



Operating Systems

NỘI DUNG

- Chương 1: Tổng quan
- Chương 2: Quản lý tiến trình
- Chương 3: Deadlock
- Chương 4: Quản lý bộ nhớ
- Chương 5: Hệ thống file
- Chương 6: Quản lý nhập xuất

Tài liệu tham khảo

- [1] Andrew S. Tanenbaum, Modern Operating Systems, Pearson, 2015
- [2] Abraham Silberschatz, Peter Baer Galvin, Greg Gagne, Operating System Concepts, John Wiley, 2013
- [3] William Stallings, Operating Systems: Internals and Design Principles, Pearson, 2015

3

Chương 1

Tổng quan



Nội dung

- Tổng quan về hệ thống máy tính
- Tổng quan về hệ điều hành

5

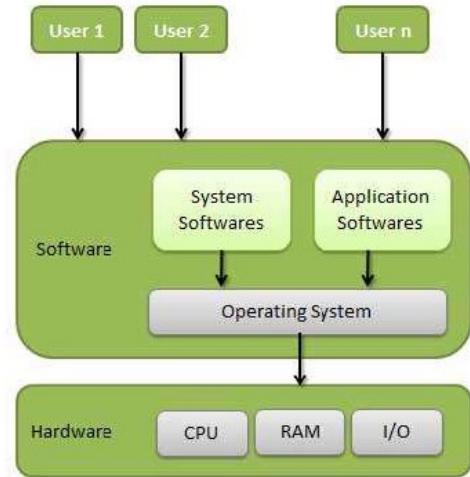
Tài liệu tham khảo

- Andrew S. Tanenbaum, Modern Operating Systems, Pearson, 2015
 - Chương 1
- Abraham Silberschatz, Peter Baer Galvin, Greg Gagne, Operating System Concepts, John Wiley, 2013
 - Chương 1,2

6

1. Tổng quan về hệ thống máy tính

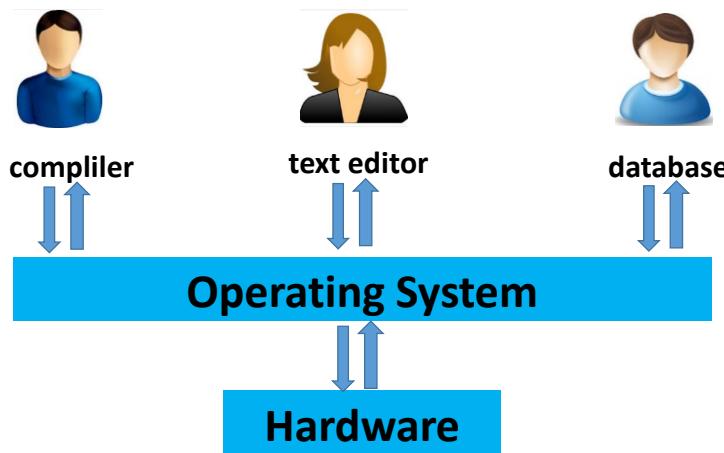
- Một hệ thống máy tính gồm có
 - Phần cứng
 - Hệ điều hành
 - Các chương trình ứng dụng
 - Người sử dụng



7

Tổng quan về hệ thống máy tính (tt)

- Bốn thành phần của hệ thống máy tính



8

Tổng quan về hệ thống máy tính (tt)

- Phần cứng: cung cấp các tài nguyên tính toán cơ bản: CPU, bộ nhớ, các thiết bị nhập xuất
- Hệ điều hành: điều khiển và phối hợp việc sử dụng phần cứng cho những ứng dụng khác nhau và người dùng khác nhau
- Chương trình ứng dụng: chương trình dịch, hệ cơ sở dữ liệu, game,... sử dụng tài nguyên của máy tính để giải quyết yêu cầu của người sử dụng

9

2. Tổng quan về Hệ điều hành

- Khái niệm về hệ điều hành
- Các chức năng cơ bản
- Lịch sử phát triển của hệ điều hành
- Phân loại hệ điều hành

10

2.1 Khái niệm về Hệ điều hành

• Hệ điều hành là một chương trình/hệ chương trình hoạt động giữa người sử dụng (user) và phần cứng của máy tính.

• Mục tiêu của HĐH:

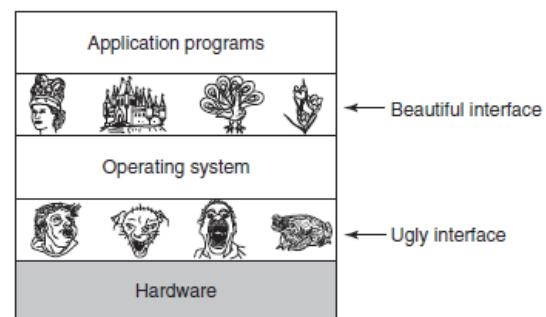
- Làm cho máy tính dễ sử dụng và hiệu quả hơn.
- Quản lý và cấp phát tài nguyên hệ thống một cách có hiệu quả

11

Khái niệm về Hệ điều hành (tt)

• Bản chất của Hệ điều hành:

- HĐH là một máy tính mở rộng
 - HĐH cung cấp khả năng trừu tượng hóa
→ giúp thao tác của người dùng dễ dàng hơn mà không cần quan tâm đến kiến trúc phức tạp của phần cứng



- HĐH là một hệ thống quản lý tài nguyên

• HĐH phải có chức năng quản lý, cấp phát tài nguyên để các chương trình có thể hoạt động chính xác và hiệu quả

Ví dụ: máy in

12

2.2 Các chức năng cơ bản của HĐH

- Cung cấp một môi trường để người sử dụng có thể thực thi các chương trình
- Điều khiển và phối hợp việc sử dụng phần cứng
- Cung cấp các dịch vụ cơ bản cho các ứng dụng
- Phân chia thời gian xử lý, định thời CPU
- Phối hợp và đồng bộ hoạt động giữa các tiến trình
- Quản lý tài nguyên hệ thống (bộ nhớ, file,...)
- Kiểm soát lỗi, bảo vệ

13

2.3. Phân loại hệ điều hành

1. Hệ điều hành dành cho máy MainFrame (MainFrame OS)
2. Hệ điều hành dành cho máy Server (Server OS)
3. Hệ điều hành dành cho máy nhiều CPU (Multiprocessor OS)
4. Hệ điều hành dành cho máy tính cá nhân (PC)
5. Hệ điều hành dành cho thiết bị cầm tay (Handheld Computer OS)
6. Hệ điều hành nhúng (Embedded OS)
7. Hệ điều hành dành cho các nút cảm biến (Sensor-Node OS)
8. Hệ điều hành đáp ứng thời gian thực (Real-Time OS)
9. Hệ điều hành dành cho thẻ chíp (SmartCard OS)

14

2.3.1 Hệ điều hành dành cho máy MainFrame

- HĐH dành cho máy tính lớn, tốc độ cao, thường được dùng trong các trung tâm dữ liệu lớn của công ty
- Số lượng I/O rất lớn, có thể đến hàng ngàn đĩa và hàng triệu gigabyte dữ liệu
- Thường được dùng làm máy chủ Web với quy mô lớn
- Xử lý nhiều công việc cùng một lúc, cung cấp ba loại dịch vụ:
 - Xử lý hàng loạt: xử lý các tác vụ thường ngày mà không có sự tương tác với người dùng
 - Xử lý giao dịch: xử lý hàng nghìn các yêu cầu nhỏ trong mỗi giây
 - Chia sẻ thời gian: cho phép nhiều người dùng từ xa thực thi trên máy tính cùng một thời điểm (truy vấn một cơ sở dữ liệu lớn)

15

2.3.2 Hệ điều hành dành cho máy Server

- HĐH dùng cho máy chủ cung cấp các dịch vụ qua mạng (truy cập tài nguyên, sử dụng dịch vụ,...)
- Máy tính sử dụng HĐH này phải có cấu hình phần cứng và tốc độ xử lý cao
- Các hệ điều hành server phổ biến là Solaris, FreeBSD, Linux và Windows Server 20xx

16

2.3.3 Hệ điều hành dành cho máy nhiều CPU

- Dùng cho các hệ thống có nhiều CPU
- Các CPU cùng chia sẻ hệ thống đường truyền dữ liệu, đồng hồ, bộ nhớ, các thiết bị ngoại vi
- **Ưu điểm:**
 - Nhiều bộ xử lý song song → nhanh
 - Độ tin cậy cao:
 - Các chức năng được xử lý trên nhiều bộ xử lý
 - Một bộ xử lý hỏng sẽ không ảnh hưởng đến toàn bộ hệ thống.

17

2.3.4 Hệ điều hành dành cho máy tính cá nhân

- Hỗ trợ đa chương trình.
- Được xây dựng đáp ứng cho nhu cầu sử dụng của người dùng cá nhân: xử lý văn bản, bảng tính, trò chơi, truy cập Internet,...
- Các HĐH phổ biến:
 - Linux: Ubuntu, Fedora, Debian,...
 - Windows 7, 8, 10
 - Apple's OS

18

2.3.5 Hệ điều hành dành cho thiết bị cầm tay

- HĐH dùng cho PDA (Personal Digital Assistant)
 - Máy tính bảng
 - Điện thoại thông minh
- Hầu hết các thiết bị đầu có CPU đa lõi, hỗ trợ các chức năng cảm biến, định vị (GPS), kết nối không dây, máy ảnh và nhiều ứng dụng của bên thứ ba
- Các HĐH phổ biến:
 - Android của Google
 - iOS của Apple

19

2.3.6 Hệ điều hành nhúng

- HĐH chạy trên các thiết bị điều khiển.
 - Lò vi sóng
 - Tivi
 - Ô tô
 - Các thiết bị điều khiển thông minh khác
- Các phần mềm được cài đặt sẵn trong ROM và người dùng không thể tải và cài đặt phần mềm trên HĐH này
- → Không cần bảo vệ các ứng dụng → đơn giản hóa thiết kế.
- Phổ biến:
 - Embedded Linux
 - QNX
 - VxWorks

20

2.3.7 Hệ điều hành dành cho các nút cảm biến

- Các nút cảm biến là các máy tính nhỏ kết nối với nhau sử dụng giao tiếp không dây.
- Mỗi nút cảm biến là một máy tính có CPU, RAM, ROM, và một hoặc nhiều cảm biến môi trường, chạy bằng pin có tích hợp radio, công suất hạn chế, hoạt động liên tục (thường là ngoài trời).
- HĐH loại này hoạt động dựa trên sự phản hồi các sự kiện bên ngoài hoặc thực hiện các phép đo định kỳ dựa trên đồng hồ bên trong.
- **Đặc điểm:**
 - Nhỏ gọn, đơn giản, tốn ít bộ nhớ RAM, tiết kiệm pin
 - Các chương trình đều được cài sẵn
 - Ví dụ: TinyOS.

21

2.3.8 Hệ điều hành đáp ứng thời gian thực

- HĐH đáp ứng yêu cầu cao về tốc độ, độ chính xác của CPU.
- Sử dụng trong các thiết bị chuyên dụng như: điều khiển các thử nghiệm khoa học, điều khiển trong y khoa, dây chuyền công nghiệp, thiết bị gia dụng, quân sự,...
- HĐH phải được viết tốt, thời gian xử lý nhanh, cho kết quả chính xác trong khoảng thời gian có hạn định.
- **Hệ điều hành thời gian thực cứng:**
 - Công việc được hoàn tất đúng lúc.
 - Hạn chế (hoặc không có) bộ nhớ phụ, tất cả dữ liệu nằm trong bộ nhớ chính (RAM/ROM).
 - Thường sử dụng trong điều khiển công nghiệp, robotics
- **Hệ điều hành thời gian thực mềm:**
 - Mỗi công việc có một độ ưu tiên riêng và sẽ được thi hành theo độ ưu tiên đó.
 - Thường sử dụng trong một số lĩnh vực như multimedia,....

22

2.3.9 Hệ điều hành dành cho thẻ chíp

- HĐH nhỏ, được dùng trong các smart card.
- Hạn chế về bộ nhớ và tốc độ xử lý
- Một số thẻ sử dụng Java: bộ nhớ ROM được cài đặt một trình thông dịch cho Máy ảo Java (JVM). Các Java applets được tải xuống thẻ và được thông dịch bởi trình thông dịch JVM.
- Đặc điểm:
 - Multiprogramming
 - Schedule
 - Resource management
 - Protection

23

2.4 Lịch sử phát triển của hệ điều hành

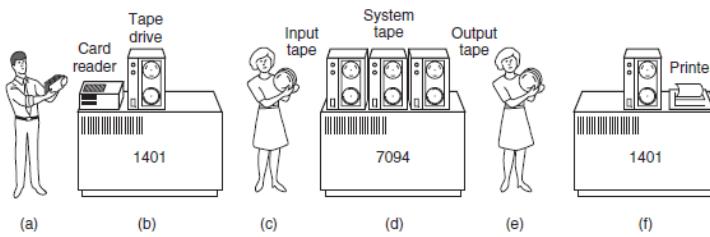
- Thế hệ 1 (1945 – 1955)
 - Máy tính dùng đèn điện tử chân không (Vacuum Tubes)
 - Mỗi máy được một nhóm thực hiện từ a→z (thiết kế, xây dựng, lập trình, vận hành, bảo trì).
 - Chưa có ngôn ngữ lập trình mà sử dụng ngôn ngữ máy, sử dụng bảng điều khiển
 - 1950: sử dụng phiếu đục lỗ để viết/đọc chương trình
 - Chủ yếu là xử lý các tính toán số học

24

Lịch sử phát triển của hệ điều hành (tt)

• Thế hệ 2 (1955 – 1965)

- Thiết bị bán dẫn (Transistors and Batch Systems) → tin cậy hơn.
- Phân chia: thiết kế, xây dựng, vận hành, lập trình, bảo trì.
- Mainframe
- Viết chương trình trên giấy (hợp ngữ, FORTRAN) → đục lỗ trên phiếu → đưa phiếu vào máy → xuất kết quả ra máy in.

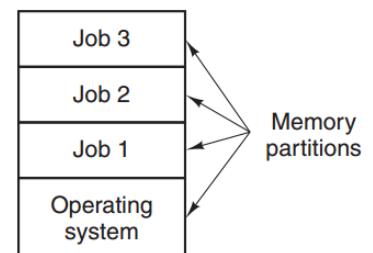


25

Lịch sử phát triển của hệ điều hành

• Thế hệ 3 (1965 – 1980)

- Máy IBM 360 ra đời, tiếp theo là hàng loạt các dòng 370, 4300, 3080, 3090,... sử dụng mạch tích hợp (IC).
- Kích thước, giá giảm
- Các chương trình được viết cho một máy có thể chạy trên tất cả các máy khác.
- Nhiều thiết bị ngoại vi → thao tác điều khiển phức tạp hơn.
- Các hệ điều hành phải có chức năng điều phối, giải quyết các yêu cầu tranh chấp thiết bị.
 - Hệ điều hành đa chương.
 - Hệ điều hành chia sẻ thời gian (CTSS của MIT)
 - MULTICS, UNIX, hệ thống các máy mini (DEC PDP-1).

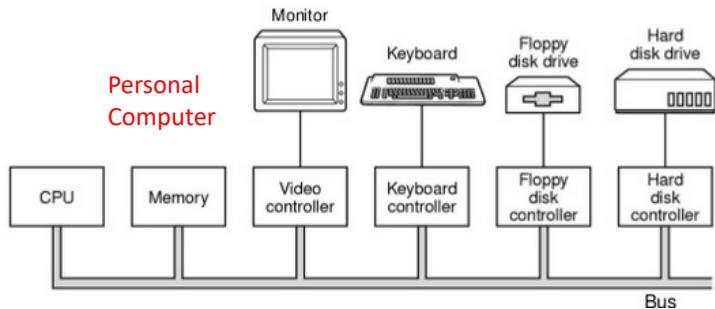


26

Lịch sử phát triển của hệ điều hành

• Thế hệ 4 (từ 1980)

- Máy tính cá nhân (Personal Computers)
- Hệ thống IBM PC với hệ điều hành MS-DOS và Windows sau này.
- Các hệ điều hành tựa Unix phát triển mạnh trên nhiều hệ máy khác nhau như Linux.
- Hệ điều hành mạng
- Hệ điều hành phân tán.



27

Lịch sử phát triển của hệ điều hành

• Thế hệ 5 (từ 1990)

- Hệ điều hành cho thiết bị di động (Mobile Computers)
- Trong thập niên 90:
 - N9000 (Nokia): kết hợp giữa điện thoại và PDA (Personal Assistant Digital)
 - GS88 (Ericsson)
 - Symbian OS (Samsung, Sony Ericsson, Motorola, Nokia)
- Thời điểm hiện tại:
 - IOS (Apple)
 - Android (Google)
 - Windows Phone (Microsoft)

28

Chương 2

Quản lý tiến trình



29
www.cungnhanlaptrinh.com

Nội dung

1. Tiến trình (process)
2. Luồng (thread)
3. Truyền thông giữa các tiến trình
4. Đồng bộ hóa tiến trình
5. Điều phối tiến trình.

Tài liệu tham khảo

- Andrew S. Tanenbaum, Modern Operating Systems, Pearson, 2015
 - Chương 2
- Abraham Silberschatz, Peter Baer Galvin, Greg Gagne, Operating System Concepts, John Wiley, 2013
 - Chương 3, 4, 5, 6
- William Stallings, Operating Systems: Internals and Design Principles, Pearson, 2015
 - Chương 3, 4

31

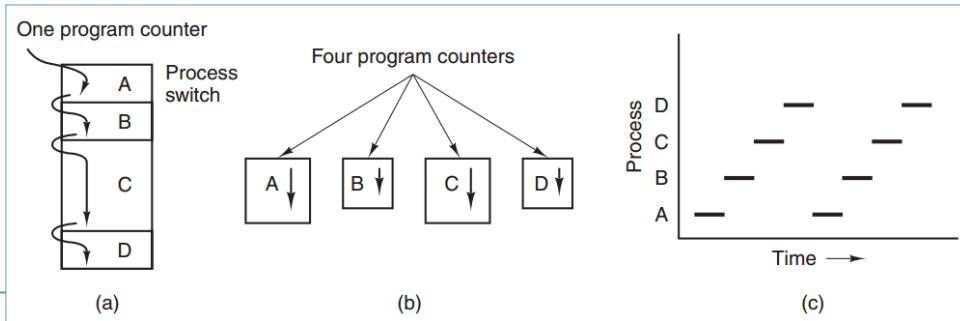
2.1. Tiến trình

- Mô hình
- Hiện thực

32

2.1.1 Mô hình Tiến trình

- Chương trình phần mềm khi thực thi phải được nạp vào bộ nhớ chính
- CPU đọc các chỉ thị từ bộ nhớ chính để xử lý
- Hệ thống máy tính hiện đại cho phép nhiều chương trình được nạp vào bộ nhớ và thực hiện đồng thời → cần có cơ chế kiểm soát hoạt động của các chương trình khác nhau → **khái niệm tiến trình (process)**



33

Mô hình Tiến trình (tt)

- Tiến trình là một chương trình đang được thực thi
- Một tiến trình cần sử dụng các tài nguyên:
 - CPU
 - Bộ nhớ
 - Tập tin
 - Thiết bị nhập xuất
 - ...

Task Manager					
Processes Performance App history Startup Users Details Services					
Name	Status	26% CPU	40% Memory	3% Disk	0% Network
Apps (5)					
> Foxit Reader 7.0, Best Reader for...		3.2%	46.6 MB	0 MB/s	0 Mbps
> Google Chrome (5)		0.6%	527.6 MB	0 MB/s	0 Mbps
> Microsoft PowerPoint		12.7%	279.0 MB	0 MB/s	0 Mbps
> Task Manager		0.4%	27.9 MB	0 MB/s	0 Mbps
> Windows Explorer		0.9%	53.8 MB	0 MB/s	0 Mbps
Background processes (123)					
> Adobe Genuine Software Integri...		0%	2.5 MB	0 MB/s	0 Mbps
> Adobe Genuine Software Servic...		0%	2.8 MB	0 MB/s	0 Mbps
> Adobe Update Service (32 bit)		0%	0.7 MB	0 MB/s	0 Mbps
> AlpsAlpine Pointing-device Driver		0%	0.6 MB	0 MB/s	0 Mbps
> AlpsAlpine Pointing-device Driver		0%	1.9 MB	0 MB/s	0 Mbps
> AlpsAlpine Pointing-device Driv...		0%	0.6 MB	0 MB/s	0 Mbps

34

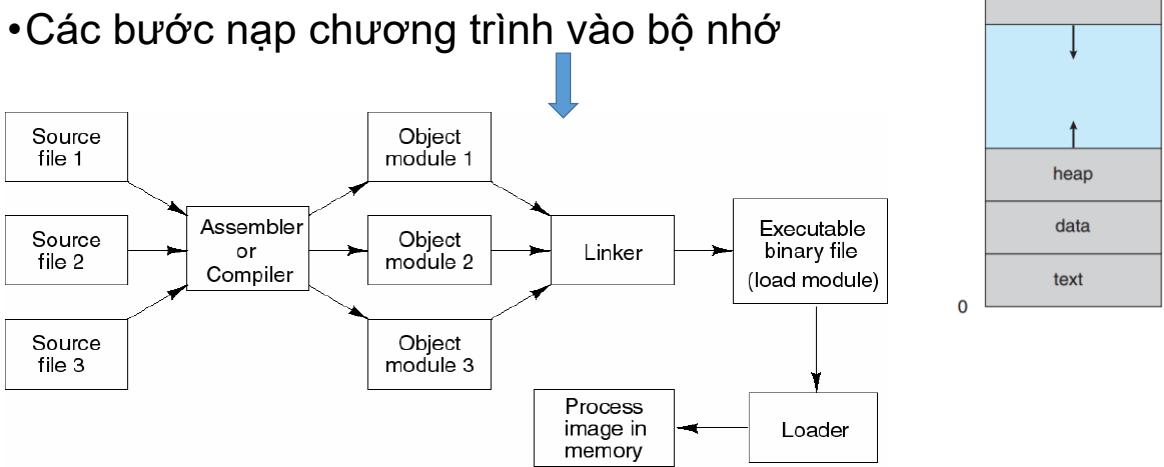
Mô hình Tiến trình (tt)

- Một tiến trình bao gồm:
 - Text section (program code), data section (chứa global variables)
 - program counter (PC), process status word (PSW), stack pointer (SP), memory management registers,...
- Phân biệt chương trình và tiến trình:
 - Chương trình: là một thực thể thụ động (tập tin có chứa một danh sách các lệnh được lưu trữ trên đĩa - file thực thi)
 - Tiến trình: là một thực thể hoạt động, với một bộ đếm chương trình xác định các chỉ lệnh kế tiếp để thực hiện và một số tài nguyên liên quan.
 - Một chương trình sẽ trở thành một tiến trình khi một tập tin thực thi được nạp vào bộ nhớ.

35

Mô hình Tiến trình (tt)

- Tiến trình trong bộ nhớ
- Các bước nạp chương trình vào bộ nhớ



36

2.1.2 Hiện thực Tiến trình

- Vai trò của hệ điều hành trong quản lý tiến trình
- Tạo tiến trình
- Các trạng thái của tiến trình
- Các thao tác với tiến trình
- Thực thi các tiến trình
- PCB – Process Control Block

37

Vai trò của hệ điều hành trong quản lý tiến trình

- Tạo và hủy tiến trình của người dùng và tiến trình hệ thống
- Dừng, khôi phục (resume) tiến trình
- Cung cấp các cơ chế để đồng bộ hóa tiến trình
- Cung cấp các cơ chế liên lạc giữa các tiến trình
- Cung cấp cơ chế xử lý bế tắc (deadlock)

38

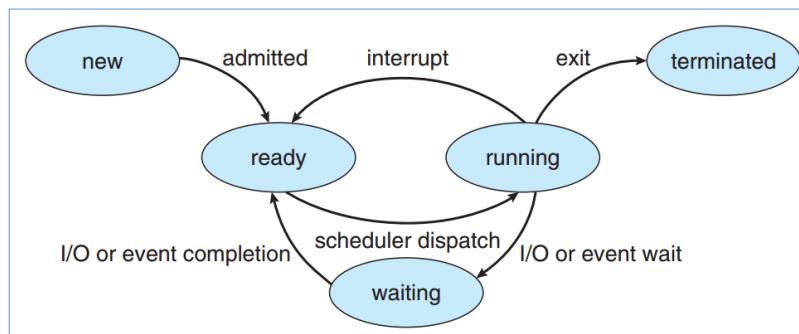
Tạo tiến trình

- Các bước hệ điều hành khởi tạo tiến trình:
 - Cấp phát định danh duy nhất cho tiến trình (process number, process identifier, pid)
 - Cấp phát không gian nhớ để nạp tiến trình
 - Khởi tạo khối dữ liệu Process Control Block (PCB) lưu các thông tin về tiến trình được tạo.
 - Thiết lập các mối liên hệ cần thiết (vd: sắp PCB vào hàng đợi định thời,...)

39

Các trạng thái của tiến trình

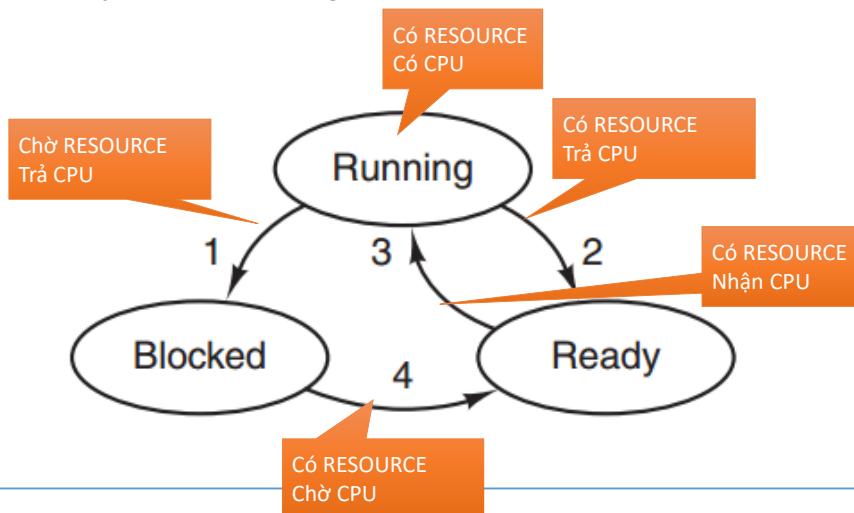
- New: tiến trình đang được tạo lập.
- Running: các chỉ thị của tiến trình đang được xử lý.
- Waiting: tiến trình chờ được cấp phát một tài nguyên, hay chờ một sự kiện xảy ra.
- Ready: tiến trình chờ được cấp phát CPU.
- Terminated: tiến trình kết thúc xử lý.



40

Các trạng thái của tiến trình (tt)

- Quá trình chuyển đổi trạng thái của tiến trình



41

Các trạng thái của tiến trình (tt)

- Ví dụ về quá trình chuyển đổi trạng thái của tiến trình

- Cho đoạn chương trình C trên Linux:


```
/* test.c */
int main(int argc, char** argv)
{
    printf("Hello \n");
    exit (0);
}
```
- Biên dịch
 - gcc test.c –o test
- Thực thi chương trình test
 - ./test
- Tiến trình test được tạo ra, thực thi và kết thúc.
- Trạng thái của tiến trình Test:
 - new → ready → running → waiting (do chờ I/O khi gọi printf) → ready →running → terminated

42

Các thao tác với tiến trình

- **Tiến trình được tạo khi:**

- Hệ thống khởi động
- Một tiến trình đang chạy thực hiện lời gọi hệ thống tạo tiến trình mới.
- Người dùng tạo
- Bắt đầu một batch job

43

Các thao tác với tiến trình (tt)

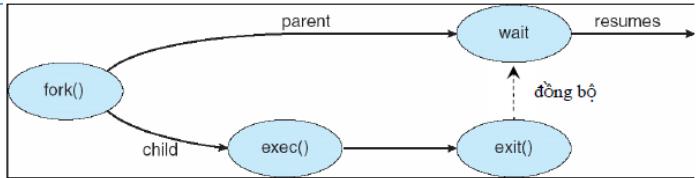
- **Tiến trình mới:**

- Nhận tài nguyên từ HĐH hoặc từ tiến trình cha
- Tài nguyên: nếu tiến trình con được tạo từ tiến trình cha:
 - Tiến trình cha và con chia sẻ mọi tài nguyên
 - Tiến trình con chỉ chia sẻ một phần tài nguyên của cha
- **Trình tự thực thi:**
 - Tiến trình cha và con thực thi đồng thời (concurrently)
 - Tiến trình cha đợi đến khi các tiến trình con kết thúc
- **Không gian địa chỉ (address space):**
 - Không gian địa chỉ của tiến trình con được nhân bản từ cha
 - Không gian địa chỉ của tiến trình con được khởi tạo từ template

44

Các thao tác với tiến trình (tt)

```
#include <stdio.h>
#include <unistd.h>
int main (int argc, char *argv[]){
    int pid;
    pid = fork();
    if (pid > 0) {
        printf("This is parent process");
        wait (NULL);
        exit(0);
    }
    else if (pid == 0) {
        printf("This is child process");
        execp("/bin/ls", "ls", NULL);
        exit(0);
    }
    else {
        printf("Fork error\n");
        exit(-1);
    }
}
```



45

Các thao tác với tiến trình (tt)

• Kết thúc tiến trình

- Tiến trình kết thúc trong các trường hợp:
 - Tiến trình tự kết thúc bình thường: tiến trình kết thúc khi thực thi lệnh cuối và gọi **exit**
 - Tiến trình bị lỗi và tự kết thúc (do trình biên dịch xử lý)
 - Tiến trình bị lỗi và buộc phải kết thúc do phần cứng (ví dụ thực hiện một lệnh không hợp lệ, tham chiếu đến bộ nhớ không tồn tại, lỗi chia cho không,...) → phát sinh một ngắt (interrupted)
 - Tiến trình kết thúc do tiến trình khác (ví dụ tiến trình cha)
 - Gọi **abort** với tham số là pid của tiến trình cần được kết thúc
 - Khi tiến trình kết thúc, HĐH thu hồi tất cả các tài nguyên của tiến trình

46

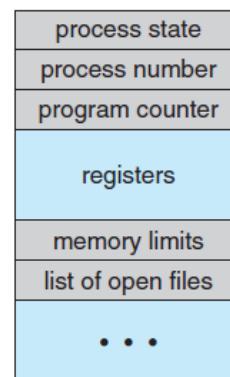
Thực thi các tiến trình

- Để thực thi được mô hình tiến trình, HĐH duy trì một bảng gọi là bảng tiến trình.
- Mỗi phần tử trong bảng là một khối quản lý tiến trình (Process Control Block - PCB)
- Mỗi PCB quản lý một tiến trình
- Mỗi tiến trình khi được tạo sẽ được cấp phát một (PCB)

47

PCB – Process Control Block

- Là một cấu trúc dữ liệu lưu các thông tin:
 - Định danh (Process Number)
 - Trạng thái tiến trình (Process State)
 - Bộ đếm chương trình (Program counter)
 - Các thanh ghi (CPU registers)
 - Thông tin lập thời biểu CPU: độ ưu tiên, con trỏ đến hàng đợi,...
 - Thông tin quản lý bộ nhớ
 - Thông tin tài khoản: lượng CPU, thời gian sử dụng,...
 - Thông tin trạng thái I/O

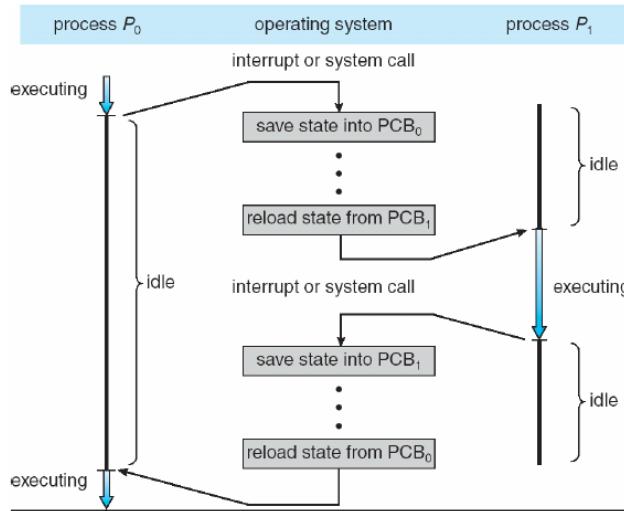


Process control block (PCB).

48

PCB (tt)

- Sơ đồ chuyển CPU giữa các tiến trình



49

2.2. Luồng (thread)

- Mô hình
- Hiện thực

50

2.2.1 Mô hình luồng

- Nhìn lại giải pháp đa tiến trình
 - Các tiến trình độc lập, không có sự liên lạc với nhau
 - Mỗi tiến trình có một không gian địa chỉ và một dòng xử lý duy nhất → tiến trình thực hiện chỉ có một tác vụ tại một thời điểm.
 - Muốn trao đổi thông tin với nhau, các chương trình cần được xây dựng theo mô hình liên lạc đa tiến trình (IPC – Inter-Process Communication) → Phức tạp, chi phí cao

51

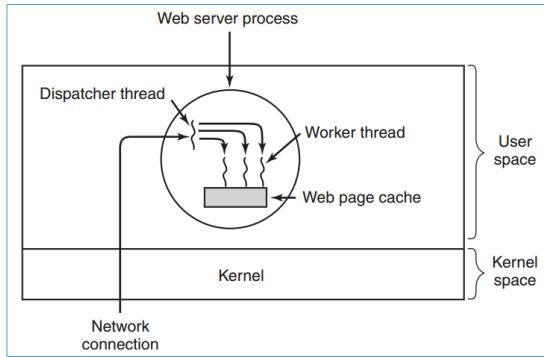
Mô hình luồng (tt)

- Để tăng tốc độ và sử dụng CPU hiệu quả hơn:
 - Cần nhiều dòng xử lý cùng chia sẻ một không gian địa chỉ
 - Các dòng xử lý hoạt động song song tương tự như tiến trình phân biệt
 - Mỗi dòng xử lý được gọi là một luồng (thread)
- Hầu hết các HĐH hiện đại đều được thực hiện theo mô hình luồng
- Ví dụ:
 - Các xử lý trên máy chủ web
 - Ứng dụng Word

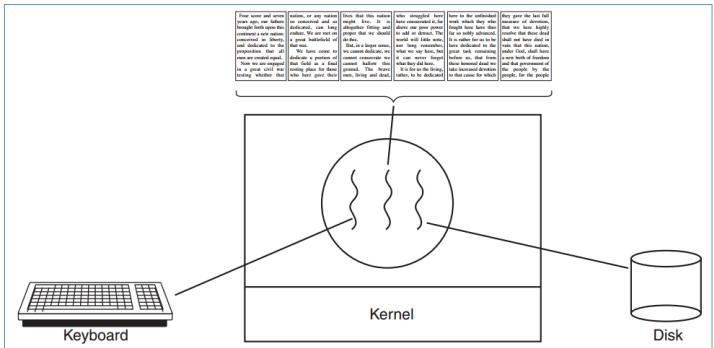
52

Mô hình luồng (tt)

- Web server xử lý đa luồng



- Ứng dụng Word với ba luồng xử lý

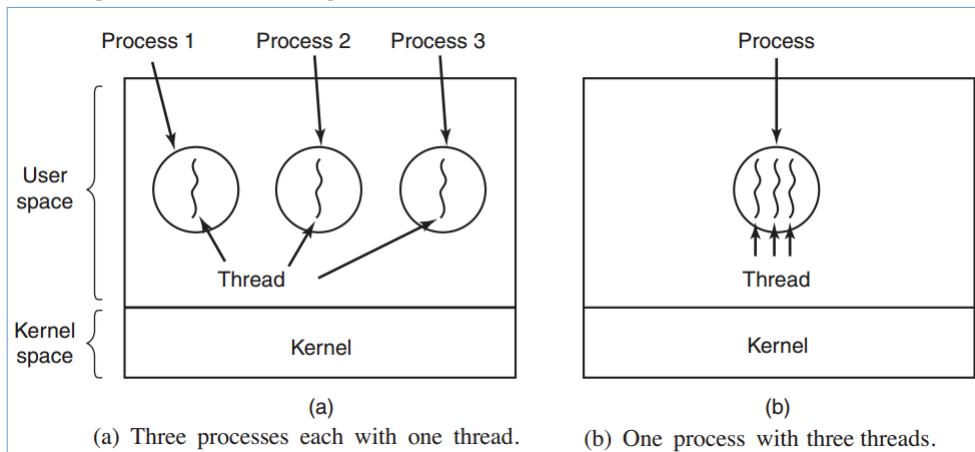


Andrew S. Tanenbaum, Modern Operating Systems, Pearson, 2015.

53

Mô hình luồng (tt)

- Đơn luồng và đa luồng

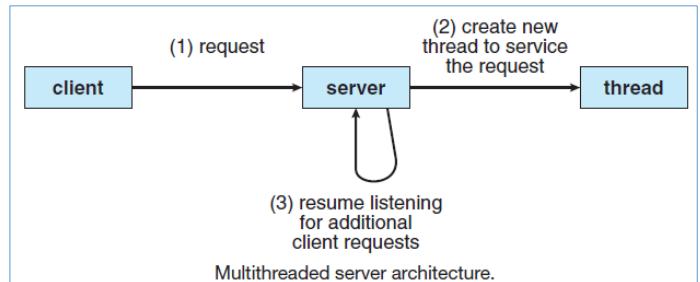


Andrew S. Tanenbaum, Modern Operating Systems, Pearson, 2015.

54

Mô hình luồng (tt)

- Luồng là một dòng xử lý trong một tiến trình
- Mỗi tiến trình luôn có một luồng chính (dòng xử lý cho hàm main())
- Ngoài luồng chính, tiến trình còn có thể có nhiều luồng con khác
- Các luồng của một tiến trình:
 - Chia sẻ không gian vùng code và data
 - Có vùng stack riêng



55

Mô hình luồng (tt)

- Các trạng thái của luồng:
 - Spawn: luồng được tạo.
 - Khi một tiến trình mới được tạo, một luồng chính cũng được tạo ra.
 - Một luồng trong tiến trình có thể tạo ra một luồng khác, cung cấp con trỏ lệnh, các đối số, thanh ghi ngũ cảnh, không gian stack và luồng mới tạo được đưa vào ready queue.
 - Blocked:
 - Khi một luồng phải chờ một sự kiện → blocked (lưu các thanh ghi người dùng, bộ đếm chương trình, con trỏ stack).
 - CPU có thể chuyển sang thực thi một luồng khác trong cùng tiến trình hoặc tiến trình khác.
 - Unblock: khi sự kiện mà luồng đang chờ đã sẵn sàng, luồng chuyển sang trạng thái ready (được đưa vào ready queue)
 - Finish: khi luồng kết thúc

56

Mô hình luồng (tt)

• Ưu điểm của luồng:

- **Tính đáp ứng (responsiveness):** chương trình tiếp tục chạy nếu một phần của nó bị blocked hay đang thực thi một thao tác dài → đáp ứng nhanh đến người sử dụng, đặc biệt trong các hệ thống chia sẻ thời gian thực.
- **Chia sẻ tài nguyên (resource sharing):** các luồng chia sẻ bộ nhớ và tài nguyên của tiến trình, và các luồng có liên quan (luồng tạo ra nó) → cho phép một ứng dụng có nhiều luồng khác nhau hoạt động trong cùng một không gian địa chỉ

57

Mô hình luồng (tt)

• Ưu điểm của luồng (tt)

• **Tính kinh tế (economy):**

- Cấp phát bộ nhớ và tài nguyên cho việc tạo tiến trình: tốn kém.
- Luồng chia sẻ tài nguyên của tiến trình → tạo luồng và chuyển ngữ cảnh ít tốn thời gian (thấp hơn 5 lần so với tiến trình)
- Thời gian tạo tiến trình gấp 30 lần so với tạo luồng

• **Khả năng mở rộng (Scalability):** tận dụng được kiến trúc đa xử lý (Utilization of multiprocessor architecture):

- Luồng có thể chạy song song trên các CPU khác nhau.
- Đa luồng trên máy nhiều CPU làm tăng tính đồng thời.

