	Hiện thư số <số>
(dấu cách)	Hiện thư ngay phía trước
+	Hiện thư ngay tiếp theo
l cmd	thực hiện lệnh cmd
dq	xóa thư hiện thời và ra khỏi mail
m user	gửi thư hiện thời cho người dùng
s tên-file	ghi thư hiện thời vào file có tên
r [tên-file]	trả lời thư hiện thời (có thể từ file)
d <số>	xóa thư số
u	khôi phục thư hiện thời
u <số>	khôi phục thư số
m <user>...	chuyển tiếp thư tới các người dùng khác
q	ra khỏi mail

Thực hiện các lệnh theo chỉ dẫn trên đây để quản trị được hộp thư của cá nhân.

5.1.3. Lệnh talk

Trong Linux cho phép sử dụng lệnh talk thay thế cho lệnh write.

5.2 Cấu hình Card giao tiếp mạng

Để các máy có thể giao tiếp được với nhau trong mạng theo giao thức TCP/IP, thiết bị dùng làm phương tiện giao tiếp đó là Card giao tiếp mạng (network card). Để quản lý thiết bị này Linux cung cấp lệnh ifconfig. Lệnh này dùng để xem các thông tin về cấu hình mạng hiện thời của máy cũng như gán các địa chỉ cho các card giao tiếp mạng (interface). Ngoài ra ta cũng có thể dùng lệnh này để kích hoạt hoặc tắt một card mạng.

/sbin/ifconfig <giao diện> [<địa chỉ>] [arp | -arp][broadcast <địa chỉ>]
[netmask <mặt nạ mạng>]

Trong đó:

- **<giao diện> :** tên của thiết bị giao tiếp mạng, chẳng hạn eth0 cho card mạng đầu tiên, eth1 cho card mạng thứ hai.
- <**địa chỉ**> **:** địa chỉ mạng sẽ gán cho giao diện này.
- **up :** tuỳ chọn này sẽ kích hoạt giao diện được chỉ ra.
- **down :** tuỳ chọn này sẽ tắt giao diện được chỉ ra.
- **arp | -arp :** cho phép hay cấm giao thức ARP trên giao diện này.
- **broadcast <địa chỉ> :** xác định địa chỉ quảng bá cho giao diện này.
- **netmask <mặt nạ mạng> :** xác định mặt nạ mạng cho giao diện này.

Để xem cấu hình của máy hiện tại ta dùng lệnh

 # ifconfig

Muốn chỉ xem các thông tin về một card mạng nào đó thôi ta dùng lệnh:

 # ifconfig eth0

Muốn kích hoạt một card mạng ta dùng lệnh

 # ifconfig eth0 up

Muốn tắt một card mạng ta dùng lệnh

 # ifconfig eth0 down

Muốn đặt lại địa chỉ cho một card mạng ta dùng lệnh:

 # ifconfig eth0 203.162.9.154 netmask 255.255.255.248

Ngoài ra nếu máy tính có cài giao diện GNOME cùng các package quản lý mạng của GNOME thì ta có thể sử dụng lệnh có giao diện đồ hoạ giúp cho việc cấu hình các tham số card mạng dễ dàng hơn. Để có công cụ này ta phải cài đặt package redhat-config-network-xxx.rpm trong đó xxx là số hiệu phiên bản của chương trình.

Trong giao diện đồ hoạ GNOME ta đánh lệnh redhat-config-network, một hộp thoại sẽ hiện lên cho phép ta thay đổi các tham số cho từng card mạng được cài trên máy.

5.3. Các dịch vụ mạng

5.3.1 Hệ thông tin mạng NIS

Khi sử dụng hệ thống mạng nói chung, mục đích của chúng ta là làm cho môi trường mạng trở nên trong suốt đối với người dùng. Một trong những điểm quan trọng là làm cho các dữ liệu quan trọng như là thông tin về người dùng, về các trạm trong mạng là đồng nhất trên tất cả các trạm làm việc. NIS (Network Information System) là một ứng dụng cung cấp các tiện ích truy nhập cơ sở dữ liệu để phân phối thông tin, chẳng hạn như dữ liệu trong /etc/passwd và /etc/group cho tất cả các máy trạm trên mạng. Điều này làm cho mạng trở nên một hệ thống duy nhất. NIS được xây dựng trên việc sử dụng dịch vụ RPC (Remote Procedure Call). Nó bao gồm một thư viện máy chủ, thư viện máy trạm và các công cụ quản trị. Ban đầu NIS được gọi là những trang vàng (Yellow Pages –YP). Cùng với sự phát triển của NIS mà có sự xuất hiện khác nhau trong các phiên bản. NIS truyền thống được xây dựng trên thư viện libc 4/5. NIS+ là sự mở rộng của NIS song vẫn hỗ trợ bảo mật thông tin. NYS là một phiên bản chuẩn hỗ trợ cả NIS và NIS+.

Hoạt động của NIS: NIS lưu trữ cơ sở dữ liệu về thông tin quản trị mạng trong các file maps. Các file này được đặt trên một NIS server trung tâm, từ đó các NIS client có thể truy nhập đến các thông tin thông qua dịch vụ RPC. Các file maps thường là các file theo định dạng DMB, một dạng cơ sở dữ liệu đơn giản. Các file maps được tạo ra từ các file văn bản như /etc/hosts hay /etc/passwd. Mỗi file văn bản này có thể có nhiều file maps khác nhau tùy thuộc vào khóa của nó.

Ví dụ nếu khóa là tên máy trạm thì ta có file hosts.byname, nếu khóa là địa chỉ IP thì ta có file hosts.byname.

File chủ	File maps tương ứng
/etc/hosts	hosts.addr Hosts.byname
/etc/networks	network.byname network.byaddr
/etc/passwd	passwd.byname passwd.byid
/etc/groups	Groups.byname group.byid
/etc/services	service.byname service.bynumber
/etc/rpc	rpc.bynumber rpc.byname
/etc/protocol	protocol.byname protocol.bynumber
/usr/lib/aliases	mail.aliases

Mỗi một file maps có một tên ngắn hơn để đẽ nhớ đối với người dùng gọi là các nickname. Để hiển thị danh sách các nickname ta dùng lệnh *ypcat*:

#ypcat -x

Các chương trình máy chủ của NIS thường có tên là ypserv. Trong các mạng cỡ nhỏ ta chỉ cần một máy làm máy chủ NIS. Một miền (domain) NIS là một tập hợp các máy trạm được quản lý bởi một máy chủ NIS. Để hiển thị và đặt tên cho một miền ta sử dụng lệnh:

#domainname nis-domain

Tên miền NIS sẽ cho biết máy chủ của miền nào các ứng dụng sẽ truy cập để nhận thông tin cần thiết. Để biết được máy chủ nào trong mạng là NIS server, các chương trình ứng dụng phải hỏi *ypbind*, một chương trình chạy ngầm có nhiệm vụ phát hiện các NIS server trên mạng. Nó sẽ phát các gói tin quảng bá để tìm các máy chủ NIS trên mạng hoặc sử dụng các thông tin trong các file cấu hình người quản trị đã cung cấp.

5.3.2. Cài đặt và cấu hình cho máy chủ NIS

Với NIS ta có khái niệm máy chủ NIS chính và máy chủ NIS phụ, một miền chỉ có thể có một máy chủ NIS chính. Khi trong mạng có nhiều máy trạm làm việc, một máy chủ NIS có thể bị quá tải, hoặc khi có sự cố thì toàn bộ hệ thống mạng đó sẽ không thể hoạt động được. Các máy chủ NIS phụ sẽ giúp giải quyết vấn đề này. Việc cài đặt các máy chủ NIS phụ chỉ khác máy chủ NIS chính ở chỗ tạo ra các file map. Chúng không được tạo ra bằng *makedbm* mà được lấy về từ máy chủ chính.

Bây giờ ta tìm hiểu cách cài đặt máy chủ NIS chính. Trước tiên ta phải cài đặt phần mềm *ypserv* lên máy tính. Chương trình sẽ nằm trong package ypserv-xxx.rpm. Ta có thể cài đặt bằng lệnh:

#rpm -ivh ypserv-xxx.rpm

#mkdir /var/yp/nis-domain

Để tạo các file cơ sở dữ liệu ta sử dụng chương trình ***makedbm***. Do đó phải đảm bảo chương trình đã được cài trên máy, việc tạo lập sẽ được tiến hành thông qua một makefile. Trong file này sẽ chứa các lệnh cần thiết để tạo ra file maps. Sau khi cài đặt phần mềm ta dùng lệnh ***make***:

#domainname nis-domain

#cd /var/yp

#make

Các file map không tự động cập nhật mỗi khi ta sửa thông tin quản trị. Do vậy mỗi khi có sự thay đổi, ta cần thực hiện lại lệnh ***make*** để cập nhật sự sửa đổi.

5.3.3. Cài đặt các máy trạm NIS

Trước tiên ta cần cài đặt phần mềm ***ypbind*** lên máy trạm bằng lệnh:

#rpm -ivh ypbind-xxx.rpm

Bước tiếp theo là chỉ ra tên của máy chủ và tên miền NIS mà trạm này sẽ sử dụng bằng cách thay đổi thông tin trong file ***/etc/yp.conf*** như sau:

#/etc/yp.conf domainname nis-domain
server lnserver

Dòng đầu tiên cho biết máy trạm này thuộc vào miền NIS có tên là ***nis-domain***. Nếu không có dòng lệnh này thì ta có thể chỉ ra bằng cách đánh lệnh ***domainname*** tại dấu nhắc dòng lệnh. Dòng thứ 2 chỉ ra tên máy chủ NIS. Địa chỉ IP của tên máy chủ này phải xuất hiện trong file ***/etc/hosts***. Hoặc ta có thể sử dụng địa chỉ IP ngay trên dòng này.

Khi ta sử dụng máy tính thường xuyên phải thay đổi miền NIS, ta có thể chỉ ra nhiều miền NIS và các máy chủ tương ứng với nó bằng lệnh server. File cấu hình dưới đây cho phép thực hiện điều đó:

#yp.conf
server ln-server1 domainname1
server ln-server2 domainname2

Khi muốn sử dụng một miền khác thì ta chỉ cần đánh lại lệnh *domainname* để xác định miền ta tương ứng.

Sau khi đã tạo ra các file cấu hình cơ bản, ta nên kiểm tra xem chương trình ypbind đã hoạt động hay chưa. Trước hết, khởi động *ypbind,* sau đó dùng tiện ích *ypcat* để lấy thông tin quản lý bởi NIS server. Để xem thông tin về địa chỉ IP của các trạm ta dùng lệnh:

#ypbind

#ypcat hosts.byname
192.168.50.1 may1

192.168.50.1 may2

...

Nếu ta không nhận được kết quả như trên hoặc ta nhận được một thông báo lỗi "can't bind to servers domain", có nghĩa là hệ thống NIS hoạt động chưa tốt, ta có thể kiểm tra xem tên miền và tên máy chủ trong file yp.conf đã chính xác chưa và sau đó *ping* máy chủ. Nếu máy chủ đã hoạt động ta kiểm tra xem sự hoạt động của ypserv bằng lệnh rpcinfo:

#rpcinfo –u serverhost ypserv
program 10004 version 2 ready and waiting

Nếu ta nhận được thông báo như trên là ypserv đang hoạt động tốt.

5.3.4. Lựa chọn các file map

Khi sử dụng NIS ta cần xác định những file cấu hình nào của các máy trạm sẽ được thay thể bởi NIS. Thông thường NIS được sử dụng để tra cứu các thông tin về máy trạm và tài khoản người dùng. Mặc dù ta đã sử dụng NIS như là một hệ quản trị tập trung, hệ thống này vẫn cho phép các máy trạm làm việc được quyền tự do lựa chọn sử dụng các file cấu hình cục bộ hoặc sử dụng từ NIS server. Thứ tự được chỉ ra trong file */etc/nsswitch.conf*.

Ví dụ sau cho biết thứ tự sử dụng dịch vụ của các hàm *gethostbyname(), gethostbyaddr()* và *getservbyname().* Các dịch vụ được liệt kê trước sẽ được sử dụng, nếu không thành công thì sử dụng dịch vụ sau đó.

#nsswitch.conf hosts: nis dns files services files nis

Dưới đây là danh sách các dịch vụ có thể sử dụng trong file */etc/nsswitch.conf.* Các file, chương trình cụ thể được sử dụng sẽ phụ thuộc vào từng loại dịch vụ:

- *nisplus hay nis+*: sử dụng NIS+ server cho miền NIS hiện thời. Tên của server được chỉ ra trong file /etc/nis.conf.
- *nis*: sử dụng NIS server cho domain hiện thời. Tên của server được chỉ ra trong file

- /*etc/yp.conf*. Với thành phần *hosts*, các file map là *hosts.byname* và *hosts.byaddr* sẽ được sử dụng.
- *dns*: sử dụng DNS server, dịch vụ này được sử dụng cho mình thành phần hosts. Tên của máy chủ được đặt trong file */etc/resolv.conf*.
- *files*: sử dụng các file cấu hình cục bộ, ví dụ: /etc/passwd cho thành phần passwd.
- *dbm*: tìm thông tin trong các file cơ sở dữ liệu /var/dbm. Tên của các file là tên của các file map tương ứng của dịch vụ NIS.

Các thành phần được hỗ trợ hiện thời của NYS là: hosts, networks, passwd, group, shadow, services, protocols, rpc, và một số file khác.

Nếu có từ khóa [NOTFOUND=return] trong các thành phần của file *nsswitch.conf*, NIS sẽ thoát ra ngay mà không sử dụng tiếp các dịch vụ sau trong trường hợp nó không tìm thấy thông tin ở dịch vụ trước đó. Chỉ khi nào dịch vụ trước bị lỗi, NIS mới dùng tiếp dịch vụ sau. Trong ví dụ dưới NIS chỉ sử dụng các file cục bộ khi khởi động hoặc DNS, NIS server bị hỏng.

#/ect/nsswitch.conf
hosts: nis dns [NOTFOUND=return] files network: nis [NOTFOUND=return] files services: file nis
protocol: files nis rpc: files nis

5.3.5. Sử dụng các file map passwd và group

Một trong những ứng dụng chính của NIS là đồng bộ thông tin về các tài khoản của người sẻ dụng trên tất cả các máy trạm trong miền NIS. Khi đó thông tin về người dùng trên trạm được liệt kê một phần nhỏ trong */etc/passwd*, phần còn lại được lưu trong file map *passwd.byname*. Việc chọn nis trong file */etc/nsswitch.conf* chưa đủ để NIS có thể hoạt động.

Khi sử dụng tài khoản của người dùng được cung cấp bởi NIS, trước tiên phải đảm bảo số hiệu của người dùng trong file *passwd* phải trùng với số hiệu của người dùng đó trên NIS. Nếu số hiệu người dùng, số hiệu nhóm của người dùng đó khác với thông tin trong miền NIS, ta cần sửa lại cho trùng nhau.

Trước tiên ta thay đổi số hiệu người dùng (uid) và số hiệu nhóm (gid) trong đó file */etc/passwd* và */etc/group* trên trạm cục bộ sang các giá trị mới của NIS. Sau đó đổi quyền sở hữu của tất cả các file bằng cách thay đổi số hiệu uid và gid cũ sang uid và gid mới. Giả sử người dùng anhnv có số hiệu uid là 501, thuộc nhóm sinhvien có gid là 423, ta sửa đổi quyền sở hữu như sau:

#find / -uid 501 – print > /tmp/uid/uid.501
#find / -gid 501 – print > /tmp/uid/gid.501
#cat /tmp/uid.501 | xargs chown anhnv
#cat /tmp/gid.501 | xargs chgrp sinhvien

Sau khi thực hiện các công việc trên số hiệu uid và gid của một người dùng trên máy trạm sẽ đồng nhất với NIS. Bước tiếp theo là sửa đổi file */etc/nsswitch.conf* như sau:

#/etc/nsswitch.conf
passwd: nis files group: nis files

File trên quy định lệnh login và các lệnh thuộc họ này sẽ truy vấn NIS server khi một người dùng muốn truy nhập, nếu không tìm thấy nó sẽ tìm tiếp đến các file cục bộ. Thông thường ta loại bỏ hầu hết người dùng khỏi các file cục bộ, chỉ giữ lại root hoặc các tài khoản chung như mail, news,… cho một số tác vụ cần chuyển đổi số hiệu uid sang tên và ngược lại. Ví dụ trong chương trình quản lý công việc cron sử dụng lệnh *su* để tạm trở thành news. Nếu news không có trong */etc/passwd*, chương trình trên sẽ không thực hiện được.

Khi người dùng muốn thay đổi mật khẩu, họ không thể dùng lệnh *passwd* như khi chưa có NIS. Lệnh *passwd* chỉ có tác dụng sửa đổi các file cấu hình cục bộ. NIS cung cấp một công cụ là *yppasswd*, nó không những cho phép sửa đổi mật khẩu người dùng trên NIS mà còn thay

đổi các thuộc tính khác như shell… Chương trình này được thực hiện khi khởi động hệ thống bằng cách chạy thêm dịch vụ *rpc.yppasswd*. Vì thói quen người dùng có thể gõ lệnh *passwd* khi muốn thay đổi mật khẩu. Giải pháp ở đây là thay đổi *passwd* bằng một liên kết đến *yppasswd.*

#cd /bin
#mv passwd passwd.old
#ln yppasswd passwd

5.4 Hệ thống file trên mạng

Linux có dịch vụ chia sẻ file trên mạng máy tính. Khi ta muốn có khả năng các máy Linux có thể chia sẻ tài nguyên là các file với nhau, dịch vụ NFS sẽ cung cấp khả năng này. Dịch vụ này cho phép chia sẻ file cho các người dùng trên mạng LAN, các file này có khả năng xuất hiện đối với các người dùng như là các file ở trên máy của mình.

5.4.1 Cài đặt NFS

Để cài đặt dịch vụ này ta cần chuẩn bị một package là nfs-utils-xxx.rpm trong đó xxx là số hiệu phiên bản. Đăng nhập với quyền root và sử dụng lệnh:

 # rpm –ivh nfs-utils-xxx.rpm

Nếu không có lời thông báo lỗi thì việc cài đặt đã thành công.

NFS sử dụng thủ tục RPC (Remote Procedure Calls) để gửi và nhận yêu cầu giữa các máy chủ và máy trạm trên mạng, do vậy dịch vụ ánh xạ cổng portmap (dịch vụ quản lý các yêu cầu RPC) phải được khởi động trước. Trên máy chủ NFS dự định sẽ chia sẻ các file dữ liệu phải khởi động hai dịch vụ nfs và portmap bằng lệnh:

 # service nfs start

 # service portmap start

Để NFS hoạt động thì ta cần phải khởi động các dịch vụ sau:

- Portmapper: tiến trình này không làm việc trực tiếp với NFS mà tham gia quản lý các yêu cầu RPC từ máy trạm gửi đến.
- Mountd: tiến trình này sẽ ánh xạ các file trên máy chủ tới các thư mục trên máy trạm yêu cầu. Nó sẽ huỷ bỏ ánh xạ này nếu có lệnh umount từ máy trạm.
- Nfs: là tiến trình chính thực hiện các nhiệm vụ của giao thức NFS. Nó có nhiệm vụ cung cấp cho các máy trạm các thư mục hoặc file được yêu cầu.

Ta có thể kiểm tra các thông tin về các dịch vụ NFS bằng lệnh:

#rpcinfo -p

5.4.2 Khởi động và dừng NFS

Việc khởi động dịch vụ NFS cũng khá đơn giản và đã được giới thiệu ở trên bằng cách khởi động portmap và nfs.

 # service nfs start

hoặc

#/etc/init.d/nfs start

Việc dừng (tắt) dịch vụ này cũng khá đơn giản, ta dùng lệnh sau:

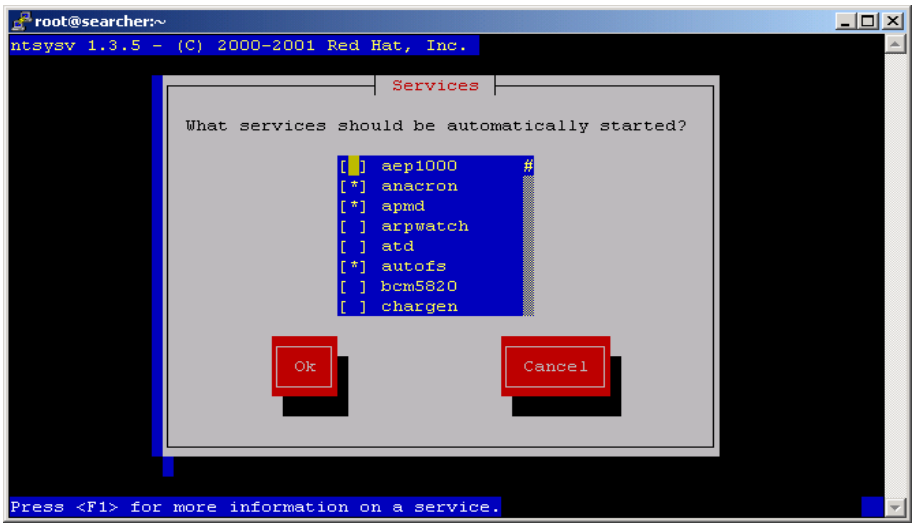
#service nfs stop

hoặc

#/etc/init.d/nfs stop

Ta có thể đặt cho dịch vụ này được tự động khởi động khi ta khởi động máy tính bằng cách dùng lệnh:

#setup



Sau đó chọn "System services", tiếp đó ta sẽ nhận được một danh sách các dịch vụ hiện đang có trong hệ thống. Muốn cho dịch vụ nào được tự động khởi động ta chỉ cần chọn dịch vụ đó, ở đây ta chọn dịch vụ có tên nfs. Chọn OK và cuối cùng chọn Quit.

5.4.3 Cấu hình NFS server và Client

Cấu hình nfs ta chỉ cần sửa file /etc/exports, đây là file chứa danh sách các thư mục được chia sẻ cho các máy khác. Nó cũng đồng thời chứa danh sách các máy trạm có quyền được truy cập và quyền truy cập của các máy trạm này. Một chú ý về định dạng của file này như sau: các dòng trống sẽ được bỏ qua, các dòng bắt đầu bằng dấu # được coi là các dòng chú thích và sẽ được bỏ qua. Các dòng dài quá ta có thể ngắt trên nhiều dòng bằng cách sử dụng dấu ngắt dòng (\).

Cấu trúc của mỗi dòng khai báo trong file này như sau:

Tên thư mục	Danh sách địa chỉ các máy trạm, quyền truy nhập của các máy trạm đó	Danh sách địa chỉ các máy trạm, quyền truy nhập của các máy trạm đó
/software/project	192.168.0.172(rw)	192.168.0.127(ro)
/software/setup	192.168.0.0/28(ro)	

Trong đó, các tham số có ý nghĩa như sau: tên thư mục là đường dẫn đến thư mục ta muốn chia sẻ cho các máy khác. Danh sách địa chỉ các máy trạm, có thể là địa chỉ IP hoặc tên máy (được liệt kê trong file /etc/hosts). Trong trường hợp muốn liệt kê danh sách các máy có địa chỉ gần kề nhau trong một khoảng nào đó ta có thể có cách viết rút gọn như sau: chẳng hạn ta muốn liệt kê các địa chỉ của máy trạm trong khoảng từ 192.168.0.0 đến các máy trạm có địa chỉ 192.168.0.15 ta chỉ cần viết 192.168.0.0/28. Quyền truy nhập được viết dưới dạng (rw) chỉ quyền đọc và ghi, còn (ro) thì các máy trạm chỉ có quyền đọc trên thư mục đó, (noaccess) cấm các máy trạm truy nhập vào các thư mục con của thư mục chia sẻ. Chú ý, giữa địa chỉ của máy và quyền truy nhập không có dấu cách.

5.4.4 Sử dụng mount

Để đọc dữ liệu trên các thư mục đã được chia sẻ này ta có thể dùng cách sau: ta sẽ dùng lệnh mount, nhưng trong trường hợp này ta phải cần quyền quản trị (root). Cú pháp của lệnh sẽ như sau:

#mount <tên_máy_chủ:/tên_thư_mục_chia_sẻ> </tên_thư_mục_cần_ánh_xạ>

Lưu ý, trước khi ra lệnh này ta phải tạo ra thư mục cần ánh xạ (trong trường hợp nó chưa tồn tại).

Ví dụ, để ánh xạ thư mục /software/project trên một máy chủ 192.168.0.33 vào thư mục /mnt/project trên máy hiện tại ta dùng lệnh sau:

#mount 192.168.0.33:/software/project /mnt/project

Bây giờ thư mục /mnt/project trên máy hiện tại sẽ bình đẳng như các thư mục khác trên máy. Ta có thể sao chép, đọc các file trên thư mục này.

5.4.5 Unmount

Sau khi thực hiện xong các thao tác cần thiết, ta có thể hủy bỏ ánh xạ này bằng lệnh umount như sau:

#umount /mnt/project

Sau lệnh này thì ta không còn có khả năng thao tác với thư mục trên máy chủ được nữa, nếu muốn ta lại phải ánh xạ lại.

Ngoài ra muốn xem trạng thái hoạt động của dịch vụ nfs ta có thể dùng lệnh:

#/etc/init.d/nfs status

Nó sẽ hiển thị thông tin về trạng thái hiện tại của dịch vụ này đang chạy hay đã dừng lại.

rpc.mountd (pid 936) is running...
nfsd (pid 948 947 946 945 944 943 942 941) is running... rpc.rquotad (pid 931) is running...

5.4.6 Mount tự động qua tệp cấu hình

Bây giờ nếu ta muốn hệ thống sẽ tự động ánh xạ thư mục này khi máy khởi động để cho những người dùng không có quyền quản trị có thể dùng được thì ta có thể sử dụng cách sửa đổi nội dung của file /etc/fstab.

Cũng tương tự như lệnh mount ở trên, trong file /etc/fstab cũng có các trường giống như đã nói ở trên. Mỗi một dòng trong file này sẽ có cấu trúc như sau:

<tên_máy_chủ:/đường_dẫn_đến_thư_mục_chia_sẻ>
</đường_dẫn_đến_thư_mục_cục_bộ> nfs

tham số nfs chỉ cho hệ điều hành biết kiểu file là nfs.

Ví dụ ta có thể thêm dòng 192.168.0.33:/software/project /mnt/project nfs vào cuối file /etc/fstab.

CÂU HỎI VÀ BÀI TẬP

1. Trình bày các lệnh cấu hình card giao tiếp mạng trên Linux
2. Trình bày quy trình cài đặt và cấu hình cho máy chủ NIC.
3. Trình bày quy trình cài đặt và cấu hình các máy trạm NIC
4. Thực hành các thao tác liên quan đến hệ thống mạng trên Linux

Chương 6. LẬP TRÌNH SHELL VÀ LẬP TRÌNH C TRÊN LINUX

6.1. Cách thức pipes và các yếu tố cơ bản lập trình trên shell

6.1.1. Cách thức pipes

Trong Linux có một số loại shell, shell ngầm định là bash. Shell cho phép người dùng chạy từng lệnh shell (thực hiện trực tiếp) hoặc dãy lệnh shell (file script) và đặc biệt hơn là theo dạng thông qua ống dẫn (pipe).

- Trong một dòng lệnh của shell có thể thực hiện một danh sách các lệnh tuần tự nhau dạng:

<lệnh> [; <lệnh>]...

Như vậy danh sách lệnh là dãy các lệnh liên tiếp nhau, cái sau cách cái trước bởi dấu chấm phảy ";"

Ví dụ: ***$ cal 10 1999; cal 11 1999 ; cal 12 1999***

Shell cho người dùng cách thức đặc biệt thực hiện các lệnh tuần tự nhau, cái ra của lệnh trước là cái vào của lệnh sau và không phải thông qua nơi lưu trữ trung gian.

- Sử dụng ống dẫn là cách thức đặc biệt trong UNIX và Linux, được thể hiện là một cách thức của shell để truyền thông liên tiến trình. ống dẫn được tổ chức theo kiểu cấu trúc dữ liệu dòng xếp hàng "vào trước ra trước" FIFO "First In First Out".

Mô tả cách thức sử dụng đường ống trong shell như sau:

<lệnh phức hợp> là hoặc <lệnh> hoặc (<lệnh>[;<lệnh>]...)

Vậy đường ống có dạng

<lệnh phức hợp> | <lệnh phức hợp>

Lệnh phức hợp phía sau có thể không có đối số. Trong trường hợp đó, thông tin kết quả từ lệnh phía trước trở thành thông tin input của lệnh ngay phía sau mà không chịu tác động theo cách thông thường của lệnh trước nữa.

Ví dụ: ***$ cal 1999 | more***

Nội dung lịch năm 1999 (lệnh ***cal*** đóng vai trò tiến trình A) không được in ngay ra màn hình như thông thường theo tác động của lệnh ***cal*** nữa mà được lưu lên một "file" tạm thời kiểu "ống dẫn" của hệ thống và sau đó trở thành đối số của lệnh ***more*** (lệnh ***more*** đóng vai trò tiến trình B).

Trong chương trình, có thể dùng ống dẫn làm file vào chuẩn cho các lệnh đọc tiếp theo.

Ví dụ: ***ls -L | ***

thì ký hiệu "\" chỉ ra rằng ống dẫn được dùng như file vào chuẩn.

6.1.2. Các yếu tố cơ bản để lập trình trong shell

Shell có công cụ cho phép có thể lập trình trên shell làm tăng thêm độ thân thiện khi giao tiếp với người dùng. Các đối tượng tham gia công cụ như thế có thể được liệt kê:

- Các biến (trong đó chú ý tới các biến chuẩn),
- Các hàm vào - ra
- Các phép toán số học,
- Biểu thức điều kiện,
- Cấu trúc rẽ nhánh,
- Cấu trúc lặp.

** Một số nội dung trong chương trình shell*

- Chương trình là dãy các dòng lệnh shell song được đặt trong một file văn bản (được soạn thảo theo soạn thảo văn bản),

- Các dòng lệnh bắt đầu bằng dấu # chính là dòng chú thích, bị bỏ qua khi shell thực hiện chương trình,

- Thông thường các bộ dịch lệnh shell là sh (/bin/sh) hoặc ksh (/bin/ksh)

Để thực hiện một chương trình shell ta có các cách sau đây:

$sh <<tên chương trình>

hoặc

$sh <tên chương trình>

hoặc nhờ đổi mod của chương trình:

$chmod u+x <tên chương trình>

và chạy chương trình

$<tên chương trình>

- Phần lớn các yếu tố ngôn ngữ trong lập trình shell là tương đồng với lập trình C.

** Các biến trong file script*

Trong shell có thể kể tới 3 loại biến:

- Biến môi trường (biến shell đặc biệt, biến từ khóa, biến shell xác định trước hoặc biến shell chuẩn) được liệt kê như sau (các biến này thường gồm các chữ cái hoa):

HOME : đường dẫn thư mục riêng của người dùng,
MAIL: đường dẫn thư mục chứa hộp thư người dùng,
PATH: thư mục dùng để tìm các file thể hiện nội dung lệnh,
PS1: dấu mời ban đầu của shell (ngầm định là $),
PS2: dấu mời thứ 2 của shell (ngầm định là >),
PWD: Thư mục hiện tại người dùng đang làm,
SHELL: Đường dẫn của shell (/bin/sh hoặc /bin/ksh)
TERM: Số hiệu gán cho trạm cuối,
USER: Tên người dùng đã vào hệ thống,

Trong ***.profile*** ở thư mục riêng của mỗi người dùng thường có các câu lệnh dạng:

<biến môi trường> = <giá trị>

- Biến người dùng: Các biến này do người dùng đặt tên và có các cánh thức nhận giá trị các biến người dùng từ bàn phím (lệnh read).

Biến được đặt tên gồm một xâu ký tự, quy tắc đặt tên như sau: ký tự đầu tiên phải là một chữ cái hoặc dấu gạch chân (_), sau tên là một hay nhiều ký tự khác. Để tạo ra một biến ta chỉ cần gán biến đó một giá trị nào đó. Phép gán là một dấu bằng (=). Ví dụ:

myname="TriThanh"

Chú ý: không được có dấu cách (space) đằng trước hay đằng sau dấu bằng. Tên biến là phân biệt chữ hoa chữ thường. Để truy xuất đến một biến ta dùng cú pháp sau; $tên_biến. Chẳng hạn ta muốn in ra giá trị của biến myname ở trên ta chỉ cần ra lệnh: ***echo $myname.***

Một số ví dụ về cách đặt tên biến:

$ no=10 # đây là một cách khai báo hợp lệ

Nhưng cách khai báo dưới đây là không hợp lệ

$ no =10 # có dấu cách sau tên biến
$ no= 10 # có dấu cách sau dấu =
$ no = 10 # có dấu cách cả đằng trước lẫn đằng sau dấu =

Ta có thể khai báo một biến nhưng nó có giá trị NULL như trong những cách sau:

$ vech=
$ vech=""

Nếu ta ra lệnh in giá trị của biến này thì ta sẽ thu được một giá trị NULL ra màn hình (một dòng trống).

- Biến tự động (hay biến-chỉ đọc, tham số vị trí) là các biến do shell đã có sẵn; tên các biến này cho trước. Có 10 biến tự động:

$0, $1, $2,..., $9

Tham biến "$0" chứa tên của lệnh, các tham biến thực bắt đầu bằng "$1" (nếu tham số có vị trí lớn hơn 9, ta phải sử dụng cú pháp ${} – ví dụ, ${10} để thu được các giá trị của chúng). Shell ***bash*** có ba tham biến vị trí đặc biệt, "$#", "$@", và "$#". "$#" là số lượng tham biến vị trí (không tính "$0"). "$*" là một danh sách tất cả các tham biến vị trí loại trừ "$0", đã được định dạng như là một xâu đơn với mỗi tham biến được phân cách bởi ký tự $IFS. "$@" trả về tất cả các tham biến vị trí được đưa ra dưới dạng N xâu được bao trong dấu ngoặc kép.

Sự khác nhau giữa "$*" và "$@" là gì và tại sao lại có sự phân biệt? Sự khác nhau cho phép ta xử lý các đối số dòng lệnh bằng hai cách. Cách thứ nhất, "$*", do nó là một xâu đơn, nên có thể được biểu diễn linh hoạt hơn không cần yêu cầu nhiều mã shell. "$@" cho phép ta xử lý mỗi đối số riêng biệt bởi vì giá trị của chúng là N đối số độc lập.

Dòng ra (hay dòng vào) tương ứng với các tham số vị trí là các "từ" có trong các dòng đó.

Ví dụ: *$chay vao chuong trinh roi*

Nếu chay là một lệnh thì dòng vào này thì:

$0 có giá trị chay $1 có giá trị vao $2 có giá trị chuong
$3 có giá trị trinh $4 có giá trị roi

Một ví dụ khác về biến vị trí giúp ta phân biệt được sự khác nhau giữa biến $* và $@:

```
#!/bin/bash
#testparm.sh
function cntparm
{
        echo –e "inside cntparm $# parms: $*"
}
cntparm '$*'
cntparm '$@'
echo –e "outside cntparm    $*    parms\n"
echo –e "outside cntparm    $@    parms\n"
```

Khi chạy chương trình này ta sẽ thu được kết quả:

```
$./testparm.sh Kurt Roland Wall
inside cntparm 1 parms: Kurt Roland Wall inside cntparm 3 parms: Kurt Roland Wall
outside cntparm: Kurt Roland Wall
outside cntparm: Kurt Roland Wall
```

Trong dòng thứ nhất và thứ 2 ta thấy kết quả có sự khác nhau, ở dòng thứ nhất biến "$*" trả về tham biến vị trí dưới dạng một xâu đơn, vì thế cntparm báo cáo một tham biến đơn. Dòng thứ hai gọi cntparm, trả về đối số dòng lệnh của là 3 xâu độc lập, vì thế cntparm báo cáo ba tham biến.

7.1.2.3. Các ký tự đặc biệt trong bash

Ký tự	**Mô tả**	**Ký tự**	**Mô tả**
<	Định hướng đầu vào	~	Thư mục home của user hiện tại
>	Định hướng đầu ra	`	Thay thế lệnh
(	Bắt đầu subshell	;	Chia cắt lệnh
)	Kết thúc subshell	#	Lời chú giải

Ký tự	Mô tả	Ký tự	Mô tả
|	Ký hiệu dẫn	‘	Trích dẫn mạnh
\	Dùng để hiện ký tự đặc biệt	“	Trích dẫn yếu
&	Thi hành lệnh chạy ở chế độ ngầm	$	Biểu thức biến
{	Bắt đầu khối lệnh	*	Ký tự đại diện cho chuỗi
}	Kết thúc khối lệnh	?	Ký tự đại diện cho một ký tự

Các ký tự đặc biệt của bash

Dấu chia cắt lệnh, ; , cho phép thực hiện những lệnh bash phức tạp đánh trên một dòng. Nhưng quan trọng hơn, nó là kết thúc lệnh theo lý thuyết POSIX.

Ký tự chú giải, # , khiến bash bỏ qua mọi ký tự từ đó cho đến hết dòng. điểm khác nhau giữa các ký tự trích dẫn mạnh và trích dẫn yếu, ‘ và “, tương ứng là: trích dẫn mạnh bắt bash hiểu tất cả các ký tự theo nghĩa đen; trích dẫn yếu chỉ bảo hộ cho một vài ký tự đặc biệt của bash .

6.2. Một số lệnh lập trình trên shell

6.2.1. Sử dụng các toán tử bash

Các toán tử string

Các toán tử string, cũng được gọi là các toán tử thay thế trong tài liệu về bash, kiểm tra giá trị của biến là chưa gán giá trị hoặc không xác định. Bảng dưới là danh sách các toán tử này cùng với miêu tả cụ thể cho chức năng của từng toán tử.

Toán tử	Chức năng
${var:- word}	Nếu biến tồn tại và xác định thì trả về giá trị của nó, nếu không thì trả về word
${var:= word}	Nếu biến tồn tại và xác định thì trả về giá trị của nó, nếu không thì gán biến thành word, sau đó trả về giá trị của nó
${var:+ word}	Nếu biến tồn tại và xác định thì trả về word, còn không thì trả về null
${var:?message}	Nếu biến tồn tại và xác định thì trả về giá trị của nó, còn không thì hiển thị “bash: $var:$message” và thoát ra khỏi lệnh hay tập lệnh hiện thời.
${var: offset[:length]}	Trả về một xâu con của var bắt đầu tại offset của độ dài length. Nếu length bị bỏ qua, toàn bộ xâu từ offset sẽ được trả về.

Các toán tử string của bash

Để minh hoạ, hãy xem xét một biến shell có tên là status được khởi tạo với giá trị defined. Sử dụng 4 toán tử string đầu tiên cho kết quả status như sau:

$echo ${status:-undefined}
defined
$echo ${status:=undefined}
defined
$echo ${status:+undefined}
undefined
$echo ${status:?Dohhh\! undefined}
defined

Bây giờ sử dụng lệnh unset để xoá biến status, và thực hiện vẫn các lệnh đó, được output như sau:

$unset status
$echo ${status:-undefined}
undefined
$echo ${status:=undefined}
undefined
$echo ${status:+undefined}
undefined
$unset status
$echo ${status:?Dohhh\! undefined}
bash:status Dohhh! Undefined

Cần thiết unset status lần thứ hai vì ở lệnh thứ ba, echo ${status:+undefined}, khởi tạo lại status thành undefined.

Các toán tử substring đã có trong danh sách ở bảng trên đặc biệt có ích. Hãy xét biến foo có giá trị Bilbo_the_Hobbit. Biểu thức ${foo:7} trả về he_Hobbit, trong khi ${foo:7:5} lại trả về he_Ho.

Các toán tử Pattern-Matching

Các toán tử pattern-matching có ích nhất trong công việc với các bản ghi độ dài biến hay các xâu đã được định dạng tự do được định giới bởi các ký tự cố định. Biến môi trường $PATH là một ví dụ. Mặc dù nó có thể khá dài, các thư mục riêng biệt được phân định bởi dấu hai chấm. Bảng dưới là danh sách các toán tử Pattern-Matching của bash và chức năng của chúng.

Toán tử	Chức năng
${var#pattern}	Xoá bỏ phần khớp (match) ngắn nhất của pattern trước var và trả về phần còn lại
${var##pattern}	Xoá bỏ phần khớp (match) dài nhất của pattern trước var và trả về phần còn lại
${var%pattern}	Xoá bỏ phần khớp ngắn nhất của pattern ở cuối var và trả về phần còn lại
${var%%pattern}	Xoá bỏ phần khớp dài nhất của pattern ở cuối var và trả về phần còn lại
${var/pattern/string}	Thay phần khớp dài nhất của pattern trong var bằng string. Chỉ thay phần khớp đầu tiên. Toán tử này chỉ có trong bash 2.0 hay lớn hơn.
${var//pattern/string}	Thay phần khớp dài nhất của pattern trong var bằng string. Thay tất cả các phần khớp. Toán tử này có trong bash 2.0 hoặc lớn hơn.

Các toán tử bash Pattern-Matching

Thông thường quy tắc chuẩn của các toán tử bash pattern-matching là thao tác với file và tên đường dẫn. Ví dụ, giả sử ta có một tên biến shell là mylife có giá trị là /usr/src/linux/Documentation/ide.txt (tài liệu về trình điều khiển đĩa IDE của nhân). Sử dụng mẫu "/*" và "*/" ta có thể tách được tên thư mục và tên file.

```
#!/bin/bash
################################################
myfile=/usr/src/linux/Documentation/ide.txt echo '${myfile##*/}=' ${myfile##*/}
echo 'basename $myfile =' $(basename $myfile)
echo '${myfile%/*}=' ${myfile%/*}
echo 'dirname $myfile =' $(dirname $myfile)
```

Lệnh thứ 2 xoá xâu matching "*/" dài nhất trong tên file và trả về tên file. Lệnh thứ 4 làm khớp tất cả mọi thứ sau "/", bắt đầu từ cuối biến, bỏ tên file và trả về đường dẫn của file. Kết quả của tập lệnh này là:

$ ./pattern.sh
${myfile##/} = ide.txt basename $myfile = ide.txt*
${myfile%/} = /usr/src/linux/Documentation dirname $myfile =*
/usr/src/linux/Documentation

Để minh hoạ về các toán tử pattern-matching và thay thế, lệnh thay thế mỗi dấu hai chấm trong biến môi trường $PATH bằng một dòng mới, kết quả hiển thị đường dẫn rất dễ đọc (ví dụ này sẽ sai nếu ta không có bash phiên bản 2.0 hoặc mới hơn):

$ echo –e ${PATH//:/\\n}
/usr/local/bin
/bin
/usr/bin
/usr/X11R6/bin
/home/kwall/bin
/home/wall/wp/wpbin

Các toán tử so sánh chuỗi

- str1 = str2 : str1 bằng str2
- str1 != str2 : str1 khác str2
- -n str : str có độ dài lớn hơn 0 (khác null)
- -z str : str có độ dài bằng 0 (null)

Các toán tử so sánh số học

- -eq : bằng
- -ge : lớn hơn hoặc bằng
- -gt : lớn hơn
- -le : nhỏ hơn hoặc bằng
- -lt : nhỏ hơn
- -ne : khác

6.2.2. Điều khiển luồng

Các cấu trúc điều khiển luồng của bash, nó bao gồm:

- if – Thi hành một hoặc nhiều câu lệnh nếu có điều kiện là true hoặc false.
- for – Thi hành một hoặc nhiều câu lệnh trong một số cố định lần.
- while – Thi hành một hoặc nhiều câu lệnh trong khi một điều kiện nào đó là true hoặc false.
- until – Thi hành một hoặc nhiều câu lệnh cho đến khi một điều kiện nào đó trở thành true hoặc false.
- case – Thi hành một hoặc nhiều câu lệnh phụ thuộc vào giá trị của biến.
- select – Thi hành một hoặc nhiều câu lệnh dựa trên một khoảng tuỳ chọn của người dùng.

** Cấu trúc rẽ nhánh có điều kiện if*

Bash cung cấp sự thực hiện có điều kiện lệnh nào đó sử dụng câu lệnh if, câu lệnh if của bash đầy đủ chức năng như của C. Cú pháp của nó được khái quát như sau:

if condition then
statements
[elif condition statements]
[else

statements]

fi

Đầu tiên, ta cần phải chắc chắn rằng mình hiểu if kiểm tra trạng thái thoát của câu lệnh last trong condition. Nếu nó là 0 (true), sau đó statements sẽ được thi hành, nhưng nếu nó khác 0, thì mệnh đề else sẽ được thi hành và điều khiển nhảy tới dòng đầu tiên của mã fi. Các mệnh đề elif (tuỳ chọn) (có thể nhiều tuỳ ý) sẽ chỉ thi hành khi điều kiện if là false. Tương tự, mệnh đề else (tuỳ chọn) sẽ chỉ thi hành khi tất cả else không thỏa mãn. Nhìn chung, các chương trình Linux trả về 0 nếu thành công hay hoàn toàn bình thường, và khác 0 nếu ngược lại, vì thế không có hạn chế nào cả.

Chú ý: Không phải tất cả chương trình đều tuân theo cùng một chuẩn cho giá trị trả về, vì thế cần kiểm tra tài liệu về các chương trình ta kiểm tra mã thoát với điều kiện if. Ví dụ chương trình diff, trả về 0 nếu không có gì khác nhau, 1 nếu có sự khác biệt và 2 nếu có vấn đề nào đó. Nếu một câu điều kiện hoạt động không như mong đợi thì hãy kiểm tra tài liệu về mã thoát .

Không quan tâm đến cách mà chương trình xác định mã thoát của chúng, bash lấy 0 có nghĩa là true hoặc bình thường còn khác 0 là false. Nếu ta cần cụ thể để kiểm tra một mã thoát của lệnh, sử dụng toán tử ***$?*** ngay sau khi chạy lệnh. ***$?*** trả về mã thoát của lệnh chạy ngay lúc đó.

Phức tạp hơn, bash cho phép ta phối hợp các mã thoát trong phần điều kiện sử dụng các toán tử && và || được gọi là toán tử logic AND và OR. Cú pháp đầy đủ cho toán tử AND như sau:

command1 && command2

Câu lệnh ***command2*** chỉ được chạy khi và chỉ khi ***command1*** trả về trạng thái là số 0 (true).

Cú pháp cho toán tử OR thì như sau:

command1 || command2

Câu lệnh ***command2*** chỉ được chạy khi và chỉ khi ***command1*** trả lại một giá trị khác 0 (false).

Ta có thể kết hợp lại cả 2 loại toán tử lại để có một biểu thức như sau:

command1 && comamnd2 || command3

Nếu câu lệnh ***command1*** chạy thành công thì shell sẽ chạy lệnh ***command2*** và nếu ***command1*** không chạy thành công thì ***command3*** được chạy.

Ví dụ:

$ rm myf && echo "File is removed successfully" || echo "File is not removed"

Nếu file myf được xóa thành công (giá trị trả về của lệnh là 0) thì lệnh "*echo File is removed successfully*" sẽ được thực hiện, nếu không thì lệnh "*echo File is not removed*" được chạy.

Giả sử trước khi ta vào trong một khối mã, ta phải thay đổi một thư mục và copy một file. Có một cách để thực hiện điều này là sử dụng các toán tử ***if*** lồng nhau, như là đoạn mã sau:

```
if cd /home/kwall/data then
        if cp datafile datafile.bak then
                # more code here
        fi
fi
```

Tuy nhiên, bash cho phép ta viết đoạn mã này súc tích hơn nhiều như sau:

```
if cd /home/kwall/data && cp datafile datafile.bak then
# more code here fi
```

Cả hai đoạn mã đều thực hiện cùng một chức năng, nhưng đoạn thứ hai ngắn hơn nhiều, gọn nhẹ và đơn giản. Mặc dù ***if*** chỉ kiểm tra các mã thoát, ta có thể sử dụng cấu trúc ***[…]*** lệnh ***test*** để kiểm tra các điều kiện phức tạp hơn. ***[condition]*** trả về giá trị biểu thị condition là true hay false. ***test*** cũng có tác dụng tương tự.

Một ví dụ khác về cách sử dụng cấu trúc ***if:***

```
#!/bin/sh
# Script to test if..elif...else
#
if [ $1 -gt 0 ]; then echo "$1 is positive"
elif [ $1 -lt 0 ]
then
        echo "$1 is negative"
elif [ $1 -eq 0 ]
then
        echo "$1 is zero"
else
        echo "Opps! $1 is not number, give number"
fi
```

Số lượng các phép toán điều kiện của biến hiện tại khoảng 35, khá nhiều và hoàn chỉnh.

Ta có thể kiểm tra các thuộc tính file, so sánh các xâu và các biểu thức số học.

Chú ý: Các khoảng trống trước dấu mở ngoặc và sau dấu đóng ngoặc trong ***[condition]*** là cần phải có. Đây là điều kiện cần thiết trong cú pháp shell của bash.

Danh sách các toán tử ***test*** file phổ biến nhất:

Toán tử	Điều kiện true
-d file	file tồn tại và là một thư mục
-e file	file tồn tại
-f file	file tồn tại và là một file bình thường (không là một thư mục hay một file đặc biệt)
-r file	file cho phép đọc
-s file	file tồn tại và khác rỗng
-w file	file cho phép ghi
-x file	file khả thi hoặc nếu file là một thư mục thì cho phép tìm kiếm trên file
-O file	file của người dùng hiện tại
-G file	file thuộc một trong các nhóm người dùng hiện tại là thành viên
file1 -nt file2	file1 mới hơn file2
file1 -ot file2	file1 cũ hơn file2

Ví dụ chương trình shell cho các toán tử ***test*** file trên các thư mục trong biến $PATH. Mã cho chương trình descpath.sh như sau:

```
#!/bin/bash
##################################
IFS=:
for dir in $PATH;
do
        echo $dir
        if [ -w $dir ]; then
                echo -e "\tYou have write permission in $dir"
```

```
        else
                echo –e "\tYou don't have write permission in $dir"
        fi
        if [ -0 $dir ]; then
                echo -e "\tYou own $dir"
        else
                echo –e "\tYou don't own $dir"
        fi
        if [ -G $dir ]; then
                echo -e "\tYou are a member of $dir's group"
        else
                echo -e "\tYou aren't a member of $dir's group"
        fi
done Chương trình descpath.sh
```

Vòng lặp for (giới thiệu trong phần dưới) sẽ duyệt toàn bộ các đường dẫn thư mục trong biến PATH sau đó kiểm tra các thuộc tính của thư mục đó. Kết quả như sau (kết quả có thể khác nhau trên các máy khác nhau do giá trị của biến PATH khác nhau):

/usr/local/bin
You don't have write permission in /usr/local/bin
You don't own /usr/local/bin
You aren't a member of /usr/local/bin's group
/bin
You don't have write permission in /bin
You don't own /bin
You aren't a member of /bin's group
/usr/bin
You don't have write permission in /usr/bin
You don't own /usr/bin
You aren't a member of /usr/bin's group
/usr/X11R6/bin
You don't have write permission in /usr/X11R6/bin
You don't own /usr/X11R6/bin
You aren't a member of /usr/X11R6/bin's group
/home/kwall/bin
You have write permission in /home/kwall/bin
You own /home/kwall/bin
You are a member of /home/kwall/bin's group
/home/kwall/wp/wpbin
You have write permission in /home/kwall/wp/wpbin
You own /home/kwall/wp/wpbin
You are a member of /home/kwall/wp/wpbin's group

Các biểu thức trong phần điều kiện cũng có thể kết hợp với nhau tạo thành các biểu thức phức tạp hơn bằng các phép toán logic.

Toán tử	**Ý nghĩa**
! expression	Logical NOT
expression1 -a expression2	Logical AND
expression1 -o expression2	Logical OR

** Các vòng lặp đã quyết định: for*

Như đã thấy ở chương trình trên, ***for*** cho phép ta chạy một đoạn mã một số lần nhất định. Tuy nhiên cấu trúc ***for*** của bash chỉ cho phép ta lặp đi lặp lại trong danh sách các giá trị

nhất định bởi vì nó không tự động tăng hay giảm con đếm vòng lặp như là C, Pascal, hay Basic. Tuy nhiên vòng lặp ***for*** là công cụ lặp thường xuyên được sử dụng bởi vì nó điều khiển gọn gàng trên các danh sách, như là các tham số dòng lệnh và các danh sách các file trong thư mục.

Cú pháp đầy đủ của for là:

for value in list
do
done
statements using $value

list là một danh sách các giá trị, ví dụ như là tên file. Giá trị là một thành viên danh sách đơn và ***statements*** là các lệnh sử dụng value. Một cú pháp khác của lệnh ***for*** có dạng như sau:

for ((expr1; expr2; expr3))
do
…....
repeat all statements between do and done until expr2 is TRUE
done

Linux không có tiện ích để đổi tên hay copy các nhóm của file. Trong MS-DOS nếu ta có 17 file có phần mở rộng a*.doc, ta có thể sử dụng lệnh COPY để copy *.doc thành file *.txt. Lệnh DOS như sau:

C:\ cp doc*.doc doc*.txt

sử dụng vòng lặp for của bash để bù đắp những thiếu sót này. Đoạn mã dưới đây có thể được chuyển thành chương trình shell thực hiện đúng như những gì ta muốn:

```
for docfile in doc/*.doc do
  cp $docfile ${docfile%.doc}.txt
done
```

Sử dụng một trong các toán tử pattern-matching của bash, đoạn mã này làm việc copy các file có phần mở rộng là *.doc bằng cách thay thế .doc ở cuối của tên file bằng .txt.

Một ví dụ khác về vòng for đơn giản như sau:

```
#!/bin/bash
for i in 1 2 3 4 5
do
        echo "Welcome $i times"
done
```

Ta cũng có một cấu trúc về ***for*** như sau, chương trình này cũng có cùng chức năng như chương trình trên nhưng ta chú ý đến sự khác biệt về cú pháp của lệnh ***for.***

```
#!/bin/bash
for (( i = 0 ; i <= 5; i++ ))
do
 echo "Welcome $i times"
done
```

$ chmod +x for2

$ ./for2

```
Welcome 0 times
Welcome 1 times Welcome 2 times Welcome 3 times
Welcome 4 times
Welcome 5 times
```

Tiếp theo là một ví dụ về vòng ***for*** lồng nhau:

```
#!/bin/bash
for (( i = 1; i <= 5; i++ ))     ### Outer for loop ###
```

```
do
        for (( j = 1 ; j <= 5; j++ )) ### Inner for loop ###
        do
                echo -n "$i "
        done
done
```

Ví dụ khác về cách sử dụng cấu trúc ***if*** và ***for*** như sau:

```
#!/bin/sh
#Script to test for loop
#
#
if [ $# -eq 0 ]
then
        echo "Error - Number missing form command line argument"
        echo "Syntax : $0 number"
        echo "Use to print multiplication table for given number"
        exit 1 fi n=$1
for i in 1 2 3 4 5 6 7 8 9 10
do
        echo "$n * $i = `expr $i \* $n`"
done
```

Khi ta chạy chương trình với tham số:

$ chmod 755 mtable

$ ./mtable 7

Ta thu được kết quả như sau:

```
7 * 1 = 7
7 * 2 = 14
...
7 * 10 = 70
```

** Các vòng lặp không xác định: while và until*

Vòng lặp ***for*** giới hạn số lần mà một đoạn mã được thi hành, các cấu trúc ***while*** và ***until*** của bash cho phép một đoạn mã được thi hành liên tục cho đến khi một điều kiện nào đó xảy ra. Chỉ với chú ý là đoạn mã này cần viết sao cho điều kiện cuối phải xảy ra nếu không

sẽ tạo ra một vòng lặp vô tận. Cú pháp của nó như sau:

while condition do

statements

done

Cú pháp này có nghĩa là khi nào condition còn true, thì thực hiện statements cho đến khi condition trở thành false (cho đến khi một chương trình hay một lệnh trả về khác 0):

until condition do

statements

done

Cú pháp ***until*** có nghĩa là trái ngược với ***while***: cho đến khi condition trở thành true thì thi hành statements (có nghĩa là cho đến khi một lệnh hay chương trình trả về mã thoát khác 0)

Cấu trúc ***while*** của bash khắc phục thiếu sót không thể tự động tăng, giảm con đếm cua vòng lặp for.

Ví dụ, ta muốn copy 150 bản của một file, thì vòng lặp while là một lựa chọn để giải quyết bài toán này.

```
#!/bin/sh
#
declare -i idx idx=1
while [ $idx != 150]
do
        cp somefile somefile.$idx idx=$idx+1
done
```

Chương trình này giới thiệu cách sử dụng tính toán số nguyên của bash. Câu lệnh ***declare*** khởi tạo một biến, idx, định nghĩa là một số nguyên. Mỗi lần lặp idx tăng lên, nó sẽ được kiểm tra để thoát khỏi vòng lặp. Vòng lặp until tuy cũng có khả năng giống while nhưng không được dùng nhiều vì rất khó viết và chạy chậm.

Một ví dụ nữa về cách sử dụng vòng lặp while được minh họa trong chương trình in bản nhân của một số:

```
#!/bin/sh
#Script to test while statement
#
if [ $# -eq 0 ]
then
        echo "Error - Number missing form command line argument"
        echo "Syntax : $0 number"
        echo " Use to print multiplication table for given number"
        exit 1
fi
n=$1
i=1
while [ $i -le 10 ]
do
        echo "$n * $i = `expr $i \* $n`"
        i=`expr $i + 1`
done
```

** Các cấu trúc lựa chọn: case và select*

Cấu trúc điều khiển luồng tiếp theo là ***case***, hoạt động cũng tương tự như lệnh switch của C. Nó cho phép ta thực hiện các khối lệnh phụ thuộc vào giá trị của biến. Cú pháp đầy đủ của ***case*** như sau:

case expr in pattern1)

statements ;; pattern2) statements ;;

…

[*)

esac

statements ;;]

expr được đem đi so sánh với từng pattern, nếu nó bằng nhau thì các lệnh tương ứng sẽ được thi hành. Dấu ;; là tương đương với lệnh break của C, tạo ra điều khiển nhảy tới dòng đầu tiên của mã ***esac***. Không như từ khoá ***switch*** của C, lệnh case của bash cho phép ta kiểm tra giá trị của ***expr*** dựa vào pattern, nó có thể chứa các ký tự đại diện. Cách làm việc của cấu trúc case như sau: nó sẽ khớp (match) biểu thức expr với các mẫu pattern1, pattern2,…nếu có một mẫu nào đó khớp thì khối lệnh tương ứng với mẫu đó sẽ được thực thi, sau đó nó thoát ra khỏi lệnh case. Nếu tất cả các mẫu đều không khớp và ta có sử dụng mẫu * (trong nhánh ****)***),

ta thấy đây là mẫu có thể khớp với bất kỳ giá trị nào (ký tự đại diện là *), nên các lệnh trong nhánh này sẽ được thực hiện.

Cấu trúc điều khiển ***select*** (không có trong các phiên bản bash nhỏ hơn 1.14) chỉ riêng có trong Korn và các shell bash. Thêm vào đó, nó không có sự tương tự như trong các ngôn ngữ lập trình quy ước. ***select*** cho phép ta dễ dàng trong việc xây dựng các menu đơn giản và đáp ứng các chọn lựa của người dùng. Cú pháp của nó như sau:

select value [in list]

do

statements that manipulate $value

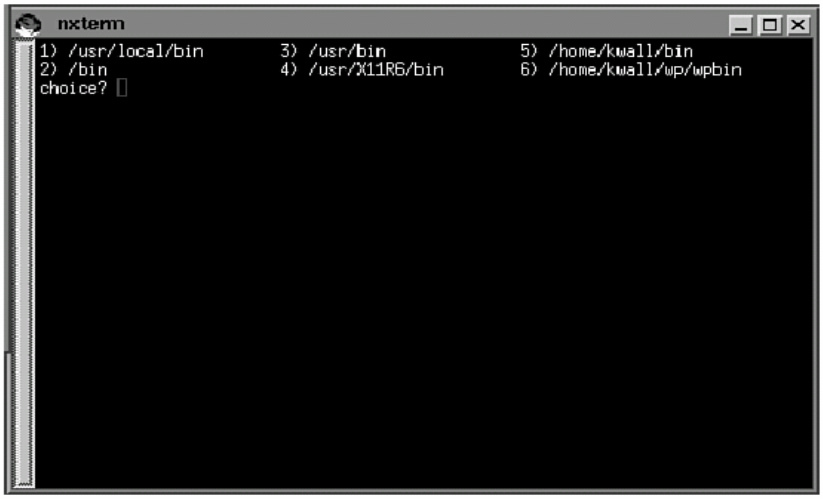
done

Dưới đây là một ví dụ về cách sử dụng lệnh select:

```
#!/bin/bash
# menu.sh – Createing simple menus with select
##########################################
IFS=: PS3="choice? "
# clear the screen clear
select dir in $PATH
do
        if [ $dir ]; then
                cnt=$(ls –Al $dir | wc -l)
                echo "$cnt files in $dir"
        else
                echo "Dohhh! No such choice!"
        fi
        echo –e "\nPress ENTER to continue, CTRL –C to quit"
        read clear
done
```

Chương trình tạo các menu bằng select

Lệnh đầu tiên đặt ký tự IFS là : (ký tự phân cách), vì thế select có thể phân tích hoàn chỉnh biến môi trường $PATH. Sau đó nó thay đổi lời nhắc default khi ***select*** bằng biến PS3. Sau khi xoá sạch màn hình, nó bước vào một vòng lặp, đưa ra một danh sách các thư mục nằm trong $PATH và nhắc người dùng chọn lựa như là minh hoạ trong hình dưới.



Nếu người dùng chọn hợp lệ, lệnh ***ls*** được thực hiện kết quả được gửi cho lệnh đếm từ ***wc*** để đếm số file trong thư mục và hiển thị kết quả có bao nhiêu file trong thư mục đó. Do ***ls*** có thể sử dụng mà không cần đối số, script đầu tiên cần chắc chắn là $dir khác null (nếu nó là null, ls sẽ hoạt động trên thư mục hiện hành nếu người dùng chọn 1 menu không hợp lệ). Nếu

người dùng chọn không hợp lệ, một thông báo lỗi sẽ được hiển thị. Câu lệnh ***read*** (được giới thiệu sau) cho phép người dùng đánh vào lựa chọn của mình và nhấn Enter để lặp lại vòng lặp hay nhấn Ctrl + C để thoát.

Chú ý: Như đã giới thiệu, các vòng lặp script không kết thuc nếu ta không nhấn Ctrl+C. Tuy nhiên ta có thể sử dụng lệnh break để thoát ra.

6.2.3 Các hàm shell

Các hàm chức năng của bash là một cách mở rộng các tiện ích sẵn có trong shell, nó có các điểm lợi sau:

- Thi hành nhanh hơn do các hàm shell luôn thường trực trong bộ nhớ.
- Cho phép việc lập trình trở nên dễ dàng hơn vì ta có thể tổ chức chương trình thành các module.

Ta có thể định nghĩa các hàm shell sử dụng theo hai cách:

```
function fname
{
        commands
}
```

Hoặc là

```
fname()
{
        commands
}
```

Cả hai dạng đều được chập nhận và không có gì khác giữa chúng. Để gọi một hàm đã định nghĩa đơn giản là gọi tên hàm cùng với các đối số mà nó cần.

Nếu so sánh với C hay Pascal, hàm của bash không được chặt chẽ, nó không kiểm tra lỗi và không có phương thức trả về đối số bằng giá trị. Tuy nhiên giống như C và Pascal, các biến địa phương có thể khai báo cục bộ đối với hàm, do đó tránh được sự xung đột với biến toàn cục. Để thực hiện điều này ta dùng từ khoá local như trong đoạn mã sau:

```
Function foo
{
        local myvar
        local yourvar=1
}
```

Trong ví dụ về các biến vị trí ở trên ta cũng thấy được cách sử dụng hàm trong bash.

Các hàm shell giúp mã của ta dễ hiểu và dễ bảo dưỡng. Sử dụng các hàm và các chú thích ta sẽ đỡ rất nhiều công sức khi ta phải trở lại nâng cấp đoạn mã mà ta đã viết từ thời gian rất lâu trước đó.

6.2.4. Các toán tử định hướng vào ra

Ta đã được biết về các toán tử định hướng vào ra, > và <. Toán tử định hướng ra cho phép ta gửi kết quả ra của một lệnh vào một file. Ví dụ như lệnh sau:

```
$ cat $HOME/.bash_profile > out
```

Nó sẽ tạo một file tên là out trong thư mục hiện tại chứa các nội dung của file bash_profile, bằng cách định hướng đầu ra của cat tới file đó.

Tương tự, ta có thể cung cấp đầu vào là một lênh từ một file hoặc là lệnh sử dụng toán tử đầu vào, <. Tacó thể viết lại lệnh cat để sử dụng toán tử định hướng đầu vào như sau:

```
$ cat < $HOME/.bash_profile > out
```

Kết quả của lệnh này vẫn như thế nhưng nó cho ta hiểu thêm về cách sử dụng định hướng đầu vào đầu ra.

Toán tử định hướng đầu ra, >, sẽ ghi đè lên bất cứ file nào đang tồn tại. Đôi khi điều này là không mong muốn, vì thế bash cung cấp toán tử nối thêm dữ liệu, >>, cho phép nối thêm dữ liệu vào cuôi file. Hay xem lệnh thêm bí danh cdlpu vào cuối của file .bashrc của tôi:

$echo "alias cdlpu='cd $HOME/kwall/projects/lpu' " >> $HOME/.bashrc

Một cách sử dụng định hướng đầu vào là đầu vào chuẩn (bàn phím). Cú pháp của lệnh này như sau:

Command << label

Input … Label

Cú pháp này nói lên rằng ***command*** đọc các input cho đến khi nó gặp label.

Ví dụ

```
#!/bin/bash
#######################################
USER=anonymous
PASS=kwall@xmission.com
ftp -i -n << END open ftp.caldera.com user $USER $PASS cd /pub
ls close END
```

6.2.5. Hiện dòng văn bản

Lệnh **echo** hiện ra dòng văn bản được ghi ngay trong dòng lệnh có

Cú pháp lệnh:

echo [tùy chọn] [xâu ký tự]…

Các tùy chọn:

-n : hiện xâu ký tự và dấu nhắc trên cùng một dòng.
-e : bật khả năng thông dịch được các ký tự điều khiển.
-E : tắt khả năng thông dịch được các ký tự điều khiển.
--help : hiện hỗ trợ và thoát. Một số bản Linux không hỗ trợ tham số này. Ví dụ, dùng lệnh **echo** với tham số **-e**

 # echo -e 'thử dùng lệnh echo \n'

sẽ thấy hiện ra chính dòng văn bản ở lệnh:

thử dùng lệnh echo

#

ở đây ký tự điều khiển '\n' là ký tự xuống dòng.

6.2.5. Lệnh read đọc dữ liệu cho biến người dùng

Lệnh read có dạng read <tên biến>

Ví dụ chương trình shell có tên thu1.arg có nội dung như sau:

```
#!/bin/sh
# Chuong trinh hoi ten nguoi va hien lai echo "Ten anh la gi?"
read name
echo "Xin chao, $name , anh go $# doi so"
echo "$*"
```

Sau đó, ta thực hiện

$chmod u+x thu1.arg

$thu1.arg Hoi ten nguoi va hien lai

Sẽ thấy xuất hiện

```
Ten anh la gi? Tran Van An
Xin chao, Tran Van An, anh go 6 doi so
Hoi ten nguoi va hien lai
```

6.2.6. Lệnh set

Để gán kết quả đư ra từ lệnh shell ra các biến tự động, ta dùng lệnh set

Dạng lệnh: ***set*** ***`<lệnh>`***

Sau lệnh này, kết quả thực hiện lệnh không hiện lên màn hình mà gán kết quả đó tương ứng cho các biến tự động. Một cách tự động các từ trong kết quả thực hiện lệnh sẽ gán tương ứng cho các biến tự động (từ $1 trở đi).

Xem xét một ví dụ sau đây (chương trình thu2.arg) có nội dung:

```
#!/bin/sh
# Hien thoi diem chay chuong trinh nay set `date`
echo "Thoi gian: $4 $5"
echo "Thu: $1"
echo "Ngay $3 thang $2 nam $6"
```

Sau khi đổi mode của File chương trình này và chạy, chúng ta nhận được:

```
Thoi gian: 7:20:15 EST
Thu: Tue
Ngay 20 thang Oct nam 1998
```

Như vậy,

```
$# = 6
$* = Tue Oct 20 7:20:15 EST 1998
$1 = Tue      $2=Oct      $3 = 20      $4 = 7:20:15
$5 = EST      $6 = 1998
```

6.2.7. Tính toán trên các biến

Các tính toán trong shell được thực hiện với các đối số nguyên. Các phép toán gồm có: cộng (+), trừ (-), nhân (*), chia (/), mod (%).

Biểu thức thực hiện theo các phép toán đã nêu. Tính toán trên shell có dạng:

`expr <biểu thức>`

Ví dụ, chương trình với tên ***cong.shl*** sau đây:

```
#!/bin/sh
# Tinh va in hai so
tong = `expr $1 + $2`
echo "Tong = $tong" Sau đó, khi đổi mod và chạy
$cong.shl 5 6
```

sẽ hiện ra:

```
Tong = 11
```

6.2.8. Chương trình ví dụ

```
/* Program 5 */
#!/bin/sh
# Chuong trinh liet ke cac thu muc con cua 1 thu muc
# Minh hoa cach su dung if then fi, while do done
# va cac CT test, expr
if test $# -ne 1
then
        echo Cu phap: $0 \<Ten thu muc\>
        exit 1
fi
cd $1   # Chuyen vao thu muc can list
if test $? -ne 0 # Neu thu muc khong ton tai thi ra khoi CT
then
        exit 1
```

```
if ls -lL |\
# Liet ke ca cac thong tin cua symbolic link
# Su dung sub-shell de tu giai phong bien
{
        sum=0
        # Lenh read x y de bo di dong 'total 1234..' cua lenh ls -lL
        read x y ;
        while read mode link user group size month day hour name do
                if [ -d $name ]
                then
                        echo $name $size \($mode\)
                fi
        done
}
```

6.3. Lập trình C trên UNIX

6.3.1. Trình biên dịch gcc

Hệ điều hành UNIX luôn kèm theo bộ dịch ngôn ngữ lập trình C với tên gọi là cc (C compiler). Trong Linux, bộ dịch có tên là ***gcc*** (GNU C Compiler) với ngôn ngữ lập trình không khác nhiều với C chuẩn. ***gcc*** cho người lập trình kiểm tra trình biên dịch. Tiến trình biên dịch bao gồm bốn giai đoạn:

- Tiền xử lý
- Biên dịch
- Tập hợp
- Liên kết

Ta có thể dừng tiến trình sau một trong những giai đoạn để kiểm tra kết quả biên dịch tại giai đoạn ấy. ***gcc*** cũng có thể chấp nhận ngôn ngữ khác của C, như ANSI C hay C truyền thống. Như đã nói ở trên, ***gcc*** thích hợp biên dịch C++ hay Objective-C. Ta có thể kiểm soát lượng cũng như kiểu thông tin cần debug, tất nhiên là có thể nhúng trong tiến trình nhị phân hóa kết quả và giống như hầu hết các trình biên dịch, gcc cũng thực hiện tối ưu hóa mã.

Trước khi bắt đầu đi sâu vào nghiên cứu **gcc**, ta xem một ví dụ sau:

```
#include<stdio.h>
int main (void)
{
fprintf( stdout, "Hello, Linux programming world!\n");
return 0;
}
```

*Một chương trình điển hình dùng để minh hoạ việc sử dụng **gcc***

Để biên dịch và chạy chương trình này hãy gõ:

```
$ gcc hello.c –o hello
$ ./hello
Hello, Linux programming world!
```

Dòng lệnh đầu tiên chỉ cho **gcc** phải biên dịch và liên kết file nguồn **hello.c**, tạo ra tập tin thực thi, bằng cách chỉ định sử dụng đối số **-o hello**. Dòng lệnh thứ hai thực hiện chương trình, và kết quả cho ra trên dòng thứ 3.

Có nhiều chỗ mà ta không nhìn thấy được, **gcc** trước khi chạy **hello.c** thông qua bộ tiền xử lý của **cpp**, để mở rộng bất kỳ một **macro** nào và chèn thêm vào nội dung của những file **#include**. Tiếp đến, nó biên dịch mã nguồn tiền xử lý sang mã **obj** . Cuối cùng, trình liên kết, tạo ra mã nhị phân cho chương trình **hello.**

Ta có thể tạo lại từng bước này bằng tay, chia thành từng bước qua tiến trình biên dịch. Để chỉ cho **gcc** biết phải dừng việc biên dịch sau khi tiền xử lý, ta sử dụng tuỳ chọn **–E** của **gcc**:

$ gcc –E hello.c –o hello.cpp

Xem xét **hello.cpp** và ta có thể thấy nội dung của **stdio.h** được chèn vào file, cùng với những mã thông báo tiền xử lý khác. Bước tiếp theo là biên dịch **hello.cpp** sang mã **obj**. Sử dụng tuỳ chọn **–c** của **gcc** để hoàn thành:

$ gcc –x cpp-output -c hello.cpp –o hello.o

Trong trường hợp này, ta không cần chỉ định tên của file output bởi vì trình biên dịch tạo một tên file **obj** bằng cách thay thế **.c** bởi **.o**. Tuỳ chọn –x chỉ cho **gcc** biết bắt đầu biên dịch ở bước được chỉ báo trong trường hợp này với mã nguồn tiền xử lý.

Làm thế nào **gcc** biết chia loại đặc biệt của file? Nó dựa vào đuôi mở rộng của file ở trên để xác định rõ phải xử lý file như thế nào cho đúng. Hầu hết những đuôi mở rộng thông thường và chú thích của chúng được liệt kê trong bảng dưới.

Phần mở rộng	**Kiểu**
.c	Mã nguồn ngôn ngữ C
.c, .cpp	Mã nguồn ngôn ngữ C++
.i	Mã nguồn C tiền xử lý
.ii	Mã nguồn C++ tiền xử lý
.S, .s	Mã nguồn Hợp ngữ
.o	Mã đối tượng biên dịch (obj)
.a, .so	Mã thư viện biên dịch

Các phần mở rộng của tên file đối với ***gcc***

Liên kết file đối tượng, và cuối cùng tạo ra mã nhị phân:

$ gcc hello.o –o hello

Trong trường hợp , ta chỉ muốn tạo ra các file **obj,** và như vậy thì bước liên kết là không cần thiết.

Hầu hết các chương trình C chứa nhiều file nguồn thì mỗi file nguồn đó đều phải được biên dịch sang mã **obj** trước khi tới bước liên kết cuối cùng. Giả sử có một ví dụ, ta đang làm việc trên **killerapp.c** là chương trình sử dụng phần mã của **helper.c**, như vậy để biên dịch **killerapp.c** ta phải dùng dòng lệnh sau:

$ gcc killerapp.c helper.c –o killerapp

gcc qua lần lượt các bước tiền xử lý - biên dịch – liên kết, lúc này tạo ra các file **obj** cho mỗi file nguồn trước khi tạo ra mã nhị phân cho **killerapp**.

Một số tuỳ chọn dòng lệnh của **gcc**:

- **-o FILE :** Chỉ định tên file output; không cần thiết khi biên dịch sang mã obj. Nếu FILE không được chỉ rõ thì tên mặc định sẽ là a.out.
- **-c** : Biên dịch không liên kết.
- **-DF00=BAR** : Định nghĩa macro tiền xử lý đặt tên F00 với một giá trị của BAR trên dòng lệnh.
- **-IDIRNAME** : Trước khi chưa quyết định được DIRNAME hãy tìm kiếm những file include trong danh sách các thư mục(tìm trong danh sách các đường dẫn thư mục)
- **-LDIRNAME** : Trước khi chưa quyết định được DIRNAME hãy tìm kiếm những file thư viện trong danh sách các thư mục. Với mặc định gcc liên kết dựa trên những thư viện dùng chung
- **-static** : Liên kết dựa trên những thư viện tĩnh

-lF00 : Liên kết dựa trên libF00

-g : Bao gồm chuẩn gỡ rối thông tin mã nhị phân

-ggdb : Bao gồm tất cả thông tin mã nhị phân mà chỉ có chương trình gỡ rối GNU- gdb mới có thể hiểu được

-O : Tối ưu hoá mã biên dịch

-ON : Chỉ định một mức tối ưu hoá mã N, 0<=N<=3.

-ANSI : Hỗ trợ chuẩn ANSI/ISO của C, loại bỏ những mở rộng của GNU mà xung đột với chuẩn(tuỳ chọn này không bảo đảm mã theo ANSI).

-pedantic : Cho ra tất cả những cảnh báo quy định bởi chuẩn

-pedantic-erors : Thông báo ra tất cả các lỗi quy định bởi chuẩn ANSI / ISO của C.

-traditional : Hỗ trợ cho cú pháp ngôn ngữ C của Kernighan và Ritchie (giống như cú pháp định nghĩa hàm kiểu cũ).

-w : Chặn tất cả thông điệp cảnh báo.

-Wall : Thông báo ra tất cả những cảnh báo hữu ích thông thường mà gcc có thể cung cấp.

-werror : Chuyển đổi tất cả những cảnh báo sang lỗi mà sẽ làm ngưng tiến trình biên dịch.

-MM : Cho ra một danh sách sự phụ thuộc tương thích được tạo.

-v : Hiện ra tất cả các lệnh đã sử dụng trong mỗi bước của tiến trình biên dịch.

6.3.2. Công cụ GNU make

Trong trường hợp ta viết một chương trình rất lớn được cấu thành bởi từ nhiều file, việc biên dịch sẽ rất phức tạp vì phải viết các dòng lệnh gcc rất là dài. Để khắc phục tình trạng này, công cụ GNU make đã được đưa ra. GNU make được giải quyết bằng cách chứa tất cả các dòng lệnh phức tạp đó trong một file gọi là makefile. Nó cũng làm tối ưu hóa tiến trình dịch bằng cách phát hiện ra những file nào có thay đổi thì nó mới dịch lại, còn file nào không bị thay đổi thì nó sẽ không làm gì cả, vì vậy thời gian dịch sẽ được rút ngắn.

Một makefile là một cơ sở dữ liệu văn bản chứa cách luật, các luật này sẽ báo cho chương trình make biết phải làm gì và làm như thế nào. Một luật bao gồm các thành phần như sau:

Đích (target) – cái mà make phải làm

Một danh sách các thành phần phụ thuộc (dependencies) cần để tạo ra đích

Một danh sách các câu lệnh để thực thi trên các thành phần phụ thuộc

Khi được gọi, GNU make sẽ tìm các file có tên là GNUmakefile, makefile hay Makefile. Các luật sẽ có cú pháp như sau:

target: dependency1, dependency2, ….
command command
……

Target thường là một file như file khả thi hay file object ta muốn tạo ra. Dependency là một danh sách các file cần thiết như là đầu vào để tạo ra target.

Command là các bước cần thiết (chẳng hạn như gọi chương trình dịch) để tạo ra target.

Dưới đây là một ví dụ về một makefile về tạo ra một chương trình khả thi có tên là editor (số hiệu dòng chỉ đưa vào để tiện theo dõi, còn nội dung của makefile không chứa số hiệu dòng). Chương trình này được tạo ra bởi một số các file nguồn: editor.c, editor.h, keyboard.h, screen.h, screen.c, keyboard.c.

1. editor : editor.o screen.o keyboard.o
2. gcc -o editor.o screen.o keyboard.o
3. editor.o : editor.c editor.h keyboard.h screen.h
4. gcc -c editor.c

5. screen.o : screen.c screen.h
6. gcc -c screen.c
7. keyboard.o : keyboard.c keyboard.h
8. gcc -c keyboard.c
9. clean:
*10. rm *.o*

Để biên dịch chương trình này ta chỉ cần ra lệnh make trong thư mục chứa file này.

Trong makefile này chứa tất cả 5 luật, luật đầu tiên có đích là ***editor*** được gọi là đích ngầm định. Đây chính là file mà make sẽ phải tạo ra, editor có 3 dependencies editor.o, screen.o, keyboard.o. Tất cả các file này phải tồn tại thì mới tạo ra được đích trên. Dòng thứ 2 là lệnh mà make sẽ gọi thực hiện để tạo ra đích trên. Các dòng tiếp theo là các đích và các lệnh tương ứng để tạo ra các file đối tượng (object).

6.3.3. Làm việc với file

Trong Linux, để làm việc với file ta sử dụng mô tả file (file descriptor). Một trong những thuận lợi trong Linux và các hệ thống UNIX khác là giao diện file làm như nhau đối với nhiều loại thiết bị. Đĩa từ, các thiết bị vào/ra, cổng song song, giả máy trạm (pseudoterminal), cổng máy in, bảng mạch âm thanh, và chuột được quản lý như các thiết bị đặc biệt giống như các tệp thông thường để lập trình ứng dụng. Các socket TCP/IP và miền, khi kết nối được thiết lập, sử dụng mô tả file như thể chúng là các file chuẩn. Các ống (pipe) cũng tương tự các file chuẩn.

Một mô tả file đơn giản chỉ là một số nguyên được sử dụng như chỉ mục (index) vào một bảng các file mở liên kết với từng tiến trình. Các giá trị 0, 1 và 2 liên quan đến các dòng (streams) vào ra chuẩn: *stdin, stderr* và *stdout*; ba dòng đó thường kết nối với máy của người sử dụng và có thể được chuyển tiếp (redirect).

Một số lời gọi hệ thống sử dụng mô tả file. Hầu hết các lời gọi đó trả về giá trị -1 khi có lỗi xảy ra và biến *errno* ghi mã lỗi. Mã lỗi được ghi trong trang chính tuỳ theo từng lời gọi hệ thống. Hàm *perror()* được sử dụng để hiển thị nội dung thông báo lỗi dựa trên mã lỗi.

Hàm open()

Lời gọi open() sử dụng để mở một file. Khuôn mẫu của hàm và giải thích tham số và cờ của nó được cho dưới đây:

#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
*int open(const char *pathname, int flags);*
*int open(const char *pathname, int flags, mode_t mode);*

Đối số pathname là một xâu chỉ ra đường dẫn đến file sẽ được mở. Thông số thứ ba xác định chế độ của file Unix (các bit được phép) được sử dụng khi tạo một file và nên được sử dụng khi tạo một file . Tham số flags nhận một trong các giá trị O_RDONLY, O_WRONLY hoặc O_RDWR

Cờ	Chú giải
O_RDONLY	Mở file để đọc
O_WRONLY	Mở file để ghi
O_RDWR	Mở file để đọc và ghi
O_CREAT	Tạo file nếu chưa tồn tại file đó
O_EXCL	Thất bại nếu file đã có
O_NOCTTY	Không điều khiển tty nếu tty đã mở và tiến trình không điều khiển tty
O_TRUNC	Cắt file nếu nó tồn tại

Cờ	Chú giải
O_APPEND	Nối thêm và con trỏ đặt ở cuối file
O_NONBLOCK	Nếu một tiến trình không thể hoàn thành mà không có trễ, trả về trạng thái trước đó
O_NODELAY	Tương tự O_NONBLOCK
O_SYNC	Thao tác sẽ không trả về cho đến khi dữ liệu được ghi vào đĩa hoặc thiết bị khác

Các giá trị cờ của hàm open()

open() trả về một mô tả file nếu không có lỗi xảy ra. Khi có lỗi , nó trả về giá trị -1 và đặt giá trị cho biến errno. Hàm create() cũng tương tự như open() với các cờ O_CREATE | O_WRONLY | O_TRUNC

Hàm close()

Chúng ta nên đóng mô tả file khi đã thao tác xong với nó. Chỉ có một đối số đó là số mô tả file mà lời gọi open() trả về. Dạng của lời gọi close() là:

```
#include <unistd.h>
int close(int fd);
```

Tất cả các khoá (lock) do tiến trình xử lý trên file được giải phóng, cho dù chúng được đặt mô tả file khác. Nếu tiến trình đóng file làm cho bộ đếm liên kết bằng 0 thì file sẽ bị xoá. Nếu đây là mô tả file cuối cùng liên kết đến một file được mở thì bản ghi ở bảng file mở được giải phóng. Nếu không phải là một file bình thường thì các hiệu ứng không mong muốn có thể xảy ra.

Hàm read()

Lời gọi hệ thống ***read()*** sử dụng để đọc dữ liệu từ file tương ứng với một mô tả file.

```
#include <unistd.h>
ssize_t read(int fd, void *buf, size_t count);
```

Đối số đầu tiên là mô tả file mà được trả về từ lời gọi ***open()*** trước đó. Đối số thứ hai là một con trỏ tới bộ đệm để sao chép dữ liệu và đối số thứ ba là số byte sẽ được đọc. ***read()*** trả về số byte được đọc hoặc -1 nếu có lỗi xảy ra.

Hàm write()

Lời gọi hệ thống ***write()*** sử dụng để ghi dữ liệu vào file tương ứng với một mô tả file.

```
#include <unistd.h>
ssize_t write(int fd, const void *buf, size_t count);
```

Đối số đầu tiên là số mô tả file được trả về từ lời gọi ***open()*** trước đó. Đối số thứ hai là con trỏ tới bộ đệm (để sao chép dữ liệu, có dung lượng đủ lớn để chứa dữ liệu) và đối số thứ ba xác định số byte sẽ được ghi. write() trả về số byte đọc hoặc -1 nếu có lỗi xảy ra

Hàm ftruncate()

Lời gọi hệ thống ftruncate() cắt file tham chiếu bởi mô tả file fd với độ dài được xác định bởi tham số length

```
#include <unistd.h>
int ftruncate(int fd, size_t length);
```

Trả về giá trị 0 nếu thành công và -1 nếu có lỗi xảy ra.

Hàm lseek()

Hàm lseek() đặt vị trí đọc và ghi hiện tại trong file được tham chiếu bởi mô tả file files tới vị trí offset

```
#include <sys/types.h>
#include <unistd.h>
off_t lseek(int fildes, off_t offset, int whence);
```

Phụ thuộc vào giá trị của whence, giá trị của offset là vị trí bắt đầu (SEEK_SET), vị trí hiện tại (SEEK_CUR), hoặc cuối file (SEEK_END). Giá trị trả về là kết quả của offset: bắt đầu file, hoặc một giá trị của off_t , giá trị -1 nếu có lỗi.

Hàm fstat()

Hàm fstat () đưa ra thông tin về file thông qua việc mô tả các file, nơi kết quả của struct stat được chỉ ra ở con trỏ chỉ đến buf().Kết quả trả về giá trị 0 nếu thành công và nhận giá trị -1 nếu sai (kiểm tra lỗi).

```
#include <sys/stat.h>
#include <unistd.h>
int fstat(int filedes, struct stat *buf);
```

Sau đây là định nghĩa của struct stat:

```
struct stat
{
dev_t   st_dev;          / * thiết bị */
int_t   st_ino ;         /* inode */
mode_tst_mode;           /* chế độ bảo vệ */
nlink_t st_nlink;        /* số lượng các liên kết cứng */
uid_t   st_uid;          /* số hiệu của người chủ */
gid_t   st_gid;          /* số hiệu nhóm của người chủ*/
dev_t   st_rdev;         /* kiểu thiết bị */
off_t   st_size;         /* kích thước bytes */
unsigned long st_blksize;  /* kích thước khối*/
unsigned long st_blocks;   /* Số lượng các khối đã sử dụng*/
time_t  st_atime;        /* thời gian truy cập cuối cùng*/
time_t  st_mtime;        /* thời gian cập nhật cuối cùng */
time_t  st_ctime;        /* thời gian thay đổi cuối cùng */
};
```

Hàm fchown()

Lời gọi hệ thống fchown() cho phép tathay đổi người chủ và nhóm người chủ kết hợp với việc mở file.

```
#include <sys/types.h>
#include <unistd.h>
int fchown(int fd, uid_t owner, gid_t group);
```

Tham số đầu tiên là mô tả file, tham số thứ hai là số định danh của người chủ, và tham số thứ ba là số định danh của nhóm người chủ**.** Người dùng hoặc nhóm người dùng sẽ được phép sử dụng khi giá trị -1 thay đổi. Giá trị trả về là 0 nếu thành công và –1 nếu gặp lỗi (kiểm tra biến errno).

Thông thường người dùng có thể thay đổi nhóm các file thuộc về họ. Chỉ root mới có quyền thay đổi người chủ sở hữu của nhiều nhóm.

Hàm fchdir( )

Lời gọi hàm fchdir( ) thay đổi thư mục bằng cách mở file được mô tả bởi biến fd. Giá trị trả về là 0 nếu thành công và –1 nếu có lỗi (kiểm tra biến errno).

```
#include <unistd.h>
int fchdir(int fd);
```

6.3.4. Thư viện liên kết

Phần này sẽ giới thiệu cách tạo ra và sử dụng thư viện (các module chương trình đã được viết và được tái sử dụng nhiều lần). Thư viện gốc của C/C++ trên Linux chính là *glibc,* thư viện này cung cấp cho người dùng rất nhiều lời gọi hệ thống. Các thư viện trên Linux thường được tổ chức dưới dạng tĩnh (static library), thư viện chia sẻ (shared library) và động

(dynamic library - giống như DLL trên MS Windows). Thư viện tĩnh được liên kết cố định vào trong chương trình trong tiến trình liên kết. Thư viện dùng chung được nạp vào bộ nhớ trong khi chương trình bắt đầu thực hiện và cho phép các ứng dụng cùng chia sẻ loại thư viện này. Thư viện liên kết động được nạp vào bộ nhớ chỉ khi nào chương trình gọi tới.

** Thư viện liên kết tĩnh*

Thư viện tĩnh và các thư viện dùng chung (shared library) là các file chứa các file được gọi là các module đã được biên dịch và có thể sử dụng lại được. Chúng được lưu trữ dưới một định dạng đặc biệt cùng với một bảng (hoặc một bản đồ) phục vụ cho tiến trình liên kết và biên dịch. Các thư viện liên kết tĩnh có phần mở rộng là .a. Để sử dụng các module trong thư viện ta cần thêm phần *#include* file tiêu đề (header) vào trong chương trình nguồn và khi liên kết (sau tiến trình biên dịch) thì liên kết với thư viện đó. Dưới đây là một ví dụ về cách tạo và sử dụng một thư viên liên kết tĩnh.

Có 2 phần trong ví dụ này, phần thứ nhất là mã nguồn cho thư viện và phần thứ 2 cho chương trình sử dụng thư viện.

Mã nguồn cho file liberr.h

```
/*
* liberr.h
*/
#ifndef _LIBERR_H
#define _LIBERR_H
#include <stdarg.h>
/* in ra một thông báo lỗi tới việcgọi stderr và return hàm gọi */
void err_quit(const char *fmt, ... );
/* in ra một thông điệp lỗi cho logfile và trả về hàm gọi */
void log_ret(char *logfile, const char *fmt, ...);
/* in ra một thông điệp lỗi cho logfile và thoát */
void log_quit( char *logfile, const char *fmt , ...);
/* in ra một thông báo lỗi và trả lại hàm gọi */
void err_prn(const char *fmt, va_list ap, char *logfile);
#endif //_LIBERR_H
```

Mã nguồn file liberr.c

```
#include <errno.h>
#include <stdarg.h>
#include <stdlib.h>
#include <stdio.h>
#include "liberr.h"
#define
MAXLINELEN 500
void err_ret(const char *fmt,...)
{
        va_list ap;
        va_start(ap, fmt); err_prn(fmt, ap, NULL); va_end(ap);
        return;
}
void err_quit(const char *fmt,...)
{
        va_list ap;
        va_start(ap, fmt); err_prn(fmt, ap, NULL); va_end(ap);
        exit(1);
}
void log_ret(char *logfile, const char *fmt,...)
```

```
{
        va_list ap;
        va_start(ap, fmt); err_prn(fmt,ap, logfile); va_end(ap);
        return;
}
void log_quit(char *logfile, const char *fmt,... )
{
        va_list ap;
        va_start(ap, fmt); err_prn(fmt, ap,logfile); va_end(ap);
        exit(1);
}
extern void err_prn( const char *fmt, va_list ap, char *logfile)
{
        int save_err;
        char buf[MAXLINELEN]; FILE *plf;
        save_err = errno;
        vsprintf(buf,fmt, ap);
        sprintf( buf+strlen(buf), ": %s", strerror(save_err));
        strcat(buf, "\n"); fflush(stdout);
        if(logfile !=NULL){
                if((plf=fopen(logfile, "a") ) != NULL){
                        fputs(buf, plf);
                        fclose(plf);
                }else
                fputs("failed to open log file \n", stderr);
        }else   fputs(buf, stderr);
        fflush(NULL);
        return;
}
```

Để tạo một thư viện tĩnh, bước đầu tiên là dịch đoạn mã của form đối tượng:

$gcc –H –c liberr.c –o liberr.o

Tiếp theo:

$ar rcs liberr.a liberr.o

```
/*
* Mã nguồn file testerr.c
*/
#include <stdio.h>
#include <stdlib.h>
#include "liberr.h"
#define ERR_QUIT_SKIP 1
#define LOG_QUIT_SKIP 1
int main(void)
{
FILE *pf;
fputs("Testing err_ret()...\n", stdout);
if((pf = fopen("foo", "r")) == NULL)
err_ret("%s %s", "err_ret()", "failed to open foo");
fputs("Testing log_ret()...\n", stdout);
if((pf = fopen("foo", "r")) == NULL);
log_ret("errtest.log", "%s %s", "log_ret()", "failed to open foo");
#ifndef ERR_QUIT_SKIP
```

```
fputs("Testing err_quit()...\n", stdout);
if((pf = fopen("foo", "r")) == NULL)
err_ret("%s %s", "err_quit()", "failed to open foo");
#endif /* ERR_QUIT_SKIP */
#ifndef LOG_QUIT_SKIP
fputs("Testing log_quit()...\n", stdout);
if((pf = fopen("foo", "r")) == NULL)
log_ret("errtest.log", "%s %s", "log_quit()", "failed to open foo");
#endif /* LOG_QUIT_SKIP */
return EXIT_SUCCESS;
}
```

Biên dịch chương trình kiểm tra, ta sử dụng dòng lệnh:

$ gcc -g errtest.c -o errtest -L. -lerr

Tham số -L. chỉ ra đường dẫn tới thư mục chứa file thư viện là thư mục hiện thời, tham số –lerr chỉ rõ thư viện thích hợp mà chúng ta muốn liên kết. Sau khi dịch ta có thể kiểm tra bằng cách chạy chương trình.

** Thư viện dùng chung*

Thư viện dùng chung có nhiều thuận lợi hơn thư viện tĩnh.Thứ nhất, thư viện dùng chung tốn ít tài nguyên hệ thống, chúng sử dụng ít không gian đĩa vì mã nguồn thư viện dùng chung không biên dịch sang mã nhị phân nhưng được liên kết và được dùng tự động mỗi lần dùng. Chúng sử dụng ít bộ nhớ hệ thống vì nhân chia sẻ bộ nhớ cho thư viện dùng chung này và tất cả các chương trình đều sử dụng chung miền bộ nhớ này. Thứ 2, thư viện dùng chung nhanh hơn vi chúng chỉ cần nạp vào một bộ nhớ. Lí do cuối cùng là mã nguồn trong thư viện dùng chung dễ bảo trì. Khi các lỗi được sửa hay thêm vào các đặc tính, người dùng cần sử dụng thư viện nâng cấp. Đối với thư viện tĩnh, mỗi chương trình khi sử dụng thư viện phải biên dịch lại.

Trình liên kết (linker)/module tải (loader) ***ld.so*** liên kết tên biểu tượng tới thư viện dùng chung mỗi lần chạy. Thư viện dùng chung có tên đặc biệt (gọi là soname), bao gồm tên thư viện và phiên bản chính. Ví dụ: tên đầy đủ của thư viện C trong hệ thống là libc.so.5.4.46, tên thư viện là libc.so, tên phiên bản chính là 5, tên phiên bản phụ là 4, 46 là mức vá (patch level). Như vậy, soname thư viện C là libc.5. Thư viện libc6 có soname là libc.so.6, sự thay đổi phiên bản chính là sự thay đổi đáng kể thư viện. Phiên bản phụ và patch level thay đổi khi lỗi được sửa nhưng soname không thay đổi và bản mới có sự thay khác biệt đáng kể so với bản cũ.

Các chương trình ứng dụng liên kết dựa vào soname. Tiện ích ***idconfig*** tạo một biểu tượng liên kết từ thư viện chuẩn libc.so.5.4.46 tới soname libc.5 và lưu trữ thông tin này trong /etc/ld.so.cache. Trong lúc chạy, ld.so đọc phần lưu trữ, tìm soname thích hợp và nạp thư viện hiện tai vào bộ nhớ, kết nối hàm ứng dụng gọi tới đối tượng thích hợp trong thư viện.

Các phiên bản thư viện khác nhau nếu:

- Các giao diện hàm đầu ra thay đổi.
- Các giao diện hàm mới được thêm.
- Chức năng hoạt động thay đổi so với đặc tả ban đầu
- Cấu trúc dữ liệu đầu ra thay đổi
- Cấu trúc dữ liệu đầu ra được thêm

Để duy trì tính tương thích của thư viện, cần đảm bảo các yêu cầu:

- Không thêm vào những tên hàm đã có hoặc thay đổi hoạt động của nó
- Chỉ thêm vào cuối cấu trúc dữ liệu đã có hoặc làm cho chúng có tính tuỳ chọn hay được khởi tạo trong thư viện
- Không mở rộng cấu trúc dữ liệu sử dụng trong các mảng

Xây dựng thư viện dùng chung hơi khác so với thư viện tĩnh, tiến trình xây dựng thư viện dùng chung được minh hoạ dưới đây:

- Khi biên dịch file đối tượng, sử dụng tùy chọn -fpic của gcc nó sẽ tạo ra mã độc lập vị trí (position independence code) từ đó có thể liên kết hay sử dụng ở bất cứ chỗ nào
- Không loại bỏ file đối tượng và không sử dụng các tùy chọn ***–fomit –frame –pointer*** của ***gcc***, vì nếu không sẽ ảnh hưởng đến tiến trình gỡ rối (debug)
- Sử dụng tuỳ chọn ***-shared*** and ***–soname*** của ***gcc***
- Sử dụng tuỳ chọn ***–Wl*** của ***gcc*** để truyền tham số tới trình liên kết ***ld.***
- Thực hiện tiến trình liên kết dựa vào thư viện C, sử dụng tuỳ chọn ***–l*** của ***gcc***

Trở lại thư viện xử li lỗi , để tạo thư viện dùng chung trước hết xây dụng file đối tượng:

$ gcc -fPiC -g -c liberr.c -o liberr.o

Tiếp theo liên kết thư viện:

$ gcc -g -shared -Wl,-soname,liberr.so -o liberr.so.1.0.0 liberr.o -lc

Vì không thể cài đặt thư viện này như thư viện hệ thống trong /usr hay /usr/lib chúng ta cần tạo 2 kiên kết, một cho soname.Và cho trình liên kết khi kết nối dựa vào liberr, sử dụng ***–lerr***:

$ ln -s liberr.so.1.0.0 liberr.so

Bây giờ, để dử dụng thư viện dùng chung mới chúng ta quay lại chương trình kiểm tra, chúng ta cần hướng trình liên kết tới thư viện nào để sử dụng và tìm nó ở đâu, vì vậy chúng ta sẽ sử dụng tuỳ chọn –l và –L:

$ gcc -g errtest.c -o errtest -L. -lerr

Cuối cùng để chạy chương trình, chúng ta cần chỉ cho ld.so nơi để tìm thư viện dùng chung :

$ LD_LIBRARY_PATH=$(pwd) ./errtest

** Sử dụng đối tượng dùng chung theo cách động*

Một cách để sử dụng thư viện dùng chung là nạp chúng tự động mỗi khi chạy không giống như nhũng thư viện liên kết và nạp một cách tự động. Ta có thể sử dụng giao diện dl (dynamic loading) vì nó tạo sự linh hoạt cho lập trình viên hay người dùng.

Giả sử ta đang tạo một ứng dụng sử lý đồ hoạ. Trong ứng dụng, ta biểu diễn dữ liệu ở một dạng không theo chuẩn nhưng lại thuận tiện cho ta xử lý, và ta cần có nhu cầu chuyển dữ liệu đó ra các định dạng thông dụng đã có (số lượng các định dạng này có thể có hàng trăm loại) hoặc đọc dữ liệu từ các định dạng mới này vào để xử lý. Để giải quyết vấn đề này ta có thể sử dụng giải pháp là thư viện. Nhưng khi có thêm một định dạng mới thì ta lại phải biên dịch lại chương trình. Đây lại là một điều không thích hợp lắm. Khả năng sử dụng thư viện động sẽ giúp ta giải quyết vấn đề vừa gặp phải. Giao diện ***dl*** cho phép tạo ra giao diện (các hàm) đọc và viết chung không phụ thuộc vào định dạng của file ảnh. Để thêm hoặc sửa các định dạng của file ảnh ta chỉ cần viết thêm một module để đảm nhận chức năng đó và báo cho chương trình ứng dụng biết là có thêm một module mới bằng cách chỉ cần thay đổi một file cấu hình trong một thư mục xác định nào đó.

Giao diện ***dl*** (cũng đơn thuần được xây dựng như một thư viện - thư viện ***libdl***) chứa các hàm để tải (load), tìm kiếm và giải phóng (unload) các đối tượng chia sẻ. Để sử dụng các hàm này ta thêm file <*dlfcn.h*> vào phần *#include* vào trong mã nguồn, và khi dịch thì liên kết nó với thư viện ***libdl*** bằng cách sử dụng tham số và tên ***–ldl*** trong dòng lệnh dịch. ***dl*** cung cấp 4 hàm xử lí các công việc cần thiết để tải, sử dụng và giải phóng đối tượng dùng chung.

Truy cập đối tượng chia sẻ

Để truy cập một đối tượng chia sẻ, dùng hàm dlopen() có đặc tả như sau:

void *dlopen(const char *filename, int flag);

dlopen() truy cập đối tượng chia sẻ bằng filename và bằng cờ. Filename có thể là đường dẫn đầy đủ, tên file rút gọn hay NULL. Nếu là NULL ***dlopen()*** mở chương trình đang chạy, đó là chương trình của bạn, nếu filename là đường dẫn ***dlopen()*** mở file đó, nếu là tên rút gọn ***dlopen()*** sẽ tìm trong vị trí sau để tìm file:

$LD_ELF_LiBRARY_PATH,

$LD_LIBRARY_PATH, /etc/ld.so.cache, /usr/lib, và /lib.

Cờ có thể là RTLD_LAZY, có nghĩa là các ký hiệu (symbol) hay tên hàm từ đối tượng truy cập sẽ được tìm mỗi khi chúng được gọi, hoặc cờ có thể là RTLD_NOW, có nghĩa tất cả ký hiệu từ đối tượng truy cập sẽ được tìm trước khi hàm ***dlopen()*** trả về. ***dlopen()*** trả điều khiển tới đối tượng truy nhập nếu nó tìm thấy từ filename hay trả về giá trị NULL nếu không tìm thấy.

Sử dụng đối tượng chia sẻ

Trước khi có thể sử dụng mã nguồn trong thư viện ta phải biết đang tìm cái gì và tìm ở đâu. Hàm dlsym() sẽ giúp điều đó:

void *dlsym(void *handle, char *symbol);

dlsym() tìm ký hiệu hay tên hàm trong truy cập và trả lại con trỏ kiểu void tới đối tượng hay NULL nếu không thành công.

Kiểm tra lỗi

Hàm ***dlerror()*** sẽ giúp ta kiểm tra lỗi khi sử dụng đối tượng truy cập động:

const char *dlerror(void);

Nếu một trong các hàm lỗi, ***dlerror()*** trả về thông báo chi tiết lỗi và gán giá trị NULL cho phần bị lỗi.

Giải phóng đối tượng chia sẻ

Để bảo vệ tài nguyên hệ thống đặc biệt bộ nhớ, khi ta sử dụng xong module trong một đối tượng chia sẻ, thì giải phóng chúng. Hàm ***dlclose()*** sẽ đóng đối tượng chia sẻ:

int dlclose(void *handle);

Sử dụng giao diện dl

Để minh hoạ cách sử dụng ***dl***,chúng ta quay lại thư viện xử lí lỗi, sử dụng một chương trình khác như sau:

```
/*
* Mã nguồn chương trình dltest.c
* Dynamically load liberr.so and call err_ret()
*/
#include <stdio.h>
#include <stdlib.h>
#include <dlfcn.h>
int main(void)
{
void *handle;
void (*errfcn)(); const char *errmsg; FILE *pf;
handle = dlopen("liberr.so", RTLD_NOW);
if(handle == NULL) {
fprintf(stderr, "Failed to load liberr.so: %s\n", dlerror());
exit(EXIT_FAILURE);
}
dlerror();
errfcn = dlsym(handle, "err_ret");
if((errmsg = dlerror()) != NULL) {
```

```
fprintf(stderr, "Didn't find err_ret(): %s\n", errmsg);
exit(EXIT_FAILURE);
}
if((pf = fopen("foobar", "r")) == NULL)
errfcn("couldn't open foobar");
dlclose(handle);
return EXIT_SUCCESS;
}
```

Biên dịch ví dụ trên bằng lệnh:

$ gcc -g -Wall dltest.c -o dltest -ldl

Như ta có thể thấy, chúng ta không liên kết dựa vào liberr hay liberr.h trong mã nguồn. Tất cả truy cập tới liberr.so thông qua ***dl***. Chạy chương trình bằng cách sau:

$ LD_LIBRARY_PATH=$(pwd) ***./dltest***

Nếu thành công thì ta nhận được kết quả như sau:

couldn't open foobar: No such file or directory

6.3.5 Các công cụ cho thư viện

Công cụ ***nm***

Lệnh ***nm*** liệt kê toàn bộ các tên hàm (symbol) được mã hoá trong file đối tượng (object) và nhị phân (binary). Lệnh nm sử dụng cú pháp sau:

nm [options] file

Lệnh ***nm*** liệt kê những tên hàm chứa trong file.

Tuỳ chọn	Miêu tả
-C| -demangle	Chuyển tên ký tự vào tên mức người dùng để cho dễ đọc.
-s|-print-armap	Khi sử dụng các file lưu trữ (phần mở rộng là ".a"), in ra các chỉ số của module chứa hàm đó.
-u| -undefined-only	Chỉ đưa ra các hàm không được định nghĩa trong file này, tức là các hàm được định nghĩa ở một file khác.
- l | -line-numbers	Sử dụng thông tin gỡ rối để in ra số dòng nơi hàm được định nghĩa

Công cụ ***ar***

Lệnh ar sử dụng cú pháp sau:

ar {dmpqrtx} [thành viên] file

Lệnh ar tạo, chỉnh sửa và trích các file lưu trữ. Nó thường được sử dụng để tạo các thư viện tĩnh- những file mà chứa một hoặc nhiều file đối tượng chứa các chương trình con thường được sử dụng (subrountine) ở định dạng tiền biên dịch (precompiled format), lệnh

ar cũng tạo và duy trì một bảng mà tham chiếu qua tên ký tự tới các thành viên mà trong đó chúng được định nghĩa. Chi tiết của lệnh này đã được trình bày trong chương trước.

Công cụ ***idd***

Lệnh ***nm*** liệt kê các hàm được định nghĩa trong một file đối tượng, nhưng trừ khi ta biết những gì thư viện định nghĩa những hàm nào. Lệnh ***idd*** hữu ích hơn nhiều. ***idd*** liệt kê các thư viện được chia sẻ mà một chương trình yêu cầu để mà chạy. Cú pháp của nó là:

idd [options] file

Lệnh ***idd*** in ra tên của thư viện chia sẻ mà file này sử dụng.

Ví dụ: chương trình thư "client mutt" cần 5 thư viện chia sẻ, như được minh hoạ sau đây:

$ idd /usr/bin/mutt

libnsl.so.1 => /lib/libns1.so.1 (0x40019000) libslang.so.1 => /usr/lib/libslang.so.1 (0x4002e000) libm.so.6 => /lib/libm/so.6 (0x40072000)
libc.so.6 => /lib/libc.so.6 (0x4008f000)
/lib/id-linux.so.2 => /lib/id-Linux.so.2 (0x4000000)

Tìm hiểu lệnh idconfig

Lệnh ***idconfig*** sử dụng cú pháp sau:

ldconfig [tuỳ chọn] [libs]

Lệnh ***ldconfig*** xác định rõ các liên kết động (liên kết khi chạy) được yêu cầu bởi thư viện được chia sẻ nằm trong các thư mục /usr/lib và /lib. Dưới đây là các tùy chọn của lệnh này:

Các tuỳ chọn	**Các miêu tả**
-p	Đơn thuần chỉ in ra nội dung của /etc/ld.so.cache, một danh sách hiện thời các thư viện được chia sẻ mà ld.so biết.
-v	Cập nhật /etc/ld.so.cache , liệt kê số phiên bản của mỗi thư viện, quét các thư mục và bất kỳ liên kết mà được tạo ra hoặc cập nhật.

Các tuỳ chọn của hàm idconfig

Biến môi trường và file cấu hình.

Chương trình tải (loader) và trình liên kết (linker) ***ld.so*** sử dụng 2 biến môi trường. Biến thứ nhất là $LD_LIBRARY, chứa danh sách các thư mục chứa các file thư viện được phân cách bởi dấu hai chấm để tìm ra các thư viện cần thiết khi chạy. Nó giống như biến môi trường $PATH. Biến môi trường thứ hai là $LD_PRELOAD, một danh sách các thư viện được người dùng thêm vào được phân cách nhau bởi khoảng trống (space).

ld.so cũng cho phép sử dụng 2 file cấu hình mà có cùng mục đích với biến môi trường được đề cập ở trên. File /etc/ld.so.conf chứa một danh sách các thư mục mà chương trình tải và trình liên kết (loader/linker) nên tìm kiếm các thư viện chia sẻ bên cạnh /usr/lib và /lib. /etc/ld.so.preload chứa danh sách các file thư viện được phân cách bằng một khoảng trống các thư viện này là thư viện người dùng tạo ra.

CÂU HỎI VÀ BÀI TẬP

1. Trình bày các yếu tố cơ bản trong lập trình shell
2. Trình bày ý nghĩa, chức năng và tác dụng của trình biên dịch gcc.
3. Thực hành các lệnh trong lập trình shell
4. Thực hành các lệnh trong lập trình C

ĐỀ THI THAM KHẢO

Đề 1:

Câu 1: Trình bày khái niệm và cấu trúc siêu khối

Câu 2: Trong thư mục người dùng /home/tuanpv có các thư mục con là vanban, bangtinh. Hãy viết các lệnh của Linux để:

1. Tạo tại thư mục vanban một thư mục con có tên là hopdong. Sao chép các tệp tin có 2 ký tự phần tên là HD trong thư mục vanban vào thư mục vừa tạo
2. Liệt kê các tệp tin có phần tên bắt đầu bởi ký tự "M" trong thư mục bangtinh lên màn hình (cho hiện các tệp tin có thuộc tính ẩn nếu có)
3. Xác lập quyền chỉ đọc cho các tệp trong thư mục bangtinh.
4. Xoá tất cả các tệp tin 2 ký tự "nh" thuộc phần tên trong thư mục vanban.

Câu 3: Lập chương trình liệt kê tên và sao chép các tệp tin trong thư mục /home/user1/vidu1 sang thư mục /home/user2/vidu

Đề 2:

Câu 1: Trình bày khái niệm và cấu trúc inode

Câu 2: Trong thư mục người dùng /home/minhnd có các thư mục con là musics, games. Hãy viết các lệnh của Linux để:

1. Xoá đi các tệp tin có phần mở rộng là mp3 trong thư mục musics; Xoá thư mục lines trong thư mục games
2. Tạo ra tại thư mục người dùng một thư mục con có tên temp, trong thư mục này tạo hai thư mục con ngang cấp có tên vidu1 và vidu2.
3. Liệt kê các tiến trình đang chạy trong hệ thống.
4. Nén thư mục games thành tệp tin luugames.tar

Câu 3: Lập chương trình đọc và hiển thị nội dung của 1 file không cấu trúc

Đề 3:

Câu 1: Trình bày tên và tác dụng của các thư mục đặc biệt trong Linux

Câu 2: Trong thư mục người dùng /home/cuongpv có các thư mục con là tailieu, tapchi. Hãy viết các lệnh của Linux để:

1. Nối nội dung các tệp sach1, sach2 trong thư mục tailieu thành tệp tapsach đặt tạo thư mục người dùng.
2. Liệt kê các tệp tin trong thư mục tapchi (kể cả tệp tin có thuộc tính ẩn)
3. Nén các tệp tin trong thư mục tailieu thành tệp luutl.zip đặt tại thư mục người dùng.
4. Xóa các tệp tin có ký tự "h" của phần tên trong thư mục tailieu

Câu 3: Cho hai vector m chiều a = (a_1, a_2, a_3,..., a_m) và b = (b_1, b_2, b_3,..., b_m). Hãy lập chương trình để tính tích vô hướng của a và b theo công thức a.b = $a_1.b_1 + a_2.b_2 + ... + a_m.b_m$.

Đề 4:

Câu 1: Trình bày cấu trúc thông tin lưu trữ về tài khoản người của một dùng trong trong file **/etc/passwd**

Câu 2: Trong thư mục người dùng /home/dungnv có các thư mục con là vanhoc và kythuat. Hãy viết các lệnh của Linux để:

1. Tìm tệp tin có chứa nội dung là "Happy Birthday"
2. Thiết lập quyền truy cập thư mục kythuat cho tất cả các người dùng.

3. Tạo một thư mục có tên là nghethuat trong thư mục người dùng, sau đó chép tất cả các tệp có ký tự "01" ở cuối phần tên trong thư mục vanhoc vào thư mục vừa tạo.
4. Liệt kê cấu hình của máy hiện tại

Câu 3: Cho số n, Hãy lập chương trình để thực hiện tính giá trị của hàm cos(x) theo công thức: $Cos(x) = 1- x^2/2! + x^4/4! - x^6/6! + ... (-1)^n x^{2x}/(2n)!$

Đề 5:

Câu 1: Trình bày các yếu tố cơ bản trong lập trình shell

Câu 2: Trong thư mục người dùng /home/thanghv có các thư mục con là congvan, quyetdinh. Hãy viết các lệnh của Linux để:

1. Tạo ra tại thư mục người dung một thư mục con có tên là saoluu,
2. Sao chép tất cả các tệp tại thư mục quyetdinh vào thư mục vừa tạo
3. Thiết lập quyền truy cập thư mục congvan cho nhóm người dùng hanhchinh
4. Xóa các tệp có hai ký tự đầu phần tên là "GM" trong thư mục congvan

Câu 3: Cho số thực a. Hãy lập sơ đồ thuật toán để thực hiện tìm số tự nhiên n nhỏ nhất sao cho $1+1/2 + 1/3 + ... + 1/n > a$

TRƯỜNG ĐẠI HỌC ĐÀ LẠT
KHOA CÔNG NGHỆ THÔNG TIN

ThS. Đặng Thanh Hải

BÀI GIẢNG TÓM TẮT

HỆ ĐIỀU HÀNH

Dành cho sinh viên ngành Công Nghệ Thông Tin

(Lưu hành nội bộ)

Đà Lạt 2008

LỜI NÓI ĐẦU

Giáo trình " Hệ điều hành" được biên soạn theo chương trình đào tạo hệ thống tín chỉ của trường Đại Học Đà Lạt. Mục đích biên soạn giáo trình nhằm cung cấp cho sinh viên ngành Công Nghệ Thông Tin những kiến thức về hệ điều hành.

Tuy có rất nhiều cố gắng trong công tác biên soạn nhưng chắc chắn rằng giáo trình này còn nhiều thiếu sót. Chúng tôi xin trân trọng tiếp thu tất cả những ý kiến đóng góp của các bạn sinh viên, cũng như của các đồng nghiệp trong lĩnh vực này để hoàn thiện giáo trình, phục vụ tốt hơn cho việc dạy và học tin học đang ngày càng phát triển ở nước ta.

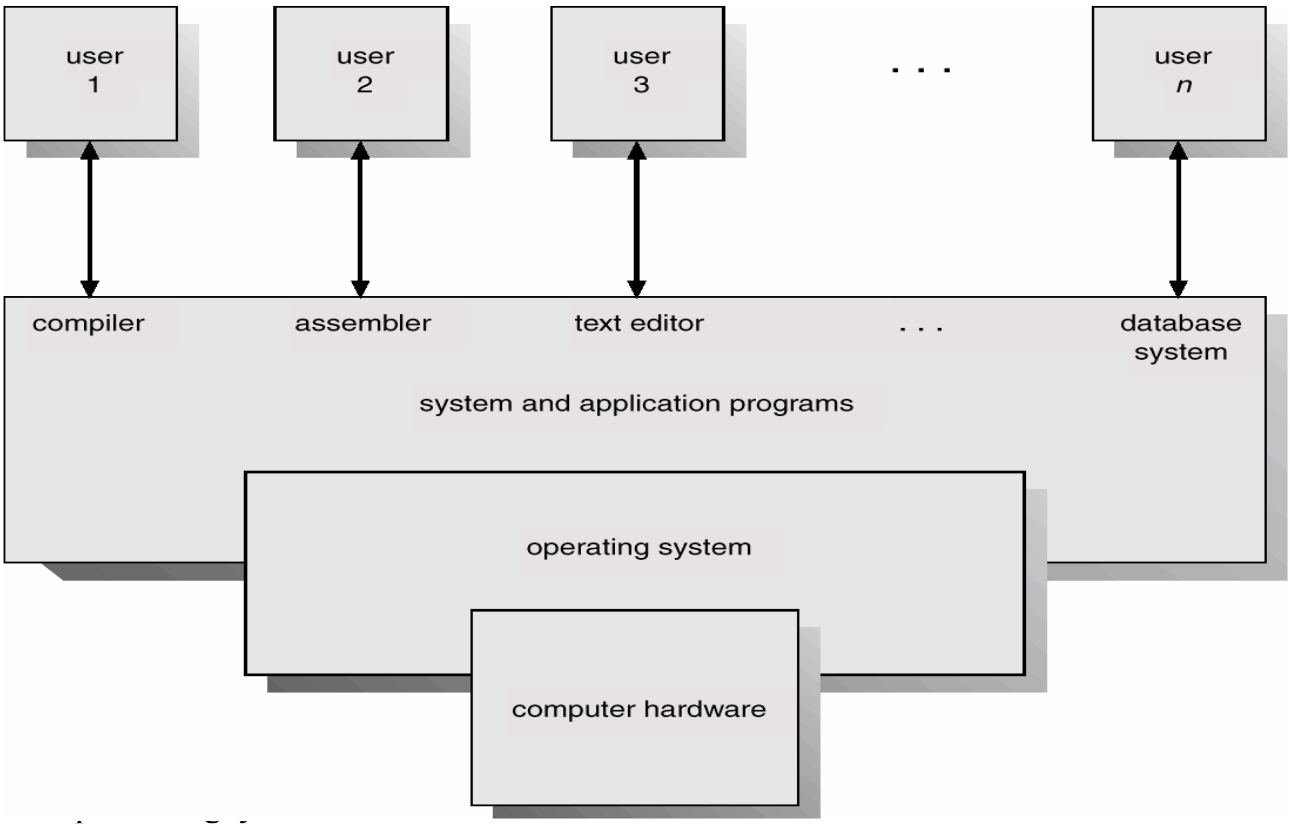
Khoa Công Nghệ Thông Tin
Trường Đại Học Đà Lạt

MỤC LỤC

CHƯƠNG I
TỔNG QUAN HỆ ĐIỀU HÀNH

I. 1 KHÁI NIỆM HỆ ĐIỀU HÀNH

- Hệ điều hành là một chương trình được xem như **trung gian** giữa người sử dụng máy tính và phần cứng máy tính với mục đích **thực hiện** các chương trình giúp cho người dùng sử dụng máy tính **dễ dàng hơn**, sử dụng phần cứng một cách có **hiệu quả.**
- Hệ điều hành là một phần quan trọng của hệ thống máy tính. Một hệ thống máy tính thường bao gồm các phần: phần cứng, hệ điều hành, các chương trình ứng dụng và người sử dụng.
  - **Phần cứng** : Bao gồm tài nguyên cơ bản của máy tính (CPU, memory, I/O devices).
  - **Hệ điều hành**: Điều khiển và kết hợp sử dụng phần cứng trong các ứng dụng khác nhau của nhiều người dùng khác nhau.
  - **Các chương trình ứng dụng** : Sẽ sử dụng tài nguyên hệ thống để giải quyết vấn đề của người sử dụng (Trình biên dịch, hệ thống cơ sở dữ liệu, games, chương trình thương mại).
  - **Người sử dụng** : Người, các máy tính khác.
- Mô hình hệ thống máy tính



Hình 1.1

- Hệ điều hành **Điều khiển chương trình** – Điều khiển thực hiện các chương trình người sử dụng và các hoạt động của thiết bị nhập xuất.
- **Hệ điều hành còn được gọi là Kernel(nhân)** – Đây là các phần cốt lõi của chương trình, thường trú trong bộ nhớ, và thực hiện hầu hết các nhiệm vụ điều hành chính.

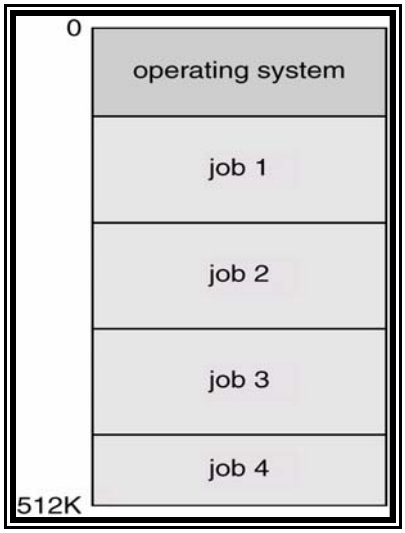
I.2 PHÂN LOẠI HỆ ĐIỀU HÀNH

I.2.1 Hệ điều hành xử lý theo lô đơn giản

- 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 gọi là bộ giám sát thường trực được thiết kế để giám sát việc thực hiện dãy công việc một cách tự động, chương trình này 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.

I.2.2 Hệ điều hành xử lý theo lô đa chương

- Đa chương làm gia tăng khai thác CPU bằng cách tổ chức các công việc sao cho CPU luôn luôn phải trong tình trạng làm việc.
- Cách thực hiện là hệ điều hành lưu trữ 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 các công việc tiếp theo.
- Mô hình bộ nhớ cho hệ điều hành đa chương:



Hình 1.2

- Các đặc trương của hệ điều hành đa chương:
  - Việc nhập xuất phải thực hiện thường xuyên bởi hệ thống.
  - Quản lý bộ nhớ – hệ thống phải cấp phát bộ nhớ cho các công việc.
  - Lập lịch CPU – hệ thống phải chọn giữa các công việc nào thật sự được chạy.
  - Cấp phát các thiết bị.

I.2.3 Hệ điều hành đa nhiệm

- Hệ điều hành đa nhiệm là một sự mở rộng logic của hệ điều hành đa chương. Nhiều công việc cùng được thực hiện thông qua cơ chế chuyển đổi CPU như hệ đa chương nhưng thời gian mỗi lần chuyển đổi diễn ra rất nhanh.
- Hệ điều hành đa nhiệm được phát triển để cung cấp việc sử dụng bên trong của một máy tính có giá trị hơn.
- Một chương trình khi thi hành được gọi là tiến trình. Trong khi thi hành 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 đa nhiệm cho phép nhiều người sử dụng chia xẻ máy tính một cách đồng bộ do thời gian chuyển đổi nhanh nên họ có cảm giác là các tiến trình đang chạy được thi hành cùng lúc.
- Hệ điều hành đa nhiệm phức tạp hơn hệ điều hành đa chương và nó phải có thêm các chức năng: quản trị và bảo vệ bộ nhớ, sử dụng bộ nhớ ảo, …
- Hệ điều hành đa nhiệm hiện nay rất thông dụng.

I.2.4 Hệ điều hành tương tác

- Hệ điều hành cung cấp cơ chế truyền thông trực tiếp giữa người sử dụng và hệ thống. Khi hệ điều hành kết thúc thực hiện một lệnh, nó sẽ tìm ra lệnh kế tiếp từ người sử dụng thông qua bàn phím.
- Hệ thống cho phép người sử dụng truy cập dữ liệu và mã chương trình một cách trực tiếp.

I.2.5 Hệ điều hành giao diện bàn giấy (Desktop)

- Hệ điều hành này có cách giao diện với người sử dụng giống như một bàn làm việc, tức trên màn hình trình bày rất nhiều biểu tượng chương trình, công cụ làm việc. Hệ điều hành có đặc điểm là:
  - Cài đặt trên *máy tính cá nhân* – hệ thống máy tính được thiết kế cho một người sử dụng đơn lẻ.
  - Các thiết bị hỗ trợ đắc lực là thiết bị nhập xuất – bàn phím, mouse, màn hình, máy in.
  - Thuận tiện cho người dùng và đáp ứng nhanh.
  - Có thể kế thừa kỹ thuật để phát triển hệ điều hành lớn hơn.
- Một số hệ điều hành khác nhau sử dụng bàn giấy hiện nay (Windows, UNIX, Linux)

I.2.6 Hệ thống song song

- Ngoài các hệ thống tín chỉ có một bộ xử lý còn có các hệ thống có nhiều bộ xử lý cùng chia xẻ hệ thống đường truyền dữ liệu, đồng hồ, bộ nhớ và các thiết bị ngoại vi.
- Thuận lợi của hệ thống xử lý song song:
  - Xử lý nhiều công việc cùng lúc thật sự
  - Tăng độ tin cậy

- Trong hệ thống xử lý song song được thành hai loại:

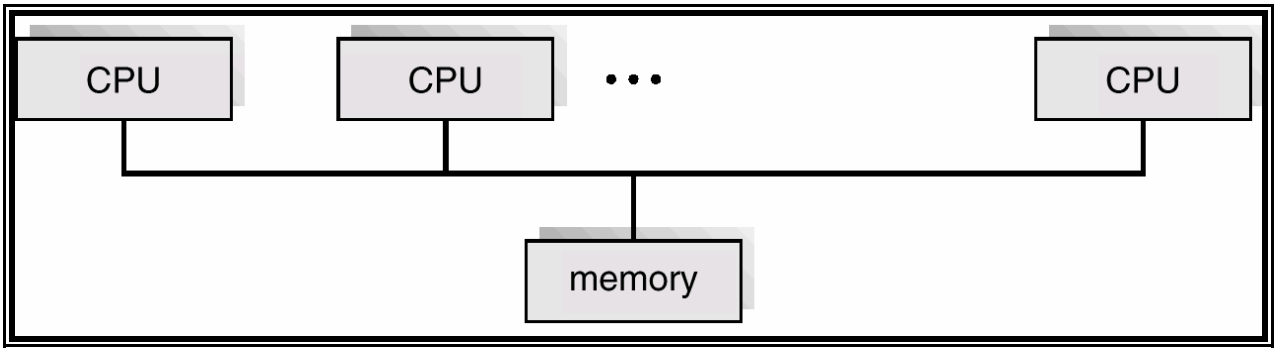
Đa xử lý đối xứng

- Mỗi bộ xử lý chạy một bản sao hệ điều hành.
- Nhiều tiến trình có thể chạy cùng lúc mà không gây hỏng.
- Hầu hết các thế hệ hệ điều hành đều hỗ trợ đa xử lý đối xứng

Đa xử lý không đối xứng

- Mỗi bộ xử lý được gắn vào một công việc cụ thể; Bộ xử lý chủ lập lịch và cấp phát công việc cho bộ xử lý phụ.
- Phổ biến nhiều trong hệ thống cực kỳ lớn.

- Kiến trúc hệ thống đa bộ xử lý đối xứng:

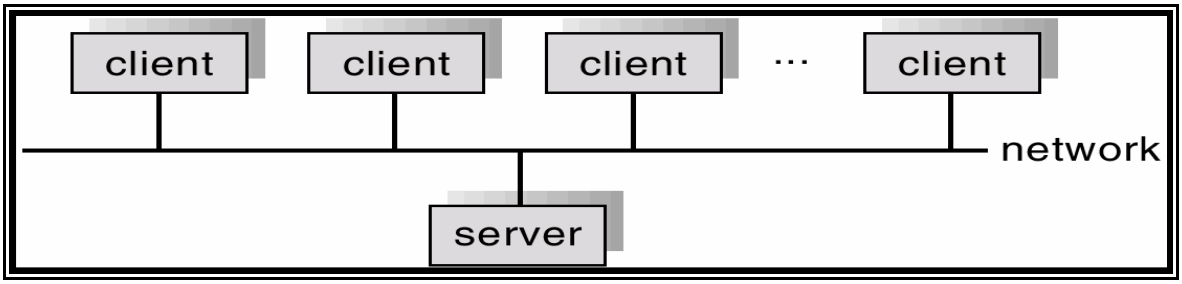


Hình 1.3

I.2.7 Hệ thống phân tán

- Hệ thống thực hiện phân tán việc tính toán giữa các bộ xử lý .
- Mỗi bộ xử lý có vùng nhớ riêng; các bộ xử lý truyền thông với nhau qua hệ thống mạng tốc độ cao.
- Thuận lợi của hệ thống phân tán:
  - Chia xẻ tài nguyên
  - Tăng tốc độ tính toán
  - Đáng tin cậy
  - Truyền thông

- Trong hệ thống yêu cầu cơ sở hạ tầng về mạng. Mạng cục bộ (LAN) hoặc mạng diện rộng (WAN), cũng có thể là hệ thống client-server hoặc peer-to-peer.
- Mô hình hệ thống Client- server:



Hình 1.4

I.2.8 Hệ thống cầm tay

- Máy trợ lý cá nhân kỹ thuật số (PDAs) (personal digital assistant – PDA)
- Vấn đề cần giải quyết :
  - Bộ nhớ bị giới hạn
  - Bộ xử lý chậm
  - Màn hình hiển thị nhỏ

I.3.LỊCH SỬ PHÁT TRIỂN HỆ ĐIỀU HÀNH

- **Thế hệ 1: 1945 – 1955**

  - Năm 1940 Howard Aiken và John Von Neumam đã thành công trong việc xây dựng một máy tính dùng ống chân không.

  -Loại máy này sử dụng khoảng 1000 ống chân không, kích thước lớn nhưng khả năng xử lý chậm

  -Thời kỳ này ngôn ngữ lập trình là ngôn ngữ máy (nhị phân)

  -Việc điều hành máy, thiết kế chương trình đều do một nhóm người.

  -Năm 1950 phiếu đục lỗ ra đời và có thể viết chương trình trên phiếu đục lỗ .

- **Thế hệ 2: 1955 – 1965**
  - Thời kỳ này máy tính được chế tạo bằng thiết bị bán dẫn.
  - Công việc lập trình được thực hiện trên giấy bằng ngôn ngữ (assembler, fortran) sau đó được đục lỗ trên phiếu và cuối cùng đưa phiếu vào máy.
  - Hệ thống xử lý theo lô ra đời. Các công việc lưu trữ vào băng từ, chuyển điều khiển đến các công việc khác nhau được thực hiện bởi một chương trình thường trú- Đây chính là tiền thân của hệ điều hành
  - Với hệ thống máy tính này đã có sự phân biệt rõ ràng giữa người thiết kế , người xây dựng, vận hành, lập trình và bảo trì máy.

- **Thế hệ 3: 1965 – 1980**
  - Thời kỳ này máy tính được chế tạo bằng IC do đó:
    - Kích thước và giá cả máy tính giảm đáng kể
    - Máy tính trở nên phổ biến hơn
    - Các thiết bị ngoại vi dành cho máy tính càng nhiều
    - Các thao tác điều kiển máy tính ngày càng phức tạp
  - Hệ điều hành ra đời nhằm điều phối, kiểm soát hoạt động và giải quyết các yêu cầu tranh chấp thiết bị.
  - Một số hệ điều hành ra đời: MULTICS, UNIX

- **Thế hệ 4: 1980 -**
  - 1980 IBM cho ra đời máy tính cá nhân PC với hệ điều hành MS-DOS
  - Có nhiều hệ điều hành đa nhiệm, giao diện ngày càng thân thiện với người sử dụng ra đời.
  - Hiện nay hệ điều hành mạng được phát triển mạnh mẽ. (Windows, Linux)

CÂU HỎI

1. Trình bày các khái niệm hệ điều hành?
2. Trình bày khái niệm hệ điều hành đa nhiệm? Sự khác nhau giữa hệ điều hành đa chương và hệ điều hành đa nhiệm?
3. Ngày nay một hệ điều hành được thiết kế phải là những loại hệ điều hành nào?

CHƯƠNG II
CẤU TRÚC HỆ ĐIỀU HÀNH

II.1 CÁC THÀNH PHẦN CỦA HỆ ĐIỀU HÀNH

Hệ điều hành cung cấp một môi trường làm việc cho các chương trình thi hành. Nó cung cấp các dịch vụ cho người sử dụng, giao tiếp với người sử dụng. các thành phần bên trong của hệ điều hành

II.1.1 Quản lý tiến trình

- ***Tiến trình*** là một chương trình đang thực hiện. Một tiến trình cần các tài nguyên bao gồm thời gian CPU , bộ nhớ , files, và thiết bị nhập xuất, để hoàn tất các công việc của mình.
- Vai trò của việc quản lý tiến trình trong hệ điều hành.
  - Tạo, huỷ tiến trình của người sử dụng và của hệ thống
  - Ngưng và cho phép chạy lại các tiến trình.
  - Cung cấp cơ chế :
    - Đồng bộ hóa tiến trình
    - Truyền thông giữa các tiến trình
    - Kiểm soát deadlock

II.1.2 Quản lý bộ nhớ chính

- Bộ nhớ là một dãy lớn các word hoặc byte, mỗi phần tử có một địa chỉ. Nó là nơi lưu trữ, truy xuất dữ liệu một cách nhanh chóng.
- Bộ nhớ chính là thiết bị lưu trữ có thể thay đổi. Nó sẽ làm mất hết dữ liệu trong trường hợp hệ thống bị hỏng.
- Vai trò quản lý bộ nhớ chính trong hệ điều hành:
  - Lưu trữ thông tin các vùng nhớ hiện được sử dụng bởi ai.
  - Quyết định tiến trình nào được nạp vào bộ nhớ khi bộ nhớ có chỗ trống.
  - Cấp phát và thu hồi bộ nhớ khi cần thiết.

II.1.3 Quản lý tập tin

- Một file là một sự thu thập các thông tin có liên quan được định nghĩa bởi người tạo ra nó. Thường file thể hiện cho chương trình và dữ liệu.
- Vai trò quản lý file trong hệ điều hành:
  - Tạo và xóa file.
  - Tạo và xoá thư mục.
  - Cung cấp các thao tác trên file và thư mục.
  - Ánh xạ file vào hệ thống lưu trữ phụ.
  - Backup tập tin trên các thiết bị lưu trữ

II.1.4 Quản lý hệ thống nhập xuất

- Một trong những mục tiêu của hệ điều hành là che dấu những đặc thù của thiết bị phần cứng đối với người sử dụng thay vào đó là một lớp thân thiện hơn, người sử dụng dễ thao tác hơn.
- Một hệ thống nhập/ xuất bao gồm**:**
  - Hệ thống buffer-caching
  - Giao tiếp thiết bị
  - Bộ điều khiển cho các thiết bị phần cứng

II.1.5 Quản lý hệ thống lưu trữ phụ

- Chính vì bộ nhớ chính thường thay đổi và quá nhỏ lưu trữ tất cả dữ liệu và chương trình một cách lâu dài,hệ thống máy tính cung cấp bộ nhớ phụ để back up từ bộ nhớ chính.
- Hầu hết hệ thống máy tính ngày nay sử dụng đĩa như thành phần cơ bản lưu trữ cả chương trình và dữ liệu.
- Vai trò quản lý đĩa trong hệ điều hành:
  - Quản lý bộ nhớ còn trống
  - Cấp phát lưu trữ
  - Lập lịch đĩa

II.1.6 Hệ thống bảo vệ

- *Bảo vệ* truy cập bởi các chương trình, các tiến trình, hoặc người sử dụng.
- Cơ chế bảo vệ phải là:
  - Phân biệt giữa cho phép hay không được phép.
  - Chỉ rõ điều khiển bị lợi dụng.
  - Cung cấp các biện pháp phải tuân thủ.

II.1.7 Hệ thống dòng lệnh

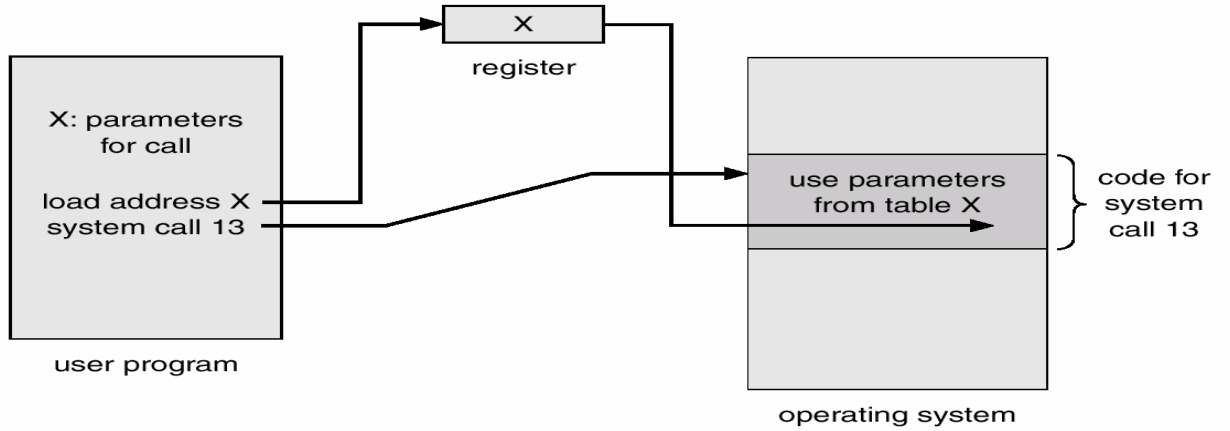
- Một trong những phần quan trọng của chương trình hệ thống trong một hệ điều hành là cơ chế **dòng lệnh**, đây là sự giao tiếp giữa người sử dụng và hệ điều hành.
- Các lệnh đưa vào hệ điều hành thông qua bộ điều khiển lệnh. Trong các hệ thống đa nhiệm một chương trình có thể đọc và thông dịch các lệnh điều khiển được thực hiện một cách tự động.
- Chức năng hệ thống dòng lệnh là lấy lệnh kế tiếp và thi hành.
- Các lệnh có quan hệ với việc tạo và quản lý các tiến trình, kiểm soát nhập xuất, quản lý bộ lưu trữ phụ, quản lý bộ nhớ chính, truy xuất hệ thống tập tin và cơ chế bảo vệ.

II.2 CÁC DỊCH VỤ HỆ ĐIỀU HÀNH

- Thực hiện chương trình – hệ thống có khả năng nạp một chương trình vào bộ nhớ và thi hành nó.
- Thực hiện nhập xuất – Từ chương trình người dùng không thể thực hiện nhập xuất trực tiếp, hệ điều hành phải cung cấp các cách thức để thực hiện nhập xuất.
- Các thao tác trên hệ thống file – chương trình có khả năng đọc, ghi, tạo và xoá file.
- Truyền thông – Trao đổi thông tin giữa các tiến trình đang thực hiện cùng lúc trên máy tính hay trên các hệ thống trên mạng. Thực hiện bằng cách thông qua bộ nhớ dùng chung hay qua các thông điệp.
- Phát hiện lỗi – bảo đảm phát hiện lỗi trong CPU, bộ nhớ, thiết bị nhập xuất hoặc trong chương trình người sử dụng

II.3 LỜI GỌI HỆ THỐNG

- Lời gọi hệ thống là giao diện giữa chương trình đang chạy và hệ điều hành. Thông thường là các chỉ thị bằng ngôn ngữ assembler.
- Có ba phương pháp được sử dụng truyền tham số giữa chương trình đang chạy và hệ điều hành.
    - Truyền tham số qua các thanh ghi.
    - Lưu trữ các tham số trong một bảng trong bộ nhớ và địa chỉ của bảng được truyền qua tham số vào thanh ghi.
    - Các chương trình thực hiện *Push* các tham số vào stack và được pop bởi hệ điều hành.



Hình 2.1

- Các loại lời gọi hệ thống:
    - Điều khiển tiến trình
    - Quản lý file
    - Quản lý thiết bị
    - Truyền thông

II.4 CHƯƠNG TRÌNH HỆ THỐNG

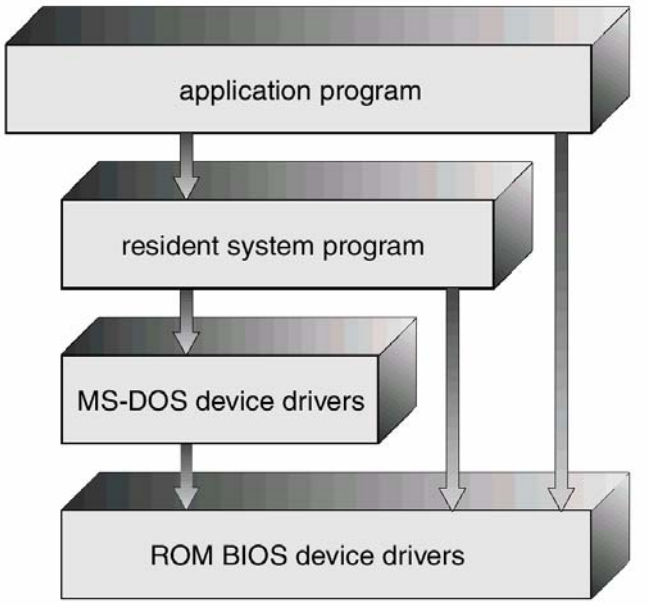
Hệ điều hành là một tập hợp các chương trình hệ thống. Chương trình hệ thống cung cấp một môi trường thuận tiện cho việc phát triển và thực hiện chương trình. Chúng được chia thành :

- **Thao tác trên file**: các chương trình này tạo, xoá, sao chép, in , liệt kê và các thao tác tổng quát trên tập tin và thư mục.
- **Thông tin các trạng thái :** Cung cấp các thông tin về ngày, giờ, khối lượng bộ nhớ hoặc dung lượng đĩa, số lượng người dùng, hoặc thông tin về trạng thái. Những thông tin này được định dạng và xuất trên các thiết bị xuất như terminal hay tập tin.
- **Mô tả tập tin:** Một số trình soạn thảo văn bản bao gồm việc tạo và mô tả tập tin lưu trên đĩa.
- **Hỗ trợ ngôn ngữ lập trình :** Chương trình dịch, hợp ngữ, và thông dịch cho một số ngôn ngữ lập trình. Các chương trình này có thể được cung cấp chung với hệ điều hành hay là một phần riêng.
- **Nạp và thực hiện chương trình :** Hệ thống cung cấp các bộ nạp, định vị, liên kết ngoài ra còn cung cấp chức năng debug.
- **Truyền thông:** Hệ thống cung cấp cơ chế tạo sự kết nối ảo giữa các tiến trình, người sử dụng, và hệ thống máy tính khác.
- **Chương trình ứng dụng:** Thông thường hệ điều hành kèm theo một số chương trình ứng dụng như định dạng, sao chép đĩa,…

II.5 CẤU TRÚC HỆ THỐNG

II.5.1 Cấu trúc đơn giản

- Các hệ điều hành đơn giản thường không có cấu trúc được định nghĩa tốt, thường bắt đầu từ một hệ thống nhỏ, đơn giản và có giới hạn.
- MS-DOS là một hệ điều hành có cấu trúc đơn giản, nó cung cấp những chức năng cần thiết nhất trong một không gian nhỏ nhất do sự giới hạn của phần cứng và không chia thành những đơn thể rõ rệt
- Các chương trình ứng dụng có thể truy cập trực tiếp các thủ tục nhập xuất cơ bản và ghi trực tiếp lên màn hình hay bộ điều khiển đĩa.



Hình 2.2

- Hệ điều hành Unix bao gồm hai phần: hạt nhân và các chương trình hệ thống. hạt nhân được chia thành một chuỗi giao tiếp và driver thiết bị .
- Những gì dưới lời gọi hệ thống và bên trên phần cứng là hạt nhân (kernel)
- Hạt nhân cung cấp hệ thống tập tin, lập lịch CPU, quản trị bộ nhớ và các chức năng của hệ điều hành khác thông qua lời gọi hệ thống.

(the users)		
shells and commands compilers and interpreters system libraries		
system-call interface to the kernel		
signals terminal handling character I/O system terminal drivers	file system swapping block I/O system disk and tape drivers	CPU scheduling page replacement demand paging virtual memory
kernel interface to the hardware		
terminal controllers terminals	device controllers disks and tapes	memory controllers physical memory

Hình 2.3

II.5.2 Cấu trúc theo lớp

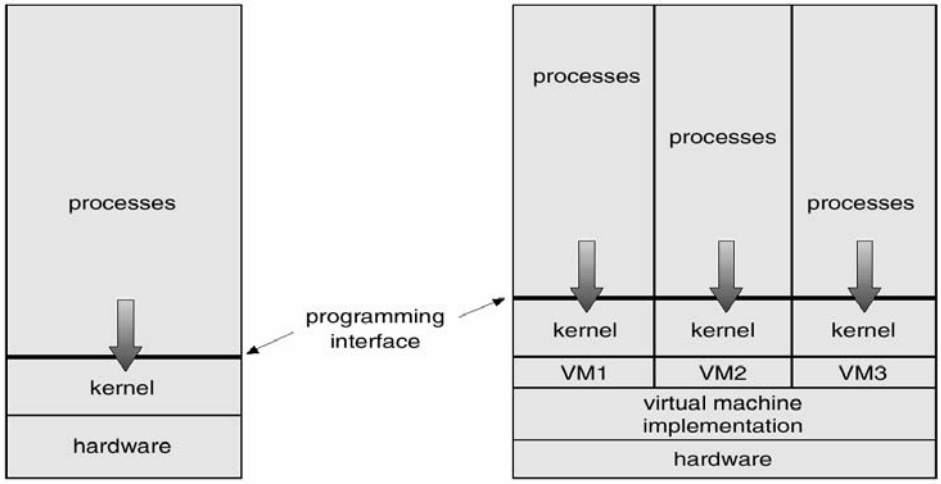
- Các phiên bản mới của Unix được thiết kế sử dụng phần cứng phức tạp hơn, do đó hệ điều hành được chia thành nhiều phần nhỏ hơn

- Việc chia hệ thống thành nhiều phần nhỏ nó che dấu thông tin, không cho chương trình của người sử dụng có thể cài đặt những hàm truy xuất cấp thấp, thay vào đó là các lớp giao tiếp bên trong.
- Hệ điều hành chia thành nhiều lớp. Lớp dưới cùng là phần cứng, lớp trên cùng là giao tiếp với người sử dụng.
- Một lớp của hệ điều hành bao gồm một số cấu trúc dữ liệu và các hàm có thể được gọi bởi lớp ở phía trên và bản than nó gọi những chức năng của lớp bên dưới. Mỗi lớp cài đặt chỉ sử dụng những thao tác do lớp dưới cung cấp.

Lớp 6: Chương trình của người sử dụng
Lớp 5: Driver thiết bị và bộ lập lịch
Lớp 4: Bộ nhớ ảo
Lớp 3: Kênh nhập xuất
Lớp 2: Lập lịch CPU
Lớp 1: Thông dịch các chỉ thị
Lớp 0: Phần cứng

II.6 MÁY ẢO

- Một máy ảo cung cấp một giao diện giống hệt các lớp phần cứng.
- Hệ điều hành tạo ra các tiến trình ảo, mỗi việc thực hiện trên bộ xử lý với bộ nhớ ảo của nó.
- Tài nguyên của máy tính thật được chia xẻ để tạo ra máy ảo.
- Lập lịch CPU cũng được tạo ra như là người sử dụng có bộ xử lý riêng.
- Spooling và hệ thống file được cung cấp một card reader ảo và một line máy in ảo.



Hình 2.4

- Thuận tiện, Bất lợi của máy ảo:
  - Khái niện máy ảo đưa ra chế độ bảo vệ tài nguyên hệ thống hoàn chỉnh từ các máy ảo khác. Các máy ảo là độc lập nhau không trực tiếp chia xẻ tài nguyên.

- Một hệ thống máy ảo là một phương tiện hoàn chỉnh để nghiên cứu hệ điều hành và phát triển nó. Các hệ thống được phát triển trên máy ảo thay vì trên máy thật bởi vậy hệ thống trên máy thật không bị phá vỡ.
- Khái niệm máy ảo đôi khi cũng khó thực hiện các yêu cầu chính xác như trên máy thật.

II.7 QUÁ TRÌNH NẠP HỆ ĐIỀU HÀNH

- Khi bật máy,c*hương trình Bootstrap* – (là đoạn mã lưu trữ trong ROM) được thi hành để kiểm tra các thiết bị máy tính có hoạt động tốt không. Nếu mọi thiết bị đều sẵn sàng thì chương trình này đọc bootsector(đĩa mềm) hay masterboot(đĩa cứng) vào bộ nhớ tại địa chỉ 0:7C00h và trao quyền điều khiển tại đây.
- Trong bootsector bao gồm bảng tham số đĩa để mô tả tổ chức vật lý và tổ chức logic trên đĩa và một chương trình mồi hệ điều hành.
- Từ đó chương trình mồi hệ điều hành trong bootsector sẽ nạp các phần còn lại của hệ điều hành (kernel) vào bộ nhớ và hệ điều hành bắt đầu hoạt động.
- Đối với đĩa cứng thì có dung lượng lớn, do đó một hệ điều hành có thể không sử dụng hết dung lượng trên đĩa. Một đĩa cứng có thể chứa đồng thời nhiều hệ điều hành, mỗi hệ điều hành được lưu trữ mỗi phân vùng riêng biệt và mỗi phân vùng gọi là partition.
- Masterboot là sector đầu tiên trên đĩa cứng vật lý và bao gồm một bảng mô tả các partition hiện có trên đĩa như là sector bắt đầu, sector kết thúc, thuộc tính của các partition tương ứng. Ngoài ra masterboot còn chứa một đoạn chương trình thực hiện đọc bảng tham số partition để xác định partition active. Từ đó nạp bootsector của partition active đó vào bộ nhớ và chuyển quyền điều khiển cho chương trình mồi hệ điều hành trong bootsector.

CÂU HỎI

1. Khi xây dựng một hệ điều hành phải bao gồm những thành phần nào? Trình bày các chức năng của các thành phần đó? Liệt kê các thành phần không thể thiết được của một hệ điều hành đa nhiệm?
2. Các chương trình ứng dụng giao tiếp với hệ điều hành thông qua thành phần nào của hệ điều hành?
3. Lời gọi hệ thống được cài đặt trong hệ điều hành bằng kỹ thuật gì? Việc truyền tham số cho lời gọi hệ thống bằng kỹ thuật nào? Trình bày kỹ thuật đó cho việc truyền tham trị và tham biến ?
4. Trình bày quá trình hệ điều hành được khởi động trên một hệ thống máy tính? Nếu sector đầu tiên trên ổ đĩa bị hỏng thì việc cài đặt hệ điều hành lên ổ đĩa đó có được thực hiện không? Giải thích?
5. Trên một ổ đĩa cứng có thể thể cài đặt được nhiều hệ điều hành không ? Vì sao?
6. Vì sao có loại virus được gọi là B-Virus? Các biện pháp phòng và diệt loại B-Virus?
7. Cài đặt phần mềm máy ảo *VMware workstation* ?

CHƯƠNG III
GIỚI THIỆU MỘT SỐ HỆ ĐIỀU HÀNH

III.1 Hệ điều hành MS-DOS

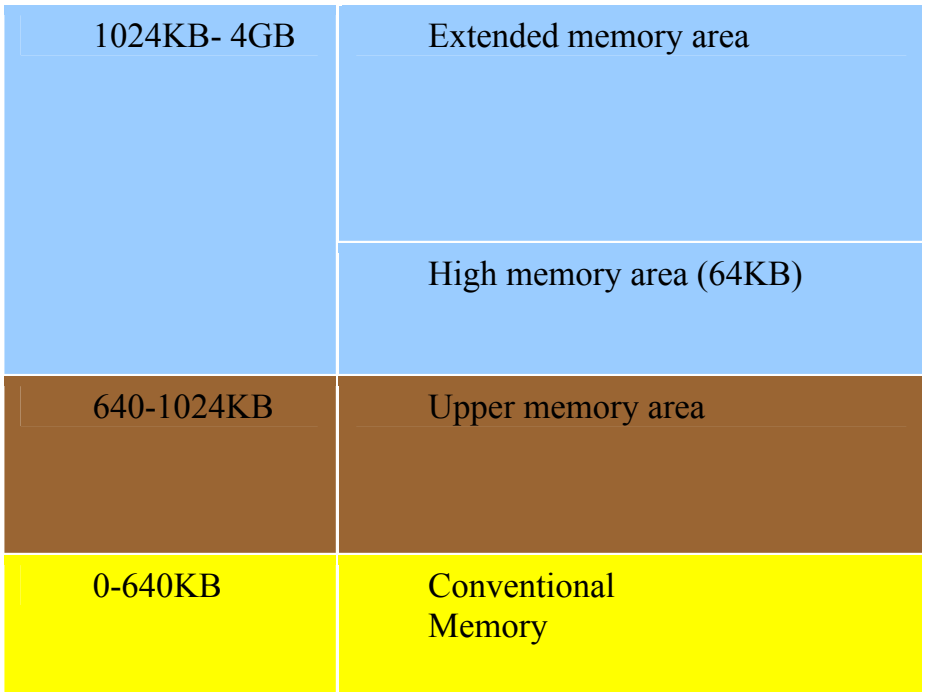
III.1.1 Giới thiệu

- MS-DOS là một hệ điều hành đầu tiên chạy trên máy PC và được thiết kế bởi Microsoft
- MS-DOS là hệ điều hành đơn nhiệm, một người □ung , và có cấu trúc đơn giản nên yêu cầu cấu hình máy thấp, bộ nhớ chính 640KB, giao diện theo cơ chế dòng lệnh
- MS-DOS có thể được cài đặt trên đĩa mềm hoặc đĩa cứng

III.1.2 Cấu trúc hệ điều hành MS-DOS

- Tổ chức của MS-DOS bao gồm:
  - Chương trình mồi hệ điều hành được nạp vào bootsertor
  - Chương trình shell: giao tiếp giữa người sử dụng và hệ điều hành.
  - Chương trình chứa các chức năng cơ bản của hệ điều hành
  - Chương trình nhập xuất
  - Hệ thống các chương trình tiện ích.

- Cấu trúc bộ nhớ



Hình 3.1

- Nội dung hệ điều hành
  - IO.SYS : hệ thống nhập xuất
  - MSDOS.SYS: hệ thống tập tin, giao tiếp dòng lệnh
  - CONFIG.SYS: cài đặt driver thiết bị

- COMMAND.COM : tập lệnh nội trú(internal)
- AUTOEXEC.BAT : chứa tập lệnh DOS chạy tự động khi hệ điều hành bắt đầu

III.1.3 Lịch sử phát triển

- Năm 1980 IBM sản xuất máy tính cá nhân đầu tiên với bộ xử lý 8088, 16 bit và yêu cầu Microsoft thiết kế một hệ điều hành. MS-DOS ra đời 1981
- Version 1.0 ra đời 8/1981 bao gồm 4000 dòng hợp ngữ.
  - Quản lý 12K bộ nhớ
  - Được tổ chức thành 3 tập tin : IBMIO.COM- chứa hệ thống nhập xuất, IBMMS-DOS chứa hệ thống tập tin trên đĩa, chương trình giao tiếp, COMMAND.COM chứa các lệnh xử lý.
  - Được cài đặt trên đĩa mềm hai mặt 320KB

- Version 2.0 : ra đời 3/1983 cung cấp hệ thống tập tin cấp bậc
  - Sử dụng đĩa cứng 10MB
  - Cho phép cài đặt các bộ điều khiển thiết bị trong tập tin CONFIG.SYS

- Version 3.0 : ra đời 8/1984 quản lý được đĩa cứng 20MB, đĩa mềm 1.2MB, sử dụng đĩa ảo – lấy RAM làm đĩa ảo
- Version 4.0 : ra đời 7/1988 hỗ trợ đĩa cứng lớn hơn 32MB đến 2GB, sử dụng vùng nhớ mở rộng đĩa ảo
- Version 5.0 : ra đời 4/1991 tận dụng bộ nhớ mở rộng dùng để lưu trữ các device driver.
- Version 6.0: 1993 có các đặc điểm:
  - Tăng dung lượng đĩa sử dụng DBLSPACE
  - Tạo bộ nhớ cho đĩa với SmartDrv
  - Tối ưu bộ nhớ dùng Memmaker
  - Cứu tập tin bằng tiện ích Undelete
  - Kiểm tra đĩa bằng Scandisk

III.1.4 Cài đặt hệ điều hành

- Khởi động hệ điều hành DOS và thực hiện lệnh Format ổ đĩa: /s
- Hoặc Sys ổ đĩa:

III.1.5 Tập lệnh

- Lệnh thông tin hệ thống
  - DATE: Thiết lập hoặc hiển thị ngày hệ thống
  - TIME: Thiết lập hoặc hiển thị giờ hệ thống
  - PROMPT: Định nghĩa dấu nhắc hệ thống
  - SET: Định nghĩa biến môi trường
  - VER: Hiển thị phiên bản hệ điều hành

- Lệnh làm việc với đĩa
  - DISKCOPY: Sao chép đĩa mềm

- FORMAT: Định dạng đĩa
- LABEL: xem, đặt nhã đĩa
- VOL: Hiển thị nhãn và số serial của đĩa

- Lệnh làm việc với thư mục
  - Ổ đĩa:↵ Chuyển ổ đĩa hiện hành
  - CHDIR hoặc CD : chuyển thư mục hiện hành
  - DIR: hiển thị nội dung thư mục
  - MKDIR hoặc MD: tạo thư mục mới
  - PATH: Định nghĩa đường dẫn
  - RMDIR hoặc RD: xoá thư mục khác rỗng
  - TREE: Hiển thị cấu trúc cây thư mục

- **Thiết lập môi trường**
  - BUFFERS: Mô tả số buffer đĩa
  - DEVICE: Cài đặt device driver
  - FILES: Số tập tin tối đa được mô tả
  - LASTDRIVER: Số ổ đĩa tối đa

- **Sử dụng tập tin Batch**
  - CALL : Gọi tập tin batch khác với tham số
  - ECHO: Hiển thị thông điệp lên màn hình
  - FOR: Thực hiện lệnh DOS nhiều lần
  - GOTO: Nhảy đến một vị trí thi hành lệnh khác trong tập tin batch
  - IF: Kiểm tra điều kiện thi hành lệnh batch
  - PAUSE: Dừng lệnh bacth
  - REM: Không thi hành lệnh batch

- **Ví dụ tạo ra đĩa khởi động cài đặt driver CDROM**

- Copy các tập tin himem.sys, oakcdrom.sys, aspi2dos.sys, mscdex.exe, guest.exe, smartdrv.exe vào đĩa mềm.
- Tạo tập tin config.sys

[memu]
menuitem=NONE, Khoi dong khong co CDROM
Menuitem=CDROM, Khoi dong co CDROM
Menuitem=USB, Khoi dong co USB
[common]
Device =himem.sys /testm:off
Lastdrive=z
[CDROM]
Device=oakcdrom.sys /d:mscd001
[USB]
Device=aspi2dos.sys /int/all
[NONE]

- **Tạo tập tin autoexec.bat**

```
@echo off
Goto %config%
:CDROM
Mscdex.exe /d:mscd001
Goto end
:USB
Guest.exe
Goto end
:NONE
:END
Smartdrv.exe
```

III.2 Hệ điều hành Windows

III.2.1 Giới thiệu

- Windows được thiết kế bởi Microsoft.
- Windows là hệ điều hành đa nhiệm, nhiều người dùng.
- Giao diện người dung thân thiện, chế độ đồ họa
- Cài đặt và thay đổi cấu hình hệ thống dễ dàng, khái niệm Plug and play
- Có tính ổn định cao, nếu có tiến trình nào bị hỏng thì hệ thống huỷ bỏ tiến trình đó mà không ảnh hưởng đến toàn bộ hệ thống.
- Có tính bảo mật cao.

III.2.2 Lịch sử phát triển

- 11/1983 Microsoft tuyên bố sự ra đời hệ điều hành Windows
- 11/1987 version 2.0 ra đời có sự thay đổi về giao diện, cửa sổ có thể chồng lên nhau, menu, dialog box
- 5/1990 version 3.0 cho phép truy cập đến bộ nhớ 16Mb bộ nhớ. Có Program Manager, Task Manager, File Manager.
- 4/1992 version 3.1 hỗ trợ Multimedia(sound& music). Windows trở thành HĐH chiến lược của Microsoft.
- 9/1995 Windows 95. Độc lập thiết bị, hỗ trợ CD-Rom, kỹ thuật Plug and play, hỗ trợ mạng cục bộ, truy xuất từ xa. Cải tiến giao diện đồ họa.
- 1998 Windows 98 ra đời với hệ thống tập tin FAT32, hỗ trợ tên tập tin dài. Tiếp theo WindowsMe được tích hợp rất nhiều driver thiết bị
- 2000 Windows 2K ra đời là sự lai ghép Windows 98 và Windows NT hỗ 2 hệ thống tập tin FAT32 và NTFS. Hỗ trợ tính bảo mật trên hệ thống tập tin NTFS. Hệ điều hành mạng Client/server. Có 2 phiên bản Windows 2K server và professional
- 2001 : Windows Xp ra đời với cải tiến giao diện. Hỗ trợ mạng Internet. Đây là hệ điều hành thông dụng nhất hiện nay.

III.2.3 Các tiện ích của Windows

- Quản lý các driver thiết bị gắn vào máy tính: **Device Manager**

- Xem cấu hình máy tính: **dxdiag**
- Xem ,cài đặt các thông số máy tính: **control panel**
- Khởi động các dịch vụ HĐH: **Administrator tool – Service**
- Cài đặt thêm các thành phần HĐH, gỡ bỏ các chương trình: **Administrator Tool-Add or remove programs**
- Quản lý đĩa: **Administrator Tool – Computer Management – Disk management**
- Tạo tài khoản người dùng: **Administrator Tool – Computer Management-Local users and Groups**
- Xem huỷ các tiến trình đang chạy : **Task manager**
- Bảo mật folder trên hệ thống NTFS
- Chia xẻ folder

III.3 Hệ điều hành Linux

III.3.1 Đặc điểm

- Multi: Tasking, Threading, User
- Multi-platform: Chạy trên nhiều nền tảng phần cứng (khác Intel).
- Open Source: Bao gồm cả kernel, drivers, công cụ phát triển
- Hỗ trợ nhiều hệ thống File: Minix-1, Xenix, System V , MS-DOS, VFAT, FAT-32, ISO 9660 (CD-ROMs). EXT, và EXT2
- Multiple Networking Protocols: Các giao thức nền tảng được hỗ trợ bởi Kernel như: TCP, Ipv4, Ipv6, AX.25, X.25, IPX, v.v…
- Multiprocessor Simultaneous Multiprocessing (SMP)
- Virtual Memory PagingMemory Protection: Hệ thống và các quá trình được bảo vệ lẫn nhau do đó không quá trình nào có thể làm cho toàn hệ thống sụp đổ.
- TCP/IP Networking: bao gồm ftp, telnet,…
- Client and Server Support: Bao gồm Netware, và Windows (SMB)
- Ký hiệu Linux Kernel
    - Các phiên bản của Linux. Các phiên bản của HDH Linux được xác định bởi hệ thống số dạng X.YY.ZZ. Nếu YY là số chẵn => phiên bản ổn định. YY là số lẻ => phiên bản thử nghiệm
    - Ví dụ:
    - Kernel 2.4.2
    - 2 là Số chính
    - .4 là số phụ , phiên bản ổn định
    - .2 Patch Level, phiên bản ổn định (nếu số lẻ là phiên bản đang thử nghiệm)

III.3.2 Lịch sử phát triển

- 1969 Kend Thompson thiết kế môi trường nghiên cứu và phát triển chương trình đây chính là tiền thân HĐH Unix. Unix được viết bằng hợp ngữ
- 1973 Unix được viết lại bằng ngôn ngữ C, dễ hiểu hơn

1975 Mã nguồn của Unix được cung cấp cho các trường đại học. Unix được nhiều công ty, tổ chức phát triển

- Năm 1991 Linus Torvalds, sinh viên của đại học tổng hợp Helsinki, Phần lan, bắt đầu xem xét Minix, một phiên bản của Unix làm ra với mục đích nghiên cứu cách tạo ra một hệ điều hành 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 định của mình về Linux
- 1/1992, Linus cho ra version 0.12 với shell và C compiler. Linus không cần Minix nữa để recompile HDH của mình. Linus đặt tên HDH của mình là Linux
- 1994, phiên bản chính thức 1.0 được phát hành

III.3.3 Cài đặt hệ điều hành

- Các thao tác chuẩn bị cài đặt
  - Kiểm tra phần cứng máy tính
    - ✓ CPU, RAM, HDD: Tùy thuộc vào phiên bản Linux mà ta sẽ cài đặt.
    - ✓ Ví dụ: Để cài đặt RedHat Linux 6.0 ta cần cấu hình phần cứng tối thiểu như sau: CPU Intel 133MHz, 16MB RAM, 2Gb HDD Kiểm tra phần mềm
  - Phiên bản Linux sẽ cài đặt.
  - Ví dụ: RedHat Linux 7.3 (phiên bản này yêu cầu CPU 233MHz ↑, 64MB RAM, 4Gb HDD
- Phân hoạch đĩa cứng
  - Đối với hệ điều hành Linux nó đòi hỏi phải có ít nhất 2 partition của đĩa cứng để có thể cài đặt thành công.
  - Partition thứ nhất: Dùng để chứa hệ điều hành. Dung lượng cho partition này tuỳ theo các package mà bạn cài đặt, thông thường khoảng 2Gb là đủ.
    - ✓ Swap Partition: Dung lượng cho parttion chỉ cần bằng dung lượng của RAM là vừa đủ
    - ✓ Đặc biệt đối với các hệ thống Linux mà sau này muốn cài đặt hệ quản trị CSDL Oracle lên thì ta phải cho swap space lớn hơn hoặc bằng 500MB
- Các chế độ cài đặt
  - Server
  - Workstation
  - Custom
  - Upgrade

III.3.4 Tập lệnh

- Quá trình khởi động RedHat Linux
  Tập tin đầu tiên mà hệ điều hành xem xét đến là /etc/inittab

```
# Default runlevel. The runlevels used by RHS are:
#0 – halt (Do NOT set initdefault to this)
#1 – Single user mode
#2 – Multiuser, without NFS
#3 – Full multiuser mode
#4 – unused
```

#5 – X11
#6 – reboot (Do NOT set initdefault to this) id:3:initdefault:

- Tập lệnh cơ bản
  - Login : Đăng ký sử dụng
  - Who : cho biết thông tin về người sử dụng
  - Date: thông báo về thời gian
  - Exit, logout : chấm dứt phiên làm việc
  - Passwd: thay đổi password
  - Man: giúp đỡ
  - Ls : Liệt kê nội dung thư mục
  - Cp: sao chép tập tin
  - Mv: Đổi tên một tập tin
  - Rm: xóa tập tin
  - Cat: Hiển thị tập tin
  - Pwd: Cho biết thư mục hiện hành
  - Cd: Thay đổi thư mục hiện hành
  - Rmdir: xóa thư mục
  - Chmod: thay đổi quyền tập tin
  - Chgrp: thay đổi nhóm
  - Chown: thay đổi người sở hữu
  - df,du: thông hệ thống tập tin
  - Useradd: Tạo người dùng
  - Usergroup:Tạo nhóm người dùng
  - Su: chuyển đổi người dùng
  - Biên dịch chương trình c: gcc

    gcc -o output input.c
  - chown: change owner. Thay đổi quyền sở hữu tập tin cho user khác. Chỉ được chạy bởi root

    chown user1 hello.txt
  - chgrp: change group. Thay đổi quyền sở hữu tập tin cho nhóm khác . Chỉ được chạy bởi root

    chgrp users hello.txt
  - chmod: change mode: Thay đổi quyền của file, file của ai người đó mới được thay đổi (ngoại trừ root, có quyền trên tất cả các file

    chmod g+r hello.txt
    chmod o-x hello.txt
    chmod u+xwr hello.txt

CÂU HỎI VÀ BÀI TẬP

1. Trình bày các đặc điểm của hệ điều hành DOS, Windows, Linux? Sực khác nhau giữa các hệ điều hành trên.
2. Cài đặt và sử dụng tập lệnh hệ điều hành MS-DOS?
3. Tạo đĩa CDRom có khả năng khởi động hệ điều hành DOS nhận diện được CDROM và USB ?
4. Cài đặt và sử dụng tập lệnh hệ điều hành Windows?
5. Cài đặt và sử dụng tập lệnh hệ điều hành Linux?
6. Cài đặt nhiều hệ điều hành trên một máy tính?

