HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG
CƠ SỞ THÀNH PHỐ HỒ CHÍ MINH
KHOA CÔNG NGHỆ THÔNG TIN

GIÁO TRÌNH

HỆ ĐIỀU HÀNH

(OPERATING SYSTEM)

BIÊN SOẠN
NINH XUÂN HẢI - HUỲNH TRỌNG THƯA

NĂM 2008

LỜI MỞ ĐẦU

Hệ Điều Hành (Operating Systems) là một thành phần không thể thiếu trong một hệ thống máy tính. Một máy tính mặc dù đắt tiền, cấu hình cao nhưng nếu không có hệ điều hành thì hầu như không thể sử dụng được. Hệ điều hành điều khiển mọi hoạt động của máy tính, giúp việc sử dụng máy tính trở nên đơn giản, dễ dàng và hiệu qủa hơn rất nhiều. Do vậy môn học "Hệ điều hành" là môn học quan trọng và rất cần thiết trong chương trình đào tạo chuyên nghành tin học ở hệ cao đẳng và kỹ sư.

Giáo trình "Hệ điều hành" được biên soạn theo chương trình đào tạo chuyên nghành tin học ở hệ cao đẳng và kỹ sư của Bộ giáo dục và đào tạo. Giáo trình được chia thành 6 chương, chương 1, 2, 3, 4 do giảng viên Ninh Xuân Hải biên soạn, chương 5, 6 do giảng viên Huỳnh Trọng Thưa biên soạn. Tuy rằng chúng tôi đã có nhiều cố gắng trong công tác biên soạn nhưng chắc chắn giáo trình vẫn còn nhiều thiếu sót, nên rất mong được bạn đọc cũng như các đồng nghiệp đóng góp ý kiến để giáo trình ngày càng hoàn thiện, nhằm mục đích phục vụ tốt hơn cho việc dạy và học tin học đang ngày càng phát triển ở nước ta.

Mọi sự góp ý hoặc thắc mắc xin gởi về địa chỉ Email: hainx@ptithcm.edu.vn hoặc htthua@ptithcm.edu.vn.

Ngày 21 Tháng 11 Năm 2008

GV. biên soạn

Ninh Xuân Hải - Huỳnh Trọng Thưa

CHƯƠNG I

GIỚI THIỆU HỆ ĐIỀU HÀNH

Chương "GIỚI THIỆU VỀ HỆ ĐIỀU HÀNH " sẽ giới thiệu và giải thích các vấn đề sau:

1.1 Hệ điều hành là gì, các khái niệm của hệ điều hành.

1.2 Lịch sử phát triển của hệ điều hành

1.3 Các loại hệ điều hành

1.4 Các dịch vụ của hệ điều hành.

1.5 Cấu trúc của hệ điều hành

1.6 Nguyên lý thiết kế hệ điều hành

1.1 CÁC KHÁI NIỆM

1.1.1 Hệ điều hành là gì?

Hệ điều hành (operating systems) là chương trình đóng vai trò trung gian giữa người sử dụng và phần cứng của máy tính. Hệ điều hành che dấu sự phức tạp, đa dạng của phần cứng, giúp việc sử dụng máy tính trở nên đơn giản, hiệu quả. Nhiệm vụ của hệ điều hành là quản lý tài nguyên của máy tính, thực thi các chương trình ứng dụng, hỗ trợ các chức năng mạng, vv …

1.1.2 Các thành phần của một hệ thống máy tính

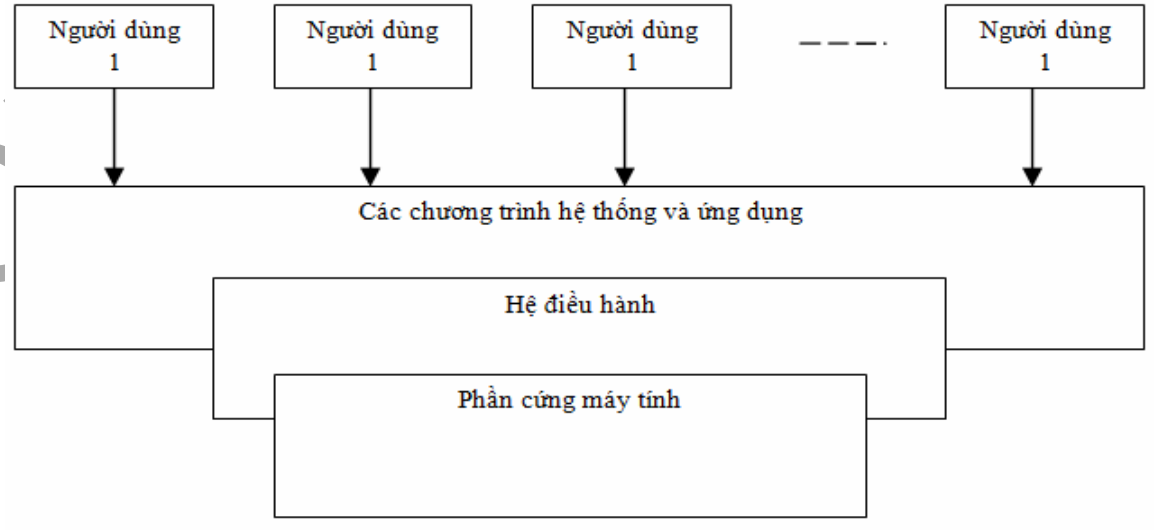
Một hệ thống máy tính được chia thành 4 thành phần sau: phần cứng, hệ điều hành, chương trình ứng dụng/chương trình hệ thống, người sử dụng.