58

2.2.2 Hiện thực luồng

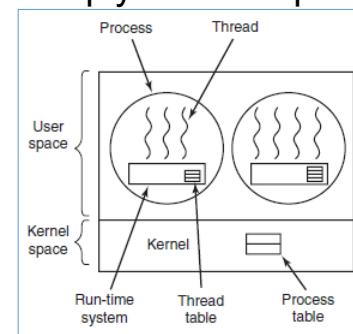
- Phân loại luồng
- Mô hình đa luồng
- Khối quản lý luồng
- Tạo luồng
- So sánh luồng và tiến trình

59

Phân loại luồng

• User-Level Thread (ULT)

- Việc quản lý luồng được thực hiện bởi ứng dụng
- Bất kỳ ứng dụng nào cũng có thể được lập trình đa luồng bằng cách sử dụng Thread library, là một thư viện các quy trình để quản lý ULT.
- Vai trò của Thread library:
 - Khởi tạo, định thời và quản lý luồng
 - Truyền thông giữa các luồng
 - Lưu giữ và khôi phục ngữ cảnh của luồng
 - Không cần hỗ trợ từ kernel



60

Phân loại luồng (tt)

• User-Level Thread (tt)

- Ưu điểm: tạo và quản lý nhanh.
- Nhược điểm:
 - Nếu kernel là single threaded, một luồng bị blocked → tất cả các luồng khác cũng bị blocked.
 - Không tận dụng kiến trúc nhiều CPU: hai luồng của một tác vụ không thể chạy trên hai CPU.
- Một số thư viện User threads:
 - POSIX pthreads
 - Mach C-threads.
 - Solaris UI-threads

61

Phân loại luồng (tt)

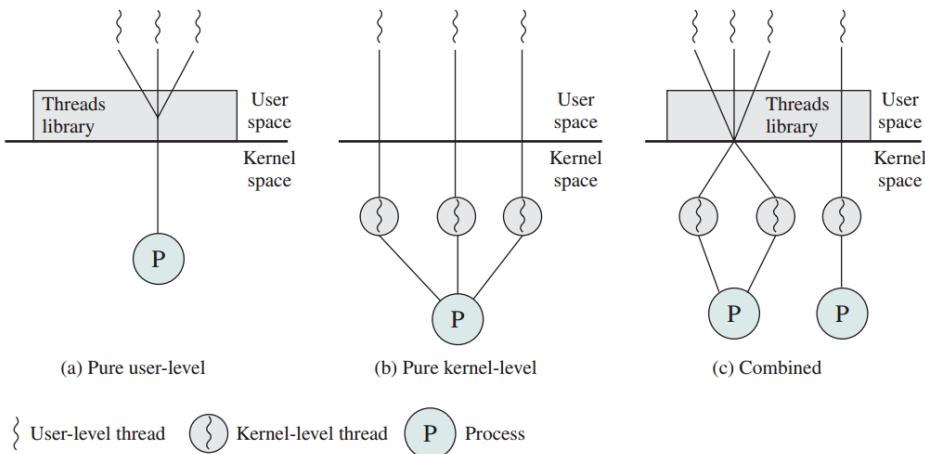
• Kernel-Level Thread:

- Kernel thread được hỗ trợ trực tiếp bởi nhân hệ điều hành.
- Kernel thực hiện việc tạo luồng, định thời và quản lý trong không gian kernel.
- Bởi vì việc quản lý thread được thực hiện bởi hệ điều hành, kernel thread được tạo và quản lý chậm hơn user thread.
- Kernel quản lý luồng → nếu một luồng bị blocked → có thể chuyển một luồng khác trong ứng dụng để thực thi → trong môi trường nhiều CPU, kernel có thể định thời cho luồng trên các CPU khác nhau.

62

Phân loại luồng (tt)

- User-Level and Kernel-Level Threads



Nguồn: William Stallings, Operating Systems: Internals and Design Principles, Pearson, 2015

63

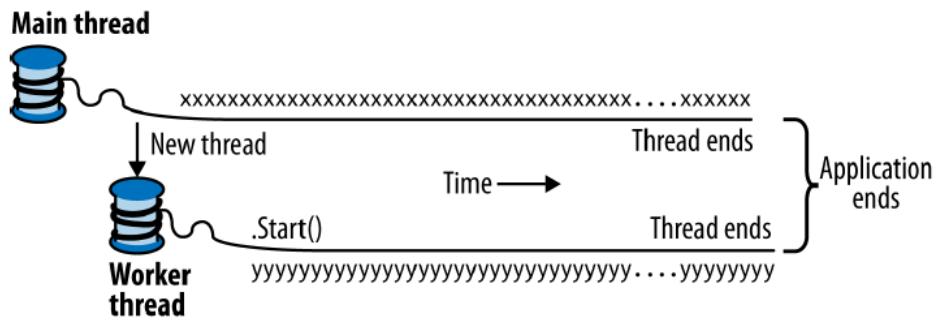
Khối quản lý luồng (Thread Control Block – TCB)

- TCB chứa các thông tin của mỗi luồng
 - ID của luồng
 - Không gian lưu các thanh ghi
 - Con trỏ tới vị trí xác định trong ngăn xếp
 - Trạng thái của luồng
 - Thông tin chia sẻ giữa các luồng trong một tiến trình
 - Các biến toàn cục
 - Các tài nguyên sử dụng như tập tin,...
 - Các tiến trình con
 - Thông tin thống kê
 - ...

64

Tạo luồng

- Khi tiến trình thực thi, một luồng chính được tạo
- Luồng chính có thể tạo các luồng con



65

Tạo luồng (tt)

```
using System;
using System.Threading;
class ThreadTest
{
    static void Main()
    {
        Thread t = new Thread (WriteY);
        t.Start(); // running WriteY()
        for (int i = 0; i < 1000; i++) Console.Write ("x");
    }
    static void WriteY()
    {
        for (int i = 0; i < 1000; i++) Console.Write ("y");
    }
}
```

// Typical Output:
 xxxxxxxxxxxxxxxxyyyyyyyyy
 xxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 yyyyyyyyyyyyyyyyyyyyyyyyyyyyy

66

So sánh luồng và tiến trình

- Tại sao không dùng nhiều tiến trình để thay thế cho việc dùng nhiều luồng ?
 - Các tác vụ điều hành luồng (tạo, kết thúc, điều phối, chuyển đổi,...) ít tốn chi phí thực hiện hơn so với tiến trình
 - Liên lạc giữa các luồng thông qua chia sẻ bộ nhớ, không cần sự can thiệp của kernel

67

So sánh luồng và tiến trình

Tiến trình	Luồng
Các tác vụ điều hành tiến trình tốn nhiều tài nguyên	Các tác vụ điều hành luồng ít tốn chi phí thực hiện hơn so với tiến trình
Tiến trình chuyển đổi cần tương tác với hệ điều hành.	Liên lạc giữa các luồng thông qua chia sẻ bộ nhớ, không cần sự can thiệp của kernel
Trong nhiều môi trường xử lý, mỗi tiến trình thực thi có nguồn tài nguyên bộ nhớ và tập tin riêng của mình	Tất cả các luồng có thể chia sẻ các tập tin mở, tiến trình con.
Nếu một tiến trình bị khóa (blocked), không có tiến trình nào khác có thể thực thi cho đến khi tiến trình ban đầu được unblocked	Khi một tiến trình bị khóa và chờ, luồng kế tiếp trong cùng một task có thể chạy
Nhiều tiến trình hoạt động độc lập với các tiến trình khác	Một luồng có thể đọc, ghi, thay đổi dữ liệu của luồng khác

68

2.3. Truyền thông giữa các tiến trình

• Mục tiêu:

- Chia sẻ thông tin.
- Hợp tác hoàn thành tác vụ:
 - Chia nhỏ tác vụ để có thể thực thi song song.
 - Dữ liệu ra của tiến trình này là dữ liệu đầu vào cho tiến trình khác
- HĐH cần cung cấp cơ chế để các tiến trình có thể trao đổi thông tin với nhau.

69

2.3.1 Các dạng tương tác giữa các tiến trình

- Tín hiệu (Signal)
- Pipe
- Vùng nhớ chia sẻ
- Trao đổi thông điệp (Message)
- Sockets

70

2.3.1.1 Tín hiệu (Signal)

- Sử dụng tín hiệu để thông báo cho tiến trình khi có một sự kiện xảy ra
- Mỗi tiến trình có một bảng biểu diễn các tín hiệu khác nhau.
- Mỗi tín hiệu (*signal handler*) có một trình xử lý tương ứng.
- Các tín hiệu được gởi đi bởi :
 - Phần cứng (ví dụ lỗi do các phép tính số học)
 - Kernel gởi đến một tiến trình (ví dụ: báo cho tiến trình khi có một thiết bị I/O rối).
 - 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 Ctrl-C để ngắt xử lý của tiến trình)

71

Tín hiệu (Signal)

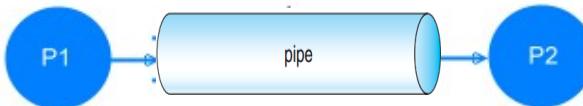
- Một số tín hiệu của UNIX

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

72

2.3.1.2 Pipe

- 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 tiến trình ghi dữ liệu vào pipe
 - Một tiến trình đọc dữ liệu từ pipe.
- Thứ tự dữ liệu truyền qua pipe: FIFO
- Một pipe có kích thước giới hạn (thường là 4096 ký tự)



73

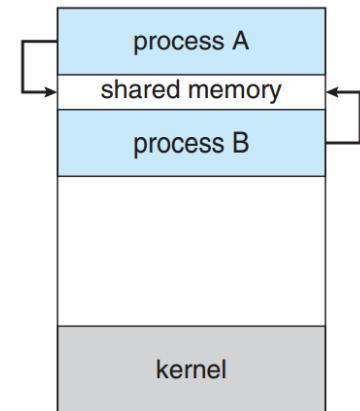
Pipe (tt)

- Một tiến trình có thể sử dụng một pipe do nó tạo ra hay kế thừa từ tiến trình cha.
- Vai trò của Hệ điều hành:
 - Cung cấp các lời gọi hệ thống (hàm) read/write cho các tiến trình thực hiện thao tác đọc/ghi dữ liệu trong pipe
 - Đồng bộ hóa việc truy xuất :
 - Nếu pipe trống: tiến trình đọc sẽ bị khóa
 - Nếu pipe đầy: tiến trình ghi sẽ bị khóa
- Ví dụ sử dụng pipe: lệnh trong command line
 - ls | more (Linux)
 - dir | more (Windows)

74

2.3.1.3 Vùng nhớ chia sẻ

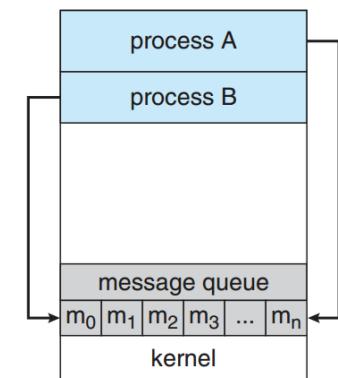
- Cho phép nhiều tiến trình cùng truy xuất đến một vùng nhớ chung gọi là *vùng nhớ chia sẻ (shared memory)*.
- Các tiến trình chia sẻ một vùng nhớ vật lý thông qua không gian địa chỉ của tiến trình.
- Vùng nhớ chia sẻ tồn tại độc lập với các tiến trình
- Một tiến trình muốn truy xuất đến vùng nhớ chia sẻ cần phải kết nối vùng nhớ đó vào không gian địa chỉ riêng của tiến trình để thao tác trên đó.



75

2.3.1.4 Trao đổi thông điệp (Message)

- Các tiến trình liên lạc với nhau thông qua việc gửi thông điệp.
- HĐH cung cấp các hàm IPC chuẩn (Interprocess communication):
 - Send ([destination], message): gửi một thông điệp
 - Receive ([source], message): nhận một thông điệp
- Khi hai tiến trình muốn liên lạc với nhau:
 - Thiết lập một mối liên kết giữa hai tiến trình
 - Sử dụng các hàm IPC thích hợp để trao đổi thông điệp
 - Hủy liên kết.



76

2.3.1.5 Sockets

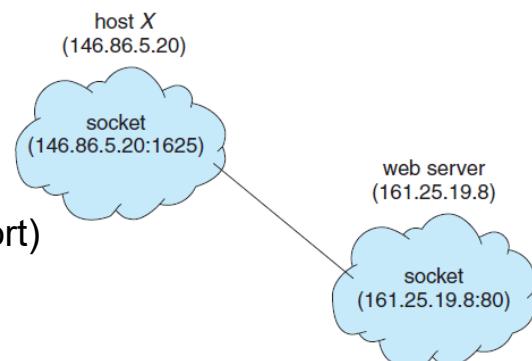
- Được dùng để trao đổi dữ liệu giữa các máy tính trên mạng
- Một socket là một thiết bị truyền thông hai chiều để gửi và nhận dữ liệu giữa các máy trên mạng.
- Mỗi socket là một đầu nối giữa một máy đến một máy tính khác
- Thao tác đọc/ghi là sự trao đổi dữ liệu ứng dụng trên nhiều máy khác nhau.
- 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)
 - Liên lạc điện thoại (socket đóng vai trò tổng đài) .

77

Sockets (tt)

- Truyền thông giữa hai tiến trình dùng socket

- Các bước trao đổi dữ liệu:
 - Tạo socket
 - Gắn kết socket với một địa chỉ (IP, Port)
 - Liên lạc:
 - Liên lạc trong chế độ không kết nối
 - Liên lạc trong chế độ có kết nối
 - Hủy socket



78

2.3.2 Vấn đề tranh chấp tài nguyên

- Trong một hệ thống có nhiều tiến trình cùng chạy
- Các tiến trình có thể chia sẻ tài nguyên chung (file system, CPU...)
- Nhiều tiến trình truy xuất đồng thời một tài nguyên mang bản chất không chia sẻ được → Xảy ra vấn đề tranh đoạt điều khiển (Race Condition)

79

Race Condition

- Ví dụ 1: đếm số người truy cập website, có 2 tiến trình chia sẻ đoạn code cập nhật biến đếm như sau:

```

counter = 0

    > P1
    n = counter;
    n = n +1;
    counter = n;

    > P2
    n = counter;
    n = n + 1;
    counter = n;

    counter?

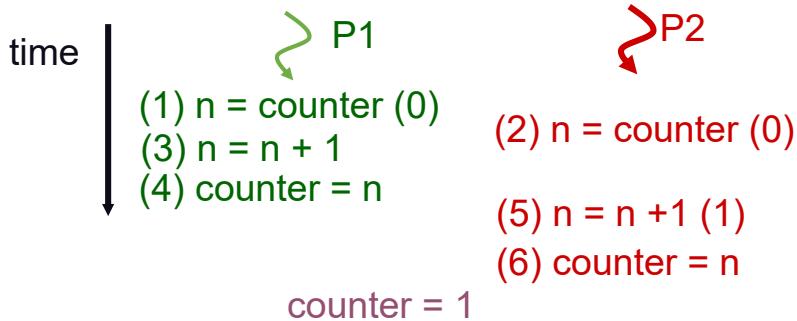
```

80

Race Condition (tt)

- Đếm số người truy cập website: tình huống 1

counter = 0

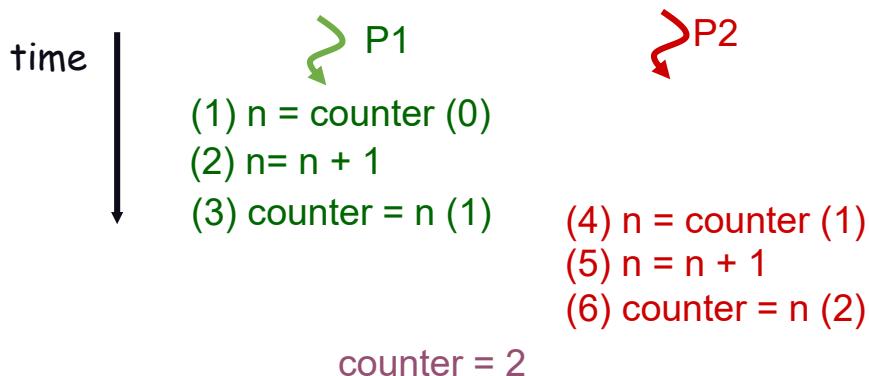


81

Race Condition (tt)

- Đếm số người truy cập website: tình huống 2

counter = 0



82

Race Condition (tt)

- Ví dụ 2: Bài toán rút tiền từ ngân hàng

```
if (tongtien >= tienrut)
    tong tien = tong tien – tien rut
```

- Nếu có hai tiến trình cùng thực hiện?

83

Race Condition (tt)

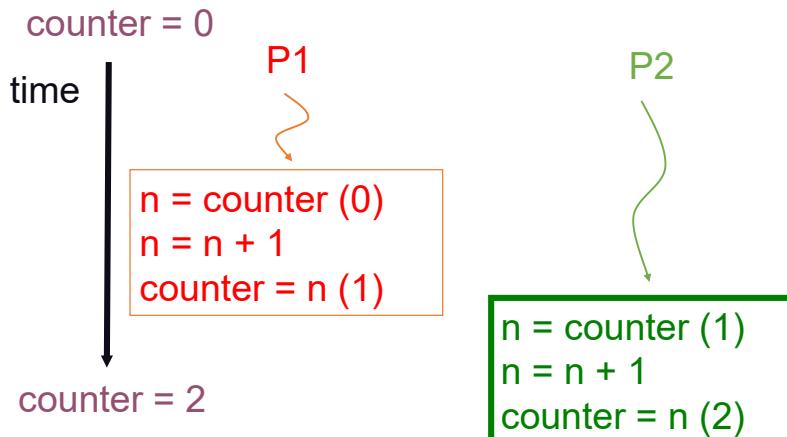
- Lý do xảy ra Race condition:

- Hai hoặc nhiều tiến trình cùng đọc/ghi dữ liệu trên một vùng nhớ chung
- Một tiến trình xen vào quá trình truy xuất tài nguyên của một tiến trình khác
- Giải pháp: đảm bảo cho một tiến trình hoàn tất việc truy xuất tài nguyên chung trước khi có tiến trình khác can thiệp.

84

Race Condition (tt)

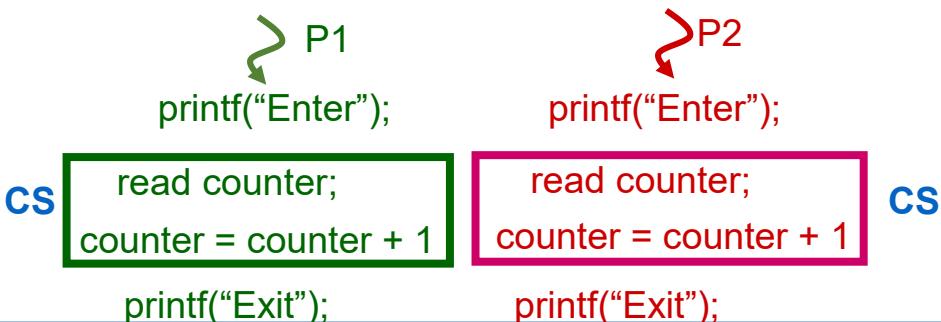
- Giải pháp cho ví dụ 1:



85

Miền găng (CriticalSection) & Khả năng độc quyền (Mutual Exclusion)

- Miền găng (CS) là đoạn chương trình có khả năng gây ra tình trạng race condition
- Để không xảy ra tình trạng Race condition → bảo đảm tính “độc quyền truy xuất” (Mutual Exclusion) cho miền găng (CS)



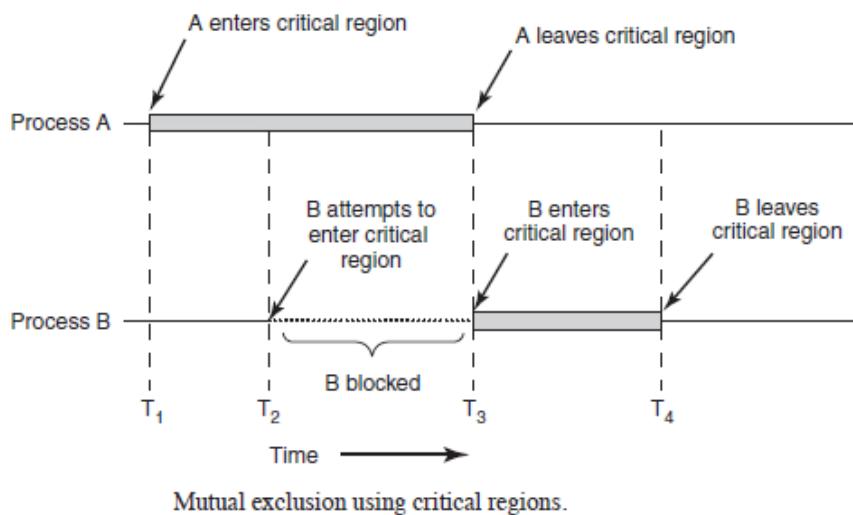
86

Giải pháp cho vấn đề tranh chấp tài nguyên

- **Đồng bộ hóa tiến trình:** bảo đảm tại một thời điểm chỉ có duy nhất một tiến trình được xử lý trong miền găng.
- **Bốn mục tiêu đồng bộ hóa tiến trình:**
 1. **Mutual exclusion:** Không có hai tiến trình cùng ở trong miền găng cùng 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 được vào miền găng.
 4. Không có giả định nào đặt ra về tốc độ của các tiến trình hoặc số lượng CPU.

87

Giải pháp cho vấn đề tranh chấp tài nguyên (tt)



88

Giải pháp cho vấn đề tranh chấp tài nguyên (tt)

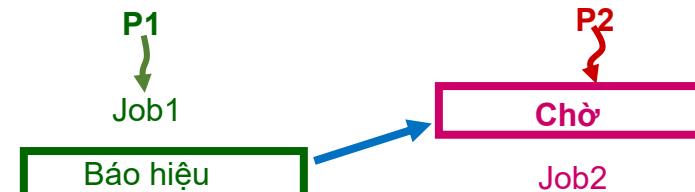
• Mô hình đảm bảo Mutual Exclusion:

- Thêm các đoạn code đồng bộ hóa vào chương trình gốc



• Mô hình tổ chức phối hợp giữa hai tiến trình:

- Thêm các đoạn code đồng bộ hóa vào 2 chương trình gốc
- Không áp dụng cho nhiều tiến trình



89

2.3.3 Đồng bộ hóa tiến trình

• Nhóm giải pháp Busy Waiting

• Phần mềm

- Sử dụng các biến cờ hiệu
- Sử dụng việc kiểm tra luân phiên
- Giải pháp của Peterson

• Phần cứng

- Cấm ngắt
- Chỉ thị TSL

• Nhóm giải pháp Sleep & Wakeup

- Mutex
- Semaphore
- Monitor
- Message

90

2.3.3.1 Các giải pháp “Busy waiting”

While (chưa có quyền) donothing()

CS;

Từ bỏ quyền sử dụng CS

- Nhận xét:

- Không đòi hỏi sự trợ giúp của Hệ điều hành
- Tiếp tục tiêu thụ CPU trong khi chờ đợi vào miền găng

91

Sử dụng biến cờ hiệu

- Sử dụng biến **lock** làm khóa chốt, khởi động là 0
 - lock = 0: CS rảnh
 - lock = 1: CS bận (có tiến trình đang truy cập)
- Tiến trình muốn vào miền găng phải kiểm tra giá trị của lock
 - lock = 0:
 - đặt lock = 1
 - vào miền găng
 - sau khi ra khỏi miền găng đặt lock = 0.
 - lock = 1: chờ

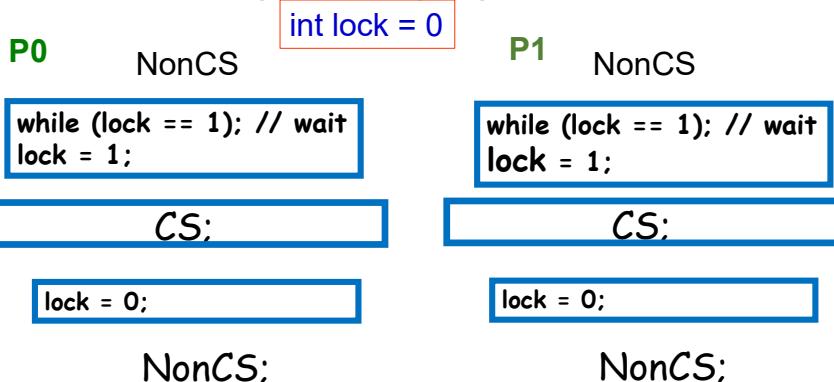
```
while (TRUE) {
    while (lock == 1); // wait
    lock = 1;
    critical-section ();
    lock = 0;
    Noncritical-section ();
}
```

92

Sử dụng biến cờ hiệu (tt)

• Nhận xét:

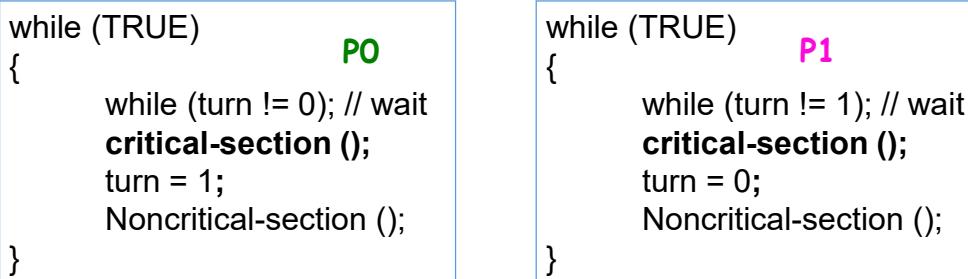
- Có thể áp dụng cho nhiều tiến trình
- Hai tiến trình có thể cùng ở miền găng tại một thời điểm



93

Kiểm tra luân phiên

- Hai tiến trình sử dụng chung biến turn, khởi động = 0 hoặc 1
 - turn = 0: P0 được vào miền găng, P1 chờ
 - turn = 1: P1 được vào miền găng, P0 chờ
 - Khi tiến trình P0 rời khỏi miền găng: đặt turn = 1
 - Khi tiến trình P1 rời khỏi miền găng: đặt turn = 0



94

Kiểm tra luân phiên (tt)

- Nhận xét:

- Chỉ áp dụng cho 2 tiến trình
- Bảo đảm Mutual Exclusion: ngăn chặn được tình trạng hai tiến trình cùng vào miền găng do kiểm tra biến turn
- Vi phạm Progress: một tiến trình ở ngoài miền găng có thể ngăn chặn tiến trình khác vào miền găng

95

Kiểm tra luân phiên (tt)

- Vi phạm Progress :

int turn = 1



P0 không vào được CS lần 2 khi P1 dừng trong NonCS !

96

Peterson

- Kết hợp ý tưởng của 1 & 2, các tiến trình chia sẻ các biến:
 - int turn //đến phiên tiến trình nào
 - int interested [2] // xác định tiến trình muốn vào CS
interested [i] = TRUE: tiến trình Pi muốn vào miền găng.
- Khởi đầu:
 - interested [0] = interested [1] = FALSE;
 - turn = 0 hay 1.
- Pi muốn vào được CS:
 - interested [j] = FALSE
 - turn = i
- Pi rời khỏi miền găng:
 - interested [i] = FALSE

97

Peterson (tt)

```

#define FALSE 0
#define TRUE 1
#define N 2 /*number of processes */
int turn; /* whose turn is it? */
int interested [N]; /* all values initially 0 (FALSE) */
void enter_region (int process) { /* process is 0 or 1 */
    int other = 1 - process; /* the opposite of process */
    interested [process] = TRUE; /* show that you are interested */
    turn = process; /* set flag */
    while (turn == process && interested[other] == TRUE);
}
void leave_region (int process) { /* process: who is leaving */
    interested [process] = FALSE;
}

```

98

Peterson (tt)

- Nhận xét: đáp ứng được cả 3 điều kiện
 - Mutual Exclusion :
 - P_i chỉ có thể vào CS khi: $interested[j] == False$, $turn == i$
 - Do biến $turn$ chỉ có thể nhận giá trị 0 hay 1 nên chỉ có 1 tiến trình vào CS
- Progress
 - Sử dụng 2 biến $interested[i]$ riêng biệt \Rightarrow tiến trình đổi phương không khoá lẫn nhau được.
- Bounded Wait : $interested[i]$ và $turn$ đều có thay đổi giá trị
- Không thể mở rộng cho N tiến trình

99

Cấm ngắt

- Cho phép tiến trình cấm tất cả các ngắt (kể cả ngắt đồng hồ) trước khi vào CS, và phục hồi ngắt khi ra khỏi CS
- \rightarrow hệ thống không thể tạm dừng hoạt động của tiến trình đang xử lý để cấp phát CPU cho tiến trình khác.
- Nhận xét:
 - Thiếu thận trọng: cho phép tiến trình người dùng được phép thực hiện lệnh cấm ngắt.
 - 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 vào CS \rightarrow không đảm bảo Mutual Exclusion.

NonCS;

Disable Interrupt;

CS;

Enable Interrupt;

NonCS;

100

Chỉ thị TSL (Test-and-Set)

- Giải pháp này yêu cầu sự trợ giúp của cơ chế phân cứng.
- Hệ thống cung cấp một lệnh đặc biệt cho phép kiểm tra và cập nhật nội dung một vùng nhớ chung.
- Hai chỉ thị TSL xử lý đồng thời (trên hai bộ xử lý khác nhau) sẽ được xử lý tuần tự.
- Đảm bảo truy xuất độc quyền:
 - Sử dụng thêm một biến **lock**
 - khởi gán lock = FALSE
 - Tiến trình muốn vào CS phải kiểm tra nếu lock = FALSE → vào CS

101

Chỉ thị TSL (tt)

```
boolean test_and_set (boolean *target) {
    boolean rv = *target;
    *target = true;
    return rv;
}
```

```
do {
    while (test and set (&lock));
    critical-section ();
    lock = false;
    Noncritical-section ();
} while (true);
```

102

Chỉ thị TSL (tt)

• Nhận xét về TSL:

- Cần được sự hỗ trợ của cơ chế phần cứng
- Không dễ cài đặt, nhất là trên các máy có nhiều bộ xử lý
- Dễ mở rộng cho N tiến trình

• Nhận xét chung về các giải pháp Busy waiting

- Sử dụng CPU không hiệu quả do liên tục kiểm tra điều kiện khi chờ vào CS.
- **Khắc phục:** khoá các tiến trình chưa đủ điều kiện vào CS, nhường CPU cho tiến trình khác → giải pháp **Sleep & Wake up**

103

Các giải pháp “Sleep & Wake up”

- Các tiến trình phải từ bỏ CPU khi chưa được vào CS
- Khi CS trống, sẽ được đánh thức để vào CS
- Cần phải sử dụng các thủ tục do hệ điều hành cung cấp để thay đổi trạng thái tiến trình

```
if (chưa có quyền vào
    CS) Sleep();
```



```
CS
```

```
Wakeup (other
    process);
```

104

Sleep & Wake up (tt)

- Hệ Điều hành hỗ trợ 2 hàm đặc quyền:
 - Sleep(): Tiến trình tự gọi hàm này để chuyển sang trạng thái Blocked
 - WakeUp (P): Tiến trình P nhận trạng thái Ready
- Khi một tiến trình chưa đủ điều kiện vào CS → gọi Sleep () → chuyển sang trạng thái bị khóa cho đến khi có một tiến trình khác gọi WakeUp để giải phóng cho nó.
- Một tiến trình khi ra khỏi CS sẽ gọi WakeUp (P) để đánh thức một tiến trình khác (P) đang bị khóa, để tiến trình này vào CS

105

Sleep & Wake up (tt)

```

int busy; //busy = 0: CS trống
int blocked; //số tiến trình bị Blocked chờ vào CS
while (TRUE) {
    if (busy) {
        blocked = blocked + 1;
        sleep();
    }
    else //busy = 0
        busy = 1;
    critical-section ();
    busy = 0;
    if(blocked) {
        wakeup(process);
        blocked = blocked - 1;
    }
    Noncritical-section ();
}

```

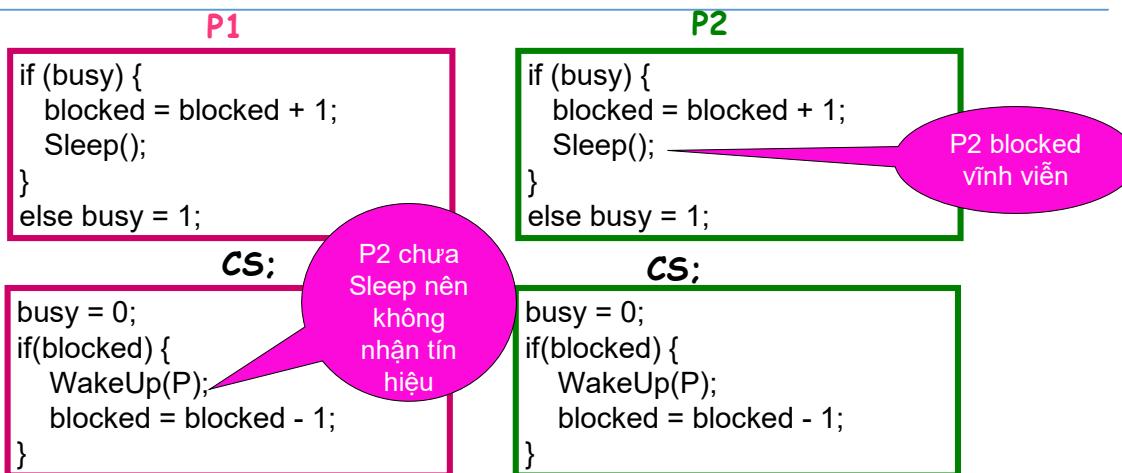
106

Sleep & Wake up (tt)

- Có thể xảy ra mâu thuẫn truy xuất
 - Giả sử P1 vào CS, khi chưa xử lý xong P1 hết thời gian sử dụng CPU, P2 được kích hoạt.
 - P2 muốn vào CS nhưng kiểm tra thấy P1 đang ở trong CS → P2 tăng giá trị biến *blocked* và **sẽ** gọi *Sleep* để tự khóa.
 - **Trước khi P2 gọi Sleep**, P1 lại được tái kích hoạt, xử lý xong và ra khỏi CS.
 - P1 thấy có một tiến trình đang chờ (*blocked=1*) nên gọi *WakeUp* và giảm giá trị của *blocked* (*blocked=0*)
 - Do P2 chưa thực hiện *Sleep* nên không thể nhận tín hiệu *WakeUp*.
 - Khi P2 được tiếp tục xử lý, nó mới gọi *Sleep* và tự khóa vĩnh viễn

107

Vấn đề với Sleep & WakeUp



108

Mutex locks

- Là giải pháp đơn giản được sử dụng để ngăn chặn race conditions
- Mutex là một biến có hai giá trị: khóa/không khóa
- Hai hàm đặc quyền được hỗ trợ từ phần cứng:
 - mutex-lock: tiến trình muốn vào CS
 - mutex-unlock: tiến trình ra khỏi CS, mở khóa cho các tiến trình khác có thể vào CS
- Một tiến trình muốn vào CS, gọi mutex-lock và trước khi rời khỏi CS gọi mutex-unlock

109

Mutex locks (tt)

```

mutex_lock() {
    while (!available); /* busy wait */
    available = false;
}

mutex_unlock() {
    available = true;
}
  
```

```

do {
    mutex_lock ();
    Critical_section
    mutex_unlock ();
    Noncritical_section
} while (true);
  
```

110

Semaphore

- Được Dijkstra đề xuất vào 1965
- Một semaphore s là cấu trúc dữ liệu gồm có:
 - Một biến nguyên dương `count`
 - Hàng đợi `queue`: danh sách các tiến trình đang bị khóa trên semaphore s
 - Hai phương thức:
 - `Wait(semaphore s)`:
 - nếu `s.count > 0`: `s.count = s.count - 1`
 - Ngược lại (`s.count = 0`): tiến trình phải chờ đến khi `s.count > 0`.
 - `Signal(semaphore s)`:
 - `s.count = s.count + 1`
 - Nếu có một hoặc nhiều tiến trình đang chờ trên semaphore s (`count (queue) > 0`) → hệ thống sẽ chọn một trong các tiến trình trong hàng đợi này cho tiếp tục xử lý.

111

Cài đặt Semaphore (Sleep & Wakeup)

```
struct semaphore {
    int count;
    queueType queue;
};
```

```
void Wait(semaphore s) {
    s.count--;
    if (s.count < 0) {
        /* place this process in s.queue */;
        /* block this process */;
    }
}
```

```
void Signal(semaphore s) {
    s.count++;
    if (s.count <= 0) {
        /* remove a process P from s.queue */;
        /* place process P on ready list */;
    }
}
```

112

Sử dụng Semaphore

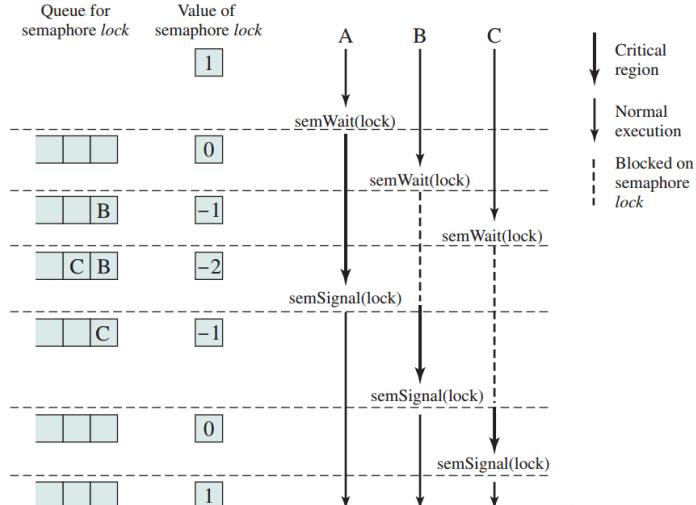
- Truy xuất độc quyền với Semaphore
 - Cho phép bảo đảm nhiều tiến trình cùng truy xuất đến CS mà không có sự mâu thuẫn truy xuất.
 - n tiến trình cùng cấu trúc chương trình sau:

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

113

Sử dụng Semaphore

- Mutual Exclusion



William Stallings, Operating Systems: Internals and Design Principles, Pearson, 2015

114

Sử dụng Semaphore

- Tổ chức đồng bộ hóa với Semaphore
 - 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 count = 0.

```
P1:
while (TRUE)
{
    job1();
    Signal (s); // đánh thức P2
}
```

```
P2:
while (TRUE)
{
    Wait (s); // chờ P1
    job2();
}
```

115

Nhận xétSemaphore

- Semaphore được xem như là một resource
 - Các tiến trình “yêu cầu” semaphore s: gọi Wait (s)
 - Nếu không hoàn tất được Wait (s): chưa được cấp resource → Blocked, được đưa vào s.queue
- Cần có sự hỗ trợ của HĐH: Sleep() & Wakeup()
- Là một cơ chế tốt để thực hiện đồng bộ
- Mở rộng cho N tiến trình
- Khó sử dụng đúng: nếu thiếu hoặc đặt sai vị trí down và up

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

116

Ví dụ về Semaphores

The Producer-Consumer Problem



Buffer



Producer



Consumer

- It has to stop when buffer is full.
- If producer is producing at that time won't allow consumer to consume data.
- It has to stop when buffer is empty.
- If consumer is consuming that time won't allow producer to produce.

117

Semaphores - The Producer-Consumer Problem (tt)

- Sử dụng giải pháp sleep -wakeup

```
#define N 100
int count = 0;
/* number of slots in the buffer */
/* number of items in the buffer */

void producer(void)
{
    int item;

    while (TRUE) {
        item = produce_item();
        /* repeat forever */
        if (count == N) sleep();
        /* generate next item */
        /* if buffer is full, go to sleep */
        insert_item(item);
        /* put item in buffer */
        count = count + 1;
        /* increment count of items in buffer */
        if (count == 1) wakeup(consumer); /* was buffer empty? */
    }
}
```

118

Semaphores - The Producer-Consumer Problem (tt)

```

void consumer(void)
{
    int item;

    while (TRUE) {
        if (count == 0) sleep();           /* repeat forever */
        item = remove_item();            /* if buffer is empty, got to sleep */
        count = count - 1;               /* take item out of buffer */
        if (count == N - 1) wakeup(producer); /* decrement count of items in buffer */
        consume_item(item);             /* was buffer full? */
        /* print item */
    }
}

```

119

Semaphores - The Producer-Consumer Problem (tt)

• Vấn đề:

- Ban đầu count= 0:
 - C(consumer) thấy count = 0 → chuẩn bị sleep, nhưng bị block.
 - P(producer): count = count + 1, gọi wakeup để đánh thức C.
 - Tuy nhiên, có thể C vẫn chưa sleep một cách hợp lý, vì vậy tín hiệu wakeup bị mất. Khi C tiếp tục chạy → kiểm tra giá trị biến count đã đọc trước đó: count= 0 → sleep.
 - Khi P lặp dây bộ đếm (count=N) → sleep.
 - Cả P và C đều sleep.