CHƯƠNG IV
HỆ THỐNG QUẢN LÝ TẬP TIN

IV.1 Khái niệm tập tin – 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.
  - Các thông tin trong tập tin là bền vững không bị ảnh hưởng bởi các xử lý ngoại trừ người sử dụng muốn xóa.
  - Tập tin được quản lý bởi hệ điều hành.
- Thư mục
  - Thư mục lưu trữ các tập tin theo một qui định.
  - Một số hệ thống coi thư mục cũng như là tập tin.
  - Hệ thống quản lý tập tin
  - Các tập tin được quản lý bởi hệ điều hành với một cơ chế riêng gọi là hệ thống quản lý tập tin.
- Hệ thống quản lý tập tin bao gồm: cách hiển thị , các yếu tố cấu thành tập tin, cách truy xuất, các thao tác trên tập tin và thư mục,…

IV.2 Mô hình quản lý và tổ chức tập tin

- Tập tin
  - Tên tập tin:
    - Mỗi tập tin được quản lý bằng một tên.
    - Cách đặt tên tập tin mỗi hệ diều hành là khác nhau.
  - Cấu trúc tập tin:
    - Dãy các byte không có cấu trúc.
    - Dãy các record có chiều dài cố định
  - Kiểu tập tin: Các hệ điều hành hỗ trợ cho nhiều loại tập tin khác nhau
    - Tập tin thường: là tập tin text hay nhị phân chứa các thông tin của người sử dụng.
    - Thư mục: là một loại tập tin hệ thống dùng lưu trữ cấu trúc hệ thống tập tin
    - Tập tin có ký tự đặc biệt: liên quan đến nhập xất thông qua các thiết bị nhập xuất
  - Thuộc tính của tập tin: Các thông tin về tập tin gọi là thuộc tính tập tin. Thuộc tính tập tin thường là các thông in:
    - Bảo vệ: bảo vệ việc truy xuất từ người sử dụng
    - Mật khẩu: Mật khẩu cần thiết khi truy xuất.
    - Người tạo: Chỉ danh người tập tin.
    - Người sở hữu: Chỉ danh người sở hữu hiện tại.
    - Chỉ đọc: 0: đọc ghi, 1: chỉ đọc
    - Ẩn: 0 bình thường, 1 không hiển thị khi liệt kế
    - hệ thống: 0 bình thường, 1 tập tin hệ thống .

- Khoá: 0 không khóa, 1 bị khóa
- Độ dài record: số byte trong một record
- Thời gian tạo: ngày , giờ tạo tập tin
- Thời gian truy xuất sau cùng: ngày , giờ truy xuất gần nhất
- Thời gian thay đổi cuối cùng: ngày, giờ thay đổi tập tin
- Kích thước hiện thời: Số byte tập tin

- Thư mục
  - Hệ thống thư mục cấp bậc:
    - Một thư mục thường chứa các entry. Mỗi entry thể hiện cho một tập tin(chứa các thuộc tính tập tin)
    - Số lượng thư mục trên mỗi hệ điều hành là khác nhau.
    - Một số hệ thống chỉ có một thư mục duy nhất, nhưng một số hệ thống có thư mục gốc, trong thư mục gốc có các thư mục con và trong các thư mục con lại có các thư mục con nữa.
  - Đường dẫn
    - Trong hệ thống tổ chức thư mục cấp bậc theo hình cây có hai cách để xác định một tập tin: đường dẫn thuyệt đối, đường dẫn tương đối.
    - Trong hầu hết các hệ điều hành tổ chức thư mục "." và ".." để chỉ ra thư mục hiện hành và thư mục cha.

IV.3 Các chức năng hệ thống tập tin

Một hệ thống tập tin thông thường bao gồm các chức năng sau:

- Tạo tập tin
- Xoá tập tin
- Mở tập tin
- Đọc, ghi tập tin
- Tìm kiếm tập tin
- Đổi tên tập tin
- Tạo thư mục
- Xóa thư mục
- Đổi tên thư mục

IV.4 Cài đặt hệ thống tập tin

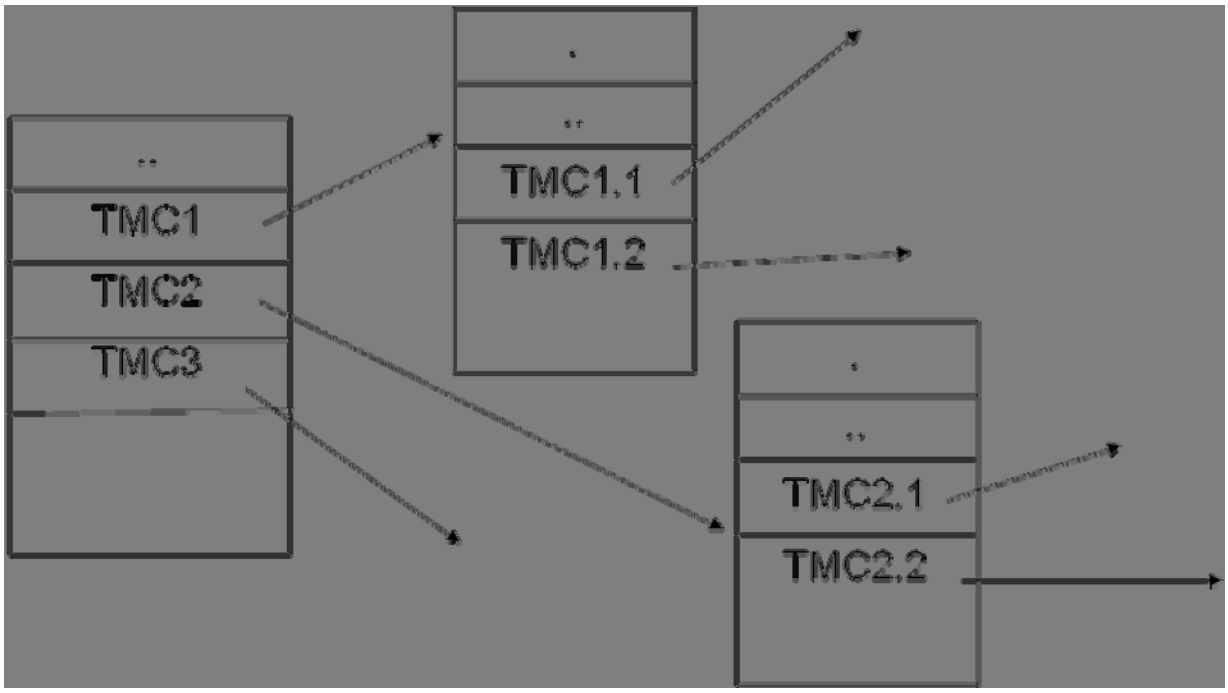
- **Cấu trúc tập tin**
  - Mỗi tập tin hay thư mục (tập tin đặc biệt) được thể hiện bằng một Entry tập tin. Cấu trúc của Entry tập tin lưu trữ thông tin về tập tin tương ứng.

  - Trước khi tập tin được đọc, ghi hệ thống phải biết đường dẫn do người sử dụng cung cấp từ đó định vị được cấu trúc entry của tập tin

  - Cấu trúc một entry tập tin:

Hình 4.1

- **Cấu trúc thư mục**
  - Thư mục chứa các tập tin do đó chứa các entry của tập tin, kích thước mỗi Entry tuỳ thuộc mỗi hệ thống
  - Trong hệ thống quản lý tập tin cần phải định vị cấu trúc thư mục gốc, và các thư mục con

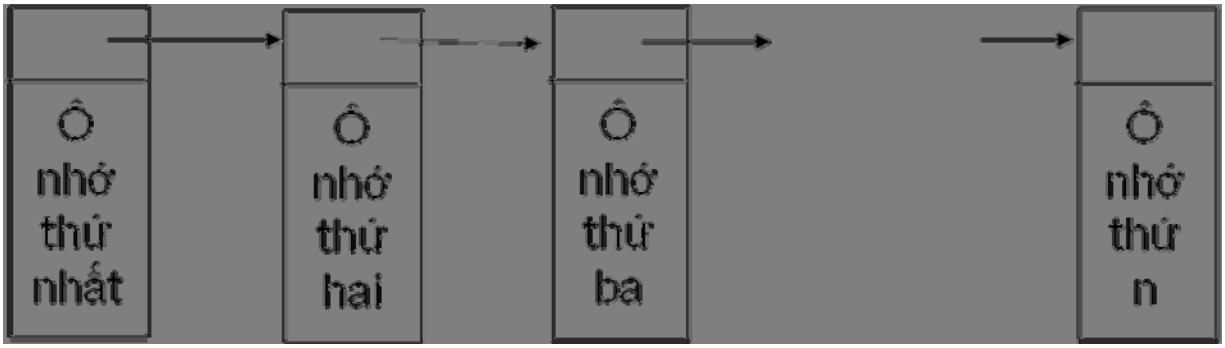


Hình 4.2

- **Quản lý vùng nhớ còn trống trên đĩa**

Hệ thống quản lý tập tin còn phải thực hiện lưu trữ các thông tin về các ô nhớ đã sử dụng chưa? Các ô nhớ đã sử dụng thuộc về tập tin, thư mục nào?

- Phương pháp định vị liên tiếp: lưu trữ nội dung tập tin trên một dãy các khối liên tiếp nhau.
  - Ưu điểm : dễ cài đặt, dễ thao tác trên các tập tin
  - Khuyết điểm: Xử lý phức tạp khi kích thước tập tin thay đổi, sự phân mảnh trên đĩa gây lãng phí
- Phương pháp định vị bằng danh sách liên kết: Không có sự phân mảnh vì các ô nhớ được cấp phát hết, truy xuất chậm vì các ô nhớ chứa nội dung tập tin nằm rải rác.



Hình 4.3

IV.5 Hệ thống tập tin MS-DOS

Cấu trúc tổng quát hệ thống tập tin MS-DOS bao gồm các cấu trúc:

- **Bảng tham số đĩa**: lưu trữ các thông tin về đĩa và các thông tin cần thiết cho hệ thống tập tin.
- Cấu trúc **FAT**(File Allocation Table): Lưu trữ các thông tin địa chỉ các ô nhớ còn trống, các ô nhớ nào thuộc về mỗi tập tin,…
- Cấu trúc **thư mục gốc**(ROOT):Cấu trúc thư mục gốc bao gồm các Entry của tập tin, thư mục con.
- Vùng **lưu trữ dữ liệu**(Data): Lưu trữ nội dung tập tin và các cấu trúc thư mục con. Cấu trúc thư mục con tương tự cấu trúc thư mục gốc.

Hình 4.5

- **Bảng tham số đĩa(BPB):** nằm trong bootsec bắt đầu tại địa chỉ offset 0 và chứa các thông tin:

Offset	**Kích thước(B)**	**Tên, ý nghĩa**	
+0h	3	JMP	Lệnh nhảy đến đoạn mồi HĐH
0h+3	8	Version	Tên Cty, Version của HĐH
+0Bh	2	SecSiz	Số byte trong một sector
+0Dh	1	ClustSiz	Số sector của cluster

+0Eh	2	ResSec	Số Sector trước bảng FAT
+10h	1	FatCnt	Số bảng FAT cài đặt
+11h	2	RootSiz	Số Entry tối đa của ROOT
+13h	2	TotSecs	Tổng số sector đĩa <32MB
+15h	1	Media	Byte chỉ danh đĩa
+16h	2	FatSiz	Số sector trong bảng FAT
+18h	2	TrkSecs	Số sector trên mỗi Track
+1Ah	2	HeadCnt	Số đầu đọc ghi
……			
+20h	4	TotSec	Tổng số sector đĩa >32MB
…….			
+3eh			Bắt đầu đoạn CTmồi HĐH

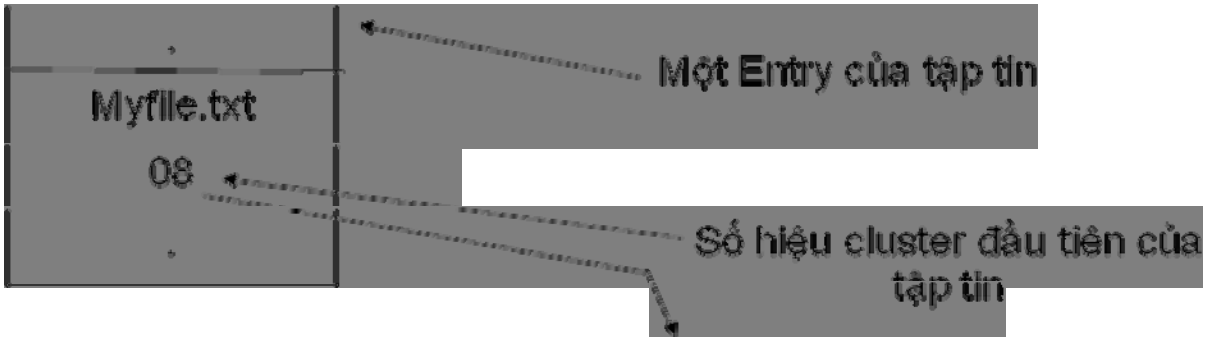
- Đối với đĩa cứng sector đầu tiên là masterboot chứa bảng tham số partition. Địa chỉ offset bảng partition bắt đầu tại 01BEh . Với một đĩa cứng có thể chi thành 4 partition, mỗi partition được thể hiện bằng một Entry(16byte) có cấu trúc như sau:

Offset	**Kích thước(B)**	**Tên, ý nghĩa**	
+0h	1	Boot	=0 không Active, =80h Active
+1h	1	HdBeg	Số mặt bắt đầu
+2h	2	SecCylBeg	Số cylinder bắt đầu(10bit) và sector bắt đầu (6bit)
+4h	1	Sys	=0Unknown, =1: Dos FAT 12bit, ; =4: Dos FAT 16bit
+5h	1	HdEnd	Số mặt kết thúc

+6h	2	SecCylEnd	Số cylinder kết thúc(10bit) và sector kết thúc (6bit)
+8h	4	SecLogic	Sector bắt đầu tương đối
+0Ch	4	TotSecs	Tổng số sector của partition

- **Cấu trúc Fat(file allocation table)**
  - Khái niệm Cluster: Khi đĩa được format đơn vị nhỏ nhất trên đĩa là Sector. Đối đĩa cứng lớn có nhiều sector mà Dos không thể quản lý được. Trong trường hợp này để giảm số sector cần quản lý bằng cách định nghĩa cluster là tập hợp các sector. Lúc này Dos chỉ quản lý Cluster thay vì sector.
  - Số hiệu các entry trong bảng Fat thể hiện cho các cluster có số hiệu tương ứng trong vùng dữ liệu.
  - Fat thể hiện thông tin về các cluster còn trống hay không? Và các cluster đã sử dụng thuộc về tập tin nào.
  - Do Trong Fat 2 phần tử đầu tiên dành riêng không sử dụng nên số hiệu các entry được đánh số từ 2 trở đi. Chính vì vậy số hiệu cluster cũng được đánh số từ 2 trở đi.
  - Fat có 3 loại : fat 12bit, fat 16bit, fat 32bit. Đối với Fat 12bit thì kích thước mỗi Entry trong Fat là 12 bit và quản lý được số cluster tối đa là (2^{12} -2). Tương tự đối với Fat 16bit và Fat 32bit.
  - Fat 12bit dùng cho đĩa <32MB, Fat 16Bit dùng đĩa <2GB và Fat 32Bit dùng cho đĩa cứng > 2GB(HĐH Windows)
  - **Nội dung của Fat**
    - Dos quản lý tập tin bằng cách giá trị entry của cluster này chứa giá trị là số thứ tự entry tiếp theo nó, cứ thế các cluster của một tập tin tạo thành một chuỗi cho đến khi gặp dấu hiệu kết thúc tập tin.
    - Tuỳ thuộc vào loại fat 12bit hay 16bit các entry có giá trị như sau:
      - 0(000) : Cluster tương ứng số hiệu entry còn trống
      - (0)002- (F)FEF : Cluster đang chứa dữ liệu của một tập tin nào đó, giá trị là của nó là số hiệu cluster kế tiếp
      - (F)FF0 – (F)FF6 : Dành riêng không sử dụng
      - (F)FF7 : Cluster hỏng
      - (F)FF8- (F)FFF : Cluster cuối cùng của tập tin

Một ví dụ nội dung của Fat:



	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f
00			03	04	05	Ff	00	00	09	0a	0b	15	00	00	00	00
01	00	00	00	00	00	16	17	19	f7	1a	1b	Ff	00	00	00	00
02	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...

Hình 4.5

Tập tin myfile.txt dài 10 cluster: 8,9,0a,0b,15,16,17,19,1a,1b
Cluster 18h : đánh dấu hỏng
Cluster 2,3,4,5: thuộc một tập tin nào đó
Các cluster khác còn trống

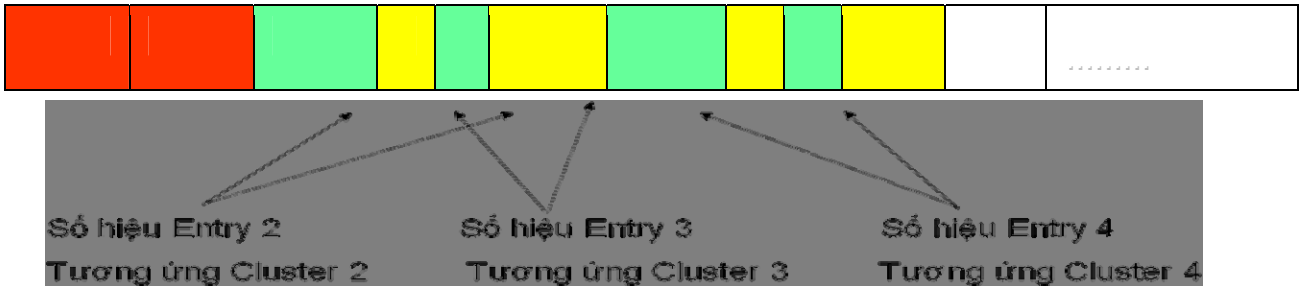
Tìm các cluster của một tập tin

```
PointType ReadEntryFile(unsigned int ClustBegin)
{
        PointType ListClust, last;
        ListClust=last=NULL;
        unsigned clust;
        clust=ClustBegin;
        while(1)
        {
                InsertLast(ListClust, clust);
                clust=NextEntry(clust)
                if (clust==0x0fff)
                        break;

        }
```

```
    return ListClust;
}
```

Đọc nội dung các Entry trong Fat:

- Nếu Fat 16 bit thì địa chỉ entry kế tiếp bằng địa chỉ entry hiện thời +2
- Nếu Fat 12 bit : Xét 3 byte một : số hiệu entry chẵn là 12 bit thấp và số hiệu entry lẻ là 12 bit cao
-



Hình 4.6

Hình 4.7

Các biến dùng chung:

unsigned X, X1, Addr;

Trường hợp số hiệu Entry Chẵn

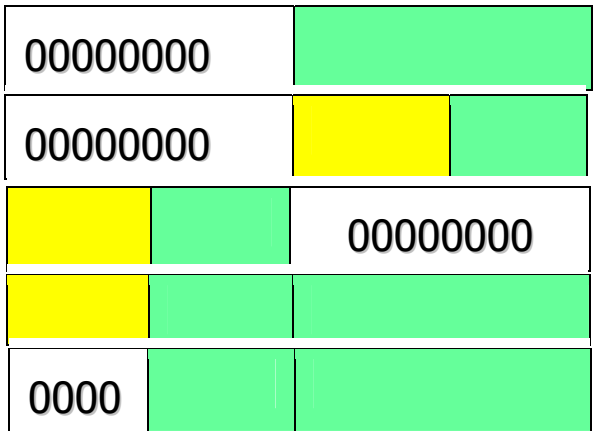
X = FAT[Addr];

X1= FAT[Addr+1];

X1= X1 <<8;

X = X + X1;

X = X & 0x0FFF;



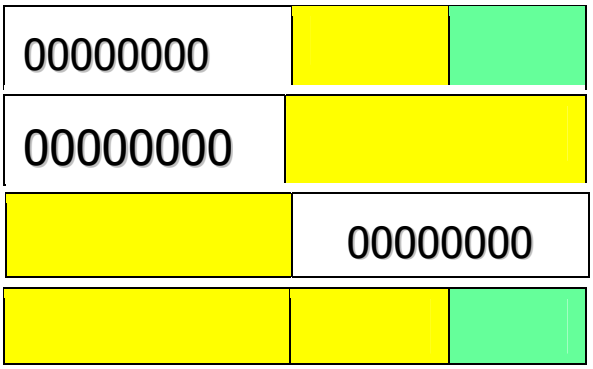
Trường hợp số hiệu Entry lẻ

X = FAT[Addr];

X1= FAT[Addr+1];

X1= X1 <<8;

X = X + X1;



X = X >>4;

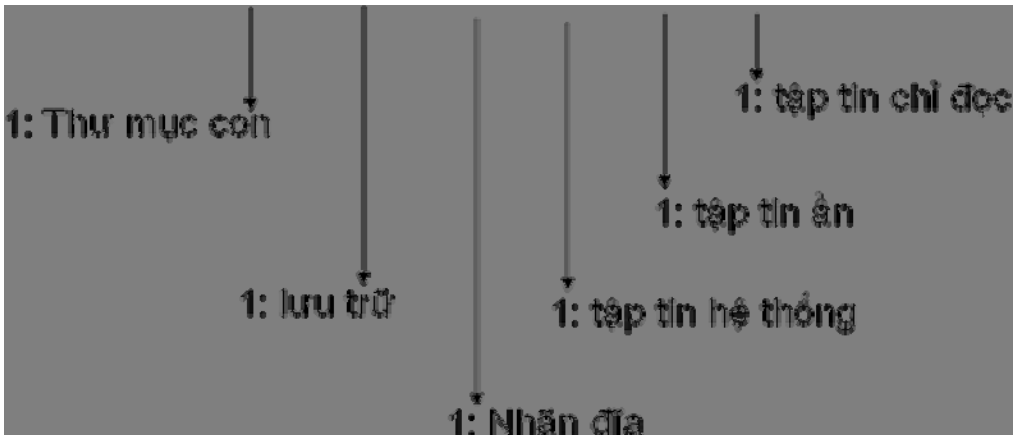
Hàm đọc nội dung Entry Fat 12bit

```
unsigned  NextEntry ( unsigned Index)
{       unsigned X, X1, Addr;
        Addr= (Index *3 )/ 2;
        X = FAT[Addr];
        X1= FAT[Addr+1];
        X1= X1 <<8;
        X = X + X1;
        if ((Index %2)==0)
                X= X & 0X0FFF;
        else
                X = X >>4;
        return X;
}
```

- **Cấu trúc một Entry Trong thư mục gốc hay thư mục con:** Mỗi Entry thể hiện cho một tập tin hay thư mục con và lưu trữ các thông tin cần thiết. Kích thước Entry là 32 byte

Offset	**Kích thước**	**Nội dung**
+0h	8	Tên tập tin hay thư mục
+8h	3	Phần mở rộng tên tập tin
+0bh	1	Thuộc tính tập tin
+0ch	0ah	Dành riêng không sử dụng
+16h	2	Thời gian tạo
+18h	2	Ngày tạo
+1ah	2	Số hiệu cluster đầu tiên
+1ch	4	Kích thước tập tin (bytes)

- Byte đầu tiên của entry thể hiệc các thông tin sau:
  - 0: Entry này còn trống
  - . : Thư mục cha
  - 0E5 : Entry của tập tin này tạm thời bị xóa
  - Ký tự bất kỳ : tên của một tập tin
- Diễn dải byte thuộc tính:

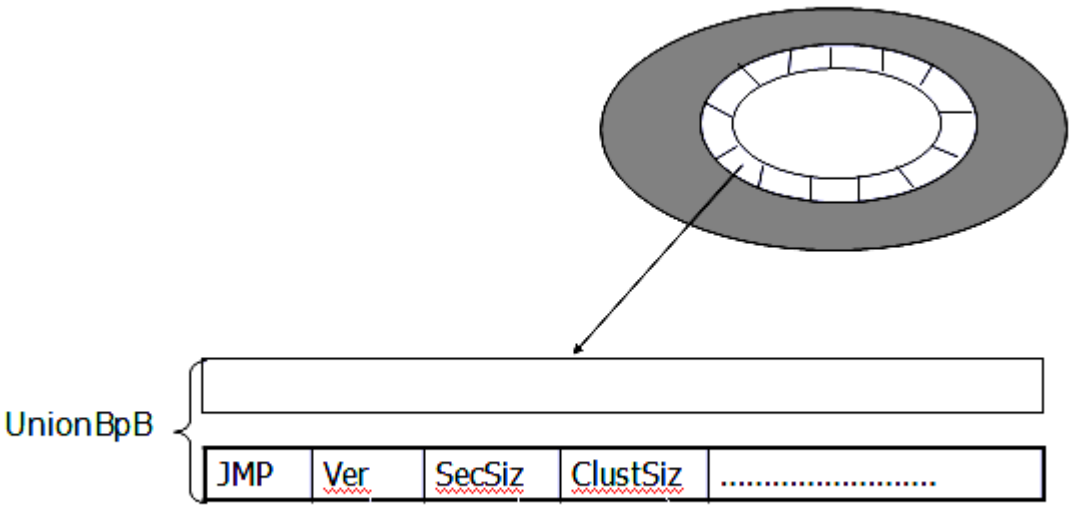


Hình 4.8

- **Khai báo bảng tham số đĩa**

```
Typedef struct {
                unsigned char     JMP[3];
                unsigned char     Ver[8];
                unsigned          SecSiz;
                unsigned char     ClustSiz;
                unsigned          ResSec;
                unsigned char     FatCnt;
                unsigned          RootSiz;
                unsigned          TotSec;
                unsigned char     Media;
                unsigned          FatSiz;
                unsigned          TrkSec;
                unsigned          HeadCnt;
                unsigned          HidSec;
               }EntryBPB;

typedef union {
                unsigned char     Sector[512];
```

```
    EntryBPB    Entry;
}UnionBPB;
```



Hình 4.9

Ví dụ

```
#include <iostream.h>
#include<dos.h>
Typedef struct {.....
}EntryBPB
Typedef Union {.....}UnionBPB
Int main()
{
        UnionBPB  bpb;
        ReadDisk (0,1, bpb.sector);
        cout<<bpb.Entry.secsiz;
        cout <<bpb.Entry.Clustsiz;
        ……..
        Return 1;
}
```

- **Khai báo kiểu thời gian tạo tập tin, thư mục**

```
typedef struct {
                unsigned      S:5 ;
                unsigned      M: 6 ;
                unsigned      H:5;
                }Time;
typedef union {
```

```
        unsigned    intTime;
        Time        T ;
        }UnionTime;
```

- **Khai báo kiểu ngày tháng năm tạo tập tin, thư mục**

```
typedef struct {
        unsigned    D: 5 ;
        unsigned    M: 6 ;
        unsigned    Y:7;
        }Date;
typedef union {
        unsigned    intDate;
        Date        Day ;
        }UnionDate;
```

- **Khai báo cấu trúc byte thuộc tính tập tin**

```
typedef struct {
        unsigned char    ReadOnly : 1 ;
        unsigned char    Hidden   : 1 ;
        unsigned char    System   : 1 ;
        unsigned char    Volume   : 1 ;
        unsigned char    SubDir   : 1 ;
        unsigned char    Archive  : 1 ;
        unsigned char    DR       : 2 ;
    }Attrib;
typedef union {
        unsigned char    charAtt;
        Attrib      Attr ;
        }UnionAttrib;
```

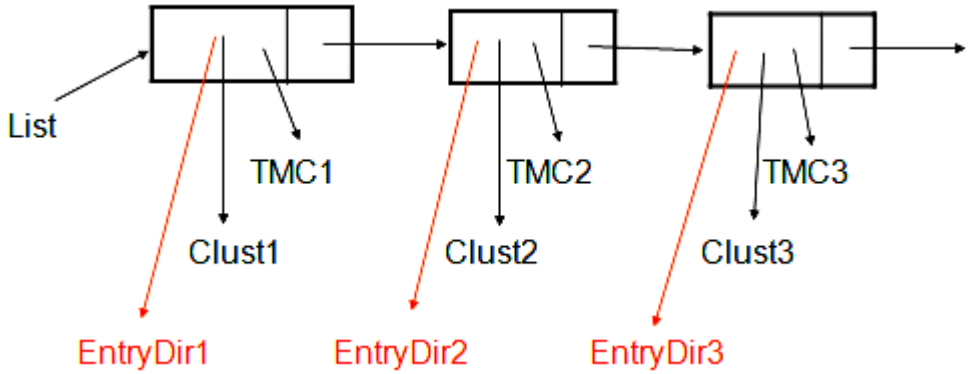
- **Khai báo cấu trúc một Entry tập tin**

```
typedef struct {
        unsigned char    FileName[8] ;
        unsigned char    Ext[3] ;
        UnionAttrib      Attribute ;
        unsigned char    DR[10] ;
        UnionTime        CreateTime ;
        UnionDate        CreateDate ;
        unsigned         ClustBegin ;
        long             FileSize;
    }EntryDir;
```

```
typedef union {
        unsigned char       Entry[32];
        EntryDir            EntDir ;
}UnionDir;
```

- **Danh sách liên kết**

```
typedef struct  Node{
            void  *Data;
            Node *Next;
        }NodeType;
typedef  NodeType *PointerType;
```



Hình 4.10

- **Hàm InsertLast**

```
int  InsertLast (PointerType &List, PointerType &Last,  void* Item)
{
        PointerType  Temp;
        Temp= new  NodeType;
        if( !Temp)
            return 0;
        Temp->Data = Item;
        Temp->Next = NULL;
        if(List==NULL)
            List=Temp;
        else Last->Next=Temp;
        Last = Temp;
        return 1;
}
```

- **Hàm đọc Sector từ đĩa với số hiệu theo kiểu vật lý**

*int ReadDiskBios(unsigned char *Buff,unsigned Side, unsigned TrackSec, unsigned Sector, unsigned SecNum)*

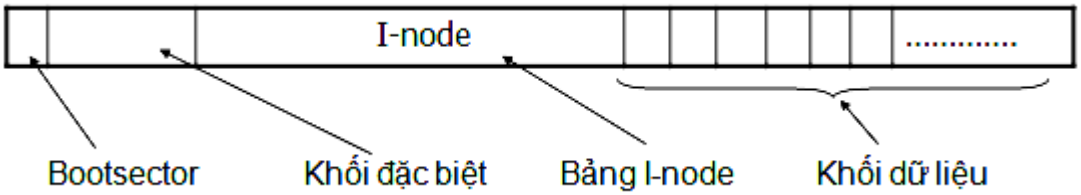
```
{
        union  REGS  u, v;
        struct  SREGS  s;
        int k, i=0;
        v.x.cflag=1;
        while(i<2) && (v.x.cflag!=0))
        {
                u.h.ah= 0x2;// Đọc đĩa
                u.h.dl=0 ; ổ đĩa mềm A
                u,h.dh=Side;
                u.x.cx=TrackSec;  // Track và Sector bắt đầu
                u.h.al=SecNum;
                s.es=FP_SEG(Buff);
                u.x.bx=FP_OFF(Buff);
                int86x (0x13, &u, &v, &s);
                i++;
        }
        k=v.h.ah;
    return (!v.x.cflag);
}
```

- **Hàm đọc đĩa với đầu vào là sector logic**

```
int ReadDisk(char *Buff, long SectorBegin, int SecNum)
{
        unsigned Side, Track, Sector, X;
        unsigned char X1;
    Sector=(unsigned)(1+(SectorBegin)%Bpb.TrkSec);
    Side=(unsigned)(((SectorBegin/Bpb.TrkSec)%Bpb.HeadCnt);
    Track=(unsigned)((SectorBegin)/(Bpb.TrkSec*Bpb.HeadCnt));
    X=Track;
    X=X & 0XFF00;
    X=X>>2;
    X=X & 0X00FF;
    X=X | Sector;
    Track = Track <<8;
    TrackSec=Track | X; //Track ( 10), Sector(6)
    if ( ReadDiskBios(Buff, Side, TrackSec, Sector, SecNum))
        return 1;
    else
        return 0;
}
```

IV.6 Hệ thống tập tin Unix

- **Cấu trúc tổng quát**

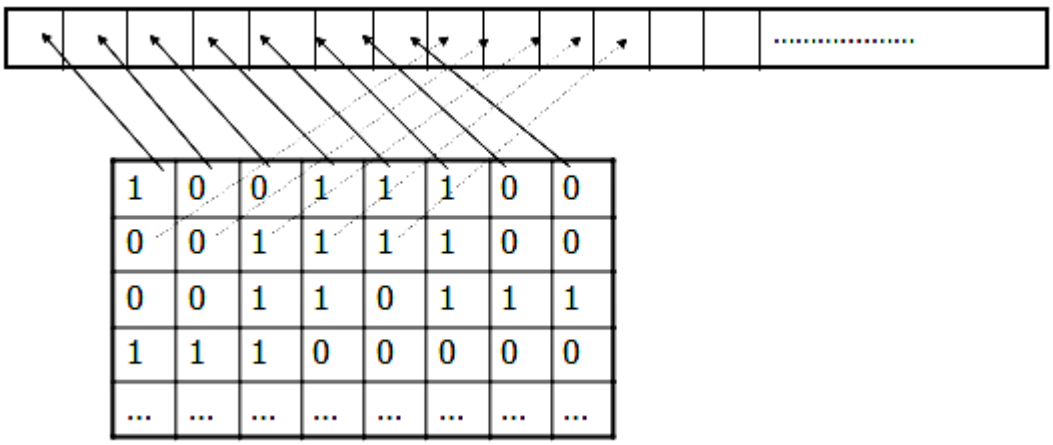


Hình 4.11

- BootSector : chứa chương trình mồi hệ điều hành
- Khối đặc biệt: lưu trữ các thông tin quan trọng về toàn bộ hệ thống tập tin, (số I-node, số khối đĩa, dãy các ô nhớ còn trống trên đĩa,…)
- Sau khối đặc biệt là bảng I-Node được đánh số từ 1 đến tối đa.
- Khối dữ liệu là vùng nhớ lưu trữ nội dung tập tin, thư mục.

- **Cách quản lý các ô nhớ còn trống trong khối dữ liệu:**

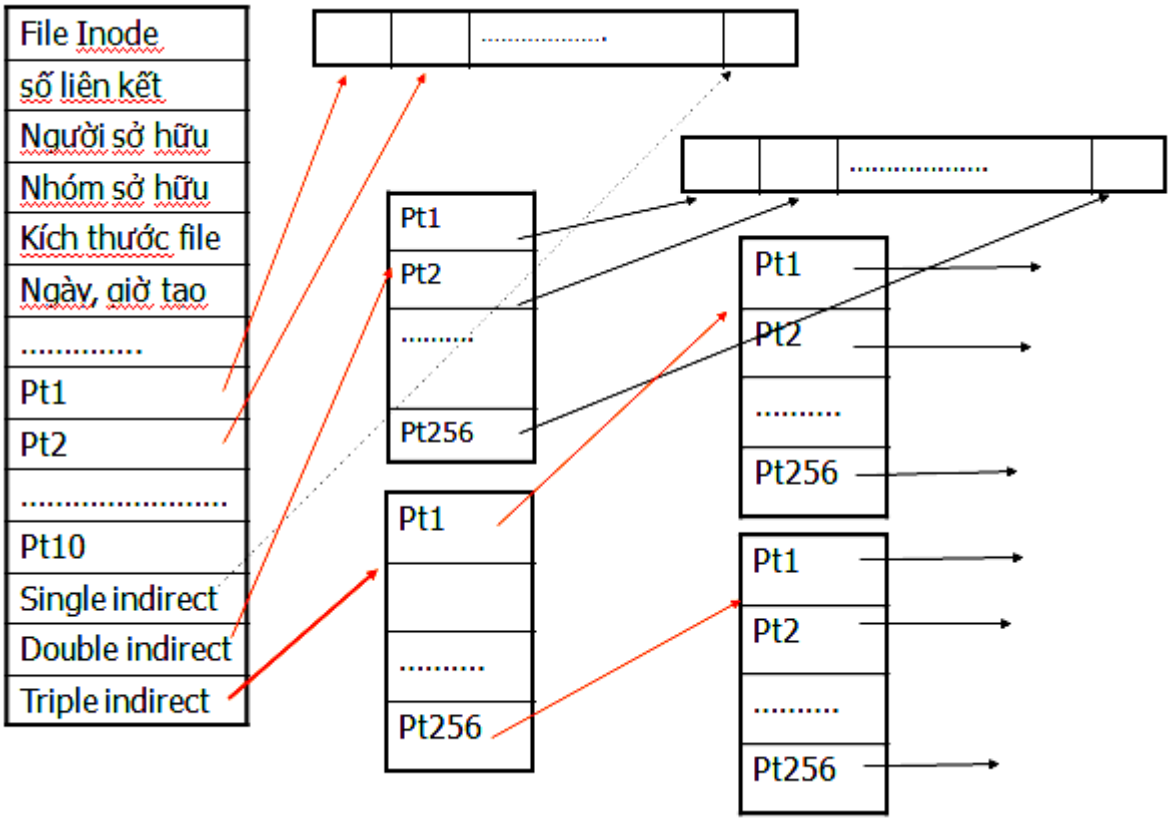
Dùng phương pháp Bitmap: Với đĩa có n ô nhớ sẽ được ánh xạ thành n bit với giá trị 1 là còn trống, giá trị 0 là đã chứa dữ liệu. Như vậy một đĩa 20MB cần 20000 bit để lưu trữ thông tin. Chiếm khoảng 3 ô nhớ.



Hình 4.11

- Tổ chức lưu trữ tập tin:
  - Mỗi file trong unix tương ứng với một I-Node. Một I-Node có kích thước 64byte bao gồm các thông tin về file: file Node, quyền sở hữu file của người sử dụng, quyền sở hữu nhóm, kích thước file, thời điểm tạo,thời điểm truy cập sau cùng, thời điểm thay đổi sau cùng, địa các ô nhớ chứa nội dung,…
  - Phần đánh địa chỉ các ô nhớ nội dung tập tin được chia thành 2 phần: phần đầu gồm 10 pt chứa được 10 địa chỉ ô nhớ. Phần thứ 2 gồm 3 con trỏ gián tiếp: Single indirect, double indirect, triple indirect.
  - Đối với tập tin có kích thước nhỏ hơn 10 ô nhớ dữ liệu thì các con trỏ gián tiếp không được sử dụng để ghi địa chỉ.

- Khi một file có kích thước lớn hơn 10 ô nhớ dữ liệu thì con trỏ single indirect được sử dụng chỉ đến một ô nhớ dành riêng, ô nhớ này lại chứa 256 địa chỉ của ô nhớ dữ liệu.
- Tương tự con trỏ double chỉ đến một ô nhớ chứa 256 địa chỉ ô nhớ và mỗi ô nhớ này lại chứa 256 địa chỉ ô nhớ dữ liệu.
- Một file lớn nhất sử dụng cả 3 con trỏ gián tiếp là 16 GB.



Hình 4.13

- **Cấu trúc thư mục**

Cấu trúc thư mục được sử dụng trong Unix vô cùng đơn giản, mỗi entry bao gồm tên tập tin, số hiệu I-Node của file. Mỗi Entry có kích thước 16 byte:

I-Node	Tên tập tin

Hình 4.14

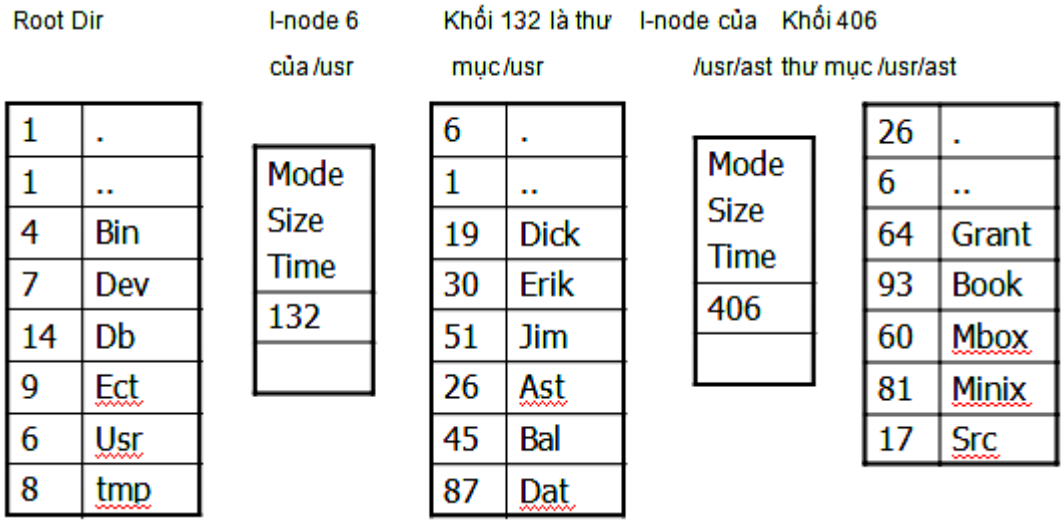
Khi một file được mở , hệ thống file phải xác định vị trí khối dữ liệu trên đia nhờ vào đường dẫn được cung cấp.

Ví dụ: cách truy tìm file dựa vào đường dẫn được cung cấp :

/usr/ast/mbox

- Trước hết đọc I-Node của thư mục gốc – là I-Node đầu tiên trong bảng I-Node

- Đọc từng entry trong thư mục gốc so sánh với thư mục usr từ đó tìm ra I-node của thư mục usr.
- Từ I-node này tiếp tục xác định các entry trong thư mục và so sánh với ast khi đó tìm được I-node của /usr/ast. Tiếp tục tương tự sẽ tìm được I-node của /usr/ast/mbox



Hình 4.15

CÂU HỎI VÀ BÀI TẬP

1. Trình bày các thành phần cấu trúc hệ thống tập tin?
2. Trình các chức năng hệ thống tập tin?
3. Trình bày cấu trúc tổ chức hệ thống tập tin MS-DOS
4. Trình bày cấu trúc tổ chức hệ thống tập tin Unix
5. Trình bày sự khác biệt giữa 2 cấu trúc tổ chức hệ thống tập tin MS-DOS và Unix?
6. Viết chương trình đọc và xuất ra màn hình bảng tham số đĩa mềm 1.44mb?
7. Viết chương trình xem nội dung thư mục gốc ?
8. Viết chương trình xem nội dung thư mục bất kỳ ?
9. Viết chương trình xem nội dung tập tin ?
10. Viết chương trình xoá tập tin , phục hồi tập tin ?
11. Viết chương trình copy tập tin ?
12. Viết chương trình sắp xếp đĩa (dồn cluster) ?
13. Viết chương trình xoá các cluster mồ côi ?
14. Viết chương trình tạo thư mục?

CHƯƠNG V
HỆ THỐNG QUẢN LÝ NHẬP XUẤT

V.1 Các khái niệm

Một trong các chức năng chính của hệ điều hành là hệ thống quản lý thiết bị nhập xuất. Hệ điều hành chỉ thị cho các thiết bị, kiểm soát các ngắt và lỗi. Ngoài ra hệ điều hành phải cung cấp cơ chế giao tiếp đơn giản và tiện dụng giữa các thiết bị và các phần còn lại của hệ thống. Việc giao tiếp này phải độc lập với thiết bị.

V.1.1 Thiết bị nhập xuất

Có nhiều cách nhìn khác nhau về thiết bị nhập xuất. Đối với các lập trình viên thì quan niệm các thiết bị nhập xuất là các lệnh thực thi được trên các thiết bị nhập xuất, chúng thực thi các chức năng gì và thông báo lỗi của chúng bao gồm những gì.

V.1.2 Thiết bị logic

Mục tiêu chung của thiết bị logic là dễ biểu diễn. Thiết bị logic được tổ chức thành nhiều lớp. Lớp dưới cùng giao tiếp với phần cứng, lớp trên cùng giao tiếp tốt , thân thiện với người sử dụng. Thiết bị logic độc lập với thiết bị ví dụ như có thể viết chương trình truy xuất file trên đĩa mềm hay đĩa cứng mà không cần mô tả 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 4 lớp: Kiểm soát lỗi, điều khiển thiết bị, phần mềm hệ điều hành độc lập thiết bị, phần mềm người sử dụng.

V.1.3 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 theo sơ đồ sau:

Các lớp	Chức năng nhập xuất
Xử lý của người sử dụng	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

V.2 Mô hình tổ chức và quản lý việc nhập xuất

V.2.1 Mô hình

V.2.1.1 các thiết bị nhập xuất

Các thiết bị nhập xuất có thể chia tương đối thành 2 loại thiết bị khối và thiết bị thuần tự:

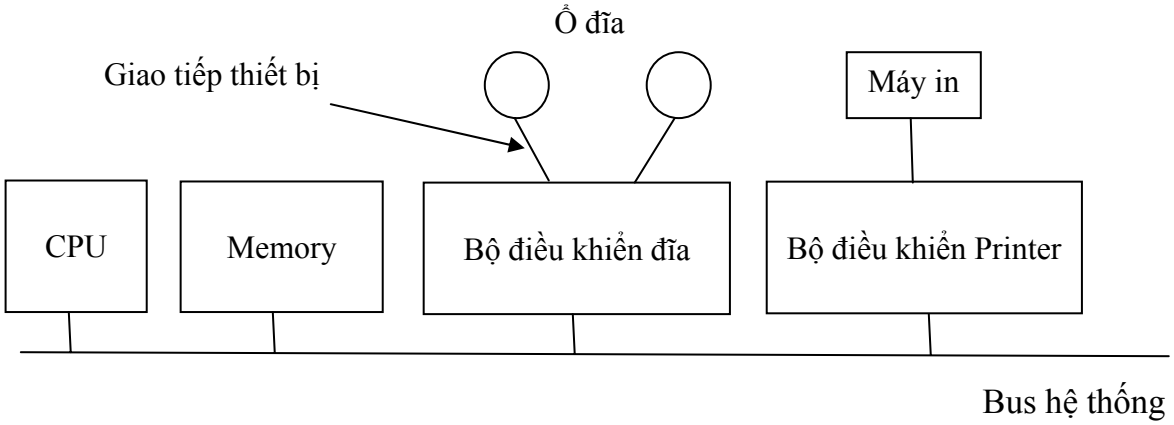
- Thiết bị khối là thiết bị mà thông tin được lưu trữ trong các khối có kích thước cố định và được lưu trữ trong những khối có kích thước cố định và được định vị bởi địa chỉ. Kích thước thông thường của một khối khoảng từ 128 byte đến 1024 byte. Đặc điểm của thiết bị khối là chúng có thể được truy xuất từng khối riêng biệt và chương trình có thể có thể truy xuất một khối bất kỳ nào đó. Đĩa là một ví dụ truy xuất khối.
- Một dạng thiết bị thứ hai là thiết bị tuần tự. Ở dạng thiết bị này việc gởi và nhận thông tin là chuỗi các bits, không có xác định địa chỉ và không thể tìm kiếm. Màn hình, bàn phím, máy in, card mạng, mouse là các thiết bị tuần tự.

V.2.1.2 Điều khiển thiết bị

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

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 2 hay 4 , 8 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.

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



Hình 5.1

V.2.1.3 DMA

Đa số các loại thiết bị, đặc biệt là các thiết bị dạng khối hỗ trợ cơ chế DMA (Direct Memory Access). Để hiểu cơ chế này trước hết phải 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 lấy dữ liệu từ buffer chuyển về bộ nhớ chính bằng cách tạo ra một vòng lặp đọc lần lượt từng byte. Thao tác này làm lãng phí thời gian của CPU, do đó để tối ưu dùng cơ chế DMA.

V.2.1 Thiết bị logic

V.2.1.1 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 khoá lại bằng cách thực hiện lệnh WAIT theo một biến điều kiện hoặc RECEIVE theo một thông điệp.

V.2.1.2 Device Drivers

Tất cả các đoạn mã độc lập thiết bị đều được chuyển đến device driver. 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 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 sector, track, cylinder, head,…giúp cho các thao tác đĩa được thực hiện tốt.

V.2.1.3 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 driver 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 driver vì lý do hiệu quả hay những lý do khác nào đó.

V.2.1.4 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.

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.

V.2.2 Các chức năng

V.2.2.1 Điều khiển 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 chữa nếu cần thiết. Thông thường khối các byte được tổ chức thành từng bit và đặt trong buffer của bộ điều khiển. Sau 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 hình bằng cách quét các tia dọc, 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 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.

Bộ điều khiển nhập xuất	**Địa chỉ nhập xuất**	**Vector 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,… 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.

V.2.2.2 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 byte phải chuyển. Quá trình này lập cho tới khi số byte 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.

V.2.2.3Thiết bị Logic

Kiểm soát ngắt

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 khiển và gửi những thông điệp đến cho các tiến trình bị khoá. Tổng quát, chức năng của ngắt là làm cho một tiến trình đang bị khoá được thi hành trở lại.

Device Driver

Chức năng của device driver 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 rãnh, 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 ví trí của đầu đọc đã đúng chưa.

Nghĩa là device driver 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 driver phải chờ cho tới khi bộ điều khiển thực hiện xong bằng cách tự khoá lại cho tới khi một ngắt phát sinh mở khoá cho nó. Hai là hệ điều hành chấm dứt mà không chờ, vì vậy driver không cần thiết phải khoá.

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 hành đợi, nó sẽ thực hiện tiếp, nếu không nó sẽ khoá lại chờ đến yêu cầu tiếp theo.

Phần mềm nhập xuất độc lập thiết bị

Giao tiếp đồng nhất cho device driver
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

Các chức năng của phần mềm nhập xuất độc lập thiết bị

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 với người sử dụng.

Trước tiên phải có chức năng tạo một ánh xạ giữa các thiết bị ví dụ với Unix tên /dev/tty() dành riêng để mô tả I-node cho một file đặc biệt, và I-node này chứa só thiết bị chính, được dùng để xác định driver thích hợp và số thiết bị hư, được dung 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 hay không 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 hoá quá trình hoạt động của hệ thống ví dụ buffer của 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.

Phần mềm nhập xuất phạm vi người sử dụng

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 sử dụng.

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 email chẳng hạn.

CÂU HỎI VÀ BÀI TẬP

1. Trình bày các khái niệm thiết bị nhập xuất, thiết bị logic?
2. Trình bày hệ thống quản lý nhập xuất?
3. Trình bày mô hình quản lý nhập xuất?
4. Trình bày các chức năng hệ thống nhập xuất?

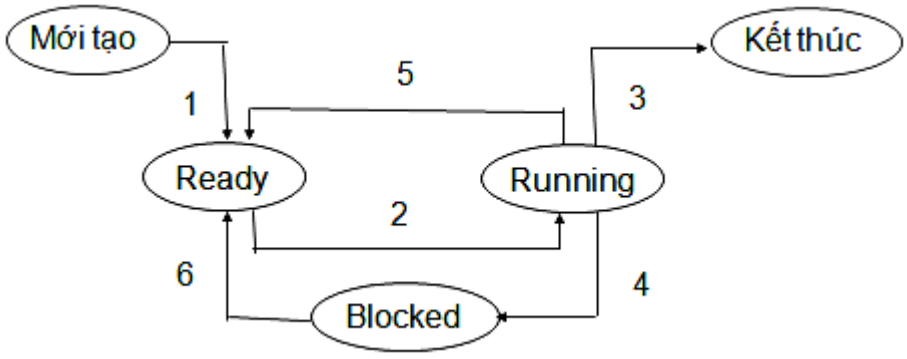
CHƯƠNG VI
HỆ THỐNG QUẢN LÝ TIẾN TRÌNH

VI.1 Khái niệm tiến trình

- Trong hệ thống đa chương có thể thể thực hiện nhiều tác vụ đồng thời.
- Việc thực hiện đồng thời này được hiện bằng cách chuyển đổi CPU qua lại giữa các chương trình. Điều này tạo cảm giác có nhiều chương trình thực hiện đồng thời.
- Trong hệ thống như vậy tất cả phần mềm được tổ chức thành một số tiến trình.
- Một tiến trình là một chương trình đang được xử lý, sở hữu con trỏ lệnh , tập các thanh ghi, biến và để hoàn thành nhiệm vụ của mình một tiến trình phải sử dụng các tài nguyên máy tính như CPU, bộ nhớ chính, các tập tin và thiết bị nhập xuất.
- Ý tưởng là có thể xem như mỗi tiến trình sở hữu một CPU ảo cho riêng mình, nhưng trong thực tế chỉ có một bộ xử lý thật sự được chuyển đổi qua lại giữa các tiến trình.
- Hệ điều hành chịu trách nhiệm sử dụng một thuật toán điều phối để quyết định thời điểm cần dừng một tiến trình để thực hiện một tiến trình khác

VI.2 Các trạng thái của tiến trình

- Trạng thái của một tiến trình tại một thời điểm được xác định bằng hoạt động hiện thời của tiến trính đó.
- Tại một thời điểm một tiến trình có thể nhận một trong các trạng thái sau đây:
  - Mới tạo: Tiến trình đang được tạo lập.
  - Running: các chỉ thị của tiến trình đang được xử lý.
  - Blocked: Tiến trình chờ được cấp phát một tài nguyên hay chờ một sự kiện nào đó xảy ra.
  - Ready: Tiến trình chờ cấp phát CPU để xử lý.
  - Kết thúc : Tiến trình hoàn tất xử lý.
- Mô hình chuyển đổi giữa các trạng thái:



Hình 6.1

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

(2) Bộ lập lịch cấp phát cho tiến trình một khoảng thời gian sử dụng CPU

(3) Tiến trình kết thúc

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

(5) Bộ lập lịch thu hồi CPU và cấp phát cho tiến trình khác

(6) Tài nguyên mà tiến trình yêu cầu đã được cấp phát hay thao tác nhập xuất đã hoàn tất.

VI.3 Cài đặt 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 như sau:
  - Chỉ danh của tiến trình: Để phân biệt các tiến trình
  - Trạng thái tiến trình: Xác định hoạt động hiện hành của tiến trình
    - ✓ Ngữ cảnh của tiến trình: quản lý các tài nguyên của tiến trình:
    - ✓ Trạng thái CPU : nội dung các thanh ghi.
    - ✓ Bộ nhớ chính: Danh sách các ô nhớ được cấp phát 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 tài nguyên do tiến trình tạo lập.
  - Thông tin giao tiếp: 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.
    - ✓ Độ ưu tiên: Giúp bộ lập lịch lựa chọn tiến trình được cấp pháp CPU.
  - Thông tin thống kê: thống kê về hoạt động của tiến trình: thời gian sử dụng CPU, thời gian chờ.

VI.4 Tiểu trình

- Trong hệ điều hành mỗi tiến trình có không gian địa chỉ và có một dòng xử lý, nhưng đôi khi người sử dụng mốn có nhiều dòng xử lý cùng chia xẻ trong cùng không gian địa chỉ và các dòng xử lý này hoạt động song song tương tư như các tiến trình phân biệt khác.
- Mỗi dòng xử lý phân biệt này gọi là một tiểu trình.
- Mỗi tiểu trình xử lý tuần tự đoạn mã của minh và sở hữu con trỏ lệnh tập các thanh ghi, stack riêng. Các tiểu trình chia sẻ CPU như các tiến trình độc lập.
- Một tiến trình có thể sở hữu nhiều tiểu trình .
- Các tiểu trình trong một tiến trình có thể chia sẻ tài nguyên của tiến trình cha (các biến toàn cục)

VI.5 Lập lịch tiến trình

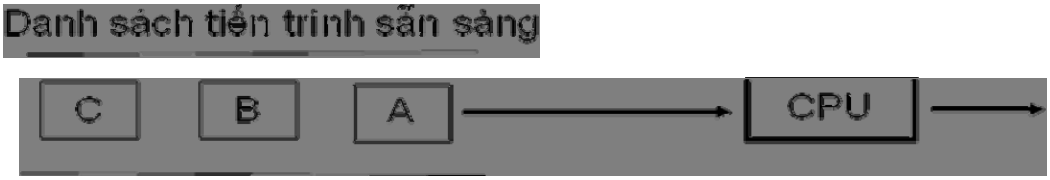
- Trong hệ thống đa nhiệm tại một thời điểm có thể nhiều tiến trình đồng thời sẵn sàng để xử lý. Mục tiêu là chuyển đổi CPU qua lại các tiến trình thường xuyên.
- Để thực hiện điều này hệ điều hành phải lựa chọn tiến trình kế tiếp để xử lý. Bộ lập lịch sẽ sử dụng thuật toán để thực hiện.
- Mục tiêu của bộ lập lịch: Hệ điều hành xây dựng nhiều chiến lược khác nhau thực hiện lập lịch nhưng phải đạt các mục tiệu như sau:
  - Sự công bằng: Các tiến trình chia sẻ CPU một cách công bằng. Không tiến trình nào chờ vô hạn mới được cấp pháp CPU

- Tính hiệu quả: Hệ thống phải tận dụng CPU 100% thời gian
- Thời gain đáp ứng hợp lý: Cực tiểu hóa thời gian hồi đáp cho cac tương tác của người sử dụng.
- Thời gian lưu lại hệ thống : Cực tiểu hóa thời gian hoàn tất các tác vụ xử lý theo lô.
- Thông lượng tối đa: Cực đại hóa số công việc được xử lý trong một đơn vị thời gian

- Tất cả mục tiêu trên thường không thỏa hết vì chính bản thân chúng có sự mâu thuẫn với nhau.
- Lập lịch tiến trình:
  - Hệ điều hành tổ chức một danh sách chứa các tiến trình đang sẵng sàng
  - Hệ điều hành sẽ chọn một tiến trình trong danh sách sẵng sàng để cấp phát CPU.
  - Các chiến lược lập lịch tiến trình

VI.5.1 Chiến lược lập lịch tiến trình FIFO

CPU được cấp phát cho tiến trình đầu tiên trong danh sách sẵn sàng- là tiến trình được đưa vào hệ thống sớm nhất.

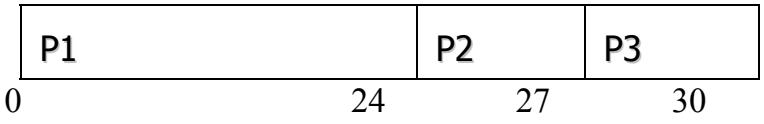


Hình 6.2

Ví dụ:

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

Thứ tự cấp phát CPU cho các tiến trình:

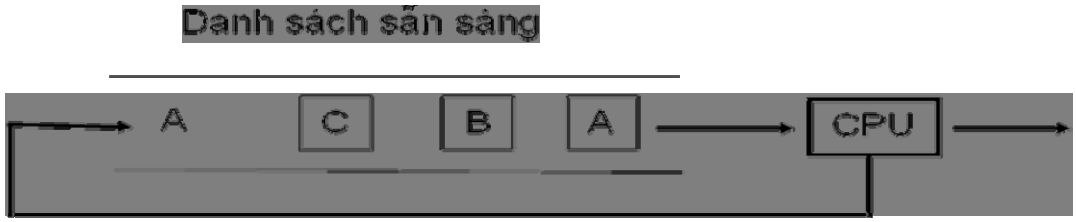


- Thời gian chờ được xử lý của P1 : 0
- Thời gian chờ được xử lý của P2 : 24-1=23
- Thời gian chờ được xử lý của P3 : 24+3 -2 =25

- Thời gian chờ trung bình là : (0+ 23+ 25)/3 =16 milisecondes
- Thời gian chờ trung bình không đạt cực tiểu và xảy ra hiện tượng tích luỹ thời gian tất cả tiến trình phải chờ một tiến trình có yêu cầu thời gian dài kết thúc.

VI.5.2 Chiến lược Round Robin

- Trong chiến lược này danh sách sẵn sàng được sử dụng như danh sách vòng. Bộ điều lập lịch lần lượt cấp phát cho từng tiến trình trong danh sách một khoảng thời gian sử dụng CPU gọi là Quantum
- Khi một tiến trình sử dụng hết thời gian Quantum dành cho nó thì hệ điều hành thu hồi CPU cấp cho tiến trình khác trong danh sách.
- Nếu tiến trình bị Blocked hoặc kết thúc trước khi hết Quantum thì hệ điều hành cũng thu hồi CPU.
- Nếu một tiến trình sử dụng hết Quantum mà chưa xử lý xong sẽ được đưa vào cuối danh sách sẵng sàng để chờ cấp phát CPU lần sau.



Hình 6.3

Ví dụ:

Tiến trình	Thời điểm vào	Thời gian xử lý
P1	0	24
P2	1	3
P3	4	3

Với Quantum = 4 thứ tự cấp phát CPU như sau:

P1	P2	P3	P1	P1	P1	P1	P1

0 4 7 10 14 18 22 26 30

- Thời gian chở xử lý P1: 0
- Thời gian chờ xử lý P2 : 4-1=3
- Thời gian chờ xử lý P3: 7-2 = 5
- Thời gian chờ xử lý P1 lần sau: 10-4=6
- Thời gian chờ trung bình : (0+3+5+6)/3 =4.66 Milisecondes

- Thời gian của Q quá bé thì chuyển đổi CPU giữa các tiến trình quá nhiều khiến việc sử dụng CPU không hiệu quả.
- Nếu Q quá lớn thì tăng thời gian hồi đáp và giảm khả năng tương tác của hệ thống

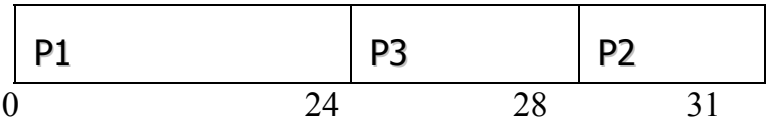
VI.5.3 Chiến lược gán độ ưu tiên

- Mỗi tiến trình được gán một độ ưu tiên tương ứng, tiến trình nào có độ ưu tiên cao hơn sẽ được chọn cấp phát CPU đầu tiên.
- Độ ưu tiên được định nghĩa trong nội tại hoặc từ bên ngoài.
- Chiến lược độ ưu tiên không độc quyền: Khi một tiến trình được đưa vào danh sách sẵn sàng , độ ưu tiên của nó được so sánh với độ ưu tiên của tiến trình đang xử lý. Bộ lập lịch sẽ thu hồi CPU từ tiến trình hiện hành để cấp phát cho tiến trình mới nếu độ ưu tiên của nó cao hơn độ ưu tiên của tiến trình hiện hành
- Chiến lược độ ưu tiên độc quyền: CPU vẫn được cấp phát cho tiến trình hiện hành mặc dù tiến trình mới vào có độ ưu tiên cao hơn độ ưu tiên của tiến trình hiện hành.

Ví dụ : Chiến lược độ ưu tiên độc quyền

Tiến trình	Thời điểm vào	Độ ưu tiên	Thời gian xử lý
P1	0	3	24
P2	1	2	3
P3	2	1	4

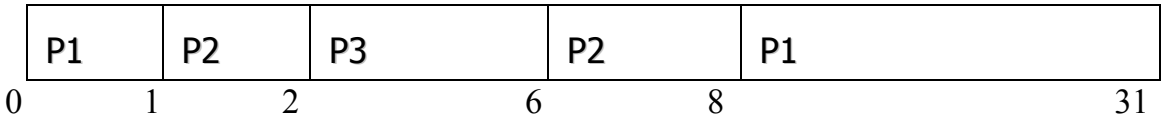
Thứ tự cấp phát CPU như sau:



Ví dụ : Chiến lược độ ưu tiên không độc quyền

Tiến trình	Thời điểm vào	Độ ưu tiên	Thời gian xử lý
P1	0	3	24
P2	1	2	3
P3	2	3	4

Thứ tự cấp phát CPU như sau:



- Với chiến lược này tiến trình có độ ưu tiên thấp sẽ đợi CPU vô hạn. Để tránh trường hợp này thì bộ lập lịch phải giảm dần độ ưu tiên của các tiến trình sau một chu kỳ thời gian.

VI.5.4 Chiến lược công việc ngắn nhất được thực hiện trước:

- Đây là thuật giải dành cho hệ thống xử lý theo lô, khi mà thời gian chạy của mỗi công việc được biết trước.
- Giả sử a, b, c, d lần lượt là thời gian của 4 công việc . Nếu cho 4 công việc này chạy theo thứ tự đó thì thời gian chạy trung bình là :
  (4a+3b+2c+d) /4 .
- Dễ dàng thấy là nếu chọn công việc ngắn cho chạy trước thì giá trị thời gian chạy trung bình là nhỏ nhất.

VI.6 Đồng bộ hóa tiến trình

- Sự liên lạc giữa các tiến trình
  - Trong môi trường đa nhiệm các tiến trình không chạy độc lập mà thường xuyên có nhu cầu trao đổi thông tin với nhau.
  - Nguyên tắc chung trao đổi thông tin giữa các tiến trình: Sử dụng vùng nhớ được chia sẻ, Trao đổi thông điệp.
  - Vấn đề nảy sinh : xảy ra hiện tượng đua nhau sử dụng vùng nhớ chia xẻ dẫn đến kết quả không chính xác - Cần phải đồng bộ hóa tiến trình.
- Điều kiện đua: Nếu có nhiều tiến trình đọc , ghi dữ liệu vào vùng nhớ dùng chung và kết quả cuối cùng phụ thuộc thời điểm tiến trình nào chạy thật sự gọi là điều kiện đua.
- Vùng găng:
  - Để tránh điều kiện đua, nếu hệ điều hành không cho phép có nhiều tiến trình đọc hoặc ghi vào vùng nhớ lưu trữ chung tại cùng một thời điểm cần phải đạt sự loại trừ lẫn nhau

- Một phần nào đó của chương trình mà tại đó thực hiện truy cập đến vùng nhớ dùng chung gọi là vùng găng

- Để tránh điều kiện đua thì hệ điều hành phải được thiết kế sao cho không cho phép 2 hay nhiều hơn tiến trình đồng thời trong vùng găng.
- Bốn điều kiện cần đảm bảo để thiết kế hệ điều hành cho phép nhiều tiến trình sử dụng vùng nhớ dùng chung một cách đúng đắn và hiệu quả:
  - (1) Không cho phép có nhiều hơn một tiến trình đồng thời trong vùng găng.
  - (2) Khi lập trình các tiến trình không được phép có bất kỳ giả định nào về tốc độ CPU và số lượng CPU.
  - (3) Không cho phép một tiến trình ở ngoài vùng găng của mình làm Blocked một tiến trình khác.
  - (4) Không cho phép bất kỳ một tiến trình nào đợi thời gian quá lâu mới có thể vào vùng găng của mình.

VI.6.1 Các phương pháp thực hiện loại trừ nhau vào vùng găng

VI.6.1.1 Dùng biến khóa

Hệ điều hành sử dụng một biến dùng chung, gọi là biến khóa lock được khởi tạo =0. Khi một tiến trình muốn vào vùng găng, nó thực hiện kiểm tra biến lock

```
int lock =0;

if (lock==0) (*)
        { lock =1;
         tiến trình trong vùng găng;
        }
else
        { tiến trình đợi cho đến khi lock =0
        }
```

Với phương pháp này vi phạm điều kiện (1) khi bộ lập lịch thu hồi CPU của một tiến trình tại (*), tiến trình này đã ghi nhận biến lock =0 và chuẩn bị vào găng. Lúc này bộ lập lịch cấp phát CPU cho tiến trình khác, tiến trình này kiểm tra biến lock thấy vẫn =0 và được phép vào găng. Nếu bộ lập lịch lại cấp phát CPU cho tiến trình trước thì tiến trình này tiếp tục chạy và thực hiện vào găng. Vậy tại một thời điểm có 2 tiến trình vào găng.

VI.6.1.2 Luân phiên ngặt

Ý tưởng: Dùng một biến dùng chung turn=0; và mỗi tiến trình có đoạn mã sau:

```
whie   (TRUE)                          while(TRUE)
{                                  {
        while (turn !=0);                  while(turn !=1);
        Tronggăng();                       TrongGăng();
        turn=1;                                turn =0;
        ngoàigăng();                       NgoàiGăng();
}                                  }
```

Phương pháp này vi phạm điều kiện (4) vì theo phương pháp này hai tiến trình phải thay phiên ngặt vào ra vùng găng. Nếu một tiến trình vào ra vùng găng không luân phiên sẽ gây ra tiến trình còn lại đợi quá lâu không vào vùng găng.

VI.6.1.3 Giải pháp Peterson

```
#define  FALSE  0
#define  TRUE   1
#define  N      2
int turn;
int interested[N]; /* khởi gán bằng FALSE*/

void Vàogăng(int Process)
{
        int other;
        other = 1- Process;
        interested[Process]= TRUE;
        turn = Process;
        while (turn ==Process && interested[other] ==TRUE);
}


void RaGăng(int Process)
{
        interested[Process]=FALSE;
}
```

Khi một tiến trình nào đó muốn vào vùng găng thi gọi hàm **Vaogang()** và truyền tham số là số hiệu tiến trình.

Khi tiến trình muốn ra khỏi vùng găng nó gọi hàm **RaGăng()**

Mặc dù giải pháp Peterson là chấp nhận được vì thỏa mãn 4 điều kiện nhưng bị hàn chế:

1. Khi một tiến trình đợi vào vùng găng tiến trình vẫn sử dụng thời gian CPU – Lãng phí CPU.

2. Khi đưa ra khái niệm độ ưu tiên cho các tiến trình giải pháp Peterson không đáp ứng được. (xét trường hợp tiến trình có độ ưu tiên thấp hơn trong vùng găng và tiến trình có độ ưu tiên cao đợi vào vùng găng.)

Giải pháp gọi

VI.6.1.4 Giải pháp gọi lời gọi hệ thống SLEEP vào WAKEUP sẽ làm blocked tiến trình đợi vào vùng găng

SLEEP: Chuyển tiến trình gọi nó về trạng thái blocked cho đến khi tiến trình khác gởi đến nó tín hiệu đổi trạng thái (WAKEUP)
WAKEUP(Process) : Chuyển tiến trình Process về trạng thái Ready (tiến trình đã gọi SLEEP trước đó)

Xét bài toán : "Sản xuất – tiêu thụ ": Có hai tiến trình SảnXuất và TiêuThụ dùng chung buffer có kích thước cố định. Tiến trình SảnXuất đặt sản phẩm vào buffer nếu buffer còn trống. Tiến trình TiêuThụ lấy sản phẩm từ buffer nếu buffer khác rỗng.

```
#define  N  100
int count=0;
void SảnXuất(void)
{
        int item;
        while (TRUE)
        {
                SảnXuấtSảnPhẩm(&Item);
                if (count == N)  SLEEP();
                ĐặtSảnPhẩm(Item);
                count++;
                if (count == 1)  WAKEUP(TieuThụ);
        }
}


void  TiêuThụ(void)
                {
                        int Item;
                        while( TRUE)
                        {
                                if (count == 0)  SLEEP();
                                LấySảnPhẩm(&Item);
                                count--;
                                if (count == N-1)  WAKEUP (SảnXuất);
                                TiêuThụsảnPhẩm(Item);
                        }
                }
```

Với giải pháp này hệ thống có thể dẫn đến tình trạng Deadlock tức cả hai tiến trình đều rơi vào trạng thái Block, không có tiến trình wakeup do sử dụng biến dùng chung

count không được thực hiện theo thao tác nguyên tử. Kết quả là tín hiệu WAKEUP bị mất khi tiến trình được WAKEUP chưa thật sự SLEEP.

- Cần duy trì một biến đếm cho mỗi tiến trình để đếm tín hiệu WAKEUP được gởi đến từ tiến trình khác . Mỗi khi gọi SLEEP tiến trình kiểm tra biến đếm , nếu biến đếm >0 thì giảm biến đếm xuống 1 và tiến trình vẫn không bị blocked.
- Đó chính là ý tưởng để xây dựng khái niệm Semaphore

VI.6.1.5 Semaphore

Semaphore là một kiểu nguyên không âm. Một semaphore s =0 chỉ ra rằng không tín hiệu WAKEUP nào được gởi đến. Có hai thao tác nguyê tử trên semaphore được định nghĩa như sau:

DOWN(s) : if (s >0) s=s-1;
else SLEEP();
UP(s): if (s >0) s=s+1;
else nếu có một hoặc nhiều tiến trình đang bị Blocked trên semaphore s . Hệ điều hành chọn ngẫu nhiên một tiến trình cho phép thoát khỏi trạng thái Blocked. (Trong khi đó s vẫn =0.)
ngược lại: không có tiến trình nào Blocked trên s thì: s= s+1 ;

- Tất cả công đoạn kiểm tra giá trị s, thay đổi s, gọi SLEEP được tích hợp thành một thao tác duy nhất không phân chia được- gọi là thao tác nguyên tử .
- Một semaphore s được khởi tạo =1 và được sử dụng bởi nhiều tiến trình để đảm bảo chỉ một trong chúng là vào được vùng găng tại một thời điểm gọi là **semaphore nhị phân**. Vì vậy mỗi tiến trình chỉ cần gọi toán tử DOWN(s) trước khi vào vùng găng và gọi UP(s) sau khi ra khỏi vùng găng thì có thể đảm bảo được sự loại trừ lẫn nhau.
- Các loại semaphore khác gọi là **semaphore đồng bộ hóa**, nó đảm bảo một dãy các sự kiện nào đó là xuất hiện hoặc không xuất hiện.

- **Áp dụng Semaphore để giải quyết bài toán Sản suất – tiêu thụ**

```
#define  N  100
typedef  int Semaphore;
Semaphore  Mutex = 1;
Semaphore Empty =N;
Semaphore Full = 0;
void SảnXuất (void)
{
        int Item;
        while(TRUE)
        {
        SảnXuấtSảnPhẩm(&Item);
        down(&Empty);
        down(&Mutex);
```

```
            ĐặtSảnPhẩm(Item);
            up(&Mutex);
            up(&Full);
            }
      }

void TiêuThụ (void)
      {     int Item;
            while(TRUE)
            {     down(&Full);
                  down(&Mutex);
                  LấySảnPhẩm(&Item);
                  up(&Mutex);
                  up(&Empty);
                  TiêuThụsảnPhẩm(Item);
            }
}
```

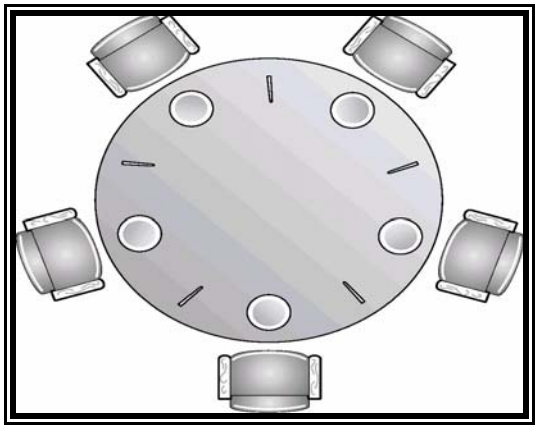
VI.6.2 Áp dụng Semaphore để giải quyết bài toán cổ điển

VI.6.2.1 Bài toán" Bữa ăn tối của các nhà hiền triết"

Có 5 nhà hiền triết ngồi quanh một bàn tròn trong một bữa ăn tối. Mỗi người có một dĩa mì Spaghetti. Mỗi người cần phải có 2 nĩa để có thể ăn mì. Giữa 2 dĩa có một nĩa.

Giả định rằng cuộc đời của nhà hiền triết chỉ luân phiên nhau 2 hành vi: ăn và suy nghĩ. Khi nhà hiền triết cảm thấy đói ông ta muốn lấy 2 nĩa bên trái và phải theo thứ tự nào đó. Nếu lấy được cả 2 nĩa ông ta bắt đầu ăn. Sau đó đặt nĩa xuống và tiếp tục suy nghĩ.

Yêu cầu viết chương trình cho mỗi nhà hiền triết sao cho không bị "kẹt".



Hình 6.4

```
#define  N  5
      void  HiềnTriết (int i)
```

```
{
        while(TRUE)
        {
                SuyNghĩ();
                LấyNĩa(i);
                LấyNĩa((i+1)%N); // Nhà hiền triết I lấy nĩa bên trái, phải
                Ă n();
                ĐặtNĩa(i);
                ĐặtNĩa((i+1)%N);
        }
}
```

Lời giải 1

```
#define  N  5
typedef int Semaphore;
Semaphore Mutex=1;
void  HiềnTriết (int i)
{       while(TRUE)
        {       SuyNghĩ();
                down(&Mutex);
                LấyNĩa(i);
                LấyNĩa((i+1)%N); // Nhà hiền triết I lấy nĩa bên trái, phải
                Ă n();
                ĐặtNĩa(i);
                ĐặtNĩa((i+1)%N);
                up(&Mutex);
        }
}
```

Lời giải trên đúng nhưng không tối ưu tài nguyên – tại một thời điển chỉ có 1 nhà hiền triết ở trạng thái ăn trong khi đó có thể có 2

Lời giải 2

```
#define  N  5
#define LEFT (i-1)%N
#define RIGHT (i+1)%N
#define THINKING 0
#define HUNGRY   1
#define EATING     2
typedef int Semaphore;
int State [N];
Semaphore Mutex=1;
Semaphore S[N];// Khởi gán =0
```

void ***HiềnTriết*** *(int i)*

```
{       while(TRUE)
        {       SuyNghĩ();
                LấyNĩa(i);// Lấy nĩa trái và phải
                Ă n();
                ĐặtNĩa(i);
        }
}
void LấyNĩa (int i)
{       down (&Mutex);
        State[i]=HUNGRY;
        Test(i);
        up(&Mutex);
        down(&S[i]);
        return ;
}

void ĐặtNĩa (int i)
{       down (&Mutex);
        State[i]=THINKING;
        Test(LEFT);
        Test(RIGHT);
        up(&Mutex);
        return ;
}

void Test (int i)
{       if( State[i]==HUNGRY  && State[LEFT]!=EATING &&
        State[RIGHT]!=EATING)
        {       State[i]=EATING;
                up(&S[i]);
        }
}
```

VI.6.2.2 Bài toán" Độc giả và nhà văn"

Một cơ sở dữ liệu mà tiến trình muốn đọc(độc giả) hoặc ghi lên đó (nhà văn) . Hệ thống cho phép đồng thời có nhiều tiến trình đọc cơ sở dữ liệu nhưng chỉ duy nhất một tiến trình ghi lên CSDL tại một thời điểm. Khi có một tiến trình ghi lên CSDL thì không có tiến trình nào được phép truy cập đến CSDL kể cả tiến trình đọc .

Yêu cầu: Lập trình cho hai tiến trình "Độc giả " và "nhà văn"

```
Seamphore  Mutex =1;
Semaphore  Db=1;// Truy cập vào DBF của tiến trình Writer
int  rc; // Đếm số tiến trình đọc
```

```
void Reader (void )
{       while (TRUE)
        {       down (Mutex);
                rc= rc+1;
                if ( rc==1)
                        down(db);
                up(Mutex);
                ReadDBF();
                down(Mutex);
                rc=rc-1;
                if (rc==0)
                        up(db);
                up(Mutex);
        }
}


void Writer (void )
{       while (TRUE)
        {       CreateData();
                down(db);
                WriteData();
                up(db);
        }
}
```

CÂU HỎI VÀ BÀI TẬP

1. Trình bày khái niệm tiến trình và giải thích quá trình chuyển đổi giữa các trạng thái tiến trình?
2. Trình bày mục tiêu của bộ lập lịch tiến trình?
3. Hãy tính thời gian chờ được xử lý trung bình của các tiến trình theo thuật toán Round Robin với Q=3

Tiến trình	Thời điểm vào	Thời gian xử lý
P1	0	10
P2	1	5
P3	4	7
P4	5	8

Hãy tính thời gian chờ được xử lý trung bình của các tiến trình theo thuật toán độ ưu tiên không độc quyền kết hợp Round Robin(Q=3) đối với các tiến trình có cùng độ ưu tiên.

Biết rằng:

- Độ ưu tiên của P1, P2, P3, P4 lần lượt là 2, 3,1, 2. (1>2>3)
- Sau mỗi lần xử lý độ ưu tiên của tiến trình giảm đi 1.

4. Hãy tính thời gian chờ được xử lý trung bình của các tiến trình theo thuật toán độ ưu tiên không độc quyền , với độ ưu tiên 1>2>3 ….

Tiến trình	Thời điểm vào	Độ ưu tiên	Thời gian xử lý
P1	0	1	7
P2	1	3	5
P3	4	2	3
P4	5	1	8

5. Trình bày nguyên tắc trao đổi thông giữa các tiến trình. Lý do cần đồng bộ hoá các tiến trình?
6. Thảo luận sự vi phạm điều kiện (4) cần đảm bảo để thiết kế hệ điều hành cho phép nhiều tiến trình sử dụng vùng nhớ dùng chung một cách đúng đắn và hiệu quả của thuật toán luân phiên ngặt?
7. Trình bày nhược điểm của giải pháp đồng bộ hoá Peterson và giải pháp gọi lời gọi hệ thống Sleep() và wakeup() ?
8. Trình khái niệm Semaphore và định nghĩa 2 toán tử Down() và Up() trên semaphore ? Cho một ví dụ tình huống sử dụng semaphore nhị phân?
9. Sử dụng semaphore thực hiện đồng bộ hóa hai tiến trình SendMessage() và ReceiveMessage(). Biết rằng hai tiến trình trên sử dụng chung một hàng đợi Queue, tiến trình ReceiceMessage() nhận message từ bàn phím và ghi vào đầu hàng đợi và tiến trình SendMessage() lấy message từ hàng đợi Queue và gởi lên mạng máy tính. Kích thước của hàng đợi Queue là N, mỗi phần tử hàng đợi chứa được một message. Khi hàng đợi đầy tiến trình ReceiceMessage() phải tạm ngưng, khi hàng đợi rỗng tiến trình SendMessage() phải tạm ngưng.
10. Sử dụng semaphore tạo ra hai tiến trình sau sao cho nb<na<nb+10:

Tiến trình 1:

```
While(TRUE)
{
        na=na+1;
}
```

Tiến trình 2:

```
While(TRUE)
{
        nb=nb+1;
}
```

CHƯƠNG VII

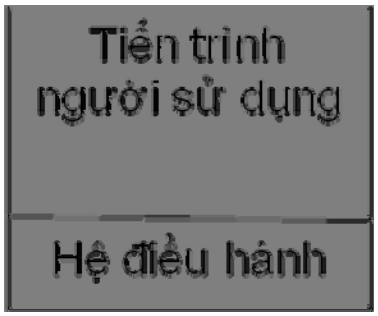
HỆ THỐNG QUẢN LÝ BỘ NHỚ

VII.1 Giới thiệu

- Bộ nhớ chính là thiết bị lưu trữ duy nhất thông qua đó CPU có thể trao đổi thông tin với môi trường ngoài.
- Nhu cầu tổ chức quản lý bộ nhớ là nhiệm vụ hàng đầu của hệ điều hành.
- Bộ nhớ chính được tổ chức như một mảng một chiều các ô nhớ. Việc trao đổi thông tin với môi trường ngoài được thực hiện thông qua các thao tác đọc ghidữ liệu vào địa chỉ cụ thể của bộ nhớ.
- Hầu hết các hệ điều hành hiện đại đều cho phép chế độ đa nhiệm nhằm nâng cao hiệu quả sử dụng CPU.

VII.2 Quản lý bộ nhớ không phân trang, không Swapping

- Bộ nhớ chỉ được chia sẻ cho hệ điều hành và một chương trình duy nhất của người sử dụng. Tức tại một thời điểm một phần bộ nhớ sẽ do HĐH chiếm giữ và phần còn lại thuộc về một tiến trình người sử dụng, tiến trình này có toàn quyền sử dụng vùng nhớ dành cho nó.
- Mô hình

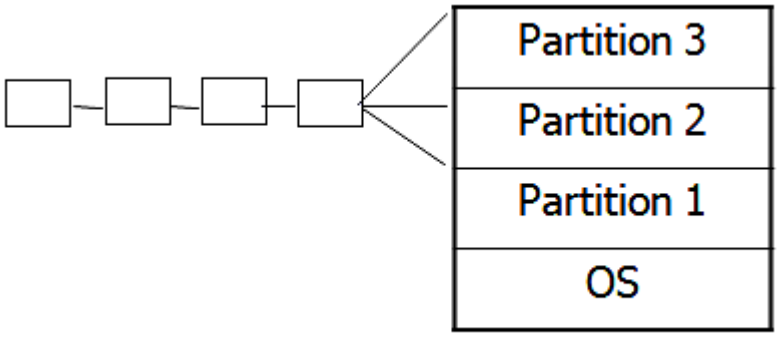


Hình 7.1

- Để bảo vệ vùng nhớ của HĐH khỏi sự xâm phạm của tiến trình người dùng cần phải tổ chức thanh ghi giới hạn.
- Địa chỉ cao nhất của vùng nhớ được cấp phát cho HĐH sẽ được nạp vào thanh ghi giới hạn.
- Tất cả các địa chỉ được tiến trình người dùng truy xuất đến sẽ được so sánh với nội dung thanh ghi giới hạn, nếu địa chỉ lớn hơn nội dung thanh ghi giới hạn thì đây là một địa chỉ hợp lệ, ngược lại một ngắt được phát sinh để thông báo cho hệ thống về một truy xuất bất hợp lệ.
- Với tổ chức như vậy thì chỉ có thể xử lý một chương trình duy nhất tại một thời điểm.
- Trong thực tế có rất nhiều tiến trình phải trải qua phần lớn thời gian chờ thao tác nhập xuất hoàn thành trong suốt thời gian này CPU nhàn rỗi ◊ để nâng cao hiệu suất sử dụng CPU cần cho phép chế độ đa chương.

VII.3 Quản lý bộ nhớ với những phân đoạn cố định

- Hệ điều hành chia bộ nhớ thành n vùng nhớ cố định(có thể không bằng nhau)
- Việc phân chia này được thực hiện vào lúc khởi động hệ thống và không thay đổi suốt quá trình chạy.
- Với tổ chức như vậy cần duy trì một hàng đợi duy nhất để lưu trữ những tiến trình chưa được cấp phát bộ nhớ.



Hình 7.2

- Tất cả tiến trình được đặt trong một hàng đợi duy nhất. 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 và cho xử lý.
- Nếu kích thước của tiến trình không vừa đúng bằng kích thước phân vùng chứa nó, phần bộ nhớ không sử dụng đến trong phân vùng sẽ bị lãng phí.
  - Xảy ra hiện tượng phân mảnh nội vi.
  - Mức độ đa chương của hệ thống bị giới hạn bởi số lượng phân vùng
- Vấn đề bảo vệ giữa các phân vùng: Để bảo vệ cần tổ chức hai thanh ghi : thanh ghi nền và thanh ghi giới hạn.
  - Khi tiến trình được tạo lập, nạp vào thanh ghi nền địa chỉ bắt đầu của phân vùng được cấp phát cho tiến trìnhvà nạp vào thanh ghi giới hạn kích thước của tiến trình
  - Sau đó mỗi đị 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 và các địa chỉ được đối chiếu với thanh ghi giới hạn để bảo đảm tiến trình không truy xuất ngoài phạm vi được cấp phát cho nó.

VII.4 Quản lý bộ nhớ với những phân đoạn động

- Như tổ chức quản lý bộ nhớ trên gây ra lãng phí bộ nhớ vì vậy một phương pháp mới được đề xuất là cấp pháp cho tiến trình vùng nhớ vừa đủ kích thước của tiến trình.
- Khi một tiến trình kết thúc vùng nhớ đã đã cấp cho nó sẽ được giải phóng và được cấp pháp cho tiến trình khác.
- Với giải pháp này không còn hiện tượng phân mảnh nội vi nhưng lại xuất hiện phân mảng ngoại vi. Khi các tiến trình lần lượt vào ra khỏi hệ thống, dần dần xuất hiện các khe hở giữa các tiến trình có thể dẫn đến tình huống tổng vùng nhớ còn trống đủ để thỏa mãn yêu cầu nhưng các vùng nhớ này không liên tục.

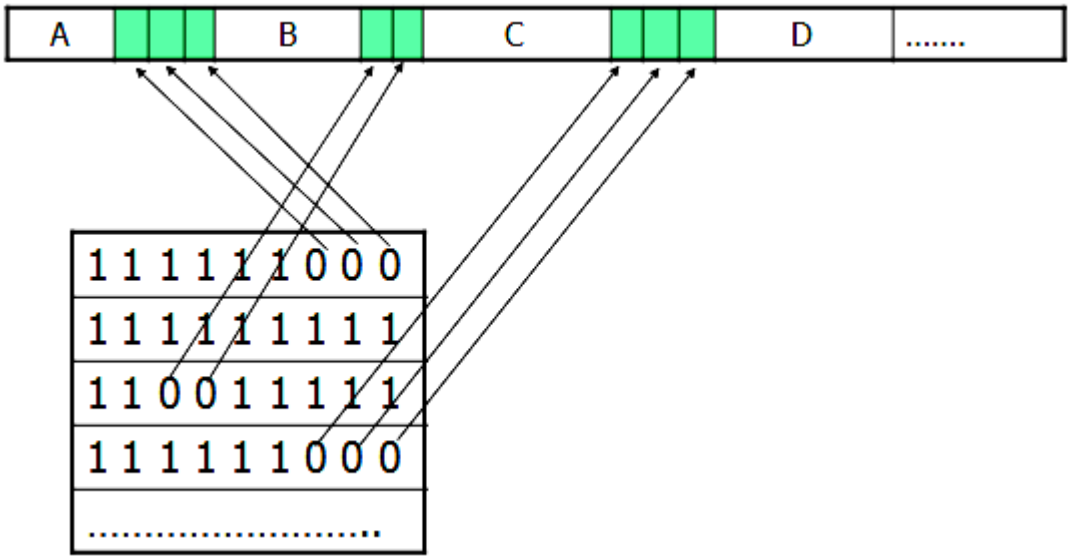
- Có thể áp dụng kỹ thuật dồn bộ nhớ để kết hợp các mảnh bộ nhớ rời rạc thành một vùng nhớ lớn liên tục.
- Một vấn đề khác nảy sinh khi kích thước của tiến trình tăng trưởng trong quá trình xử lý mà không còn vùng nhớ còn trống gần kề để mở rộng vùng nhớ cho tiến trình.

- Quản lý các ô nhớ còn trống: Cần phải lưu trữ thông tin các ô nhớ còn trống để cấp phát cho các tiến trình, Có hai phương pháp chính: Bitmap và danh sách liên kết.
- Phương pháp Bitmap:

Chia bộ nhớ thành từng đơn vị nhỏ (vài bytes) . Xây dựng bitmap, ứng với mỗi bit trong bitmap là một đơn vị bộ nhớ.

- Bit được đánh dấu là 1 khi đơn vị bộ nhớ tương ứng đã được cấp phát
- Bit được đánh dấu 0 khi đơn vị bộ nhớ tương ứng chưa được cấp phát.

Thao tác cấp phát bộ nhớ là : Giả sử tiến trình cần k đơn vị bộ nhớ, HĐH duyệt bitmap và tìm ra k bit liên tiếp bằng 0

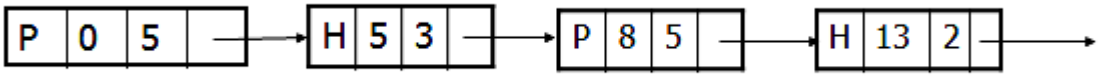


Hình 7.3

- Quản vùng nhớ còn trống bằng danh sách liên kết

Tổ chức một danh sách liên kết, mỗi phần tử tương ứng với một tiến trình hay lỗ hổng. Mỗi phần tử trong danh sách có 4 trường :

- Cờ biểu thị tiến trình (P) hay lỗ hổng (H)
- Địa chỉ bắt đầu của vùng nhớ tương ứng
- Kích thước của vùng nhớ
- Con trỏ Next



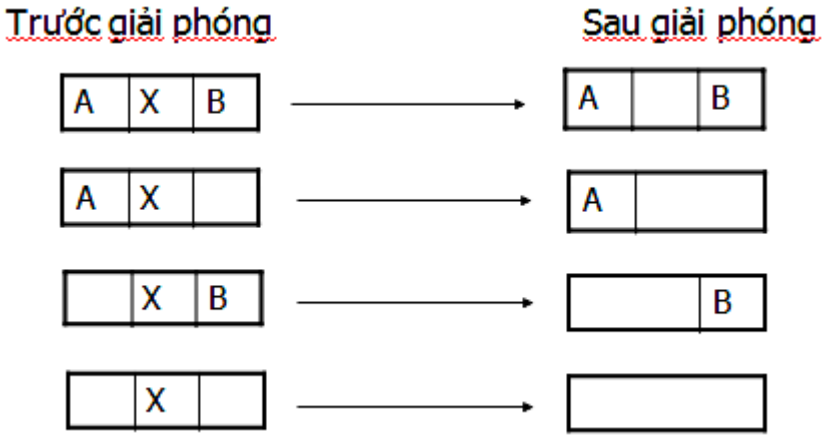
Hình 7.4

- **Thao tác cấp phát bộ nhớ**

- First Fit : Xác định lỗ hổng đầu tiên trong danh sách có kích thước đủ lớn để cấp phát cho tiến trình và lỗ hổng này được chia làm 2 phần : 1 phần cho tiến trình và phần kia là lỗ hổng mới.
- Best Fit : xác định lỗ hổng bé nhất có kích thước đủ lớn để cấp phát cho tiến trình.
- Worst Fit : Cấp phát phân đoạn tự do lớn nhất đủ lớn để cấp phát cho riến trình.

Có thể làm tăng tốc độ bằng cách cho cả 3 thuật toán trên bằng cách tổ chức 2 danh sách: 1 cho tiến trình và một cho lỗ hổng. Tuy nhiên lại làm chậm thao tác giải phóng bộ nhớ.

- **Thao tác giải phóng bộ nhớ**
  - Khi thực hiện thao tác giải phóng chú ý cần xem xét các trường hợp khác nhau của 2 phần tử kề nhau. Nhờ đó thực hiện thao tác kết hợp hai lỗ hổng kề nhau một cách phù hợp:



Hình 7.5

- Quá trình Swapping

Trong hệ thống đa nhiệm nhiều người dùng thường là bộ nhớ không đủ chỗ để lưu trữ tất cả các tiến trình vì thế hệ thống dùng DSLK để lưu trữ tạm thời một số tiến trình nào đó chưa thật sự chạy, khi được cấp phát CPU thì hệ thống phải mang nó vào bộ nhớ chính . Việc di chuyển tiến trình từ bộ nhớ vào đĩa và ngược lại gọi là quá trình swapping.

- Bộ nhớ ảo

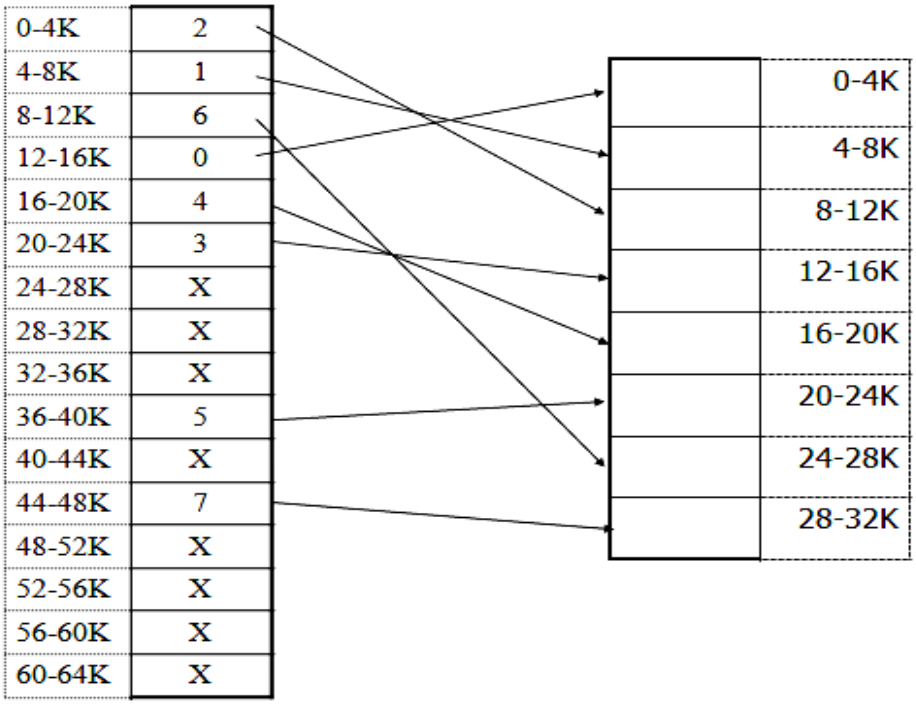
Bộ nhớ ảo được đưa ra nhằm khắc phục tình trạng kích thước chương trình vượt quá kích thước bộ nhớ vật lý. Hệ điều hành chia chương trình thành nhiều phần và giữa lại những phần chương trình đang chạy hiện thời vừa đủ chứa trong bộ nhớ , phần còn lại lưu trữ trên đĩa. HĐH theo dõi và quản lý tất cả công việc swapping giữa đĩa và bộ nhớ . Dĩ nhiên bộ nhớ ảo vẫn có thể thiết kế cho hệ thống đa nhiệm.

- Sự phân trang (paging)

Hầu hết những hệ thống có sử dụng bộ nhớ ảo đều sử dụng kỹ thuật phân trang. Địa chỉ ảo là địa chỉ tham khảo từ chương trình . Tập tất cả địa chỉ ảo gọi là vùng địa chỉ ảo, đối với những hệ thống không dùng bộ nhớ ảo , địa chỉ ảo cũng là địa chỉ vật lý.

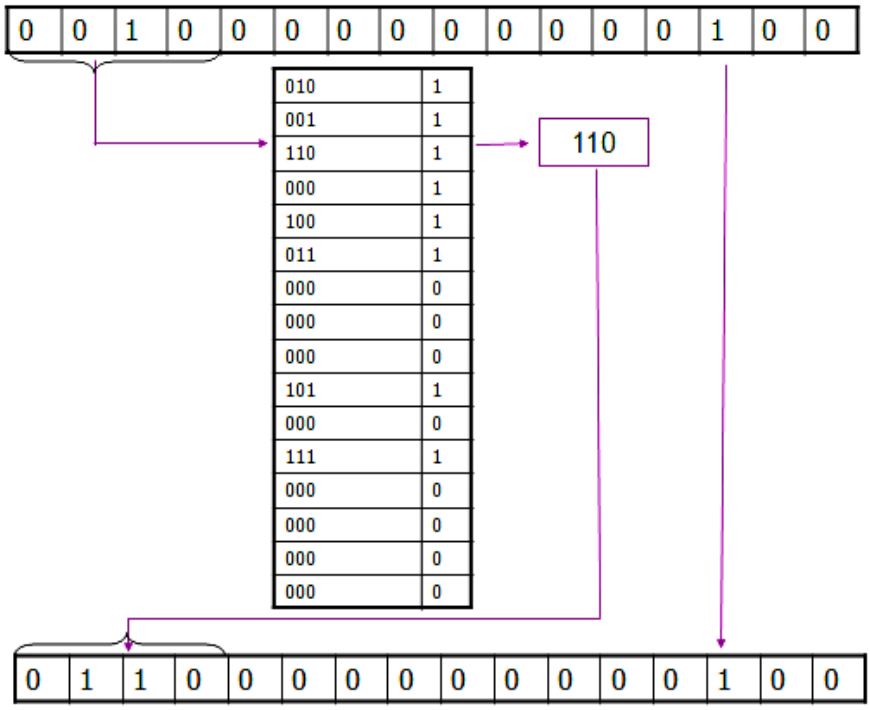
Vùng địa chỉ ảo được chi thành nhiều trang (page) có kích thước bằng nhau tướng ứng với những khung trang (Page frame) trong bộ nhớ vật lý. Trang và khung trang luôn có kích thước bằng nhau.

Ví dụ: có 32K bộ nhớ vật lý chia thành 8 khung trang có kích thước 4K và 64K bộ nhớ ảo chia thành 16 trang kích thước 4K. Trong bất kỳ thời điểm nào chỉ có 8 trang từ bộ nhớ ảo ánh xạ vào bộ nhớ vật lý như sau:



Hình 7.6

- Nếu lệnh nào đó trong chương trình tham khảo đến địa chỉ ảo không thuộc về bất kỳ trang nào đang được ánh xạ hiện thời gọi là lỗi trang. Trong trường hợp này HĐH tìm ra một khung trang nào ít sử dụng nhất trục xuất khỏi bộ nhớ và tìm kiếm trên đĩa trang nào chứa địa chỉ mà chương trình muốn tham khảo đến để chuyển vào bộ nhớ vật lý. Tiếp đến hệ thống cập nhật ánh xạ bộ nhớ cho trang mới.
- Bảng trang
  Hệ thống phải duy trì một bảng trang thể hiện ánh xạ từ bộ nhớ ảo vào bộ nhớ vật lý và chứa thông tin trang nào hiện thời đang được ánh xạ.



Hình 7.7

VII.5 Các thuật toán thay thế trang

Mỗi khi gặp lỗi trang HĐH phải chọn 1 trong các trang nào đó trục xuất ra khỏi bộ nhớ. Nếu nội dung trang đó bị thay đổi thì phải ghi nội dung mới của trang lên đĩa. Vấn đề là chọn trang nào để trục xuất để giảm tổn phí.

Xét ví dụ một tiến trình truy xuất đến một dãy các trang sau:

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

giả sử có 3 khung trang và ban đầu cả 3 khung trang đều rỗng.

VII.5.1 Thuật toán FIFO

- Ghi nhận thời điểm 1 trang được mang vào bộ nhớ chính khi cần thay thế trang , thì trang lâu nhất sẽ được chọn để trục xuất.

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	1	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
*	*	*	*		*	*	*	*	*	*			*	*			*	*	*

Hình 7.8

- Với thuật toán này có thể có trang được chọn để thay thế có thể chứa nhiều dữ liệu cần thiết thường xuyên được nạp vào sớm , do vậy khi bị chuyển ra bộ nhớ phụ sẽ nhanh chóng gây ra lỗi trang.

VII.5.2 Thuật toán tối ưu

- Thay thế trang sẽ lâu nhất được sử dụng trong tương lai

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
*	*	*	*		*		*			*			*				*		

Hình 7.9

- Thuật toán này bảo đảm lỗi trang là ít nhất tuy nhiên thuật toán này không khả thi vì không thể biết trước dãy các trang được truy xuất theo thứ tự.

VII.5.3 Thuật toán lâu nhất chưa sử dụng (LRU)

- Với mỗi trang ghi nhận thời điểm cuối cùng trang được truy cập, trang được thay thế là trang lâu nhất chưa sử dụng.

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
*	*	*			*		*	*	*	*			*		*		*		

Hình 7.10

- Thuật toán đòi hỏi một cơ chế xác định thứ tự các trang theo thời điểm truy xuất cuối cùng. Dùng stack
  - Cần tổ chức một stack lưu trữ số hiệu các trang
  - Mỗi khi thực hiện một truy xuất đến 1 trang , số hiệu của trang sẽ được xóa khỏi vị trí hiện hành trong stack và đưa lên đỉnh stack.
  - Trang ở đỉnh stack là trang được chọn để thay thế.

VII.5.4 Thuật toán Not Recently Used (NRU)

- Thuật toán dựa trên sự tồn tại 2 bit gắn liền với mỗi trang để chỉ ra trang đó có vừa mới được tham khảo hay là được ghi lên hay không. Những bit này chứa trong bảng trang

- ✓ Bit R =1: nếu trang đó truy cập ngược lại chưa được truy cập
- ✓ Bit M =1 : Trang đó đã bị thay đổi ngược lại chưa bị thay đổi

- Cứ mỗi lần truy cập bộ nhớ cần phải cập nhật lại các bit này
- Khi một tiến trình bắt đầu cả 2 bit trong tất cả các trang được đặt bằng 0. theo một chu kỳ thời gian bit R được đặt lại bằng 0 để phân biệt những trang mới được truy cập với trang đã truy cập trước đó.
- Khi xuất hiện một lỗi trang HĐH duyệt toàn bộ bảng trang và chia chúng thành 4 lớp dựa vào các bit R, M như sau:

Class 0 : R=0, M=0
Class 1: R=0, M=1
Class 2: R=1, M=0
Class 3: R=1, M=1

Sau đó thuật toán chọn ra một trang để thay thế thuộc thuộc lớp thấp nhất khác rỗng.

CÂU HỎI VÀ BÀI TẬP

1. Trình bày hiện tượng phân mảnh nội vi và ngoại vi trong quá trình tổ chức quản lý bộ nhớ phân đoạn cố định và động ?
2. Trình bày thuật toán cấp phát bộ nhớ (thuật toán first Fit, Best Fit, Worst Fit) trong kỹ thuật quản lý bộ nhớ động bằng danh sách liên kết và kỹ thuật bitmap?
3. Trình bày thuật toán giải phóng bộ nhớ trong kỹ thuật quản lý bộ nhớ động bằng danh sách liên kết ?
4. Trình bày phương pháp chuyển đổi địa chỉ ảo sang địa chỉ vật lý trong phương pháp quản lý bộ nhớ ảo? Kích thước trang và khung trang, nội dung bảng trang như mục VII.4 ?
5. Trong một thống cài đặt bộ nhớ ảo theo kỹ thuật phân trang, giả sử một chương trình truy cập đến các trang sau: 1, 0,4, 2, 1, 3, 1, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 3, 0, 1. Giả sử hệ thống có 3 khung trang lúc đầu còn trống. Có bao nhiêu lỗi trang xảy ra khi hệ thống sử dụng thuật toán thay thế trang LRU, FIFO ?

---HẾT---

TÀI LIỆU THAM KHẢO

[1] **Giáo trình hệ điều hành 1, Lê Khắc Nhiên Ân**, Hoàng Kiếm
[2] **Giáo trình hệ điều hành 2, Lê Khắc Nhiên Ân**, Hoàng Kiếm
[3] **Hỗ trợ kỹ thuật lập trình hệ thống**, Nguyễn Tín
[4] **Virus tin học huyền thoại và thực tế**, Ngô Anh Vũ
[5] **Operating Systems**, Andrew S.Tanenbaum
[6] **Operating System Concepts**, Abraham Siberschatz, Peter B. Galvin

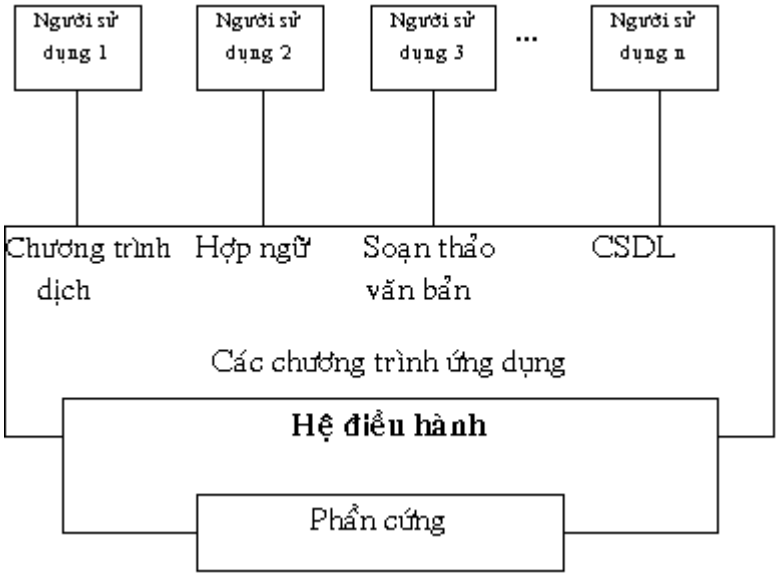
Mục lục

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)

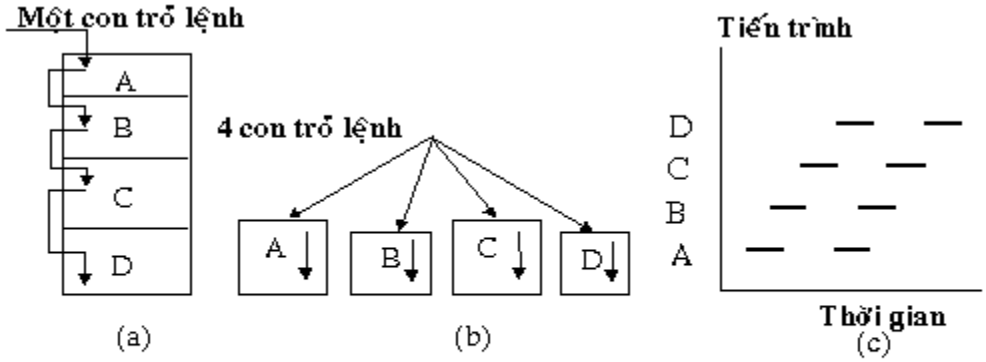
Để hỗ trợ sự đa chương, máy tính phải có khả năng thực hiện nhiều tác vụ đồng thời. Nhưng việc điều khiển nhiều hoạt động song song ở cấp độ phần cứng là rất khó

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)

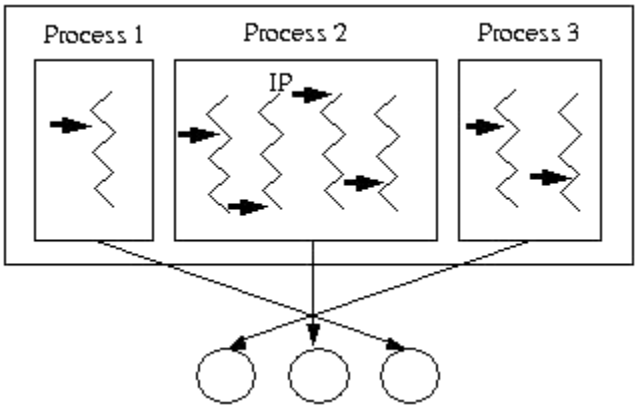
Trong hầu hết các hệ điều hành, **mỗi tiến trình có một không gian địa chỉ và chỉ có một dòng xử lý**. Tuy nhiên, có nhiều tình huống người sử dụng mong muốn có nhiều dòng xử lý cùng chia sẻ một không gian địa chỉ, và các dòng xử lý này hoạt động song song tương tự như các tiến trình phân biệt (ngoại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

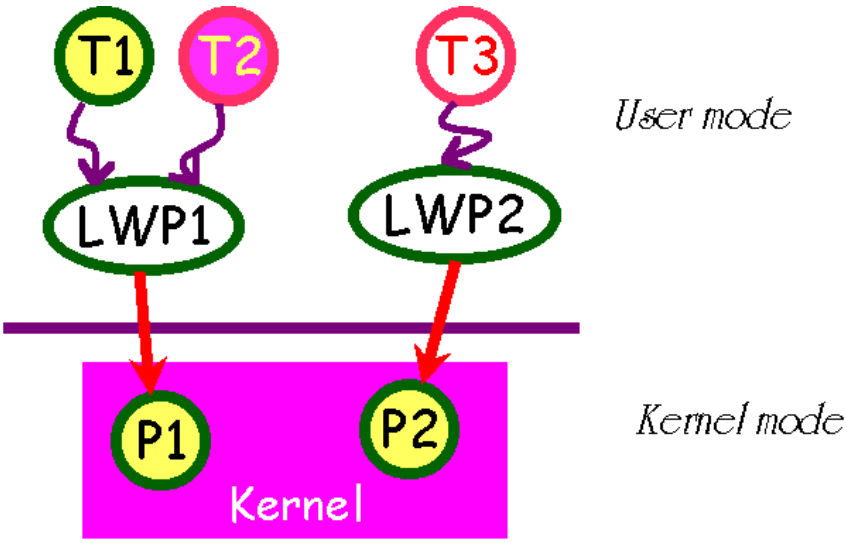
Tiến trình
Không gian địa chỉ
Tài nguyên toàn cục
Các thông tin thống kê

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

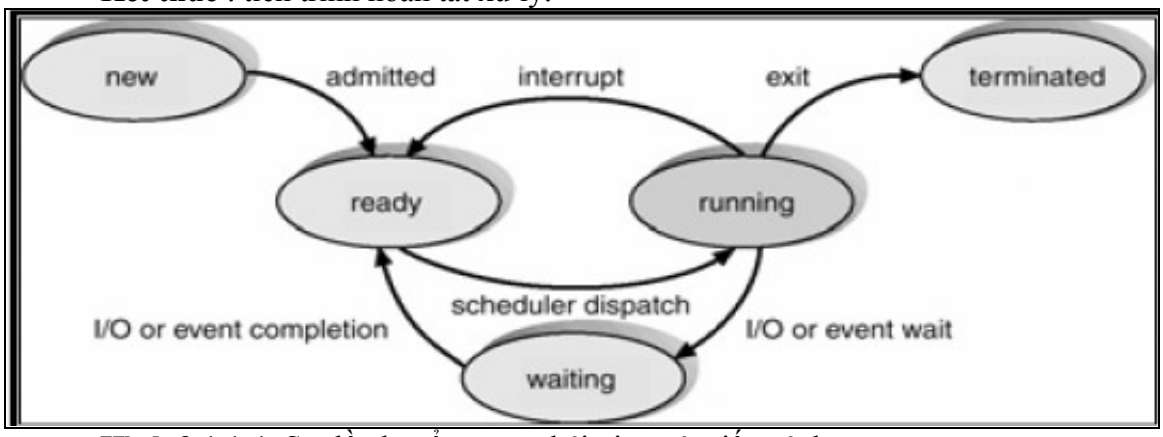
3.1. Tổ chức quản lý tiến trình

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.**

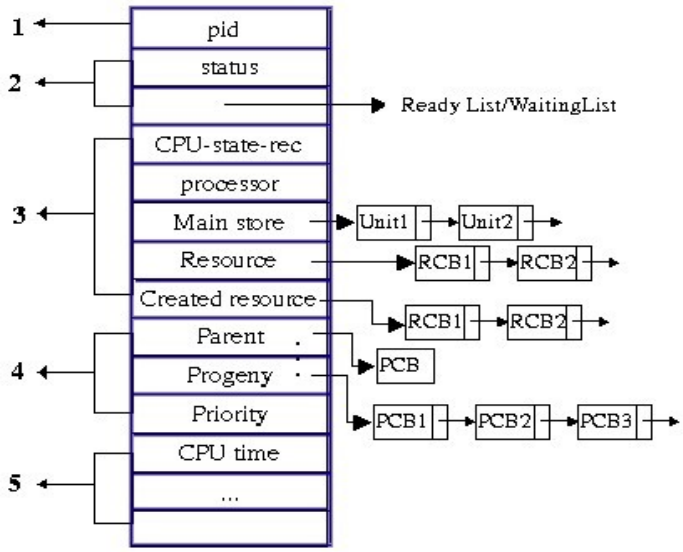
Hình 3.1.2-1. Hai chế độ xử lý

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.



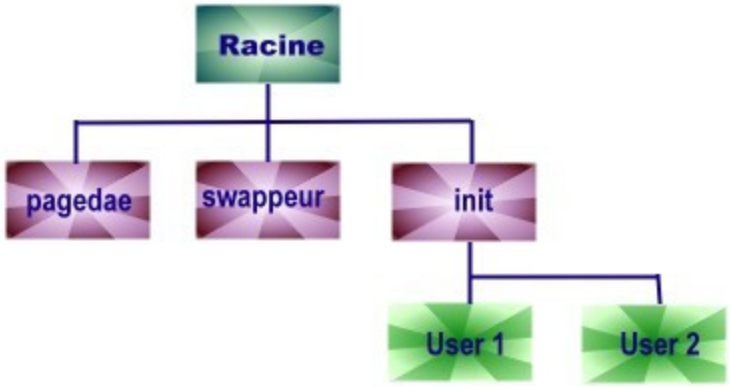
Hình 3.1.3-1. Khối điều khiển tiến trình

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.

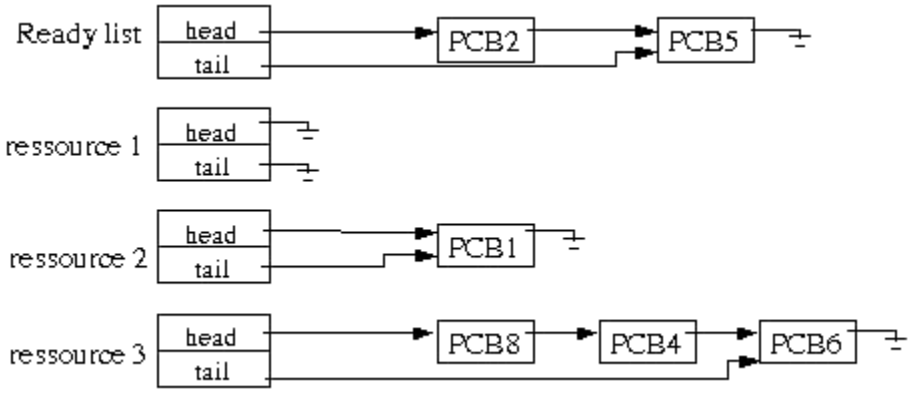
Đối với các hệ thống tương tác(time sharing), các hệ thời gian thực (real time),cần phải sử dụng nguyên lý điều phối không độc quyền để các tiến trình quan trọng có cơ hội hồi đáp kịp thời. Tuy nhiên thực hiện điều phối theo nguyên lý không độc quyền đòi hỏi những cơ chế phức tạp trong việc phân định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

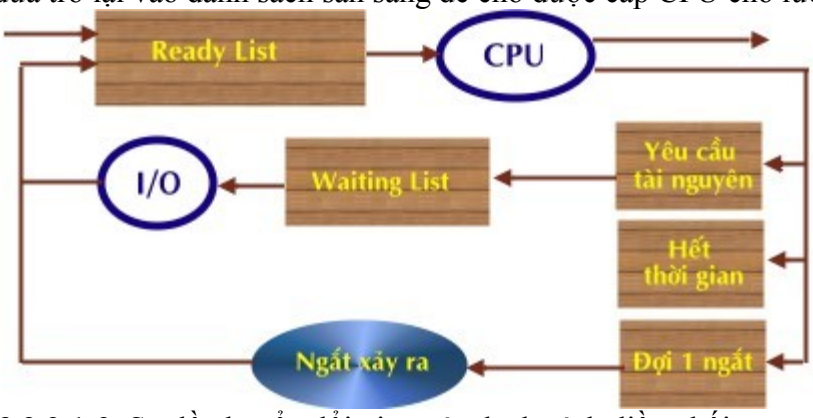


Hình 3.2.2.1-1. Các danh sách điều phối

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.

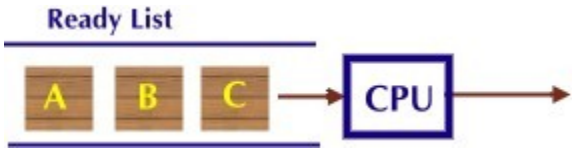
Trong nhiều hệ điều hành, có thể không có bộ điều phối tác vụ hoặc tách biệt rất ít đối với bộ điều phối tiến trình. Một vài hệ điều hành lại đưa ra một cấp độ điều phối trung gian kết hợp cả hai cấp độ điều phối tác vụ và tiến trình

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.

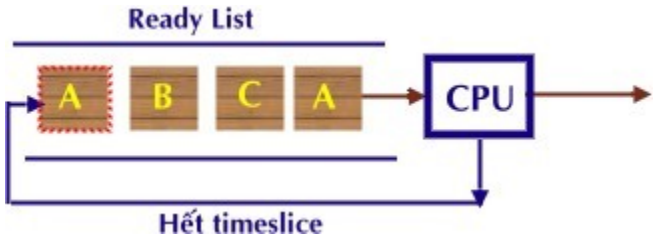
- **Thảo luận** : *Thời gian chờ trung bình không đạt cực tiểu, và biến đổi đáng kể đối với các giá trị về thời gian yêu cầu xử lý và thứ tự khác nhau của các tiến trình trong danh sách sẵn sàng. Có thể xảy ra hiện tượng tích lũy thời gian chờ, khi các tất cả các tiến trình (có thể có yêu cầu thời gian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

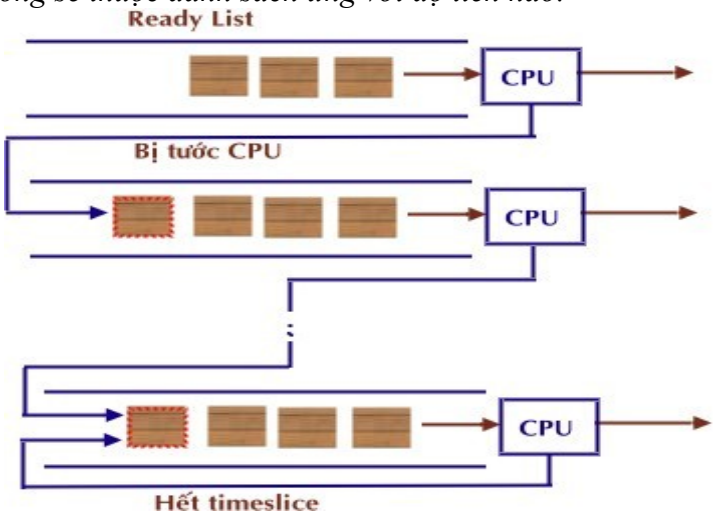
3.2.3.5. Chiến lược điều phối với nhiều mức độ ưu tiên

- **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

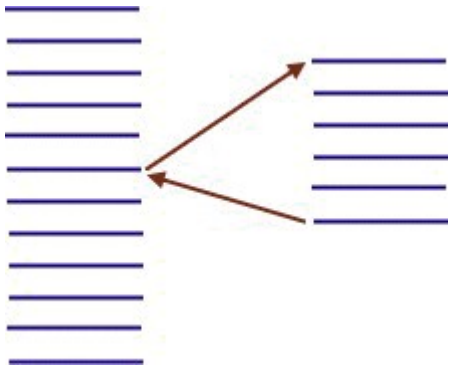
Mỗi tiến trình sỡ hữu một bảng biểu diễn các tín hiệu khác nhau. Với mỗi tín hiệu sẽ có tương ứng một trình xử lý tín hiệu *(signal handler)* qui định các xử lý của tiến trình khi nhận được tín hiệu tương ứng.

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.



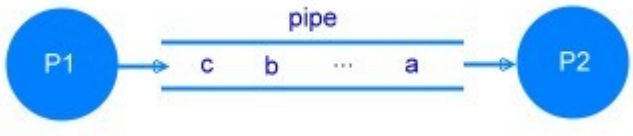
Hình 4.2.1-1. Liên lạc bằng tín hiệu

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 !

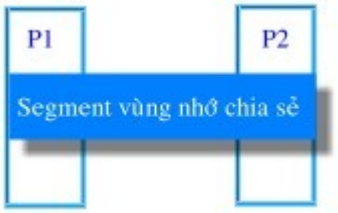
Cơ chế này cho phép truyền dữ liệu với cách thức không cấu trúc.

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**:**

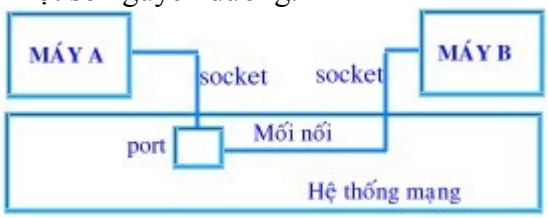
Một liên kết được thành lập giữa hai tiến trình. Trước khi mối liên kết này được thiết lập, một trong hai tiến trình phải đợi có một tiến trình khác yêu cầu kết nối.Có thể sử dụng socket để liên lạc theo mô hình client-serveur. Trong mô hình này, server sử dụng

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.

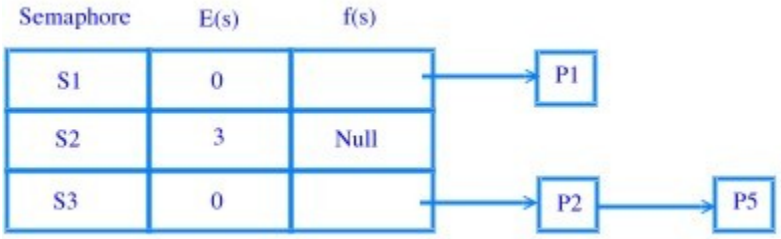
4.4.2.1. Semaphore

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 !

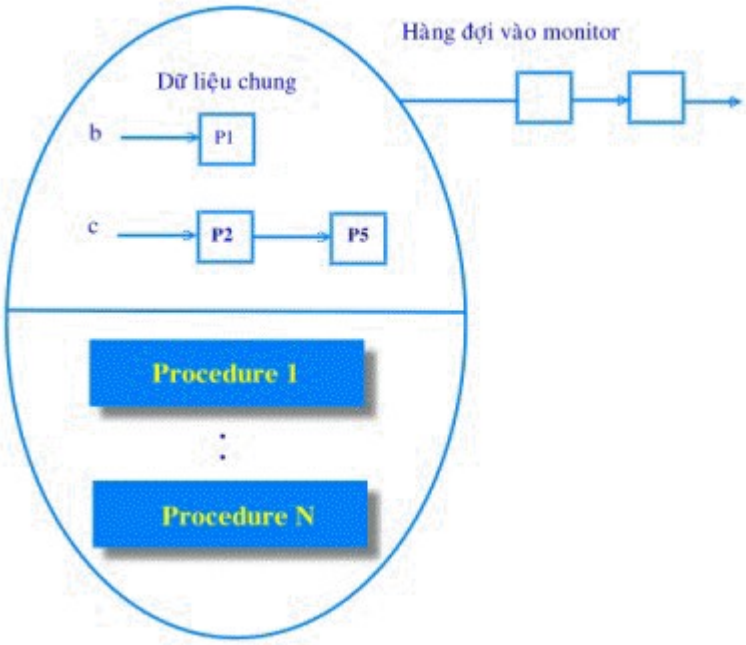
Vì thế việc sử dụng đúng cách semaphore để đồng bộ hóa phụ thuộc hoàn toàn vào lập trình viên và đòi hỏi lập trình viên phải hết sức thận trọng.

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

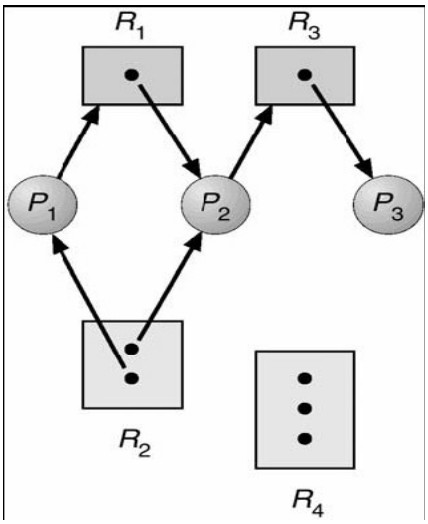
E. Một tập các đỉnh V được chia làm hai loại nút P = {P1, P2,…,Pn} là tập hợp các quá trình hoạt động trong hệ thống, và R = {R1, R2, ..., Rm} là tập hợp chứa tất cả các loại tài nguyên trong hệ thống.

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

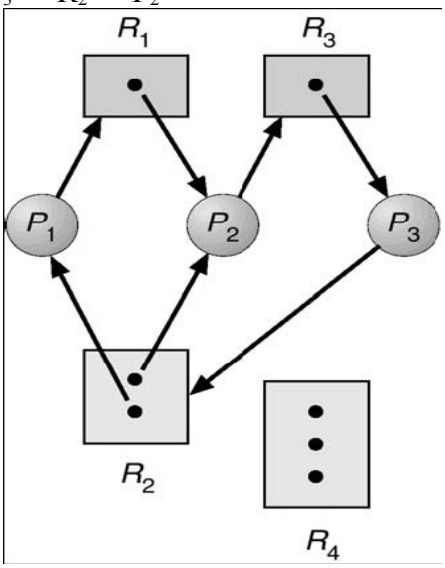
Đồ thị cấp phát tài nguyên hiển thị rằng, nếu đồ thị không chứa chu trình, thì không có quá trình nào trong hệ thống bị deadlock. Nếu đồ thị có chứa chu trình, thì deadlock có thể tồn tại. Nếu mỗi loại tài nguyên có chính xác một thể hiện, thì một chu trình ngụ ý rằng một deadlock xảy ra. Nếu một chu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

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

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

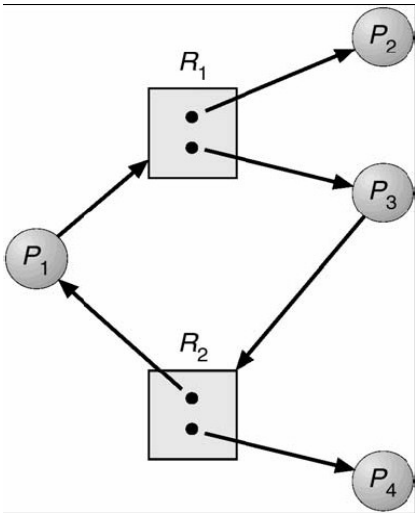


Đồ thị cấp phát tài nguyên với deadlock

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.

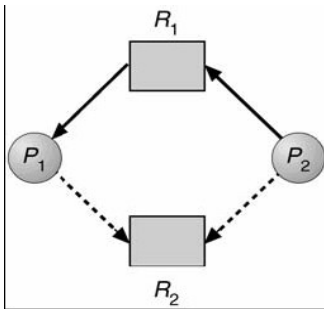
5.5.2. Giải thuật đồ thị cấp phát tài nguyên

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.

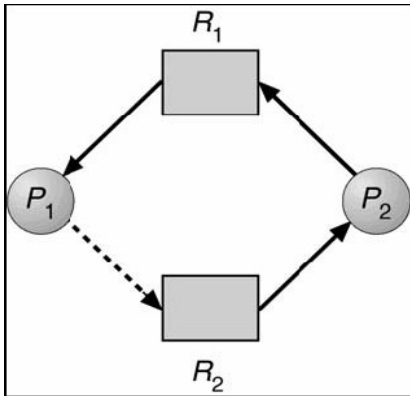
Giả sử rằng Pi yêu cầu tài nguyên Rj. Yêu cầu có thể được gán chỉ nếu chuyển cạnh yêu cầu Pi → Rj tới cạnh gán Rj→Pi không dẫn đến việc hình thành chu trình trong đồ thị cấp phát tài nguyên. Chú ý rằng chúng ta kiểm tra tính an toàn bằng cách dùng giải thuật phát hiện chu trình. Một giải thuật để phát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

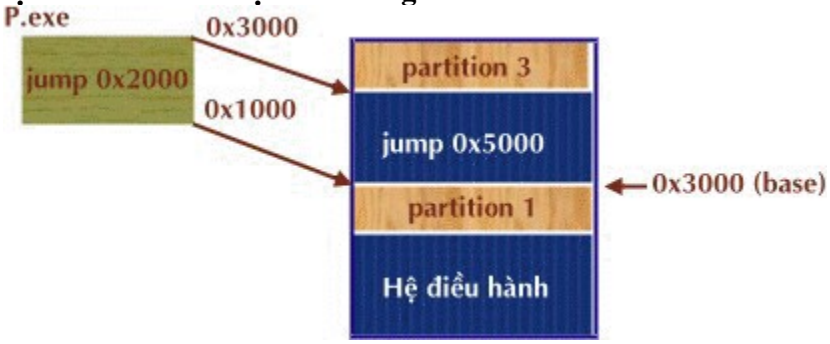
Chương trình của người sử dụng chỉ thao tác trên các địa chỉ ảo, không bao giờ nhìn thấy các địa chỉ vật lý . Địa chỉ thật sự ứng với vị trí của dữ liệu trong bô nhớ chỉ được xác định khi thực hiện truy xuất đến dữ liệu.

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**.

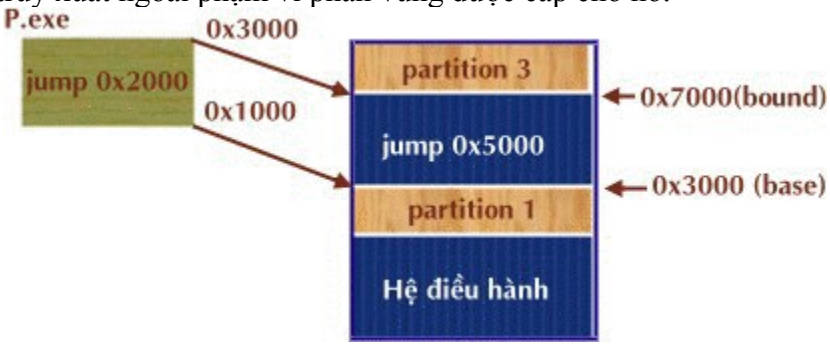


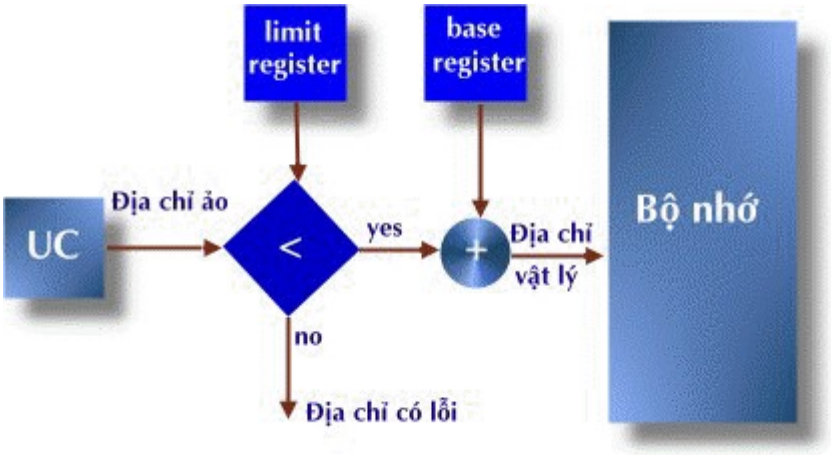
Thảo luận:

- 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.

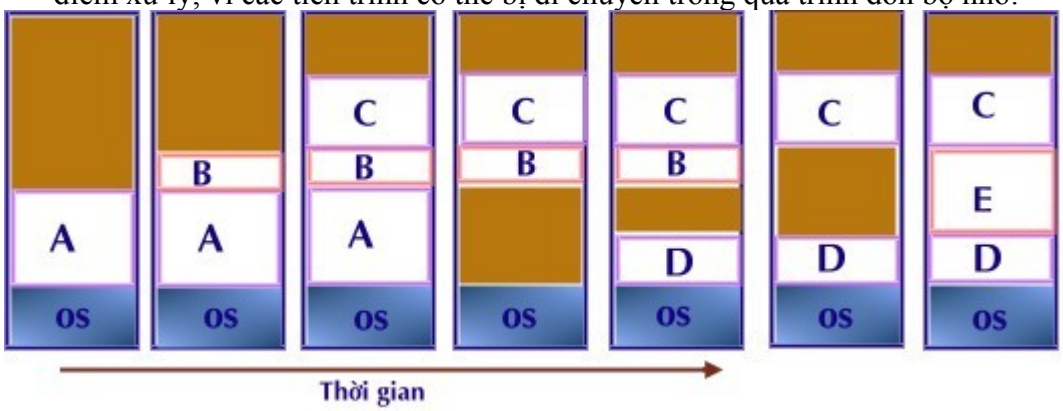




Hai thanh ghi hỗ trợ chuyển đổi địa chỉ

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:

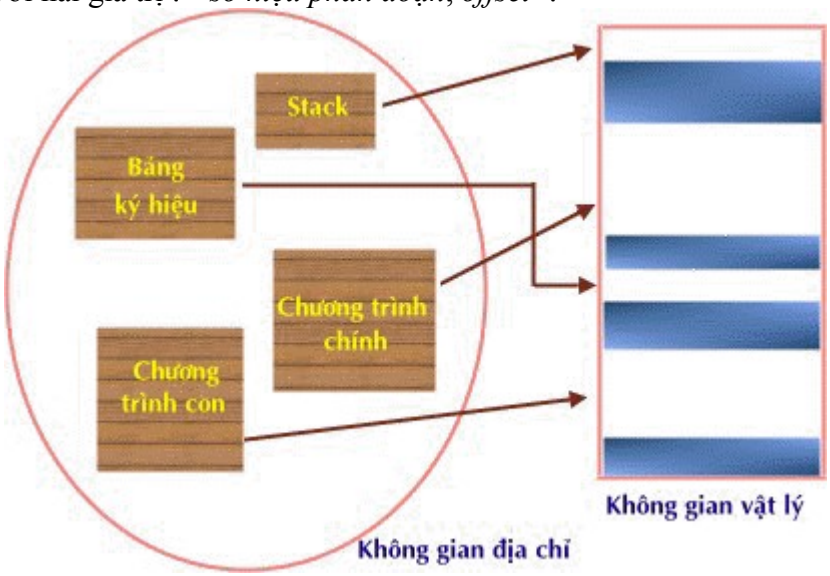
- Dời chỗ tiến trình: di chuyển tiến trình đến một vùng nhớ khác đủ lớn để thỏa mãn nhu cầu tăng trưởng của tiến trình.

- 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:

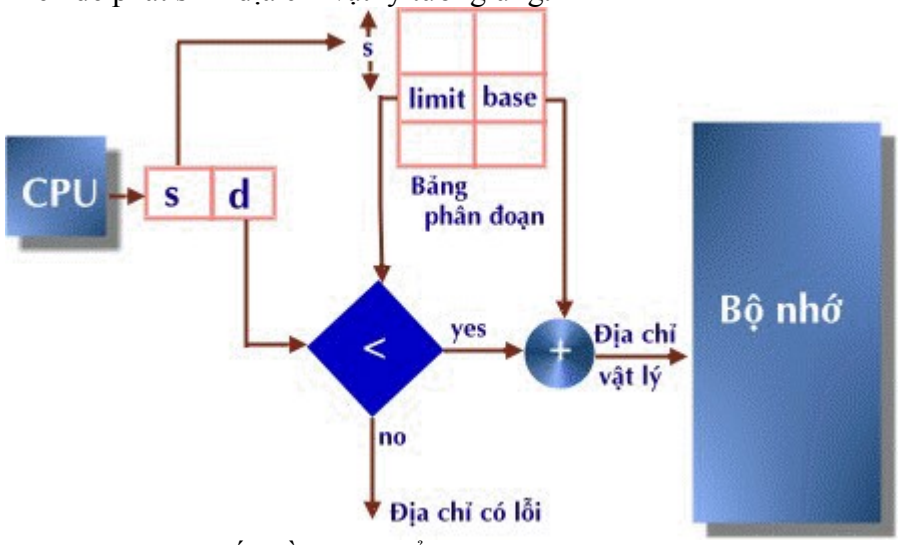
Cần phải xây dựng một ánh xạ để chuyển đổi các địa chỉ 2 chiều được người dùng định nghĩa thành địa chỉ vật lý một chiều. Sự chuyển đổi này được thực hiện qua một *bảng phân đoạn*. Mỗi thành phần trong bảng phân đoạn bao gồm một *thanh ghi nền* và

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.

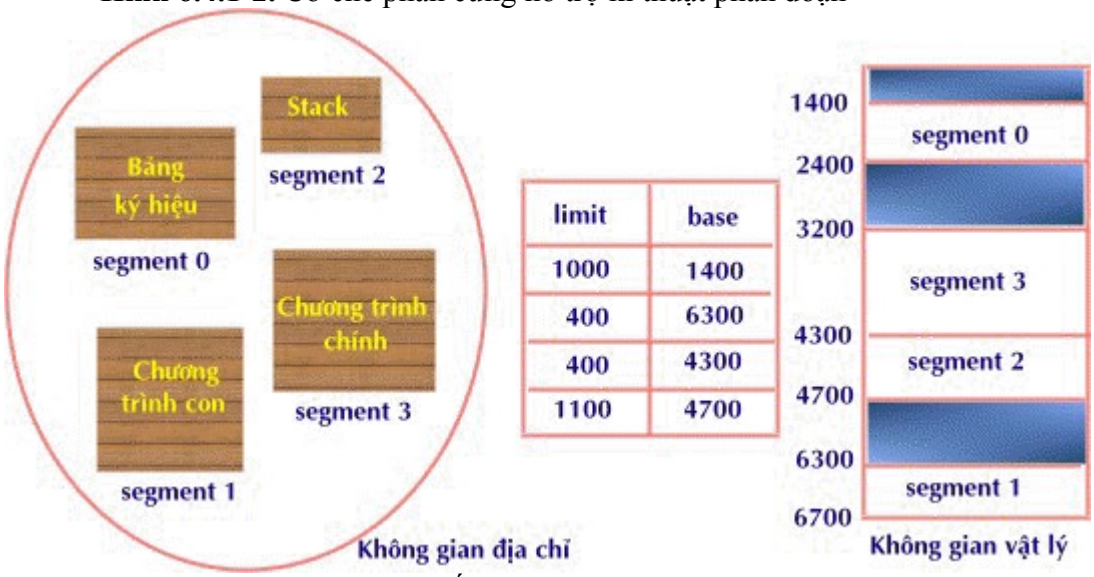
Chuyển đổi địa chỉ:

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

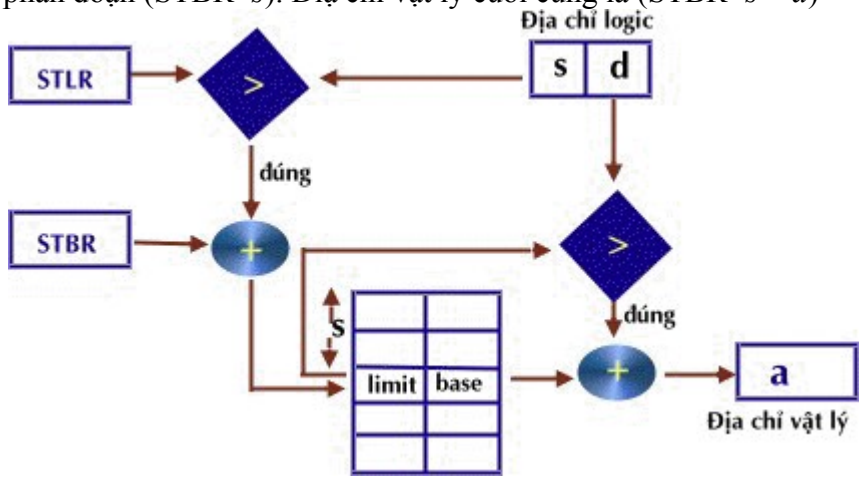


Hình 6.4.1-3. Hệ thống phân đoạn

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*)

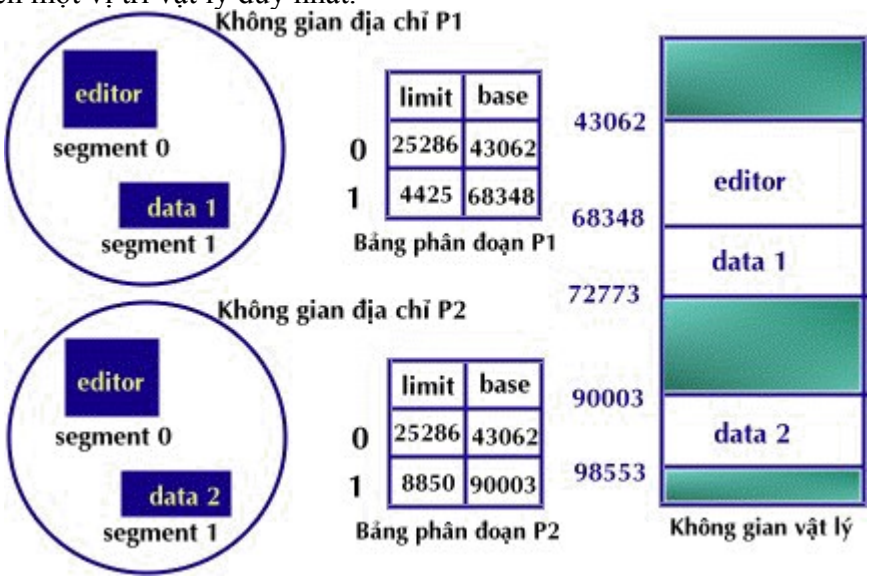


Hình 6.4.1-4. Sử dụng STBR, STLR và bảng phân đoạn

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.

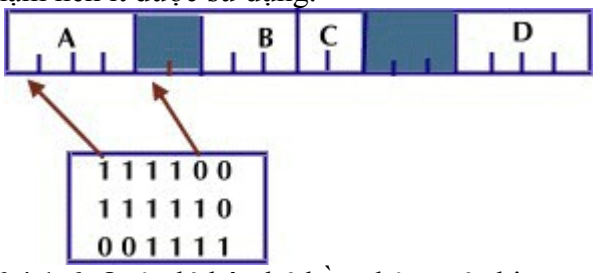
Chia sẻ phân đoạn: Một ưu điểm khác của kỹ thuật phân đoạn là khả năng chia sẻ ở mức độ phân đoạn. Nhờ khả năng này, các tiến trình có thể chia sẻ với nhau từng phần chương trình (ví dụ các thủ tục, hàm), không nhất thiết phải chia sẻ toàn bộ chương trình như trường hợp phân trang. Mỗi tiến trình có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

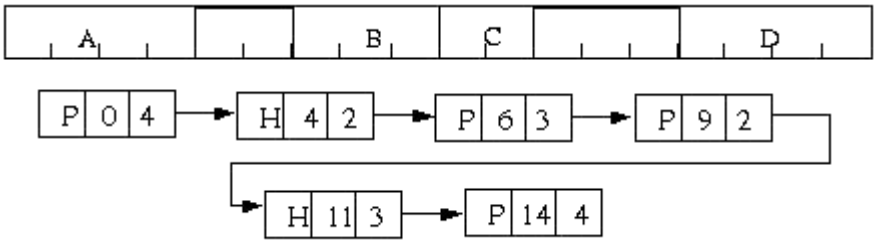
Thảo luận:

- 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.

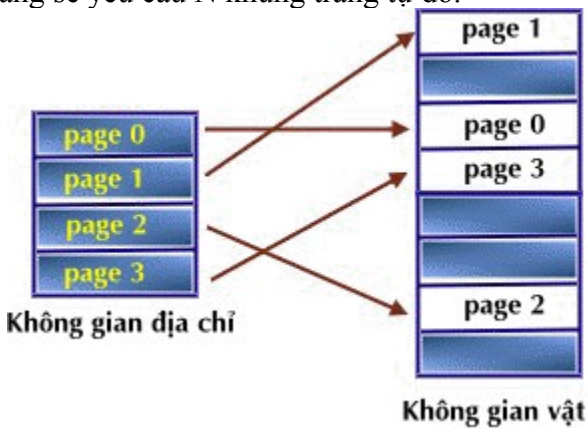
Trong hệ thống sử dụng kỹ thuật phân đoạn , hiện tượng phân mảnh ngoại vi lại xuất hiện khi các khối nhớ tự do đều quá nhỏ, không đủ để chứa một phân đoạn.

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:

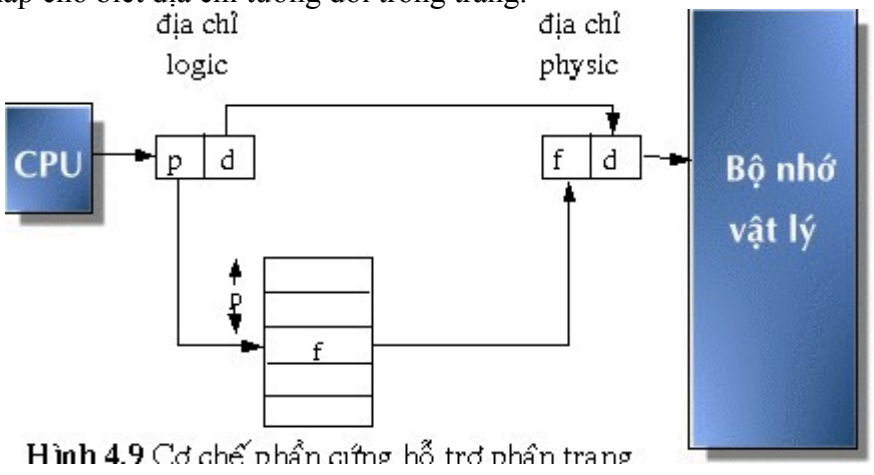
Cơ chế phần cứng hỗ trợ thực hiện chuyển đổi địa chỉ trong cơ chế phân trang là bảng trang (*pages table*). Mỗi phần tử trong bảng trang cho biết các địa chỉ bắt đầu của vị trí lưu trữ trang tương ứng trong bộ nhớ vật lý (số hiệu khung trang trong bộ nhớ vật lý đang chứa trang).

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.



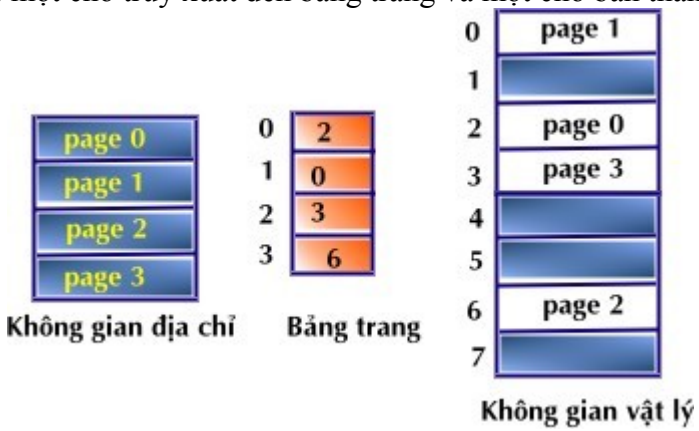
Hình 6.4.2-2. Cơ chế phần cứng hỗ trợ phân trang

Cài đặt bảng trang:

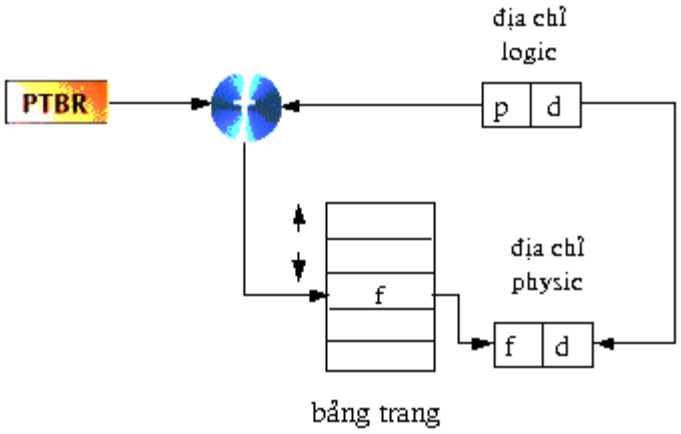
Trong trường hợp đơn giản nhất, bảng trang một tập các thanh ghi được sử dụng để cài đặt bảng trang. Tuy nhiên việc sử dụng thanh ghi chỉ phù hợp với các bảng trang

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!



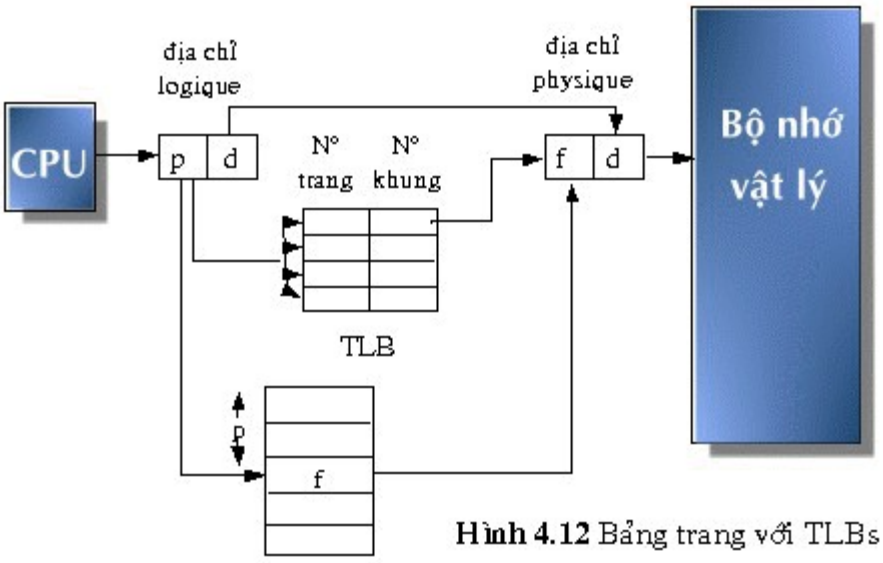
Hình 6.4.2-3. Mô hình bộ nhớ phân trang



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.



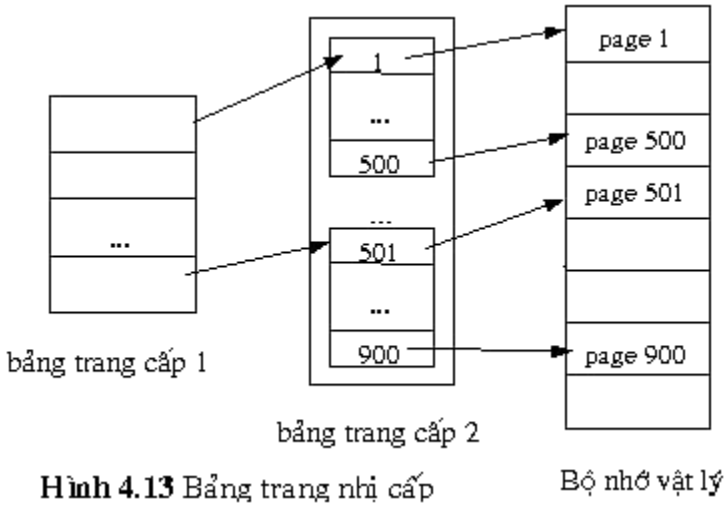
Hình 4.12 Bảng trang với TLBs

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

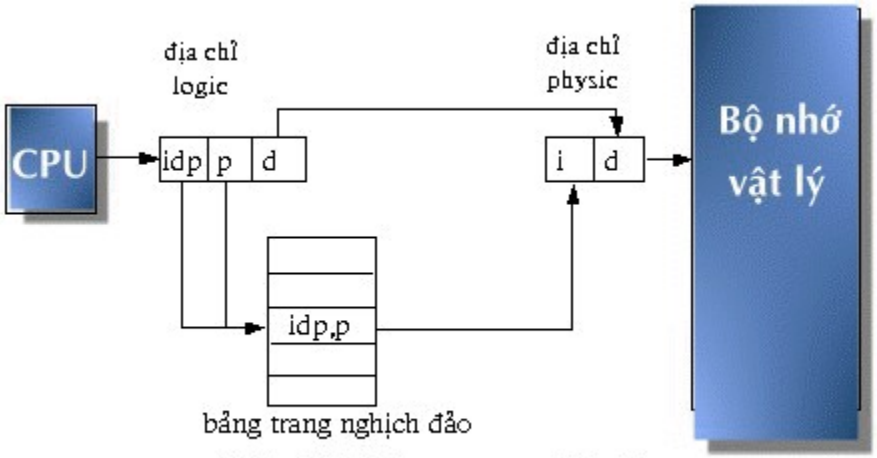
Hình 6.4.2-6. Bảng trang cấp 2

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).

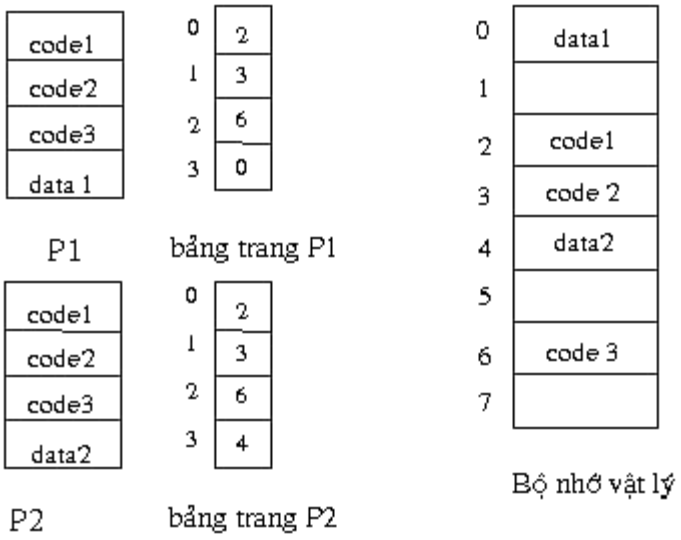
- *Hợp lệ* : trang tương ứng thuộc về không gian địa chỉ của tiến trình.
- *Bất hợp lệ* : trang tương ứng không nằm trong không gian địa chỉ của tiến trình, điều này có nghĩa tiến trình đã truy xuất đến một địa chỉ không được phép.

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

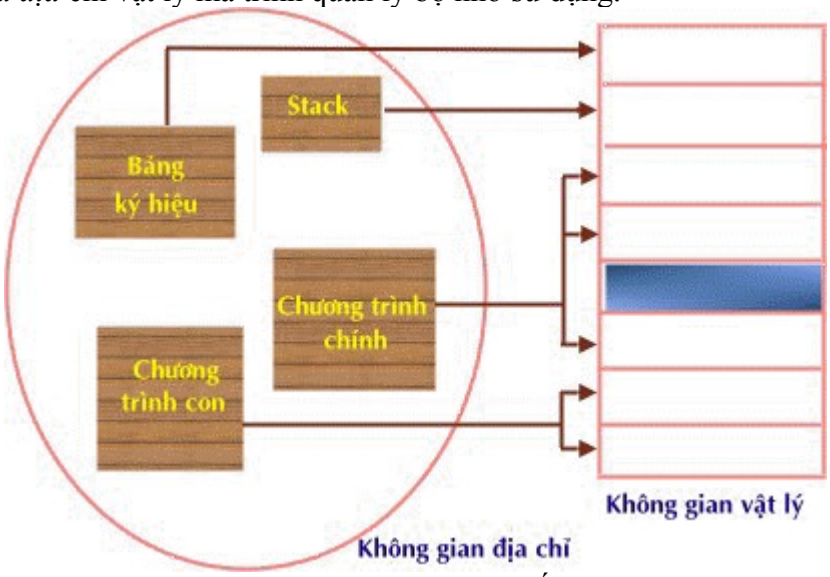
Kỹ thuật phân đoạn thõa mãn được nhu cầu thể hiện cấu trúc logic của chương trình nhưng nó dẫn đến tình huống phải cấp phát các khối nhớ có kích thước khác nhau cho các phân đoạn trong bộ nhớ vật lý. Điều này làm rắc rối vấn đề hơn rất nhiều so với việc

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*.

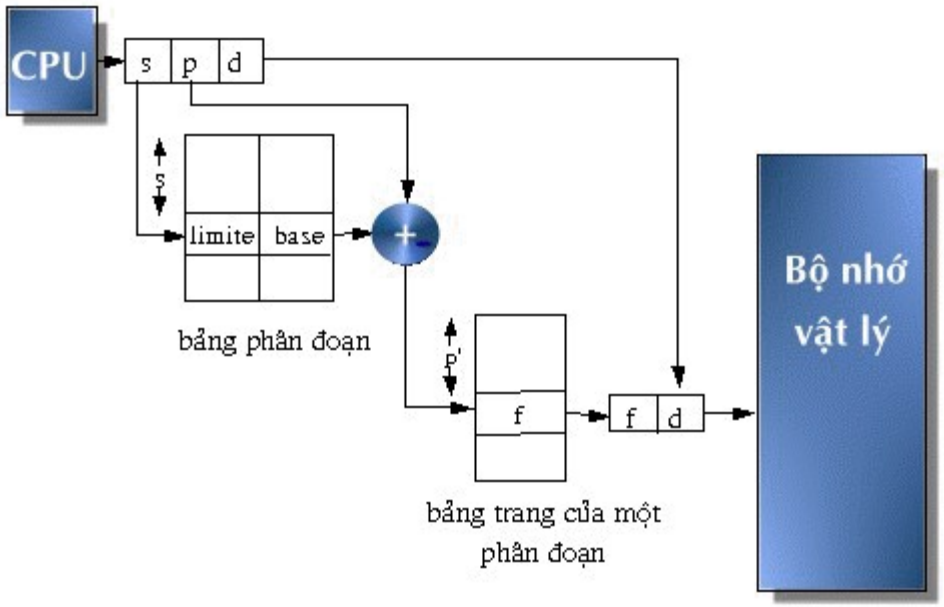
6.4.3. Phân đoạn kết hợp phân trang (Paged segmentation)

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.

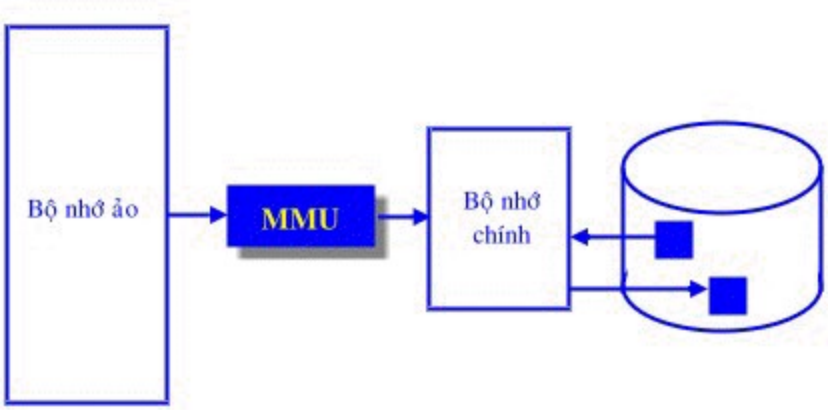
Để giải phóng lập trình viên khỏi các suy tư về giới hạn của bộ nhớ, mà cũng không tăng thêm khó khăn cho công việc lập trình của họ, người ta nghĩ đến các kỹ thuật tự động, cho phép xử lý một chương trình có kích thước lớn chỉ với một vùng nhớ có kích thước nhỏ . Giải pháp được tìm thấy với khái niệm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

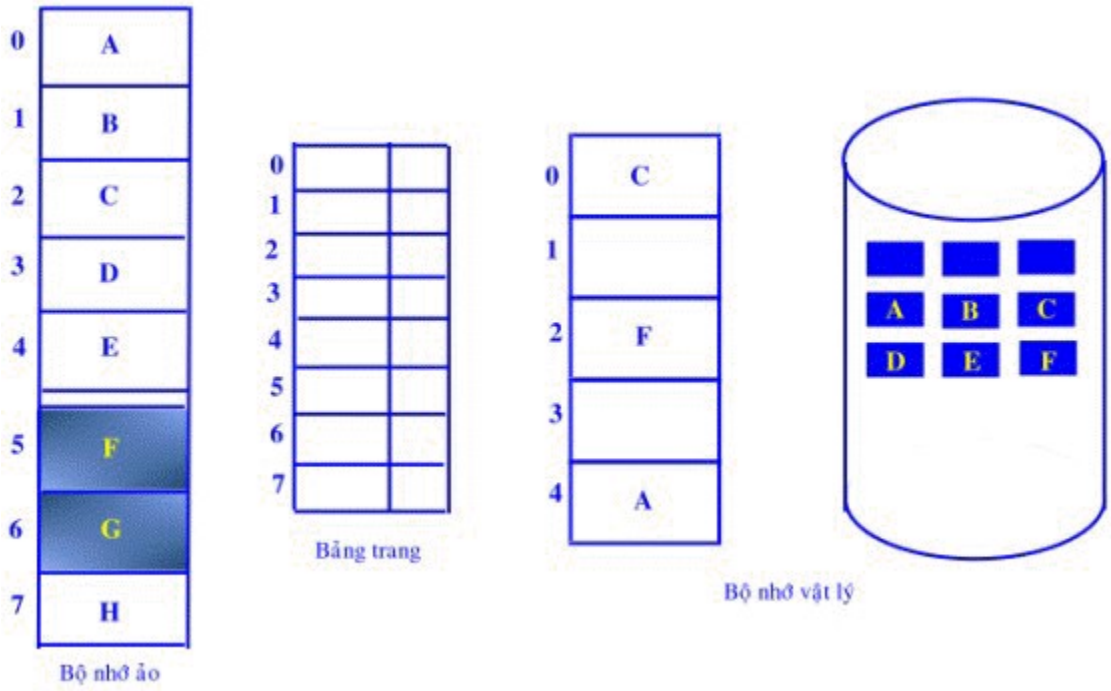
- *valid* : trang tương ứng là hợp lệ và đang ở trong bộ nhớ chính .
- *invalid* : hoặc trang bất hợp lệ (không thuộc về không gian địa chỉ của tiến trình) hoặc trang hợp lệ nhưng đang được lưu trên bộ nhớ phụ.

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.*

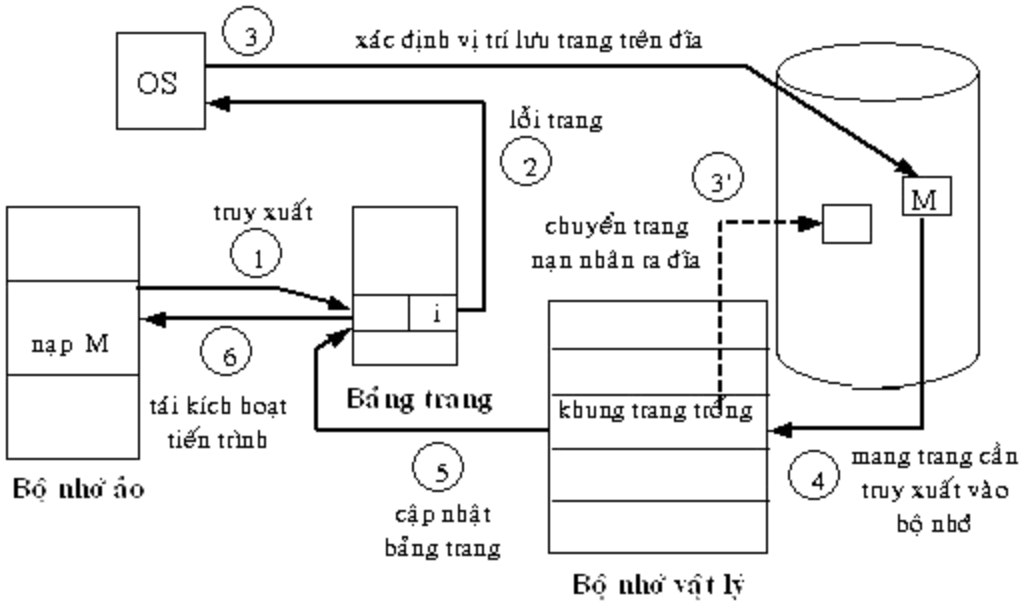


Hình 6.5.1.2-1. Bảng trang với một số trang trên bộ nhớ phụ

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 »**

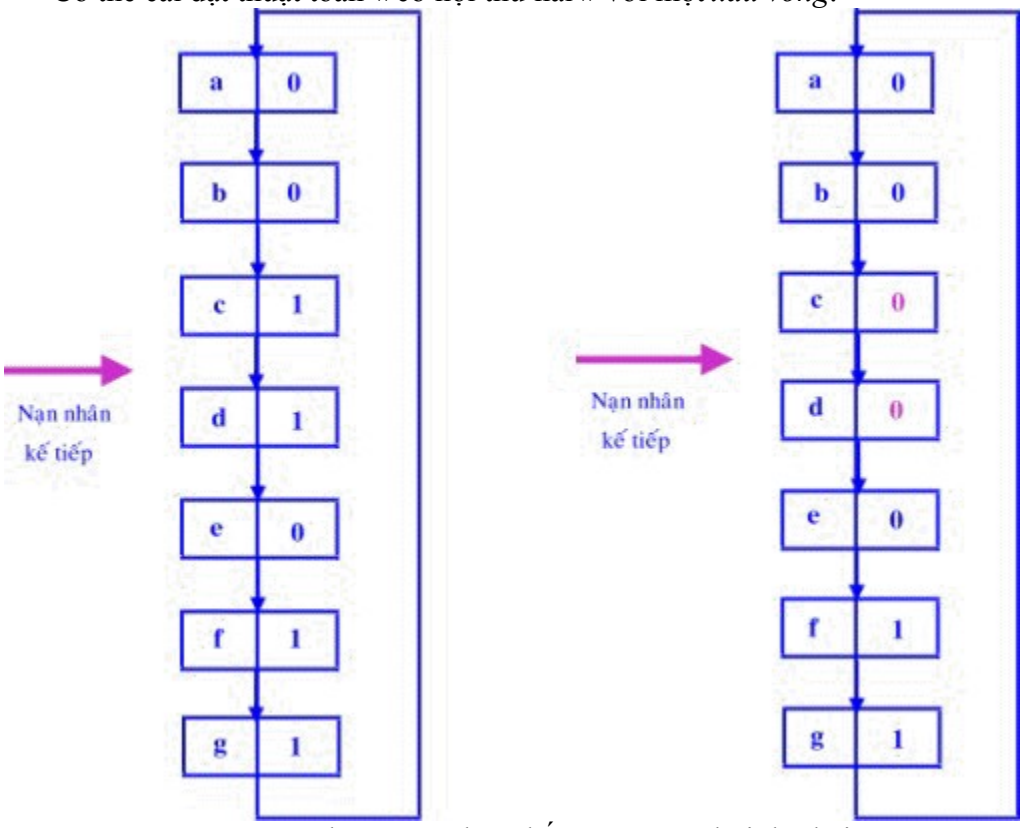
Tiếp cận: Sử dụng một bit reference duy nhất. Thuật toán cơ sở vẫn là FIFO, tuy nhiên khi chọn được một trang theo tiêu chuẩn FIFO, kiểm tra bit reference của trang đó :

- 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.