+ Phần cứng (hardware) : CPU, bộ nhớ, các thiết bị nhập/xuất,…

+ Hệ điều hành (operating systems): điều khiển và phối hợp việc sử dụng phần cứng cho nhiều ứng dụng với nhiều người sử dụng khác nhau.

+ Chương trình ứng dụng và chương trình hệ thống (system and applications programs): là các chương trình giải quyết những vấn đề của người sử dụng như là chương trình dịch, hệ quản trị cơ sở dữ liệu, chương trình trò chơi, chương trình thương mại,…

+ Người sử dụng (user): người sử dụng hoặc máy tính.



Hình 1.1: Các thành phần của một hệ thống máy tính

1.1.3 Các thành phần của một hệ thống nhập/xuất

Một hệ thống nhập/xuất gồm ba thành phần sau:

+ Hệ thống bộ nhớ đệm (buffer-caching system)

+ Chương trình điều khiển thiết bị (Drivers for specific hardware devices).

+ Chương trình giao tiếp với chương trình điều khiển thiết bị (A general device-driver interface).

<table>
<tr><td colspan="2">Chương trình giao tiếp với chương trình điều khiển thiết bị</td></tr>
<tr><td>Chương trình điều khiển thiết bị</td><td>Hệ thống bộ nhớ đệm</td></tr>
</table>

Hình 1.2: Các thành phần của một hệ thống nhập/xuất

1.1.4 Các thành phần của hệ điều hành

Hệ điều hành gồm có ba thành phần sau:

+ Bộ cấp phát tài nguyên (Resource allocator): Quản lý và cấp phát tài nguyên.

+ Chương trình kiểm soát (Control program): Kiểm soát việc thực thi chương trình và kiểm soát hoạt động của các thiết bị nhập/xuất.

+ Phần nhân (Kernel): là chương trình "lõi" của hệ điều hành, được thực thi trước tiên và tồn tại trong bộ nhớ cho đến khi tắt máy (các chương trình khác gọi là chương trình ứng dụng).

<table>
<tr><td>Bộ cấp phát tài nguyên</td><td>Chương trình kiểm soát</td></tr>
<tr><td colspan="2">Phần nhân</td></tr>
</table>

Hình 1.3: Các thành phần của hệ điều hành

1.2 LỊCH SỬ PHÁT TRIỂN CỦA HỆ ĐIỀU HÀNH

+ Giai đoạn 1 (1945 – 1955): đã có máy tính lớn nhưng chưa có hệ điều hành.

+ Giai đoạn 2 (1956 – 1965): hệ thống xử lý theo lô (Batch systems)

+ Giai đoạn 3 (1966 – 1980): hệ thống xử lý đa chương (Multiprogramming systems) , hệ thống xử lý đa nhiệm (Multitasking systems).

+ Giai đoạn 4 (1981 - 2007): hệ thống đa xử lý (Multiprocessor systems), hệ thống xử lý phân tán (Distributed systems), hệ thống xử lý thời gian thực (Real-time systems), hệ thống nhúng (Embedded systems).

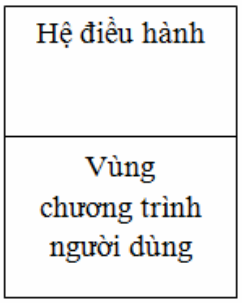
1.3 PHÂN LOẠI HỆ THỐNG MÁY TÍNH

Một hệ thống máy tính gồm hai phần là hệ điều hành và phần cứng tương ứng để thực thi hệ điều hành.

1.3.1 Hệ thống xử lý theo lô (Batch Systems)

Đây là hệ điều hành đầu tiên, thô sơ nhất. Đối với hệ điều hành này thì tại một thời điểm chỉ có một công việc trong bộ nhớ, khi thực hiện xong một công việc, công việc khác sẽ được tự động nạp vào và cho thực thi. Hệ điều hành có một chương trình, gọi là bộ giám sát, thường trú trong bộ nhớ chính, giám sát việc thực hiện dãy các công việc theo thứ tự và tự động.