120

Semaphores - The Producer-Consumer Problem (tt)

- Giải pháp dùng Semaphore:

```
#define N 100           /* number of slots in the buffer */
typedef int semaphore;
semaphore mutex = 1;    /* semaphores are a special kind of int */
semaphore empty = N;   /* controls access to critical region */
semaphore full = 0;    /* counts empty buffer slots */
/* counts full buffer slots */

void producer(void)
{
    int item;

    while (TRUE) {           /* TRUE is the constant 1 */
        item = produce_item(); /* generate something to put in buffer */
        down(&empty);        /* decrement empty count */
        down(&mutex);        /* enter critical region */
        insert_item(item);   /* put new item in buffer */
        up(&mutex);          /* leave critical region */
        up(&full);           /* increment count of full slots */
    }
}
```

121

Semaphores - The Producer-Consumer Problem (tt)

- Giải pháp dùng Semaphore (tt):

```
void consumer(void)
{
    int item;

    while (TRUE) {           /* infinite loop */
        down(&full);        /* decrement full count */
        down(&mutex);        /* enter critical region */
        item = remove_item(); /* take item from buffer */
        up(&mutex);          /* leave critical region */
        up(&empty);           /* increment count of empty slots */
        consume_item(item);  /* do something with the item */
    }
}
```

122

Monitor

- Đầu xuất bởi Hoare (1974) & Brinch (1975)
- Là giải pháp đồng bộ hóa do NNLT cung cấp
 - Hỗ trợ cùng các chức năng như Semaphore
 - Dễ sử dụng và kiểm soát hơn Semaphore
 - Bảo đảm Mutual Exclusion một cách tự động
 - Sử dụng biến điều kiện để thực hiện đồng bộ hóa
- Là một module chương trình định nghĩa:
 - Các CTDL, đối tượng dùng chung
 - Các phương thức xử lý các đối tượng này
 - Bảo đảm tính đóng gói

123

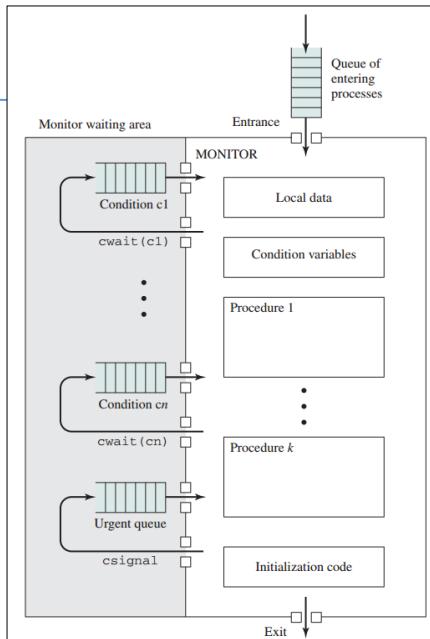
Monitor

- Các đặc điểm chính của Monitor:
 - Các biến dữ liệu cục bộ chỉ có thể truy cập được bằng thủ tục bên trong Monitor.
 - Một tiến trình hoạt động bên trong monitor bằng cách gọi các phương thức trong monitor.
 - Tại một thời điểm, chỉ có một tiến trình duy nhất được hoạt động bên trong một monitor
 - Các tiến trình không thể vào monitor → đưa vào hàng đợi

124

Monitor

- Cấu trúc Monitor:



125

Monitor (tt)

- Đồng bộ hóa với các biến điều kiện
 - x.Wait(): Tiến trình gọi hàm sẽ bị blocked → hàng đợi trên biến x
 - x.Signal(): Giải phóng 1 tiến trình đang bị blocked trên biến điều kiện x
 - x.queue: danh sách các tiến trình blocked trên biến điều kiện x
- Tiến trình sau khi gọi Signal:
 - Blocked, nhường quyền vào monitor cho tiến trình được đánh thức
 - Tiếp tục xử lý hết chu kỳ, rồi blocked

126

Monitor (tt)

- Cài đặt monitor

```
monitor monitor_name {
    /* shared variable declarations */
    function Procedure1 (...) {
        ...
    }
    function Procedure2 (...) {
        ...
    }
    ...
    function Proceduren (...) {
        ...
    }
    initialization code (...) {
        ...
    }
}
```

127

Monitor (tt)

- Cài đặt:

```
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
        status(Q) = ready;
        enter(Q,ready-list);
    }
}
```

128

Monitor (tt)

- Sử dụng: với mỗi nhóm tài nguyên cần chia sẻ, có thể định nghĩa một monitor, tất cả các thao tác trên tài nguyên này phải thỏa một số điều kiện nào đó

Pi:

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

129

Monitor (tt)

- Nhận xét:

- Đảm bảo truy xuất độc quyền bởi trình biên dịch mà không do lập trình viên → nguy cơ thực hiện đồng bộ hóa sai giảm rất nhiều.
- Đòi hỏi phải có một ngôn ngữ lập trình định nghĩa khái niệm monitor.

130

Message

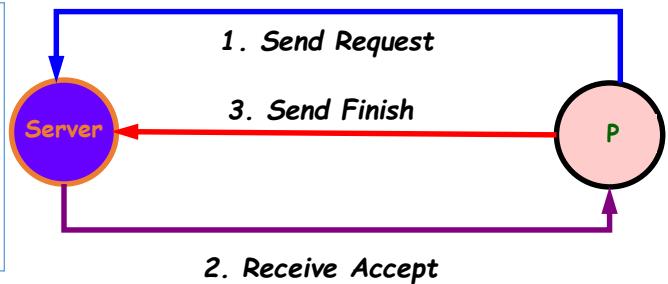
- Đồng bộ hóa trên môi trường phân tán
- Được hỗ trợ bởi HĐH
- Một tiến trình kiểm soát việc sử dụng tài nguyên và nhiều tiến trình khác yêu cầu tài nguyên.
- Tiến trình có yêu cầu tài nguyên:
 - Gởi một message đến tiến trình kiểm soát
 - → chuyển sang trạng thái blocked cho đến khi nhận được thông điệp đánh thức từ tiến trình kiểm soát
 - Khi sử dụng xong tài nguyên → gởi một thông điệp khác đến tiến trình kiểm soát để báo kết thúc truy xuất.
- Tiến trình kiểm soát:
 - Nhận được message yêu cầu tài nguyên
 - Khi tài nguyên sẵn sàng → gởi một thông điệp đến tiến trình đang bị khóa trên tài nguyên đó để đánh thức tiến trình này.

131

Message (tt)

- Trao đổi thông điệp với hai primitive Send và Receive để thực hiện sự đồng bộ hóa:

```
while (TRUE)
{
  Send(process controller, request message);
  Receive(process controller, accept message);
  critical-section ();
  Send(process controller, end message);
  Noncritical-section ();
}
```



132

Các bài toán đồng bộ hóa kinh điển

- Producer – Consumer
- Readers – Writers
- Dining Philosophers

133

2.4. Điều phối tiến trình (Scheduling)

- Mục tiêu điều phối
- Các cấp điều phối
- Các giải thuật điều phối (cấp điều phối thời gian ngắn)
- Vấn đề điều phối luồng

134

Mục tiêu điều phối (tt)

• Một số khái niệm:

- CPU utilization (% sử dụng CPU, Độ lợi CPU)
- Throughput: thông lượng tối đa
- Turnaround-time (Thời gian quay vòng – hoàn thành)
- Response time (Thời gian đáp ứng)
- Waiting time (Thời gian chờ)
- Average turn-around time (Thời gian quay vòng trung bình)

135

Mục tiêu điều phối (tt)

• Mục tiêu chung

- Công bằng sử dụng CPU, tận dụng CPU tối đa
- Cân bằng sử dụng các thành phần của hệ thống

• Hệ điều hành xử lý theo lô

- Tối ưu thông lượng tối đa (throughput)
- Giảm thiểu turnaround time: $T_{quit} - T_{arrive}$
- Tận dụng CPU

• Hệ điều hành tương tác

- Đáp ứng nhanh: giảm thiểu thời gian chờ
- Cân đối mong muốn của người dùng

• Hệ điều hành thời gian thực

- Tránh mất dữ liệu
- Đảm bảo chất lượng trong các hệ thống đa phương tiện

136

Mục tiêu điều phối (tt)

• Tiêu chí lựa chọn tiến trình

- Chọn tiến trình vào RQ trước
- Chọn tiến trình có độ ưu tiên cao hơn

• Thời điểm lựa chọn tiến trình

- Điều phối độc quyền (non-preemptive scheduling): tiến trình đang ở trạng thái Running sẽ tiếp tục sử dụng CPU cho đến khi kết thúc hoặc bị block vì chờ I/O hay các dịch vụ của hệ thống (độc chiếm CPU)
- Điều phối không độc quyền (preemptive scheduling): tiến trình đang running phải trả CPU khi:
 - Xử lý xong (kết thúc)
 - Bị block chờ I/O hay các dịch vụ của hệ thống
 - Hết thời gian sử dụng CPU
 - Có tiến trình có độ ưu tiên hơn vào ReadyQueue

137

2.4.2 Các cấp điều phối

• Điều phối tác vụ (job scheduling)

- Chọn tác vụ và nạp những tiến trình của tác vụ đó vào bộ nhớ chính để thực hiện
- Quyết định mức độ đa chương của hệ thống
- Chức năng điều phối tác vụ được kích hoạt khi tiến trình được tạo hoặc kết thúc
- Bộ điều phối tác vụ cần chọn luân phiên một cách hợp lý giữa tiến trình hướng nhập xuất (I/O bounded) và các tiến trình hướng xử lý (CPU bounded) → cân bằng hoạt động của CPU và các thiết bị ngoại vi

138

Các cấp điều phối (tt)

•Điều phối tiến trình (process scheduling)

- Chọn một tiến trình ở trạng thái sẵn sàng để cấp phát CPU
- Tần suất hoạt động cao (~100ms), xảy ra khi có ngắt (đồng hồ, thiết bị ngoại vi,...) → cần tăng tốc độ của bộ điều phối tiến trình
- Một số HĐH không có bộ điều phối tác vụ
- Một số HĐH kết hợp cả hai cấp độ điều phối (cấp độ điều phối trung gian)

139

2.4.3 Các giải thuật điều phối

•Điều phối trong Hệ điều hành xử lý theo lô (Batch Systems)

- First-Come, First-Served (FCFS)
- Shortest Job First (SJF)
- Shortest Remaining Time Next (SRTF)

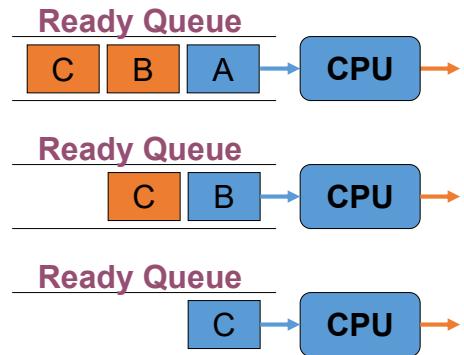
•Điều phối trong Hệ điều hành tương tác (Interactive Systems)

- Round Robin
- Priority
- HRRN
- Multiple Queues

140

2.4.3.1 First-Come First-Served (FCFS)

- Tiên trình vào hàng đợi (ready queue) trước sẽ nhận CPU trước.
- Thời điểm lựa chọn tiên trình
 - Độc quyền



141

FCFS (tt)

- Ví dụ:

P	Arrival Time	Service Time
P1	0	24
P2	1	3
P3	2	3

- Biểu đồ Gantt:



Thời gian đáp ứng:

$$P1: 0 - 0 = 0$$

$$P2: 24 - 1 = 23$$

$$P3: 27 - 2 = 25$$

$$Avg = (23 + 25) / 3 = 16$$

Thời gian hoàn thành:

$$P1: 24 - 0 = 24$$

$$P2: 27 - 1 = 26$$

$$P3: 30 - 2 = 28$$

$$Avg = (24 + 26 + 28) / 3 = 26.3$$

Thời gian chờ:

$$P1: 24 - 0 - 24 = 0$$

$$P2: 27 - 1 - 3 = 23$$

$$P3: 30 - 2 - 3 = 25$$

$$Avg = (23 + 25) / 3 = 16$$

142

FCFS (tt)

- Đơn giản, dễ cài đặt
- Chịu đựng hiện tượng tích lũy thời gian chờ
 - Tiến trình có thời gian xử lý ngắn đợi tiến trình có thời gian xử lý dài
 - Ưu tiên tiến trình cpu-bounded
- Có thể xảy ra tình trạng độc chiếm CPU
- Không phù hợp với các hệ phân chia thời gian

143

2.4.3.2 Shortest Job First (SJF) Shortest Remaining Time Next (SRT)

- Shortest Job First
 - Chọn tiến trình có thời gian xử lý ngắn nhất
 - Thời điểm lựa chọn tiến trình: độc quyền (non-preemptive)
- Shortest Remaining Time Next
 - Chọn tiến trình có thời gian xử lý còn lại ngắn nhất
 - Thời điểm lựa chọn tiến trình: không độc quyền (khi có một tiến trình mới vào RQ, thời gian xử lý còn lại của các tiến trình được tính toán lại và tiến trình có thời gian xử lý còn lại ngắn nhất sẽ được nhận CPU)

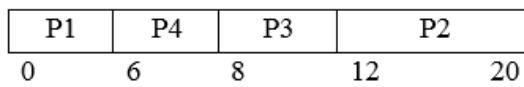
144

SJF – SRT (tt)

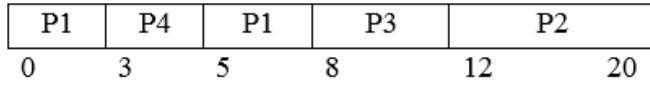
- Ví dụ

P	Arrival Time	Service Time
P1	0	6
P2	1	8
P3	2	4
P4	3	2

- Thời điểm lựa chọn độc quyền



- Thời điểm lựa chọn không độc quyền



145

SJF – SRT (tt)

- Nhận xét:

- Tối ưu thời gian chờ
- Cơ chế không độc quyền có thể xảy ra tình trạng “đói” (starvation) đối với các process có CPU-burst lớn khi có nhiều process với CPU-burst nhỏ đến hệ thống
- Cơ chế độc quyền không phù hợp cho hệ thống chia sẻ thời gian
- Thời gian sử dụng CPU còn lại cho lần tiếp theo của mỗi tiến trình được tính theo công thức:

$$\tau_{n+1} = \alpha t_n + (1 - \alpha) \tau_n$$

- t_n là độ dài của thời gian xử lý lần thứ n
- t_{n+1} là giá trị dự đoán cho lần xử lý tiếp theo
- $0 \leq \alpha \leq 1$

146

2.4.3.3 Điều phối với độ ưu tiên (Priority)

- Mỗi tiến trình được gán một độ ưu tiên để phân biệt tiến trình quan trọng với tiến trình bình thường
- Tiêu chí lựa chọn tiến trình
 - Tiến trình có độ ưu tiên cao nhất
- Thời điểm lựa chọn tiến trình
 - Độc quyền
 - Không độc quyền

147

Điều phối với độ ưu tiên (tt)

- Ví dụ:

P	Arrival Time	Priority	Service Time
P1	0	3	24
P2	1	1	3
P3	2	2	3

- Độ ưu tiên độc quyền

P1		P2	P3
0		24	27

- Độ ưu tiên không độc quyền

P1	P2	P3	P1
0	1	4	7

148

Điều phối với độ ưu tiên (tt)

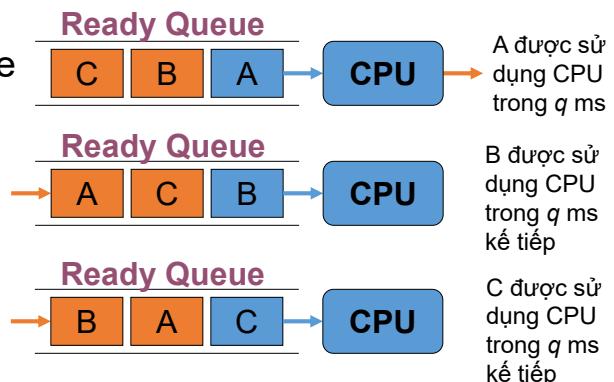
• Nhận xét:

- Có thể xảy ra tình trạng Starvation (đói CPU): các tiến trình độ ưu tiên thấp có thể không bao giờ thực thi được
- → Giải pháp Aging – tăng độ ưu tiên cho tiến trình chờ lâu trong hệ thống
- Gán độ ưu tiên còn dựa vào:
 - Yêu cầu về bộ nhớ
 - Số lượng file được mở
 - Tỉ lệ thời gian dùng cho I/O trên thời gian sử dụng CPU
 - Các yêu cầu bên ngoài

149

2.4.3.4 Round Robin (RR) Điều phối xoay vòng

- Mỗi tiến trình chỉ sử dụng một lượng q (quantum) cho mỗi lần sử dụng CPU
- Tiêu chí lựa chọn tiến trình
 - Thứ tự vào hàng đợi Ready Queue
- Thời điểm lựa chọn tiến trình
 - Không độc quyền



150

Round Robin (tt)

- Ví dụ:
 - RR (q=4)

P	Arrival Time	Service Time
P1	0	24
P2	1	3
P3	2	3

- Biểu đồ Gantt:

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

151

Round Robin (tt)

- Nhận xét:
 - Loại bỏ hiện tượng độc chiếm CPU
 - Phù hợp với hệ thống tương tác người dùng
 - Thời gian chờ đợi trung bình của giải thuật RR thường khá lớn nhưng thời gian đáp ứng nhỏ
 - Hiệu quả phụ thuộc vào việc lựa chọn quantum q
 - q quá lớn => FCFS (giảm tính tương tác)
 - q quá nhỏ => chủ yếu thực hiện chuyển đổi ngữ cảnh (context switching)
 - Thường q = 10 -100 milliseconds

152

2.4.3.5 Highest Response Ratio Next (HRRN)

- Cải tiến giải thuật SJF
- Định thời theo chế độ độc quyền
- Độ ưu tiên được tính theo công thức:

$$p = (tw + ts)/ts$$

- tw: thời gian chờ (waiting time)
- ts: thời gian sử dụng CPU (service time)
- Tiêu chí chọn: chọn p lớn nhất \rightarrow Ưu tiên cho các tiến trình có thời gian sử dụng CPU ngắn và các tiến trình chờ lâu.
- p được tính lại khi có tiến trình kết thúc

153

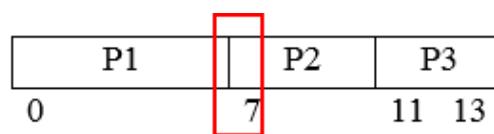
Highest Response Ratio Next (tt)

- Ví dụ:

P	Thời gian đến	Thời gian sử dụng CPU
P1	0	7
P2	1	4
P3	5	2

- Độ ưu tiên tại thời điểm (7):

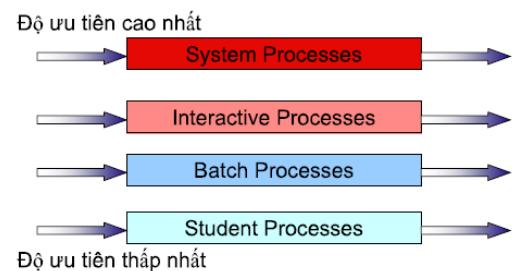
- P2: $(6+4)/4 = 2.5$
- P3: $(2+2)/2 = 2$
- \rightarrow chọn P2



154

2.4.3.6 Multilevel Queue Scheduling

- Sử dụng nhiều hàng đợi ready (ví dụ foreground, background)
- Mỗi hàng đợi chứa các tiến trình có cùng độ ưu tiên và có giải thuật điều phối riêng, ví dụ:
 - foreground: RR
 - background: FCFS
- HĐH phải định thời cho các hàng đợi
- Ví dụ:



155

Multilevel Queue Scheduling (tt)

- Fixed priority scheduling:
 - Chọn tiến trình trong hàng đợi có độ ưu tiên từ cao đến thấp.
 - Có thể có starvation.
- Time slice:
 - Mỗi hàng đợi được nhận một khoảng thời gian chiếm CPU và phân phối cho các process trong hàng đợi khoảng thời gian đó.
 - Ví dụ: 80% cho hàng đợi foreground định thời bằng RR và 20% cho hàng đợi background định thời bằng giải thuật FCFS

156

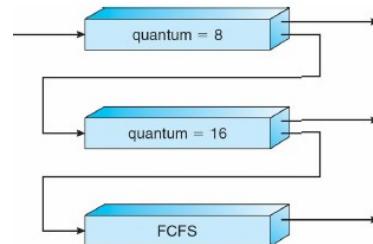
2.4.3.7 Multilevel Feedback Queue

- Trong hệ thống Multilevel Feedback Queue, các tiến trình có thể được di chuyển giữa các queue tùy theo đặc tính của nó tại mỗi thời điểm, Ví dụ:
 - Tiến trình sử dụng CPU quá lâu → chuyển sang một hàng đợi có độ ưu tiên thấp hơn
 - Tiến trình chờ quá lâu trong một hàng đợi có độ ưu tiên thấp → chuyển lên hàng đợi có độ ưu tiên cao hơn (*aging*, giúp tránh starvation)

157

Multilevel Feedback Queue (tt)

- Ví dụ: Có 3 hàng đợi
 - Q0, dùng RR với quantum 8ms
 - Q1, dùng RR với quantum 16ms
 - Q2, dùng FCFS
- Giải thuật
 - Tiến trình mới sẽ vào hàng đợi Q0.
Khi đến lượt, sẽ được cấp phát quantum là 8 ms, nếu chưa xử lý xong → chuyển xuống cuối hàng đợi Q1
 - Tại Q1, tiến trình thực thi sẽ được cấp phát quantum là 16ms, nếu chưa xử lý xong → chuyển xuống cuối hàng đợi Q2



158

Bài tập

- Cho các tiến trình và thời gian đến hàng đợi, thời gian sử dụng CPU mô tả trong bảng sau

P	Arrival time	CPU burst	Priority
P1	1	3	3
P2	2	1	2
P3	3	5	1
P4	4	4	4
P5	5	2	5

- Vẽ biểu đồ Gantt, tính thời gian đáp ứng, thời gian hoàn thành, thời gian chờ với các thuật toán điều phối:
 - FCFS
 - Priority (thời điểm lựa chọn độc quyền)
 - Priority (thời điểm lựa chọn không độc quyền)
 - SJF (thời điểm lựa chọn độc quyền)
 - SRT (thời điểm lựa chọn không độc quyền)
 - HRRN
 - Round Robin (với $q = 2$)

159

2.4. Vấn đề điều phối luồng

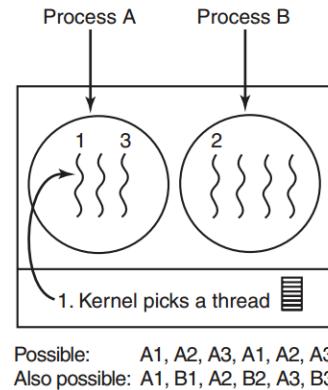
- Các luồng liên lạc với nhau thông qua các biến toàn cục của tiến trình
- Cơ chế điều phối luồng phụ thuộc vào cách cài đặt luồng:
 - Cài đặt trong kernel-space
 - Cài đặt trong user-space
 - Cài đặt trong kernel-space và user-space

160

Vấn đề điều phối luồng (tt)

- Cài đặt trong kernel-space:

- Bảng quản lý thread lưu ở phần kernel-space
- HĐH chịu trách nhiệm điều phối thread

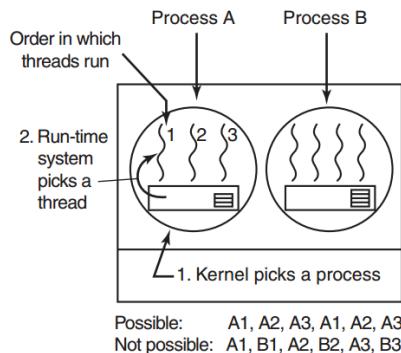


161

Vấn đề điều phối luồng (tt)

- Cài đặt trong user-space

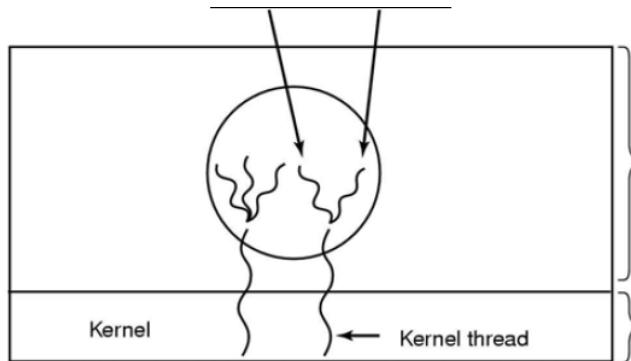
- Bảng quản lý thread lưu ở phần user-space
- Tiến trình chịu trách nhiệm điều phối thread



162

Vấn đề điều phối luồng (tt)

- Cài đặt trong kernel-space và user-space
 - Một số luồng mức user được cài đặt bằng một luồng mức kernel
 - Một luồng của HĐH quản lý một số luồng của tiến trình



163

Chương 3

Deadlock



164
www.cunghoclaptrinh.com

Nội dung

1. Đặc điểm sử dụng tài nguyên của các tiến trình
2. Tình trạng deadlock
3. Giải pháp xử lý

165

Tài liệu tham khảo

- Andrew S. Tanenbaum, Modern Operating Systems, Pearson, 2015
 - Chương 6
- Abraham Silberschatz, Peter Baer Galvin, Greg Gagne, Operating System Concepts, John Wiley, 2013
 - Chương 7
- William Stallings, Operating Systems: Internals and Design Principles, Pearson, 2015
 - Chương 6

166

3.1. Đặc điểm sử dụng tài nguyên của các tiến trình

- Trong môi trường multiprogramming nhiều tiến trình có thể yêu cầu cùng một tài nguyên để có thể thực thi.
- Có hai loại tài nguyên:
 - Preemptable: tài nguyên có thể chia sẻ giữa các tiến trình (vd: bộ nhớ)
 - Nonpreemptable: tài nguyên không thể chia sẻ giữa các tiến trình (vd: đầu ghi đĩa)
- Các bước tiến trình sử dụng tài nguyên :
 - Yêu cầu tài nguyên (request)
 - Sử dụng tài nguyên (use)
 - Giải phóng (trả) tài nguyên (release)

167

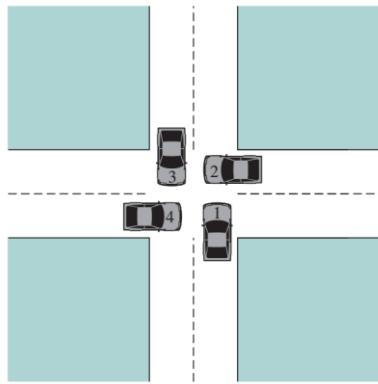
Đặc điểm sử dụng tài nguyên của các tiến trình (tt)

- Khi tiến trình yêu cầu tài nguyên:
 - Nếu tài nguyên có sẵn: sử dụng
 - Nếu tài nguyên không có sẵn: chuyển sang trạng thái chờ.
- Tài nguyên mà các tiến trình yêu cầu có thể là một tài nguyên không thể chia sẻ.
- Các tiến trình chờ tài nguyên có thể sẽ kẹt bao giờ thay đổi lại trạng thái được vì các tài nguyên mà nó yêu cầu đang bị giữ bởi các tiến trình khác → tắc nghẽn (deadlock)
- *Deadlock là tình trạng khi hệ thống tồn tại một tập hợp các tiến trình bị khóa, trong đó mỗi tiến trình đều đang chờ một sự kiện mà chỉ một tiến trình khác có thể tạo ra, và kết quả là không một tiến trình nào có thể hoàn thành.*

168

Đặc điểm sử dụng tài nguyên của các tiến trình (tt)

- Ví dụ: tình trạng kẹt xe



169

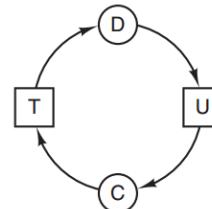
3.2. Tình trạng deadlock

- Điều kiện xảy ra deadlock
- Đồ thị cấp phát tài nguyên

170

Điều kiện xảy ra deadlock

- **Ngăn chặn lẫn nhau (Mutual exclusion):** Một tài nguyên bị chiếm bởi một tiến trình, và không tiến trình nào khác có thể sử dụng tài nguyên này
- **Giữ và đợi (Hold-and-wait):** Một tiến trình đang giữ ít nhất một tài nguyên và chờ một số tài nguyên khác đang bị một tiến trình khác chiếm giữ.
- **Không có đặc quyền (No-preemption):** không thể thu hồi tài nguyên đã cấp cho tiến trình mà phải chờ tiến trình trả lại tài nguyên sau khi đã sử dụng xong
- **Chờ đợi vòng tròn (circular wait):** Một tập tiến trình $\{P_0, P_1, \dots, P_n\}$
 - P_0 chờ một tài nguyên do P_1 chiếm giữ
 - P_1 chờ một tài nguyên khác do P_2 chiếm giữ, ...,
 - P_{n-1} chờ tài nguyên do P_n chiếm giữ
 - P_n chờ tài nguyên do P_0 chiếm giữ



171

Đồ thị cấp phát tài nguyên (Resource allocation graph)

- Dùng để mô tả một cách chính xác tình trạng bế tắc
- Là một đồ thị có hướng $G=(V, E)$ với V là tập đỉnh, E là tập các cạnh
- V được chia thành hai tập con
 - $P = \{P_0, P_1, \dots, P_n\}$ là tập các tiến trình trong hệ thống
 - $R = \{R_0, R_1, \dots, R_m\}$ là tập các loại tài nguyên trong hệ thống thỏa mãn $P \cap R = \emptyset$

172

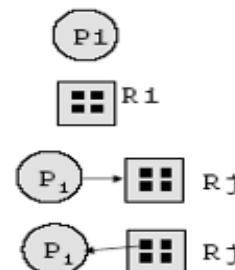
Đồ thị cấp phát tài nguyên (tt)

- Một cạnh có hướng từ P_i tới R_j (ký hiệu $P_i \rightarrow R_j$): tiến trình P_i chờ tài nguyên R_j - gọi là **cạnh yêu cầu**
- Một cạnh có hướng từ R_j tới P_i (ký hiệu $R_j \rightarrow P_i$): tài nguyên R_j đã được cấp phát cho tiến trình P_i - gọi là **cạnh gán**
- **Biểu diễn:**
 - P_i : hình tròn
 - R_j : hình chữ nhật. R_j có thể có nhiều thể hiện \rightarrow biểu diễn mỗi thể hiện là một chấm nằm trong hình vuông.
 - Một cạnh yêu cầu trở tới chỉ một hình vuông R_j
 - Một cạnh gán gán tới một trong các dấu chấm trong hình vuông

173

Đồ thị cấp phát tài nguyên (tt)

- Khi P_i yêu cầu một thể hiện R_j , một cạnh yêu cầu được chèn vào đồ thị cấp phát tài nguyên.
- Khi yêu cầu được đáp ứng, cạnh yêu cầu được truyền tới cạnh gán.
- Khi quá trình không còn cần truy xuất tới tài nguyên, nó giải phóng tài nguyên \rightarrow cạnh gán bị xoá.



174

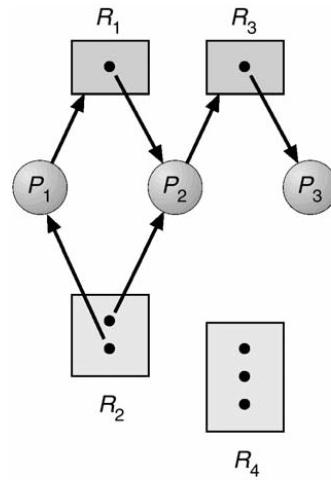
Đồ thị cấp phát tài nguyên (tt)

- **Ví dụ:** các tập P, R, và E:

- $P = \{P_1, P_2, P_3\}$
- $R = \{R_1, R_2, R_3, R_4\}$
- $E = \{P_1 \rightarrow R_1, P_2 \rightarrow R_3, R_1 \rightarrow P_2, R_2 \rightarrow P_2, R_3 \rightarrow P_3\}$

- **Trạng thái tiến trình:**

- P_1 đang giữ một thẻ hiện của R_2 và đang chờ một thẻ hiện của R_1
- P_2 đang giữ một thẻ hiện của R_1 và R_2 và đang chờ một thẻ hiện của R_3
- P_3 đang giữ một thẻ hiện của R_3



175

Đồ thị cấp phát tài nguyên (tt)

- Nếu đồ thị không chứa chu trình: **không** có deadlock.
- Nếu đồ thị có chứa chu trình: **có thẻ** có deadlock.
- Nếu một chu trình bao gồm chỉ một tập hợp các loại tài nguyên, mỗi loại tài nguyên chỉ có một thẻ hiện:
 - Có deadlock
 - Mỗi tiến trình chứa trong chu trình bị deadlock.
 - Một chu trình trong đồ thị là điều kiện cần và đủ để tồn tại deadlock.
- Nếu mỗi loại tài nguyên có nhiều thẻ hiện:
 - Chu trình không có deadlock.
 - Một chu trình trong đồ thị là điều kiện **cần nhưng chưa đủ** để tồn tại deadlock.

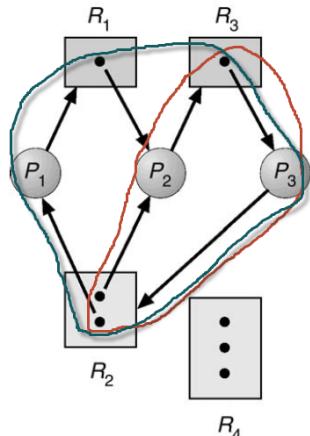
176

Đồ thị cấp phát tài nguyên (tt)

P3 yêu cầu một thể hiện của R2 \rightarrow có deadlock

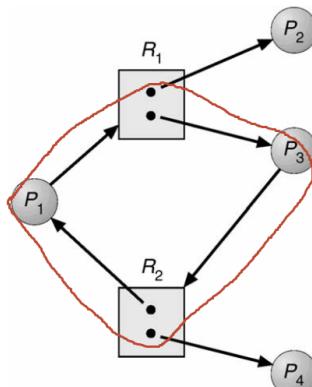
P1 \rightarrow R1 \rightarrow P2 \rightarrow R3 \rightarrow P3 \rightarrow R2 \rightarrow P1

P2 \rightarrow R3 \rightarrow P3 \rightarrow R2 \rightarrow P2



Có chu trình nhưng không có deadlock

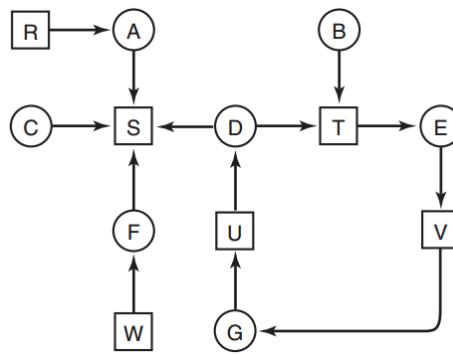
P1 \rightarrow R1 \rightarrow P3 \rightarrow R2 \rightarrow P1



177

Đồ thị cấp phát tài nguyên (tt)

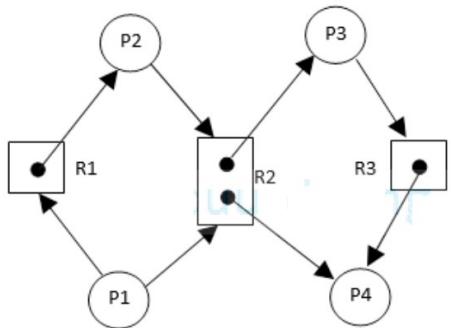
- Xét đồ thị cấp phát tài nguyên sau, cho biết hệ thống có xảy ra tình trạng deadlock hay không?



178

Đồ thị cấp phát tài nguyên (tt)

- Xét đồ thị cấp phát tài nguyên sau, cho biết hệ thống có xảy ra tình trạng deadlock hay không?



179

3.3. Giải pháp xử lý deadlock

- Không quan tâm: có thể bỏ qua, xem như deadlock không bao giờ xảy ra trong hệ thống.
- Ngăn chặn deadlock
- Tránh xảy ra deadlocks
- Phát hiện tình trạng deadlock và khôi phục.

180

3.3.1 Ngăn chặn deadlock

- Ngăn chặn một trong bốn điều kiện làm xảy ra deadlock
 - Loại trừ lẫn nhau (mutual exclusion)
 - Giữ và chờ cấp thêm tài nguyên (Hold and Wait)
 - Không đòi lại tài nguyên từ quá trình đang giữ chúng (No Preemption)
 - Tồn tại chu trình trong đồ thị cấp phát tài nguyên (Circular Wait)

181

Ngăn chặn mutual exclusion

- Đảm bảo hệ thống không có các tài nguyên không thể chia sẻ: gần như không tránh được

182

Ngăn chặn Giữ và chờ ...

- Đảm bảo khi một tiến trình yêu cầu tài nguyên, nó không giữ bất cứ tài nguyên nào khác
 - Bắt buộc mỗi tiến trình phải yêu cầu toàn bộ tài nguyên cần thiết một lần, nếu có đủ tài nguyên hệ thống sẽ cấp phát, nếu không đủ tài nguyên, tiến trình sẽ bị block.
 - Khi yêu cầu tài nguyên, tiến trình không được giữ một tài nguyên khác, nếu có thì phải trả tài nguyên đang giữ trước khi yêu cầu.
- Hạn chế :
 - Hiệu quả sử dụng tài nguyên rất thấp
 - Có khả năng starvation

183

Ngăn chặn không có đặc quyền

- Cách 1:** Nếu một tiến trình đang giữ một số tài nguyên và yêu cầu tài nguyên không có sẵn thì tiến trình phải trả tất cả tài nguyên hiện đang giữ
- Cách 2:** nếu tài nguyên mà một tiến trình yêu cầu không có sẵn:
 - Nếu các tài nguyên đó đang được cấp phát tới các tiến trình khác đang chờ tài nguyên bổ sung → thu hồi lại → cấp cho tiến trình đang yêu cầu.
 - Nếu tài nguyên đó đang được cấp phát tới một tiến trình không đợi tài nguyên, tiến trình đang yêu cầu phải chờ và một số tài nguyên nó đang giữ có thể được đòi lại nếu có tiến trình khác yêu cầu.
- Được áp dụng cho tài nguyên mà trạng thái của nó có thể được lưu và cập nhật dễ dàng như các thanh ghi CPU, không gian bộ nhớ...

184

Ngăn chặn tạo chu trình trong đồ thị cấp phát tài nguyên

- Cấp phát tài nguyên theo một thứ tự phân cấp như sau:
 - Gọi $R = \{R_1, R_2, \dots, R_m\}$ là tập hợp loại tài nguyên.
 - Đánh số thứ tự cho mỗi loại tài nguyên bằng cách định nghĩa hàm ánh xạ một-một $F: R \rightarrow N$ (N là tập hợp các số tự nhiên)
 - Ví dụ: $F(\text{ổ băng từ}) = 1, F(\text{đĩa từ}) = 5, F(\text{máy in}) = 12$
 - Một tiến trình đang giữ tài nguyên R_i thì chỉ được yêu cầu tài nguyên R_j nếu $F(R_j) > F(R_i)$

185

3.3.2 Tránh deadlock

- Sử dụng thuật toán cấp phát tài nguyên với đầu vào là các thông tin yêu cầu cấp phát tài nguyên của tiến trình để quyết định cấp phát tài nguyên sao cho deadlock không xảy ra.
- Có nhiều thuật toán tránh deadlock
- Thuật toán đơn giản:
 - Biết trước số thể hiện của mỗi loại tài nguyên mà mỗi tiến trình sẽ sử dụng.
 - → Hệ thống sẽ có đủ thông tin để xây dựng thuật toán cấp phát không gây ra bế tắc

186

Tránh deadlock (tt)

- Mục tiêu các thuật toán là kiểm tra trạng thái cấp phát tài nguyên “động” để đảm bảo điều kiện tạo chu trình chờ không xảy ra
- Trạng thái cấp phát tài nguyên được xác định bởi:
 - số lượng tài nguyên rỗi
 - số lượng tài nguyên đã cấp phát cho các tiến trình
 - số lượng tối đa tài nguyên mà mỗi tiến trình yêu cầu
- Hai thuật toán:
 - Thuật toán đồ thị cấp phát tài nguyên
 - Thuật toán banker

187

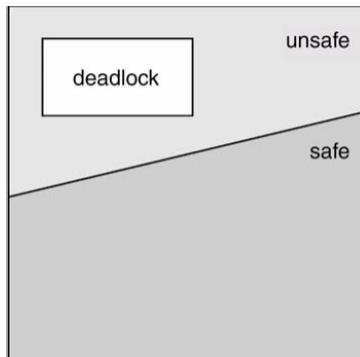
Khái niệm Trạng thái an toàn (safe-state)

- Một trạng thái cấp phát tài nguyên được gọi là an toàn nếu hệ thống có thể cấp phát tài nguyên cho các tiến trình **theo một thứ tự nào đó** mà vẫn tránh được deadlock.
- Nói cách khác, một hệ thống ở trong trạng thái an toàn nếu và chỉ nếu ở đó tồn tại một **thứ tự an toàn**
- Thứ tự của các tiến trình $\langle P_1, P_2, \dots, P_n \rangle$ là một thứ tự an toàn khi: với mỗi thứ tự P_i , các tài nguyên mà P_i yêu cầu vẫn có thể được thoả mãn bởi:
 - tài nguyên hiện có + các tài nguyên được giữ bởi tất cả P_j , với $j < i$.

188

Trạng thái an toàn (tt)

- Trạng thái an toàn không là trạng thái bế tắc
- Trạng thái bế tắc là trạng thái không an toàn
- Trạng thái không an toàn có thể không là trạng thái bế tắc



189

Trạng thái an toàn (tt)

- Ví dụ: Xét một hệ thống có 12 tài nguyên là 12 băng từ và 3 tiến trình P0, P1, P2 với các yêu cầu cấp phát:
 - P0 yêu cầu tối đa 10 băng từ
 - P1 yêu cầu tối đa 4 băng từ
 - P2 yêu cầu tối đa 9 băng từ
- Giả sử tại thời điểm t0:
 - P0 đang giữ 5 băng từ
 - P1 và P2: mỗi tiến trình giữ 2 băng từ.
 - → có 3 băng từ rỗng.
- Tại thời điểm t0, hệ thống ở trạng thái an toàn: thứ tự cấp phát $\langle P1, P0, P2 \rangle$ thỏa mãn điều kiện an toàn
- Giả sử ở thời điểm t1, P2 có yêu cầu và **được cấp phát 1 băng từ**
 → Hệ thống không ở trạng thái an toàn nữa

P	Yêu cầu nhiều nhất	Yêu cầu hiện tại
P0	10	5
P1	4	2
P2	9	2

190

Giải thuật đồ thị cấp phát tài nguyên

- Được áp dụng cho các tài nguyên chỉ có 1 thể hiện
- Sử dụng đồ thị cấp phát tài nguyên và bổ sung thêm một cạnh báo trước (claim edge)
- Cạnh báo trước $P_i \rightarrow R_j$ cho biết P_i có thể yêu cầu cấp phát tài nguyên R_j , được biểu diễn trên đồ thị bằng các đường có nét đứt.
- Các tài nguyên mà tiến trình cần phải được thông báo trước: tất cả các cạnh báo trước của P_i phải xuất hiện trong đồ thị cấp phát tài nguyên.