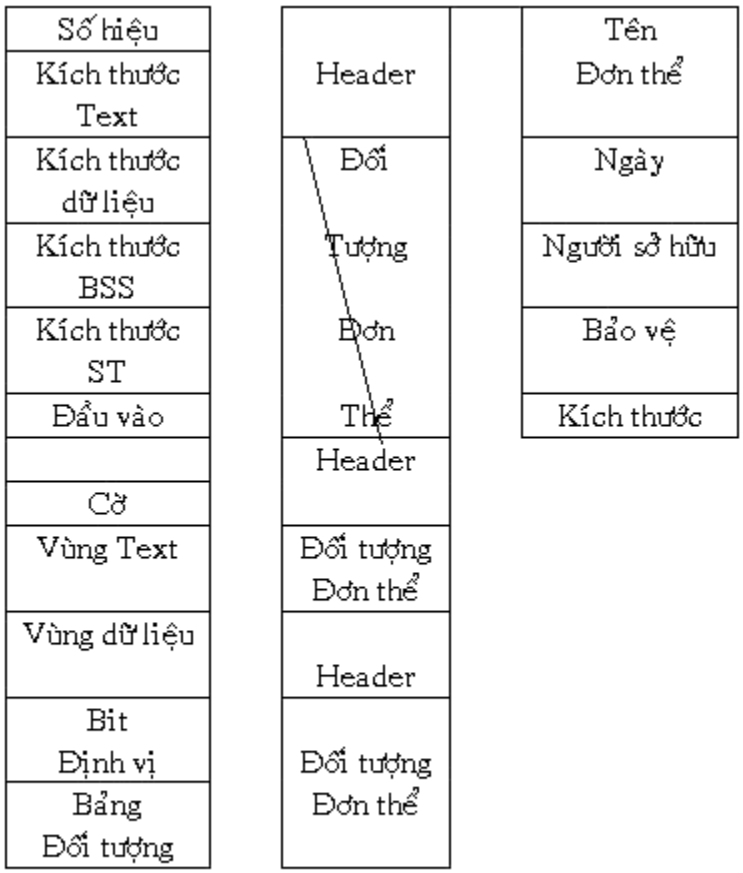
- *Tập tin thường* : là tập tin text hay tập tin nhị phân chứa thông tin của người sử dụng.
- *Thư mục* : là những tập tin hệ thống dùng để lưu giữ cấu trúc của hệ thống tập tin.
- *Tập tin có ký tự đặc biệt* : liên quan đến nhập xuất thông qua các thiết bị nhập xuất tuần tự như màn hình, máy in, mạng.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

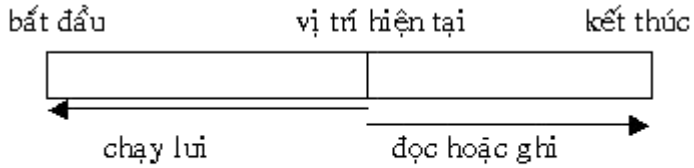
Truy xuất tập tin :

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 :

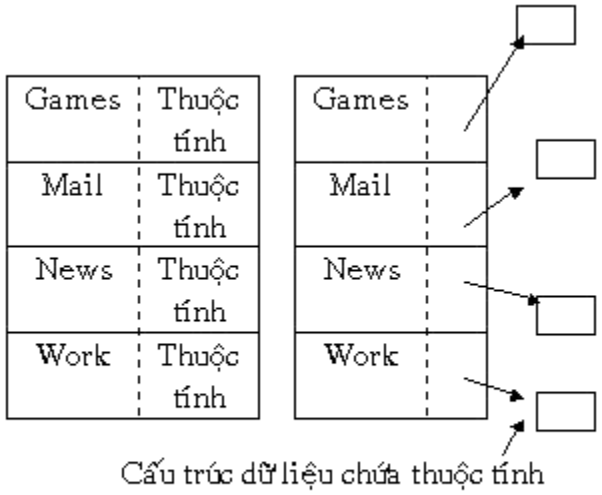
Một thư mục thường thường chứa một số *entry*, mỗi entry cho một tập tin. Mỗi entry chứa tên tập tin, thuộc tính và địa chỉ trên đĩa lưu dữ liệu hoặc một entry chỉ chứa tên tập tin và một con trỏ, trỏ tới một cấu trúc, trên đó có thuộc tính và vị trí lưu trữ của tập tin.

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.

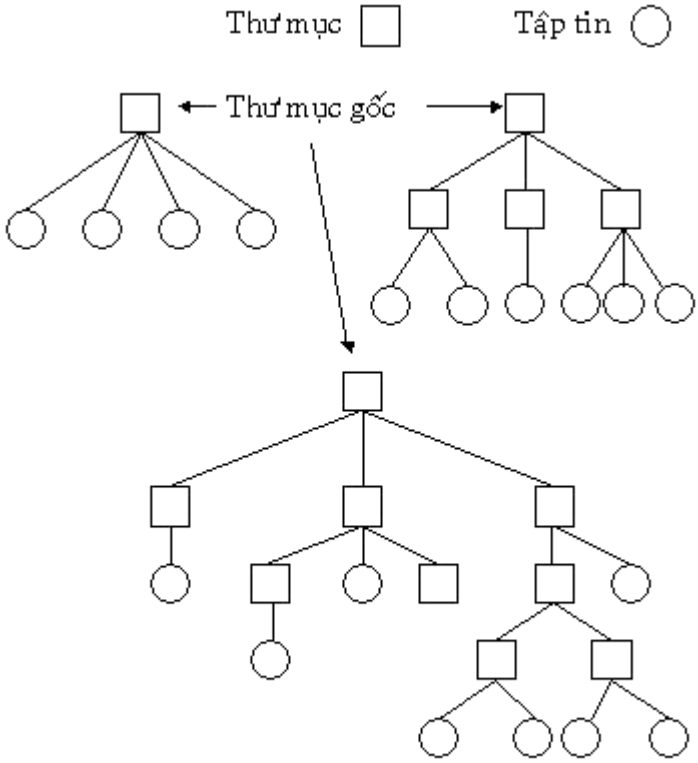
Số lượng thư mục trên mỗi hệ thống là khác nhau. Thiết kế đơn giản nhất là hệ thống chỉ có thư mục đơn(còn gọi là thư mục một cấp), chứa tất cả các tập tin của tất cả người dùng, cách này dễ tổ chức và khai thác nhưng cũng dễ gây ra khó khăn khi có nhiều người sử dụng vì sẽ có nhiều tập tin trùng tên.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

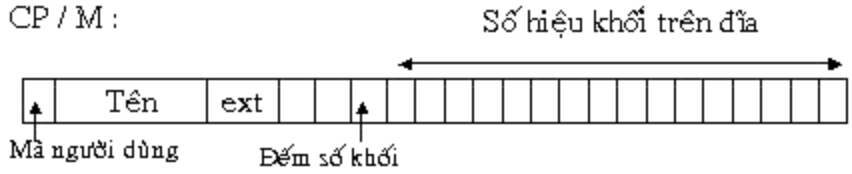
7.3.1. Bảng quản lý tệp tin, thư mục

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

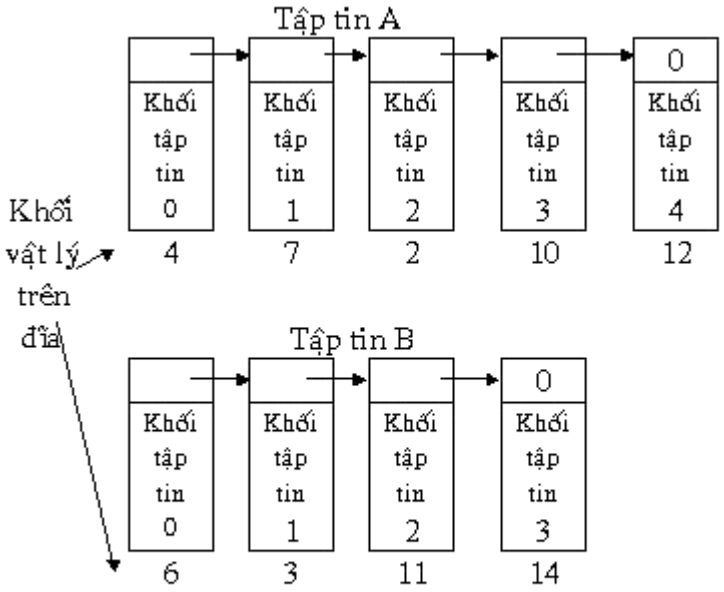
Định vị liên tiếp:

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:



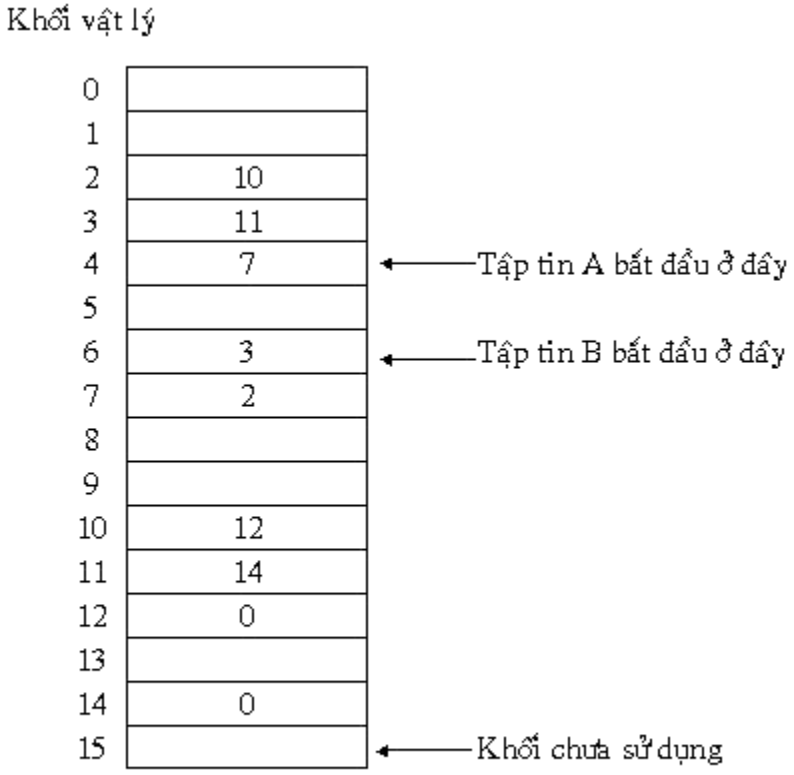
Hình 9.2 Định vị bằng danh sách liên kết

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

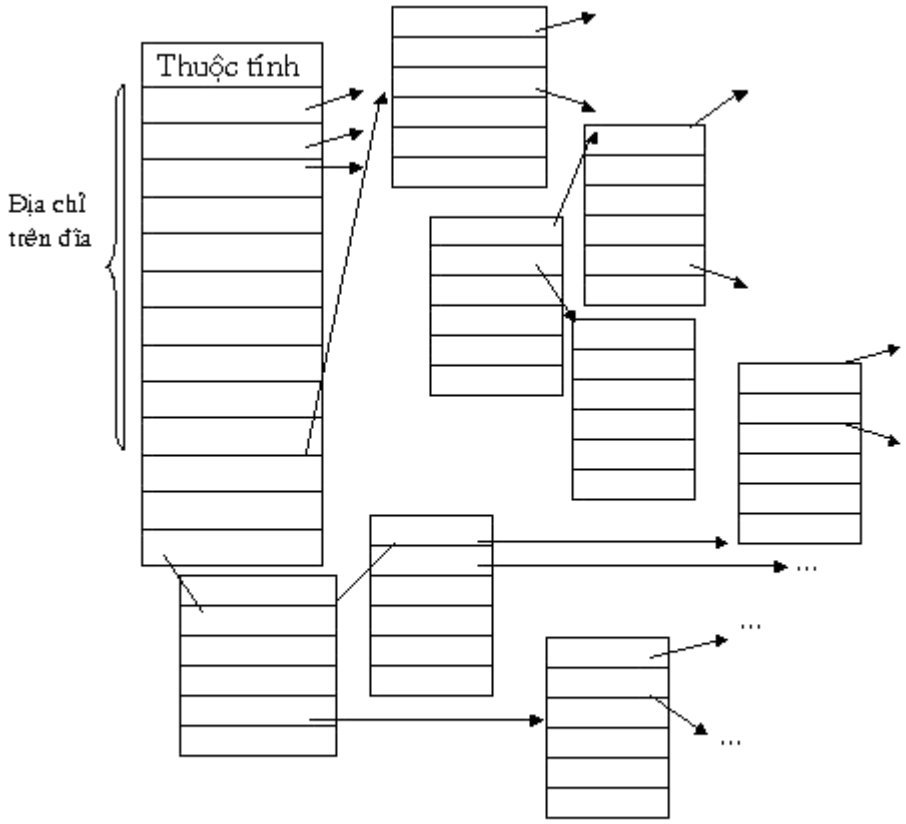
Hình 7.3.2.2-2. Bảng chỉ mục của danh sách liên kết

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

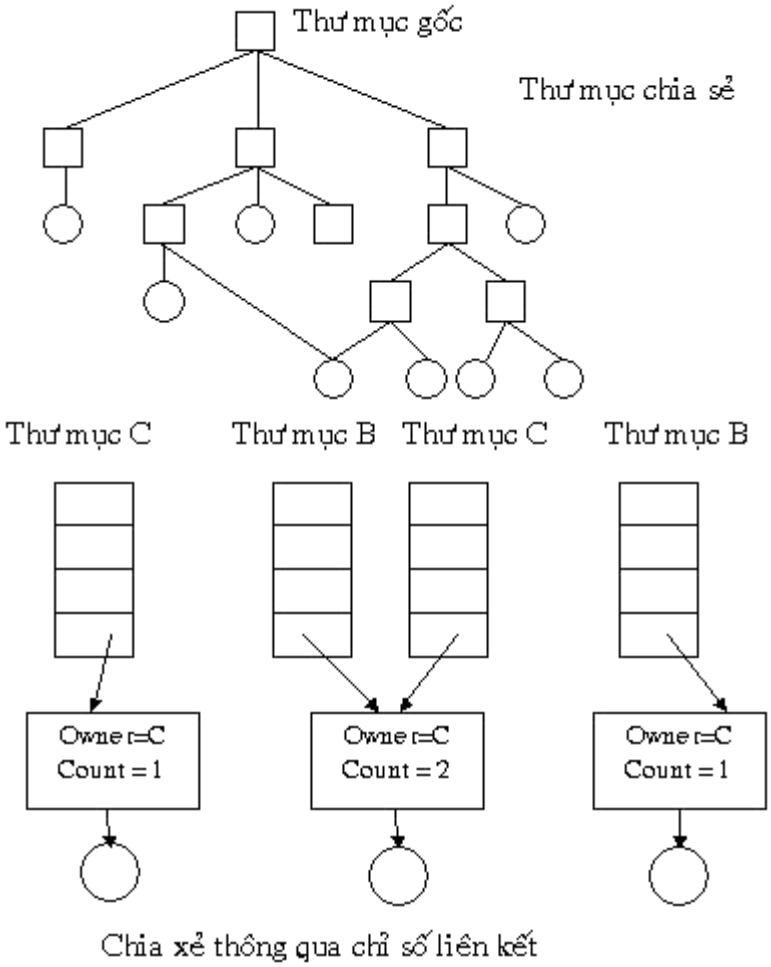
Hình 7.3.2.2-3. Cấu trúc I-node

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.

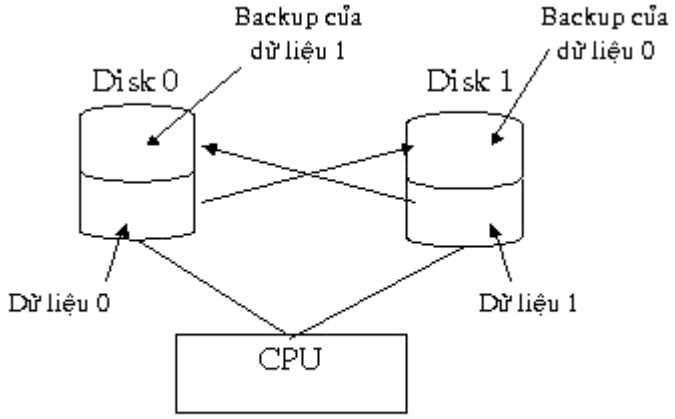
7.3.4.2. Sao lưu

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.

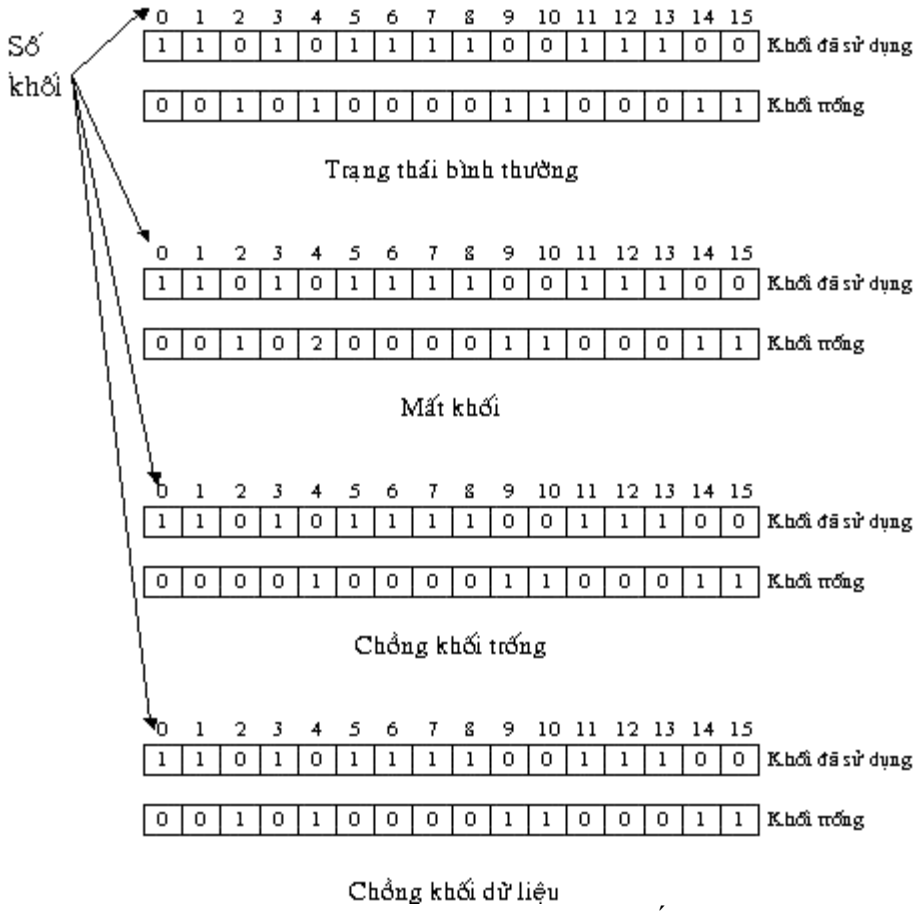


Hình 9.7 Backup

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 :

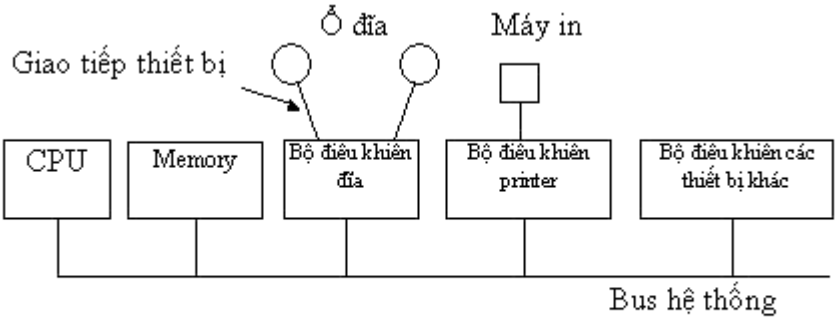
- Bộ xử lý kiểm soát trực tiếp các thiết bị ngoại vi.
- Hệ thống có thêm bộ điều khiển thiết bị. Bộ xử lý sử dụng cách thực hiện nhập xuất thứ nhất. Theo cách này bộ xử lý được tách rời khỏi các mô tả chi tiết của các thiết bị ngoại vi.
- Bộ xử lý sử dụng thêm cơ chế ngắt.
- Sử dụng cơ chế DMA, bộ xử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

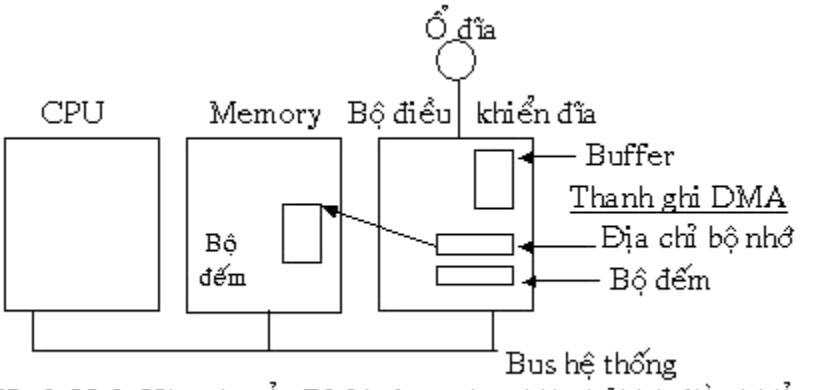
Hệ điều hành thực hiện nhập xuất bằng cách ghi lệnh lên các thanh ghi của bộ điều khiển. Ví dụ : bộ điều khiển đĩa mềm của IBMPC chấp nhận 15 lệnh khác nhau như : READ, WRITE, SEEK, FORMAT, RECALIBRATE, một số lệnh có tham số và các tham số cũng được nạp vào thanh ghi. Khi một lệnh đã được chấp nhận,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

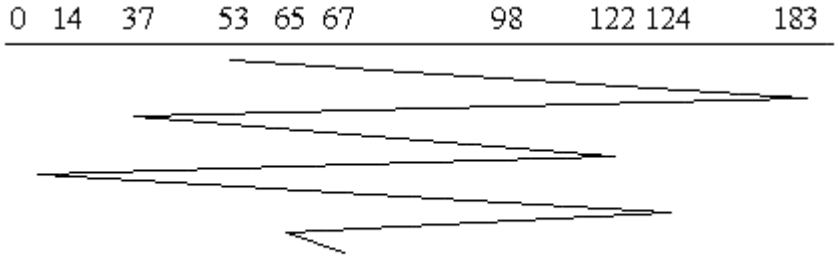
Một đĩa bao gồm nhiều cylinder, mỗi cylinder chứa nhiều track trên các head. Mỗi track được chia làm nhiều sector (từ 8 đến 32). Mỗi sector có số byte là như nhau dù vị trí của nó ở gần tâm hay ở ngoài rìa đĩa, những khoảng trống thừa không dùng đến.
Một đặc điểm thiết bị cài đặt quan trọng cho driver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

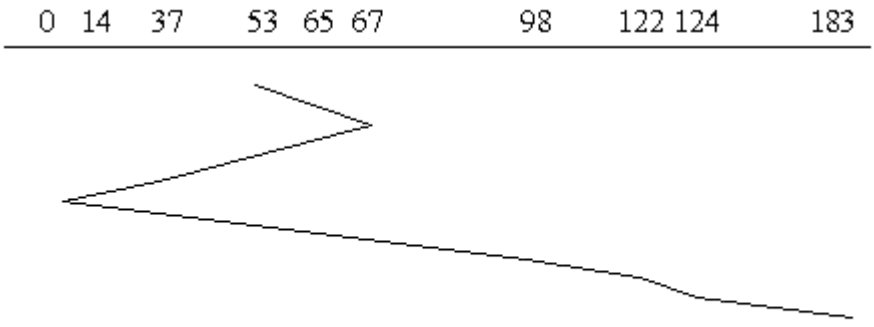


Hình 12.1 Phương pháp FCFS

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

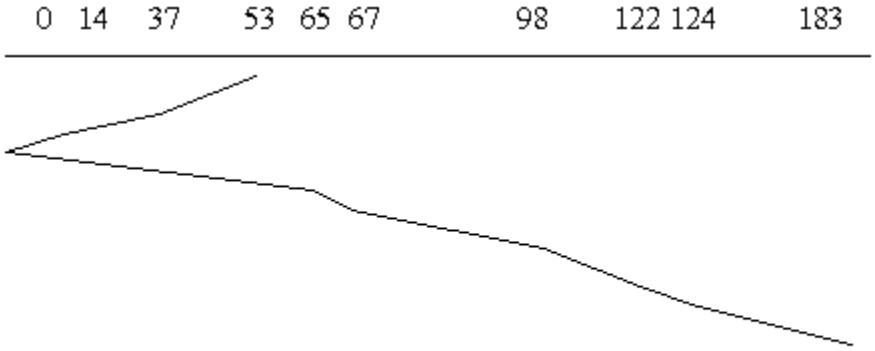
Hình 9.1.2.2-1. Lập lịch SSTF

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 :



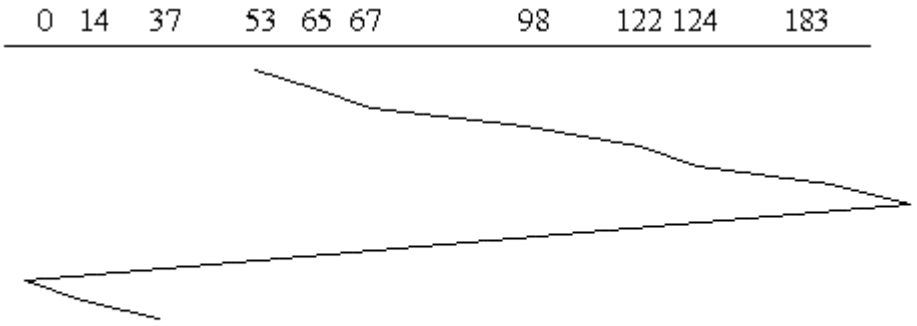
Hình 12.3 Phương pháp SCAN

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 :

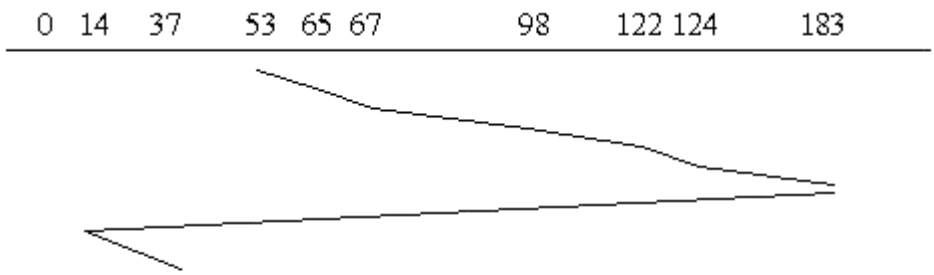


Hình 12.4 Phương pháp C-SCAN

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.

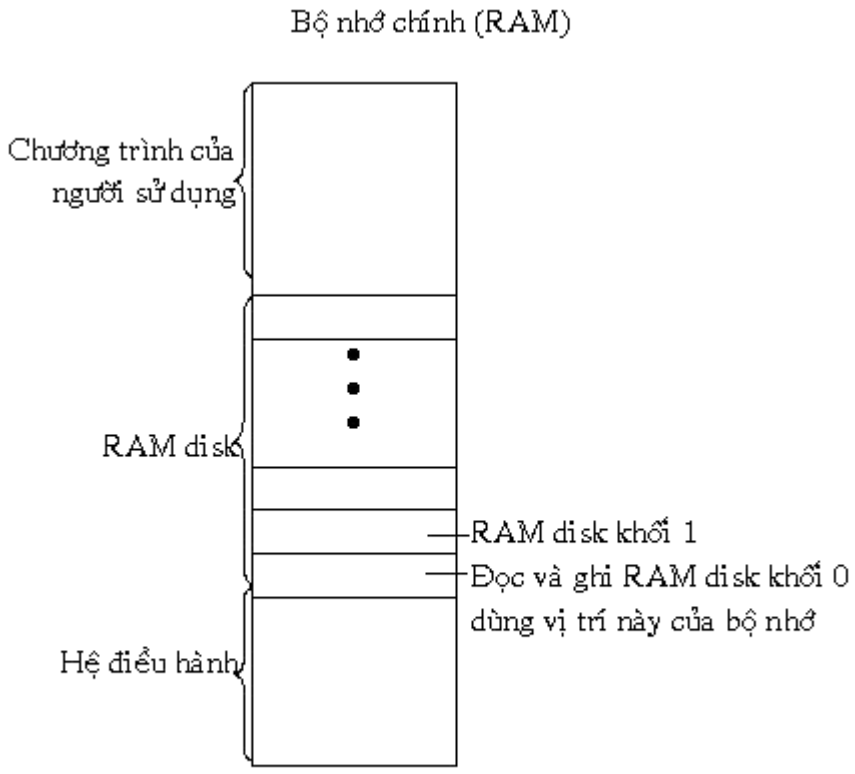
9.1.3. Quản lý lỗi

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

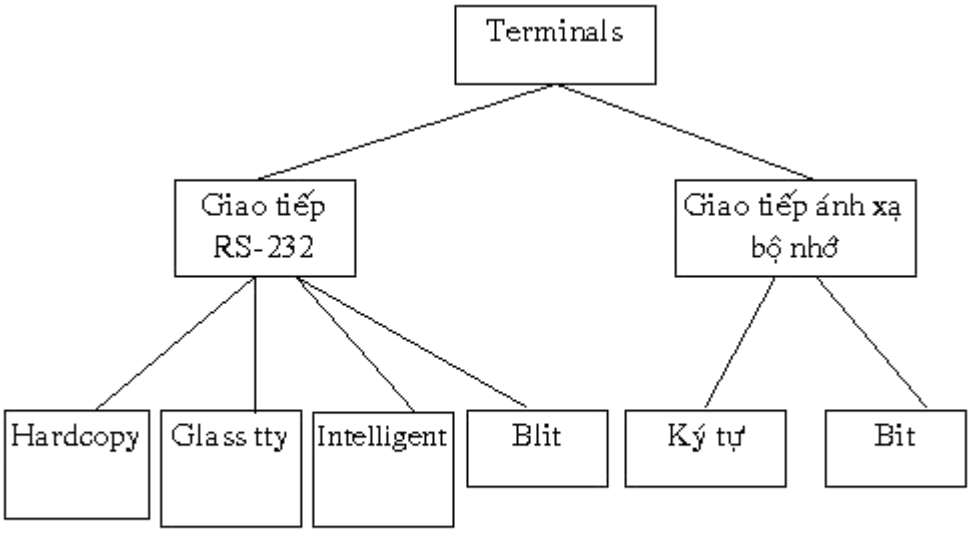
Như vậy sau bốn lần thứ tự các sector đọc được vẫn là từ 1 đến 17

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

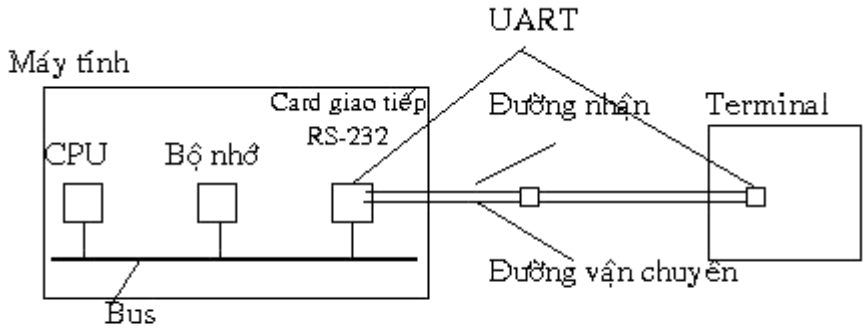
Dưới quan điểm của hệ điều hành, terminal được chia làm hai loại lớn dựa vào cách liên lạc với hệ điều hành. Loại thứ nhất bao gồm những loại terminal giao tiếp theo chuẩn RS-232. Loại thứ hai là những terminal dùng ánh xạ bộ nhớ. Mỗi loại được chia làm nhiều loại nhỏ như hình sau :



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

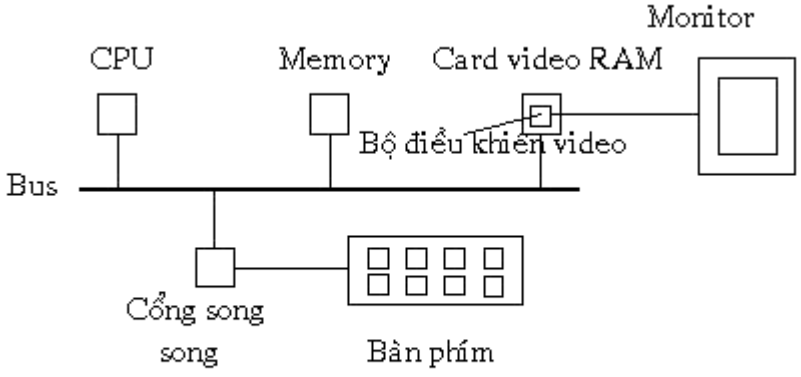
Hình 9.2.1-2. RS-232

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

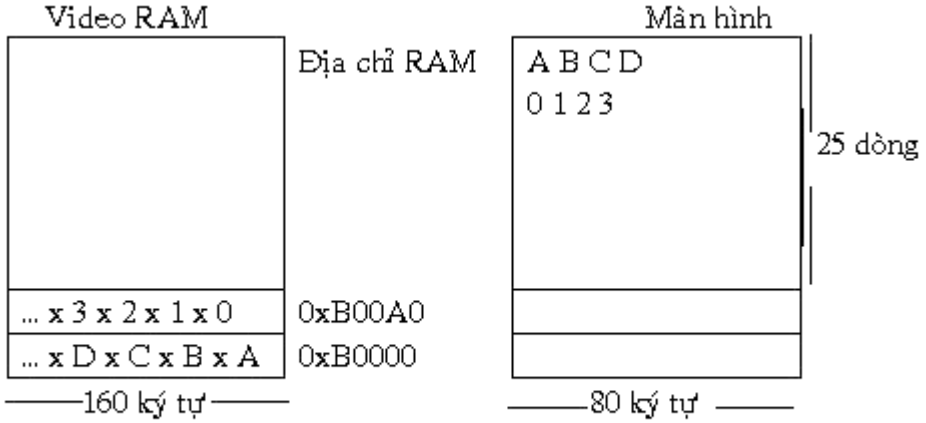
Hình 9.2.2-1. Terminal ánh xạ bộ nhớ

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

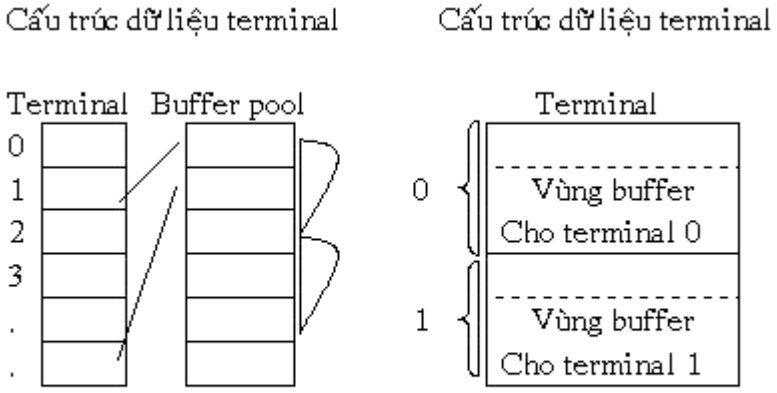
Khi CPU ghi một ký tự vào video RAM, nó xuất hiện trên màn hình theo mỗi lần hiển thị (1/50 giây cho mono, 1/60 cho màu). CPU có thể nạp 4K ảnh màn hình đã được tính trước vào video RAM trong vài phần triệu giây. Với tốc độ 9600 bps, ghi 2000 ký tự vào terminal RS-232 mất khoảng 2083 phần triệu giây.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

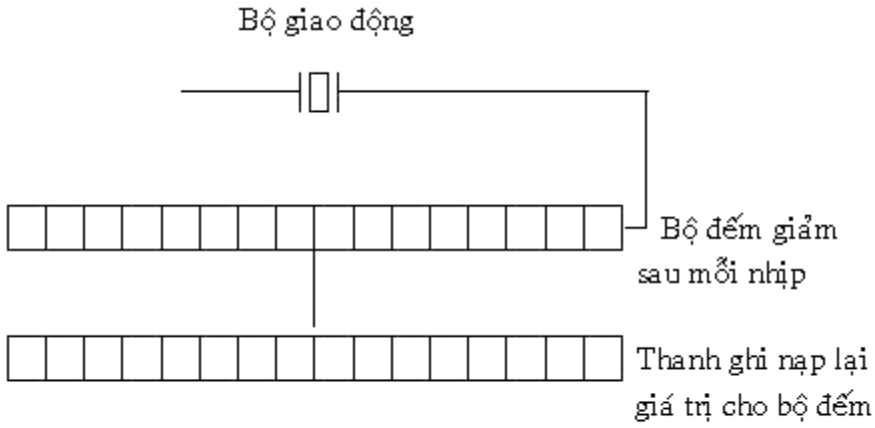
9.3. CÀI ĐẶT ĐỒNG HỒ

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.

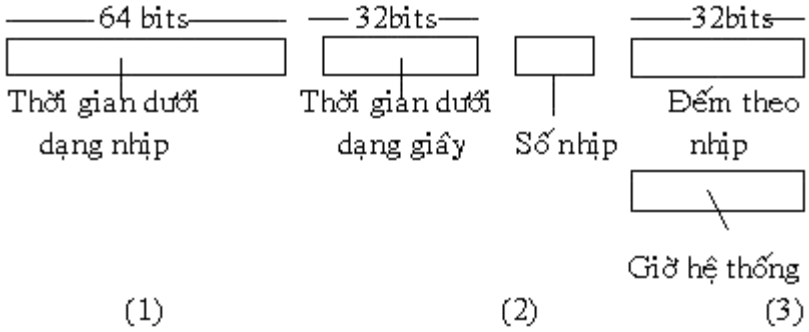
Ưu điểm của đồng hồ có thể lập trình là ngắt định kỳ được điều khiển bởi phần mềm. Nếu sử dụng tin thể thạch anh có tần số 1 MHz, bộ đếm sẽ có nhịp là mỗi micro giây. Với thanh ghi 16 bit, ngắt có thể được lập trình để xảy ra trong khoảng từ 1 đến 65535 msec.

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.

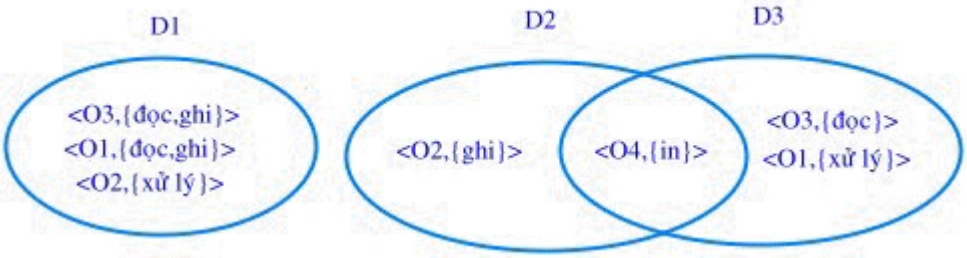
Mỗi tiến trình trong hệ thống đều hoạt động trong một miền bảo vệ *(protection domain)* nào đó. Một miền bảo vệ sẽ xác định các tài nguyên (đối tượng) mà những tiến trình hoạt động trong miền bảo vệ này có thể sử dụng, và các thao tác hợp lệ các tiến trình này có thể thực hiện trên những tài nguyên đó.

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 ghi		đọc Ghi	

Hình 10.3-1. Ma trận quyền truy xuất

Cơ chế bảo vệ được cung cấp khi ma trận quyền truy xuất được cài đặt (với đầy đủ các thuộc tính ngữ nghĩa đả mô tả trên lý thuyết), lúc này người sử dụng có thể áp dụng các chiến lược bảo vệ bằng cách đặc tả nội dung các phần tử tương ứng trong ma trận _ xác định các quyền truy xuất ứng với từng miền bảo vệ , và cuối cùng, hệ điều hành sẽ quyết định cho phép tiến trình hoạt động trong miền bảo vệ thích hợp.
Ma trận quyền truy xuất cũng cung cấp một cơ chế thích hợp để định nghĩa và thực hiện một sự kiểm soát nghiêm nhặt cho cả phương thức liên kết tĩnh và động các tiến trình với các miền bảo vệ :
Có thể kiểm soát việc chuyển đổi giữa các miền bảo vệ nếu quan niệm miền bảo vệ cũng là một đối tượng trong hệ thống, và bổ sung các cột mô tả cho nó trong ma trận quyền truy xuất.
Khi đó tiến trình được phép chuyển từ miền bảo vệ **Di** sang miền bảo vệ **Dj** nếu phần tử **access**(i,j) chứa đựng quyền « chuyển » (switch).

object domain	**F_1**	**F_2**	**F_3**	**Máy in**	**D_1**	**D_2**	**D_3**	**D_4**
D_1	đọc		đọc			chuyển		
D_2				in			chuyển	chuyển
D_3		đọc	xử lý					
D_4	đọc ghi		đọc ghi		chuyển			

Hình 10.3-2. Ma trận quyền truy xuất với domain là một đối tượng

Có thể kiểm soát việc sửa đổi nội dung ma trận (thay đổi các quyền truy xuất trong một miền bảo vệ) nếu quan niệm bản thân ma trận cũng là một đối tượng.

Các thao tác sửa đổi nội dung ma trận được phép thực hiện bao gồm : sao chép quyền (copy), chuyển quyền (transfer), quyền sở hữu (owner), và quyền kiểm soát (control)

- ***Copy***: nếu một quyền truy xuất *R* trong *access[i,j]* được đánh dấu là *R** thì có thể sao chép nó sang một phần tử *access[k,j]* khác (mở rộng quyền truy xuất *R* trên cùng đối tượng *Oj* nhưng trong miền bảo vệ *Dk*).
- ***Transfer*** : nếu một quyền truy xuất *R* trong *access[i,j]* được đánh dấu là *R+* thì có thể chuyển nó sang một phần tử *access[k,j]* khác (chuyển quyền truy xuất *R+* trên đối tượng *Oj* sang miền bảo vệ *Dk*).
- ***Owner*** : nếu *access[i,j]* chứa quyền truy xuất *owner* thì tiến trình hoạt động trong miền bảo vệ *Di* có thể thêm hoặc xóa các quyền truy xuất trong bất kỳ phần tử nào trên cột *j* (có quyền thêm hay bớt các quyền truy xuất trên đối tượng *Oj* trong những miền bảo vệ khác).
- ***Control*** : nếu *access[i,j]* chứa quyền truy xuất *control* thì tiến trình hoạt động trong miền bảo vệ *Di* có thể xóa bất kỳ quyền truy xuất nào trong các phần tử trên dòng *j* (có quyền bỏ bớt các quyền truy xuất trong miền bảo vệ *Dj*).

object domain	**F_1**	**F_2**	**F_3**
D_1	xử lý		ghi+
D_2	xử lý	đọc*	xử lý
D_3	xử lý		

(a)

object domain	**F_1**	**F_2**	**F_3**
D_1	xử lý		
D_2	xử lý	đọc*	xử lý
D_3	xử lý	đọc	ghi+

(b)

Hình 10.3-3. Ma trận quyền truy xuất với quyền *copy* , *transfer* (a) trước, (b) sau cập nhật

object domain	**F_1**	**F_2**	**F_3**
D_1	owner xử lý		Ghi
D_2		đọc* owner	đọc* owner ghi*

D_3	xử lý		

(a)

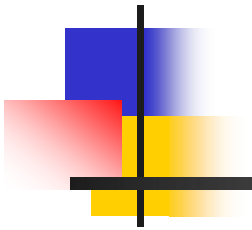
object domain	**F_1**	**F_2**	**F_3**
D_1	owner xử lý		
D_2		owner đọc* ghi*	đọc* owner ghi*
D_3		ghi	

(b)

Hình 10.3-4. Ma trận quyền truy xuất với quyền *owner* (a) trước, (b) sau cập nhật

object domain	**F_1**	**F_2**	**F_3**	**Máy in**	**D_1**	**D_2**	**D_3**	**D_4**
D_1	đọc		đọc			chuyển		
D_2				in			chuyển	control chuyển
D_3		đọc	xử lý					
D_4	ghi		Ghi		chuyển			

Hình 10.3-5. Ma trận quyền truy xuất đã sửa đổi nội dung so với H5.3 nhờ quyền *control*

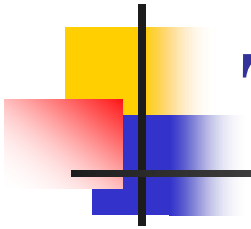


HỆ ĐIỀU HÀNH NÂNG CAO

Trường đại học Khoa học tự nhiên

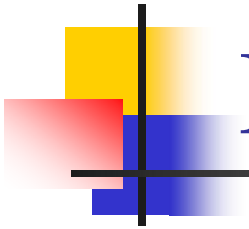
Khoa Công nghệ Thông tin

Trần Hạnh Nhi



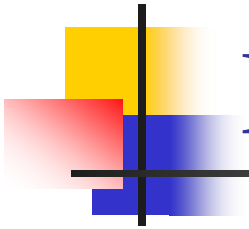
Tổ chức

- **Phụ trách Lý thuyết :**
  - **Trần Hạnh Nhi**
- **Phụ trách thực hành:**
  - **Phạm Nguyễn Anh Huy**
  - **Trần Anh Tuấn**
  - **Lê Thụy Anh**
  - **Đinh Bá Tiến**
- **Trang web của môn học :**



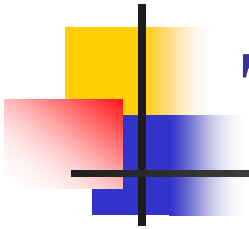
Mục tiêu

- **Kết quả mong đợi về lý thuyết :**
  - **Hiểu được cách thức Hệ điều hành làm việc**
  - **Nắm được các nguyên lý thiết kế Hệ điều hành**
  - **Biết được một số cơ chế, chiến lược cơ bản để giải quyết các nhiệm vụ của Hệ điều hành**
- **Kết quả cần đạt được về thực hành**
  - **Vận dụng được các kiến thức lý thuyết để cài đặt giả lặp một số module của Hệ điều hành**
  - **Sử dụng được các cơ chế hỗ trợ của một Hệ điều hành cụ thể (Windows NT) để giải quyết các bài toán cơ bản.**



Kiến thức yêu cầu

- **Kiến trúc Máy tính**
- **Hệ điều hành cơ bản**
- **Lập trình C/C++**

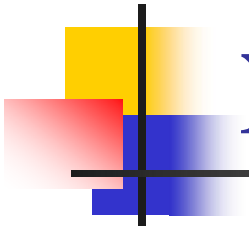


Tính điểm

- 70% Lý thuyết + 30% Thực hành
- Lý thuyết :
  - 1 bài thi cuối khoá (không tham khảo tài liệu)
  - Mỗi sinh viên làm bài độc lập
- Thực hành: 2 bài tập lớn
  - Thời hạn và cách thức nộp bài sẽ do giáo viên phụ trách thực hành qui định
  - Mỗi nhóm thực hành gồm 2 sinh viên
- Bắt buộc có nộp bài thực hành mới được thi lý thuyết

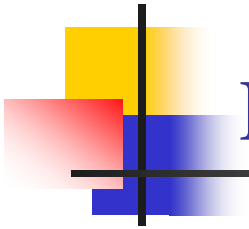
Tài liệu tham khảo

- ***Trần Hạnh Nhi*** **: Giáo trình Hệ điều hành Nâng cao**
- ***A.Silberschatz & P/Galvin*** **: OS concepts (5e)**
  - **Slides :**
- ***W. Stallings*** **: Operating Systems**
- ***A.Tanenbaum et al*** **: OS Design and Implementation**
  - **Minix :**
- ***R.Finkel*****:: An OS vade mecum**
  - **Book online :**
- ***Jeffrey Richter*** **: Advanced Windows**
- ***Tiến Huy- Đan Thư- Hạnh Nhi*** **: Kỹ thuật lập trình trên Windows NT**



Nội dung

- **Chương 1 : Tổ chức Hệ điều hành**
- **Chương 2 : Quản lý tiến trình**
- **Chương 3 : Liên lạc giữa các tiến trình**
- **Chương 4 : Quản lý bộ nhớ chính**
- **Chương 5 : An toàn hệ thống**



Bài giảng 1 : Giới thiệu

- Tại sao phải tìm hiểu về Hệ điều hành ?
- Hệ điều hành là gì ?
  - Vai trò trong hệ thống ?
  - Chức năng ?
  - Kiến trúc ?
- Các nguyên lý thiết kế Hệ điều hành

Tại sao cần tìm hiểu Hệ điều hành ?

- **Để phá vỡ sự "bí ẩn" của hệ thống :**
  - **Tại sao máy tính có thể "biết" được nội dung đĩa ?**
  - **Tại sao có thể vừa soạn thảo, vừa nghe nhạc trên cùng 1 máy tính (có 1 CPU ?)**
  - **Tại sao 1 ứng dụng kích thước 1 M có thể hoạt động trên Windows mà bị báo "Not enough memory" trên DOS ?**
- **Để khai thác tốt hơn môi trường làm việc :**
  - **Lập trình trên môi trường đa nhiệm (multitask), đa xử lý(multiprocessing) với các mô hình multiprocess, multithreads..**
  - **Sử dụng bộ nhớ hiệu quả**
  - **sử dụng các cơ chế Thông tin liên lạc, an toàn & bảo mật...**
- **Vì là môn học bắt buộc ☺**

Hệ điều hành, anh là ai ?

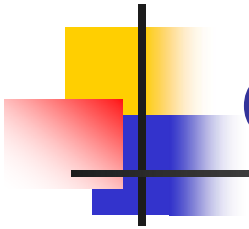
Ứng dụng

Giao diện ảo

Hệ điều hành

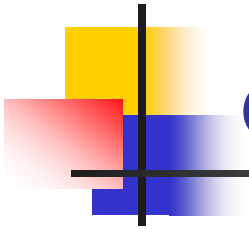
Giao diện vật lý

Phần cứng



Chức năng của Hệ điều hành

- **Quản trị tài nguyên (resource principle) :**
  - **Tài nguyên : CPU, Mem, IO; Files, ports, mailboxes...**
  - **Đối tượng sử dụng tài nguyên : Process, Thread**
  - **Nhiệm vụ : Cung cấp các giải thuật cấp phát, quản lý tài nguyên.cho các đối tượng hoạt động trong hệ thống**
  - **Mục tiêu : Cấp phát đầy đủ, công bằng R cho Ps; Sử dụng hiệu quả Rs, Nâng cao thông lượng Ps...**
- **Trừu tượng hoá hệ thống (beautification principle)**
  - **Nhiệm vụ : Cung cấp các giải thuật để che dấu chi tiết phần cứng, tạo 1 môi trường dễ làm việc hơn (hope) cho user**
  - **Mục tiêu : tạo môi trường an toàn, tạo sự trừu tượng hoá, độc lập thiết bị**
  - **Ví dụ : device driver**



Các thành phần

Quản lý bộ nhớ phụ

Quản lý tiến trình

Quản lý nhập xuất

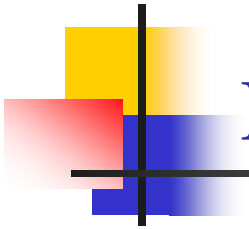
Hệ thống tập tin

Quản lý bộ nhớ chính

Hệ thống bảo vệ

Bộ thông dịch lệnh

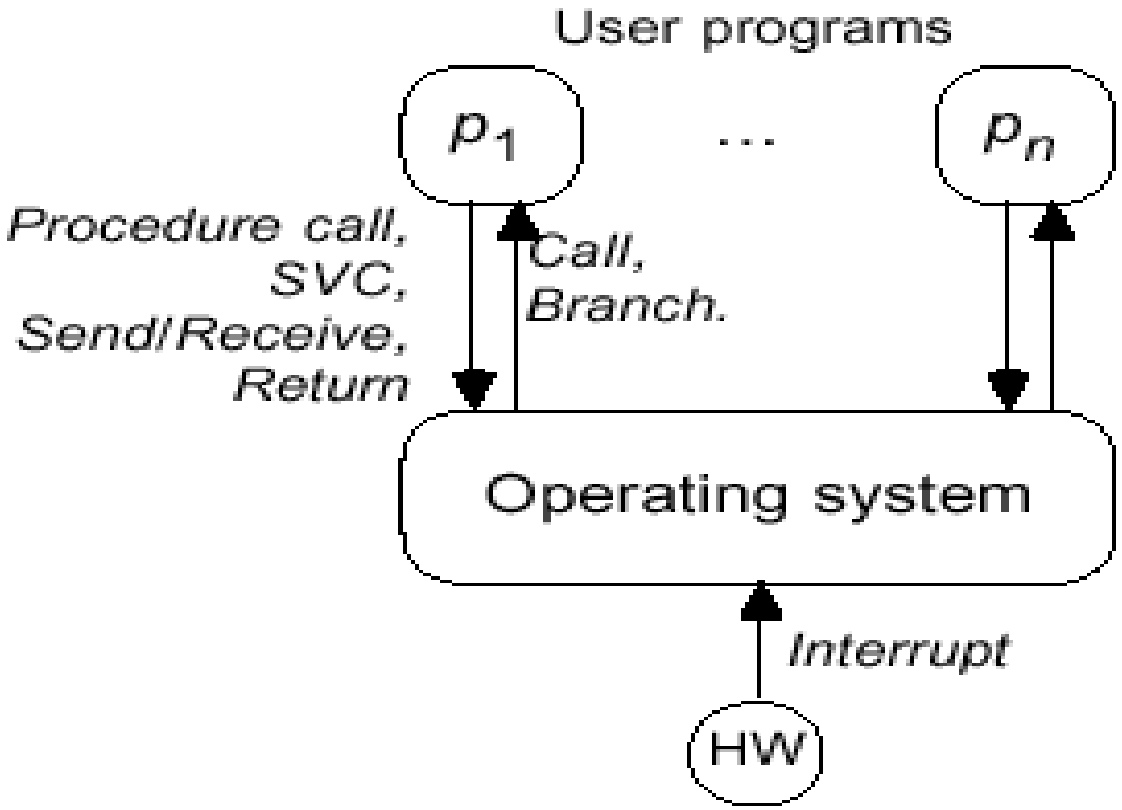
Giao tiếp mạng

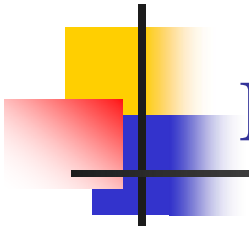


Kiến trúc Hệ điều hành

- Đơn giản (Monolithic)
- Hạt nhân (Kernel)
- Phân lớp (Layered)
- Máy ảo (Virtual Machine)
- Hướng đối tượng (OOOS)
- Exokernel

Monolithic

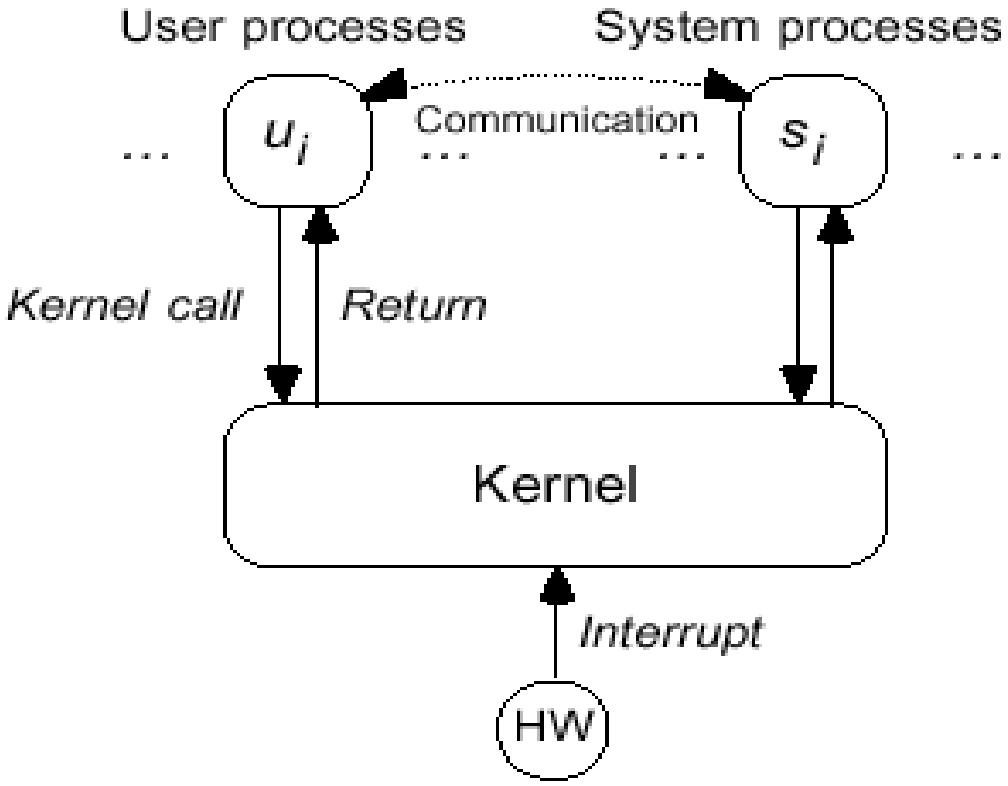


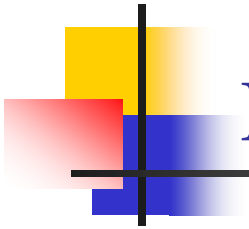


Monolithic

- OS = Thư viện tiện ích
- Có thể tổ chức thành nhiều module : CPU scheduling, Mem Management, Device management...nhưng chỉ có 1 trong những module này hoạt động tại một thời điểm
- Đơn nhiệm
- Quyền điều khiển được chuyển đổi thông qua lời gọi hàm
- Khi tầm vóc phát triển hệ thống trở nên thiếu tin cậy.
- Ví dụ : MS-DOS, Ultrix (mature Unix)

Kernel

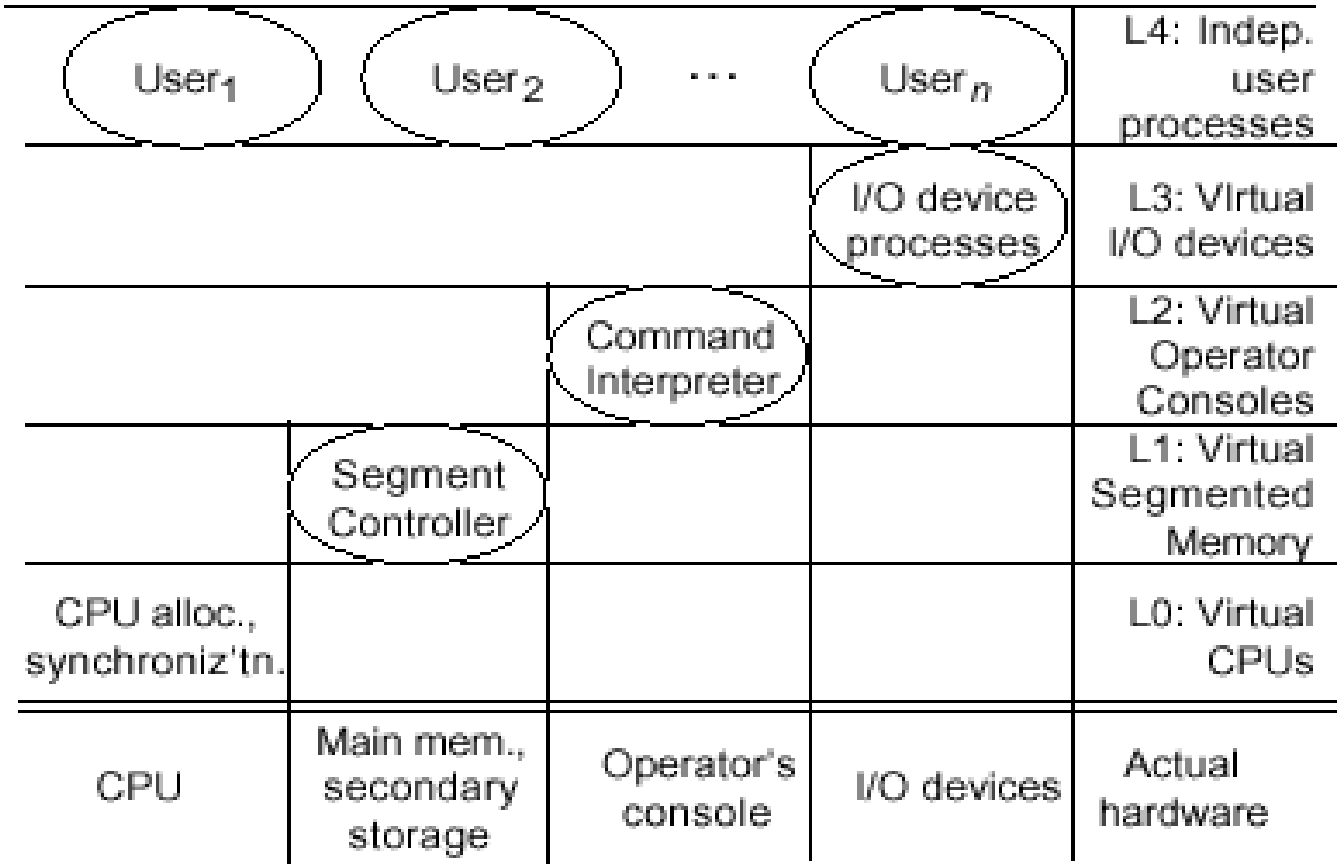


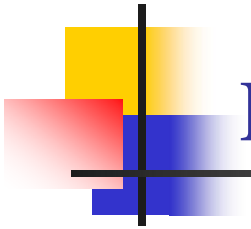


Kernel

- OS = Kernel + System processes
- Kernel được bảo vệ
- Đa nhiệm
- Kernel chịu trách nhiệm phân chia thời gian sử dụng CPU, Giao tiếp giữa các tiến trình
- Chỉ có 2 mức kernel/non-kernel =>kernel lớn, thiếu tin cậy như trước
- Định nghĩa cứng các giao tiếp với ứng dụng trong kernel
- Ví dụ : Windows NT

Layered

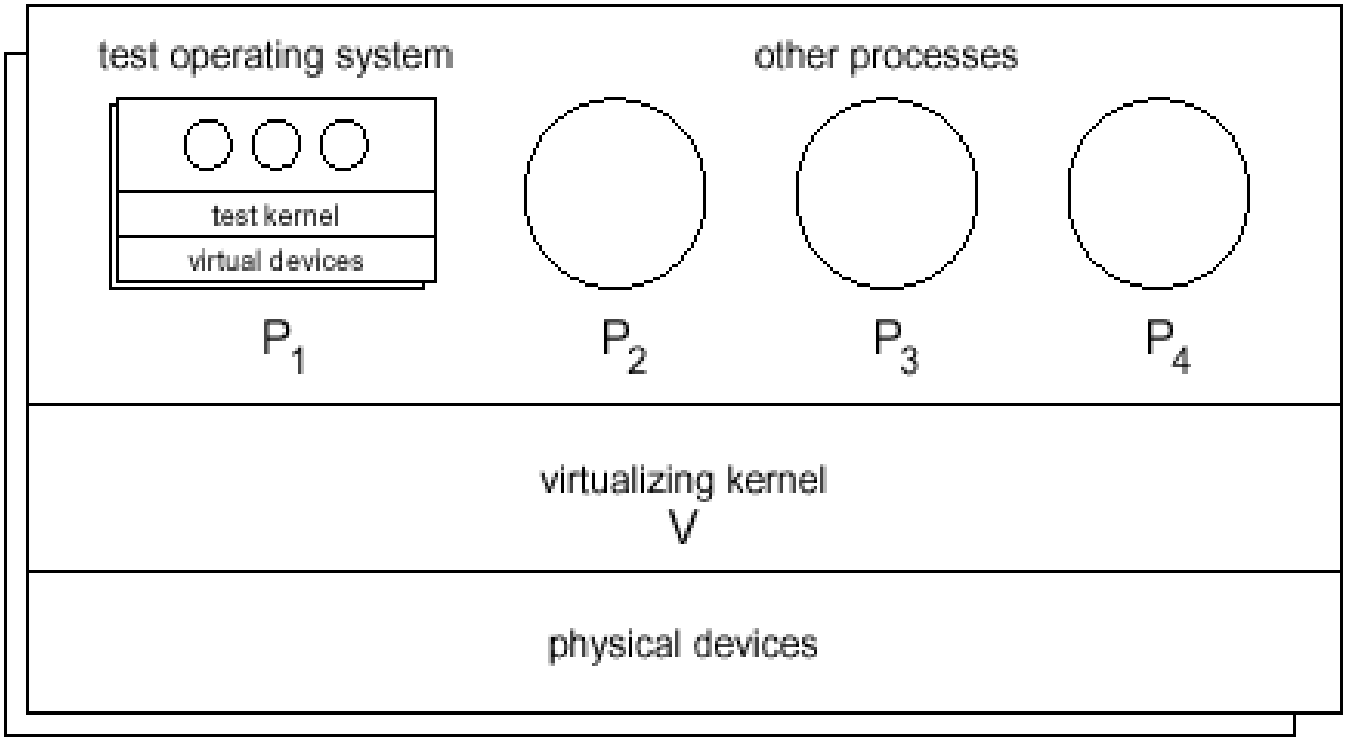


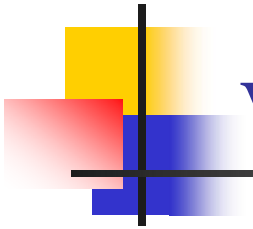


Layered

- OS = các lớp trừu tượng hoá một tác vụ quản lý
- Lớp trên được sử dụng các hàm xử lý tài nguyên thuộc tác vụ do lớp dưới cung cấp
- Khó xác định được các lớp xử lý rạch ròi, thứ tự lớp ?
  - Tạo tiến trình -> PM gọi MM
  - Bộ nhớ đầy -> MM gọi PM
- Xếp lớp theo hàm xử lý , thay vì tác vụ
  - Seg management- P scheduling- Seg creation- P creation

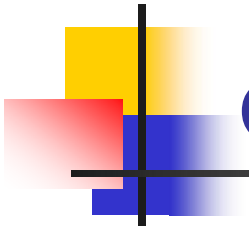
Virtual Machine





Virtual Machine

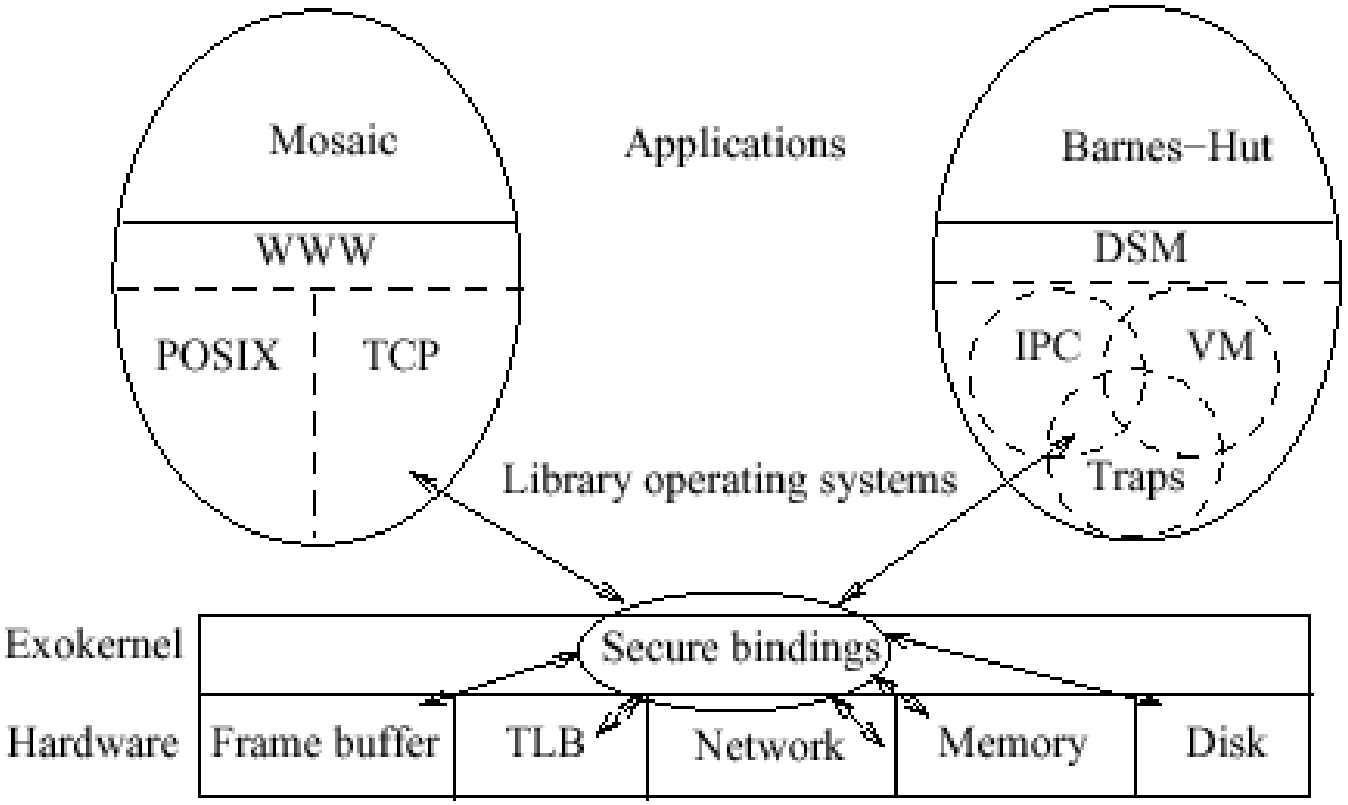
- OS = Virtualizing kernel + virtual machines
- Virtual machine = physical hardware
- Virtualizing kernel tạo ra nhiều VM trên 1 máy tính.
- Process interface = hardware interface
- Ưu điểm :
  - Môi trường thuận lợi cho sự tương thích (compatibility)
  - Tăng tính an toàn hệ thống do cung cấp các VM độc lập.
  - Dể phát triển các HDH đơn nhiệm cho mỗi VM
- Khuyết điểm:
  - Phức tạp cho việc giả lặp (transput, add translation...)
- Ví dụ : CMS(conversational Monitor System) trên VM/370 (hỗ trợ hardware)

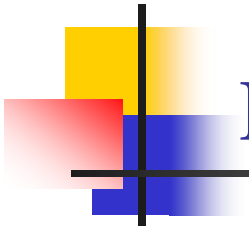


OOOS

- OS = tập các đối tượng
- Tiến trình, tập tin, hàm, khối nhớ...
- Một hàm xử lý (kernel/non-kernel mode) thao tác trên một tập các đối tượng.
- Che dấu thông tin
- Ví dụ :CAP, StarOS, iMAX432

Exokernel



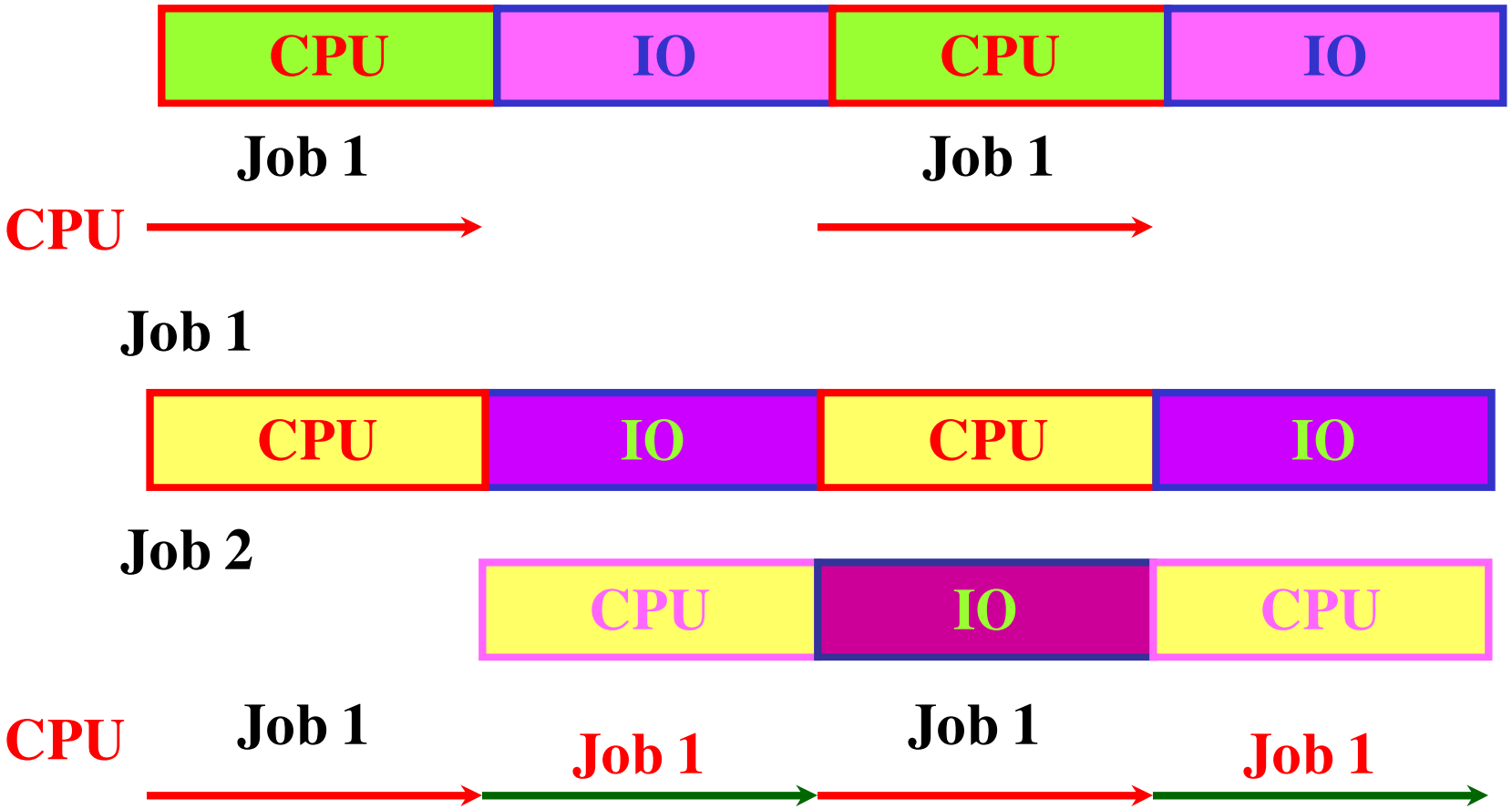


Exokernel

- **Hướng đến một HDH linh đông trong giao tiếp với ứng dụng, cho phép ứng dụng chuyên biệt hoá hệ điều hành theo nhu cầu đặc thù một cách dễ dàng**
- **OS = Exokernel + Library OS**
- **Ứng dụng có thể phát triển các mô hình tổ chức VM, IPC theo nhu cầu riêng**
- **Ví dụ : ý tưởng của project do Dawson R Engler et al phát triển tại MIT**

Bài 2 : CÁC MÔ HÌNH XỬ LÝ ĐỒNG HÀNH

Xử lý đồng hành, để tăng hiệu suất sử dụng CPU



Xử lý đồng hành, để tăng tốc độ xử lý

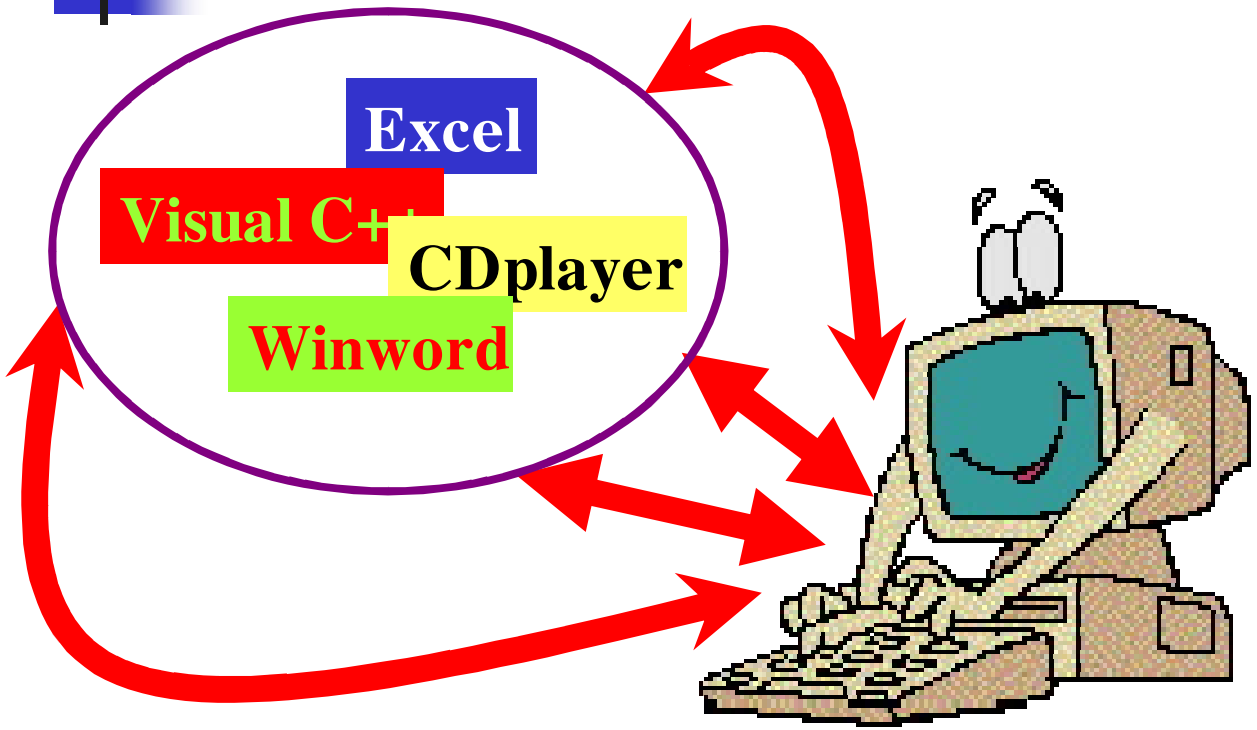
- **Job : kq = a*b + c*d;**
- **Xử lý tuần tự :**

```
kq1 = a * b;
kq2 = c* d;
kq = kq1 + kq2;
```

- **Xử lý đồng hành :**

kq1 = a*b;	kq2 = c*d;
kq = kq1 + kq2;	

Xử lý đồng hành, những khó khăn ?



- Tài nguyên giới hạn, ứng dụng "vô hạn"
- Nhiều hoạt động đan xen

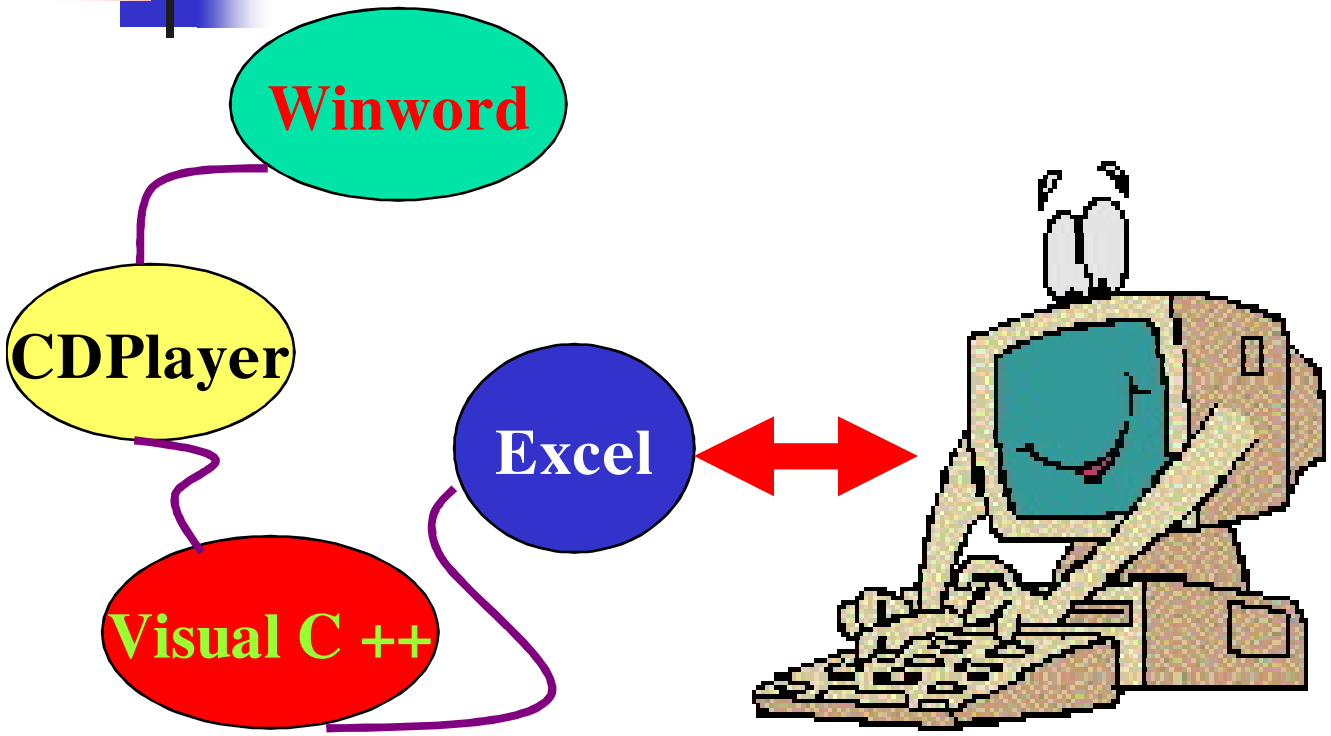
??? Phân chia tài nguyên ?

??? Chia sẻ tài nguyên ?

??? Bảo vệ?

HĐH : " Giải quyết nhiều công việc đồng thời, đâu có dễ !

Giải pháp

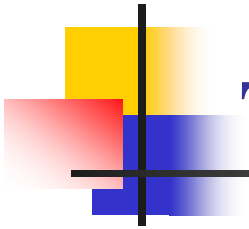


HĐH : " Ai cũng có phần khi đến lượt mà ! "

-"Chia để trị", cô lập các hoạt động.

- Mỗi thời điểm chỉ giải quyết 1 yêu cầu.

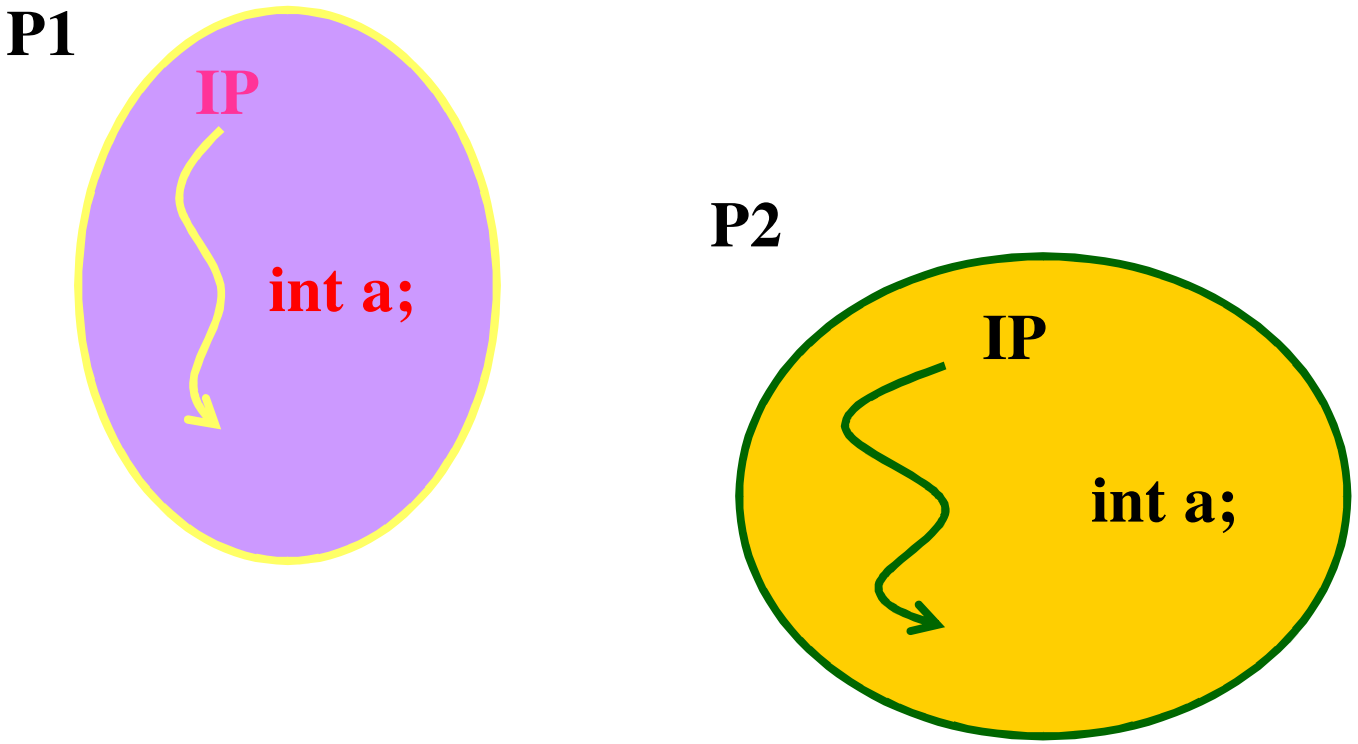
- Ảo hoá tài nguyên : biến ít thành nhiều



Thuật ngữ

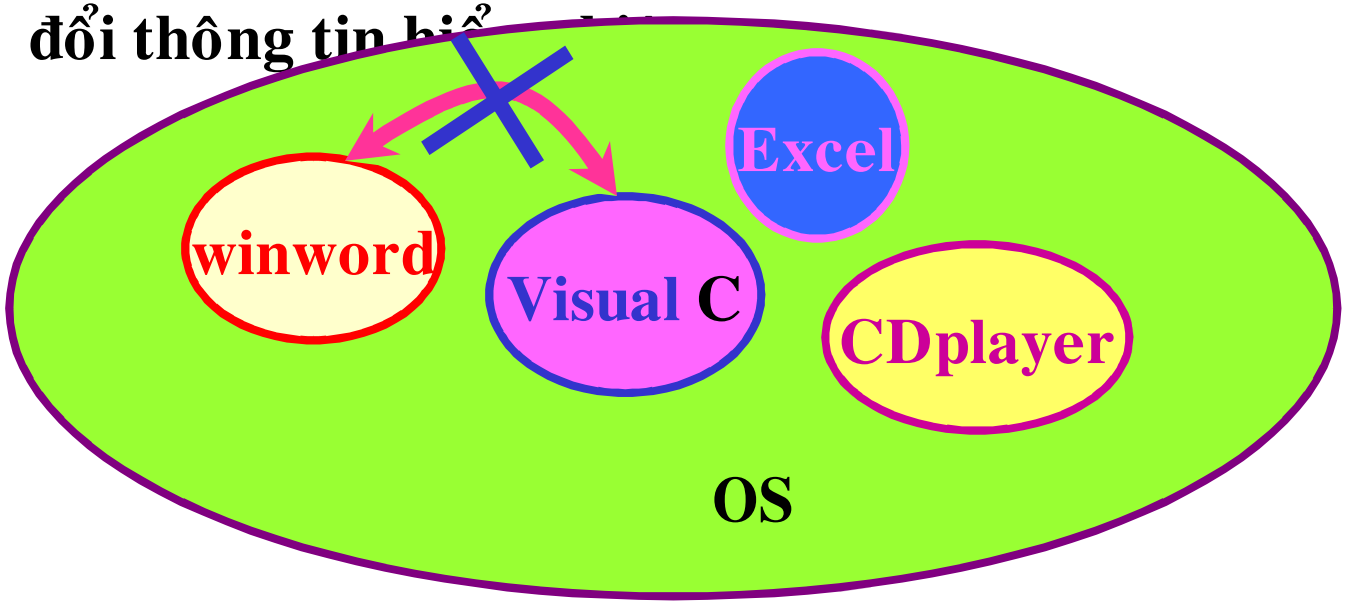
- **Concurrency (đồng hành): mô hình xử lý nhiều tác vụ đồng thời.**
- **Multitasking (đa nhiệm) : cho phép nhiều tác vụ/ công việc được xử lý đồng thời**
- **Multiprogramming (đa chương) : cho phép nhiều chương trình được thực hiện đồng thời (trên 1 CPU)**
- **Multiprocessing (đa xử lý): nhiều bộ xử lý làm việc đồng thời**

Khái niệm tiến trình



Mô hình đa tiến trình (MultiProcesses)

- **Hệ thống là một tập các tiến trình hoạt động đồng thời**
- **Các tiến trình độc lập với nhau => không có sự trao đổi thông tin hiể**



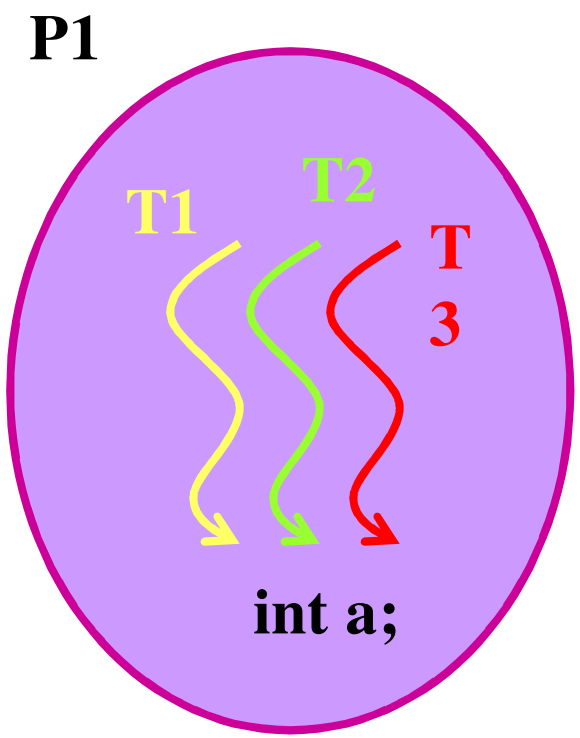
Mô hình đa tiểu trình (MultiThreads)

- **Muốn nhiều dòng xử lý đồng thời cùng chia sẻ tài nguyên (server, OS, các chương trình tính toán song song)**

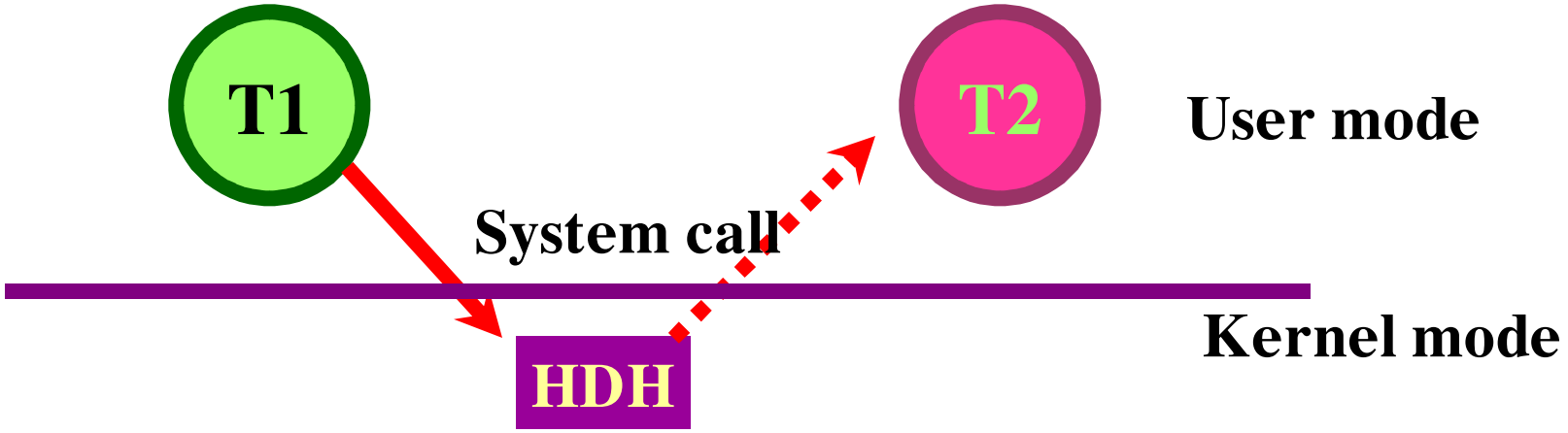
☞ **TIỂU TRÌNH (THREAD)**

Khác biệt giữa Tiểu trình & Tiến trình

- Tiểu trình : 1 dòng xử lý
- Tiến trình :
  - 1 không gian địa chỉ
  - 1 hoặc nhiều tiểu trình
- Các tiến trình là độc lập
- Các tiểu trình trong cùng 1
  tiến trình không có sự bảo vệ
  lẫn nhau (cần thiết ?).

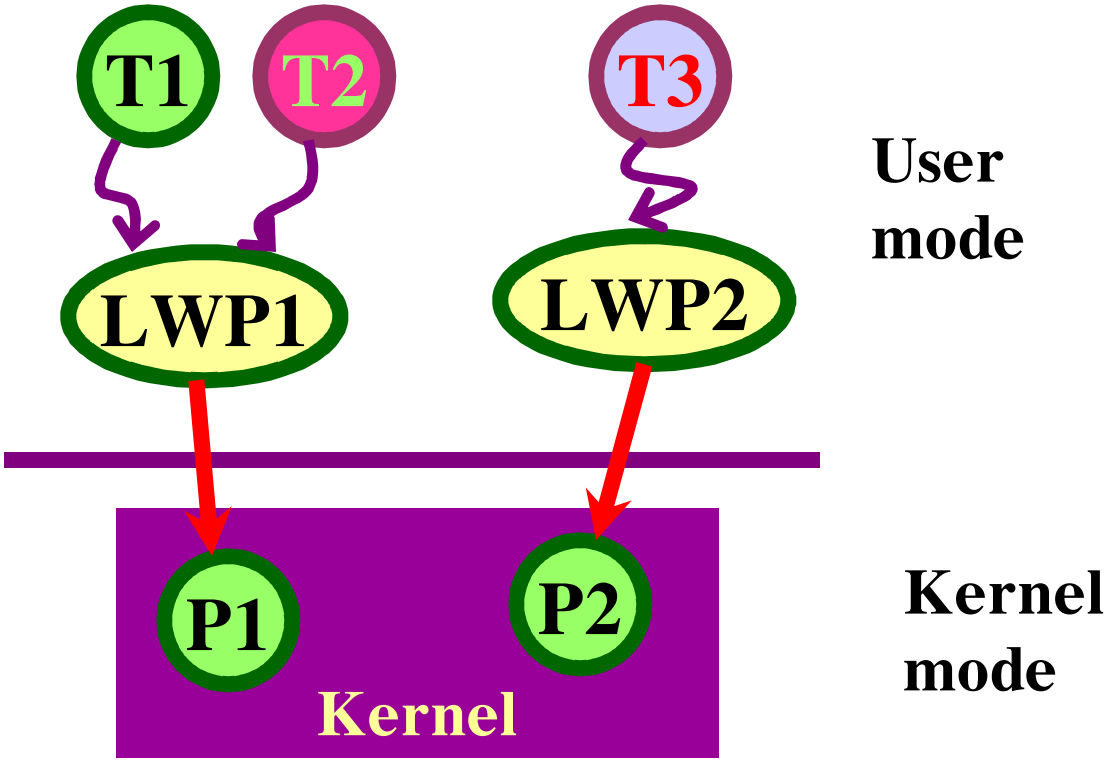


Tiểu trình hạt nhân (Kernel thread)

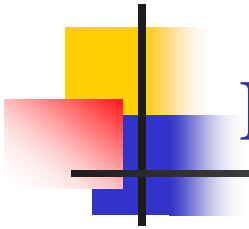


Khái niệm tiểu trình được xây dựng bên trong hạt nhân

Tiểu trình người dùng (User thread)



Khái niệm tiểu trình được hỗ trợ bởi một thư viện hoạt động trong user mode

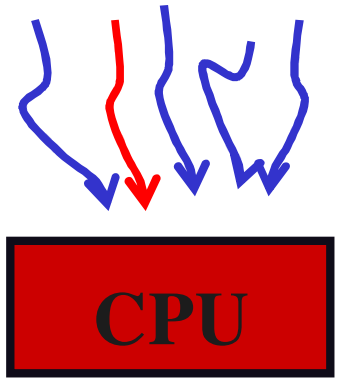


Bài 3 : QUẢN LÝ TIẾN TRÌNH

- Phân chia CPU cho các tiến trình ?
  - Tiếp cận
  - Mục tiêu ?
  - Tổ chức ?
  - Chiến lược ?
- Trạng thái tiến trình ?
- Lưu trữ thông tin tiến trình ?
- Các thao tác trên tiến trình ?
- Bảo vệ tiến trình ?
- Trao đổi thông tin giữa các tiến trình ?

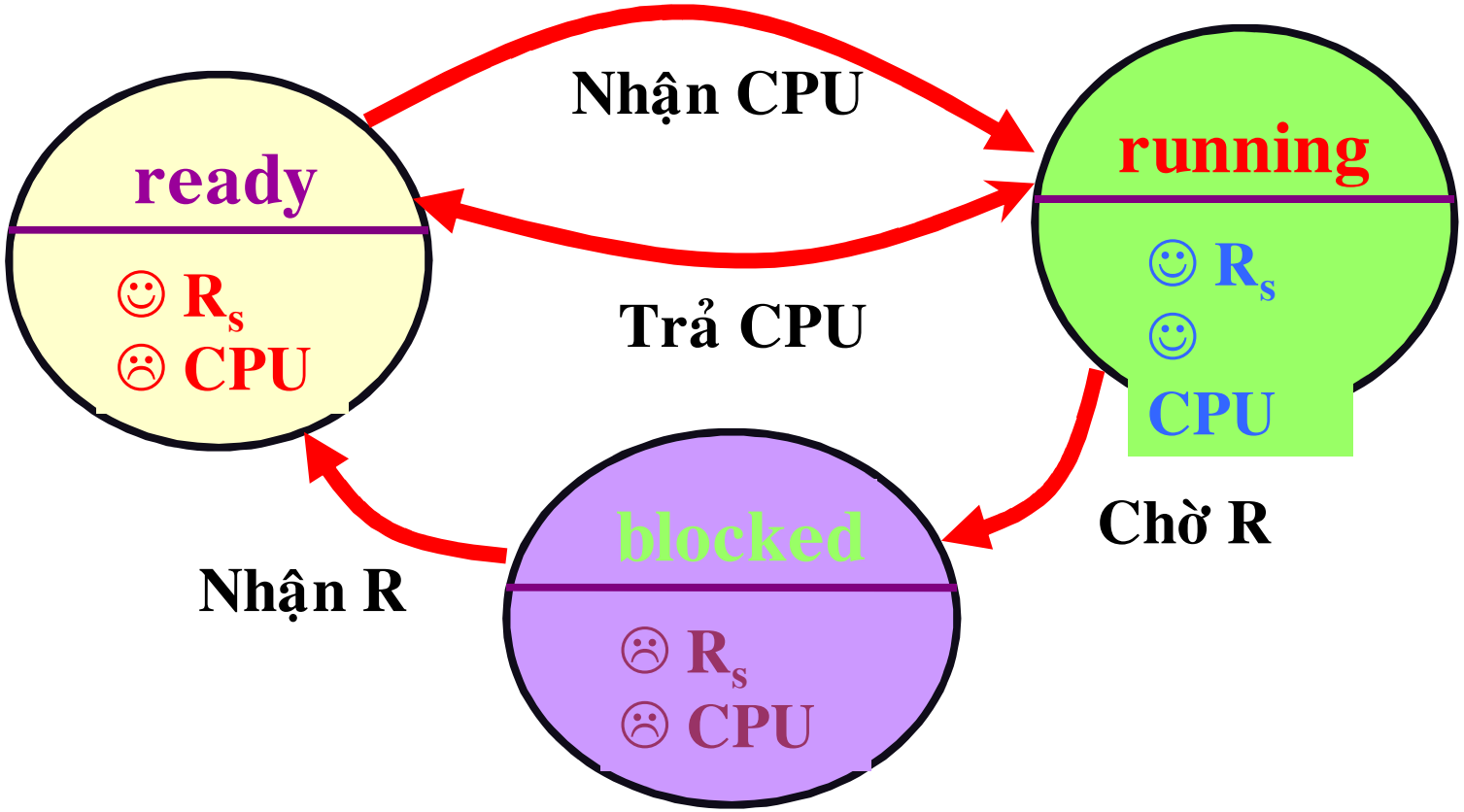
Phân chia CPU ?

- 1 CPU vật lý : làm thế nào để tạo ảo giác mỗi tiến trình sở hữu CPU riêng của mình ?
- Dispatcher luân chuyển CPU giữa các tiến trình:
  - Ngữ cảnh xử lý riêng biệt cho mỗi tiến trình (PCB)



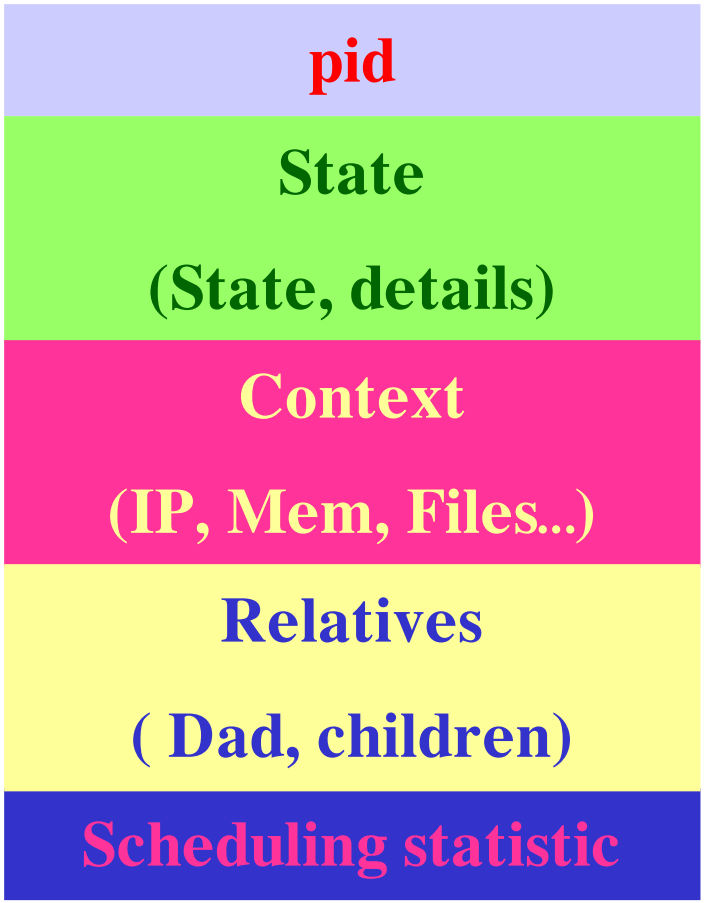

```
while(1)
{
   interrupt P_cur
   save state P_cur
   Scheduler gets P_next
   load state P_next
   jump to it
}
```

Trạng thái tiến trình ?



Khối quản lý tiến trình trong mô hình multiprocesses

Process control Block
PCB



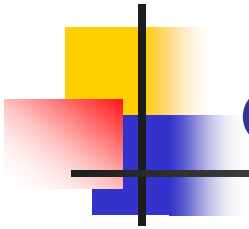
PCB và TCB trong mô hình multithreads

PCB

pid
Threads list
Context (Mem, global ressources...)
Relatives (Dad, children)
Scheduling statistic

Thread Control Block
TCB

tid
State (State, details)
Context (IP, local stack...)



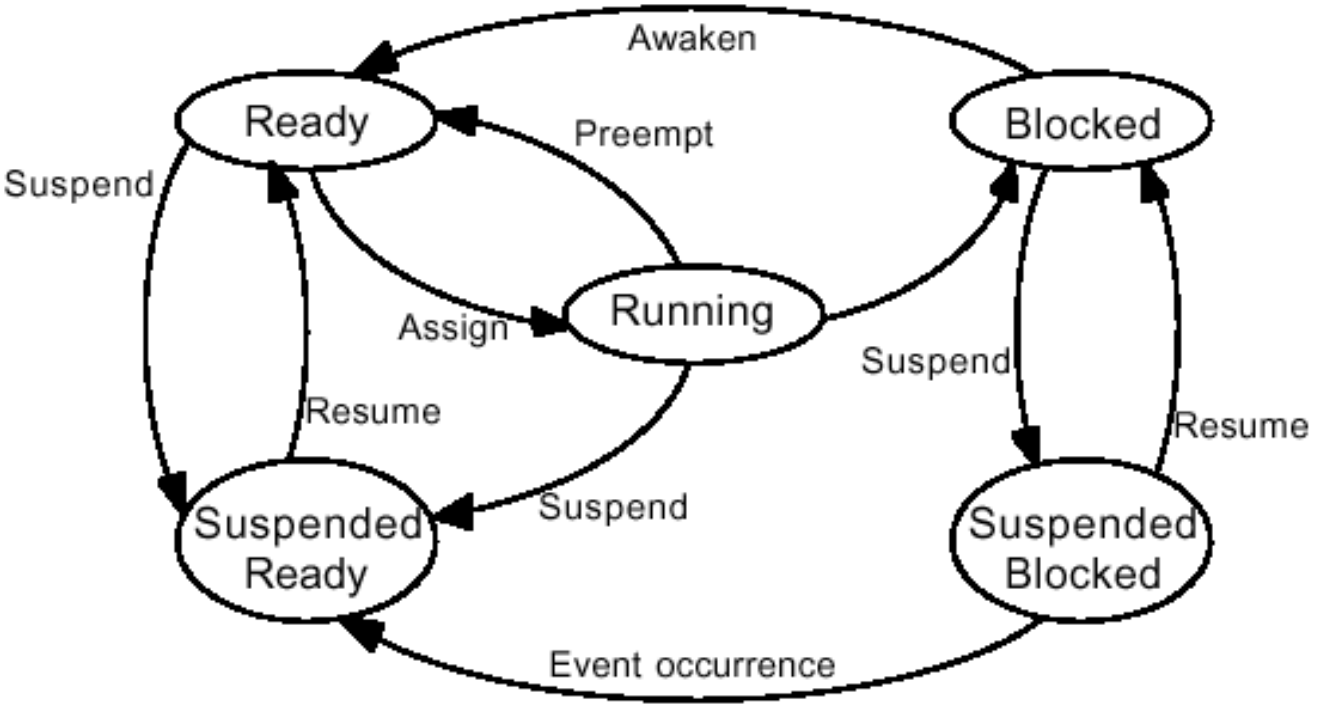
Các thao tác trên tiến trình

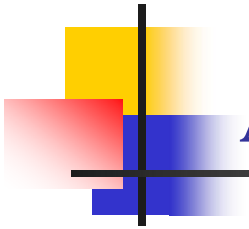
- **Tạo lập tiến trình :**
  - **Cấp phát tài nguyên cho tiến trình con ?**
  - **Hoạt động của cha và con độc lập**
- **Kết thúc tiến trình :**
  - **Thu hồi tài nguyên ?**
  - **Ép buộc kết thúc ?**
- **Thay đổi trạng thái tiến trình :**

  Assign(), Block(), Awake(), Resume(), Suspend()

Trạng thái tiến trình ?

- **Có nhu cầu Suspend &Resume :**
  - **Hệ thống quá tải**
  - **Kiểm soát hoạt động của tiến trình con**

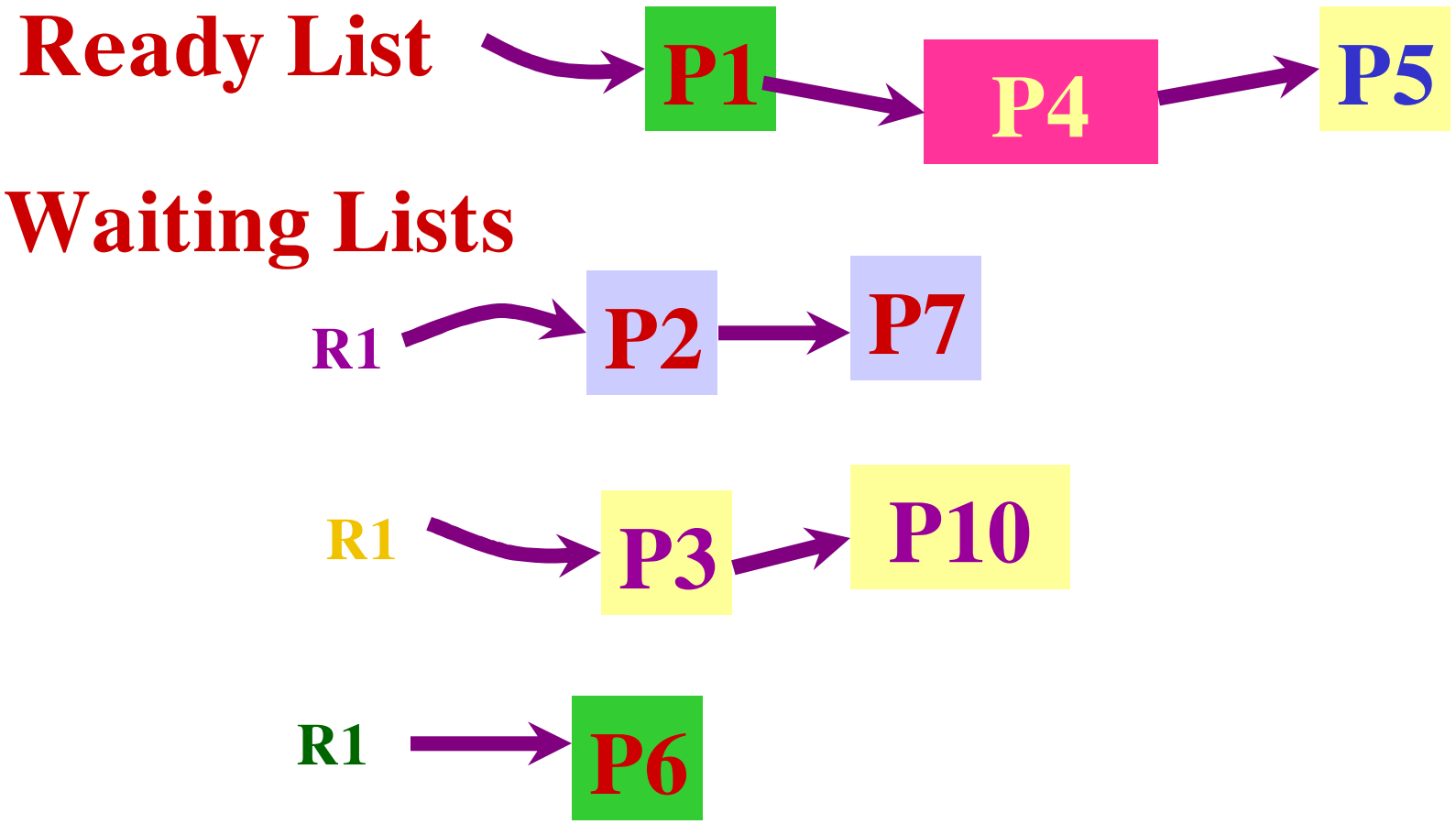


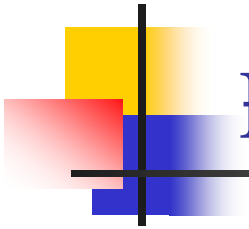


An ninh trật tự cho môi trường đa tiến trình !

- **Bảo vệ tiến trình :**
  - **Ngăn cản các tiến trình xâm phạm tài nguyên, can thiệp vàoxử lý của nhau => KGĐC riêng biệt, 2 mode xử lý**
  - **Bảo đảm quyền tiến triển xử lý cho mỗi tiến trình => công bằng trong các chiến lược phân phối tài nguyên.**
- **Trao đổi thông tin , phối hợp hoạt động ?**
  - **Nhu cầu ?**
  - **Vấn đề ? => Chương kế tiếp**
  - **Giải pháp ?**

Các danh sách tiến trình





Điều phối tiến trình

- Mục tiêu ?
- Các cấp độ điều phối
- Thời điểm ra quyết định điều phối ?
- Đánh giá chiến lược điều phối ?
- Một số chiến lược điều phối

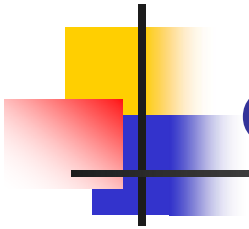
Điều phối tiến trình

SCHEDULER

chọn một tiến trình nhận cpu

DISPATCHER

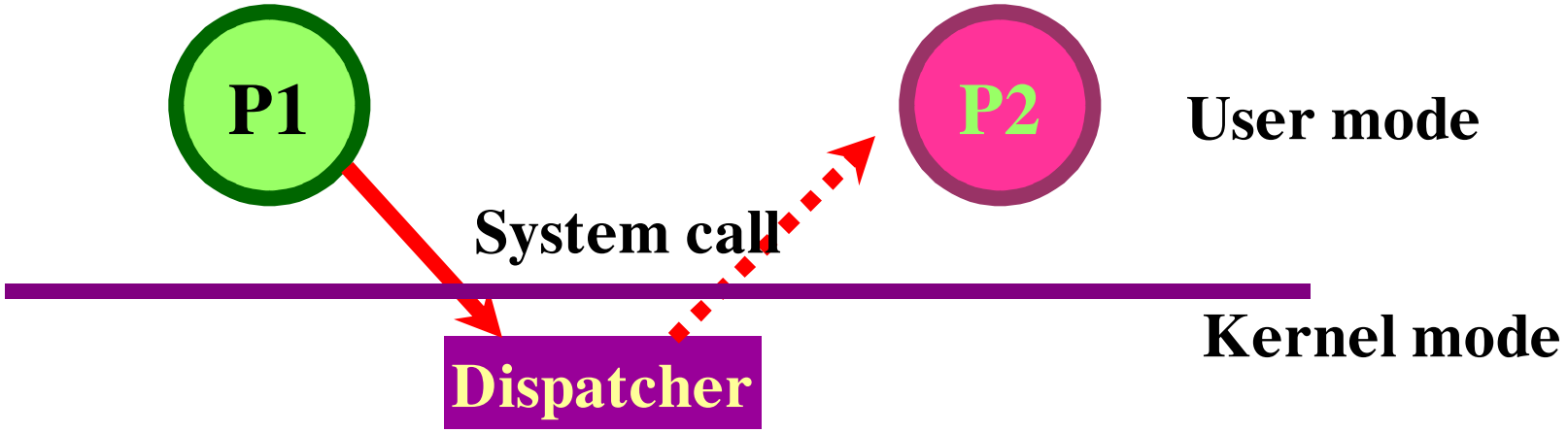
chuyển đổi ngữ cảnh



Chuyển đổi ngữ cảnh (context switching)

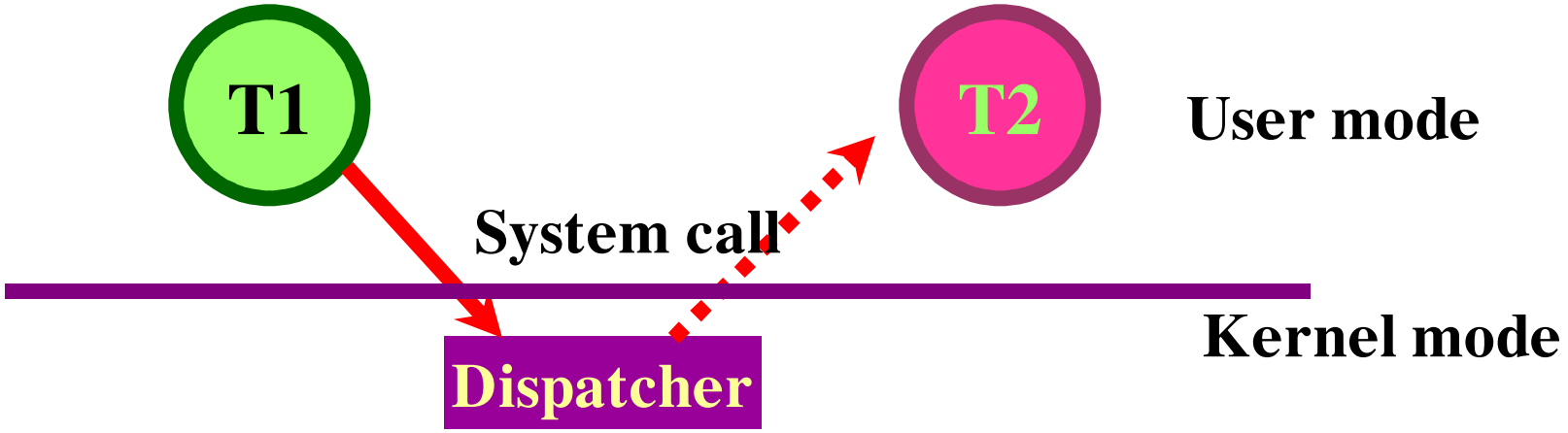
- **Kịch bản :**
  - **Lưu ngữ cảnh tiến trình hiện hành**
  - **Nạp ngữ cảnh tiến trình được chọn kế tiếp**
- **Chi tiết cụ thể phụ thuộc vào phần cứng**
  - **general-purpose & floating point registers, co-processor state...**
- **Chi phí chuyển đổi ngữ cảnh :**
  - **Giữa các tiến trình ?**
  - **Giữa các tiểu trình ?**

Chuyển đổi ngữ cảnh giữa các tiến trình



- Chuyển đổi mode xử lý
- Chuyển đổi IP và các thanh ghi khác của CPU
- Chuyển đổi không gian địa chỉ

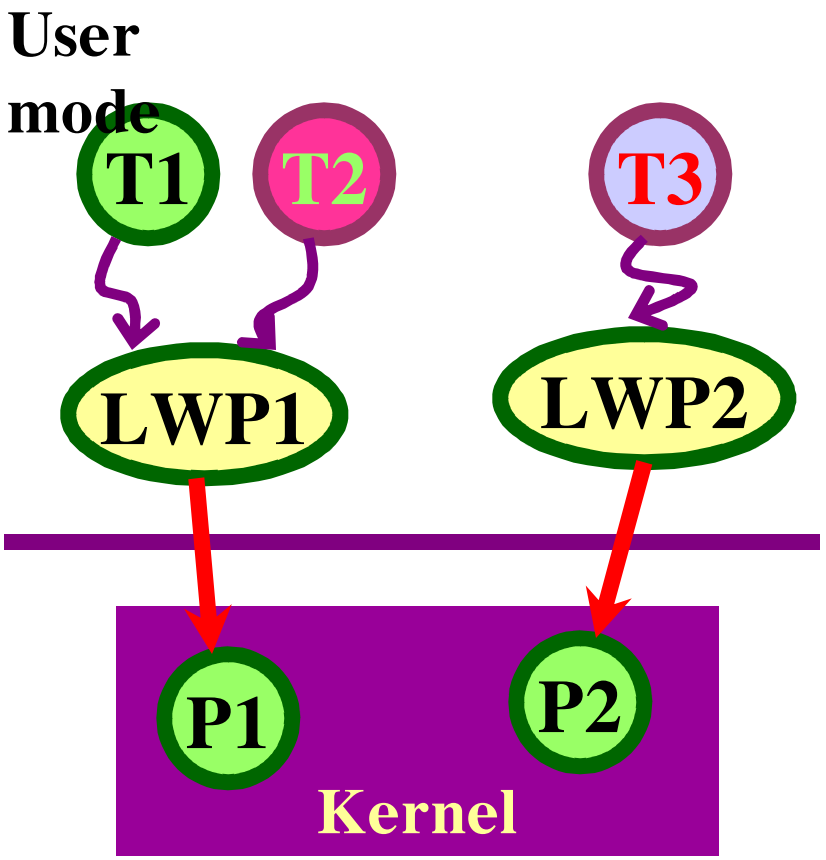
Tiểu trình hạt nhân (Kernel thread)

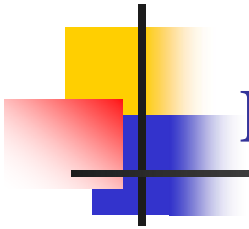


- Khái niệm tiểu trình được xây dựng bên trong hạt nhân
- Distpatcher làm việc với đơn vị là tiểu trình

Tiểu trình người dùng (User thread)

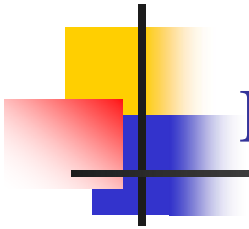
- Khái niệm tiểu trình được hỗ trợ bởi một thư viện hoạt động trong user mode
- Distpatcher của hạt nhân làm việc với đơn vị là tiến trình
- ThreadDistpatcher làm việc với đơn vị là tiểu trình
  - P –– LWP - T
- Không cần chuyển đổi chế độ xử lý khi chuyển đổi các tiểu trình cùng thuộc 1 tiến trình.





Lựa chọn tiến trình ?

- Tác vụ của Scheduler
- Mục tiêu ?
  - Sử dụng CPU hiệu quả
  - Đảm bảo tất cả các tiến trình đều tiến triển xử lý
- Tiêu chuẩn lựa chọn ?
  - Tất cả các tiến trình đều như nhau ?
  - Đề xuất một độ ưu tiên cho mỗi tiến trình ?
- Thời điểm lựa chọn ? (Thời điểm kích hoạt Scheduler())

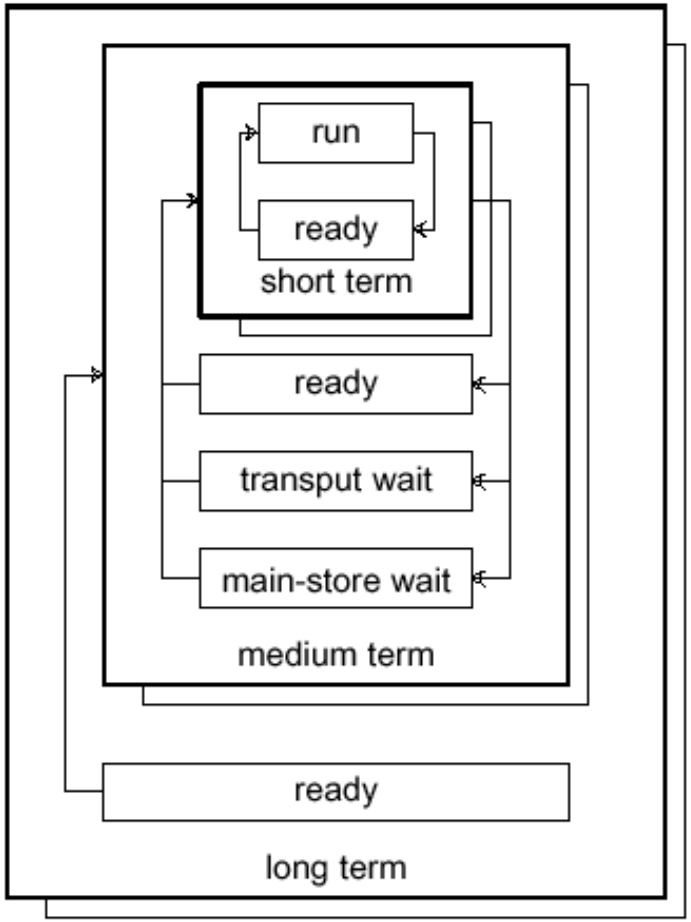


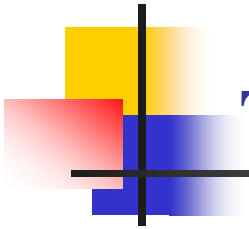
Mục tiêu điều phối

- **Hiệu qủa (Efficiency)**
  - **Thời gian**
    - **Đáp ứng (Response time)**
    - **Hoàn tất(Turnaround Time = T_{quit} -T_{arrive}):**
    - **Chờ (Waiting Time = $T_{in\ Ready}$) :**
  - **Thông lượng (Throughput = # jobs/s)**
    - **Hiệu suất Tài nguyên**
    - **Chi phí chuyển đổi**
- **Công bằng (Fairness) : Tất cả các tiến trình đều có cơ hội nhận CPU**

Các cấp độ điều phối

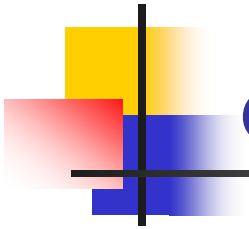
- **Longterm scheduling : chọn tiến trình kế tiếp được khởi động (mang vao bộ nhớ và nhận trạng thái ready)**
- **Mediumterm scheduling : quyết định chuyển tiến trình đang running sang trạng thái blocked.**
- **Shorterm scheduling : chọn 1 tiến trình ở trạng thái ready để chuyển sang trạng thái running.**
- **Không có sự phân biệt rõ**





Thời điểm ra quyết định điều phối

- **Điều phối độc quyền (non-preemptive scheduling): tiến trình được chọn độc chiếm CPU**
- **Điều phối không độc quyền (preemptive scheduling): tiến trình được chọn có thể bị « cướp » CPU bởi tiến trình có độ ưu tiên cao hơn**

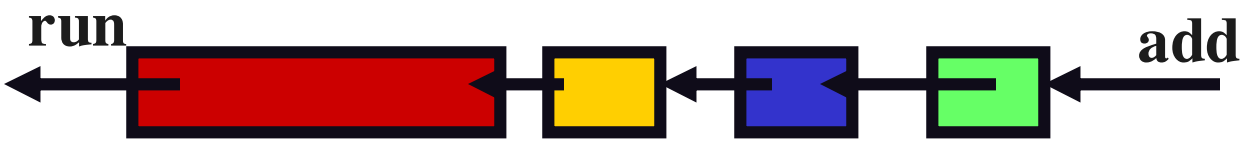


Các chiến lược điều phối

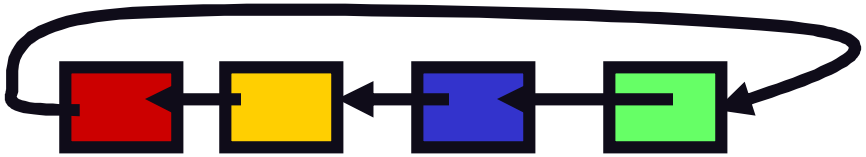
- **FIFO**
- **RR**
- **SJF**
- **MULTILEVELFEEDBACK**
- **LOTTERY**

FIFO – RR -SJF

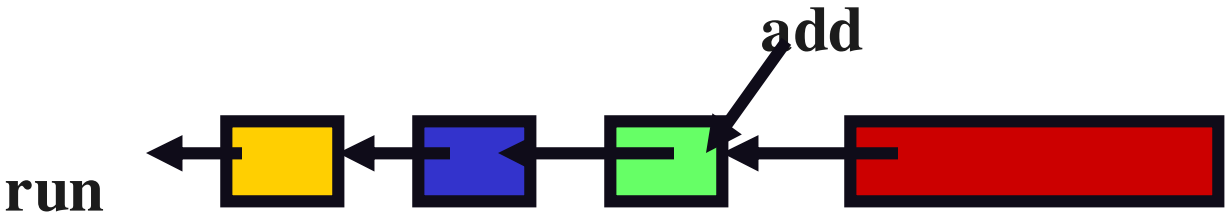
- **FIFO**



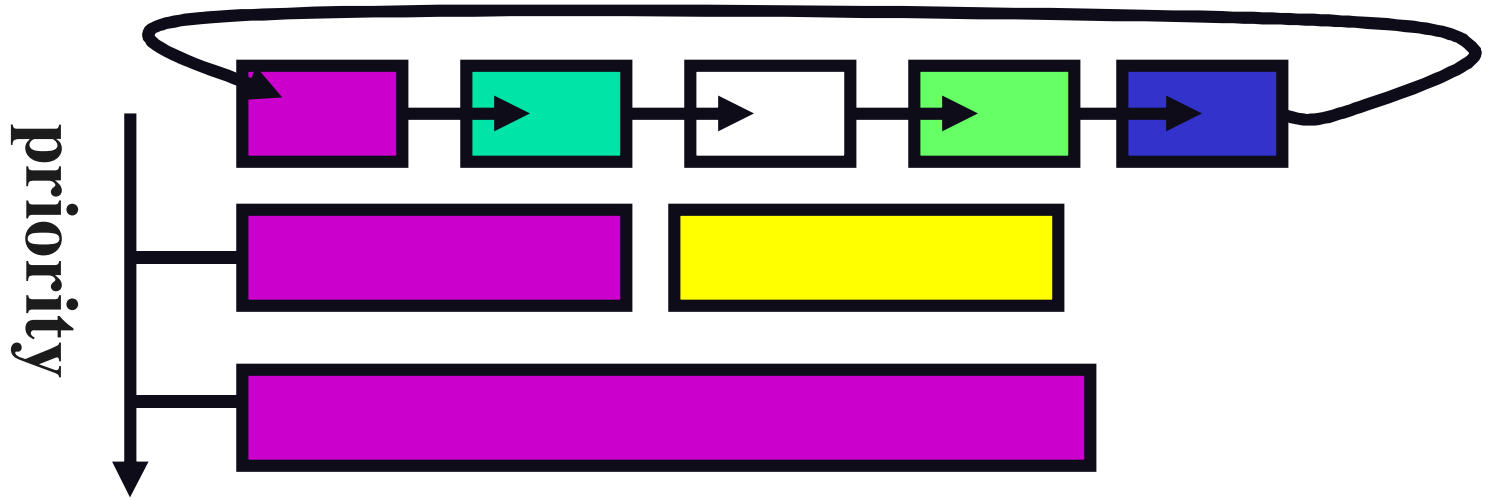
- **RR**



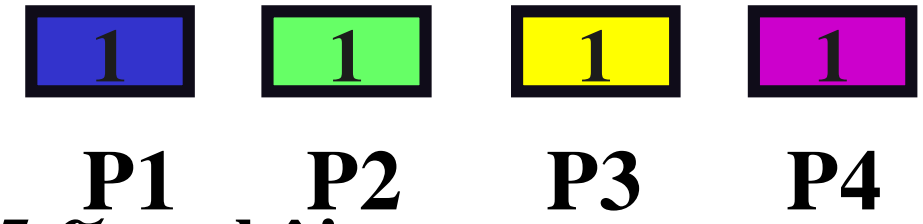
- **SJF**



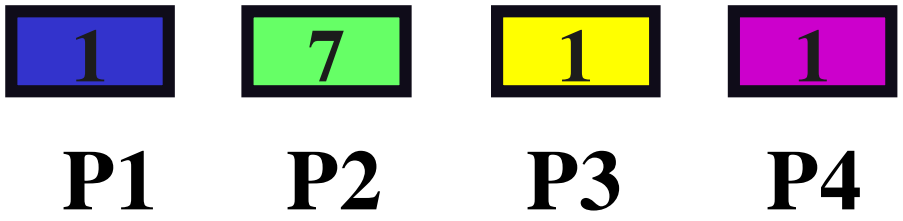
Multilevel Feedback



Lottery



P2 có 25 % cơ hội

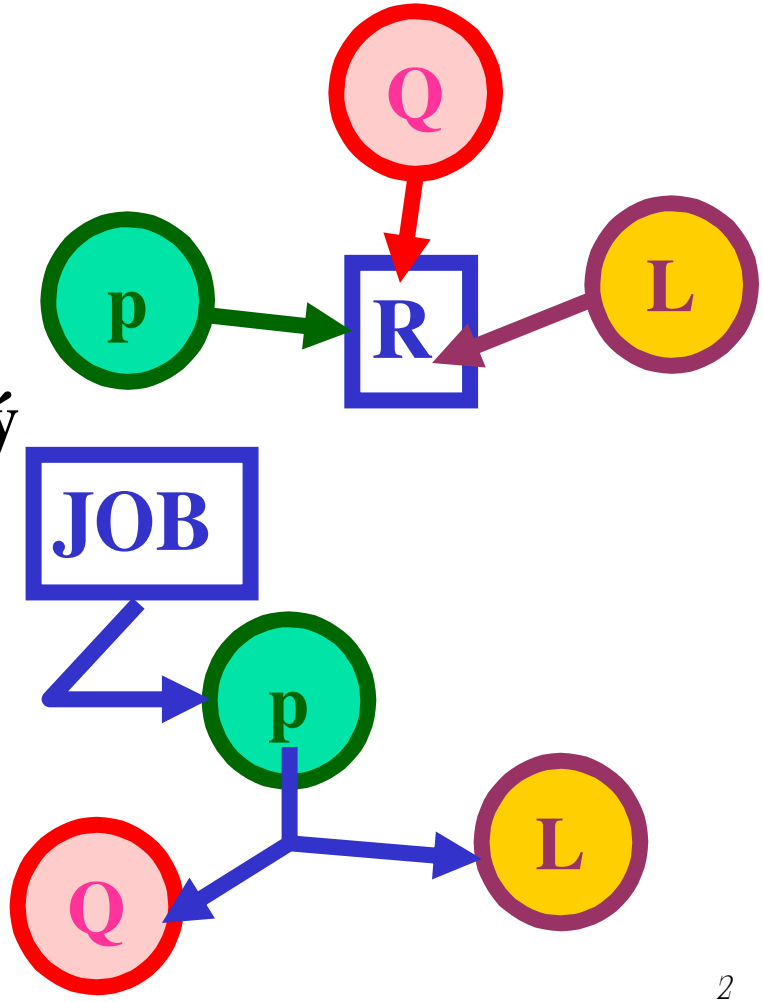


P2 có 70 % cơ hội

BÀI 4 : LIÊN LẠC GIỮA CÁC TIẾN TRÌNH & VẤN ĐỀ ĐỒNG BỘ HOÁ

Nhu Cầu Liên Lạc

- **Chia sẻ thông tin**
- **Phối hợp tăng tốc độ xử lý**

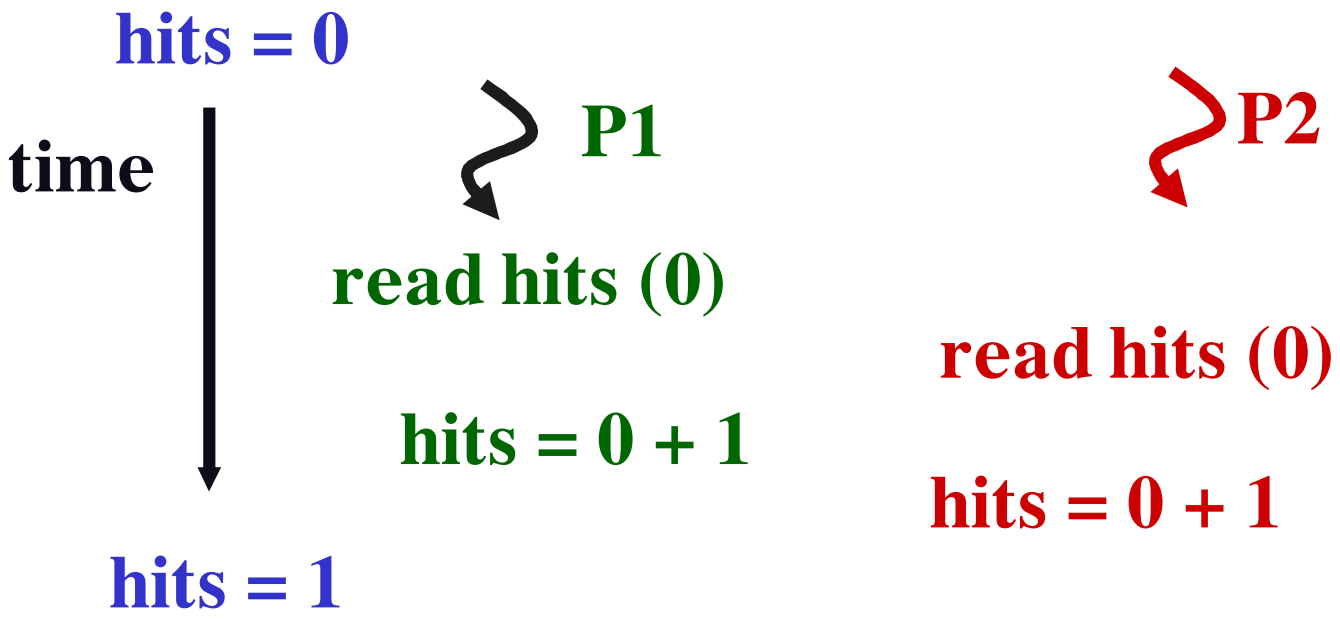


Các Cơ Chế Liên Lạc

- **Signal**
  - ☹Không truyền được dữ liệu
- **Pipe**
  - ☹ Truyền dữ liệu không cấu trúc
- **Shared Memory**
  - ☺ Broadcast
  - ☹ Mâu thuẫn truy xuất => nhu cầu đồng bộ hoá
- **Message**
  - ☺ Liên lạc trên môi trường phân tán
- **Socket**
  - ☺ Liên lạc trên nhiều môi trường khác biệt

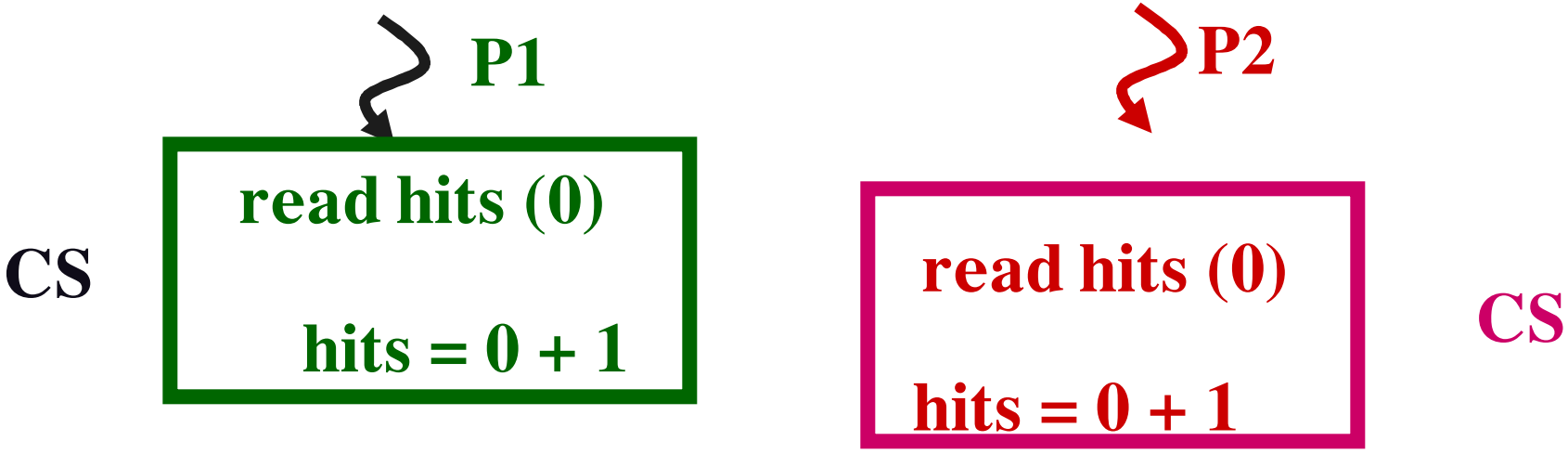
Race condition

- **P1 và P2 chia sẻ biến chung hits**



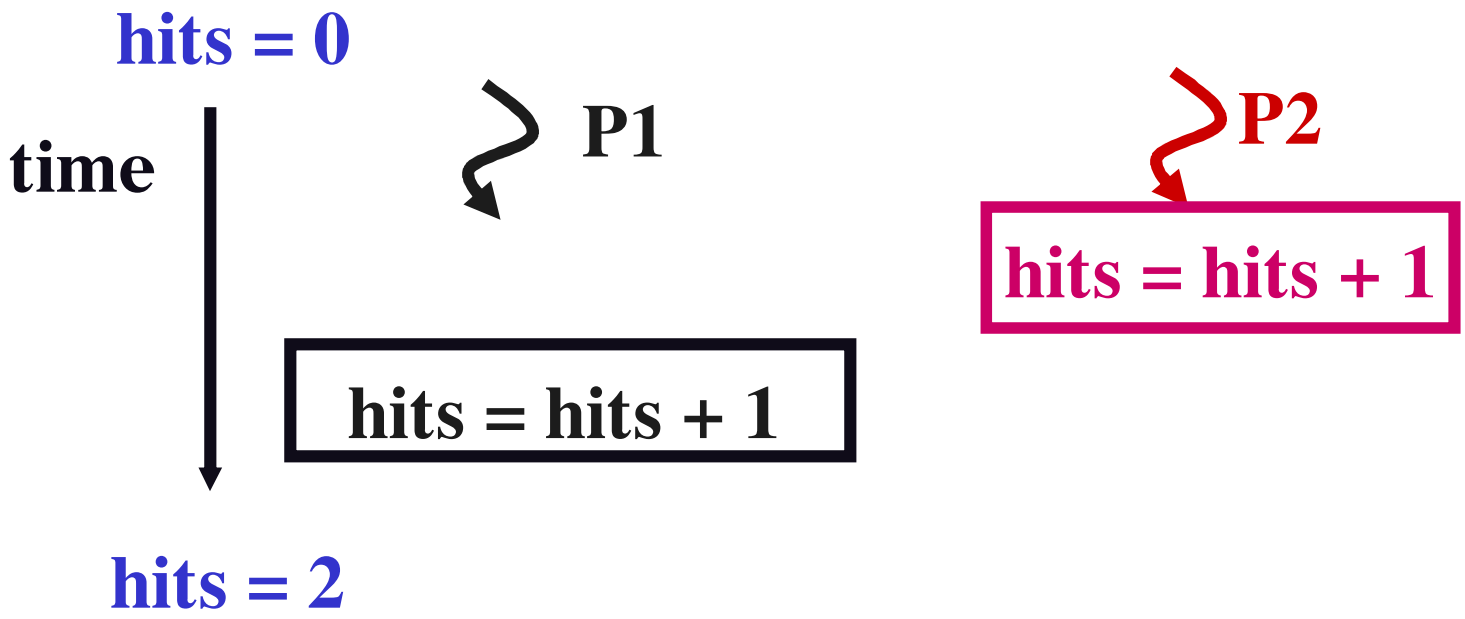
☹ **Kết quả cuối cùng không dự đoán được !**

Miền găng (critical section)

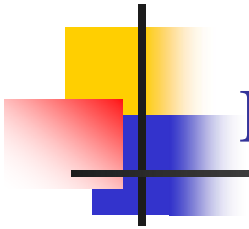


CS là đoạn chương trình có khả năng gây ra hiện tượng race condition

Giải pháp tổng quát



Bảo đảm tính "độc quyền truy xuất" miền găng tại một thời điểm



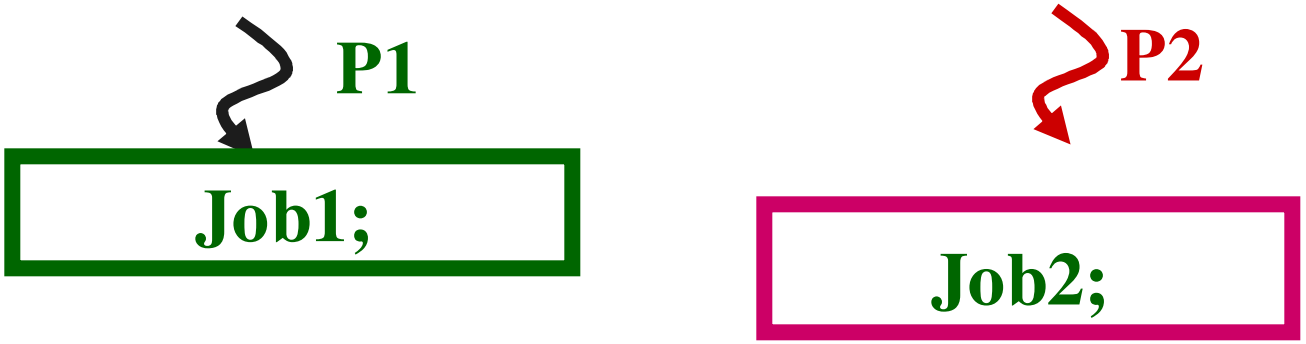
Mô hình đảm bảo độc quyền truy xuất

Kiểm tra và dành quyền vào CS

CS;

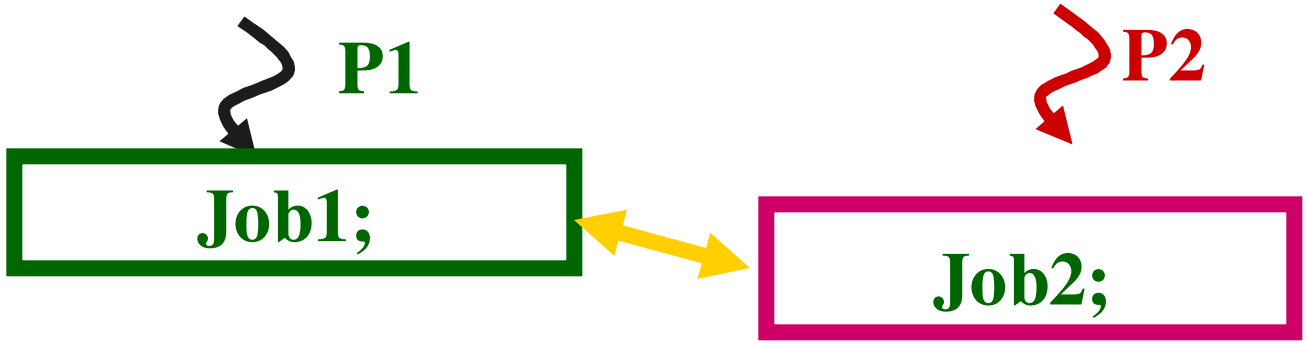
Từ bỏ quyền sử dụng CS

Rendez-Vous



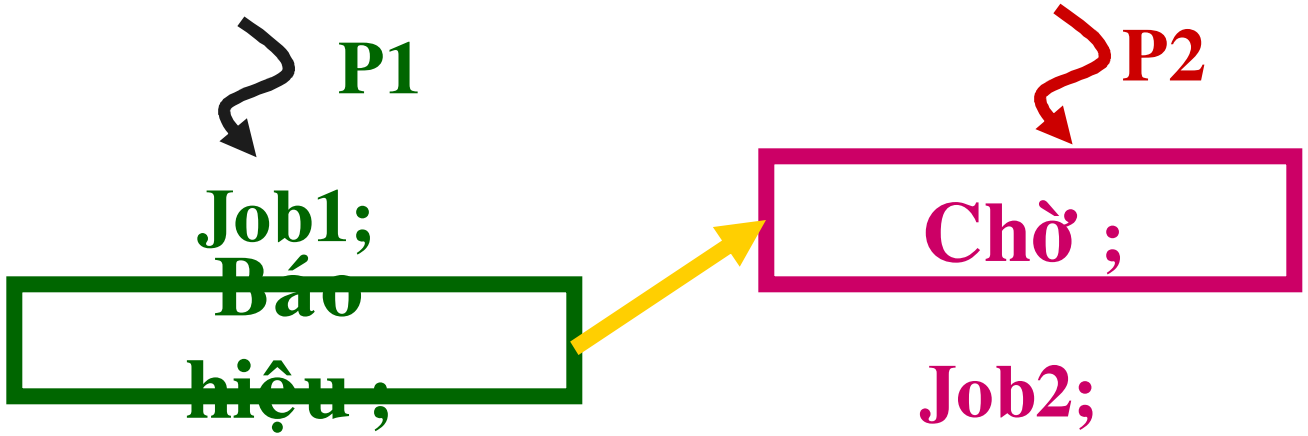
Làm thế nào bảo đảm trình tự thực hiện Job1 - Job2 ?

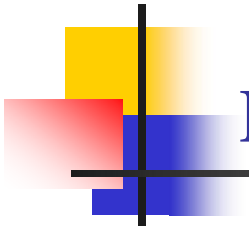
Giải pháp



Hai tiến trình cần trao đổi thông tin về diễn tiến xử lý

Mô hình tổ chức phối hợp hoạt động giữa hài tiến trình





Bài toán đồng bộ hoá

- **Nhiều tiến trình chia sẻ tài nguyên chung đồng thời :**
  - **Tranh chấp ?**
  - **Nhu cầu "độc quyền truy xuất" (mutual exclusion)**
- **Các tiến trình phối hợp hoạt động :**
  - **Tương quan diễn tiến xử lý ?**
  - **Nhu cầu "hò hẹn" (rendez-vous)**

HỆ ĐIỀU HÀNH

Tài liệu tham khảo He:

- Slide bài giảng
  - http://fit.hcmup.edu.vn/~haits
  - http://www.box.net/shared/2v0t4sqeot
- Lê Khắc Nhiên Ân, giáo trình "Hệ điều hành cơ bản", ĐHKHTN TPHCM
- Trần Hạnh Nhi, giáo trình "Hệ điều hành nâng cao", ĐHKHTN TPHCM

* Tổng quan về hệ thống máy tính

Kiến trúc máy tính gồm 2 phần chính:

+ HSA (Kiến trúc hệ thống phần cứng)

+ ISA (Kiến trúc bộ lệnh)

* Các thành phần chính của hệ thống máy tính:

+ HSA (Kiến trúc hệ thống phần cứng)

Để có thể hoạt động, bắt buộc phải có:

- *- Hệ thống xử lý*
- *- Hệ thống lưu trữ*
- *- Hệ thống nhập xuất*

Hệ thống lưu trữ

- *Bộ nhớ trong*

  (RAM, ROM, Registers, Cache, Port)
- *Bộ nhớ ngoài*

  (HD, FD, CD, DVD, USB disk, ...)

* Các thành phần chính của hệ thống máy tính:

+ HSA (Kiến trúc hệ thống phần cứng)

+ ISA (Kiến trúc bộ lệnh)

- *- Kiểu dữ liệu*
- *- Tập thanh ghi*
- *- Tập lệnh*

* Các hệ thống số

- Hệ nhị phân (cơ số 2):

Hình thức thể hiện: 01001b, 11111001b, 11100b..

- Hệ thập phân (cơ số 10):

Hình thức thể hiện: 9, 249, 28d, ..

- Hệ thập lục phân (cơ số 16):

Hình thức thể hiện: 9h, 0F9h, 1Ch, ..

Các kiểu dữ liệu trong máy tính

- **bit**: đơn vị lưu trữ nhỏ nhất
- **Byte**: đơn vị truy xuất của chương trình
- **Word**: đơn vị truy xuất của máy tính
  (có kích thước phụ thuộc vào CPU & lưu ngược theo đơn vị Byte – xem ví dụ)
- **Chuỗi ký tự**: lưu trữ theo thứ tự bình thường
- **Số BCD**: lưu trữ mỗi chữ số của 1 số thập phân bằng một (hoặc nửa) Byte

Tổ chức dữ liệu trên bộ nhớ trong

- **Byte**: đơn vị truy xuất bộ nhớ trong của phần mềm
  (gồm 8 bit - bit phải nhất là bit 0 & bit trái nhất là bit 7)
- **Word**: đơn vị truy xuất của phần cứng (có kích thước phụ thuộc vào CPU) hoặc 1 kiểu dữ liệu của phần mềm (có kích thước phụ thuộc vào phần mềm tương ứng)
- **Chuỗi ký tự**: lưu trữ theo thứ tự bình thường
- **Số nguyên**: lưu ngược theo đơn vị Byte (khảo sát các ví dụ cụ thể)

* Tổ chức thanh ghi:

+ Khái niệm thanh ghi:

- Là các vùng nhớ nhỏ lưu dữ liệu trong các chip xử lý
- Có tốc độ truy xuất rất nhanh & tần suất sử dụng cao

+ Kích thước thanh ghi:

Tính theo đơn vị bit – tùy thuộc chức năng của chip

Các thanh ghi trong CPU 8bit có kích thước 8bit, trong CPU 16bit có kích thước 16bit, trong CPU 32bit có kích thước 32bit đồng thời có luôn các thanh ghi của CPU 16 bit

+ Số lượng thanh ghi:

Thường rất ít, tùy thuộc mức độ xử lý & thiết kế của chip

CPU Intel 16 bit có 14 thanh ghi, phân thành 5 nhóm

* Tổ chức thanh ghi của CPU 16 bit:

+ Nhóm thanh ghi đoạn (Segment register):

- CS (Code Segment): lưu chỉ số của segment chứa CT ngôn ngữ máy.
- DS (Data Segment): lưu chỉ số segment chứa dữ liệu của CT.
- ES (Extra Segment): lưu chỉ số segment chứa dữ liệu bổ sung của CT.
- SS (Stack Segment): lưu chỉ số segment chứa ngăn xếp của CT.

(Trên CPU 32bit có thêm 2 thanh ghi FS, GS có chức năng tương tự như ES)

CT muốn truy xuất 1 vùng nhớ thì phải

đưa chỉ số segment của vùng nhớ đó vào một thanh ghi đoạn

* Tổ chức thanh ghi của CPU 16 bit:

+ Nhóm thanh ghi con trỏ & chỉ mục (Pointer & Index Reg.)

- IP (Instruction Pointer): lưu offset của ô nhớ chứa lệnh kế tiếp.
- BP (Base Pointer): lưu offset của ô nhớ cần truy xuất.
- SP (Stack Pointer): lưu offset đỉnh ngăn xếp.
- SI (Source Index): lưu offset vùng nhớ nguồn cần đọc lên.
- DI (Destination Index): lưu offset vùng nhớ đích cần ghi xuống.

Mỗi thanh ghi trong nhóm này phải đi kèm với 1 thanh ghi trong nhóm thanh ghi đoạn mới biểu thị được vùng nhớ /ô nhớ cần truy xuất.

* Tổ chức thanh ghi của CPU 16 bit:

+ Nhóm thanh ghi đa dụng (General Register)

- AX (Accumulator Register): lưu các dữ liệu số, giá trị mặc định, …
- BX (Base Register): đóng vai trò chỉ số mảng,, cũng có thể lưu dữ liệu
- CX (Count Register): có thể dùng để định số lần lặp.
- DX (Data Register): lưu dữ liệu /kết quả tính toán (~AX).

Mỗi thanh ghi trong nhóm này đều cho phép sử dụng đến từng Byte, bằng cách thay chữ 'X' thành chữ 'H' để chỉ Byte cao, hoặc 'L' để chỉ Byte thấp (AH, BL, CL)

→ **Xem như có thêm 8 thanh ghi mới**

* Tổ chức thanh ghi của CPU 16 bit:

+ Thanh ghi cờ (Flag Register)

Không có tên, mỗi bit là 1 cờ, biểu diễn trạng trái của lệnh vừa thực hiện, hoặc đặt trạng trái cho lệnh thực hiện

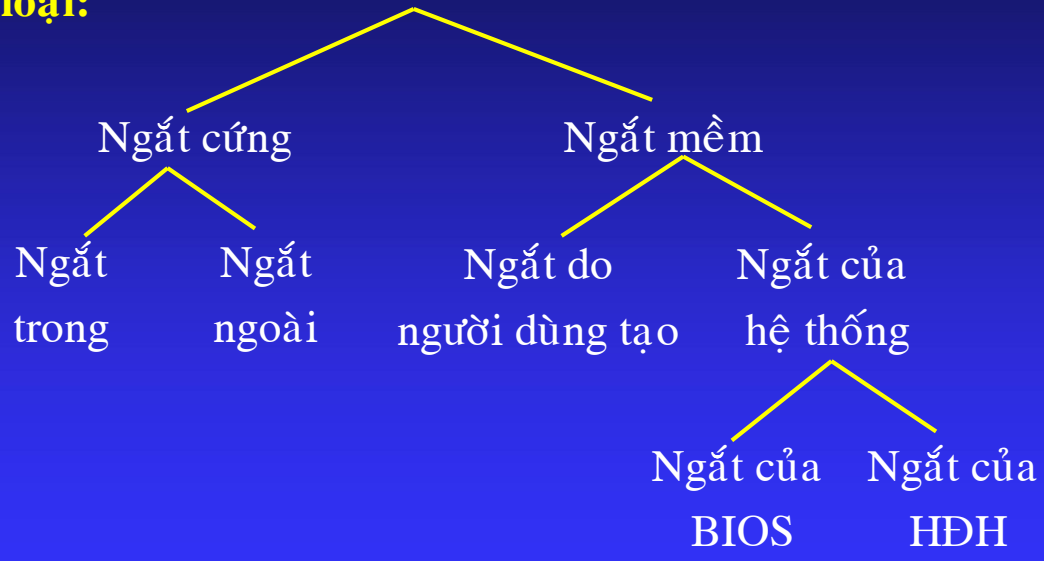
* Hệ thống ngắt (Interrupt):

+ Khái niệm:

- Là các chương trình con (thủ tục /hàm) có sẵn trong máy
- Các hàm ngắt không chỉ có sẵn trong ROM BIOS mà còn được Hệ Điều Hành bổ sung thêm
- Hàm ngắt không có tên mà thay vào đó là 1 con số (0..FF)
- Mỗi hàm ngắt lại chứa bên trong nó nhiều hàm con
- Tham số in./output của hàm ngắt được truyền qua thanh ghi

Hệ thống ngắt (Interrupt):

+ Phân loại:



Hệ thống ngắt (Interrupt):

+ Chương trình ngắt:

Thường gồm nhiều đoạn chương trình con bên trong, mỗi đoạn thực hiện 1 công việc cụ thể nào đó, chỉ số của đoạn thể hiện trong AH. Khi đó thân hàm ngắt có dạng (mã giả):

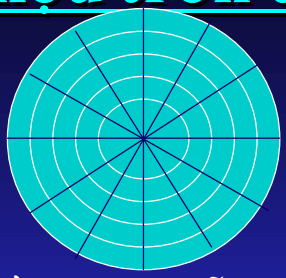
```
switch (AH) {
case 0:
        .. .. ..    // hàm con 0
        break;
case 1:
        .. .. ..    // hàm con 1
        break;
.. .. ..
case N:
        .. .. ..    // hàm con N
        break;
}
```

Một Số Thiết Bị

- Đĩa mềm
- Đĩa cứng
- Card màn hình

Tổ chức dữ liệu trên đĩa từ

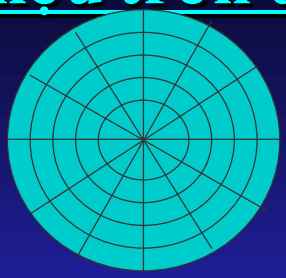
+ Cấu trúc vật lý :



- Hình tròn, gồm nhiều mặt, mỗi mặt có nhiều đường tròn đồng tâm, trên các đường tròn có các cung tròn, thông thường mỗi cung chứa 4096 điểm từ (=4096bit = 512 byte)
- Mỗi mặt có tương ứng 1 đầu đọc để đọc hoặc ghi dữ liệu.
- Mỗi lần đọc /ghi ít nhất 1 cung tròn (512 B)
- Các cung tròn, đường tròn & đầu đọc (hoặc mặt) có các từ gốc tương ứng là sector, track (hoặc cylinder) & head.
- Mỗi lần truy xuất (đọc hoặc ghi đĩa) chỉ có thể thực hiện trên N sector liên tiếp (N>=1)

Tổ chức dữ liệu trên đĩa từ

+ Cấu trúc vật lý :



- Để truy xuất 1 sector cần phải chỉ ra vị trí của sector đó. Vị trí sector được thể hiện bằng 3 thông số: chỉ số sector, track & head
- Head được đánh số theo thứ tự từ trên xuống bắt đầu từ 0, Track được đánh số theo thứ tự từ ngoài vào bắt đầu từ 0, Sector được đánh chỉ số theo thứ tự bắt đầu từ 1 theo chiều ngược với chiều quay của đĩa.
- Địa chỉ của sector vật lý có ký hiệu : (sector, track, head)

Tổ chức dữ liệu trên đĩa từ

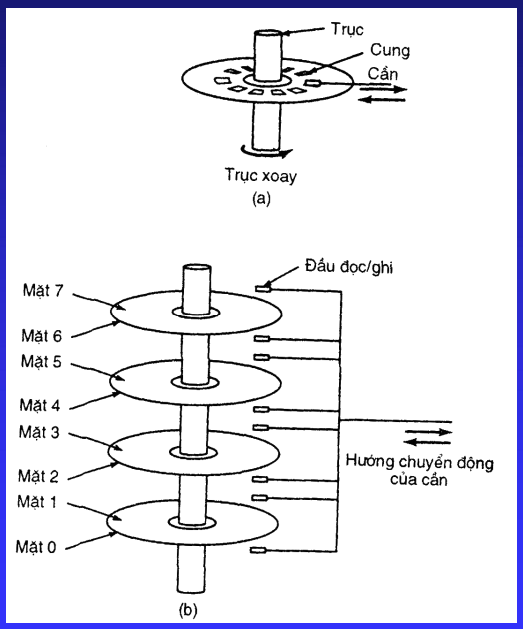
+ Tổ chức đĩa logic:

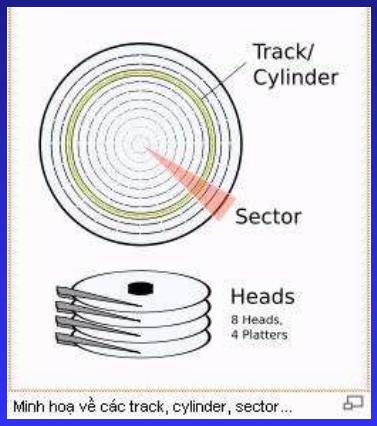
- Là 1 dãy các sector được đánh chỉ số theo thứ tự tăng dần bắt đầu từ 0
- Đĩa thật sự là đĩa vật lý nhưng vì truy xuất phải dùng đến 3 tham số rất bất tiện nên khái niệm đĩa logic được đưa ra để dễ hiểu, dễ thao tác /tính toán hơn.
- Mỗi sector trên đĩa logic tương ứng với 1 sector duy nhất trên đĩa vật lý sao cho sau khi truy xuất sector K thì truy xuất tiếp sang sector K+1 là nhanh hơn so với tất cả các sector khác.

Đĩa cứng

- **Ổ đĩa cứng**, hay còn gọi là **ổ cứng** (***Hard Disk Drive*-HDD**) là thiết bị dùng để lưu trữ dữ liệu trên bề mặt các tấm đĩa hình tròn phủ vật liệu từ tính.

Đĩa cứng





Đĩa cứng

- Trên một track chia làm nhiều cung bằng nhau là 1 sector và được đánh số từ một. Qua track mới đánh số lại. Số sector trên mỗi track là bằng nhau.
- Cylinder là tập hợp các track có cùng số hiệu trên tất cả các mặt.
- Cluster là tập hợp các sector (đĩa mềm 1 cluster = 1 sector)

Đĩa cứng

- Đơn vị cơ sở của đĩa là sector (thường = 512 byte)→Đĩa là thiết bị xuất nhập theo khối.
- Một sector được xác định bằng 2 cách:
  - Sector tương đối: sector vật lý xác định bởi head (đánh số từ 0), track (đánh số từ 0) trên head và sector trên track (đánh số từ 1).
  - Sector tuyệt đối: sector logic xác định bởi một thông tin. Cách đánh số xác định sector tuyệt đối như sau: đánh số hết trên một cylinder sau đó trở về mặt 0 để đánh số tiếp theo cho mặt 0 của cylinder tiếp theo.

Đĩa cứng

- Ví dụ: cho một đĩa cứng có 160 mặt, 640 cylinder và 80 sector/track. Hỏi ổ cứng có dung lượng bao nhiêu MB?
- Dung lượng = Số sector/đĩa * 512 (byte)
- DL = Số sector/cylinder * 640 * 512 (byte)
- DL = Số sector/track * 160 * 640 * 512 (byte)
- DL = (80 * 160 * 640 * 5120) : (1024^2) (MB)

Card Màn Hình

- Chất lượng màn hình phụ thuộc vào card màn hình. Có 3 yếu tố để xác định chất lượng của card màn hình:
  - Độ phân giải ví dụ 800x600
  - Số màu = số màu của 1 pixel được quy định bởi số bit màu quản lý cho 1 pixel.
  - Kích thước bộ nhớ = 1p * số trang. Trong đó 1p = độ phân giải * số byte màu. (1p là một trang màn hình).

Card màn hình

- Ví dụ: 1 card màn hình sử dụng độ phân giải 800x600 có thể hiển được 65536 màu. Tốc độ load hình là 24 frame/s. Hỏi card màn hình này có dung lượng là bao nhiêu Mb.
- 65536 màu = 2^{16} màu = 2 byte màu
- Tốc độ load hình 24 frame/s → số trang là 24
- Kích thước bộ nhớ = 1p * số trang = 1p * 24
- Kích thước = độ phân giải * số byte màu * 24
- Kích thước = 800*600*2*24 (byte) ≈ 22 (MB)

Câu hỏi ôn tập

- 1/ Cho một đĩa cứng có dung lượng 640 MB, đĩa này sử dụng 640 mặt. Biết số sector trên 1 track gấp đôi số track trên một mặt. Hỏi số sector trên một cylinder là bao nhiêu?
- 2/ Cho một card màn hình có dung lượng 8 MB, sử dụng 24 frame/s. Hỏi card màn hình này có thể sử dụng được những độ phân giải thông thường nào khi số màu hiển thị là 256 màu.
- 3/Tìm công thức xác định số hiệu sector tuyệt đối từ sector tương đối và ngược lại.

- DL = Số Sec/Đĩa * 512 (byte)
- DL = Số Sec/mặt * Số Mặt/Đĩa * 512
- DL = Số Sec/mặt * Số Mặt * 512
- DL = Số Sec/Track * Số Track/mặt * Số Mặt * 512
- DL = x * x/2 * 640 * 512 (byte) = 640 (MB) = 640*1024*1024 → x?
- Số Sec/Cyl = Số Sec/Track * Số Track/Cyl
- Số Sec/Cyl = x * Số Track/Cyl
- Số Sec/Cyl = x * Số mặt/Đĩa

Tổng quan về Hệ điều hành

Trần Sơn Hải
Khoa Toán – Tin học
Đại học Sư phạm TPHCM

Heavily reference to Operating System Slide of Hoang Than Anh Tuan, University of Pedagogy, HCMC

Nội dung

- Hệ điều hành là gì?
- Vai trò và đặc điểm của hệ điều hành
- Lịch sử phát triển của hệ điều hành
- Các chức năng cơ bản của hệ điều hành

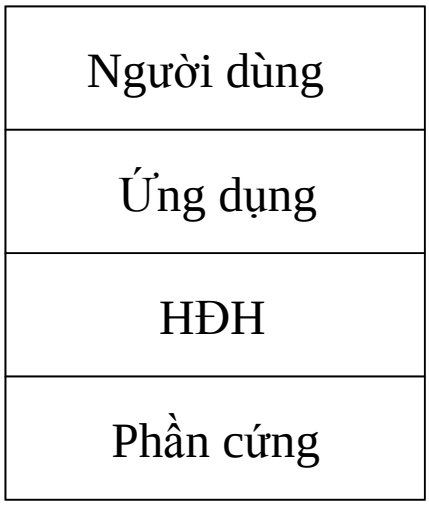
Hệ điều hành là gì?

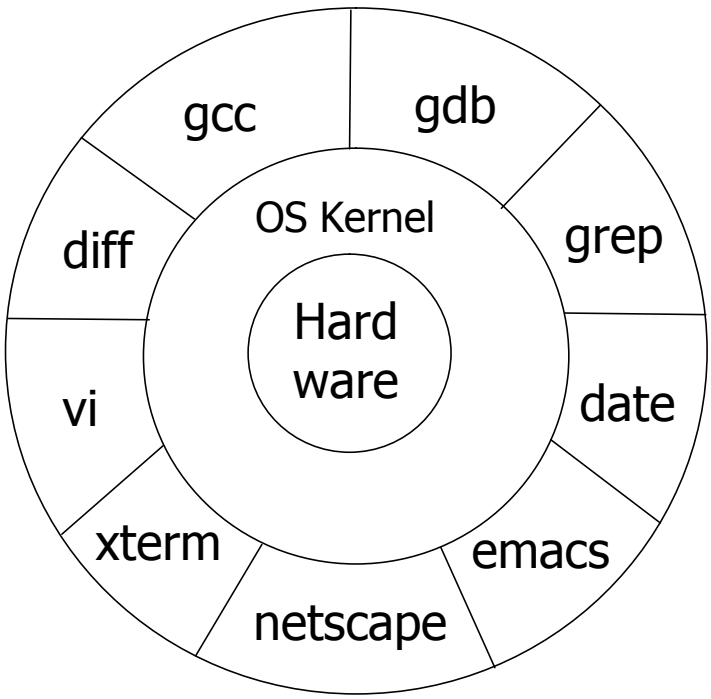
- Là một chương trình (phần mềm)!
- Chương trình đầu tiên được chạy sau khi khởi động.
- Điều phối việc thi hành các phần mềm khác
- Cung cấp các dịch vụ thông dụng sử dụng bởi người dùng và các chương trình ứng dụng khác

Định nghĩa HĐH

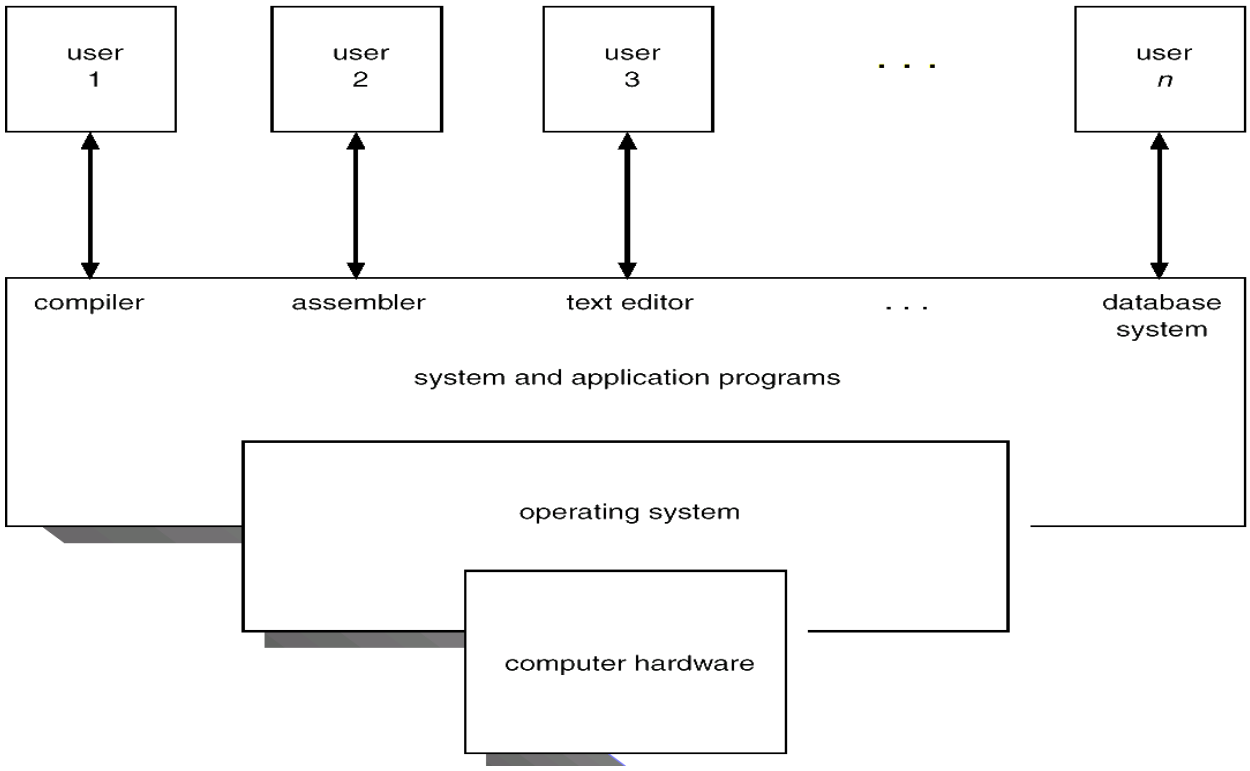
- ***HĐH*** là một chương trình hoạt động như là một trung gian giữa người dùng và phần cứng máy tính.
- Mục đích của HĐH:
  - Kiểm soát và thực thi các chương trình ứng dụng.
    - Quản lý tiến trình
    - Quản lý CPU
  - Giúp người sử dụng dễ sử dụng máy vi tính.
  - Sử dụng tài nguyên phần cứng máy tính một cách hiệu quả
    - Quản lý đĩa cứng
    - Quản lý bộ nhớ

Vai trò của HĐH

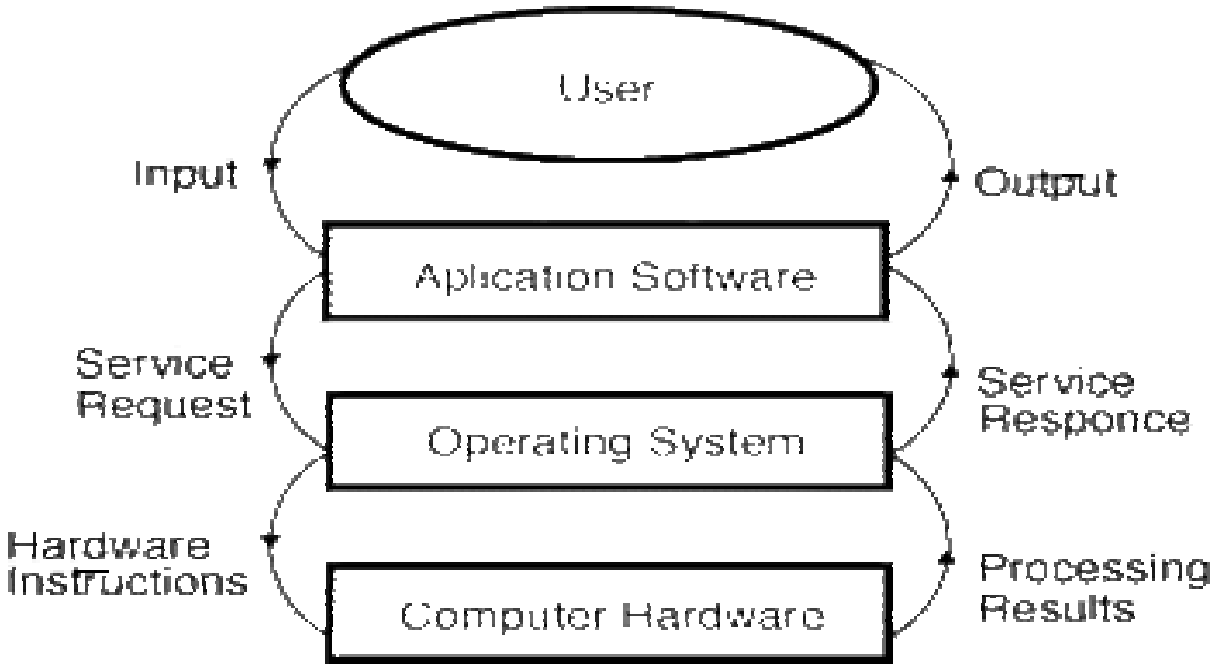




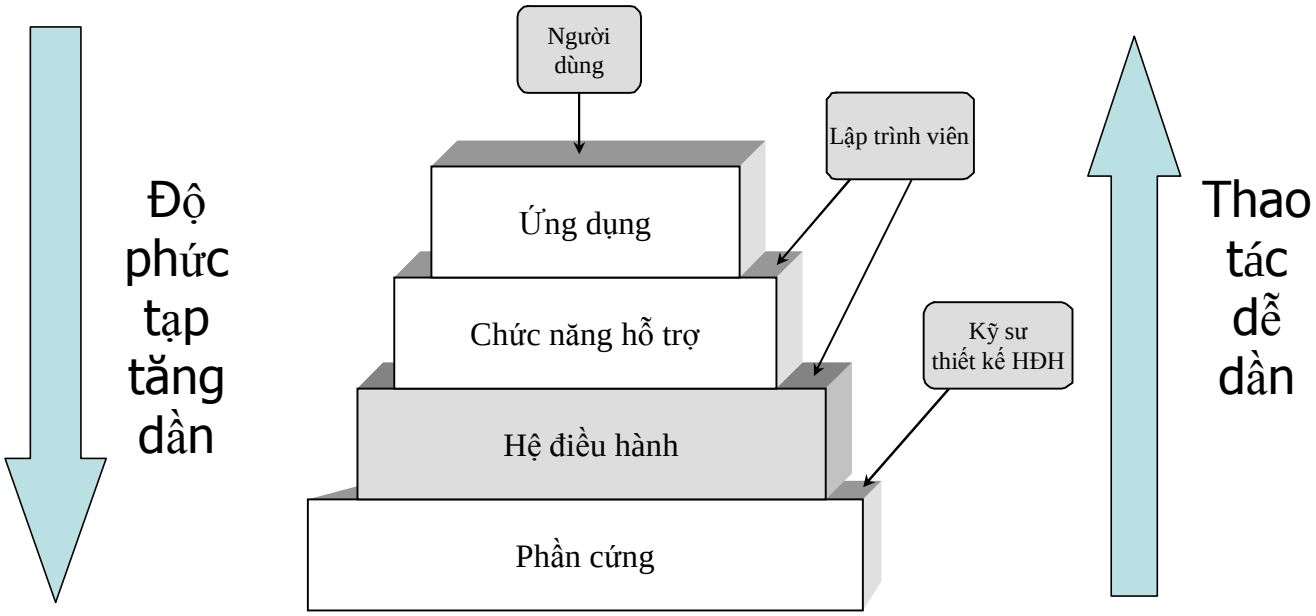
Vai trò của HĐH (trạng thái tĩnh) (tt)



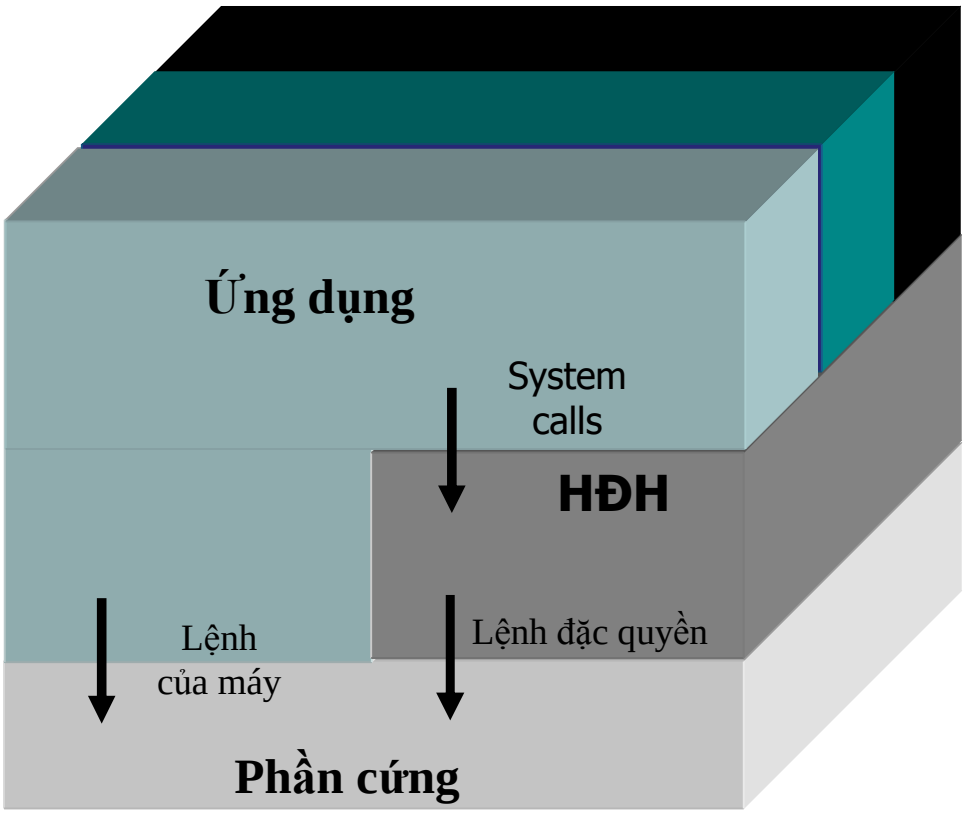
Vai trò của HĐH (trạng thái động) (tt)



Phân lớp trong một hệ thống



Mô hình khác



Nhiều ứng dụng

Một HĐH

Một Phần cứng

HĐH là một PM có tính phản ứng

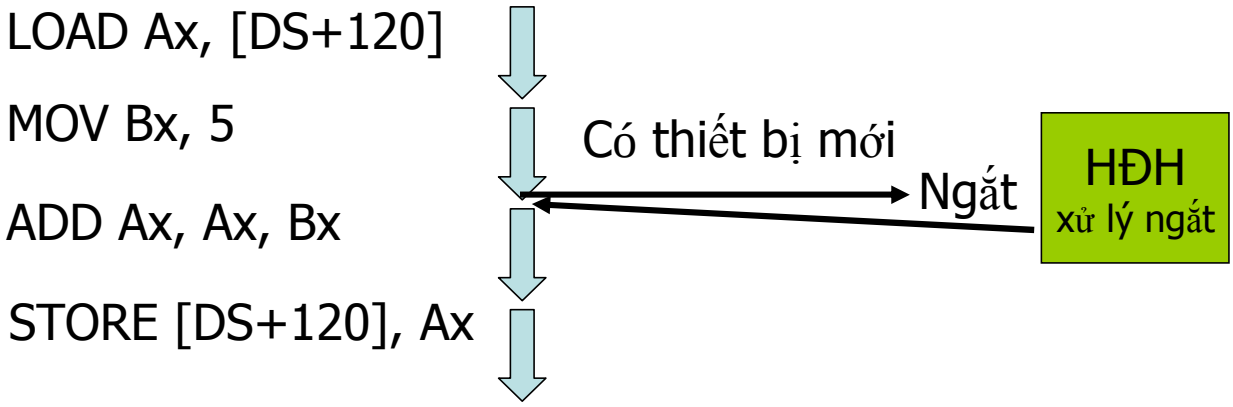
- Nó chờ đợi các *sự kiện* (*event).*
- Khi một sự kiện xảy ra, HĐH phản ứng.
  - HĐH xử lý sự kiện đó.
    - Có thêm một chương trình muốn chạy.
    - Có thêm thiết bị mới gắn vào
- Việc xử lý sự kiện phải tốn càng ít thời gian càng tốt.
- Loại sự kiện
  - Ngắt (interrupts)
  - Lời gọi hệ thống (System calls)

Ngắt

- ***Ngắt*** là một cách thức mà phần cứng thông báo cho HĐH về các tình huống đặc biệt đòi hỏi HĐH phải lưu ý
- Ngắt làm cho CPU tạm dừng thi hành chỉ thị kế tiếp
- Thay vì đó, quyền điều khiển được chuyển sang cho HĐH

Minh họa ngắt

Chương trình A



Xử lý ngắt

- *Bộ xử lý ngắt* là một phần của mã lệnh HĐH dùng để thao tác liên quan đến hoàn cảnh phát sinh ra ngắt
  - Các con trỏ trỏ đến các bộ xử lý ngắt được lưu trong vector ngắt (*interrupt vector*)
  - Vector ngắt được lưu trong một vị trí bộ nhớ được định trước

Xử lý ngắt (tt)

- Khi một ngắt xảy ra:
  - CPU chuyển sang chế độ Kernel
  - Chuyển quyền điều khiển cho bộ xử lý ngắt thích hợp
  - Địa chỉ của bộ xử lý ngắt được tìm thấy bằng cách sử dụng số thứ tự của ngắt như là một chỉ số đề vector ngắt:
    - Jump &int[interrupt#]

Vector ngắt

- Địa chỉ vector ngắt và số thứ tự ngắt là một phần trong đặc tả phần cứng
- HĐH tải các địa chỉ ngắt vào trong vector ngắt trong quá trình khởi động

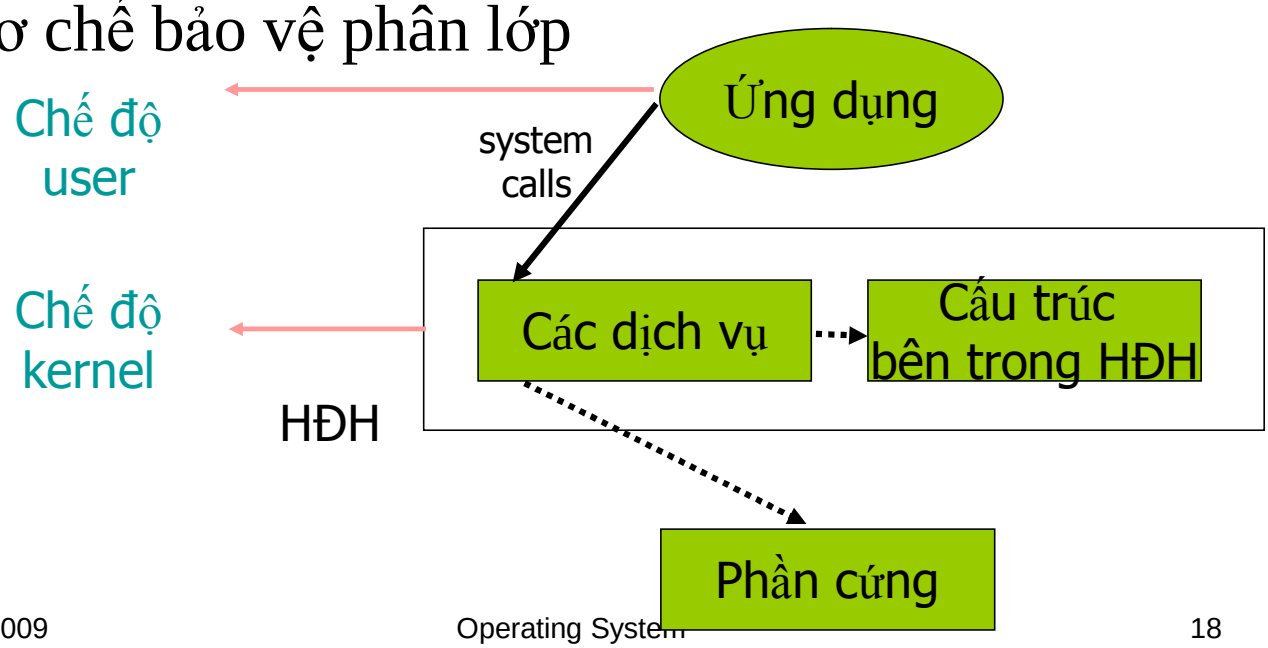
Các loại ngắt

- *Ngắt không đồng bộ* được tạo ra bởi các thiết bị ngoại vi tại những thời điểm không thể biết trước
- *Ngắt bên trong* (*đồng bộ*) được tạo ra bởi CPU do một tình huống có thể lường trước
  - Tình huống gây lỗi
  - Vấn đề có tính tạm thời

Lời gọi hệ thống

- Hệ điều hành cung cấp một tập hợp các dịch vụ hệ thống (các chức năng mà hệ điều hành cung cấp, thường liên quan đến phần cứng).
  - Nhằm giúp cho các chương trình ứng dụng không cần quan tâm đến chi tiết phần cứng
  - Tập hợp các lời gọi hệ thống được gọi là API của HĐH (OS *API)*
  - Được đóng gói thành một thư viện
- Khi một chương trình muốn sử dụng một dịch vụ nào đó của HĐH chúng sẽ phát sinh ***một lời gọi hệ thống (system call)***.
- Các lời gọi hệ thống thường thấy
  - Mở/đọc/ghi/đóng file
  - Lấy ngày giờ hiện tại
  - Tạo tiến trình mới
  - Yêu cầu thêm bộ nhớ

- Chương trình người dùng không thể truy cập vào các cấu trúc nội tại của HĐH; chỉ có thể thông qua lời gọi hệ thống (protection!)
- Cơ chế bảo vệ phân lớp



Chế độ thi hành CPU

- CPU hỗ trợ ít nhất là 2 chế độ thi hành (dual mode):
- *User* mode (chế độ người dùng)
  - Mã lệnh của chương trình người dùng
- *Kernel* (supervisor, privileged, monitor, system) mode
  - Mã lệnh của HĐH
- Chế độ thi hành được chỉ ra bởi một bit trong word trạng thái xử lý (*processor status word) (PSW)*

Kernel Mode

- Gần như kiểm soát hoàn toàn phần cứng:
  - Các chỉ thị CPU đặc biệt
  - Truy cập không giới hạn đến bộ nhớ, đĩa, …

Các chỉ thị chỉ có trong Kernel mode

- Tải/lưu các thanh ghi đặc biệt
  - VD: các thanh ghi dùng để định nghĩa các địa chỉ vùng nhớ có thể truy cập được
- Ánh xạ các trang bộ nhớ đến không gian địa chỉ của một tiến trình xác định
- Chỉ thị đặc mức độ ưu tiên của ngắt
- Chỉ thị để kích hoạt thiết bị I/O

Bảo vệ Kernel mode

- Đoạn mã của HĐH thi hành ở chế độ Kernel
  - Ngắt, lời gọi hệ thống
- Chỉ đoạn mã của HĐH mới được thi hành ở chế độ Kernel
- Đoạn mã của người dùng không bao giờ được thi hành trong chế độ Kernel

Các lời gọi hệ thống trong Unix

- Process control
  - **`fork(), exec(), wait(), abort()`**
- File manipulation
  - **`chmod(), link(), stat(), create()`**
- Device manipulation
  - **`open(), close(), ioctl(), select()`**
- Information maintenance
  - **`time(), acct(), gettimeofday()`**
- Communications
  - **`socket(), accept(), send(), recv()`**

Câu hỏi

- HĐH đọc dữ liệu vào, thi hành tính toán, xuất ra kết quả và thoát.
  - Đúng hay sai?

SAI!!!

- HĐH là một chương trình có *tính phản ứng* (*reactive program*)

Câu hỏi

- Hệ điều hành là gì. Ví dụ các hệ điều hành mà bạn biết.
- Ngắt là gì?
- Vẽ mô hình phân lớp trong một hệ thống và trình bày vai trò của hệ điều hành trong đó.
- Tìm hiểu các lệnh trong MS-DOS. (dir, copy, copy con, del, ipconfig, netstat, ipconfg /all, nslookup,

Lịch sử phát triển của HĐH

Lịch sử phát triển HĐH

Trần Sơn Hải
Khoa Toán – Tin học
Đại học Sư phạm TPHCM

Heavily reference to Operating System Slide of Hoang Than Anh Tuan, University of Pedagogy, HCMC

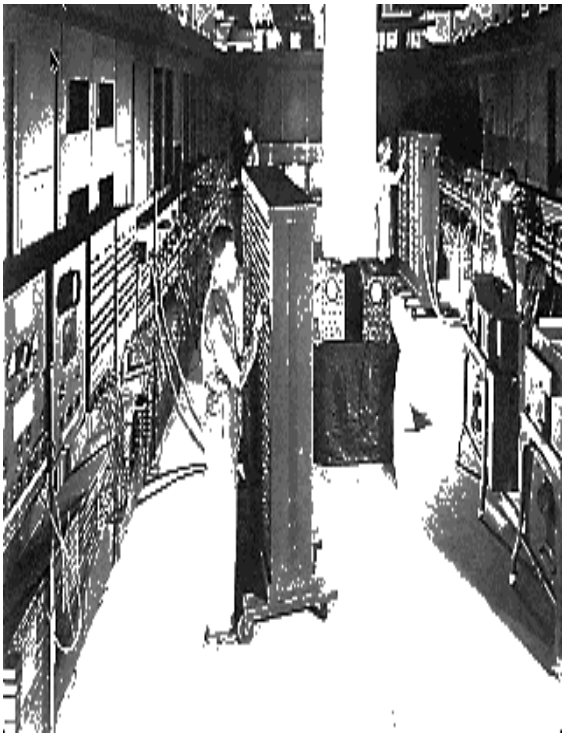
Hệ điều hành là gì?

- **Thế nào là một hệ điều hành?**
  - Một nhà cung cấp sự trừu tượng hóa
  - Một nhà cấp phát tài nguyên
  - Ngoài ra:
    - Một điều phối viên
    - Một người bạn: giúp máy tính dễ sử dụng
    - Một phù thủy: làm cho hệ thống có vẻ có nhiều hơn cái thật sự nó có (nhiều vi xử lý, nhiều bộ nhớ hơn)
  - *Công việc của hệ điều hành dựa theo phần cứng.*

HĐH bao gồm những gì?

- File systems
- Device drivers
- Networking protocols
- System utilities
- Shells
- Libraries
- Accessories
- Browser
- ……..

Thời kỳ khởi đầu ENIAC: (1945—1955)



2. Dự án chế tạo máy **ENIAC**(Electronic Numerical Integrator and Computer) được BRL (Ballistics Research Laboratory – Phòng nghiên cứu đạn đạo quân đội Mỹ) bắt đầu vào năm 1943 dùng cho việc tính toán chính xác và nhanh chóng các bảng số liệu đạn đạo cho từng loại vũ khí mới.

Các thông số:

-18000 bóng đèn chân không

- nặng hơn 30 tấn

- Tiêu thụ một lượng điện năng vào khoảng 140kW và chiếm một diện tích xấp xỉ 1393 m2 .

- 5000 phép cộng/ 1s

- Đặc biệt sử dụng hệ đếm thập phân

Lịch sử HĐH : Lịch sử phát triển phần cứng

- **Khởi đầu: Không có HĐH**
  - » Máy tính là thiết bị thực nghiệm kỳ lạ.
  - » Lập trình bằng ngôn ngữ máy.
  - » Dùng các bảng tổng đài để điều khiển máy tính.
  - » Người sử dụng ngồi trước bảng điều khiển.
  - » Không có sự chồng nhau giữa việc tính toán, I/O và thời gian suy nghĩa của người dùng.
  - » Lập trình bằng cách đưa phiếu đục lỗ vào bằng tay.
  - » Đã có thư viện được viết dùng chung cho mọi người → tiền thân của hệ điều hành.
  - » **Vấn đề: chờ đợi quá lâu, quá nhiều.**

Giai đoạn 1: máy tính đắt, nhân công rẻ. (1948 – 1975)

- Sử dụng máy tính hiệu quả hơn: tách rời máy và người.
- HĐH hỗ trợ ***làm việc theo lô****:* một chương trình tải công việc của người dùng vào, thi hành nó và làm tiếp công việc kế tiếp.
- Nếu chương trình lỗi, HĐH ghi lại nội dung của bộ nhớ và lưu lại ở đâu đó.
- Sử dụng tài nguyên hiệu quả hơn, nhưng khó debug!

Các kênh dữ liệu và ngắt cho phép I/O và tính toán chồng nhau.

- Vùng đệm và xử lý ngắt được hỗ trợ bởi hệ điều hành.
- Xuất hiện công việc spool.

Vấn đề

- Tiện ích còn thấp; mỗi thời điểm chỉ chạy một công việc.
- Không có sự bảo vệ giữa các công việc khác nhau.
- Công việc có thời gian thi hành ngắn sẽ phải đợi rất lâu nếu nó được sắp sau công việc có thời gian thi hành dài hơn.

Giải pháp

- Bảo vệ phần cứng: bảo vệ vùng nhớ và tái định vị vùng nhớ.
- Lập trình đa chương (Multiprogramming): nhiều người dùng có thể chia sẻ cùng một hệ thống.
- Công việc nhỏ có thể nhanh chóng được hoàn thành.
- HĐH phải quản lý tương tác giữa các công việc đồng thời.
- HĐH trở nên một khoa học quan trọng.

•**OS/360**: HĐH đầu tiên thiết kế cho một họ các máy tính: từ máy tính nhỏ nhất đến máy tính lớn nhất.

Vấn đề

•OS/360 được giới thiệu vào năm 1963; và đến năm 1968 nó mới thật sự hoạt động.

Hơn 1000 lỗi khi phát hành

•Hệ thống cực kỳ phức tạp.

•Tất cả đều là mã hợp ngữ (assembly code).

Giai đoạn 2:Máy tính rẻ hơn - nhân công đắt! (1975-1985)

Giúp con người tăng năng suất.

Chia sẻ thời gian: cho phép nhiều người sử dụng máy tính cùng một lúc.

- Thiết bị cuối rẻ: mọi người đều có thể mua.
- Dữ liệu được lưu trữ: dùng các hệ thống file.
- Thử nghiệm cung cấp thời gian phản hồi chấp nhận được (tránh tình trạng tranh chấp tài nguyên; *sự đổ vỡ (thrashing)*).

Thị trường được định hướng bởi các ứng dụng theo chiều dọc.

CTSS:

- Phát triển tại MIT.
- Một trong những hệ thống chia sẻ thời gian đầu tiên.
- Tiên phong trong việc lập lịch
- Khởi nguồn cho MULTICS.

MULTICS:

- Hợp tác phát triển bởi MIT, Bell Labs, General Electric.
- Viễn cảnh: một máy tính hỗ trợ cho mọi người. Mọi người sẽ mua dịch vụ tính toán như là mua điện.
- Rất nhiều ý tưởng!
- Xây dựng rất khó so với dự tính.
- Công nghệ đã bắt kịp.

UNIX:

- Ken Thompson và Dennis Ritchie xây dựng một hệ thống "do Lập trình viên vì lập trình viên".
- Ban đầu được cài đặt bằng hợp ngữ. Được viết lại bằng C sau này.
- Ý tưởng mới: HĐH có thể di chuyển được!
- Các trường ĐH được cung cấp đoạn mã để tham khảo.
- ĐH Berkeley thêm vào hỗ trợ bộ nhớ ảo cho VAX.
- DARPA chọn UNIX làm nền tảng mạng (arpanet).
- UNIX trở thành HĐH thương mại.
- Các ý tưởng quan trọng được phổ biến thông qua UNIX
- HĐH được viết trên ngôn ngữ cấp cao.
- HĐH có thể di chuyển được không phụ thuộc vào nền tảng phần cứng.
- Cơ chế đường ống (pipe)
- Hệ thống file có thể được nạp.

Giai đoạn 3: Máy tính rất rẻ - nhân công rất đắt. (1981 - ...)

Đưa máy tính đến từng trạm đầu cuối!

CP/M: HĐH thương mại đầu tiên.

IBM cần một phần mềm cho PC mới của họ, nhưng CP/M không nắm bắt được thời cơ.

Tiếp cận Bill Gates (Microsoft) để xem họ có thể xây dựng một cái vậy không.

Bill Gates mua 86-DOS, và tạo nên MS-DOS.

Mục đích chính: hoàn thành nhanh và chạy được các chương trình CP/M hiện hành.

HĐH trở thành một thư viện gồm các thủ tục con và các lệnh có thể thi hành được.

Personal workstations

- The PERQ
- The Xerox Alto
- The SUN Workstation (Stanford University Network)

Personal computers

- The Apple II
- The IBM PC
- The Macintosh

Ứng dụng trong công nghiệp

- Word processors
- Spreadsheets
- Databases

Thị trường chia theo chiều ngang

- Hardware
- Operating systems
- Applications

Graphical User Interfaces

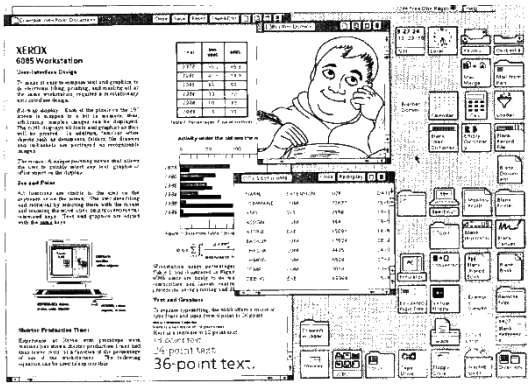
- Xerox Star: 1981
  - Chuột, windows
- Apple Lisa/Machintosh: 1984
  - “Look and Feel” suit 1988
- Microsoft Windows:
  - Win 1.0 (1985)
  - Win 3.1 (1990)
  - Win 95 (1995)
  - Win NT (1993)
  - Win 2000 (2000)
  - Win XP (2001)

Single Level

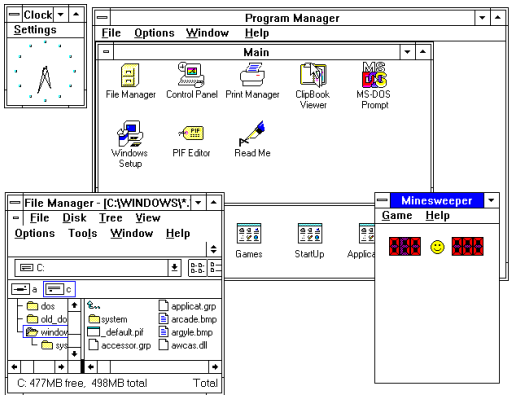
HAL/Protection

No HAL/ Full Prot

Xerox Star



Windows 3.1



Giai đoạn 4: Mạng xuất hiện!

Việc kết nối trở nên quan trọng, bức thiết.

Người ta muốn chia sẻ *dữ liệu* chứ không phải phần cứng.

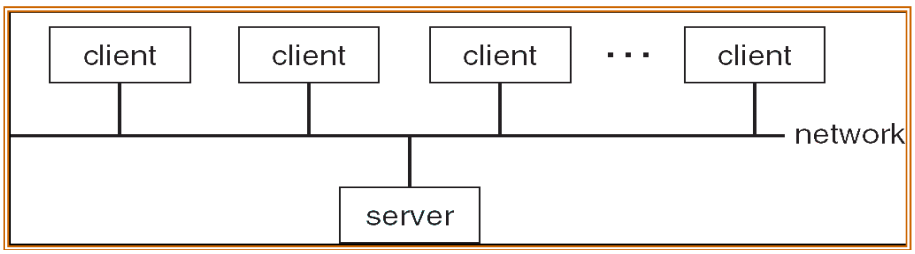
Ứng dụng mạng đẩy đến cho nền công nghiệp.

- WWW
- Email
- News

Việc bảo vệ và lập trình ***đa chương*** trở nên kém quan trọng cho các máy tính cá nhân. Nó quan trọng hơn cho các máy tính chủ (***server***)

Thị trường tiếp tục phân hóa theo chiều ngang:

- Cung cấp dịch vụ Internet
- Thông tin trở thành một phương tiện
- Quảng cáo trên máy tính.



HĐH ngày nay

Hệ thống lớn và phức tạp với vô số vấn đề.

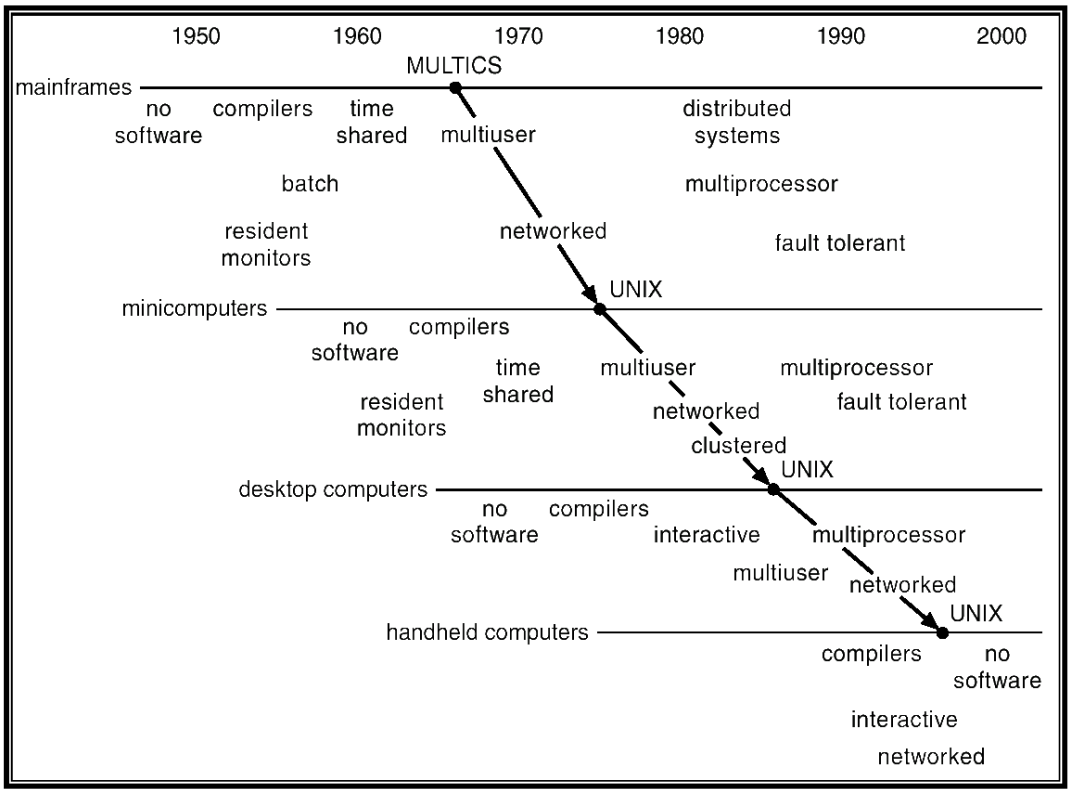
- hàng triệu dòng lệnh.
- hàng trăm, hàng ngàn người phát triển.

Tương tác phức tạp

- Không đồng bộ.
- Chạy trên mọi nền tảng.
- Phân lớp người dùng theo nhu cầu.
- Hiệu năng sử dụng!

Khó hiểu

- Không còn như nguyên bản được tạo ra.
- Quá lớn để cho một người có thể nắm bắt được.
- Không được kiểm lỗi đầy đủ (OS/360 phát hành với 1000 lỗi).
- Khó dự đoán hành động; tối ưu chủ yếu dựa vào cảm tính (đoán).



Phân loại hệ điều hành

- Theo số người dùng
  - Single và Multi user
- Theo số tác vụ
  - Single và Mutil task
- Theo giao diện
  - Command line và Graphic

Câu Hỏi

- Trình bày ngắn gọn về các giai đoạn phát triển của hệ điều hành.
- Liệt kê các hệ điều hành nhiều dùng và nhiều tác vụ. Win XP là hệ điều hành mutil user- mutil task như Linux đúng không?
- Liệt kê các lệnh cơ bản trong hệ điều hành Linux.

Tóm lại

- Không hệ điều hành
- Batch monitor
  → Multiprogramming
- Time sharing
  → Graphical UI
- Thư viện API của Hệ Điều Hành (như Windows API)
- Network
- PDA
- Kiến trúc của hệ điều hành MS-DOS?
- Bài tập nghiên cứu: tìm hiểu và so sánh tập lệnh trong MS-DOS và LINUX.

Tiến trình

Trần Sơn Hải

Khoa Toán – Tin học

Đại học Sư phạm TPHCM

Heavily reference to Operating System Slide of Hoang Than Anh Tuan, University of Pedagogy, HCMC

Nội dung

- Khái niệm đa chương
- Tiến trình
- Luồng (tiểu trình)
- Không gian địa chỉ
- Sự liên quan giữa tiến trình, tiểu trình và không gian địa chỉ
- Lập lịch tiến trình

Tiến trình là gì?

- ***Tiến trình*** = một thể hiện của việc thi hành một chương trình
  - Thường gọi là “HeavyWeight Process”
- Tiến trình là một sự trừu tượng hóa cung cấp bởi HĐH, chỉ ra những gì là cần thiết để thi hành một chương trình
  - Một ngữ cảnh tính toán tách biệt cho mỗi ứng dụng
- Ngữ cảnh tính toán
  - Trạng thái CPU + không gian địa chỉ + môi trường

Tiến trình là gì? (tt)

- Tiến trình thường gồm có hai phần:
  - Một dãy các lệnh mà nó cần phải thi hành
    - Code
    - Trạng thái CPU
  - Các tài nguyên của riêng nó
    - Trạng thái bộ nhớ chính (không gian địa chỉ)
    - Trạng thái nhập xuất (file đang thao tác)

Trạng thái CPU=Nội dung các thanh ghi

- Word trạng thái tiến trình (PSW)
  - chế độ thi hành, kết quả lệnh cuối cùng, mức ngắt
- Thanh ghi lệnh (IR)
  - Chỉ thị hiện tại đang được thi hành
- Bộ đếm chương trình (PC)
- Con trỏ ngăn xếp (SP)
- Các thanh ghi công dụng chung

Không gian địa chỉ

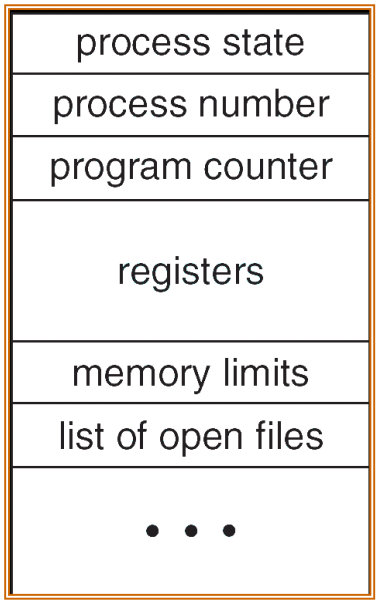
- Data
  - Dữ liệu định trước (thời điểm biên dịch)
- Code
  - Đoạn mã chương trình
- Heap
  - Dữ liệu được cấp phát động
- Stack
  - Hỗ trợ lời gọi hàm

Môi trường

- Các thực thể bên ngoài
  - Thiết bị đầu cuối
  - File đang mở
  - Kênh kết nối
    - Cục bộ
    - Với các máy khác

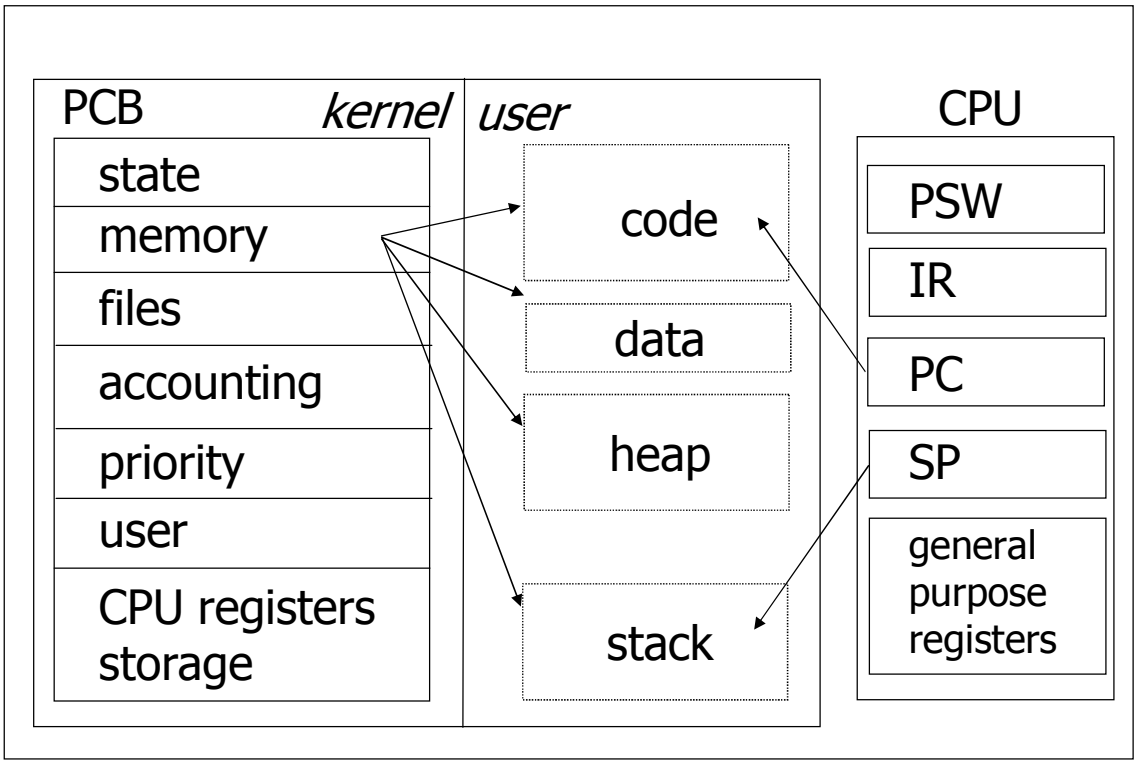
Khối điều khiển tiến trình (PCB)

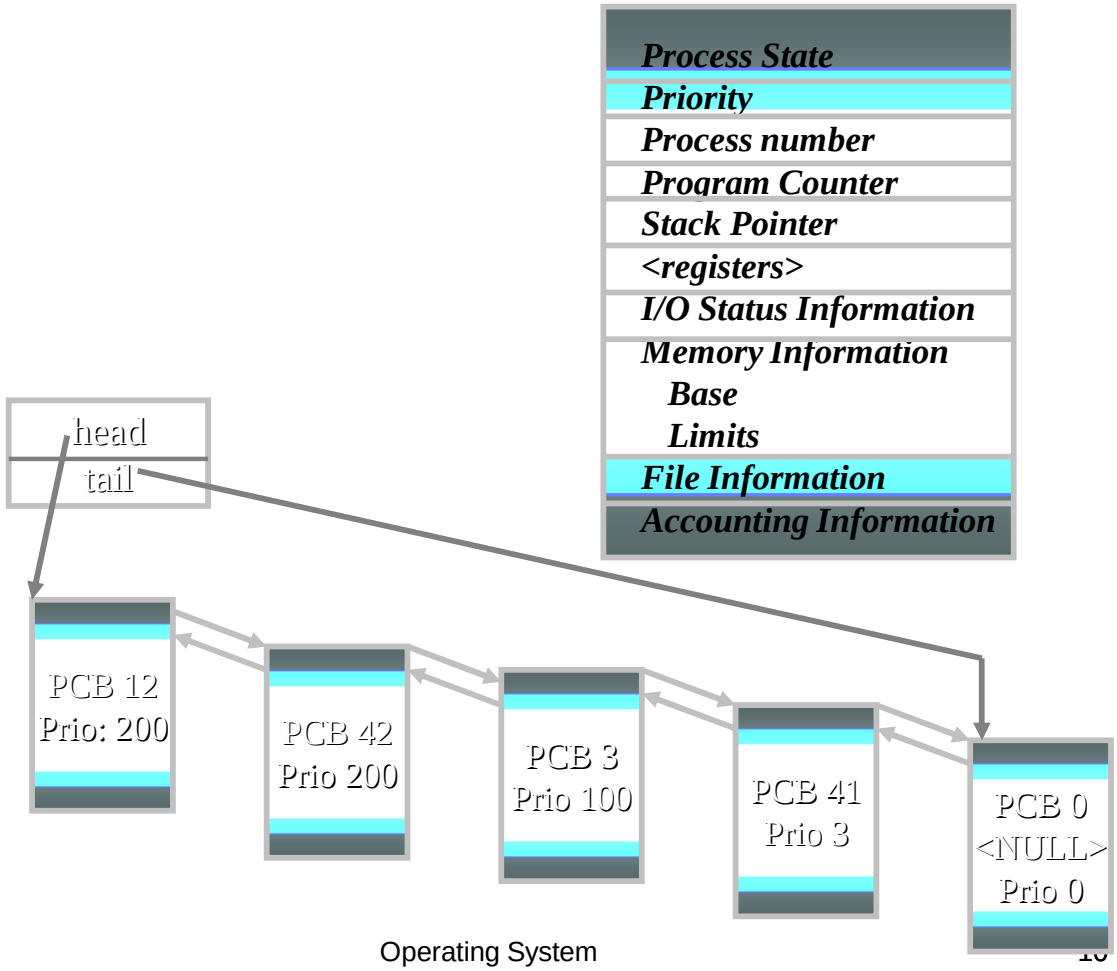
- Ngữ cảnh tính toán của mỗi tiến trình được lưu trong một ***khối điều khiển tiến trình (Process Control Block: PCB)***



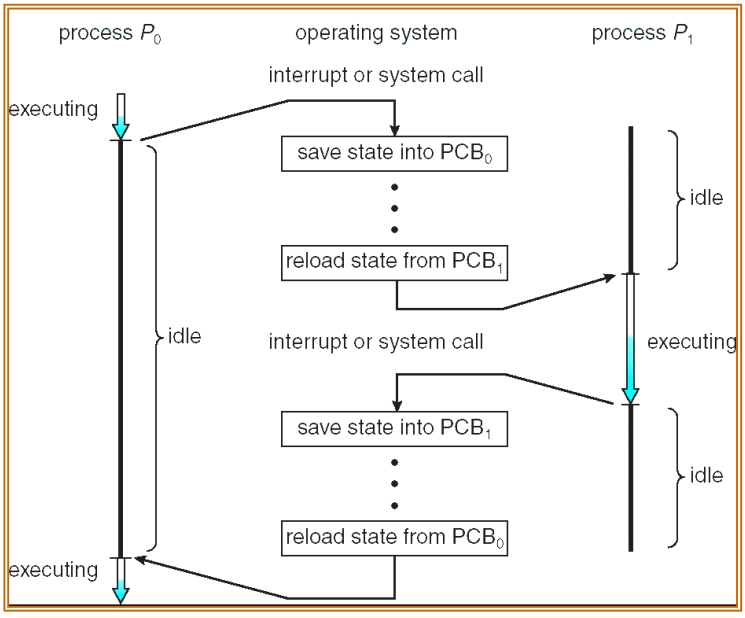
Process Control Block

Process control block (PCB)



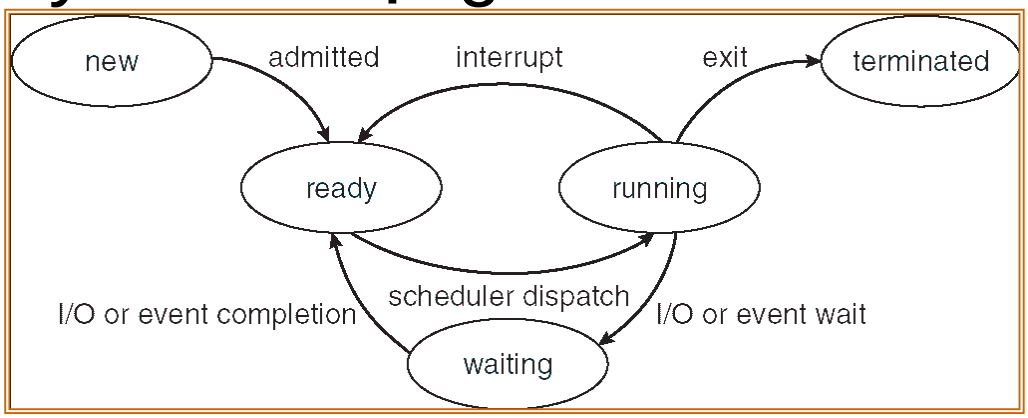


Chuyển đổi ngữ cảnh tiến trình



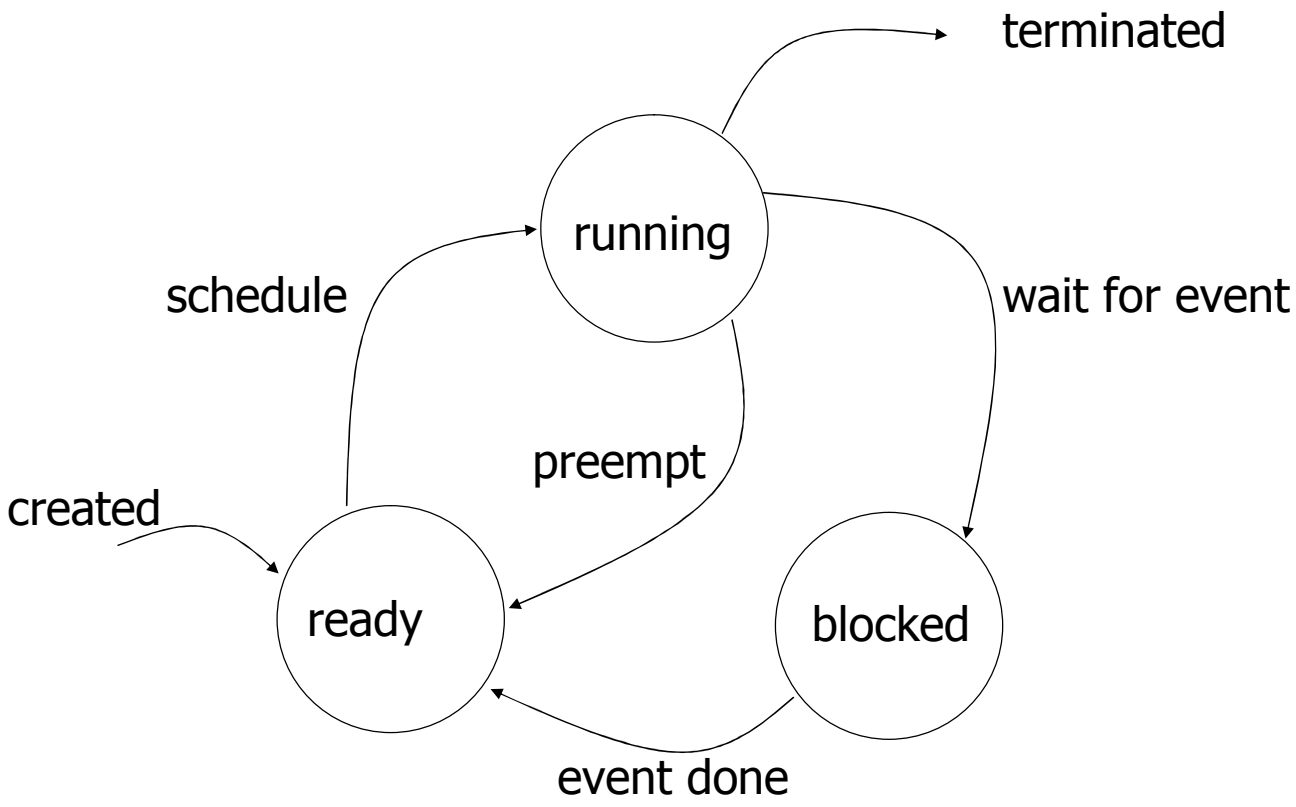
- CPU chuyển đổi tiến trình này sang tiến trình khác
- ***Chuyển đổi ngữ cảnh (context switching) → overhead***
- Trạng thái của tiến trình luôn thay đổi

Chuyển đổi trạng thái của tiến trình

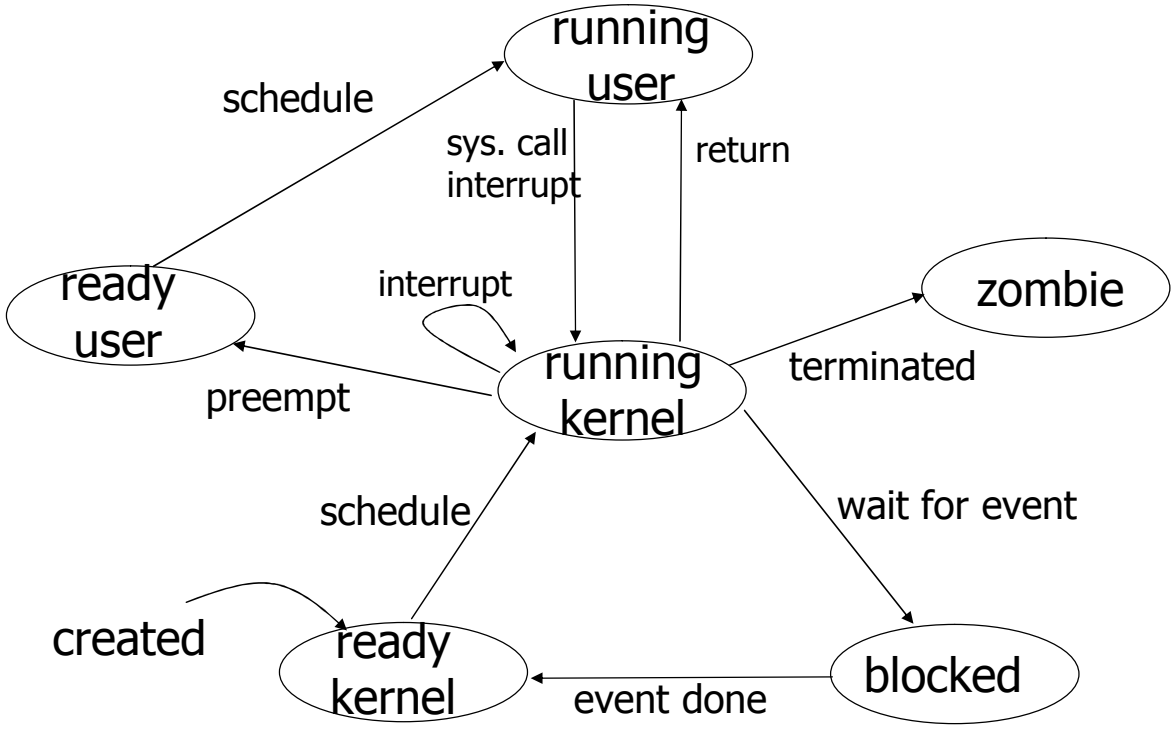


- ***Trạng thái của tiến trình***
  - **new**: Tiến trình vừa được tạo (chạy chương trình)
  - **ready**: Tiến trình sẵn sàng để chạy
  - **running**: Tiến trình đang chạy (thi hành lệnh)
  - **waiting**: Tiến trình chờ đợi một sự kiện
  - **terminated**: Tiến trình kết thúc thi hành lệnh

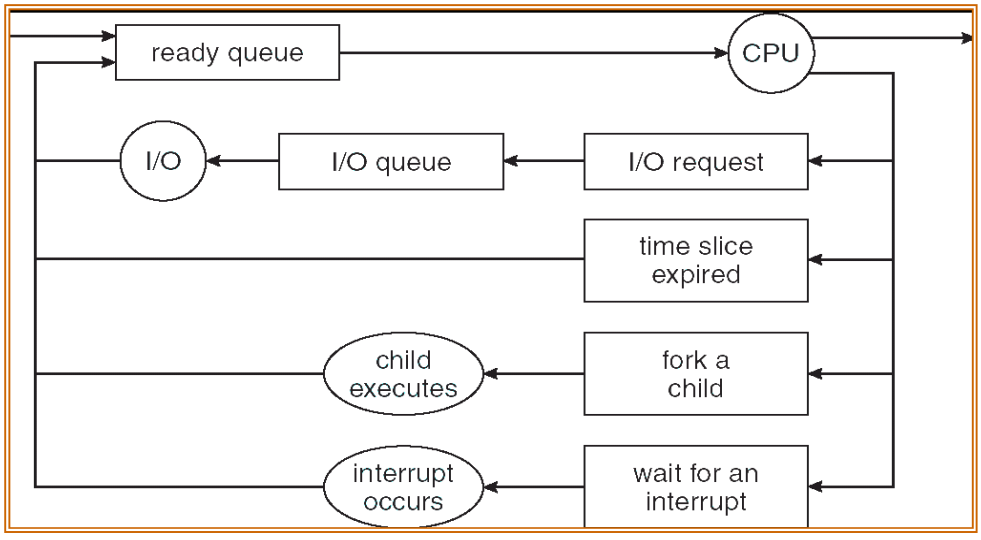
Trạng thái của tiến trình (khác)



Môi trường UNIX

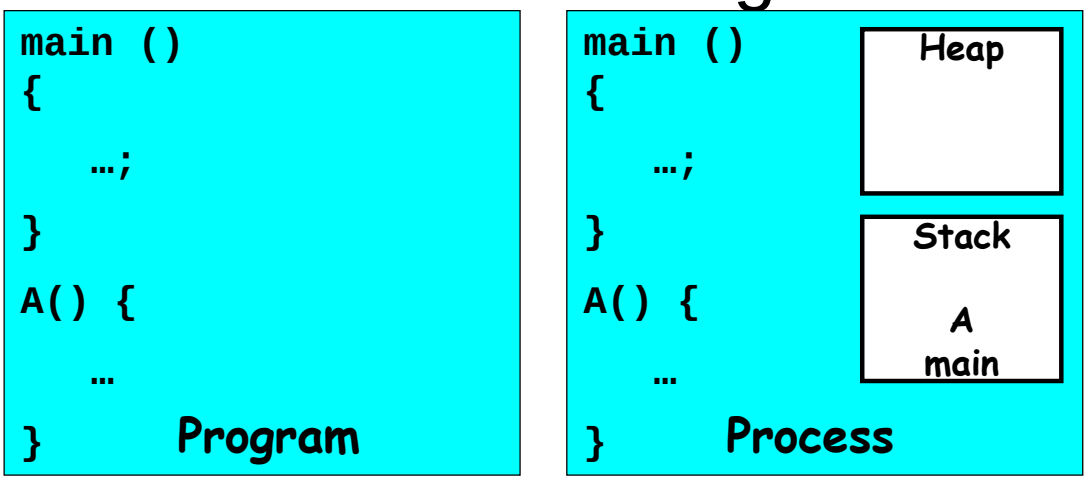


Điều phối tiến trình



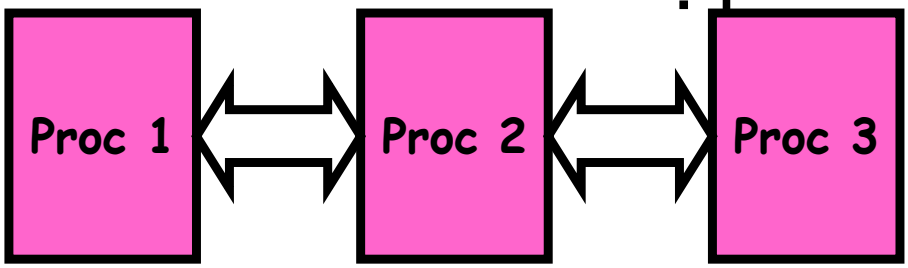
- Có nhiều hàng đợi:
  - ready queue: hàng đợi chứa các tiến trình sẵn sàng chạy
  - I/O queue: hàng đợi chứa các tiến trình sẵn sàng thi hành I/O
- Lựa chọn tiến trình nào → ***điều phối tiến trình***

Tiến trình = Chương trình ?



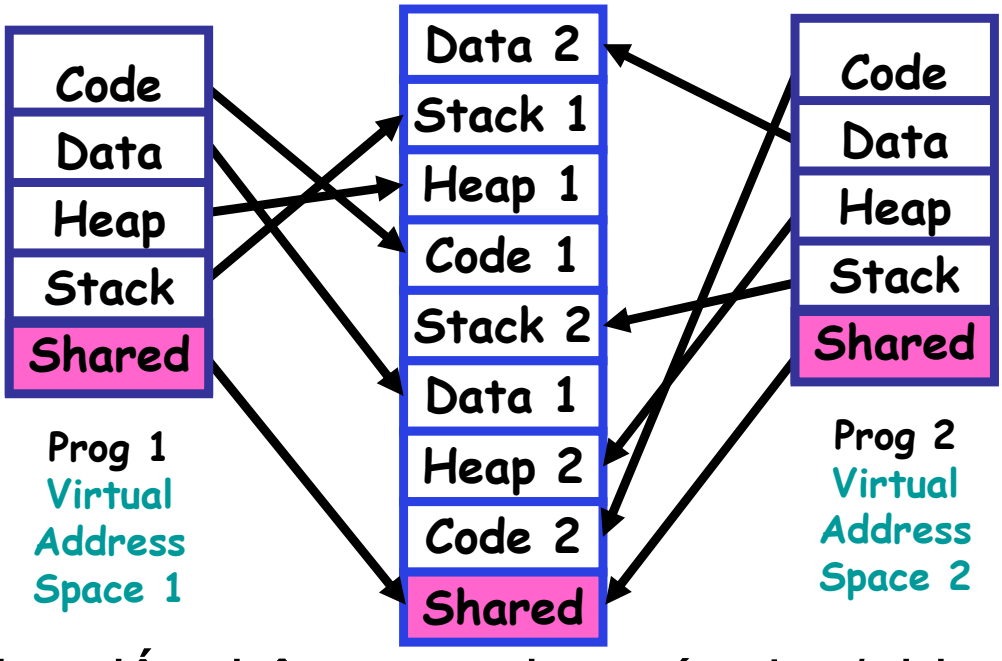
- Một chương trình có thể có nhiều tiến trình
  - Mở *Notepad.exe* xem file a.txt → 1 tiến trình.
  - Mở *Notepad.exe* xem file b.txt → 1 tiến trình.
- Chương trình nhìn từ góc độ mã lệnh chỉ là một phần của tiến trình.

Nhiều tiến trình hợp tác



- Có những công việc cần có nhiều tiến trình hợp tác với nhau để hoàn thành.
- Thời gian để tạo tiến trình
  - Tạo khối PCB
  - Tạo không gian địa chỉ
- Thời gian chuyển đổi các tiến trình
- Cần có một cơ chế giao tiếp:
  - Tách biệt không gian địa chỉ của các tiến trình với nhau
  - Ánh xạ vùng nhớ chia sẻ
  - Truyền thông điệp
    - `send()` và `receive()`

Giao tiếp bằng Shared-Mem



- Giao tiếp thông qua thao tác đọc/ghi trên vùng nhớ chung

Giao tiếp giữa các tiến trình (IPC)

- (Inter-Process Communication) Cơ chế cho phép các tiến trình giao tiếp với nhau và đồng bộ hóa hành động của chúng
- Hệ thống thông điệp – các tiến trình giao tiếp với nhau không cần phải qua các biến dùng chung
- IPC cung cấp hai thao tác cơ bản:
  - **send**(*message*)
  - **receive**(*message*)
- Nếu tiến trình *P* và *Q* muốn giao tiếp với nhau, chúng phải:
  - tạo một đường giao tiếp giữa chúng
  - trao đổi các thông điệp thông qua send/receive

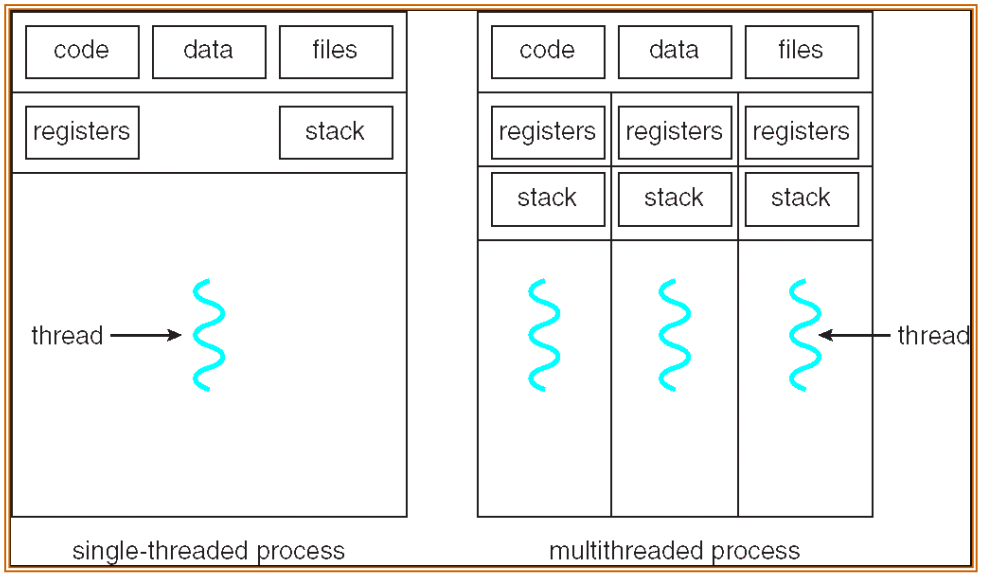
Tiểu trình (Thread = LightWeight Process)

- Tiểu trình (luồng): một sự thi hành (hoạt động) bên trong một tiến trình
- Một tiến trình đa luồng bao gồm nhiều hoạt động cùng tồn tại và thi hành
- Tách biệt:
  - Trạng thái CPU, ngăn xếp
- Chia sẻ:
  - Mọi thứ khác
    - Data, Code, Heap, môi trường
  - Đặc biệt: Không gian địa chỉ (Tại sao?)

Tiểu trình (tt)

- MultiThreading = một chương trình được tạo ra bằng một số các hoạt động đồng thời.
- HeaveWeight Process = Tiến trình với duy nhất một tiểu trình

Đơn tiểu trình và đa tiểu trình



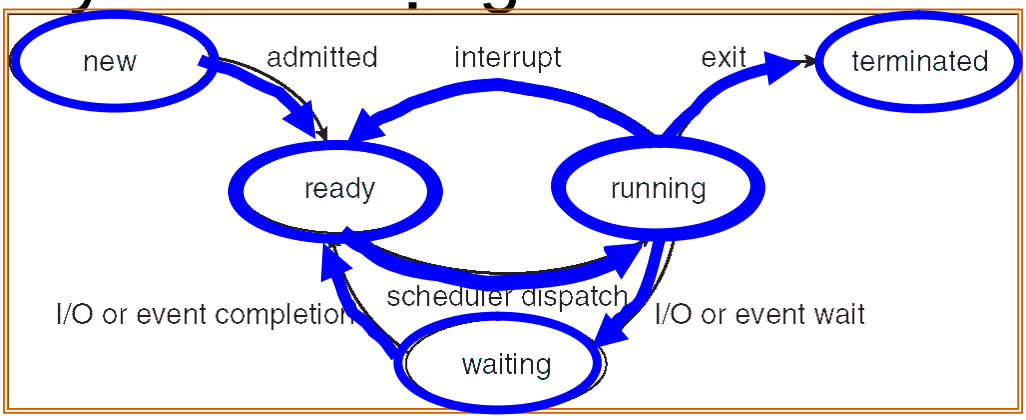
Một số ví dụ về chương trình đa tiểu trình

- Database server:
  - Nhiều kết nối và cơ sở dữ liệu cùng một lúc
- Network Server:
  - Truy cập đồng thời từ môi trường mạng
  - Một tiến trình – nhiều thao tác đồng thời
  - File Server, Web server, ...
- Paralell Programming (có nhiều CPU)
  - Chia chương trình thành nhiều thread để tận dụng nhiều CPU
  - Còn gọi là Multi - Processing

Hỗ trợ tiểu trình

- HĐH
  - Ưu điểm: lập lịch tiểu trình được thực hiện bởi OS
    - Tối ưu hóa CPU
  - Khuyết điểm: nhiều tiểu trình → overhead
- Mức người dùng
  - Ưu điểm: overhead thấp
  - Khuyết điểm: OS không nhận ra cụ thể
    - VD: một tiểu trình bị block do I/O sẽ block tất cả các tiểu trình khác cùng một tiến trình

Chuyển đổi trạng thái của Thread



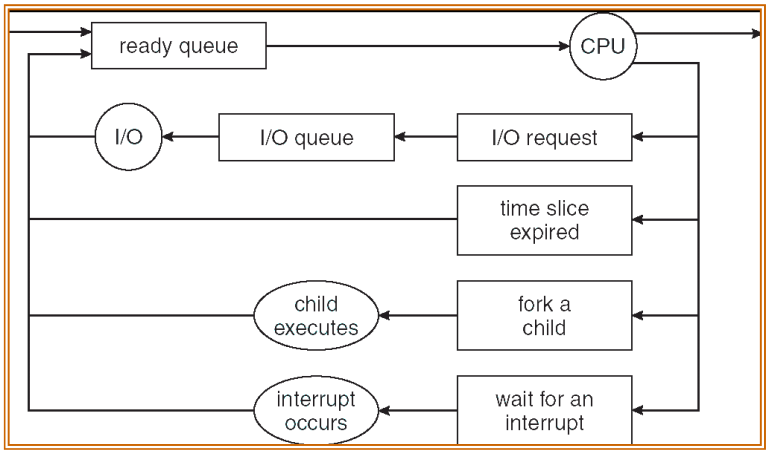
- Tương tự như tiến trình:
  - new: Tiểu trình được tạo mới
  - ready: Tiểu trình đang chờ để chạy
  - running: Tiểu trình đang được thi hành
  - waiting: Tiểu trình đang chờ sự kiện
  - terminated: Tiểu trình kết thúc thi hành
- Thông tin tiểu trình lưu trong TCB

Lập trình đa chương

- *Lập trình đa chương*: Có nhiều công việc (tiến trình, tiểu trình) trong hệ thống
  - Tận dụng thời gian chồng nhau của CPU và I/O
  - Chia sẻ thời gian trên một CPU
  - Thi hành đồng thời trên nhiều CPU
  - Kết hợp cả hai
- Ưu điểm?
  - Tính phản hồi nhanh, tối ưu hóa, thi hành đồng thời
- Khuyết điểm?
  - Overhead, phức tạp

Lập lịch các tiến trình (STS)

- Tình huống:
  - Có nhiều tiến trình nhưng tại một thời điểm chỉ có một tiến trình có thể được thực thi (trạng thái là running)
  - Vấn đề: chọn tiến trình nào để thực thi ở bước kế tiếp (từ trạng thái ready chuyển sang trạng thái running)
- ***Lập lịch*** là thao tác quyết định tiến trình nào được quyền thực thi.

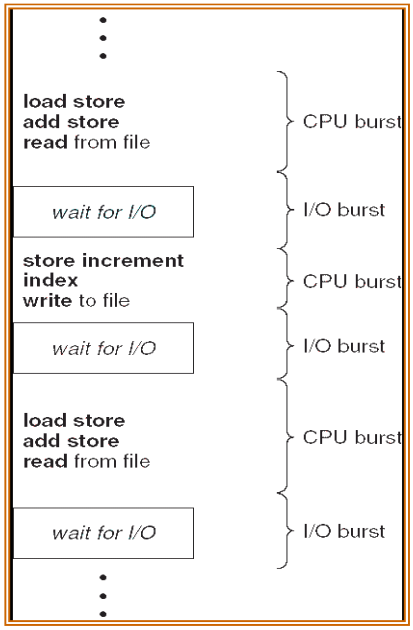


Lập lịch (tt)

- Giả sử:
  - Một tiến trình chỉ có một tiểu trình (HeaveWeight Process)
    - *Lưu ý*: Hệ điều hành lập lịch ở mức tiểu trình
  - Các tiến trình là độc lập với nhau
    - Không có hợp tác, chia sẻ tài nguyên với nhau
    - Các tiến trình hợp tác → đồng bộ hóa tiến trình (chương sau)
  - Mô hình thực thi của các tiến trình là một chuỗi thời gian sử dụng CPU và I/O xen kẽ nhau
    - Chỉ tập trung vào lập lịch cho thời gian CPU

Lập lịch (tt)

- Chương trình sẽ sử dụng CPU trong một khoảng thời gian.
- Sau đó thi hành thao tác I/O
- Tiếp tục sử dụng CPU,...



Tính chất khối việc!

? Có phụ thuộc khối lượng công việc không?

Có:

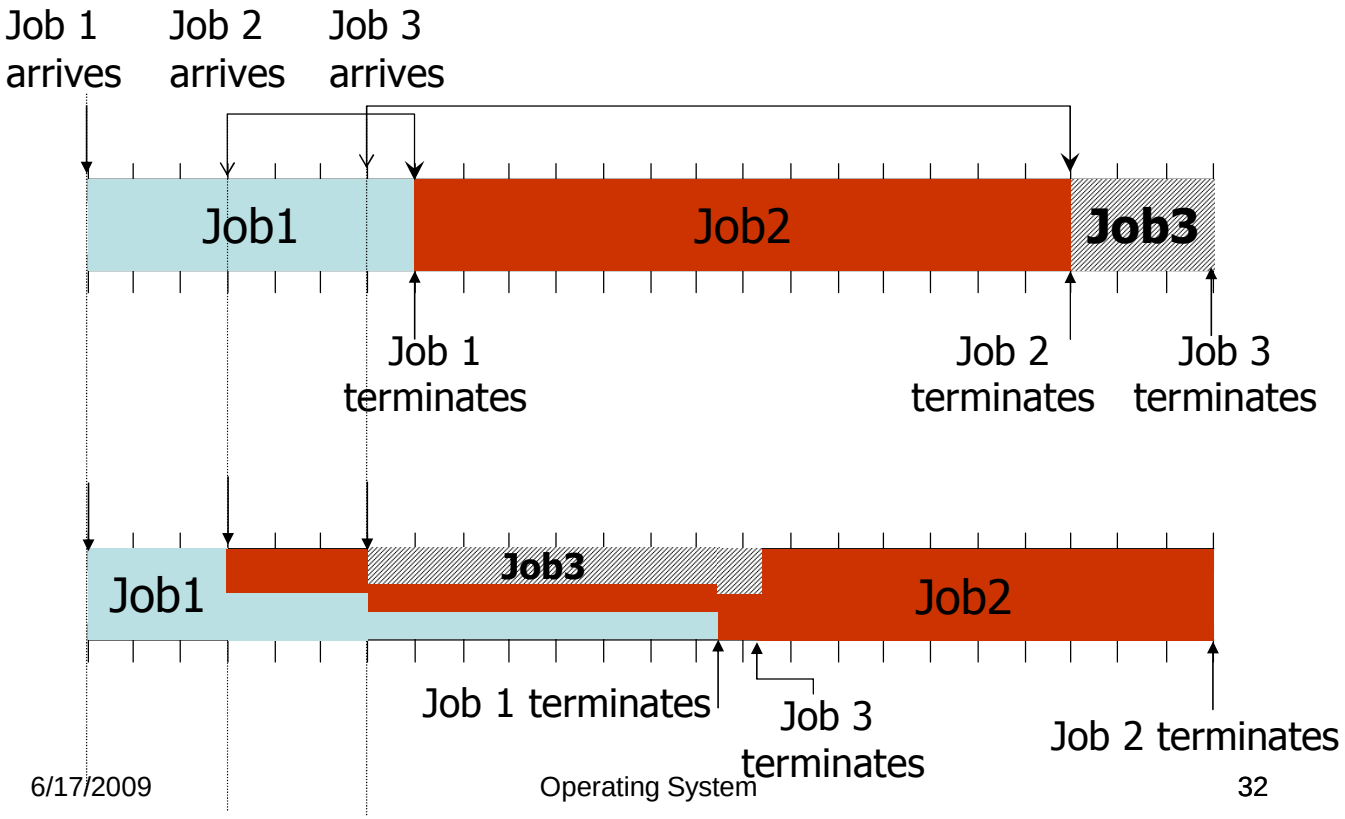
- Nếu mọi công việc đều sử dụng CPU hoặc I/O như nhau thì lập trình đa chương không cải tiến tính tối ưu hoạt động

- Một bộ công việc hỗn hợp sẽ được tạo bởi ***bộ lập lịch dài hạn*** (long-time scheduling)
  - Công việc sẽ được phân định là sử dụng nhiều CPU hoặc nhiều I/O tùy thuộc vào quá trình thi hành trước đó của nó
- Quan tâm vào ***bộ lập lịch ngắn hạn*** (Short-Time Scheduling)

Tiêu chuẩn đánh giá việc lập lịch

- Tối thiểu hóa ***thời gian phản hồi*** (response time)
- Tối thiểu hóa ***thời gian lưu lại trong hệ thống*** (turnaround time)
  - $T_{turn\ around} = T_{kết\ thúc} - T_{bắt\ đầu}$
- Tối đa hóa ***throughput***
  - *Throughput = số công việc xử lý trong một đơn vị thời gian*
- ***Sự công bằng*** (Fairness)
  - Các tiến trình đều được đối xử công bằng. Không có tiến trình nào phải chờ quá lâu
- Tối ưu hóa CPU (Efficient)

Turn around time

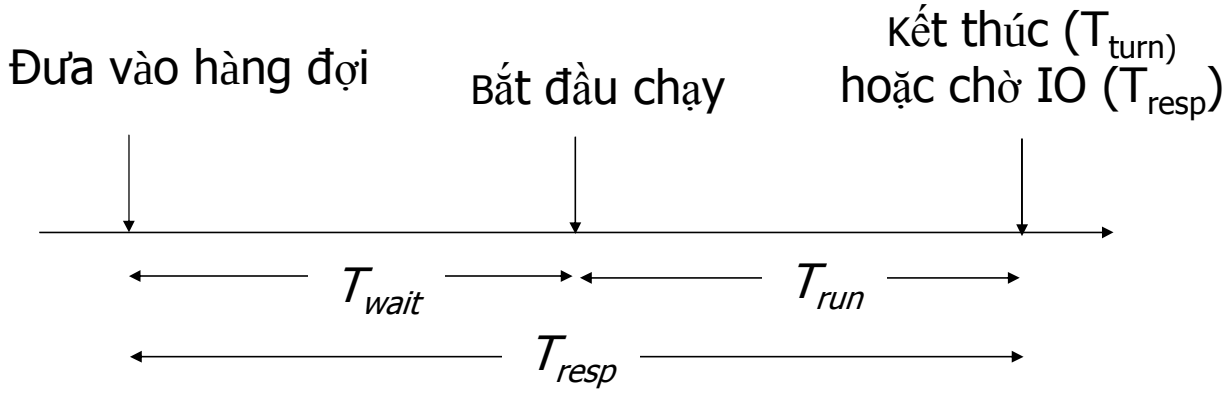


Thời gian chạy (T_{run})

- Thời gian chạy = thời gian sử dụng CPU
  - Không tính thời gian chờ IO dù thật sự lúc đó chương trình vẫn "đang chạy"
- Bởi vì theo STS: các tiến trình phụ thuộc nhiều vào I/O thường có thời gian khá ngắn
  - Trình soạn thảo!!!

Các độ đo thông dụng

- Thời gian phản hồi (response time) <> Thời gian lưu lại hệ thống (turnaround time)
- **Tỉ số phản hồi** *slowdown=*T_{turn} / T_{run}



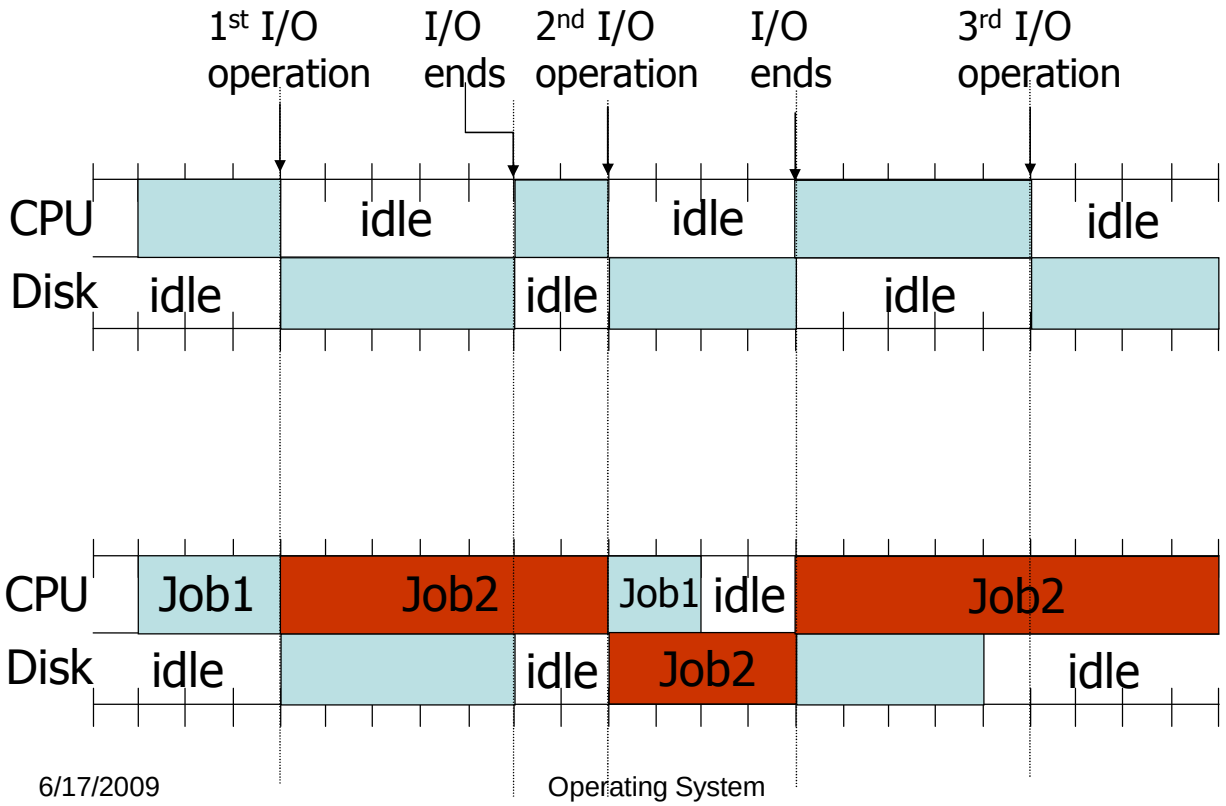
$T_{resp} = T_{wait} + T_{run}$

Các độ đo khác

- Thời gian chờ đợi: giá trị trung bình của T_{wait}
- Tỉ số phản hồi (slowdown)

  $slowdown = T_{resp} / T_{run}$
- Throughput: cực đại hóa số tiến trình hoàn thành trong một đơn vị thời gian → phụ thuộc vào chi tiết cụ thể của tiến trình → ít hữu dụng.

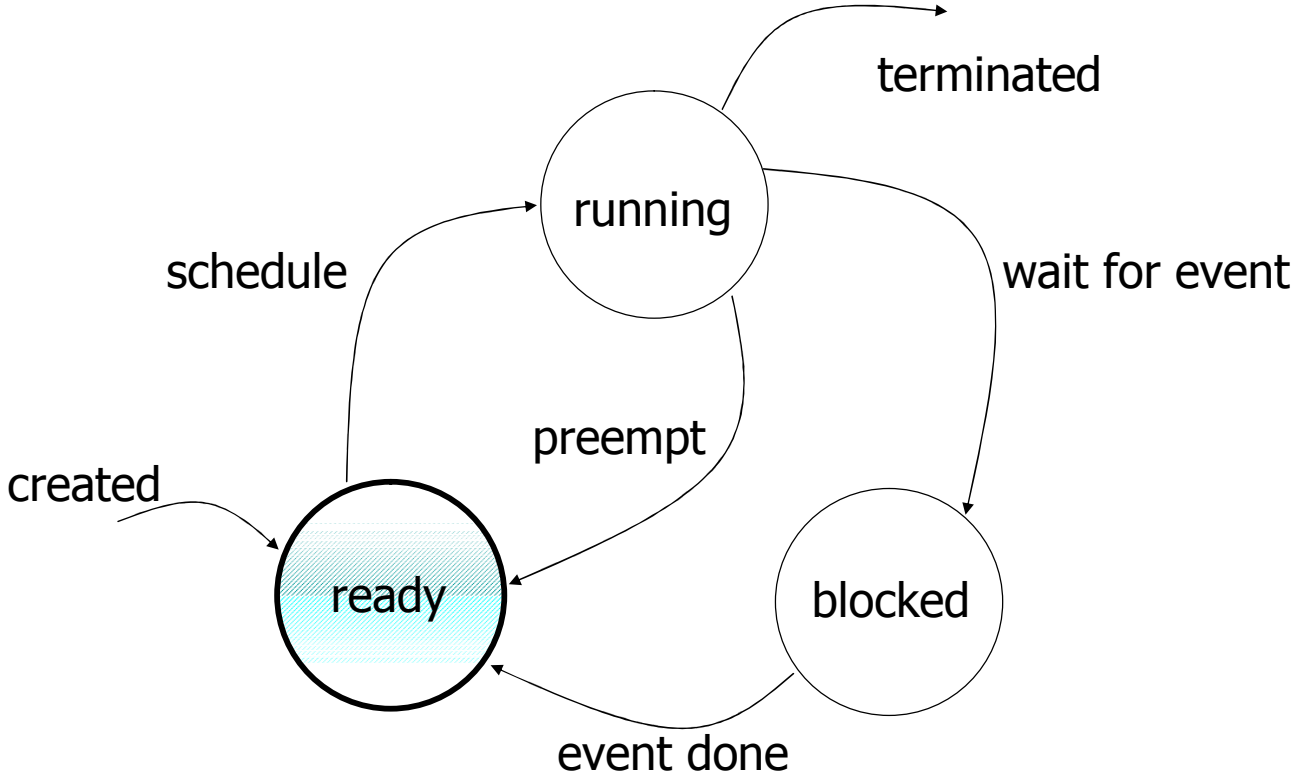
Tối ưu hóa CPU



Chi phí cho lập trình đa chương

- Overhead cho sự chuyển đổi
  - Lưu và phục hồi ngữ cảnh sẽ làm lãnh phí CPU
- Giảm hiệu năng
  - Tạm bằng lòng với tài nguyên (ít hơn mong muốn)
  - Cache → không hiệu quả
- Phức tạp
  - Sự đồng bộ hóa, điều khiển song song, tránh deadlock, cơ chế bảo vệ

Lập lịch ngắn hạn (STS)



Lập lịch ngắn hạn (STS) (tt)

- Mẫu thi hành tiến trình bao gồm sự chuyển đổi liên tiếp giữa sử dụng CPU và chờ đợi IO
  - CPU – I/O – CPU – I/O...
- Các tiến trình sẵn sàng để thi hành được lưu trong một hàng đợi sẵn sàng (chạy) (*ready* (*run*) *queue)*
- STS lập lịch cho các tiến trình từ hành đợi sẵn sàng một khi CPU chuyển sang trạng thái rảnh.

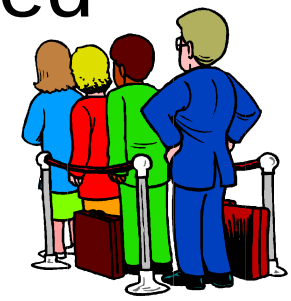
Lập lịch Off-line vs. On-line

- Thuật toán Off-line
  - Lấy tất cả thông tin về tất cả các công việc cần phải lập lịch (biết trước tương lai)
  - Cho ra trình tự đã được lập lịch
  - Không cần cơ chế cưỡng ép
- Thuật toán On-line
  - Công việc xuất hiện vào những thời điểm không thể đoán trước (không biết trước tương lai).
  - Rất ít thông tin
  - Cần cơ chế cưỡng ép

First-Come, First-Served (FCFS)

- Lập lịch các công việc theo thứ tự xuất hiện của chúng.
  - Off-line FCFS lập lịch theo thứ tự xuất hiện trong dữ liệu đầu vào của nó
- Thi hành lần lượt mỗi công việc cho đến khi hoàn thành
  - Nguyên thủy: hoàn thành kể cả tính I/O
  - Hiện đại: dừng lại khi bị block (gặp I/O)
- Có cả on-line lẫn off-line
- Đơn giản, dùng làm cơ sở để phân tích các pp khác
- Thời gian phản hồi kém

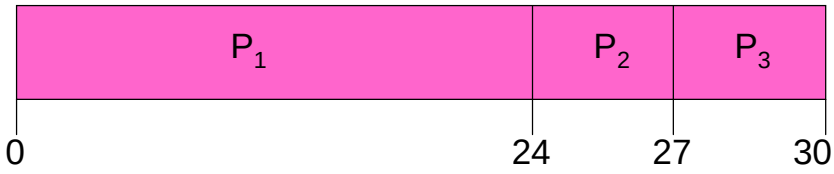
First-Come, First-Served



- Ví dụ:

Process	Burst Time
P_1	24
P_2	3
P_3	3

  - VD: 3 tiến trình vào hàng đợi theo thứ tự: P_1 , P_2 , P_3
    Sơ đồ Gantt:



  - Thời gian chờ P_1 = 0; P_2 = 24; P_3 = 27
  - Thời gian chờ trung bình: (0 + 24 + 27)/3 = 17
  - Thời gian hoàn thành trung bình: (24 + 27 + 30)/3 = 27
- *Điểm yếu:* Các tiến trình có thời gian CPU ngắn vào sau tiến trình có thời gian CPU dài.

First-Come, First-Served

- Điểm yếu:
  - Giả sử vào hàng đợi theo thứ tự: P_2 , P_3 , P_1
    Sơ đồ Gantt:

  - Thời gian chờ P_1 = 6; P_2 = 0; P_3 = 3
  - Thời gian chờ trung bình: (6 + 0 + 3)/3 = 3
  - Thời gian hoàn thành trung bình: (3 + 6 + 30)/3 = 13
- Trường hợp 2:
  - Thời gian chờ trung bình tốt hơn (3 < 17)
  - Thời gian hoàn thành trung bình tốt hơn (13 < 27)

Round Robin (RR)

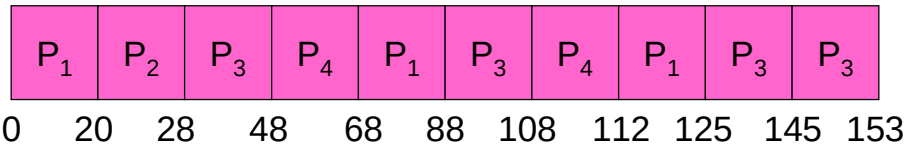
- Mô hình FCFS: Không tốt cho những tiến trình thời gian ngắn!
  - Phụ thuộc hoàn toàn vào thứ tự
- ***Mô hình Round Robin***
  - Mỗi tiến trình sẽ nhận được một khoảng thời gian sử dụng CPU khá nhỏ (*time quantum*), thường là 10-100 milli giây
  - Sau khi khoảng thời gian này kết thúc, tiến trình sẽ bị cưỡng chế chuyển vào hàng đợi sẵn sàng (không cho dùng CPU nữa).
  - Giả sử có *n* tiến trình trong hàng đợi và time quantum là $q \Rightarrow$
    - Mỗi lần chạy tiến trình sẽ có tối đa *q* đơn vị thời gian
    - Không có tiến trình nào phải đợi quá *(n-1)q* đơn vị thời gian
- Đánh giá hiệu năng
  - *q* lớn $\Rightarrow$ FCFS
  - *q* nhỏ $\Rightarrow$ thời gian overhead lớn → không hiệu quả

Round Robin (q=20)

- Ví dụ:

Process	Burst Time
P_1	53
P_2	8
P_3	68
P_4	24

  - Sơ đồ Gantt:



  - Thời gian chờ
    P_1=(68-20)+(112-88)=72
    P_2=(20-0)=20
    P_3=(28-0)+(88-48)+(125-108)=85
    P_4=(48-0)+(108-68)=88
  - Thời gian chờ trung bình = (72+20+85+88)/4=66¼
  - Thời gian hoàn thành trung bình = (125+28+153+112)/4 = 104½
- Đánh giá:
  - Tốt cho các tiến trình có thời gian CPU ngắn
  - Thêm thời gian chuyển đổi ngữ cảnh cho các tiến trình có thời gian CPU dài

Câu Hỏi

- Ví dụ:

Process	Burst Time
P_1	53
P_2	8
P_3	68
P_4	24

  - Sơ đồ Gantt?
  - Thời gian chờ P_1?
    P_2 ?
    P_3?
    P_4?
  - Thời gian chờ trung bình ?
  - Thời gian hoàn thành trung bình ?
- Khi dùng Round Robin với q =40, First Come- First Service và FCFS trong trường hợp xấu nhất với 4 process trên.

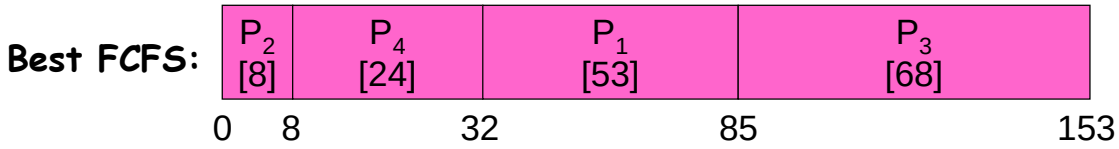
So sánh FCFS và Round Robin

- Giả sử thời gian chuyển đổi ngữ cảnh không đáng kể, RR hay FCFS tốt hơn?
- Xét ví dụ: 10 tiến trình, mỗi tiến trình sử dụng 100s CPU
  q = 1s
  Tất cả tiến trình vào hàng đợi cùng 1 thời điểm
- Thời gian hoàn thành:

P #	FIFO	RR
1	100	991
2	200	992
…	…	…
9	900	999
10	1000	1000

  - Cả RR và FCFS đều hoàn thành 10 tiến trình tại cùng 1 thời điểm
  - Thời gian phản hồi của RR rất tệ!
    - Không nên dùng trong trường hợp các tiến trình có thời gian sử dụng CPU gần nhau

Round Robin với q khác nhau



	Quantum	P_1	P_2	P_3	P_4	Average
Wait Time	Best FCFS	32	0	85	8	31¼
	Q = 1	84	22	85	57	62
	Q = 5	82	20	85	58	61¼
	Q = 8	80	8	85	56	57¼
	Q = 10	82	10	85	68	61¼
	Q = 20	72	20	85	88	66¼
	Worst FCFS	68	145	0	121	83½
Completion Time	Best FCFS	85	8	153	32	69½
	Q = 1	137	30	153	81	100½
	Q = 5	135	28	153	82	99½
	Q = 8	133	16	153	80	95½
	Q = 10	135	18	153	92	99½
	Q = 20	125	28	153	112	104½
	Worst FCFS	121	153	68	145	121¾

Sự cưỡng chế (Pre-emptive)

- ***Sự cưỡng chế*** là hành động dừng một công việc đang chạy để lập lịch cho công việc khác
- → chuyển đổi ngữ cảnh
  - Ví dụ: Tiến trình P1 đang chạy (sử dụng CPU)→ dừng tiến trình P1 lại (chuyển ra hàng đợi ready) và giao CPU cho tiến trình P2 nào đó.
  - Lưu ý: Tiến trình P1 không bị dừng bởi thao tác I/O hoặc các sự kiện khác

Sử dụng sự cưỡng chế

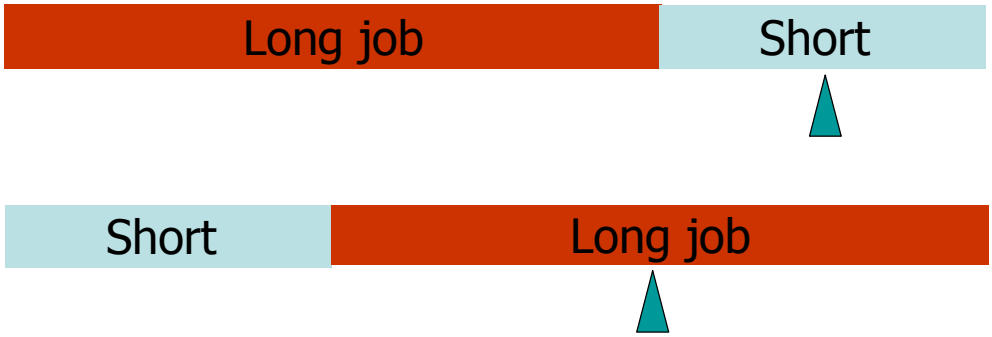
- Thuật toán SST on-line
  - Tương thích với sự thay đổi điều kiện
    - Vd: có công việc mới
  - Bổ sung cho việc thiếu thông tin
    - Vd: thời gian chạy
- Sự cưỡng chế theo chu kỳ giúp hệ thống nằm trong tầm kiểm soát
- Cải thiện tính công bằng

Các thuật tóan cải tiến FCFS

- Xét trường hợp tốt nhất của FCFS: tiến trình thời gian ngắn vào trước, tiến trình thời gian dài vào sau
- ***Shortest Job First (SJF)***:
  - Chọn tiến trình có thời gian chạy là ít nhất (không phụ thuộc thứ tự vào)
  - Còn gọi là “Shortest Time to Completion First” (STCF)
- ***Shortest Remaining Time First (SRTF)***:
  - Là một phiên bản SJF có cưỡng chế (Preemptive version of SJF): nếu có tiến trình mới vào và thời gian sử dụng CPU ít hơn thời gian còn lại của tiến trình đang chiếm CPU thì dừng tiến trình đang chạy và chuyển quyền cho tiến trình mới vào.
  - Còn gọi là “Shortest Remaining Time to Completion First” (SRTCF)
- Ý tưởng chính
  - Cho phép công việc có thời gian thi hành CPU ngắn ra ngoài CPU càng nhanh càng tốt
  - Kết quả là thời gian phản hồi trung bình sẽ tốt hơn

Shortest Job First (SJF)

- Công việc có thời gian ít nhất sẽ được thi hành trước
- Độ đo thời gian phàn hồi là tốt nhất



- Chỉ có off-line
  - Tất cả các công việc và thời gian thi hành phải được biết trước

Shortest Remaining Time first (SRT)

- Biết: thời gian thi hành công việc
- Không biết: thời điểm công việc bắt đầu (được nạp vào hàng đợi)
- Khi có một công việc mới:

  Nếu thời gian thi hành của nó nhỏ hơn thời gian thi hành còn lại của công việc đang được thi hành hiện tại thì:

  cưỡng chế dừng công việc đang thi hành hiện tại và lập lịch cho công việc vừa được tạo ra

  Ngược lại, tiếp tục công việc hiện tại và chèn công việc mới vào hàng đợi theo thứ tự thời gian còn lại phải thi hành
- Khi công việc hiện tại kết thúc, chọn công việc nằm ở đầu hàng đợi để thi hành

So sánh SJF, SRTF, FCFS, RR

- SJF vs SRTF
  - Tốt nhất để tối thiểu hóa thời gian phản hồi trung bình. (SJF: non-preemptive, SRTF: preemptive)
  - SRTF ít nhất là tương đương với SJF
- SRTF vs FCFS và RR
  - Nếu thời gian sử dụng của các tiến trình là như nhau → SRTF = FCFS
  - Nếu thời gian sử dụng của các tiến trình là biến động lớn → SRTF, RR giúp cho các tiến trình có thời gian ngắn không chờ quá lâu.

So sánh SJF, SRTF, FCFS, RR (tt)

- SRTF có thể làm phát sinh trường hợp ***"đói CPU" (starvation)*** cho các tiến trình có thời gian sử dụng CPU tương đối lâu
  - Ví dụ: Trường hợp các tiến trình có thời gian sử dụng ngắn liên tục được đưa vào. → tiến trình có thời gian sử dụng dài sẽ không được phép sử dụng CPU → ***tình trạng đói CPU (starvation).***
- Cả 4 phương pháp đều yêu cầu phải biết thời gian mà một tiến trình sẽ dùng CPU.
  - Làm sao biết?

So sánh SJF, SRTF, FCFS, RR (tt)

- Dùng SRTF để làm cơ sở đánh giá các phương pháp khác (vì là phương pháp tối ưu) về thời gian phản hồi trung bình.
- Ưu điểm
  - Thời gian phản hồi trung bình của SRTF là tốt nhất.
- Khuyết điểm
  - Phải dự đoán thời gian sử dụng CPU của tiến trình
  - Không công bằng

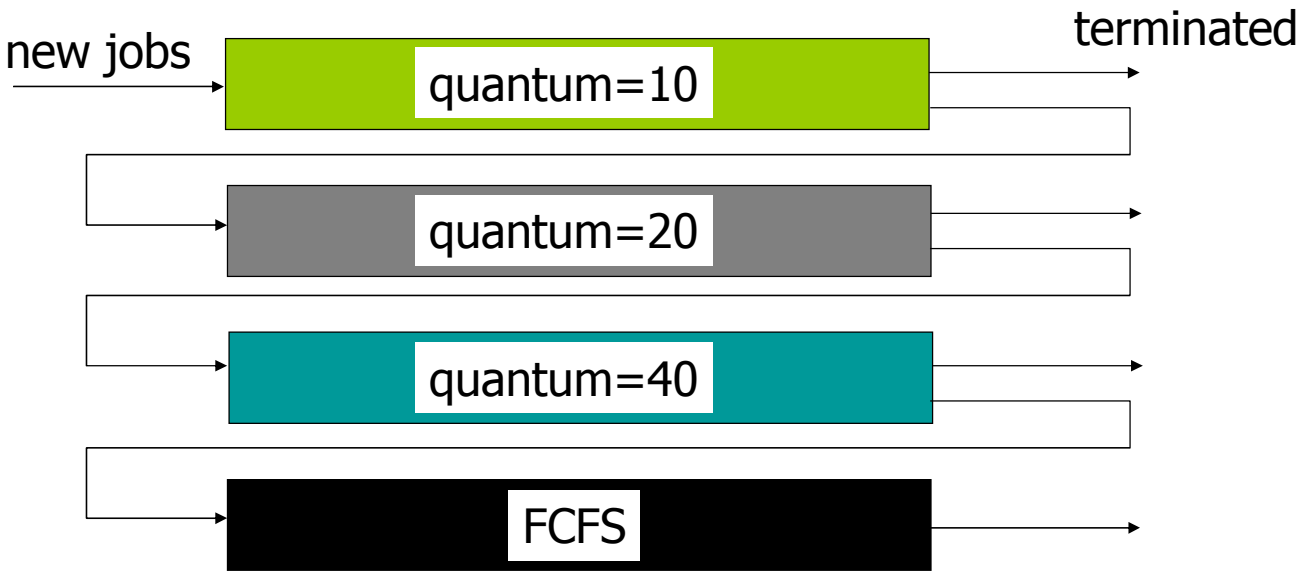
Dự đoán thời gian sử dụng CPU

- Yêu cầu người dùng nhập vào
  - Khó khả thi: người dùng không biết
  - Người dùng có thể đưa vào thời gian thi hành ngắn để mong kết thúc công việc sớm
- ***Adaptive (thích ứng)***: Dự đoán tương lai bằng cách quan sát quá khứ.
  - Nếu trong quá khứ tiến trình (chương trình) thường dùng CPU nhiều (CPU-bound) thì có thể trong tương lai nó sẽ sử dụng nhiều.
  - Nếu trong quá khứ tiến trình (chương trình) thường thao tác I/O (I/O bound) thì có thể trong tương lai nó sẽ sử dụng CPU ít.

Dự đoán thời gian sử dụng CPU (tt)

- Gọi t_i là thời gian sử dụng CPU tại lần thứ i.
- Ý tưởng thời gian sử dụng CPU tại lần thứ n là:
  - $T_n = f(t_{n-1}, t_{n-2}, ...)$
  - $T_n = \alpha t_{n-1} + (1-\alpha)T_{n-1}$ ($\alpha \in [0,1]$)

Hàng đợi phản hồi đa mức (Multilevel feedback queues)



Multilevel feedback queues

- Độ ưu tiên được ngầm định trong mô hình này
- Rất linh hoạt
- Tình trạng đói CPU có thể có

Nhiều công việc ngắn vào => công việc dài sẽ bị "đói"

- Giải pháp:
  - Để nguyên
  - Lão hóa (aging)

FCFS Hiện Đại

- Một tiến trình sẽ chấm dứt hoạt động và chuyển sang trạng thái mới khi hết thời gian hoặc đến thời điểm truy xuất.
- Ví dụ cho 2 tiến trình A(10,2,2) (thời gian hoạt động là 10, thời điểm bắt đầu IO là 2 sau khi bắt đầu tiến trình; trong đó thời gian IO là 2) và B(8,2,2). Hỏi A B ở trạng thái nào theo FCFS lúc 9,32?
- A running và B ready

1	2	3	4	5	6	7	8	9	10
A_R	A_R	A_{IO} B_R	A_{IO} B_R	A_R B_{IO}	A_R B_{IO}	A_R	A_R	A_R	A_R

Round Robin Hiện Đại

- Tại thời điểm m có 2 tiến trình: A running xong q và B IO xong thì thứ tự đưa vào hàng đợi là B trước A sau.
- Tại thời điểm m A running xong q và A đến thời điểm bắt đầu IO thì thời điểm IO sẽ đưa vào chu kỳ sau.
- Ví dụ cho 2 tiến trình A(10,2,2) (thời gian 10 hoạt động, thời điểm bắt đầu IO là 2 sau khi bắt đầu tiến trình; trong đó thời gian IO là 2 và bắt đầu IO) và B(9,3,2). Hỏi A B ở trạng thái nào theo RR với q=2 lúc 9,82?

A running xong q và B IO xong thì thứ tự đưa vào hàng đợi là B trước A sau.
A running xong q và A đến thời điểm bắt đầu IO thì thời điểm IO sẽ đưa vào chu kỳ sau.
Ví dụ A(10,2,2) và B(9,3,2). Trạng thái nào theo RR với q=2 lúc 9,82?
B running và A ready

1	2	3	4	5	6	7	8	9	10
A_R	A_R	B_R	A_{IO} B_R	A_{IO} B_R	A_R B_{IO}	A_R B_{IO}	B_R	B_R	A_R

Tóm tắt

- Tiến trình = một thể hiện của việc thi hành một chương trình.
- Đa chương = nhiều tiến trình có thể cùng được thi hành. Tại mỗi thời điểm chỉ có một tiến trình ở trạng thái được thi hành.
- Lập lịch = quyết định tiến trình nào sẽ được chuyển trạng thái từ sẵn sàng sang chạy.

Tóm tắt (tt)

- FCFS: Vào trước sẽ được cấp phát CPU trước
  - Ưu: đơn giản
  - Khuyết: tiến trình ngắn sẽ chờ tiến trình dài.
- Round Robin: Cấp mỗi tiến trình một khoảng thời gian định trước (quantumn) khi nó nhận được CPU.
  - Ưu: Các tiến trình ngắn sẽ kết thúc nhanh chóng
  - Khuyết: tiến trình có thời gian sử dụng CPU gần nhau → không hiệu quả.

Tóm tắt

- SJF/SRTF: Cấp phát cho tiến trình có thời gian thi hành/thời gian còn lại là ít nhất.
  - Ưu: thời gian phản hồi trung bình là tốt nhất.
  - Khuyết: khó dự đoán thời gian sử dụng CPU, không công bằng.
- Multi-level feedback: sử dụng nhiều hàng đợi với độ ưu tiên khác nhau. Tự động chuyển đổi mức độ ưu tiên của các tiến trình.

Câu Hỏi

- 1/ Cho A (10,0,4) B(8,2,1) và C(9,1,2). Hỏi khi T=11,32 thì A B C ở trạng thái nào theo FCFS.
- 2/ Cho A (10,1,2) B(10,3,2) và C(10,0,2). Hỏi khi T=9,82 thì A B C ở trạng thái nào theo RR với q=2.
- 3/ Bài tập về nhà:viết chương trình input vào số tiến trình, thông tin của từng tiến trình. Output ra trạng thái của các tiến trình tại thời điểm bất kỳ vẽ sơ đồ.

Giao tiếp giữa các tiến trình

1-23,..

38

86

Nội dung

- Tổng quan về giao tiếp tiến trình (tiểu trình)
- Vấn đề Producer/Consumer (sản xuất / tiêu thụ)
- Miền găng
- Đồng bộ bằng giải pháp phần cứng
- Semaphores
- Monitors
- Truyền thông điệp
- Các bài toán cổ điển về đồng bộ hóa

Tổng quan về giao tiếp tiến trình

- *Tiến trình độc lập* không ảnh hưởng và không bị ảnh hưởng bởi việc thực thi của các tiến trình khác.
- *Tiến trình hợp tác (không độc lập)* có thể ảnh hưởng và bị ảnh hưởng bởi việc thực thi của các tiến trình khác.
- Ưu điểm của việc hợp tác tiến trình:
  - Chia sẻ thông tin
  - Tăng tốc tính toán (xử lý song song): thời gian I/O và thời gian CPU
  - Tính module hóa
  - Tiện lợi

Hợp tác bằng việc chia sẻ

- Các tiến trình sử dụng và cập nhập dữ liệu chia sẻ như các biến, file và cơ sở dữ liệu dùng chung.
- Thao tác ghi phải độc lập từng đôi một để ngăn ngừa tình trạng đụng độ, có thể dẫn đến tính không toàn vẹn dữ liệu.
- Các miền găng dùng để cung cấp sự toàn vẹn dữ liệu.
- Một tiến trình đòi hỏi miền găng phải không bị chờ mãi mãi: deadlock hoặc starvation.

Hợp tác bằng việc giao tiếp

- Giao tiếp cung cấp phương cách để đồng bộ hóa nhiều hoạt động.
- Có khả năng deadlock
  - Mỗi tiến trình đều chờ thông điệp từ một tiến trình khác.
- Có khả năng xảy ra tình trạng đói (starvation)
  - Hai tiến trình gởi thông điệp cho nhau trong khi một tiến trình khác chờ thông điệp.

Ví dụ 1: hợp tác tiểu trình

- Các tiểu trình độc lập:

Thread A	Thread B
x = 1;	y = 2;

- Các tiểu trình có hợp tác với nhau (y=0):

Thread A	Thread B
x = 1;	y = 2;
x = y+1;	y = y*2;

  - x=?
- Một cách đơn giản: x = bao nhiêu trong ví dụ sau

Thread A	Thread B
x = 1;	x = 2;

Khái niệm cơ bản trong đồng bộ hóa

- ***Thao tác nguyên tố (Atomic Operation)***: là thao tác một khi chạy là luôn luôn hoàn thành, không bị cắt ngang nửa chừng.
  - Là khái niệm cơ bản để tạo nên nguyên lý lập trình đa chương
  - Ví dụ: phép load, store là thao tác nguyên tố
  - Ví dụ: phép nhân 3*4.5 ???
- *Lưu ý: Trong các bài toán đồng bộ hóa, cần chú ý rằng một tiểu trình (tiến trình) có thể bị ngắt tại bất cứ thời điểm nào.*

Ví dụ 2: đồng bộ hóa tiến trình

- Giả sử có 2 tiểu trình:

Thread A		Thread B	
r1=0	load r1, M[i]		
		r1=0	load r1, M[i]
r1=1	add r1, r1, 1		
		r1=-1	sub r1, r1, 1
M[i]=1	store r1, M[i]		
		M[i]=-1	store r1, M[i]

- Vấn đề:

Vấn đề của đồng bộ hóa

- Tiến trình (tiểu trình) có thể bị lấy lại CPU bất cứ lúc nào (bị cưỡng chế hoặc không bị cưỡng chế).
- Dữ liệu (biến) có thể được chia sẻ bởi nhiều tiến trình (tiểu trình)
  - Không bảo đảm sự toàn vẹn dữ liệu
  - Giá trị của các biến dùng chung không thể định trước, phụ thuộc vào quá trình thực thi của các tiến trình.

Ví dụ 3: Mua sữa

Thời điểm	Chồng	Vợ
3:00	Nhìn vào tủ lạnh. Thấy hết sữa cho con	
3:05	Đi ra cửa hàng	
3:10	Đến cửa hàng	Nhìn vào tủ lạnh. Thấy hết sữa cho con
3:15	Mua sữa cho con	Đi ra cửa hàng
3:20	Mang sữa về bỏ trong tủ	Đến cửa hàng
3:25		Mua sữa cho con
3:30		Mang sữa về bỏ trong tủ

Một vài khái niệm

- ***Đụng độ (Race condition)***: tình huống mà nhiều tiến trình (tiểu trình) cùng truy cập và thao tác dữ liệu chia sẻ một cách đồng thời. Giá trị dữ liệu cuối cùng phụ thuộc vào sự luân phiên thi hành của các tiến trình.
- Giải quyết đụng độ → ***đồng bộ hóa***

Một vài khái niệm

- ***Đồng bộ hóa (synchronization)***: là công việc sử dụng các *thao tác nguyên tố* để bảo đảm sự hợp tác giữa các tiến trình đạt được kết quả mong muốn.
- ***Loại trừ hỗ tương (độc quyền truy xuất) (mutual exclusion)***: bảo đảm rằng chỉ có một tiểu trình (tiến trình) được quyền thao tác tại một vùng nhớ tại một thời điểm.
  - Loại trừ các tiến trình khác làm việc như mình
- ***Miền găng (critical section)***: đoạn mã chỉ cho phép một tiểu trình (tiến trình) thi hành tại một thời điểm.
  - Tiểu trình (tiến trình) được gọi là đi vào miền găng.
  - Loại trừ hỗ tương và miền găng là hai khái niệm cùng một mục đích.

Giải pháp 1 cho việc mua sữa

- Đưa ra một qui định (dùng miếng giấy thông báo):
  - Trước khi đi mua sữa dán miếng giấy thông báo
  - Mua sữa về gỡ bỏ miếng giấy thông báo
  - Nếu thấy miếng giấy thì không cần phải đi mua (chờ)
- Đưa cho máy tính thực hiện

```
if (noMilk) {
  if (noNote) {
   leave Note;
   buy Milk;
   remove Note;
  }
}
```


Giải pháp 1 cho việc mua sữa

```
if (noMilk) {
  if (noNote) {
   leave Note;
   buy Milk;
   remove Note;
  }
}
```

```
if (noMilk) {
  if (noNote) {
   leave Note;
   buy Milk;
   remove Note;
  }
}
```

→ vẫn có khả năng sai

Giải pháp 2 cho việc mua sữa

- Giải pháp 1 có vẻ "dán giấy thông báo" quá trễ

Thread A

```
leave note A;
if (noNote B) {
   if (noMilk) {
      buy Milk;
   }
}
remove note A;
```

Thread B

```
leave note B;
if (noNoteA) {
   if (noMilk) {
      buy Milk;
   }
}
remove note B;
```

→ có khả năng cả hai người không ai đi mua sữa

Giải pháp 3 cho việc mua sữa

Thread A

```
leave note A;
while (note B) {
   do nothing;
}
if (noMilk) {
   buy milk;
}
remove note A;
```

Thread B

```
leave note B;
if (noNote A) {
     if (noMilk) {
          buy milk;
     }
}
remove note B;
```

- A: nếu thấy B đã dán giấy thì chờ (vòng while), ngược lại mua sữa.
- B: nếu thấy A đã dán giấy thì gỡ bỏ giấy báo của mình, nhường cho A mua. Ngược lại thì đi mua.

Phân tích giải pháp 3

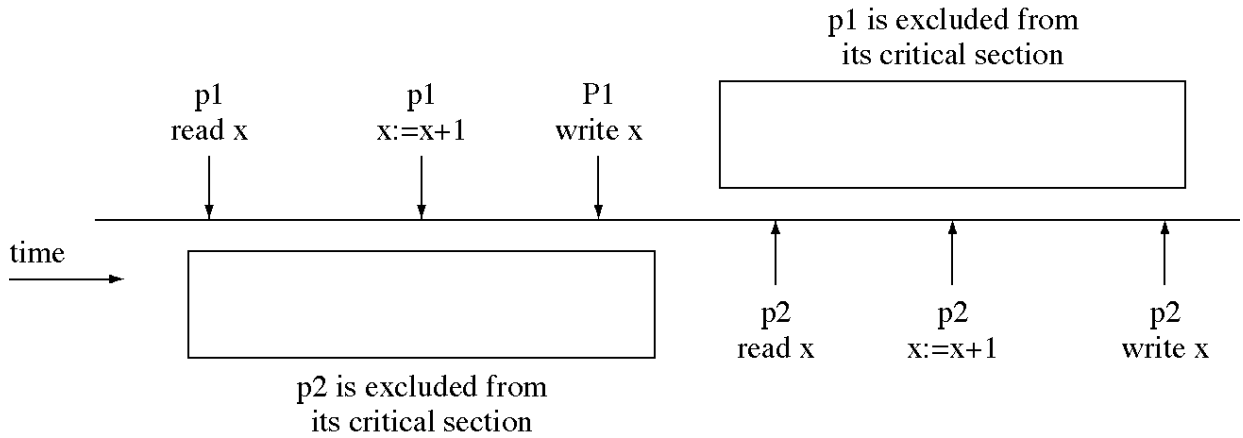
```
if (noMilk) {
  buy milk;
}
```

được gọi là miền găng. Mục đích của cả 3 giải pháp là bảo vệ miền găng này. Tại một thời điểm bất kỳ chỉ có thể có nhiều nhất là một tiến trình, tiểu trình vào miền găng.

Phân tích

- Rất phức tạp dù ví dụ rất đơn giản
- Đoạn mã của A và B khác nhau dù hai người cùng làm một việc. Nếu có rất nhiều tiểu trình thì sao???
- Trong khi A chờ đợi, nó vẫn sử dụng CPU → busy-waiting

Miền găng (critical section)



- Lập trình đa chương cho phép song song theo lý thuyết để sử dụng các thiết bị hiệu quả hơn. Nhưng chúng ta đã trả giá bởi tính đúng đắn không còn.
- Do đó chấp nhận gỡ bỏ tính song song → tính đúng đắn

Vấn đề miền găng

- *n* tiến trình đấu tranh với nhau để sử dụng một số dữ liệu nào đó.
- Mỗi tiến trình có một đoạn mã, gọi là *miền găng* (*critical section (CS)*), tại đó dữ liệu chia sẻ được truy cập.
- Vấn đề: bảo đảm rằng khi một tiến trình đang thực thi trong miền găng của nó, không có tiến trình nào khác được quyền thực thi trong miền găng của nó.

Ngữ cảnh miền găng (1)

- Khi một tiến trình thi hành đoạn mã thao tác trên dữ liệu chia sẻ (hay tài nguyên), chúng ta nói rằng tiến trình đó đang trong miền găng của nó.
- Việc thực thi các miền găng phải có tính duy nhất: tại bất kỳ thời điểm nào, chỉ có duy nhất một tiến trình được quyền thực thi trong miền găng của nó (ngay cả với nhiều bộ xử lý).
- Vì vậy mỗi tiến trình phải yêu cầu quyền trước khi vào miền găng.

Ngữ cảnh miền găng (2)

- Đoạn mã thể hiện yêu cầu này được gọi làEntry Section (ES).
- Miền găng (CS) có thể theo sau là Leave/Exit Section (LS).
- Phần đoạn mã còn lại là Remainder Section (RS).
- Vấn đề của miền găng là thiết kế một giao thức mà các tiến trình có thể sử dụng để hành động của chúng sẽ không phụ thuộc vào thứ tự mà sự thi hành của chúng được chen vào.

Ngữ cảnh miền găng (3)

```
repeat
   entry section
    critical section
   exit section
    remainder section
forever
```

- Giải pháp = cần phải chỉ ra các lệnh được thực thi của ***entry section*** và ***exit section***
- Không có bất cứ sự giả sử nào về tốc độ CPU và thứ tự thi hành của các tiến trình

Giải pháp cho vấn đề miền găng

- Có 3 yêu cầu mà một giải pháp đúng cần phải thỏa mãn:

1. ***Mutual Exclusion***: không có 2 tiến trình cùng ở trong miền găng một lúc
2. ***Progress***: Một tiến trình bên ngoài miền găng không được ngăn cản các tiến trình khác vào miền găng
3. ***Bounded Waiting***: không có tiến trình nào phải chờ vô hạn để vào miền găng

- Chỉ cần một trong ba điều kiện trên sai thì giải pháp đưa ra là sai.

Phân loại các giải pháp cho CS

- Giải pháp phần mềm
  - Các thuật toán mà tính đúng đắn của nó không dựa trên bất kỳ một giả thuyết nào khác.
- Giải pháp phần cứng
  - Dựa trên các mã lệnh máy đặc biệt.
- Giải pháp HĐH
  - Cung cấp các hàm và cấu trúc dữ liệu cho lập trình viên thông qua lời gọi hệ thống.
- Giải pháp ngôn ngữ lập trình –
  - Là một phần của ngôn ngữ.

Giải pháp phần mềm

- Xét trường hợp có 2 tiến trình:
  - Thuật toán 1, 2, 3 → sai.
  - Thuật toán 4 → đúng (Peterson's algorithm).
- Tổng quát hóa cho trường hợp n tiến trình:
  - Thuật toán Bakery.
- Qui ước:
  - 2 tiến trình: P_0 và P_1
  - Hoặc P_i, P_j
  - Hoặc Larry và Jim

Cấu trúc của các tiến trình

- Cấu trúc tổng quát của tiến trình P_i (P_j)

```
do {
    entry section
        critical section
    leave section
        remainder section
} while (1);
```

- Lưu ý: Các tiến trình có thể chia sẻ các biến dùng chung để đồng bộ hóa hoạt động của chúng.

Thuật toán 1

- Ý tưởng: sử dụng một biến luân phiên.
  - Biến luân phiên mang giá trị nào thì tiến trình tương ứng được phép vào miền găng.
  - Sau khi ra khỏi miền găng thì đặt giá trị biến luân phiên cho phép tiến trình còn lại vào miền găng.

Thuật toán 1- Larry/Jim version

- Biến dùng chung:
  - **string turn**; khởi tạo **turn = "Larry" hoặc "Jim"**
  - **turn = "Larry"** $\Rightarrow$ *Larry* có thể vào trong miền găng
- Tiến trình *Larry*

```
do {
   while (turn != "Larry");
      critical section
   turn ="Jim";
      remainder section
} while (1);
```

- Tiến trình Jim tương tự nhưng hoán đổi Larry và Jim.

Thuật toán 1- Pi/Pj version

- Biến dùng chung:
  - **int turn**; khởi tạo **turn = 0**
  - **turn = i** $\Rightarrow$ P_i có thể vào miền găng
- Tiến trình P_i

```
do {
   while (turn != i) ;
      critical section
   turn = j;
      remainder section
} while (1);
```

- Thỏa "mutual exclusion" và "bounded waiting", nhưng không thỏa "progress" (???).

Thuật toán 1: không thỏa tính progress

- Giả sử P_i vào miền găng được (turn = i).
- P_i ra khỏi miền găng → nhường quyền vào miền găng cho P_j → turn = j
- Xét trường hợp P_j không thích vào miền găng (do bận xử lý một việc gì đó) và P_i muốn vào miền găng trở lại. Nhưng turn = j → P_i không vào miền găng được.
- → P_j ở ngòai miền găng nhưng đã ngăn cản không cho P_i vào miền găng

Thuật toán 2

- Ý tưởng: dùng hai biến cho hai tiến trình
  - Một tiến trình A trước khi vào miền găng cần bảo đảm tiến trình B còn lại không muốn vào miền găng.
  - Nếu tiến trình B không muốn vào miền găng, thì tiến trình A đặt cờ báo hiệu nó muốn vào miền găng. Ngược lại phải chờ.
  - Khi ra khỏi miền găng, tiến trình A đặt giá trị biến báo hiệu cho biến là nó không muốn vào miền găng nữa

Thuật toán 2 - Larry/Jim version

- Biến dùng chung
  - **boolean flag-larry, flag-jim**;
    khởi tạo **flag-larry = flag-jim = false**
  - **flag-larry= true** ⇒ *Larry* sẵn sàng để vào miền găng
- Tiến trình *Larry*

```
do {
    while (flag-jim);
  flag-larry = true;
    critical section
  flag-larry = false;
    remainder section
} while (1);
```

Thuật toán 2 - Pi/Pj version

- Biến dùng chung
  - **boolean flag[2]**; khởi tạo **flag [0] = flag [1] = false**
  - **flag [i] = true** $\Rightarrow$ P_i sẵn sàng vào miền găng
- Tiến trình P_i

```
do {
    while (flag[j]);
  flag[i] = true;
    critical section
  flag [i] = false;
    remainder section
} while (1);
```

- Thỏa mãn tính "progress", nhưng không thỏa mãn "mutual exclusion" và "bounded waiting"(???).

Thuật toán 2: không thỏa mãn mutual exclusion

Tiến trình 1:

```
do {
  while (flag-jim);

  flag-larry = true;
     critical section
  flag-larry = false;

     remainder
section
} while (1);
```

Tiến trình 2:

```
do {

    while (flag-larry);

  flag-jim = true;
     critical section
  flag-jim = false;

     remainder section
} while (1);
```

Thuật toán 3

- Ý tưởng: dùng hai biến như thuật toán 2 nhưng thể hiện mong muốn vào miền găng trước khi quan tâm đến mong muốn của tiến trình khác.
  - Nhằm loại bỏ sự vi phạm mutual exclusion

Thuật toán 3 - Larry/Jim version

- Biến dùng chung
  - **boolean flag-larry, flag-jim**;
    khởi tạo **flag-larry = flag-jim = false**
  - **flag-larry= true** ⇒ *Larry* sẵn sàng vào miền găng
- Tiến trình *Larry*

```
do {
flag-larry = true;
  while (flag-jim);
    critical section
flag-larry = false;
      remainder section
} while (1);
```

Thuật toán 3 - Pi/Pj version

- Biến dùng chung
  - **boolean flag[2]**; khởi tạo **flag [0] = flag [1] = false**
  - **flag [i] = true** $\Rightarrow$ P_i muốn vào miền găng
- Tiến trình P_i

```
do {
   flag[i] = true;
   while (flag[j]);
      critical section
   flag [i] = false;
      remainder section
} while (1);
```

- Thỏa mãn “mutual exclusion”, nhưng không thỏa mãn “progress” và “bounded waiting”(???).

Thuật toán 4

- Ý tưởng: kết hợp ý tưởng của thuật toán 1, 2 và 3. Dùng một biến luân phiên và hai biến đại diện cho hai tiến trình.
  - Thể hiện mong muốn vào miền găng của mình trước (đặt giá trị tương ứng với tiến trình)
  - Nhường quyền vào miền găng cho tiến trình còn lại (đặt giá trị biến luân phiên)
  - Một tiến trình chỉ được vào miền găng khi có mong muốn vào miền găng và đến lượt mình được vào miền găng.

Thuật toán 4 - Larry/Jim version

- Kết hợp biến dùng chung của thuật toán 1 và 2/3.
- Tiến trình *Larry*

```
do {
   flag-larry = true;
   turn = “Jim”;
   while (flag-jim and turn = “Jim”);
      critical section
   flag-larry = false;
      remainder section
} while (1);
```


Thuật toán 4 - Pi/Pj version

- Kết hợp biến dùng chung của thuật toán 1 và 2/3.
- Tiến trình P_i

```
do {
   flag [i]:= true;
   turn = j;
   while (flag [j] and turn = j);
      critical section
   flag [i] = false;
      remainder section
} while (1);
```

- Thỏa mãn cả ba điều kiện → giải quyết vấn đề miền găng cho 2 tiến trình.

Thuật toán 5 - Larry/Jim version

- Như thuật toán 4, nhưng hoán đổi 2 câu lệnh đầu tiên cho nhau.
- Tiến trình *Larry*

```
do {
    turn = "Jim";
   flag-larry = true;
   while (flag-jim and turn = "Jim");
     critical section
   flag-larry = false;
     remainder section
}while (1);
```


Thuật toán Bakery

- Miền găng với n tiến trình tranh chấp:
- Trước khi vào miền găng, tiến trình nhận một con số. Tiến trình có con số nhỏ nhất được vào miền găng.
- Nếu hai tiến trình P_i và P_j nhận cùng một số thì nếu $i < j$, thì P_i được vào trước, ngược lại P_j được vào trước.
- Bộ tạo số luôn tạo ra các con số theo thứ tự tăng: 1,2,3,3,3,3,4,5...

Thuật toán Bakery

- Chọn một số:
  - max (a_0,…, a_{n-1})
- Thứ tự từ vựng (ticket #, PID #)
  - (a,b) < (c,d) nếu $a < c$ hoặc nếu $a = c$ và $b < d$
- Dữ liệu dùng chung:

  boolean choosing[n];

  int number[n];

  Cấu trúc dữ liệu được khởi tạo bằng **false** và **0**.

Thuật toán Bakery

```
do {
   choosing[i] = true;
   number[i] = max(number[0], …, number[n – 1]) +1;
   choosing[i] = false;
   for (j = 0; j < n; j++) {
      while (choosing[j]) ;
      while ((number[j]!=0) &&
   ((number[j],j)<(number[i],i)) ;
   }
     critical section
   number[i] = 0;
     remainder section
} while (1);
```

Tiến trình bị lỗi?

- Nếu cả 3 điều kiện (ME, progress, bounded waiting) đều được thỏa thì một giải pháp hợp lệ sẽ không bị ảnh hưởng trong trường hợp phần RS bị lỗi.
- Tuy nhiên trong trường hợp nếu lỗi xảy ra đúng ở trong miền găng thì nó sẽ chiếm miền găng vĩnh viễn.

Khuyết điểm của giải pháp phần mềm

- Giải pháp phần mềm dễ bị đổ vỡ.
- Tiến trình đang yêu cầu vào miền găng ở trạng thái “busy waiting” (vẫn tiêu tốn CPU).
- Nếu miền găng dài, sẽ hiệu quả hơn nếu block các tiến trình đang chờ.

Giải pháp phần cứng: cấm ngắt

- Khi ở trong miền găng, các tiến trình khác không được chen ngang vào miền găng.

```
Process Pi:
repeat
  disable interrupts
    critical section
  enable interrupts
    remainder section
forever
```


Các chỉ thị phần cứng đặc biệt

- Truy xuất đến một vùng nhớ sẽ không cho phép các tiến trình khác truy xuất vào cùng địa chỉ.
- Mở rộng: cung cấp hai chỉ thị nguyên tố để thi hành thao tác đọc ghi trên cùng một vùng nhớ.
- Sự thi hành của chỉ thị này thỏa mãn “mutually exclusive”.
- Cần bổ sung thêm cơ chế khác để thỏa mãn 2 tính chất còn lại.

Chỉ thị test-và-set

- Mô tả trong C++:

```
bool testset(int& i)
{
  if (i==0) {
    i=1;
    return true;
  } else {
    return false;
  }
}
```

- Thuật toán sử dụng testset cho Mutual Exclusion:
  - Biến dùng chung b được khởi tạo về 0
  - Chỉ duy nhất tiến trình P_i đặt b=1 được vào CS

```
Process Pi:
repeat
  repeat{}
  until testset(b);
     CS
  b:=0;
     RS
forever
```


TestAndSet trong phần cứng

- Kiểm tra và thay đổi nội dung của một word một cách không thể phân chia:

```
boolean TestAndSet(boolean &target)
{
    boolean rv = target;
    target = true;
    return rv;
}
```

Mutual Exclusion với TestAndSet

- Biến dùng chung:

  boolean lock = false;

- Tiến trình P_i

```
do {
  while (TestAndSet(lock)) ;
     critical section
  lock = false;
     remainder section
}
```


Swap trong phần cứng

- Hoán đổi hai biến (không thể phân chia).

```
void Swap(boolean &a, boolean &b) {
  boolean temp = a;
  a = b;
  b = temp;
}
```

Mutual Exclusion với Swap

Biến dùng chung:

boolean lock = false;

- Tiến trình P_i

```
do {
   key = true;
   while (key == true)
         Swap(lock,key);
      critical section
   lock = false;
      remainder section
}
```


Semaphores (1)

- Là một công cụ đồng bộ hóa được cung cấp bởi HĐH không đòi hỏi "busy waiting".
- Một semaphore S là một biến nguyên mà ngoài lệnh khởi tạo ra, chỉ có thể được truy xuất thông qua hai thao tác độc quyền truy xuất và nguyên tố:
  - wait(S)
  - signal(S)

CS với n tiến trình

- Dữ liệu dùng chung:

  semaphore mutex; ***//***khởi tạo *mutex* = 1
- Tiến trình *Pi:*

```
do {
    wait(mutex);
        critical section
    signal(mutex);
        remainder section
} while (1);
```


Semaphores (2)

- Truy cập với 2 thao tác

  wait (*S*):

  while *S*≤ 0 do *no-op*;

  ***S*--;**

  signal (*S*):

  S*++*;
- Để tránh “busy waiting”: khi một tiến trình phải đợi, nó sẽ được đặt vào hàng đợi block.

Semaphores (3)

- Semaphore bản chất là một cấu trúc:

```
type semaphore = record
                   count: integer;
                   queue: list of process
                 end;
var S: semaphore;
```

- Khi một tiến trình phải đợi một semaphore S, nó sẽ bị block và đặt vào hàng đợi của semaphore tương ứng.
- Thao tác signal lấy một tiến trình từ trong hàng đợi và đặt nó vào trong danh sách các tiến trình ở trạng thái sẵn sàng.

Thao tác trên Semaphore

```
wait(S):
  S.count--;
  if (S.count<0) {
    block this process
    place this process in S.queue
  }

signal(S):
  S.count++;
  if (S.count<=0) {
    remove a process P from S.queue
    place this process P on ready list
  }
```

Cài đặt semaphore (1)

- Định nghĩa cấu trúc:

```
typedef struct {
   int value;
   struct process *L;
} semaphore;
```

- Giả sử có 2 thao tác cơ bản:
  - **Block** tạm cho tiến trình chờ.
  - **wakeup(*P*)** khôi phục lại sự thi hành của tiến trình bị block **P**.

Cài đặt semaphore

-

```
wait(S):
        S.value--;
        if (S.value < 0) {
                add this process to S.L;
                block;
        }
signal(S):
        S.value++;
        if (S.value <= 0) {
                remove a process P from S.L;
                wakeup(P);
        }
```

Công cụ đồng bộ hóa tiến trình

- B được thi hành sau A.
- Dùng semaphore *flag* khởi tạo ban đầu bằng 0
- Mã:

P_i	P_j
⋮	⋮
A	*wait*(*flag*)
signal(*flag*)	*B*

Deadlock và Starvation

- **Deadlock:**
  - Gọi S và Q là hai semaphore được khởi tạo 1

P_0	P_1
wait(*S*);	*wait*(*Q*);
wait(*Q*);	*wait*(*S*);
⋮	⋮
signal(*S*);	*signal*(*Q*);
signal(*Q*)	*signal*(*S*);

- **Starvation** : không được lấy ra khỏi hàng đợi block của semaphore

Ưu khuyết điểm của Semaphore

- Semaphore cung cấp một công cụ mạnh mẽ để hỗ trợ cơ chế độc quyền truy xuất và đồng bộ hóa các tiến trình.
- Tuy nhiên wait(S) và signal(S) nằm rải rác ở các tiến trình. Vì vậy, khó để hiểu ý nghĩa của nó.
- Sử dụng phải đúng trong tất cả các process.
- Một tiến trình sử dụng không đúng sẽ ảnh hưởng đến các tiến trình khác.
- Vai trò lập trình viên nặng nề

Monitors (1)

- Định nghĩa tổng quan:
  - Monitor là một cấu thành của ngôn ngữ cấp cao cung cấp các chức năng tương đương như semaphore nhưng dễ điều khiển hơn.
  - Monitor là một cấu thành của ngôn ngữ cấp cao hỗ trợ điều khiển truy cập đến dữ liệu chia sẻ.
- Một số ngôn ngữ hỗ trợ:
  - Concurrent Pascal, Modula-3, uC++, Java...
- Có thể được cài đặt bằng semaphore.

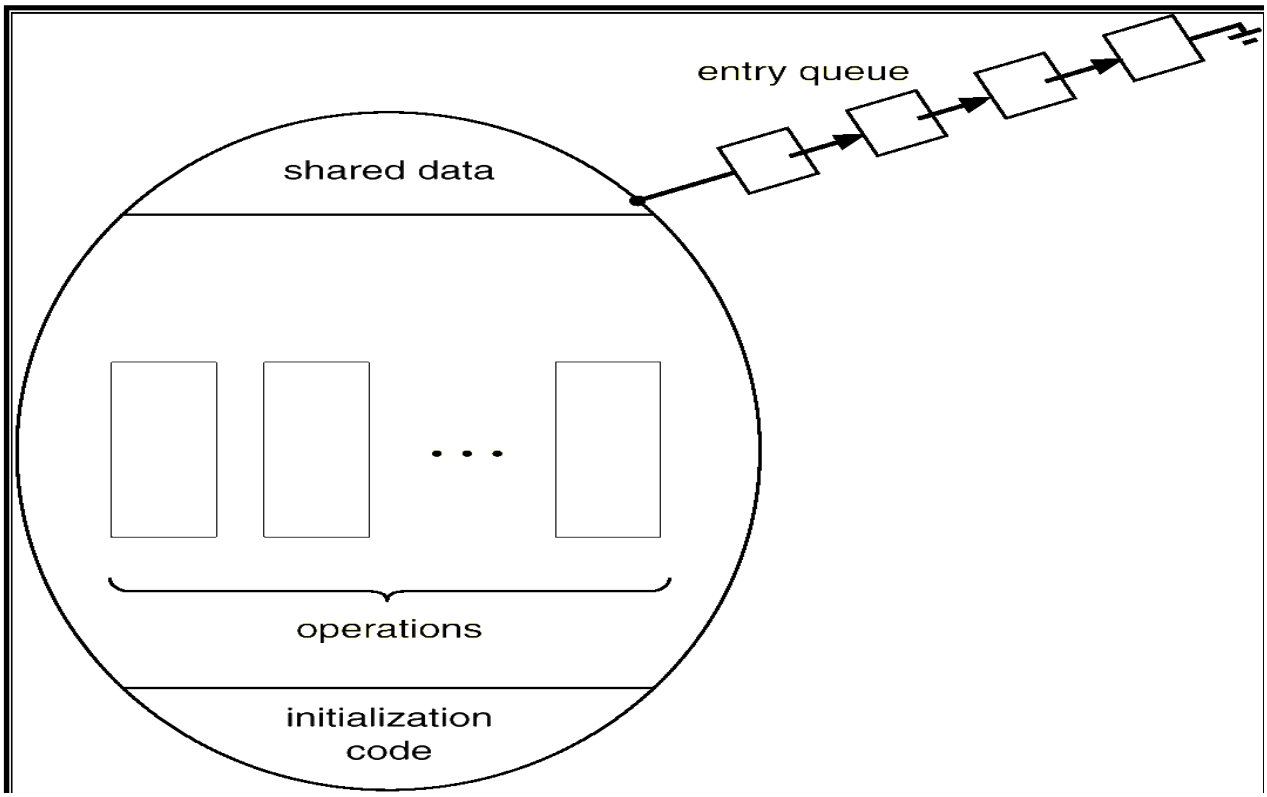
Monitors (2)

- Monitor là một module phần mềm chứa:
  - Một hay nhiều thủ tục (thao tác)
  - Một đoạn mã khởi tạo
  - Các biến cục bộ
- Tính chất:
  - Các biến cục bộ chỉ có thể được truy cập bởi các thủ tục của monitor.
  - Một tiến trình vào monitor bằng cách gọi một thủ tục của nó.
  - Tại một thời điểm bất kỳ, chỉ có duy nhất một tiến trình có thể ở trong monitor.

Cấu trúc monitor

```
monitor monitor-name
{
    shared variable declarations
    procedure body P1 (...) {  . . .  }
    procedure body P2 (...) {  . . .  }
      . . .
    procedure body Pn (...) {  . . .  }
      . . .
    {  initialization code  }
}
```

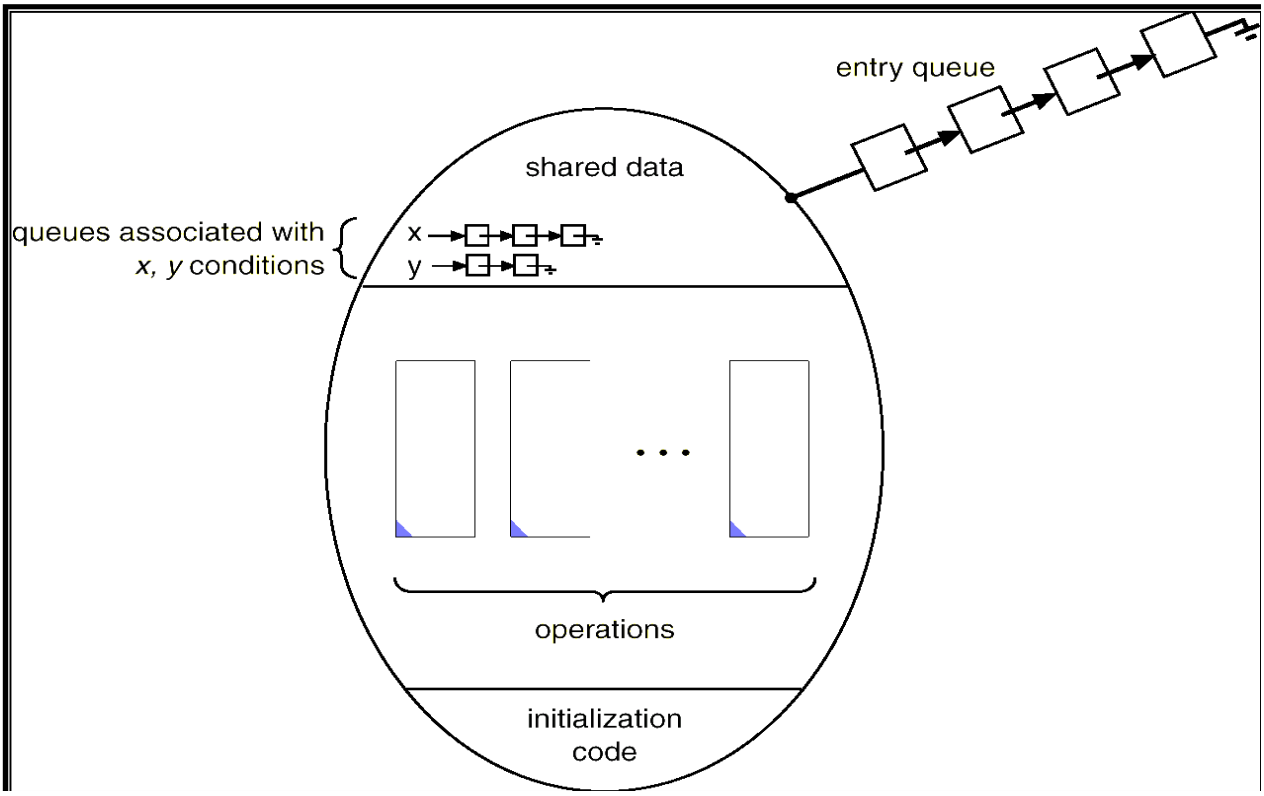
Mô hình của monitor



Các đặc trưng của monitor

- Monitor bảo đảm sự độc quyền, không cần phải lập trình rõ ràng như các phương pháp khác.
- Vì vậy, dữ liệu dùng chung được bảo vệ bằng cách đặt nó vào trong monitor.
- Có thể đồng bộ hóa các tiến trình bằng cách sử dụng các biến điều kiện để biểu diễn các điều kiện mà một tiến trình có thể phải chờ trước khi được thi hành trong monitor.

Monitor có biến điều kiện



Điều kiện trong monitor (1)

- Để cho phép một tiến trình chờ trong monitor, một biến điều kienẹ phải được khai báo, ví dụ:

  condition x, y;

- Các biến điều kiện chỉ có thể được truy xuất bên trong monitor.
- Các biến điều kiện chỉ có thể được sử dụng với 2 thao tác **cwait** và **csignal** thi hành trên một hàng đợi đi kèm.

Điều kiện trong monitor (2)

- Thao tác **cwait** và **csignal**:
  - Lời gọi

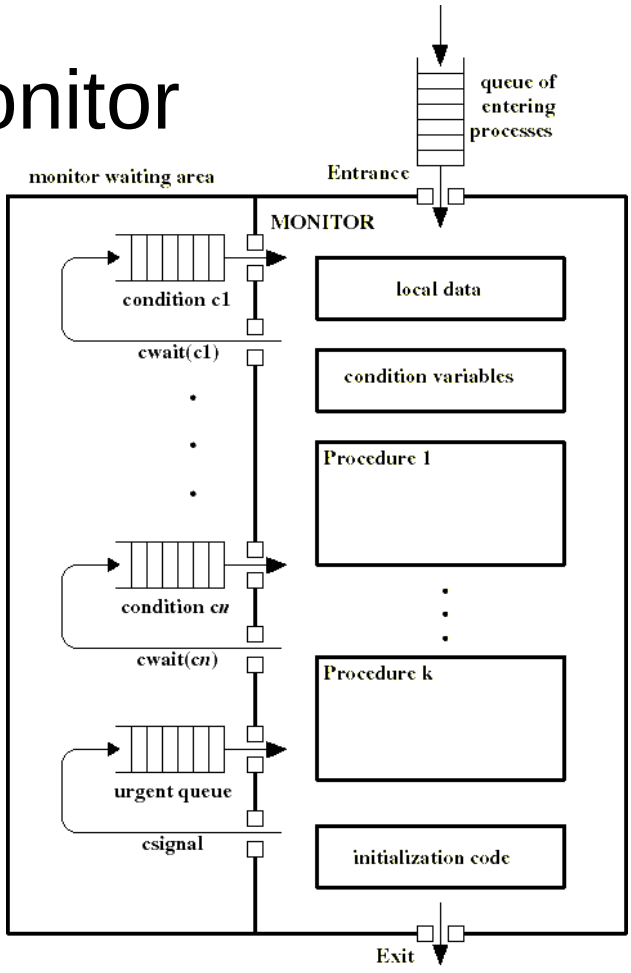
    x.cwait(); or cwait(x);

    nghĩa rằng tiến trình gọi nó sẽ bị treo co đến khi tiến trình khác đánh thức.

    x.csignal(); or csignal(x);
  - Thao tác **csignal** khôi phục một và chỉ một tiến trình bị treo. Nếu không có tiến trình nào đang bị treo, thì thao tác **csignal** không có ý nghĩa.

Ngữ cảnh của monitor

- Chờ đợi các tiến trình từ hàng đợi vào hoặc hàng đợi điều kiện.
- Một tiến trình tự đặt nó vào hàng đợi cn bằng lời gọi cwait(cn).
- csignal(cn) lấy ra một tiến trình trong hàng đợi điều kiện cn và đưa vào thi hành trong monitor.
- Do đó csignal(cn) khóa tiến trình đang gọi nó và đặt vào hàng đợi khẩn cấp(trừ khi csignal là thao tác cuối).



Khuyết điểm của semaphore và monitor

- Sử dụng biến dùng chung để xác lập độc quyền truy xuất hoặc đồng bộ hóa.
- Trong một hệ phân tán hoặc môi trường mạng, mỗi tiến trình có một không gian bộ nhớ riêng → không thể dùng chung biến (bộ nhớ)

Truyền thông điệp (messaging)

- Công dụng:
  - Cho phép các tiến trình liên lạc với nhau mà không cần phải có các biến dùng chung.
- Hai thao tác nguyên tố cơ bản:
  - **send***(destination, message*) hoặc **send**(*message*)
  - **receive***(source, message*) hoặc **receive**(*message*)
- Kích thước của thông điệp có thể cố định hoặc linh hoạt

Truyền thông điệp (2)

- Trường hợp sử dụng:
  - Giữa các tiến trình trong cùng một máy.
  - Giữa các tiến trình trong mạng hoặc một hệ phân tán.
- Trong cả hai trường hợp, tiến trình có thể hoặc bị khóa hoặc không bị khóa khi truyền hoặc nhận thông điệp.

Truyền thông điệp (3)

- Vì vậy việc truyền thông điệp có thể bị khóa hoặc không bị khóa:
  - **Nếu bị khóa** → đồng bộ **(synchronous).**
  - **Nếu không bị khóa** → không đồng bộ (**asynchronous**)**.**
- Thao tác **send** và **receive** có thể bị khóa hoặc không bị khóa.

Đồng bộ hóa trong truyền thông điệp

- Với bên gởi: tự nhiên hơn nếu không bị khóa sau khi gởi:
  - Có thể gởi nhiều thông điệp đến nhiều địa chỉ.
  - Nhưng thường là mong đợi sự phản hồi từ phía người nhận (trường hợp gởi sai).
- Với bên nhận: tự nhiên hơn nếu bị khóa sau khi báo chờ nhận:
  - Bên nhận thường cần thông tin trước khi tiếp tục xử lý.
  - Nhưng có thể bị khóa vĩnh viễn nếu bên gởi không thể gởi (bị lỗi).

Các cách truyền thông điệp

- Truyền trực tiếp (direct communication):
  - Tên của tiến trình được dùng cho địa chỉ nguồn và địa chỉ đích.
- Truyền gián tiếp:
  - Các thông điệp được gởi đến một "hộp thư" chung chứa một hàng đợi các thông điệp.
  - Bên gởi đặt thông điệp vào trong hộp thư, và bên nhận lấy thông điệp từ trong hộp thư đó ra.

Truyền trực tiếp

- Các tiến trình phải biết rõ lẫn nhau:
  - **send**(*P, message*) – gởi một thông điệp đến tiến trình P
  - **receive**(*Q, message*) – nhận một thông điệp từ tiến trình Q
- Tính chất của đường truyền:
  - Được tự động tạo.
  - Một đường truyền được liên kết với duy nhất một cặp tiến trình đang trao đổi thông tin.
  - Mỗi cặp có đúng một đường truyền.
  - Có thể là một chiều hoặc hai chiều.

Truyền gián tiếp

- Các thông điệp được đưa vào và lấy ra từ trong hộp thư (còn gọi là cổng – port)
  - Mỗi hộp thư có một id duy nhất.
  - Các tiến trình chỉ có thể liên lạc với nhau nếu chúng chia sẻ với nhau một hộp thư.
- Tính chất của đường truyền
  - Đường truyền chỉ được thành lập không chúng chia sẻ với nhau một hộp thư dùng chung.
  - Một đường truyền có thể được sử dụng bởi nhiều tiến trình.
  - Mỗi cặp tiến trình có thể có nhiều đường truyền.
  - Có thể một chiều hoặc hai chiều.

Truyền gián tiếp (2)

- Thao tác
  - Tạo một hộp thư mới
  - Gởi và nhận thông điệp qua hộp thư
  - Hủy hộp thư
- Các thao tác nguyên tố:

  send(*A, message*) – gởi một thông điệp đến hộp thư A.

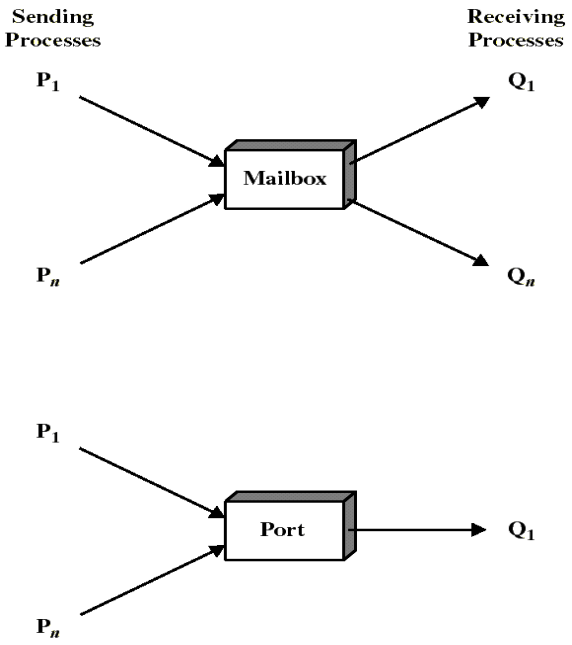
  receive(*A, message*) – nhận một thông điệp từ hộp thư A.

Truyền gián tiếp (3)

- Chia sẻ hộp thư
  - P_1, P_2, và P_3 dùng chung hộp thư A.
  - P_1 gởi; P_2 và P_3 nhận.
  - Ai lấy được thư?
- Giải pháp có thể:
  - Cho phép mỗi đường truyền chỉ có đúng 2 tiến trình.
  - Cho phép tại mỗi thời điểm chỉ có một tiến trình được thực hiện thao tác nhận.
  - Cho phép ngẫu nhiên chọn bên nhận và thông báo cho bên gởi biết ai đã nhận.

Hộp thư và cổng

- Một hộp thư có thể được dùng bởi 2 tiến trình.
- Một hộp thư có thể được chia sẻ giữa nhiều bên gởi và bên nhận:
- Cổng là một hộp thư liên kết một bên nhận và nhiều bên gởi
  - Dùng trong ứng dụng client/server: server chính là bên nhận.



Mutual Exclusion với truyền thông điệp

- Tạo một hộp thư ***mutex*** dùng bởi n tiến trình.
- send() không bị khóa.
- receive() bị khóa khi **mutex** rỗng.
- Khởi tạo: send(***mutex***, "go");
- Tiến trình Pi đầu tiên thi hành *receive()* sẽ vào trong miền găng. Các tiến trình khác bị khóa đến khi tiến trình trên kết thúc

```
Process Pi:
var msg: message;
repeat
   receive(mutex,msg);
   CS
   send(mutex,msg);
   RS
forever
```

Các bài toán cổ điển về đồng bộ hóa

- Bài toán Producer – Consumer
- Bài toán Readers – Writers
- Bài toán Triết gia dùng bữa

Bài toán Producer – Consumer

- Tiến trình sản xuất sản xuất thông tin được tiêu thụ bởi tiến trình tiêu thụ.
- Tại bất cứ thời điểm nào, một tiến trình sản xuất có thể tạo dữ liệu nào đó.
- Tại bất cứ thời điểm nào, một tiến trình tiêu thụ có thể muốn lấy một số dữ liệu nào đó.
- Dữ liệu được lưu vào trong bộ đệm.
- Nếu vùng đệm là hữu hạn, nhà sản xuất sẽ bị “khóa” nếu dữ liệu mới sẽ làm tràn bộ đệm.
- Nếu vùng đệm rỗng, nhà tiêu thụ sẽ bị khóa khi nó cần dữ liệu.

Phân tích

- Có 3 vấn đề cần đồng bộ
  - Truy xuất độc quyền (Mutual Exclusion)
  - Khi đầy thì không được sản xuất thêm
  - Khi rỗng thì không được tiêu thụ thêm

Giải pháp với semaphore

- Cần 3 semaphore
  - **Mutex** để truy xuất độc quyền
  - **Full** để kiểm soát đầy
  - **Empty** để kiểm soát rỗng

Giải pháp với Semaphore (2)

- Dữ liệu dùng chung

semaphore full, empty, mutex;
nextp, nextc; // con trỏ để thêm vào, lấy ra

- Khởi tạo:

full = 0, empty = n, mutex = 1

Tiến trình Producer

```
do {
    ...
  produce an item in nextp
    ...
  wait(empty);
  wait(mutex);
    ...
  add nextp to buffer
    ...
  signal(mutex);
  signal(full);
} while (1);
```

Tiến trình Consumer

```
do {
    wait(full)
    wait(mutex);
        …
    remove an item from buffer to nextc
        …
    signal(mutex);
    signal(empty);
        …
    consume the item in nextc
        …
} while (1);
```


- Câu hỏi:
  - Điều gì xảy ra nếu đảo 2 câu lệnh wait của 1 trong 2 tiến trình???
  - Điều gì xảy ra nếu đảo 2 câu lệnh signal của 1 trong 2 tiến trình???

Giải pháp với Monitor

- Cần lưu một vùng đệm:
  - buffer: array[0..k-1] of items;
- Cần hai biến điều kiện:
  - notfull: csignal(notfull) vùng đệm chưa đầy.
  - notemty: csignal(notempty) vùng đệm chưa rỗng.
- Các biến con trỏ:
  - nextin: con trỏ để thêm vào.
  - nextout: con trỏ để lấy ra.
  - count: số mặt hàng trong vùng đệm.
- Các thủ tục
  - append(): thêm vào một mặt hàng (Producer)
  - take(): lấy ra một mặt hàng (Consumer)

```
ProducerI:
repeat
  produce v;
  append(v);
forever

ConsumerI:
repeat
  take(v);
  consume v;
forever
```


Giải pháp với Monitor

```
Monitor boundedbuffer:
  buffer: array[0..k-1] of items;
  nextin:=0, nextout:=0, count:=0: integer;
  notfull, notempty: condition;

  append(v):
    if (count=k) cwait(notfull);
    buffer[nextin]:= v;
    nextin:= nextin+1 mod k;
    count++;
    csignal(notempty);

  take(v):
    if (count=0) cwait(notempty);
    v:= buffer[nextout];
    nextout:= nextout+1 mod k;
    count--;
    csignal(notfull);
```

Giải pháp truyền thông điệp

- Nhà sản xuất sẽ đặt các mặt hàng (trong các thông điệp) vào trong hộp thư ***mayconsume***.
- ***mayconsume*** hành động như là vùng đệm của chúng ta: nhà tiêu thụ chỉ có thể tiêu thụ nêu có ít nhất một thông điệp trong hộp thư đó.
- Hộp thư ***mayproduce*** được khởi tạo ban đầu với k thông điệp trống (k= kích thước vùng đệm).
- Kích thước của ***mayproduce*** giảm xuống khi một mặt hàng được sản xuất và tăng lên nếu một mặt hàng được tiêu thu.

Giải pháp truyền thông điệp (2)

```
Producer:
var pmsg: message;
repeat
  receive(mayproduce, pmsg);
  pmsg := produce();
  send(mayconsume, pmsg);
forever

Consumer:
var cmsg: message;
repeat
  receive(mayconsume, cmsg);
  consume(cmsg);
  send(mayproduce, null);
forever
```

Bài toán Readers - Writers

- Nhiều thao tác đọc và ghi diễn ra đồng thời trên một cơ sở dữ liệu.
- Trong khi một tiến trình đang ghi thì các tiến trình khác không được ghi hay đọc cơ sở dữ liệu.
- Các thao tác đọc có thể tiến hành đồng thời.

Phân tích

- Có 3 vấn đề cần đồng bộ
  - Khi một tiến trình đang ghi thì không một tiến trình nào được đọc.
  - Khi một tiến trình đang đọc thì không được ghi, và các tiến trình đọc khác vẫn đọc bình thường
  - Độc quyền truy xuất trên các biến dùng chung

Giải pháp với semaphore

- Dữ liệu dùng chung

  semaphore mutex, wrt;
  int readcount;

- Khởi tạo

  mutex = 1, wrt = 1, readcount = 0

Giải pháp với semaphore (2)

- **readcount** lưu số tiến trình đang đọc hiện tại.
- **mutex** cung cấp độc quyền truy xuất trên biến readcount.
- **wrt** cung cấp độc quyền truy xuất cho tiến trình ghi.

Tiến trình Writer

wait(wrt);

…

writing is performed

…

signal(wrt);

Tiến trình Reader

wait(mutex);
readcount++;
if (readcount == 1)
wait(wrt);
signal(mutex);
…
reading is performed
…
wait(mutex);
readcount--;
if (readcount == 0)
signal(wrt);
signal(mutex);

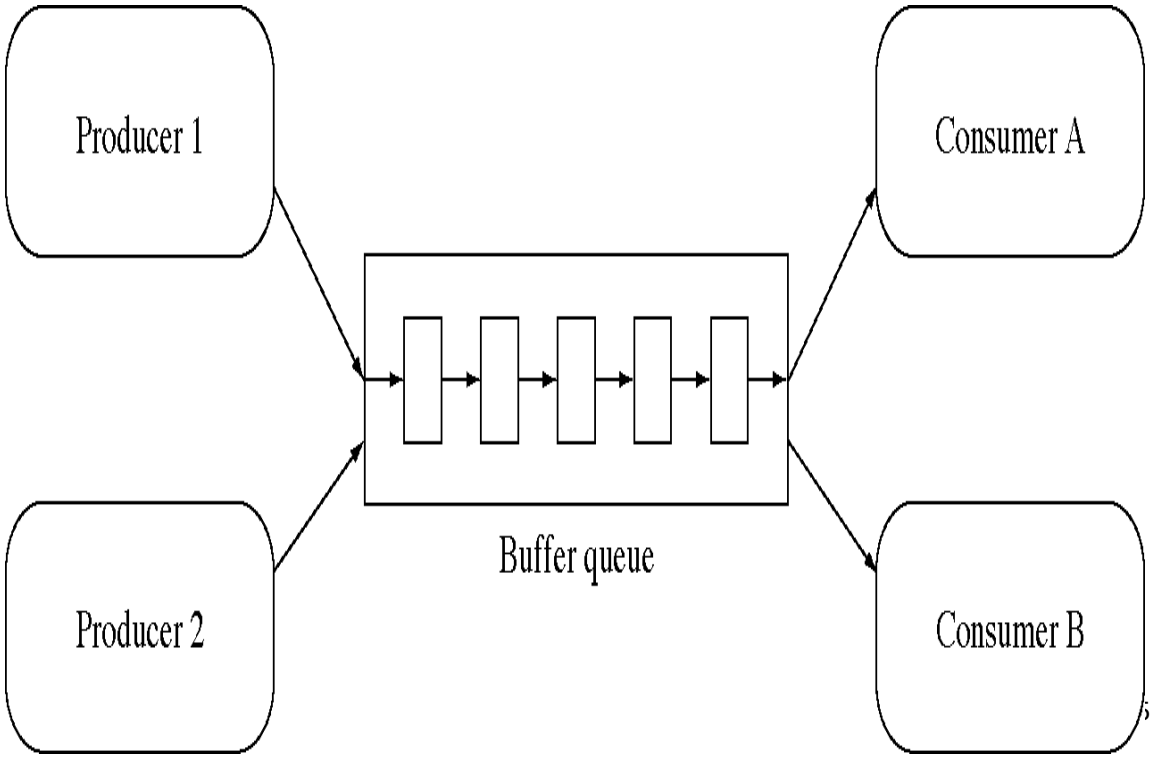
Bài toán Sản xuất / Tiêu thụ

- Một khuôn mẫu cho việc hợp tác giữa các tiến trình
  - *Producer* cung cấp thông tin được tiêu thụ bởi một tiến trình *Consumer*.

Bài toán Sản xuất / Tiêu thụ (2)

- Cần một vùng đệm để lưu các mục hàng được sản xuất và tiêu thụ sau đó:
  - *unbounded-buffer* : không giới hạn kích thước.
  - *bounded-buffer*: có kích thước xác định.

Multiple Producers and Consumers

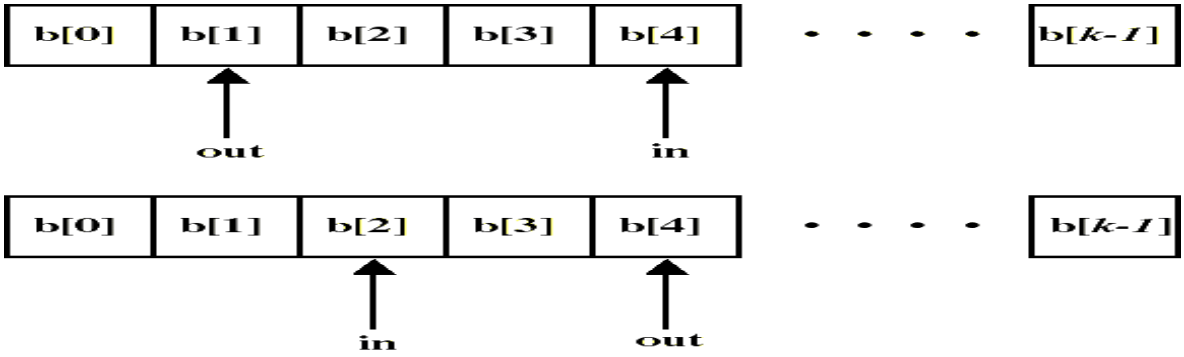


Ngữ cảnh

- Tiến trình sản xuất sản xuất thông tin được tiêu thụ bởi tiến trình tiêu thụ.
- Tại bất cứ thời điểm nào, một tiến trình sản xuất có thể tạo dữ liệu nào đó.
- Tại bất cứ thời điểm nào, một tiến trình tiêu thụ có thể muốn lấy một số dữ liệu nào đó.
- Dữ liệu được lưu vào trong bộ đệm.
- Nếu vùng đệm là hữu hạn, nhà sản xuất sẽ bị "khóa" nếu dữ liệu mới sẽ làm tràn bộ đệm.
- Nếu vùng đệm rỗng, nhà tiêu thụ sẽ bị khóa khi nó cần dữ liệu.

Ý tưởng

- Vùng đệm được cài đặt bởi mảng vòng với 2 con trỏ là **in** và **out**.
- Biến **in** trỏ đến vị trí tự do kế tiếp trong vùng đệm.
- Biến **out** trỏ đến vị trí mục hàng đầu tiên.



Bounded-Buffer: Giải pháp vùng nhớ chia sẻ

- Dữ liệu chia sẻ

```
#define BUFFER_SIZE 10
typedef struct {
  . . .
} item;
item buffer[BUFFER_SIZE];
int in = 0;
int out = 0;
```

Bounded-Buffer – Tiến trình SX

```
item nextProduced;

while (1) {
  while (((in + 1) % BUFFER_SIZE) ==
out);
                                    /* do
nothing */
    buffer[in] = nextProduced;
    in = (in + 1) % BUFFER_SIZE;
}
```


Bounded-Buffer – Consumer Process

```
item nextConsumed;

while (1) {
    while (in == out);    /* do nothing
*/
    nextConsumed = buffer[out];
    out = (out + 1) % BUFFER_SIZE;
}
```

Vấn đề

- Các tiến trình đồng hành thường cần chia sẻ dữ liệu và tài nguyên.
- Nếu không kiểm soát việc truy xuất tài nguyên → mất tính toàn vẹn dữ liệu.
- Hành động thi hành bởi các tiến trình đồng hành sẽ phụ thuộc và thứ tự thi hành của các tiến trình.

Ví dụ về sự không toàn vẹn

- Có 2 tiến trình P1, P2 cùng chia xuất đến biến a.
- Các tiến trình có thể bị ngắt bất cứ lúc nào.
- Nếu P1 bị ngắt sau khi nhập dữ liệu và P2 thi hành hoàn toàn.
- Sau đó dữ liệu xuất ra bởi P1 sẽ là kí tự đọc bởi P2!!!

```
static char a;

void echo()
{
   cin >> a;
   cout << a;
}
```

Toàn vẹn dữ liệu

- Bảo vệ toàn vẹn dữ liệu đòi hỏi cơ chế bảo đảm sự thi hành một cách có thứ tự của các tiến trình hợp tác.
- Giải pháp vùng nhớ chia sẻ ở trên cho phép tối đa n-1 mục được ở trong vùng đệm tại một thời điểm.
  - Đếm số mục hiện tại đang ở trong vùng đệm

Bounded-Buffer – bộ đếm

- Dữ liệu chia sẻ

```
#define BUFFER_SIZE 10
typedef struct {
  . . .
} item;
item buffer[BUFFER_SIZE];
int in = 0;
int out = 0;
int counter = 0;
```

Bounded-Buffer – tiến trình SX

```
item nextProduced;

while (1) {
    while (counter == BUFFER_SIZE);
                                    /* do
nothing */
    buffer[in] = nextProduced;
    in = (in + 1) % BUFFER_SIZE;
    counter++;
}
```


Bounded-Buffer – Tiến trình TT

```
item nextConsumed;

while (1) {
    while (counter == 0);  /* do nothing */
    nextConsumed = buffer[out];
    out = (out + 1) % BUFFER_SIZE;
    counter--;
}
```

Bounded-Buffer – biến đếm (1)

- Câu lệnh

  counter++;
  counter--;

  phải là câu lệnh nguyên tố.
- Câu lệnh nguyên tố là câu lệnh không thể bị ngắt.

Bounded-Buffer – Biến đếm (2)

- count++:

  register1 = counter
  register1 = register1 + 1
  counter = register1

- count--:

  register2 = counter
  register2 = register2 – 1
  counter = register2

Bounded-Buffer – Biến đếm (3)

- Nếu cả hai nhà tiêu thụ và sản xuất đều cố gắng cập nhật vùng đệm một cách đồng thời, có thể có sự thi hành xen lẫn các câu lệnh.
- Sự chen ngang phụ thuộc vào việc lên lịch các tiến trình SX và TT.

Bounded-Buffer – Biến đếm (4)

- Một khả năng có thể:
  producer: **register1 = counter** (*register1 = 5*)
  producer: **register1 = register1 + 1** (*register1 = 6*)
  consumer: **register2 = counter** (*register2 = 5*)
  consumer: **register2 = register2 – 1** (*register2 = 4*)
  producer: **counter = register1** (*counter = 6*)
  consumer: **counter = register2** (*counter =*

Đụng độ (race condition)

- **Race condition**: tình huống mà nhiều tiến trình cùng truy cập và thao tác dữ liệu chia sẻ một cách đồng thời. Dữ liệu cuối cùng phụ thuộc vào tiến trình cuối cùng.
- Để ngăn ngừa đụng độ, các tiến trình đồng hành phải được **đồng bộ hóa**.