191

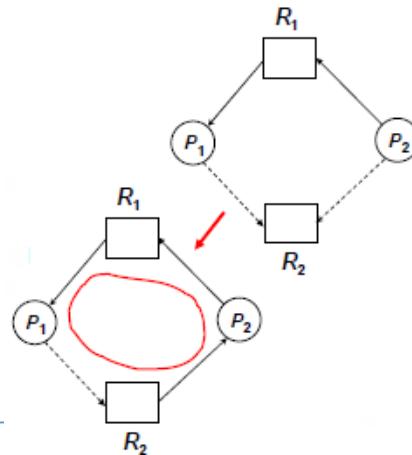
Giải thuật đồ thị cấp phát tài nguyên (tt)

- Giả sử P_i yêu cầu tài nguyên R_j . Yêu cầu này chỉ có thể được chấp nhận nếu ta chuyển cạnh báo trước $P_i \rightarrow R_j$ thành cạnh cấp phát $R_j \rightarrow P_i$ mà không tạo ra một chu trình
- Kiểm tra bằng cách sử dụng thuật toán phát hiện chu trình trong đồ thị: Nếu có n tiến trình trong hệ thống, thuật toán phát hiện chu trình có độ phức tạp tính toán $O(n^2)$
 - Nếu không có chu trình: trạng thái an toàn \rightarrow cấp phát
 - Ngược lại: trạng thái không an toàn.

192

Giải thuật đồ thị cấp phát tài nguyên (tt)

- Ví dụ: Giả sử P2 yêu cầu cấp phát R2
- Mặc dù R2 rãnh nhưng chúng ta không thể cấp phát R2, vì nếu cấp phát ta sẽ có chu trình trong đồ thị và gây ra chờ vòng → Hệ thống ở trạng thái không an toàn
- Nhận xét: không thể áp dụng tới hệ thống cấp phát tài nguyên với nhiều thể hiện của mỗi loại tài nguyên



193

Thuật toán banker

- Còn gọi là giải thuật nhà băng
- Được dùng cho các hệ thống mà mỗi tài nguyên có nhiều thể hiện, kém hiệu quả hơn thuật toán đồ thị phân phối tài nguyên
- Thuật toán banker có thể dùng trong ngân hàng: không bao giờ cấp phát tài nguyên (tiền) gây nên tình huống sau này không đáp ứng được nhu cầu của tất cả các khách hàng

194

Thuật toán banker (tt)

- Khi một tiến trình mới đưa vào hệ thống, hệ thống phải biết tiến trình cần tối đa bao nhiêu thẻ hiện của mỗi loại tài nguyên để có thể thực thi.
- Khi tiến trình yêu cầu các tài nguyên, hệ thống phải xác định việc cấp phát các tài nguyên này có đảm bảo hệ thống ở trạng thái an toàn hay không.
 - an toàn: cấp tài nguyên
 - không an toàn: tiến trình phải chờ cho tới khi đủ tài nguyên.

195

Thuật toán banker (tt)

- Gọi n là số tiến trình trong hệ thống và m là số loại tài nguyên trong hệ thống, cần có các cấu trúc dữ liệu:
 - **Available**: một vector có chiều dài m biểu diễn số lượng tài nguyên sẵn có của mỗi loại.
 - Nếu $\text{Available}[j] = k \rightarrow$ có k thẻ hiện của loại tài nguyên R_j sẵn dùng.
 - **Max**: ma trận $n \times m$ biểu diễn số lượng tối đa yêu cầu của mỗi tiến trình đối với mỗi loại tài nguyên
 - Nếu $\text{Max}[i, j] = k \rightarrow P_i$ có thẻ yêu cầu nhiều nhất k thẻ hiện của loại tài nguyên R_j .

196

Thuật toán banker (tt)

- **Allocation:** ma trận $n \times m$ biểu diễn số lượng tài nguyên của mỗi loại mà mỗi tiến trình đang giữ.
 - Nếu $\text{Allocation} [i, j] = k \rightarrow P_i$ hiện được cấp k thẻ hiện của loại tài nguyên R_j
- **Need:** ma trận $n \times m$ biểu diễn số lượng tài nguyên của mỗi loại mà mỗi tiến trình đang cần thêm.
 - Nếu $\text{Need} [i, j] = k \rightarrow$ tiến trình P_i có thể cần thêm k thẻ hiện của loại tài nguyên R_j để hoàn thành tác vụ của nó.
 - $\text{Need} [i, j] = \text{Max} [i, j] - \text{Allocation} [i, j]$
- Số lượng và giá trị các biến trên biến đổi theo trạng thái của hệ thống

197

Thuật toán banker (tt)

- Qui ước so sánh hai vector: gọi X và Y là các vector có chiều dài n , ta có:
 - $X \leq Y$ nếu và chỉ nếu $X[i] \leq Y[i] \forall i$
 - **Thí dụ:**
 - $X = (1, 7, 3, 2)$
 - $Y = (0, 3, 2, 1)$
 - $\Rightarrow Y \leq X$.

198

Thuật toán banker (tt)

• Thuật toán trạng thái an toàn:

1. Với Work và Finish là hai vector độ dài m và n. Khởi tạo:
 - Work = Available
 - Finish[i] = false for $i = 0, 1, \dots, n - 1$.
2. Tìm i sao cho $\text{Finish}[i] == \text{false}$ và $\text{Need}[i] \leq \text{Work}$
 - Nếu không tìm được i , chuyển đến bước 4
3. $\text{Work} = \text{Work} + \text{Allocation}[i]$,
 $\text{Finish}[i] = \text{true}$
 Chuyển đến bước 2
4. Nếu $\text{Finish}[i] == \text{true}$ Vì thì hệ thống ở trạng thái an toàn
 - Độ phức tạp tính toán của thuật toán trạng thái an toàn:
 $O(m \cdot n^2)$

199

Thuật toán banker (tt)

• Thuật toán yêu cầu tài nguyên:

1. Nếu $\text{Request}[i] \leq \text{Need}[i]$, chuyển đến bước 2
 Ngược lại thông báo lỗi (không có tài nguyên rồi)
2. Nếu $\text{Request}[i] \leq \text{Available}$, chuyển đến bước 3.
 Ngược lại P_i phải chờ vì không có tài nguyên
3. Giả sử hệ thống cấp phát các tài nguyên cho tiến trình $P_i \rightarrow$ cập nhật trạng thái như sau :
 - $\text{Available} = \text{Available} - \text{Request}[i]$
 - $\text{Allocation}[i] = \text{Allocation}[i] + \text{Request}[i]$
 - $\text{Need}[i] = \text{Need}[i] - \text{Request}[i]$
4. Áp dụng thuật toán kiểm tra trạng thái an toàn:
 - Nếu hệ thống ở trạng thái an toàn \rightarrow cấp phát tài nguyên cho P_i
 - Ngược lại $\rightarrow P_i$ phải chờ và trạng thái của hệ thống được khôi phục như cũ

200

Thuật toán banker (tt)

• Ví dụ: hệ thống gồm

- 3 loại tài nguyên A, B, C.
 - A có 10 thể hiện
 - B có 5 thể hiện
 - C có 7 thể hiện
- 5 tiến trình P0, P1, P2, P3, P4
- Giả sử trạng thái của hệ thống tại thời điểm T0:

1. Tại thời điểm T0, hệ thống có ở trạng thái an toàn hay không
2. Giả sử P1 yêu cầu cấp phát (1,0,2). Yêu cầu này có thể được thỏa mãn hay không?
Tương tự với:
P4(3,3,0)
P0(0,2,0)
P3(0,2,1)

	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			

201

Thuật toán banker (tt)

	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			

- Ma trận Need = Max – Allocation

	Need		
	A	B	C
P0	7	4	3
P1	1	2	2
P2	6	0	0
P3	0	1	1
P4	4	3	1

- Hệ thống hiện ở trong trạng thái an toàn do thứ tự $\langle P1, P3, P4, P0, P2 \rangle$ thỏa tiêu chuẩn an toàn.

202

Thuật toán banker (tt)

- Giả sử rằng quá trình P1 yêu cầu thêm (1,0,2)

- Request[1](1,0,2) \leq Available (3,3,2) \rightarrow True
- Request[1](1,0,2) \leq Need[1] (1,2,2) \rightarrow True
- Cập nhật trạng thái mới:

	<u>Allocation</u>	<u>Need</u>	<u>Available</u>
	A B C	A B C	A B C
P_0	0 1 0	7 4 3	2 3 0
P_1	3 0 2	0 2 0	
P_2	3 0 2	6 0 0	
P_3	2 1 1	0 1 1	
P_4	0 0 2	4 3 1	

- Kiểm tra bằng thuật toán trạng thái an toàn

203

3.3.3 Phát hiện Deadlock

- Nếu không áp dụng phòng tránh hoặc ngăn chặn deadlock thì hệ thống có thể bị deadlock. Khi đó:
 - Cần có thuật toán kiểm tra trạng thái để xem có deadlock xuất hiện hay không
 - Thuật toán khôi phục nếu deadlock xảy ra
- Cần có giải thuật áp dụng cho hệ thống mà **mỗi loại tài nguyên chỉ có một thể hiện** và hệ thống mà **mỗi loại tài nguyên có nhiều thể hiện**

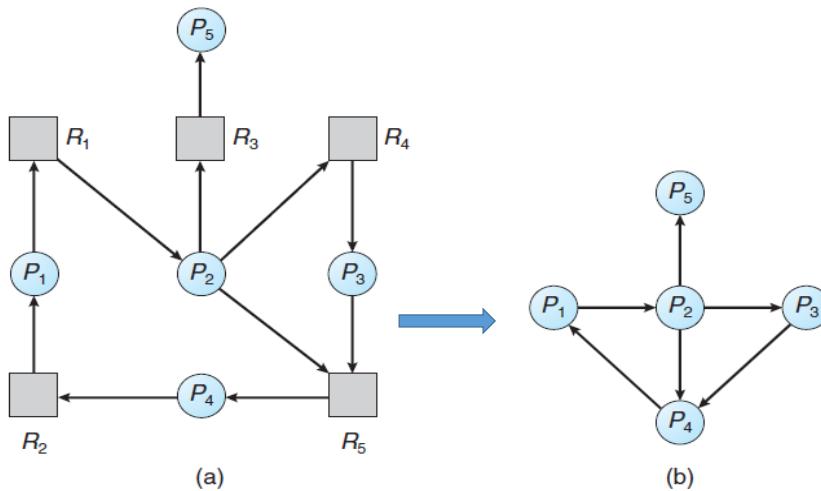
204

Tài nguyên chỉ có một thể hiện

- **Sử dụng thuật toán đồ thị chờ** (wait-for) có được từ đồ thị cấp phát tài nguyên bằng cách xóa các đỉnh tài nguyên và nối các cung liên quan
- $P_i \rightarrow P_j$: P_i đang chờ P_j giải phóng tài nguyên mà P_i cần
- $P_i \rightarrow P_j$: tồn tại trong đồ thị chờ nếu và chỉ nếu đồ thị cấp phát tài nguyên tương ứng có hai cung:
 - $P_i \rightarrow R_q$
 - $R_q \rightarrow P_j$
- Hệ thống có deadlock nếu **đồ thị chờ có chu trình**
- Để phát hiện deadlock:
 - Cần cập nhật đồ thị chờ
 - Thực hiện định kỳ thuật toán phát hiện chu trình

205

Tài nguyên chỉ có một thể hiện (tt)



206

Tài nguyên có nhiều thể hiện

- Sử dụng các cấu trúc dữ liệu tương tự như giải thuật Banker:
 - **Available**: vector m phần tử biểu diễn số lượng thể hiện của mỗi loại tài nguyên
 - **Allocation**: Ma trận $n \times m$ biểu diễn số thể hiện của mỗi loại tài nguyên đang được cấp phát cho các tiến trình
 - **Request**: Ma trận $n \times m$ biểu diễn yêu cầu hiện tại của mỗi tiến trình.
 - Nếu $\text{Request}[i][j] = k \rightarrow$ tiến trình P_i yêu cầu cấp phát k thể hiện của tài nguyên R_j

207

Tài nguyên có nhiều thể hiện (tt)

1. Khởi tạo:
 - $\text{Work} = \text{Available}$
 - For $i = 0, 1, \dots, n-1$
 - nếu $\text{Allocation}[i] \neq 0$ gán $\text{Finish}[i] = \text{false}$
 - ngược lại gán $\text{Finish}[i] = \text{true}$
2. Tìm i sao cho $\text{Finish}[i] == \text{false}$ và $\text{Request}[i] \leq \text{Work}$. Nếu không tìm thấy i , chuyển đến bước 4
3. $\text{Work} = \text{Work} + \text{Allocation}[i]$, $\text{Finish}[i] = \text{true} \rightarrow$ chuyển đến bước 2
4. Nếu $\text{Finish}[i] == \text{false}$ với $0 \leq i \leq n-1$ thì hệ thống đang bị bế tắc (và tiến trình P_i đang bế tắc).
- Độ phức tạp tính toán của thuật toán: $O(m \cdot n^2)$

208

Tài nguyên có nhiều thể hiện (tt)

- Ví dụ: 5 tiến trình P0, P1, P2, P3, P4
3 loại tài nguyên A, B, C.
 - A có 7 thể hiện
 - B có 2 thể hiện
 - C có 6 thể hiện.
- Trạng thái cấp phát tài nguyên tại thời điểm T0:

	Allocation			Request			Available		
	A	B	C	A	B	C	A	B	C
P0	0	1	0	0	0	0	0	0	0
P1	2	0	0	2	0	2			
P2	3	0	3	0	0	0			
P3	2	1	1	1	0	0			
P4	0	0	2	0	0	2			

- Hệ thống không ở trong trạng thái deadlock (thứ tự $\langle P0, P2, P3, P4, P1 \rangle \rightarrow Finish[i] = \text{true}$ cho với mọi i)

209

Tài nguyên có nhiều thể hiện (tt)

- Giả sử P2 yêu cầu thêm 1 thể hiện của tài nguyên C

	Request		
	A	B	C
P_0	0	0	0
P_1	2	0	2
P_2	0	0	1
P_3	1	0	0
P_4	0	0	2

- Hệ thống bị deadlock bao gồm các tiến trình P1, P2, P3, P4.

210

Sử dụng giải thuật phát hiện deadlock

- Khi nào sử dụng giải thuật phát hiện deadlock? Phụ thuộc vào hai yếu tố:
 - Deadlock có khả năng xảy ra thường xuyên?
 - Bao nhiêu tiến trình sẽ bị ảnh hưởng khi có deadlock?
- Sử dụng thuật toán phát hiện:
 - Theo định kỳ: có thể có nhiều chu trình trong đồ thị, không biết được tiến trình/request nào gây ra bế tắc
 - Khi có yêu cầu cấp phát tài nguyên: tốn tài nguyên CPU

211

3.3.4 Phục hồi deadlock

1. Kết thúc tiến trình

- Huỷ bỏ tất cả tiến trình bị deadlock:
 - phá vỡ chu trình deadlock
 - chi phí cao
- Hủy bỏ từng tiến trình tại mỗi thời điểm cho đến khi chu trình deadlock bị xóa, tiêu chí chọn để hủy một tiến trình:
 - Độ ưu tiên
 - Thời gian đã thực hiện và thời gian còn lại
 - Số lượng và các loại tài nguyên đã sử dụng
 - Các tài nguyên cần cấp phát thêm
 - Số lượng các tiến trình phải kết thúc
 - Tiến trình là tương tác hay xử lý theo lô (batch)

212

Phục hồi deadlock

2. Lấy lại tài nguyên

- Chọn tài nguyên nào và tiến trình nào để thực hiện?
 - Thứ tự ưu tiên.
 - Số lượng tài nguyên mà tiến trình bị deadlock đang giữ
 - Thời gian mà tiến trình đã thực thi
- Khôi phục trạng thái của tiến trình đã chọn như thế nào?
 - Khôi phục tiến trình về trạng thái an toàn và khởi động lại từ trạng thái đó: khó
 - Hủy bỏ tiến trình và khởi động lại để phá vỡ deadlock.
 - Yêu cầu hệ thống phải lưu trữ thông tin về trạng thái của tất cả các tiến trình đang chạy.
- Làm thế nào để tránh tình trạng một tiến trình luôn bị bắt buộc giải phóng tài nguyên?

213

Chương 4

Quản lý bộ nhớ



NỘI DUNG

1. Vấn đề quản lý bộ nhớ
2. Mô hình quản lý bộ nhớ thực
3. Mô hình quản lý bộ nhớ ảo

Tài liệu tham khảo

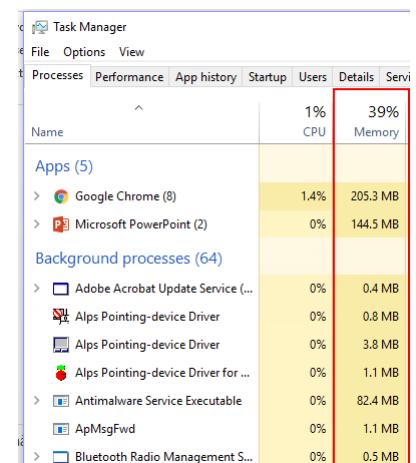
- Andrew S. Tanenbaum, Modern Operating Systems, Pearson, 2015
 - Chương 3
- Abraham Silberschatz, Peter Baer Galvin, Greg Gagne, Operating System Concepts, John Wiley, 2013
 - Chương 8, 9

4.1. Vấn đề quản lý bộ nhớ

- Bộ nhớ chính là nơi mà CPU có thể truy cập trực tiếp
- Dữ liệu của chương trình phải được ghi vào bộ nhớ chính trước khi được xử lý
- Việc trao đổi thông tin giữa CPU với môi trường ngoài thông qua thao tác đọc/ghi dữ liệu vào một địa chỉ trong bộ nhớ
- Bộ nhớ chính được tổ chức như mảng một chiều các từ nhớ (word), mỗi từ nhớ có một địa chỉ

Vấn đề quản lý bộ nhớ (tt)

- Trong hệ thống đơn chương, bộ nhớ chính được chia thành hai phần:
 - Một phần dành cho hệ điều hành
 - Một phần dành cho các chương trình đang được thực thi.
- Trong một hệ thống đa chương:
 - Phần bộ nhớ dành cho các chương trình người dùng phải được chia nhỏ hơn
- Nhiệm vụ phân chia bộ nhớ được thực hiện động bởi hệ điều hành gọi là quản lý bộ nhớ.



Vấn đề quản lý bộ nhớ (tt)

- Quản lý bộ nhớ là công việc của hệ điều hành cùng với sự hỗ trợ của phần cứng nhằm phân phối, sắp xếp các tiến trình trong bộ nhớ sao cho hiệu quả.
- Ba chức năng chính của HĐH đối với quản lý bộ nhớ:
 - Theo dõi những phần nào của bộ nhớ đang được sử dụng.
 - Quyết định những tiến trình nào sẽ được nạp vào bộ nhớ.
 - Định vị và tái định vị không gian bộ nhớ khi cần thiết.
- Mục tiêu quản lý bộ nhớ: nạp càng nhiều tiến trình vào bộ nhớ càng tốt (gia tăng mức độ đa chương)

219

Vấn đề quản lý bộ nhớ (tt)

- Các yêu cầu đối với việc quản lý bộ nhớ
 - Cấp phát bộ nhớ cho các tiến trình
 - Tái định vị (relocation): khi swapping,...
 - Bảo vệ: phải kiểm tra truy xuất bộ nhớ có hợp lệ không
 - Chia sẻ: cho phép các tiến trình chia sẻ vùng nhớ chung
 - Kết gán địa chỉ nhớ luận lý của user vào địa chỉ thực
- Hàng đợi nhập hệ thống: các chương trình trên đĩa đang chờ được nạp vào bộ nhớ

220

Vấn đề quản lý bộ nhớ (tt)

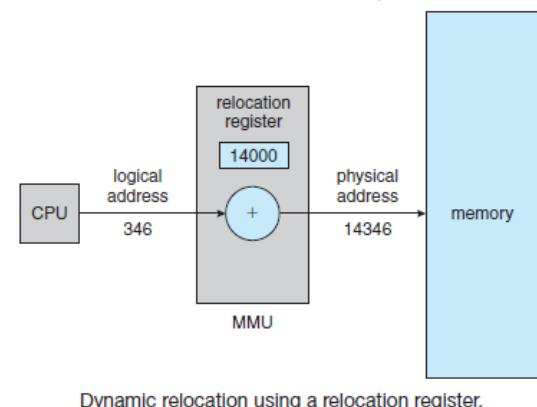
- **Địa chỉ vật lý** (physical address - địa chỉ thực) là một vị trí thực trong bộ nhớ chính.
- **Địa chỉ luận lý** (logical address): địa chỉ được CPU tạo ra, là một vị trí nhớ được diễn tả trong một chương trình.
 - Các trình biên dịch (compiler) tạo ra mã lệnh chương trình mà trong đó mọi tham chiếu bộ nhớ đều là địa chỉ luận lý
 - **Địa chỉ tuyệt đối** (absolute address): ánh xạ đến một vị trí thực trong bộ nhớ.
 - **Địa chỉ tương đối** (relative address) (khả tái định vị - relocatable address): độ dời tương đối so với một vị trí xác định bởi địa chỉ tuyệt đối. (VD: 12 byte so với vị trí bắt đầu chương trình)

221

Vấn đề quản lý bộ nhớ (tt)

• Chuyển đổi địa chỉ:

- Khi một lệnh được thực thi, các tham chiếu đến địa chỉ luận lý phải được chuyển đổi thành địa chỉ thực.
- **MMU (Memory Management Unit):**
cơ chế phần cứng được sử dụng để chuyển đổi địa chỉ luận lý thành địa chỉ vật lý tại thời điểm xử lý



222

Vấn đề quản lý bộ nhớ (tt)

- Thời điểm gán kết (chuyển đổi) địa chỉ:

- Thời điểm biên dịch:

- Nếu biết trước địa chỉ bộ nhớ của tiến trình thì có thể kết gán địa chỉ tuyệt đối lúc biên dịch (Ví dụ: nếu biết rằng một tiến trình của người dùng sẽ bắt đầu từ vị trí R, thì mã trình biên dịch được tạo sẽ bắt đầu tại vị trí đó).

- Thay đổi vị trí nạp chương trình → biên dịch lại (MS-DOS)

- Thời điểm nạp:

- Tại thời điểm biên dịch, nếu chưa biết tiến trình sẽ nằm ở đâu trong bộ nhớ thì TBD phải sinh mã khả tái định vị.

- Vào thời điểm nạp, phải chuyển đổi địa chỉ khả tái định vị thành địa chỉ thực dựa trên một địa chỉ nền (base address).

- Địa chỉ thực được tính toán vào thời điểm nạp chương trình → phải nạp lại nếu địa chỉ nền thay đổi

223

Vấn đề quản lý bộ nhớ (tt)

- Thời điểm gán kết (chuyển đổi) địa chỉ:

- Thời điểm thực thi:

- Trong quá trình thực thi, tiến trình có thể được di chuyển từ segment này sang segment khác trong bộ nhớ thì quá trình gán kết địa chỉ được thực hiện tại thời điểm thực thi

- CPU tạo ra địa chỉ luận lý cho process

- Cần sự hỗ trợ của phần cứng cho việc ánh xạ địa chỉ.

- Ví dụ: trường hợp địa chỉ luận lý là relocatable thì có thể dùng thanh ghi base và limit,...

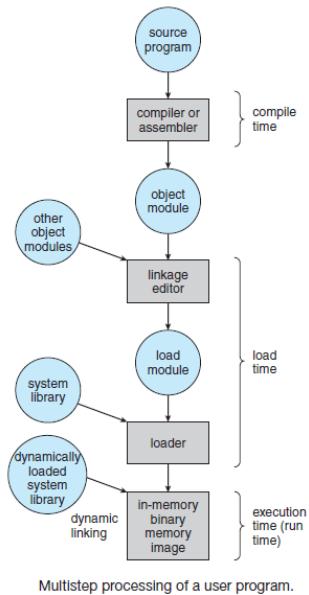
- Sử dụng các cơ chế swapping, paging, segmentation

- Được sử dụng trong hầu hết các hệ điều hành

224

Vấn đề quản lý bộ nhớ (tt)

- Các bước thực thi chương trình



225

Vấn đề quản lý bộ nhớ (tt)

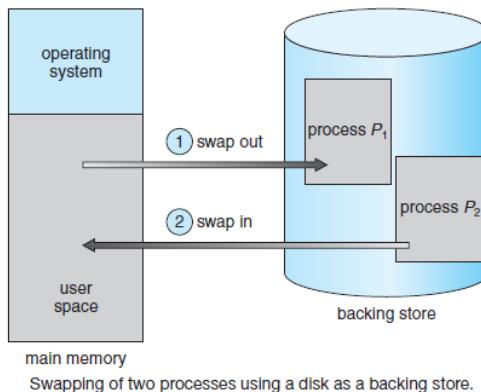
- Phân mảnh nội (Internal Fragmentation) – mỗi block được cấp phát lớn hơn yêu cầu bộ nhớ một ít
- Phân mảnh ngoại vi (External Fragmentation) – tổng bộ nhớ trống thỏa yêu cầu, nhưng không liên tục
- Giải pháp phân mảnh ngoại vi: kết hợp
 - Chuyển các vùng trống thành một khối bộ nhớ liên tục
 - Chỉ thực hiện được nếu HĐH hỗ trợ biên dịch địa chỉ trong thời gian thực thi

226

Vấn đề quản lý bộ nhớ (tt)

• Swaping:

- Tiến trình ở trạng thái chờ được tạm thời chuyển ra bộ nhớ phụ (swap out), sau đó được chuyển vào bộ nhớ chính (swap in) để tiếp tục xử lý



227

Vấn đề quản lý bộ nhớ (tt)

• Swaping (tt)

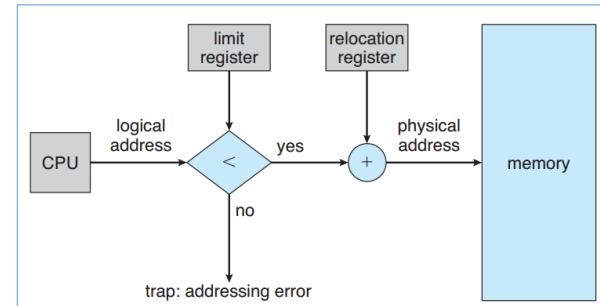
- Tăng số lượng các tiến trình đồng hành → hiệu quả trong môi trường đa chương
- Cần xác định vị trí của tiến trình trong bộ nhớ chính khi được chuyển ra/vào, phụ thuộc vào thời điểm kết gán địa chỉ
 - Thời điểm nạp, cần phải cấp phát lại đúng vùng nhớ cho tiến trình
 - Thời điểm xử lý: có thể nạp tiến trình vào một vùng nhớ bất kỳ
- Cần sử dụng bộ nhớ phụ
- Thời gian chuyển đổi: mỗi tiến trình cần được phân chia thời gian sử dụng CPU sao cho không cảm thấy bị chậm trễ do swap

228

Vấn đề quản lý bộ nhớ (tt)

• Memory Protection (bảo vệ bộ nhớ)

- Ngăn một tiến trình truy cập bộ nhớ không hợp lệ
- Cần có hỗ trợ của phần cứng, sử dụng hai thanh ghi:
 - relocation register: ghi vị trí bắt đầu của tiến trình trong bộ nhớ vật lý
 - limit register: ghi phạm vi địa chỉ logic



229

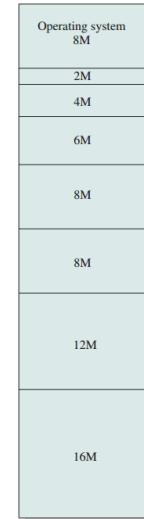
4.2. Mô hình quản lý bộ nhớ thực

- Cấp phát liên tục:
 - Phân vùng cố định
 - Phân vùng động
- Cấp phát không liên tục
 - Phân trang
 - Phân đoạn

230

4.2.1 Phân vùng cố định

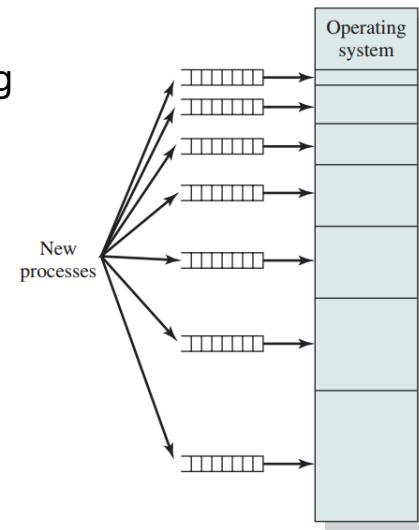
- Bộ nhớ được chia thành n phân vùng (partition) có kích thước cố định
- Các tiến trình có nhu cầu bộ nhớ sẽ được lưu trữ trong hàng đợi
- Có hai cách tổ chức:
 - Sử dụng nhiều hàng đợi
 - Sử dụng một hàng đợi



231

Phân vùng cố định (tt)

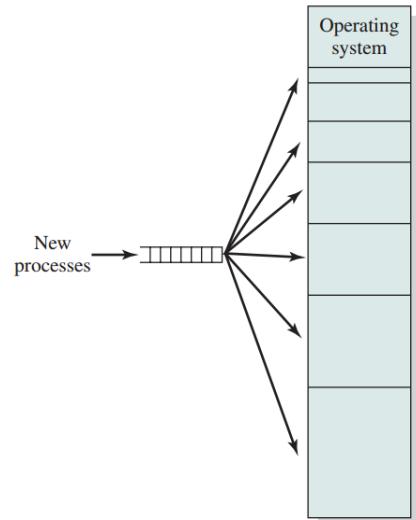
- Sử dụng nhiều hàng đợi:
 - Mỗi phân vùng có một hàng đợi tương ứng
 - Tiến trình được tạo sẽ được đưa vào hàng đợi của phân vùng có kích thước nhỏ nhất đủ chứa nó
 - Hạn chế: có thể có một số hàng đợi trống vì không có tiến trình có kích thước phù hợp



232

Phân vùng cố định (tt)

- Sử dụng một hàng đợi:
 - Chỉ có một hàng đợi chung cho tất cả partition
 - Khi cần nạp một tiến trình vào bộ nhớ chính → chọn partition nhỏ nhất còn trống



233

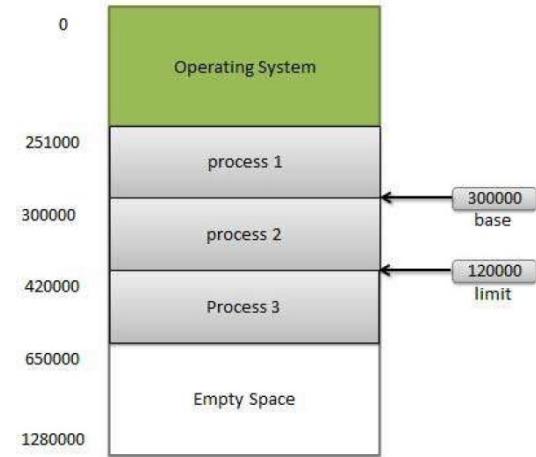
Phân vùng cố định (tt)

- Nhận xét:
 - Đơn giản, chi phí thấp
 - Kích thước tiến trình không vừa đúng bằng kích thước phân vùng chứa nó, phần bộ nhớ còn lại sẽ lãng phí (phân mảnh nội)
 - Số lượng phân vùng được cài đặt cố định → ảnh hưởng đến mức độ đa chương
 - → trong môi trường đa chương cần tái định vị (relocation) và bảo vệ các tiến trình.

234

Phân vùng cố định (tt)

- Tái định vị vào thời điểm nạp chương trình: cập nhật lại các địa chỉ trong chương trình khi chương trình được nạp vào bộ nhớ
- Bảo vệ: sử dụng hai thanh ghi
 - Thanh ghi nền (base register)
 - Thanh ghi giới hạn (limit register)
 - Khi tiến trình được tạo:
 - Nạp vào thanh ghi nền địa chỉ bắt đầu của phân vùng được cấp cho tiến trình
 - Nạp vào thanh ghi giới hạn kích thước của tiến trình



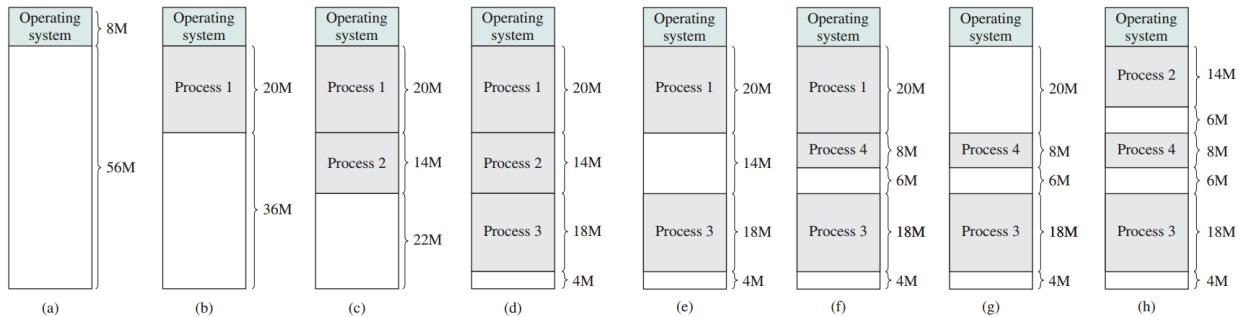
235

4.2.2 Phân vùng động

- Cấp phát vùng nhớ vừa đúng với kích thước của tiến trình được tạo
- Thu hồi vùng nhớ khi tiến trình kết thúc
- → Kích thước vùng nhớ cấp cho tiến trình và vị trí bắt đầu của các phân vùng là động

236

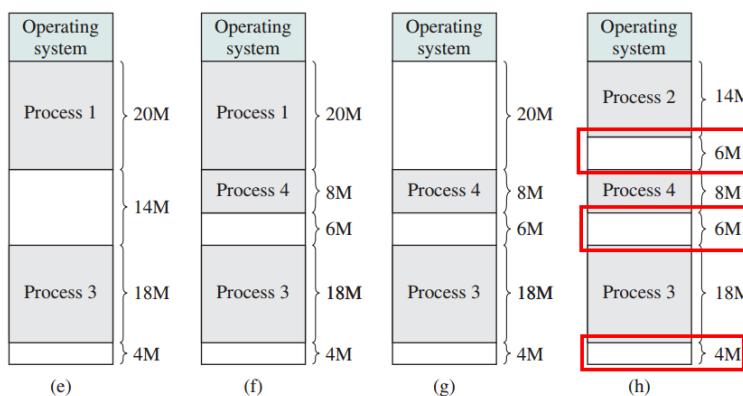
Phân vùng động (tt)



237

Phân vùng động (tt)

- Nhận xét: Không phân mảnh nội, nhưng phân mảnh ngoại vi
 - Ví dụ: process mới có kích thước 7M?



238

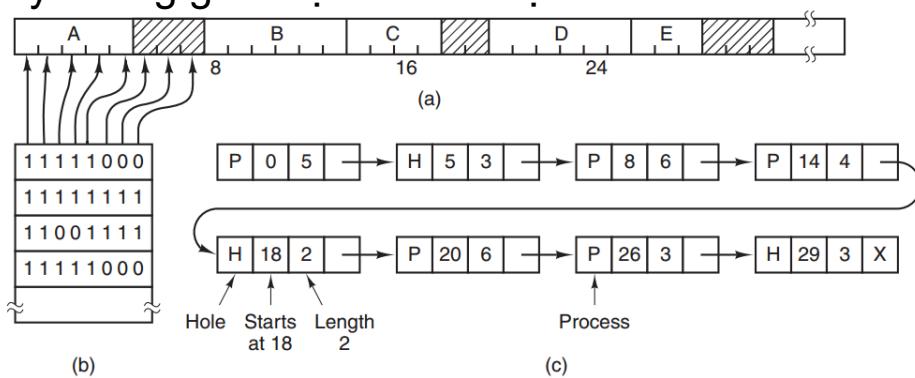
Phân vùng động (tt)

- Quản lý không gian bộ nhớ: có hai phương pháp
 - Sử dụng mảng bit: bộ nhớ được chia thành những đơn vị cấp phát, mỗi bit trên dãy bit cho biết trạng thái của một đơn vị
 - 0: còn trống
 - 1: đã cấp phát cho tiến trình
 - Sử dụng Danh sách liên kết: mỗi phần tử quản lý một vùng trên bộ nhớ, gồm ba trường chính:
 - trường thứ nhất cho biết khối nhớ đã cấp phát (P: process) hay đang còn trống (H: Hole)
 - trường thứ hai cho biết thứ tự của đơn vị cấp phát đầu tiên trong block (Starts)
 - trường thứ ba cho biết block gồm bao nhiêu đơn vị cấp phát (Length)

239

Phân vùng động (tt)

- Quản lý không gian bộ nhớ - ví dụ



(a) một phần không gian bộ nhớ với 5 tiến trình và 3 vùng trống

(b) quản lý các đơn vị cấp phát bằng mảng các bit

(c) quản lý các đơn vị cấp phát bằng danh sách liên kết

240

Phân vùng động (tt)

• Các thuật toán cấp phát vùng nhớ cho tiến trình:

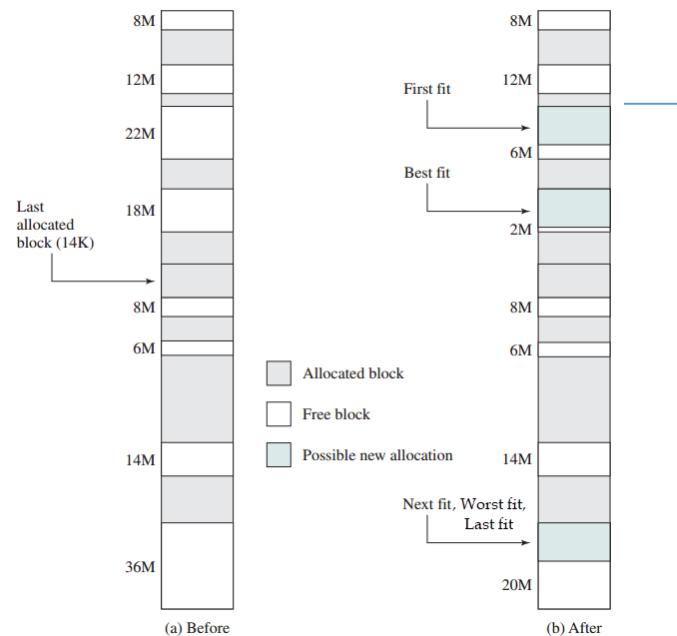
- **Best-fit:** chọn khối nhớ có kích thước nhỏ nhất vừa đúng bằng kích thước của tiến trình cần được nạp vào bộ nhớ.
- **First-fit:** chọn khối nhớ trống đầu tiên có kích thước đủ lớn để nạp tiến trình.
- **Last-fit:** chọn khối nhớ trống cuối cùng có kích thước đủ lớn để nạp tiến trình.
- **Next-fit:** tương tự như First-fit nhưng 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
- **Worst-fit:** chọn khối nhớ có kích thước lớn nhất để nạp tiến trình

241

Phân vùng động (tt)

• Ví dụ: cấp phát khối nhớ cho tiến trình có kích thước 16M theo các thuật toán:

- First-fit
- Best-fit
- Next-fit
- Last-fit
- Worst-fit



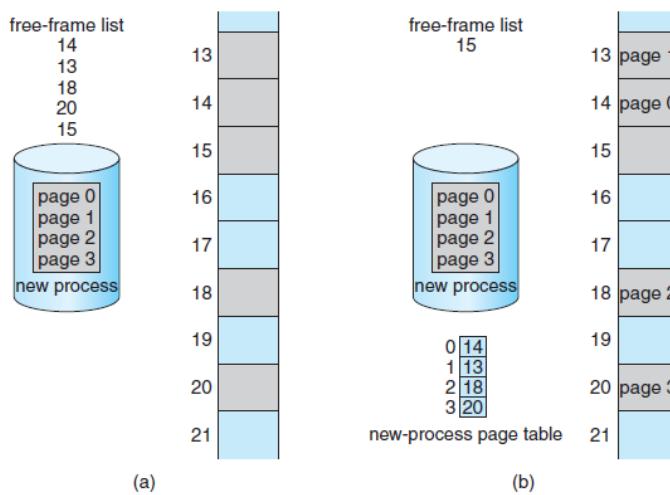
242

4.2.3 Phân trang (paging)

- Cơ chế phân trang cho phép không gian địa chỉ thực (physical address space) của một process có thể không liên tục nhau
- Bộ nhớ vật lý được chia thành các khối (block) có kích thước cố định và bằng nhau, gọi là khung trang (page frame), kích thước của frame là lũy thừa của 2 (512 bytes -16Mb)
- Bộ nhớ luận lý (không gian địa chỉ luận lý) cũng được chia thành các khối có cùng kích thước với khung trang, gọi là trang (page).
- Khi cần nạp một tiến trình, các trang của tiến trình sẽ được nạp vào các khung trang còn trống

243

Phân trang (tt)

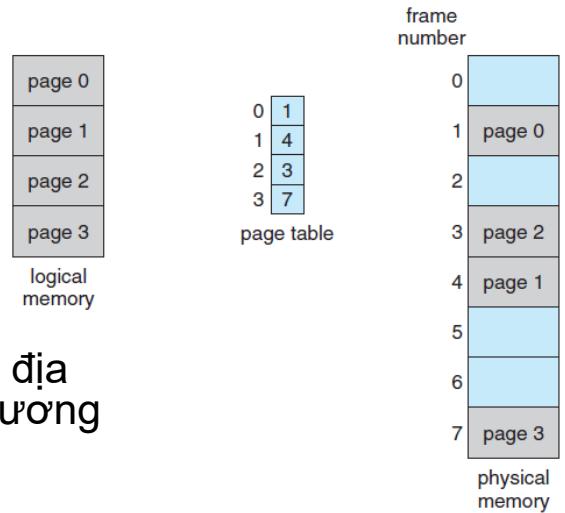


Free frames (a) before allocation and (b) after allocation.