Đụng độ do cập nhật dữ liệu

Chia sẻ mức cân đối thu chi;

Code for p1:

```
. . .
balance += amount;
. . .
```

Code for p1:

```
. . .
Load    R1, balance
Load    R2, amount
Add     R1, R2
Store   R1, balance
. . .
```

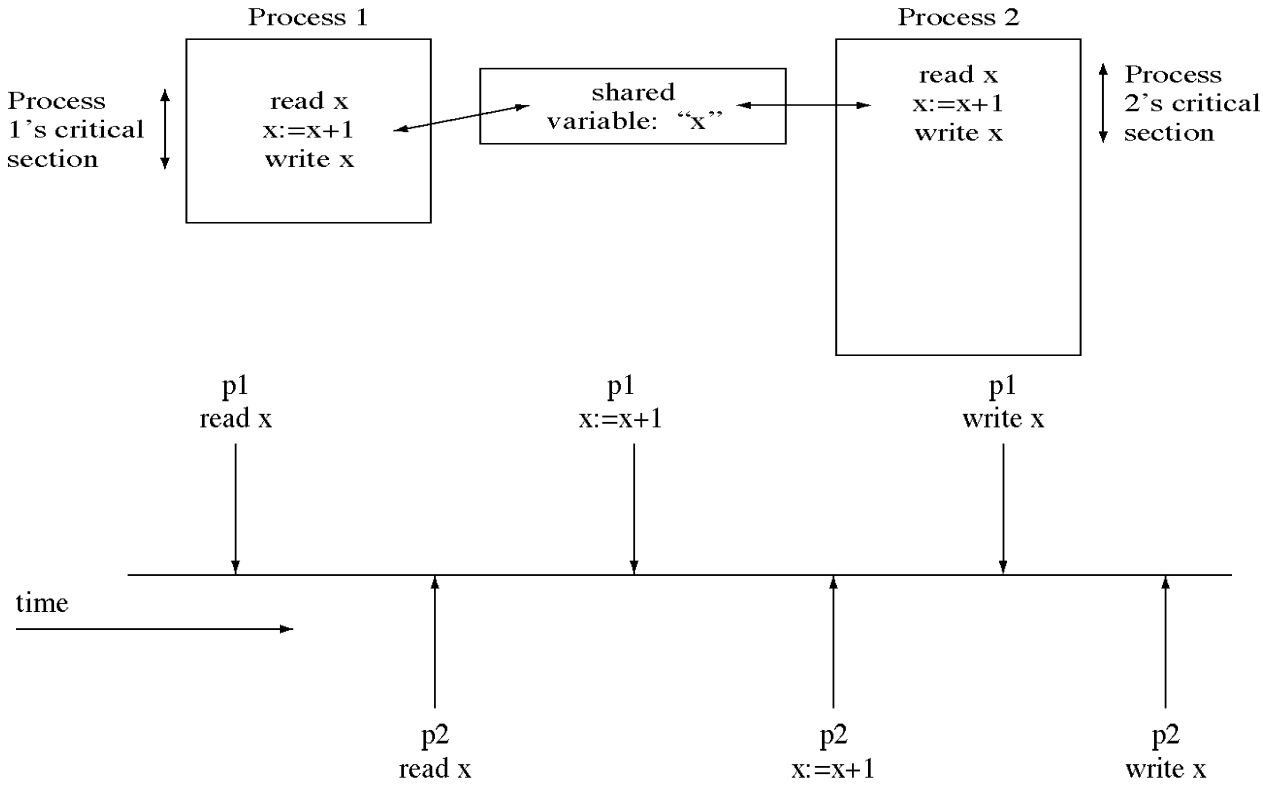
Code for p2:

```
. . .
balance += amount;
. . .
```

Code for p2:

```
. . .
Load    R1, balance
Load    R2, amount
Add     R1, R2
Store   R1, balance
. . .
```

Đụng độ do cập nhật dữ liệu



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.

Deadlock

Trường hợp deadlock có thể phát sinh nếu **4 điều kiện** sau xảy ra cùng một lúc trong hệ thống:

1) Loại trừ hỗ tương: chỉ một quá trình tại cùng một thời điểm có thể sử dụng tài nguyên không chia sẽ. Nếu một quá trình khác yêu cầu tài nguyên đó, quá trình yêu cầu phải tạm dừng cho đến khi tài nguyên được giải phóng.

2) Giữ và chờ cấp thêm tài nguyên: quá trình phải đang giữ ít nhất một tài nguyên và đang chờ để nhận tài nguyên thêm mà hiện đang được giữ bởi quá trình khác.

3) Không đòi lại tài nguyên từ quá trình đang giữ chúng: Các tài nguyên không thể bị đòi lại; nghĩa là, tài nguyên có thể được giải phóng chỉ tự ý bởi quá trình đang giữ nó, sau khi quá trình đó hoàn thành tác vụ.

4) Tồn tại chu trình trong đồ thị cấp phát tài nguyên: một tập hợp các quá trình {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.

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.

Trạng thái 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.

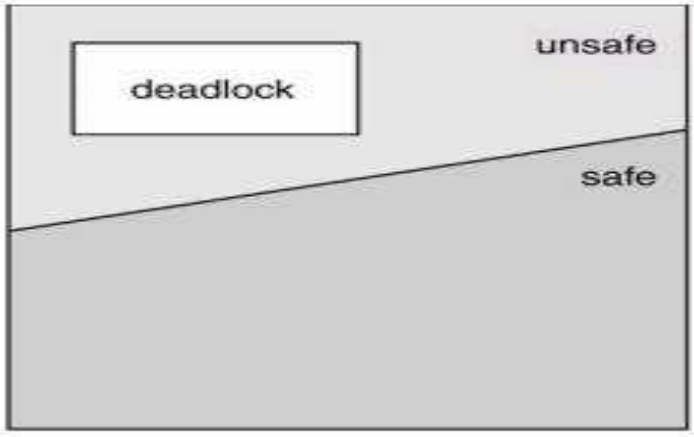
Trong trường hợp này, nếu những tài nguyên mà quá trình Pi yêu cầu không sẵn dùng tức thì thì Pi có thể chờ cho đến khi tất cả Pj hoàn thành.

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

Trạng thái 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



Các giải pháp tránh deadlock

Giải pháp đồ thị cấp phát tài nguyên: 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.

Các giải pháp tránh deadlock

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**.

Các giải pháp tránh deadlock

Nhiều cấu trúc dữ liệu phải được duy trì để cài đặt giải thuật Banker như sau:

- **Available**: một vector có chiều dài m hiển thị số lượng tài nguyên sẵn dùng của mỗi loại. Nếu Available[j]= k, có k thể hiện của loại tài nguyên Rj sẵn dùng.
- **Max**: một ma trận n x m định nghĩa số lượng tối đa yêu cầu của mỗi quá trình. Nếu Max[i , j] = k, thì quá trình Pi có thể yêu cầu nhiều nhất k thể hiện của loại tài nguyên Rj.
- **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 Pi hiện được cấp k thể hiện của loại tài nguyên Rj.
- **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 Pi có thể cần thêm k thể hiện của loại tài nguyên Rj để hoàn thành tác vụ của nó. Chú ý rằng, Need[i, j]=Max[i, j]– Allocation [i, j].

Giải Thuật An Toàn

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

1) Gọi Work và Finish là các vector có chiều dài m và n tương ứng. Khởi tạo Work:=Available và Finish[i]:=false cho i = 1, 2, …,n.
2) Tìm i thỏa:
   a) Finish[i] = false
   b) Need i $\leq$ Work.

   Nếu không có i nào thỏa, di chuyển tới bước 4
3) Work:=Work + Allocation i
   Finish[i] := true
   Di chuyển về bước 2.
4) Nếu Finish[i] = true cho tất cả i, thì hệ thống đang ở trạng thái an toàn.

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

Giải Thuật Yêu Cầu Cấp Phát Tài Nguyên

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

1) Nếu Requesti $\leq$ Needi, di chuyển tới bước 2. Ngược lại, phát sinh một điều kiện lỗi vì quá trình vượt quá yêu cầu tối đa của nó.
2) Nếu Requesti $\leq$ Available, di chuyển tới bước 3. Ngược lại, Pi phải chờ vì tàinguyên không sẵn có.
3) Giả sử hệ thống cấp phát các tài nguyên được yêu cầu tới quá trình Pi bằng cách thay đổi trạng thái sau:

   Available := Available – Requesti;
   Allocationi := Allocationi + Requesti;
   Needi := Needi – Requesti;

   Nếu kết quả trạng thái cấp phát tài nguyên là an toàn, thì Pi được cấp phát tài nguyên của nó. Tuy nhiên, nếu trạng thái mới là không an toàn, thì Pi phải chờ Requesti và trạng thái cấp phát tài nguyên cũ được phục hồi.

Ví dụ: Trạng Thái An Toàn

Xét một hệ thống với 5 quá trình từ P0 tới P4, và 3 loại tài nguyên A, B, C. Loại tài nguyên A có 10 thể hiện, loại tài nguyên B có 5 thể hiện và loại tài nguyên C có 7 thể hiện. Giả sử rằng tại thời điểm T0 trạng thái hiện tại của hệ thống như sau:

	Allocation			Max			Available		
	A	B	C	A	B	C	A	B	C
P0	0	1	0	7	5	3	3	3	2
P1	2	0	0	3	2	2			
P2	3	0	2	9	0	2			
P3	2	1	1	2	2	2			
P4	0	0	2	4	3	3			

Ví dụ: Trạng Thái An Toàn

Nội dung ma trận ***Need*** được định nghĩa là ***Max-Allocation*** và là

	Need		
	A	B	C
P0	7	4	3
P1	1	2	2
P2	6	0	2
P3	0	1	1
P4	4	3	1

Chúng ta khẳng định rằng hệ thống hiện ở trong trạng thái an toàn. Thật vậy, thứ tự <P1, P3, P4, P2, P0> thỏa tiêu chuẩn an toàn (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.)

Ví dụ: Trạng Thái An Toàn

Giả sử bây giờ P1 yêu cầu thêm một thể hiện loại A và hai thể hiện loại C, vì thế Request1 = (1, 0, 2). Để quyết định yêu cầu này có thể được cấp tức thì hay không, trước tiên chúng ta phải kiểm tra Request1 ≤ Available (nghĩa là, (1, 0, 2)) ≤ (3, 3, 2)) là đúng hay không. Sau đó, chúng ta giả sử yêu cầu này đạt được và chúng ta đi đến trạng thái mới sau:

	Allocation			Max			Available		
	A	B	C	A	B	C	A	B	C
P0	0	1	0	7	4	3	2	3	0
P1	3	0	2	0	2	0			
P2	3	0	2	6	0	0			
P3	2	1	1	0	1	1			
P4	0	0	2	4	3	1			

Ví dụ: Trạng Thái An Toàn

Tuy nhiên, chúng ta cũng thấy rằng, khi hệ thống ở trong trạng thái này, một yêu cầu (3, 3, 0) bởi P4 không thể được gán vì các tài nguyên là không sẳn dùng.

Một yêu cầu cho (0, 2, 0) bởi P0 không thể được cấp mặc dù tài nguyên là sẳn dùng vì trạng thái kết quả là không an toàn.

Hãy tính toán và giải thích rõ các ý trên?

Quản lý bộ nhớ

Nội dung

- Vì sao phải quản lý bộ nhớ
- Không gian địa chỉ lôgic - không gian địa chỉ vật lý
- Swapping
- Cấp phát bộ nhớ liên tục
- Phân trang (Paging)
- Phân đoạn (Segmentation)
- Kết hợp phân đoạn và phân trang

Vì sao phải quản lý bộ nhớ

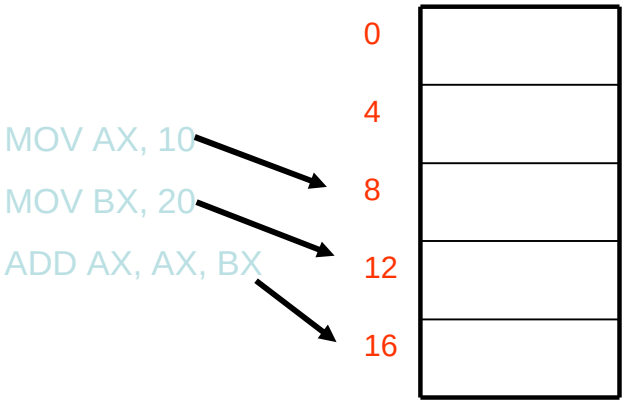
- Một chương trình muốn chạy thì phải được nạp vào trong bộ nhớ chính.
  - Vấn đề:
    - Khi nào nạp?
    - Nạp vào đâu?
    - Nạp những phần nào?
- Quản lý bộ nhớ giúp tối ưu hóa hoạt động của bộ nhớ

  →Tối ưu hóa số tiến trình cùng lúc ở trong bộ nhớ chính → nâng cao tính đa chương

  →Tận dụng tối đa bộ nhớ của máy tính

Bộ nhớ

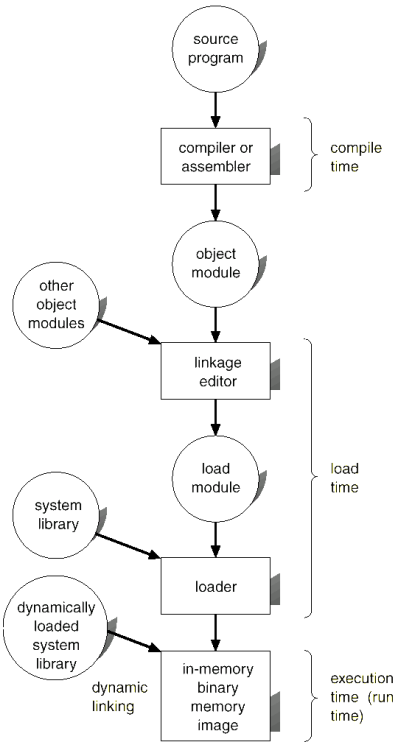
- Là một dãy các ô nhớ liên tục nhau
- Mỗi ô nhớ (một word) có một địa chỉ
- Chương trình = tập các câu lệnh (chỉ thị máy) + dữ liệu
- Nạp chương trình vào bộ nhớ ⇔ đặt các chỉ thị và dữ liệu vào các ô nhớ ⇔ xác định ánh xạ giữa các chỉ thị, dữ liệu vào địa chỉ trong bộ nhớ



Ánh xạ chỉ thị, dữ liệu vào địa chỉ bộ nhớ

Việc ánh xạ chỉ thị, dữ liệu vào địa chỉ bộ nhớ có thể xảy ra tại 3 thời điểm:

- **Thời điểm biên dịch**: nếu địa chỉ vùng nhớ được biết trước thì *mã lệnh tuyệt đối (có địa chỉ tuyệt đối)* có thể được tạo ra ngay tại thời điểm biên dịch. Nếu địa chỉ bắt đầu của vùng nhớ bị thay đổi thì sẽ phải biên dịch lại.
- **Thời điểm nạp**: Tạo ra các *mã lệnh có thể tái định vị* (*relocatable* code) nếu địa chỉ vùng nhớ không thể biết tại thời điểm biên dịch.
- **Thời điểm thi hành**: Việc kết hợp mã lệnh và địa chỉ sẽ được trì hoãn cho đến lúc chạy chương trình nếu tiến trình đó có thể bị di chuyển từ phân đoạn nhớ này đến phân đoạn nhớ khác. Cần phải có hỗ trợ từ phần cứng để ánh xạ địa chỉ (ví dụ: thanh ghi cơ sở và thanh ghi giới hạn (*base registers, limit registers*)).



Cơ chế nạp động (Dynamic loading)

- Các hàm sẽ không được nạp cho đến khi nó được gọi
- Sử dụng bộ nhớ tốt hơn: các hàm không được sử dụng sẽ không bao giờ được nạp.
- Có ý nghĩa khi một lượng lớn các mã lệnh dùng để xử lý các trường hợp ít khi xảy ra.
- Không cần hỗ trợ từ hệ điều hành.

Cơ chế liên kết động (dynamic linking)

- Liên kết lúc thực thi tiến trình.
- Một đoạn lệnh nhỏ, gọi là *stub*, dùng để định vị hàm của thư viện đang ở trong bộ nhớ.
- Stub thi hành hàm vừa tìm thấy.
- HĐH cần phải kiểm tra xem hàm cần tìm có nằm trong bộ nhớ của tiến trình hay không.

Overlays

- Chỉ lưu trong bộ nhớ chỉ thị và dữ liệu đang cần.
- Cần thiết khi một tiến trình lớn hơn dung lượng nhớ có thể cấp phát cho nó.

Một số khái niệm

- Khi một chương trình thực hiện, nội dung của nó sẽ được đưa vào bộ nhớ máy tính.
- Khi đó, hệ điều hành phải định ra một vùng nhớ trống cho chương trình hoạt động.
- Thao tác này gọi là location.

Một số khái niệm

- Sau một thời gian hoạt động sẽ có một số chương trình kết thúc→ bộ nhớ bị phân mảnh.
- Một số chương trình khác muốn hoạt động nhưng các khối nhớ trống còn lại không đủ cấp cho nó.
- Khi đó hệ điều hành phải thực hiện thao tác relocation định vị lại các chương trình hiện có để tạo ra khối nhớ trống lớn nhất cho chương trình.

Một số khái niệm

- Cơ chế overlay là cơ chế cho phép chia nhỏ 1 chương trình ra thành nhiều phần.
- Mỗi phần là 1 trang (page). Tại một thời điểm chỉ thực hiện một hoặc một số trang.
- Số trang thực hiện mỗi lần là số **khung trang k**. Các phần còn lại sẽ được để lại ở bộ nhớ ngoài.

Các kỹ thuật thay thế trang trong cơ chế overlay

- Fifo:
  - Ví dụ: A cần thực hiện nội dung các page theo thứ tự sau: 2 9 6 8 2 4 3 7 5 3 9. Với số khung trang k = 3. (Mỗi thời điểm tối đa thực hiện được 3 trang).
  - Đầu tiên 2 được nạp vào bộ nhớ. 2 (trạng thái bộ nhớ)
  - Tiếp theo 9 được nạp vào bộ nhớ. 2 9
  - Tiếp theo 6 được nạp vào bộ nhớ. 2 9 6
  - Sau đó sẽ thay 2 bởi 8. 8 9 6
  - Sau đó sẽ thay 9 bởi 2. 8 2 6
  - ??? 8 2 4
  - ??? 3 2 4
  - ??? 3 7 4
  - ??? 3 7 5
  - ??? 3 7 5
  - ??? 9 7 5

Các kỹ thuật thay thế trang trong cơ chế overlay

- Ít sử dụng nhất trong tương lai:
  - Trang được thay thế là trang ít sử dụng nhất trong tương lai. Nếu 2 trang trong tương lai có số lần xuất hiện bằng nhau và khác 0 thì ta sẽ chọn trang xa nhất để thay thế. Nếu hai trang trong tương lai đều không xuất hiện thì chọn trang đầu tiên tính từ trên xuống.
  - Ví dụ: B cần thực hiện nội dung các page theo thứ tự sau: 2 9 6 8 5 2 4 3 7 5 3 8 9. Với số khung trang k = 3. (Mỗi thời điểm tối đa thực hiện được 3 trang).
  - Đầu tiên 2 được nạp vào bộ nhớ. 2 (trạng thái bộ nhớ)
  - Tiếp theo 9 được nạp vào bộ nhớ. 2 9
  - Tiếp theo 6 được nạp vào bộ nhớ. 2 9 6
  - Sau đó sẽ thay 6 bởi 8. 2 9 8
  - Sau đó sẽ thay 9 bởi 5. 2 5 8
  - ??? 2 5 8
  - ??? 4 5 8
  - ??? 3 5 8
  - ??? 3 5 7
  - ??? 3 5 7
  - ??? 8 5 7
  - ??? 9 5 7

Câu hỏi

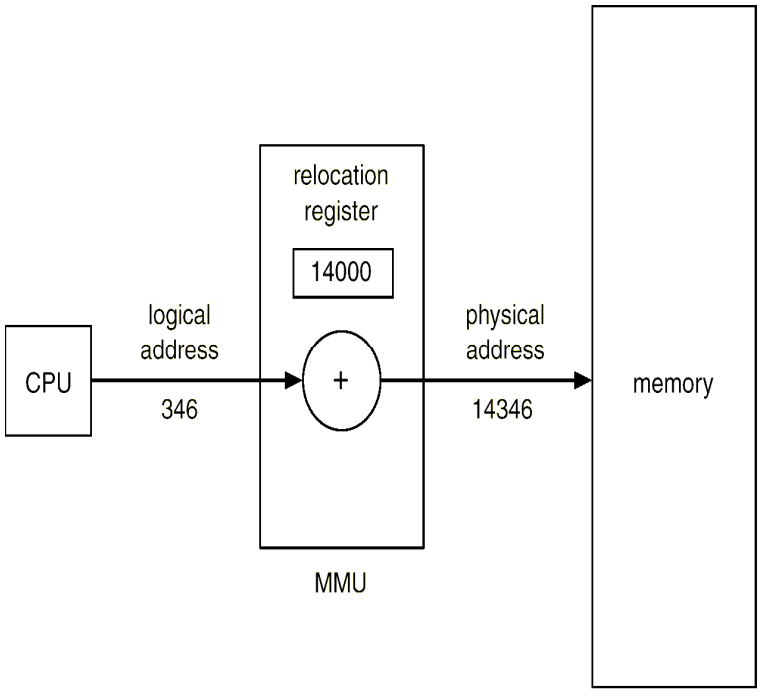
- A cần thực hiện nội dung các page theo thứ tự sau: 2 9 6 8 2 4 3 7 5 3 9. Với số khung trang k = 3. (Mỗi thời điểm tối đa thực hiện được 3 trang). Hãy thực hiện theo kỹ thuật ít sử dụng nhất trong tương lai.
- B cần thực hiện nội dung các page theo thứ tự sau: 2 9 6 8 5 2 4 3 7 5 3 8 9. Với số khung trang k = 3. (Mỗi thời điểm tối đa thực hiện được 3 trang). Hãy hiện thực theo kỹ thuật thay thế trang fifo trong cơ chế overlay.

Không gian địa chỉ lôgic và vật lý

- Khái niệm một *không gian địa chỉ lôgic* được kết buộc với một *không gian địa chỉ vật lý* đóng vai trờ trung tâm trong một việc quản lý bộ nhớ tốt.
  - *Địa chỉ lôgic* – được phát sinh bởi bộ xử lý; còn được gọi là *địa chỉ ảo*.
  - *Địa chỉ vật lý* – địa chỉ được nhìn thấy bởi đơn vị quản lý bộ nhớ.
  - *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 lôgic và địa chỉ vật lý như nhau trong mô hình kết buộc địa chỉ tại thời điểm biên dịch và nạp;
- Địa chỉ lôgic và địa chỉ vật lý khác nhau trong mô hình kết buộc địa chỉ tại thời điểm thi hành.

Đơn vị quản lý Bộ nhớ Memory-Management Unit (MMU)

- Thiết bị phần cứng để ánh xạ địa chỉ ảo thành địa chỉ vật lý.
- Trong mô hình MMU, mỗi địa chỉ phát sinh bởi một tiến trình được cộng thêm giá trị của thanh ghi tái định vị (relocation register) tại thời điểm nó truy xuất đến bộ ngớ.
- Chương trình người dùng chỉ quan tâm đến *địa chỉ lôgic*; nó không thấy *địa chỉ vật lý thật sự*.

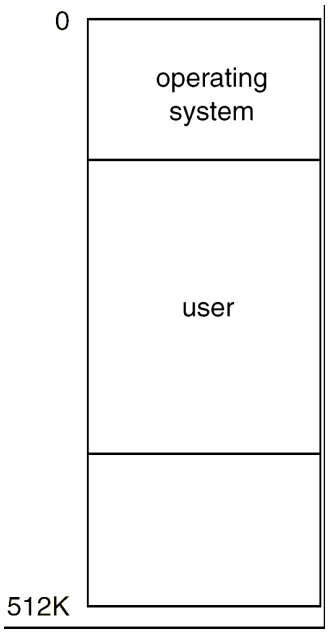


Các kiểu cấp phát vùng nhớ

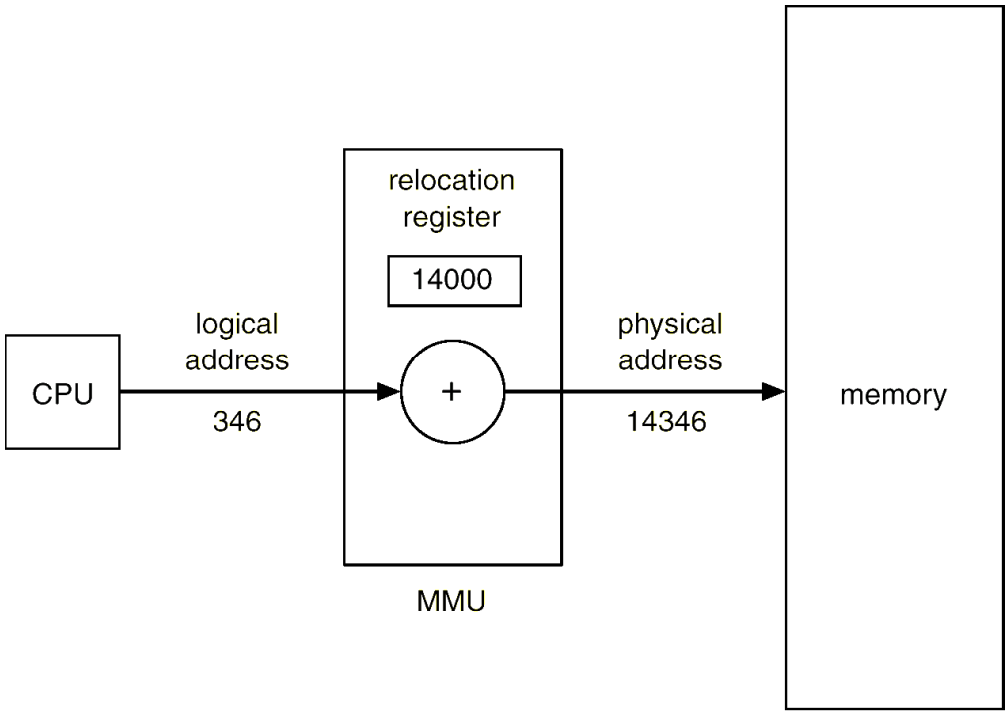
- Cấp phát bộ nhớ liên tục với một phân vùng
- Cấp phát bộ nhớ liên tục với nhiều phân vùng có kích thước cố định
- Cấp phát bộ nhớ liên tục với nhiều phân vùng có kích thước động
- Cấp phát bộ nhớ không liên tục bằng phân trang

Cấp phát liên tục một phân vùng

- Bộ nhớ chính thường chia làm hai phân vùng:
  - Phần chứa HĐH, thường nằm ở vùng nhớ thấp cùng với bảng vector ngắt.
  - Các tiến trình người dùng nằm ở vùng nhớ cao.
- Cấp phát chỉ trên một vùng
  - Thanh ghi tái định vị được dùng để bảo vệ các tiến trình người dùng với nhau và bảo vệ dữ liệu và đoạn mã của HĐH.
  - Thanh ghi tái định vị lưu giá trị địa chỉ vật lý nhỏ nhất; thanh ghi giới hạn (limit register) chứa phạm vi địa chỉ lôgic → mọi địa chỉ lôgic phải nhỏ hơn thanh ghi giới hạn.
  - Thường dùng trong các hệ đơn chương



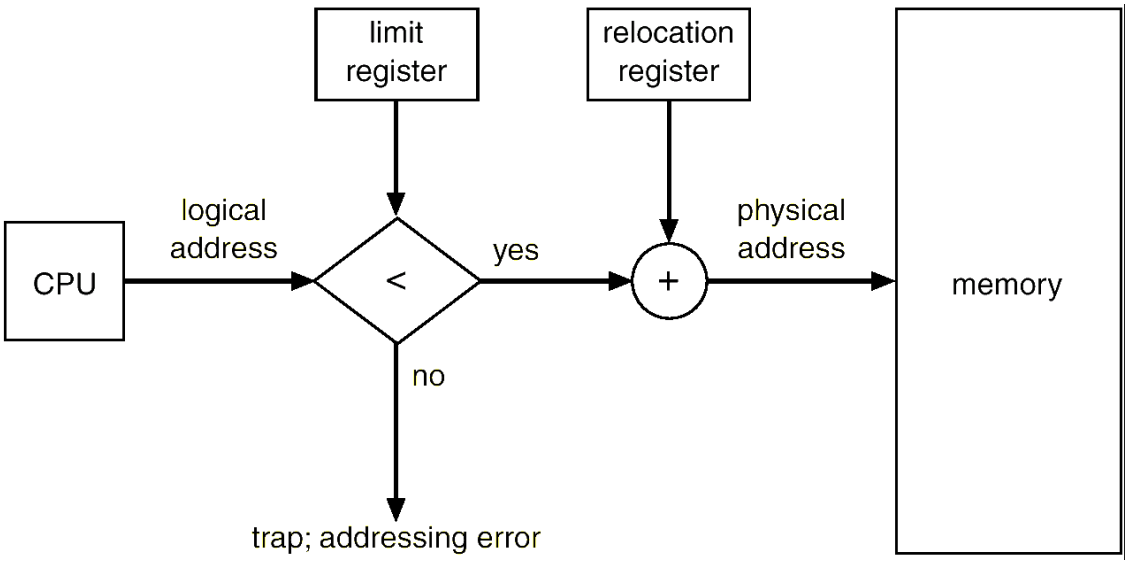
9.03



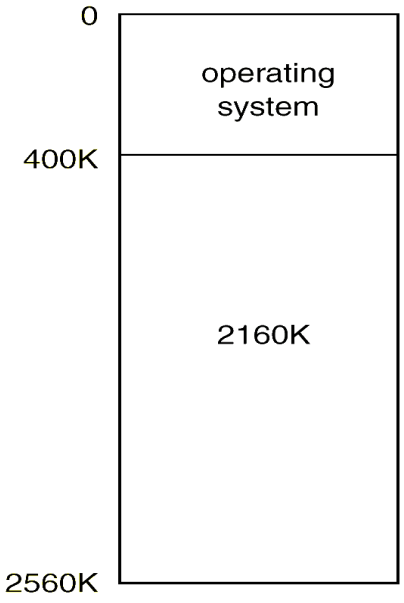
Cấp phát liên tục nhiều phân vùng cố định

- Cấp phát liên tục với nhiều phân vùng cố định
  - Bộ nhớ được phân thành các phân vùng có kích thước cố định (có thể khác nhau hoặc bằng nhau)
  - Khi một tiến trình vào, nó sẽ được cấp phát cho một phân vùng có kích thước đủ để chứa nó.
- Khuyết điểm:
  - Mỗi phân vùng chỉ được lưu một tiến trình → số tiến trình đồng thời bị giới hạn bởi số phân vùng
  - Nếu kích thước của tiến trình không vừa đúng bằng kích thước của phân vùng chứa nó, phần bộ nhớ không được sử dụng sẽ lãng phí → hiện tượng phân mảnh nội vi (internal fragmentation).
  - Sử dụng thanh ghi tái định vị và thanh ghi giới hạn để bảo vệ các tiến trình khỏi bị xâm phạm lẫn nhau.

Cấp phát liên tục nhiều phân vùng cố định



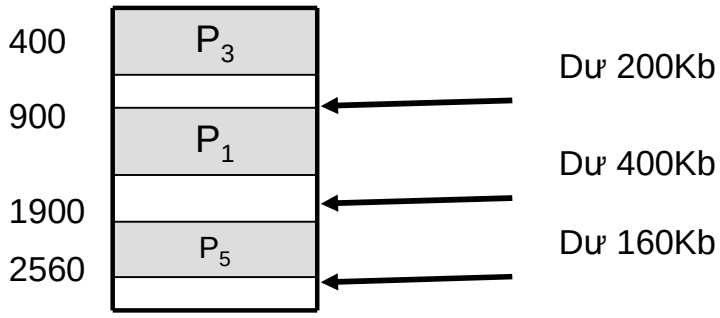
Cấp phát liên tục nhiều phân vùng cố định



job queue

process	memory	time
P_1	600K	10
P_2	1000K	5
P_3	300K	20
P_4	700K	8
P_5	500K	15

Cấp phát liên tục nhiều phân vùng cố định

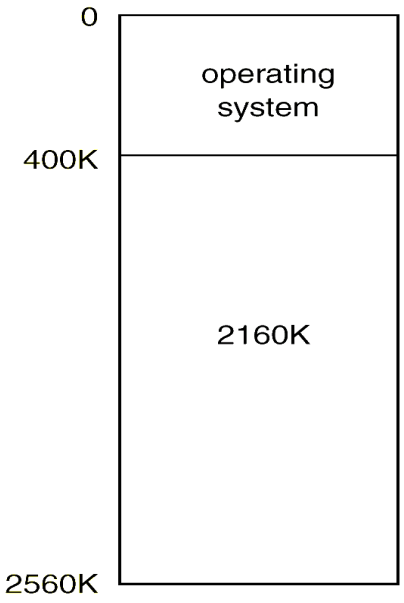


Giả sử các phân vùng cố định là 500 Kb, 1000Kb và 656 Kb.

Cấp phát liên tục nhiều phân vùng động

- Cấp phát liên tục với nhiều phân vùng động
  - Khi một tiến trình vào hệ thống, cấp phát cho nó vùng nhớ vừa đúng với kích thước của tiến trình.
  - Khi tiến trình kết thúc vùng nhớ đó sẽ bị thu hồi và cấp phát cho các tiến trình khác
  - Kích thước của vùng nhớ động
  - Vị trí của vùng nhớ cũng động
- Khuyết điểm:
  - Không còn hiện tượng phân mảnh nội vi nhưng xuất hiện hiện tượng phân mảnh ngoại vi (external fragmentation) → có khả năng tổng vùng nhớ còn trống có thể đủ để cấp phát cho nhu cầu của tiến trình nhưng không thể vì chúng nằm rải rác.
  - Giải pháp cho phân mảnh ngoại vi dồn vùng nhớ → phức tạp, chi phí cao.
  - Nếu trong quá trình xử lý tiến trình có nhu cầu sử dụng thêm vùng nhớ → phải dời chỗ tiến trình hoặc cấp phát thêm nếu còn trống phần ngay sau nó → cần phải có một thuật toán lường trước hoặc tối ưu việc cấp phát vùng nhớ tránh phải dời chỗ tiến trình nhiều.

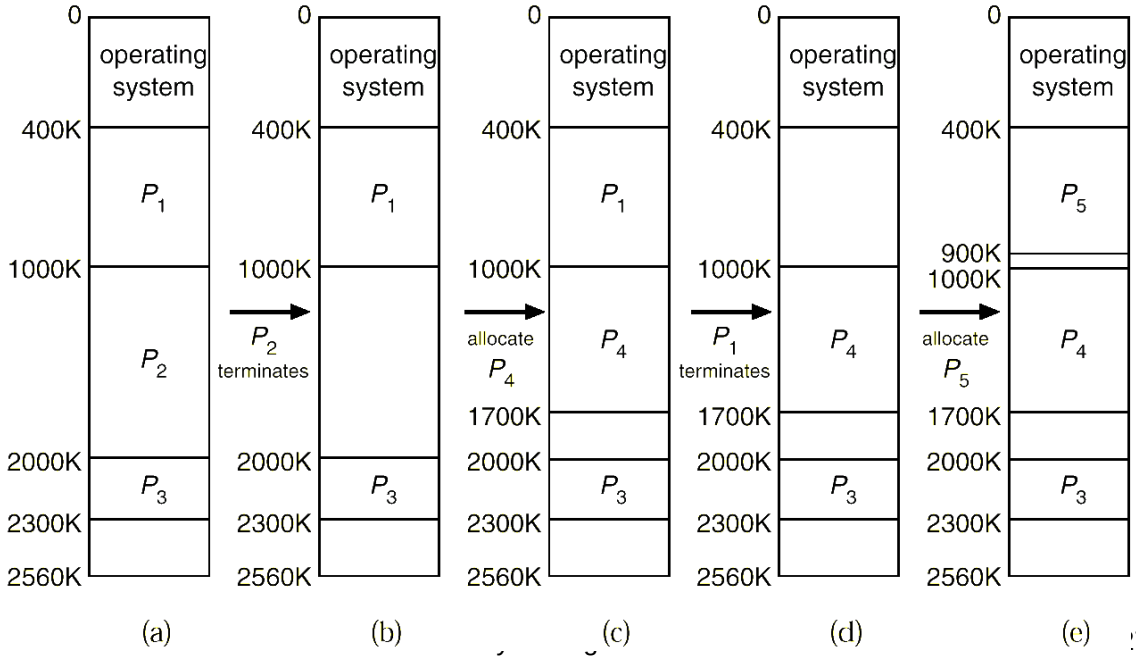
Cấp phát liên tục nhiều phân vùng động



job queue		
process	memory	time
P_1	600K	10
P_2	1000K	5
P_3	300K	20
P_4	700K	8
P_5	500K	15

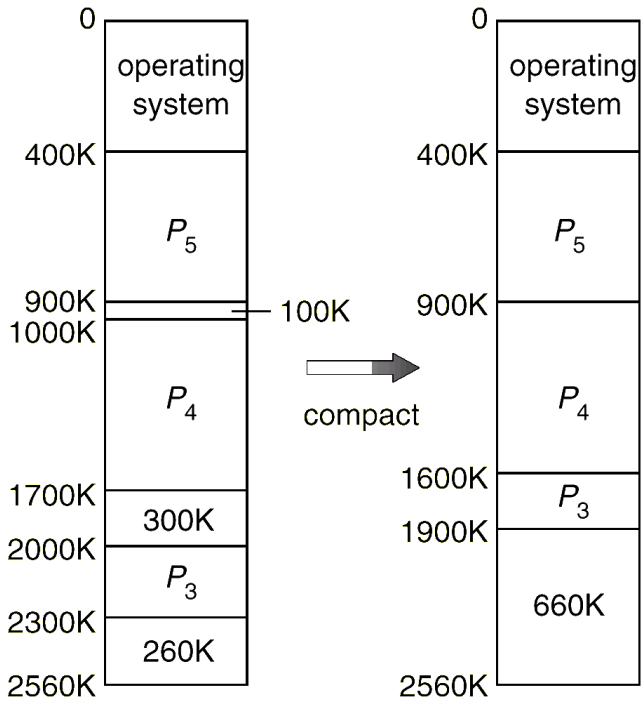
Cấp phát liên tục nhiều phân vùng động

job queue		
process	memory	time
P_1	600K	10
P_2	1000K	5
P_3	300K	20
P_4	700K	8
P_5	500K	15

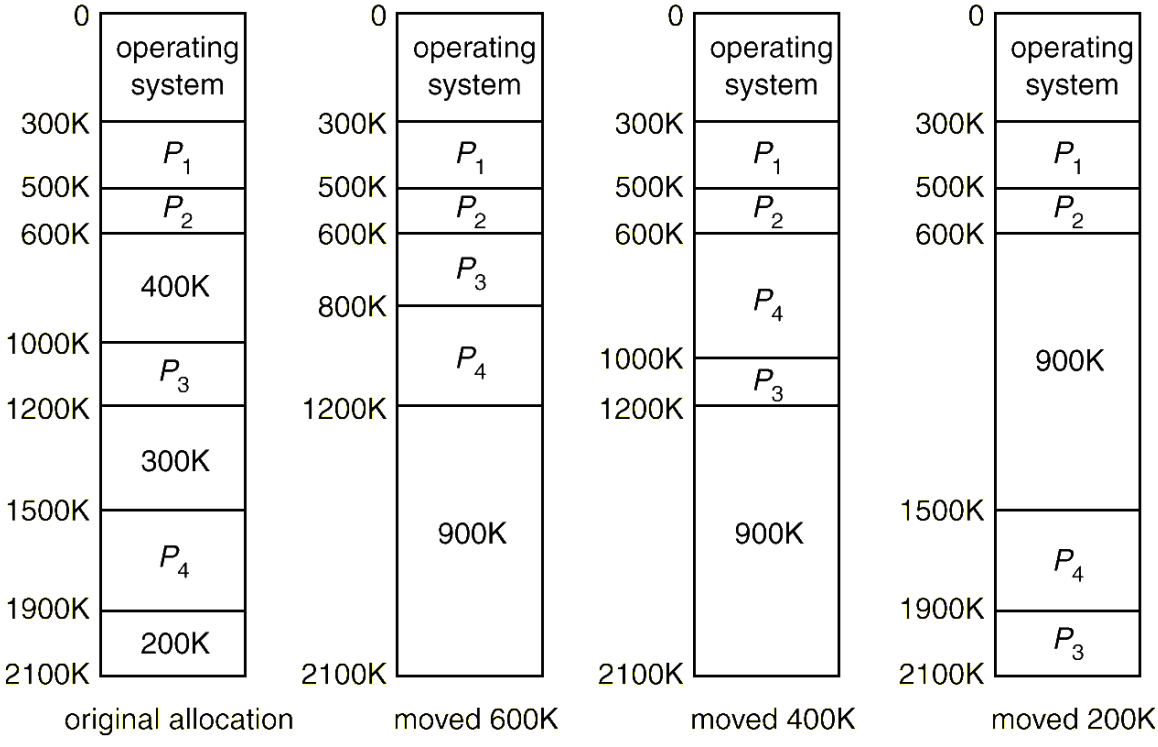


Relocation

job queue		
process	memory	time
P_1	600K	10
P_2	1000K	5
P_3	300K	20
P_4	700K	8
P_5	500K	15



Relocation



Cấp phát liên tục nhiều phân vùng động

Chọn vùng nhớ trống nào để cấp phát cho một tiến trình khi có nhu cầu

- **First-fit**: Cấp phát cho vùng trống đầu tiên có thể chứa được tiến trình.
- **Best-fit**: Cấp phát vùng trống nhỏ nhất có thể chứa được tiến trình; phải tìm kiếm trên toàn bộ danh sách các vùng trống.
- **Worst-fit**: Cấp phát vùng trống lớn nhất; phải tìm kiếm trên toàn bộ danh sách

First-fit tốt hơn về tốc độ và Best-fit tối ưu hóa việc sử dụng bộ nhớ

Câu hỏi bài tập

- Vẽ sơ đồ cấp phát liên tục nhiều phân vùng động theo:
  - First fit ?
  - Best fit ?
  - Worst fit?

  Trong trường hợp của hình bên

job queue

process	memory	time
P_1	600K	10
P_2	1000K	5
P_3	300K	20
P_4	700K	8
P_5	500K	15

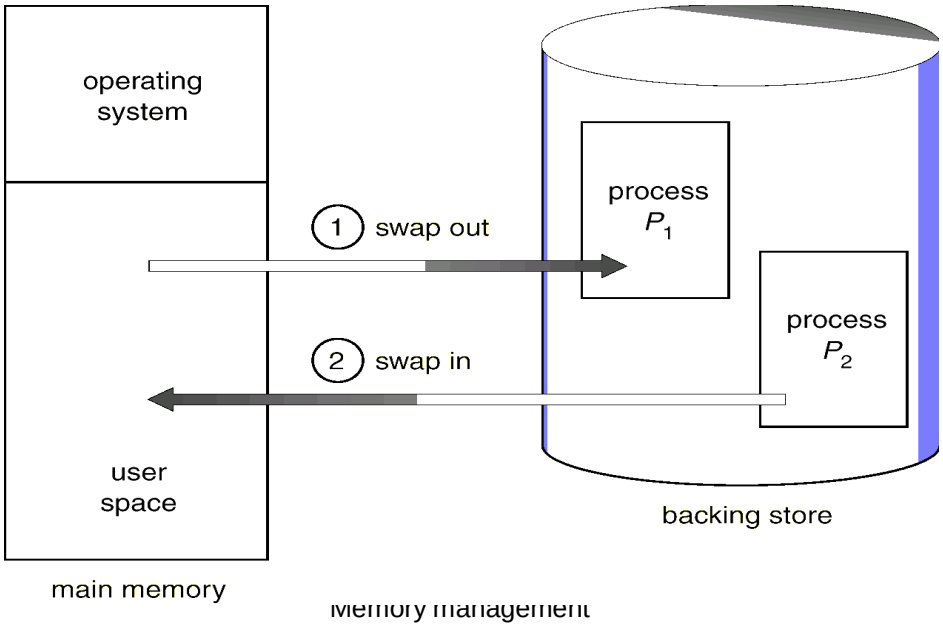
Swapping

- Một tiến trình có thể bị chuyển ra ngoài tạm thời đến một vùng lưu trữ nào đó khi tiến trình đó phải chờ đợi trong khoảng thời gian dài để giải phóng vùng nhớ cho tiến trình khác (swap out).
- Khi tiến trình kết thúc việc chờ, nó có thể được mang vào lại bộ nhớ chính để xử lý (swap in).
- Thao tác swap cũng có thể xảy ra khi dùng các thuật toán điều phối tiến trình có độ ưu tiên

Swapping

- Nâng cao mức độ đa chương
- Khi tiến trình được mang lại bộ nhớ chính, nó sẽ được nạp vào vùng nhớ nào?
  - Kết buộc lúc nạp → phải đưa vào vùng nhớ cũ của nó
  - Kết buộc lúc thi hành → thay đổi thanh ghi tái định vị
- Cần sử dụng bộ nhớ phụ đủ lớn để lưu các tiến trình bị khóa.
- Thời gian swap chủ yếu là thời gian chuyển tiến trình từ vùng nhớ chính đến bộ nhớ phụ → một tiến trình cần phải có khoảng thời gian xử lý đủ lớn.

Mô hình thao tác swapping



Khuyết điểm của các pp cấp phát liên tục

- Xảy ra hiện tượng phân mảnh bộ nhớ:
  - Phân mảnh nội vi (internal fragmentation)
    - Kích thước phân vùng cố định
  - Phân mảnh ngoại vi (external fragmentation)
    - Kích thước phân vùng động

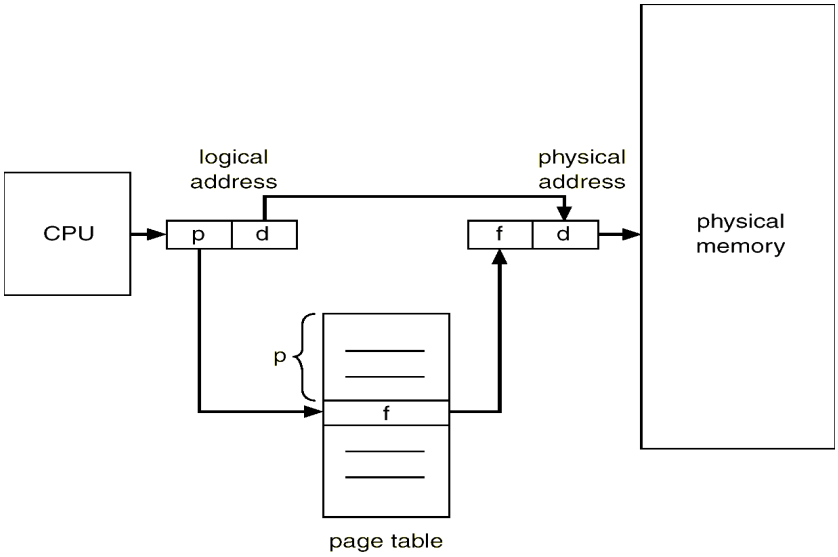
Cấp phát bộ nhớ bằng pp phân trang (Paging)

- Không gian địa chỉ lôgic của một tiến trình có thể không liên tục.
- Chia bộ nhớ vật lý thành các khối có kích thước cố định gọi là khung (frame) (kích thước là số mũ của 2, từ 512 đến 8192 bytes).
- Chia bộ nhớ lôgic thành các khối có cùng kích thước và gọi là trang (pages).
- Lưu trạng thái của tất cả các khung (frame).
- Để chạy một chương trình có kích thước n trang, cần phải tìm n khung trống và nạp chương trình vào.
- Tạo một bảng trang để chuyển đổi địa chỉ lôgic sang địa chỉ vật lý.
- Có hiện tượng phân mảnh bộ nhớ nội vi.

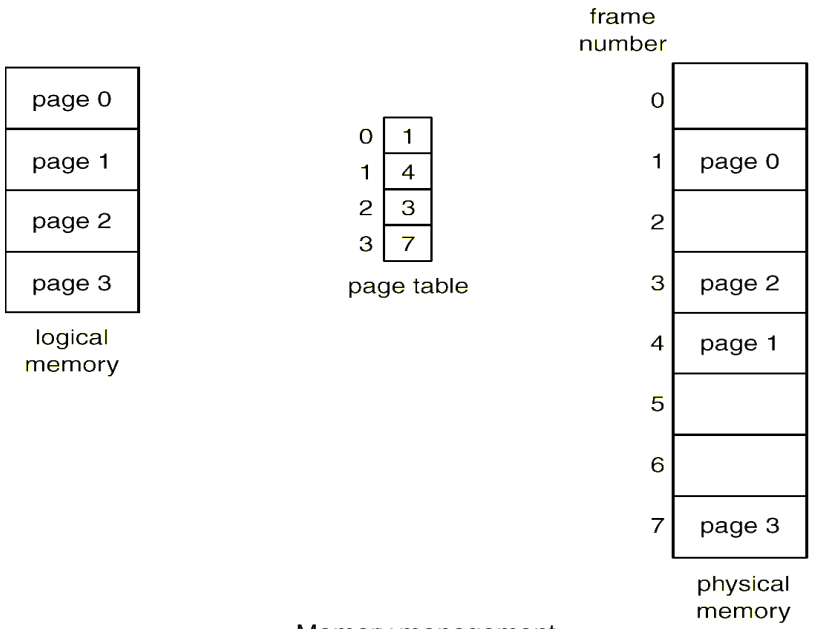
Mô hình chuyển đổi địa chỉ

- Địa chỉ được tạo ra bởi CPU gồm có hai phần:
  - *Số trang (Page number) (p)* – được dùng như là một chỉ số của một bảng trang chứa địa chỉ cơ sở của mỗi trang trong bộ nhớ vật lý.
  - *Page offset (d)* – kết hợp với địa chỉ cơ sở để định ra không gian địa chỉ vật lý được gởi đến bộ nhớ.

Kiến trúc chuyển đổi địa chỉ

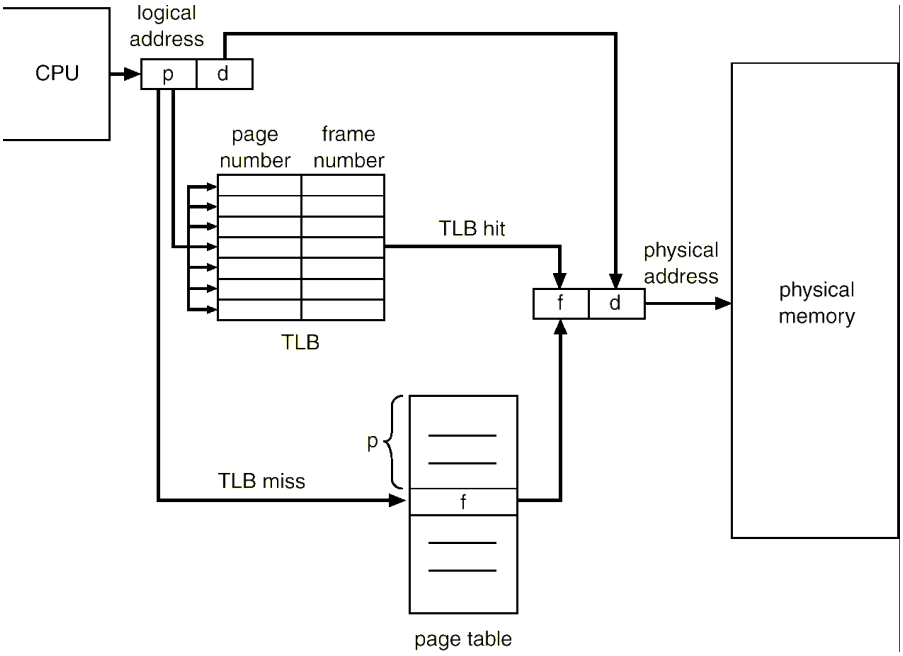


Ví dụ trang



Cài đặt bảng trang

- Bảng trang được đặt trong bộ nhớ.
- *Page-table base register (*PTBR) chỉ đến bảng trang.
- *Page-table length register* (PRLR) cho biết kích thước của bảng trang.
- Với mô hình này, mọi sự truy cập chỉ thị/dữ liệu đều đòi hỏi hai lần truy cập vùng nhớ: 1) truy cập bảng trang; 2) chỉ thị hoặc dữ liệu → có vẻ chậm.
- Khắc phục vấn đề hai lần truy cập vùng nhớ bằng cách sử dụng một vùng đệm phần cứng tra cứu nhanh đặc biệt (special fast-lookup hardware) gọi là *associative registers* hoặc *translation look-aside buffers (TLBs)*



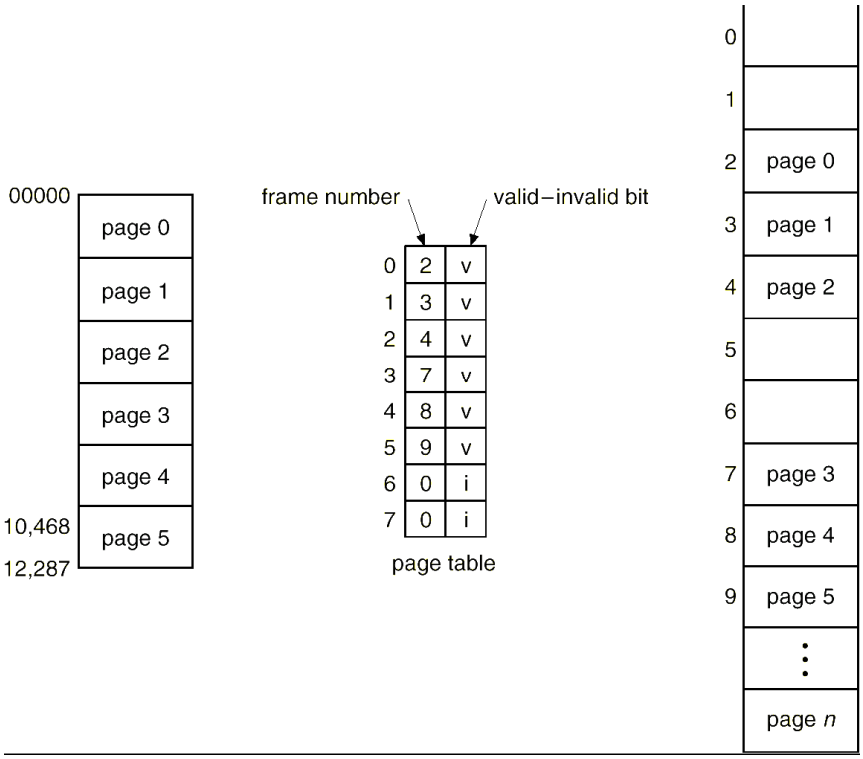
Effective Access Time

- Thời gian tìm kiếm trong associate registers = ε đơn vị thời gian
- Giả sử thời gian truy cập bộ nhớ là 1 ms
- Gọi α là tỉ lệ số lần số trang được tìm thấy trong các associative registers.
- Effective Access Time (EAT)

$$EAT = (1 + \varepsilon)\,\alpha + (2 + \varepsilon)(1 - \alpha)$$
$$= 2 + \varepsilon - \alpha$$

Bảo vệ vùng nhớ

- Làm sao biết trang nào của tiến trình nào? Cần bảo vệ các tiến trình truy xuất vào trang không phải của mình.
- Việc bảo vệ vùng nhớ được cài đặt bằng cách liên kết một khung với một bit, gọi là bit kiểm tra hợp lệ (valid-invalid bit).
- *Valid-invalid* bit được đính kèm vào mỗi ô trang bảng trang:
  - "valid" chỉ ra rằng trang đi kèm là nằm trong không gian địa chỉ lôgic của tiến trình vì vậy truy xuất trang này là hợp lệ.
  - "invalid" chỉ ra rằng trang đi kèm không nằm trong không gian địa chỉ lôgic của tiến trình.



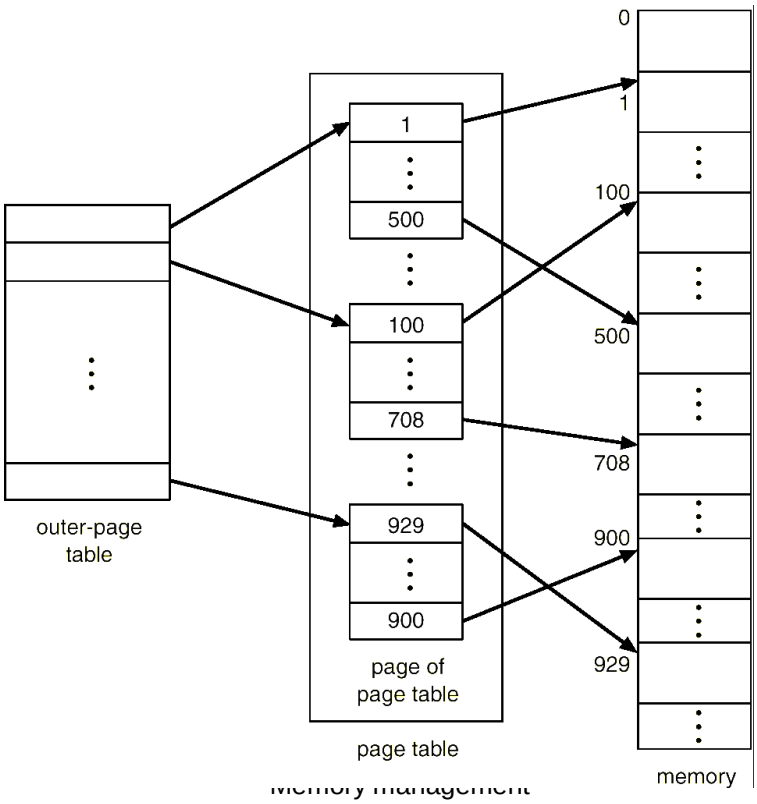
Tổ chức bảng trang

- Chỉ một bảng trang (cho mỗi tiến trình)
- Tiến trình nhiều trang → quản lý bộ nhớ rất lớn → số trang nhiều → kích thước của bảng trang phải lớn → không tối ưu.
- Giải pháp:
  - Phân trang đa cấp (multilevel paging)
  - Bảng trang nghịch đảo

Phân trang đa cấp

- Phân chia bảng trang thành các phần nhỏ
- Bản thân bảng trang cũng được phân trang

Mô hình phân trang hai cấp



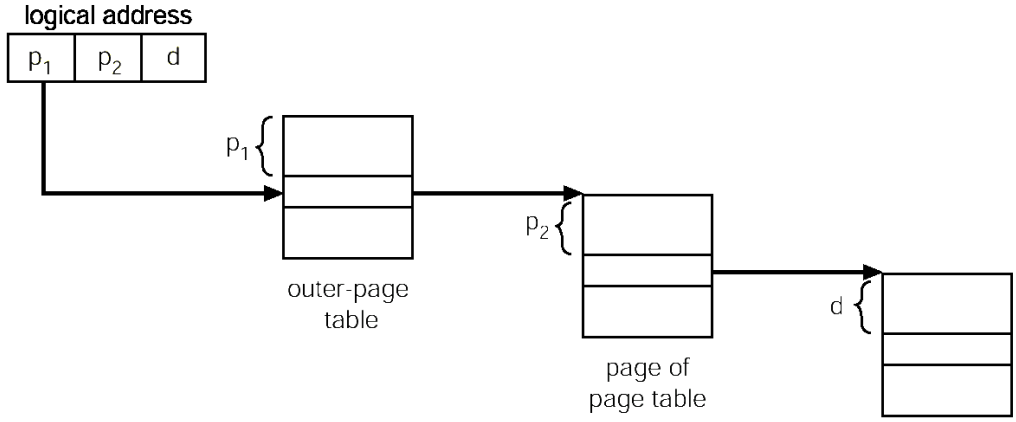
Ví dụ phân trang 2 cấp

- Một địa chỉ lôgic (trên máy 32 bit với kích thước trang là 4K) được chia thành:
  - Page number: 20 bit.
  - Page offset: 12 bit.
- Bởi vì bảng trang được phân trang, số trang tiếp tục được phân chia thành:
  - Page number: 10-bit.
  - Page offset: 10 bit.
- Địa chỉ lôgic sẽ có cấu trúc như sau:

page number		page offset
p_i	p_2	d
10	10	12

Chuyển đổi địa chỉ

- Address-translation scheme for a two-level 32-bit paging architecture



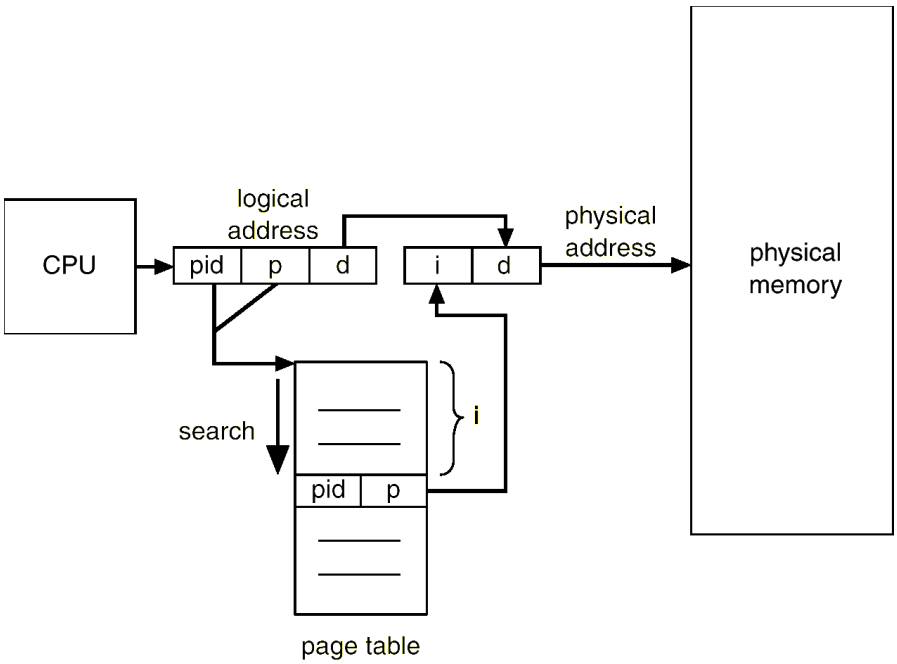
Phân tích

- Nếu chỉ dùng một trang
  - bảng trang sẽ có 2^{20} phần tử.
- Nếu dùng phân trang 2 cấp
  - Bảng trang ngoài cần 2^{10} phần tử
  - Bảng trang trong vẫn có 2^{20} phần tử nhưng có thể được đặt ở vùng nhớ phụ

Bảng trang nghịch đảo

- Sử dụng một bảng trang duy nhất cho mọi tiến trình
- Một entry cho mỗi khung (bộ nhớ vật lý).
- Một entry bao gồm địa chỉ ảo của khung và tiến trình sở hữu khung đó.
- Giảm kích thước lưu trữ bảng trang , nhưng tăng thời gian tìm kiếm bảng trang.
- Dùng bảng băm để tăng tốc tìm kiếm.

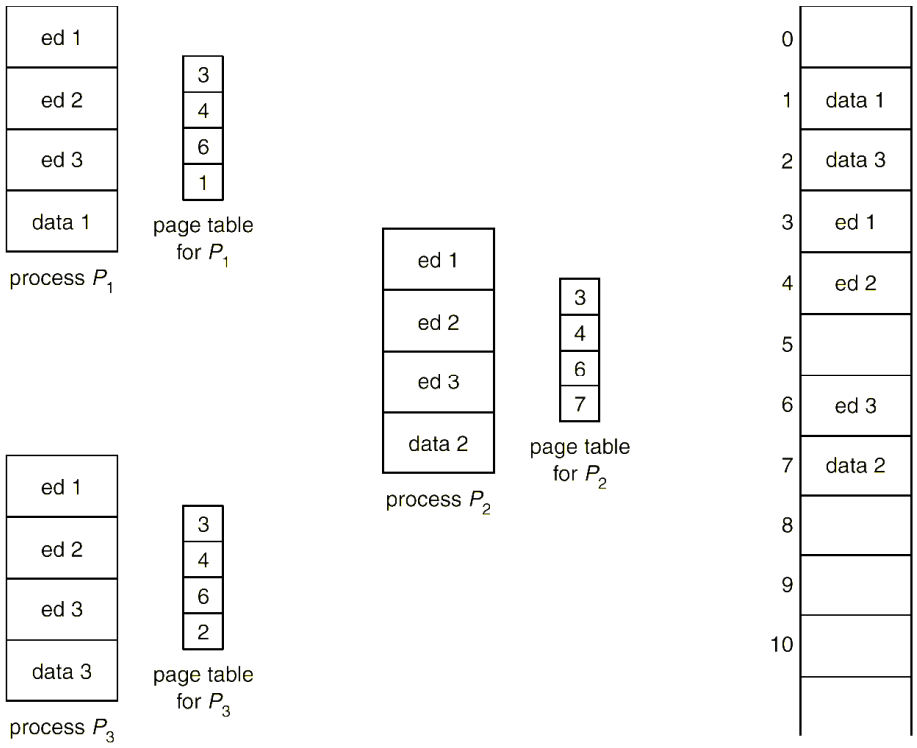
Kiến trúc bảng trang nghịch đảo



Chia sẻ và bảo vệ

- Chia sẻ và bảo vệ
  - Ở cấp độ trang
  - Đứng ở khía cạnh người dùng, khá bất tiện
    - Vd: Đoạn mã nằm trên nhiều trang → phải chia sẻ tất cả các trang đó với nhau.
  - Đoạn mã phải nằm ở một vị trí giống nhau trông tất cả các không gian địa chỉ của của tiến trình muốn chia sẻ

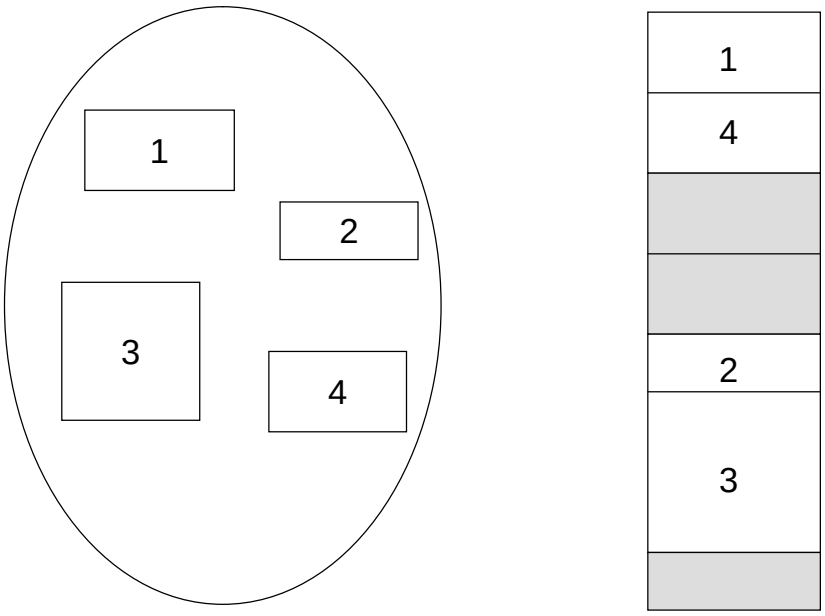
Ví dụ chia sẻ trang



Phân đoạn

- Hỗ trợ quản lý bộ nhớ theo góc độ người dùng.
- Một chương trình bao gồm nhiều phân đoạn (segment):
  - Đoạn mã
  - Biến toàn cục, dữ liệu
  - stack
  - heap

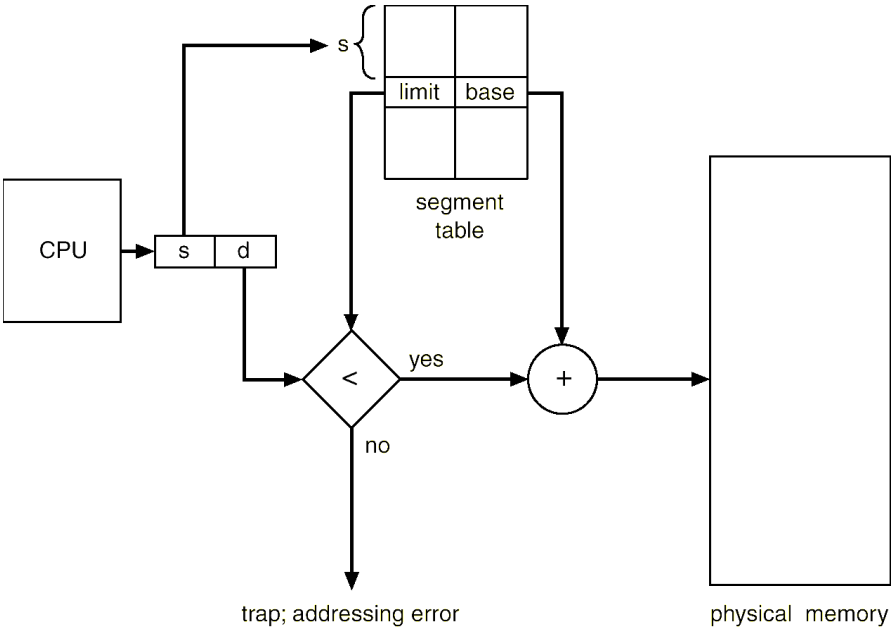
Góc độ người dùng



Kiến trúc phân đoạn

- Địa chỉ lôgic = <segment-number, offset>
- *Bảng phân đoạn (Segment table)* – chuyển đổi địa chỉ hai chiều thành địa chỉ vật lý một chiều. Mỗi ô trong bảng gồm có:
  - Thanh ghi nền (base) – chứa địa chỉ vật lý bắt đầu của phân đoạn trong bộ nhớ chính.
  - *Thanh ghi giới hạn (limit)* – chỉ kích thước của phân đoạn.
- *Segment-table base register (STBR)* lưu trữ địa chỉ của bảng phân đoạn trong vùng nhớ.
- *Segment-table length register (STLR)* chỉ số segment được sử dụng bởi 1 chương trình

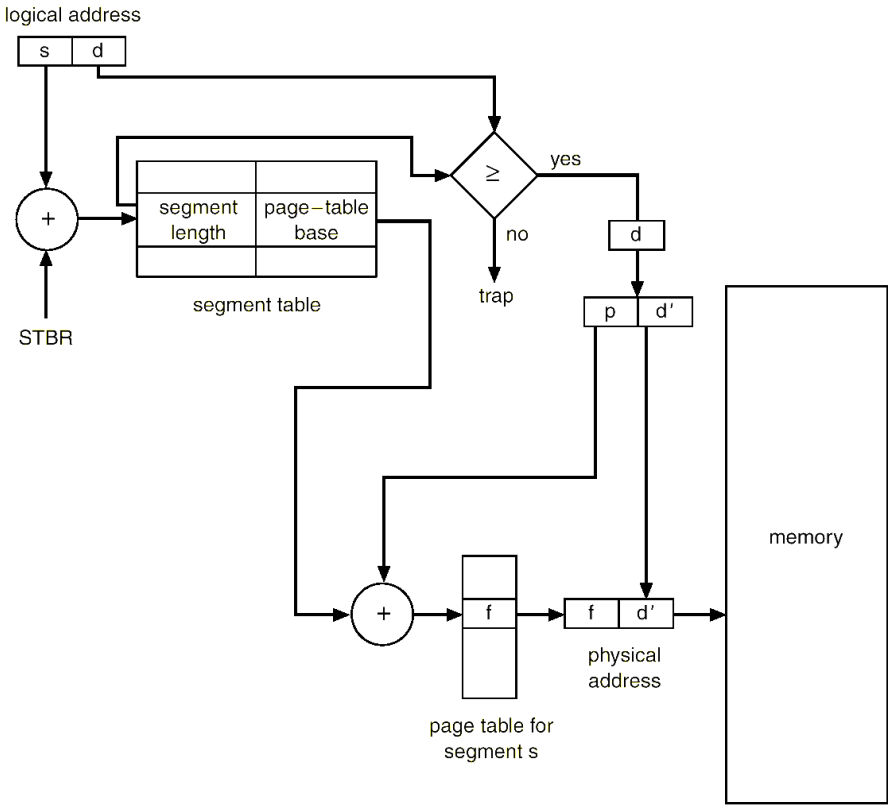
Chuyển đổi địa chỉ



Kiến trúc phân đoạn (tt)

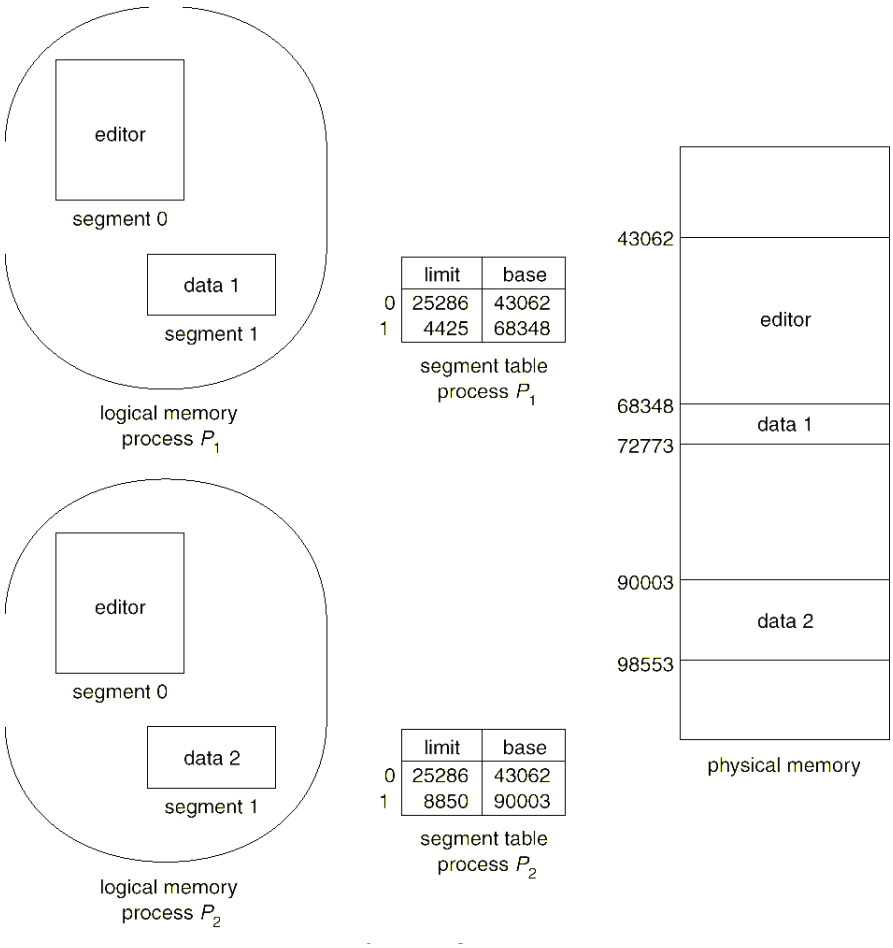
- Tái định vị.
  - Động (dynamic partition)
  - Thông qua bảng phân đoạn
- Chia sẻ.
  - Có thể chia sẻ các phân đoạn giữa các chương trình
  - Sử dụng chung chỉ số sement
- Cấp phát.
  - first fit/best fit
  - Có hiện tượng phân mảnh ngoại vi

Chuyển đổi địa chỉ



Kiến trúc phân đoạn (tt)

- Bảo vệ
  - Mỗi entry thêm một bit “valid bit”. Nếu valid bit = 0 $\Rightarrow$ truy cập phân đoạn không hợp lệ
  - Hỗ trợ phân quyền theo từng thao tác read/write/execute



Kết hợp phân trang và phân đoạn

- Ý tưởng:
  - Phân trang trong mỗi phân đoạn
  - Bộ nhớ = nhiều phân đoạn
  - Phân đoạn = nhiều trang
- Giải quyết tình trạng phân mảnh ngoại vi
- Mỗi phần tử của bảng phân đoạn gồm hai thành phần:
  - Thanh ghi giới hạn (limit): kích thước của phân đoạn (giống với phân đoạn thuần)
  - Thanh ghi cơ sở (base): chứa địa chỉ của bảng trang của phân đoạn đó (khác với phân đoạn thuần)

Các tiêu chuẩn so sánh các pp quản lý vùng nhớ

- Hỗ trợ phần cứng
- Hiệu năng
- Sự phân mảnh
- Khả năng tái định vị
- Hỗ trợ swap
- Chia sẻ
- Bảo vệ

Bộ nhớ ảo

Trần Sơn Hải
Khoa Toán – Tin học
Đại học Sư phạm TPHCM

Heavily reference to Operating System Slide of Hoang Than Anh Tuan, University of Pedagogy, HCMC

Bộ nhớ ảo: Ý tưởng

- Hai đặc trưng quan trọng của kiến trúc phân đoạn và phân trang:
  - Mọi sự truy xuất vùng nhớ của một tiến trình đều được chuyển đổi địa chỉ lúc thi hành (run-time) → có thể swap-in, swap-out.
  - Một tiến trình được phân ra thành một số phần (trang hoặc đoạn) và không nhất thiết phải nằm liên tục nhau

Bộ nhớ ảo: Ý tưởng (tt)

- Nếu hai tính chất trên được bảo đảm thì không nhất thiết tất cả các trang hoặc phân đoạn phải nằm trong bộ nhớ chính lúc thi hành.
- Ưu điểm:
  - Có nhiều tiến trình trong bộ nhớ hơn → giải thuật lập lịch sẽ tối ưu hơn → nâng cao mức độ đa chương
  - Một tiến trình có thể lớn hơn kích thước của bộ nhớ chính

Nguyên lý cục bộ

Các thao tác truy cập vùng nhớ có khuynh hướng cụm lại (cluster).
Sau một khoảng thời gian đủ dài, cụm này có thể sẽ thay đổi, nhưng trong một khoảng thời gian ngắn, bộ xử lý chủ yếu chỉ làm việc trên một số cụm nhất định.
Sau một khoảng thời gian đủ dài, cụm này có thể sẽ thay đổi, nhưng trong một khoảng thời gian ngắn, bộ xử lý chủ yếu chỉ làm việc trên một số cụm nhất định.

Giải thích

Các câu lệnh cơ bản chủ yếu là tuần tự (thi hành từ trên xuống dưới). Câu lệnh không tuần tự là câu lệnh rẽ nhánh (câu lệnh điều kiện) thường chiếm tỉ lệ khá ít.

Trong một khoảng thời gian ngắn, các chỉ thị thông thường nằm trong một số hàm, thủ tục nhất định.

Hầu hết các câu lệnh lặp chứa một số ít các chỉ thị và lặp lại nhiều lần. Do đó trong suốt thời gian lặp, việc tính toán hầu như chỉ diễn ra trong một vùng nhỏ liên tục.

Khi truy cập vào một cấu trúc dữ liệu trước đó, thông thường các câu lệnh đặt liền nhau sẽ truy cập đến các thành phần khác nhau của cùng một cấu trúc dữ liệu.

Các vấn đề liên quan đến bộ nhớ ảo

- Cần có sự hỗ trợ phần cứng về kiến trúc phân trang và phân đoạn
  - Đã khảo sát
- Cần có thuật toán hiệu quả để quản lý việc chuyển đổi các trang, phân đoạn từ bộ nhớ chính vào bộ nhớ phụ và ngược lại
  - Nguyên lý cục bộ
  - Đĩa cứng hoạt động theo khối
  - Dự đoán được các trang và phân đoạn dựa vào lịch sử truy xuất vùng nhớ trước đó.

Quản lý việc chuyển đổi giữa vùng nhớ chính và vùng nhớ phụ

- Các chính sách cần xét:
  - Chính sách **nạp** (fetch policy): khi nào thì một trang được nạp vào bộ nhớ?
  - Chính sách **đặt** (placement policy): trang hoặc phân đoạn sẽ được đặt ở đâu trong bộ nhớ chính?
  - Chính sách **thay thế** (replacement policy): chọn trang nào đưa ra khỏi bộ nhớ phụ khi cần nạp một trang mới vào bộ nhớ chính?

Cài đặt bộ nhớ ảo

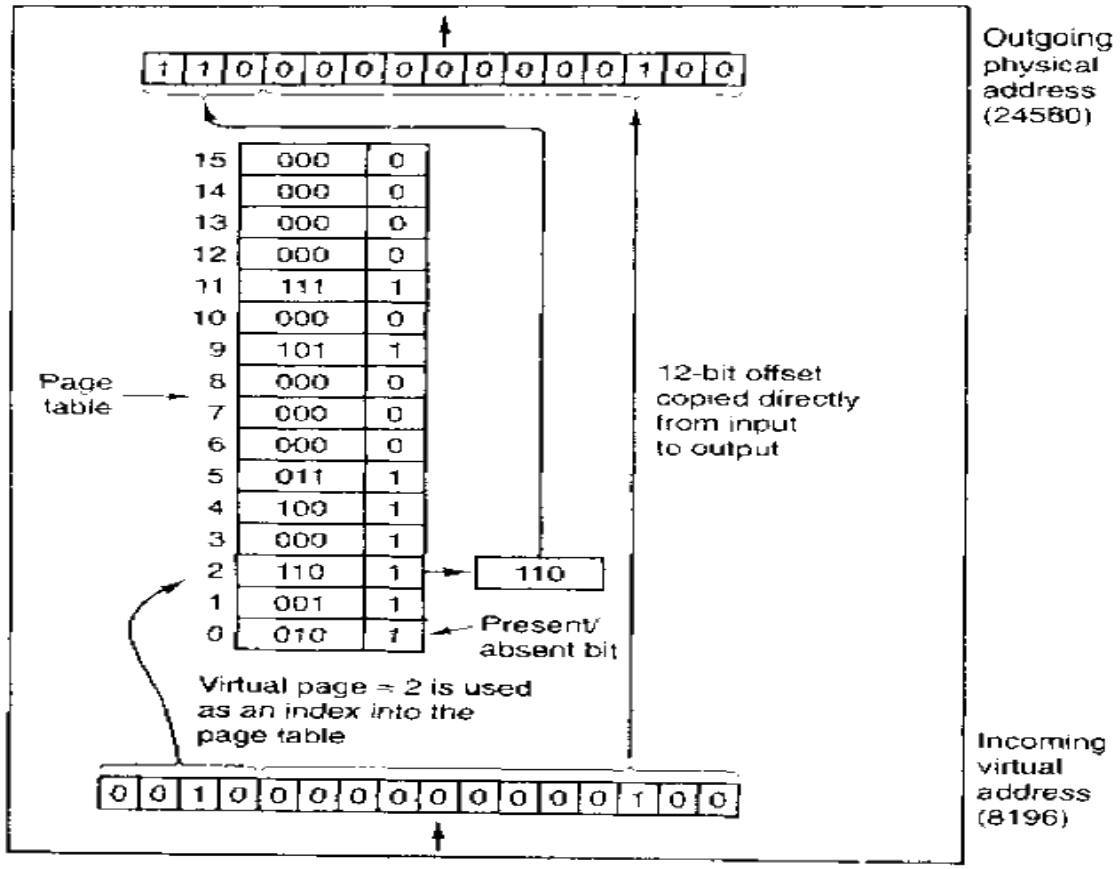
- Kỹ thuật phân trang theo yêu cầu (demand paging)
- Kỹ thuật phân đoạn theo yêu cầu (demand segmentation)
  - Khó vì kích thước không đồng nhất

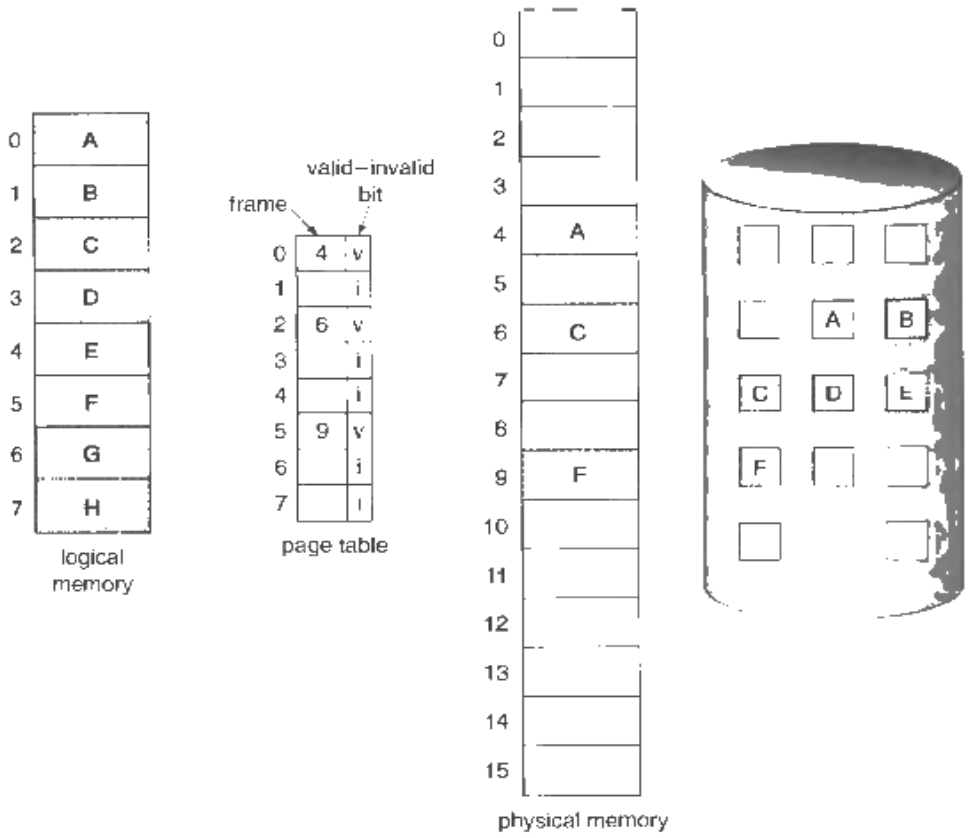
Kỹ thuật phân trang theo yêu cầu

- Phân trang theo yêu cầu = Phân trang + swapping
- Tiến trình là một tập các trang thường trú trên bộ nhớ phụ.
- Một trang chỉ được nạp vào bộ nhớ chính khi có yêu cầu.
- Khi có yêu cầu về một trang nào đó, cần có cơ chế cho biết trang đó đang ở trên đó hoặc ở trong bộ nhớ
  - Sử dụng bit valid/invalid
  - Valid: có trong bộ nhớ chính
  - Invalid: trang không hợp lệ hoặc trang đang nằm trong bộ nhớ phụ

Cơ chế phần cứng

- Bảng trang
  - Phải phản ánh được một trang đang nằm trong bộ nhớ chính hay bộ nhớ phụ và tương ứng đang nằm ở vị trí nào (trong bộ nhớ chính hoặc bộ nhớ phụ)
- Bộ nhớ phụ
  - Dùng một không gian trên đĩa cứng thường gọi là không gian swapping.



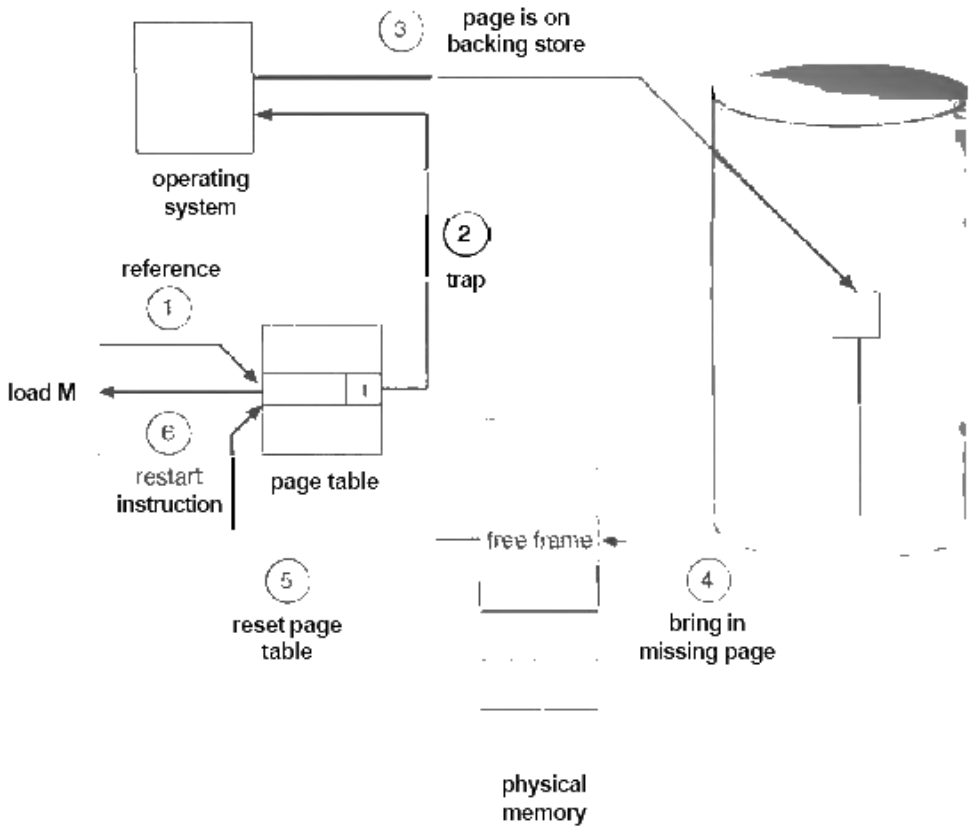


Quá trình xử lý một trang không có trong bộ nhớ chính (lỗi trang)

1. Kiểm tra trang được truy xuất có hợp lệ hay không?
2. Nếu truy xuất không hợp lệ → kết thúc

Ngược lại → bước 3

3. Tìm vị trí chứa trang muốn truy xuất trên đĩa cứng.
4. Tìm một khung trang trống trên bộ nhớ chính
    1. Nếu tìm thấy → bước 5
    2. Nếu không tìm thấy khung trang trống, tìm một khung trang "nạn nhân" và chuyển nó ra bộ nhớ phụ, cập nhật bảng trang
5. Chuyển trang muốn truy xuất từ bộ nhớ phụ vào bộ nhớ chính, cập nhật bảng trang, bảng khung trang.
6. Tái kích hoạt tiến trình tại chỉ thị truy xuất đến trang



Thay thế trang

- Là cơ chế thay thế một trang đang nằm trong bộ nhớ nhưng chưa cần sử dụng bằng một trang đang nằm trong đĩa (không gian swapping) đang được yêu cầu.
- Hai thao tác:
  - Chuyển trang từ bộ nhớ chính ra bộ nhớ phụ
  - Mang trang từ bộ nhớ phụ vào vào nhớ chính
- Giảm số lần thao tác bằng bit cập nhập (dirty bit)
  - Bit cập nhật = 1: nội dung trang đã bị thay đổi → cần lưu lại trên đĩa
  - Bit cập nhật = 0: nội dung trang không bị thay đổi → không cần lưu lại trên đĩa

Một phần tử trong bảng trang

Page number	Valid bit	Dirty bit

Hiệu quả của phân trang theo yêu cầu

- Gọi p là xác suất xảy ra lỗi trang:
  - P = 0: không có lỗi trang
  - P = 1: mỗi truy xuất đến bộ nhớ đều có lỗi trang
- MA: thời gian truy xuất đến bộ nhớ chính
- TDP: thời gian xử lý lỗi trang = [+ swapout] + swapin + tái kích hoạt
- TEA: thời gian thật sự để thực hiện một truy xuất bộ nhớ

TEA = (1-p)MA + p*TDP

- Mấu chốt là p: p càng thấp thì hiệu quả của phân trang theo yêu cầu càng cao

Thuật toán thay thế trang

- Ý tưởng chính:
  - Chọn trang nạn nhân là trang mà sau khi thay thế sẽ gây ra ít lỗi trang nhất.
- Các thuật toán:
  - FIFO
  - Tối ưu (ít sử dụng nhất trong tương lai)
  - LRU
  - Xấp xỉ LRU

Thuật tóan FIFO

- Ý tưởng:
  - Ghi nhận thời điểm một trang được đưa vào bộ nhớ
  - Thay thế trang ở trong bộ nhớ lâu nhất
- Có thể không cần ghi nhận thời điểm đưa mộ trang vào bộ nhớ. Sử dụng danh sách trang theo kiểu FIFO → trang thay thế luôn là trang đầu
- Dễ hiểu, dễ cài đặt, nhưng không lôgic trong trường hợp những trang đầu tiên được nạp vào thường những trang quan trọng, chứa dữ liệu truy xuất thường xuyên → chuyển nó ra sẽ gây lỗi trang cho những lần truy xuất sau
- Nghịch lý Belady: số lượng lỗi trang sẽ tăng lên nếu số lượng khung trang tăng lên.

Thuật toán tối ưu

- Ý tưởng:
  - Thay thế trang sẽ được lâu sử dụng nhất trong tương lai.
- Hoàn hảo về mặt ý tưởng nhưng không khả thi về mặt thực tế
  - Làm sao dự đoán được chuỗi các trang truy xuất trong tương lai.

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

- Ý tưởng:
  - Ghi nhận thời điểm cuối cùng trang được truy cập
  - Thay thế trang chưa được truy cập lâu nhất
- Dùng quá khứ gần để dự đoán tương lai
  - FIFO: thời điểm nạp vào
  - Tối ưu: thời điểm sẽ truy cập

Least-recently-used LRU

- Các cách cài đặt:
  - Sử dụng bộ đếm
    - Mỗi phần tử trong bảng trang có một thành phần ghi nhận thời điểm truy xuất mới nhất.
    - CPU có một bộ đếm, tăng khi có một truy xuất đến bộ nhớ
    - Cập nhật thời điểm theo bộ đếm
    - Trang có thời điểm truy xuất nhỏ nhất sẽ bị thay thế
  - Sử dụng stack
    - Lưu các số hiệu trang
    - Khi một trang được truy xuất → chuyển số hiệu trang lên đầu stack
    - Thay thế trang có số hiệu ở đáy stack

- LRU đòi hỏi phần cứng hỗ trợ khá nhiều
  - Biến bộ đếm
  - Stack
- Tìm các thuật toán xấp xỉ LRU

Thuật toán xấp xỉ LRU

- Có 3 thuật toán
  - Sử dụng nhiều bit tham khảo (reference bit)
  - Cơ hội thứ hai
  - Cơ hội thứ hai cải tiến

- Ý tưởng chính: bit tham khảo được thêm vào mỗi phần tử trong bảng trang
  - Ban đầu = 0
  - Có truy xuất → 1
  - Sau mỗi chu kỳ qui định trước, kiểm tra bit này và gán nó trở lại là 0.
  - Biết được trang nào đã được truy xuất gần đây nhưng không biết được thứ tự truy xuất.

Thuật toán nhiều bit tham khảo

- Ý tưởng:
  - 1 bit tham khảo chỉ biết được thông tin của 1 chu kỳ
  - Nhiều bit tham khảo sẽ biết được thông tin của nhiều chu kỳ.
- Sử dụng thêm 8 bit tham khảo cho mỗi phần tử trong bảng trang
- Sau một chu kỳ, một ngắt được phát sinh, HĐH sẽ đặt bit tham khảo của trang đó (0 hoặc 1) vào bit cao nhất trong 8 bit, loại bỏ bit cuối cùng (thấp nhất)
- 8 bit sẽ lưu trữ tình hình truy xuất đến trang trong 8 chu kỳ gần nhất
- 10001000 sẽ tốt hơn 01111111
- Nếu xem là số nguyên không dấu thì trang được thay thế là trang có số tương ứng nhỏ nhất.

Thuật toán cơ hội thứ hai

- Ý tưởng:
  - Sử dụng một một bit tham khảo duy nhất
  - Ý tưởng như FIFO có cải tiến
    - Nếu bit tham khảo = 0 → thay thế trang
    - Ngược lại, cho trang này cơ hội thứ hai đặt bit tham khảo về 0, chọn trang FIFO kế tiếp. Trang được cho cơ hội thứ hai đặt vào cuối hàng đợi.
- Một trang đã được cho cơ hội thứ hai sẽ không bị thay thế trước khi các trang còn lại bị thay thế
- Có thể cài đặt bằng một xâu vòng (danh sách liên kết vòng).

Thuật toán cơ hội thứ hai nâng cao

- Ý tưởng:
  - Xét cặp bit: reference bit và dirty bit
  - (0,0): không truy xuất, không sửa đổi → trang tốt nhất để thay thế.
  - (0,1): không truy xuất, có sửa đổi → cần lưu lại trang thay thế.
  - (1,0): có truy xuất, chưa sửa đổi → có khả năng sẽ được sử dụng tiếp.
  - (1,1): có truy xuất, có sửa đổi → có khả năng được sử dụng tiếp và nếu thay thế cần lưu lại.
  - Lớp đầu tiên có độ ưu tiên thấp nhất và lớp cuối cùng có độ ưu tiên cao nhất.

Cấp phát khung trang

- Trả lời câu hỏi:
  - Mỗi tiến trình sẽ được cấp phát bao nhiêu khung trang?
- Các hướng tiếp cận:
  - Cấp phát cố định:
    - Cấp phát công bằng
    - Cấp phát theo tỉ lệ
  - Cấp phát theo độ ưu tiên

Cấp phát cố định

- Mỗi tiến trình sẽ được cấp phát một số lượng khung trang cố định ngay từ đầu cho đến khi kết thúc thi hành.
- Có 2 hướng
  - Cấp phát công bằng
    - M khung trang, n tiến trình → mỗi tiến trình m/n
  - Cấp phát theo tỉ lệ
    - S_i: kích thước bộ nhớ ảo của tiến trình I
    - S = sum(S_i)
    - M khung trang
    - Tiến trình I sẽ có: (S_i/S)*M khung trang

Cấp phát theo độ ưu tiên

- Số khung trang dành cho mỗi tiến trình phụ thuộc vào độ ưu tiên của tiến trình tại thời điểm xác định.
- Nếu tiến trình p_i phát sinh lỗi trang, chọn một trong các khung trang của nó để thay thế hoặc một khung trang của các tiến trình có độ ưu tiên thấp hơn.

Thay thế toàn cục và Thay thế cục bộ

- Thay thế toàn cục
  - Trang "nạn nhân" có thể là bất cứ khung trang nào của hệ thống, không nhất thiết phải là khung trang của tiến trình đó.
- Thay thế cục bộ
  - Trang nạn nhân là một trong số khung trang của tiến trình đó.
- Có vẻ thay thế toàn cục sẽ linh hoạt hơn nhưng có thể gây ra hiệu ứng trì trệ hệ thống (thrashing)
- Thrashing: tự đọc tài liệu

Hệ thống quản lý tập tin

Tổng quan

- Khái niệm cơ bản
- Mô hình quản lý tập tin
- Cài đặt hệ thống quản lý tập tin
- Truy xuất hệ thống quản lý tập tin

Các khái niệm cơ bản

- Bộ nhớ ngoài (secondary storage)
  - Có khả năng lưu trữ dữ liệu trong thời gian dài
  - Công dụng:
    - Dữ liệu có kích thước rất lớn → bộ nhớ trong không đủ để chứa
    - Phải được lưu lại trong thời gian dài, ngay cả khi chương trình không chạy
- Tập tin
  - Là đơn vị lưu trữ thông tin cơ bản của bộ nhớ ngoài
  - Là sự trừu tượng hóa của bộ nhớ ngoài
  - Được quản lý bởi hệ điều hành
  - Có một tên
  - Tập hợp các sector

Các khái niệm cơ bản

- Thư mục
  - Là một tập tin đặc biệt, có thể lưu trữ nhiều tập tin khác.
- Hệ thống quản lý tập tin
  - Là một phần của hệ điều hành cung cấp các chức năng quản lý tập tin (thư mục) như:
    - Cách hiển thị tập tin
    - Cách đặt tên tập tin
    - Tổ chức thông tin của tập tin
    - Thao tác trên tập tin
    - Sử dụng và bảo vệ tập tin

Mô hình quản lý tập tin

- Cách đặt tên:
  - Phụ thuộc vào hệ điều hành
  - Thường là các chữ cái và số
  - Phân biệt chữ hoa, chữ thường???
  - Chia làm 2 phần được phân cách bởi dấu ".":
    - Phần tên: tên của tập tin
    - Phần mở rộng: để xác định loại tập tin

Cấu trúc của tập tin

- Có thể chia làm 3 loại
  - Dãy tuần tự các byte không cấu trúc
  - Dãy các record có kích thước cố định
  - Cấu trúc cây: gồm nhiều record, không nhất thiết phải có kích thước bằng nhau. Mỗi record có thể có khóa để tìm kiếm nhanh hơn.

Phân loại các kiểu tập tin

- Tập tin bình thường:
  - Tập tin văn bản: có phân biệt ký tự xuống dòng → thích hợp với các chương trình soạn thảo
  - Tập tin nhị phân: có qui định cấu trúc riêng.
- Thư mục: là tập tin dùng để lưu trữ các tập tin khác
- Tập tin dành riêng cho hệ điều hành: thường được gắn với một thiết bị phần cứng.

Truy xuất tập tin

- Truy xuất tuần tự
  - Để đến được byte (record) thứ i phải đi qua i byte (record) trước đó.
- Truy xuất ngẫu nhiên

Các thuộc tính của tập tin

- Mô tả các thông tin của tập tin
- Ví dụ:
  - Bảo vệ: ai được quyền truy xuất.
  - Ngày giờ tạo tập tin
  - …

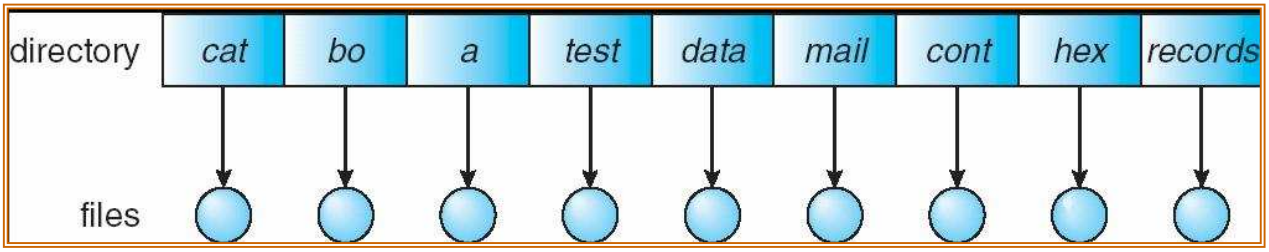
Hệ thống thư mục

- Thư mục chứa nhiều entry, mỗi entry tương ứng với một tập tin.
- Entry chứa:
  - Tên tập tin, thuộc tính, địa chỉ trên bộ nhớ ngoài lưu tập tin.
  - Hoặc tên tập tin và một con trỏ trỏ đến cấu trúc chứa thuộc tính, địa chỉ trên bộ nhớ ngoài.
- Một tập tin luôn nằm trong một thư mục → khi có yêu cầu mở một tập tin, HĐH tìm trên thư mục đã chỉ ra cho đến kho tìm được tập tin cần tìm, đưa vào bộ nhớ chính. Thao tác sau đó hầu hết sẽ làm việc trên bộ nhớ chính.

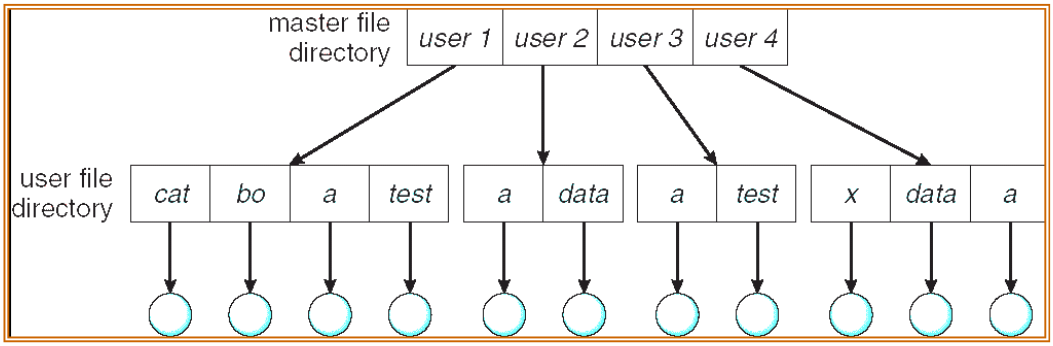
Hệ thống thư mục

- Theo dạng thư mục đơn:
  - Chỉ có một thư mục và mọi tập tin đều đặt trong thư mục đó → khó đặt tên tập tin vì có thể trùng nhau.
- Theo dạng hai cấp:
  - Có một thư mục gốc, trong thư mục gốc có nhiều thư mục con và các tập tin sẽ được lưu vào trong các thư mục con.
- Theo dạng đa cấp
  - Có một thư mục gốc, một thư mục có thể chứa các thư mục con và các tập tin.

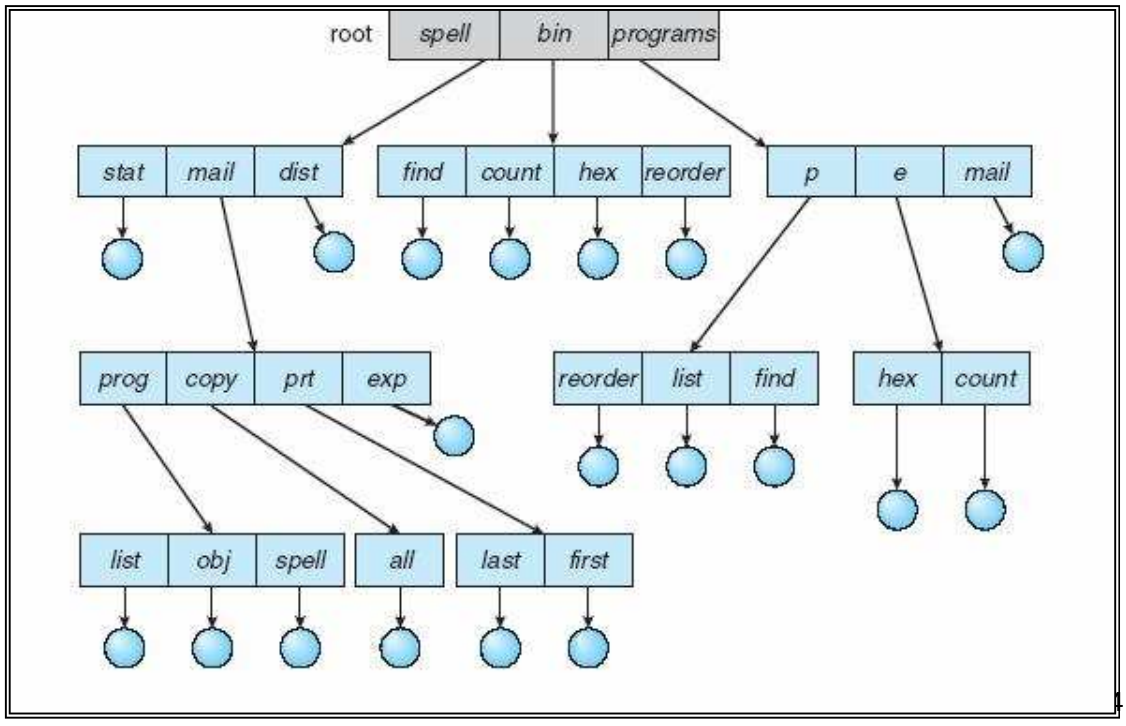
Hệ thống thư mục 1 cấp



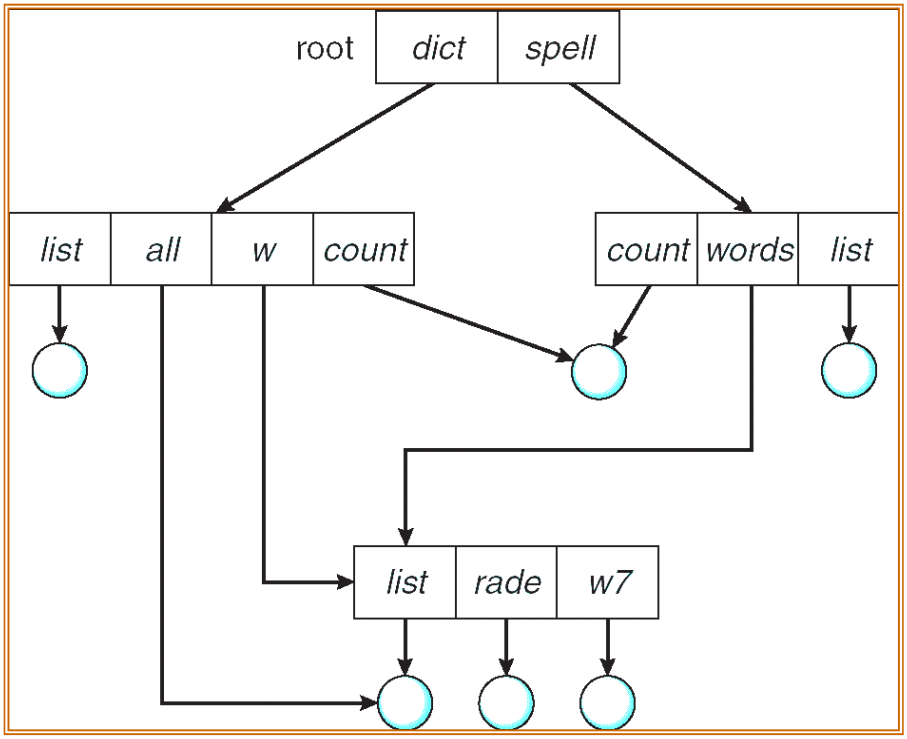
Hệ thống thư mục 2 cấp



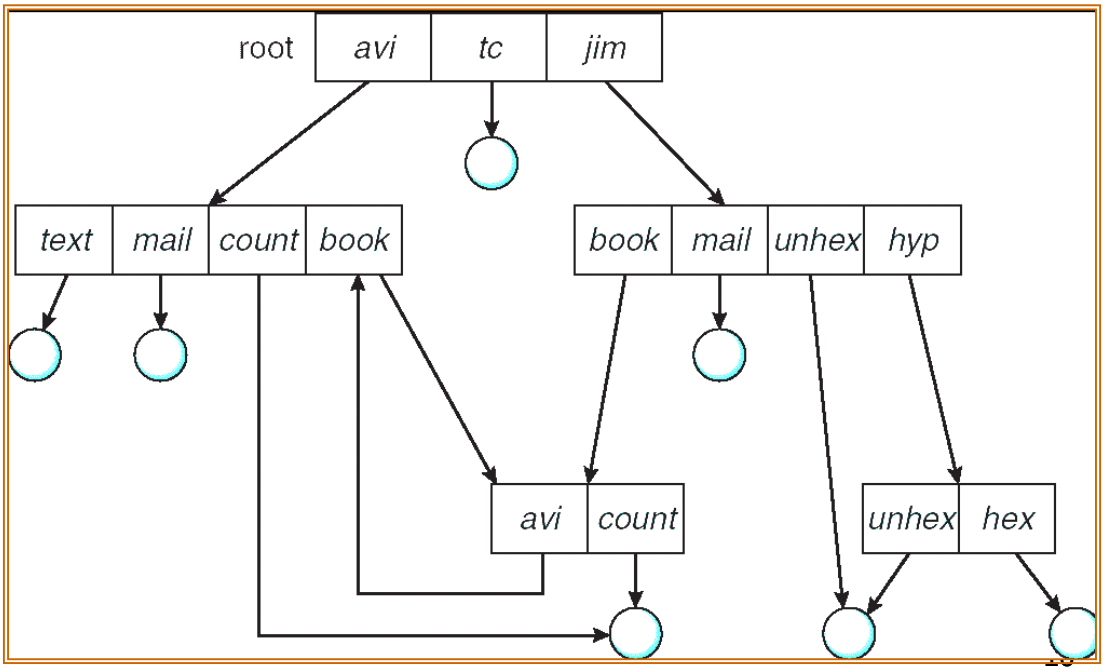
Hệ thống thư mục đa cấp (cấu trúc cây)



Hệ thống thư mục đa cấp (không có chu trình)



Hệ thống thư mục tổng quát



Đường dẫn

- Đường dẫn: cho phép xác định vị trí của tập tin trong hệ thống thư mục.
- Trong hệ thống thư mục theo kiểu đa cấp (cây thư mục) có hai cách để xác định một tập tin.
- Đường dẫn tuyệt đối: mỗi tập tin được gán một đường dẫn từ thư mục gốc đến tập tin.
- Đường dẫn tương đối: người dùng có thể qui định một thư mục hiện hành, mọi đường dẫn không cần chỉ ra từ thư mục gốc mà chỉ cần chỉ ra từ thư mục hiện hành
- Lưu ý: “.” thư mục hiện hành, “..” thư mục cha

Các chức năng

- Tập tin:
  - Tạo
  - Xóa
  - Mở
  - Đóng
  - Đọc
  - Ghi
  - Thêm
  - Tìm
  - Lấy thuộc tính và thay đổi thuộc tính
  - Đổi tên

Các chức năng

- Thư mục:
  - Tạo
  - Xóa
  - Mở (thư mục)
  - Đóng
  - Đọc thư mục
  - Đổi tên
  - Liên kết
  - Bỏ liên kết

Hệ Thống Quản Lý Tập Tin

- Đóng vai trò trung gian nhờ vào đó người dùng chỉ cần cung cấp tên có thể xác định được tập tin nằm ở những sector nào.
- Hai đối tượng chủ yếu trên HTQLTT là tập tin và thư mục.

Hệ Thống Quản Lý Tập Tin

- Multi System là cơ chế cho phép trên 1 đĩa cứng có nhiều hệ điều hành.
- Mỗi vùng trên đĩa cứng thuộc về một hệ điều hành gọi là 1 partition.
- Một hệ điều hành có thể có nhiều partition khi đó nó sẽ có 1 partition chính: primary partition, còn lại là extended partition.

Hệ Thống Quản Lý Tập Tin

- Khi đĩa cứng có nhiều hệ điều hành phải tồn tại 1 hệ điều hành được khởi động đầu tiên thì partition chính của hệ điều hành đó được gọi là active.
- Để quản lý các partition, ta sử dụng sector đầu tiên trên đĩa gọi là Master Boot Record. Mỗi partition sẽ chiếm 16 bytes thông tin trên Master Boot Record

Hệ Thống Quản Lý Tập Tin

- Mỗi partition sẽ chiếm 16 bytes thông tin trên Master Boot Record

Offset	Độ lớn	Ý Nghĩa	Ví dụ
0	1	Active partition	80h hoặc 0
1	1	Head bắt đầu	
2	1	Track bắt đầu	
3	1	Sector bắt đầu	
4	1	Byte mô tả loại hệ điều hành	F9 – DOS, FA - Win
5	1	Head kết thúc	
6	1	Track kết thúc	
7	1	Sector kết thúc	
8	4	Khoảng cách từ Master Boot đến partition	
12	4	Dung lượng partition	

Hệ Thống Quản Lý Tập Tin

- Mutil Volume: là cơ chế cho phép trên 1 partition có thể chia nhỏ ra nhiều phần. Mỗi phần được gọi là 1 volume.
- Mỗi volume sẽ tương ứng với 1 ổ đĩa logic.
- Một volume sẽ có cấu trúc như sau:

Boot Sector	Quản lý	Data

- Với DOS, Win, 1 partition thường tương ứng với 1 volume, 1 ổ đĩa logic
- Đĩa mềm được xem là 1 volume.

Cài đặt hệ thống quản lý tập tin

- Mỗi tập tin được lưu trên nhiều block (khối) khác nhau của 1 volume.Mỗi block có kích thước 2^k sector và đánh số từ 0.
- Đĩa mềm 1 block = 1 sector. Một block có các trạng thái sau:
  - Free: chưa thuộc về tập tin nào
  - Used: thuộc vào duy nhất một tập tin
- Vấn đề là hệ thống cần phải biết tất cả các khối của một tập tin nào đó → bảng phân phối vùng nhớ
- Các phương pháp chính
  - Định vị liên tiếp (mô hình tuần tự)
  - Định vị bằng chỉ số (mô hình Index)
  - Định vị bằng danh sách liên kết (Mô hình liên kết)
  - Định vị bằng danh sách liên kết sử dụng chỉ số (Mô hình liên kết chỉ số)
  - I-node

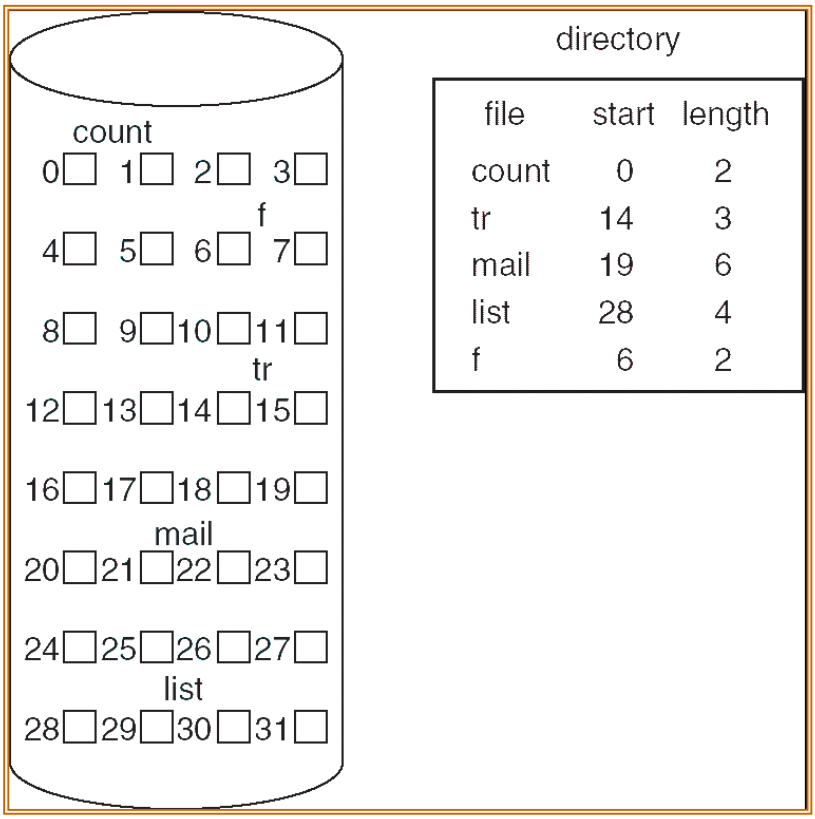
Định vị liên tiếp

- Các khối của tập tin được cấp phát liên tiếp nhau → chỉ cần lưu khối đầu tiên và kích thước của tập tin.
- Cấu trúc một ô của vùng quản lý:

```
Struct oquanly {
    char filename[11];
    int blockbatdau;
    long size;
}
```

- Ưu điểm:
  - Dễ cài đặt
  - Đọc nhanh
- Khuyết điểm:
  - Không thể lưu trữ khi bị phân mảnh
  - Phải biết kích thước tối đa của tập tin

Định vị liên tiếp



Định vị chỉ số

- Cấu trúc một ô của vùng quản lý:

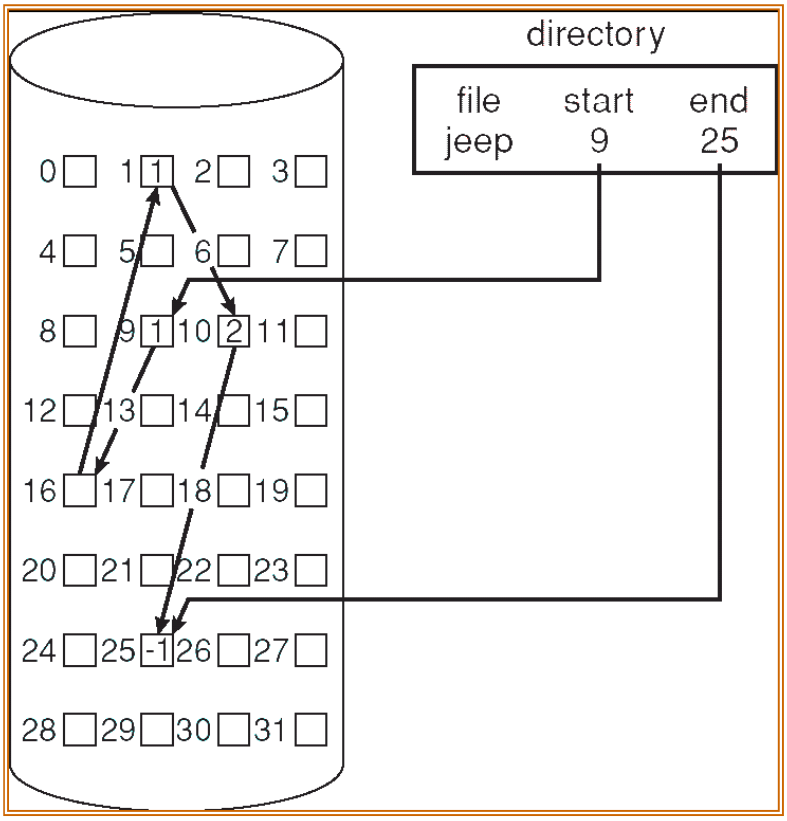
```
Struct oquanly {
    char filename[11];
    int blockIndex[max];//mảng chỉ số các block
    long size;
}
```

- Ưu điểm:
  - Có thể lưu trữ khi phân mảnh
  - Đọc nhanh
- Khuyết điểm:
  - Giới hạn kích thước tập tin

Định vị bằng danh sách liên kết

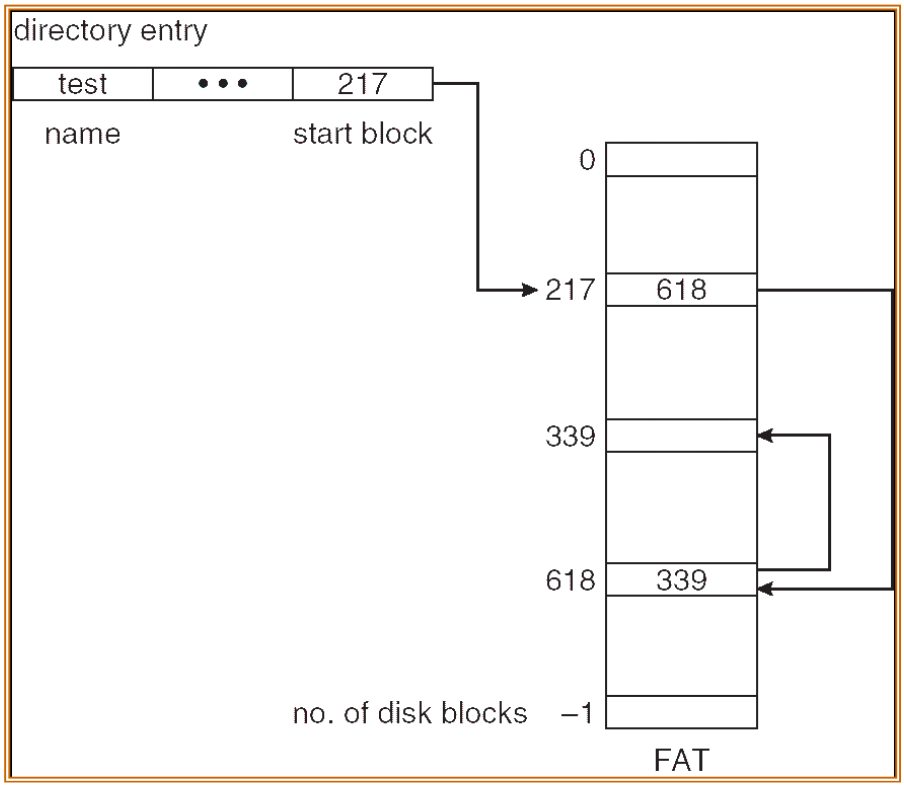
- Các khối của một tập tin sẽ trỏ đến khối kế tiếp. Khối cuối cùng trỏ đến NULL → chỉ cần lưu khối đầu tiên.
- Ưu điểm:
  - Không bị phân mảnh
- Khuyết điểm
  - Truy xuất tuần tự

Định vị bằng danh sách liên kết



Định vị bằng DSLK sử dụng chỉ số

- Thay vì dùng con trỏ, sử dụng chỉ số để liên kết các khối.
- Ưu điểm:
  - Có thể truy xuất ngẫu nhiên
- Khuyết điểm:
  - Giới hạn kích thước bảng chứa các chỉ số liên kết

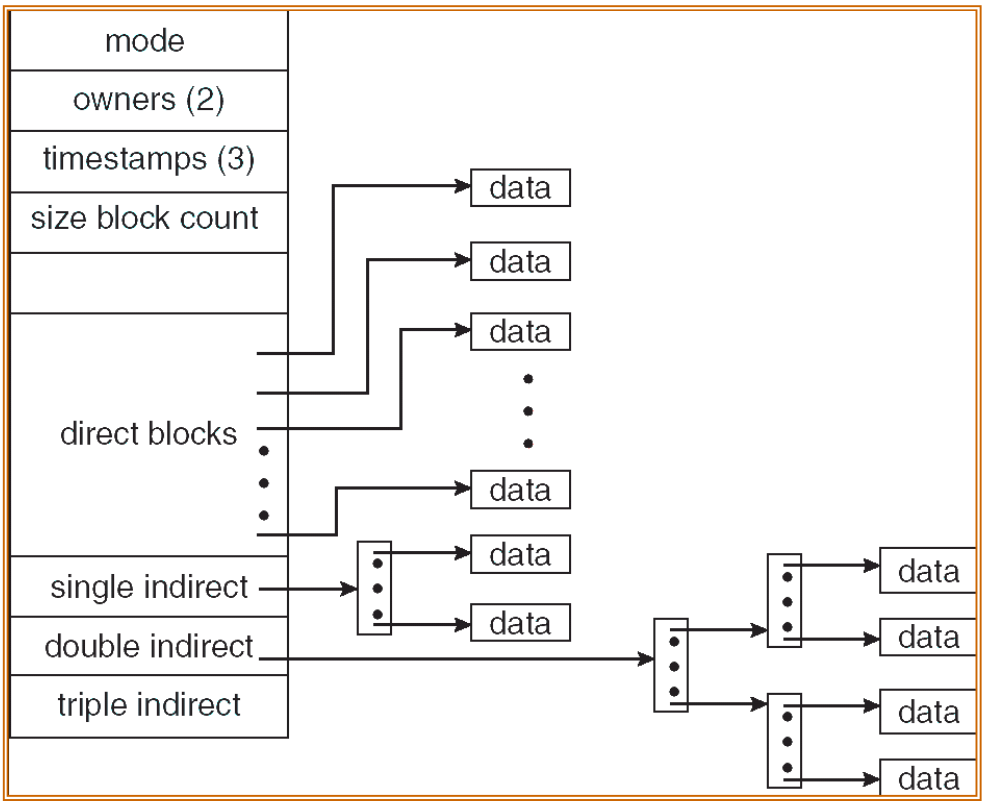


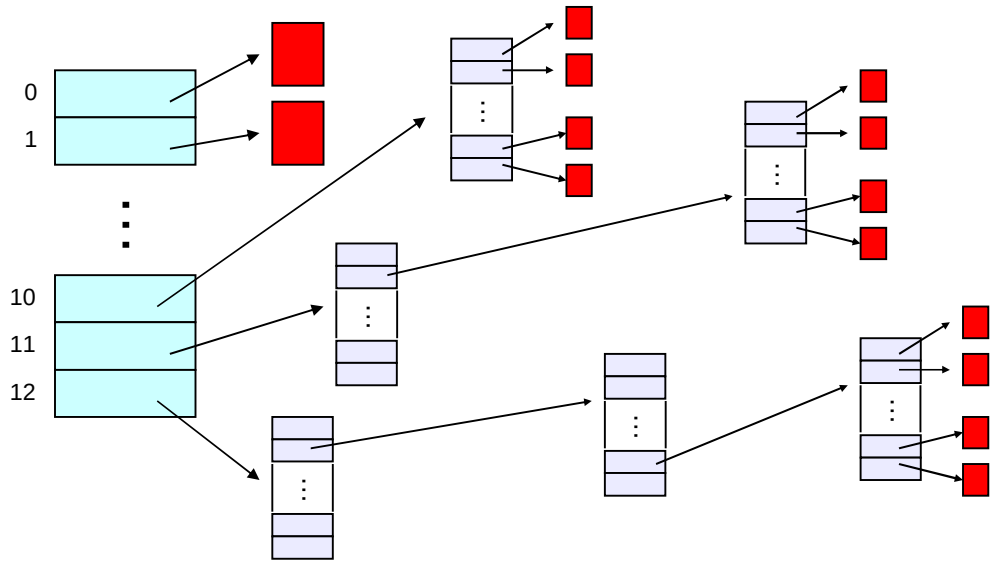
Câu Hỏi

- Master Boot Record là gì? Một ổ cứng có thể chia tối đa bao nhiêu partion? Một Partion có phải luôn tương ứng với một ổ đĩa Logic (ví dụ C D) hay không?
- Theo phương pháp định vị bằng chỉ số, nếu một ô quản lý có kích thước 59 bytes thì giới hạn kích thước một tập tin là bao nhiêu trong trường hợp 1 block = 1KB.
- Bạn đang ở thư mục “C:\Documents and Settings\All Users” hãy viết đường dẫn tương đối đến “C:\Windows”?

Định vị bằng I-node

- I-node gồm 2 phần: phần lưu thuộc tính và phần địa chỉ.
- Phần địa chỉ có thể chia làm 2 phần
  - Phần đầu: 10 phần tử chứa địa chỉ trực tiếp của 10 khối đầu tiên
  - Phần tử thứ 11 chứa địa chỉ của một khối mà khối đó chứa 2^{10} phần tử, mỗi phần tử chứa địa chỉ của một khối thật sự của tập tin
  - Phần tử thứ 12 chứa địa chỉ của một khối mà mỗi phần tử trong khối như là phần tử thứ 11.
  - Phần tử thứ 13 chứa địa chỉ của một khối mà mỗi phần tử trong khối như là phần tử thứ 12.


mode
owners (2)
timestamps (3)
size block count
direct blocks
single indirect
double indirect
triple indirect
data
data
data
data
data
data
data
data
data


0
1
10
11
12

Mô hình tổ chức Volume của UNIX

- Theo dạng inode

Boot Sector	Khối khởi động	Danh sách inode	Data

Khối khởi động chứa chương trình khởi động.

Danh sách các inode được phân thành các ô. Mỗi ô gọi là một inode (64 bytes)

Mỗi inode quản lý cho một tập tin hoặc một thư mục con.

Mô hình tổ chức Volume của UNIX

25 bytes thông tin về tập tin và thư mục con (tên, thuộc tính, quyền sở hữu....)			
1(3bytes)	2	3	4
5	6	7	8
9	10	11	12
13			

39 bytes phân thành 13 ô. Mỗi ô chứ chỉ số của 1 block.

10 ô đầu tiên chứa chỉ số block dữ liệu của tập tin.

Ô 11 chứa chỉ số block mà block đó được phân thành nhiều ô (3 bytes) chứa chỉ số block dữ liệu.

Ô 12 chứa chỉ số block mà block này được chia thành các ô. Mỗi ô chứa chỉ số của 1 block có vai trò như của ô 11.

Ô 12 chứa chỉ số block mà block này được chia thành các ô. Mỗi ô chứa chỉ số của 1 block có vai trò như của ô 12.

Quản lý đĩa

- Tập tin được lưu trên đĩa → lưu như thế nào?
  - Lưu trên n byte liên tiếp → tập tin lớn thì quản lý khó
  - Lưu trên các khối → thường dùng
- Kích thước khối hầu như được đặt là 512byte (1 sector chuẩn)
  - Có thể là 1K, 2K

Lưu danh sách khối trống

- Công dụng:
  - Nhanh chóng xác định khối để cấp phát khi có nhu cầu lưu tập tin hoặc tạo mới tập tin.
- Các phương pháp:
  - Bảng bit:
    - N khối → n bit.
    - Bit = 0 → khối còn trống
    - Bit = 1 → khối đã sử dụng
    - Kích thước bảng bit: disk size / (8 * block size)
    - Ví dụ: đĩa 16Gb, block 512 byte → bảng bit chiếm 4MB

- Các phương pháp lưu khối trống
  - Danh sách các vùng trống
    - Mỗi phần tử bao gồm vị trí bắt đầu và kích thước của vùng trống
  - Danh sách khối trống
    - Mỗi khối được gán một con số
    - Danh sách các khối trống được lưu trong một vùng dành riêng

Giới thiệu về hệ thống quản lý quyền sở hữu file của UNIX

- Tiến trình và file đều được sở hữu bởi một người dùng nào đó.
- Một tập tin sẽ có user owner và group owner
  - Chủ sở hữu có quyền kiểm soát chính và có thể bị tước đoạt bởi root (user cao nhất).
  - Mỗi file có một chủ sở hữu chính và thuộc vào một hay nhiều nhóm chủ sở hữu.
  - Chủ sở hữu có thể xác định quyền truy cập cho mọi thành viên còn lại (trừ root) bằng chmod.
  - Người sở hữu không nhất thiết phải thuộc về một nhóm sở hữu tập tin.

Hệ thống FAT 12, 16, 32

BS	FAT1	FAT2	DIRECTORY	Data
	Vùng quản lý			Vùng dữ liệu

- Mô hình này được áp dụng cho hệ điều hành DOS và Windows.
- FAT1 và FAT2 là hai bảng giống nhau. FAT2 là bảng dự phòng của FAT1.
- FAT (File Allocation Table) được phân thành các ô, mỗi ô quản lý trong 1 block.
  - Ô thứ i quản lý Block thứ i-2
  - Kích thước mỗi ô có thể là 12, 16 hay 32 Bit.

Hệ thống FAT 12, 16, 32

BS	FAT1	FAT2	DIRECTORY	Block 0 1 2 3 …
	Vùng quản lý			Vùng dữ liệu

- FAT (File Allocation Table) được phân thành các ô, mỗi ô quản lý trong 1 block.
  - Ô thứ I quản lý Block thứ i-2
  - Kích thước mỗi ô có thể là 12, 16 hay 32 Bit.
  - Giá trị của các ô có thể là:
    - 0 là free
    - FF7 là bad
    - FF8 – FFF là cuối file
    - Khác là ô tiếp theo

Hệ thống FAT 12, 16, 32

BS	FAT1	FAT2	DIRECTORY	Block 0 1 2 3 …
	Vùng quản lý			Vùng dữ liệu

- Root Directory được phân thành các ô, mỗi ô quản lý cho 1 tập tin hay thư mục.
  - Mỗi ô của Root Directory có kich thước 32 bytes
  - Cấu trúc 32 bytes này như sau:

Hệ thống FAT 12, 16, 32

BS	FAT1	FAT2	DIRECTORY	Block 0 1 2 3 …
	Vùng quản lý			Vùng dữ liệu

Cấu trúc một ô của Root Directory

Offset	Độ lớn	Ý nghĩa
0	8	Filename
8	3	Extension
11	1	Thư mục con (16) hay tập tin (>=32)
12	10	Chưa dùng đến
22	2	Ngày tạo
24	2	Giờ tạo
26	2	chỉ số ô đàu tiên trên bảng FAT
28	4	Kích thước file

Thông tin về mô hình FAT trên đĩa mềm 1,4 MB

- Có 2880 sector
- Vùng quản lý gồm 32 sector:
  - FAT1: 9 sector
  - FAT2: 9 sector
  - Root Directory: 14 sector. Chia thành (14*512)/32=224 ô. → có tối đa 224 tập tin trên thư mục gốc.
- Vùng dữ liệu 2847 sector.
- Thư mục con cũng chiếm một số block trên volume. Nội dung của các block này sẽ có cấu trúc như Root Directory. Nghĩa là cũng phân thành các ô, mỗi ô quản lý cho 1 tập tin.

Ví dụ

- Trên thư mục gốc có tập tin emoi.c (5,2,7): dự liệu của tập tin emoi trên các block 5, 2, 7 hay nói cách khác chỉ số cá ô quản lý là ô (7,4,9), tập tin m.txt (9,4,3) và thư mục L (6). Hãy vẽ bảng FAT, Root Dir?
- Nhắc lại: ô thứ I của FAT quản lý block thứ i-2.
- Root Dir được chia thành các ô (32 bytes), ta tập trung vào thể hiện 4 thông tin tên file, phần mở rộng, giá trị byte 11 biểu diễn tập tin hay thư mục và giá trị bytes26-27 biểu chỉ số ô đầu tiên trong bảng FAT.

Ví dụ

Ô thứ 0

0 (free)	0 (free)
9	FFF
5	4
FFF	FFF
0 (free)	6

emoi	c	32	7
m	txt	32	11
L		16	8

32 là thư mục; 16 là tập tin

7, 11, 8 là chỉ số ô đầu tiên trong bảng FAT của tập tin emoi.c, m.txt và thư mục L.

Ví dụ

Block

- Giả sử thư mục L (6) trong ví dụ chứa tập tin A(1) và B.doc(8) thì nội dung của block 6 như sau:

.		16	8
..		16	0
A		32	3
B	doc	32	10

Chỉ số ô FAT đầu tiên của thư mục hiện hành

Chỉ số ô FAT đầu tiên của thư mục cha nếu cha là thư mục gốc thì bằng 0

Ví dụ

- Giả sử thư mục L (6) trong ví dụ chứa tập tin A(1) và B.doc(8) thì nội dung của block 6 và bảng FAT như sau:

.		16	8
..		16	0
A		32	3
B	doc	32	10

0 (free)	FFF
9	FFF
5	4
FFF	FFF
FFF	6

Bài tập FAT và RD

- Dung lượng đĩa sẽ quyết định kích thước 1 ô trên bảng FAT.
- FAT12 → 2^{12} ô = 4096 ô → tối đa là 4094 block. (vì mỗi ô quản lý 1 block) Tính toán khoảng gần đúng.
- Ví dụ: cho đĩa có dung lượng 20MB. Mỗi block là 2 sector. Hỏi đĩa sử dụng bảng FAT nào là hợp lý?
  - Số ô = số block + 2 = $(20*1024^2)/(2*512)$
    $= 5*2^{12}+2 < 2^{15}$ --> Nên dùng FAT 16.
- Nếu dùng FAT 32 thì sao?
  - Mỗi block có kích thước nhỏ đi → 1 block có thể chỉ là 1 sector.

Bài tập FAT và RD

- Cho một volume có 20 block như sau
- Thư mục gốc \:
  - E (0)
    - M.TXT (1,19,18)
    - C (2)
      - U.TXT (4,5)
      - V (6)
        - » N.TXT (7,8)
        - » I.TXT (9,10)
  - M.TXT (3,17,16)
- Hãy vẽ FAT, RD và các block liên quan (block nội dung của thư mục có cấu trúc tượng tự RD thêm vào . và ..) .
- Quy ước có phần mở rộng là tập tin, có chứa con thì sẽ phải là thư mục.

Cho một đĩa có dung lượng 40GB. Mỗi block gồm 16 sector. Hỏi ta sử dụng bảng FAT nào là thích hợp nhất?

Câu Hỏi Ôn tập

1. So sánh thời gian các lệnh copy, move, delete trong tất cả các trường hợp có thể có.
2. Tại sao trong UNIX không có system call detete(…) để xoá file mà chỉ có system call unlink(…) để xoá một link đến file?
3. Trình bày các thao tác của DOS/Win khi tiến hành khi delete một file?

Hệ thống quản lý nhập/xuất

Tổng quan

- Giới thiệu về thiết bị nhập xuất
- Các kỹ thuật quản lý thao tác nhập xuất
- Các vấn đề về thiết kế hệ thống quản lý nhập xuất trong HĐH
- Kỹ thuật vùng đệm nhập xuất
- Quản lý hệ thống nhập xuất đĩa

Phân loại các thiết bị nhập/xuất

- Human readable
  - Dùng để giao tiếp với người dùng
  - Máy in
  - Thiết bị đầu cuối liên quan đến hiển thị
    - Màn hình hiển thị
    - Bàn phím
    - Chuột

Phân loại các thiết bị nhập/xuất (2)

- Machine readable
  - Dùng để giao tiếp với thiết bị điện tử
  - Đĩa

Phân loại các thiết bị nhập/xuất (2)

- Giao tiếp
  - Dùng để giao tiếp với các thiết bị ở xa (remote device)
  - Modem

Sự khác nhau giữa các thiết bị I/O

- Tốc độ
- Ứng dụng
- Mức độ phức tạp để kiểm soát
- Đơn vị truyền
- Biểu diễn dữ liệu
- Phát sinh lỗi

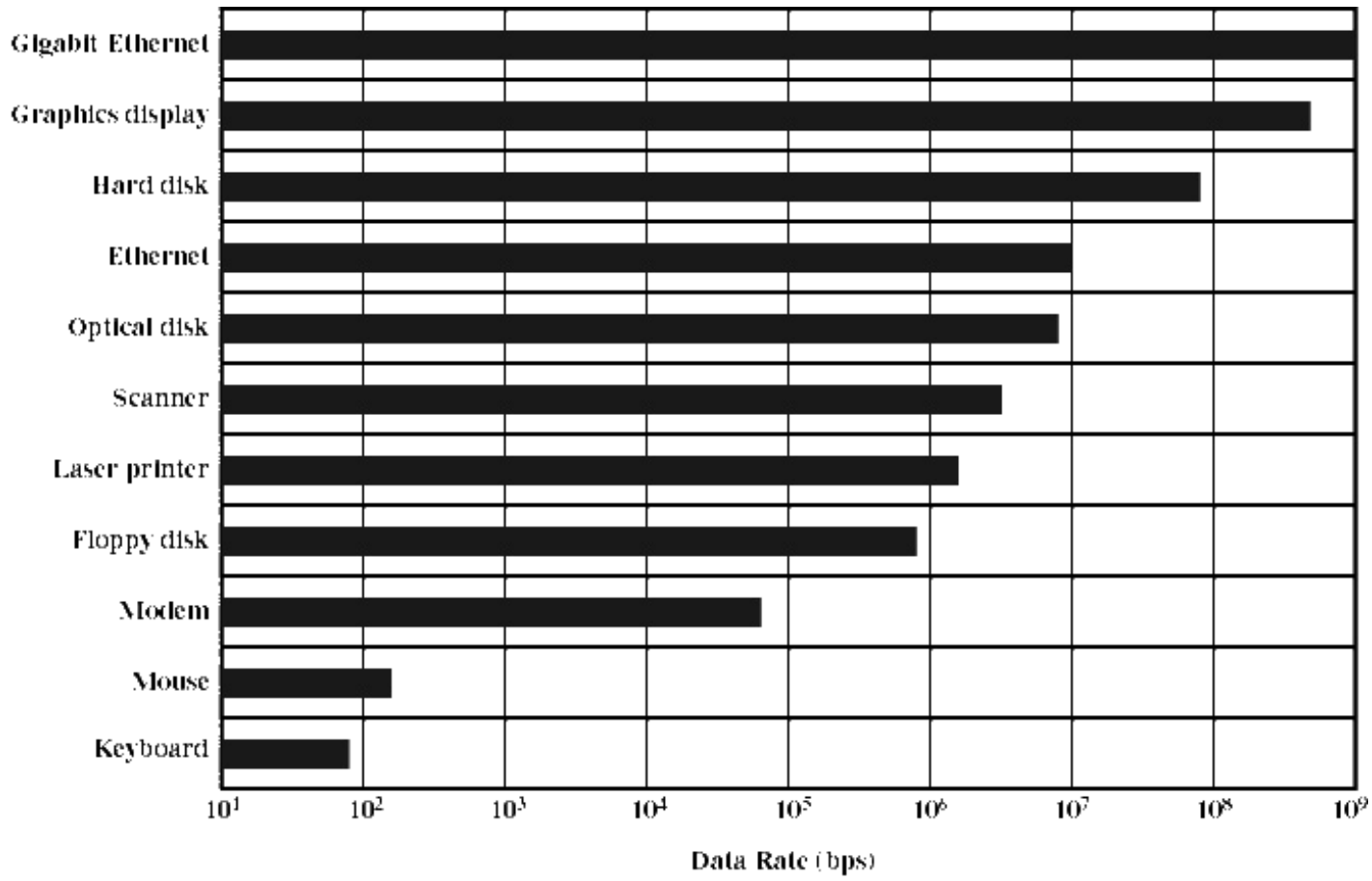


Figure 11.1 Typical I/O Device Data Rates

Sự khác nhau giữa các thiết bị I/O (2)

- Ứng dụng
  - Đĩa dùng để lưu trữ đòi hỏi phải có hệ thống quản lý tập tin đi kèm
  - Đĩa dùng để làm bộ nhớ ảo đòi hỏi phải có sự hỗ trợ phần cứng và phần mềm.
  - Các thiết bị đầu cuối được sử dụng bởi quản trị sẽ có độ ưu tiên cao hơn.

Sự khác nhau giữa các thiết bị I/O (3)

- Độ phức tạp để kiểm soát:
  - Máy in so với đĩa
- Đơn vị truyền
  - Có thể được truyền theo byte hoặc khối (có thể là bit)
- Biểu diễn dữ liệu
  - Cơ chế mã hóa
- Phát sinh lỗi
  - Thiết bị khác nhau xử lý lỗi phát sinh khác nhau

Kỹ thuật xử lý I/O

- Programmed I/O
  - Tiến trình phải busy-waiting cho thao tác nhập xuất hoàn thành
- Interrupt-driven I/O
  - Phát sinh lệnh I/O
  - Bộ xử lý tiếp tục thực thi các chỉ thị khác
  - Module I/O gởi một ngắt đến bộ xử lý khi hoàn thành thao tác I/O

Kỹ thuật quản lý I/O (2)

- Direct Memory Access (DMA)
  - Module DMA điều khiển việc chuyển đổi dữ liệu giữa bộ nhớ chính và thiết bị I/O
  - Bộ xử lý chỉ bị ngắt khi toàn bộ khối đã được chuyển

Lịch sử phát triển của quản lý I/O

- Bộ xử lý trực tiếp điều khiển thiết bị ngoại vi
- Bộ điều khiển (Controller) hay module I/O module được thêm vào
  - Bộ xử lý dùng programmed I/O không có hỗ trợ ngắt
  - Bộ xử lý không cần phải quản lý chi tiết các thiết bị ngoại vi

Lịch sử phát triển của quản lý I/O

- Bộ điều khiển hoặc module I/O module có hỗ trợ ngắt
  - Bộ xử lý không cần phải bỏ thời gian chờ thao tác I/O thi hành
- Direct Memory Access
  - Khối dữ liệu được chuyển vào bộ nhớ mà không cần xử lý của bộ xử lý
  - Bộ xử lý chỉ liên quan vào lúc bắt đầu và lúc kết thúc chuyển khối.

Lịch sử phát triển của quản lý I/O

- Module I/O là một bộ xử lý riêng biệt
  - Có tập lệnh chuyên biệt
- Bộ xử lý I/O
  - Module I/O có bộ nhớ riêng

Các vấn đề thiết kế liên quan đến HĐH

- Tính hiệu quả
  - Hầu hết các thiết bị I/O đều rất chậm so với bộ nhớ chính
    - Sử dụng cơ chế đa chương cho phép một vài tiến trình chờ I/O trong khi tiến trình khác thi hành
  - Cơ chế swapping
    - Bản chất cũng là thao tác nhập/xuất

Các vấn đề thiết kế liên quan đến HĐH

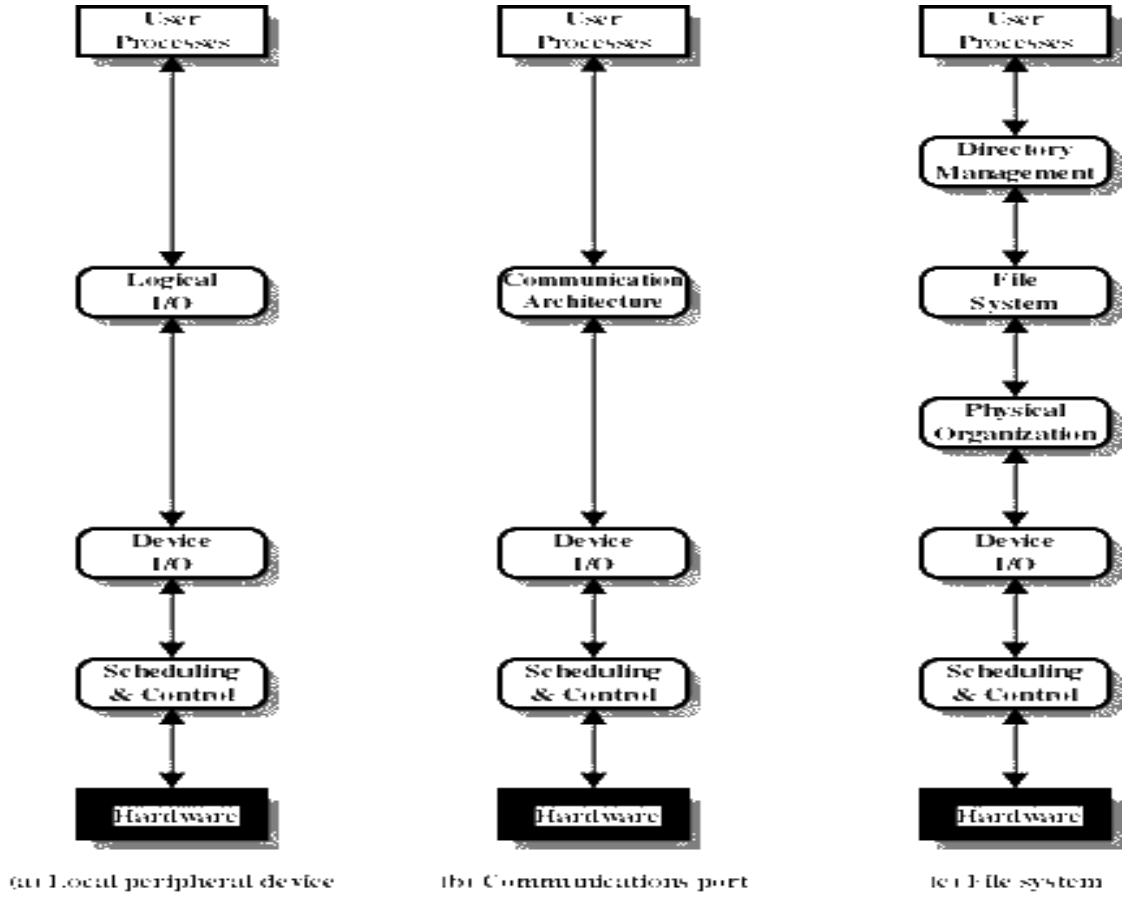
- Tính tổng quát
  - Mong muốn xử lý tất cả các thiết bị I/O giống nhau
  - Che giấu hầu hết chi tiết của thiết bị nhận xuất để tiến trình (mức cao) xem các thiết bị ở một vài chức năng thông dụng như read, write, open, close, lock, unlock

Mô hình lôgic các chức năng nhập xuất

- Giao tiếp đơn giản (thiết bị ngoại vi cục bộ)
  - Logical I/O
  - Device I/O
  - Scheduling and Control
- Giao tiếp thông quan cổng kết nối
  - Communication architecture
  - Device I/O
  - Scheduling and Control

Mô hình lôgic các chức năng nhập xuất

- Hệ thống tập tin
  - Directory management
  - File system
  - Physical organization
  - Device I/O
  - Scheduling and Control



(a) Local peripheral device (b) Communications port (c) File system

Figure 11.5 A Model of I/O Organization

Kỹ thuật vùng đệm nhật/xuất

- Lý do
  - Các tiến trình phải chờ hoàn thành I/O trước khi xử lý tiếp
  - Thao tác I/O vẫn phải tốn bộ nhớ

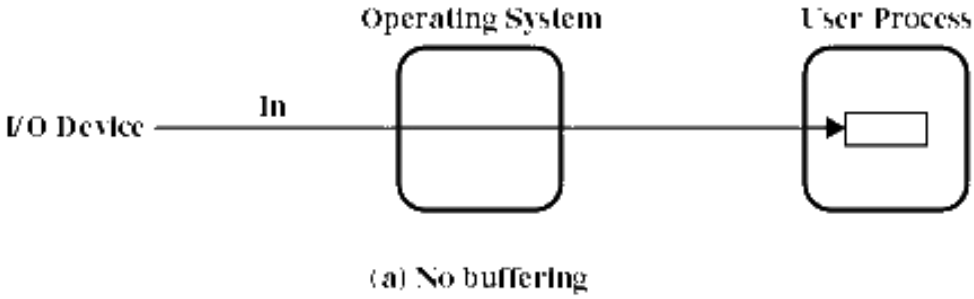
Kỹ thuật vùng đệm nhật/xuất (2)

- Các thiết bị I/O có hai dạng hỗ trợ nhập/xuất cơ bản:
  - Hướng khối (block – oriented)
  - Hướng dòng (stream – oriented)
- Block-oriented
  - Thông tin được lưu trong những khối có kích thước cho trước
  - Đơn vị chuyển là một khối / một lần
  - Dùng cho đĩa
- Stream-oriented
  - Truyền thông tin như là một dòng các byte
  - Dùng cho các thiết bị đầu cuối, máy in, chuột, cổng giao tiếp,…

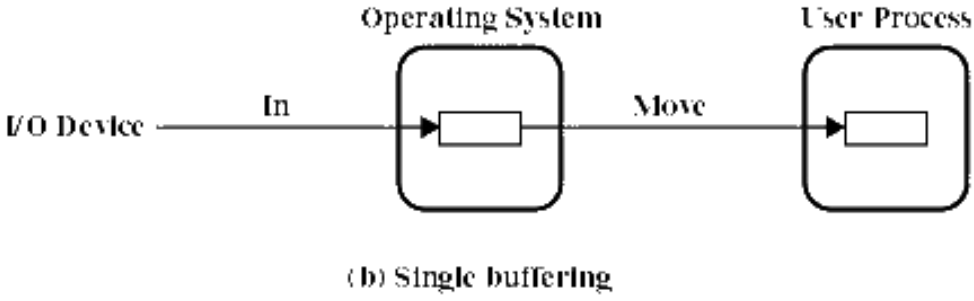
Single Buffer

- HĐH dành riêng một vùng đệm trong bộ nhớ chính cho thao tác I/O
- Block-oriented
  - Dữ liệu vào được chuyển vào vùng đệm theo đơn vị khối
  - Khối đó sẽ chuyển đến cho tiến trình người dùng khi cần
  - Khối khác được chuyển vào vùng đệm, trong khi tiến trình đang xử lý khối trước đó.
- Stream-oriented
  - Như khối nhưng với đơn vị là dòng. Một dòng được kết thúc bởi dấu CF.

I/O Buffering



(a) No buffering



(b) Single buffering

Figure 11.6 I/O Buffering Schemes (input)

Double Buffer

- Dùng hai vùng đệm thay vì một
- Một tiến trình có thể chuyển dữ liệu vào và ra một vùng đệm trong khi hệ điều hành truyền thông tin hoặc thao tác trên vùng đệm còn lại

Circular Buffer

- Nhiều hơn hai vùng đệm
- Trong trường hợp thao tác nhập xuất gắn liền với tiến trình.

I/O Buffering

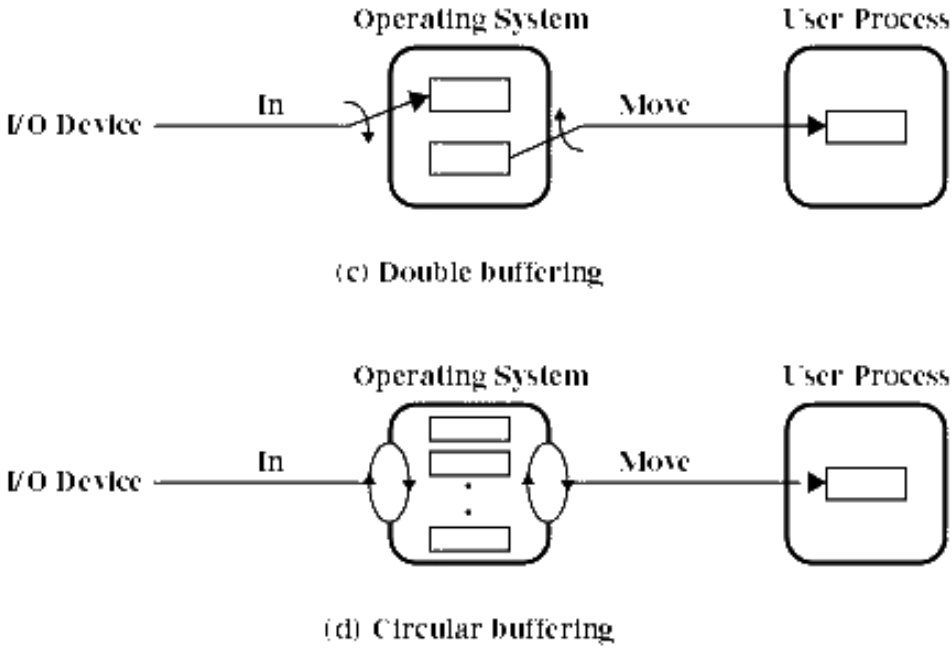
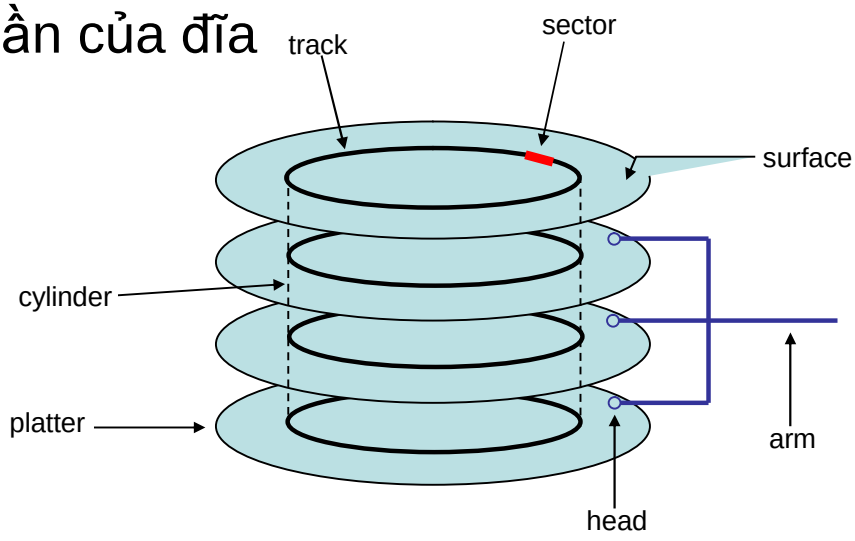


Figure 11.6 I/O Buffering Schemes (input)

Nhắc lại cấu trúc đĩa

- Các thành phần của đĩa
  - platters
  - surfaces
  - tracks
  - sectors
  - cylinders
  - arm
  - heads



Các tham số liên quan đến hiệu năng truy xuất đĩa

- Để đọc hay ghi, đầu đĩa phải di chuyển đến track và sector tương ứng
- Seek time
  - Thời gian dùng để định vị đầu đĩa đến track mong muốn
- Rotational delay hoặc rotational latency
  - Thời gian dùng để di chuyển đầu đĩa đến sector tương ứng (vị trí đầu tiên của sector)
- Transfer time
  - Thời gian chuyển dữ liệu (từ hoặc vào bộ nhớ chính)

Thời gian thi hành tác vụ I/O

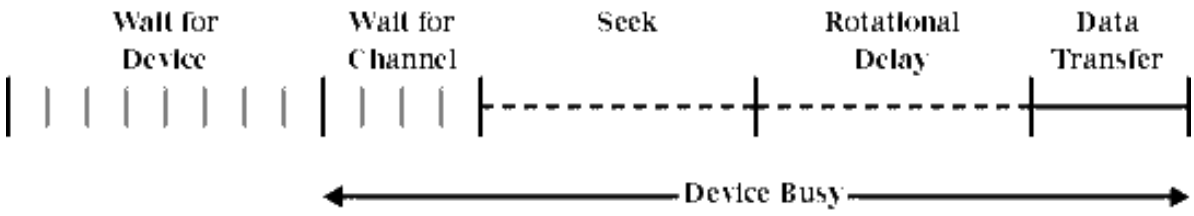


Figure 11.7 Timing of a Disk I/O Transfer

Tham số hiệu năng đĩa

- Access time = seek time + rotational delay
  - Thời gian để đặt đầu đĩa và đúng vị trí để đọc và ghi
- Dữ liệu sẽ được chuyển khi đầu đĩa nằm đúng vị trí sector.

Chính sách lập lịch trên đĩa

- Mấu chốt là Seek time.
  - Vì sao?
- Với một đĩa có thể một số các yêu cầu I/O
- Cần có cơ chế xử lý các yêu cầu này thích hợp → tăng hiệu năng xử lý I/O

Chính sách lập lịch trên đĩa (2)

- First-in, first-out (FIFO)
  - Xử lý yêu cầu một cách tuần tự
  - Công bằng cho tất cả tiến trình
  - Nếu nhiều tiến trình thì xác suất xem như là ngẫu nhiên

Chính sách lập lịch trên đĩa (3)

- Có ưu tiên
  - Mục tiêu không phải là tối ưu đĩa mà để phục vụ cho các mục tiêu khác
  - Các công việc có thời gian thi hành ngắn có thể có độ ưu tiên cao hơn
  - Khả năng tương tác cao

Chính sách lập lịch trên đĩa (4)

- Last-in, first-out
  - Tốt cho các hệ thống xử lý giao dịch (transaction processing system)
    - Thiết bị được giao cho người dùng gần đây nhất → đầu đọc di chuyển ít nhất
  - Có thể xảy ra tình trạng đói (starvation)

Chính sách lập lịch trên đĩa (5)

- First Come, First Service
  - Phương pháp này dễ lập trình nhưng không cung cấp dịch vụ tốt.
  - Ví dụ: cần phải đọc các khối theo thứ tự sau **98 183 37 122 14 75** và đầu đọc đang ở vị trí 59.
  - Đầu đọc sẽ lần lượt đi qua các khối **59 98 183 37 122 12 75**.

Chính sách lập lịch trên đĩa (6)

- Shortest Service Time First (SSTF)
  - Chọn yêu cầu I/O mà đòi hỏi ít di chuyển nhất đầu đĩa từ vị trí hiện tại
  - Luôn luôn cho ra seek time nhỏ nhất.
  - Ví dụ: cần đọc các khối **98 183 37 122 14 124 65 và 67**. Giả sử đầu đọc đang ở vị trí 53.
  - Đầu đọc sẽ lần lượt qua các khối 53 65 67 37 4 98 122 124 183

Chính sách lập lịch trên đĩa (7)

- SCAN
  - Đầu đọc di chuyển theo một hướng nhất định. Ví dụ cần đọc các khối 98 183 37 122 14 124 65 67. Giả sử đầu đọc ở vị trí 53.
  - Đầu đọc sẽ lần lượt qua các khối 53 37 14 0 65 67 98 122 124 183 (theo SCAN định hướng về 0)

Chính sách lập lịch trên đĩa (7)

- C-SCAN
  - Như SCAN, giới hạn chỉ di chuyển theo 1 chiều
  - Quay về vị trí bắt đầu ngay khi đụng track ở xa nhất, không cần phải tiếp tục.
  - Ví dụ cần đọc các khối 98 183 37 122 14 124 65 67. Giả sử đầu đọc ở vị trí 53.
  - Đầu đọc sẽ lần lượt qua các khối 53 65 67 98 122 124 183 199 0 14 37

Disk Scheduling Algorithms

Table 11.3 Disk Scheduling Algorithms [WIED87]

Name	Description	Remarks
Selection according to requestor		
RSS	Random scheduling	For analysis and simulation
FIFO	First in first out	Fairest of them all
PRI	Priority by process	Control outside of disk queue management
LIFO	Last in first out	Maximize locality and resource utilization
Selection according to requested item:		
SSTF	Shortest service time first	High utilization, small queues
SCAN	Back and forth over disk	Better service distribution
C-SCAN	One way with fast return	Lower service variability
N-step-SCAN	SCAN of N records at a time	Service guarantee
FSCAN	N-step-SCAN with N = queue size at beginning of SCAN cycle	Load-sensitive

Câu hỏi ôn tập

- Ví dụ: cần đọc các khối 98 183 124 65 12 và 67. Giả sử đầu đọc đang ở vị trí 53.
- Đầu đọc sẽ lần lượt qua các khối nào và vẽ sơ đồ theo các thuật toán đọc đĩa FCFS, SSTF, SCAN và C-SCAN.
- Từ đó kết luận xem cách đọc nào tối ưu nhất.

Địa chỉ tương đối = (head, cylinder, sector)

Head : 0 → Số mặt trên đĩa – 1

Cylinder: 0 → Số cylinder trên đĩa – 1

Sector : 1 → Số sector trên track

Địa chỉ tuyệt đối = head * Số sector/track + track * số sector/cylinder + sector - 1

Chương 1:Tổng quan về Hệ điều hành

- Khái niệm về hệ điều hành
- Phân loại các hệ điều hành
- Cấu trúc của hệ điều hành
- Lịch sử phát triển của hệ điều hành

Chương 1:Tổng quan về Hệ điều hành

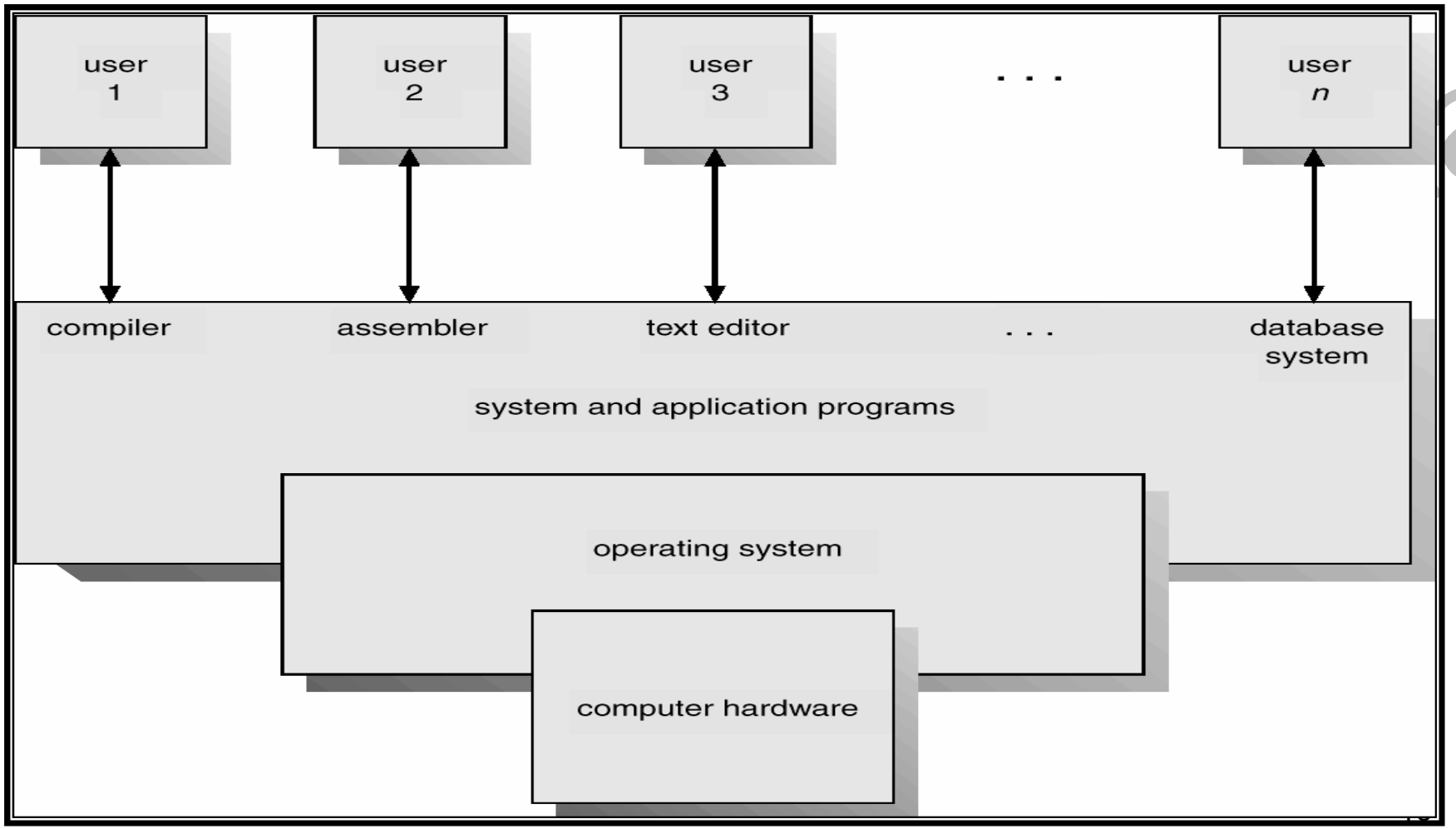
- Khái niệm về hệ điều hành
  - Các thành phần cộng tác trong Hệ thống CNTT gồm: phần cứng, HĐH, các ứng dụng và người sử dụng (user)
  - Phần cứng (CPU, bộ nhớ, thiết bị IO…): tài nguyên máy tính
  - Chương trình ứng dụng (trình biên dịch, hệ quản trị CSDL, phần mềm thương mại, trò chơi…) sử dụng tài nguyên máy tính để giải quyết các yêu cầu của user.
  - HĐH là chương trình hoạt động giữa user và phần cứng máy tính, điều khiển quản lý tài nguyên và phối hợp sử dụng phần cứng cho những ứng dụng khác nhau, giúp giao tiếp người-máy thuận lợi hiệu quả.

Chương 1:Tổng quan về Hệ điều hành

Users

Chương 1:Tổng quan về Hệ điều hành

Mô hình trừu tượng của 1 máy tính



Chương 1:Tổng quan về Hệ điều hành

- Khái niệm về hệ điều hành (tt)
  - HĐH là bộ điều phối tài nguyên của máy tính (thời gian sử dụng CPU, bộ nhớ, đĩa, thiết bị IO) cho các ứng dụng
  - Khi có nhiều yêu cầu khai thác tài nguyên, HĐH phải giải quyết vấn đề tranh chấp và quyết định cấp phát tài nguyên như thế nào là hiệu quả nhất
  - Để gia tăng hiệu quả khai thác tài nguyên, HĐH phải quản lý các bộ ĐKTB của nhà sản xuất (vd: VGA card, sound card, modem, printer, HDD…)

➔ Mục tiêu cơ bản của HĐH:

  - Giúp cho việc thi hành các chương trình dễ dàng hơn
  - Hỗ trợ các thao trên hệ thống máy tính hiệu quả hơn

Chương 1:Tổng quan về Hệ điều hành

- Phân loại các hệ điều hành
  - Hệ thống xử lý theo lô (batch) đơn giản
  - Hệ thống xử lý theo lô đa chương
  - Hệ thống chia xẻ thời gian
  - Hệ thống song song
  - Hệ thống phân tán
  - Hệ thống xử lý thời gian thực

Chương 1:Tổng quan về Hệ điều hành

1. ***Hệ thống xử lý theo lô đơn giản (đơn nhiệm, đơn chương)***

   Hệ điều hành xử lý theo lô thực hiện các công việc lần lượt theo những chỉ thị định trước. Việc thực hiện dãy các công việc một cách tự động nhờ một chương trình luôn nằm thường trú trong bộ nhớ máy tính được gọi *bộ giám sát thường trực*

   Ưu điểm:Thời gian thực hiện chương trình nhanh.

   Nhược điểm: CPU còn nhiều thời gian nhàn rỗi khi làm việc thiết bị nhập xuất:

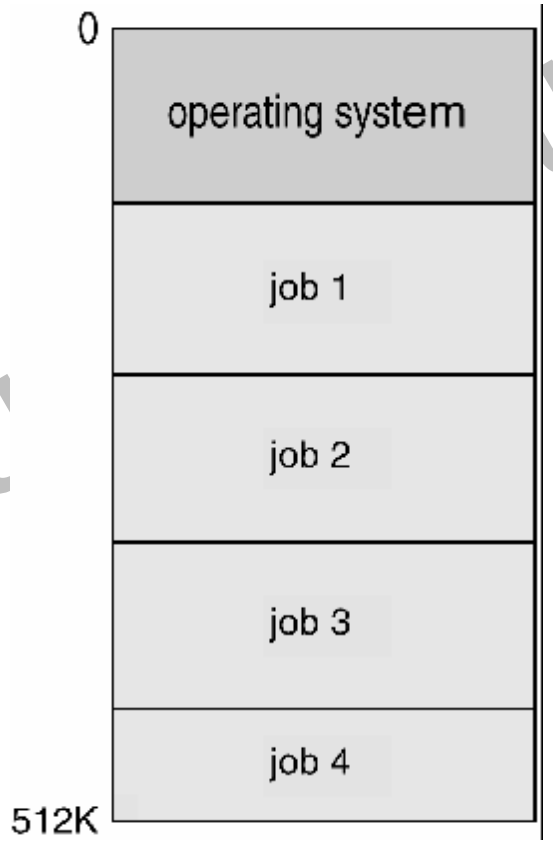
   Khắc phục: Xử lý off_line, Spooling.

Chương 1:Tổng quan về Hệ điều hành

Multi-programmed Systems

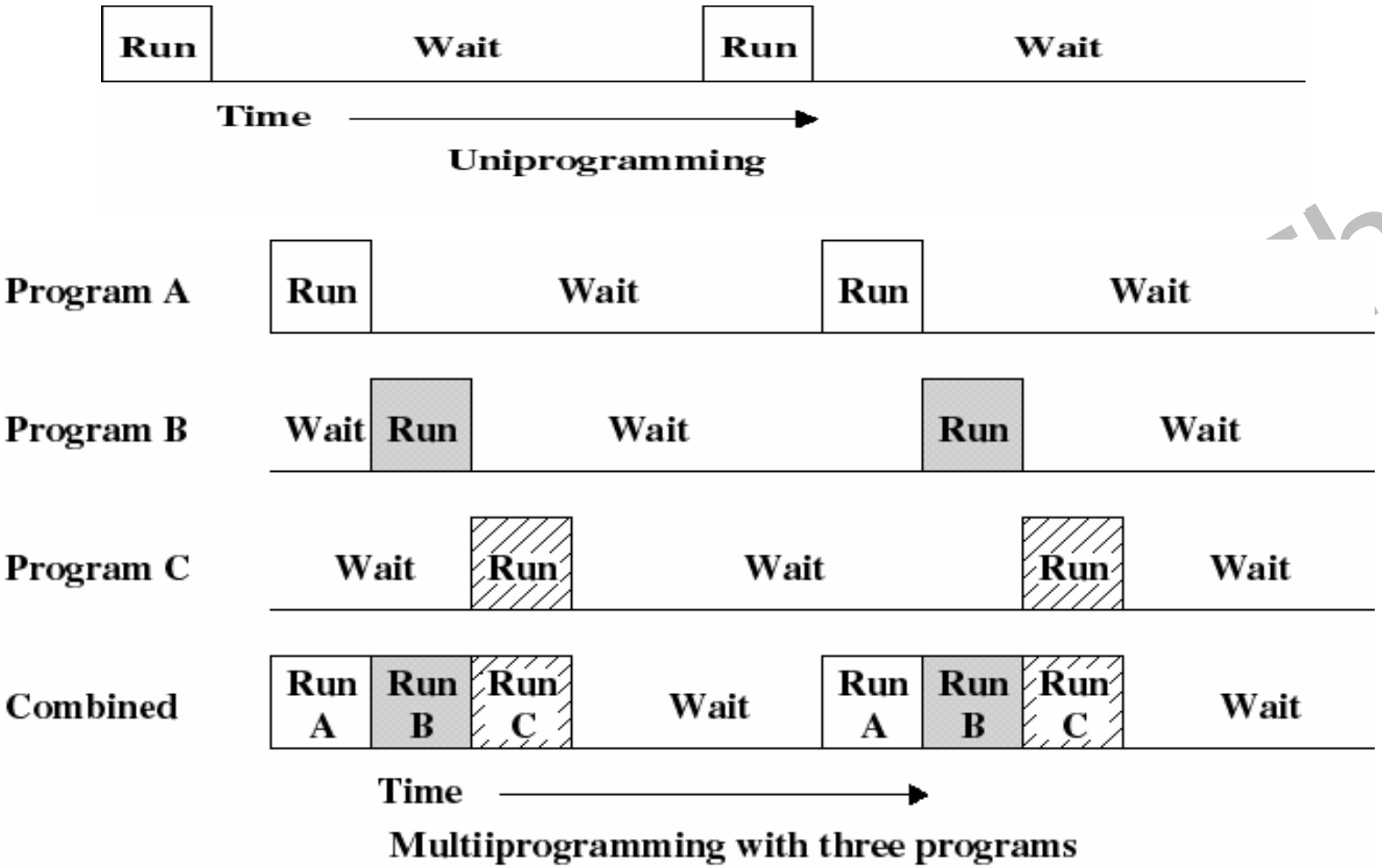
2. Hệ thống xử lý đa chương
   - Job pool: cấu trúc dữ liệu cho phép OS lựa chọn công việc được thực thi kế tiếp
   - Nhiều công việc được nạp vào bộ nhớ chính cùng lúc, thời gian xử lý của CPU được phân chia giữa các công việc đó
   - Tận dùng được thời gian rảnh, khi một công việc nào đó phải chờ I/O thì phải nhường CPU cho công việc khác (overlapping CPU - I/O).

Yêu cầu đối với OS
   - Job Scheduling
   - Memory management
   - CPU scheduling
   - Allocation of devices
   - Protection



Chương 1:Tổng quan về Hệ điều hành

So sánh multi-programming và uni-programming



Chương 1:Tổng quan về Hệ điều hành

3. ***Hệ thống chia sẻ thời gian*** Time-Sharing (multitasking) Systems
   - Multi-programmed systems không cung cấp khả năng tương tác với users
   - CPU luân phiên chuyển đổi thực thi giữa các công việc

     Quá trình chuyển đổi xảy ra ***thường xuyên hơn***, mỗi công việc chỉ được chia một phần nhỏ thời gian CPU

     Cung cấp sự tương tác giữa hệ thống với user
     - Khi kết thúc thực thi một lệnh, OS sẽ chờ lệnh kế tiếp từ bàn phím chứ không phải từ card reader

     Một công việc chỉ được chiếm CPU để xử lý khi nó nằm trong bộ nhớ chính

     Khi cần thiết, một công việc nào đó có thể được chuyển từ bộ nhớ chính ra thiết bị lưu trữ, nhường bộ nhớ chính cho công việc khác.

Chương 1:Tổng quan về Hệ điều hành

Yêu cầu đối với OS trong Time-Sharing Systems

- Định thời công việc (job scheduling)
- Quản lý bộ nhớ (Memory Management)
  - Các công việc được hoán chuyển giữa bộ nhớ chính và đĩa
  - Virtual memory: cho phép một công việc có thể được thực thi mà không cần phải nạp hoàn toàn vào bộ nhớ chính
- Quản lý các process (Process Management)
  - Định thời CPU (CPU scheduling)
  - Đồng bộ các công việc (synchronization)
  - Tương tác giữa các công việc (process communication)
  - Tránh Deadlock
- Quản lý hệ thống file, hệ thống lưu trữ (disk management)
- Phân bổ các thiết bị, tài nguyên
- Cơ chế bảo vệ (protection)

Chương 1:Tổng quan về Hệ điều hành

4. ***Hệ thống song song***

- Gồm nhiều bộ vi xử lý cùng chia sẻ hệ thống đường dẫn dữ liệu, đồng bộ, bộ nhớ và các thiết bị ngoại vi. Các bộ vi xử lý liên lạc bên trong với nhau.
- Với nhiều bộ vi xử lý công việc thực hiện được thực hiện sẽ nhanh hơn, những không phải có n vi xử lý nhanh hơn gấp n lần so hệ thống 1 vi xử lý.
- Độ tin cậy trong hệ thống nhiều vi xử lý là rất cao.
- Hệ thống đa xử lý thường sử dụng cách đa xử lý đối xứng. Một số hệ thống đa xử lý bất đối xứng.

Symmetric multiprocessing (SMP)

  - Các processor vận hành cùng một hệ điều hành duy nhất.
  - Nhiều ứng dụng thực thi cùng lúc với performance cao.
  - Đa số các hệ điều hành hỗ trợ SMP.

Asymmetric multiprocessing

  - Mỗi processor thực thi một công việc khác nhau, master processor định thời và phân công việc cho các slave processors.

Chương 1:Tổng quan về Hệ điều hành

5. Hệ thống phân tán.

- Hệ thống phân tán: trên mỗi máy trong mạng cài đặt một hệ điều hành khác nhau. Máy chủ chủ yếu thực hiện một số chức năng sau: Quản lý các kho dữ liệu tập trung, cung cấp một số dịch vụ truyền dữ liệu, tổ chức xử lý khi có yêu cầu từ máy trạm. Mô hình khách chủ(Client-Server) phân chia quản lý.
- Nguyên nhân xây dựng HĐH phân tán:
  - Chia sẻ tài nguyên.
  - Tăng tốc độ tính toán.
  - An toàn
  - Thông tin liên lạc được với nhau

  Yêu cầu cơ sở hạ tầng mạng máy tính
  - LAN,WAN

  Dựa trên mô hình client-server hoặc peer-to-peer

Chương 1:Tổng quan về Hệ điều hành

6. Hệ thống thời gian thực

Hệ thống thời gian thực (Real-Time Systems)

- Thường dùng trong các thiết bị chuyên dụng như điều khiển các thử nghiệm khoa học, điều khiển trong y khoa, dây chuyền công nghiệp.
- Ràng buộc tương đối chặt chẽ về thời gian: hard và soft real-time.

Hard real-time:

- Hạn chế (hoặc không có) bộ nhớ phụ, tất cả dữ liệu nằm trong bộ nhớ chính (RAM) hoặc ROM
- Yêu cầu thời gian đáp ứng, xử lý rất nghiêm ngặt, thường sử dụng trong điều khiển công nghiệp, công nghệ robotics.

Soft real-time

- Thường xuất hiện trong lĩnh vực multimedia, thực tế ảo (virtual reality) với yêu cầu mềm dẻo hơn về thời gian.

Chương 1:Tổng quan về Hệ điều hành

1.3.1 Các thành phần của hệ thống gồm:

- Quản lý tiến trình
- Quản lý bộ nhớ chính
- Quản lý bộ nhớ phụ
- Quản lý hệ thống nhập xuất
- Quản lý hệ thống tập tin
- Hệ thống bảo vệ
- Hệ thống cơ chế dòng lệnh

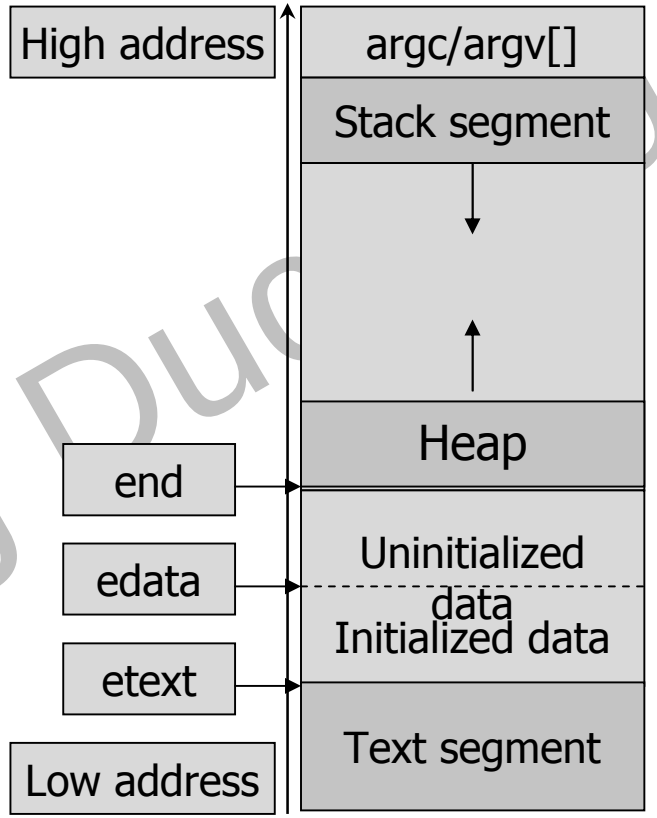
Chương 1:Tổng quan về Hệ điều hành

- Quản lý tiến trình
  - Tiến trình là một loạt các công việc được thi hành (proram, batch processing, spooling, connecting…)
  - Tiến trình sử dụng tài nguyên máy tính (CPU, bộ nhớ, tập tin, thiết bị…) để phục vụ công việc của nó
  - Khi tiến trình khởi tạo, nó đòi hỏi nhiều tài nguyên hệ thống.
  - Khi tiến trình kết thúc, HĐH phải thu hồi hoặc tái tạo tài nguyên để có thể dùng lại cho các tiến trình khác
  - Tiến trình được biên dịch thành các tập tin thụ động trên đĩa
  - Khi tiến trình được kích hoạt, HĐH sẽ khởi tạo tài nguyên ban đầu theo yêu cầu, nạp tập chỉ thị vào bộ nhớ và thi hành theo cơ chế tuần tự. Tiến trình chuyển sang hoạt động

Chương 1:Tổng quan về Hệ điều hành

Cấu trúc 1 tiến trình của UNIX

```
1.  int a = 0, b, *c;
2.  int main( int argc, char *argv[ ] ) {
3.        b= increase(a);
4.        c =(int*)malloc(10*sizeof(int));
5.        c[5]= b;
6.  }
7.  int  increase(int x) {
8.        return x ++;
9.  }
```



Chương 1:Tổng quan về Hệ điều hành

- Quản lý bộ nhớ chính
  - Bộ nhớ chính là trung tâm của các thao tác và xử lý
  - Về mặt vật lý, bộ nhớ là các chip nhớ tĩnh điện
  - Về mặt luận lý, bộ nhớ là mảng các bit nhớ tổ chức theo đơn vị lưu trữ (byte, word hay dword)
  - Mỗi ô nhớ được HĐH định vị bằng cơ chế đánh địa chỉ riêng
  - Các bộ ĐKTB, HĐH, ứng dụng… đều lưu trữ dữ liệu vào bộ nhớ
  - Các chương trình muốn thi hành phải được ánh xạ thành địa chỉ tuyệt đối, nạp vào bộ nhớ chính để hệ thống truy xuất đến
  - Khi tiến trình kết thúc, dữ liệu vẫn còn trong bộ nhớ cho đến khi một tiến trình khác ghi chồng lên (hoặc tắt máy)

Chương 1:Tổng quan về Hệ điều hành

- Quản lý bộ nhớ chính (tt)
  - Do ứng dụng có nhu cầu sử dụng bộ nhớ khác nhau, nên HĐH phải có nhiều kế hoạch quản trị bộ nhớ hiệu quả
  - Kế hoạch quản trị bộ nhớ của HĐH phụ thuộc vào đặc điểm phần cứng và nhu cầu sử dụng của user
  - Vai trò của HĐH trong việc quản lý bộ nhớ chính:
    - Cấp phát và thu hồi bộ nhớ khi cần thiết
    - Quyết định tiến trình nào được nạp vào bộ nhớ chính, địa chỉ nạp
    - Lưu giữ thông tin về các vị trí trong bộ nhớ đã sử dụng, tiến trình nào đang sử dụng

Chương 1:Tổng quan về Hệ điều hành

- Quản lý bộ nhớ phụ
  - Nhược điểm của bộ nhớ chính:
    - Quá nhỏ để có thể lưu giữ mọi dữ liệu và chương trình
    - Mất dữ liệu khi tắt máy
  - Hệ thống lưu trữ phụ dùng đĩa lưu trữ chương trình, dữ liệu
  - Vai trò của HĐH trong việc quản lý đĩa:
    - Định vị lưu trữ, truy xuất đĩa
    - Quản lý vùng trống
    - Lập lịch cho đĩa
  - Hiệu năng của hệ thống tùy thuộc rất nhiều vào tốc độ đọc/ghi. Vì vậy HĐH phải có cơ chế quản lý đĩa hiệu quả.

Chương 1:Tổng quan về Hệ điều hành

- Quản lý hệ thống nhập xuất
  - Mỗi nhà sản xuất đều cung cấp các device driver ĐKTB
  - Nhiệm vụ của HĐH là tạo mặt giao tiếp thân thiện giữa user và thiết bị thông qua các giao thức ĐKTB tổng quát
  - Một hệ thống nhập xuất bao gồm:
    - Hệ thống buffer cahing
    - Giao tiếp điều khiển thiết bị tổng quát
    - Bộ điều khiển cho các thiết bị phần cứng

Chương 1:Tổng quan về Hệ điều hành

- Quản lý hệ thống tập tin
  - Máy tính có thể lưu trữ thông tin trên nhiều dạng thiết bị vật lý khác nhau (băng từ, đĩa từ, đĩa quang, thẻ nhớ…)
  - Mỗi dạng thiết bị lưu trữ có đặc điểm riêng về tổ chức vật lý, tốc độ truy xuất, khả năng lưu trữ, tốc độ truyền…
  - HĐH định nghĩa đơn vị lưu trữ logic là tập tin cùng với cấu trúc đĩa luận lý để truy xuất thông qua bộ điều khiển đĩa
  - Tập tin: tập hợp thông tin của chương trình và/hoặc dữ liệu
  - Các loại tập tin:
    - Thi hành: chứa ứng dụng, phần mềm. Vd: COM, EXE
    - Văn bản: chứa ký tự ASCII, chương trình nguồn
    - Nhị phân: chứa dữ liệu của các ứng dụng

Chương 1:Tổng quan về Hệ điều hành

- Quản lý hệ thống tập tin (tt)
  - Vai trò của HĐH trong việc quản lý tập tin:
    - Tạo, xóa tập tin
    - Tạo, xóa thư mục
    - Hỗ trợ các thao tác trên tập tin và thư mục
    - Ánh xạ tập tin trên hệ thống lưu trữ phụ
    - Backup tập tin trên các thiết bị lưu trữ

Chương 1:Tổng quan về Hệ điều hành

- Hệ thống bảo vệ
  - Hệ thống có nhiều user, nhiều tiến trình đồng thời
  - HĐH cung cấp cơ chế đảm bảo tài nguyên (tập tin, bộ nhớ, CPU, đồng hồ, thiết bị ngoại vi…) chỉ được truy xuất bởi những tíên trình có quyền
  - Hệ thống bảo vệ là cơ chế kiểm soát quá trình truy xuất của chương trình, tiến trình, user trên tài nguyên hệ thống
  - Gia tăng độ an toàn khi kiểm tra lỗi của các giao tiếp giữa những hệ thống nhỏ bên trong

Chương 1:Tổng quan về Hệ điều hành

- Hệ thống cơ chế dòng lệnh
  - Mỗi HĐH có những giao tiếp khác nhau: cơ chế dòng lệnh, giao diện có các biểu tượng, cửa sổ thao tác dùng chuột…
  - Cơ chế dòng lệnh là giao tiếp tương tác lệnh giữa user và HĐH
  - Các HĐH thiết kế cơ chế dòng lệnh bên trong hạt nhân hoặc tách cơ chế dòng lệnh thành một ứng dụng đặc biệt:
    - DOS: thi hành khi bắt đầu công việc (COMMAND.COM)
    - Unix: thi hành khi user login lần đầu tiên
  - Các lệnh được đưa vào HĐH nhờ bộ thông dịch lệnh qua cơ chế dòng lệnh hoặc Shell và được thực hiện tuần tự
  - Các lệnh có quan hệ với việc tạo và quản lý các tiến trình, kiểm soát nhập xuất, quản lý bộ nhớ, quản lý đĩa, truy xuất tập tin và cơ chế bảo vệ.

Chương 1:Tổng quan về Hệ điều hành

- Các dịch vụ của hệ điều hành
  - Thi hành chương trình: Nạp chương trình, chấm dứt bình thường hay bất thường (lỗi nghiêm trọng)
  - Thao tác nhập xuất: cung cấp dịch vụ hỗ trợ nhập xuất
  - Thao tác trên hệ thống tập tin, truy xuất đĩa
  - Truyền thông điệp giữa các tiến trình (máy đơn, mạng)
  - Phát hiện lỗi do CPU, bộ nhớ, thiết bị, tiến trình gây ra. Mỗi dạng lỗi được HĐH giải quyết tương ứng

Chương 1:Tổng quan về Hệ điều hành

- Lời gọi hệ thống
  - Cung cấp giao tiếp giữa tiến trình và HĐH. Có 2 dạng:
    - Cấp thấp: các lệnh hợp ngữ
    - Cấp cao: các hàm/thủ tục thiết kế bằng NNLT cấp cao.
  - Trong các NLT cấp cao, user không quan tâm đến chi tiết mà chỉ cần thông qua các hàm hay các lệnh để gọi thực hiện
  - Có 3 phương pháp chuyển tham số cho HĐH: thanh ghi, ngăn xếp hoặc cấu trúc bảng
  - Các loại lời gọi hệ thống: kiểm soát tiến trình, tao tác tập tin, thao tác thiết bị, truyền thông điệp

Chương 1:Tổng quan về Hệ điều hành

- Các chương trình hệ thống
  - Thao tác tập tin, mô tả tập tin
  - Thông tin trạng thái (ngày giờ, dung lượng đĩa, bộ nhớ…)
  - Hỗ trợ các NNLT: trình biên dịch, thông dịch tích hợp trong nhân hay phát hành riêng
  - Nạp và thi hành chương trình: nạp, định vị, liên kết, debug
  - Thông điệp: dùng liên lạc giữa các tiến trình, các máy
  - Các chương trình ứng dụng đi kèm: định dạng, sao chép đĩa, soạn thảo văn bản, vẽ hình đơn giản…

Chương 1:Tổng quan về Hệ điều hành

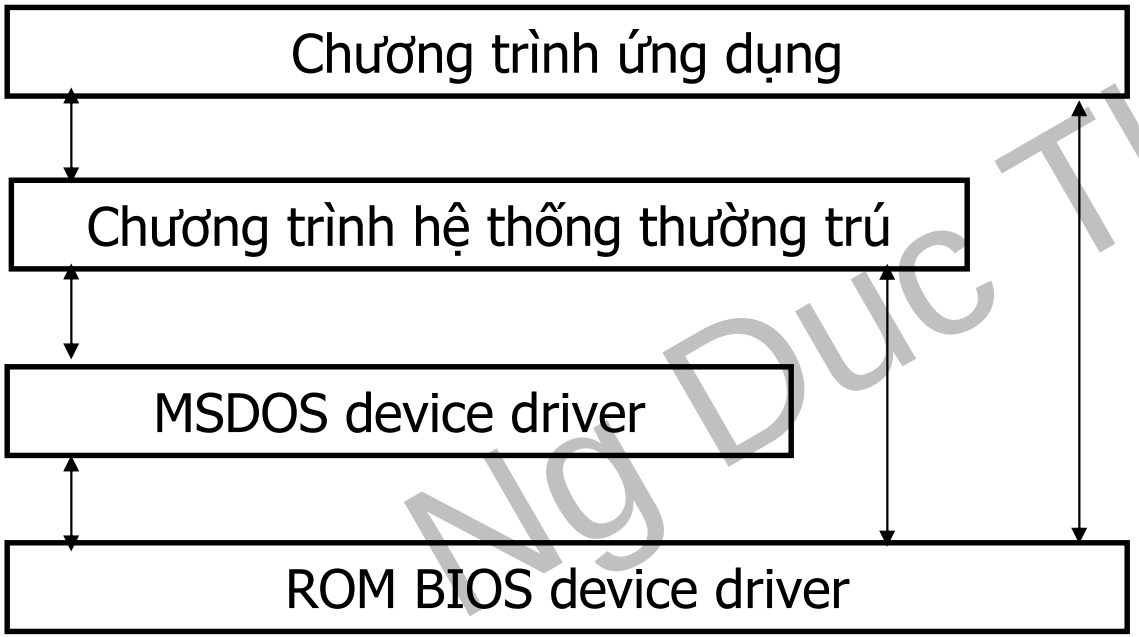
- Cấu trúc hệ thống
  - Cấu trúc đơn giản
  - Cấu trúc theo lớp
  - Cấu trúc Máy ảo
  - Mô hình client-server

Chương 1:Tổng quan về Hệ điều hành

- Cấu trúc đơn giản
  - Khởi đầu của HĐH là hệ thống nhỏ, đơn giản và có giới hạn
  - Các HĐH cấu trúc đơn giản quen thuộc: MSDOS, UNIX
  - Hệ điều hành MSDOS:
    - HĐH cấu trúc đơn giản, không chia thành những đơn thể rõ rệt
    - Cung cấp những chức năng cần thiết nhất trong không gian nhỏ, chạy trên phần cứng giới hạn
    - Giữa giao diện và chức năng không phân chia rõ rệt
    - Các ứng dụng có thể truy xuất trực tiếp thủ tục nhập xuất

Chương 1:Tổng quan về Hệ điều hành

Cấu trúc của MS-DOS



Chương 1:Tổng quan về Hệ điều hành

- Cấu trúc đơn giản
  - HĐH UNIX: cấu trúc gồm hạt nhân và chương trình hệ thống
    - Hạt nhân gồm chuỗi giao tiếp và bộ ĐKTB (device driver)
    - Hạt nhân cung cấp hệ thống tập tin, lập lịch CPU, quản trị bộ nhớ và những chức năng khác thông qua lời gọi hệ thống
    - Lời gọi hệ thống định nghĩa giao tiếp lập trình cho UNIX
    - Các chương trình hệ thống dùng lời gọi hệ thống do hạt nhân hỗ trợ để cung cấp những chức năng hữu ích như biên dịch và thao tác tập tin

Chương 1:Tổng quan về Hệ điều hành

Cấu trúc UNIX

<table>
<tr><td colspan="3">Người sử dụng</td></tr>
<tr><td colspan="3">Shell và lệnh
Biên dịch và thông dịch
Thư viện hệ thống</td></tr>
<tr><td colspan="3">Giao tiếp lời gọi hệ thống với hạt nhân</td></tr>
<tr><td>Tín hiệu kiểm soát hệ thống, nhập xuất tuần tự của bộ điều khiển terminal</td><td>Hệ thống tập tin chuyển đổi giữa hệ thống nhập xuất khối và bộ điều khiển đĩa</td><td>Lập lịch CPU, thay thế phân trang, yêu cầu phân trang trong bộ nhớ ảo</td></tr>
<tr><td colspan="3">Giao tiếp giữa hạt nhân với phần cứng</td></tr>
<tr><td>Bộ kiểm soát terminal</td><td>Bộ kiểm soát đĩa</td><td>Bộ kiểm soát bộ nhớ</td></tr>
</table>

Chương 1:Tổng quan về Hệ điều hành

- Cấu trúc theo lớp
  - Chia chức năng hệ thống thành nhiều phần nhỏ
  - Cấm ứng dụng sử dụng các hàm truy xuất cấp thấp
  - HĐH được chia thành nhiều lớp: lớp phần cứng, lớp hệ thống, lớp giao tiếp user…
  - Mỗi lớp gồm một số CTDL và các hàm được gọi từ lớp trên. Bản thân lớp chỉ gọi được các chức năng của lớp dưới hỗ trợ
  - Mỗi lớp không cần biết HĐH và các lớp khác được cài đặt như thế nào. Nó chỉ biết nhiệm vụ và các thao tác trên lớp

Chương 1:Tổng quan về Hệ điều hành

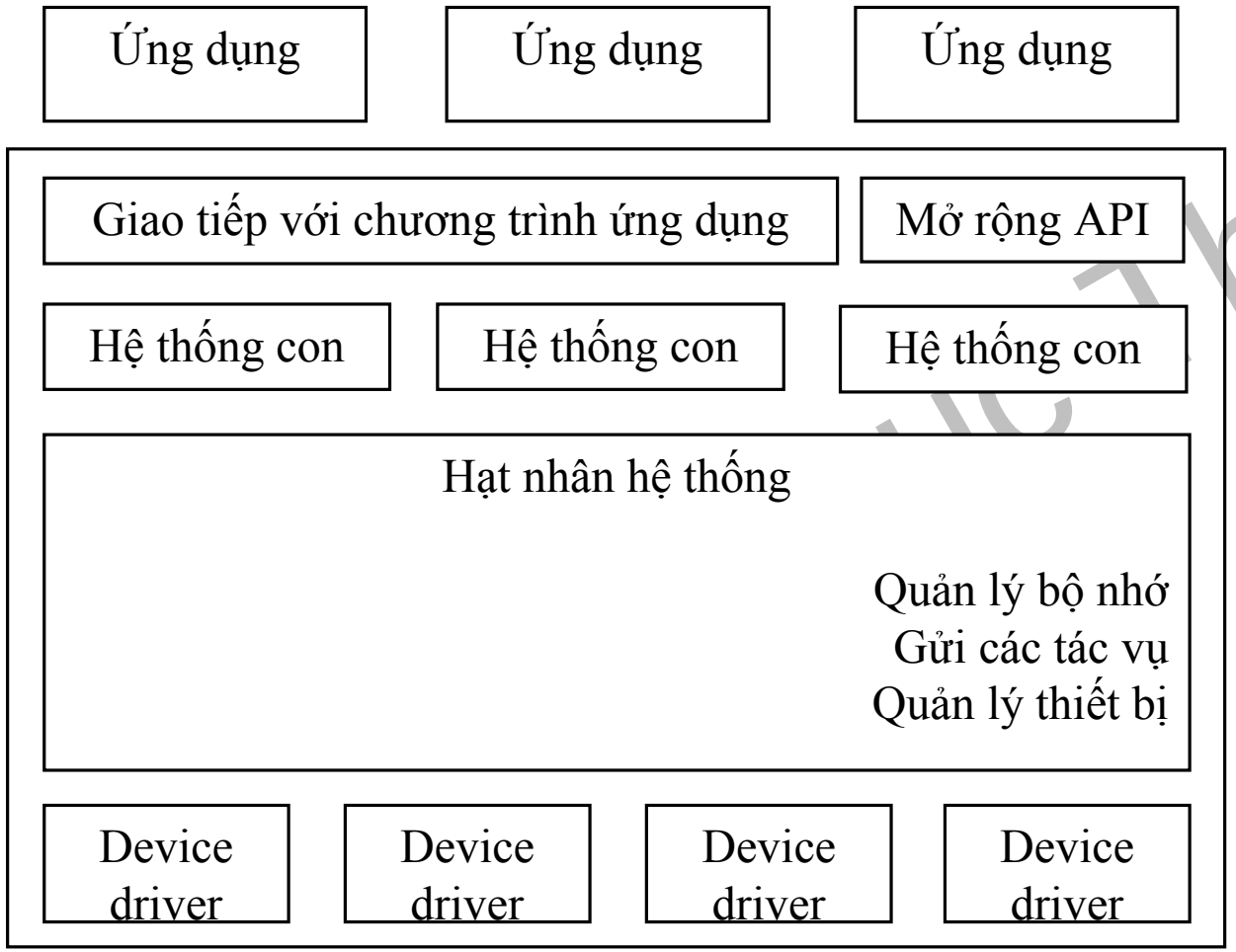
Cấu trúc của Hệ điều hành THE

- HĐH Technische Hogesschool Eindhoven
  - Lớp 5: Chương trình ứng dụng
  - Lớp 4: Tạo buffer cho thiết bị nhập xuất
  - Lớp 3: Device driver thao tác màn hình
  - Lớp 2: Quản lý bộ nhớ ảo
  - Lớp 1: Lập lịch CPU
  - Lớp 0: Phần cứng

Chương 1:Tổng quan về Hệ điều hành

- Các lớp của HĐH VENUS
  - Lớp 6: Chương trình ứng dụng
  - Lớp 5: Device driver và bộ lập lịch
  - Lớp 4: Bộ nhớ ảo
  - Lớp 3: Kênh nhập xuất
  - Lớp 2: Lập lịch CPU
  - Lớp 1: Thông dịch các địa chỉ
  - Lớp 0: Phần cứng

Chương 1:Tổng quan về Hệ điều hành



Chương 1:Tổng quan về Hệ điều hành

- Máy ảo (Virtual Machine)
  - HĐH = <Ứng dụng>|<Hệ thống>|<Nhân>|<Phần cứng>
  - Nhân HĐH dùng chỉ thị của phần cứng để tạo thư viện Lời gọi hệ thống
  - Các chương trình hệ thống có thể sử dụng lời gọi hệ thống hoặc chỉ thị phần cứng: (SysCall+HWare) ∈ SysClass
  - Các ứng dụng có thể gọi các chương trình hệ thống: (SysFunc+SysClass) ∈ Virtual Machine
  - Bằng kỹ thuật lập lịch CPU và bộ nhớ ảo, HĐH có thể tạo nhiều tiến trình phức ảo hoạt động như một máy tính có bộ xử lý và bộ nhớ riêng.

Chương 1:Tổng quan về Hệ điều hành

- Máy ảo (tt)
  - Các tiến trình của máy ảo hoạt động như lời gọi hệ thống cùng hệ thống tập tin không truy xuất trực tiếp phần cứng
  - Tài nguyên của máy tính được chia xẻ để tạo các máy ảo: CPU ảo, bộ nhớ ảo, thiết bị ảo, tập tin (và đĩa) ảo.
  - Máy ảo thực hiện ở 2 mức: giám sát và sử dụng
  - Vấn đề cơ bản của máy ảo: hệ thống đĩa (ảo), vận chuyển dữ liệu và thời gian thi hành
  - Lợi thế của máy ảo: mỗi ứng dụng hoạt động độc lập vì được cung cấp môi trường như một máy riêng (CPU, bộ nhớ, bộ chỉ thị, thiết bị ngoại vi…).

Chương 1:Tổng quan về Hệ điều hành

- Mô hình client-server
  - Các đoạn mã hệ thống có xu hướng chuyển dần lên lớp cao.
  - Chức năng của hạt nhân được chuyển bớt cho các tiến trình của user, chỉ còn lại các chức năng cơ bản (vd: kiểm soát quá trình thông tin giữa client-server)
  - HĐH được chia thành các phần nhỏ, mỗi phần kiểm soát một mặt của hệ thống
  - Mô hình hoạt động: client gửi yêu cầu, server thực hiện, gửi kết quả cho client
  - Ưu điểm: tương thích với mô hình hệ thống phân tán

Chương 1:Tổng quan về Hệ điều hành

- Lịch sử phát triển các hệ điều hành
  - Thế hệ 1: 1945-1955
  - Thế hệ 2: 1955-1965
  - Thế hệ 3: 1965-1980
  - Thế hệ 4: 1980 - nay

Chương 1:Tổng quan về Hệ điều hành

- Thế hệ 1: 1945-1955
  - Máy tính điện tử dùng đèn do Howard Aiken và John von Neuman thiết kế. Kích thước lớn. Tốc độ tính toán chậm
  - Chưa có NNLT và HĐH. Chưa có sự phân công: người thiết kế, lập trình, thao tác, quản lý, sử dụng… đều là một
  - Lập trình bằng ngôn ngữ máy tuyệt đối
  - Sử dụng bảng điều khiển thực hiện các tính năng cơ bản
  - 1950: phiếu đục lỗ ra đời, thay cho bảng điều khiển

Chương 1:Tổng quan về Hệ điều hành

- Thế hệ 2: 1955-1965
  - Máy tính điện tử bán dẫn được sản xuất cho khách hàng
  - Kích thước và tốc độ được cải thiện
  - Có sự phân chia rõ ràng giữa người thiết kế, xây dựng, vận hành, lập trình, bảo trì.
  - NNLT: sử dụng Hợp ngữ, FORTRAN mã hóa chỉ thị trên phiếu đục lỗ. Xuất kết quả ra máy in
  - Hệ thống xử lý theo lô (lưu trên băng từ) ra đời
  - Chương trình điều khiển hệ xử lý theo lô là tiền thân của các HĐH ngày nay

Chương 1:Tổng quan về Hệ điều hành

- Thế hệ 3: 1965-1980
  - Máy IBM360 được trang bị mạch tích hợp IC
  - Kích thước giảm thiểu. Tốc độ gia tăng.
  - Các thiết bị ngoại vi bắt đầu xuất hiện. Thao tác điều khiển dần dần phức tạp hơn.
  - Hệ điều hành ra đời, từ đơn giản đến phức tạp: hệ đa chương, hệ chia xẻ thời gian
  - Hệ máy mini bắt đầu phát triển

Chương 1:Tổng quan về Hệ điều hành

- Thế hệ 4: 1980-nay
  - Máy tính cá nhân (PC-Personal Computer) ra đời
  - Hệ máy IBMPC sử dụng HĐH MS-DOS chiếm lĩnh thị trường máy cá nhân trên toàn thế giới
  - HĐH mạng, HĐH phân tán phát triển mạnh
  - Giao diện đồ họa (GUI-Graphic User Interface), Multimedia, Internet…

Chương 2: Giới thiệu 1 số Hệ điều hành

- Hệ điều hành MSDOS
- Hệ điều hành Windows
- Hệ điều hành Windows NT
- Hệ điều hành Novel Netware
- Hệ điều hành Unix

Chương 2: Giới thiệu 1 số Hệ điều hành

- Hệ điều hành MSDOS
  - Khái niệm
  - Lịch sử phát triển
  - Quá trình khởi động
  - Tập lệnh của MSDOS

Chương 2: Giới thiệu 1 số Hệ điều hành

- Khái niệm về MSDOS
  - Microsoft Disk Operating System là một trong những HĐH được sử dụng rộng rãi nhất trên thế giới
  - MSDOS (gọi tắt là DOS) là hệ điều hành có cấu trúc đơn giản, đơn nhiệm (monotasking), một người dùng, truy xuất hệ thống nhiều cấp từ cao đến thấp
  - Giao tiếp dòng lệnh dùng tham số (Shell từ DOS 4.0)
  - Đòi hỏi cấu hình thấp: CPU AT/XT, 640KB bộ nhớ, màn hình màu CGA/EGA/VGA hoặc đơn sắc. Cho phép mở rộng thiết bị

Chương 2: Giới thiệu 1 số Hệ điều hành

- Khái niệm về MSDOS (tt)
  - Tổ chức các chương trình của MSDOS bao gồm:
    - Chương trình khởi động
    - Chương trình Shell
    - Chương trình chức năng
    - Chương trình nhập xuất
    - Hệ thống các tiện ích

Chương 2: Giới thiệu 1 số Hệ điều hành

- Lịch sử phát triển của MSDOS
  - Tháng 8-1981, Microsoft phát hành HĐH MSDOS 16 bits cho máy PC có CPU 8088 theo đơn đặt hàng của IBM
  - Version 1.0 có 4000 dòng Hợp ngữ, 12KB bộ nhớ, tổ chức thành 3 tập tin:
    - IBMBIO.COM: hệ thống nhập xuất tuần tự và đĩa
    - IBMMSDOS.COM: hệ thống tập tin, kiểm soát IO, giao tiếp
    - COMMAND.COM: trình xử lý lệnh (nội trú, ngoại trú)
  - Đề xuất khái niệm thiết bị IO độc lập (CON, PRN, AUX), định dạng tập tin ứng dụng COM, EXE. Hỗ trợ xử lý lệnh theo lô
  - Version 1.1: Đĩa mềm lưu trữ 5.2 inch (360 KB, 2 mặt).

Chương 2: Giới thiệu 1 số Hệ điều hành

- Lịch sử phát triển của MSDOS (tt)
  - Version 2.0: phát hành vào 3-1983 gồm 20,000 dòng lệnh, cài trên đĩa cứng 10MB.
  - Chuyển IBMMIO.COM thành tập danh sách liên kết thiết bị.
  - Sử dụng cấu trúc đĩa thứ bậc, thẻ file, đường ống, lọc sắp xếp, device driver cấu trúc mở…
  - Cài đặt thiết bị, định dạng ngày tháng, tiền tệ, mã quốc gia vào lúc khởi động trong CONFIG.SYS.
  - Version 2.11 được chuyển thành hơn 60 ngôn ngữ

Chương 2: Giới thiệu 1 số Hệ điều hành

- Lịch sử phát triển của MSDOS (tt)
  - Version 3.0 phát hành 8-1984 gồm 40,000 dòng lệnh.
  - Hỗ trợ máy IBMPC AT, 20MB đĩa cứng, đĩa mềm 1.2MB
  - Sử dụng cấu hình CMOS, đĩa ảo VDISK
  - Version 3.1 (11-1984) giới thiệu Microsoft Network
  - Version 3.2 (1986) hỗ trợ đĩa mềm 3.5". Cung cấp các lệnh REPLACE, XCOPY, DRIVER.SYS
  - Version 3.3 hỗ trợ cổng truyền thông, ổ đĩa 1.44MB, đĩa cứng 32MB trên máy AT 80286, 80386
  - Version 4.0 (7-1988) hỗ trợ đĩa cứng đến 2GB. Sử dụng RAM mở rộng làm đĩa ảo. Giới thiệu MS-DOS Shell.

Chương 2: Giới thiệu 1 số Hệ điều hành

- Lịch sử phát triển của MSDOS (tt)
  - Version 5.0 (4-1991), sử dụng RAM mở rộng để lưu user device driver. Tận dụng vùng nhớ 640-1MB để giải phóng bớt vùng nhớ cơ sở
  - Cải tiến Shell cho phép nạp nhiều ứng dụng cùng lúc
  - Chú trọng nhiều hơn đến user. Bổ sung Help mở rộng
  - Version 6.0 (1993): tăng dung lượng đĩa bằng DBLSPACE. Tạo cache đĩa bằng SMARTDRV. Tối ưu bộ nhớ bằng MEMMAKER. Chống virus với MSAV. Phục hồi tập tin bằng UNDELETE. Kiểm tra đĩa bằng SCANDISK.

Chương 2: Giới thiệu 1 số Hệ điều hành

- Quá trình khởi động. Các khái niệm cơ bản của DOS
  - Lệnh nội trú: thư viện xử lý các lệnh phổ biến, thường trú trong bộ nhớ
  - Lệnh ngoại trú: lệnh đặc biệt, ít dùng, lưu trên đĩa dưới dạng các tập tin thi hành
  - Các dạng thông báo lỗi: không tìm thấy lệnh, lệnh sai cú pháp, không thể thực hiện lệnh
  - Khởi động máy bằng đĩa mềm hoặc đĩa cứng.
  - Mục tiêu của quá trình khởi động:
    - Kiểm tra sự hoạt động của thiết bị
    - Cài đặt chương trình của HĐH

Chương 2: Giới thiệu 1 số Hệ điều hành

- Quá trình khởi động máy tính
  - POST-Power On Self Test: do chương trình trong ROM thực hiện (trước khi HĐH được kích hoạt):
    - Kiểm tra thiết bị
    - Cài đặt thông số hệ thống vào RAM
    - Nạp Master boot/Boot sector vào bộ nhớ. Chuyển điều khiển
  - Khởi động từ đĩa mềm: POST-Boot sector-HĐH
  - Khởi động từ đĩa cứng: POST-Master boot-Boot sector-HĐH
  - Master boot chứa đoạn chương trình nạp Boot sector vào bộ nhớ chính và bảng phân hoạch đĩa cứng
  - Boot sector chứa đoạn chương trình nạp HĐH vào bộ nhớ chính và bảng tham số đĩa

Chương 2: Giới thiệu 1 số Hệ điều hành

- Quá trình khởi động HĐH MSDOS:
  - Cài đặt MSDOS vào RAM
  - Khởi tạo các vector ngắt tương ứng
  - Tìm đọc tập tin cấu hình CONFIG.SYS:
    - Nếu tìm thấy, thiết lập cấu hình theo chỉ dẫn.
    - Nếu không thấy, thiết lập cấu hình chuẩn
  - Nạp DOS Shell, chuyển điều khiển
  - Thực hiện tập tin lô AUTOEXEC.BAT
  - Xuất dấu đợi lệnh (DOS Prompt):
    - Khởi động từ đĩa mềm: A:\>_
    - Khởi động từ đĩa cứng: C:\>_

Chương 2: Giới thiệu 1 số Hệ điều hành

- Các tập lệnh của MSDOS:
  - Thông tin hệ thống
  - Làm việc với đĩa
  - Làm việc với thư mục
  - Thao tác với tập tin
  - Quản lý nhập xuất
  - Thiết lập môi trường

Chương 2: Giới thiệu 1 số Hệ điều hành

- Hệ điều hành Windows
  - Khái niệm
  - Lịch sử phát triển
  - Quá trình khởi động
  - Tập lệnh

Chương 2: Giới thiệu 1 số Hệ điều hành

- Khái niệm về Hệ điều hành Windows
  - HĐH đa nhiệm của Microsoft. Giao diện đồ họa thân thiện, dễ sử dụng, thích hợp cho mọi đối tượng người dùng
  - Yêu cầu phần cứng không cao: 386SX, VGA, 4MB RAM, 80MB HDD. Quản lý thiết bị tốt, trong suốt với user
  - Tương thích với máy IBMPC. Các HĐH khác có thể cùng hoạt động với Windows. DOS trở thành ứng dụng của Windows. HĐH có tính ổn định cao, ít hỏng hóc.
  - Khai thác được khả năng tối đa của máy. Cài đặt thay đổi cấu hình hệ thống dễ dàng: Plug and Play (Windows 95)

Chương 2: Giới thiệu 1 số Hệ điều hành

- Khái niệm về Hệ điều hành Windows (tt)
  - Giao diện cửa sổ, biểu tượng. Hỗ trợ tên file dài, giao tiếp dữ liệu OLE.
  - Chạy trong chế độ bảo vệ (protected mode). Định vị bộ nhớ phẳng. Phá vỡ rào cản 640 MB bộ nhớ cơ sở.
  - Hỗ trợ các ứng dụng 32 bits. Giao tiếp trực tiếp với các thiết bị thông tin. Liên kết trong các hệ thống mạng.
  - Dễ học, dễ sử dụng, hỗ trợ lập trình tốt… Windows đã đưa Microsoft trở thành 1 trong những công ty hàng đầu thế giới trong lĩnh vực CNTT

Chương 2: Giới thiệu 1 số Hệ điều hành

- Lịch sử phát triển của HĐH Windows
  - 11-1983 công bố. 11-1985 phát hành version 1.0
  - Version 2.0 (11-1987) thay đổi giao diện, cửa sổ phủ lấp. Mở rộng giao tiếp bàn phím, chuột cho menu và dialog box
  - 1987-1990: Windows 2.x (80386 Virtual mode). Multitask các ứng dụng DOS, Windows. Truy xuất trực tiếp phần cứng
  - 5-1990: Windows 3.0 protect mode 80x86. Truy cập 16MB bộ nhớ. Shell (Program, Task, File Manager) hoàn chỉnh.
  - 4-1992: Windows 3.1 protect mode. Hỗ trợ font True type, multimedia, OLE, common dialog box.
  - 9-1995: Windows 95 GUI hoàn chỉnh. Độc lập thiết bị, hỗ trợ CD-ROM, Plug&Play. Hỗ trợ mạng cục bộ, remote control.

Chương 2: Giới thiệu 1 số Hệ điều hành

- Quá trình khởi động của HĐH Windows
  - POST-Power On Self Test
  - Nạp các thành phần cho chế độ thực
  - Nạp VxD tĩnh và chế độ bảo vệ
  - Nạp HĐH Windows và khởi tạo desktop

Chương 2: Giới thiệu 1 số Hệ điều hành

- Nạp các thành phần cho chế độ thực (real-mode)
  - Boot sector nạp IO.SYS. Chuyển điều khiển
  - IO.SYS có chức năng quản lý cấu hình real-mode, graphic mode. IO.SYS nạp các tập tin tối thiểu, phân tích tập tin CONFIG.SYS, thi hành đồ họa (real-mode, protected mode)
  - Xác định tiểu sử phần cứng (hardware profile). Khởi tạo thiết bị phần cứng cho chế độ thực. Nạp MSDOS.SYS
  - MSDOS.SYS định vị địa chỉ các tập tin và các chức năng khởi động Windows 95.
  - Xử lý CONFIG.SYS. Nạp COMMAND.COM để xử lý lệnh ở chế độ DOS thực. Thi hành AUTOEXEC.BAT. Chuyển điều khiển cho quá trình nạp bộ quản lý máy ảo VMM32 ở chế độ thực

Chương 2: Giới thiệu 1 số Hệ điều hành

- Nạp VxD tĩnh và chế độ bảo vệ (protected-mode)
  - VMM32 nạp các thiết bị mô tả trong SYSTEM.INI. Tìm driver cho thiết bị. Nếu tìm không thấy, nạp driver mặc định
  - Chuyển từ chế thực sang chế độ bảo vệ
  - Khởi động bộ quản lý cấu hình chế độ bảo vệ. Khởi tạo các VxD có sẵn. Xác định các VxD cần nạp thêm
  - Nhận dạng thông tin phần cứng
  - Giải quyết các xung đột về tài nguyên
  - Khởi tạo các thiết bị

Chương 2: Giới thiệu 1 số Hệ điều hành

- Nạp Windows 95 files, khởi tạo desktop
  - Nạp SYSTEM.INI: thông tin cấu hình hệ thống
  - Nạp KERNEL32.DLL: hạt nhân của Windows 95
  - Nạp GDI.EXE, GDI32.DLL: cơ chế đồ họa của Windows 95
  - Nạp USER.EXE, USER32.DLL giao tiếp user, quản lý cửa sổ
  - Liên kết tài nguyên, font
  - Nạp WIN.INI chứa các cấu hình về chương trình và user
  - Nạp Shell (EXPLORER.EXE), các thành phần của desktop
  - Nếu sử dụng mạng, xuất hiện hộp thoại login.

Chương 2: Giới thiệu 1 số Hệ điều hành

- Tập lệnh (Shell) Windows 95
  - Thao tác với các đối tượng
    - Các đối tượng: file, folder, program, printer, modem…
    - Tên: dùng mô tả và xác định đối tượng
    - Thuộc tính: kích thước, vị trí, tên, thời gian…
  - Các thành phần của desktop
    - My Computer: Drives, Control Panel, Printer, DialUp
    - Shortcut: Name, Address, Icon, Application…
    - Network Neighborhood: Workgroup, Server, Printer, Shared Folder…
  - Recycled Bin: Restore Deleted Items
  - Taskbar: Start button, Quick Launch, System Tray
  - Start: Programs, Documents, Settings, Find, Run, Help, Shutdown

Chương 2: Giới thiệu 1 số Hệ điều hành

- Hệ điều hành Windows NT. Khái niệm
  - HĐH mạng dành cho LAN và Workgroup của Microsoft
  - Đúc kết từ ưu điểm của các HĐH khác như Unix, OS/2…
  - Windows NT có 2 phiên bản: Server và Workstation
  - Windows NT Workstation phục vụ nhiều chức năng mạng: print server, router TCP/IP…
  - Windows NT sử dụng profiles lưu trữ thông tin chi tiết về môi trường của user qua các phiên làm việc
  - Vừa là HĐH máy đơn, vừa là file server, application server, print server…, Windows NT là HĐH thân thiện, mạnh mẽ và linh hoạt

Chương 2: Giới thiệu 1 số Hệ điều hành

- Hệ điều hành Windows NT. Đặc điểm
  - Dễ tương thích (portability):
    - NT được viết bằng ngôn ngữ C
    - Các phần được xây dựng trên lớp phần cứng trừu tượng
    - Chạy được trên các máy Intel x86, RISC (MIPS, AXP, PowerPC)
  - Khả năng bộ nhớ: định vị đến 4GB địa chỉ bộ nhớ
  - Đa nhiệm: thực hiện nhiều tiến trình cùng lúc
  - Hỗ trợ nhiều CPU:
    - Workstation 4.0: 2 CPU
    - Server 4.0: 4 CPU
    - Chuyên dụng: 255 CPU

Chương 2: Giới thiệu 1 số Hệ điều hành

- Hệ điều hành Windows NT. Đặc điểm (tt)
  - Hệ thống tập tin NTFS (New Technology File System):
    - Tên file dài đến 255 ký tự. Phân biệt chữ hoa, thường
    - Logical Partition đến 1064 GB
    - Nén file/folder giảm kích thước 40-50%
  - An toàn: nhiều cấp độ an tòan trên mức chia xẻ, quyền truy cập dữ liệu mạng
  - Tương thích: hỗ trợ nhiều loại ứng dụng từ nhiều HĐH khác nhau (MSDOS, Windows 16bits, Windows 32 bits, OS/2, POSIX)

Chương 2: Giới thiệu 1 số Hệ điều hành

- Hệ điều hành Windows NT. Đặc điểm (tt)
  - Hỗ trợ mạng:
    - Dễ quản trị: user profiles, quản trị domain, lập lịch tác vụ…
    - Hỗ trợ API (Application Programming Interface): NetBIOS , Socket…
    - Hỗ trợ giao thức truyền thông: NetBEUI, NWLink, TCP/IP, DLC
    - Giao tiếp với Netware: các phần mềm client có thể truy xuất đến server Netware
    - Dịch vụ truy xuất từ xa (RAS-Remote Access Service) cho phép user kết nối vào LAN thông qua modem, vào WAN sử dụng X25
    - Dịch vụ Internet: hỗ trợ cho mạng TCP/IP các dịch vụ world wide web, FTP server, Gopher server…

Chương 2: Giới thiệu 1 số Hệ điều hành

- Quản trị user, nhóm user trên Windows NT
  - Bộ quản lý User Manager for Domain
  - Tạo shortcut cho User Manager for Domain
  - Lựa chọn Domain
  - Thêm user account
  - Copy user account
  - Thay đổi thuộc tính của user acount
  - Gán user vào các nhóm
  - Quản lý nhóm

Chương 2: Giới thiệu 1 số Hệ điều hành

- Sử dụng Windows NT Workstation Client
  - Windows NT Workstation có 3 thành phần chính:
    - Nghi thức mạng
    - Định hướng lại network client
    - Cấu hình tài nguyên
  - Logon/Logoff Windows NT Workstation
    - Logon cục bộ
    - Logon domain

Chương 2: Giới thiệu 1 số Hệ điều hành

- Hệ điều hành Novell Netware
  - 1983: mạng S-Net ra đời, là tiền thân của Novell Netware
  - Các năm sau, Novell giới thiệu Netware 86 (file server, truy xuất hệ thống tập tin an toàn, hỗ trợ quản trị mạng)
  - 1986: Advanced Netware hỗ trợ LAN, cho phép nối nhiều kiểu mạng trong file server hoặc workstation bên ngoài
  - Advanced Netware 286 xây dựng trên CPU 80286, multitasking, protected mode. Server hỗ trợ 4 kiểu mạng
  - Version Netware 386 3.0 (1989) là HĐH 32 bits thích hợp cho các mạng lớn. An toàn, hiệu quả, linh động, tin cậy

Chương 2: Giới thiệu 1 số Hệ điều hành

- Hệ điều hành Novell Netware (tt)
  - 1991: Netware 3.11 hỗ trợ tập tin DOS, Macintosh, Windows, OS/2, Unix và các dịch vụ in ấn
  - 1995: Netware 4.1 cung cấp các dịch vụ thiết yếu: thư mục, tập tin, thông điệp hợp nhất, định hướng đa nghi thức, quản trị mạng, an tòan, in ấn.
  - Các sản phẩm của Novell: Netware SFT III 3.12, Netware 4.1 for OS/2, Netware for Unix.

Chương 2: Giới thiệu 1 số Hệ điều hành

- Đặc điểm của HĐH Netware 4.1
  - HĐH mạng 32 bit CPU 80386, cung cấp các dịch vụ cho trạm
  - Dịch vụ xác minh user và login
  - Dịch vụ tập tin: user được gán quyền truy cập server, thư mục, tập tin, quản lý hệ thống tập tin, quyền truy cập
  - Bảo vệ an toàn dữ liệu: kiểm tra ghi dữ liệu, tạo mirror, duplicate thư mục, FAT, HotFix, TTS (Transaction Tracking System), giám sát SFT (System Fault Tolerance)
  - Định hướng: server có thể có nhiều card giao tiếp mạng (NIC-Netware Interface Connector). Netware hỗ trợ định hướng thông tin giữa những người sử dụng trên mạng

Chương 2: Giới thiệu 1 số Hệ điều hành

- Đặc điểm của HĐH Netware 4.1 (tt)
  - Thông điệp: cung cấp hệ thống quản trị thông điệp
  - Quản trị: cung cấp nhiều tiện ích quản trị, giám sát và tối ưu
  - Dịch vụ in ấn: cho phép chia xẻ tập tin, in ấn trên mạng
  - Internet: hỗ trợ WAN và mobile device
  - Cấu hình năng động: sử dụng bộ nhớ tạo cache cho thư mục, buffer định hướng, chỉ mục Turbo FAT, TTS
  - Quản trị bộ nhớ hiệu quả
  - Hệ thống tập tin: tìm kiếm theo cơ chế thang máy, cache file, ghi background, Turbo FAT, nén file, kích thước file đến 4GB, 2 triệu thư mục/volume, mở cùng lúc 10,000 files…

Chương 2: Giới thiệu 1 số Hệ điều hành

- Hệ điều hành Unix
  - Sản phẩm HĐH 32 bits của Bell Lab dành cho máy trạm, máy mini, notebook và các hệ máy đặc biệt
  - Cung cấp môi trường tốt cho phát triển ứng dụng (xử lý văn bản, xử lý tính toán, CASE/CAD/CAM)
  - Dễ chuyển đổi giữa các hệ thống, kết nối các chương trình. Từ khi IBMPC ra đời, Unix được chuyển từ máy mainframe, mini sang PC với hệ Microsoft's XENIX, AT&T System V.
  - Multitasking, multi-user, Unix là HĐH được sử dụng rộng rãi trong các chiến lược mở rộng hệ thống
  - Unix truyền thống có cơ chế dòng lệnh. Ngày nay Unix đã có các HĐH giao diện đồ họa thân thiện hơn.

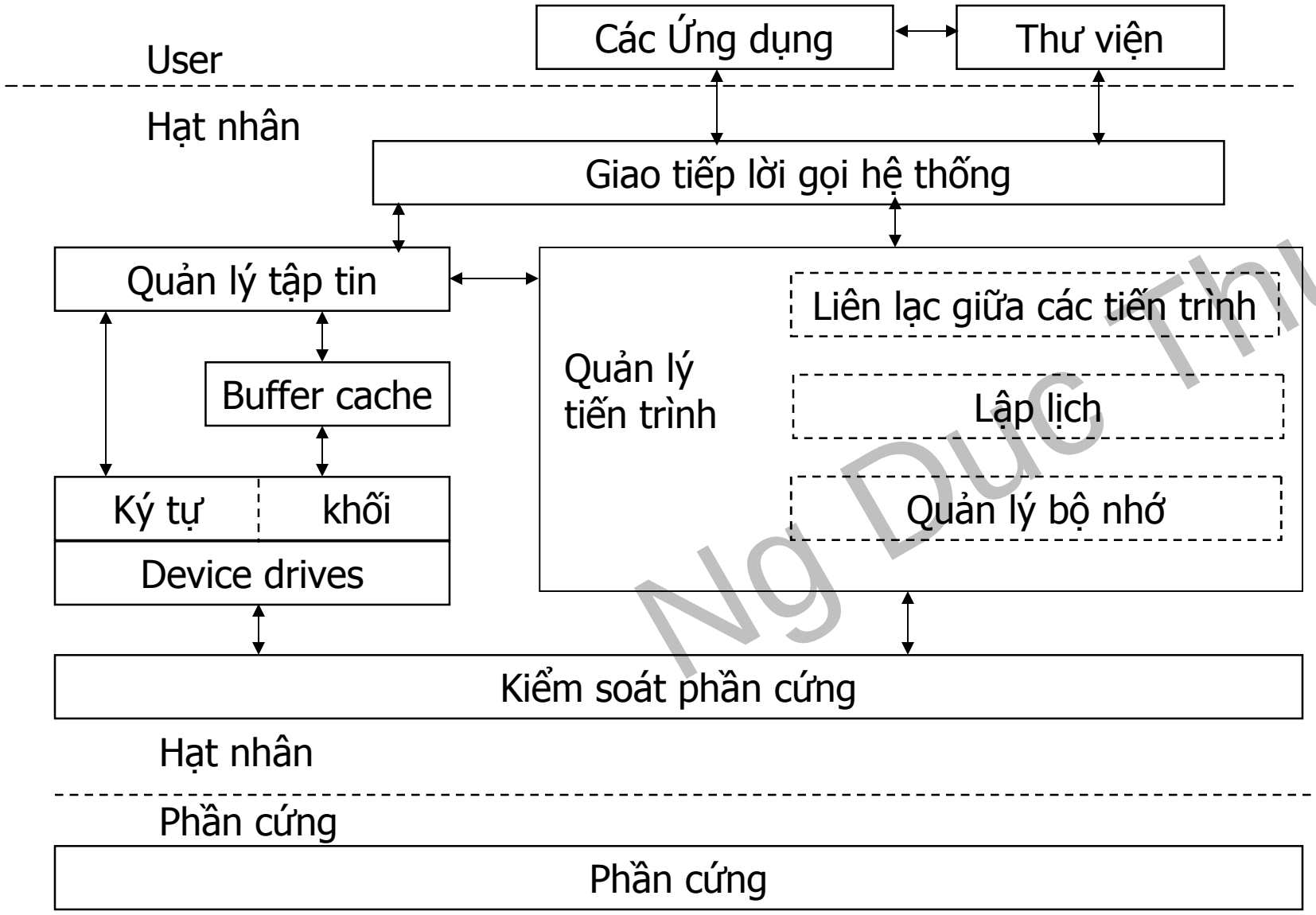
Chương 2: Giới thiệu 1 số Hệ điều hành

- Lịch sử phát triển của HĐH Unix
  - 1965-1969: Bell Lab phát triển hệ Multics trên mainframe
  - 1969: Ken Thompson thiết kế lại cho máy DEC PDP-7
  - 1970: Unix ra đời cho hệ máy PDP-11. Chưa hỗ trợ multitasking, multiprogramming. 24 (16+8) KB bộ nhớ. File tối đa 64KB. Soạn thảo văn bản. Chưa hỗ trợ bảo vệ lưu trữ.
  - 1973: Denis Ritchie viết lại Unix bằng C
  - 1975: Unix mã nguồn mở phổ biến trong các trường đại học
  - 1979: Unix với hệ thống chia xẻ thời gian ra đời. File tối đa 1 tỷ byte. Ngôn ngữ C mở rộng. Bổ sung nhiều shell mạnh hỗ trợ chuỗi, lập trình cấu trúc, chuyển file giữa các máy.

Chương 2: Giới thiệu 1 số Hệ điều hành

- Lịch sử phát triển của HĐH Unix (tt)
  - 1980: XENIX, hệ Unix của Microsoft cho bộ xử lý 16 bits. Có đặc tính sửa chữa khôi phục tập tin tự động, chia sẻ dữ liệu, tăng cường xử lý bên trong
  - Unix của đại học Berkeley: HĐH chia sẻ thời gian, hỗ trợ không gian địa chỉ lớn hơn. Bộ nhớ ảo phân trang, hệ thống tập tin nhanh và mạnh hơn. Xử lý thông tin nội tại. Hỗ trợ mạng cục bộ. Soạn thảo toàn màn hình và shell
  - 1982: Unix System III, V của AT&T hỗ trợ truy xuất công việc từ xa, điều khiển mã nguồn
  - Có rất nhiều công ty cùng tham gia xây dựng Unix. Ngày nay Unix chiếm khoảng 20% trên các hệ thống máy tính

Hạt nhân UNIX



Chương 2: Giới thiệu 1 số Hệ điều hành

- Cấu trúc HĐH Unix: user, hạt nhân và phần cứng
  - Các phần chính của nhân: Quản lý tập tin, quản lý tiến trình
  - Các thư viện liên kết với lời gọi hệ thống để truy cập HĐH
  - Các nhóm Lời gọi hệ thống: nhóm tập tin, nhóm tiến trình
  - Hệ thống quản lý bộ nhớ đồng bộ hóa tiến trình, liên lạc giữa các tiến trình, cấp phát bộ nhớ, lập lịch tiến trình
  - Hệ thống quản lý tập tin và hệ thống quản lý tiến trình cùng tương tác khi nạp tập tin vào bộ nhớ
- Unix Shell, chương trình giao tiếp của user với Unix
  - Unix có nhiều loại shell (Bourne:sh, Korn: ksh, C: csh)
  - Mỗi shell có dấu nhắc khác nhau ($, %…)
  - Các shell có thể thi hành cùng lúc, 1 shell trên 1 shell khác…

Chương 2: Giới thiệu 1 số Hệ điều hành

- Tập lệnh Unix
  - Các lệnh cơ bản
  - Các lệnh liên quan đến tập tin, thư mục
  - Các lệnh soạn thảo
  - Các lệnh tiện ích
  - Các lệnh về mạng
  - Các lệnh chuyển tập tin
  - Các lệnh thông tin liên lạc

Chương 3: Quản lý tiến trình

2.1 Khái niệm

- Tiến trình là một chương trình đang xử lý, sở hữu một con trỏ lệnh, tập thanh ghi và các biến.
- Chương trình là thực thể thụ động chứa đựng các tín hiệu điều khiển máy tính để thực hiện tác vụ nào đó.
- HĐH hỗ trợ đa chương, đa nhiệm. Trong HĐH có nhiều tiến trình cùng hoạt động. Vì vậy việc sử dụng thuật toán để điều phối các tiến trình là nhiệm vụ của HĐH.
- Bộ phận thực hiện chức năng này gọi là bộ điều phối.

Chương 3: Quản lý tiến trình

Trạng thái của một tiến trình:

- Running: *các chỉ thị tiến trình đang được xử lý.*
- Blocked: *tiến trình chờ cấp phát tài nguyên, hay sự kiện nào đó xảy ra.*
- Ready: *tiến trình chờ cấp phát CPU.*

(Create: tạo mới, Destroy: kết thúc)

Chế độ xử lý tiến trình:

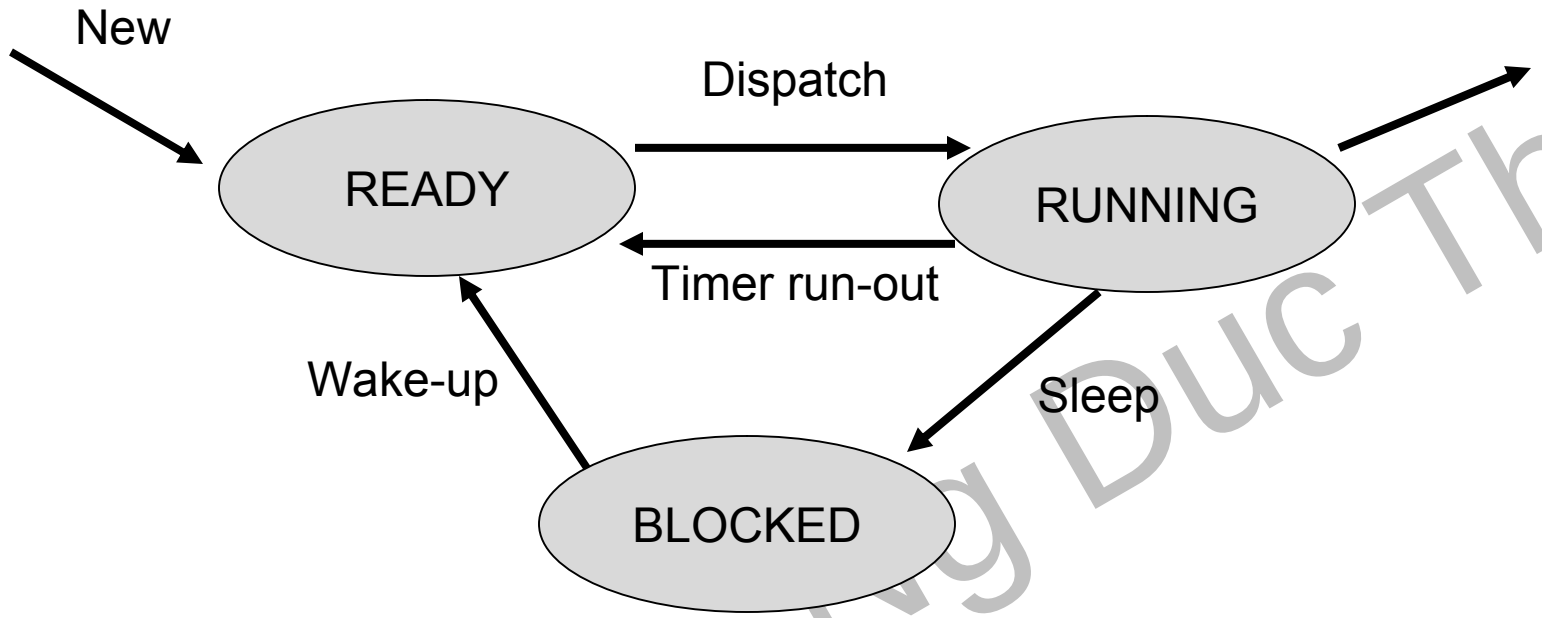
HĐH có hai chế độ xử lý tiến trình:

Chế độ đặc quyền: chỉ có HĐH mới hoạt động được với chế độ độc quyền, nhờ sự giúp đỡ phần cứng.

Chế độ không đặc quyền: người sử dụng

Chương 3: Quản lý tiến trình

Chuyển trạng thái tiến trình



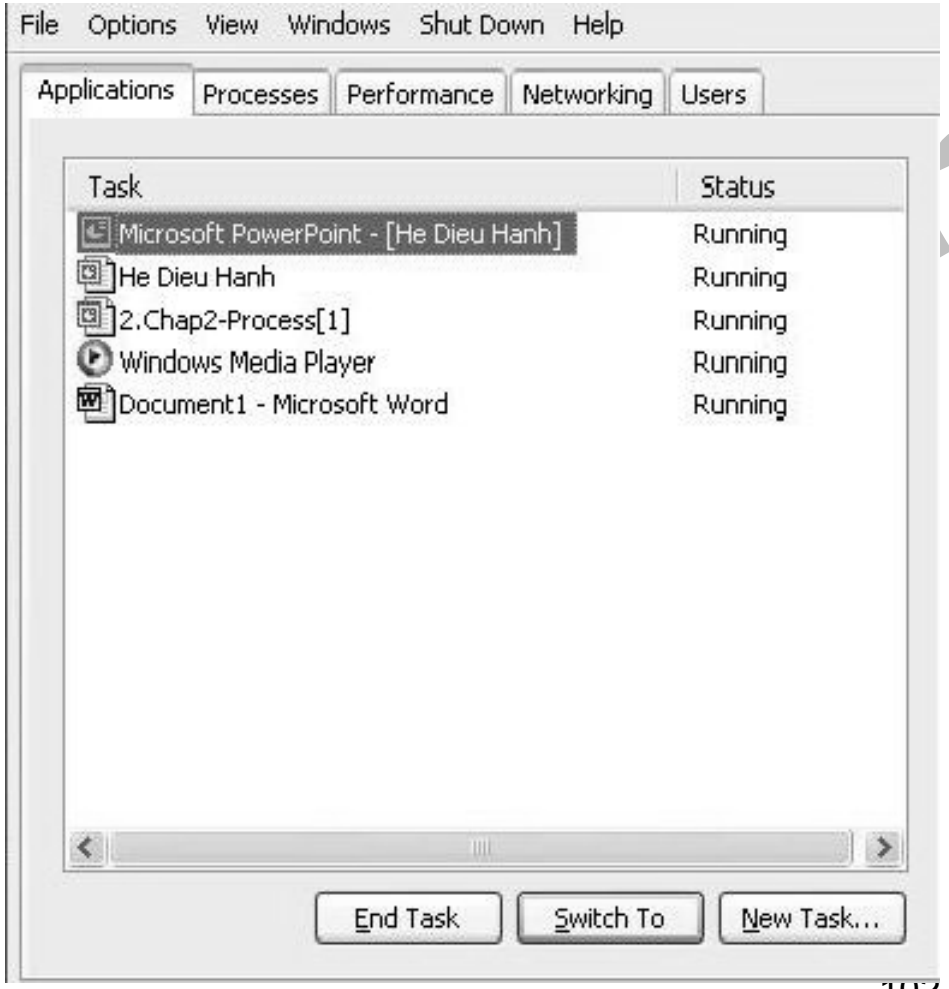
Chương 3: Quản lý tiến trình

- HĐH quản lý thông tin tiến trình qua khối điều khiển (PCB: Process Control Block). Cấu trúc dữ liệu của HĐH để quản lý quá trình
- Chứa thông tin nhận dạng, trạng thái, định vị tài nguyên cho quá trình bao gồm thông tin sau:
  - ✓ Danh định cho quá trình (PID)
  - ✓ Bộ đếm chương trình
  - ✓ Vùng lưu giá trị thanh ghi CPU
  - ✓ Độ ưu tiên của quá trình
  - ✓ Thông tin định vị bộ nhớ quá trình
  - ✓ Thông tin bảo mật
  - ✓ Con trỏ đến các quá trình cha, con
  - ✓ .....

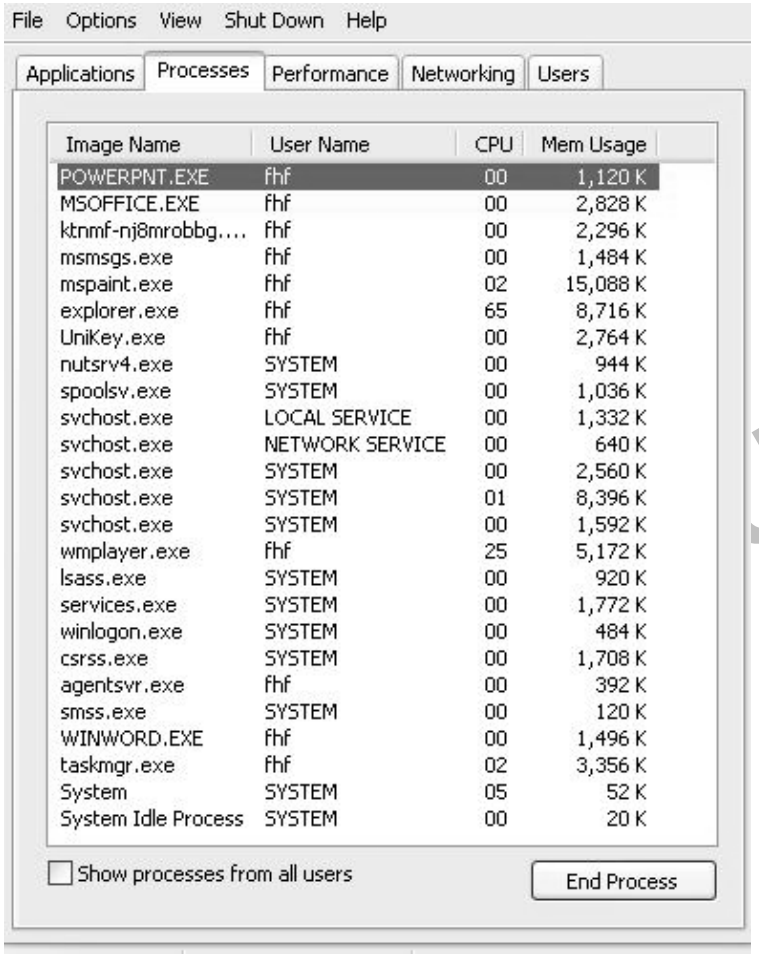
Chương 3: Quản lý tiến trình

Ví dụ

Trong Task Manager



Chương 3: Quản lý tiến trình



Chương 3: Quản lý tiến trình

Thao tác trên tiến trình

HĐH cung cấp các thao tác chủ yếu cho tiến trình là:

☞ Create: tạo lập tiến trình
- ✓ *Định danh tiến trình*
- ✓ *Đưa tiến trình vào danh sách quản lý*
- ✓ *Xác định mức độ ưu tiên*
- ✓ *Cung cấp tài nguyên ban đầu*
- ✓ *Tạo khối điều khiển tiến trình (PCB)*

☞ Destroy: kết thúc tiến trình
- ✓ *Thu hồi tài nguyên đã cấp*
- ✓ *Huỷ bỏ tiến trình khỏi danh sách quản lý*
- ✓ *Huỷ bỏ khối điều khiển tiến trình*

(Lưu ý: tiến trình con không thể tiếp tục khi tiến trình cha kết thúc)

Chương 3: Quản lý tiến trình

2.2 Tiểu trình (threads)

- Mỗi tiến trình thông thường có một không gian địa chỉ và dòng xử lý.
- Trong trường hợp người sử dụng muốn nhiều dòng xử lý cùng chia sẻ một không gian địa chỉ và các dòng dữ liệu xử lý song song như các tiến trình riêng biệt. HĐH cung cấp cơ chế như vậy gọi là tiểu trình.
- Một tiểu trình là một đơn vị xử lý cơ bản trong hệ thống.
- Một tiểu trình có thể tạo nhiều tiến trình con.
- Một tiến trình có thể sở hữu nhiều tiểu trình.

Chương 3: Quản lý tiến trình

2.3 Điều phối tiến trình

Trong môi trường đa chương nhiều tiến trình đồng thời sẵn sàng nhận xử lý. Tiến trình tiếp theo nào được chọn để xử lý cần có giải thuật thích hợp để thực hiện nhiệm vụ này. HĐH có bộ phận thực hiện nhiệm vụ này được gọi bộ điều phối tiến trình.(dispatcher)

1. Mục tiêu điều phối là:

- ✡ Sự công bằng(Fairness)
- ✡ Tính hiệu quả (Efficiency)
- ✡ Thời gian đáp ứng hợp lý(Response time)
- ✡ Thời gian lưu lại trong hệ thống (Turaround Time).
- ✡ Thông lượng tối đa (throughput)

Chương 3: Quản lý tiến trình

2. Các đặc điểm của tiến trình

Mục tiêu cơ bản trong điều phối tiến trình:

- Hướng xuất/nhập của tiến trình
- Hướng xử lý của tiến trình
- Chương trình tương tác hay xử lý theo lô
- Độ ưu tiên của tiến trình
- Thời gian đã sử dụng CPU của tiến trình.
- Thời gian còn lại của tiến trình cần giải quyết

Điều phối độc quyền và không độc quyền

Điều phối độc quyền: Tiến trình chiếm CPU đến khi hoàn tất hoặc tự nguyện giải phóng.

Điều phối không độc quyền: Tiến trình có thể bị tạm dừng bất cứ khi nào.

Chương 3: Quản lý tiến trình

3. Điều phối: HĐH sử dụng 2 loại danh sách trong điều phối:

- Danh sách sẵn sàng
- Danh sách hàng đợi

Danh sách sẵn sàng: Chứa những chương trình thường trú trong bộ nhớ chính. Ở trạng thái săn sàng

Danh sách hàng đợi: Chứa những tiến trình trạng thái Ready.

Các cấp độ điều phối:

- *Điều phối tác vụ*: quyết định số tiến trình vào bộ nhớ
- *Điều phối tiến trình*: quyết định tiến trình nào tiếp theo nhận CPU

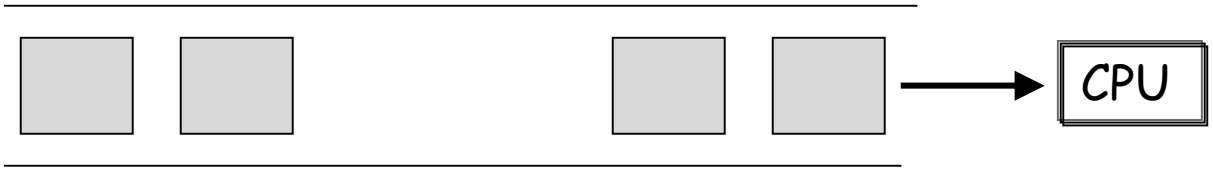
Chương 3: Quản lý tiến trình

4. Các chiến lược điều phối

A. Chiến lược FIFO(First In First Out)

Nguyên tắc: CPU sẽ được cấp phát cho tiến trình đầu tiên trong danh sách sẵn sàng. Đây thuật toán điều phối theo nguyên tắc độc quyền. CPU chỉ có thể thoát khỏi tiến trình khi kết thúc hay có yêu cầu xuất / nhập.

Danh sách sẵn sàng



Chương 3: Quản lý tiến trình

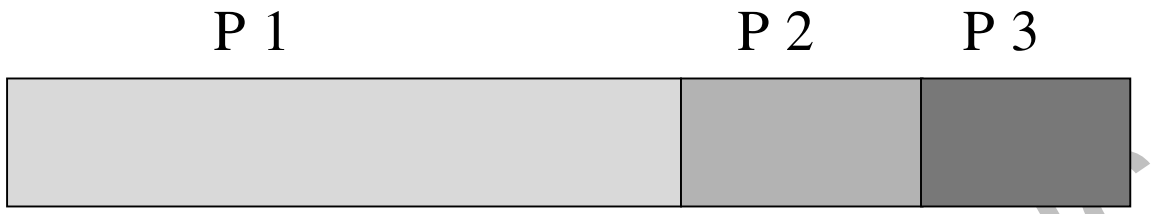
Ví dụ: chiến lược điều phối FIFO

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

Chương 3: Quản lý tiến trình

Ví dụ: chiến lược điều phối FIFO

Thứ tự cấp phát:



Thời gian chờ: P1 là 0

P2 là 24

P3 là 27

Thời gian chờ trung bình:(0+24+27)/3=17ms

Chương 3: Quản lý tiến trình

Nhận xét: chiến lược điều phối FIFO

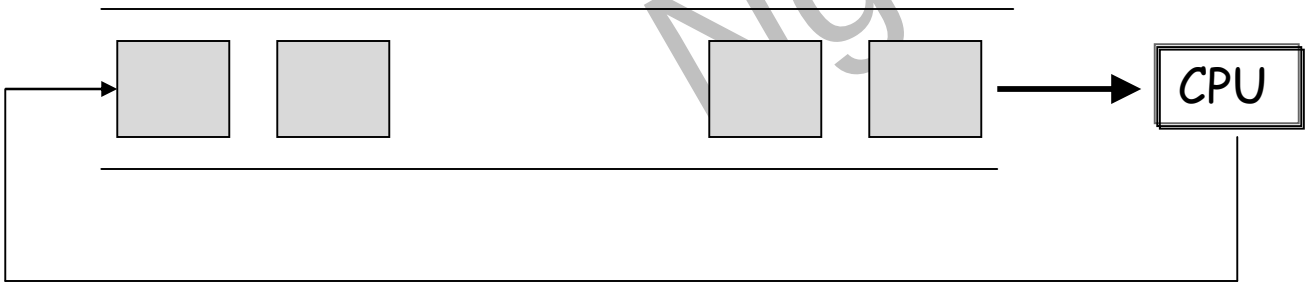
- Thời gian chờ không đạt cực tiểu. Có thể xảy hiện tượng tích lũy thời gian chờ.
- Thay đổi đáng kể thời gian chờ nếu thay đổi thứ tự dãy.
- Giải thuật không phù hợp với việc phân chia thời gian

Chương 3: Quản lý tiến trình

B. Chiến lược phân phối vòng quay(Round Robin)

Nguyên tắc: Danh sách sẵn sàng xử lý là danh sách vòng, bộ điều phối lần lượt cấp phát từng tiến trình trong danh sách khoảng thời gian t.(Giải thuật điều phối không độc quyền)

Danh sách sẵn sàng



Chương 3: Quản lý tiến trình

Ví dụ: chiến lược điều phối RR

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

Chương 3: Quản lý tiến trình

Ví dụ: chiến lược điều phối RR

Thời gian chờ trung bình:(0+4+7+6)/3=5,66ms

Nhận xét:

☞ Giải thuật phụ thuộc vào rất nhiều giá trị t

☞ t quá nhỏ -> sử dụng CPU kém hiệu quả

☞ t quá lớn -> tăng khả năng hồi đáp, giám khả năng tương tác hệ thống.

Chương 3: Quản lý tiến trình

C. Chiến lược Điều phối với độ ưu tiên

Nguyên tắc: Mỗi tiến trình khi vào hệ thống được gán giá trị ưu tiên. Tiến trình có độ ưu tiên cao sẽ được nhận CPU trước. Độ ưu tiên dựa trên rất nhiều tiêu chí như: thời gian xử lý, yêu cầu bộ nhớ.

- Giải thuật này có thể theo nguyên tắc độc quyền và không độc quyền

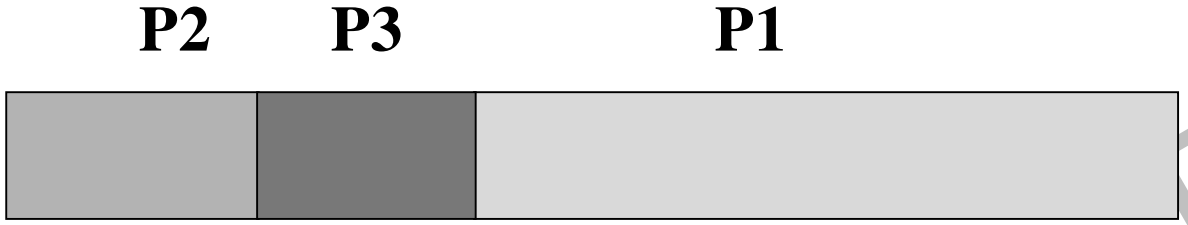
Chương 3: Quản lý tiến trình

Ví dụ: chiến lược điều phối Ưu tiên

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

Chương 3: Quản lý tiến trình

Ví dụ: chiến lược điều phối Ưu tiên



Thời gian chờ trung bình:(0+3+6)/3=2ms

Nhận xét:

- Có thể dẫn đến tiến trình có mức ưu tiên cấp thấp chờ vô hạn.
- Khắc phục bộ điều phối tiến hành giảm độ ưu tiên xuống sau mỗi ngắt đồng hồ cho đến khi tiến trình có mức ưu tiên thấp hơn tiến trình kế cận và chuyển CPU.

Chương 3: Quản lý tiến trình

D. Chiến lược công việc ngắn nhất

(SJF: Shortest Job First)

Nguyên tắc: Điều phối với độ ưu tiên, độ ưu tiên p được gán cho tiến trình là nghịch đảo của thời gian xử lý t mà tiến trình yêu cầu: p=1/t. Tiến trình kế tiếp là tiến trình yêu cầu ít thời gian nhất để kết thúc.

☞ Giải thuật này cũng có thể là độc quyền hay không độc quyền

Chương 3: Quản lý tiến trình

Ví dụ: chiến lược điều phối SJF

Tiến trình	Thời gian xử lý
P1	6
P2	8
P3	7
P4	3

Chương 3: Quản lý tiến trình

Ví dụ: chiến lược điều phối SJF

Thứ tự cấp phát:

P4 **P1** **P3** **P2**

Thời gian chờ trung bình:(3+9+16+0)/4=7ms

Nhận xét:

- Cho phép đạt thời gian chờ trung bình là cực tiểu
- Xác định thời gian yêu cầu xử lý còn lại là rất khó khăn:

Biểu thức dự đoán $\tau_{n+1} = \alpha t_n + (1-\alpha)\tau_n$

Chương 3: Quản lý tiến trình

F. Chiến lược điều phối nhiều mức độ ưu tiên

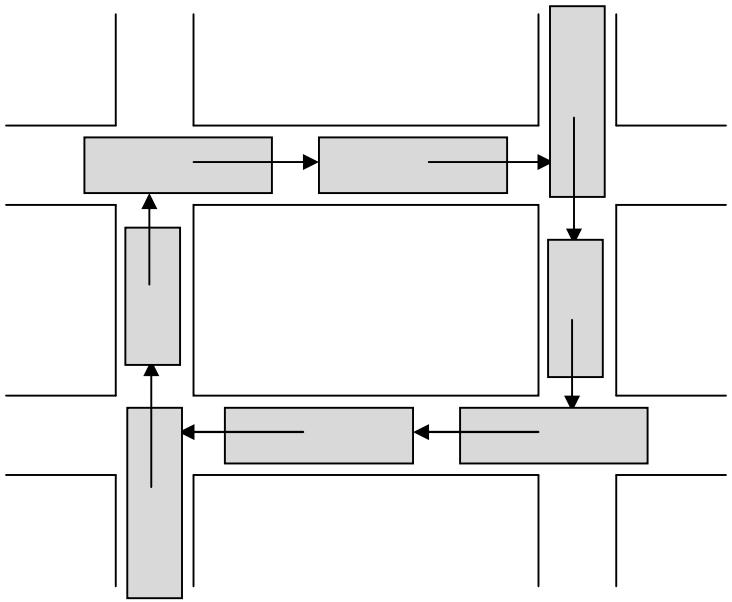
Nguyên tắc: Phân lớp tiến trình tuỳ theo độ ưu tiên của chúng để có cách điều phối thích hợp cho từng nhóm. Danh sách sẵn sàng được phân tách thành các danh sách riêng biệt theo cấp độ ưu tiên. Mỗi danh sách bao gồm các tiến trình có cùng mức độ ưu tiên và được áp dụng giải thuật điều phối thích hợp để điều phối.

Giải thuật sử dụng là giải thuật không độc quyền.

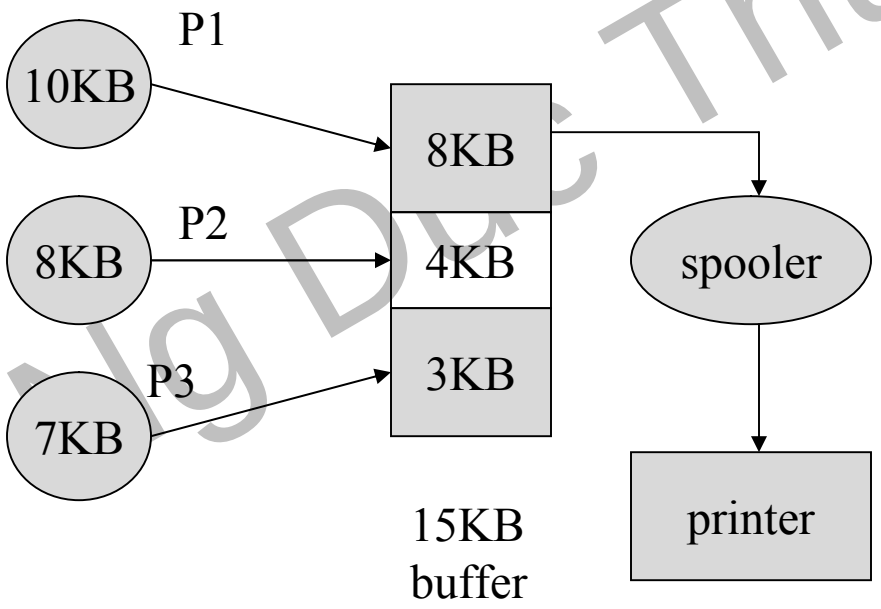
Chương 3: Quản lý tiến trình

Tiến trình deadlock : đợi một sự kiện không bao giờ xảy ra.

Một hệ thống bị deadlock : có tiến trình bị deadlock.



Tắc nghẽn trong giao thông



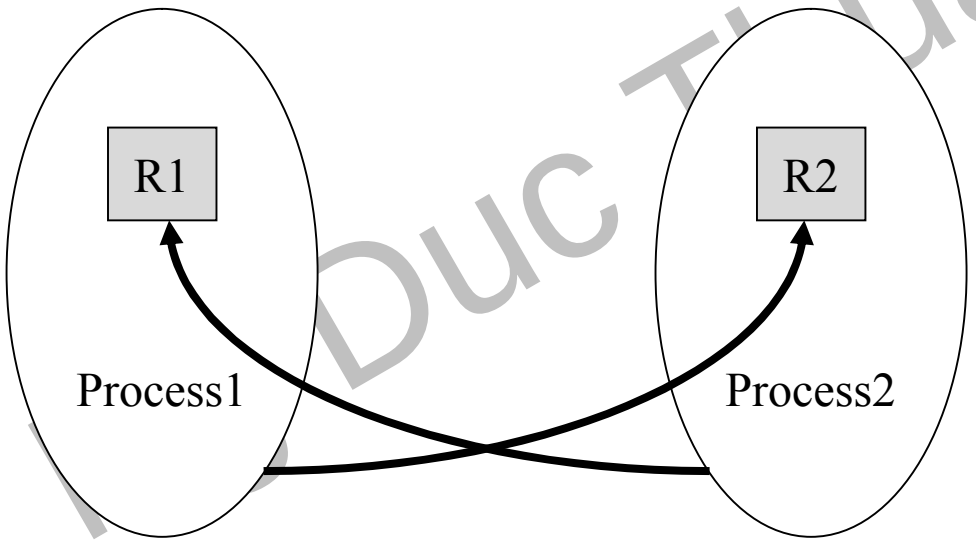
Tắc nghẽn trong quản lý in ấn

Chương 3: Quản lý tiến trình

Hai tiến trình bị deadlock:

Process 1	Process2
P(S1);	P(S2);
P(S2);	P(S1);
Critical Section;	*Critical Section;*
V(S1);	V(S2);
V(S2);	V(S1);

Dạng deadlock:



Chương 3: Quản lý tiến trình

ĐIỀU KIỆN XẢY RA DEADLOCK

Điều kiện **mutual exclusion**: các tiến trình cần thực hiện loại trừ tương hỗ trên vùng tranh chấp

Điều kiện **hold & wait**: tiến trình đang giữ tài nguyên có thể yêu cầu thêm tài nguyên khác

Điều kiện **no-preemption**: tài nguyên chỉ được giải phóng khi tiến trình dùng xong

Điều kiện **circular-wait**: các tiến trình giữ và đợi tài nguyên tạo thành vòng luẩn quẩn

Chương 3: Quản lý tiến trình

GIẢI QUYẾT DEADLOCK

Ngăn ngừa deadlock ***(deadlock prevention)***
- Qui định cấp , dùng tài nguyên nghiêm ngặt
- Không cho các điều kiện deadlock xảy ra

Tránh deadlock ***(deadlock avoidance)***
- Vẫn cho các điều kiện deadlock tồn tại
- Cấp tài nguyên hợp lý, an toàn

Phát hiện deadlock ***(deadlock detection)***

Phục hồi deadlock ***(deadlock recovery)***

Chương 3: Quản lý tiến trình

Cấm điều kiện **multual-exclusion** ?

Cấm điều kiện **hold & wait**

- Tiến trình yêu cầu tất cả tài nguyên một lần
- Chỉ được xử lý khi đã đủ tất cả tài nguyên cần thiết

Cấm điều kiện **no-preemption**

- Nếu yêu cầu tài nguyên không được, tiến trình phải giải phóng tất cả tài nguyên đang giữ và yêu cầu lại.

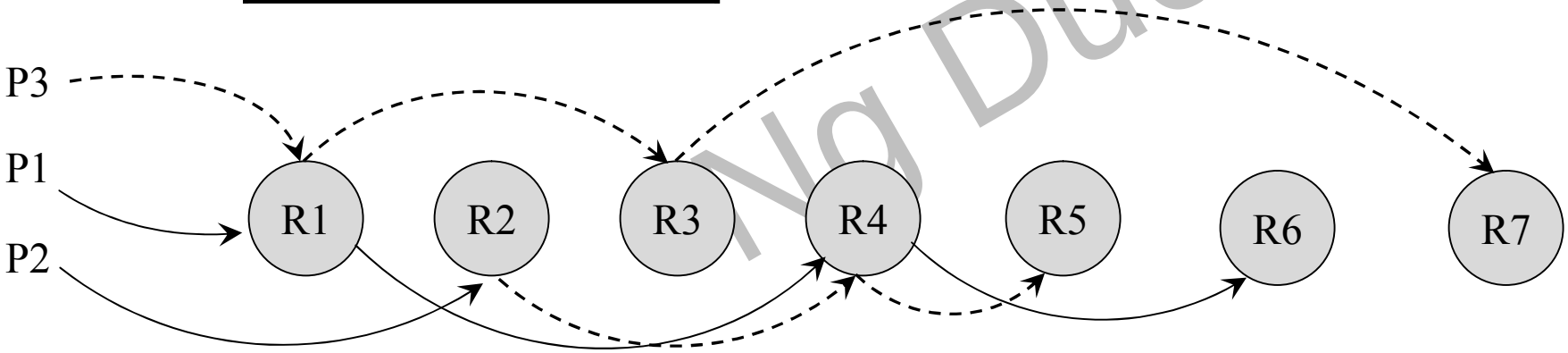
Loại bỏ **circular-wait**

- Sắp xếp tài nguyên theo trật tự và chung cấp cho tiến trình theo đúng trật tự đó

Chương 3: Quản lý tiến trình

Ví dụ

Tiến trình	Yêu cầu thực tế
P1	R6, R4, R1
P2	R2, R5, R4
P3	R3, R7, R1



Chương 3: Quản lý tiến trình

Giải thuật nhà băng (Banker's Algorithm)
- Hệ điều hành ~ Nhà Băng
- Tiến trình ~ Khách hàng
- Tài nguyên ~ Vốn vay

Ràng buộc
- Yêu cầu vay cực đại ≤ vốn nhà băng
- Khách không trả vốn nếu vay chưa đủ yêu cầu cực đại
- Khi vay đủ, khách phải trả đủ vốn sau thời gian hữu hạn

Trạng thái nhà băng
- An toàn (**Safe**): thỏa yêu cầu mọi khách, ngân hàng thu vốn đủ
- Không an toàn (**Unsafe**) : ngược lại ➔ có thể deadlock

Hệ thống phải cấp phát tài nguyên sao cho không rơi vào trạng thái Unsafe

Chương 3: Quản lý tiến trình

Trạng thái sau là an toàn. Tại sao ?

	Đang mượn	Cần tối đa	Cần thêm
P1	1	4	3
P2	4	6	2
P3	5	8	3
Vôn 12 , còn lại 2			

Trạng thái sau là không an toàn. Tại sao ?

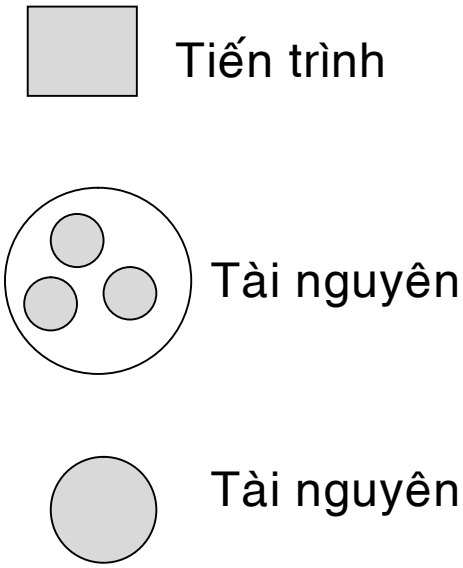
	Đang mượn	Cần tối đa	Cần thêm
P1	8	10	2
P2	2	5	3
P3	1	3	2
Vôn 12 , còn lại 1			

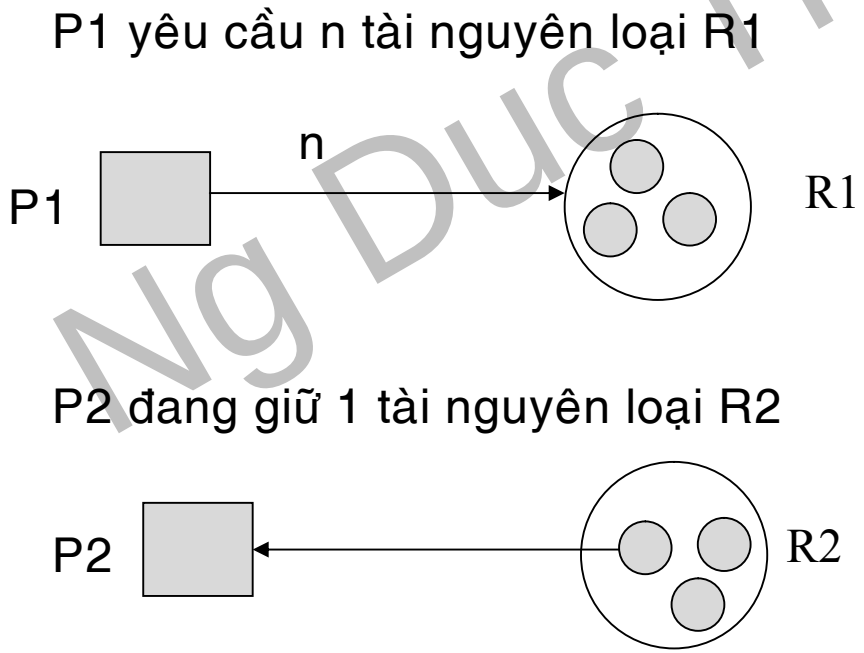
Chương 3: Quản lý tiến trình

PHÁT HIỆN DEADLOCK

Ghi nhận, theo dõi yêu cầu, sự cấp phát tài nguyên cho các tiến trình.

Dùng đồ thị RAG (*Resource Allocation Graph*)





Chương 3: Quản lý tiến trình

Giản lược RAG

1. Tài nguyên rảnh → cấp cho tiến trình yêu cầu
2. Tiến trình đủ tài nguyên
   → xoá mọi cạnh vào, xoá tiến trình
3. Lặp lại 1 với các tiến trình khác đến khi tối giản

Khi giải thuật dừng

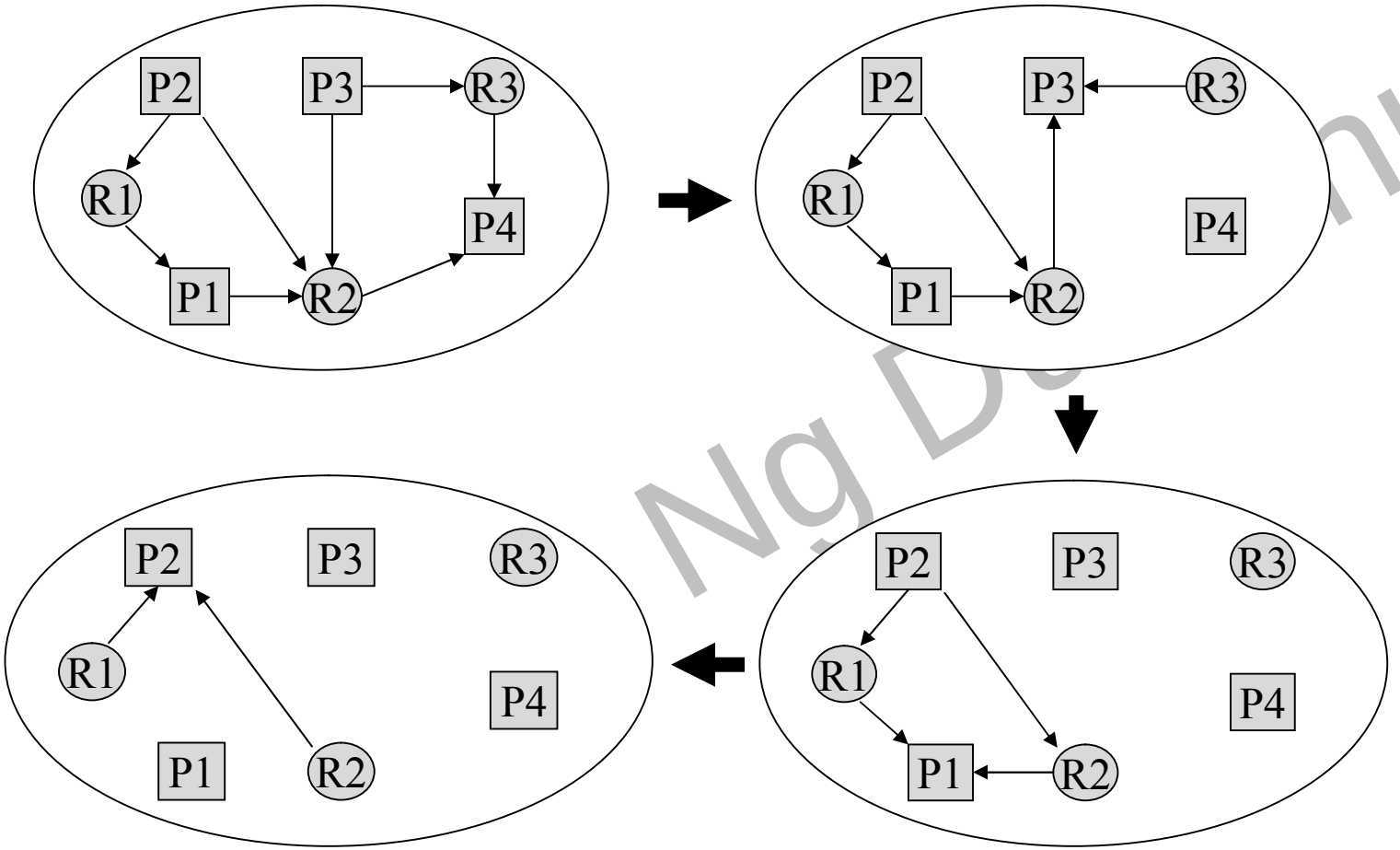
– RAG không còn cạnh: không có deadlock
– RAG có chu trình (cycle): deadlock

Nhận xét

– Phí tổn lớn

Chương 3: Quản lý tiến trình

VÍ DỤ 1 : GIẢN ƯỚC RAG



Chương 4: Hệ thống quản lý File

4.1 TỔNG QUAN VỀ DỮ LIỆU & FILE

Yêu cầu lưu trữ dữ liệu của người dùng:

- ✓ Lưu trữ lâu dài
- ✓ Truy cập nhanh
- ✓ Lưu được nhiều dữ liệu
- ✓ Chia sẻ và bảo vệ tốt
- ✓ Dễ sử dụng

Cần sự hỗ trợ của phần cứng và OS để dữ liệu lưu trữ ra bộ nhớ ngoài.

Tập tin và thư mục

- ✓ *Tập tin:* đơn vị lưu trữ thông tin do HĐH, các ứng dụng, người dùng quản lý.
- ✓ *Thư mục:* cấu trúc tổ chức tập tin trên đĩa.

Chương 4: Hệ thống quản lý File

4.2 MÔ HÌNH QLÝ VÀ TỔ CHỨC TẬP TIN

Tên tập tin:

- ✓ Tạo bởi HĐH, tiến trình hoặc user
- ✓ Cách đặt tên tập tin của mỗi HĐH là khác nhau: Số ký tự đặt tên (chữ cái, chữ số, ký tự đặc biệt), phân biệt/không phân biệt chữ hoa/thường.
- ✓ Nhiều hệ tạo tên file nhiều cụm, ngăn cách bởi dấu chấm '.'

Cấu trúc tập tin, gồm 3 loại:

- ✓ Dãy tuần tự các byte không cấu trúc.
- ✓ Dãy các record có chiều dài cố định.
- ✓ Cấu trúc cây (mỗi record có trường khóa giúp tìm kiếm nhanh).

Chương 4: Hệ thống quản lý File

4.2 MÔ HÌNH QLÝ VÀ TỔ CHỨC TẬP TIN

Kiểu tập tin

✓*Tập tin thường*: tập tin text/nhị phân chứa thông tin của người dùng.

✓*Thư mục:* (tập tin) lưu giữ cấu trúc hệ thống tập tin.

✓*Tập tin có ký tự đặc biệt*: liên quan đến nhập xuất (màn hình, máy in, mạng)

✓*Tập tin khối*: truy xuất thiết bị khối dạng đĩa

Các loại tập tin

Tập tin văn bản: tập các ký tự

Tập tin nhị phân: gồm dãy các byte tổ chức theo định dạng xác định (vd: header, text, data, relocation bits, symbol table)

Chương 4: Hệ thống quản lý File

4.2 MÔ HÌNH QLÝ VÀ TỔ CHỨC TẬP TIN

Truy xuất tập tin:

- *Truy xuất tuần tự:* thích hợp cho các thiết bị lưu trữ tuần tự (băng từ).
- *Truy xuất ngẫu nhiên:* thích hợp cho các thiết bị lưu trữ ngẫu nhiên (đĩa từ, CD-ROM...)

Thuộc tính tập tin: Bảo vệ, Mật khẩu, Người tạo, Người sở hữu, Chỉ đọc, Ẩn, Hệ thống, Lưu trữ, ASCII/Binary, Truy xuất ngẫu nhiên/Tuần tự, Temp, Khóa (lock), Độ dài record, Vị trí khóa, Ngày giờ tạo, Thời điểm truy cập, Thời điểm thay đổi, Kích thước hiện tại, Kích thước tối đa.

Chương 4: Hệ thống quản lý File

4.2 MÔ HÌNH QLÝ VÀ TỔ CHỨC TẬP TIN

Thư mục: hệ thống lưu trữ theo cấp bậc

- Thư mục chứa các đề mục tên + thuộc tính + địa chỉ tập tin.
- Khi có yêu cầu mở file, HĐH tìm trong thư mục tên file cần mở, xác định thuộc tính và địa chỉ file, đọc file vào bộ nhớ chính.
- Số lượng thư mục trên mỗi hệ thống là khác nhau. Để tiện cho việc truy xuất, sử dụng, lưu trữ, đa số các HĐH đều tổ chức thư mục theo cấu trúc phân cấp hình cây.

Đường dẫn: cách xác định địa chỉ tập tin

✓Thư mục hiện hành [.], thư mục cha [..]

✓Đường dẫn tuyệt đối: /usr/ast/mailbox

✓Đường dẫn tương đối: [.]/dir1/dir2

Chương 4: Hệ thống quản lý File

4.2 MÔ HÌNH QLÝ VÀ TỔ CHỨC TẬP TIN

Các chức năng của hệ thống tập tin

- _*Tập tin*: Tạo, Xóa, Mở, Đọc, Ghi, Thêm, Đóng, Tìm, Lấy thuộc tính, Thiết lập thuộc tính, Đổi tên.
- _*Thư mục*: Tạo, Xóa, Mở thư mục (vd: mở trước khi liệt kê), Đóng thư mục, Đổi tên, Liên kết (file có thể xuất hiện trong nhiều thư mục khác nhau), Bỏ liên kết.

Chương 4: Hệ thống quản lý File

CÁC PHƯƠNG PHÁP ĐỊNH VỊ BẢNG PHÂN PHỐI ĐĨA

Định vị liên tiếp:

- Dễ cài đặt, dễ thao tác
- Không linh động, đĩa bị phân mảnh

Định vị bằng danh sách liên kết

- Entry chỉ chứa địa chỉ đầu. Mọi khối đều được cấp phát, ít lãng phí.
- Khối dữ liệu bị thu hẹp. Truy xuất ngẫu nhiên chậm

Danh sách liên kết sử dụng index

- Tương tự như cách 2 nhưng thay con trỏ bằng bảng index
- Truy xuất ngẫu nhiên dễ dàng hơn
- Bị giới hạn bởi kích thước bộ nhớ

Chương 4: Hệ thống quản lý File

QUẢN LÝ ĐĨA

Có 2 phương pháp lưu trữ:

✓Lưu tuần tự trên N byte liên tiếp: không hiệu quả khi file lớn

✓Lưu dữ liệu trên đĩa theo đơn vị khối

Kích thước khối thông thường là 512byte, 1 hoặc 2 KB

Lưu giữ các khối trống. Có 2 phương pháp

✓Sử dụng danh sách liên kết của khối đĩa

✓Sử dụng bitmap: đĩa N khối ánh xạ thành N bit (1:trống, 0: đã dùng). Đĩa 20M cần 20Kbit để lưu trữ = 2.5 ~3 khối

Chương 4: Hệ thống quản lý File

Độ an toàn của hệ thống tập tin

Quản lý khối bị hỏng:

✓Giải pháp phần mềm: xây dựng tập tin chứa các khối bị hỏng

✓Giải pháp phần cứng: dùng sector trên đĩa lưu giữ danh sách khối hỏng

Backup:

✓Chép dự phòng bản sao thứ cấp dữ liệu (đĩa mềm, băng từ...)

✓Chia đĩa cứng làm 2 phần: dữ liệu hoạt động và backup.

Tính không đổi của hệ thống tập tin

✓Ngừng hệ thống đột ngột có thể gây mất dữ liệu

✓Hệ thống phải có cơ chế kiểm tra tính toàn vẹn dữ liệu trên 2 phần khối và tập tin

Chương 4: Hệ thống quản lý File

Truy xuất hệ thống tập tin theo MSDOS

Nhập xuất theo thẻ file

Thẻ file (Handle file): là đối tượng mà MSDOS dùng để quản lý các file đang được mở trên bộ nhớ. Khi file được mở DOS sẽ gán cho file một số xác định. (AX, BX)

- ✓Tạo: 3C
- ✓Mở: 3D
- ✓Đóng: 3E
- ✓Hủy: 41
- ✓Đặt con trỏ: 42
- ✓Đọc từ tập tin/thiết bị: 3F
- ✓Ghi lên tập tin/thiết bị: 40
- ✓IOCTL: 44
- ✓Định hướng lại: 46

Chương 4: Hệ thống quản lý File

Truy xuất hệ thống tập tin theo MSDOS

Các chức năng về tập tin và thư mục

- ✓ Tạo thư mục: 39
- ✓ Hủy thư mục: 3A
- ✓ Chuyển thư mục: 3B
- ✓ Đổi tên tập tin: 56
- ✓ Thiết lập ngày giờ: 57
- ✓ Hỏi kích thước file: 42
- ✓ Thay đổi thuộc tính: 43
- ✓ Tìm tập tin: 4E
- ✓ Tìm thư mục: 11

Chương 4: Hệ thống quản lý File

Truy xuất hệ thống tập tin theo MSDOS

Các chức năng thi hành:

- ✓ Thi hành: 4B
- ✓ Tạp PSP: 26
- ✓ Lấy địa chỉ PSP: 62
- ✓ Kết thúc thường trú: Int 27
- ✓ Kết thúc chương trình: Int 20
- ✓ Kết thúc gửi mã thoát về tiến trình cha: 4C
- ✓ Lấy mã thoát của tiến trình kết thúc: 4D
- ✓ Kết thúc qua Ctr-Brk: Int 23

Chương 4: Hệ thống quản lý File

Truy xuất hệ thống tập tin theo MSDOS

Mở tập tin mới

- ✓ AH=3Ch
- ✓ CL= (0: đọc; 1 ghi; 2 đọc/ghi)
- ✓ DS:DX: là địa chỉ xâu tên file (kết thúc 0).
- ✓ => AX chứa thẻ file

Đóng tập tin

- ✓ AH=3Eh
- ✓ BX= thẻ file

Đọc nội dung tập tin

- ✓ AH=3Fh CX= số byte cần đọc
- ✓ BX= thẻ file DS:DX: vùng đệm lưu

Ghi nội dung tập tin

- ✓ AH=40h CX= số byte cần ghi
- ✓ BX= thẻ file DS:DX: vùng đệm chứa dliệu

Chương 4: Hệ thống quản lý File

RAID (*Redundant Array of Inexpensive Disks*)

Tập hợp các đĩa cứng được hệ điều hành xem như một thiết bị lưu trữ

Dữ liệu được phân bố trên tất cả các đĩa

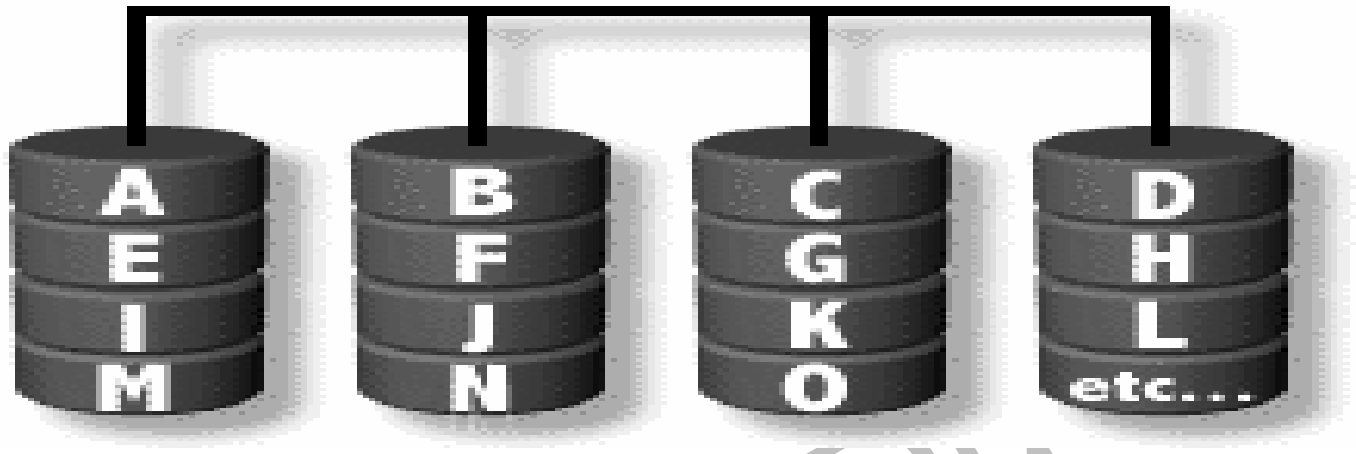
Các mục tiêu chính

- ❖Tăng dung lượng lưu trữ
- ❖Tăng hiệu suất I/O
- ❖Tăng tính sẵn sàng cao
- ❖Tăng khả năng phục hồi hệ thống

Các loại RAID

- ❖RAID 0 → RAID 10 (phổ biến RAID 0, 1, 3, 5)
- ❖Software RAID/ Hardware RAID

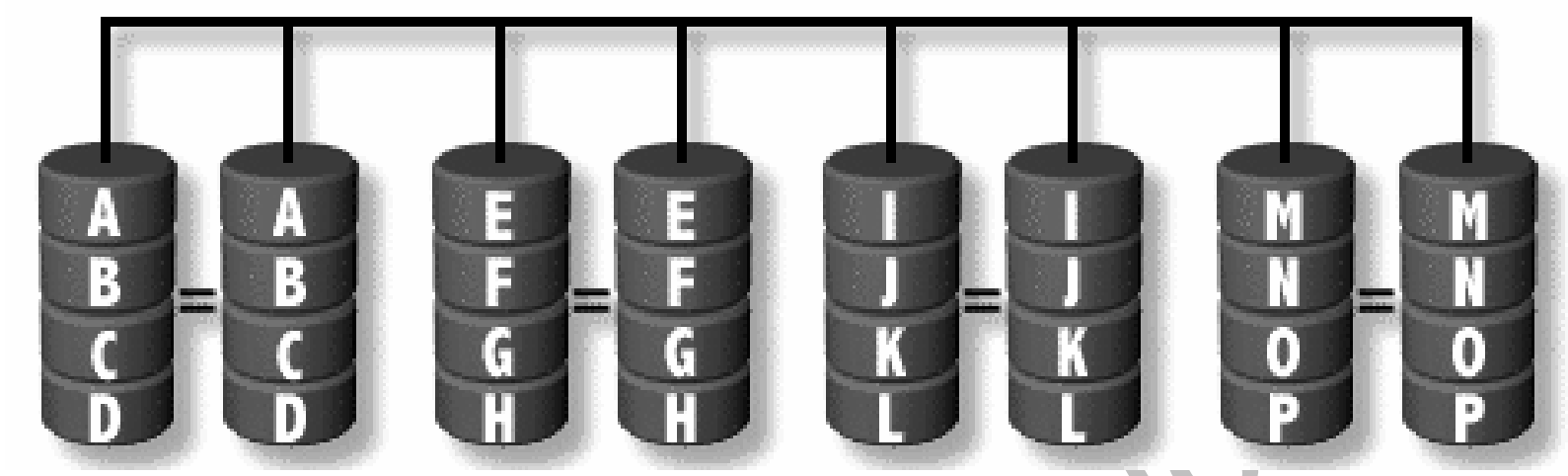
Chương 4: Hệ thống quản lý File



RAID-0

- ✓ Dữ liệu lưu trữ trải đều trên các đĩa
- ✓ Tăng không gian lưu trữ
- ✓ Tăng hiệu suất hệ thống
- ✓ Tính sẵn sàng của dữ liệu thấp

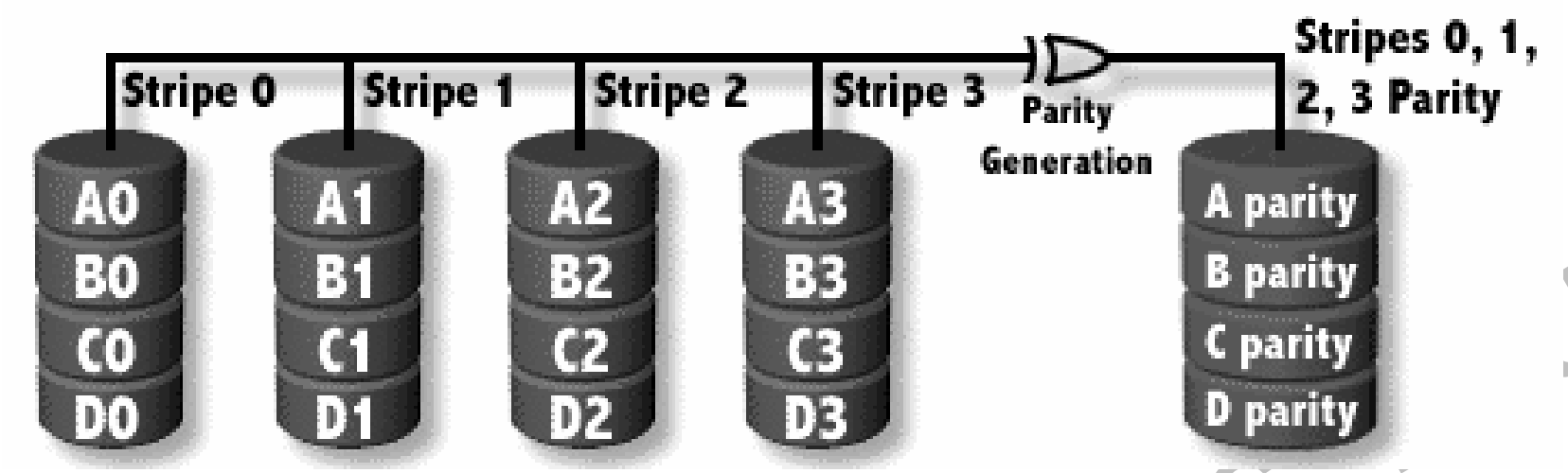
Chương 4: Hệ thống quản lý File



RAID-1

- ✓ Nhân bản dữ liệu trên các đĩa tách biệt
- ✓ Tính sẵn sàng & tốc độ đọc dữ liệu rất cao
- ✓ Yêu cầu dung lượng đĩa gấp đôi
- ✓ Tốc độ ghi chậm hơn

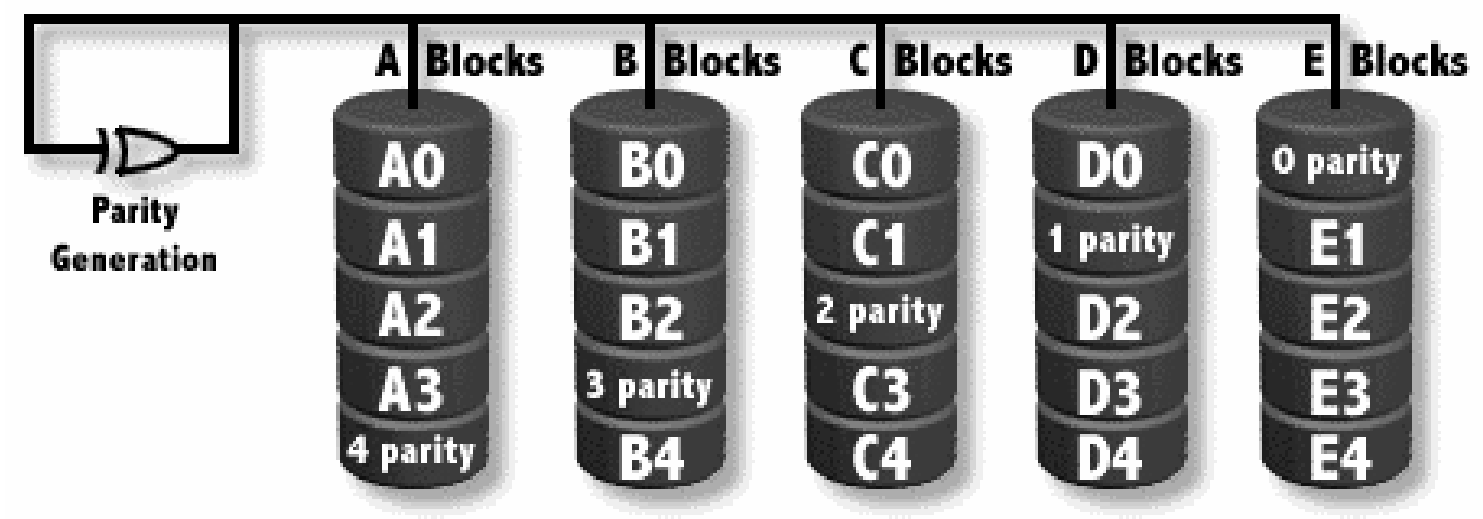
Chương 4: Hệ thống quản lý File



RAID-3

- ✓ Lưu dữ liệu trải đều trên các đĩa
- ✓ Sử dụng một đĩa lưu thông tin kiểm tra dữ liệu
- ✓ Tính sẵn sàng cao, chi phí hợp lý
- ✓ Hiệu suất I/O thấp

Chương 4: Hệ thống quản lý File



RAID-5

- ✓ Dữ liệu, thông tin kiểm tra được lưu trải đều trên các đĩa
- ✓ Tính sẵn sàng dữ liệu trung bình, chi phí hợp lý
- ✓ Tốc độ ghi thấp
- ✓ Yêu cầu phần cứng đặc biệt

Chương 5: Quản lý bộ nhớ

5.1 Đặt vấn đề

HĐH ngày nay cho phép chạy chế độ đa nhiệm=> Nhu cầu chia sẻ bộ nhớ giữa các tiến trình khác nhau.

- Hệ điều hành có nhiệm vụ cấp pháp bộ nhớ cho các tiến trình khi có yêu cầu.
- Để thực hiện tốt nhiệm vụ này, HĐH xem xét bộ nhớ dựa trên nhiều khía cạnh:
  - Sự tương tác giữa địa chỉ logic và vật lý.
  - Quản lý bộ nhớ vật lý.
  - Chia sẻ thông tin giữa các tiến trình qua bộ nhớ.
  - Bảo mọi sự truy xuất bất hợp pháp.

Chương 5: Quản lý bộ nhớ

5.1 Đặt vấn đề

Quá trình ánh xạ địa chỉ tượng trưng của 1 chương trình nguồn vào bộ nhớ chính vào 3 thời điểm:

- Thời điểm biên dịch
- Thời điểm nạp
- Thời điểm xử lý

➡Không gian địa chỉ và không gian vật lý

✓ Địa chỉ logic- địa chỉ ảo do bộ xử lý tạo ra.

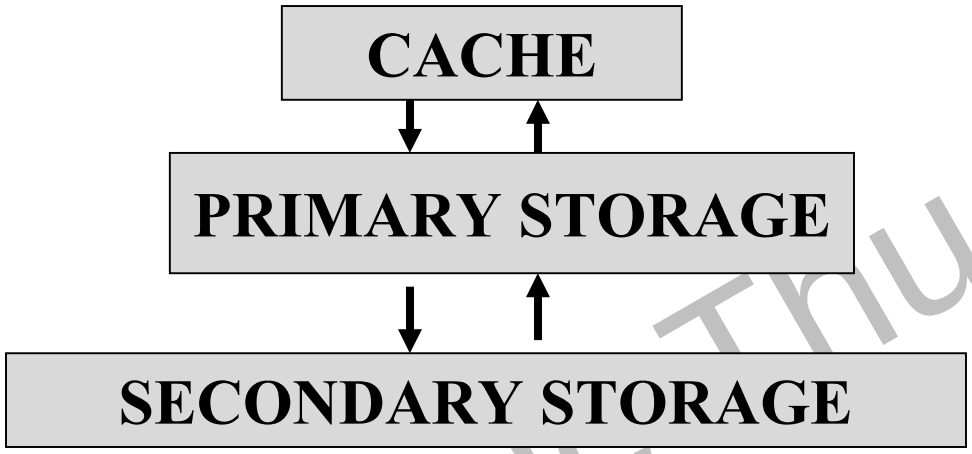
✓ Địa chỉ vật lý- địa chỉ thực.

✓ Không gian địa chỉ - là tập hợp địa chỉ ảo phát sinh bởi 1 chương trình.

✓ Không gian vật lý – là tập hợp địa chỉ vật lý tương ứng đại chỉ ảo.

Chương 5: Quản lý bộ nhớ
PHÂN CẤP BỘ NHỚ

- **Từ trên xuống**
  - ✓ Tốc độ giảm
  - ✓ Dung lượng tăng
  - ✓ Giá thành giảm



- **Các vấn đề quan tâm**

✓ Bộ nhớ chính chứa 1 hay nhiều tiến trình ?

✓ Các quy trình dùng vùng nhớ như nhau / khác nhau ?

✓ Bảo vệ vùng nhớ của OS và của từng tiến trình ?

✓ Vùng nhớ của quy trình là liên tục / gián đoạn ?

Chương 5: Quản lý bộ nhớ

CHIẾN LƯỢC QUẢN LÝ BỘ NHỚ

Chiến lược nạp (*fetch strategies*)

- Nạp phần nào của tiến trình vào bộ nhớ và khi nào nạp ?
- Nạp theo yêu cầu & nạp tiên đoán

Chiến lược sắp đặt (*placement strategies*)

Nạp tiến trình mới vào đâu ?

Chiến lược thay thế (*replacement strategies*)

Đưa tiến trình nào ra bộ nhớ phụ ?

Chương 5: Quản lý bộ nhớ

TỔ CHỨC CẤP PHÁT BỘ NHỚ

Cấp phát bộ nhớ liên tục

- Đơn lập trình
- Đa lập trình phân đoạn cố định
- Đa lập trình phân đoạn thay đổi
- Đa lập trình có thay thế vùng nhớ

Cấp phát bộ nhớ không liên tục

Chương 5: Quản lý bộ nhớ

5.2 Cấp phát liên tục

Các hệ đơn chương trình

Ý tưởng: Bộ nhớ chỉ chia sẻ cho hệ điều hành và một chương trình duy nhất người sử dụng. Một phần bộ nhớ do HĐH chiếm giữ phần còn lại thuộc về tiến trình người dùng.

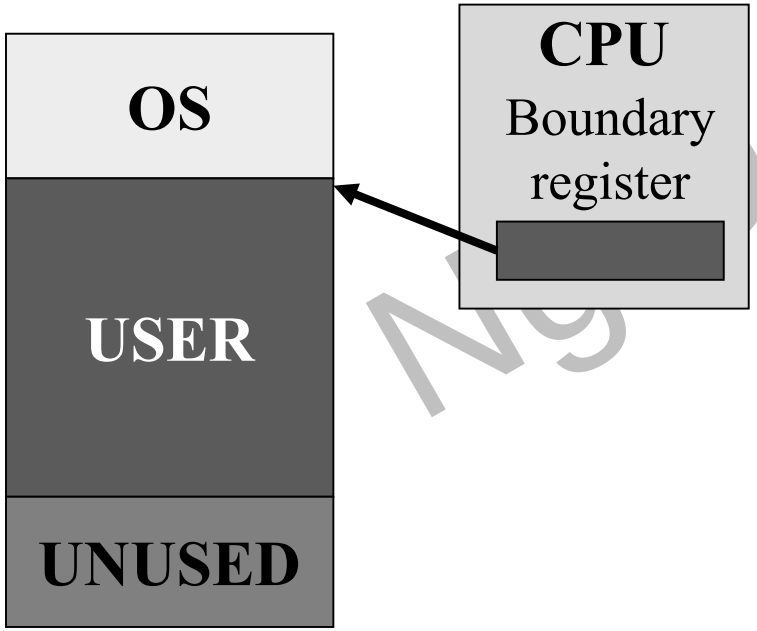
Thảo luận:

☞ Cần bảo vệ vùng nhớ khỏi sự xâm phạm tiến trình người dùng.(sử dụng thanh ghi giới hạn)

☞Tại một thời điểm chỉ có thể đáp ứng một tiến trình.

Chương 5: Quản lý bộ nhớ

✓ Mỗi lần tiến trình người dùng truy xuất cần kiểm tra với nội dung thanh ghi giới hạn =>Tốc độ truy xuất không cao.
✓Sử dụng CPU không hiệu quả.

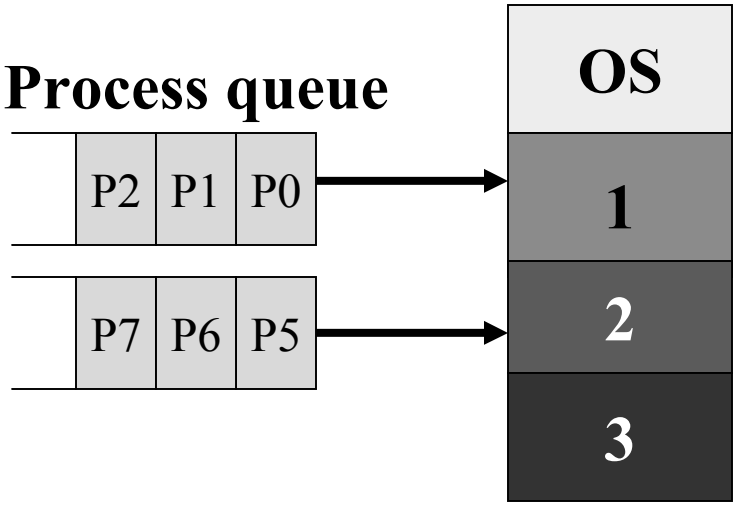


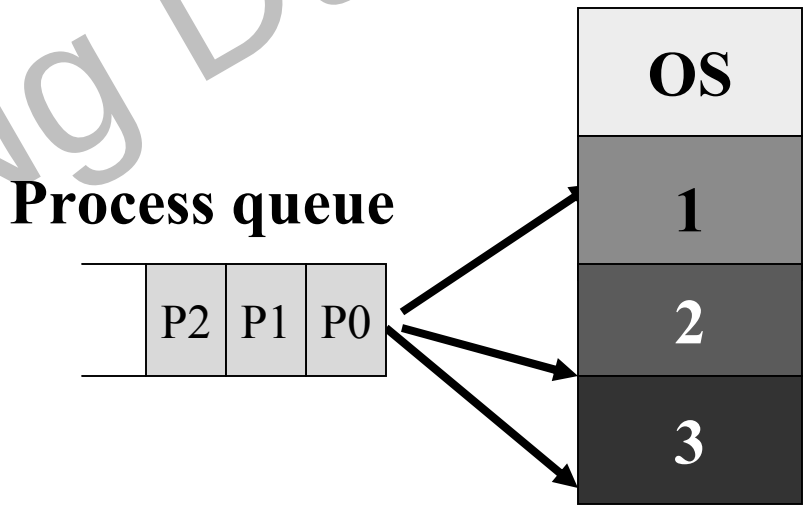
Chương 5: Quản lý bộ nhớ

2. Hệ thống đa chương trình với phân vùng cố định.

Ý tưởng: Bộ nhớ được chia thành n phân vùng có kích thước cố định (các phân vùng có kích thước khác nhau). Tiến trình có yêu cầu bộ nhớ được lưu trữ trong hàng đợi. Hàng đợi được tổ chức:

- Sử dụng mỗi phân vùng một hàng đợi.
- Sử dụng một hàng đợi duy nhất.

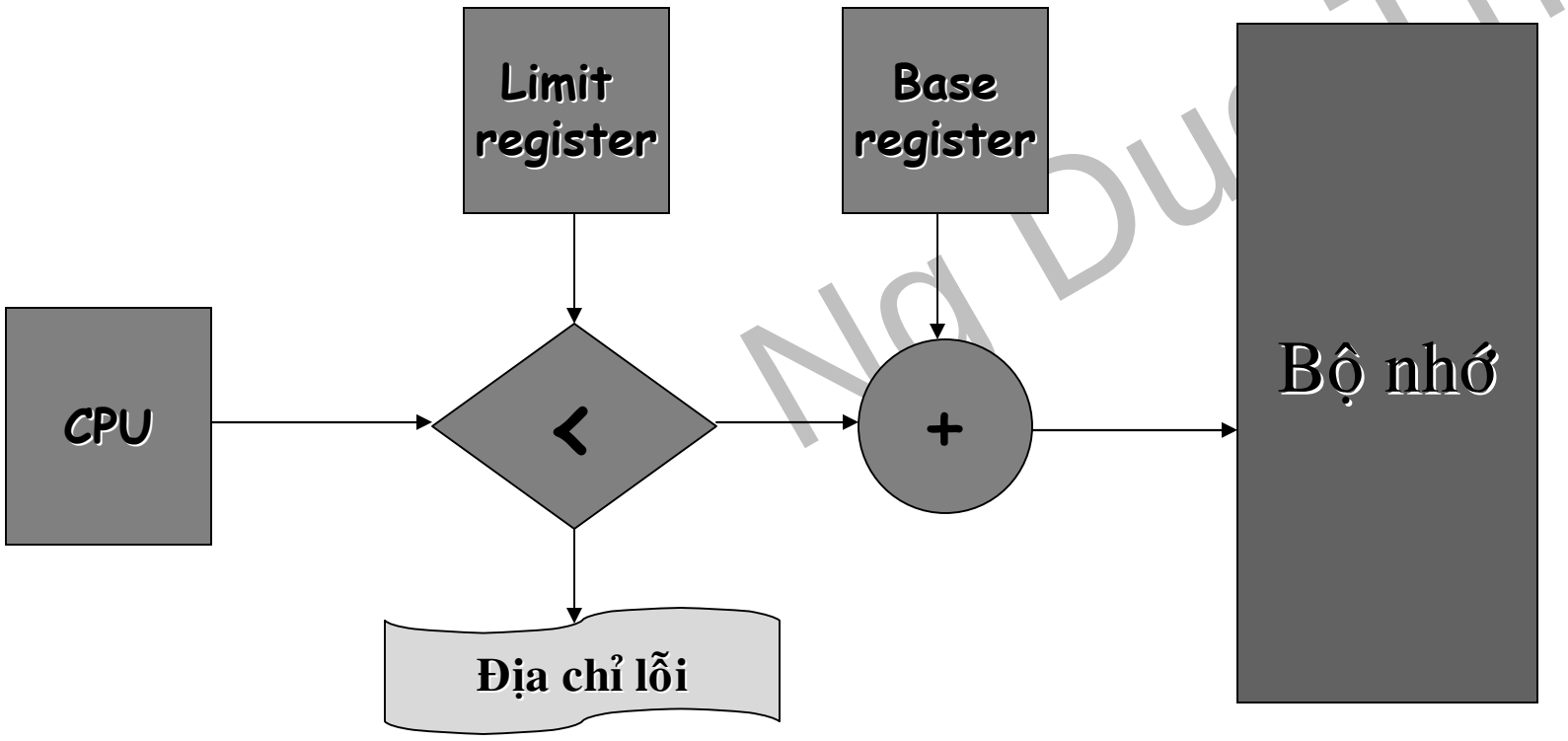




Chương 5: Quản lý bộ nhớ

Thảo luận:

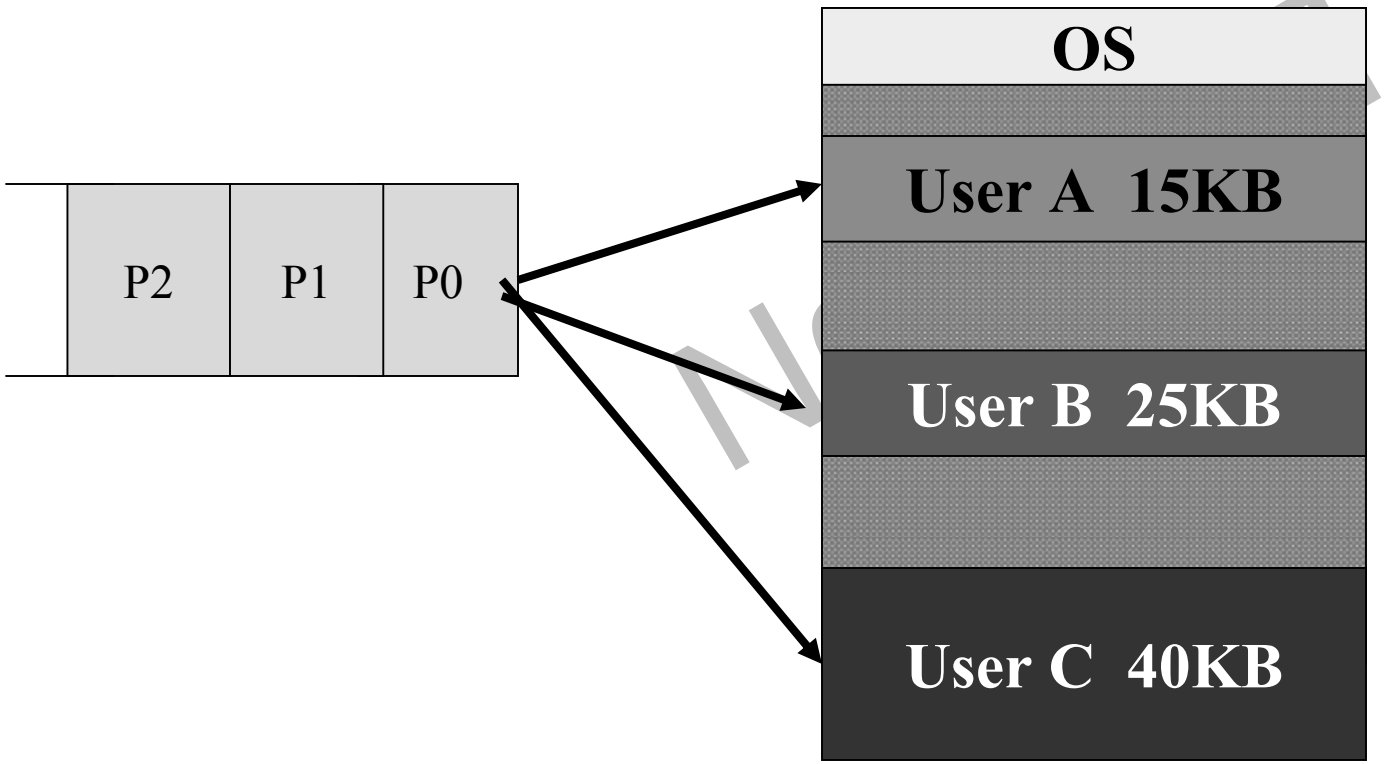
- ✓ Kích thước tiến trình khác nhau => phân mảnh nội.
- ✓ Mức độ đa chương của hệ thống bị giới hạn bởi phân vùng.



Chương 5: Quản lý bộ nhớ

3. Hệ thống đa chương trình với phân vùng động.

Ý tưởng: Tiến trình được đưa vào hệ thống, cấp phát một vùng nhớ vừa đúng kích thước tiến trình. Phần còn lại cấp cho tiến trình khác.



Chương 5: Quản lý bộ nhớ

Thảo luận:

➢Không có hiện tượng phân mảnh nội vi, nhưng xuất hiện phân mảnh ngoại vi.

➢Khi kích thước tiến trình tăng trưởng trong quá trình xử lý mà không còn vùng nhớ trống kề nhau đủ rộng.

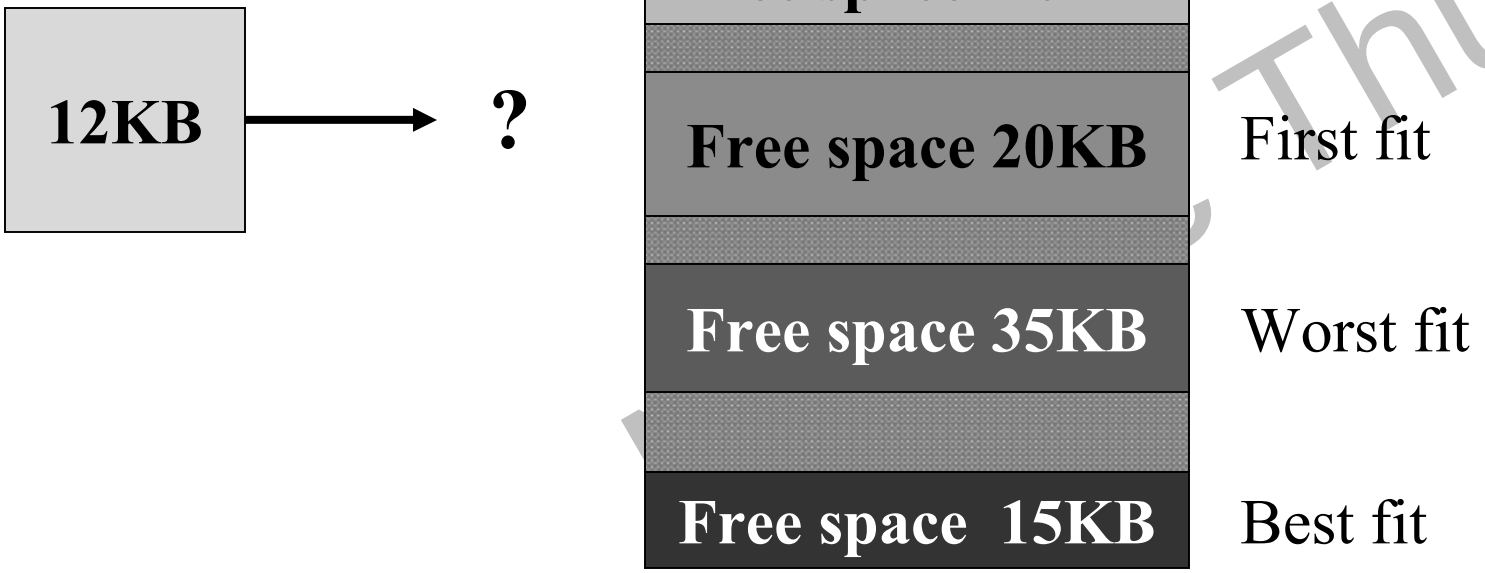
➢ Ghi lại hiện trạng bộ nhớ để cấp phát để cấp phát động cho đúng.

➢ Có 2 phương pháp:

- Quản lý bằng một bảng các bit.
- Quản lý bằng danh sách(First fit, Best fit, Worst fit)

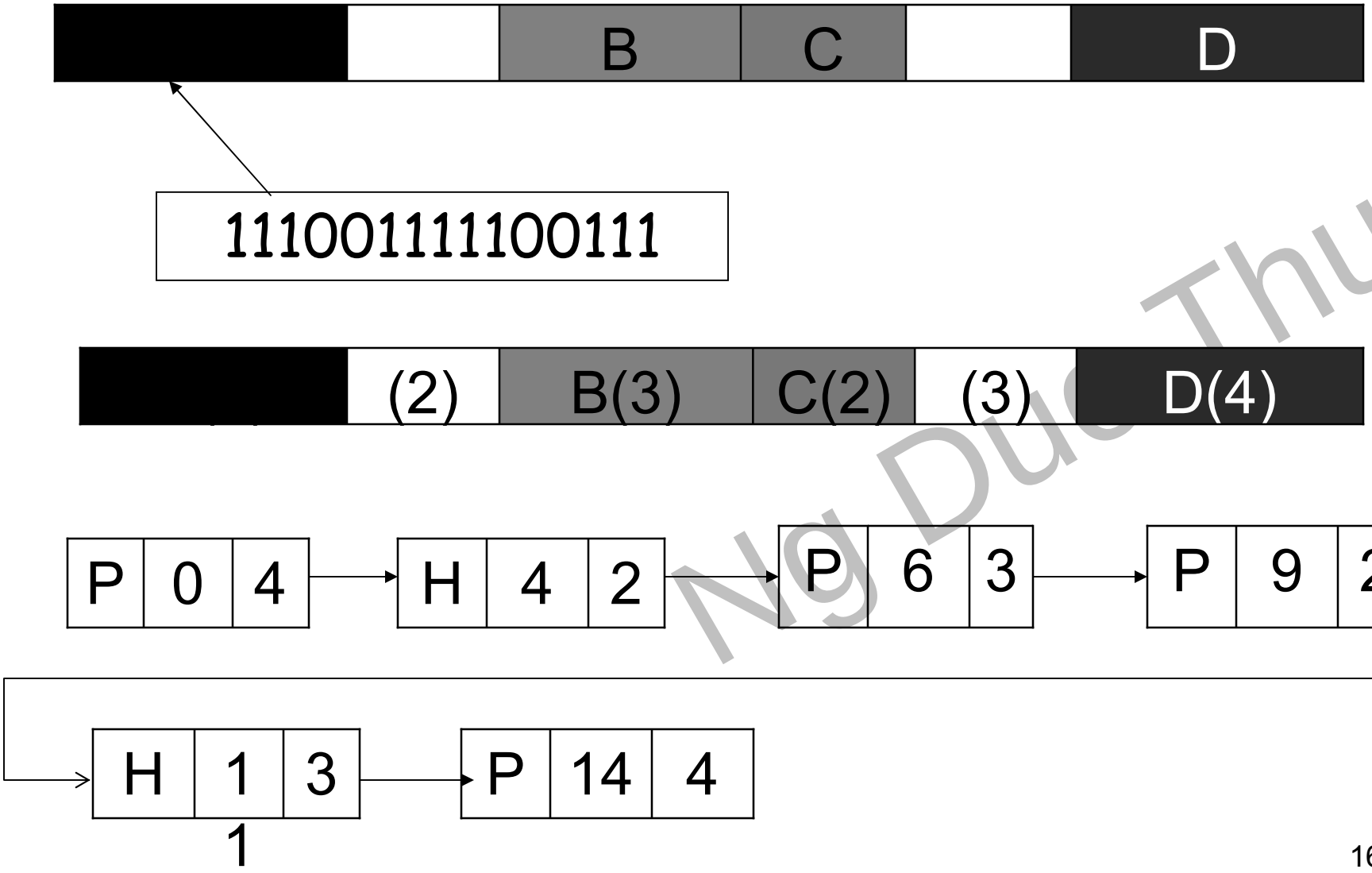
HỆ THỐNG ĐA CHƯƠNG PHÂN ĐOẠN THAY ĐỔI

- Chiến lược sắp đặt



- Vấn đề phân mảnh vùng nhớ (*fragmentation*)

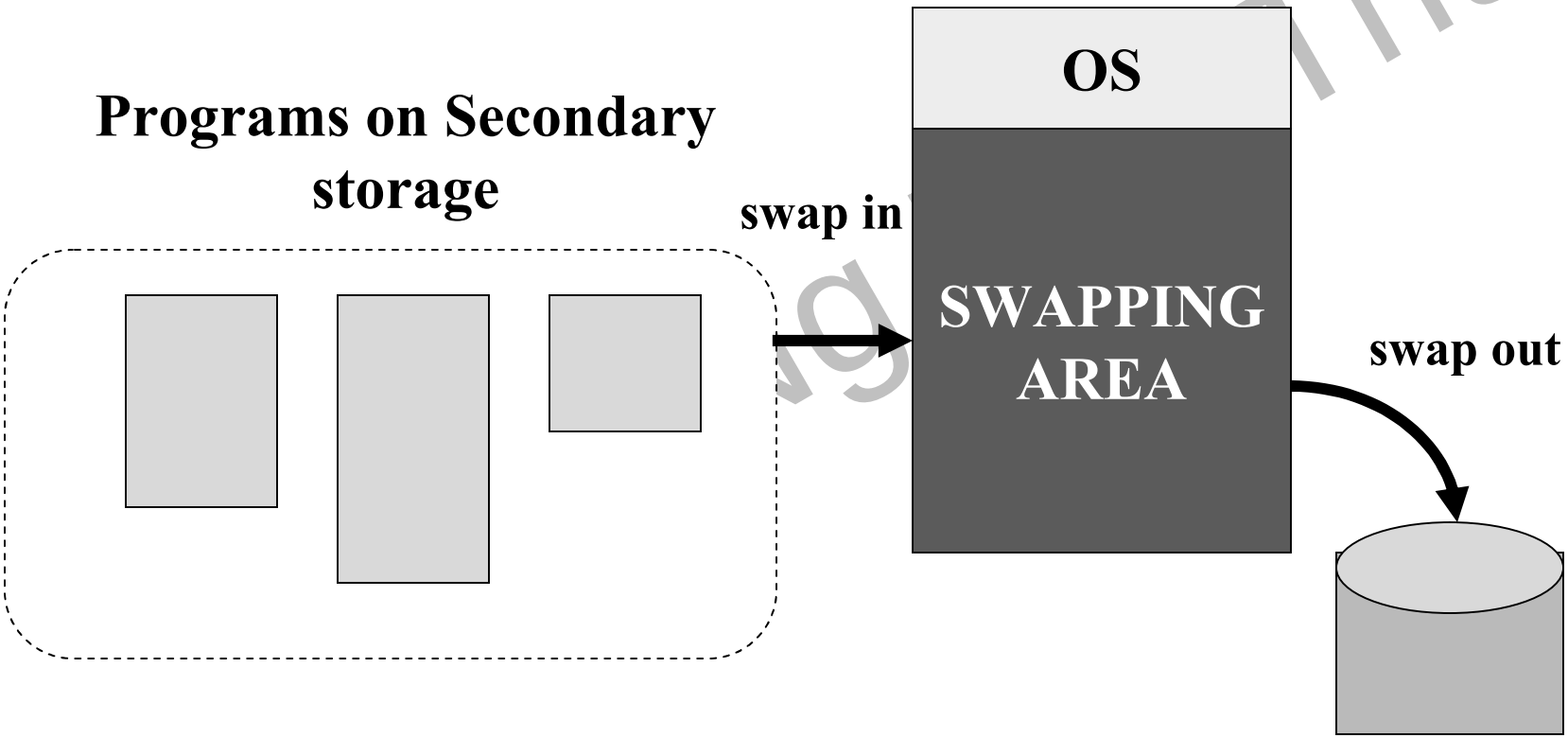
Chương 5: Quản lý bộ nhớ



Chương 5: Quản lý bộ nhớ

4. Các hệ thống đa chương với kỹ thuật "SWAPPING"

Ý tưởng: Một tiến trình chờ tương đối dài tạm thời chuyển ra bộ nhớ phụ(swap out). Khi kết thúc việc chờ tiến trình trở lại bộ nhớ chính để xử lý (swap in.)



Chương 5: Quản lý bộ nhớ

5.3 Cấp phát không liên tục

1. Phân trang(paging)

Ý tưởng:

1. Phân bộ nhớ thành các khối có kích thước cố định bằng nhau.

- ✓ Một tiến trình kích thước n trang sẽ yêu cầu n khung trang tự do.
- ✓ Cơ chế MMU(Manager Memory Unit) trong kỹ thuật phân trang.
- ✓ Hỗ trợ phần cứng thực hiện chuyển đổi địa chỉ trong cơ chế phân trang là bảng trang(pages table). Mỗi phần tử trong bảng cho biết các đại chỉ bắt đầu của vị trí lưu trữ trang tương ứng trong bộ nhớ vật lý.

Chương 5: Quản lý bộ nhớ

5.3 Cấp phát không liên tục

Chuyển đổi địa chỉ:

Mỗi địa chỉ phát bởi CPU bao gồm 2 thành phần:

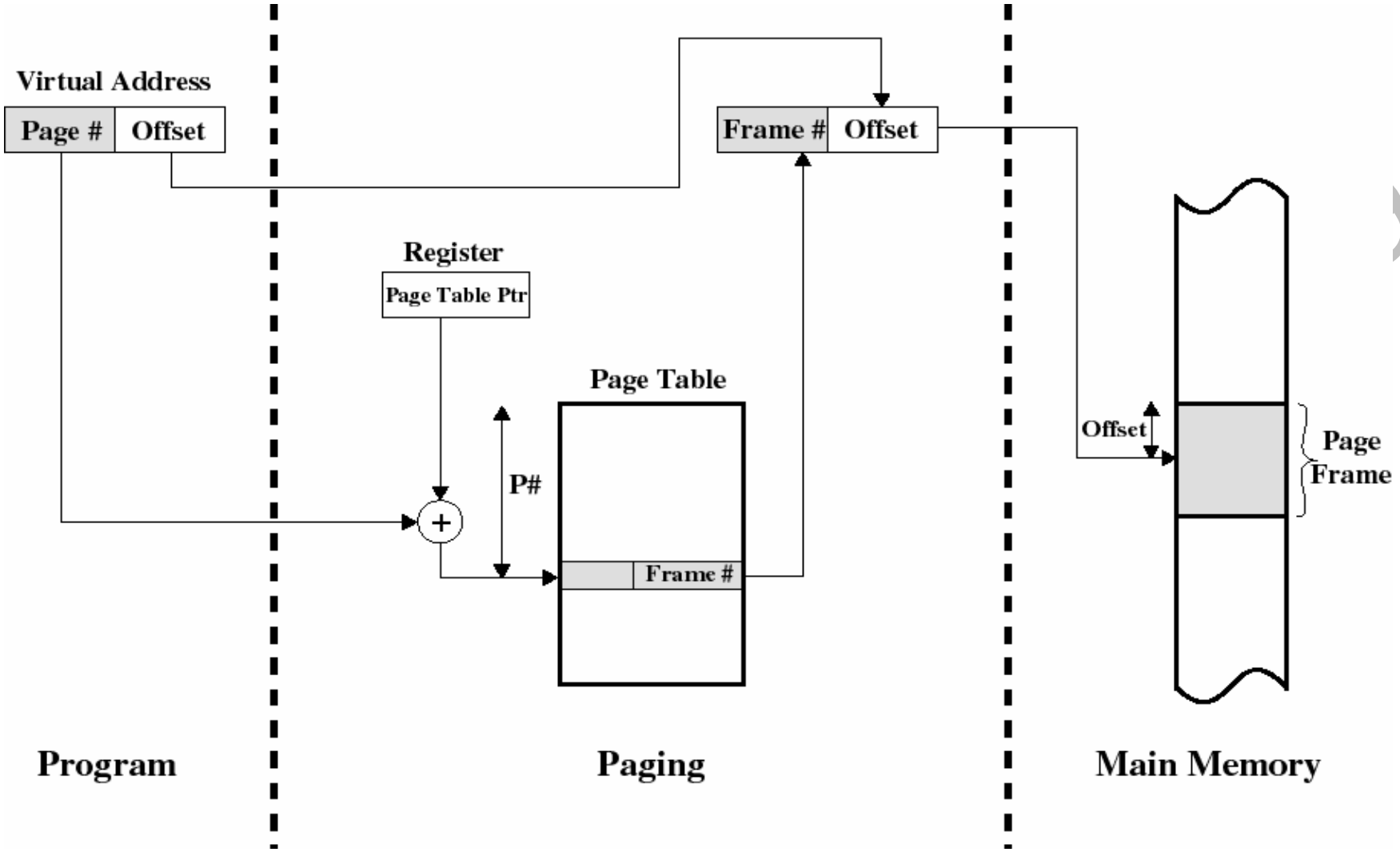
+ Số hiệu trang (*p*)

+ Địa chỉ tương đối trong trang (*d*)

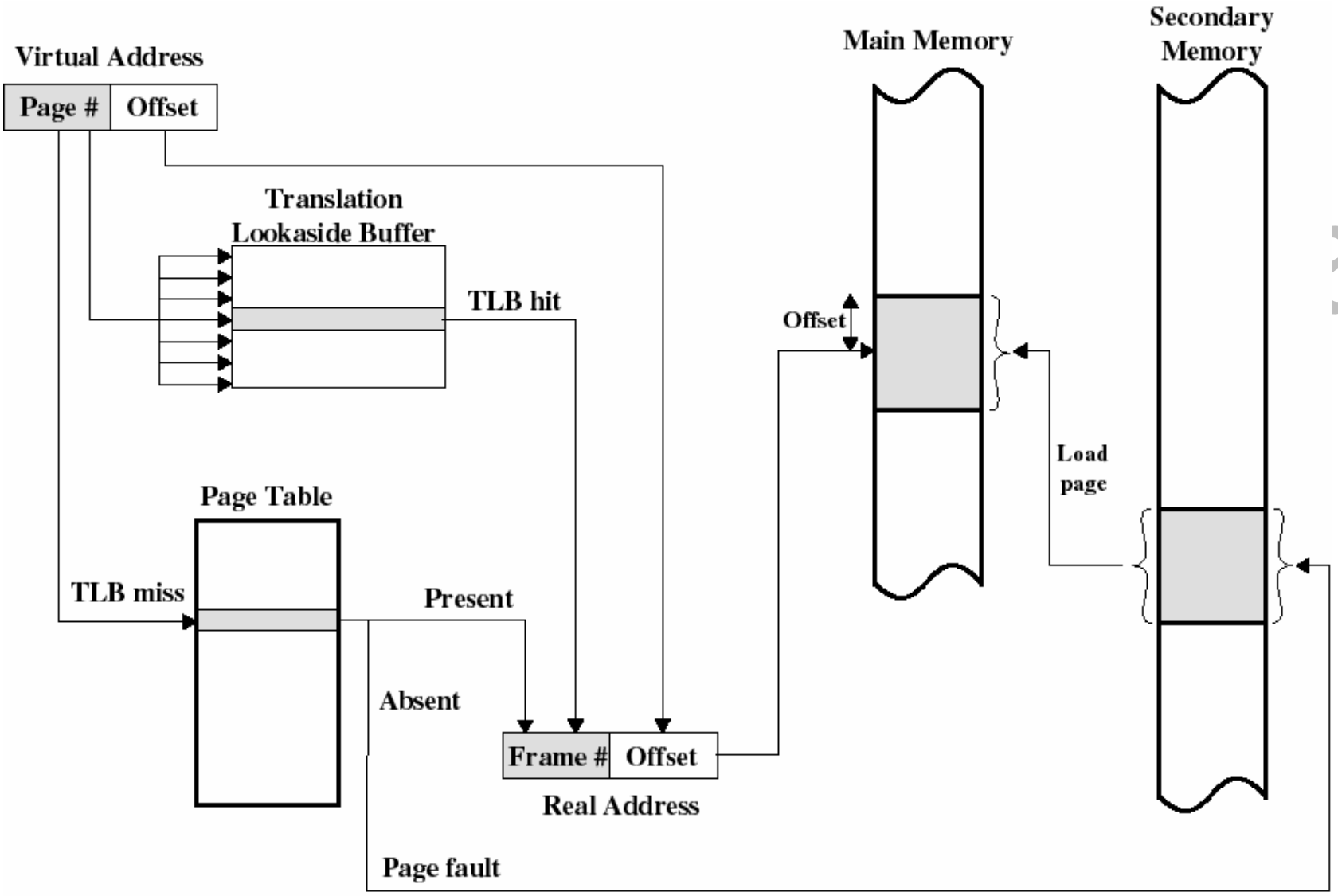
Kích thước mỗi trang do phần cứng quy định thường lũy thừa của 2 nằm trong miền trị (512 ->8192).

➢Nếu kích thước của không gian địa chỉ là 2^m và kích thước trang 2^n, thì m-n bits cao của địa chỉ ảo sẽ biểu diễn số trang, và n bits thấp cho biết địa chỉ tương đối trong trang.

ÁNH XẠ ĐỊA CHỈ TRỰC TIẾP TRONG HỆ THỐNG PHÂN TRANG



ÁNH XẠ TRANG DÙNG BỘ NHỚ KẾT HỢP



Thảo luận

- Kỹ thuật phân trang loại bỏ hiện tượng phân mảnh ngoại vi.
- Vẫn còn xuất hiện tượng phân mảnh nội vi khi kích thước tiến trình không là bội số kích thước của trang.
- Một tiến trình người dùng coi bộ nhớ phân trang như không gian liên tục, đồng nhất và chỉ chứa duy nhất một tiến trình.
- Phần cứng nhiệm vụ đổi địa chỉ logic thành địa chỉ vật lý.
- Để lưu trữ thông tin chi tiết quá trình cấp phát bộ nhớ, HĐH sử dụng một bảng khung trang.