244

Phân trang (tt)

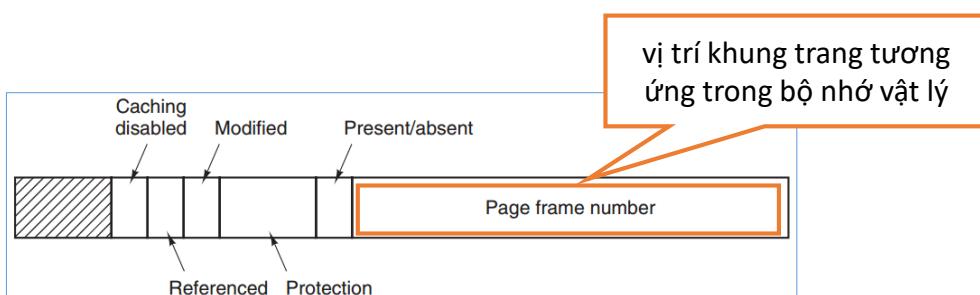
- HĐH phải thiết lập **một bảng phân trang** (page table) để ánh xạ địa chỉ luận lý thành địa chỉ thực
- Mỗi tiến trình có một bảng phân trang, được quản lý bằng một con trỏ lưu giữ trong PCB.
- Mỗi phần tử trong page table chứa địa chỉ bắt đầu của vị trí lưu trữ trang tương ứng trong bộ nhớ vật lý



245

Phân trang (tt)

- Cấu trúc một phần tử trong page table:



246

Phân trang (tt)

0	0	0	7	0
1	1	—	8	5
2	2	—	9	6
3	3	—	10	11
				12

Process A page table Process B page table Process C page table Process D page table

Frame number	Main memory
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	

(a) Fifteen available frames

Frame number	Main memory
0	A.0
1	A.1
2	A.2
3	A.3
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	

(b) Load process A

Frame number	Main memory
0	A.0
1	A.1
2	A.2
3	A.3
4	B.0
5	B.1
6	B.2
7	
8	
9	
10	
11	
12	
13	
14	

(c) Load process B

Frame number	Main memory
0	A.0
1	A.1
2	A.2
3	A.3
4	B.0
5	B.1
6	B.2
7	C.0
8	C.1
9	C.2
10	C.3
11	
12	
13	
14	

(d) Load process C

Frame number	Main memory
0	A.0
1	A.1
2	A.2
3	A.3
4	D.0
5	D.1
6	D.2
7	C.0
8	C.1
9	C.2
10	C.3
11	
12	
13	
14	

(e) Swap out B

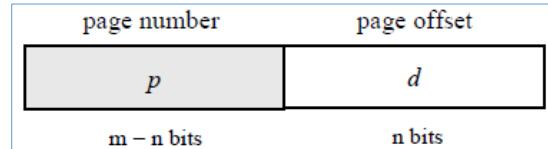
Frame number	Main memory
0	A.0
1	A.1
2	A.2
3	A.3
4	D.0
5	D.1
6	D.2
7	C.0
8	C.1
9	C.2
10	C.3
11	D.3
12	D.4
13	
14	

(f) Load process D

247

Phân trang (tt)

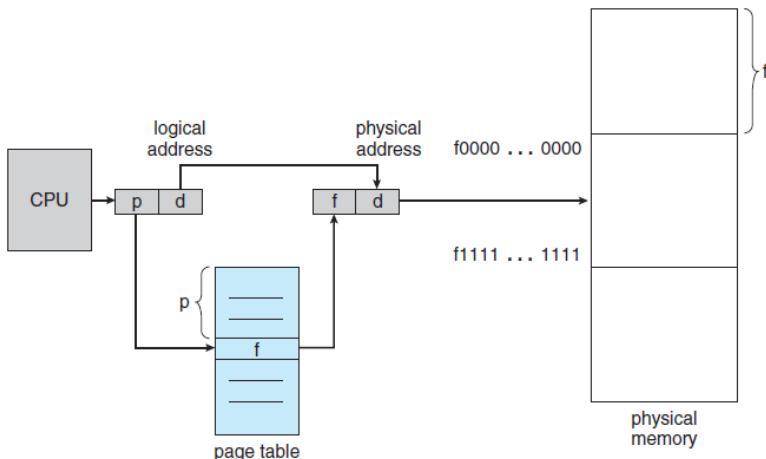
- Địa chỉ logic phát sinh từ CPU được chia thành hai phần:
 - Page number (p): số hiệu trang, sử dụng như chỉ mục đến phần tử tương ứng trong bảng trang
 - Page offset (d): địa chỉ tương đối trong trang, kết hợp với địa chỉ bắt đầu của trang để tạo ra địa chỉ vật lý mà trình quản lý bộ nhớ sử dụng (độ dời trang)
- Kích thước trang do phần cứng qui định, thường là lũy thừa của 2 (512 bytes - 16M)
- VD: Nếu kích thước của không gian địa chỉ là 2^m và kích thước trang là 2^n :
 - p: m - n bit cao của địa chỉ logic
 - d: n bit thấp của địa chỉ logic
- Bảng phân trang sẽ có tổng cộng $2^m/2^n = 2^{m-n}$ mục (entry)



248

Phân trang (tt)

- Phân cứng chuyển đổi địa chỉ logic sang địa chỉ vật lý:



249

Phân trang (tt)

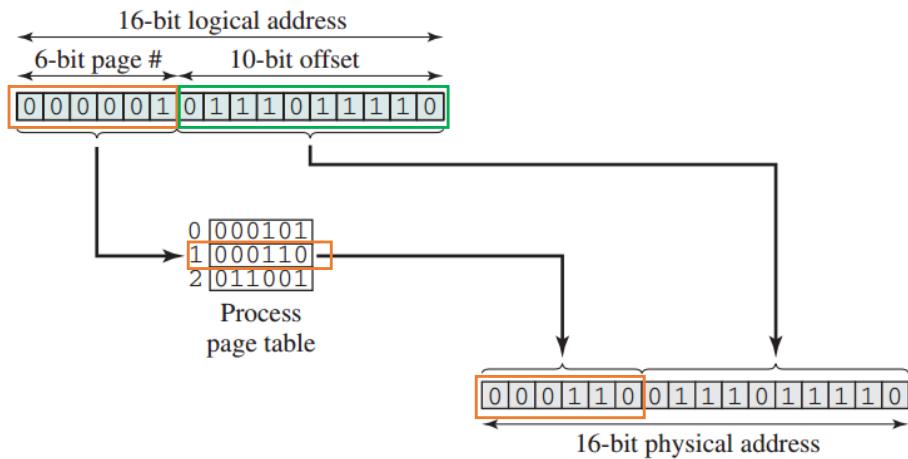
- Quá trình chuyển đổi địa chỉ logic sang địa chỉ vật lý:

- Lấy m bit cao của địa chỉ \Rightarrow số thứ tự trang (vị trí trang trong bảng trang)
- Dựa vào phần tử tương ứng trong bảng trang, tìm được số thứ tự khung vật lý k (vị trí khung trang trong bộ nhớ vật lý)
- Địa chỉ vật lý bắt đầu của khung là $k \cdot 2^n$
- Địa chỉ vật lý của byte được tham chiếu là địa chỉ bắt đầu của khung cộng với độ dời
- \Rightarrow Chỉ cần thêm số khung vào trước dãy bit biểu diễn độ lệch

250

Phân trang (tt)

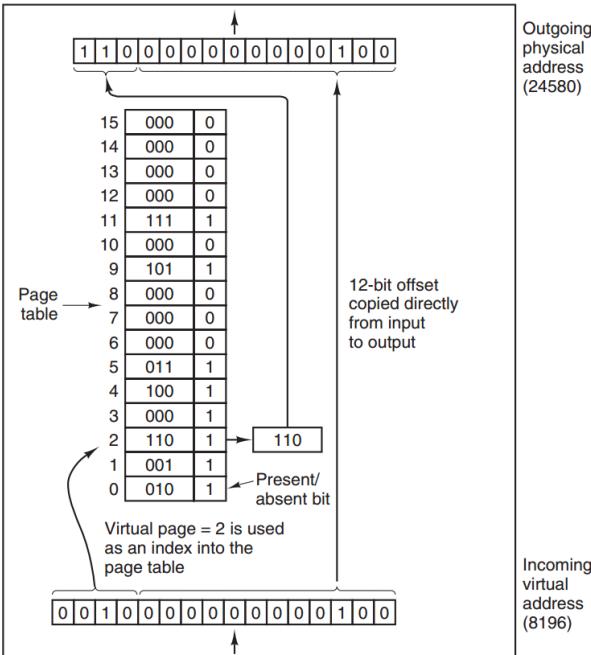
- Quá trình chuyển địa chỉ logic sang địa chỉ vật lý :



251

Phân trang (tt)

- Ví dụ:



252

Phân trang (tt)

• Cài đặt bảng trang

- Bảng phân trang thường được lưu giữ trong bộ nhớ chính
 - Mỗi tiến trình được hệ điều hành cấp một bảng phân trang
 - Thanh ghi *page-table base* (PTBR) trả đến bảng phân trang
 - Thanh ghi *page-table length* (PTLR) biểu thị kích thước của bảng phân trang (có thể được dùng trong cơ chế bảo vệ bộ nhớ)

• Nhận xét về cơ chế phân trang:

- Cấp phát không liên tục: các trang của cùng một tiến trình có thể nằm ở các vị trí không liên tục trong bộ nhớ → không phân mảnh ngoại vi
- Có phân mảnh nội: khung cuối cùng có thể không sử dụng hết kích thước

253

Phân trang (tt)

• Tăng tốc độ phân trang

- Mỗi tác vụ truy cập dữ liệu/lệnh cần hai thao tác truy xuất vùng nhớ
 - Dùng page number (p) làm index để truy cập mục trong bảng phân trang (nằm trong bộ nhớ chính) để lấy số frame
 - Dùng page offset (d) để truy xuất dữ liệu/lệnh trong frame
- → truy cập bộ nhớ chính hai lần → truy xuất chậm → HĐH dùng một bộ phận cache phần cứng có tốc độ truy xuất và tìm kiếm cao, gọi là thanh ghi kết hợp (associative register) hoặc Translation Lookaside Buffers (TLBs)

254

Phân trang (tt)

- **Thanh ghi kết hợp (Translation Lookaside Buffers - TLB):**
 - Là thanh ghi hỗ trợ tìm kiếm truy xuất dữ liệu với tốc độ rất nhanh, nằm trong MMU
 - Số mục của TLB khoảng 8 – 256 mục
 - Mỗi mục chứa thông tin về một trang (page number, frame number, modified bit, valid bit, protection code (quyền đọc / ghi / thực thi))
 - Khi có chuyển ngữ cảnh, TLB bị xóa
 - Khi TLB đầy, thay thế mục dùng LRU
 - Ánh xạ địa chỉ ảo
 - Nếu page number nằm trong TLB (hit, trúng) → lấy ngay được số frame → tiết kiệm được thời gian truy cập bộ nhớ chính để lấy số frame trong bảng phân trang.
 - Ngược lại (miss, sai), phải lấy số frame từ bảng phân trang

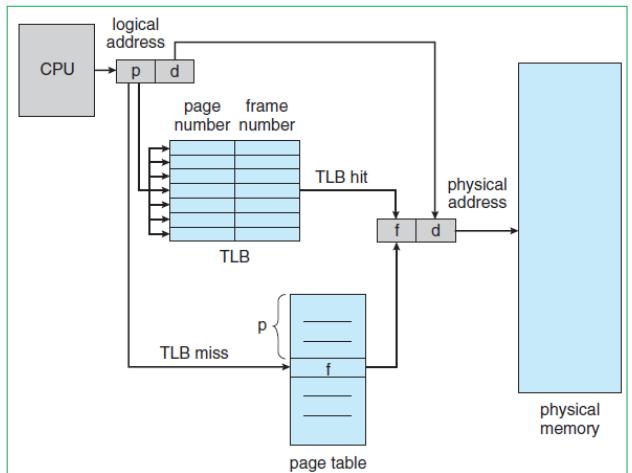
255

Phân trang (tt)

Nội dung một TLB

Valid	Virtual page	Modified	Protection	Page frame
1	140	1	RW	31
1	20	0	R X	38
1	130	1	RW	29
1	129	1	RW	62
1	19	0	R X	50
1	21	0	R X	45
1	860	1	RW	14
1	861	1	RW	75

Paging hardware với TLB



256

Phân trang (tt)

• Bảo vệ bộ nhớ:

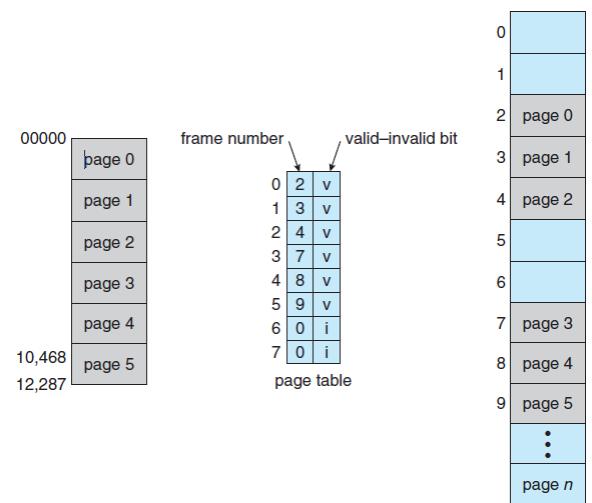
- Việc bảo vệ bộ nhớ được hiện thực với *bit bảo vệ* (protection bits) trong mỗi phần tử của bảng phân trang. Các bit này biểu thị các thuộc tính: read-only, read-write, execute-only
- Ngoài ra, phần tử trong bảng trang còn có một *valid bit* với hai giá trị:
 - 1: "valid" - cho biết là trang của process, do đó là một trang hợp lệ.
 - 0: "invalid" - cho biết là trang không của process, do đó là một trang bất hợp lệ.
- Dùng PTLR để kiểm tra truy xuất đến bảng phân trang có hợp lệ hay không

257

Phân trang (tt)

• Ví dụ:

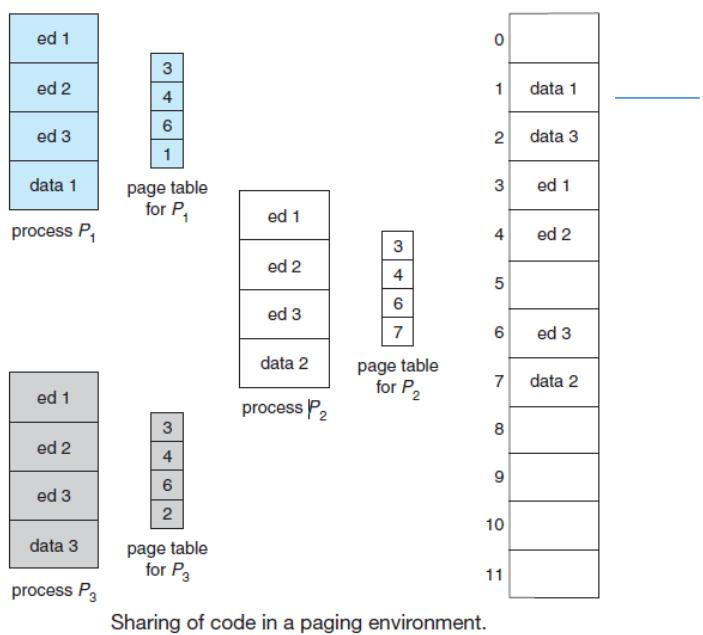
- Hệ thống với không gian địa chỉ 14 bit (0-16383), mỗi trang có kích thước 2K = 2048
- Process chỉ sử dụng các địa chỉ từ 0-10468
- Địa chỉ trong các trang 0, 1, 2, 3, 4 và 5 được đánh dấu là hợp lệ trong bảng trang
- Các tham chiếu nào ngoài địa chỉ 0-10468 đều là bất hợp lệ
- Frame 9 (chứa page 5) bị phân mảnh nội, các truy cập vào địa chỉ lên đến 12287 là hợp lệ. Các địa chỉ từ 12288-16383 là không hợp lệ (invalid)



258

Phân trang (tt)

- Chia sẻ các trang nhớ: cho phép nhiều tiến trình có thể thực thi một mã lệnh tại một thời điểm



259

Phân trang (tt)

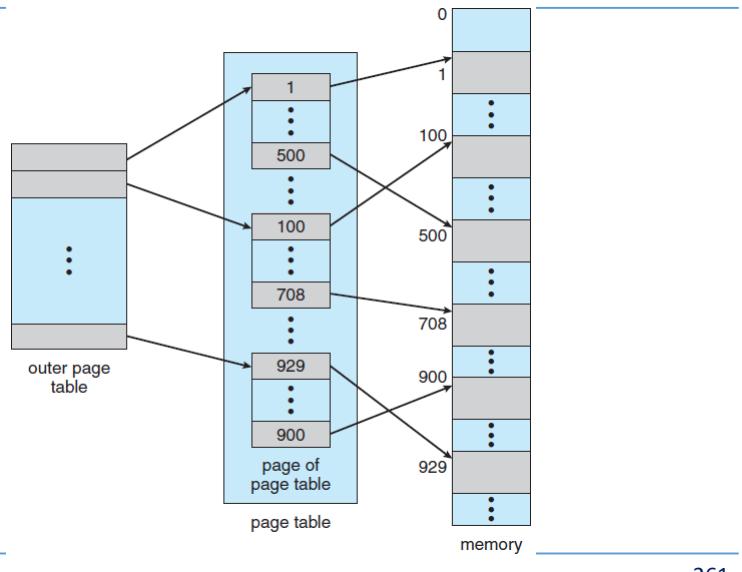
- Bảng phân trang phân cấp**

- Các hệ thống hiện đại đều hỗ trợ không gian địa chỉ ảo rất lớn (2^{32} đến 2^{64})
- Giả sử không gian địa chỉ là 2^{32} bit, kích thước trang là 4KB (2^{12}) \rightarrow bảng phân trang sẽ có $2^{32}/2^{12} = 2^{20} = 1M$ mục.
- Giả sử mỗi mục cần 4 byte thì mỗi process cần 4MB cho bảng phân trang
- Giải pháp: thay vì dùng một bảng phân trang duy nhất cho mỗi process, chia bảng phân trang thành nhiều phần nhỏ hơn (phân trang bảng phân trang), và chỉ giữ những bảng phân trang cần thiết trong bộ nhớ \rightarrow bảng phân trang 2 mức (two-level page table).

260

Phân trang (tt)

- Bảng phân trang 2 mức

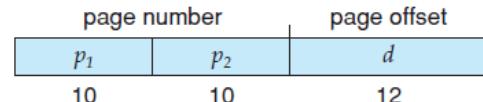


261

Phân trang (tt)

- Ví dụ:

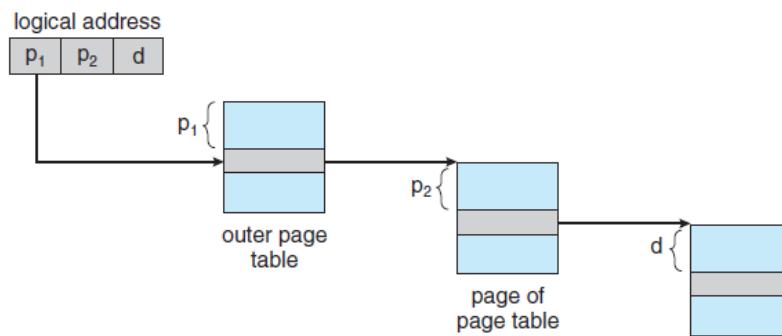
- Địa chỉ logic (trên máy 32-bit, trang cỡ $4K=2^{12}$) được chia thành:
 - Địa chỉ trang: 20 bits
 - Địa chỉ offset: 12 bits.
- Bảng trang 2 cấp: 20 bit địa chỉ trang được chia thành:
 - 10-bit địa chỉ trang cấp 1
 - 10-bit địa chỉ trang cấp 2
- Khi đó địa chỉ logic có dạng:
 - p_1 là chỉ số đến bảng phân trang mức 1 (outer page)
 - p_2 là chỉ số đến bảng phân trang mức 2



262

Phân trang (tt)

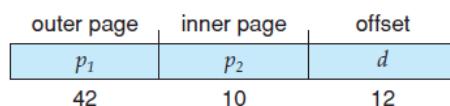
- Sơ đồ chuyển đổi địa chỉ (address-translation scheme) cho cơ chế bảng phân trang 2 mức, 32-bit địa chỉ



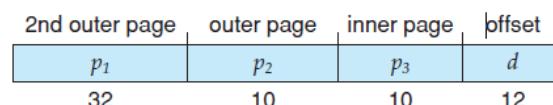
263

Phân trang (tt)

- Với hệ điều hành 64 bits, bảng trang 2 mức không còn phù hợp:
 - Giả sử trang có kích thước $4K$ (2^{12}) \rightarrow bảng trang có 2^{52} mục
 - Với bảng trang 2 mức, địa chỉ logic có dạng:



- Bảng trang mức 1 khá lớn (2^{42}) nên có thể được chia tiếp ...



264

Phân trang (tt)

• Bảng phân trang băm:

- Một phương pháp phổ biến để xử lý không gian địa chỉ lớn hơn 32 bit là sử dụng bảng trang được băm, với giá trị băm là số trang ảo.
- Để giải quyết dung độ, mỗi mục của bảng băm là một danh sách liên kết. Mỗi phần tử của danh sách gồm các trường:
 - Chỉ số trang ảo
 - Chỉ số frame
 - Con trỏ đến phần tử kế tiếp
- Chỉ số trang ảo (virtual page number) được biến đổi qua hàm băm thành một hashed value.
- Phần tử tương ứng sẽ được lưu vào danh sách liên kết tại mục có chỉ số là hashed value

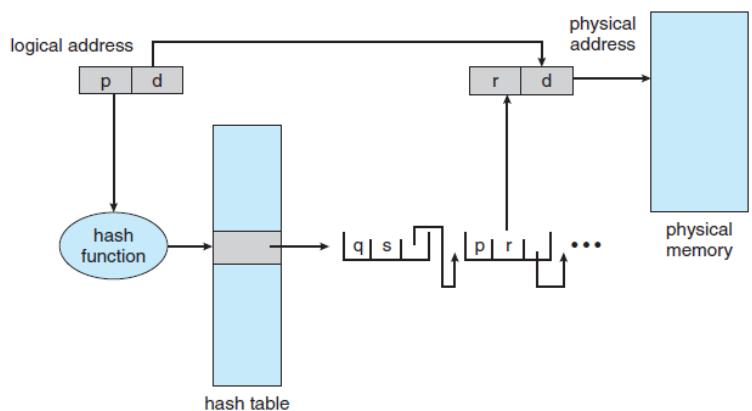
265

Phân trang (tt)

• Bảng phân trang băm:

• Giải thuật tìm trang:

- Chỉ số trang ảo được biến đổi thành hashed value (với cùng hàm băm).
- Hashed value là chỉ số của mục cần truy cập trong bảng băm.
- → Tìm trong danh sách liên kết phần tử chứa chỉ số trang ảo để lấy chỉ số frame tương ứng

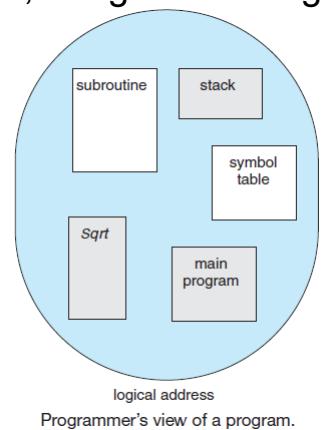


266

4.2.4 Phân đoạn (Segmentation)

• VỚI GÓC NHÌN CỦA USER:

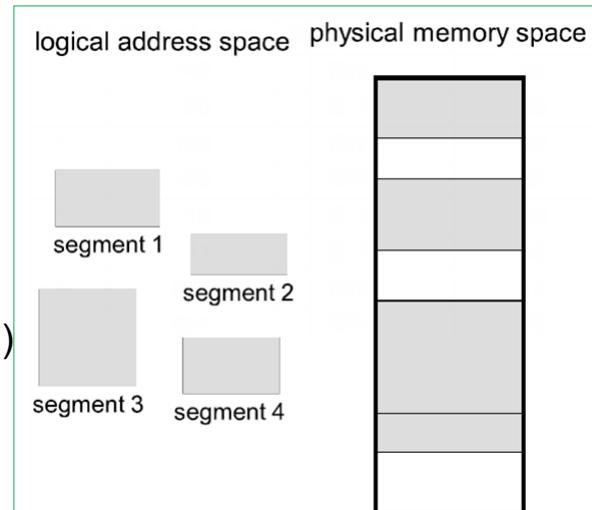
- Một chương trình người dùng có thể được chia nhỏ, trong đó chương trình và dữ liệu liên quan của nó được chia thành một số phân đoạn
- Khi chương trình được biên dịch, trình biên dịch sẽ tạo các segment
 - main, procedure, function
 - local variables, global variables, common block, stack, symbol table, array,...
- Trình loader sẽ gán mỗi segment một số định danh riêng



267

Phân đoạn (tt)

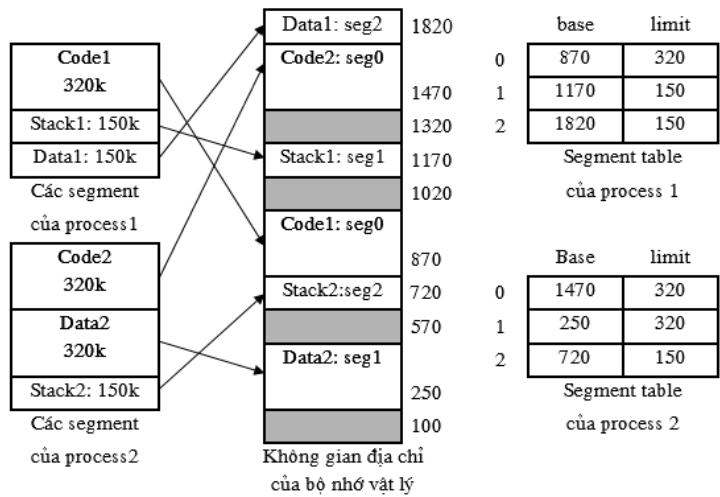
- Trong cơ chế phân đoạn, không gian địa chỉ ảo là một tập các segment
 - Là một dãy các địa chỉ liên tục từ 0
 - Mỗi phân đoạn gồm định danh, kích thước
 - Kích thước các segment có thể khác nhau, có thể thay đổi khi chương trình thực thi (stack, array,...)
- Địa chỉ luận lý bao gồm:
 - segment number
 - offset



268

Phân đoạn (tt)

- Khi một tiến trình được nạp vào bộ nhớ thì tất cả các đoạn của nó sẽ được nạp vào các phân đoạn còn trống (có thể không liên tiếp nhau) trên bộ nhớ



269

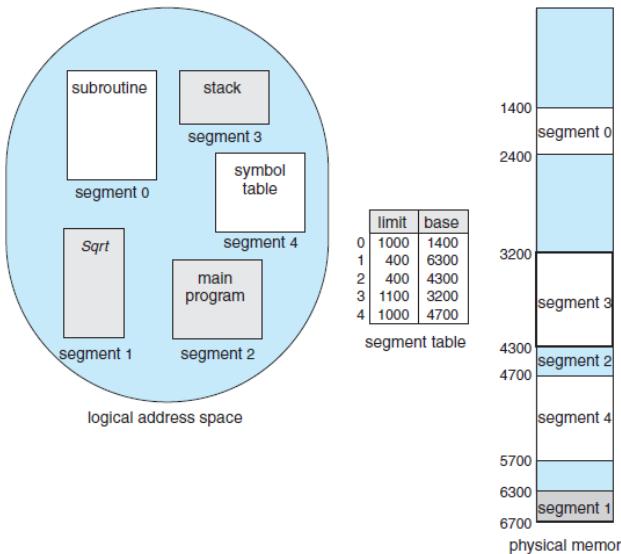
Phân đoạn (tt)

- Mỗi tiến trình có một bảng phân đoạn (SCT: Segment Control Table) riêng
- Mỗi phần tử trong bảng phân đoạn gồm tối thiểu 2 trường:
 - base**: địa chỉ khởi đầu của phân đoạn trong bộ nhớ vật lý
 - limit**: kích thước của phân đoạn, còn được dùng để kiểm soát sự truy xuất bất hợp lệ của các tiến trình

270

Phân đoạn (tt)

- Ví dụ về phân đoạn



271

Phân đoạn (tt)

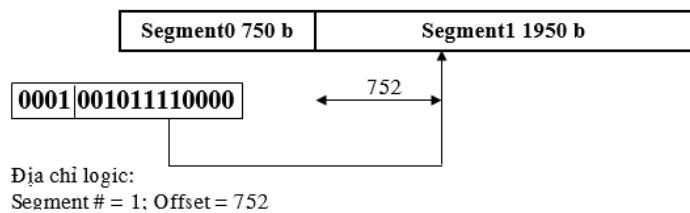
- Các bảng phân đoạn có kích thước nhỏ có thể được chứa trong các thanh ghi
- Các bảng phân đoạn có kích thước lớn được chứa trong bộ nhớ chính, HĐH dùng một thanh ghi để lưu trữ địa chỉ bắt đầu nơi lưu trữ bảng phân đoạn (**STBR: Segment Table Base Register**)
- Thanh ghi **STLR: Segment Table Length Register** được dùng để ghi kích thước hiện tại của bảng phân đoạn.
- HĐH dùng một danh sách riêng để theo dõi các segment còn trống trên bộ nhớ.

272

Phân đoạn (tt)

- Địa chỉ phát sinh từ CPU được chia thành hai phần:

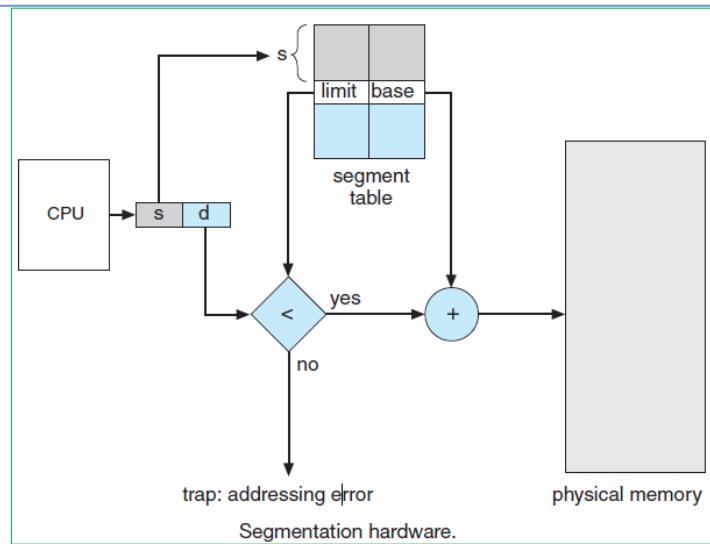
- **Segment number (s)** – Số hiệu đoạn: sử dụng như chỉ mục đến phần tử trong bảng phân đoạn, cho biết số hiệu đoạn tương ứng cần truy xuất.
- **Segment offset (o)** - Địa chỉ tương đối trong đoạn: giá trị này sẽ được kết hợp với địa chỉ bắt đầu của đoạn để xác định địa chỉ vật lý của ô nhớ cần truy xuất.



273

Phân đoạn (tt)

- Phần cứng hỗ trợ chuyển đổi địa chỉ



274

Phân đoạn (tt)

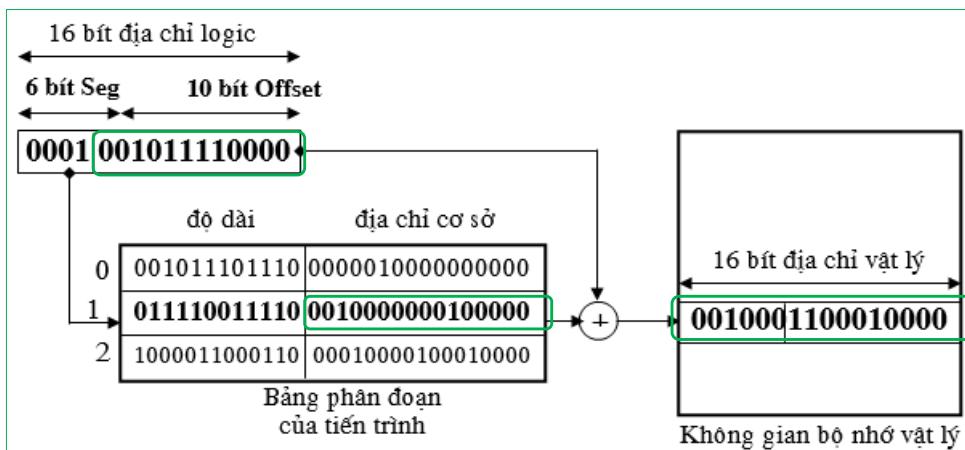
• Các bước chuyển đổi địa chỉ

- n bít trái nhất của địa chỉ logic xác định chỉ số của phân đoạn cần truy xuất trong bảng phân đoạn
- Truy xuất phần tử tương ứng thông qua chỉ số trên để tìm địa chỉ vật lý bắt đầu của phân đoạn.
- So sánh thành phần offset của địa chỉ logic (m bít phải của địa chỉ logic) với trường **length** của phân đoạn.
 - Nếu offset > length: địa chỉ truy xuất là không hợp lệ.
 - Ngược lại: địa chỉ vật lý cần tìm = địa chỉ vật lý bắt đầu của phân đoạn + offset.
- Giả sử có một địa chỉ logic gồm $n + m$ bít: $n = 6$, $m = 10 \rightarrow$ kích thước tối đa của một segment là 2^{10} bytes

275

Phân đoạn (tt)

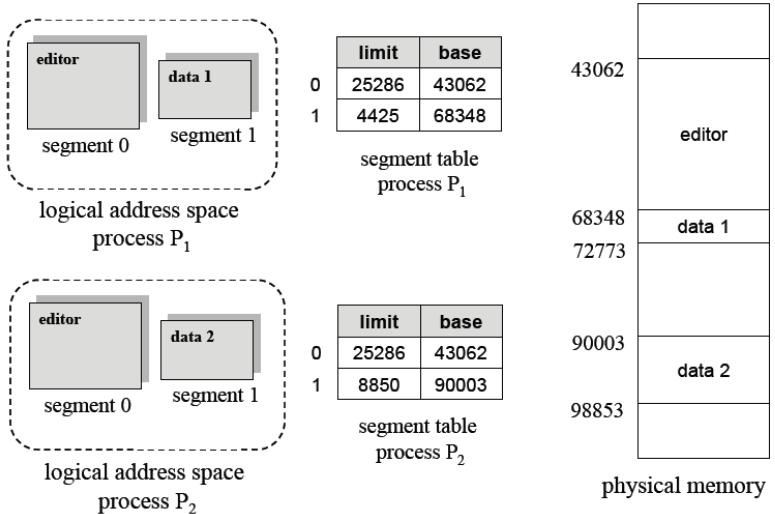
• Các bước chuyển đổi địa chỉ (tt)



276

Phân đoạn (tt)

• Chia sẻ các đoạn



277

Phân đoạn (tt)

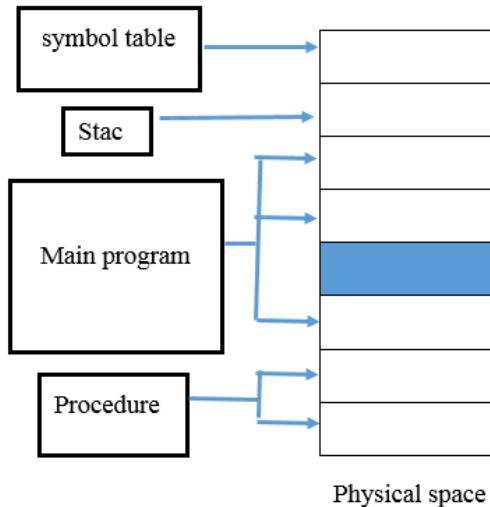
• Nhận xét

- Các segment có kích thước không bằng nhau → tương tự phân vùng động.
- Một chương trình có thể chiếm giữ hơn một phân vùng có thể không liền kề với nhau.
- Loại trừ được sự phân mảnh nội, vẫn còn phân mảnh ngoại vi (như phân vùng động), tuy nhiên do chương trình được chia nhỏ nên phân mảnh ngoại vi không đáng kể.
- Tường minh đối với người lập trình, cho phép người lập trình tổ chức chương trình và dữ liệu (có thể gán các chương trình và dữ liệu đến các đoạn nhớ khác nhau)
- Cần giải quyết vấn đề cấp phát động: best-fit, first-fit
- Segment có kích thước quá lớn có thể không nạp được vào bộ nhớ → Kết hợp phân trang với phân đoạn

278

Phân đoạn (tt)

- Kết hợp phân trang và phân đoạn



279

4.3. Mô hình quản lý bộ nhớ ảo

- Đặc điểm
- Kỹ thuật phân trang theo yêu cầu

280

4.3.1 Đặc điểm mô hình quản lý bộ nhớ ảo

- Nhìn lại paging và segmentation :
 - Các tham chiếu đến bộ nhớ được chuyển đổi động thành địa chỉ thực lúc process đang thực thi
 - Một process gồm các phần nhỏ (page hay segment), các phần này được nạp vào các vùng có thể không liên tục trong bộ nhớ chính
- Nhận xét:
 - không phải tất cả các phần của một process cần thiết phải được nạp vào bộ nhớ chính tại cùng một thời điểm:
 - Đoạn mã điều khiển các lỗi hiếm khi xảy ra
 - Các arrays, list, tables được cấp phát bộ nhớ (cấp phát tĩnh) nhiều hơn yêu cầu thực sự
 - Một số tính năng ít khi được dùng của một chương trình
 - Ngay cả khi toàn bộ chương trình đều cần dùng thì có thể không cần dùng toàn bộ cùng một lúc

281

Đặc điểm mô hình quản lý bộ nhớ ảo (tt)

- Bộ nhớ ảo (virtual memory)
 - Cơ chế được hiện thực trong hệ điều hành để cho phép thực thi một tiến trình mà chỉ cần giữ trong bộ nhớ chính một phần của không gian địa chỉ luận lý của nó, còn phần còn lại được giữ trên bộ nhớ phụ (đĩa).
 - Cần kết hợp kỹ thuật swapping để chuyển các phần của tiến trình từ bộ nhớ chính ra bộ nhớ phụ và ngược lại khi cần thiết.
- Ưu điểm của bộ nhớ ảo
 - Tăng số lượng tiến trình đồng thời được nạp vào bộ nhớ
 - Một tiến trình có kích thước lớn hơn bộ nhớ thực vẫn có thể thực thi
 - Bộ nhớ ảo cho phép giảm nhẹ công việc của lập trình viên vì không cần bận tâm đến giới hạn của vùng nhớ vật lý.

282

Đặc điểm mô hình quản lý bộ nhớ ảo (tt)

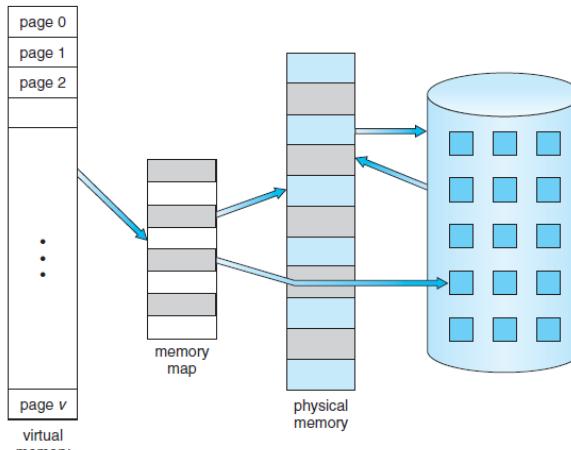
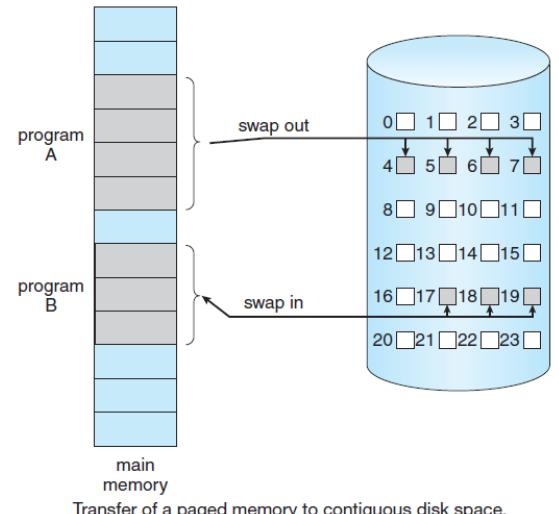


Diagram showing virtual memory that is larger than physical memory.

283

4.3.3 Phân trang theo yêu cầu (Demand paging)

- 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 (đĩa).
- Khi cần xử lý, tiến trình sẽ được nạp vào bộ nhớ chính, nhưng chỉ nạp những trang cần thiết trong thời điểm hiện tại → một trang chỉ được nạp vào bộ nhớ chính khi có yêu cầu.