Chương 5: Quản lý bộ nhớ

KHÁI NIỆM BỘ NHỚ ẢO

Là hình ảnh của bộ nhớ thực

- Địa chỉ ảo V: tham khảo bởi process
- Địa chỉ thực R : có trong bộ nhớ thực

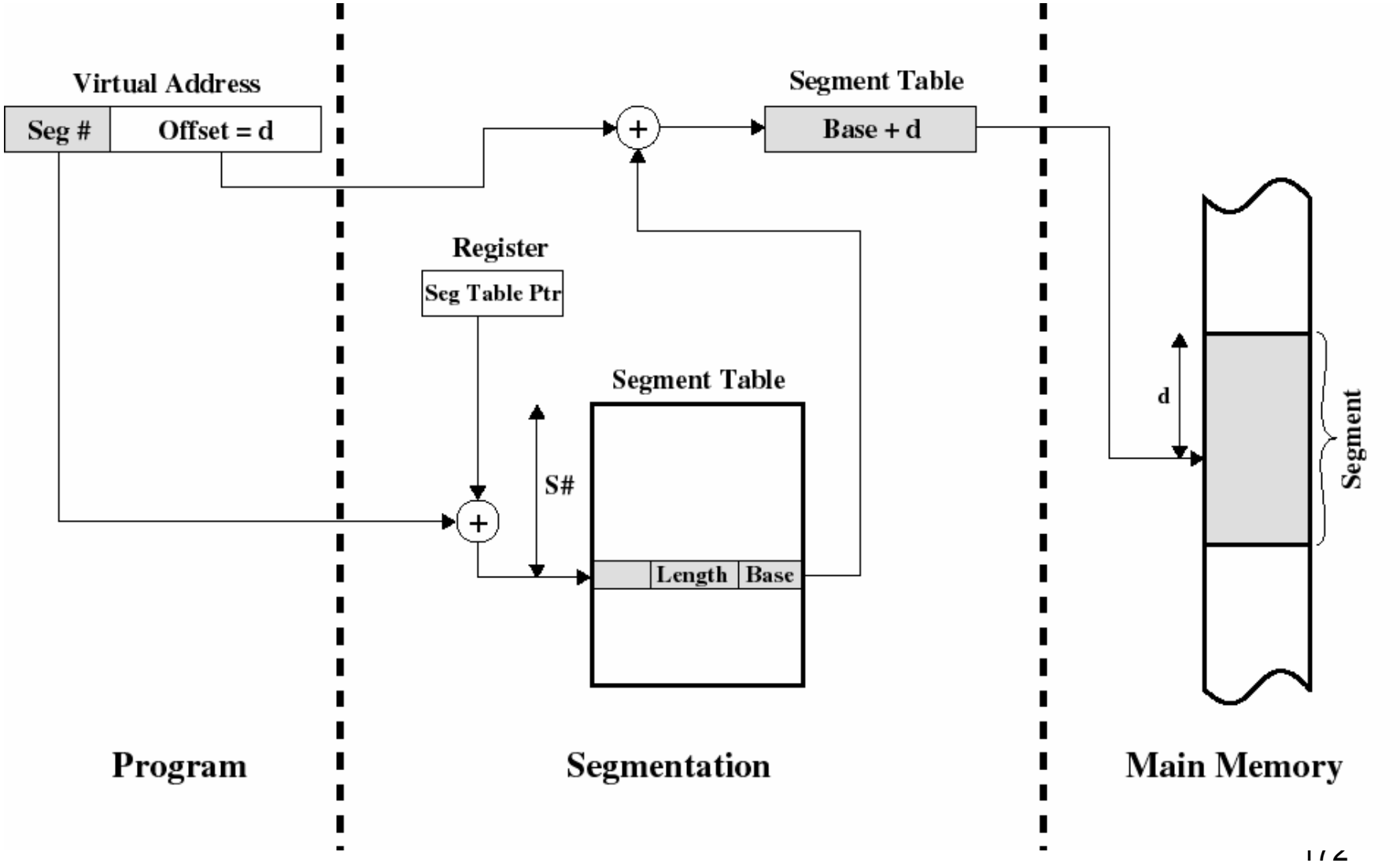
$$|V| >> |R|$$

Địa chỉ ảo được ánh xạ thành địa chỉ thực mỗi khi quá trình thực thi → dynamic address translation

Sự cần thiết của bộ nhớ ảo

- Dễ phát triển ứng dụng
- Lưu trữ được nhiều quá trình trong bộ nhớ
- Tái định vị (relocation) các quá trình
- Cho các quá trình chia sẻ vùng nhớ dễ dàng

ÁNH XẠ ĐỊA CHỈ TRONG HỆ THỐNG PHÂN ĐOẠN



Chương 5: Quản lý bộ nhớ

PHỐI HỢP PHÂN TRANG & PHÂN ĐOẠN

Địa chỉ ảo V=(s, p, d)

- s: chỉ số đoạn (segment #)
- p : chỉ số trang trong đoạn (page #)
- d : độ dời của ô nhớ trong trang (displacement)

Địa chỉ thực R=(p′, d′)

- p′ : chỉ số trang thực (frame #)
- d′ : độ dời của ô nhớ trong trang thực

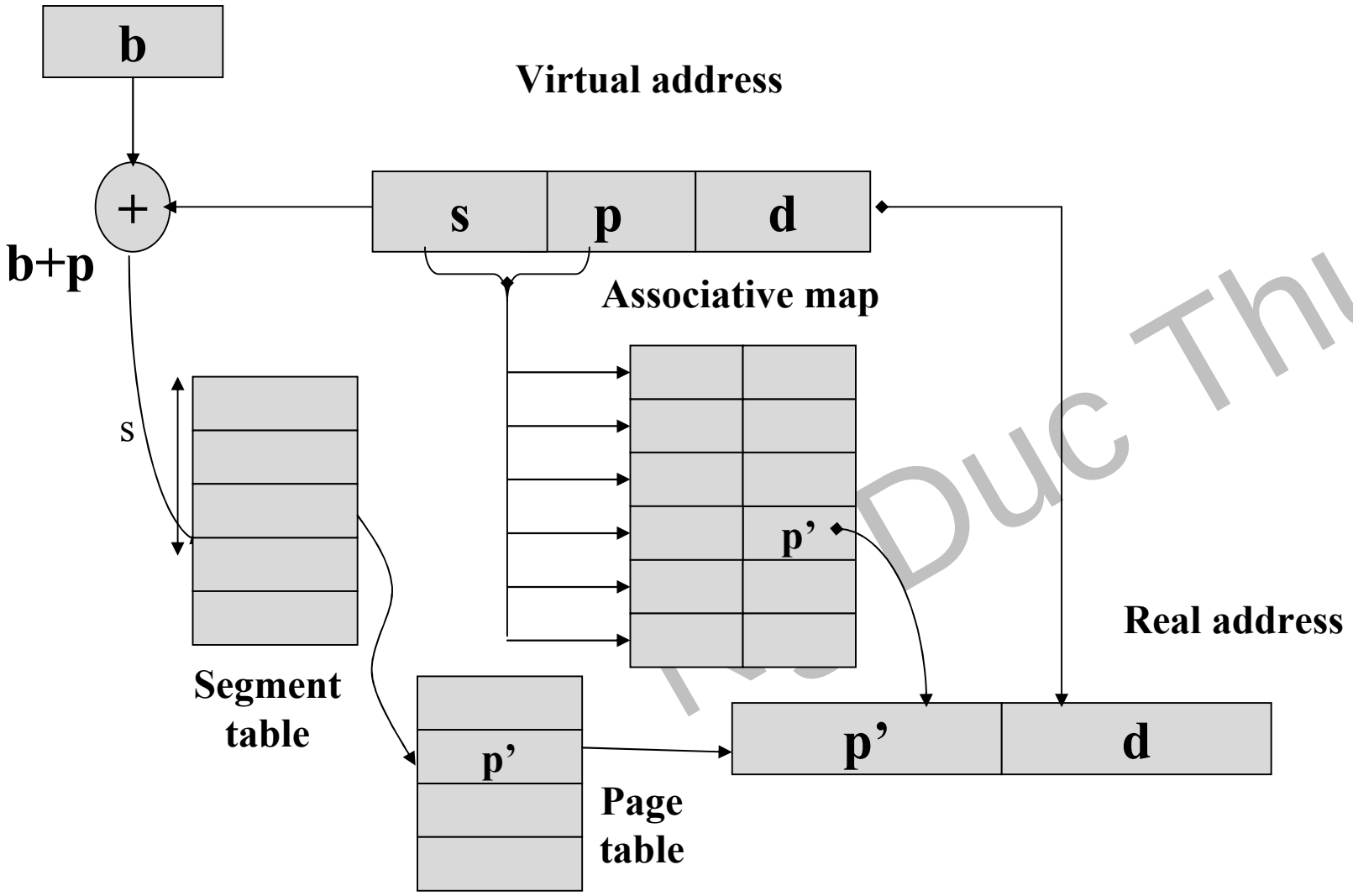
Ánh xạ địa chỉ

(s, p) → Associate memory → p′

Hoặc s→s′ (s′, p) → p′

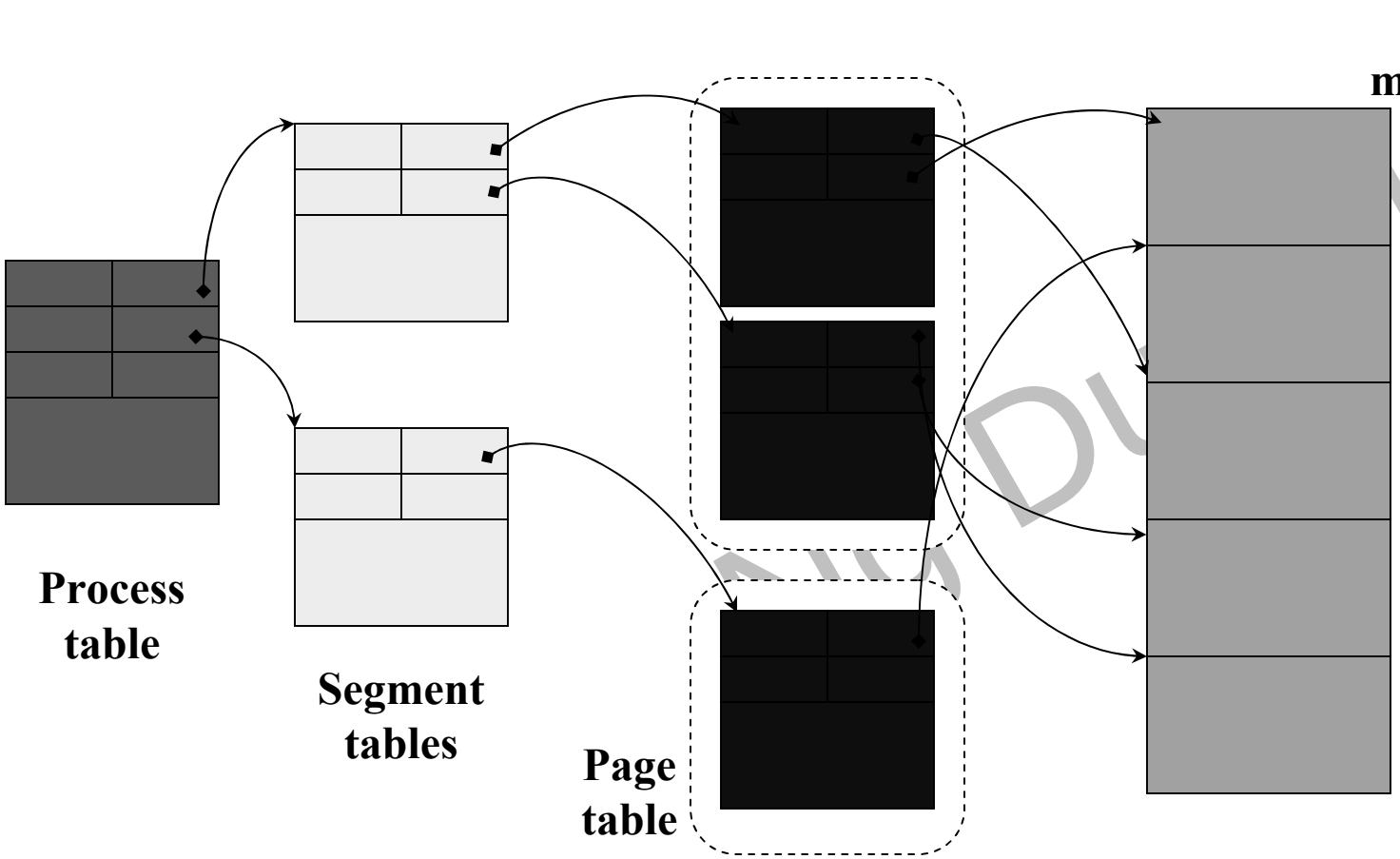
(s′ : địa chỉ đầu bảng ánh xạ trang với mỗi đoạn)

ÁNH XẠ ĐỊA CHỈ TRONG HỆ THỐNG PHÂN ĐOẠN KẾT HỢP PHÂN TRANG



CẤU TRÚC ÁNH XẠ BỘ NHỚ

(trong hệ thống phân đoạn kết hợp phân trang)



Chương 5: Quản lý bộ nhớ

CÁC CHIẾN LƯỢC QUẢN LÝ BỘ NHỚ ẢO

Các chiến lược quản lý

- **Chiến lược nạp (*Fetch strategies*)**
- **Chiến lược sắp đặt (*Placement strategies*)**
- **Chiến lược thay thế(*Replacement strategies*)**

Chiến lược nạp

- Nạp trang theo yêu cầu (*Demand paging*)
- Nạp trang tiên đoán (*Anticipatory paging*)
- Page fault và các bước xử lý page fault

Chiến lược sắp đặt

Chiến lược thay thế

CÁC GIẢI THUẬT THAY THẾ TRANG

Yêu cầu : Tối thiểu số page fault

Nguyên tắc tối ưu : Chọn trang thay thế là:

- Trang không còn dùng nữa
- Trang sẽ không dùng lại trong thời gian xa nhất

Các tiêu chuẩn (thực tế) để chọn trang thay thế

- Các trang không bị thay đổi
- Các trang không bị khóa
- Các trang không thuộc quá trình nhiều page fault
- Các trang không thuộc tập làm việc của quá trình

Một số giải thuật thay thế trang

- Thay thế trang ngẫu nhiên
- FIFO, LRU, giải thuật xấp xỉ LRU, LFU, NUR

GIẢI THUẬT TỐI ƯU (OPT)

Chọn trang thay thế là trang sẽ không được tham khảo trong thời gian lâu nhất.

Ví dụ:

	1	2	3	4	1	2	5	1	2	3	4	5
Thời điểm t	0	1	2	3	4	5	6	7	8	9	10	11
Bộ nhớ thực có 3 frame	1	1	1	1	1	1	1	1	1	1	1	1
		2	2	2	2	2	2	2	2	3	4	4
			3	4	4	4	5	5	5	5	5	5

7 page fault

GIẢI THUẬT FIFO

Chọn trang thay thế là trang ở trong bộ nhớ thực trong khoảng thời gian lâu nhất.

Nghịch lý Belady

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

Bộ nhớ thực có 3 frame

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

9 page fault

Bộ nhớ thực có 4 frame

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

10 page fault

GIẢI THUẬT LRU (*Least Recently Used*)

Chọn trang thay thế là trang đã không được tham khảo trong thời gian lâu nhất.

Thời điểm t	0	1	2	3	4	5	6	7	8	9	10	11
											0	1
Chuỗi tham khảo	1	2	3	4	1	2	5	1	2	3	4	5
Bộ nhớ thực có 3 frame	**1**	1	1	**4**	4	4	**5**	5	5	**3**	3	**5**
		2	2	2	**1**	1	1	1	1	1	**4**	4
			3	3	3	**2**	2	2	2	2	2	2

GIẢI THUẬT NUR (Not Used Recently)

Là giải thuật xấp xỉ LRU

Dùng thêm 2 bit cho mỗi trang

- Referenced bit R
- Modified bit M (còn gọi là dirty bit)

Trang sẽ thuộc 1 trong 4 nhóm, thay thế trang sẽ theo độ ưu tiên của nhóm trang

R	M	Ý nghĩa đối với trang nhớ
0	0	Chưa tham chiếu, chưa sửa đổi
0	1	Chưa tham chiếu, đã sửa đổi ?
1	0	Đã tham chiếu, chưa sửa đổi
1	1	Đã tham chiếu, đã sửa đổi

Thứ tự ưu tiên thay thế trang giảm dần

Chương 6: Quản lý xuất /nhập

- Các khái niệm cơ bản
  - HĐH phải quản lý tất cả các thiết bị nhập xuất, ra chỉ thị cho các thiết bị, kiểm soát các ngắt và lỗi
  - HĐH phải cung cấp giao tiếp đơn giản, tiện dụng giữa các thiết bị và hệ thống. Giao tiếp phải độc lập với thiết bị
  - Lập trình viên nhìn các thiết bị IO dưới góc độ phần mềm
  - Thiết bị logic: chìa khóa của vấn đề độc lập thiết bị
  - Thiết bị logic được tổ chức thành 4 lớp: kiểm sóat lỗi, điều khiển thiết bị, phần mềm HĐH độc lập thiết bị, phần mềm mức người sử dụng

Chương 6: Quản lý xuất /nhập

- Hệ thống quản lý nhập xuất
  - Tổ chức theo từng lớp. Mỗi lớp có chức năng nhất định
  - Các lớp giao tiếp với nhau theo sơ đồ:
    - Xử lý của user: tạo lời gọi nhập xuất, định dạng nhập xuất
    - Phần mềm ĐLTB: đặt tên, bảo vệ, tổ chức khối, bộ đệm định vị
    - Điều khiển thiết bị: thiết lập thanh ghi TB, kiểm tra trạng thái
    - Kiểm soát ngắt: báo cho driver khi nhập xuất hòan tất
    - Phần cứng: thực hiện thao tác nhập xuất

Chương 6: Quản lý xuất /nhập

- Mô hình tổ chức và quản lý nhập xuất
  - Mô hình
    - Thiết bị nhập xuất
    - Thiết bị logic
  - Các chức năng
    - Thiết bị nhập xuất
    - Thiết bị logic

Chương 6: Quản lý xuất /nhập

- Mô hình thiết bị nhập xuất. Các loại thiết bị IO:
  - Thiết bị khối:
    - Thông tin lưu trữ trong các khối có địa chỉ xác định.
    - Kích thước khối cố định. Thông thường từ 128-1024 byte
    - Dữ liệu truy xuất theo từng khối riêng biệt. VD: đĩa
  - Thiết bị tuần tự:
    - Dữ liệu lưu trên chuỗi các bits tuần tự, không có địa chỉ xác định.
    - Không seek được (VD: màn hình, bàn phím, máy in, card mạng, chuột…)
  - Các thiết bị khác, VD: bộ nhớ màn hình, đồng hồ…

Chương 6: Quản lý xuất /nhập

- Điều khiển thiết bị
  - Thiết bị IO có 2 phần: phần cơ là bản thân thiết bị và phần điện tử là mạch (card) điều khiển thiết bị
  - Mỗi bộ phận điều khiển có thể gắn kết /quản lý nhiều loại thiết bị khác nhau
  - Nhà sản xuất thiết bị (và bộ điều khiển) phải tuân theo chuẩn giao tiếp. VD: ANSI, IEEE, ISO…
  - Giao tiếp giữa thiết bị và bộ điều khiển là giao tiếp mức thấp
- DMA (Direct Memory Access)
  - DMA: kênh truyền dữ liệu trực tiếp từ bộ nhớ đến thiết bị IO
  - Đa số các loại thiết bị (đặc biệt dạng khối) đều hỗ trợ DMA

Chương 6: Quản lý xuất /nhập

- Thiết bị logic. Kiểm soát ngắt
  - Ngắt là tình huống phức tạp, cần được che khuất trong HĐH
  - Ngắt chỉ được tạo sau khi các tiến trình nhập xuất hoàn tất
  - Khóa tiến trình bằng lệnh WAIT hoặc RECEIVE thông điệp
- Device Drivers
  - Tất cả các đoạn mã độc lập đều chuyển đến device driver
  - Mỗi device driver kiểm soát từng loại/tập hợp thiết bị
  - Device driver phát chỉ thị và kiểm tra thực hiện chính xác

Chương 6: Quản lý xuất /nhập

- Phần mềm nhập xuất độc lập thiết bị
  - Độc lập về mặt thiết bị
  - Độc lập về mặt hệ thống
- Phần mềm nhập xuất ở mức người sử dụng
  - Một phần các phần mềm nhập xuất chứa các thư viện liên kết với chương trình của người sử dụng
  - Lời gọi hệ thống nhập xuất được các hàm thư viện thực hiện

Chương 6: Quản lý xuất /nhập

- Các chức năng thiết bị nhập xuất. Điều khiển thiết bị:
  - Chức năng của bộ điều khiển là giao tiếp với HĐH (qua bus)
  - Bộ điều khiển chuyển đổi dãy các bit tuần tự trong một khối các byte trong buffer của bộ điều khiển, hiệu chỉnh và chuyển dữ liệu vào bộ nhớ chính
  - Mỗi bộ điều khiển có các thanh ghi để liên lạc với CPU. Các thanh ghi này được ánh xạ thành một phần của bộ nhớ chính tại địa chỉ xác định của từng loại thiết bị
  - HĐH thực hiện nhập xuất bằng cách ghi lệnh lên thanh ghi bộ điều khiển. CPU rời bộ điều khiển để thực hiện công việc khác. Khi thực hiện xong, bộ điều khiển tạo ngắt gọi CPU đến lấy kết quả trong thanh ghi

Chương 6: Quản lý xuất /nhập

- Các chức năng thiết bị nhập xuất. DMA
  - DMA: truy nhập bộ nhớ trực tiếp, giúp CPU không lãng phí thời gian
  - CPU gửi cho bộ điều khiển các thông số như địa chỉ trên đĩa (nguồn), địa chỉ trong bộ nhớ (đích), số lượng byte dữ liệu
  - Sau khi đọc tòan bộ dữ liệu từ thiết bị vào buffer, kiểm tra checksum hòan tất, bộ điều khiển chuyển byte đầu tiên vào bộ nhớ chính tại dịa chỉ DMA. Thao tác được thực hiện đến khi chuyển xong dữ liệu

Chương 6: Quản lý xuất /nhập

- Thiết bị logic. Kiểm soát ngắt
  - Khi 1 ngắt xảy ra, hàm xử lý ngắt khởi tạo tiến trình xử lý ngắt
  - Chức năng của ngắt là làm cho tiến trình đang bị khóa được thi hành trở lại
- Device Drivers
  - Chức năng của device driver là nhận yêu cầu từ phần mềm nhập xuất độc lập thiết bị ở lớp trên và giám sát việc thực hiện các yêu cầu này
  - Sau khi HĐH hòan tất việc kiểm tra lỗi, driver sẽ chuyển dữ liệu cho phần mềm độc lập thiết bị, trả thông tin về trạng thái cho nơi gọi, kiểm tra hàng đợi để thực hiện tiếp hay khóa lại chờ yêu cầu tiếp theo

Chương 6: Quản lý xuất /nhập

- Phần mềm nhập xuất độc lập thiết bị
  - Chức năng cơ bản: cung cấp giao tiếp đồng nhất cho phần mềm phạm vi người sử dụng
  - Chức năng 1: tạo ánh xạ giữa thiết bị và tên gọi hình thức
  - Chức năng 2: bảo vệ thiết bị (vd: quyền truy nhập của user)
  - Chức năng 3: cung cấp khối dữ liệu độc lập thiết bị
  - Chức năng 4: cung cấp buffer đồng bộ hóa hoạt động
  - Chức năng 5: định vị lưu trữ trên thiết bị khối
  - Chức năng 6: cấp phát, giải phóng các thiết bị chuyên dụng
  - Chứcnăng 7: thông báo lỗi cho lớp trên từ các lỗi do device driver báo về

Chương 6: Quản lý xuất /nhập

- Phần mềm nhập xuất phạm vi người sử dụng
  - Thư viện nhập xuất chuẩn chứa một số hàm có chức năng nhập xuất, chạy như chương trình người dùng
  - Các hàm thư viện chuyển các tham số thích hợp cho lời gọi hệ thống
  - Chức năng của spooling là tránh trường hợp một tiến trình truy xuất và chiếm giữ thiết bị khiến các tiến trình khác không truy xuất được thiết bị đó
  - Ứng dụng của spooling: printing, sending/receiving email

Chương 6: Quản lý xuất /nhập

- Cài đặt hệ thống quản lý nhập xuất
  - Cài đặt hệ thống nhập xuất đĩa
  - Cài đặt hệ thống nhập xuất chuẩn
  - Cài đặt đồng hồ

Chương 6: Quản lý xuất /nhập

- Cài đặt hệ thống nhập xuất đĩa
  - Ưu điểm của đĩa: dung lượng, chi phí, bảo tòan thông tin
  - Cấu trúc vật lý: cylinder, track, head, sector
  - Tốc độ đĩa phụ thuộc vào các thao tác:
    - Seek: di chuyển đầu đọc đến track/cylinder (seek time *)
    - Chờ cho khối cần thiết đến dưới đầu đọc (latency time *)
    - Đọc dữ liệu từ đĩa vào bộ nhớ (transfer time)
  - HĐH cần có các thuật toán lập lịch truy xuất đĩa

Chương 6: Quản lý xuất /nhập

- Các thuật toán đọc đĩa
  - Lập lịch FCFS
  - Lập lịch SSTF
  - Lập lịch SCAN
  - Lập lịch C-SCAN
  - Lập lịch LOOK

Chương 6: Quản lý xuất /nhập

- Lập lịch FCFS (First Come, First Served)
  - Phương pháp đơn giản, dễ lập trình
  - Không cung cấp dịch vụ tốt

  Ví dụ đầu đọc đang ở khối 53, cần đọc các khối theo thứ tự

  98, 183, 37, 122, 14, 124, 65, 67

  Đầu đọc phải lần lượt đi qua các khối

  53, 98, 183, 37, 122, 14, 124, 65, 67

Chương 6: Quản lý xuất /nhập

- Lập lịch SSTF (Shortest-Seek-Time-First)
  - Di chuyển đầu đọc đến các khối cần thiết theo vị trí lần lượt gần với vị trí hiện hành của đầu đọc nhất

  Ví dụ đầu đọc đang ở khối 53, cần đọc các khối theo thứ tự

  98, 183, 37, 122, 14, 124, 65, 67

  Đầu đọc lần lượt đi qua các khối

  53, 65, 67, 37, 14, 98, 122, 124, 183
  - Thích hợp cho hệ thống cần truy xuất dữ liệu liên tục

Chương 6: Quản lý xuất /nhập

- Lập lịch SCAN
  - Đầu đọc di chuyển về 1 phía của đĩa và từ đó di chuyển qua phía kia.

  Ví dụ đầu đọc đang ở khối 53, cần đọc các khối theo thứ tự

  98, 183, 37, 122, 14, 124, 65, 67

  Đầu đọc lần lượt đi qua các khối

  53, 37, 14, 0, 65, 67, 98, 122, 124 và 183
  - Thích hợp cho hệ thống truy xuất dữ liệu khối lượng lớn

Chương 6: Quản lý xuất /nhập

- Lập lịch C-SCAN
  - Tương tự thuật tóan SCAN, chỉ khác khi di chuyển đến một đầu nào đó của đĩa, nó sẽ lập tức trở về đầu bắt đầu của đĩa

  Ví dụ đầu đọc đang ở khối 53, cần đọc các khối theo thứ tự

  98, 183, 37, 122, 14, 124, 65, 67

  Đầu đọc lần lượt đi qua các khối

  53, 65, 67, 98, 122, 124, 183, 0, 14, 37
  - Thích hợp cho hệ thống truy xuất dữ liệu khối lượng lớn

Chương 6: Quản lý xuất /nhập

- Lập lịch LOOK
  - Giống C-SCAN nhưng chỉ chuyển đến khối xa nhất ở mỗi hướng chứ không đến cuối.

  Ví dụ đầu đọc đang ở khối 53, cần đọc các khối theo thứ tự

  98, 183, 37, 122, 14, 124, 65, 67

  Đầu đọc lần lượt đi qua các khối

  53, 65, 67, 98, 122, 124, 183, 14, 37
  - Thích hợp cho hệ thống truy xuất dữ liệu khối lượng lớn

Chương 6: Quản lý xuất /nhập

- Quản lý lỗi
  - Lỗi lập trình: tìm không thấy cylinder, sector, head, địa chỉ buffer. Xử lý bằng kiểm tra tham số, thông báo lỗi.
  - Lỗi checksum tạm thời: gây ra bởi bụi trên đầu đọc
  - Lỗi checksum thường trực: đĩa hư vật lý trên các khối
  - Lỗi tìm kiếm: seek đầu đọc sai địa chỉ
  - Lỗi điều khiển: bộ điều khiển từ chối thi hành lệnh

Chương 6: Quản lý xuất /nhập

- RAM disk
  - RAM disk dùng một phần bộ nhớ chính để lưu trữ các khối dữ liệu
  - RAM disk được chia làm nhiều khối tùy theo dung lượng yêu cầu cấp phát. Mỗi khối có cùng kích thước với khối trên đĩa
  - Khi nhận được chỉ thị đọc/ghi các khối, driver tìm trong bộ nhớ RAM disk vị trí của khối, thực hiện đọc/ghi ngay trong vùng nhớ
  - RAM disk truy xuất nhanh hơn đĩa vật lý
  - HĐH phải lưu dữ liệu vào đĩa cứng trước khi người dùng shutdown

Chương 6: Quản lý xuất /nhập

- Interleave
  - Khoảng cách giữa các sector dùng để đồng bộ chức năng đọc/ghi dữ liệu trên đĩa.
  - Interleave được xác định trong quá trình format đĩa

VD: đĩa có 17 sector/track, interleave=4, sơ đồ sector như sau:

1, 14, 10, 6, 2, 15, 11, 7, 3, 16, 12, 8, 4, 17, 13, 9, 5

Lần 1: **1**, 14, 10, 6, **2**, 15, 11, 7, **3**, 16, 12, 8, **4**, 17, 13, 9, **5**
Lần 2: 1, 14, 10, **6**, 2, 15, 11, **7**, 3, 16, 12, **8**, 4, 17, 13, **9**, 5
Lần 3: 1, 14, **10**, 6, 2, 15, **11**, 7, 3, 16, **12**, 8, 4, 17, **13**, 9, 5
Lần 4: 1, **14**, 10, 6, 2, **15**, 11, 7, 3, **16**, 12, 8, 4, **17**, 13, 9, 5

Chương 6: Quản lý xuất /nhập

- Cài đặt hệ thống nhập xuất chuẩn (terminal)
  - Terminal là hệ thống nhập xuất (chỉ có bàn phím, màn hình và bộ điều hợp dữ liệu vào ra)
  - Terminal chuyển dữ liệu Một máy tính có thể liên lạc với nhiều terminal
  - HĐH chia terminal thành 2 loại: RS-232 và ánh xạ bộ nhớ
  - RS-232: hard-copy, glasstty, Intelligent, blit
  - Ánh xạ bộ nhớ: ký tự, bit

Chương 6: Quản lý xuất /nhập

- Các terminal RS-232
  - Thiết bị giao tiếp tuần tự theo bit với bàn phím, màn hình...
  - Connector 25 pins: mass, receiver, sender, 22 reserved pins
  - Data package: start bit-data to be tranfered-end bit(s)
  - Tốc độ chuyển:1200, 2400, 4800, 9600 bps (bit per second)
  - Sử dụng bộ chuyển UART gắn trên card giao tiếp
  - Hard-copy/glassttys: nhập ký tự từ bàn phím, chuyển cho máy tính, xuất ra máy in/màn hình
  - Terminal Intelligent: gửi ký tự ASCII ESC sau những ký tự khác nhau để di chuyển con trỏ trên màn hình
  - Blit: bộ xử lý mạnh với màn hình độ phân giải 1024x800

Chương 6: Quản lý xuất /nhập

- Các terminal ánh xạ bộ nhớ
  - Giao tiếp với máy tính bằng video RAM (không dùng cổng serial)
  - Chip điều khiển trên video card lấy thông tin từ video RAM, tạo ra tín hiệu video để điều khiển màn hình
  - Màn hình tạo tia điện tử quét từ trên xuống dưới.
  - Bộ điều khiển tín hiệu sẽ xác định mỗi điểm (pixel) là sáng hay tối (đối với màn hình mono), màu gì (màn hình màu)
  - Text mode (mono $B000, màu $B800): 2000 ký tự 2 byte (thuộc tính-chữ), box 9x14, bố trí trên 25 dòng, 80 cột. Graphic mode ($A000): độ phân giải 320x200, 640x480, 800x600, 1027x768... pixels

Chương 6: Quản lý xuất /nhập

- Phần mềm nhập cho terminal
  - Dữ liệu nhập từ bàn phím, chuyển cho ứng dụng
  - Khi có phím nhấn, ngắt bàn phím báo cho bộ điều khiển biết có ký tự nhập đang lưu trữ trong buffer của bộ nhớ chính. Bộ điều khiển sẽ ánh xạ lại mã ASCII của ký tự phím nhấn
  - Có 2 dạng buffer bàn phím: pool buffer, structred buffer
  - Một số terminal cần phần mềm echoing điều khiển hiển thị ký tự gõ: mật khẩu, tab, backspace, line feed, carriage return...

Chương 6: Quản lý xuất /nhập

- Phần mềm xuất cho terminal
  - Mỗi loại terminal sử dụng phần mềm xuất khác nhau
  - Terminal RS-232 sử dụng pool buffer chứa dữ liệu xuất: các ký tự được xuất tuần tự theo các tín hiệu ngắt
  - Terminal ánh xạ: các ký tự được xuất 1 lần từ video RAM. Các ký tự đặc biệt (backspace, bell, cr, lf...) được cập nhật cho phù hợp.
  - Các chức năng của phần mềm soạn thảo màn hình terminal: di chuyển con trỏ, chèn xóa ký tự/dòng, cuộn màn hình lên xuống, tạo hiệu ứng (tương phản, gạch dưới, nhấp nháy...), tạo/hủy/di chuyển/quản trị cửa sổ...

Chương 6: Quản lý xuất /nhập

- Cài đặt đồng hồ (timer)
  - Timer là thiết bị phần cứng đặc biệt, không thuộc thiết bị khối (vd đĩa) hay thiết bị tuần tự (vd bàn phím, màn hình).
  - Chức năng chính của timer
    - Kiểm soát thời gian trong ngày
    - Phân chia thời gian chia xẻ cho các tiến trình sử dụng CPU
  - Phần mềm đồng hồ hoạt động như device driver

Chương 6: Quản lý xuất /nhập

- Phần cứng đồng hồ:
  - Dạng 1: sử dụng điện thế 110/220v, tạo ngắt theo mỗi chu kỳ hiệu điện thế 50/60 MHz
  - Dạng 2: bộ dao động thạch anh, bộ đếm và bộ thanh ghi
    - Dưới tác dụng của dòng diện, tinh thể thạch anh tạo ra dao động 5-100 MHz, chuyển cho bộ đếm
    - Bộ đếm giảm dần sau mỗi dao động, tạo ngắt khi Counter=0. Bộ thanh ghi sẽ nạp lại giá trị cho bộ đếm
    - Khi ngắt đồng hồ kích hoạt, HĐH sẽ thực hiện trình xử lý ngắt của nó

Chương 6: Quản lý xuất /nhập

Phần mềm đồng hồ

- Phần cứng đồng hồ tạo các ngắt theo từng khoảng thời gian đều đặn
- Phần mềm (driver) đồng hồ có nhiệm vụ:
  - Quản lí thời gian trong ngày
  - Không cho phép tiến trình chạy lâu hơn thời gian cho phép
  - Điều phối kế hoạch sử dụng CPU
  - Cung cấp watchdog timer cho hệ thống

Chương 6: Quản lý xuất /nhập

- Quản lý thời gian trong ngày
  - Tăng bộ đếm sau mỗi nhịp đồng hồ.
  - Vấn đề lưu ý : Kích thước bộ đếm
  - Các loại bộ đếm:
    - Bộ đếm nhịp 32 bits: dao động 60MHz bị tràn sau 2 năm
    - Bộ đếm 64 bits: tốn kém
    - Bộ đếm giây 32 bits: 2^{32} lưu được 136 năm
    - Bộ đếm nhịp liên hệ với thời gian khởi động của hệ thống

Chương 6: Quản lý xuất /nhập

- Quản lý thời gian chạy của tiến trình
  - Khi một tiến trình bắt đầu, bộ lập lịch sẽ khởi tạo giá trị cho bộ đếm.
  - Giá trị này giảm dần sau mỗi ngắt đồng hồ
  - Counter=0, thời gian chạy của tiến trình kết thúc. Bộ điều khiển đồng hồ sẽ yêu cầu bộ lập lịch thiết lập giá trị cho tiến trình khác
- Điều phối kế hoạch sử dụng CPU
  - Sử dụng timer cục bộ cho từng tiến trình
  - Khởi tạo, kích hoạt timer khi tiến trình bắt đầu
  - Dừng timer khi tiến trình kết thúc

Chương 6: Quản lý xuất /nhập

- Cung cấp watchdog timer
  - Một số thiết bị nhập xuất cần đo thời gian đạt đến trạng thái sẵn sàng hoạt động

  VD: Sau 500 ms từ lúc khởi động, ổ đĩa mềm mới đạt được tốc độ cần thiết cho các tác vụ truy xuất
  - Watchdog timer đếm thời gian cho các thiết bị nhập xuất

  VD: Không tắt motor ổ đĩa mềm, chờ các thao tác nhập xuất tiếp theo. Quá thời gian không có yêu cầu truy xuất, tắt motor.

HỆ ĐIỀU HÀNH

Trường Đại Học Mở TP.HCM – Ban Trung Học Chuyên Nghiệp

MỤC LỤC

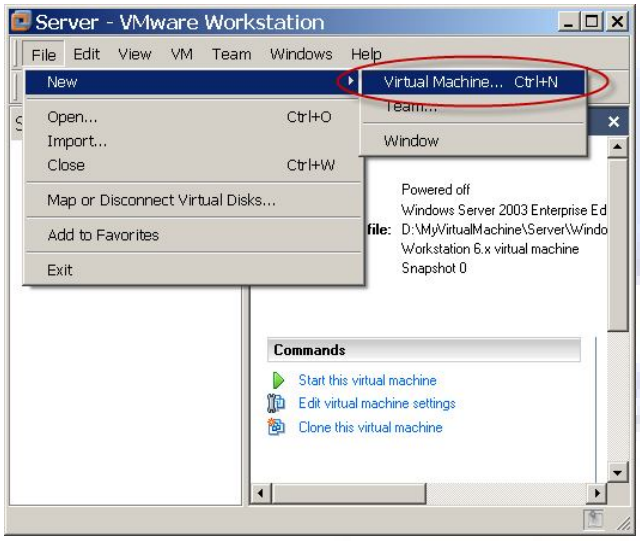
Lab 1
CÀI ĐẶT HỆ ĐIỀU HÀNH TRÊN VMWARE

Yêu cầu : cài đặt hệ điều hành Windows XP trên máy ảo

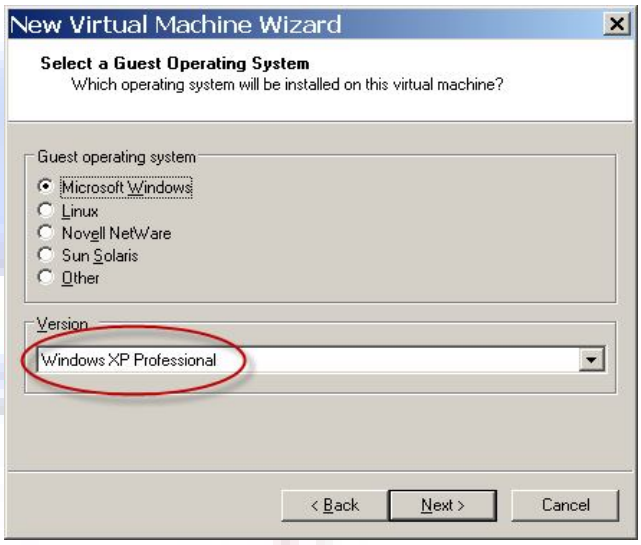
Hướng dẫn:

1. Tạo máy ảo

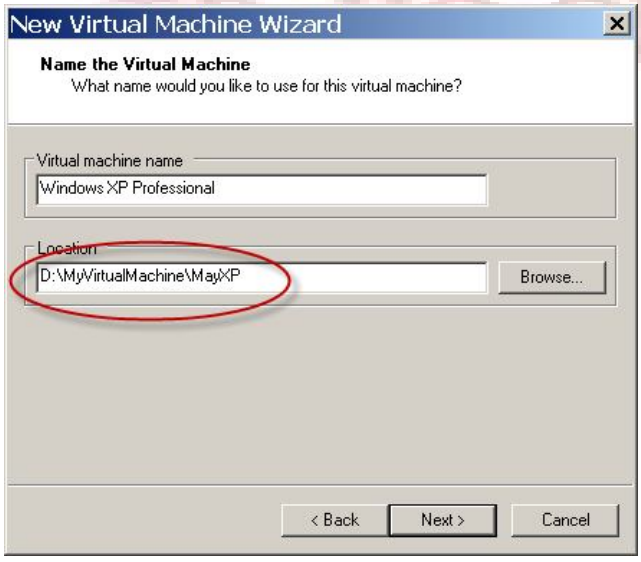
B1 : Khởi động phần mềm VMWare. Nhấn **File** \ **New** \ **Virtual Machine**. Nhấn **Next**, chọn mục **Typical**, nhấn **Next**



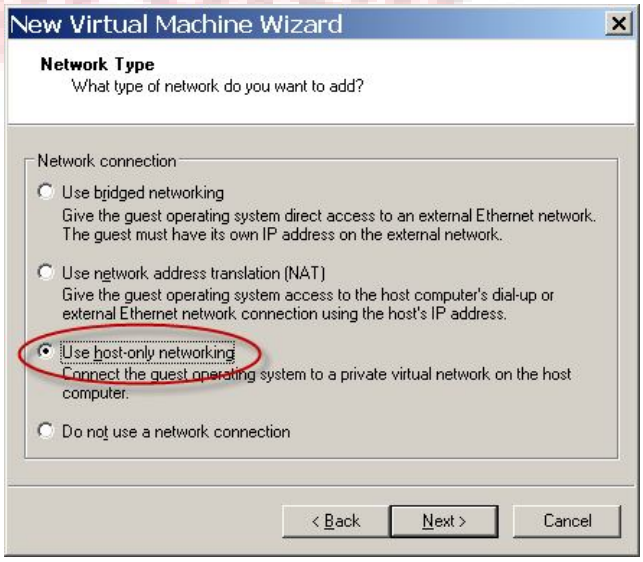
B2 : chọn hệ điều hành cần cài đặt **Windows XP Professional**. Nhấn **Next**

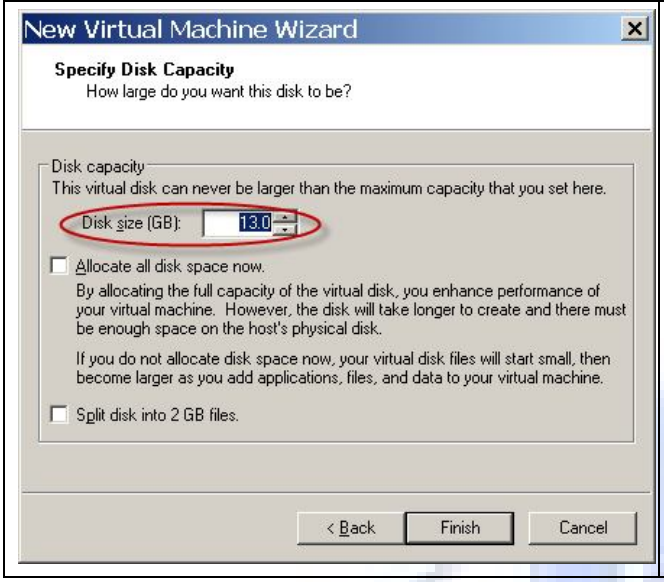
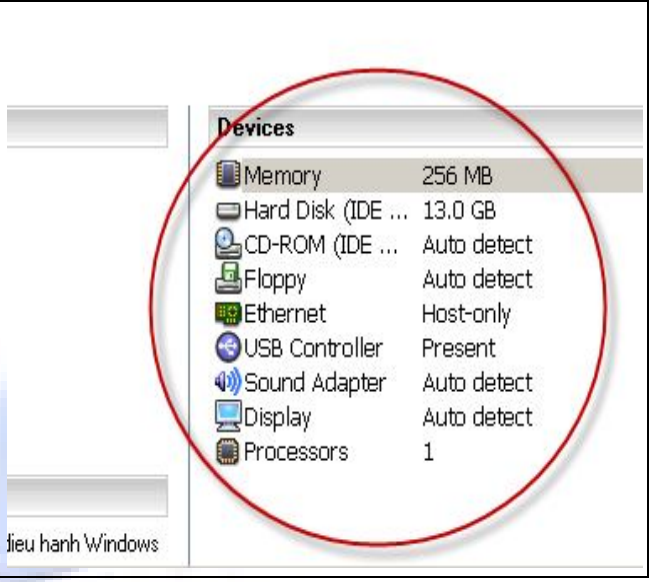


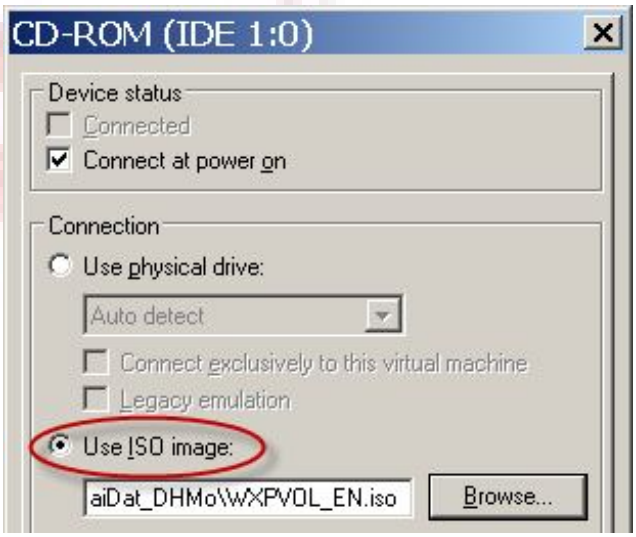
B3 : chọn thư mục lưu trữ máy ảo, nhấn **Next**



B4 : cấu hình cho card mạng **Use host only networking** (dùng cho mạng nội bộ)

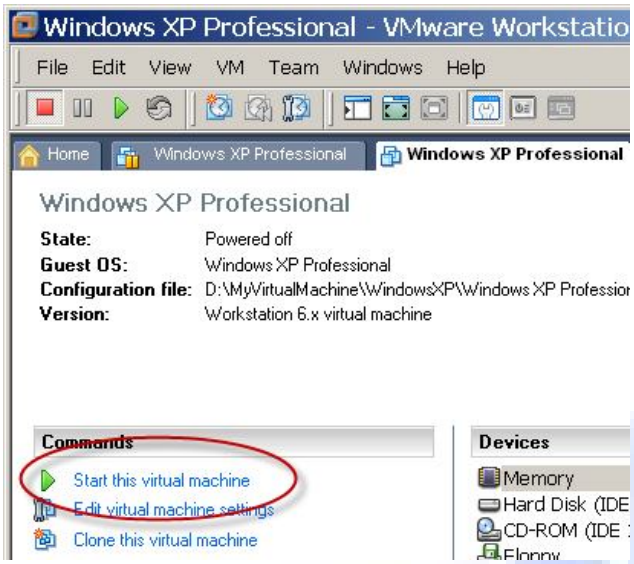


B5 : khai báo dung lượng ổ đĩa cứng là **5GB**, nhấn **Finish** để hoàn tất	**B6** : chỉnh sửa một số các thông số trong mục **Devices**
	

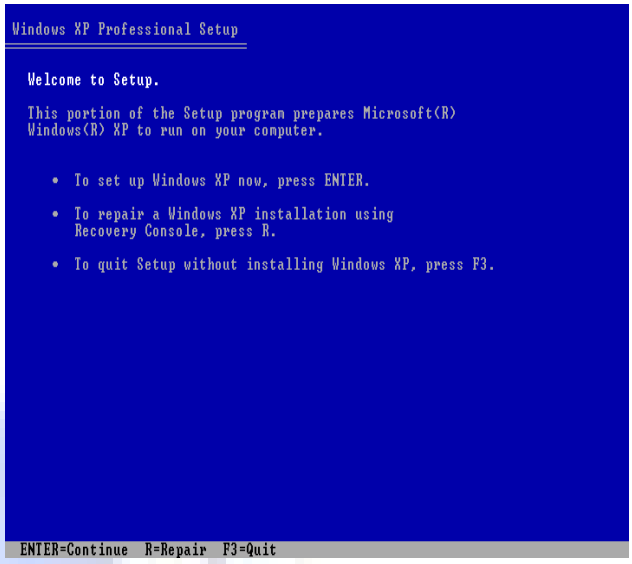
B7 : trong mục **Devices**, nhấn double vào mục **Floppy** bỏ mục **Connect Power On**	**B8** : trong mục **Devices**, nhấn double vào mục **CD-ROM** chọn mục **Use ISO image** và chỉ đường dẫn đến tập tin **WXPVOL_EN.iso**
	

2. Cài đặt Windows XP Professional

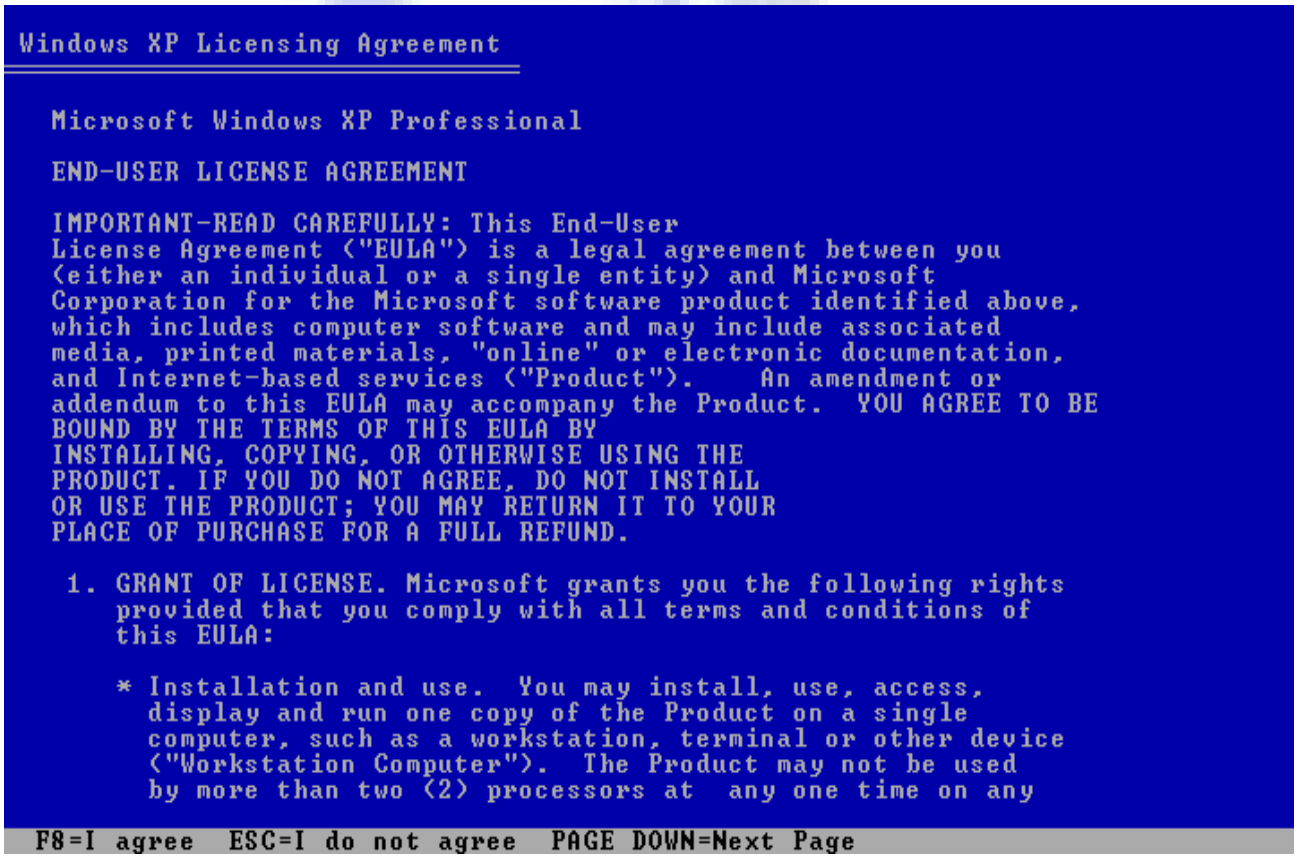
B1 : nhấn **Start this virtual machine** để khởi động máy ảo.



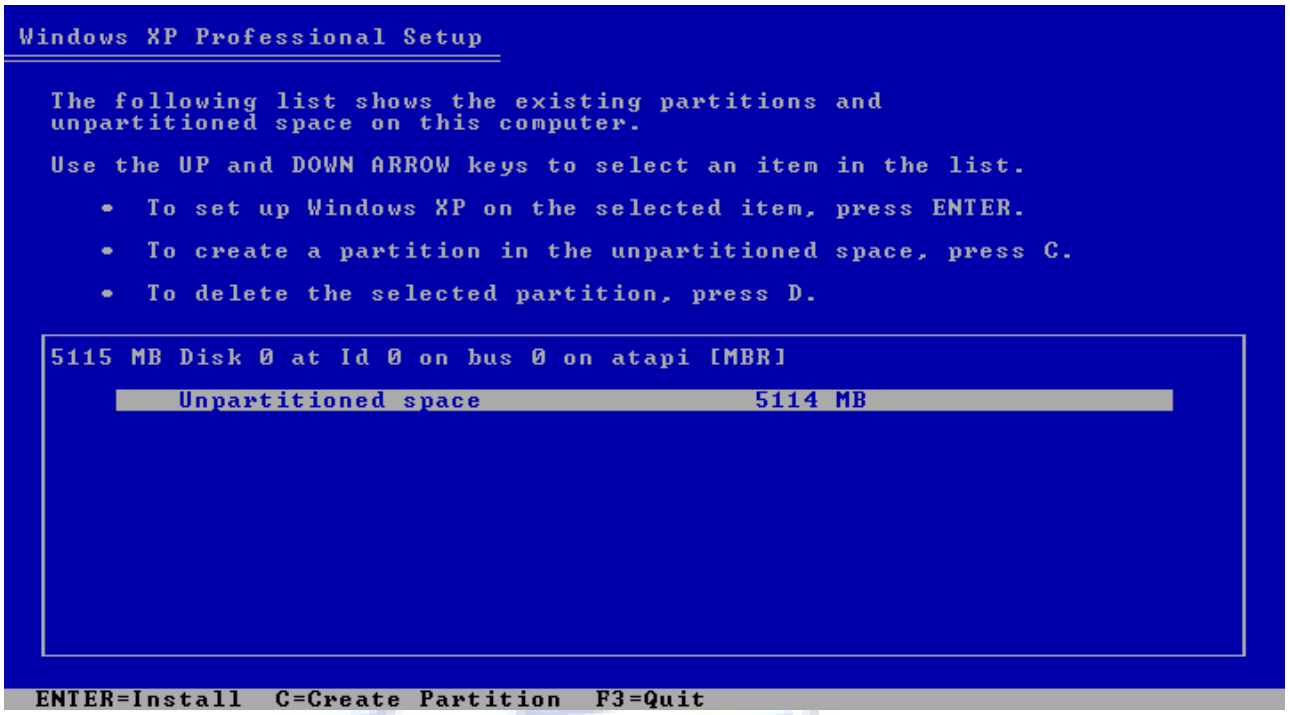
B2 : Nhấn **Enter** để tiếp tục.



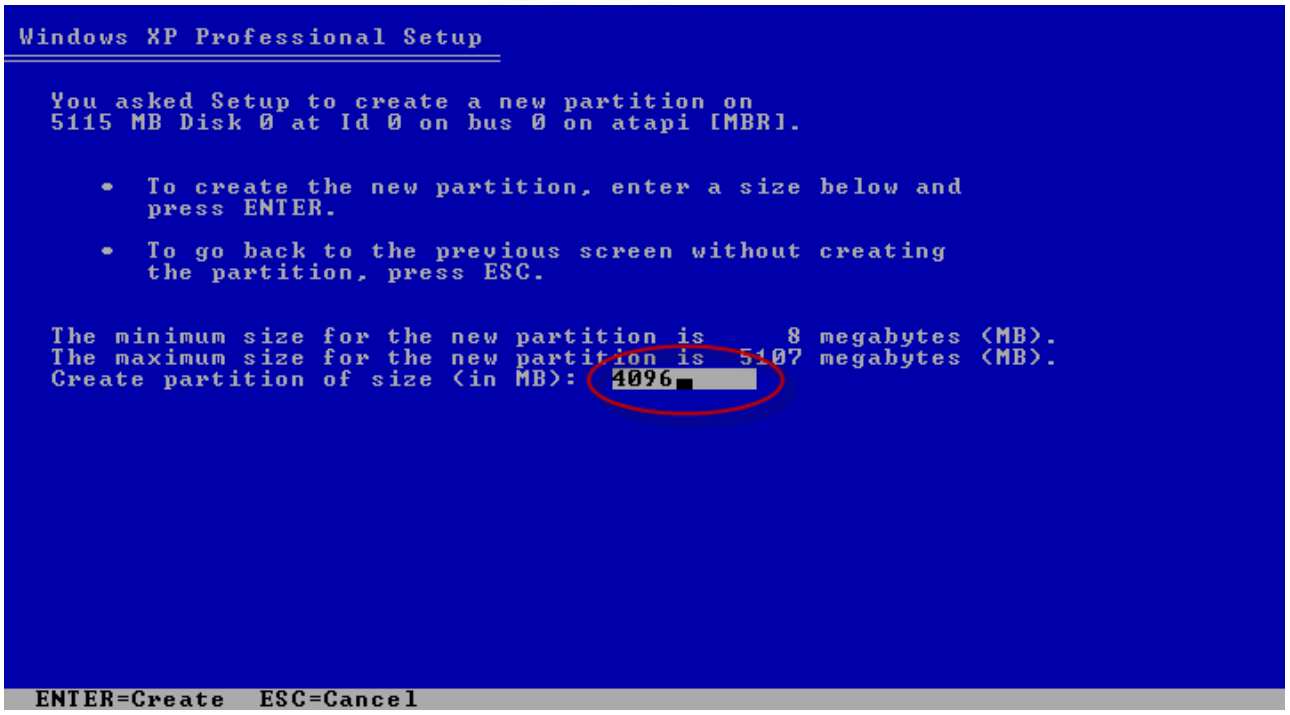
B3 : nhấn **F8** đồng ý các điều khoản.



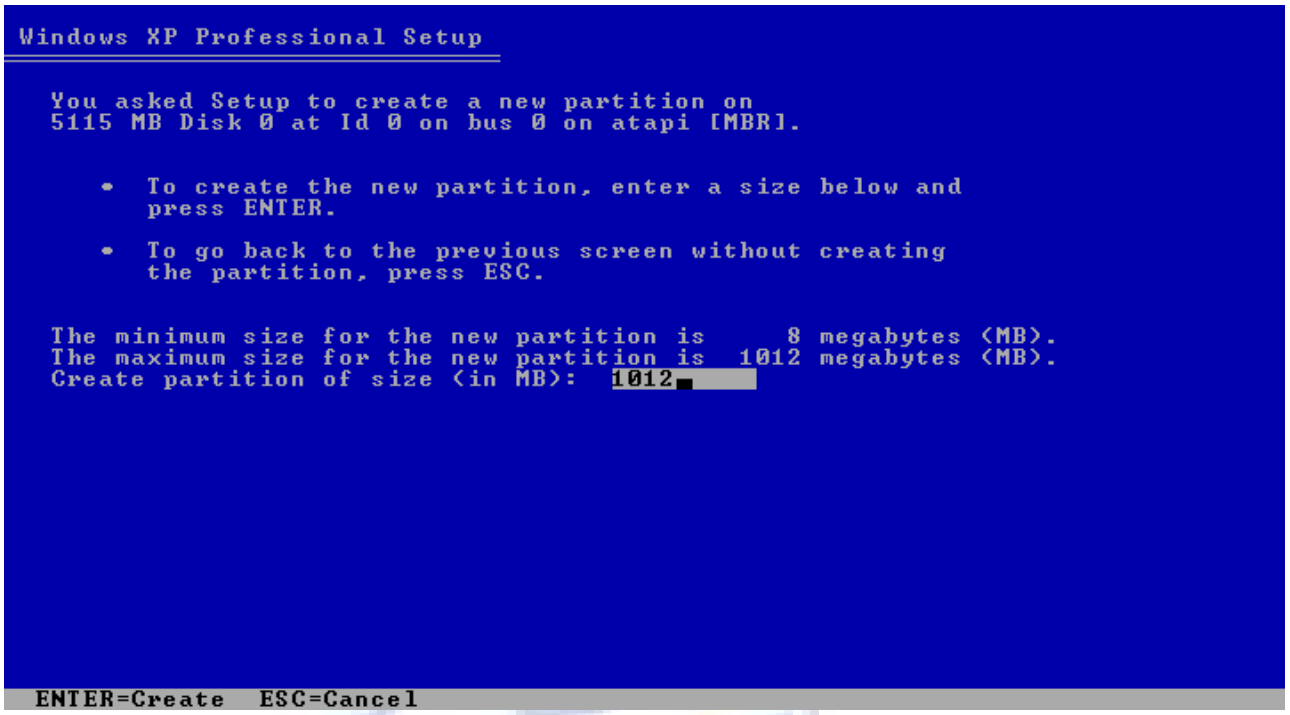
B4 : nhấn phím **C** để tạo mới partition.



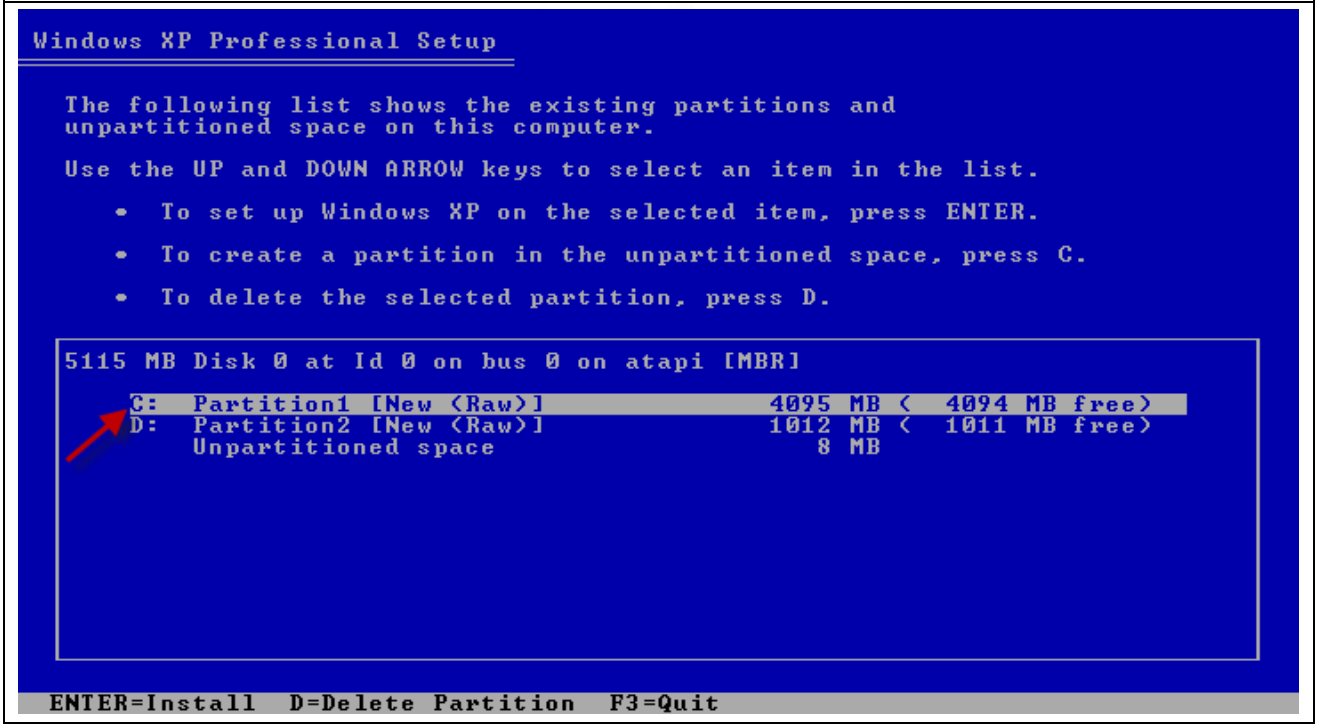
B5 : nhập dung lượng cho partition là 4GB (≈ **4096** MB) và nhấn **Enter** để tạo mới partition.



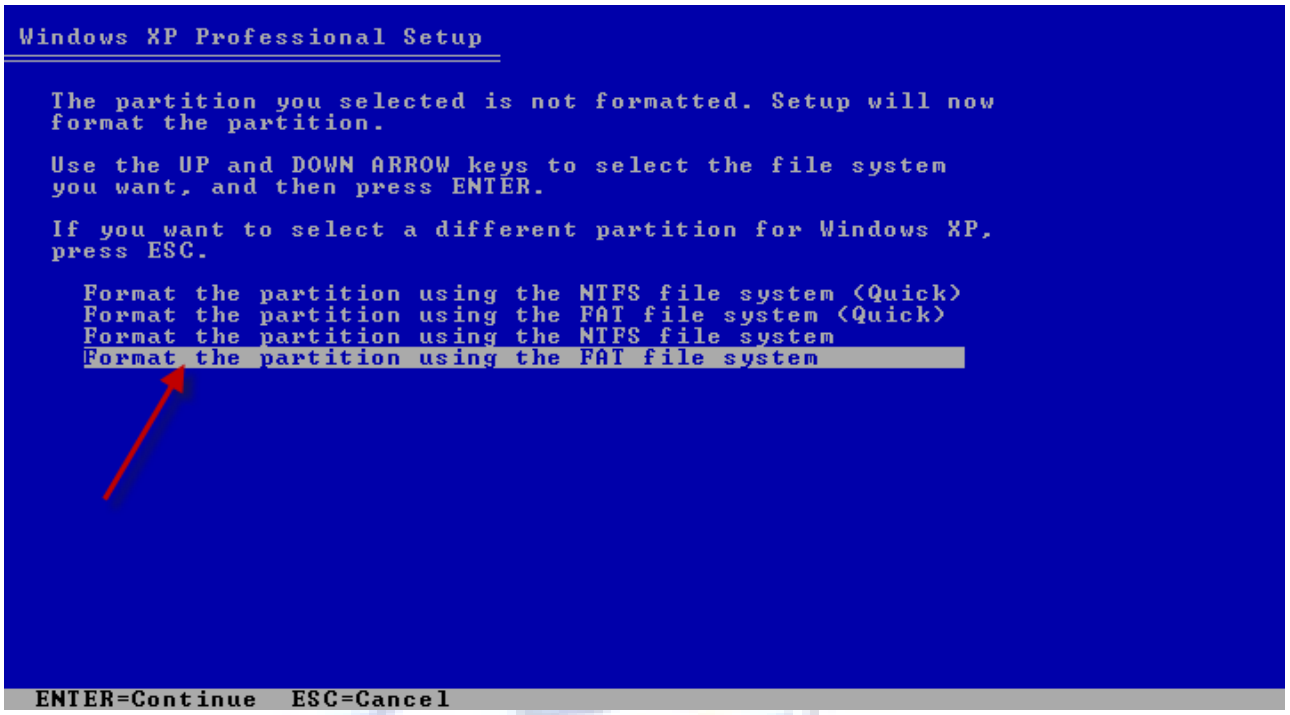
B6 : nhấn **Enter** để tạo thêm partition có dung lượng 1012 MB.



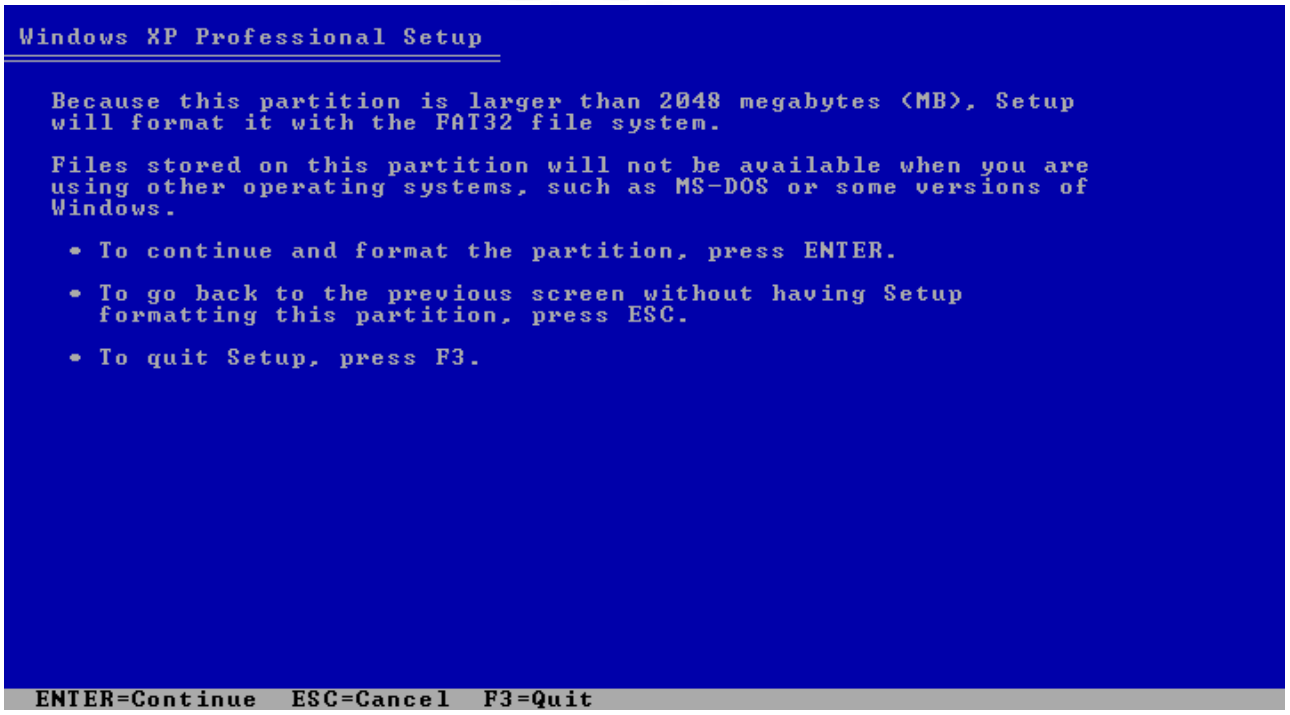
B7 : chọn ổ đĩa **C:** và nhấn **Enter** để bắt đầu cài đặt hệ điều hành lên partition này.



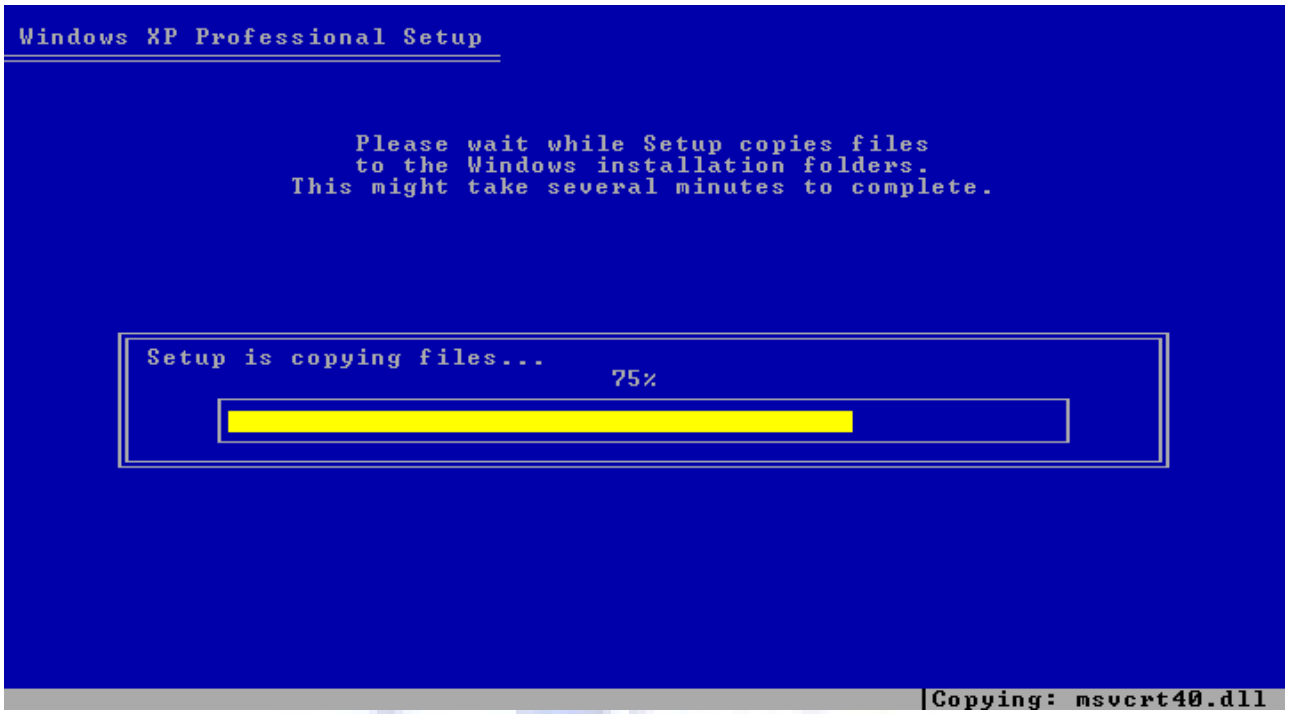
B8 : định dạng partition này là **FAT32**. Nhấn **Enter** để tiếp tục



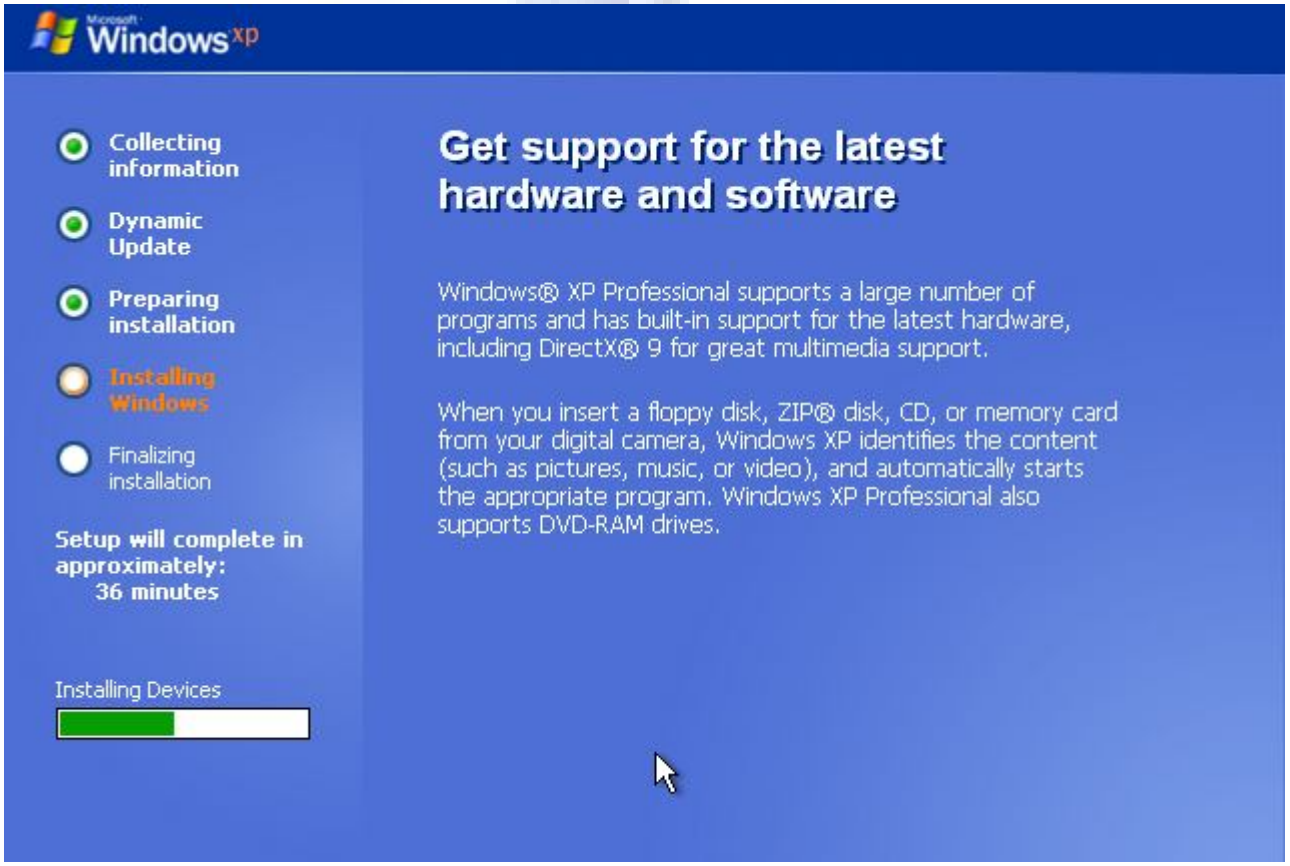
B9 : Nhấn **Enter** để tiếp tục.



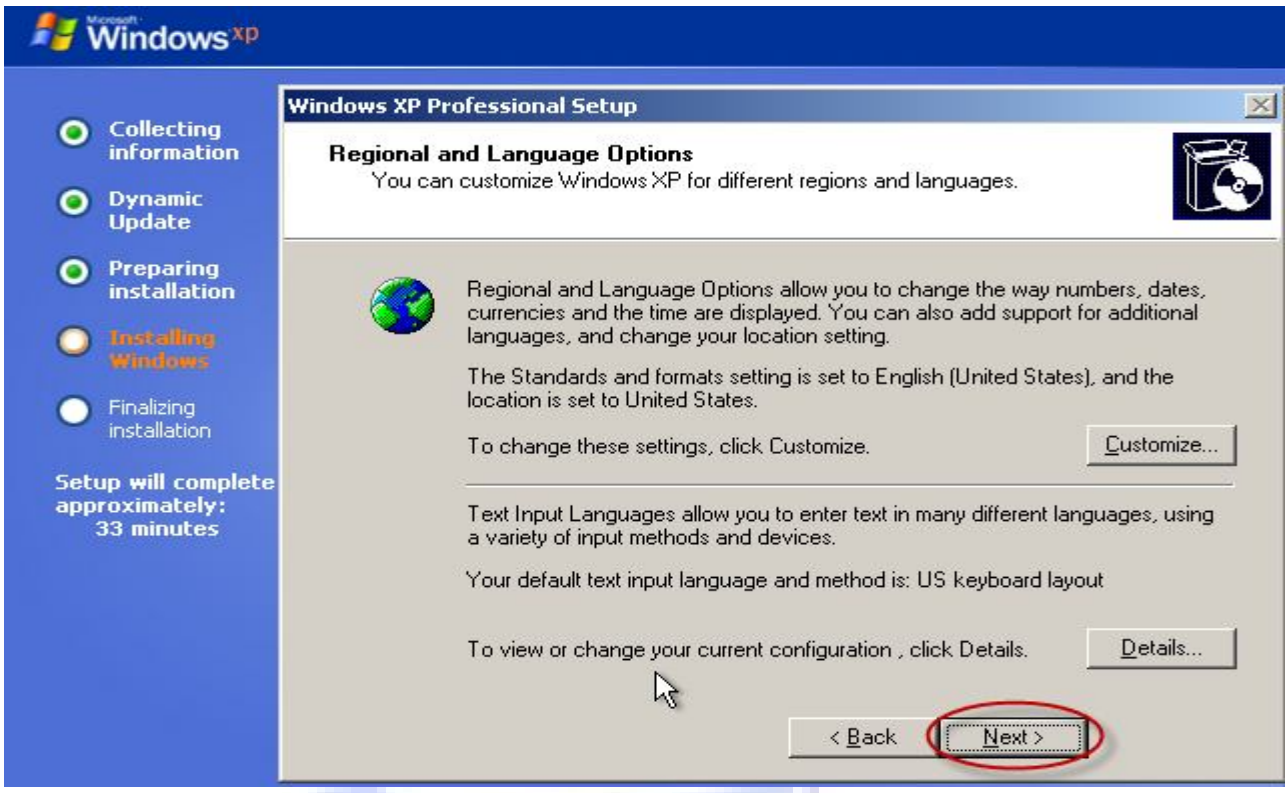
B10 : Quá trình copy các tập tin bắt đầu



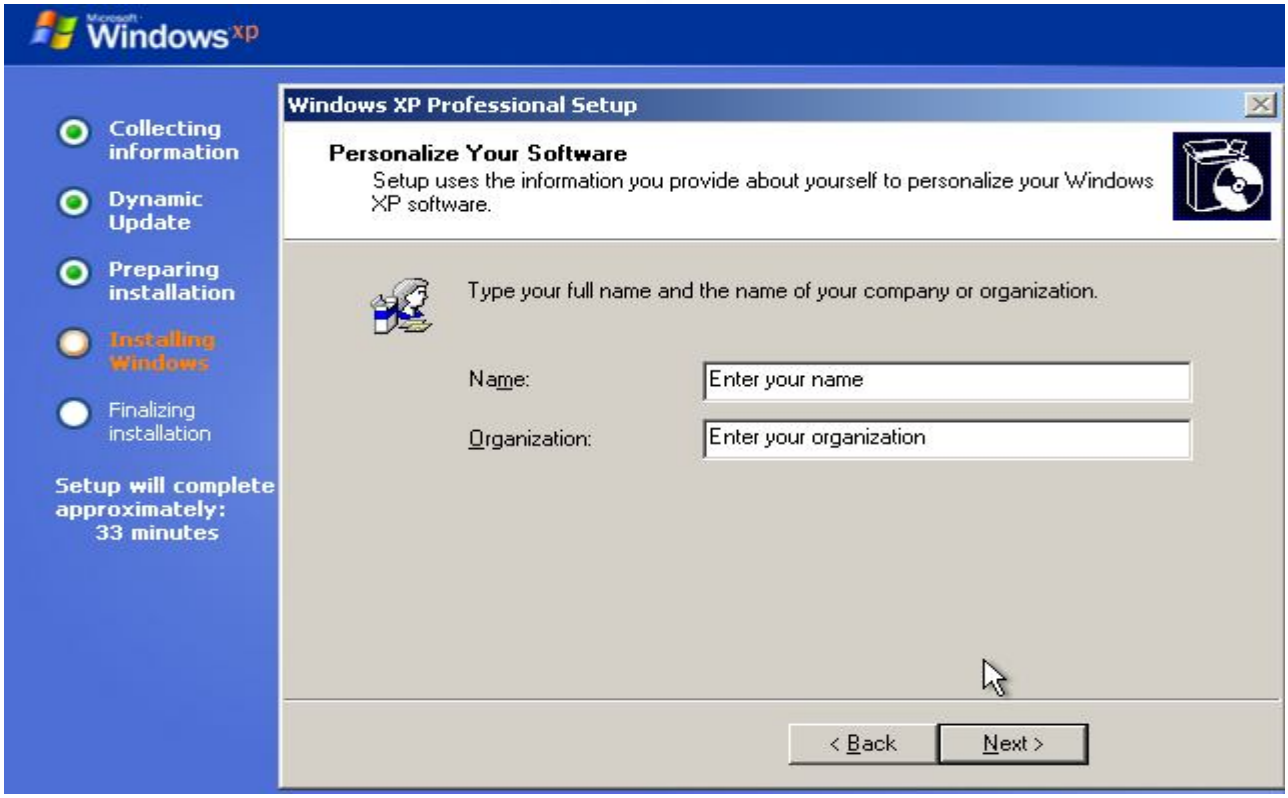
B11 : Quá trình cài đặt bắt đầu



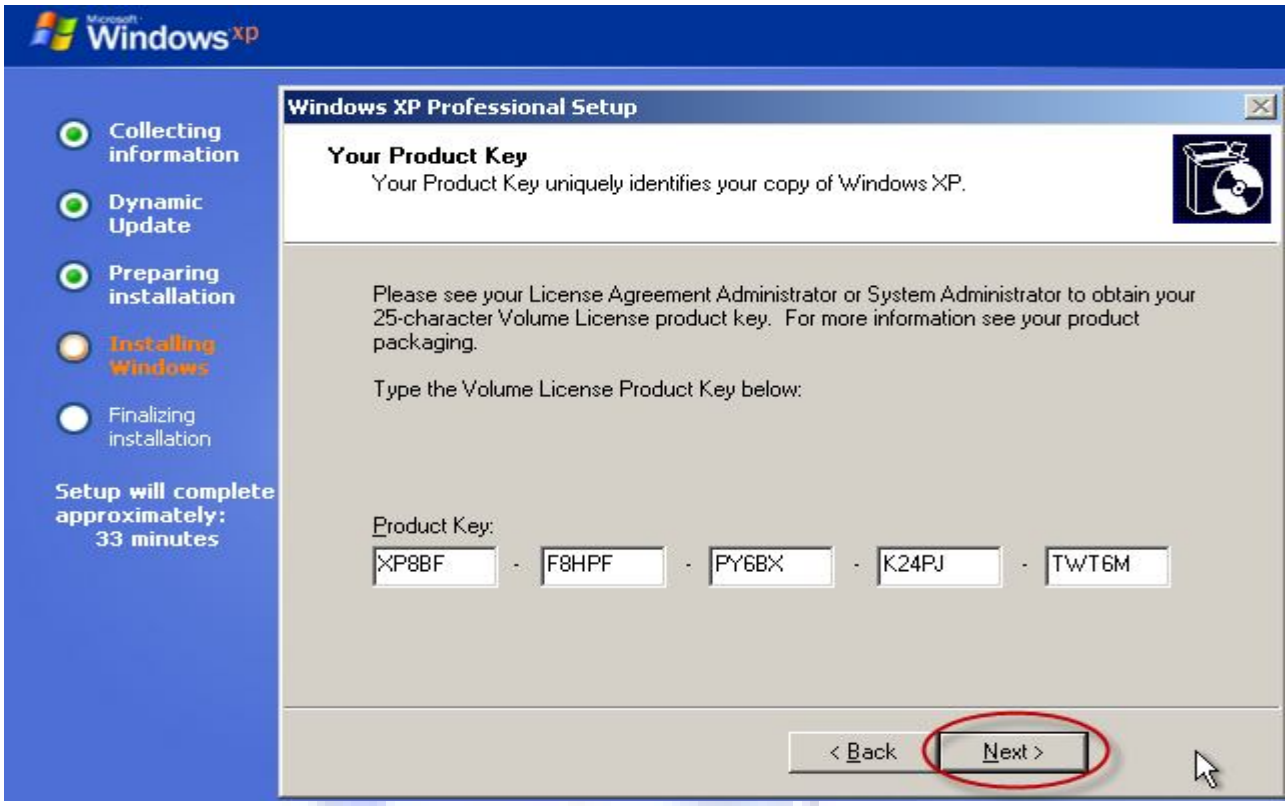
B12 : Nhấn **Next** để tiếp tục.



B13 : Nhập tên và tổ chức vào các hộp thông tin và nhấn **Next** để tiếp tục.

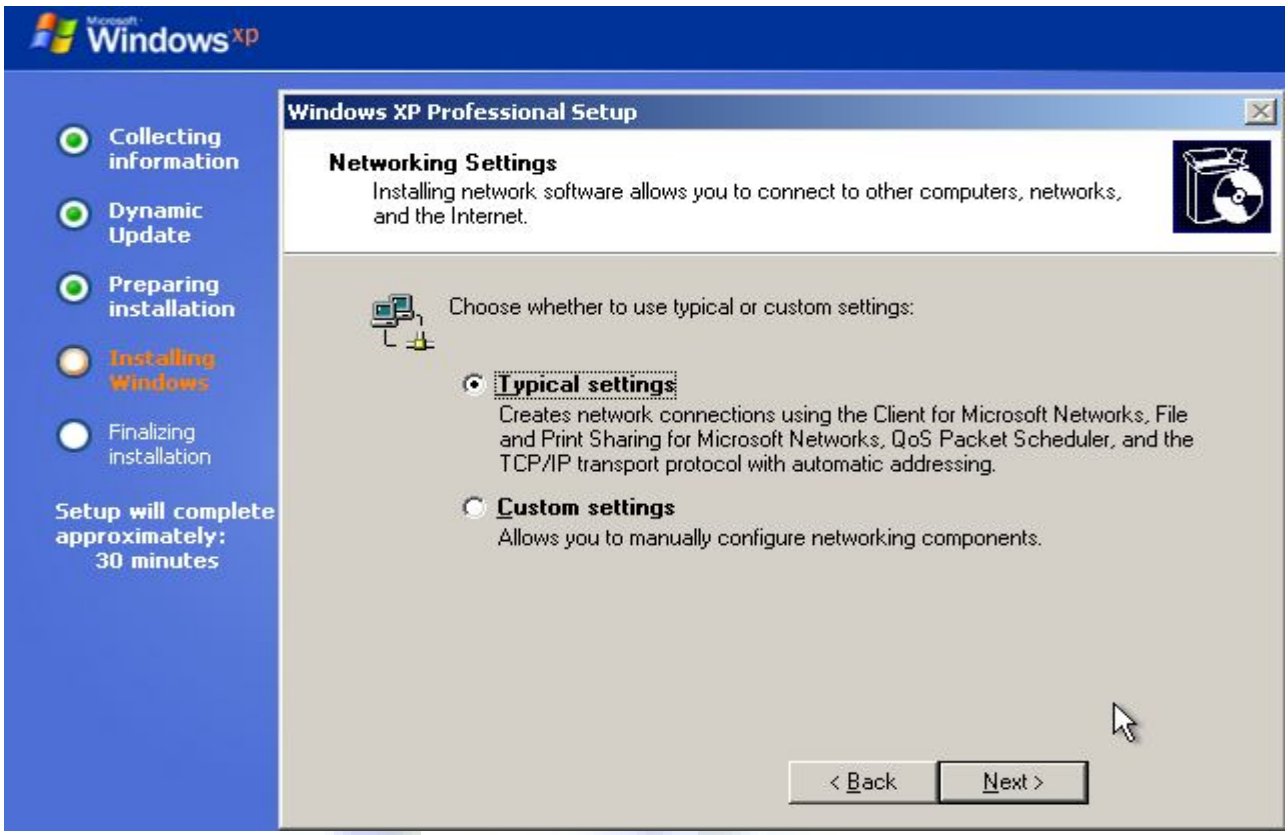


B14 : Nhập số CD-Key: XP8BF-F8HPF-PY6BX-K24PJ-TWT6M và nhấn **Next** để tiếp tục.



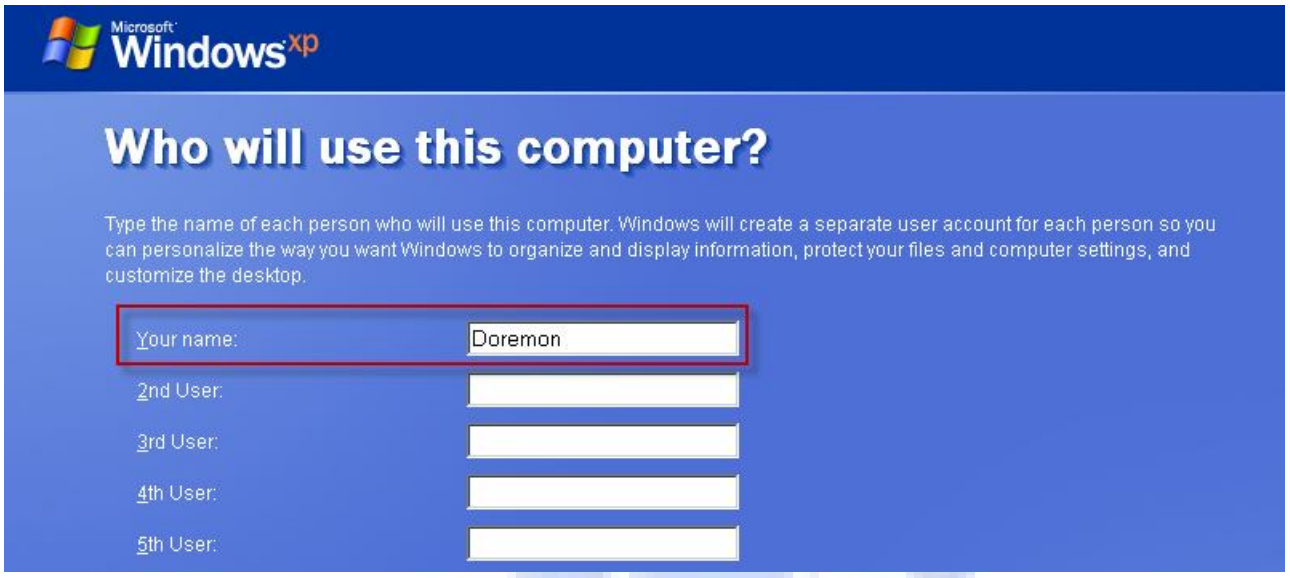
B15 : Nhập tên cho máy tính và mật khẩu của người Administrator.

B16 : Cấu hình kết nối mạng, nhấn **Next** để tiếp tục.

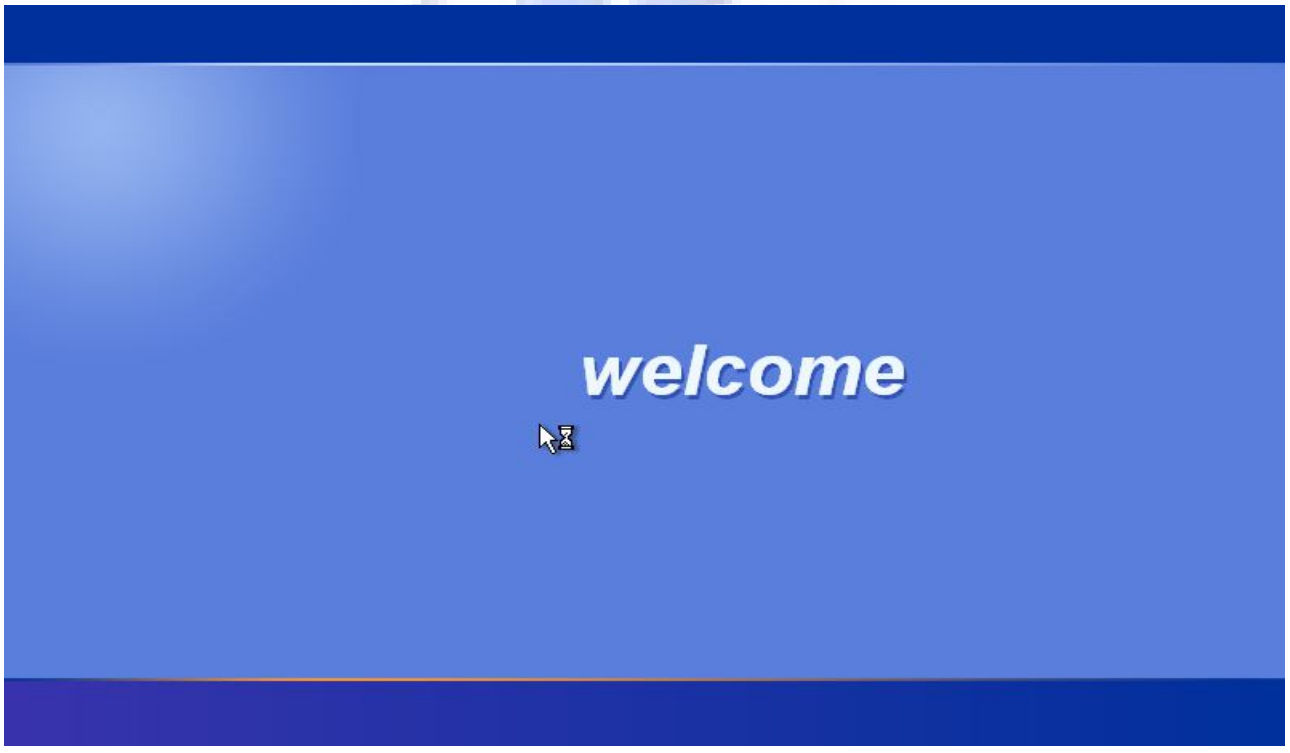


B17 : Quá trình cài đặt bước vào giai đoạn hoàn tất.

B18 : Tạo tài khoản để đăng nhập hệ thống, nhấn **Next** để tiếp tục



B19 : Màn hình Welcome chào đón người dùng



BÀI TẬP

Tạo máy ảo với cấu hình như sau: **HDD** (9GB); **RAM** (512MB)

1. Cài đặt hệ điều hành **Windows 7** trên máy ảo.
2. Cài đặt hệ điều hành **Windows Server 2003** trên máy ảo.

Lab 2
CÁC LỆNH VỀ THƯ MỤC TẬP TIN

0. Hãy tạo cây thư mục sau

D:
|_____**Computer**
|_____**Monitor**
|
|_____**M**
|
|_____**CPU**
|
|_____**RAM**
|
|_____**Cache**
|
|_____**Temp**

1. Sao chép thư mục **RAM** vào thư mục **Monitor** (tương đương với Copy → Paste)

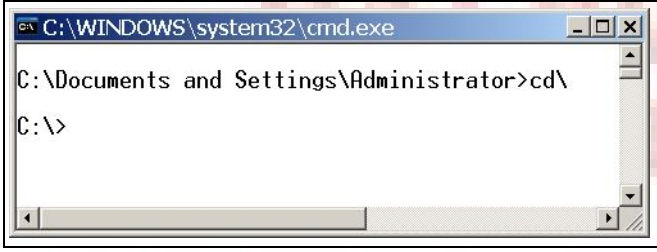
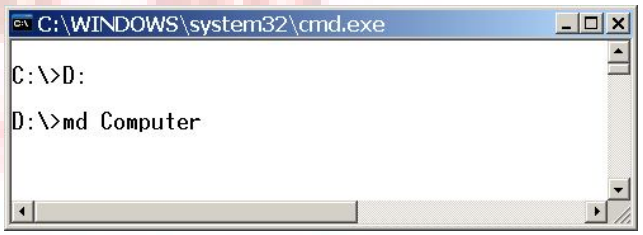
2. Di chuyển thư mục **Cache** vào thư mục **CPU** (tương đương với Cut → Paste)

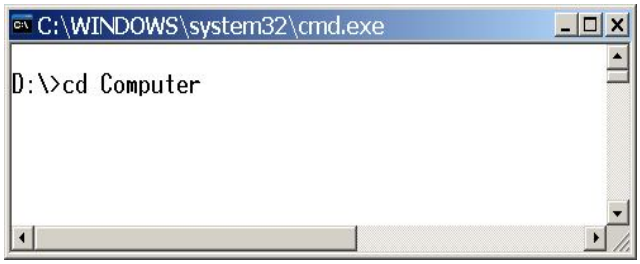
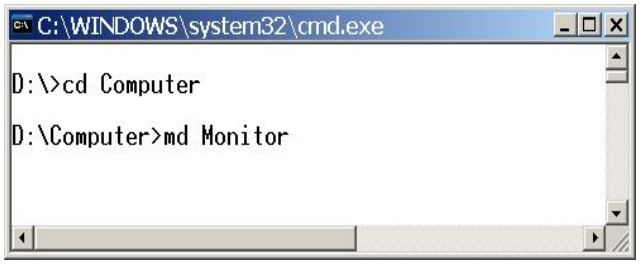
3. Đổi tên thư mục **M** thành **Mainboard**

4. Xoá thư mục **TEMP**

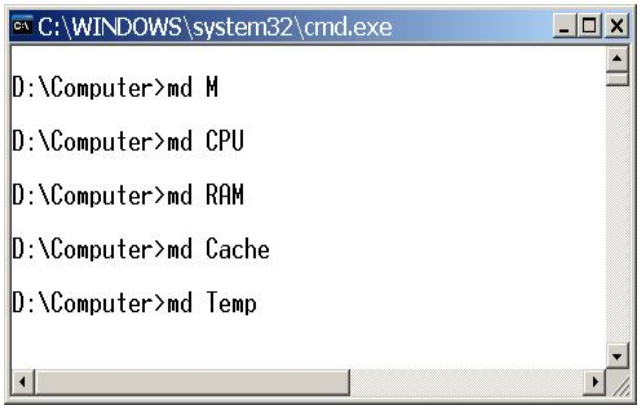
Hướng dẫn:

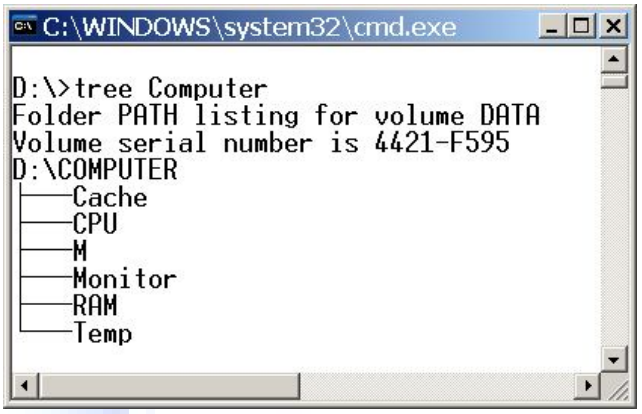
0. Tạo cây thư mục

B1: khởi động chế độ command promt trong Windows bằng cách nhấn **Start → Run →** gõ **cmd → OK**. Chuyển về thư mục gốc ổ đĩa C bằng lệnh **cd**	**B2:** chuyển đến ổ đĩa **D:** và tạo thư mục **Computer** bằng lệnh **md**
	

B3: chuyển đến thư mục **Computer** bằng lệnh **cd**. Trong lúc làm có thể sử dụng lệnh **cls** để xoá màn hình hiện tại để nhìn cho thoáng hơn.	**B4:** tạo thư mục **Monitor** trong thư mục **Computer** D:\Computer>**md Monitor**
	

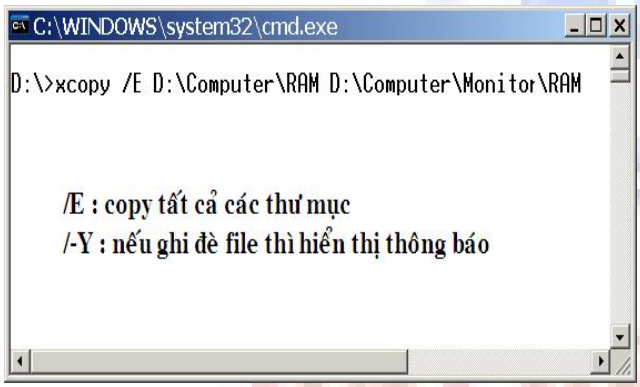
B5: dùng lệnh **md** để tạo tiếp các thư mục còn lại nằm trong thư mục **Computer**	**B6:** xem lại cây thư mục đã tạo bằng lệnh **tree**





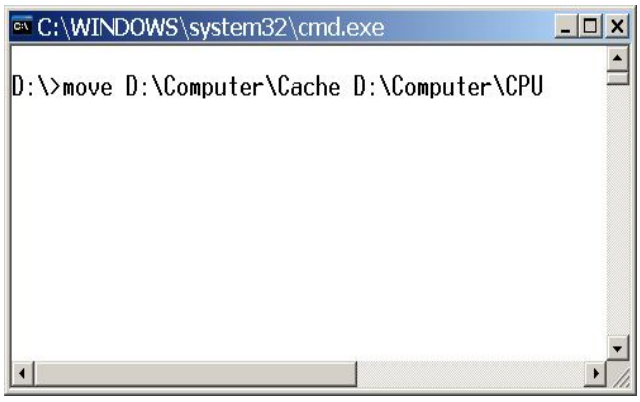
1. Sao chép thư mục **RAM** vào thư mục **Monitor**

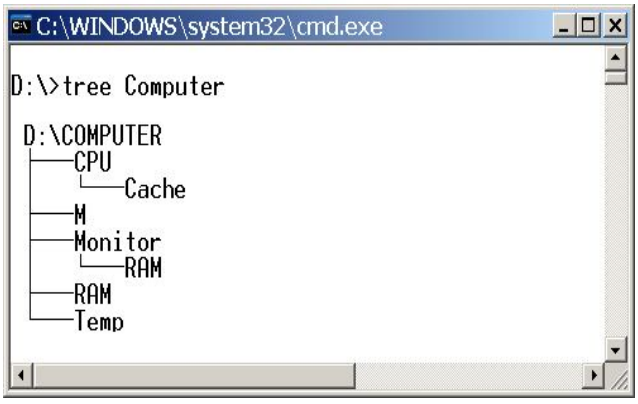
B1: ta dùng lệnh **xcopy** với cú pháp như sau: **xcopy** /E D:\Computer\RAM D:\Computer\Monitor\RAM	**B2:** xem lại cây thư mục bằng lệnh **tree**



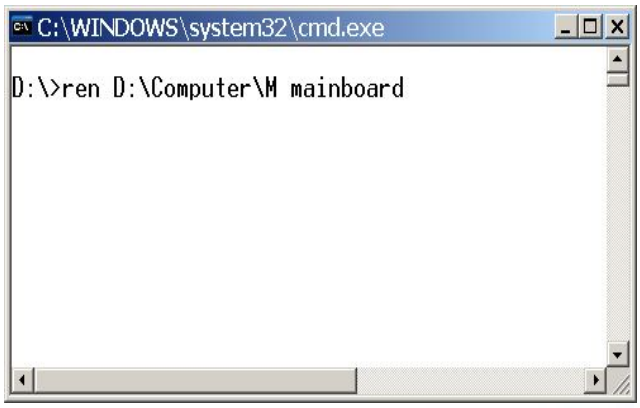
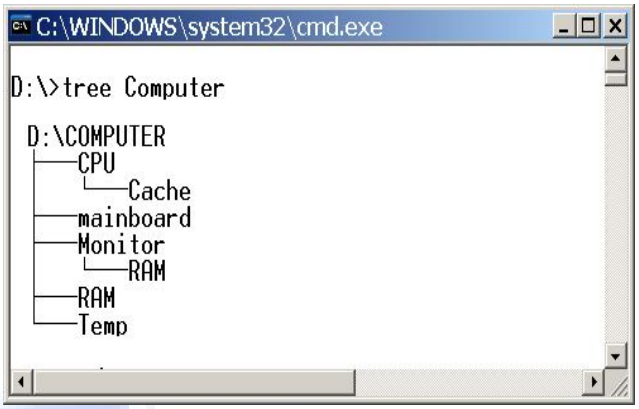
2. Di chuyển thư mục **Cache** vào thư mục **CPU**

B1: ta dùng lệnh **move** với cú pháp như sau: **move** D:\Computer\Cache D:\Computer\CPU	**B2:** xem lại cây thư mục bằng lệnh **tree**

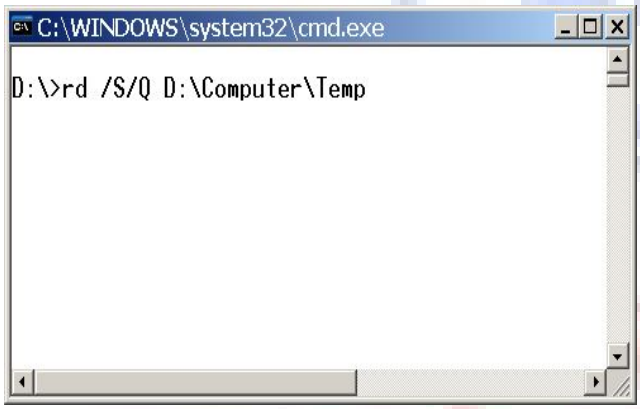
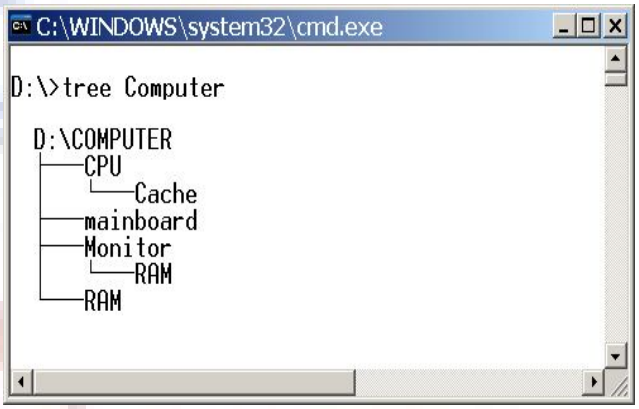




3. Đổi tên thư mục **M** thành **Mainboard**

B1: ta dùng lệnh **ren** với cú pháp như sau: **ren** D:\Computer\M Mainboard	**B2:** xem lại cây thư mục bằng lệnh **tree**
	

4. Xoá thư mục **TEMP**

B1: ta dùng lệnh **rd** với cú pháp như sau: **rd** /S /Q D:\Computer\Temp	**B2:** xem lại cây thư mục bằng lệnh **tree**
	

5. Tạo một tập tin **data.txt** trong thư mục **COMPUTER** với nội dung tuỳ ý và xem lại nội dung tập tin data.txt vừa tạo.

6. Sao chép tập tin **data.txt** sang thư mục **MAINBOARD**

7. Đổi tên tập tin **data.txt** trong thư mục **COMPUTER** thành **baitho.txt**

8. Di chuyển tập tin **data.txt** trong thư mục **MAINBOARD** sang thư mục **COMPUTER**

9. Xoá tập tin **data.txt** trong thư mục **COMPUTER**

10. Đặt thuộc tính **ReadOnly** cho tập tin **baitho.txt**

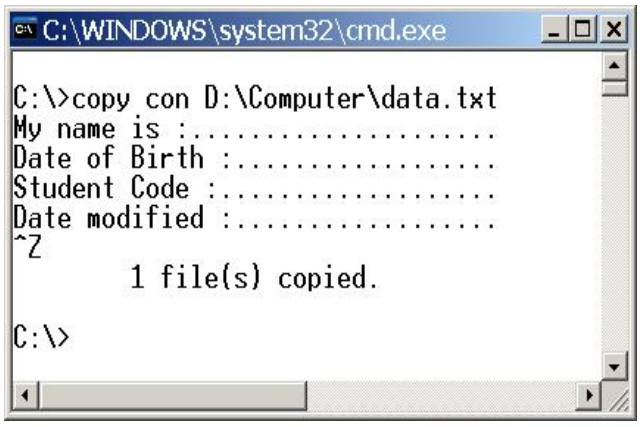
Hướng dẫn:

5. Tạo một tập tin **data.txt** trong thư mục **COMPUTER** với nội dung tuỳ ý và xem lại nội dung tập tin data.txt vừa tạo.

B1: ta dùng lệnh **copy con** với cú pháp như sau:

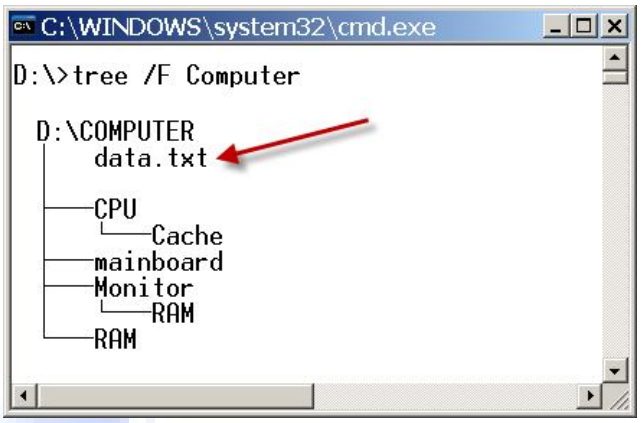
D:\> **copy con** D:\Computer\data.txt

Sau khi soạn xong nội dung ta nhấn **F6** để hoàn tất.



B2: xem lại cây thư mục bằng lệnh **tree** với cú pháp như sau:

D:\> **tree** /F Computer

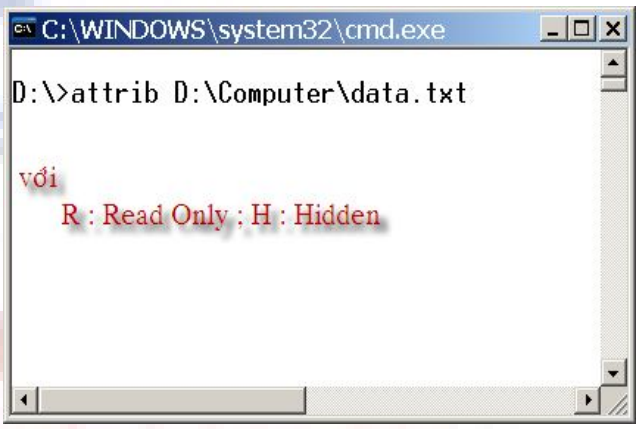


B3: để xem lại nội dung của tập tin đã tạo ta dùng lệnh **type** với cú pháp như sau:

D:\> **type** D:\Computer\data.txt

B4: xem thuộc tích của tập tin data.txt bằng lệnh **attrib** với cú pháp như sau:

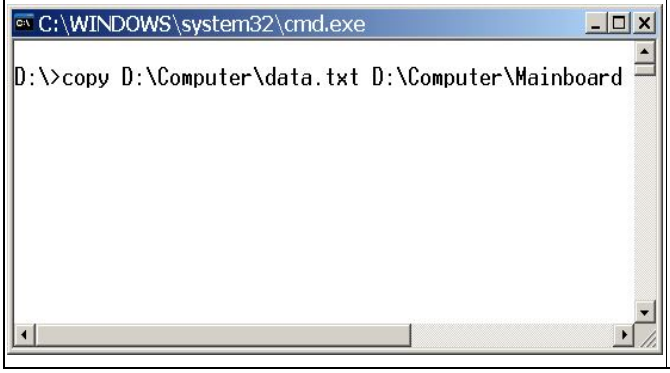
D:\> **attrib** D:\Computer\data.txt



6. Sao chép tập tin **data.txt** sang thư mục **MAINBOARD**

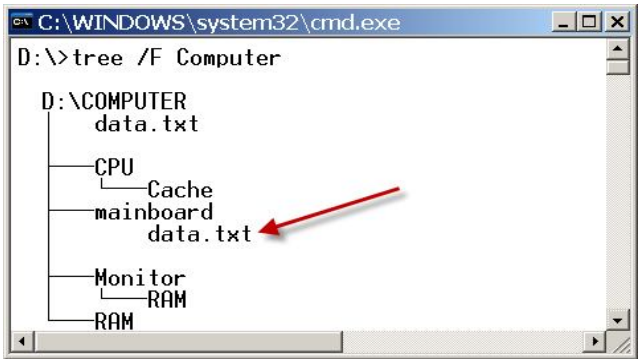
B1: để sao chép tập tin ta dùng lệnh **copy** với cú pháp như sau :

D:\>**copy** D:\Computer\data.txt D:\Computer\Mainboard

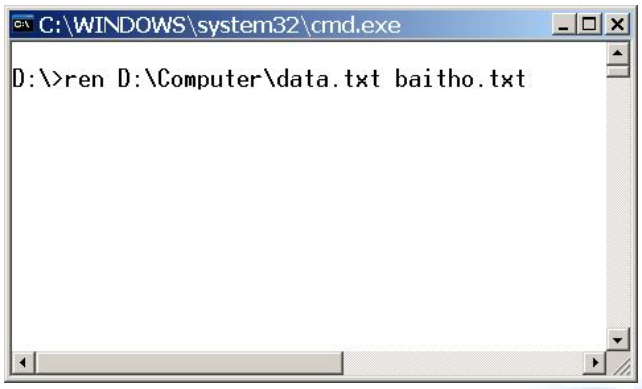
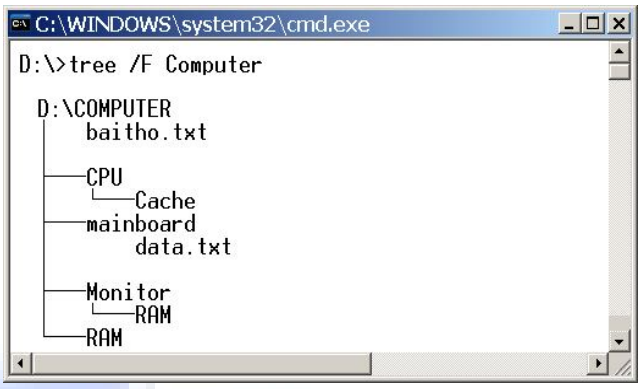


B2: xem lại cây thư mục bằng lệnh **tree** với cú pháp như sau:

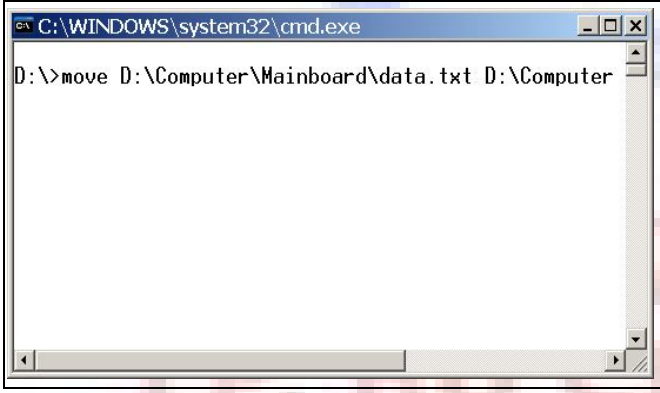
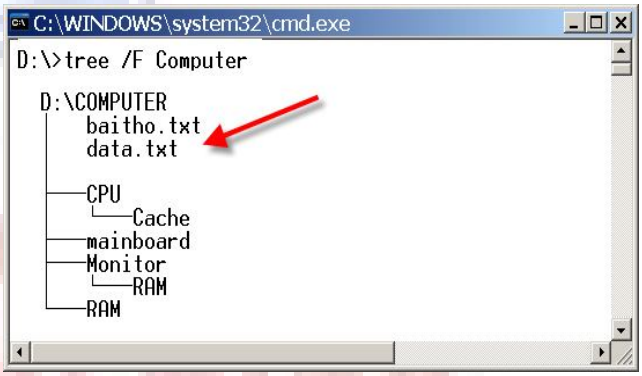
D:\> **tree /F** Computer



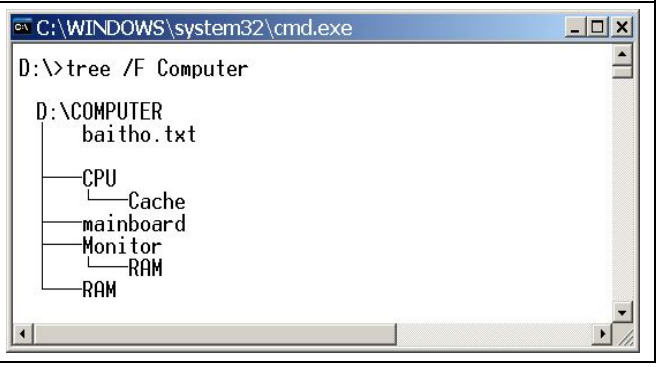
7. Đổi tên tập tin **data.txt** trong thư mục **COMPUTER** thành **baitho.txt**

B1: để đổi tên tập tin ta dùng lệnh **ren** với cú pháp như sau: D:\>**ren** D:\Computer\data.txt baitho.txt	**B2:** xem lại cây thư mục bằng lệnh **tree** với cú pháp như sau: D:\> **tree /F** Computer
	

8. Di chuyển tập tin **data.txt** trong thư mục **MAINBOARD** sang thư mục **COMPUTER**

B1: để di chuyển tập tin ta dùng lệnh **move** với cú pháp như sau: D:\> **move** D:\Computer\Mainboard\data.txt D:\Computer	**B2:** xem lại cây thư mục bằng lệnh **tree** với cú pháp như sau: D:\> **tree /F** Computer
	

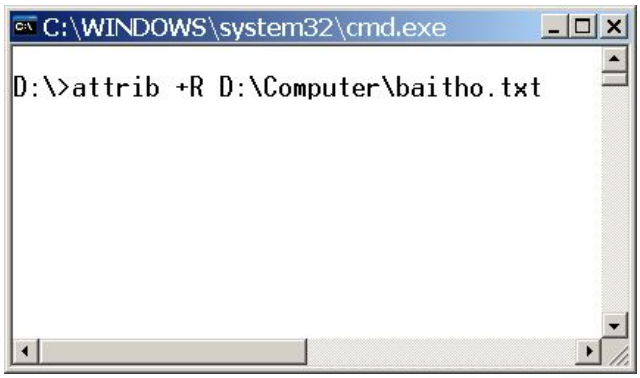
9. Xoá tập tin **data.txt** trong thư mục **COMPUTER**

B1: để xoá tập tin ta dùng lệnh **del** với cú pháp sau D:\> **del** D:\Computer\data.txt	**B2:** xem lại cây thư mục bằng lệnh **tree** D:\> **tree /F** Computer
	

10. Đặt thuộc tính **ReadOnly** cho tập tin **baitho.txt**

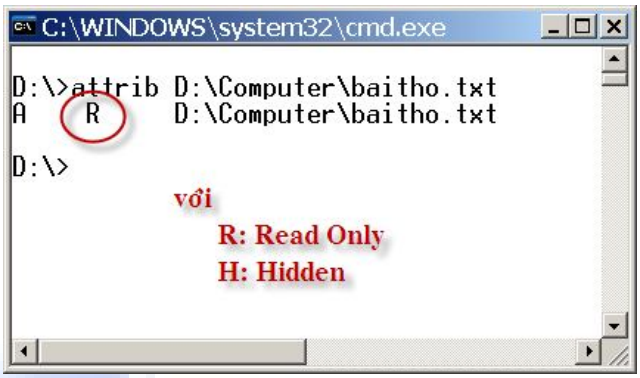
B1: để đặt thuộc tính cho một tập tin ta dùng lệnh **attrib** với cú pháp như sau:

D:\> **attrib** +R D:\Computer\baitho.txt



B2: xem lại thuộc tính của tập tin

D:\> **attrib** D:\Computer\baitho.txt



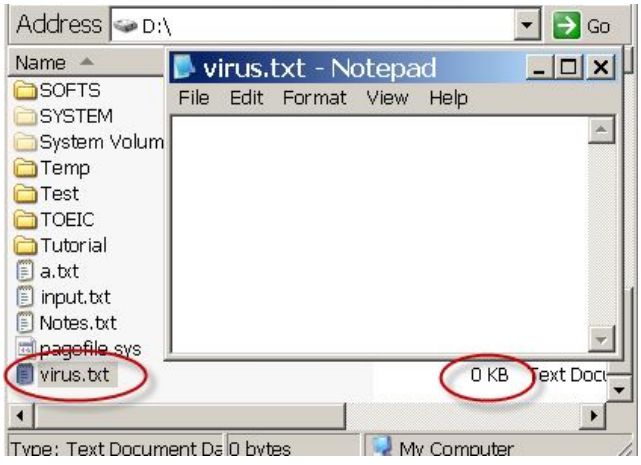
11. Tạo một tập tin che dấu nội dung của tập tin

B1: trong cửa sổ lệnh ta gõ lệnh như sau:

D:\>**notepad virus.txt:password**

B2: xuất hiện một hộp thoại chọn **Yes**, gõ nội dung tuỳ ý cho tập tin, nhấn **File → Save** để lưu trữ

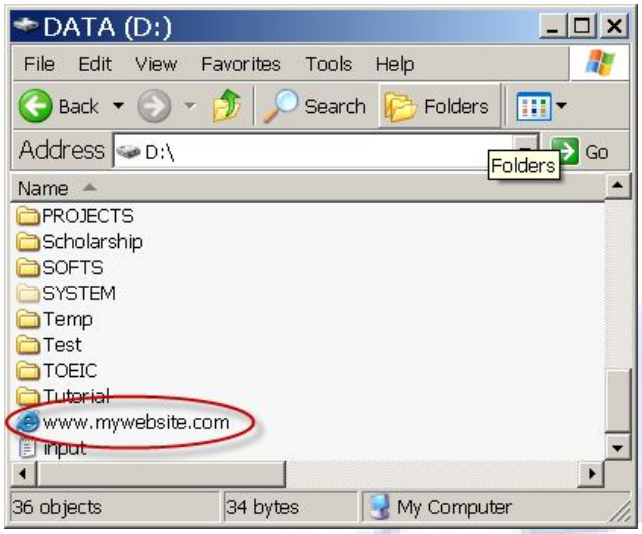
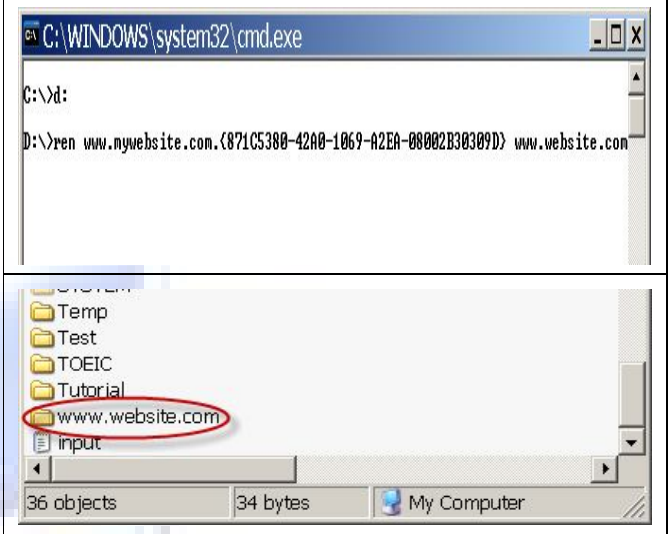
B3: mở Windows Explorer, double click vào tập tin **virus.txt** chúng ta sẽ không thấy nội dung gì của tập tin hiện ra.



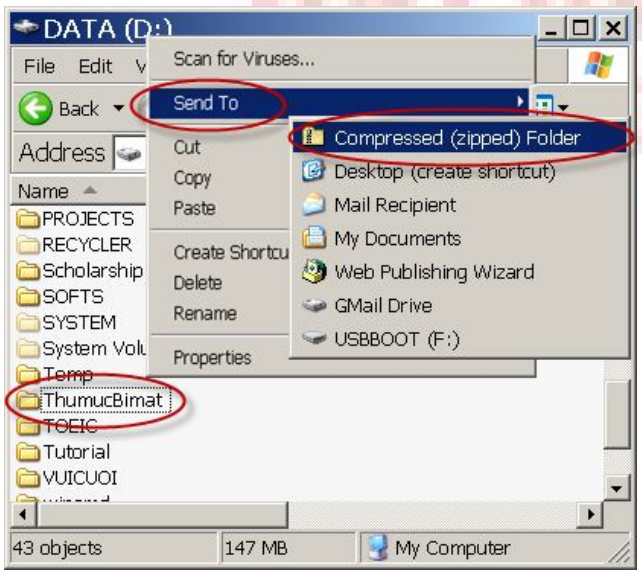
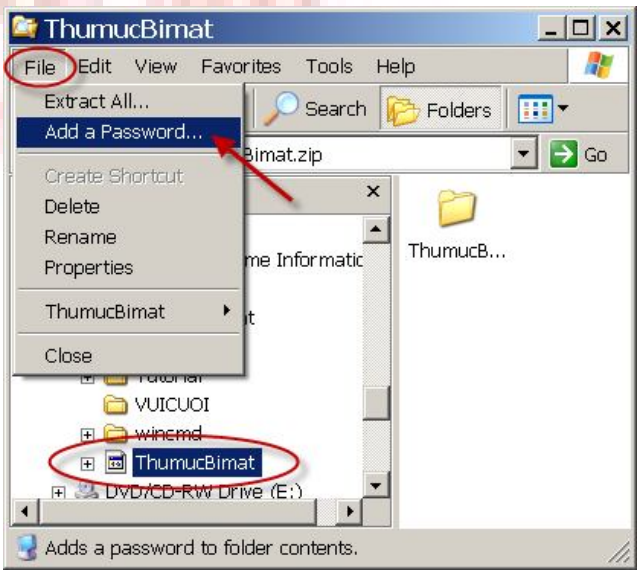
B4: để chỉnh sửa hoặc xem lại nội dung tập tin virus.txt ta gõ lệnh như B1

D:\>**notepad virus.txt:password**

12. Ẩn thư mục dưới một hình dạng khác

B1: tạo một thư mục tên **www.mywebsite.com**. Chép dữ liệu của bạn vào thư mục này. Đổi tên thư mục thành **www.mywebsite.com.{871C5380-42A0-1069-A2EA-08002B30309D}**	**B2:** Thử **double click** để mở thư mục **www.mywebsite.com**. Kết quả bạn sẽ thấy gì ? Bạn muốn thấy lại thư mục hãy gõ dòng lệnh sau : **D:\> ren www.mywebsite.com.{871C5380-42A0-1069-A2EA-08002B30309D} www.website.com**
	

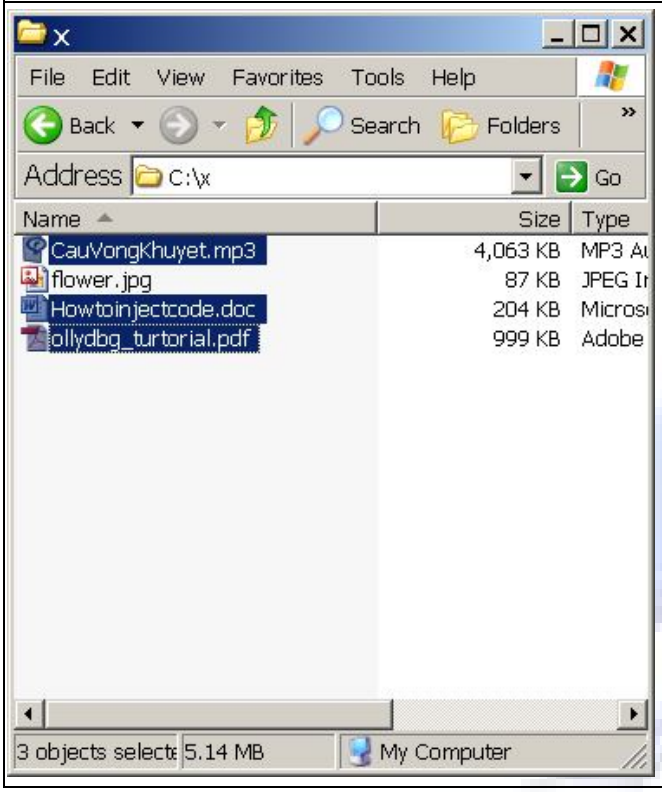
13. Đặt password bảo vệ cho thư mục

B1: Đầu tiên, bạn đưa các tập tin cần bảo vệ vào 1 thư mục, sau đó click phải vào thư mục đó, chọn **Send to ~> Compressed (zipped) Folder**. Chờ cho thư mục được nén lại với đuôi là **.zip**	**B2:** Bạn hãy mở thư mục nén này ra bằng chương trình nén của Windows. Bạn vào menu **File ~> Add a password**. Một hộp thoại hiện ra, bạn gõ pass vào ô **Password** và **Confirm Password** sau đó nhấn **OK**. Đến đây, thư mục của bạn đã được password bảo vệ.
	

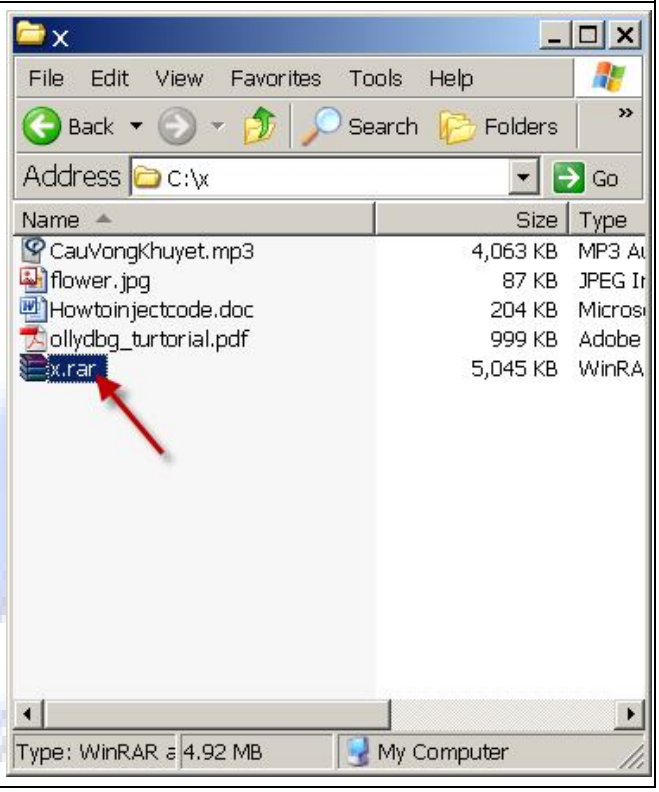
Chú ý : *nếu máy có cài chương trình nén file thì vào* ***Option*** *hoặc* ***Settings*** *của chương trình đó, tìm mục* ***File List*** *hoặc* ***Integration*** *và bỏ chọn ở phần* ***ZIP****.*

14. Đưa bí mật vào trong file

B1: chuẩn bị **tập tin hình ảnh định dạng jpg**, các tập tin cần ẩn.

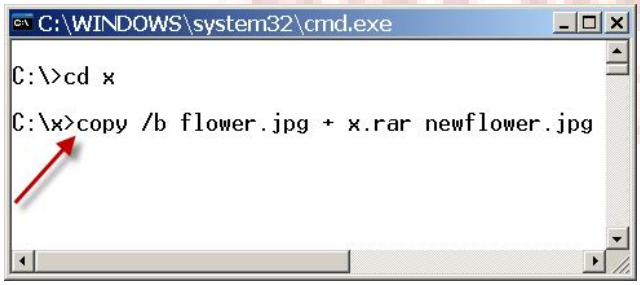


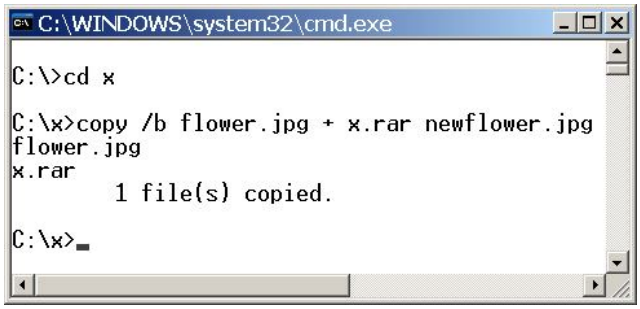
B2: Nén các tập tin cần ẩn thành một tập tin nén **.rar** hoặc **.zip**



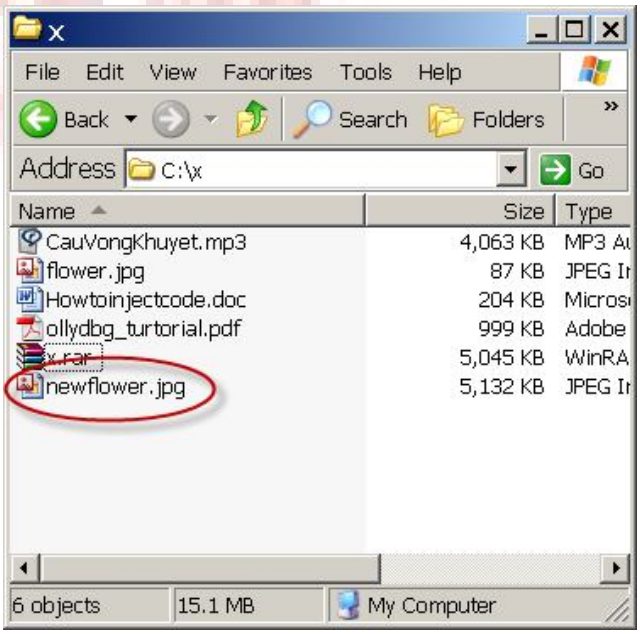
B3: Nhấn **Start ~> Run** gõ **cmd** để mở cửa sổ lệnh. Sau đó gõ lệnh sau:

Copy /b flower.jpg + x.rar newflower.jpg

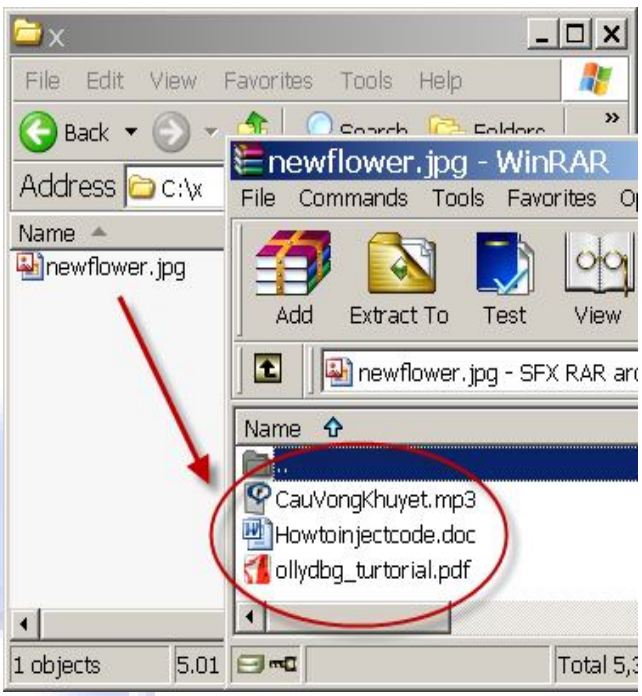




Kết quả là tập tin mới **newflower.jpg** được tạo ra có dung lượng bằng tổng các file (x.rar và flower.jpg). Như vậy là chúng ta đã hoàn thành công việc.



Lấy lại những tập tin ẩn trong file hình flower.jpg

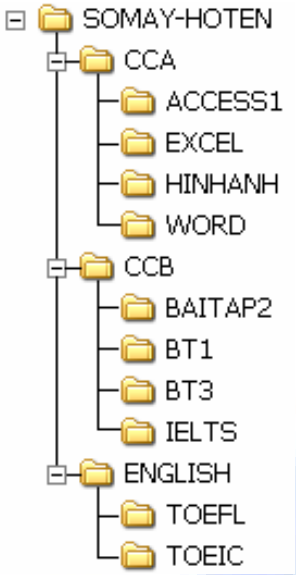
Nhận xét khi ta double click vào file newflower.jpg chỉ nhìn thấy đó là một tấm hình về một đoá hoa.	Để lấy lại file ẩn trong hình ta mở file hình này bằng chương trình giải nén winrar sẽ thấy các tập tin ẩn.
	

Thực hành : Tạo cây thư mục sau sau đó lần lượt xoá thư mục **Windows** và **Unix**

```
D:\
|_____Windows
|          |_____MSDOS
|          |
|          |_____Win98
|          |
|          |_____WinXP
|          |
|          |_____WinNT
|          |
|          |_____WinVista
|          |
|          |_____Windows7
|
|_____Unix
           |_____Solaris
           |
           |_____Linux
                      |
                      |_____Fedora
                      |
                      |_____RedHat
                      |
                      |_____Ubuntu
```

BÀI TẬP

❖ Tạo cây thư mục trên ổ đĩa **D:** có cấu trúc như sau:



Lưu ý: *Thư mục SOMAY-HOTEN học viên phải đổi lại là số máy và họ tên của mình*
Ví dụ : **007 – TranThiThuTrang**

❖ Anh (chị) hãy lần lượt thực hiện các thao tác sau:

1. Sao chép thư mục **BT1** vào thư mục **CCA**
2. Di chuyển thư mục **ACCESS1** vào thư mục **CCB**
3. Di chuyển thư mục **IELTS** vào thư mục **ENGLISH**
4. Lần lượt đổi tên các thư mục
   - Thư mục **BAITAP2** thành **ACCESS2**
   - Thư mục **BT3** thành **ACCESS3**
5. Xoá thư mục **HINHANH** trong thư mục **CCA**
6. Tạo tập tin **security.txt** với nội dung tuỳ ý và lưu vào thư mục **CCA**
7. Sao chép 2 tập tin **mspaint.exe** và **notepad.exe** trong thư mục **C:\WINDOWS\system32** vào **CCA**
8. Tạo thêm thư mục **BACKUP** trong thư mục **SOMAY-HOTEN**
9. Sao chép các tập tin trong thư mục **CCA** vào thư mục **BACKUP**
10. Đổi tên tập tin **mspaint.exe** trong thư mục **BACKUP** thành **hoasy.exe**

ÔN TẬP

- Các lệnh về thư mục
  1. Lệnh xem thư mục làm việc hiện hành: .....
  2. Chuyển thư mục làm việc đến thư mục `C:\Windows`: .....

3. Chuyển thư mục làm việc đến D:\: .....
4. Tạo thư mục `lab1`, trong đó tạo thêm thư mục `data`, `script`, `temp`:

.....
.....
.....
.....
.....

5. Chuyển thư mục làm việc sang thư mục `lab1` mới tạo và liệt kê thư mục:

.....
.....
.....

6. Xóa thư mục `temp` trong thư mục `lab1`: .....

- ## Các lệnh về file

7. Chuyển thư mục làm việc sang thư mục `lab1\data`. Dùng lệnh **copy con** tạo file `poem.txt` với nội dung là một bài thơ mình thích.
8. In ra màn hình nội dung file `poem.txt`: .....
9. Chép file `C:\Windows\greenstone.bmp` đến thư mục `lab1\data`: .....

.....
.....

- ## Sử dụng trợ giúp

10. Lệnh để xem trợ giúp của lệnh `copy`: .....
11. Lệnh xem trợ giúp của lệnh `move`: .....
12. Lệnh xem trợ giúp của `taskkill`: .....

- ## Chuyển hướng xuất nhập

13. Thực hiện lệnh liệt kê thư mục `C:\Windows` và kết quả được lưu vào file `result.txt`: .....

.....

14. Liệt kê nội dung thư mục `C:\Windows` và dừng lại sau khi in đầy màn hình: .....

.....

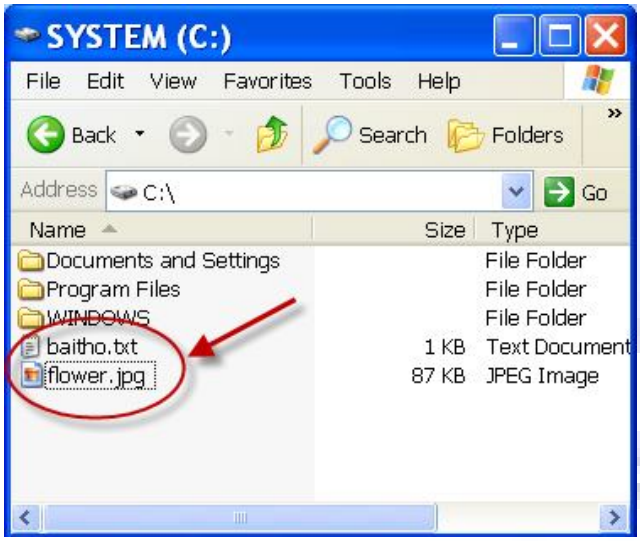
Lab 3

CHƯƠNG TRÌNH RESTORATION FILES

Chuẩn bị công cụ:

- Đĩa Hiren Boot.
- Một tập tin hình ảnh và tập tin văn bản

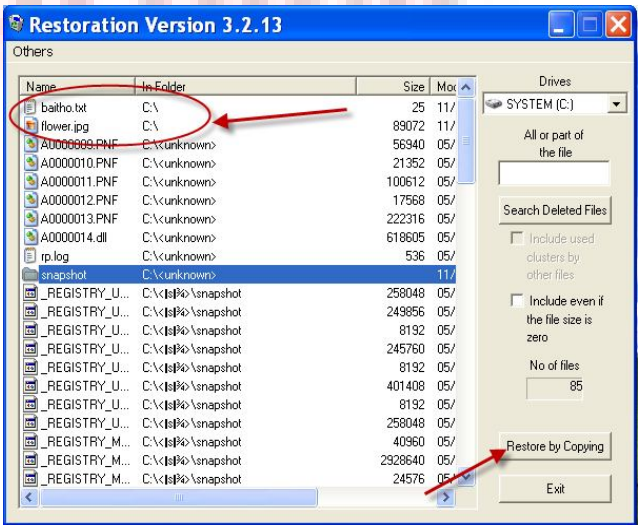
B1: tạo một tập tin văn bản với nội dung bất kỳ trên ổ đĩa C:, copy thêm một tập tin hình ảnh vào ổ đĩa C:



B2: xoá hẳn 2 file trên bằng cách chọn 2 file và nhấn tổ hợp phím **Shift** + **Delete**

B3: mở đĩa CD Hiren Boot và truy cập vào thư mục theo đường dẫn **E:\HBCD\WinTools**, chạy chương trình bằng cách nhấn đúp vào file **Restoration.bat**

B4: chọn 2 file cần phục hồi và nhấn nút lệnh **Restore by Copying**, sau đó chọn ổ đĩa D:\. Mở ổ đĩa D: và kiểm tra lại nội dung 2 file trên



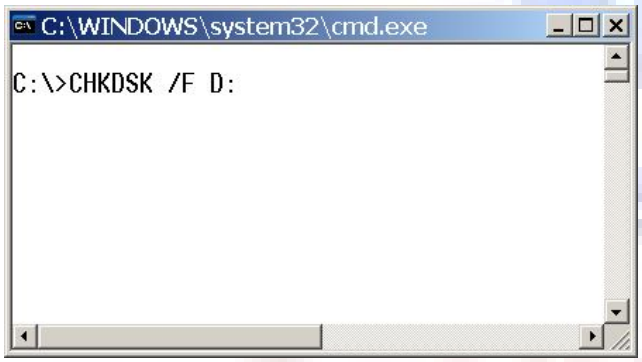
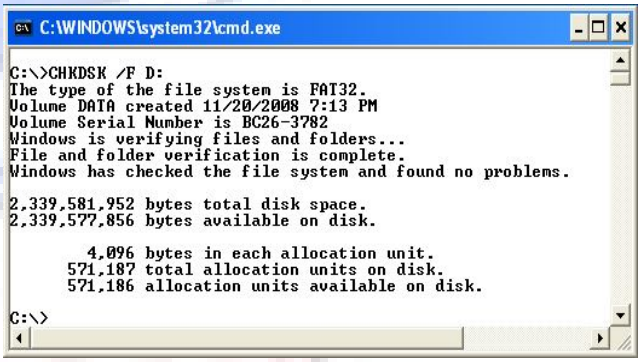
Lab 4
CÁC LỆNH VỀ HỆ THỐNG

4.1 CHKDSK

- Công dụng: tạo hoặc hiển thị báo cáo về tình trạng ổ đĩa dựa trên hệ thống tập tin đã được dùng. CHKDSK có thể liệt kê và sửa lỗi ổ cứng. Nếu CHKDSK không thể sử dụng được nó sẽ đề nghị bạn thực hiện lại thử sau khi đã khởi động lại máy. Để dùng được lệnh CHKDSK bạn phải thuộc nhóm quản trị (Administrator).
- Cú pháp:

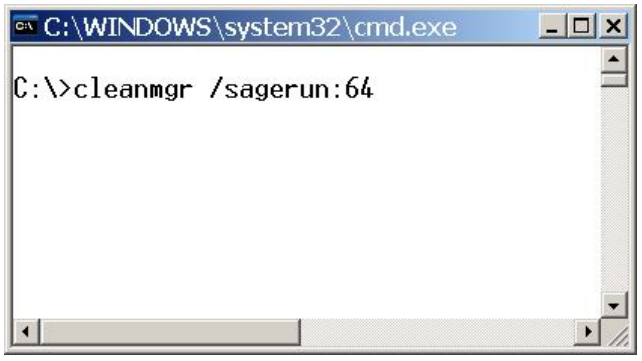
  CHKDSK [*drive***:**][[*path*] *filename*] [**/f**] [**/v**] [**/r**] [**/l**[**:***size*]] [**/x**]
- Ví dụ:

  Kiểm tra và sửa lỗi trên ổ đĩa D:

B1: ta gõ lệnh sau **CHKDSK** /F D: Tham số /F : fix errors on the disk	**B2:** màn hình thể hiện các thông tin diễn ra của tiến trình kiểm tra đĩa
	

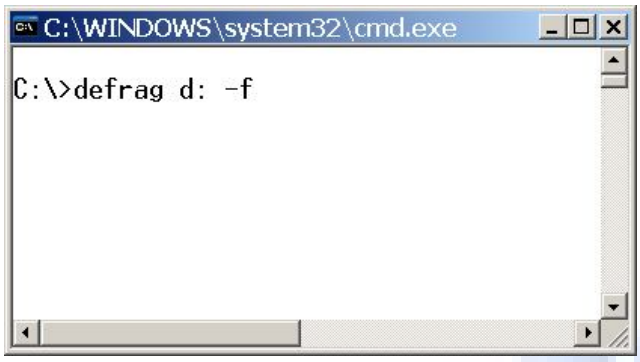
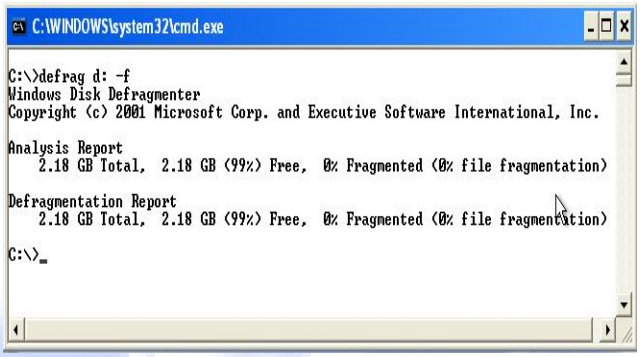
4.2 CLEANMGR

Lệnh này dùng để xoá rác các ổ đĩa, bạn cũng có thể vào **Start → Programs → Accessories → System Tools → Disk Cleanup**, tuy nhiên nếu dùng lệnh các bạn có thể lập lịch tự động để dọn dẹp rác cho máy.

B1: ta gõ lệnh **cleanmgr /sagerun : 64**	**B2:** tiến trình cleanup diễn ra trên tất cả ổ đĩa
	

4.3 DEFRAG

Lệnh này dùng để chống phân mảnh đĩa giúp tăng tốc độ truy xuất đĩa cứng. Ví dụ : thực hiện việc dồn đĩa trên đĩa D: ta làm các bước như sau :

B1: ta gõ lệnh **defrag D: -f**	**B2:** thông tin sau khi thực hiện xong lệnh defrag
	

4.4 FORMAT

- Công dụng: định dạng các loại đĩa được Windows XP chấp nhận. Bạn phải sử dụng tên đăng nhập thuộc nhóm quản trị nếu muốn định dạng một phân vùng ổ cứng.
- Cú pháp:

  format *volume* [**/fs:***file-system*] [**/v:***label*] [**/q**] [**/a:***unitsize*] [**/f:***size*] [**/t:***tracks* **/n:***sectors*] [**/c**] [**/x**] [**/1**] [**/4**] [**/8**]

  volume: tên ổ đĩa (đối đĩa mềm, đĩa gắn rời) hoặc tên phân vùng ổ cứng cần định dạng.

  /fs:*file-system* tên bảng phân hoạch tập tin FAT, FAT32 hoặc NTFS. Ổ đĩa mềm chỉ có thể dùng FAT.

  /v:label nhãn của ổ đĩa. Nếu bạn không nhập đối số /v hoặc không nhập nhãn đĩa, Windows 2000 sẽ yêu cầu bạn nhập nhãn đĩa sau khi định dạng xong. Nếu bạn định dạng cho nhiều đĩa cùng lúc thì tất cả các ổ đĩa đều có nhãn đĩa giống nhau. Không dùng **/v** với **/8**.

  /q Xóa bảng danh mục tập tin và thư mục gốc trước khi định dạng nhưng không quét để kiểm tra những vùng bị hỏng trên đĩa. Tham số này dùng đối với các ổ đĩa đã được định dạng trước đó, và nó sẽ thực hiện thao tác định dạng nhanh hơn thông thường.

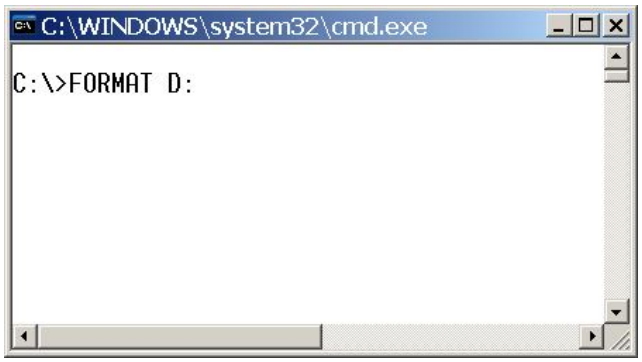
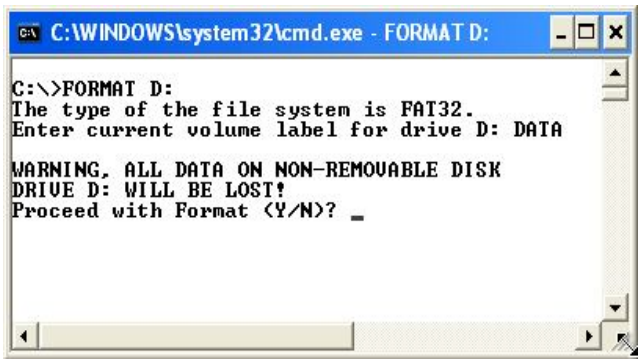
  /a:unitsize xác định độ lớn của một liên cung trên đĩa được định dạng. Nếu bạn không nhập dung lượng unitsize lệnh sẽ lấy một dung lượng thích hợp tùy vào dung lượng nhớ của đĩa được định dạng.

  /t:*tracks* số rãnh trên đĩa - có thể dùng /f thay tham số này.

  /n:*sectors* số cung trên một rãnh.

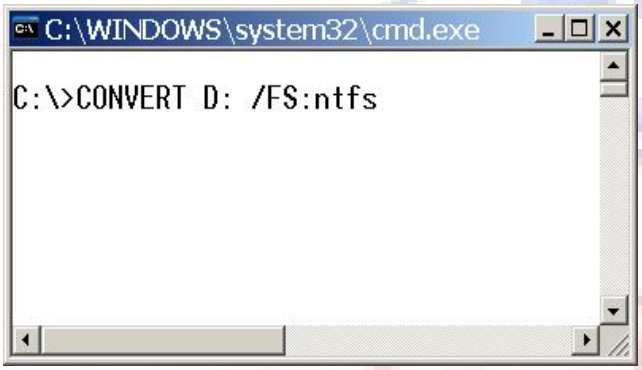
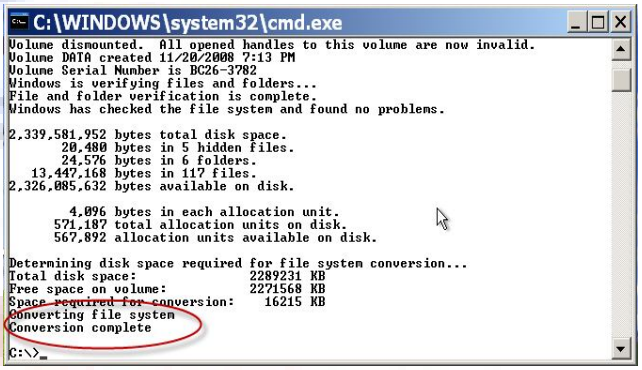
- *Ví dụ:*

  Định dạng lại ổ đĩa D: (*chú ý dữ liệu trên đĩa D sẽ bị mất hết*)

B1: ta gõ lệnh sau **FORMAT** D:	**B2:** khi dùng lệnh này dữ liệu trên đĩa sẽ bị mất, vì vậy chúng ta phải cẩn thận.
	

4.5 CONVERT

Lệnh này dùng để chuyển đổi hệ thống tập tin (file system) từ dạng **FAT32** sang **NTFS**. Chúng ta chuyển partition thành NTFS khi muốn nhu cầu bảo mật tốt hơn. *Ví dụ* : chuyển partition D từ FAT32 thành NTFS

B1: ta gõ lệnh như sau **CONVERT D: /FS:ntfs**	**B2:** diễn biến của tiến trình và khi kết thúc.
	

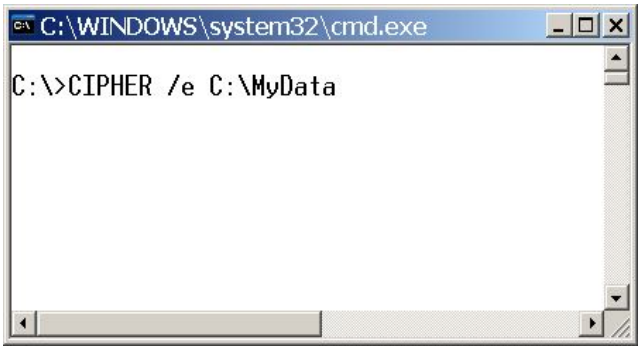
4.6 CIPHER

Lệnh này dùng để mã hoá và giải mã tập tin và thư mục. Điều kiện để mã hoá và giải mã là partition đó phải được định dạng hệ thống file là **NTFS**. Lệnh không có tác dụng với tập tin có thuộc tính *Read Only*

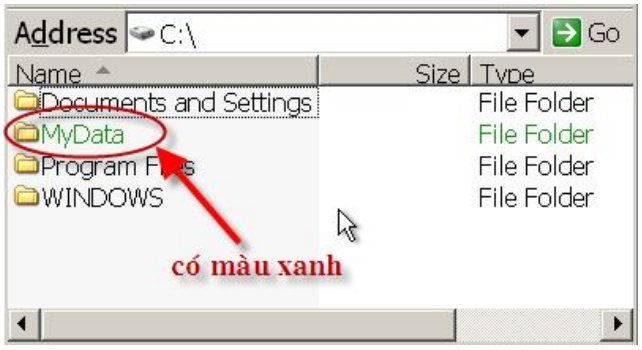
Ví dụ : mã hoá thư mục **C:\MyData**

B1: ta gõ lệnh **CIPHER /e C:\MyData**

Tham số /e : encrypt

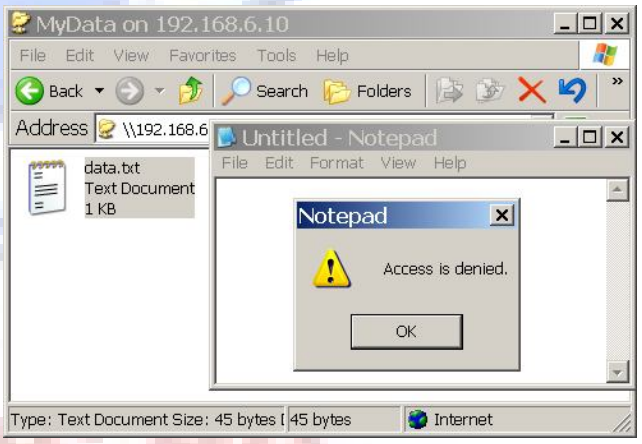


B2: thư mục mã hoá sẽ có **màu xanh** khác so với các thư mục khác. Và khi ta copy bất kỳ tập tin nào vào thư mục này thì các tập tin đó cũng sẽ được mã hoá.



B3: ta chia sẻ thư mục C:\MyData trên mạng bằng cách nhấn chuột phải vào MyData chọn **Sharing and Security…**

B4: từ máy khác truy cập vào thư mục MyData và mở thử xem nội dung của bất kỳ tập tin nào trong đó xem xảy ra hiện tượng gì ?



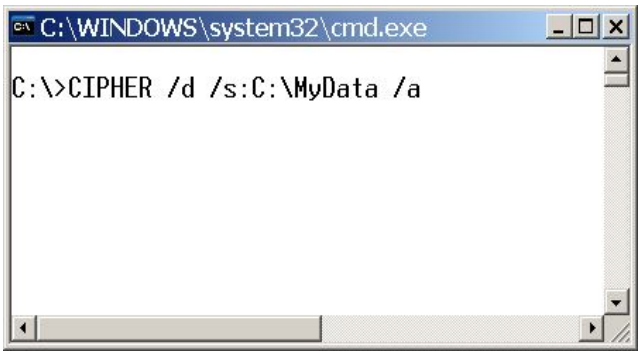
Ví dụ : giải mã lại thư mục **C:\MyData**

B1: ta gõ lệnh **CIPHER /d /s:C:\MyData /a**

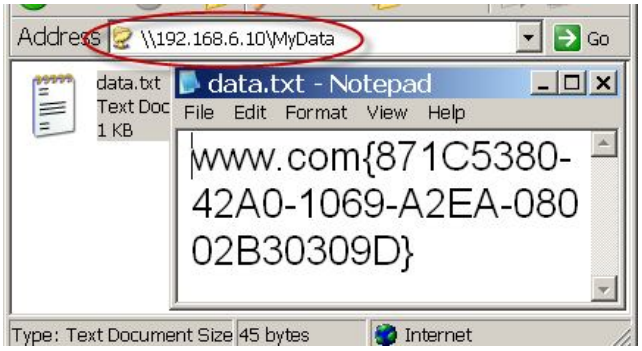
Tham số /d : decrypt

Tham số /s : duyệt tất cả tập tin và thư mục con

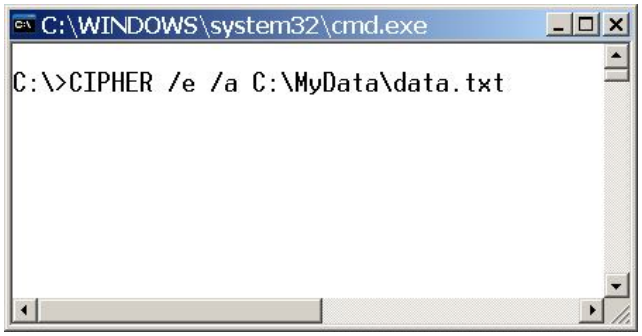
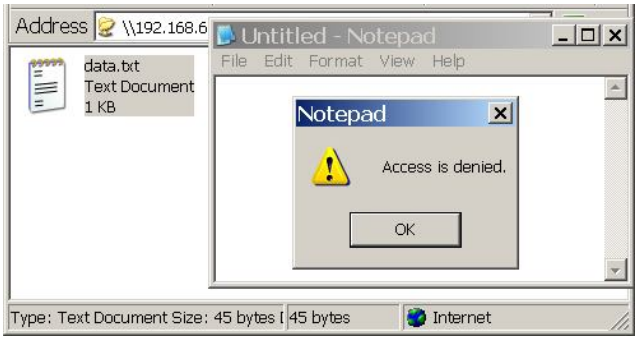
Tham số /a : thực thi decrypt cho cả tập tin và thư mục



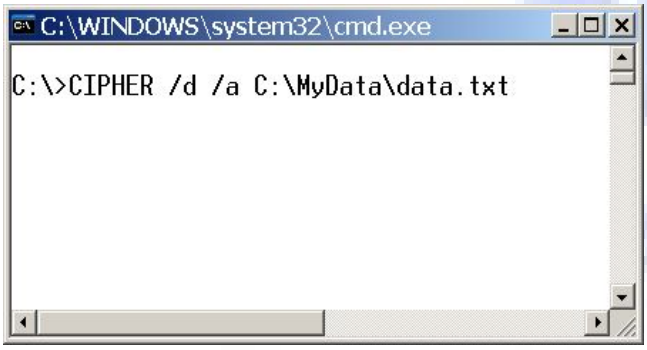
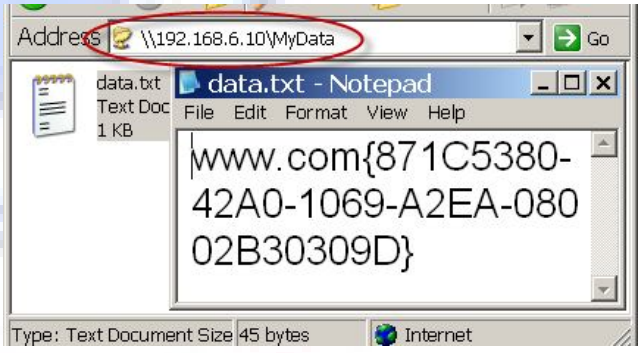
B2: thư mục C:\MyData trở lại bình thường và ta có thể truy xuất được các tập tin trong thư mục để xem nội dung của tập tin



Ví dụ : mã hoá tập tin **C:\MyData\data.txt**

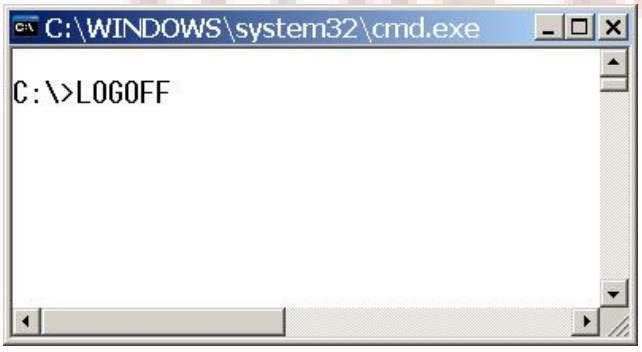
B1: ta gõ lệnh **CIPHER /e /a C:\MyData\data.txt**	**B2:** từ máy khác truy cập vào mở xem nội dung tập tin **MyData\data.txt**
	

Ví dụ : giải mã tập tin **C:\MyData\data.txt**

B1: ta gõ lệnh **CIPHER /d /a C:\MyData\data.txt**	**B2:** từ máy khác truy cập vào mở xem nội dung tập tin **MyData\data.txt**
	

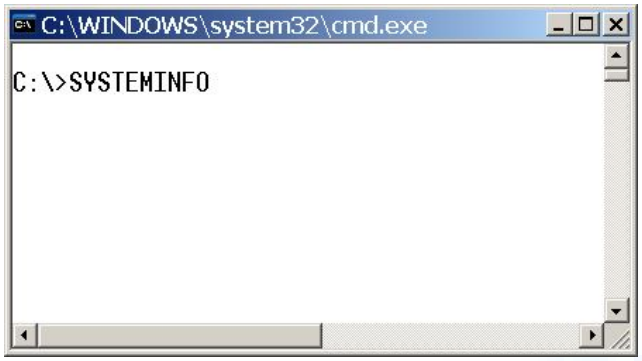
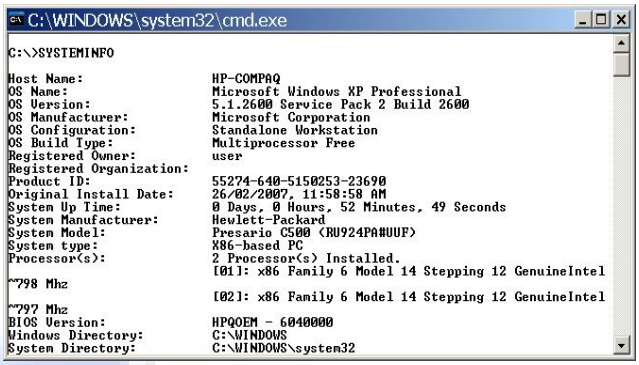
4.7 LOGOFF

Lệnh này dùng để thoát ra khỏi session đang làm việc, tương đương với thao tác Logoff khi ta nhấn Start

B1: ta gõ lệnh sau **LOGOFF**	**B2:** kết quả là màn hình đăng nhập xuất hiện
	

4.8 SYSTEMINFO

Lệnh này dùng để xem thông tin hệ thống gồm thông tin về hệ điều hành, CPU, RAM,…

B1: ta gõ lệnh sau **SYSTEMINFO**	**B2:** màn hình thể hiện các thông tin
	

4.9 SHUTDOWN

- Công dụng: shutdown máy
- Cú pháp:

  shutdown [**-s** | **-r**] [**-m** \\computername] [**-t** **xx**]

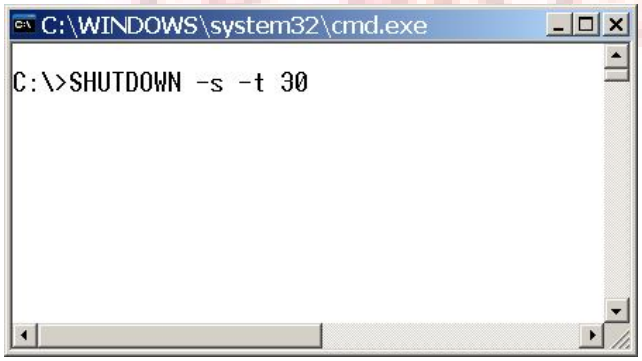
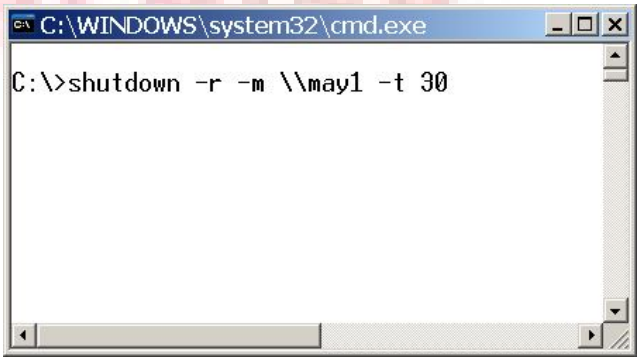
  -s : shutdown máy

  -r : shutdown và restart lại máy

  -m **\\computername** : shutdown máy từ xa

  -t **xx** : thiết lập thời gian trước khi shutdown xx giây

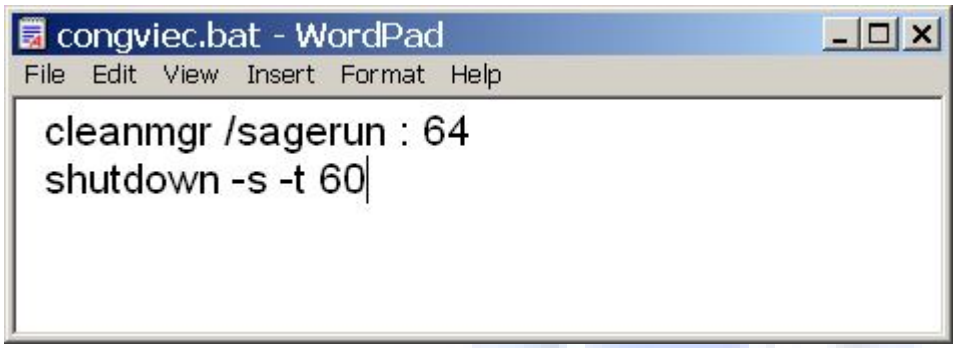
- Ví dụ:

Shutdown máy sau 30 giây	Điều khiển Shutdown máy tính khác từ xa
	

4.10 Tạo file bat thực thi nhiều công việc

Ví dụ : tạo tập tin **congviec.bat** thực hiện những công việc sau (lưu ý phần mở rộng của tập tin là **.bat**)

1. thực thi công việc dọn dẹp rác các ổ đĩa
2. shutdown máy sau 60 giây



Để thực thi ta double click và tập tin congviec.bat là xong

4.11 Tạo file bat chứa "mã độc"

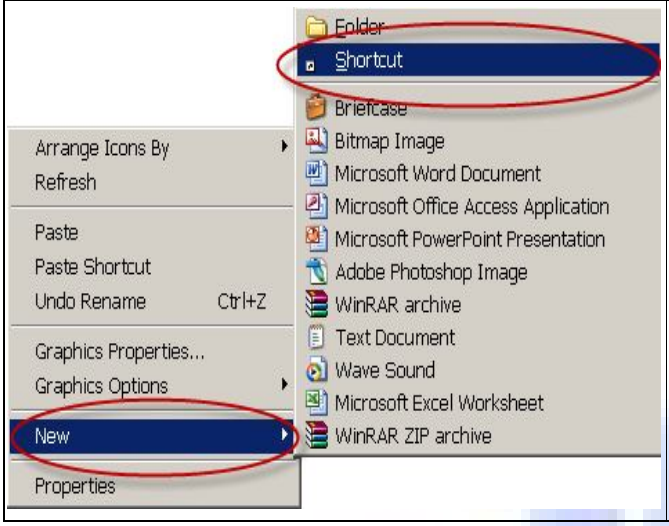
Ví dụ : tạo tập tin **virus.bat** với đoạn mã độc hại sau (lưu ý phần mở rộng của tập tin là **.bat**)

Chú ý : *Đoạn mã lệnh trên sẽ xoá các tập tin quan trọng của hệ thống Windows và gây cho hệ điều hành Windows không thể khởi động. Đoạn code này chỉ mang tính chất tham khảo phục vụ trong việc học tập, các sinh viên* ***được cảnh báo không sử dụng trong việc khác.***

4.12 Shortcut Internet Explorer "giả mạo"

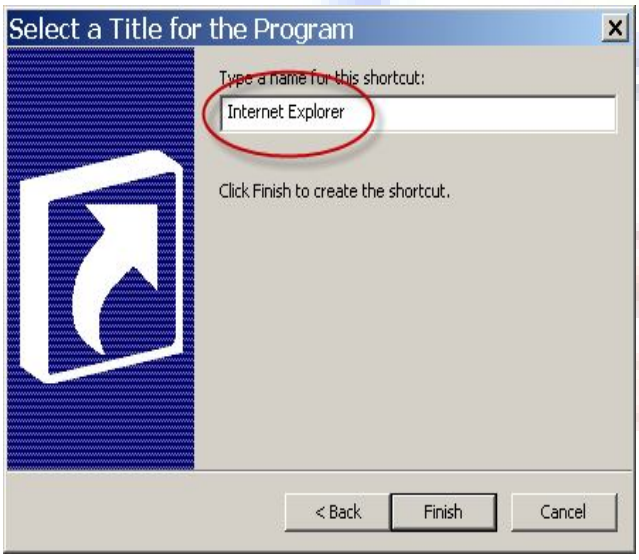
Tạo shortcut giả mạo chứa mã độc gây nhầm lẫn cho người dùng

B1: Tạo shortcut trên màn hình nền Desktop bằng cách **nhấn chuột phải** vào màn hình Desktop ~> chọn **New** ~> chọn **Shortcut**



B2: Đánh dòng lệnh **shutdown** vào ô **Type the location of the item** (*chắc các bạn cũng đoán được mã lệnh này dùng làm gì*). Nhấn **Next** tiếp tục.

B3: Đặt tên cho shortcut là **Internet Explorer**. Nhấn **Finish** hoàn tất.



B4: Thay đổi biểu tượng shortcut bằng cách nhấn chuột phải shortcut ~> **Properties** ~> **Change Icon**

Hãy thử nhấn vào biểu tượng IE xem xảy ra chuyện gì ?

Chú ý : *Các thao tác được hướng dẫn chỉ mang tính chất tham khảo phục vụ trong việc học tập nhằm giúp sinh viên có ý thức cảnh giác trước những nguy cơ tiềm ẩn, các sinh viên* ***được cảnh báo không sử dụng trong việc khác****.*

Lab 5
ĐO LƯỜNG CÁC THAO TÁC VỀ I/O

1. Mục tiêu

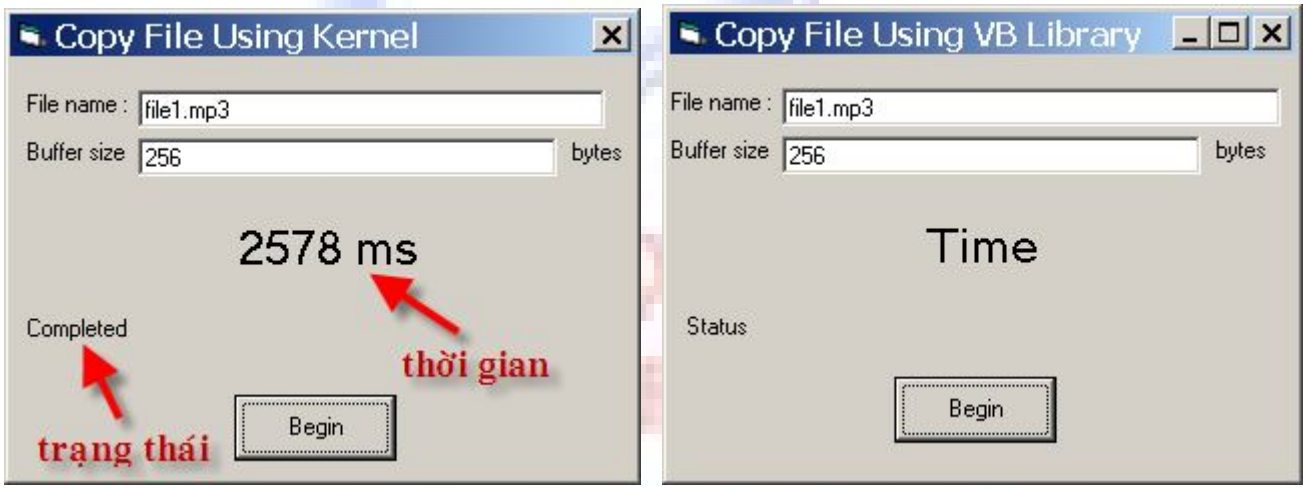
Đánh giá hiệu suất của hệ thống khi xử lý những file dữ liệu lớn. Sinh viên thực hiện viết một chương trình Visual Basic trên Windows đọc nội dung một file (có kích thước lớn), xử lý và sau đó ghi vào một file mới. Chương trình sẽ đo thời gian chạy chương trình khi sử dụng trực tiếp system call của Windows và khi sử dụng thư viện hàm của Visual Basic.

2. Kiến thức cần biết

- Lập trình Visual Basic trên Windows
- Kỹ thuật lập trình đọc ghi file
- Kỹ thuật thao tác với time trên Windows

3. Yêu cầu chi tiết

Viết 2 chương trình thực hiện việc đọc file và ghi ra 1 file mới, 1 chương trình sử dụng trực tiếp system call dùng các hàm API của Windows, một chương trình khác sử dụng các hàm xây dựng sẵn của Visual Basic (Open, Get, Put, Close) để đọc và ghi dữ liệu. Chương trình có khai báo kích thước buffer sử dụng để xử lý. Giao diện của chương trình như sau:



Trong chương trình viết đoạn lệnh dùng để đo thời gian thực thi của 2 chương trình trên, với kích thước buffer thay đổi từ **256B**, **512B**, **1KB**, **2KB**, **4KB**, **8KB**, **16KB**, **32KB**, **64KB**, **128KB**, **256KB**, **512KB**, **1MB**, **2MB**, **4MB**. Thời gian có thể được lấy bằng hàm GetTickCount (xem thêm trong mã nguồn ví dụ). và được tính chính xác đến miligiây (ms).

Tham khảo source code mẫu đính kèm (chưa hoàn chỉnh).

4. Kết quả:

Kết quả thu được, tóm tắt dưới dạng bảng như sau

Buffer \ Implementation	Exec time - System call (ms)				Exec time – Visual Basic library (ms)			
	Lần 1	**Lần 2**	**Lần 3**	**Trung bình**	**Lần 1**	**Lần 2**	**Lần 3**	**Trung bình**
256B								
512B								
1KB								
2KB								
4KB								
8KB								
16KB								
32KB								
64KB								
128KB								
256KB								
512KB								
1MB								
2MB								
4MB								

Lưu ý: mỗi lần chạy có thể khác nhau nên sinh viên cần chạy 3 lần và ghi nhận lại trong bảng kết quả.

Nhận xét, đánh giá về thời gian chạy của 2 chương trình khi tăng kích thước buffer.

.....

.....

.....

.....

Vài mô tả về platform sử dụng để chạy chương trình (tốc độ CPU, kích thước bộ nhớ)

.....

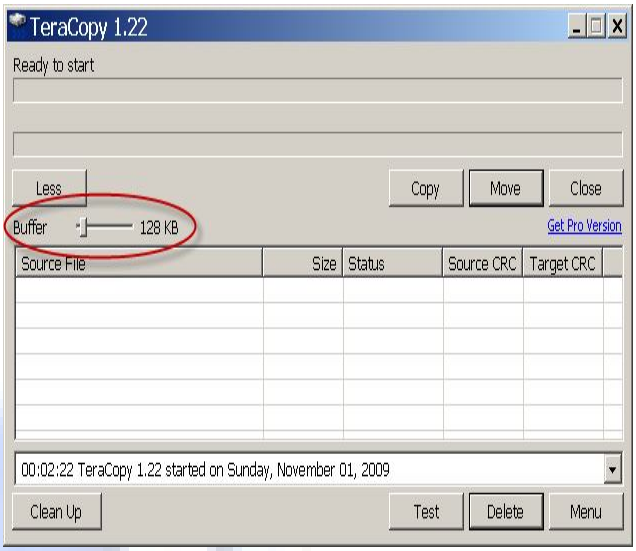
.....

.....

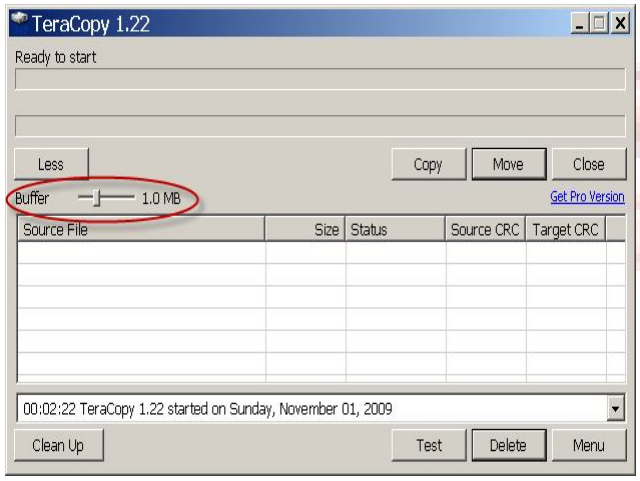
5. Sử dụng Tera Copy

B1: Cài đặt chương trình Tera Copy.

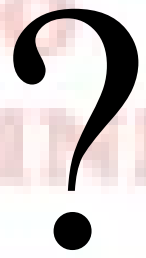
B2: Chọn buffer size là **128K**, rồi copy **file1.mpg**



B3: Thử **tăng buffer size** và copy lại **file1.mpg**. Sau đó thử tăng tiếp buffer size và thực hiện lại thao tác copy **file1.mpg**. Nhận xét ?



Câu hỏi : Hãy thử suy nghĩ và so sánh tốc độ giữa hai thao tác **Copy** và **Cut** ? Thao tác nào sẽ nhanh hơn ? Lý giải hiện tượng này.

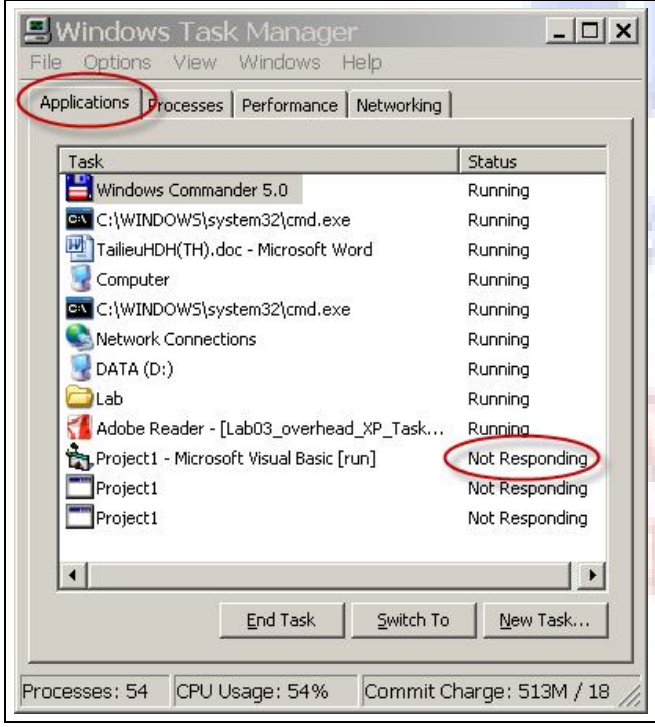
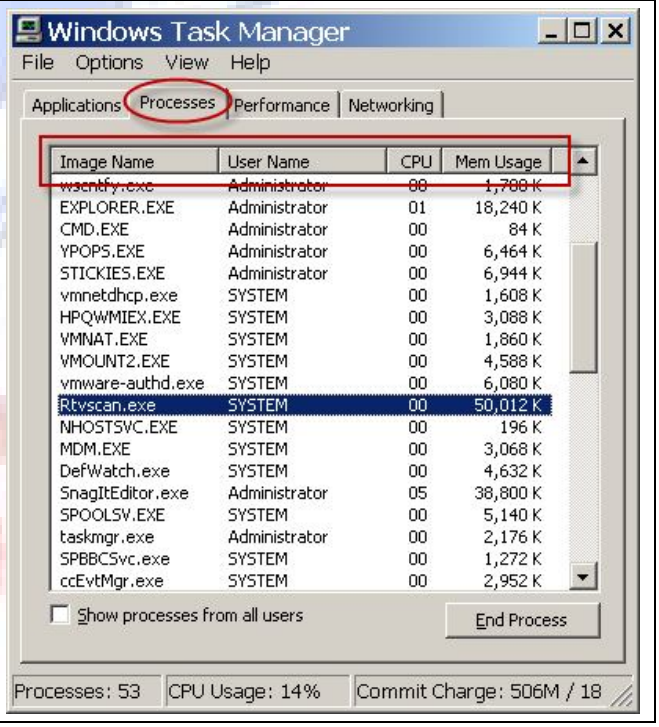


Lab 6
THEO DÕI CÁC TIẾN TRÌNH VÀ BỘ NHỚ

6.1 Đánh giá hiệu suất máy tính với công cụ Task Manager

Khởi động **Task Manager** bằng những cách sau

- Nhấn tổ hợp phím **Ctrl** + **Shift** + **ESC**.
- Nhấn **Ctrl** + **Alt** + **Del**, sau đó chọn nút **Task Manager**
- Nhấn chuột phải thanh **Taskbar** và chọn mục **Task Manager**

☞ Tab **Applications** liệt kê tất cả các ứng dụng đang chạy. Nếu một ứng dụng nào đó bị treo chúng ta sẽ thấy status là "**Not Responding**", ta chọn ứng dụng bị treo và nhấn **End Task**	☞ Tab **Processes** liệt kê tất cả các tiến trình đang chạy gồm có các thông tin về tiến trình như: tên tiến trình (image name), người dùng nào đang chạy tiến trình (username), tỷ lệ sử dụng CPU (CPU), dung lượng bộ nhớ sử dụng cho tiến trình (Mem usage)
	

a. Có bao nhiêu ứng dụng đang chạy trên máy của bạn :.....

.....

b. Liệt kê những tiến trình của hệ thống (có username là System) chạy trên máy của bạn :.....

.....

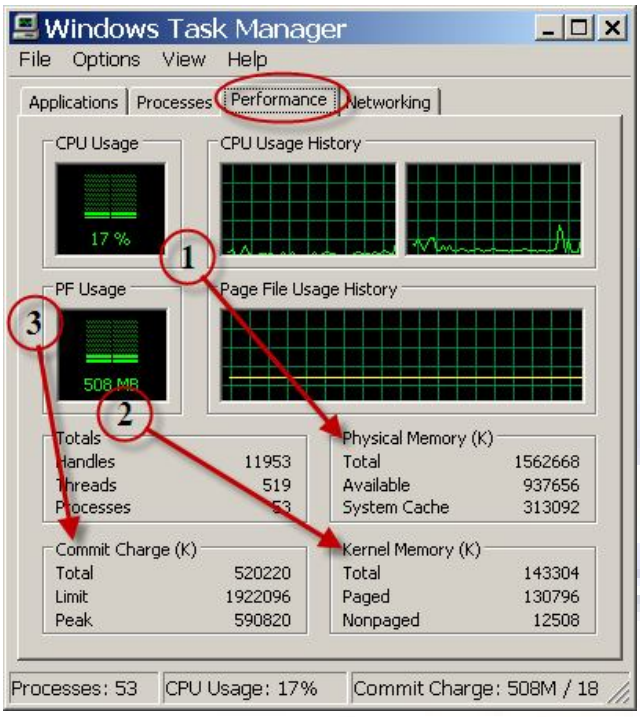
.....

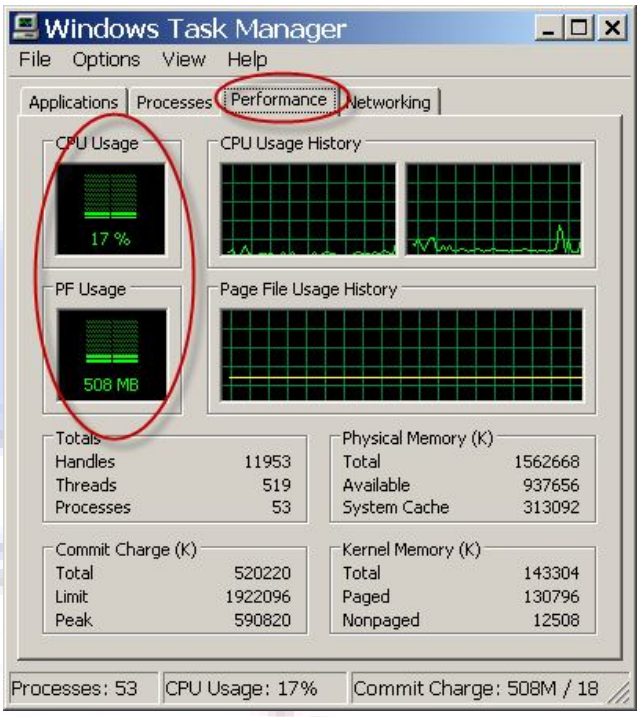
.....

c. Số lượng tiến trình đang chạy trên máy của bạn là bao nhiêu ?.....

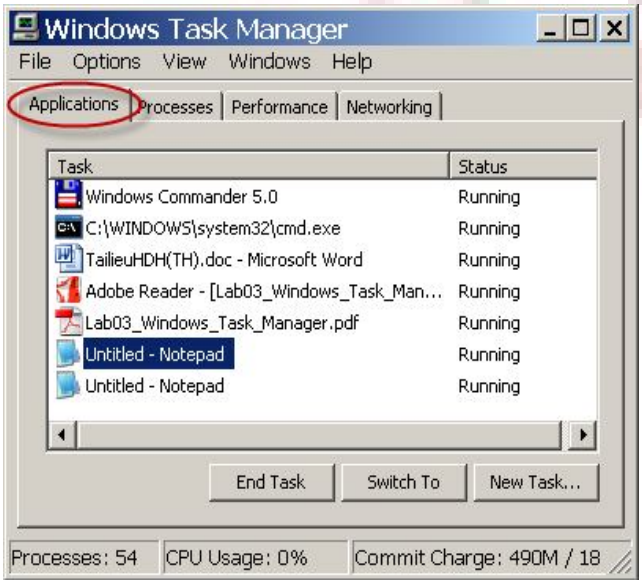
.....

☞ Tab **Perfomance** cho biết thông tin thống kê việc sử dụng CPU và RAM ❶ : dung lượng bộ nhớ vật lý (RAM) ❷ : dung lượng bộ nhớ mà hệ điều hành dùng ❸ : tổng dung lượng hiện tại mà các process đang dùng (bao gồm cả RAM và virtual RAM)	Nếu các tiến trình đang trong quá trình chạy và xử lý thì sẽ sử dụng CPU và RAM vì thế ta thấy số liệu trên các biểu đồ thống kê thay đổi liên tục. Các tiến trình mới sinh ra cũng sẽ làm cho việc sử dụng dung lượng RAM tăng theo. Nếu dung lượng RAM vật lý quá ít sẽ khiến hệ điều hành phải dùng tới virtual memory nhiều, điều này sẽ khiến việc sử dụng CPU tăng lên.





Thực hành :



1. Khởi động chương trình notepad làm 2 lần bằng cách vào **Start → Run →** gõ **notepad → OK**. Sau đó kiểm tra lại trong tab **Applications** đã thấy gì ?. Chọn một trong ứng dụng Notepad đang chạy và nhấn "**Switch To**", mô tả chuyện gì xảy ra, chức năng của Switch To ?

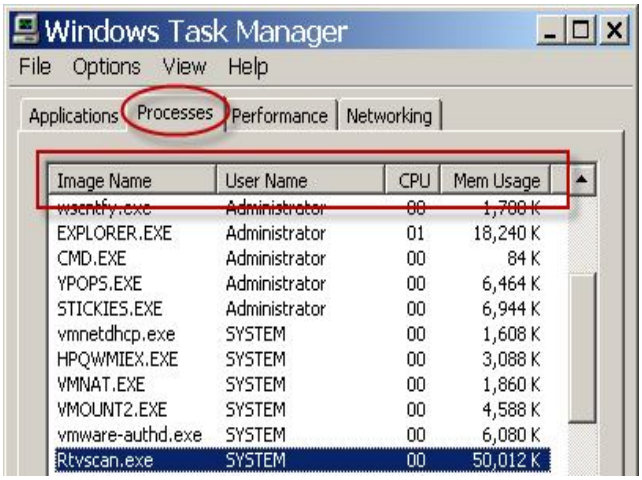
Sau đó "**End Task**", mô tả chuyện gì xảy ra ?

Cuối cùng, khởi động ứng dụng **WordPad** dùng chức năng "**New Task**…"

.....

.....

.....



2. Trong tab **Processes**, hãy xác định tiến trình nào chiếm dụng CPU nhiều nhất và cho biết nhiệm vụ của tiến trình này là gì ?

.....

.....

.....

3. Tạo một tập tin **virus.vbs** (vbscript) có mã lệnh như sau (có thể dùng Notepad để soạn)

Dim a

While (true)

a = a + 1

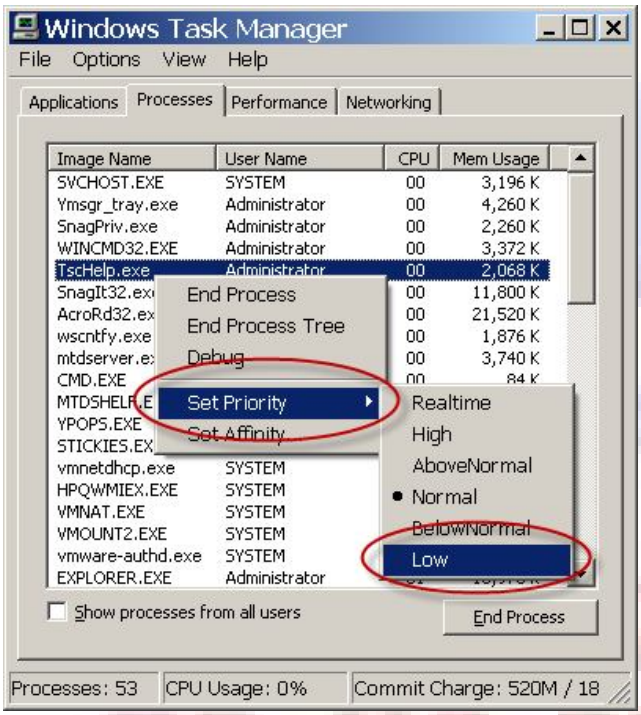
Wend

Double click vào script để chạy và theo dõi tiến trình chạy tên gì ? Ghi thông số về tỷ lệ mà tiến trình này chiếm dụng CPU.

.....

.....

.....

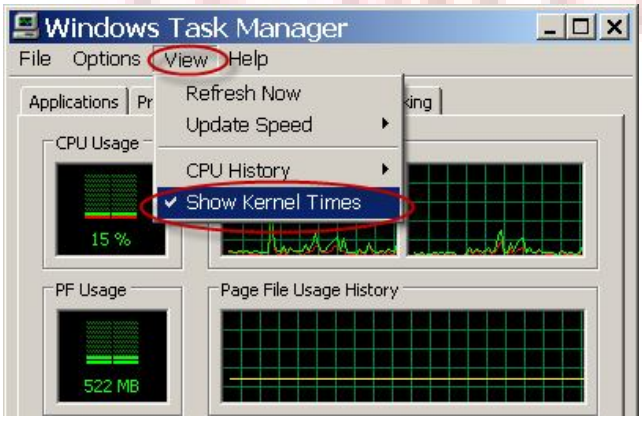


Cuối cùng chọn process này, **nhấn chuột phải** → chọn **Set Priority** → thiết lập mức **Low**. Nhận xét về kết quả như thế nào ?

.....

.....

.....



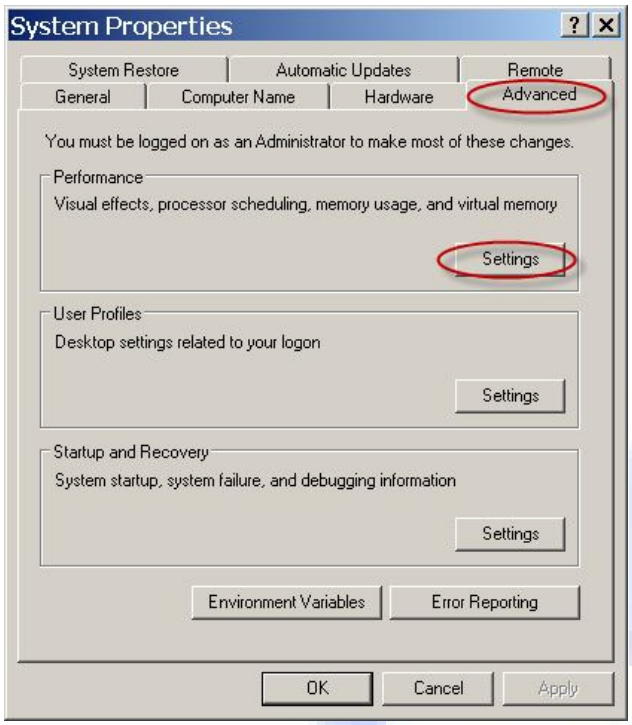
4. Chọn tab **Perfomance**, vào menu **View** → chọn **Show Kernel Times** để xem thêm thông tin về quá trình làm việc của hệ điều hành (đường biểu diễn màu đỏ trên biểu đồ). Double click trên biểu đồ để phóng lớn lên.

Nhấn giữ cửa sổ Task Manager, và di chuyển cửa sổ qua lại thật nhanh trên màn hình, quan sát sự thay đổi của CPU trên biểu đồ. Tại sao lại xảy ra sự thay đổi này ?

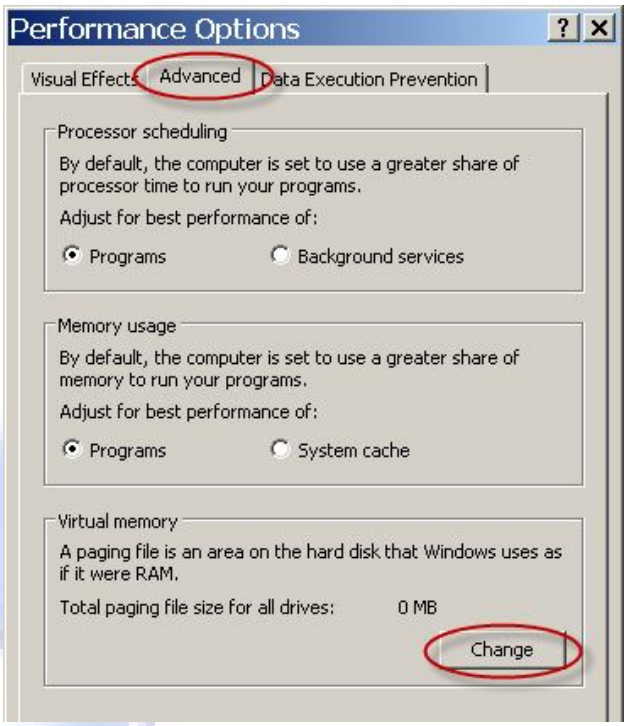
Ghi nhận các thông tin về **Physical Memory** và **Kernel Memory**. Tính toán xem **bao nhiêu phần trăm bộ nhớ** được sử dụng bởi **kernel**. Sau đó khởi động ứng dụng **WordPad 10 lần** và mô tả những thay đổi với thông tin về **Kernel Memory**.

6.2 Quản lý Bộ nhớ ảo

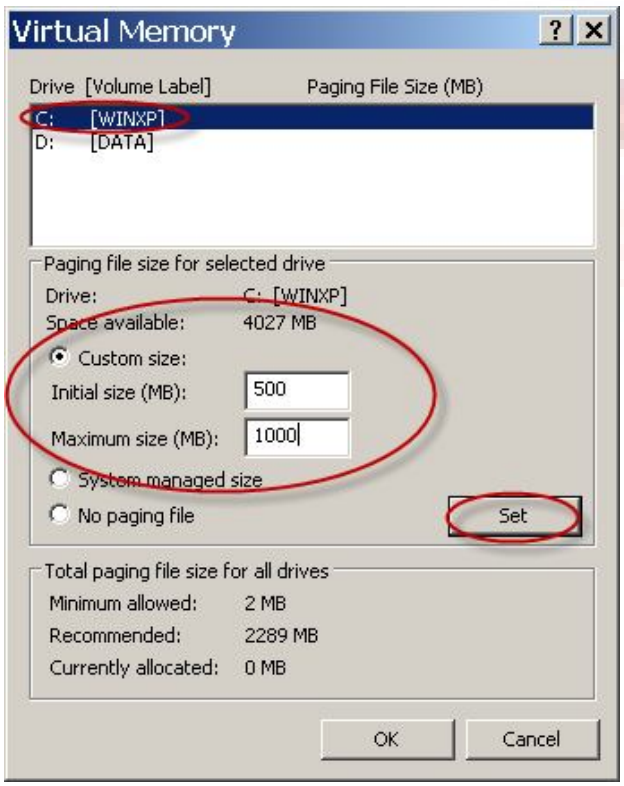
B1 : Nhấn chuột phải **My Computer** → **Properties** → chọn thẻ **Advanced** → nhấn **Settings**



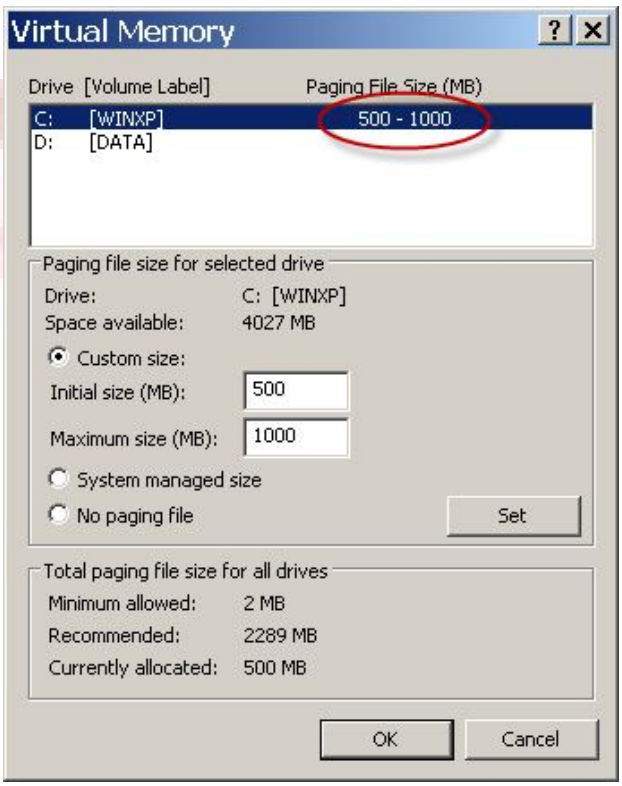
B2 : Chọn thẻ **Advanced** → nhấn **Change**



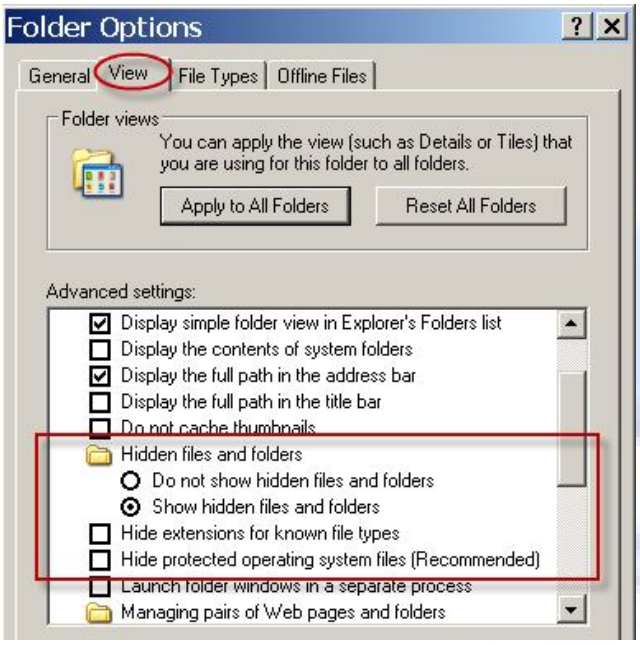
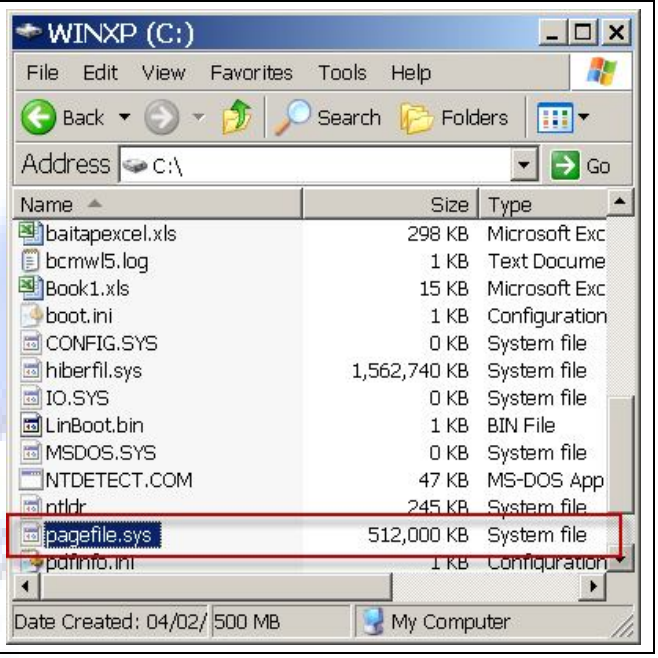
B3 : chọn ổ đĩa **C:** → nhập kích thước RAM ảo trong mục **Intial size** và mục **Maximum size**. Nhấn nút **Set**.



B4 : Kết quả sau khi thiết lập thông số cho Ram ảo. Nhấn **OK**

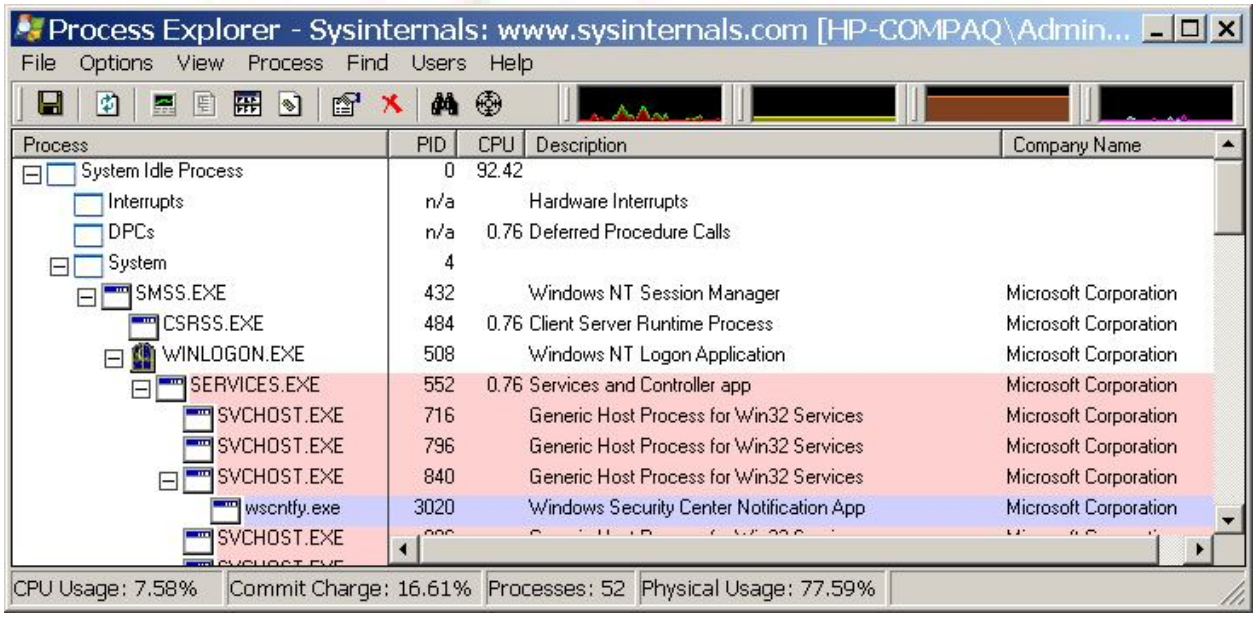


Wndows sẽ lưu trữ bộ nhớ ảo này trên đĩa cứng thông qua tập tin **pagefile.sys**. Muốn nhìn thấy được tập tin này ta làm các bước sau

B1 : Khởi động **Windows Explorer** → vào menu **Tools → Folder Options**. Chọn thẻ **View** và chọn mục **Show hidden files and floder** → tiếp theo tả bỏ các dấu check của mục **Hide extensions for known file types** và mục **Hide protected opreating system files**. Nhấn **OK**	**B2** : Truy cập vào ổ đĩa C:\, ta sẽ tìm thấy tập tin **pagefile.sys** là tập tin hệ thống và có dung lượng bằng đúng với dung lượng mà ta đã cấu hình cho RAM ảo.
	

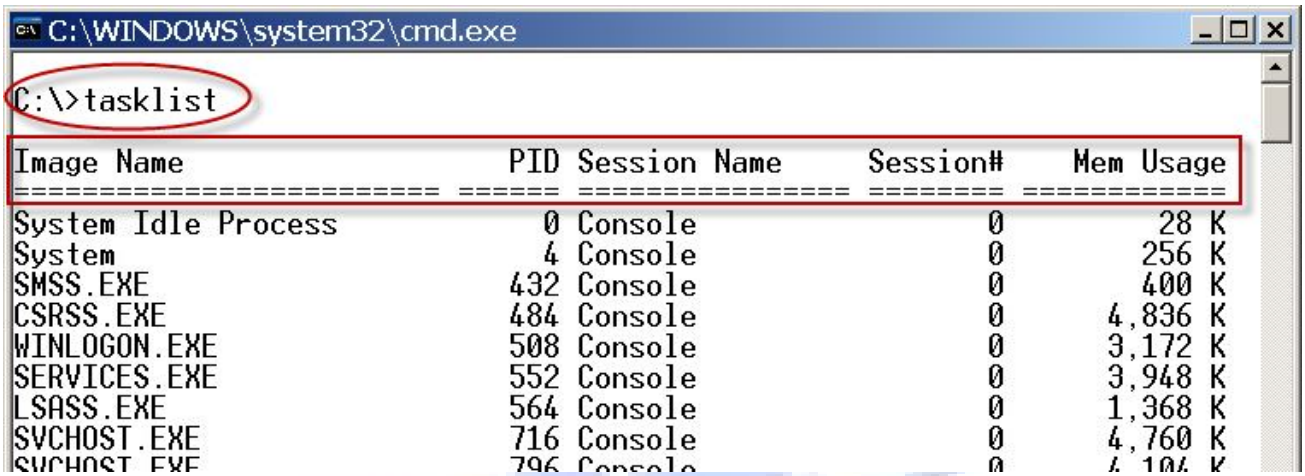
6.3 Công cụ Process Explorer

Công cụ này cho phép ta biết thông tin chi tiết hơn về một tiến trình, ví dụ thông tin về tiến trình cha, tiến trình con, tiến trình đó chạy file nào trên máy, …



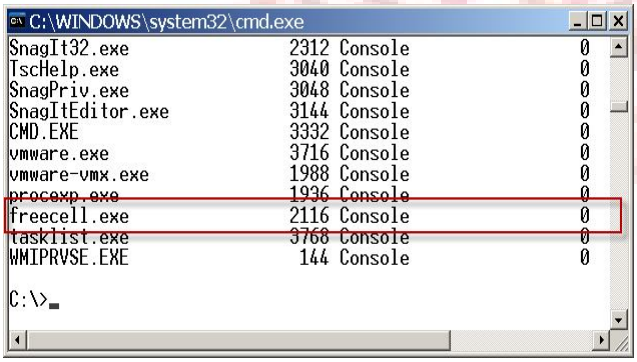
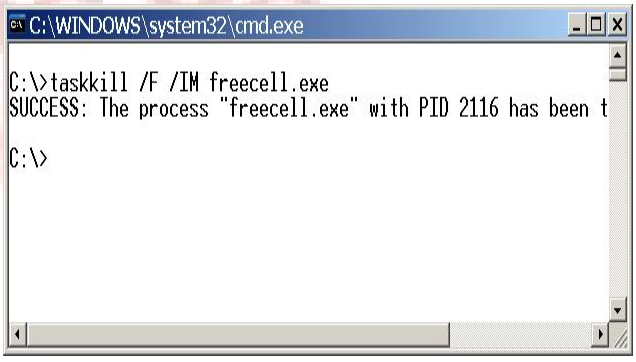
6.4 Lệnh TASKLIST

Lệnh này liệt tất cả các tiến trình đang chạy kèm với các thông tin của tiến trình như : **Image Name** (tên process), **PID** (số hiệu tiến trình), **Mem Usage** (dung lượng bộ nhớ mà tiến trình dùng)



6.5 Lệnh TASKKILL

Lệnh này dùng để kết thúc một tiến trình. Ví dụ ta muốn kết thúc tiến trình có tên freecell.exe ta sẽ làm các bước sau :

B1: giả sử khởi động ứng dụng FreeCell bằng cách vào **Start → Run →** gõ **freecell → OK**. Sau đó kiểm tra bằng lệnh tasklist xem có tiến trình này chưa. Ta nhận xét tiến trình này có tên freecell.exe và có PID là 2116.	**B2:** kết thúc tiến trình bằng lệnh taskkill với cú pháp: C:\>**taskkill /F /IM freecell.exe** Hoặc C:\>**taskkill /PID 2116 /T**
	

Lab 7
TÌM HIỂU VỀ CRACKING

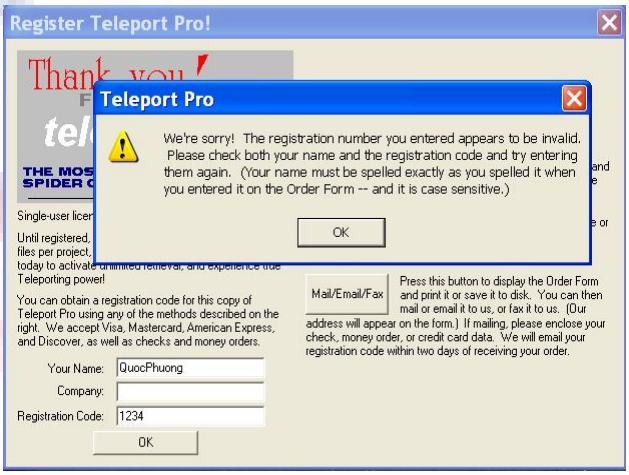
Chú ý : *Các thao tác được hướng dẫn chỉ mang tính chất tham khảo phục vụ trong việc học tập, các sinh viên* ***được cảnh báo không sử dụng trong việc khác****.*

Chuẩn bị công cụ:

- Phần mềm **W32DASM** : phần mềm dùng để dịch ngược file thực thi thành mã máy.
- Phần mềm **OllyDbg** : phần mềm dùng để debug các file thực thi.
- Sinh viên cần am hiểu mã lệnh assembly (mã máy)

7.1 Dò tìm số serial trong phần mềm Teleport pro 1.29

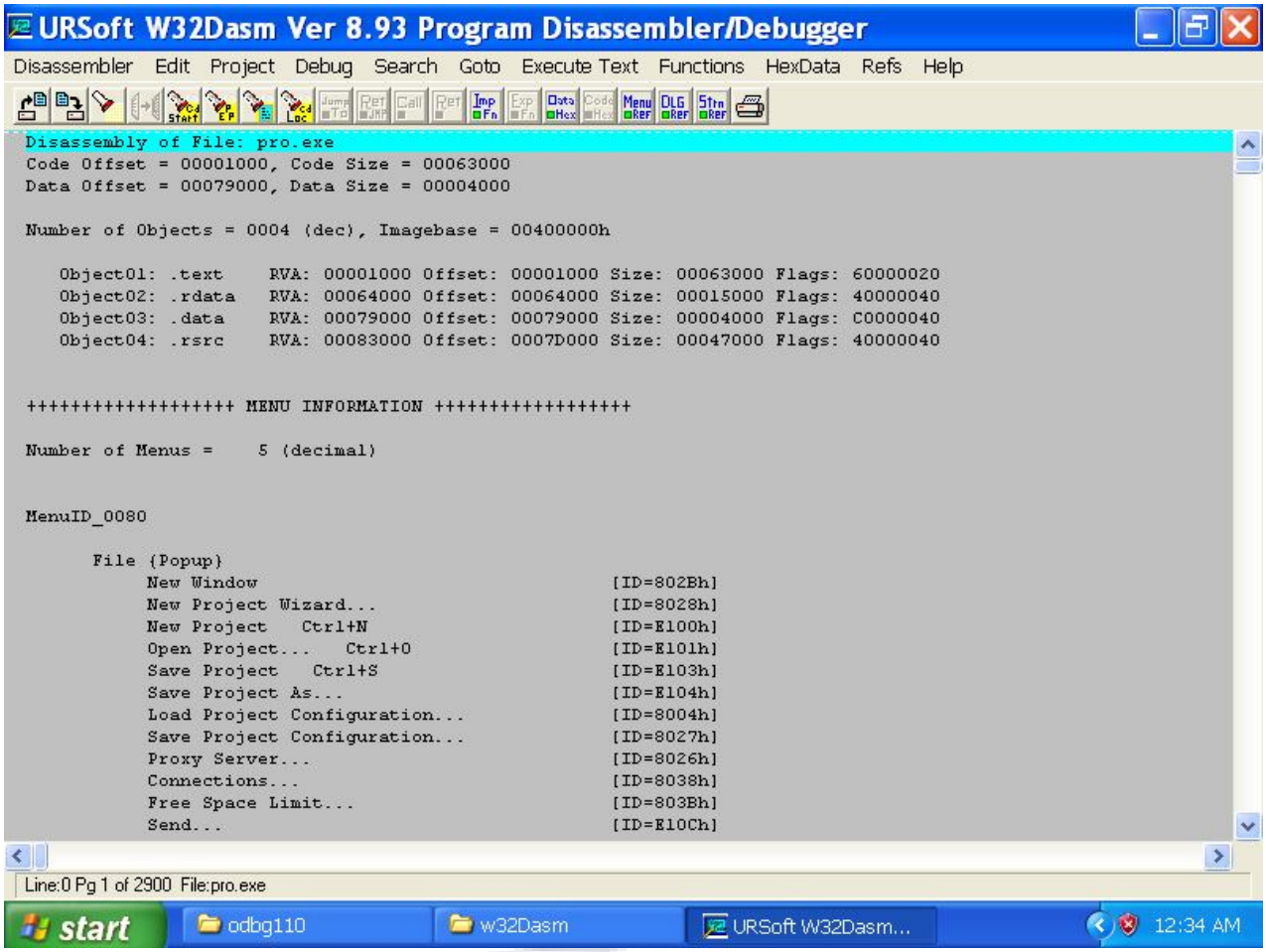
Chương trình Teleport Pro là một chương trình dùng để "hút" tất cả mọi thứ trên một website nhằm giúp chúng ta có thể duyệt Web offline mà không cần phải online. Nhưng nếu không đăng ký bạn chỉ có thể "hút" tối đa 500 file.

Cài đặt phần mềm Teleport Pro 1.29 lên Windows XP bằng cách nhấn đúp vào tập tin cài đặt **pro12.exe**	Chạy phần mềm Teleport Pro vào mục **Help** chọn **Register** để đăng ký. Ta thấy chương trình báo sai số serial.
	

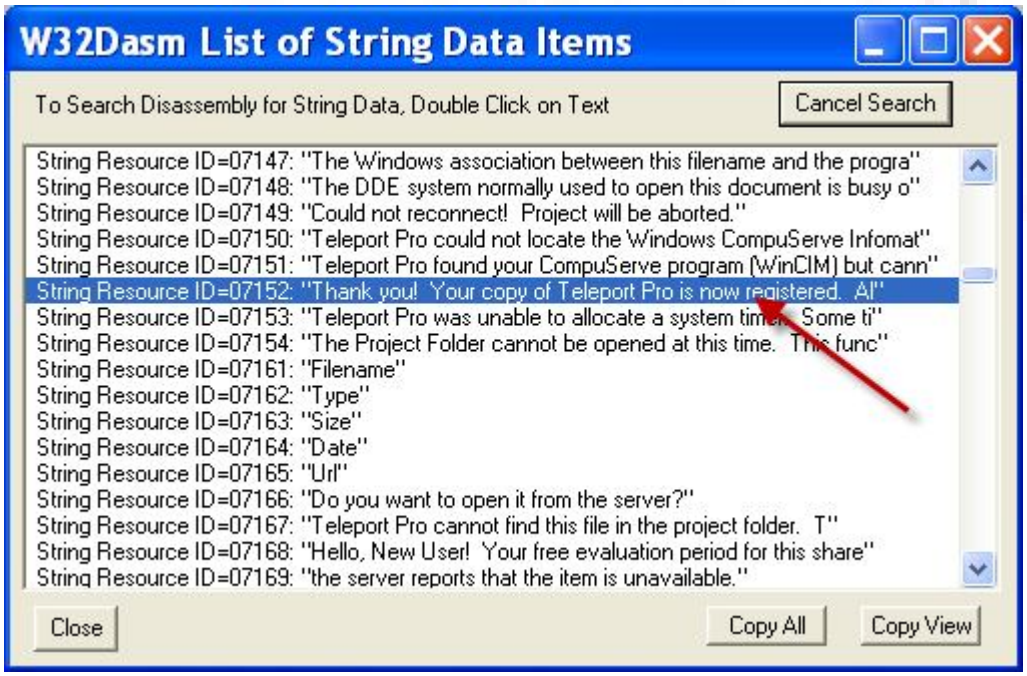
Vấn đề nằm ở đây, phải chăng lúc này chương trình sẽ kiểm tra số serial chúng ta nhập vào và số serial của chương trình, như vậy phải xảy ra một sự so sánh. Nếu như chúng ta có số serial đúng thì sự so sánh này sẽ cho kết quả đúng và chúng ta sẽ vượt qua được bức tường này.

Đến lúc này ta cần dùng công cụ **W32DASM** để tìm một số những thông tin hữu ích về phần mềm. Hãy khởi động **W32DASM** , sau đó nhấn **Disassembler ~> Open File To Disassmble…** và chỉ đến đường dẫn nơi chứa file thực thi của chương trình Teleport Pro nằm trong **C:\Program Files\Teleport Pro**

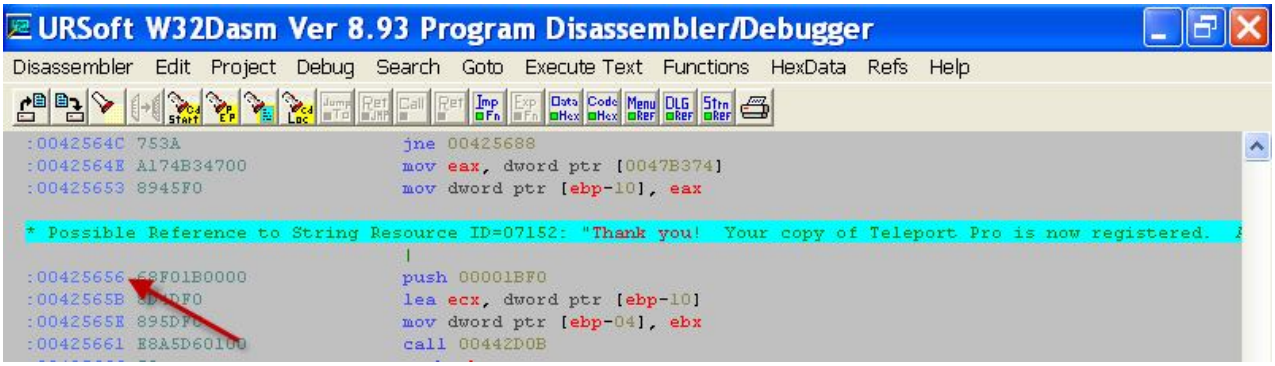
Và sau đây là giao diện của chương trình sau khi **W32DASM** đã dịch ngược thành mã máy



Ta vào tiếp mục **Refs** ~> **String Data Reference** và tìm thấy một dòng chữ đáng ngờ sau đây



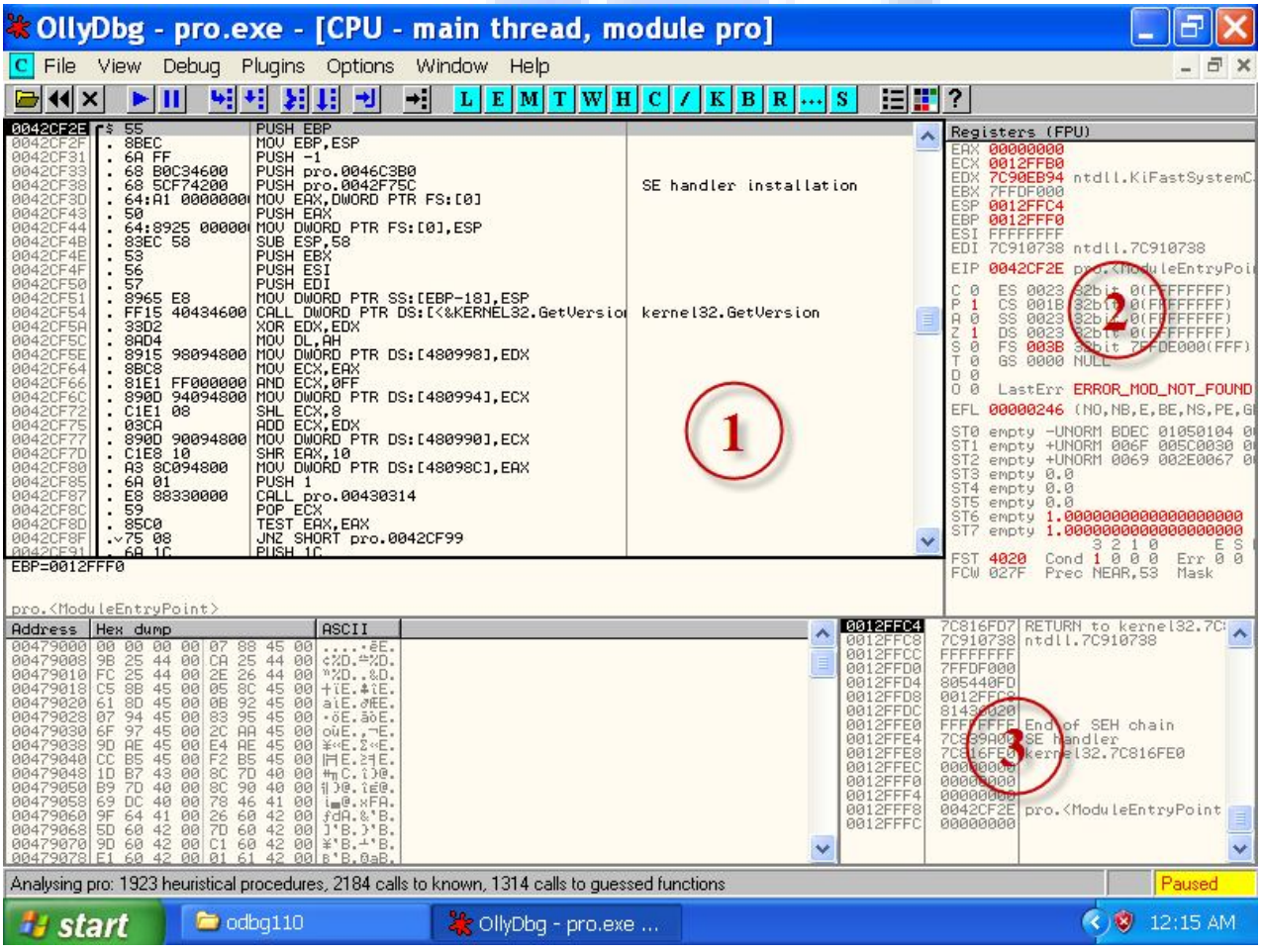
Hãy nhấn đúp vào dòng này và ghi lại địa chỉ nơi xảy ra mã lệnh thông báo này



00425656 69F01B0000 push 00001BF0

Đến lúc này ta cần dùng công cụ **OllyDbg** để dò tìm số serial từ thông tin ích ỏi ở trên. Hãy khởi động **OllyDbg** , sau đó nhấn **File** ~> **Open** và chỉ đến đường dẫn nơi chứa file thực thi của chương trình Teleport Pro nằm trong **C:\Program Files\Teleport Pro**

Và sau đây là giao diện của chương trình sau khi **OllyDbg** đã dịch ngược thành mã máy



Trong đó màn hình số ❶ là hình ảnh về mã nguồn của của tập tin thực thi Teleport Pro đã dịch ngược thành mã máy (mã lệnh assembly). Màn hình số ❷ hiển thị nội dung của các thanh ghi trong CPU. Màn hình số ❸ hiển thị nội dung của bộ nhớ RAM.

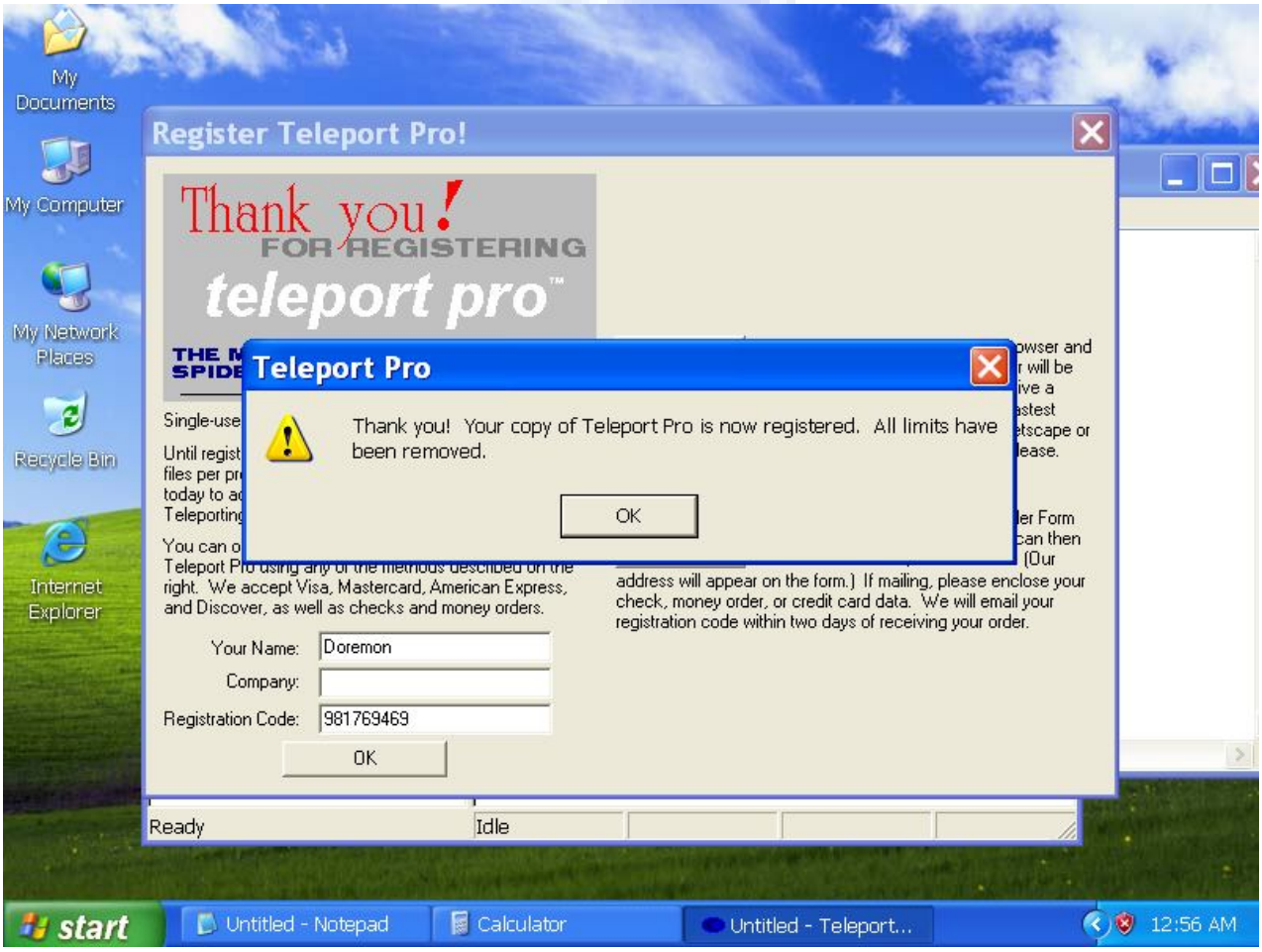
Nhấn **Ctrl + G** và gõ vào địa chỉ quý báu tìm thấy từ chương trình **W32DASM**. Di chuyển lên trên chúng ta sẽ tìm thấy một vài mã lệnh có ích. Ta sẽ đặt "bẫy" (bằng cách nhấn F2) tại địa chỉ mã lệnh sau

```
00425602 E8 54BC0100  CALL pro.00441258
```

Sau đó sẽ dò lần lượt xem chương trình đã làm gì (bằng cách nhấn F8) rồi nhập tên thông tin về username là doremon, gõ vào số serial bất kỳ, sau đó nhấn F8 và khi đến dòng mã lệnh

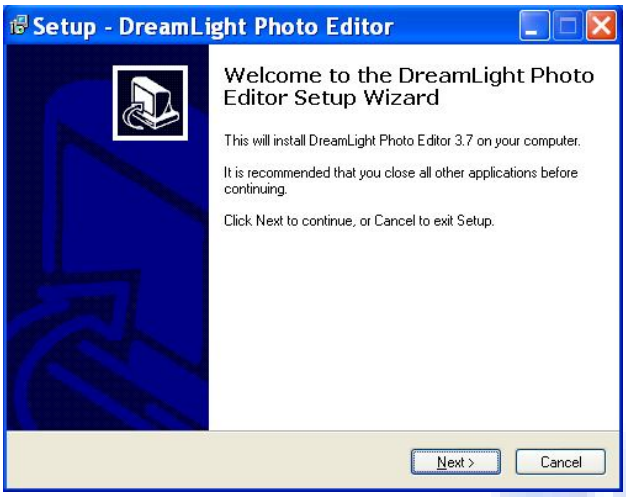
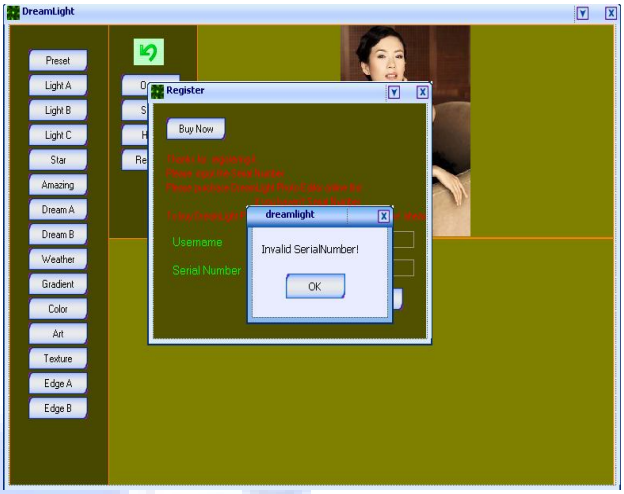
```
00425648 3945 E8      CMP DWORD PTR SS:[EBP-18],EAX
```

Hãy để ý dòng lệnh này, lệnh CMP là so sánh, vậy ra bức tường ngăn cản của chương trình nằm ở đây, đồng thời ta cũng phát hiện giá trị thanh ghi EAX trong CPU lúc này có giá trị số hex là 3A849CFD. Lấy máy tính và tính toán chuyển số này thành số thập phân ta có giá trị này là 981769469. Đây phải chăng là số serial của chương trình. Hãy khởi động lại Teleport Pro, nhập username là Doremon và số serial là 981769469 bạn sẽ thấy kết quả thật tuyệt.



7.2 Bỏ qua quá trình kiểm tra số serial của phần mềm DreamLight

Phần mềm DreamLight là phần mềm giúp chúng ta tạo các hiệu ứng trên ảnh khá đẹp và thú vị nhưng chỉ cho phép chúng ta sử dụng 30 lần, chương trình sẽ khoá và không cho chúng ta dùng nữa. Hãy thử xem !

Cài đặt phần mềm DreamLight lên Windows XP bằng cách nhấn đúp vào tập tin cài đặt **dreamlight.exe**	Chạy phần mềm DreamLight vào mục **Register** để đăng ký. Ta thấy chương trình báo sai số serial. Hãy để ý đến thông báo "**Invalid SerialNumber!**"
	

Đến lúc này ta cần dùng công cụ **OllyDbg** để tìm kiếm các thông tin hữu ích. Hãy khởi động **OllyDbg** , sau đó nhấn **File** ~> **Open** và chỉ đến đường dẫn nơi chứa file thực thi của chương trình DreamLight nằm trong **C:\Program Files\DreamLight Photo Editor**

Tiếp theo ta nhấn chuột phải chọn mục **Search For** ~> **All referenced text string**. Hãy tìm xem dòng chữ "**Invalid SerialNumber!**" có mặt ở những địa chỉ nào, hãy ghi ra giấy những địa chỉ này. Ta quay trở lại màn hình mã lệnh assembly và tìm thấy những dòng mã lệnh có ích sau đây

```
00473863 TEST AL,AL
00473865 JE dreamlig.00473905
.....
00473881 CMP EAX,OB
00473865 JNZ SHORT dreamlig.00473905
.....
004738C2 JNZ SHORT dreamlig.00473905
```

Hãy để ý các dòng lệnh JE, JNZ tất cả đều là lệnh nhảy và nhảy đến địa chỉ 00473905, mà xem xét kỹ thì 00473905 là địa chỉ chứa dòng chữ "Invalid SerialNumber", tới đây nghi ngờ các dòng lệnh này sẽ thực thi nếu số serial của ta gõ vào không đúng với chương trình thì lập tức nó sẽ báo lỗi, vậy ta sẽ xử lý không cho nó nhảy tới địa chỉ trên bằng cách sửa lại như sau: (sửa bằng cách nhấn đúp vào dòng mã lệnh đó)

```
00473863 TEST AL,AL
00473865 JNZ dreamlig.00473905
.....
```

```
00473881 CMP EAX,OB
00473865 JE SHORT dreamlig.00473905
.....
004738C2 JE SHORT dreamlig.00473905
```

Sau khi đã sửa xong, ta nhấn chuột phải, chọn **Copy to executable** ~> **All modifications ~> Copy all** và chọn Yes ghi đè lên tập tin thực thi của chương trình.

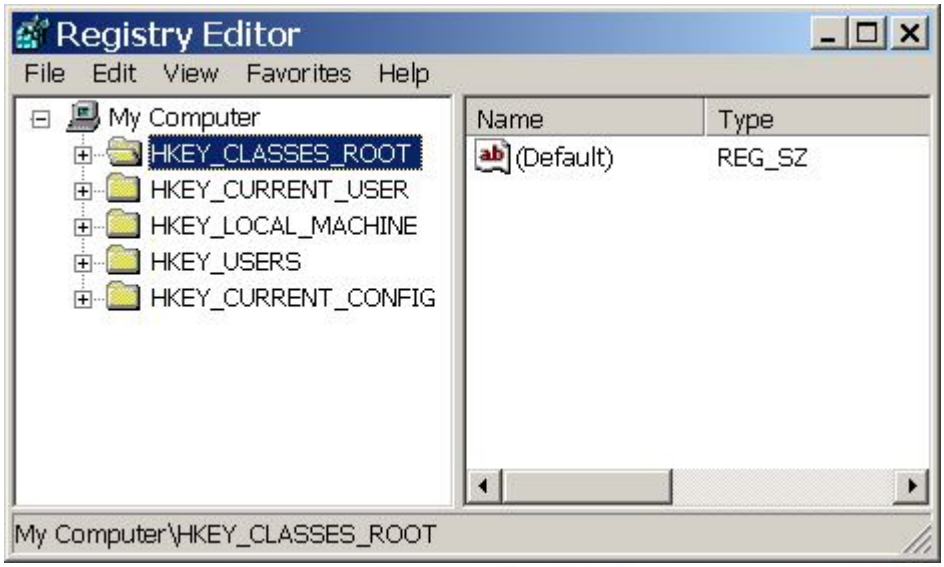
Bây giờ thử khởi động lại chương trình và đăng ký, sau đó nhấn **Open** chương trình sẽ không cản trở ta nữa.

Bài tập : Hãy thử với phần mềm **EditPlus3**, **magicphotof**, **photoshine**, **kbmusic**

Lab 8
TÌM HIỂU VỀ REGISTRY

8.1 Giới thiệu Registry

Registry là một cơ sở dữ liệu dùng để lưu trữ thông số kỹ thuật của Windows. Nó ghi nhận tất cả các thông tin khi bạn thay đổi, chỉnh sửa trong Menu Settings, Control Panel....
Để chỉnh sửa registry ta nhấn **Start → Run →** gõ **regedit → OK**.
Có 5 nhánh chính, mỗi nhánh chứa đựng những thông tin đặc biệt lưu trữ trong Registry.



HKEY_CLASSES_ROOT
Nhánh này theo dõi những tập tin liên kết (File Association), dữ liệu thông tin về các đối tượng trong lớp, hệ điều hành và các phím tắt.

HKEY_CURRENT_USER
Nhánh này liên kết với khu vực của nhánh HKEY_USERS. Nôi dung của nhánh này tuỳ thuộc vào những người dùng hiện đăng nhập máy tính. Nhánh này lưu lại các sở thích và các xác lập liên quan đến các ứng dụng dành riêng cho từng người sử dụng máy.

HKEY_LOCAL_MACHINE
Nhánh này chứa đựng các thông tin đặc biệt về máy tính bao gồm phần cứng , phần mềm và cấu hình của máy tính. Thông tin này được sử dụng cho tất cả các người dùng máy tính.

HKEY_USERS
Nhánh này chứa đựng những sở thích riêng lẻ của từng người dùng máy tính, mỗi người dùng được đại diện bởi 1 tù khoá con SID, được định vị dưới nhánh chính.

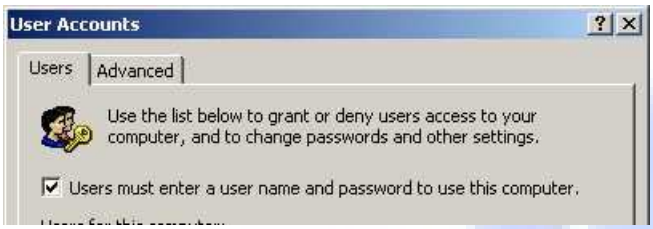
HKEY_CURRENT_CONFIG

Nhánh này liên kết với khu vực của nhánh HKEY_LOCAL_MACHINE, dành riêng cho cấu hình của phần cứng hiện hành.

8.2 Tự động đăng nhập không cần mật khẩu

Nếu bạn là người duy nhất sử dụng máy tính thì thao tác đăng nhập mỗi khi khởi động máy là không cần thiết. Bạn có thể vô hiệu hóa việc đăng nhập này bằng cách:

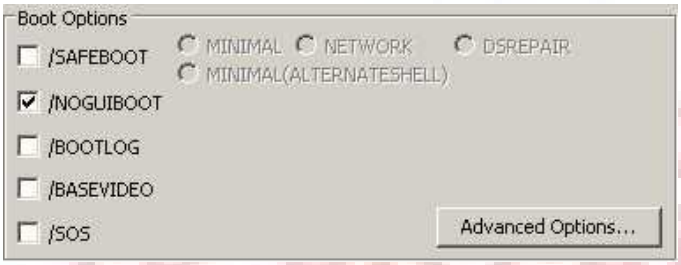
- Nhấp **Start** > **Run…** gõ lệnh ***control userpasswords2*** rồi nhấn **Enter**.
- Xuất hiện hộp thoại, bỏ dấu kiểm trước tùy chọn **Users must enter a user name and password to use this computer**



8.3 Không hiển thị logo Windows XP khi khởi động

Nếu không thích hiển thị logo của Windows XP trong quá trình khởi động, bạn làm như sau:

- Vào **Start** > **Run**… gõ ***MSCONFIG*** rồi nhấn **Enter**.
- Nhấn chọn thẻ ***BOOT.INI.***
- Đánh dấu kiểm trước tùy chọn ***/NOGUIBOOT.***

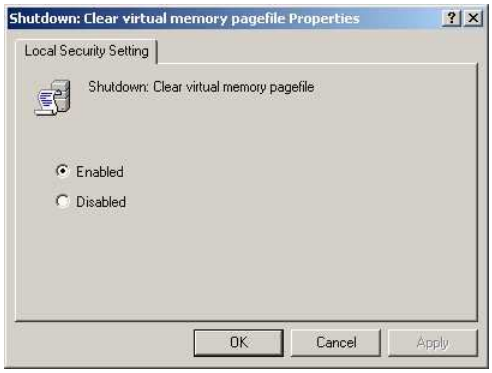


8.4 Xóa nội dung Page File (pagefie.sys) khi tắt máy

Khi bộ nhớ RAM bị thiếu, hệ thống sẽ tạo một bộ nhớ ảo lấy dung lượng từ đĩa cứng. Nội dung bộ nhớ ảo này được lưu trữ trong một tập tin tạm gọi là *Page File*. Tùy theo dung lượng chỉ định mà tập tin này có kích thước lớn hay nhỏ. Để làm trống nội dung tập tin này mỗi lần shutdown máy, bạn làm theo 1 trong 2 cách sau:

Cách 1 : Dùng **Administrative Tools**:

- Vào **Start** > **Settings** > **Control Panel**
- Nhấp đúp lên ***Administrative Tools*** > ***Local Security Policy***
- Duyệt tới các mục ***Local Policies*** > ***Security Options***
- Trong danh sách bên phải, tìm và nhấp đúp lên mục ***Shutdown: Clear Virtual Memory Pagefile*** rồi chọn ***Enable***
- Khởi động lại máy tính.



Cách 2 : Thay đổi trong **Registry**:

- Vào **Start** > **Run** gõ **Regedit** rồi nhấn **Enter**.
- Lần lượt duyệt tới các khóa **HKEY_LOCAL_MACHINE \ SYSTEM \ CurrentControlSet \ Control \ Session Manager \Memory Management**.
- Cửa sổ bên phải, nhấp đúp lên khóa ***ClearPageFileAtShutdown*** rồi sửa giá trị thành 1.
- Đóng Registry và khởi động lại máy tính.

8.5 Hiển thị hộp thông báo trước khi Logon

Với hướng dẫn này bạn sẽ làm cho máy bạn hiện một thông báo trước khi hiện hộp thoại đăng nhập (Logon). Đó có thể là nội quy công ty hay lời nhắc nhở của bạn tuỳ thích.

Tìm đến khoá chỉ ra bên dưới trong Registry Editor, tạo hai giá trị String mới nếu chưa có với tên là "**LegalNoticeCaption**" với dữ liệu là tiêu đề của hộp thoại, và giá trị tên "**LegalNoticeText**" với dữ liệu là nội dung của hộp thoại thông báo.

Minh họa trong Registry Editor		
Name	Type	Data
(Default)	REG_SZ	(not set value)
LegalNoticeCaption	REG_SZ	Thông báo!
LegalNoticeText	REG_SZ	Hiện giờ chưa có gì để thông báo!
My Computer\ HKEY_LOCAL_MACHINE\ SOFTWARE\ Microsoft\ Windows NT\ CurrentVersion\ Winlogon		

System Key: HKEY_LOCAL_MACHINE\ SOFTWARE\ Microsoft\ Windows NT\ CurrentVersion\ Winlogon

Name: LegalNoticeCaption, LegalNoticeText

Type: REG_SZ

8.6 Ẩn Tab Hardware

Khi nhấn chuột phải trên các ổ đĩa các thông tin liên quan đến ổ đĩa và chọn Properties trong menu Context sẽ hiện lên trong một cửa sổ gồm nhiều Tab, trong đó có Hardware. Ta có thể tắt Tab HardWare như sau:
Tìm đến khoá chỉ ra bên dưới trong Registry Editor, tạo mới giá trị DWORD với tên "**NoHardWareTab**" nếu chưa có và sửa dữ liệu của nó thành 1 để giấu Tab HardWare.

Minh họa trong Registry Editor		
Name	Type	Data
(Default)	REG_SZ	(not set value)
NoHardwareTab	REG-DWORD	0x00000001(1)
My Computer\ HKEY_CURRENT_USER\ Software\ Microsoft\ Windows\ CurrentVersion\ Policies\ Explorer		

User Key: HKEY_CURRENT_USER\ Software\ Microsoft\ Windows\ CurrentVersion\ Policies\ Explorer.
System Key: HKEY_LOCAL_MACHINE\ Software\ Microsoft\ Windows\ CurrentVersion\ Policies\ Explorer
Name: NoHardwareTab
Type: REG_DWORD
Value: 0 - hiện Tab HardWare, 1 - ẩn Tab HardWare.

8.7 Ẩn Tab Security

Trong hộp thoại **Properties** của tập tin và thư mục (xuất hiện trong menu context khi click phải lên Icon của tập tin và thư mục) có tab **Security** cho phép bạn thiết lập chức năng bảo mật cho tập tin và thư mục. Hướng dẫn này giúp bạn có thể tắt tab này.
Tìm đến khoá chỉ ra bên dưới trong Registry Editor, tạo mới giá trị DWORD với tên "**NoSecurityTab**" nếu chưa có và sửa dữ liệu của nó thành 1 để giấu tab **Security**.

Minh họa trong Registry Editor		
Name	Type	Data
(Default)	REG_SZ	(not set value)
NoSecurityTab	REG-DWORD	0x00000001(1)
My Computer\ HKEY_CURRENT_USER\ Software\ Microsoft\ Windows\ CurrentVersion\ Policies\ Explorer		

User Key: HKEY_CURRENT_USER\ Software\ Microsoft\ Windows\ CurrentVersion\ Policies\ Explorer.
System Key: HKEY_LOCAL_MACHINE\ Software\ Microsoft\ Windows\ CurrentVersion\ Policies\ Explorer
Name: NoSecurityTab
Type: REG_DWORD
Value: 0 - hiện, 1 - ẩn tab Security.

8.8 Ẩn mục Properties khi click phải trên biểu tượng My Computer

Hướng dẫn này giúp bạn giấu menu **Properties** khi click phải vào biểu tượng My Computer.
Tìm đến khoá chỉ ra bên dưới trong Registry Editor, tạo giá một giá trị kiểu DWORD nếu nó chưa có với tên "**NoPropertiesMyComputer**", sửa dữ liệu nó là 1 để ẩn menu **Properties** của My Computer.

Minh họa trong Registry Editor		
Name	Type	Data
(Default)	REG_SZ	(not set value)
NoPropertiesMyComputer	REG_DWORD	0x00000001(1)
My Computer\ HKEY_CURRENT_USER\ Software\ Microsoft\ Windows\ CurrentVersion\ Policies\ Explorer		

User Key: HKEY_CURRENT_USER\ Software\ Microsoft\ Windows\ CurrentVersion\ Policies\ Explorer
System Key: HKEY_LOCAL_MACHINE\ Software\ Microsoft\ Windows\ CurrentVersion\ Policies\ Explorer
Name: NoPropertiesMyComputer
Type: REG_DWORD
Value: 0: Hiện menu Properties, 1 - ẩn menu Properties

8.9 Ẩn các ổ đĩa

Hướng dẫn này giúp bạn có thể ẩn từng ổ đĩa hoặc tất các ổ đĩa từ A đến Z trong cửa sổ Explorer.
Tìm đến khoá chỉ ra bên dưới, tạo mới giá trị DWORD với tên "**NoDrives**" nếu chưa có.
Chú ý các con số sau tương ứng với từng ổ đĩa. A:1, B: 2, C: 4, D: 8, E: 16, E: 32, F: 64, G: 128, H: 256, I: 512, J: 1024, K: 2048, L: 4096, M: 8192, N: 16384, O: 32768, P: 65536, Q: 131072, R: 262144, S: 524288, T: 1048576, U: 2097152, V: 4194304, X: 8388608, Y: 16777216, Z: 33554432. All: 67108863. Từ A-Z: là cấp số nhân bội số 2 bắt đầu từ 1.
Nếu muốn ẩn 1 ổ đĩa thì sửa dữ liệu cho giá trị **"NoDrives"** tương ứng với số đặc trưng của ổ đĩa nêu trên, nếu muốn ẩn nhiều ổ đĩa thì cộng dồn các số đặc trưng đó, ví dụ muốn ẩn ổ C và D thì giá trị của NoDrives là 12, và muốn ẩn tất cả các ổ đĩa thì giá trị của Drives là 67108863.

Minh họa trong Registry Editor		
Name	Type	Data
(Default)	REG_SZ	(not set value)
NoDrives	REG_DWORD	0x03ffffff (67108863)
My Computer\ HKEY_CURRENT_USER\ Software\ Microsoft\ Windows\ CurrentVersion\ Policies\ Explorer		

User Key: HKEY_CURRENT_USER\ Software\ Microsoft\ Windows\ CurrentVersion\ Policies\ Explorer
System Key: HKEY_LOCAL_MACHINE\ Software\ Microsoft\ Windows\ CurrentVersion\ Policies\ Explorer
Name: NoDrives
Type: REG_DWORD

8.10 Đổi tên và biểu tượng các ổ đĩa

Hướng dẫn này giúp bạn thiết lập Registry để đổi tên các ổ đĩa và Icon của nó theo ý bạn.
Trình tự thực hiện các bước như sau:

Bước 1: Tìm đến khóa trong Registry Editor **HKEY_LOCAL_MACHINE\ Software\ Microsoft\ Windows\ CurrentVersion\ Explore**r. Tạo một khóa với tên **DriveIcons** nếu nó chưa có. Tạo một khóa con với tên tương ứng với tên của ổ đĩa mà bạn muốn đổi tên và Icon của nó. Ví dụ ở đây là **D.**

Bước 2: Tạo một khóa con của khóa **D** với tên **DefaultIcon**, nhập dữ liệu cho giá trị **Default** của nó là đường dẫn đầy đủ của Icon mà bạn muốn chọn làm biểu tượng cho ổ đĩa.

Bước 3: Tạo một khóa con của khóa **D** với tên là **DefaultLabel**, nhập dữ liệu cho giá trị **Default** của nó là tên mới cho ổ đĩa.
User Key: HKEY_LOCAL_MACHINE\ Software\ Microsoft\ Windows\ CurrentVersion\ Explorer\ DriveIcons

8.11 Ẩn các Icon đặc biệt trên Desktop

Các Icon **My Computer**, **My Documents**, **Internet Explorer**, **My Network Places** đôi lúc làm màn hình Desktop trở nên chật chội, chúng ta có thể tắt chúng đi.
Bảng sau giúp bạn tham khảo những giá trị đại diện cho các biểu tượng đặc biệt trên Desktop.

Tên giá trị trong Registry	Tên biểu tượng trên Desktop
{208D2C60-3AEA-1069-A2D7-08002B30309D}	My Network Connections
{20D04FE0-3AEA-1069-A2D8-08002B30309D}	My Computer
{450D8FBA-AD25-11D0-98A8-0800361B1103}	My Documents
{871C5380-42A0-1069-A2EA-08002B30309D}	Internet Explorer

Trong Windows XP, click phải trên mành hình Desktop, chọn **Properties**, chọn Tab **Desktop**, nhấn **Customize Desktop**... , chọn Tab **General**. Từ đây bỏ dấu Check để ẩn các biểu tượng một cách dễ dàng. Khi bạn chọn bỏ dấu Check thì Windows sẽ lưu vào Registry, vâng hầu như tất cả đều liên quan đến Registry.
Khi bạn đang sử dụng StartMenu kiểu Classic thì Windows sẽ ghi vào khóa **HKEY_CURRENT_USER\ Software\ Microsoft\ Windows\ CurrentVersion\ Explorer\ HideDesktopIcons\ ClassicStartMenu.**
Tương ứng với biểu tượng nào bạn bỏ dấu Check thì Windows sẽ ghi vào Registry giá trị DWORD tương đương được liệt kê ở bảng trên và gán dữ liệu cho nó là 1 để ẩn nó, 0 để hiện lại nó nếu bạn check vào ô check box.

Minh họa trong Registry Editor - Ẩn My Computer		
Name	Type	Data
(Default)	REG_SZ	(not set value)
{20D04FE0-3AEA-1069-A2D8-08002B30309D}	REG_DWORD	0x00000001(1)
My Computer\ HKEY_CURRENT_USER\ Software\ Microsoft\ Windows\ CurrentVersion\ Explorer\ HideDesktopIcons\ ClassicStartMenu.		

Khi bạn đang sử dụng Start Menu XP thì Windows thực hiện tương tự như ở khoá khác: HKEY_CURRENT_USER\ Software\ Microsoft\ Windows\ CurrentVersion\ Explorer\ HideDesktopIcons\ NewStartPanel

Chúng ta vẫn có cách khác để ẩn các biểu tượng này trên Desktop.

Tìm đến khoá chỉ ra bên dưới trong Registry Editor, tạo những giá trị DWORD mới nếu nó không tồn tại tương ứng với các giá trị được liệt kê ở bản trên và gán dữ liệu cho nó là 1 nếu bạn muốn ẩn nó đi

Minh họa trong Registry Editor - Ẩn My Computer		
Name	Type	Data
(Default)	REG_SZ	(not set value)
{20D04FE0-3AEA-1069-A2D8-08002B30309D}	REG_DWORD	0x00000001(1)
My Computer\ HKEY_CURRENT_USER\ Software\ Microsoft\ Windows\ CurrentVersion\ Policies\ NonEnum		

User Key: HKEY_CURRENT_USER\ Software\ Microsoft\ Windows\ CurrentVersion\ Policies\ NonEnum

System Key: HKEY_LOCAL_MACHINE\ Software\ Microsoft\ Windows\ CurrentVersion\ Policies\ NonEnum

Name: {20D04FE0-3AEA-1069-A2D8-08002B30309D}

Type: REG_DWORD

Value: 0 - hiện, 1 - ẩn

8.12 Ẩn hiện nút Shutdown

Hướng dẫn này giúp bạn vô hiệu hoá nút **Shutdown** trên **Start Menu**.

Tìm đến khoá chỉ ra bên dưới trong Registry Editor. Tạo một giá trị DWORD mới với tên "**NoClose**" nếu chưa có và gán dữ liệu cho nó là 1 để vô hiệu hoá nút Shutdown.

Minh họa trong Registry Editor		
Name	Type	Data
(Default)	REG_SZ	(not set value)
NoClose	REG_DWORD	0x00000001(1)
My Computer\ HKEY_CURRENT_USER\ Software\ Microsoft\ Windows\ CurrentVersion\ Policies\ Explorer		

User Key: HKEY_CURRENT_USER\ Software\ Microsoft\ Windows\ CurrentVersion\ Policies\ Explorer

Name: NoClose

Type: REG_DWORD

Value: 0 - hiện, 1 - vô hiệu hoá Shutdown

8.13 Tắt menu Log Off trên Start Menu

Tìm đến khoá chỉ ra bên dưới trong Regsitry Editor, tạo một giá trị kiểu DWORD mới với tên "**StartMenuLogOff**" và sửa dữ liệu cho nó thành 1 để không cho phép hiển thị mục LogOff.

Minh họa trong Registy Editor		
Name	Type	Data
(Default)	REG_SZ	(not set value)
StartMenuLogOff	REG_DWORD	0x00000000(0)
My Computer\ HKEY_CURRENT_USER\ Software\ Microsoft\ Windows\ CurrentVersion\ Explorer\ Advanced		

User Key: HKEY_CURRENT_USER\ Software\ Microsoft\ Windows\ CurrentVersion\ Explorer \ Advanced

Name: StartMenuLogOff

Type: REG_DWORD

Value: 1- ẩn menu Log Off, 0 - hiện menu Log Off

8.14 Tắt menu Run

Hướng dẫn này giúp bạn điều khiển Windows tắt menu Run trên Start Menu. Khi đã tắt menu Run bạn vẫn có thể dùng phím tắt **Windows + R** để mở cửa sổ Run.

Tìm đến khoá chỉ ra bên dưới trong Regsitry Editor, tạo một giá trị kiểu DWORD mới với tên "**StartMenuRun**" và sửa dữ liệu cho nó thành 0 để tắt menu Run.

Minh họa trong Registy Editor		
Name	Type	Data
(Deafult)	REG_SZ	(not set value)
StartMenuRun	REG_DWORD	0x00000000(0)
My Computer\ HKEY_CURRENT_USER\ Software\ Microsoft\ Windows\ CurrentVersion\ Explorer\ Advanced		

User Key: HKEY_CURRENT_USER\ Software\ Microsoft\ Windows\ CurrentVersion\ Explorer\ Advanced

Name: StartMenuRun

Type: REG_DWORD

Value: 0- tắt menu Run. 1- hiện menu Run trên Start Menu

8.15 Tắt menu Help & Supports trên Start Menu

Đối với những người mới làm quen với máy tính thì menu Help có tính hữu dụng, như khi bạn đã có chút kiến thức cơ bản rồi thì phần Help không cần đến nữa. Hướng dẫn này giúp bạn điều khiển Windows tắt menu Help trên Start Menu.

Tìm đến khoá chỉ ra bên dưới trong Regsitry Editor, tạo một giá trị kiểu DWORD mới với tên "**NoSMHelp**" nếu chưa có và sửa dữ liệu cho nó thành 1 tắt menu Help trên Start Menu.

Minh họa trong Registy Editor		
Name	Type	Data
(Deafult)	REG_SZ	(not set value)
NoSMHelp	REG_DWORD	0x00000001(1)
My Computer\ HKEY_CURRENT_USER\ Software\ Microsoft\ Windows\ CurrentVersion\ Policies\ Explorer		

User Key: HKEY_CURRENT_USER\ Software\ Microsoft\ Windows\ CurrentVersion\ Policies\ Explorer
System Key: HKEY_LOCAL_MACHINE\ Software\ Microsoft\ Windows\ CurrentVersion\ Policies\ Explorer
Name: NoSMHelp
Type: REG_DWORD
Value: 0 - hiển thị menu Help, 1 - tắt menu Help.

8.16 Tắt menu Search trên Start Menu

Hướng dẫn này giúp bạn tắt menu Search trên Start menu. Tắt menu này đồng nghĩa với việc vô hiệu hoá công cụ tìm kiếm của Windows. Bạn phải cân nhắc trước khi chỉnh sửa.
Tìm đến khoá chỉ ra bên dưới trong Regsitry Editor, tạo một giá trị kiểu DWORD mới với tên "**NoFind**" nếu chưa có và sửa dữ liệu cho nó thành 1 tắt menu Search trên Start Menu.

Minh họa trong Registy Editor		
Name	Type	Data
(Deafult)	REG_SZ	(not set value)
NoFind	REG_DWORD	0x00000001(1)
My Computer\ HKEY_CURRENT_USER\ Software\ Microsoft\ Windows\ CurrentVersion\ Policies\ Explorer		

User Key: HKEY_CURRENT_USER\ Software\ Microsoft\ Windows\ CurrentVersion\ Policies\ Explorer
System Key: HKEY_LOCAL_MACHINE\ Software\ Microsoft\ Windows\ CurrentVersion\ Policies\ Explorer
Name: NoFind
Type: REG_DWORD
Value: 0 - hiển thị menu Search, 1 - tắt menu Search.

8.17 Ẩn tất cả các Icon trong Systray

Tìm đến khoá chỉ ra bên dưới trong Regsitry Editor, tạo một giá trị kiểu DWORD mới với tên "**NoTrayItemsDisplay**" và sửa dữ liệu cho nó thành 1.

Minh họa trong Registy Editor		
Name	Type	Data
(Deafult)	REG_SZ	(not set value)
NoTrayItemsDisplay	REG_DWORD	0x00000001(1)
My Computer\ HKEY_CURRENT_USER\ Software\ Microsoft\ Windows\ CurrentVersion\ Policies\ Explorer		

User Key: HKEY_CURRENT_USER\ Software\ Microsoft\ Windows\ CurrentVersion\ Policies\ Explorer
System Key: HKEY_LOCAL_MACHINE\ Software\ Microsoft\ Windows\ CurrentVersion\ Policies\ Explorer
Name: NoTrayItemsDisplay
Type: REG_DWORD
Value: 0 - hiện các Icon, 1- ẩn tất cả các Icon

8.18 Tắt đồng hồ trên Systray

Tìm đến khoá chỉ ra bên dưới trong Regsitry Editor, tạo một giá trị kiểu DWORD mới với tên "**HideClock**" nếu chưa có và sửa dữ liệu cho nó thành 1 để ẩn đồng hồ.

Minh họa trong Registy Editor		
Name	Type	Data
(Deafult)	REG_SZ	(not set value)
HideClock	REG_DWORD	0x00000001(1)
My Computer\ HKEY_CURRENT_USER\ Software\ Microsoft\ Windows\ CurrentVersion\ Policies\ Explorer		

User Key: HKEY_CURRENT_USER\ Software\ Microsoft\ Windows\ CurrentVersion\ Policies\ Explorer
System Key: HKEY_LOCAL_MACHINE\ Software\ Microsoft\ Windows\ CurrentVersion\ Policies\ Explorer
Name: HideClock
Type: REG_DWORD
Value: 0 - hiển thị đồng hồ, 1 - tắt đồng hồ.

8.19 Hiển thị đồng hồ trên Taskbar theo phong cách mới

Mặc định trên Taskbar xuất hiện đồng hồ với phần đuôi bằng tiếng Anh là AM, PM. Hướng dẫn này giúp bạn thay đổi chúng thành một từ bất kỳ (nhưng phải không quá 5 ký tự) để đồng hồ của bạn sinh động hơn. Và hoàn toàn có thể Việt hóa phần đuôi đó với tiếng Việt Unicode cũng như có thể thêm vào phần đâu của đồng hồ một dòng chữ bất kỳ như hình bên dưới.
Tìm đến khoá chỉ ra bên dưới trong Regsitry Editor. Sửa dữ liệu của giá trị **s1159** thành "**Sáng**" mà mặc định là AM, **s2359** thành "**Chiều**" mà mặc định là PM, **sTimeFormat** thành "**Bây giờ là hh:mm:ss tt**". Bạn sẽ có một đồng hồ tuyệt đẹp.

Minh hoạ trong Registy Editor		
Name	Type	Data
(Deafult)	REG_SZ	(not set value)
s1159	REG_SZ	Sáng
s2359	REG_SZ	Chiều
sTimeFormat	REG_SZ	Bây giờ là hh:mm:ss tt
My Computer\ HKEY_CURRENT_USER\ Control Panel\ International		

User Key: HKEY_CURRENT_USER\ Control Panel\ International

Name: s1159, s2359, sTimeFormat

Type: REG_SZ

8.20 Vô hiệu hoá chức năng Update Windows

Hướng dẫn này giúp bạn vô hiệu hoá chức năng cập nhật phiên bản mới của Windows.

Tìm đến khoá chỉ ra bên dưới trong Registry Editor. Tạo một khoá kiểu DWORD mới với tên **"NoWindowsUpdate"** nếu chưa có và gán dữ liệu cho nó là 1 để vô hiệu hoá chức năng Update của Windows.

Minh họa trong Registry Editor		
Name	Type	Data
(Default)	REG_SZ	(not set value)
NoWindowsUpdate	REG_DWORD	0x00000001(1)
HKEY_CURRENT_USER\ SOFTWARE\ Microsoft\ Windows\ CurrentVersion\ Policies\ Explorer		

User Key: HKEY_CURRENT_USER\ Software\ Microsoft\ Windows\ CurrentVersion\ Policies\ Explorer

System Key: HKEY_LOCAL_MACHINE\ Software\ Microsoft\ Windows\ CurrentVersion\ Policies\ Explorer

Name: NoWindowsUpdate

Type: REG_DWORD

Value: 0 - mặc định, 1 - vô hiệu hoá chức năng Update.

8.21 Ẩn Start Menu

Những cái có khi chúng ta tưởng rằng không làm được vẫn có thể. Ví như hướng dẫn dưới đây giúp bạn ẩn Start Menu. Khi đó nhấn chuột vào nút Start Menu sẽ không xuất hiện menu nữa.

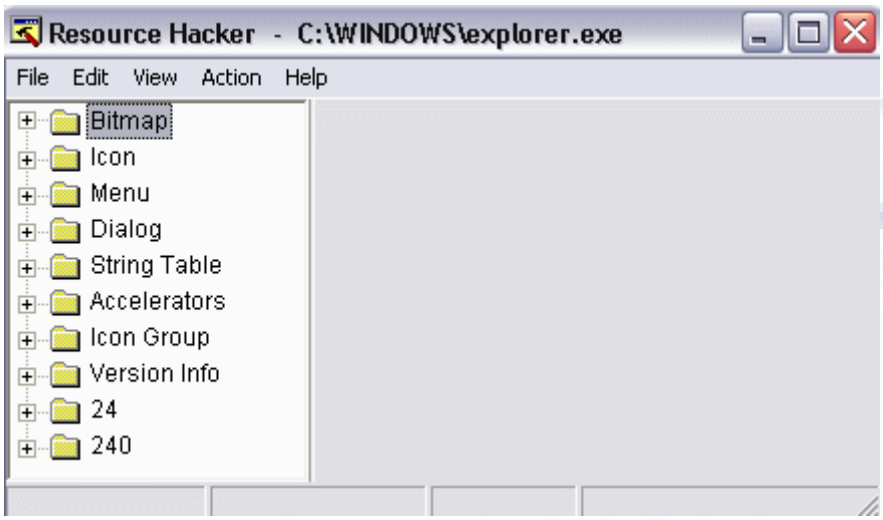
Mở Registry Editor, tìm đến khoá **HKEY_CLASSES_ROOT\CLSID\{5b4dae26-b807-11d0-9815-00c04fd91972}**. Sửa tên khoá này thành **{-5b4dae26-b807-11d0-9815-00c04fd91972}**, tức thêm dấu "-" vào sau "{". Muốn hiện lại Start Menu thì xoá dấu "-" đi. Cập nhật lại Registry để chỉ sữa chữa này có hiệu lực.

8.22 Thay đổi chữ 'Start' trên Start Menu

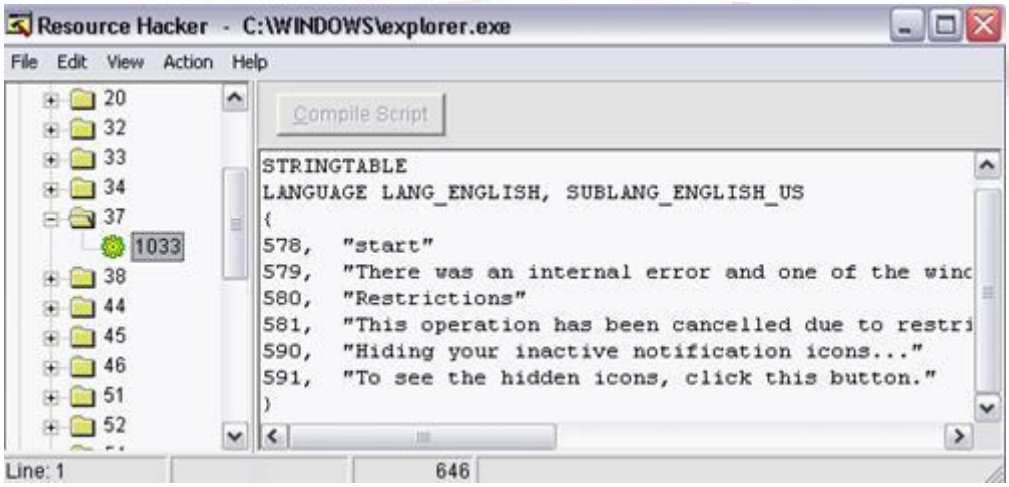
Bạn có muốn thay chữ Start thành một từ khác, có thể là tên bạn hoặc một từ khác. Điều này có thể làm được theo hướng dẫn sau. Thông thường muốn đổi chữ Start thì phải dùng một chương trình đọc File nhị phân. Những có một điều nguy hiểm là có thể làm hỏng hệ thống của bạn với một xác suất rất lớn. Nếu muốn thử thì bạn phải lưu lại tập tin Explorer.exe trong thư mục Windows, hoặc WinNT vào một nơi an toàn.

Tạo một bản copy của tập tin **explorer.exe** và đổi tên chúng thành một tên khác. Ví dụ là **MyExplorer.exe**. Lưu nó ở một thư mục khác.

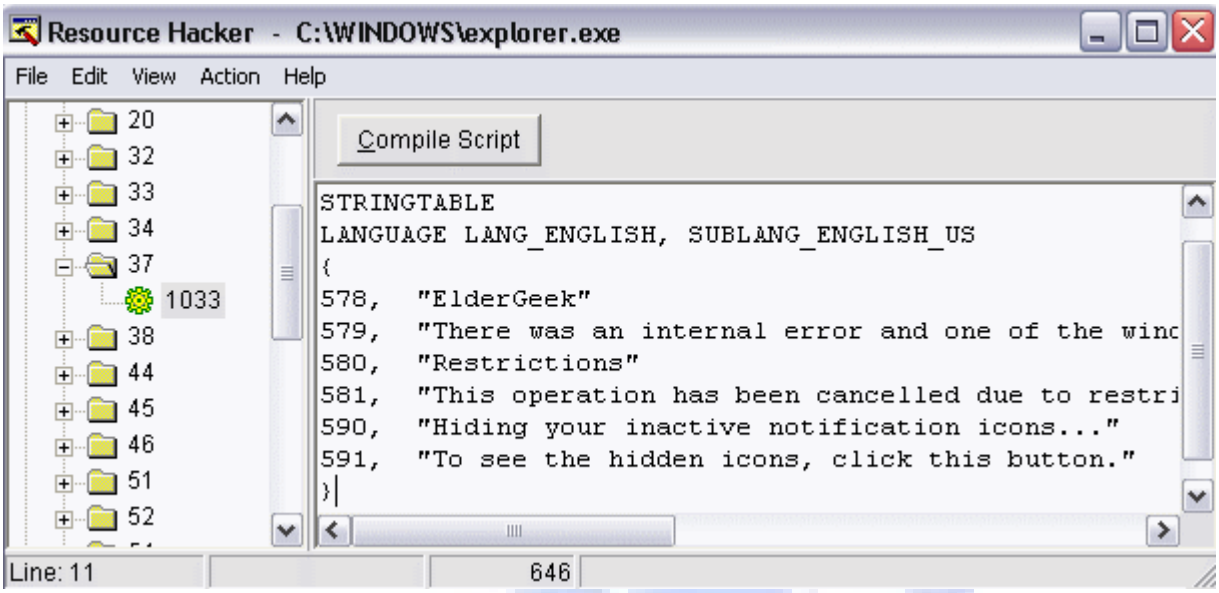
- Dùng chương trình **ResHacker.exe** để tiến hành phẫu thuật tập tin **MyExplorer**.



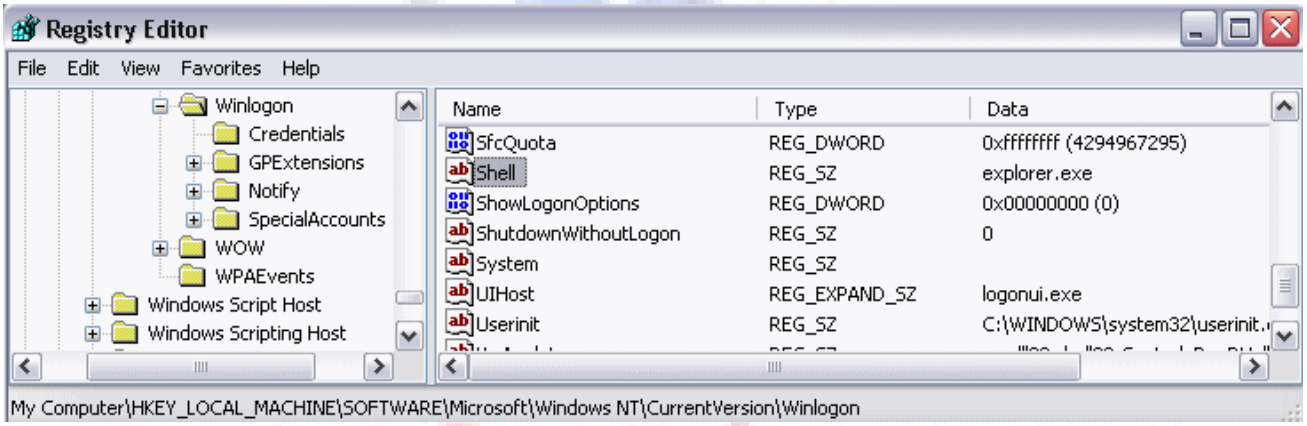
- Tìm đến nhánh String Table và chọn nhánh 37, chọn số 1033



- Bên màn hình bên phải nhìn thấy chữ "start", ta gõ lại thành chữ "ElderGreek" chẳng hạn



- Biên dịch lại file bằng cách nhấn **Compile Script**



- Vào Registry Editor tìm khoá **HKEY_LOCAL_MACHINE\ SOFTWARE\ Microsoft\ Windows NT\ CurrentVersion\ Winlogon**. Tìm khoá con của nó có tên Shell gán giá trị mới cho nó là "MyExplorer.exe". Khởi động lại máy. Khi muốn trở về trạng thái 'Start' nguyên thuỷ thì chỉ việc sửa giá trị của khoá "Shell" là "explorer.exe".

8.23 Tắt Settings của Display trong Control Panel

Điều này cũng có nghĩa là bạn không thể thay đổi được độ phân giải của màn hình mục Settings.

User Key: HKEY_CURRENT_USER\ Software\ Microsoft\ Windows\ CurrentVersion\ Policies\ System.
System Key: HKEY_LOCAL_MACHINE\ Software\ Microsoft\ Windows\ CurrentVersion\ Policies\ System.
Name: NoDispSettingsPage
Type: REG_DWORD (DWORD Value)
Value: (0 = hiển thị tab Settings, 1 = không hiển thị tab Settings)

8.24 Tắt mục ScreenSaver của Display trong Control Panel

Khi tắt mục ScreenSaver không có nghĩa là tắt chế độ ScreenSaver tự chạy, mà là ẩn không cho thay đổi, chỉnh sửa ScreenSaver. Vì vậy trước khi tắt nó đi bạn phải cân nhắc.

User Key: HKEY_CURRENT_USER\ Software\ Microsoft\ Windows\ CurrentVersion\ Policies\ System.
System Key: HKEY_LOCAL_MACHINE\ Software\ Microsoft\ Windows\ CurrentVersion\ Policies\ System.
Name: NoDispScrSavPage
Type: REG_DWORD (DWORD Value)
Value: (0 = disabled, 1 = enabled)

8.25 Tắt mục thay đổi hình nền của Display trong Control Panel

Không cho thay đổi hình nền, màu nền của Desktop.

User Key: HKEY_CURRENT_USER\ Software\ Microsoft\ Windows\ CurrentVersion\ Policies\ System.
System Key: HKEY_LOCAL_MACHINE\ Software\ Microsoft\ Windows\ CurrentVersion\ Policies\ System.
Name: NoDispBackgroundPage
Type: REG_DWORD (DWORD Value)
Value: (0 = disabled, 1 = enabled)

8.26 Tắt mục Appearance của Display trong Control Panel

User Key: HKEY_CURRENT_USER\ Software\ Microsoft\ Windows\ CurrentVersion\ Policies\ System.
System Key: HKEY_LOCAL_MACHINE\ Software\ Microsoft\ Windows\ CurrentVersion\ Policies\ System.
Name: NoDispAppearancePage
Type: REG_DWORD (DWORD Value)
Value: (0 = disabled, 1 = enabled)

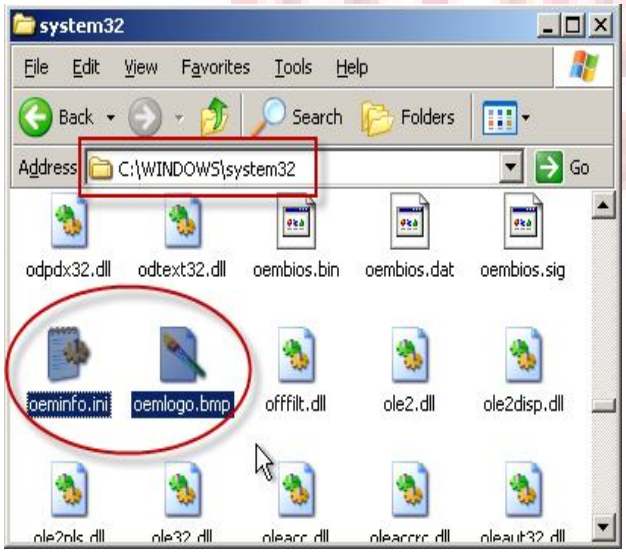
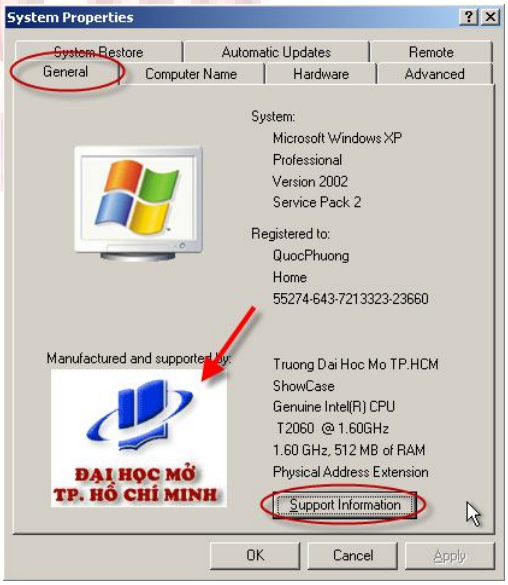
8.27 Thay đổi Wallpaper của màn hình đăng nhập

Khi cài đặt Wallpaper nền cho desktop, wallpaper màn hình đăng nhập ban đầu không bị thay đổi và vẫn còn là wallpaper mặc định. Bạn thay đổi Wallpaper bằng cách sau: Tìm tới từ khoá theo đường dẫn: **HKEY_CURRENT_USER\Control Panel\Desktop**

Value Name: Wallpaper
Data Type: REG_SZ
Cho dữ liệu của của wallpaper là đường dẫn và tên tập tin bitmap (BMP) muốn làm nền. Ví dụ: C:\windows\mylogo.bmp.
Lưu ý: Trong cùng khoá, bạn cũng có thể thiết lập các giá trị bổ sung "Pattern" = "(None)", và TitleWallpaper = ""

8.28 Cài đặt logo công ty và thông tin công ty trong My Computer

B1 : Đầu tiên bạn chuẩn bị một logo của công ty ở dạng bitmap đổi tên tập tin thành **oemlogo.bmp**	**B2** : một tập tin văn bản cho biết thông tin về công ty lưu dưới tên **oeminfo.ini** có dạng như sau:
	[General] Manufacturer=Truong Dai Hoc Mo TP.HCM Model=ShowCase [Support Information] Line1=" " Line2=" Ho tro, Tu van, Ban hang" Line3="" Line4=" " Line5=" Bo Giao Duc & Dao Tao " Line6=" Ban Trung Cap Dai Hoc Mo" Line7="" Line8=" +1 (888) 888-888 (voice)" Line9=" +1 (888) 888-889 (fax)" Line10="" Line11="" Line12=" http://www.ou.edu.vn"

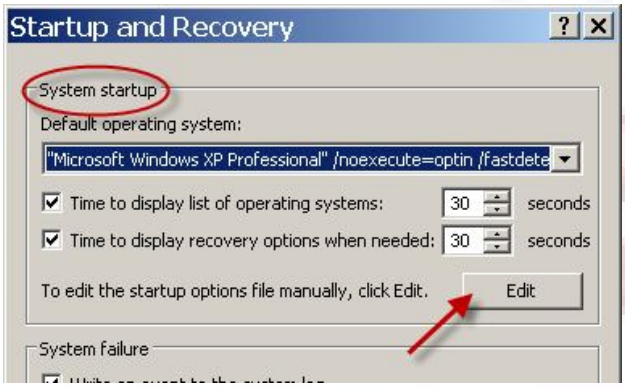
B3 : chép hai tập tin **oemlogo.bmp** và **oeminfo.ini** vào thư mục **C:\Windows\System32**	**B4** : Nhấn chuột phải **My Computer → Properies**, chúng ta sẽ thấy màn hình như sau
	

8.29 Thay đổi màn hình Boot Screen của Windows

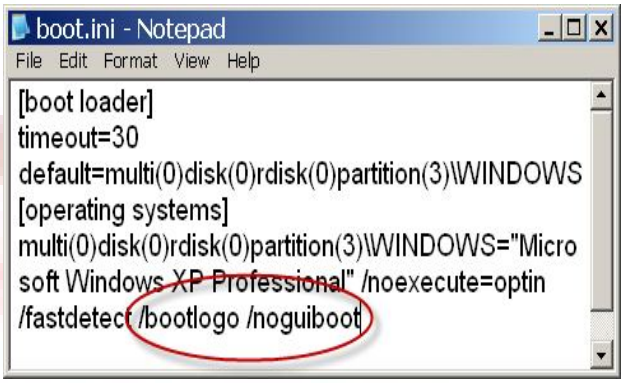
B1 : chuẩn bị một tập tin hình ảnh bitmap 640 x 480 với 16 màu. Đổi tên thành **boot.bmp** và sao chép tập tin này vào trong thư mục **C:\Windows**

B2 : nhấn chuột phải **My Computer** ~> **Properties** ~> chọn thẻ **Advanced** ~> trong mục **Startup and Recovery** nhấn **Settings**

B3 : trong mục **System startup** nhấn **Edit** để chỉnh sửa một số thông tin hệ thống.



B4 : thêm tham số sau vào **/bootlogo /noguiboot,** lưu lại và boot lại máy để xem kết quả.



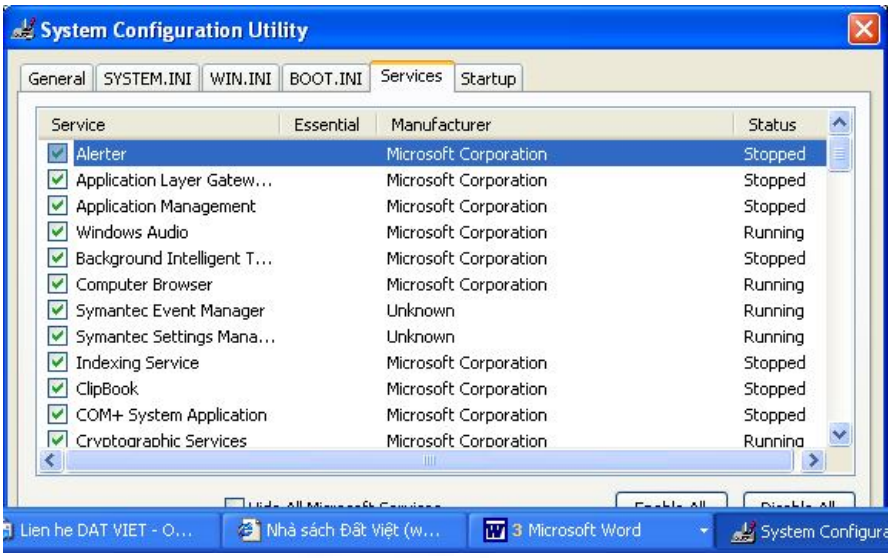
Hoặc có một cách khác là chúng ta có thể dùng phần mềm **BootSkin** (sinh viên tự tham khảo cách dùng phần mềm này)

Lab 9
TỐI ƯU HỆ THỐNG VÀ BẢO MẬT

9.1 Một số dịch vụ có thể vô hiệu hoá

Khi khởi động hệ thống, Windows sẽ tự động kích hoạt nhiều dịch vụ (services) mà người dùng không cần tới. Điều này làm thời gian khởi động máy lâu hơn và tiêu tốn nhiều tài nguyên hệ thống hơn. Do vậy, bạn có thể vô hiệu hóa những dịch vụ không cần thiết, cách thực hiện như sau:

Nhấp **Start** > **Run…** gõ lệnh **msconfig** rồi nhấn **Enter,** xuất hiện hộp thoại



Nhấp chọn thẻ **Services**. Trong danh sách bên dưới, hãy bỏ dấu kiểm trước các mục sau để vô hiệu hóa chúng.

Sau đây là một số dịch vụ không cần thiết: *Alerter, Application Management, Background Intelligent Transfer Service, Clipbook, Error Reporting Service, Fast User Switching, Help and Support, IMAPI CD-Burning COM Service, Indexing Service, IP SEC, Messenger, Net Logon, Network DDE, NT LM Security Support Provider, Performance Logs and Alerts, Portable Media Serial Number, QOS RSVP, Remote Desktop Help Session Manager, Remote Assistance, Remote Registry, Routing & Remote Access, Secondary Login, Smart Card, Smart Card Helper SSDP, Discovery Service TCP/IP NetBIOS Helper, Telnet, Uninterruptible Power Supply Service, Universal Plug and Play Device Host Upload Manager, Volume Shadow Copy Service, Web Client, Wireless Zero Configuration, WMI Performance Adapter.*

9.2 Dọn dẹp thư mục *Prefetch*

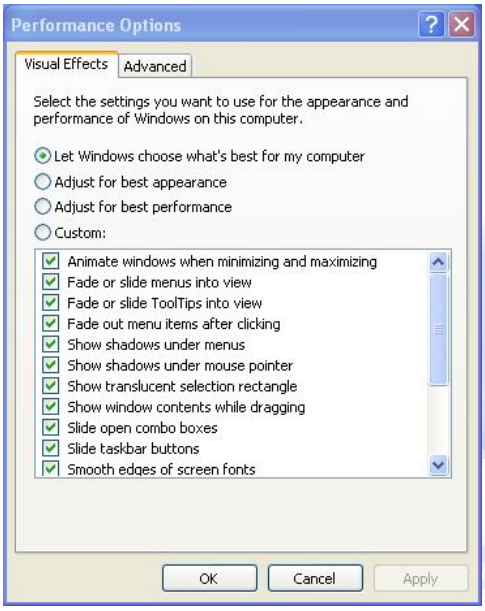
WindowsXP sử dụng một kỹ thuật ***Prefetch*,** giúp giảm thời gian khởi động các ứng dụng mới dùng gần nhất. Sau một thời gian sử dụng, thư mục Prefetch bị đầy làm tốn dung lượng đĩa cứng. Để dọn dẹp thư mục này, bạn thực hiện như sau:

- Nhấp **Start** > **Run…** gõ *Prefetch* rồi nhấn **Enter**.
- Nhấn ***Ctrl-A*** để chọn tất cả các đối tượng.
- Nhấn ***Delete*** để xóa.

9.3 Giảm hiệu ứng đồ hoạ

Mặc định, Windows XP sử dụng khá nhiều các hiệu ứng đồ họa nhằm tạo giao diện bắt mắt người dùng. Tuy nhiên, những hiệu ứng này sẽ làm giảm tốc độ truy xuất của hệ thống. Do đó, nếu thấy không cần thiết bạn có thể vô hiệu hóa chúng bằng các bước như sau:

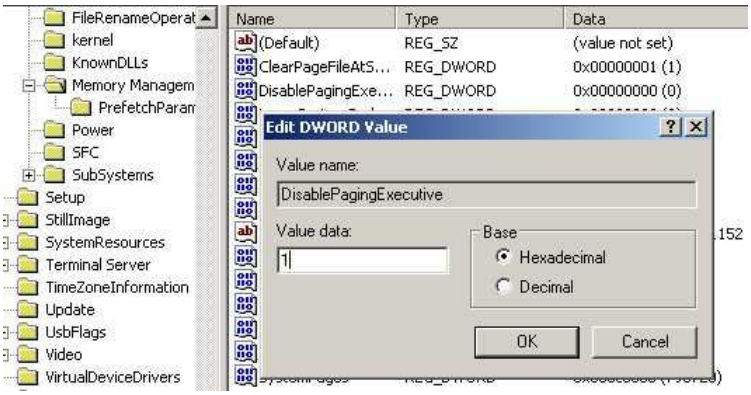
- Nhấp phải biểu tượng **My Computer** chọn **Properties** rồi chọn thẻ **Advanced.**
- Trong khung *Performance*, nhấp **Settings.**
- Danh sách các hiệu ứng được liệt kê. Để bỏ hiệu ứng nào bạn nhấp bỏ dấu chọn trong hộp kiểm. Hoặc nếu muốn vô hiệu hóa tất cả, nhấp chọn mục ***Adjust for best performance***



9.4 Tăng hiệu năng hệ thống

Nếu trong hệ thống bộ nhớ RAM nhiều hơn 512MB, bạn có thể thực hiện bỏ chế độ RAM ảo để tăng hiệu năng hệ thống:

- Vào **Start** > **Run** gõ **Regedit,** nhấn **Enter**.
- Lần lượt duyệt tới các khóa **HKEY_LOCAL_MACHINE \ SYSTEM \ CurrentControlSet \ Control \ Session Manager \ Memory Management \ DisablePagingExecutive**.
- Trong hộp thoại **Value Data**, nhập giá trị là **1**, nhấp OK.
- Khởi động lại máy tính để thay đổi có hiệu lực.



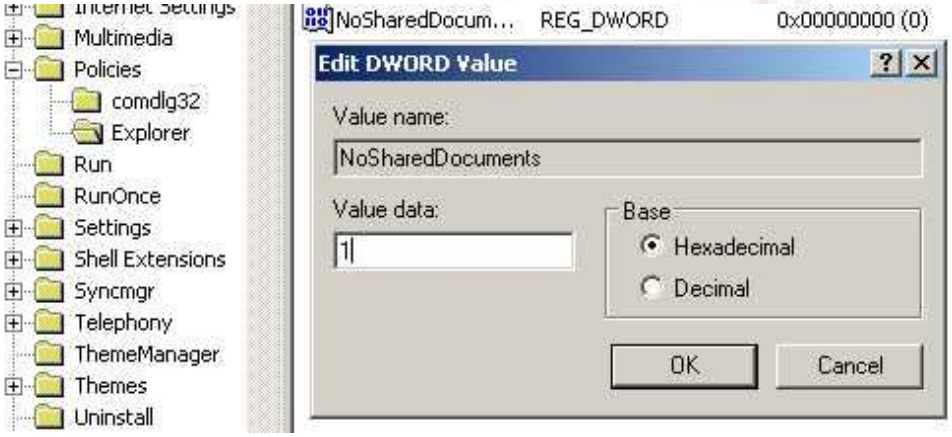
9.5 Ẩn máy tính khỏi *Network Neighborhood*

Trong mạng nội bộ, có những lúc bạn cần chia sẻ tập tin từ một máy tính, và đôi khi cũng muốn ẩn máy này đi, không cho nó xuất hiện trong *Network Neighborhood*. Để thực hiện, bạn làm như sau:
Vào **Start** > **Run** gõ lệnh ***net config server /hidden:yes*** rồi nhấn **Enter**

9.6 Vô hiệu hóa Shared documents

Để vô hiệu hóa thư mục *Shared Documents*, không cho người dùng khác có thể thấy nó trên mạng nội bộ, bạn làm như sau:

- Vào **Start** > **Run** gõ **Regedit** rồi nhấn **Enter**.
- Lần lượt duyệt tới các khoá **HKEY_CURRENT_USER \ Software \ Microsoft \ Windows \ CurrentVersion \ Policies \ Explorer **
- Ngay khóa Explorer, nhấp phải chuột chọn **New** > **DWORD Value,** nhập tên ***NoSharedDocuments.***
- Nhấp đúp lên giá trị vừa tạo và nhập giá trị 1 trong mục **Value Data**
- Khởi động lại máy tính để thay đổi có hiệu lực.



9.7 Không cho phép xóa các Printers đã thiết lập

Printers and Faxes là một trong các thành phần chính của Control Panel trong tất cả các phiên bản của Windows. Nó có nhiệm vụ thiết lập hoạt động cho các máy in, fax. Hướng dẫn này giúp bạn hạn chế chức năng xóa một máy in đã được thiết lập trong Printer and Faxes.

Mở Registry Editor, tìm đến khóa chỉ ra bên dưới. Tạo một giá trị kiểu DWORD với tên "**NoDeletePrinter**" nếu nó chưa có và nhập dữ liệu cho nó là 1 để hạn chế chức năng xóa các Printers đã được thiếp lập trong Printers and Faxes.

Minh họa trong Registry Editor		
Name	Type	Data
(Default)	REG_SZ	(not set value)
NoDeletePrinter	REG_DWORD	0x00000001(1)
My Computer\ HKEY_CURRENT_USER\ Software\ Microsoft\ Windows\ CurrentVersion\ Policies\ Explorer		

User Key: HKEY_CURRENT_USER\ Software\ Microsoft\ Windows\ CurrentVersion\ Policies\ Explorer
System Key: HKEY_LOCAL_MACHINE\ Software\ Microsoft\ Windows\ CurrentVersion\ Policies\ Explorer.
Name: NoDeletePrinter
Type: REG_DWORD
Value: 0 - mặc định, 1 - không cho xóa các Printers đã được thiết lập trong Printers and Faxes.

9.8 Tăng tốc độ lướt web

Hướng dẫn này giúp bạn chỉnh sửa Registry để tăng tốc độ lướt web.
Tìm đến khóa sau: **HKEY_LOCAL_MACHINE\ SOFTWARE\ Microsoft\ Windows\ CurrentVersion\ Explorer\ RemoteComputer\ NameSpace**. Xóa giá trị: **{D6277990-4C6A-11CF-8D87-00AA0060F5BF}**.

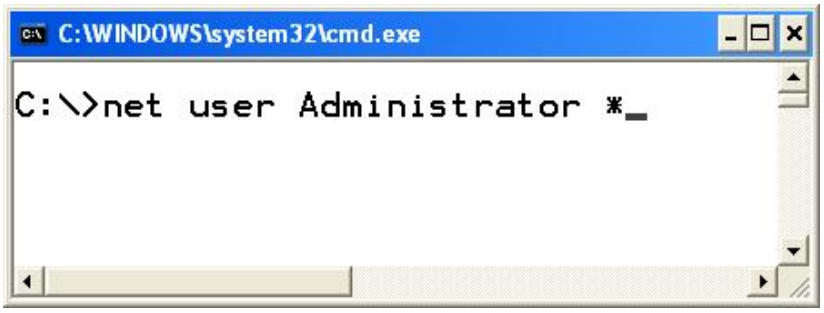
9.9 Hạn chế một số ứng dụng mà các người dùng có thể chạy

Tạo các khoá mới tên là 1, 2, 3, ... với kiểu REG_SZ và nhập đường dẫn vào cho giá trị các khoá đó để chỉ định các chương trình không cho phép người dùng chạy.

User Key: HKEY_CURRENT_USER\ Software\ Microsoft\ Windows\ CurrentVersion\ Policies\ Explorer
System Key: HKEY_LOCAL_MACHINE\ Software\ Microsoft\ Windows\ CurrentVersion\ Policies\ Explorer
Name: DisallowRun

9.10 Xoá mật khẩu Admin

Dùng đĩa CD Hiren Boot hay sử dụng lệnh **net user**.
Ví dụ : muốn xoá mật khẩu của người quản trị thì ta gõ lệnh như sau



Lab 10
VIRUS VÀ SỰ NGUY HIỂM

Chú ý : *Các thao tác được hướng dẫn chỉ mang tính chất tham khảo phục vụ trong việc học tập giúp sinh viên phát huy sáng kiến nhằm giúp đỡ cộng đồng trong việc tiêu diệt và chống lại các virus, các sinh viên* ***được cảnh báo không sử dụng trong việc khác****.*

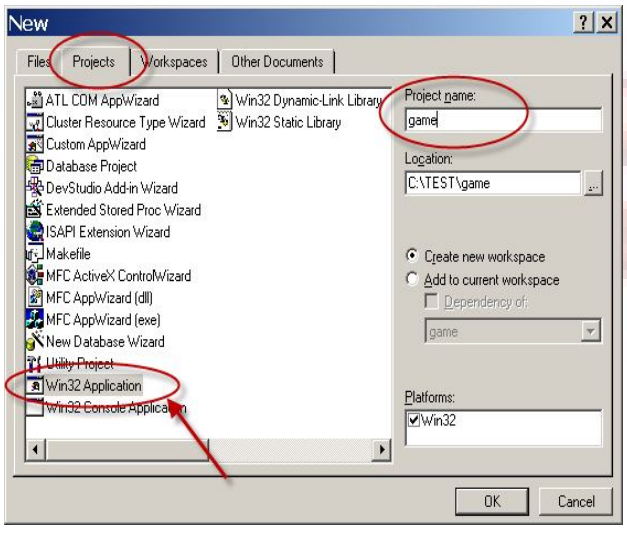
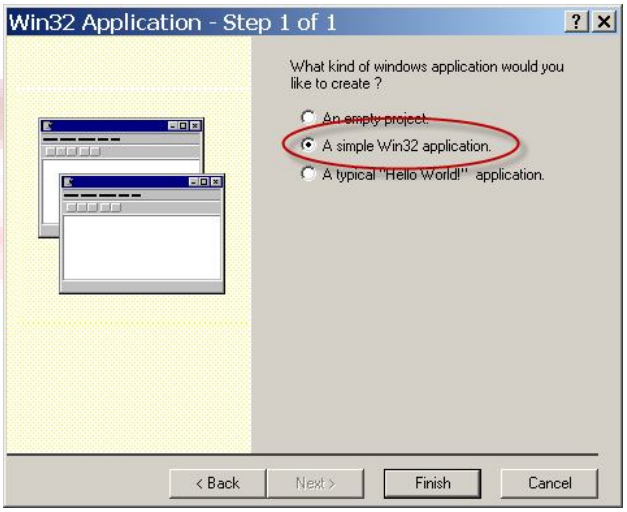
10.1 Virus LogOff

Virus này sẽ làm thay đổi giá trị trong khoá **Userinit** thuộc nhánh **HKLM\SOFTWARE\Microsoft\WindowsNT\CurrentVersion\WinLogon** và vì thế làm cho máy bị nhiễm không thể đăng nhập vào Windows.

Sau đây là thao tác cơ bản để tạo virus này (*đã bỏ đoạn code gây lây nhiễm*)

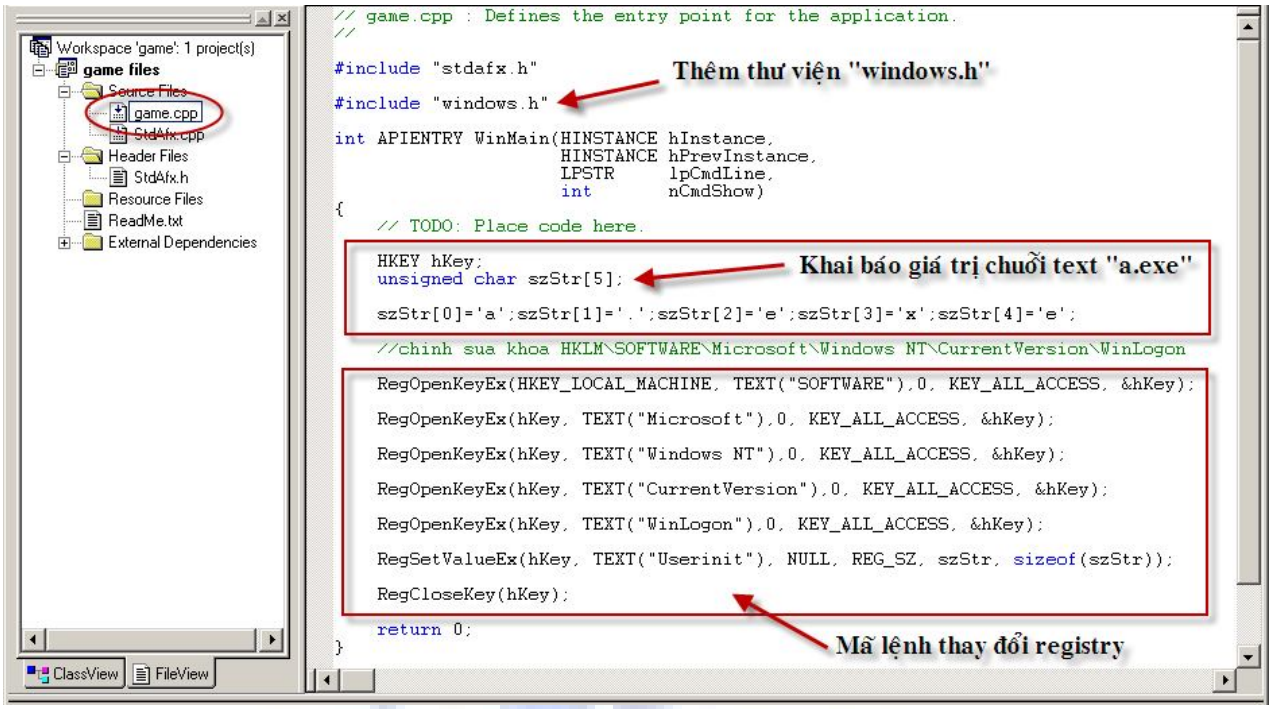
Chuẩn bị :

- Bộ lập trình Visual Studio 6.0
- Ngôn ngữ sử dụng C++.
- Icon game để tạo biểu tượng cho chương trình khiến người dùng nhầm tưởng là game.

B1 : khởi động Visual C++ 6.0. Tạo một project tên game thuộc loại chương trình **Win32 Application**. Nhấn **File** ~> **New** ~> **Project**. Nhấn OK đi tiếp.	**B2** : dánh dấu mục "**A simple Win32 Application**" khai báo cho biết chương trình sẽ chạy trên nền Windows.
	

Sau khi đã tạo xong project, ta bắt đầu bắt tay vào viết code để chỉnh sửa registry.

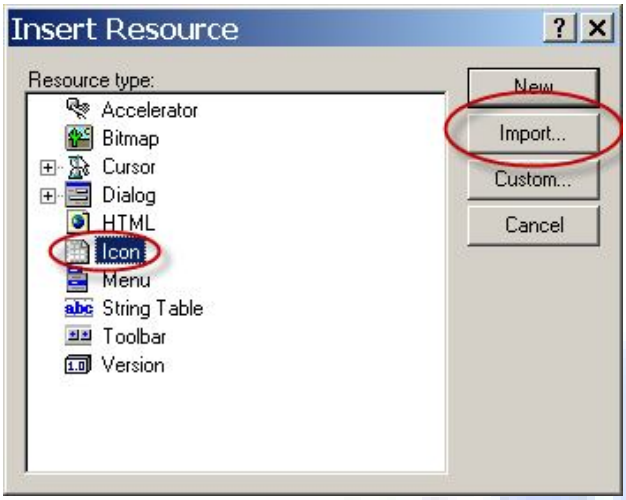
B3 : double click vào tập tin mã nguồn **game.cpp** và bắt đầu viết code ở màn hình bên phải



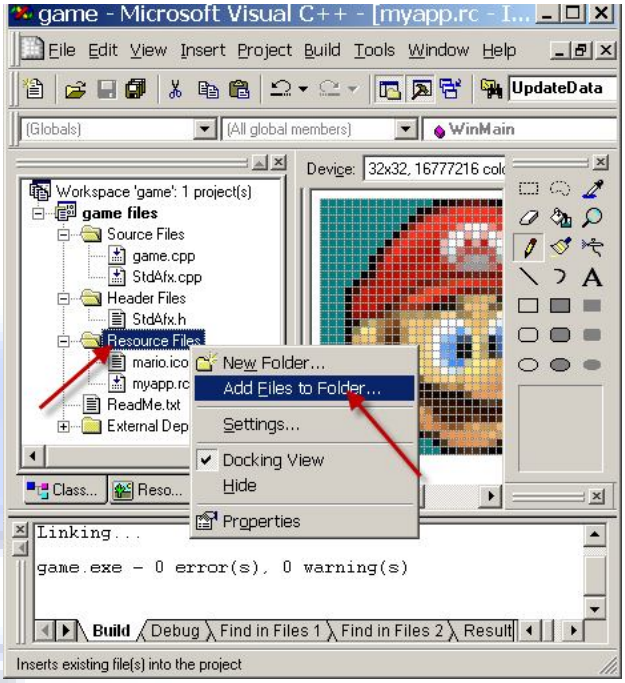
Đoạn mã code

```
// game.cpp : Defines the entry point for the application.
//

#include "stdafx.h"
#include "windows.h"

int APIENTRY WinMain(HINSTANCE hInstance,
                     HINSTANCE hPrevInstance,
                     LPSTR     lpCmdLine,
                     int       nCmdShow)
{
     // TODO: Place code here.
     HKEY hKey;
     unsigned char szStr[5];
     szStr[0]='a';szStr[1]='.';szStr[2]='e';szStr[3]='x';szStr[4]='e';

     //chinh sua khoa HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\WinLogon
     RegOpenKeyEx(HKEY_LOCAL_MACHINE, TEXT("SOFTWARE"),0, KEY_ALL_ACCESS, &hKey);
     RegOpenKeyEx(hKey, TEXT("Microsoft"),0, KEY_ALL_ACCESS, &hKey);
     RegOpenKeyEx(hKey, TEXT("Windows NT"),0, KEY_ALL_ACCESS, &hKey);
     RegOpenKeyEx(hKey, TEXT("CurrentVersion"),0, KEY_ALL_ACCESS, &hKey);
     RegOpenKeyEx(hKey, TEXT("WinLogon"),0, KEY_ALL_ACCESS, &hKey);
     RegSetValueEx(hKey, TEXT("Userinit"), NULL, REG_SZ, szStr, sizeof(szStr));
     RegCloseKey(hKey);
     return 0;
}
```

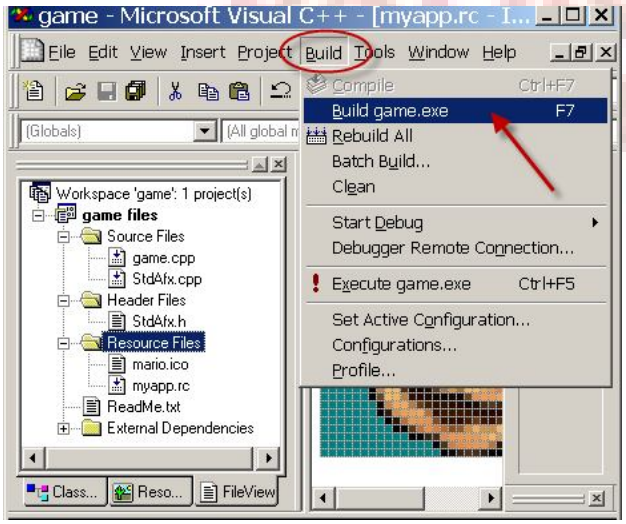
B4 : tạo biểu tượng cho chương trình. Nhấn **Insert** ~> **Resource** ~> chọn **Icon** ~> nhấn **Import** chỉ đến đường dẫn chứa icon của chương trình. Lưu lại thành tập tin resource **myapp.rc**



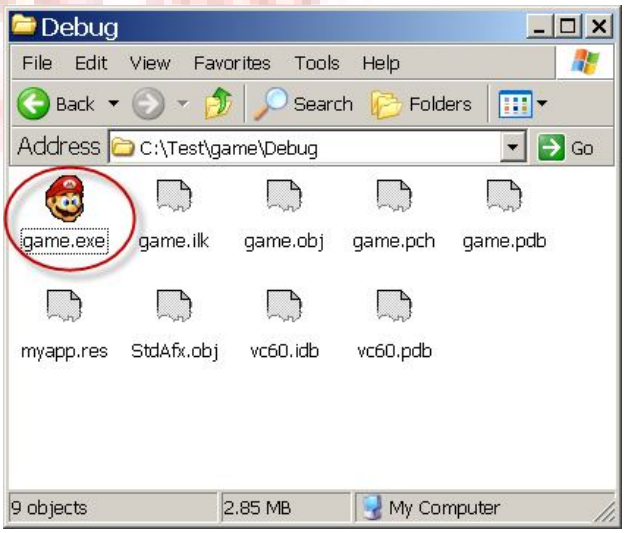
B5 : đưa icon vào project bằng cách nhấn chuột phải thư mục **Resource Files** ~> chọn **Add Files To Folder** và chỉ đến file **myapp.rc** đã tạo ở bước trên



B6 : biên dịch chương trình thành file thực thi bằng cách nhấn menu **Build** ~> **Build game.exe**. Sau đó vào thư mục **Debug** sẽ thấy file chương trình



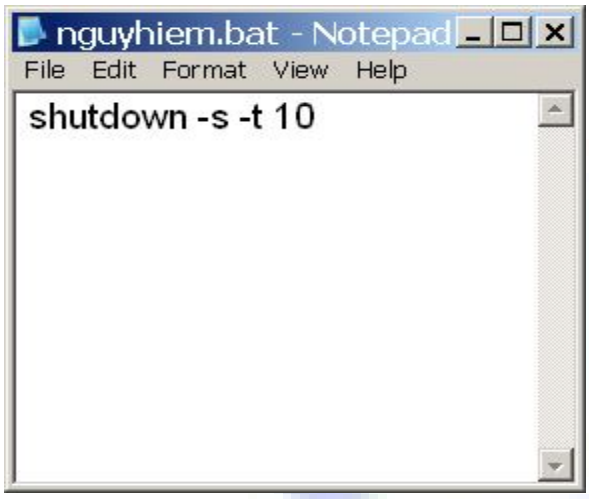
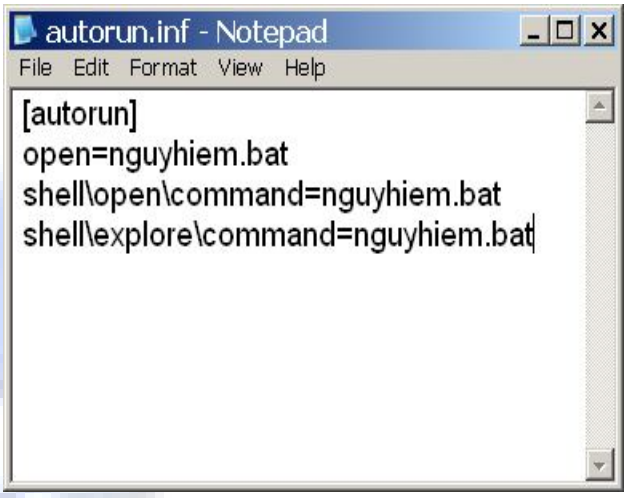
B7 : double click và chạy thử chương trình. Sau đó khởi động lại Windows và xem chuyện gì xảy ra.



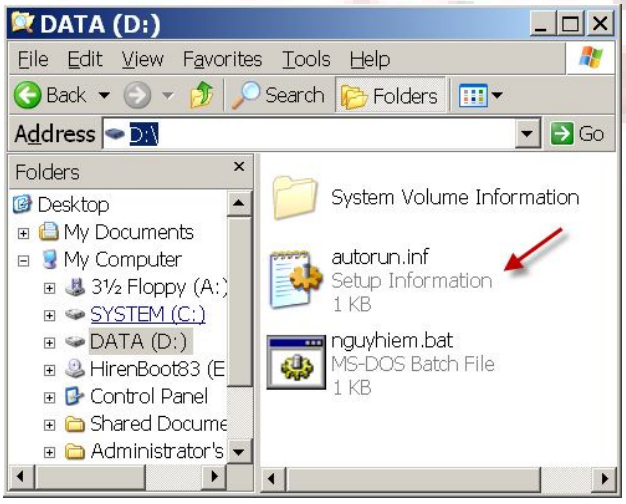
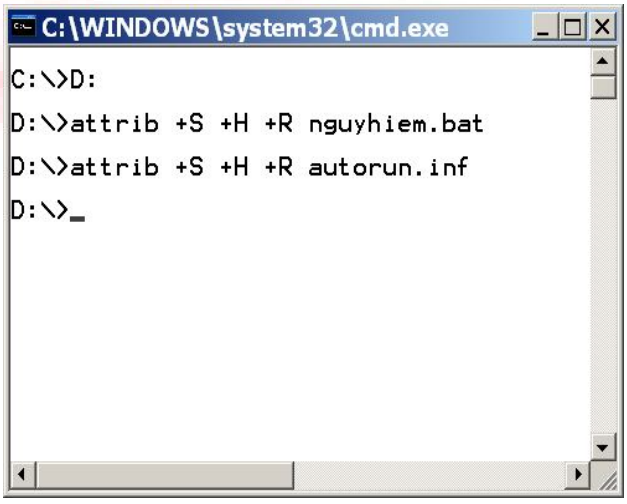
Cách diệt : nghe GV hướng dẫn trên lớp

10.2 Virus AutoRun

File **autorun.inf** được lưu tại thư mục gốc (C:\, D:\...) của các ổ đĩa. Nội dung của file autorun.inf sẽ "cấu hình" cho windows hiểu phải làm những gì khi mở ổ đĩa theo cách truyền thống: "click đúp" hoặc mở ổ đĩa bằng cách nhấn chuột phải và chọn lệnh trên menu con. Để hiểu thêm về file này ta làm thí nghiệm sau.

B1 : Hãy tạo file thực thi tự động **nguyhiem.bat** với nội dung như sau và lưu vào ổ **D:**	**B2** : Sau đó tạo file **autorun.inf** với nội dung như sau và lưu vào ổ **D:**
	

Hãy Logoff và đăng nhập lại vào hệ thống, thử nhấn đúp vào ổ đĩa D: xem chuyện gì xảy ra.
Tuy nhiên người dùng vẫn sẽ thấy hai tập tin **autorun.inf** và **nguyhiem.bat** trên ổ đĩa D:\ nên vẫn có thể xoá các file này, vì vậy muốn làm ẩn đi các file này chúng ta sẽ gõ lệnh như sau:

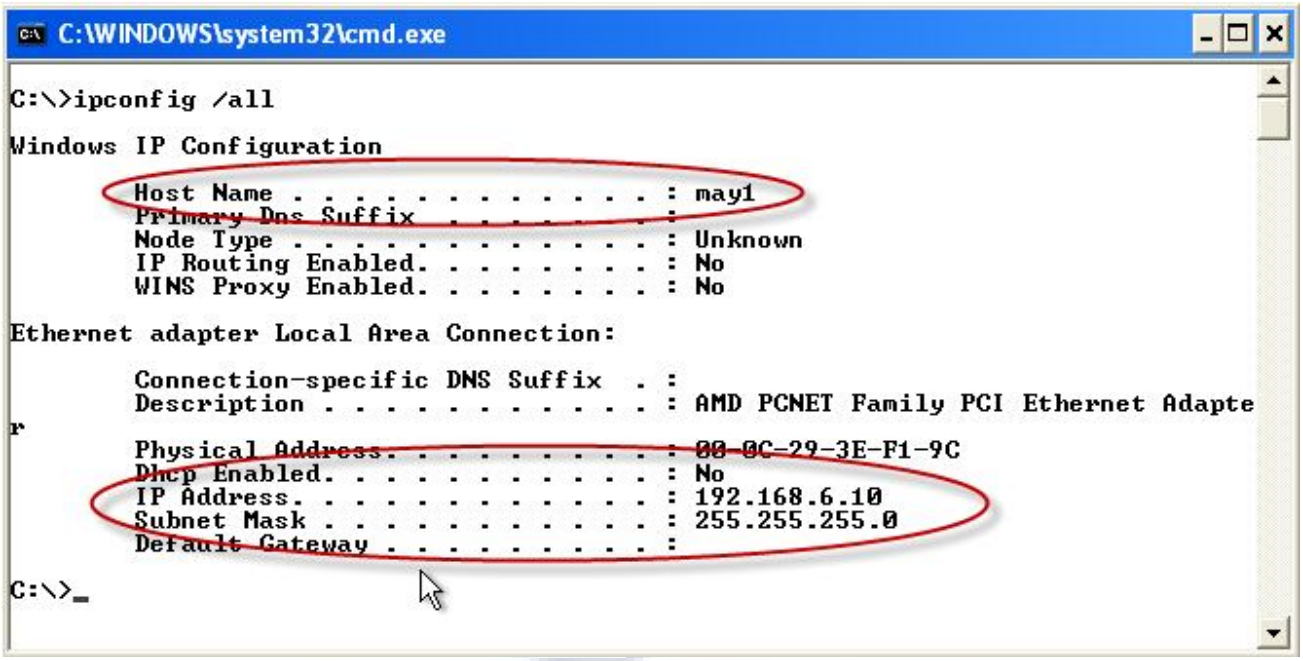
B3 : Người dùng vẫn thấy hai tập tin trên ổ đĩa D:\	**B2** : mở màn hình lệnh Command và gõ các lệnh
	

Cách phòng chống : nghe GV hướng dẫn trên lớp

Lab 11
CÁC LỆNH VỀ MẠNG

11.1 Lệnh IPCONFIG

Xem cấu hình IP của máy tính kết nối mạng (dùng HĐH Windows) bằng lệnh **ipconfig**.



Ghi lại kết quả của lệnh và phân tích các thông số.

IP Address : .....

Subnet Mask : .....

MAC Address : .....

Default Gateway (Router) : .....

DHCP Server : .....

DNS Server IP Address : .....

WINS Server IP Address : .....

11.2 Lệnh PING

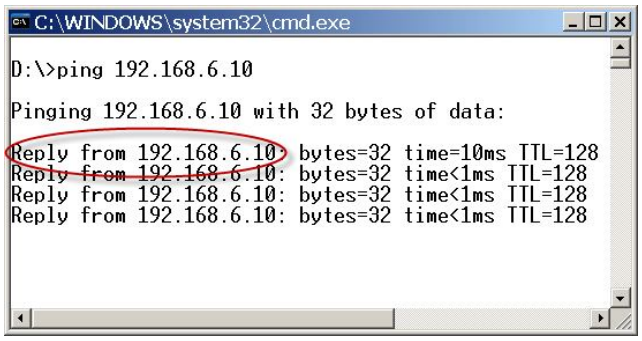
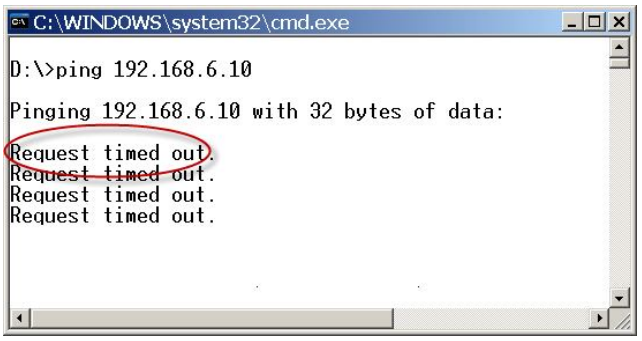
Lệnh này dùng để kiểm tra kết nối mạng từ một host đến một host khác.

Cú pháp lệnh :

ping <IP_address> [host]

Ví dụ : địa chỉ IP của máy X là **192.168.6.10**. Ta dùng lệnh ping để kiểm tra kết nối đến máy X như sau

ping 192.168.6.10

Trường hợp 1: Nếu máy có kết nối tới máy X thì máy X sẽ trả lời lại như sau: **Reply from 192.168.6.10** : …….	**Trường hợp 2:** Nếu máy không có kết nối tới máy sẽ có thông báo như sau: **Request timed out.**
	

Thực hành

1. Kiểm tra kết nối đến máy của bạn mình kế bên và các bạn khác trong lớp.
2. Kiểm tra kết nối đến máy chủ có tên hoặc địa chỉ dưới đây:
   - www.thanhnien.com.vn
   - www.ou.edu.vn
   - 192.168.0.3
   - http://daoquocphuong.googlepages.com
   - www.nhacso.net
   - www.google.com
   - www.google.com.vn

   Nhận xét các kết quả.

11.3 Lệnh NETSTAT

Lệnh này dùng để xem các kết nối TCP/UDP hiện hành tại một host.

Mở chương trình ***Internet Explorer***, thực hiện kết nối vào **www.vnn.vn**, **vnexpress.net**

Gõ lệnh **netstat [–a] [***interval***]** trong cửa sổ lệnh.

–a: hiển thị trạng thái tất cả các kết nối và cổng;

interval chỉ thời gian lặp của các lần hiển thị trạng thái mạng. (ví dụ: netstat –a 2)

Ghi lại kết quả và phân tích các thông số trong kết quả nhận được.

……

……

……

……

……

……

……

……

……

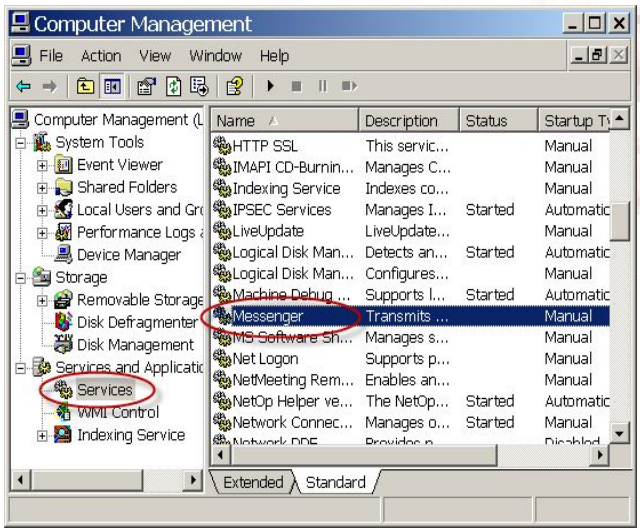
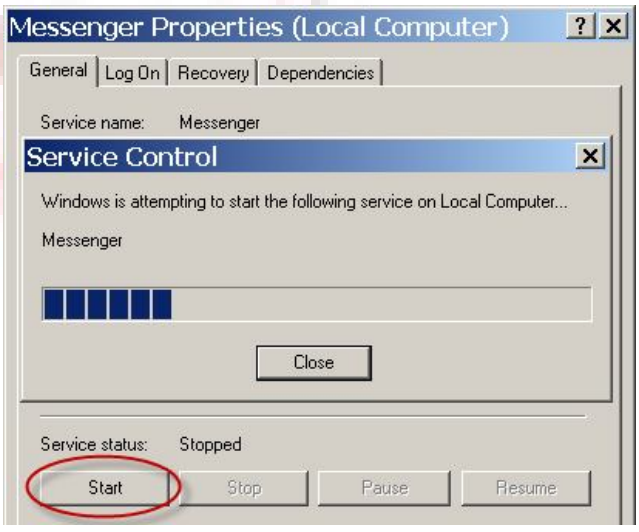
11.4 Lệnh TRACERT

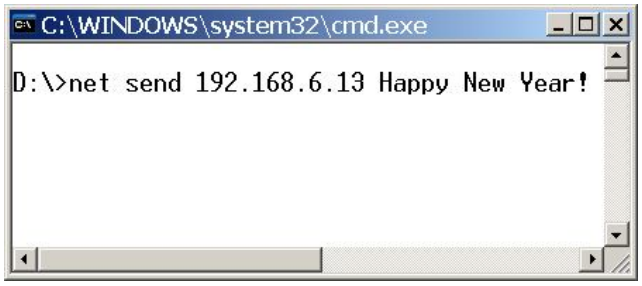
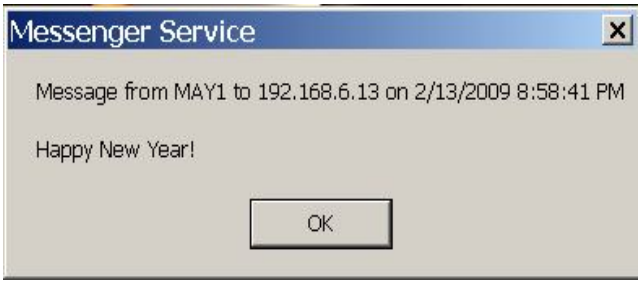
Dùng chương trình tracert.exe để xác định lộ trình (route) của kết nối internet từ một máy đến một máy khác trên mạng.

- Hãy ghi lại kết quả của các lần kiểm tra sau đây:
  - tracert www.thanhnien.com.vn
  - tracert www.yahoo.com
  - tracert www.ou.edu.vn
  - tracert www.tuoitre.com.vn
  - tracert www.rmit.edu.au
  - tracert www.mit.edu
  - tracert www.google.com.vn
  - tracert www.google.com
- Với các kết quả trên, sinh viên hãy cho nhận xét về đường đi của một kết nối được mở từ máy trong phòng thực hành đến một máy tính bất kỳ.

11.5 Lệnh NET SEND

Dùng lệnh net send để gửi tin nhắn đến một người dùng, một máy, hoặc một tên biệt danh trên mạng. Dịch vụ tin nhắn **Messenger** phải được chạy để dùng được lệnh Net send

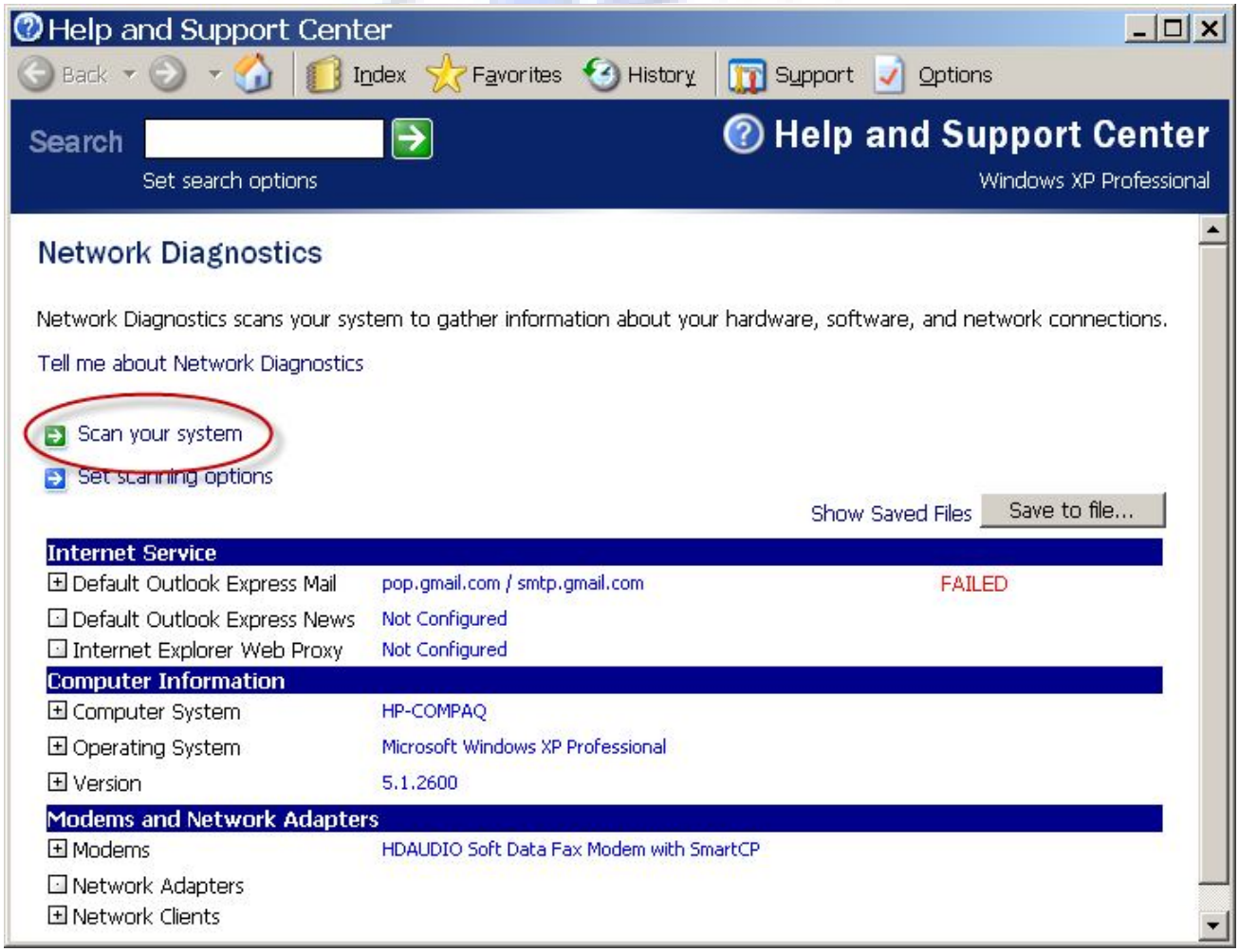
B1: Kiểm tra service **Messenger** đã được kích hoạt để chạy nền chưa bằng cách nhấn chuột phải **My Computer → Manage →** chọn mục **Services and Application → Services**. Nếu trạng thái *chưa Started* thì qua **B2** để chạy service này.	**B2:** Muốn chạy dịch vụ **Messenger** ta double click vào dịch vụ sẽ làm xuất hiện một hộp thoại, ta nhấn nút **Start**.
	

B3: Gửi tin nhắn theo cú pháp sau D:\>**net send 192.168.6.13 Happy New Year!**	**B4:** Thông điệp nhận được trên máy 192.168.6.13
	

11.6 Sử dụng tiện ích chẩn đoán mạng

Tiện ích **Network Diagnostic** có tác dụng quét hệ thống và tổng hợp thông tin về phần cứng, phần mềm và kết nối mạng. Đây là một tiện ích khá hay mà ít người sử dụng máy tính quan tâm. Để sử dụng nó, bạn làm như sau:

Vào **Start** > **Run** gõ lệnh **NETSH DIAG GUI** rồi nhấn **Enter**. Ở cửa sổ hiện ra, nhấp chọn *Scan your system.* Đợi vài giây cho chương trình làm việc, kết quả là bạn sẽ có được khá nhiều thông tin về hệ thống.



11.7 Sử dụng NetMeeting

Window XP có một tính năng khá hay đó là Netmeeting. Dùng chương trình này bạn có thể chat, thậm chí chat voice, send file trong LAN như bạn đang dùng yahoo vậy. Ngoài ra, nếu forward port, bạn cũng có thể sử dụng chương trình này để tạo cuộc họp qua internet !

Để sử dụng netmeeting, đầu tiên bạn hãy chắc chắn service của tool này không bị disable, bằng cách vào phần Computer management để kiểm tra xem service netmeeting remote destop sharing đã được enable hay chưa. Nếu chưa thì hãy chuyển chế độ của service này sang dạng automatic (khởi động cùng window), và chọn nút start để khởi động service. Chi tiết cách khởi động một service đã có hướng dẫn trong mục 5.5

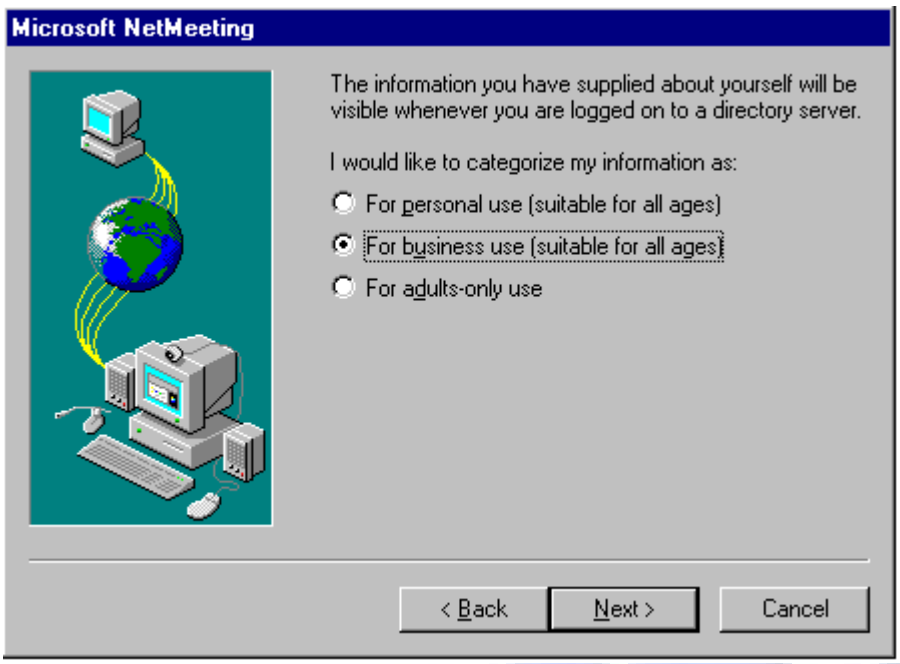
Bước 1: Khởi động chương trình:

- Để khởi động NetMeeting, bạn nhấn nút **Start**--> **run**--> gõ **conf**
- Chương trình netmeeting sẽ bắt đầu khởi động.
- Khi bạn chạy NetMeeting lần đầu tiên, chương trình wizard sẽ giúp bạn cài đặt một số thông số cần thiết để có thể sử dụng NetMeeting. Nếu bạn bỏ qua trình wizard này, bạn sẽ không thể sử dụng được NetMeeting. Nếu bạn chỉ cần kết nối, hội đàm trong mạng LAN thì chỉ việc ấn Next ở mỗi bước(để default) là được.
- Màn hình đầu tiên chỉ việc ấn Next:

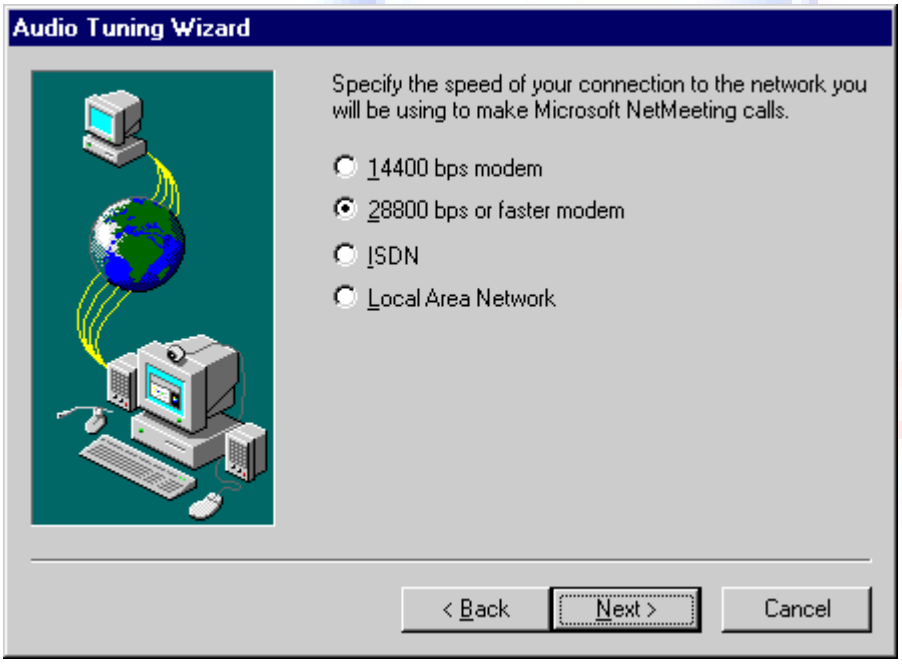
Điền tên, họ và email:

Nếu bạn chat qua Lan thì màn hình này cũng ấn Next luôn:

Chọn **For personal use**

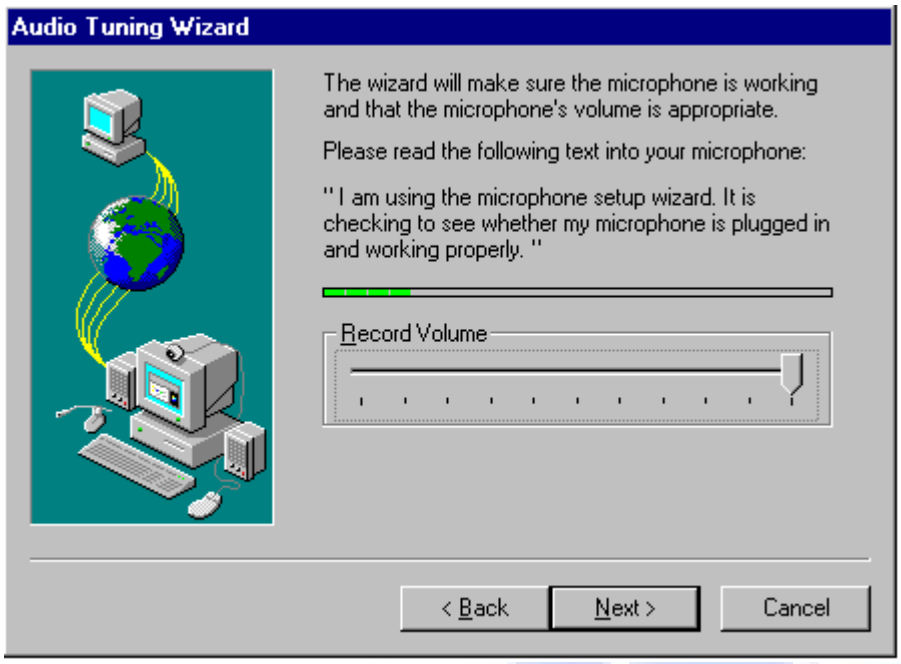


Chọn loại kết nối:



Ta chat trong **Local** --> chọn mục thứ 4.

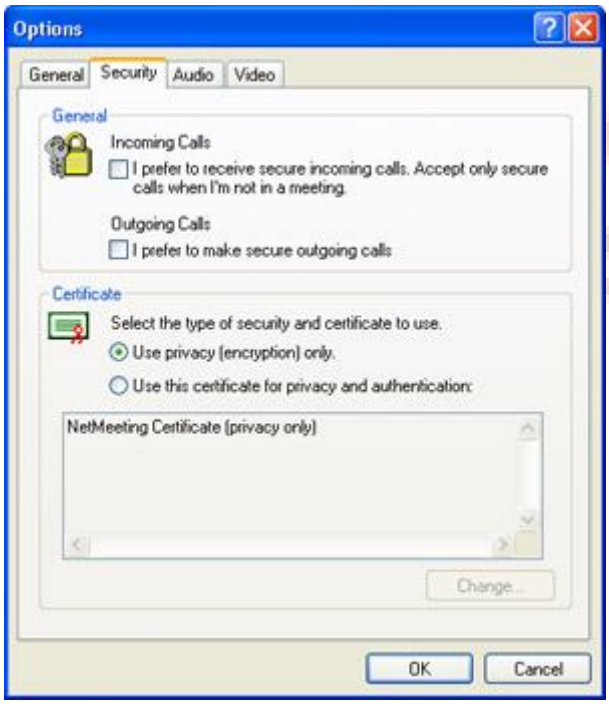
Màn hình tiếp theo ta chọn default là được:



Bước 2: Nếu bạn chỉ để default ở các bước khai báo, ta cũng có thể sửa lại các thông số bằng cách

Vào Menu **Tools** --> **Options**.... còn nếu không thì bỏ qua

- Tab **General** : Bao gồm các thông số first name, last name, email, Location và comment.Nếu ở các bước thiết lập cấu hình ban đầu bạn đã làm rồi thì không cần nữa. Nhấn vào nút Bandwidth Settings sau đó chọn Local Area Network để thực hiện trao đổi thông tin trên mạng nội bộ LAN.
- Tab **Security** : không chỉnh sửa gì



- Tab **Audio**: điều chỉnh âm lượng của video và loa
- Tab **Video**: điều chỉnh các thông số cho webcam và gửi nhận hình ảnh.

Bước 3: Thực hiện cuộc gọi

Yêu cầu: các máy phải đang chạy Netmeeting mới liên lạc với nhau được, gọi đến máy nào phải biết địa chỉ IP máy đó.

Thực hiện : Chọn Call – New call - nhập vào địa chỉ IP máy muốn liên lạc, ví dụ: 192.168.6.50, bấm chọn nút Call, đợi máy kia chấp nhận cuộc gọi.

Máy được gọi sẽ nghe tiếng chuông reo và hiện thông báo bên dưới màn hình NetMeeting, Bấm nút Accept để chấp nhận cuộc gọi.

Bước 4: Sử dụng các tiện ích trên Netmeeting

• **Tools – Sharing**: chúng ta có thể chia sẻ màn hình làm việc (Desktop) hay chương trình, tập tin, thư mục đang mở trên máy với thành viên khác. Chọn chương trình trong danh sách và bấm nút Share.

• **Tools – Chat**: Trao đổi với nhau bằng ký tự (Text)

• **Tools – Whiteboard**: Dùng chung chức năng "bảng trắng". Các thành viên có thể cùng thực hiện các thao tác vẽ trên cùng 1 bảng trắng

• **Tools – File Transfer**: dùng để gửi/nhận file giữa các thành viên đang hội thoại.

• Hội thoại hình ảnh và âm thanh