Transfer of a paged memory to contiguous disk space.

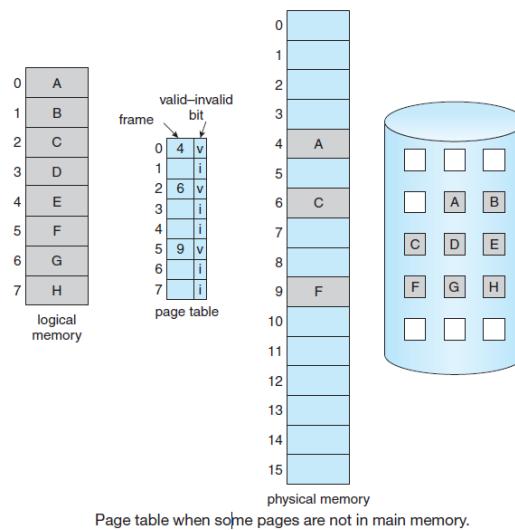
284

Phân trang theo yêu cầu (tt)

- Cần cung cấp một cơ chế phần cứng giúp phân biệt các trang đang ở trong bộ nhớ chính và các trang trên đĩa → sử dụng valid bit:
 - 1 (valid): trang hợp lệ và đang ở trong bộ nhớ chính.
 - 0 (invalid):
 - Trang bất hợp lệ (không thuộc về không gian địa chỉ của tiến trình)
 - hoặc trang hợp lệ nhưng không nằm trong bộ nhớ chính (đang được lưu trên bộ nhớ phụ)
- Khi trang không nằm trong bộ nhớ chính, phần tử tương ứng trong bảng trang sẽ được đánh dấu *invalid* và chứa địa chỉ của trang trên bộ nhớ phụ.

285

Phân trang theo yêu cầu (tt)



286

Phân trang theo yêu cầu (tt)

- **Cơ chế phần cứng:** kết hợp của cơ chế hỗ trợ kỹ thuật phân trang và kỹ thuật swapping
 - Bảng trang: phản ánh tình trạng của một trang là đang nằm trong bộ nhớ chính hay bộ nhớ phụ.
 - Bộ nhớ phụ:
 - Thường là đĩa, lưu trữ những trang không được nạp vào bộ nhớ chính.
 - Vùng không gian đĩa dùng để lưu trữ tạm các trang trong kỹ thuật swapping được gọi là không gian swapping.

287

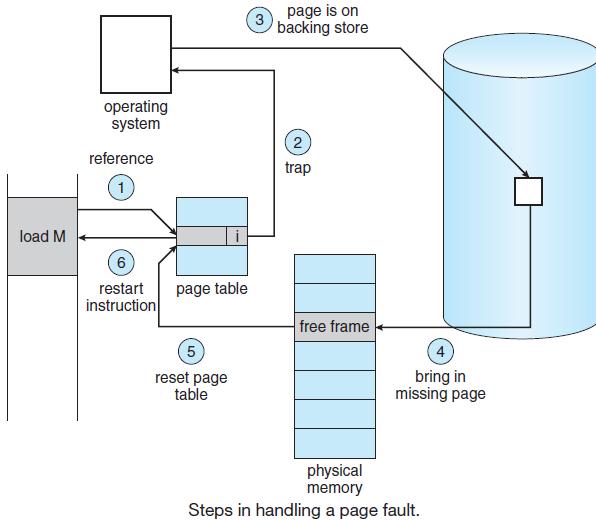
Phân trang theo yêu cầu (tt)

- **Lỗi trang:** Khi có một tham chiếu đến một trang không có trong bộ nhớ chính (present bit = 0) → phần cứng phát sinh một ngắt (page-fault trap) báo cho HĐH xử lý:
 - Kiểm tra truy xuất đến bộ nhớ là hợp lệ hay không hợp lệ
 - Nếu truy xuất không hợp lệ, kết thúc tiến trình.
 - Ngược lại:
 - Tìm vị trí chứa trang muốn truy xuất trên đĩa
 - Tìm một khung trang trống trong bộ nhớ chính → chuyển trang muốn truy xuất từ bộ nhớ phụ vào bộ nhớ chính bằng cách nạp trang cần truy xuất vào khung trang trống đã chọn, cập nhật nội dung bảng trang, bảng khung trang tương ứng
 - Nếu không còn khung trang trống → thay trang

288

Phân trang theo yêu cầu (tt)

- **Lỗi trang (tt)**



289

Phân trang theo yêu cầu (tt)

- **Copy-on-Write:**

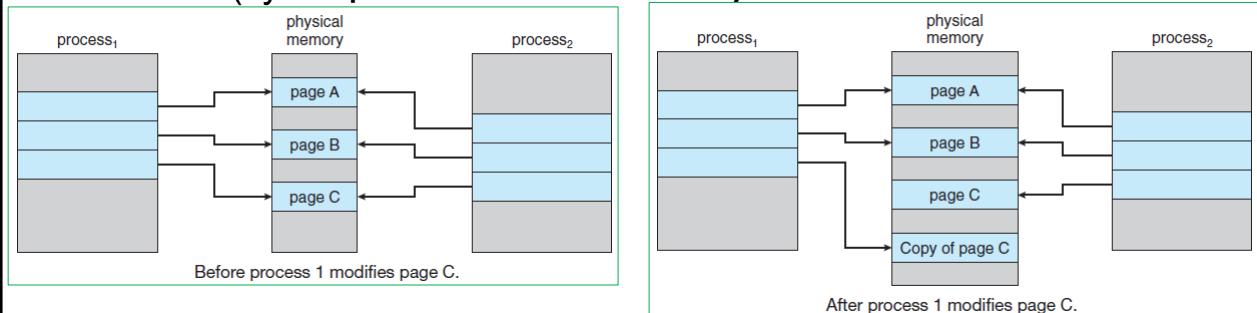
- Là một kỹ thuật phổ biến được sử dụng trong nhiều hệ điều hành (Windows XP, Linux, và Solaris)
- Cho phép tiến trình cha và con cùng chia sẻ các trang trong bộ nhớ khi mới khởi tạo tiến trình con.
- Các trang được chia sẻ này được đánh dấu là các trang **copy-on-write**, nếu một trong hai tiến trình muốn ghi vào một trang được chia sẻ, một bản sao của trang đó sẽ được tạo.
- Chỉ những trang có thể sửa đổi mới cần được đánh dấu là copy-on-write
- Tạo tiến trình hiệu quả hơn: chỉ các trang bị sửa đổi mới được copy

290

Phân trang theo yêu cầu (tt)

•Copy-on-Write (tt):

- Khi tạo bảng sao cho một trang trong copy-on-write, HĐH cần cấp phát các trang rỗi để lưu các trang copy này.
- Các trang rỗi được cấp phát từ một tập hợp các trang được xóa trắng trước đó (kỹ thuật **zero-fill-on-demand**)

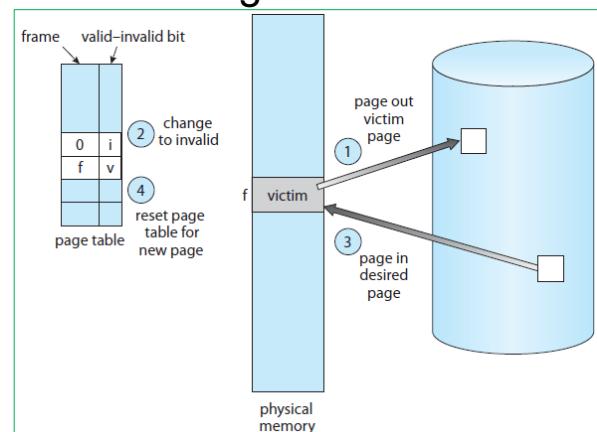


291

Phân trang theo yêu cầu (tt)

•Thay thế trang: khi đưa một trang từ bộ nhớ ngoài vào bộ nhớ chính, trường hợp không tìm được frame trống:

- Chuyển một trang trong bộ nhớ chính ra bộ nhớ ngoài và nạp một trang khác vào bộ nhớ chính.
- Dùng một giải thuật thay trang để chọn một *trang hy sinh* (*victim page*)
- Ghi *victim page* lên bộ nhớ ngoài
- Đọc trang đang cần vào frame trống (thay vào chỗ *victim page*)



292

Phân trang theo yêu cầu (tt)

• Thay thế trang (tt):

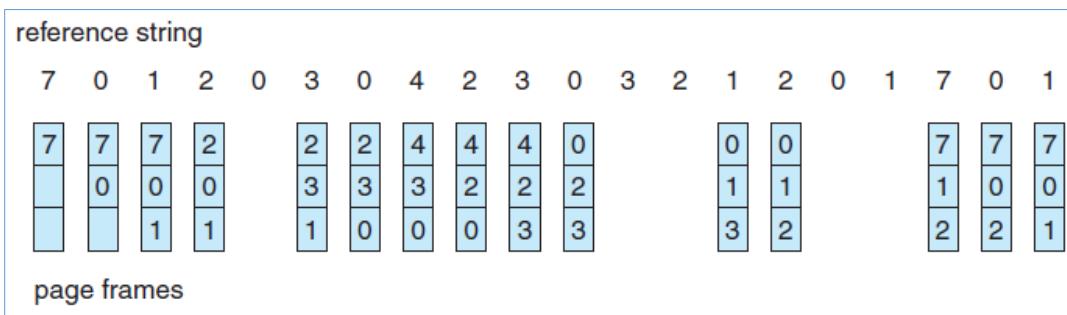
- Trong mỗi phần tử của bảng phân trang, sử dụng một bit *cập nhật* (dirty bit) để phản ánh tình trạng trang có bị sửa đổi hay không
 - dirty bit = 1: nội dung trang đã bị sửa đổi
 - Khi cần thay thế một trang, nếu dirty bit = 1 \rightarrow trang cần được lưu lại trên đĩa, ngược lại (trang không bị thay đổi) \rightarrow không cần lưu trữ trang trả lại đĩa.
- Các thuật toán thay thế trang cần phải đạt mục tiêu là chọn victim page sao cho sau khi thay thế sẽ gây ra ít lỗi trang nhất
- Thuật toán thay thế trang được đánh giá hiệu quả bằng cách xử lý trên một chuỗi các địa chỉ cần truy xuất và tính toán số lượng lỗi trang phát sinh

293

Phân trang theo yêu cầu (tt)

• Giải thuật thay trang FIFO

- Chọn trang ở trong bộ nhớ lâu nhất
- Ví dụ : sử dụng 3 khung trang, ban đầu cả 3 đều trống :



294

Phân trang theo yêu cầu (tt)

• Giải thuật thay trang FIFO (tt)

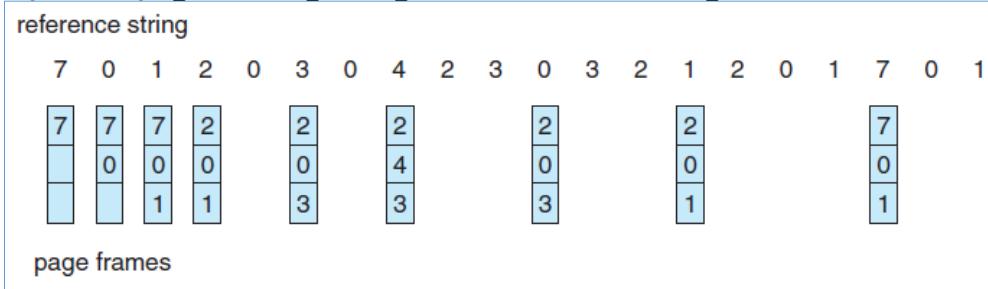
- Dễ hiểu, dễ cài đặt
- Có thể cho kết quả không tốt: trang được chọn để thay thế có thể là trang chứa nhiều dữ liệu cần thiết, thường xuyên được sử dụng nên được nạp sớm, do vậy khi bị chuyển ra bộ nhớ phụ sẽ nhanh chóng gây ra lỗi trang.
 - Số lượng lỗi trang xảy ra sẽ tăng lên khi số lượng khung trang nhiều → nghịch lý Belady
 - ví dụ chuỗi tham chiếu 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
 - 3 khung trang: 9 lỗi
 - 4 khung trang: 10 lỗi

295

Phân trang theo yêu cầu (tt)

• Giải thuật thay trang OPT(optimal)

- Thay thế trang sẽ được tham chiếu trễ nhất trong tương lai
- Ví dụ: sử dụng 3 khung trang, khởi đầu đều trống

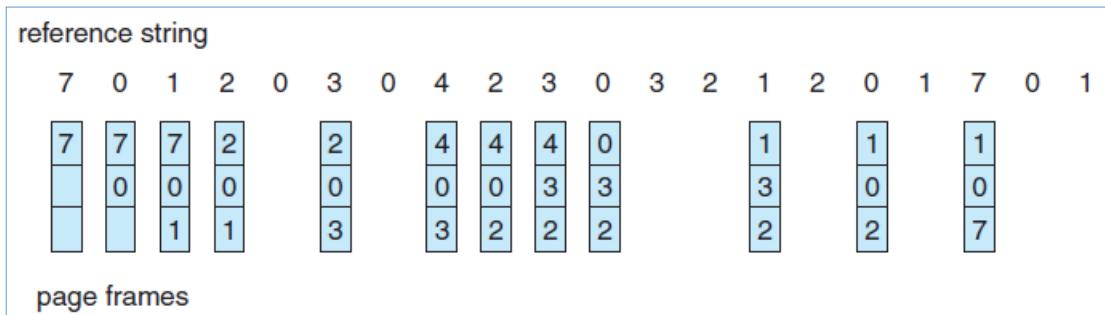


- Số lượng lỗi trang thấp nhất, không xảy ra nghịch lý Belady,
- Không khả thi, vì không thể biết trước chuỗi truy xuất của tiến trình!

296

Phân trang theo yêu cầu (tt)

- Giải thuật thay trang *Least Recently Used* (LRU)
 - Thay thế trang nhớ không được tham chiếu lâu nhất
 - Ví dụ: sử dụng 3 khung trang, khởi đầu đều trống



297

Phân trang theo yêu cầu (tt)

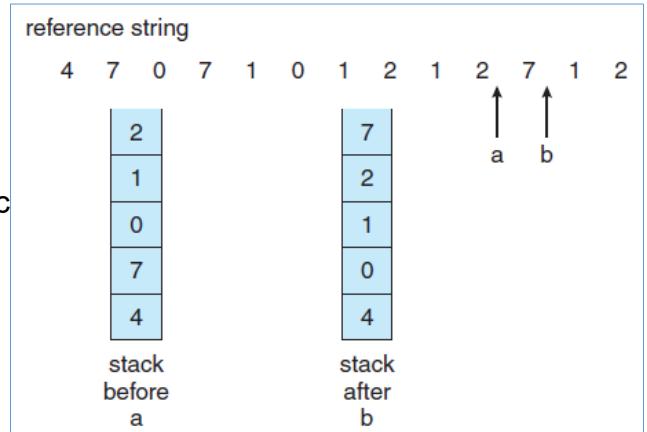
- LRU (tt)
 - Cần được cơ chế phần cứng hỗ trợ để xác định thứ tự cho các trang theo thời điểm truy xuất cuối cùng. Có hai cách:
 - Sử dụng bộ đếm:
 - Mỗi phần tử trong bảng trang được bổ sung một trường ghi nhận thời điểm truy xuất mới nhất
 - Bổ sung vào cấu trúc của CPU một bộ đếm mỗi lần có sự truy xuất bộ nhớ, giá trị của counter tăng lên 1.
 - Mỗi lần thực hiện truy xuất đến một trang, ghi giá trị của counter vào một trường lưu thời điểm truy xuất mới nhất của phần tử tương ứng với trang trong bảng trang
 - Thay thế trang có giá trị trường lưu thời điểm truy xuất mới nhất là nhỏ nhất

298

Phân trang theo yêu cầu (tt)

• LRU (tt)

- 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.



299

Chương 5

Quản lý hệ thống File



Nội dung

-
1. File và thư mục
 2. Hiện thực Hệ thống file
-

Tài liệu tham khảo

-
- Andrew S. Tanenbaum, Modern Operating Systems, Pearson, 2015
 - Chương 4
 - Abraham Silberschatz, Peter Baer Galvin, Greg Gagne, Operating System Concepts, John Wiley, 2013
 - Chương 11, 12
-

5.1. File và thư mục

- Ý nghĩa
- File và các khái niệm liên quan
- Các thao tác với file
- Thư mục

5.1.1 Ý nghĩa của hệ thống file

- Yêu cầu lưu trữ của máy tính:
 - Phải có khả năng lưu trữ một lượng thông tin rất lớn.
 - Thông tin phải tồn tại lâu dài.
 - Nhiều tiến trình phải có thể truy cập thông tin cùng một lúc.
- Bộ nhớ trong:
 - Có kích thước giới hạn
 - Dữ liệu mất khi mất nguồn điện
 - Giá thành cao
- Bộ nhớ ngoài: khả năng lưu trữ lớn, tồn tại độc lập, dễ dàng sao chép, di chuyển, giá thành rẻ.
- Hệ thống file: tổ chức bộ nhớ ngoài để có thể lưu trữ khối lượng lớn dữ liệu, chương trình, khi thực thi → nạp vào bộ nhớ trong

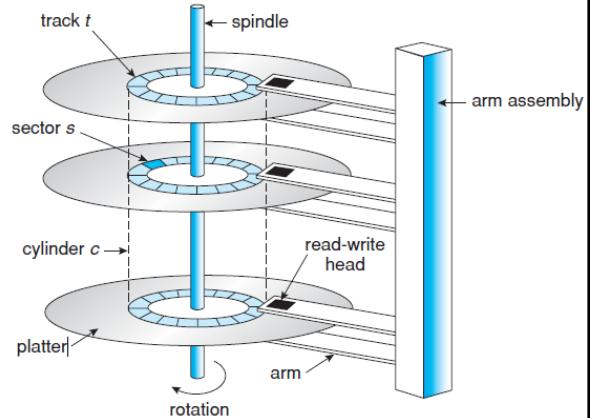
5.1.2. File và các khái niệm liên quan

• **Bộ nhớ ngoài:**

- Là các thiết bị lưu trữ bên ngoài có khả năng lưu trữ trong thời gian dài
- Chứa lượng thông tin rất lớn

• **Đĩa cứng:**

thiết bị lưu trữ thuộc bộ nhớ ngoài



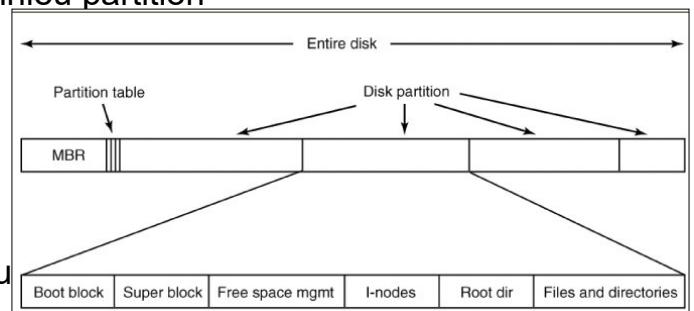
File và các khái niệm liên quan (tt)

• **Partition (phân vùng):**

- Là một tập các sector liền kề trên một đĩa, còn gọi là đĩa luận lý
- Mỗi đĩa vật lý có thể chia thành nhiều partition

• **Volume:**

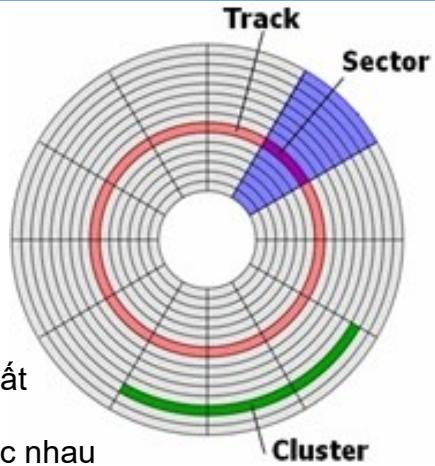
- Một volume tương tự một partition logic trên một đĩa
- Được tạo khi ta định dạng một đĩa hoặc một phần của đĩa theo hệ thống file NTFS.
- Một đĩa có thể có một hoặc nhiều volume độc lập với nhau
- Có thể tạo ra một volume trên nhiều đĩa vật lý khác nhau (Windows NT/2000)



File và các khái niệm liên quan (tt)

• Track:

- Đĩa cứng được định dạng thành các vòng tròn đồng tâm gọi là track để ghi dữ liệu
- Track có thể thay đổi vị trí khi định dạng cấp thấp ổ đĩa (low format)



• Sector:

- Mỗi rãnh chia thành nhiều phần bằng nhau gọi là sector
- Thông thường thì một sector chứa dung lượng 512 byte

• Cluster (khối):

- Gồm một hoặc nhiều sector → tăng hiệu quả truy xuất

• Cylinder (tử trụ)

- Tập hợp các track cùng bán kính ở các mặt đĩa khác nhau
- Cylinder chỉ có trên các ổ đĩa cứng

File và các khái niệm liên quan (tt)

• File (tập tin):

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

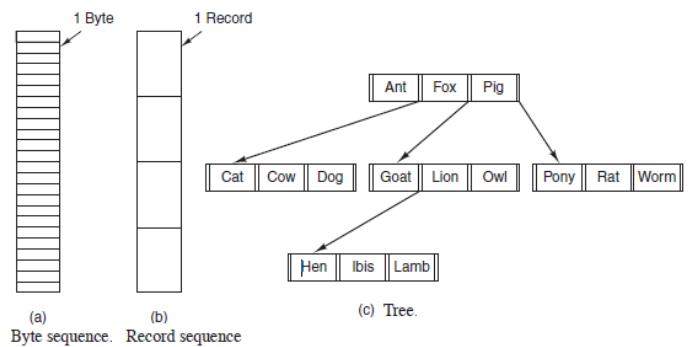
• Hệ thống quản lý file

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

File và các khái niệm liên quan (tt)

• Cấu trúc của file: gồm 3 loại:

1. Dãy tuần tự các byte không cấu trúc: hệ điều hành không biết nội dung của file: MS-DOS, UNIX, Windows
2. Dãy các record có chiều dài cố định, không còn sử dụng trong các HĐH hiện đại
3. 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, được sử dụng trên một số máy tính lớn để xử lý dữ liệu thương mại



File và các khái niệm liên quan (tt)

• Kiểu file:

- Các hệ điều hành hỗ trợ nhiều loại file khác nhau bao gồm các kiểu như:
 - 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, mạng.
 - File khống: dùng để truy xuất trên thiết bị đĩa.
 - File thường: được chia làm hai loại là file văn bản và file nhị phân.

File và các khái niệm liên quan (tt)

• File văn bản:

- Chứa các dòng văn bản, độ dài có thể khác nhau, cuối dòng có ký hiệu enter.
- Ưu điểm: có thể hiển thị, in hay soạn thảo với một editor thông thường.
- Được dùng để nhập xuất, chứa dữ liệu làm đầu vào và đầu ra cho cơ chế pipeline.

• File nhị phân:

- Gồm dãy các byte, hệ điều hành chỉ thực thi file này nếu nó có cấu trúc đúng.
- Ví dụ: một file nhị phân thi hành được của UNIX: thường có năm thành phần: header, text, data, relocation bits, symbol table.
 - Header: nhận diện tập tin.
 - 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 text và data. Nó được nạp vào bộ nhớ và định vị lại bởi những bit relocation.
 - Bảng symbol được dùng để debug

5.1.3 Thuộc tính của file

- **Tên** (name): là thông tin được lưu ở dạng mà người dùng có thể đọc
- **Định danh** (identifier): là thẻ duy nhất, thường là số, dùng để xác định file trong hệ thống file, người dùng không thể đọc
- **Kiểu** (type): này được yêu cầu cho hệ thống hỗ trợ các kiểu khác nhau
- **Vị trí** (location): con trỏ chỉ tới một thiết bị tới vị trí file trên thiết bị đó.
- **Kích thước** (size): kích thước hiện hành của file (tính bằng byte, word hay khối) và kích thước cho phép tối đa chứa trong thuộc tính này.
- **Bảo vệ** (protection): mức độ và quyền truy cập
- **Ngày** (date), **Giờ** (time), và **định danh người dùng** (user identification): thông tin về việc tạo, sửa đổi gần nhất, sử dụng gần nhất → có ích cho việc bảo vệ, bảo mật, và kiểm soát việc sử dụng

5.1.4 Các thao tác với file

- Các thao tác cơ bản:
 - Tạo file
 - Ghi file
 - Đọc file
 - Tái định vị trong file
 - Xóa file
 - Xóa nội dung file
- Truy xuất file
- Bảo toàn dữ liệu file
- Danh sách các quyền truy cập (Access Right)

5.1.4.1 Các thao tác cơ bản với file

• Tạo file:

- Tìm một khối trống cấp cho file
- Tạo một cấu trúc chứa thông tin của file ghi vào bảng thư mục

• Ghi file:

- Tìm tên file trong bảng thư mục
- Lấy được số hiệu khối nhớ đầu tiên cấp phát cho file, ghi dữ liệu vào file bắt đầu từ vị trí này
- Dùng một con trỏ ghi để ghi nhận vị trí cho lần ghi kế tiếp

• Đọc file:

- Tìm tên file trong bảng thư mục
- Biết vị trí lưu trữ file
- Đọc file vào bộ nhớ
- Dùng một con trỏ đọc để ghi nhận vị trí cho lần đọc kế tiếp

Các thao tác cơ bản với file (tt)

• Tái định vị trong file:

- Tìm tên file trong bảng thư mục
- Dùng lệnh seek để thay đổi vị trí con trỏ đọc

• Xóa file:

- Tìm tên file trong bảng thư mục
- Giải phóng tất cả các khối đĩa mà file chiếm dụng
- Xóa mục tương ứng chứa thông tin của file trong bảng thư mục

• Xóa nội dung file:

- Tìm tên file trong bảng thư mục
- Thiết lập thuộc tính kích thước file = 0
- Giải phóng vùng nhớ chứa nội dung file

Các thao tác cơ bản với file (tt)

• Các thao tác trên file thông thường phải tìm file trên bảng thư mục → chậm → HĐH sử dụng bảng open-file để chứa thông tin các file đã mở

- Khi cần thao tác với file, HĐH sẽ tìm file trong bảng open-file
- Khi đóng file, phần tử tương ứng được xóa trong bảng open-file

• Mở file:

- Lần đầu: thông tin file được đọc từ bảng thư mục và lưu vào bảng open-file trong bộ nhớ
- Các lần sau (khi file chưa đóng): lấy thông tin từ bảng open-file.
- Thủ tục do HĐH cung cấp: **Open** (tên file cần mở, chế độ mở: đọc/ ghi/ tạo mới)

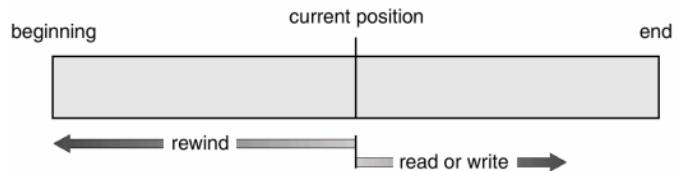
• Đóng file:

- Ghi mục tương ứng trong bảng open-file vào cấu trúc thư mục và hủy mục này trong bảng open-files
- Thủ tục do HĐH cung cấp: **Close** (tên file cần đóng)

5.1.4.2 Các phương pháp truy xuất file

• Truy xuất tuần tự:

- Thông tin trong file được xử lý có thứ tự
- Tiến trình đọc/ghi tất cả các byte trong file theo thứ tự từ đầu (các trình soạn thảo, trình biên dịch) → tự động tăng con trỏ file đến vị trí tiếp theo.
- File có thể tự khởi động lại từ vị trí đầu tiên
- Một số hệ thống file cho phép di chuyển con trỏ file đi tới/ lui n mẩu tin.
- Phù hợp với các loại băng từ
- Có hai cách truy xuất:
 - Đọc bắt đầu ở vị trí đầu file
 - SEEK đến một vị trí trong file



Các phương pháp truy xuất file (tt)

• Truy xuất ngẫu nhiên

- Truy xuất không theo thứ tự
- Sử dụng cho loại file được hình thành từ các mẩu tin luận lý có chiều dài không đổi
- Dựa trên mô hình của đĩa, vì đĩa cho phép truy xuất ngẫu nhiên tới bất cứ khối nào tập tin.
- File được hiển thị như một chuỗi các khối hay mẩu tin được đánh số, có thể đọc hoặc ghi các khối bất kỳ.
- Được sử dụng trong trường hợp phải truy xuất một khối lượng thông tin lớn (ví dụ: cơ sở dữ liệu).
- Được sử dụng trong UNIX và Windows

Các phương pháp truy xuất file (tt)

• Các phương pháp truy xuất khác

- Được xây dựng trên cơ sở của phương pháp truy xuất ngẫu nhiên
- Tạo chỉ mục cho file, chỉ mục chứa các con trỏ chỉ tới các khối khác.
- Để tìm một mẫu tin trong file, trước hết phải tìm chỉ mục → dùng con trỏ để truy xuất file → tìm mẫu tin.
- Với những file lớn → chỉ mục file quá lớn → tạo chỉ mục cho file chỉ mục:
 - File chỉ mục chính chứa các con trỏ chỉ tới các file chỉ mục thứ cấp chỉ tới các thành phần dữ liệu thật sự.

5.1.4.3 Bảo toàn dữ liệu file

- HĐH phải luôn tạo bản sao cho các file đang mở trên hệ thống, để có thể phục hồi lại khi cần thiết, có hai kỹ thuật là **DUMP có chu kỳ** và **DUMP Incremental**
- **DUMP theo chu kỳ:**
 - Sau một khoảng thời gian, nội dung của các file đang mở trên bộ nhớ chính sẽ được đổ (Dump/backup) ra lại đĩa.
 - Nếu hệ thống gặp sự cố thì tất cả các file đang mở sẽ được tái tạo lại kể từ trạng thái mà chúng được DUMP ra lần cuối cùng.
 - Dump làm tổn thời gian thực hiện của hệ thống.

Bảo toàn dữ liệu file (tt)

• DUMP Incremental:

- Chỉ lưu các thông tin được sửa đổi kể từ lần Dump sau cùng
- Thông tin cần lưu trữ ít hơn → hệ thống có thể thực hiện Dump thường xuyên hơn
- Sử dụng một trường KTCN dài 2 bit, chứa một trong các giá trị sau:
 - 00: mở không cập nhật, 01: mở có cập nhật, 10: không thay đổi so với lần Dump trước, 11: có thay đổi so với lần Dump trước.
- Cần thường xuyên kiểm tra bảng danh mục và cập nhật lại trường KTCN sau mỗi lần Dump → chậm tốc độ thực hiện của hệ thống.
- → cài đặt thêm một bảng danh mục để ghi nhận thông tin của các file đang được truy xuất (ghi/đọc) trên hệ thống chỉ dành cho Dump sử dụng → hệ thống Dump có thể hoạt động song song với các thao tác khác của hệ thống.
- Dump Incremental là một tiến trình có độ ưu tiên thấp, thường trú trong bộ nhớ, phân tích các bảng danh mục để tìm ra các file cần phải thực hiện Dump.

5.1.4.4 Danh sách các quyền truy cập (Access Right)

- Trong các hệ thống đa người dùng, quyền truy cập có thể được gán cho **User**, **User Group**, **All User**.
- Khi một group được gán quyền nào đó thì tất cả các user trong group đều được gán quyền truy cập đó.

-rw-rw-r--	1	pbg	staff	31200	Sep 3 08:30	intro.ps
drwx-----	5	pbg	staff	512	Jul 8 09:33	private/
drwxrwxr-x	2	pbg	staff	512	Jul 8 09:35	doc/
drwxrwx---	2	jwg	student	512	Aug 3 14:13	student-proj/
-rw-r--r--	1	pbg	staff	9423	Feb 24 2012	program.c
-rwxr-xr-x	1	pbg	staff	20471	Feb 24 2012	program
drwx--x--x	4	tag	faculty	512	Jul 31 10:31	lib/
drwx-----	3	pbg	staff	1024	Aug 29 06:52	mail/
drwxrwxrwx	3	pbg	staff	512	Jul 8 09:35	test/

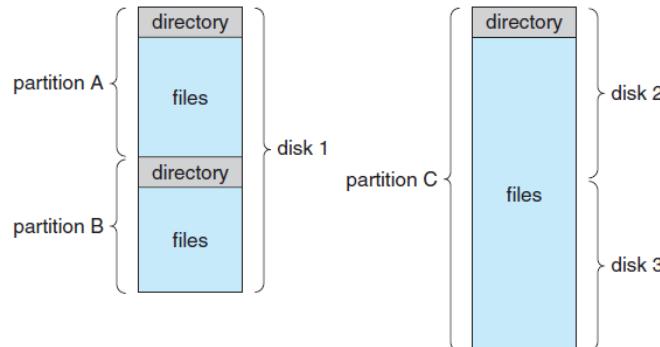
Danh sách các quyền truy cập (tt)

• Các quyền truy cập của hệ điều hành:

- **None**: không được phép truy cập thư mục chứa file.
- **Knowledge**: có thể xác định được là file đang tồn tại và ai là người sở hữu file → có thể yêu cầu chủ sở hữu cấp thêm quyền truy cập.
- **Execution**: có thể thực thi một chương trình nhưng không thể copy nó.
- **Reading**: đọc, copy và thực thi. (Một vài hệ thống chỉ cho phép đọc nhưng không thể copy nội dung)
- **Appending**: chỉ có thể thêm dữ liệu vào file, thường là ở cuối file,
- **Updating**: có thể thay đổi, xoá và thêm dữ liệu vào file.
- **Changing protection**: có thể thay đổi các quyền truy cập được gán đến user khác (thường chỉ được gán cho user sở hữu file)
- **Deletion**: có thể xoá được file từ hệ thống file.

5.1.5. Thư mục

- Đĩa chia thành các partition, mỗi partition được xem như một thiết bị lưu trữ riêng
- Mỗi partition chứa thông tin về các file chứa bên trong (volume table of contents)



Thư mục (tt)

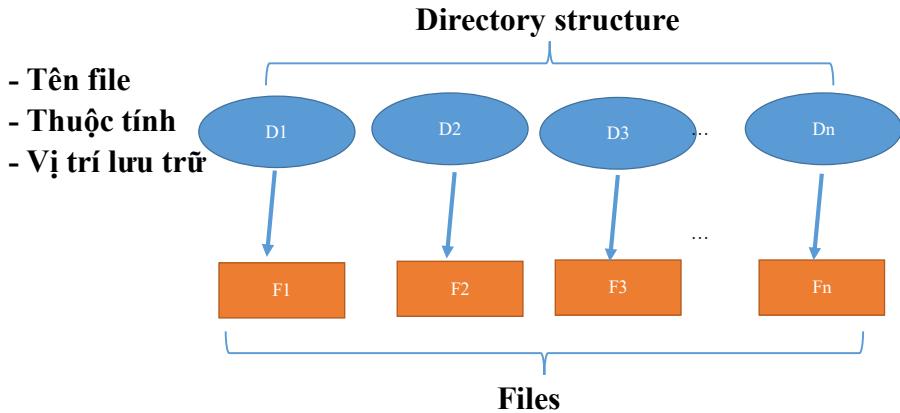
- Cấu trúc thư mục dùng để quản lý có tổ chức hệ thống tập tin, có thể đáp ứng các thao tác:
 - Tìm kiếm tập tin trong thư mục
 - Tạo một tập tin bên trong thư mục.
 - Xóa tập tin khỏi thư mục.
 - Liệt kê các tập tin trong thư mục.
 - Đổi tên một tập tin.
 - Duyệt hệ thống tập tin

Thư mục (tt)

- Cấu trúc thư mục được ghi trên đĩa là một bảng và gồm nhiều mục (directory entry), mỗi mục chứa thông tin một file (thuộc tính file, danh sách các số hiệu khối đĩa lưu trữ file)
- Khi truy xuất file, HĐH tìm file trong bảng thư mục, lấy thông tin file lưu vào bộ nhớ (bảng open-file) dành cho các lần truy xuất sau
- HĐH cung cấp các lời gọi hệ thống thao tác trên thư mục như: Create, Delete, Open, Close, Read, rename, Link, Unlink,...

Thư mục (tt)

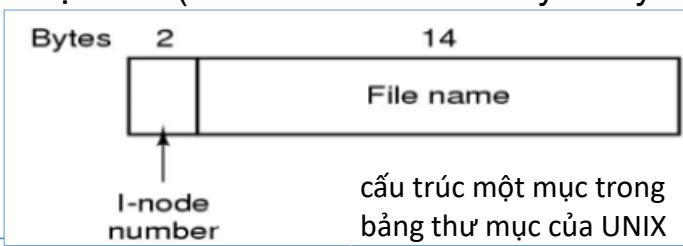
- Mô hình cấu trúc thư mục, mỗi mục (Di) quản lý một file (Fi) hoặc thư mục con



Thư mục (tt)

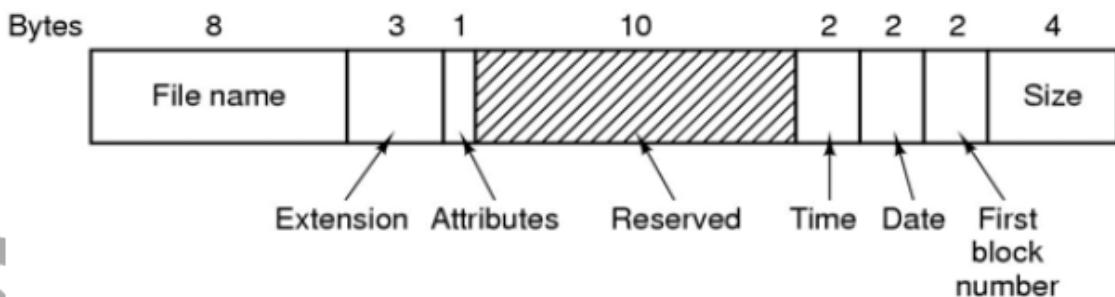
- Bảng thư mục (Directory table)**

- Mảng một chiều, mỗi phần tử lưu thông tin của một file/thư mục:
 - Thuộc tính file
 - Địa chỉ trên đĩa của file (số hiệu khối đầu tiên chứa file/số I-node của file)
- Mỗi đĩa có một bảng thư mục gọi là bảng thư mục gốc (**RDET** – Root Directory Entry Table), cài đặt ở phần đầu của đĩa và có thể có nhiều bảng thư mục con (**SDET** – Sub Directory Entry Table)



Thư mục (tt)

• Bảng thư mục (tt)

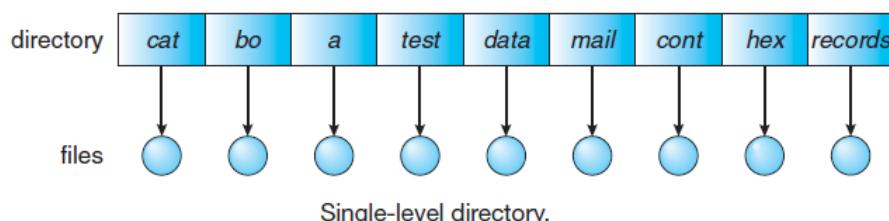


cấu trúc một mục trong bảng thư mục của MSDOS/WINDOWS (FAT)

Thư mục (tt)

• Cấu trúc thư mục dạng đơn cấp

- Tất cả file được chứa trong cùng thư mục
- Cấu trúc thư mục chứa của tất cả file của tất cả người dùng



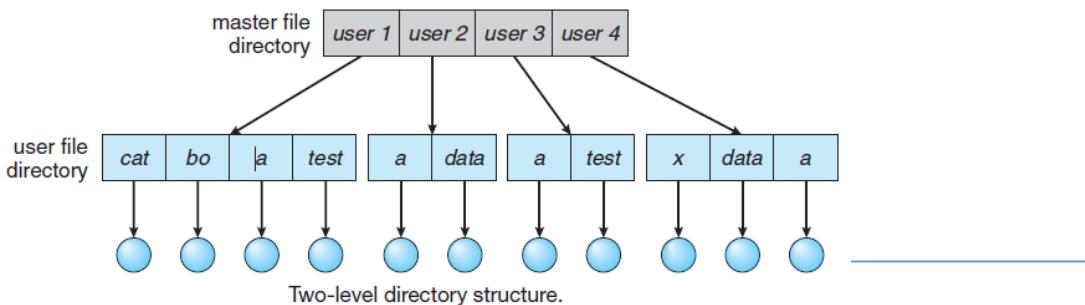
• Hạn chế:

- Thư mục chứa số lượng lớn file → khó nhớ, tìm kiếm chậm
- Nhiều người dùng → lẫn lộn tên tập tin

Thư mục (tt)

• Cấu trúc thư mục dạng hai cấp

- Mỗi người dùng có thư mục riêng (user file directory-UFD)
- Mỗi UFD có cấu trúc giống nhau nhưng chỉ quản lý file của một người dùng
- Khi người dùng đăng nhập, hệ thống MFD (master file directory) tìm kiếm load UFD dựa vào id tài khoản người dùng



Thư mục (tt)

• Cấu trúc thư mục dạng hai cấp (tt)

- Để tạo một file cho một người dùng, HĐH chỉ tìm UFD của người dùng đó để xác định một file khác cùng tên có tồn tại hay không.
- Để xóa một tập tin, HĐH chỉ tìm kiếm trong UFD cục bộ → không thể xóa nhầm file của người dùng khác có cùng tên.
- Ưu điểm:
 - Tên file của các người dùng khác nhau có thể trùng
 - Tìm kiếm nhanh
- Hạn chế: các người dùng khó có thể truy xuất các file của người dùng khác

Thư mục (tt)

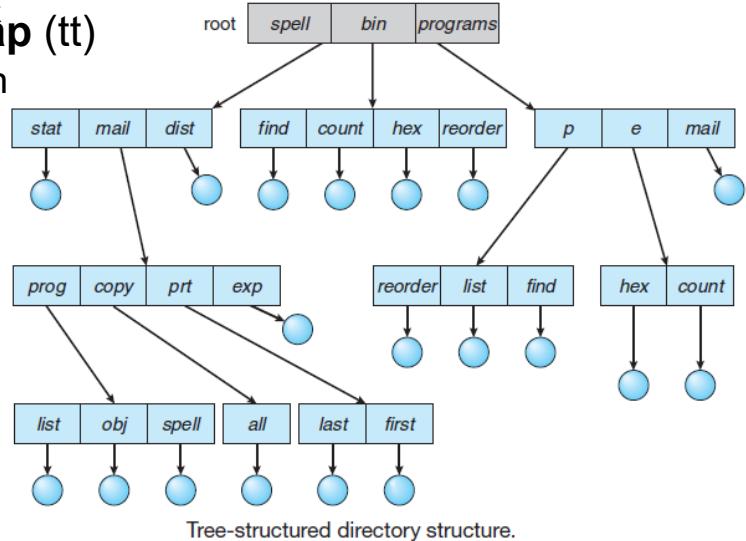
• Cấu trúc thư mục đa cấp (dạng cây)

- Là trường hợp tổng quát của thư mục hai cấp, là cấu trúc thư mục phổ biến nhất
- Cho phép người dùng tạo thư mục con và tổ chức các tập tin
- Cây có một thư mục gốc và tất cả các file trong hệ thống đều có một tên đường dẫn duy nhất
- Một thư mục (hay thư mục con) chứa tập hợp các file hay thư mục con
- Mỗi tiến trình có **thư mục hiện hành**
- Tên đường dẫn có hai dạng: **đường dẫn tuyệt đối** và **đường dẫn tương đối**
- Người dùng có thể truy xuất tới các file của người dùng khác bằng cách xác định tên đường dẫn của chúng (đường dẫn tương đối hay tuyệt đối).

Thư mục (tt)

• Cấu trúc thư mục đa cấp (tt)

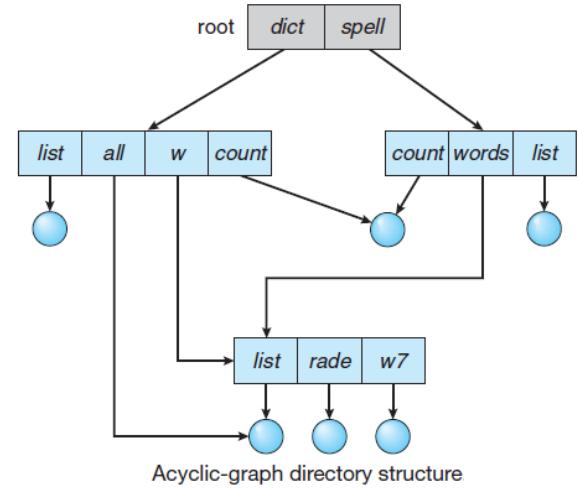
- Người dùng có thể chuyển thư mục hiện hành tới thư mục của người dùng khác và truy xuất các file bằng tên của chúng



Thư mục (tt)

• Cấu trúc Acyclic-Graph

- Cho phép chia sẻ thư mục chung giữa các user
- UFD của các user được chia sẻ sẽ chứa thư mục dùng chung
- Cách chia sẻ file/thư mục:
 - Symbolic link
 - Hard link
- Linh hoạt nhưng phức tạp: một file chia sẻ có thể có nhiều tên
- Vấn đề trong thao tác xóa.
VD: xóa file /dict/w/list



5.2. Hiện thực Hệ thống file

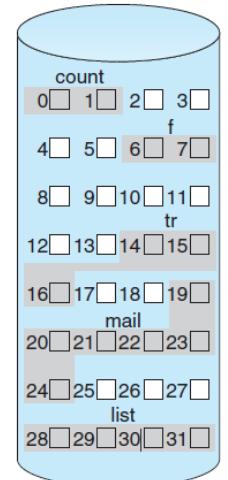
- Các phương pháp cấp phát không gian cho file
 - Cấp phát liên tục
 - Cấp phát theo danh sách liên kết
 - FAT
 - Cấp phát dùng chỉ số index
 - I-nodes
 - NTFS
- Quản lý các khối trống
- Hệ thống file trong MS-DOS

5.2.1 Cấp phát liên tục

- Lưu trữ file trên dãy các khối liên tiếp.

- Mỗi phần tử trong bảng thư mục:

- Tên file
- Vị trí khối (cluster) đầu tiên.
- Kích thước (số khối)
- Ưu điểm: Đơn giản, dễ cài đặt, dễ thao tác vì toàn bộ file được đọc từ đĩa bằng thao tác đơn giản không cần định vị lại.



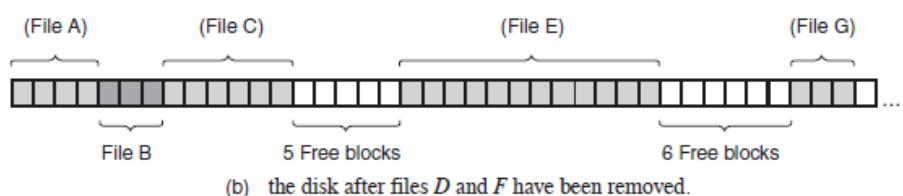
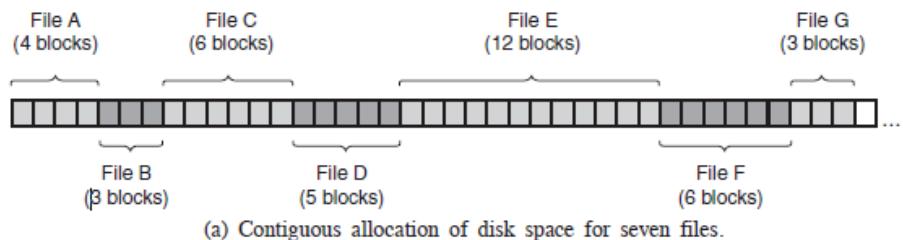
directory		
file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

Contiguous allocation of disk space.

Cấp phát liên tục (tt)

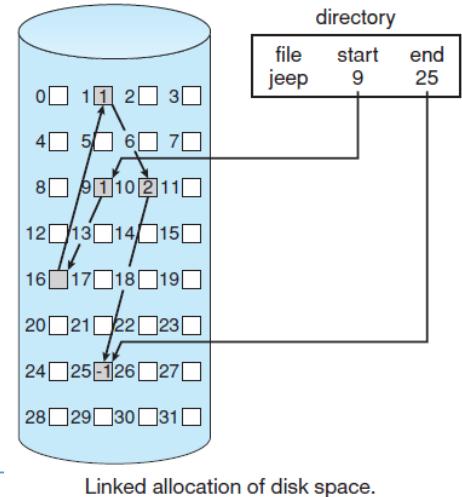
- Hạn chế:

- Không linh động.
- Phân mảnh ngoại vi



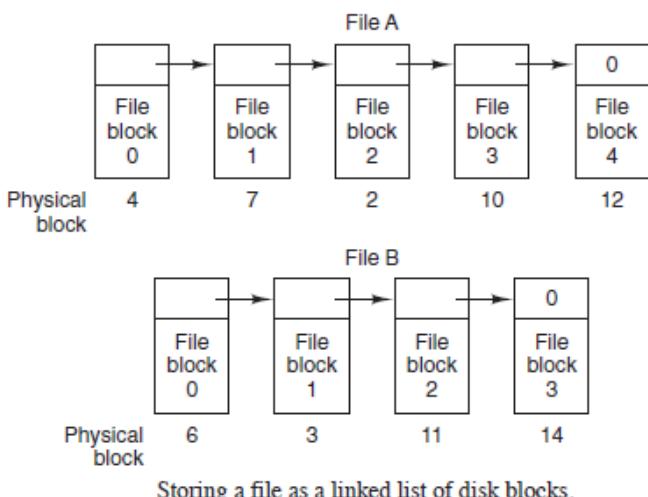
5.2.2 Cấp phát dùng danh sách liên kết

- Các khối của file có thể nằm ở các vị trí không liên tục nhau
- Mỗi mục trong bảng thư mục:
 - Tên file
 - Vị trí khối đầu tiên (start)
 - Vị trí khối kết thúc (end)
- Mỗi khối trên đĩa:
 - Con trỏ (~ 4 bytes) trỏ đến vị trí khối tiếp theo của file
 - Dữ liệu của file



Cấp phát dùng danh sách liên kết

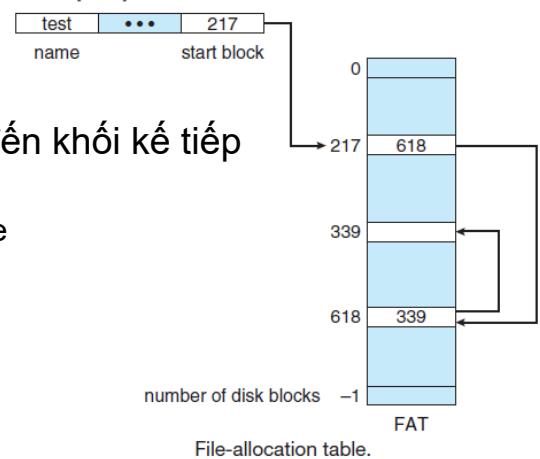
- Nhận xét:
 - Không phân mảnh ngoại vi
 - Truy xuất ngẫu nhiên chậm
 - Tốn bộ nhớ lưu các con trỏ
 - Một con trỏ lỗi → cả file



5.2.3 FAT (File Allocation Table)

• Mô hình cấp phát không liên tục, sử dụng FAT:

- Giải quyết hạn chế của cấp phát dùng danh sách liên kết
- Mỗi mục trong bảng thư mục lưu số hiệu của khối đầu tiên
- Các khối còn lại của file lưu trong bảng FAT, mỗi phần tử có con trỏ trỏ đến khối kế tiếp
- Ưu điểm:
 - Dễ quản lý các số hiệu khối đã cấp cho file
 - Truy xuất ngẫu nhiên dễ dàng
 - Kích thước file dễ mở rộng
- Hạn chế:
 - Bảng FAT có kích thước giới hạn

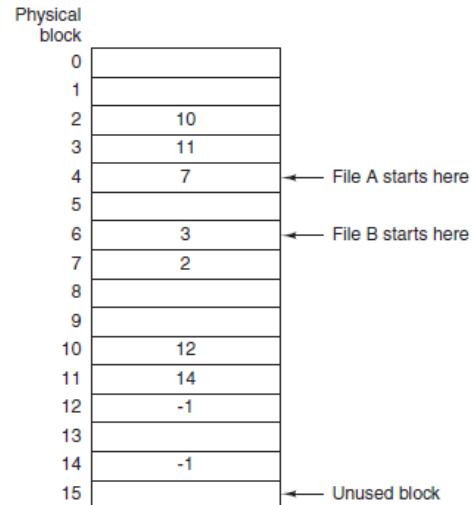


FAT (tt)

- Ví dụ 2: file A và B được cấp phát các khối như sau:
 - A: 4, 7, 2, 10, 12
 - B: 6, 3, 11, 14

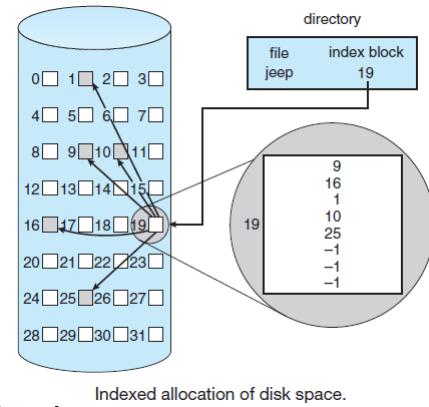
Bảng thư mục

...	(A, 4)	...	(B, 6)	...
-----	--------	-----	--------	-----



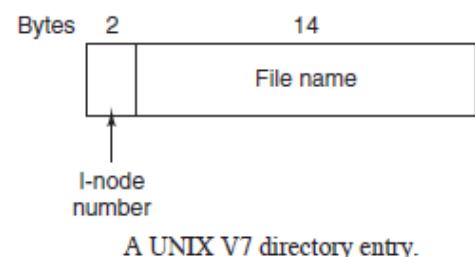
5.2.4 Cấp phát dùng index (bảng chỉ mục)

- Sử dụng bảng chỉ mục (một mảng) chứa tất cả vị trí các khối của file
- Một mục trong bảng thư mục:
 - Tên file
 - Vị trí khối của bảng chỉ mục của file (phản tử i quản lý cluster i)
- Ưu điểm
 - Hỗ trợ truy cập trực tiếp → Truy cập ngẫu nhiên tốt
 - Không phân mảnh ngoại vi
- Nhược điểm
 - Tốn không gian lưu trữ bảng index khi file có kích thước chỉ vài block
 - Nếu khối lưu bảng index bị bad → mất cả tập tin



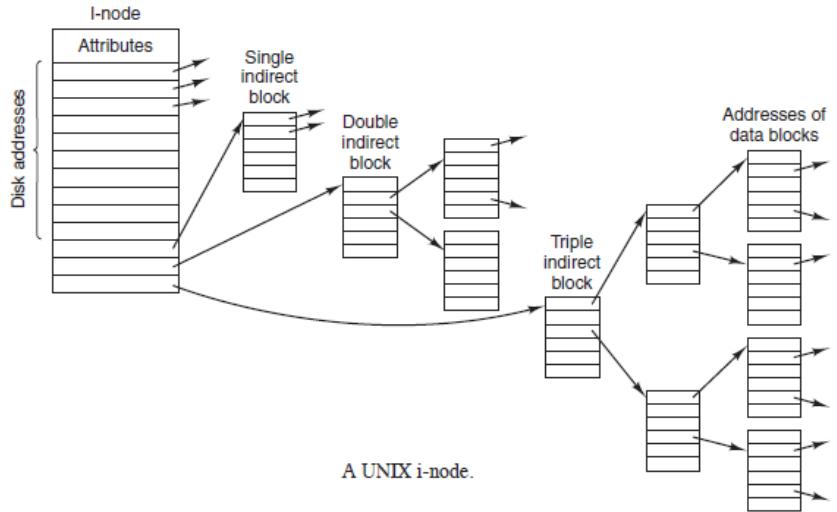
5.2.5 Bảng I-nodes

- Sử dụng trong UNIX
- Mỗi file được quản lý bằng một cấu trúc gọi là I-nodes
- Mỗi I-nodes bao gồm:
 - Phần lưu trữ thuộc tính file
 - Phần thứ 2 gồm
 - LINUX: 12 (UNIX v7: 10) con trỏ đầu trỏ đến các khối dữ liệu đầu tiên của file
 - Ba phần tử còn lại trỏ đến các bảng thứ cấp: single, double, w triple



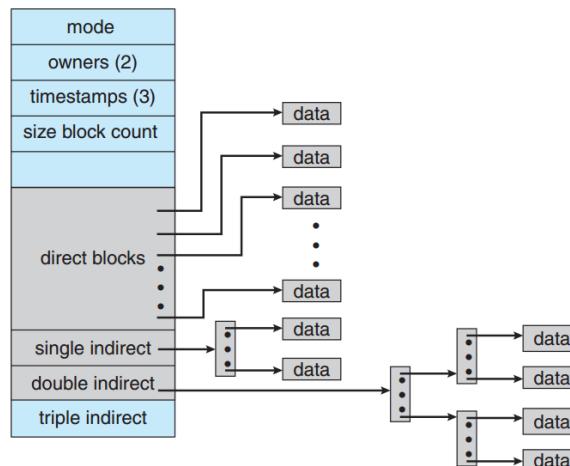
Bảng I-nodes (tt)

- Unix V7 File system



Bảng I-nodes (tt)

- UNIX inode

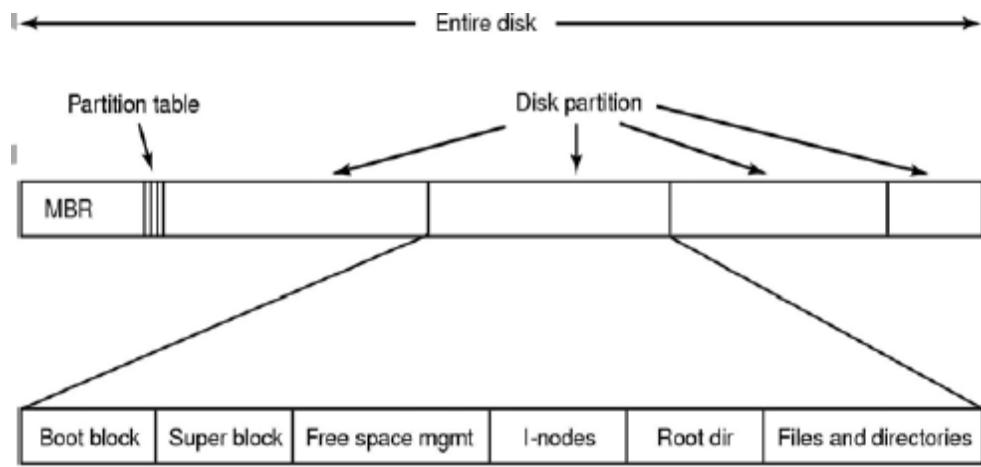


Bảng I-nodes (tt)

- Tổ chức quản lý đĩa bằng I-nodes:
 - MBR (Master Boot Record): sector đầu tiên chứa thông tin về đĩa
 - Partition Table: bảng phân vùng chứa các thông tin về mỗi phân vùng
 - Tổ chức một phân vùng gồm: boot block, super block, free space mgmt, I-nodes, root dir, files, directories
 - I-nodes: bảng chứa các i-node ghi thông tin mỗi file
 - Mỗi mục của bảng thư mục gốc (root-dir) ghi tên file và số hiệu i-node của file
 - Ưu điểm: quản lý hiệu quả khi hệ thống file lớn

Bảng I-nodes (tt)

- Tổ chức quản lý đĩa bằng I-nodes:



Bảng I-nodes (tt)

- Ví dụ: đọc file /usr/ast/mbox

Thư mục gốc		I-node 6 dành cho /user	Block 132 là thư mục /usr	I-node 26 dành cho /usr/ast	Block 406 là thư mục /usr/ast
1	-	Mode size times	6 *	Mode size times	26 *
1	..		1 **		6 **
4	bin	132	19 dick	406	64 grants
7	dev		30 erik		92 books
14	lib		51 jim		60 mbox
9	etc		26 ast		81 minix
6	usr		45 bal		17 src
8	tmp				

Tra cứu usr
tìm được i-node 6

I-node 6 cho biết
thông tin /user
được lưu tại
block 132

Block 132 là
thư mục /usr

/usr/ast là
i-node 26

I-node cho thấy
/usr/ast ở tại
block 406

/usr/ast/mbox là
i-node 60

5.2.6 NTFS (New Technology File System)

• Đặc điểm của NTFS

- NTFS là một hệ thống file mạnh và linh động, những đặc điểm nổi bật là :
 - Khả năng phục hồi
 - Bảo mật
 - Quản lý được đĩa dung lượng lớn và kích thước file lớn
 - Quản lý hiệu quả, có thể sử dụng nhiều data stream cho một file
 - Nén, mã hóa
 - Hard link, symbolic link
- Mọi phần tử trên volume là file, mỗi file là một tập hợp các thuộc tính.
- Nội dung dữ liệu của file cũng được xem là một thuộc tính.

NTFS (tt)

• Cấu trúc volume của NTFS



- Partition Boot Sector: các sector khởi động của partition (có thể đến 16 sectors) bao gồm các thông tin về cấu trúc của volume, cấu trúc của hệ thống file, mã nguồn khởi động,...
- Master File Table (MFT): chứa thông tin về file và thư mục trên đĩa.
- System file: vùng chứa các file hệ thống, gồm:
 - MFT2: bản sao của MFT.
 - Log file : thông tin về các giao tác dùng cho việc phục hồi.
 - Cluster bitmap: biểu diễn thông tin lưu trữ của các cluster
 - Bảng các thuộc tính: định nghĩa các kiểu thuộc tính hỗ trợ cho volume
- File area: vùng chứa nội dung thực sự của các tập tin

NTFS (tt)

• Master File Table (MFT):

- Lưu các thông tin về tất cả file và thư mục trên volume NTFS, kể cả các vùng trống.
- MFT được tổ chức như một bảng gồm nhiều hàng.
 - Mỗi hàng (1024 bytes) còn gọi là record mô tả cho một file hoặc một thư mục trên volume.
 - Nếu kích thước file nhỏ thì toàn bộ nội dung của file được lưu trong một hàng.
 - Trường hợp file lớn, thông tin của file sẽ lưu trong một hàng và phần còn lại của file sẽ được ghi vào các cluster trên đĩa, các con trỏ đến các cluster đó được lưu trong hàng chứa thông tin của file.
 - Mỗi hàng cũng lưu những thuộc tính cho file hay thư mục mà nó quản lý.

NTFS (tt)

- Các thuộc tính file trong NTFS:

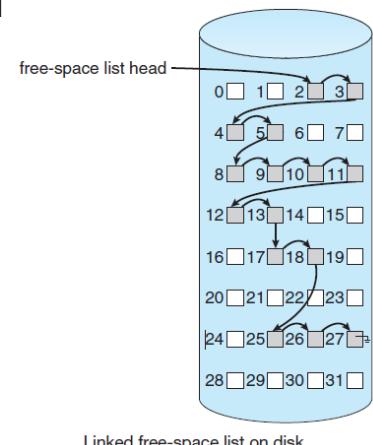
- Thông tin chuẩn: thuộc tính truy cập, thời điểm truy cập/sửa đổi lần cuối, chỉ số liên kết
- Danh sách thuộc tính: được sử dụng khi tất cả thuộc tính vượt quá một hàng của MFT
- Tên tập tin
- Mô tả an toàn: thông tin về người sở hữu và quyền truy cập
- Dữ liệu
- Chỉ mục gốc: dùng cho thư mục
- Chỉ mục định vị: dùng cho thư mục
- Thông tin volume
- Bitmap: sơ đồ mô tả hiện trạng các hàng trong MFT

5.2.7 Quản lý các khôi trống trên đĩa

- Dùng danh sách liên kết: mỗi phần tử trong DSLK là một khôi chứa một bảng gồm các số hiệu khôi trống, phần tử cuối của bảng lưu số hiệu khôi tiếp theo trong danh sách

- Nhận xét:

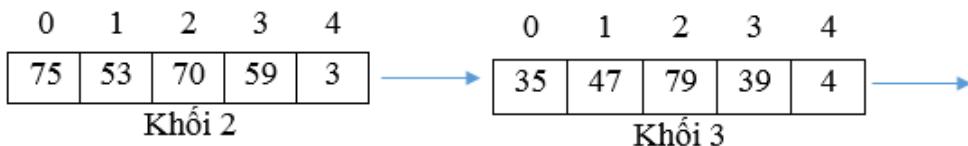
- đĩa hoàn toàn trống: tồn nhiều khôi nhớ cho DSLK
- đĩa gần đầy: tồn ít khôi nhớ cho DSLK



Quản lý các khối trống trên đĩa (tt)

- Dùng danh sách liên kết (tt)

- Ví dụ:



- Khối 2, 3 là các khối lưu số hiệu các khối trống
- Khối 2 là khối đầu tiên trong DSLK

Quản lý các khối trống trên đĩa (tt)

- Dùng vector bit (dãy bit)

- bit thứ $i = 1$: khối thứ i trống
- bit thứ $i = 0$: khối thứ i có dữ liệu
- Vector bit được lưu trên một hoặc nhiều khối đĩa, khi cần sẽ được đọc vào bộ nhớ để xử lý
- Ưu điểm: ít tốn khối nhớ so với DSLK
- Hạn chế:
 - kích thước vector là cố định
 - HĐH cần đồng bộ vector bit trong bộ nhớ và vector bit trên đĩa

1001101101101100
0110110111110111
1010110110110110
0110110110111011
1110111011101111
1101101010001111
0000111011010111
1011101101101111
1100100011101111
0111011101110111
1101111101110111

A bitmap

5.2.8 Hệ thống file trong MSDOS

- Nội dung một file được chia thành các block file lưu trữ tại các cluster trên đĩa, các cluster này có thể không nằm kề nhau.
- **Bảng FAT:**
 - Quản lý danh sách các cluster chứa nội dung của một file của tất cả các file
 - Ghi nhận trạng thái của các cluster trên đĩa: còn trống, đã cấp phát cho các file, bị bad không thể sử dụng hay dành riêng cho hệ điều hành.
- Trong quá trình khởi động máy tính HĐH nạp bảng FAT vào bộ nhớ để chuẩn bị cho việc đọc/ghi các file sau này.

Hệ thống file trong MSDOS (tt)

- Khi cần đọc/ghi file, HĐH phải dựa vào bảng FAT
- Bảng FAT hỏng → không thể ghi/đọc các file trên đĩa
- → HĐH DOS tạo ra hai bảng FAT hoàn toàn giống nhau là FAT1 và FAT2 (dự phòng)
- Nếu FAT1 bị hỏng thì DOS sẽ sử dụng FAT2 để khôi phục lại FAT1.
- **Trong hệ thống file FAT32:** nếu FAT1 bị hỏng thì HĐH sẽ chuyển sang sử dụng FAT2, sau đó mới khôi phục FAT1, và ngược lại.

Hệ thống file trong MSDOS (tt)

- Bảng FAT bao gồm nhiều phần tử, các phần tử được đánh địa chỉ bắt đầu từ 0
- Mỗi phần tử trong bảng FAT cho biết trạng thái của một cluster tương ứng trên vùng dữ liệu.
- HĐH DOS định dạng hệ thống file theo một trong 2 loại FAT là FAT12 và FAT16.
 - Mỗi phần tử trong FAT12 rộng 12 bit (1.5 byte)
 - Mỗi phần tử trong FAT16 rộng 16 bit(2 byte).

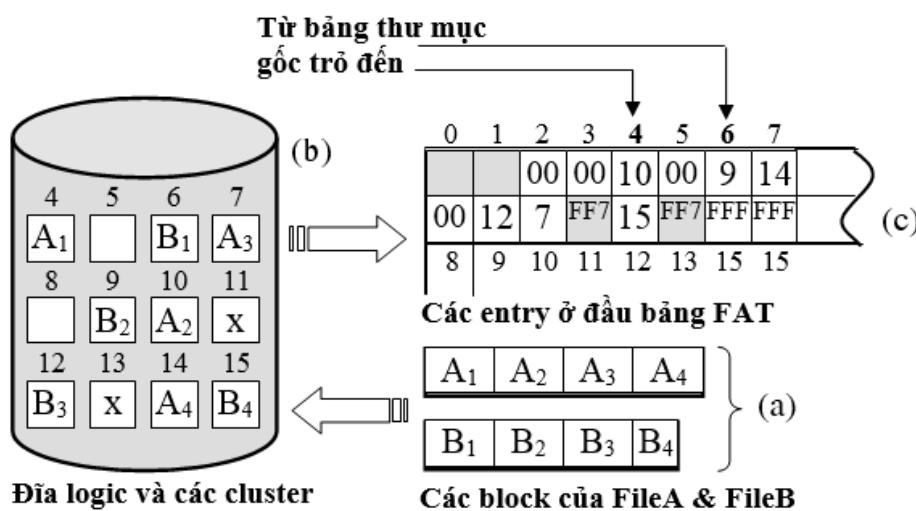
Hệ thống file trong MSDOS (tt)

- Bảng giá trị các phần tử trong FAT
 - 000h (0000h): cluster tương ứng còn để trống.
 - FF7h (FFF7h): cluster tương ứng bị bad.
 - FF0h (FFF0h) - FF6h (FFF6h): cluster tương ứng dành riêng cho hệ điều hành.
 - FF8h (FFF8h) - FFFh (FFFFh): cluster tương ứng là cluster cuối cùng trong dãy các cluster chứa nội dung của một file.
 - 002h (0002h) – FFEh (FFFEh): số hiệu của cluster trong bảng FAT, cho biết cluster tiếp theo trong dãy các cluster chứa nội dung của một file.

Hệ thống file trong MSDOS (tt)

- Trong bảng FAT, hai phần tử đầu tiên (00 và 01) được sử dụng để chứa một giá trị nhận biết khuôn dạng đĩa, được gọi là byte định danh (byte ID) của đĩa, đây là byte đầu tiên của bảng FAT.
 - Đối với đĩa cứng thì byte ID = F8h.
- Để đọc được nội dung của một file trên đĩa, HĐH phải tìm được dãy các cluster chứa nội dung của một file.
 - Cluster đầu tiên được chứa trong bảng thư mục gốc
 - Các cluster còn lại được chứa trong bảng FAT

Hệ thống file trong MSDOS (tt)



Hệ thống file trong MSDOS (tt)

• Nhận xét:

- Thao tác đọc file của DOS kém hiệu quả
 - đọc nội dung của file tại các cluster trên vùng data của đĩa
 - đọc và phân tích bảng FAT để các cluster chứa nội dung của một file.
- Khắc phục:
 - lưu danh sách các cluster chứa nội dung của một file vào một danh sách
 - khi đọc file hệ điều hành chỉ cần đọc nội dung của các cluster trên đĩa theo danh sách
 - Lưu danh sách các cluster còn trống trên đĩa → thì tốc độ ghi file nhanh
 - Windows98, windowsNT/2000

Hệ thống file trong MSDOS (tt)

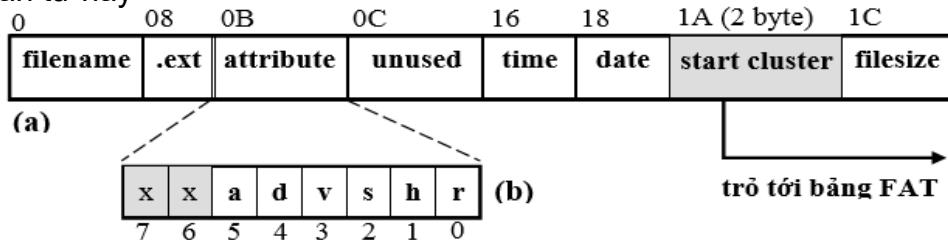
• Root Directory:

- Quản lý thông tin của các file và các thư mục đang được lưu trữ trên thư mục gốc của đĩa mềm hoặc đĩa logic trên đĩa cứng
- Số lượng phần tử trong bảng thư mục gốc được DOS quy định trước trong quá trình Format đĩa và được ghi tại word tại offset 11h trong boot sector, giá trị này không thể thay đổi.
- → tổng số file và thư mục con chứa trên thư mục gốc của đĩa là có giới hạn. Đây là một hạn chế của DOS.

Hệ thống file trong MSDOS (tt)

• Root Directory (tt)

- Mỗi phần tử trong bảng thư mục gốc có kích thước 32 byte, dùng để chứa thông tin về một file/thư mục đang lưu trên thư mục gốc của đĩa.
 - Khi một file/thư mục được tạo ra trên thư mục gốc của đĩa thì HĐH dùng một phần tử trong bảng thư mục gốc để chứa các thông tin liên quan của nó
 - Khi một file/thư mục bị xoá/di chuyển khỏi thư mục gốc thì HĐH sẽ thu hồi lại phần tử này



Cấu trúc của một phần tử trong bảng thư mục gốc của DOS

Hệ thống file trong MSDOS (tt)

• Root Directory (tt)

- Byte đầu tiên (offset 00) của một phần tử trong thư mục gốc (byte trạng thái) chứa một trong các giá trị đặc biệt sau đây:
 - 0: cho biết phần tử chưa được sử dụng.
 - E5h: phần tử là của một file đã được tạo nhưng đã bị xoá.
 - 05h: kí tự đầu tiên của tên file này thực tế là E5h, nếu file được tạo có tên bắt đầu là kí tự có mã ascii là E5h thì sẽ được thay bằng kí tự có mã là 05h, để phân biệt file này với các file đã bị xoá.
 - 2Eh (kí tự '.'): phần tử chứa thông tin của một thư mục con, nếu byte thứ 2 cũng là 2Eh ("..") thì trường start cluster sẽ chứa số hiệu cluster đầu tiên của thư mục cha, nếu là thư mục gốc thì là 0000h.

Hệ thống file trong MSDOS (tt)

• Root Directory (tt)

- Các trường trong một phần tử của bảng thư mục gốc

Offset	Nội dung	Độ lớn
00h	Tên chính của file (ASCII)	8 byte
08h	Phần mở rộng của tên File (ASCII)	3 byte
0Bh	Thuộc tính file (attribute) (0, 0, A, D, V, S, H, R)	1 byte
0Ch	Dự trữ, chưa được sử dụng (Unused)	10 byte
16h	Giờ thay đổi tập tin cuối cùng (giờ 5bit, phút 6bit, giây/2 5bit)	2 byte
18h	Ngày thay đổi tập tin cuối cùng (ngày 5 bit, tháng 4 bit năm -1980: 7 bit)	2 byte
1Ah	Cluster đầu tiên của File (start cluster)	2 byte
1Ch	Kích thước của File (filesize)	4 byte

Hệ thống file trong MSDOS (tt)

• Root Directory (tt)

- Ví dụ: một phần tử lưu tập tin có dãy byte như sau:

**44 45 54 48 49 20 20 20 44 4F 43 20 00 00 00 00
 00 00 00 00 00 00 16 4A BB 34 14 00 1E 0A 00 00**

- → các thuộc tính tập tin là:

- Tên file: DETHI.DOC
- Kích thước: offset 1Ch: 0A1E = 2590
- Cluster bắt đầu: offset 1Ah: 14h = 20
- Ngày cập nhật: offset 18h: 34BBh = 0011010 0101 11011b = 27/5/2006
- Giờ cập nhật: offset 16h: 4A16h = 01001 010000 10110h = 9:16:44

Hệ thống file trong MSDOS (tt)

• Root Directory (tt)

- Trường hợp xóa file:
 - HĐH thay kí tự đầu tiên của tên file tại phần tử trong bảng thư mục gốc bằng giá trị E5h
 - → có thể khôi phục lại được file này, bằng cách thay kí tự mã E5h ở byte đầu tiên của tên file bằng một kí tự khác
 - Khi duyệt bảng thư mục gốc gặp các phần tử có byte đầu = E5h HĐH biết đây là phần tử của file đã bị xóa nên không in ra màn hình.
 - Trường hợp không khôi phục được: do sau một thời gian, HĐH đã sử dụng phần tử trong thư mục gốc, các phần tử trong bảng FAT và các cluster trên vùng data của file đã bị xóa, cấp phát cho các file mới.

Hệ thống file trong MSDOS (tt)

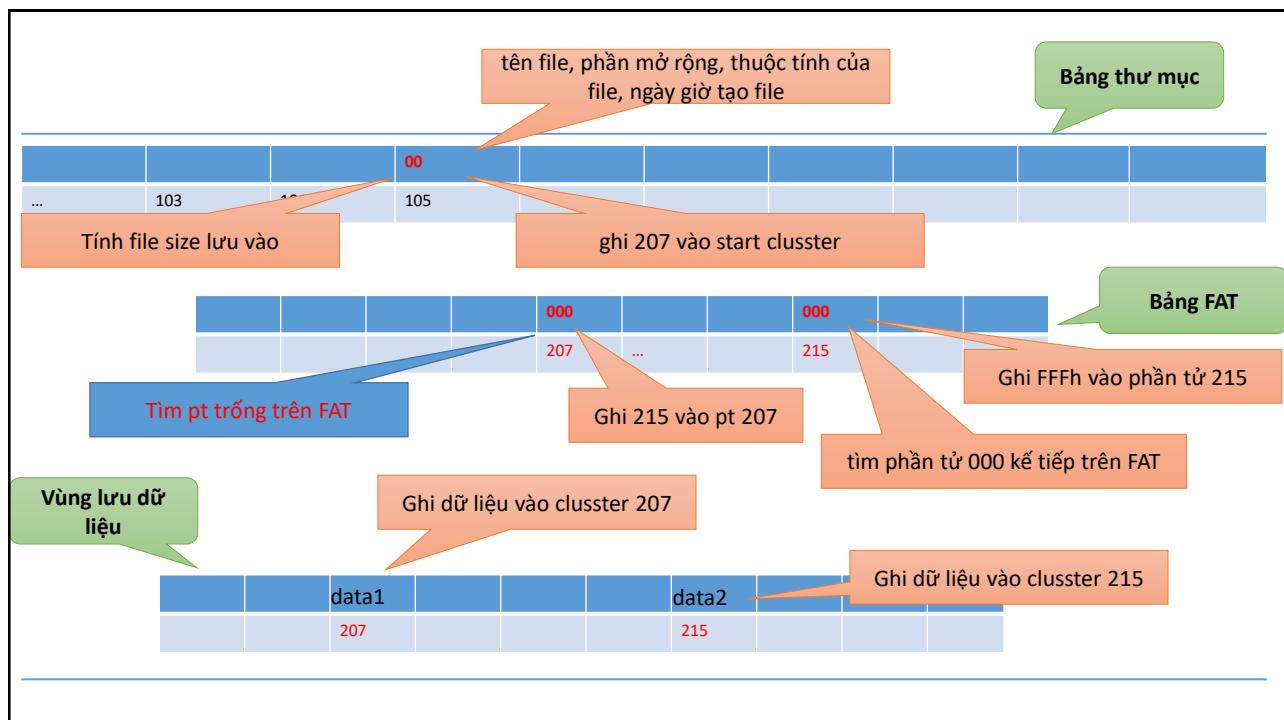
• Root Directory (tt)

- Trường hợp ghi file: HĐH DOS thực hiện các bước sau:
 1. Tìm phần tử mà byte đầu tiên của nó chứa giá trị 00. **Giả sử tìm được phần tử thứ 105.**
 2. Ghi tên file, phần mở rộng, thuộc tính của file, ngày giờ tạo file vào các trường tương ứng tại **phân tử 105** trong bảng thư mục gốc.
 3. Tìm một phần tử trong bảng FAT chứa giá trị 000h, giả sử tìm được **207** → **cluster 207** trên vùng data còn trống.
 4. Ghi giá trị **207**, vào trường **start cluster** tại offset **1Ah** của phần tử **105** trong bảng thư mục gốc.
 5. Ghi block đầu tiên của file vào **cluster 207** trên vùng data. Nếu nội dung của file chứa vừa đủ trong 1 cluster, thì DOS sẽ thực hiện bước 9, ngược lại DOS tiếp tục thực hiện bước 6.

Hệ thống file trong MSDOS (tt)

• Root Directory (tt)

6. Tiếp tục tìm một phần tử trong bảng FAT chứa giá trị 000h, giả sử tìm được phần tử 215 → cluster 215 trên vùng data còn trống.
7. Ghi giá trị 215 vào phần tử 207 trong bảng FAT và ghi block thứ hai của file vào cluster 215 trên vùng data.
8. Lặp lại các bước 6 và 7 cho đến khi ghi hết các block của file vào các cluster trên vùng data. Giả sử block cuối cùng của file được ghi vào cluster **n**.
9. Bước cuối cùng: ghi giá trị FFFh vào phần tử **n** trong bảng FAT.
10. Tính kích thước của file và ghi vào trường filesize của phần tử 105 trong bảng thư mục gốc.



Hệ thống file trong MSDOS (tt)

- **Thư mục con (subdirectory):**

- Số phần tử trong bảng thư mục gốc là cố định, không thể mở rộng, được DOS quy định trong quá trình định dạng đĩa. → khái niệm thư mục con.
- Cấu trúc của một phần tử trong bảng thư mục của thư mục con tương tự cấu trúc của một phần tử trong bảng thư mục gốc.
 - Đầu chứa thông tin của một thư mục (con) gồm:
 - trường Filename là tên của thư mục (con)
 - trường Attribute = 16 (00010000)
 - trường Start cluster = cluster đầu tiên của thư mục (con)
 - trường Filesize = 0
 - ...
 - Hai phần tử đầu tiên trong bảng thư mục của thư mục con chứa các giá trị đặc biệt.

Hệ thống file trong MSDOS (tt)

- **Thư mục con (tt)**

- **Đặc điểm thư mục con trong DOS:**

- Được lưu trữ giống các file khác trên đĩa, muốn đọc được thư mục con HĐH phải dò trong bảng FAT.
- Bảng thư mục của thư mục con
 - **có số lượng phần tử không giới hạn**
 - có thể định vị tại một vị trí bất kỳ trên vùng data của đĩa, nếu tạo quá nhiều thư mục con trên đĩa thì bảng thư mục của các thư mục con sẽ chiếm hết nhiều không gian đĩa trên vùng data.
 - Virus khó có thể tấn công bảng thư mục của thư mục con vì nó không cố định
- Kích thước của thư mục con là kích thước của tất cả các file trong thư mục con, dựa vào bảng thư mục của thư mục con và bảng FAT

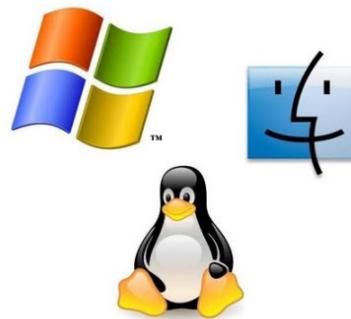
Hệ thống file trong MSDOS (tt)

• Thư mục con (tt)

- Tạo thư mục con: HĐH tạo bảng thư mục cho nó và khởi tạo 2 phần tử đầu tiên trong bảng thư mục này:
 - Phần tử thứ nhất:
 - byte đầu tiên của trường filename chứa mã ascii của kí tự dấu chấm (.)
 - phần tử này chỉ đến chính thư mục hiện hành.
 - Trường Start cluster cho biết cluster bắt đầu của thư mục này.
 - Phần tử thứ hai:
 - 2 byte đầu tiên của trường filename chứa 2 mã ascii của 2 kí tự dấu chấm (..)
 - phần tử này chỉ đến thư mục cha của nó.
 - Trường Start cluster cho biết cluster bắt đầu của thư mục cha của nó, nếu cha của nó là thư mục gốc thì trường này chứa giá trị 0.

Chương 6

QUẢN LÝ NHẬP XUẤT



Nội dung

1. Nguyên lý phần cứng I/O
2. Nguyên lý phần mềm I/O
3. Đĩa cứng

377

Tài liệu tham khảo

- Andrew S. Tanenbaum, Modern Operating Systems, Pearson, 2015
 - Chương 5
- Abraham Silberschatz, Peter Baer Galvin, Greg Gagne, Operating System Concepts, John Wiley, 2013
 - Chương 13

378

6.1 Nguyên lý phần cứng I/O

- Thiết bị nhập/xuất
- Bộ điều khiển thiết bị nhập/xuất
- Các thanh ghi nhập/xuất
- Kỹ thuật DMA
- Ngắt (Interrupt)

379

6.1.1 Thiết bị I/O

- Thiết bị I/O có thể chia làm hai loại chính:
 - Thiết bị khối (block devices):
 - lưu trữ thông tin theo các khối có kích thước cố định (512 - 65,536 byte), mỗi khối có địa chỉ riêng
 - Có thể đọc hoặc ghi từng khối một cách độc lập
 - Đĩa cứng, USB,...
 - Thiết bị tuần tự (character devices):
 - Lưu trữ thông theo dãy tuần tự các bit, không có địa chỉ
 - Thông tin được gởi/nhận một cách tuần tự
 - Màn hình, bàn phím, máy in, card mạng, chuột)
- Một số thiết bị không thuộc hai loại trên: clock, màn hình cảm ứng,...

380

Thiết bị I/O (tt)

• Đặc tính của các thiết bị I/O:

- Tốc độ truyền dữ liệu
- Dung lượng lưu trữ, tốc độ truy xuất
- Chức năng
- Đơn vị truyền dữ liệu: khôi hay ký tự
- Biểu diễn dữ liệu
- Tình trạng lỗi: nguyên nhân, cách báo lỗi,...

381

Thiết bị I/O (tt)

• Các thiết bị I/O có tốc độ giao tiếp rất khác nhau tùy vào tính chất sử dụng:

Device	Data rate	Device	Data rate
Keyboard	10 bytes/sec	FireWire 800	100 MB/sec
Mouse	100 bytes/sec	Gigabit Ethernet	125 MB/sec
56K modem	7 KB/sec	SATA 3 disk drive	600 MB/sec
Scanner at 300 dpi	1 MB/sec	USB 3.0	625 MB/sec
Digital camcorder	3.5 MB/sec	SCSI Ultra 5 bus	640 MB/sec
4x Blu-ray disc	18 MB/sec	Single-lane PCIe 3.0 bus	985 MB/sec
802.11n Wireless	37.5 MB/sec	Thunderbolt 2 bus	2.5 GB/sec
USB 2.0	60 MB/sec	SONET OC-768 network	5 GB/sec

382

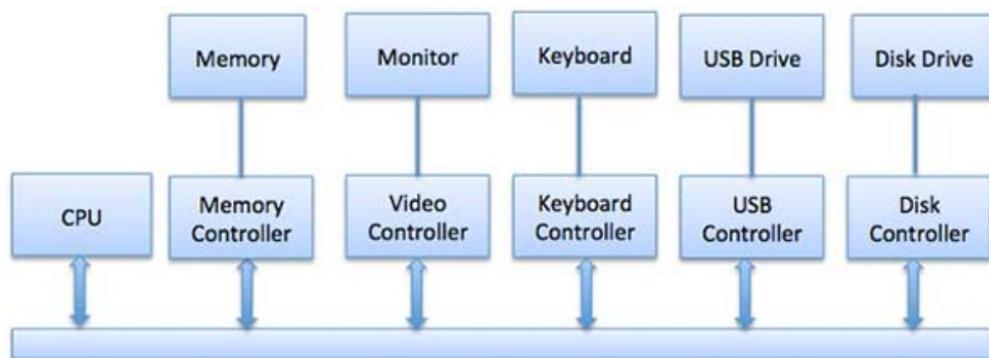
6.1.2 Bộ điều khiển thiết bị nhập/xuất (Device Controller)

- Các thiết bị I/O có thể giao tiếp với thiết bị khác
- Để máy tính có thể giao tiếp với thiết bị I/O → sử dụng một bộ phận tương thích với bộ phận điều khiển của thiết bị, gọi là device controller
- Mỗi bộ điều khiển thiết bị có một vài thanh ghi được sử dụng để giao tiếp với CPU.
- Bằng cách ghi vào các thanh ghi này, hệ điều hành có thể ra lệnh cho thiết bị cung cấp dữ liệu, chấp nhận dữ liệu, tự bật hoặc tắt hoặc thực hiện một số hành động,...
- Bằng cách đọc từ các thanh ghi này, hệ điều hành có thể biết được trạng thái của thiết bị có sẵn sàng chấp nhận một lệnh mới hay không

383

Bộ điều khiển thiết bị nhập/xuất (tt)

- Mô hình kết nối CPU, bộ nhớ, bộ điều khiển và thiết bị I/O



384

6.1.3 Các thanh ghi nhập/xuất

- **Thanh ghi lệnh:** thanh ghi chứa mã lệnh chức năng mà CPU ghi vào để controller thực hiện, chiều di chuyển thông tin của thanh ghi này là: CPU → controller (OUT).
- **Thanh ghi trạng thái:** thanh ghi chứa các bit thông tin về trạng thái hiện hành của thiết bị I/O tương ứng (bận, rãnh,...), chiều di chuyển thông tin của thanh ghi này là: controller → CPU (IN).
- **Thanh ghi dữ liệu xuất:** chứa dữ liệu mà CPU muốn xuất ra thiết bị I/O, chiều di chuyển thông tin của thanh ghi này là: CPU → controller (OUT).
- **Thanh ghi dữ liệu nhập:** chứa dữ liệu mà thiết bị I/O đưa vào hệ thống, chiều di chuyển thông tin của thanh ghi này là: controller → CPU (IN)

385

Các thanh ghi nhập/xuất (tt)

- Mỗi thanh ghi cần có địa chỉ truy xuất duy nhất.
- Có ba loại địa chỉ:
 - Địa chỉ I/O
 - Địa chỉ memory-mapped I/O
 - Địa chỉ hỗn hợp

386

Các thanh ghi nhập/xuất (tt)

• Địa chỉ I/O có hai loại:

- Địa chỉ ô nhớ dùng để truy xuất các ô nhớ trong RAM
- Địa chỉ I/O dùng để truy xuất các thanh ghi của các mạch controller.
- Ví dụ:
MOV AL, F5: đọc nội dung ô nhớ RAM ở địa chỉ F5_h vào thanh ghi AL của CPU
IN AL, F5: đọc nội dung thanh ghi của controller có địa chỉ (port) là F5_h.

• Địa chỉ memory-mapped I/O:

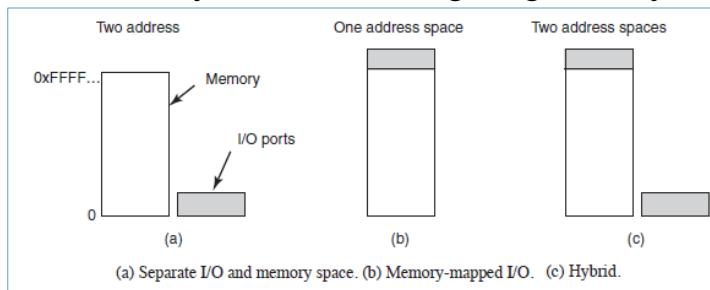
- Mỗi thanh ghi I/O có một địa chỉ ô nhớ xác định, muốn truy xuất thanh ghi I/O, CPU sẽ dùng lệnh truy xuất ô nhớ như bình thường.
- Ví dụ:
MOV AL, F5: đọc nội dung thanh ghi I/O ở địa chỉ F5_h tùy thuộc vào địa chỉ này đang được dùng cho phần tử nào.

387

Các thanh ghi nhập/xuất (tt)

• Địa chỉ hỗn hợp (hybrid): có hai loại

- Địa chỉ ô nhớ dành để truy xuất các ô nhớ trong RAM
- Địa chỉ I/O dành truy xuất các thanh ghi của các mạch controller.
- Tùy theo tính chất sử dụng của từng thanh ghi → sử dụng địa chỉ I/O (port) hay địa chỉ memory với lệnh tương ứng để truy xuất



388

6.1.4 Kỹ thuật DMA (Direct Memory Access)

- CPU giao tiếp với I/O: có 3 cách

- Special Instruction I/O

- Sử dụng các lệnh của CPU để điều khiển các thiết bị I/O: gửi dữ liệu đến I/O hoặc đọc từ I/O

- Memory-mapped I/O

- Direct memory access (DMA)

389

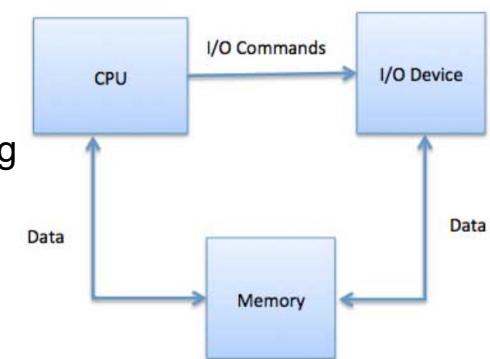
Kỹ thuật DMA (tt)

- Memory-mapped I/O

- Một không gian địa chỉ được chia sẻ giữa bộ nhớ và I/O. Thiết bị được kết nối trực tiếp với các vị trí trong bộ nhớ chính để chuyển dữ liệu đến bộ nhớ hoặc đọc dữ liệu từ bộ nhớ không cần thông qua CPU.

- HĐH phân bổ bộ đệm trong bộ nhớ và báo cho I/O dùng bộ đệm đó để gửi dữ liệu đến CPU. Thiết bị I/O hoạt động không đồng bộ với CPU, ngắt CPU khi kết thúc.

- Ưu điểm: các lệnh có thể truy cập bộ nhớ đều có thể được sử dụng để thao tác với thiết bị I/O → nhanh → được sử dụng cho các thiết bị I/O tốc độ cao

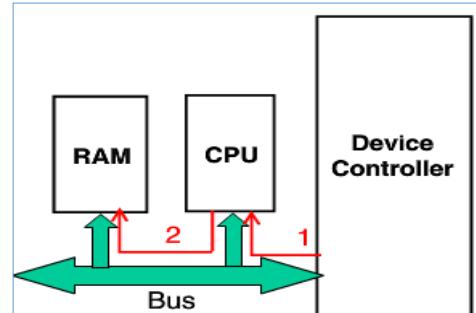


390

Kỹ thuật DMA (tt)

• Direct memory access (DMA)

- Việc chuyển thông tin giữa thiết bị I/O với bộ nhớ máy tính được thực hiện thông qua CPU theo cơ chế tuần tự.
- Mỗi lần cần di chuyển dữ liệu từ thiết bị I/O (đang nằm trong thanh ghi data in của controller), CPU phải thực hiện 2 lệnh máy liên tiếp:
 1. Lệnh IN (hay MOV) để di chuyển dữ liệu từ controller vào thanh ghi của CPU.
 2. Lệnh MOV để di chuyển dữ liệu từ thanh ghi CPU ra ô nhớ RAM xác định.
- Hạn chế: không hiệu quả
 - Tốn nhiều chu kỳ để di chuyển thông tin
 - Thông tin phải đi qua trung gian là CPU
- → Kỹ thuật DMA



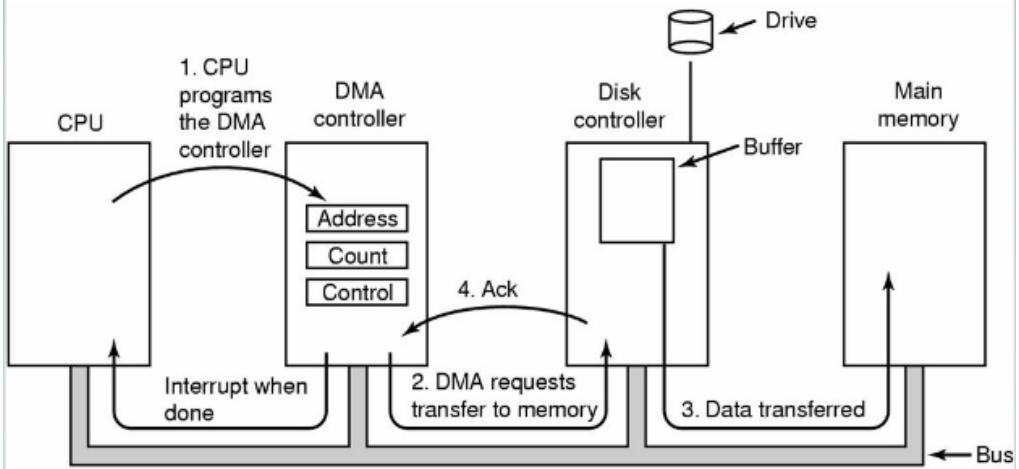
391

Kỹ thuật DMA (tt)

- DMA là một vi mạch có chức năng đặc biệt, cho phép chuyển dữ liệu trực tiếp giữa I/O và RAM mà không cần đi ngang qua CPU.
- CPU khởi động các thanh ghi của DMA, các thanh ghi này chứa địa chỉ ô nhớ và số byte cần chuyển.
- DMA chuyển số liệu và khi chấm dứt thì trả quyền điều khiển cho CPU: Bộ điều khiển DMA báo (ngắt) CPU khi hoàn thành công việc theo từng block dữ liệu.

392

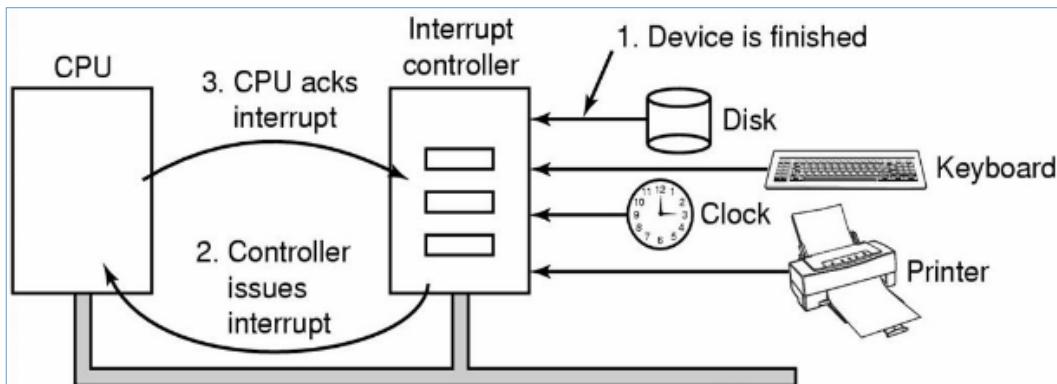
Kỹ thuật DMA (tt)



393

6.1.5 Kỹ thuật Interrupt

- Bộ điều khiển thiết bị sẽ báo (ngắt) CPU khi hoàn thành công việc theo từng đơn vị dữ liệu → CPU không phải chờ,



394

6.2 Nguyên lý phần mềm nhập/xuất

- Mục tiêu của phần mềm nhập xuất
- Lập trình nhập/xuất
- Cơ chế ngắt
- Nhập/xuất sử dụng kỹ thuật DMA

395

6.2.1 Mục tiêu của phần mềm nhập xuất

- **Độc lập thiết bị:** chương trình ứng dụng có thể truy cập bất kỳ thiết bị I/O nào không phụ thuộc vào tính chất vật lý hay loại thiết bị.
- **Đặt tên thiết bị đồng nhất:** tên thiết bị là một chuỗi hay một số nguyên giống như tên một file và không phụ thuộc vào thiết bị nào
- **Che dấu việc xử lý lỗi:** nếu có lỗi trong khi truy xuất I/O, hệ thống phần mềm I/O phải cố gắng xử lý ở cấp thấp nhất. Nếu không được, thì trình điều khiển thiết bị xử lý (cố gắng đọc lại khôi). Chỉ khi các lớp dưới không thể giải quyết được vấn đề thì các lớp trên mới được thông báo về nó.

396

Mục tiêu của phần mềm nhập xuất (tt)

- **Chuyển đổi cách nhập/xuất dữ liệu dạng bất đồng bộ thành dạng đồng bộ:**

- Ở cấp vật lý: mỗi lần cần đọc một sector đĩa, CPU sẽ xuất lệnh đọc ra controller, chờ ngắt khi có dữ liệu, đọc khối dữ liệu vào bộ nhớ.
- Ở cấp ứng dụng: nên cung cấp một hàm chức năng, ứng dụng gọi hàm này → trả về kết quả → ứng dụng có dữ liệu để xử lý tiếp.

- **Sử dụng bộ đệm dữ liệu nhập/xuất:**

- Dữ liệu từ một thiết bị I/O đưa vào máy tính cần được cache lại để cung cấp cho ứng dụng khi có yêu cầu

- **Thiết bị dùng chung (shared)**

- Tại mỗi thời điểm có thể phục vụ cho nhiều ứng dụng đồng thời

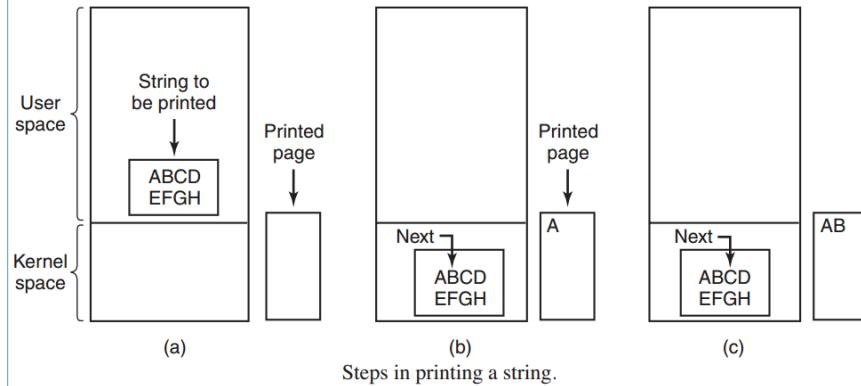
- **Thiết bị dùng riêng (delicated):**

- Chỉ phục vụ một ứng dụng tại mỗi thời điểm

397

6.2.2 Lập trình nhập/xuất

- **Ví dụ:**



```

copy_from_user(buffer, p, count);
for (i = 0; i < count; i++) {
    while (*printer_status_reg != READY) ;
    *printer_data_register = p[i];
}
return_to_user();
    
```

/* p is the kernel buffer */
 /* loop on every character */
 /* loop until ready */
 /* output one character */

398

Lập trình nhập/xuất (tt)

- Nhận xét:

- Đơn giản
- Do CPU thực hiện hoàn toàn → chiếm dụng CPU toàn thời gian cho đến khi tất cả I/O hoàn thành.
- Không hiệu quả trong các hệ thống phức tạp

399

Dùng cơ chế ngắt

- Để tận dụng CPU, khi I/O làm việc, CPU được điều phối chuyển sang công việc khác → cơ chế ngắt
- Ví dụ: ghi một chuỗi lên máy in
 - (a) Code executed at the time the print system call is made.
 - (b) Interrupt service procedure for the printer.

```
copy_from_user(buffer, p, count);
enable_interrupts();
while (*printer_status_reg != READY) ;
*printer_data_register = p[0];
scheduler();
```

(a)

```
if (count == 0) {
  unblock_user();
} else {
  *printer_data_register = p[i];
  count = count - 1;
  i = i + 1;
}
acknowledge_interrupt();
return_from_interrupt();
```

(b)

400

Dùng kỹ thuật DMA (DMA)

- Ngắt → lãng phí một lượng thời gian CPU → sử dụng DMA.
- Ý tưởng là để bộ điều khiển DMA nạp các ký tự vào máy in cùng một lúc mà không cần CPU.
- DMA được lập trình I/O để thực hiện tất cả công việc thay cho CPU
- Yêu cầu phần cứng đặc biệt (bộ điều khiển DMA) nhưng giải phóng CPU trong tiến trình I/O để thực hiện công việc khác.

401

Dùng kỹ thuật DMA (DMA)

- Ví dụ: in một chuỗi dùng DMA
 - (a) Code executed when the print system call is made.
 - (b) Interrupt-service procedure.

```
copy_from_user(buffer, p, count);
set_up_DMA_controller();
scheduler();
```

(a)

```
acknowledge_interrupt();
unblock_user();
return_from_interrupt();
```

(b)

402

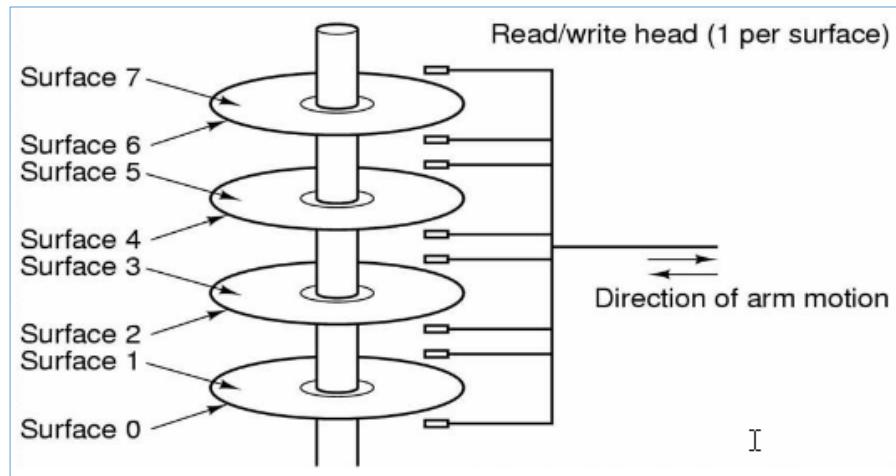
6.4. Đĩa từ

- Giới thiệu
- Định dạng đĩa
- Các thuật toán điều phối đĩa
- Xử lý lỗi

403

6.4.1 Giới thiệu về đĩa

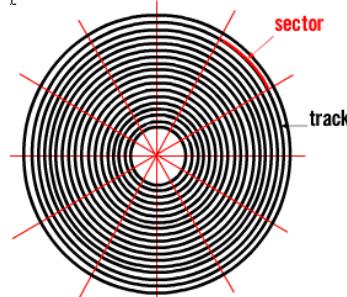
- Cấu trúc đĩa



404

Giới thiệu về đĩa (tt)

- Đĩa vật lý là không gian dữ liệu 3 chiều, gồm nhiều cylinder
- Mỗi cylinder gồm nhiều track
- Mỗi track chứa nhiều sector.
- Sector là đơn vị truy xuất tin nhỏ nhất ở cấp vật lý (từ ngoài ta không thể truy xuất từng byte dữ liệu trên disk được).
- Muốn truy xuất một sector, cần xác định tọa độ 3 chiều của nó (C,H,S) → khó.



405

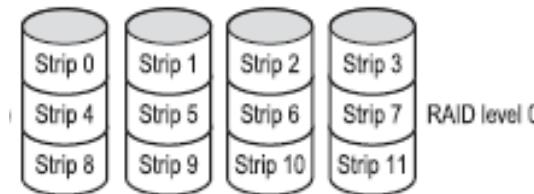
Giới thiệu về đĩa (tt)

- Đĩa SLED (Single Large Expensive Disk):
 - Tốc độ truy xuất chậm
 - Độ tin cậy thấp
 - → khắc phục bằng kỹ thuật RAID (Hệ thống đĩa dự phòng - Redundant Array of Inexpensive Disks)
- RAID:
 - Kết nối một dãy các ổ cứng có chi phí thấp lại với nhau để hình thành một đĩa logic có dung lượng lớn.
 - Ưu điểm của RAID
 - Dự phòng
 - Hiệu quả cao
 - Giá thành thấp

406

Đĩa cứng RAID 0

- Dùng kỹ thuật striping: gộp nhiều sector thành một strip, phân bổ trên nhiều đĩa → đĩa luận lý như sau:
 - phân bổ các strip theo kiểu round-robin.
 - Nếu cần truy xuất khối dữ liệu dài n strip, controller trên RAID0 sẽ biết được khối dữ liệu n strip này nằm trên n đĩa khác nhau → mỗi đĩa truy xuất 1 strip.
 - Dữ liệu truy xuất được sẽ được hợp lại/phân ra bởi controller



407

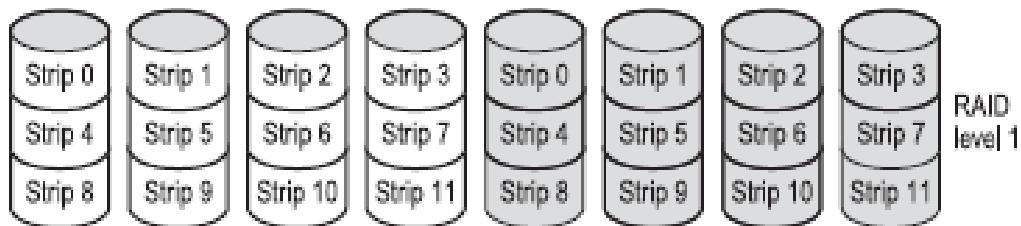
Đĩa cứng RAID 0 (tt)

- **Ưu điểm:**
 - Tăng tốc độ truy xuất lên n lần
 - Tăng hiệu quả lưu trữ.
 - Không làm mất dung lượng dữ liệu.
- **Nhược điểm**
 - Không có ổ dự phòng
 - Giảm độ tin cậy n lần.

408

Đĩa cứng RAID 1

- Dùng kỹ thuật Master/Mirror: nhân bản dự phòng
- Mỗi đĩa có sẵn (master), cần thêm một đĩa dự phòng (slave).
- Ghi dữ liệu: controller sẽ ghi dữ liệu đồng thời lên cả master lẫn slave.



409

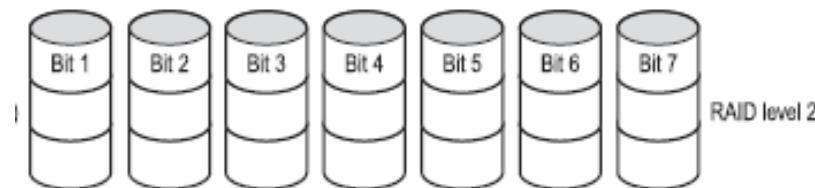
Đĩa cứng RAID 1 (tt)

- Đọc dữ liệu: controller yêu cầu master thực hiện.
- Trường hợp master trục trặc, controller đổi vai trò của master và slave → thực hiện lại việc đọc dữ liệu.
- Phục hồi: copy lại dữ liệu.
- Ưu điểm: khả năng dự phòng dữ liệu.
- Nhược điểm:
 - Sử dụng nhiều đĩa cứng → hao phí
 - Không tăng hiệu suất thực thi.
 - Tốn thời gian thay đổi ổ hoạt động khi có sự cố.

410

Đĩa cứng RAID 2

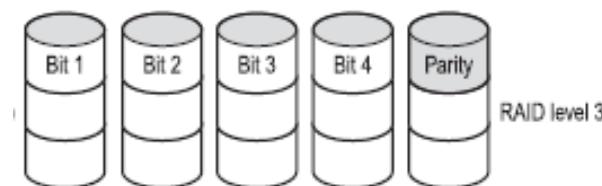
- Để lưu 32 bit dữ liệu, controller sẽ đổi 32 bit dữ liệu thành mã sửa sai Hamming 39 bit, từng bit sẽ được ghi lên một đĩa khác nhau.
- Tốc độ: tăng lên 32 lần so với SLED.
- Độ tin cậy: nếu có lỗi ở một disk, nhờ mã Hamming, controller sẽ tự sửa sai để xác định 32 bit dữ liệu gốc



411

Đĩa cứng RAID 3

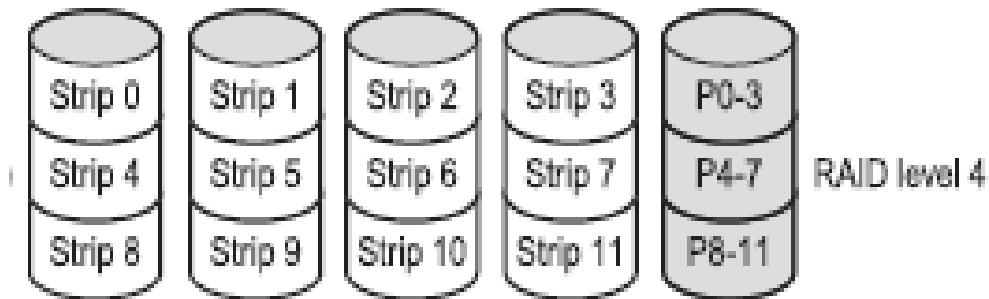
- Tương tự RAID 0, dùng kỹ thuật stripping theo từng byte: chia file thành từng byte ghi trên từng đĩa
- Sử dụng thêm 1 đĩa chứa bit phát hiện lỗi (parity).
- Tốc độ truy xuất tập tin kích thước nhỏ rất nhanh
- Độ tin cậy: nếu có lỗi ở một đĩa, nhờ bit parity phát hiện lỗi + tín hiệu báo lỗi ở đĩa thứ i, controller sẽ tự sửa sai bit i để xác định 32 bit dữ liệu gốc.



412

Đĩa cứng RAID 4

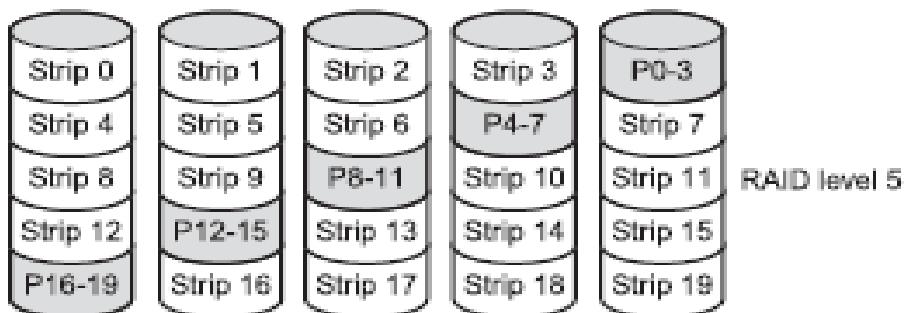
- Dữ liệu ghi trên n đĩa + 1 đĩa chứa bit phát hiện lỗi (parity).
- Độ tin cậy: nếu có lỗi ở một đĩa, nhờ bit parity phát hiện lỗi + tín hiệu báo lỗi ở đĩa thứ i.



413

Đĩa cứng RAID 5

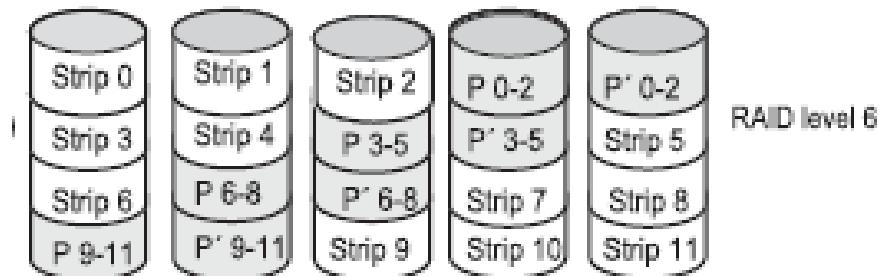
- Sử dụng kỹ thuật stripping theo từng Block và phân chia thông tin “parity” (chẵn lẻ)
- Cần ít nhất 3 đĩa
- Dữ liệu ghi vào n đĩa + 1 đĩa ghi bit parity dự phòng theo kiểu luân phiên



414

Đĩa cứng RAID 6

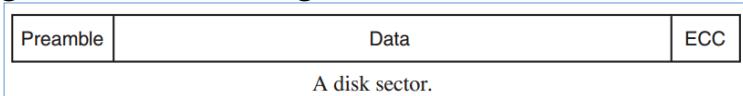
- Phát triển từ RAID 5
- Cần ít nhất 4 đĩa
- Dữ liệu ghi vào n đĩa + 2 đĩa ghi bit parity dự phòng chẵn/lẻ độc lập với nhau theo kiểu luân phiên



415

6.4.2 Định dạng đĩa

- Trước khi đĩa có thể được sử dụng, mỗi mặt đĩa phải nhận được định dạng cấp thấp do phần mềm thực hiện.
- Đĩa được định dạng tạo thành hàng loạt các track, mỗi track chứa các sector

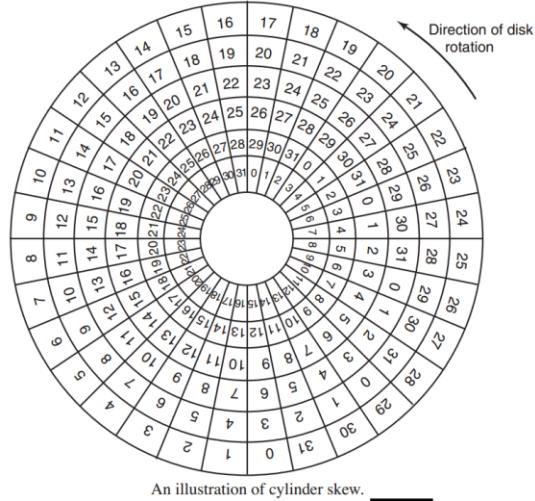


- Mỗi sector bao gồm các trường:
 - Preamble: bắt đầu với một số bit nhất định dùng để xác định vị trí bắt đầu của sector, cylinder number, sector number và các thông tin khác
 - Data: dữ liệu, kích thước được xác định bởi chương trình định dạng cấp thấp, thường là 512 byte
 - ECC: chứa thông tin được sử dụng để khôi phục lỗi, kích thước và nội dung của trường này phụ thuộc vào nhà sản xuất

416

Định dạng đĩa (tt)

- Sau khi định dạng mức thấp, dung lượng đĩa bị giảm, thường thấp hơn 20% so với dung lượng chưa được định dạng



417

6.4.3 Các thuật toán điều phối đĩa

- Tốc độ đọc/ghi đĩa phụ thuộc ba yếu tố:
 - Seek time: thời gian di chuyển đầu đọc tới track hay cylinder thích hợp
 - Rotational delay (rotational latency): thời gian quay đĩa đến sector cần đọc
 - Actual data transfer time: thời gian truyền dữ liệu thực tế.

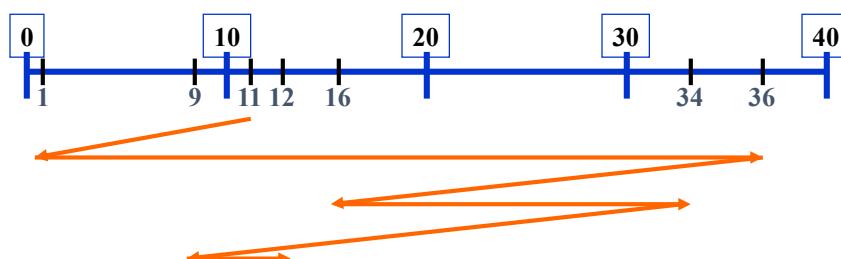


Các thuật toán điều phối đĩa (tt)

- Các thuật toán điều phối đĩa được dùng để xác định thứ tự di chuyển giữa các track trên đĩa một cách tối ưu
- Các thuật toán điển hình:
 - FCFS (First-Come, First-Served)
 - SSF (Shortest seek first)
 - SCAN (Elevator algorithm)
 - C-SCAN (One-way elevator)
 - ...

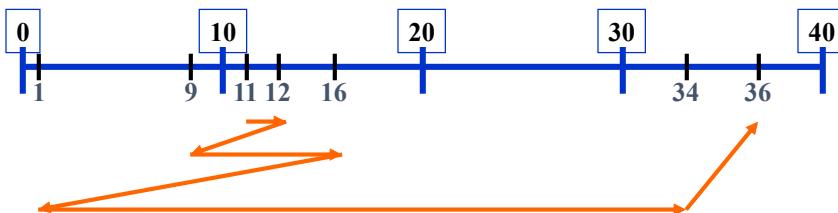
First in, First out (FIFO)

- Di chuyển theo thứ tự truy xuất, mỗi lần nhận 1 thao tác riêng lẻ
- Ví dụ: truy xuất 1, 36, 16, 34, 9, 12 và vị trí hiện tại ở Cylinder 11 (Di chuyển trung bình = $111/6 = 18.5$ cylinders)



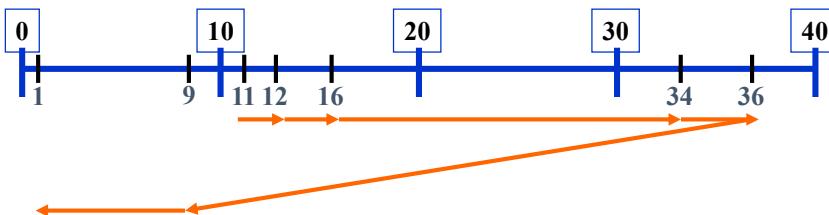
Shortest Seek First (SSF/SSTF)

- Truy xuất nào có di chuyển ít nhất thì thực hiện trước
- Ví dụ: 1, 36, 16, 34, 9, 12 và vị trí hiện tại ở Cylinder 11 : Di chuyển trung bình = $61/6 = 10.2$ cylinders



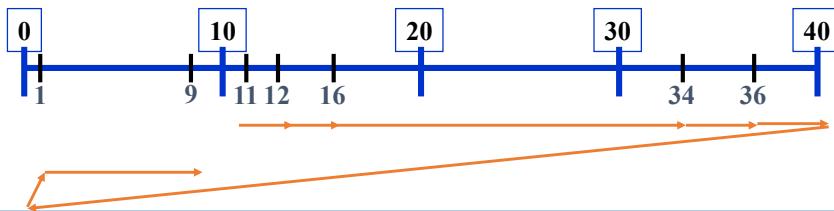
SCAN (Elevator Algorithm)

- Đầu đọc di chuyển từ 1 phía. Khi đến cylinder cuối được yêu cầu của phía đó, mới quay lại.
- Trong khi quay về, sẽ truy xuất cylinder nào được yêu cầu
- Ví dụ: truy xuất 1, 36, 16, 34, 9, 12 và vị trí hiện tại ở Cylinder 11 (Di chuyển trung bình = $60/6 = 10$ cylinders)
 - 11, 12, 16, 34, 36 end, 9, 1



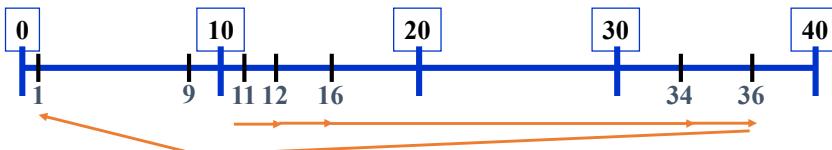
C-SCAN (Circular SCAN - One-way Elevator)

- Giống SCAN nhưng chỉ truy xuất cylinder theo 1 chiều
- Khi đến cylinder cuối cùng, đầu đọc quay về vị trí bắt đầu 0. Trong khi quay về, không phục vụ yêu cầu nào (1 chiều).
- Ví dụ: truy xuất 1, 36, 16, 34, 9, 12 và vị trí hiện tại ở Cylinder 11 (Di chuyển trung bình = $78/6 = 13$ cylinders)
 - $\rightarrow 11, 12, 16, 34, 36, 40$ end, 0, 1, 9



LOOK

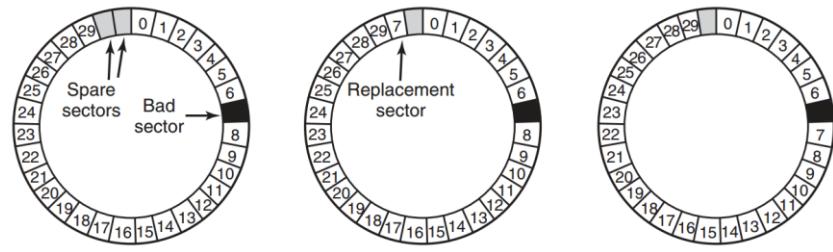
- Kết hợp giữa SCAN và C-SCAN
- Quét 1 phía, khi đến cylinder cuối được yêu cầu của phía đó rồi quay lại, phục vụ yêu cầu trên đường đi (không đụng hai biên). Ngoài ra, còn có C-LOOK.
- Ví dụ: truy xuất 1, 36, 16, 34, 9, 12 và vị trí hiện tại ở Cylinder 11 (Di chuyển trung bình = $68/6 = 11.3$ cylinders)
 - $\rightarrow 11, 12, 16, 34, 36, 9, 1$



424

6.4.4 Quản lý lỗi (Error Handling)

- Lỗi thường ở 2 dạng: tạm thời và lâu dài
- I/O Subsystem có thể phục hồi hiệu quả lỗi tạm thời
- Khi yêu cầu nhập xuất xảy ra lỗi → trả về mã lỗi



- (a) A disk track with a bad sector. (a)
 (b) Substituting a spare for the bad sector.
 (c) Shifting all the sectors to bypass the bad one