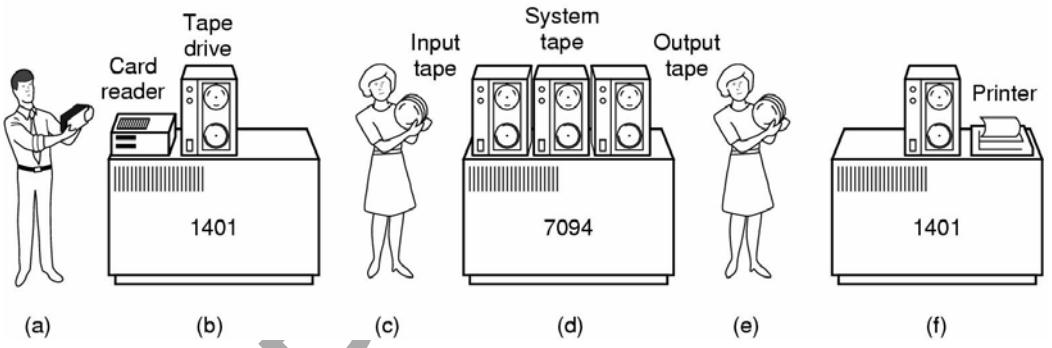
Cách bố trí bộ nhớ của hệ điều hành xử lý theo lô như sau: phần bộ nhớ ở địa chỉ thấp dành cho hệ điều hành, phần còn lại dành cho một chương trình của người dùng.



Hình 1.4: mô hình tổ chức bộ nhớ của hệ điều hành xử lý theo lô

Xem một ví dụ về cách thức làm việc với hệ thống xử lý theo lô:

- Lập trình viên mang phiếu ghi chương trình đến máy 1401
- Máy sẽ đọc chương trình từ phiếu và ghi chương trình vào băng từ
- Lập trình viên đem băng từ tới máy 7094 để thực hiện tính toán và kết qủa được ghi vào băng từ
- Lập trình viên đem băng từ chứa kết qủa tới máy 1402 để in

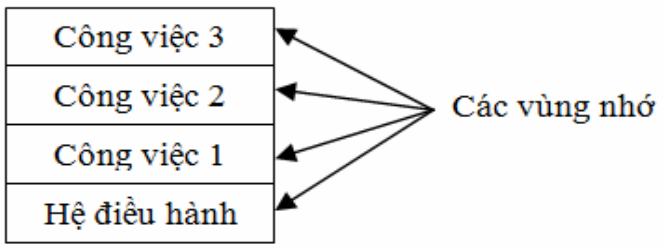


Hình 1.5: ví dụ về cách thức xử lý công việc với hệ điều hành xử lý theo lô

1.3.2 Hệ thống xử lý đa chương (MultiProgramming Systems)

Tại một thời điểm có nhiều công việc trong bộ nhớ và khi một công việc đang thực hiện, nếu có yêu cầu nhập/xuất thì CPU không nghỉ mà hệ điều hành sẽ chuyển sang thực hiện công việc khác.

Ví dụ trong bộ nhớ hiện có ba chương trình thực hiện ba công việc. Nếu công việc 1 yêu cầu nhập/xuất thì công việc 1 tạm ngừng, công việc 2 (hoặc công việc 3) sẽ được thực hiện. Khi thao tác nhập/xuất của công việc 1 xong thì công việc 1 sẽ được thực hiện tiếp, công việc 2 sẽ tạm ngừng,...



Hình 1.6: mô hình tổ chức bộ nhớ của hệ thống xử lý đa chương

*** Các chức năng của hệ điều hành trong hệ thống xử lý đa chương**

+ ***Lập lịch CPU (CPU scheduling)***: chọn một trong những công việc trong bộ nhớ cho thực thi (cho sử dụng CPU). Khi chọn cần tránh trường hợp một công việc chờ trong bộ nhớ quá lâu.

+ ***Quản lý bộ nhớ (Memory management)***: cần phải quản lý phần bộ nhớ nào đã cấp phát và cấp cho công việc nào (bộ nhớ cấp phát cho mỗi công việc phải riêng biệt), phần bộ nhớ nào chưa cấp, khi một công việc thực thi xong cần thu hồi phần bộ nhớ đã cấp cho công việc đó. Nếu một công việc truy xuất đến phần bộ nhớ đã cấp cho công việc khác thì phải ngăn cấm. Nếu bộ nhớ bị phân mảnh quá nhiều, cần dồn bộ nhớ, vv…

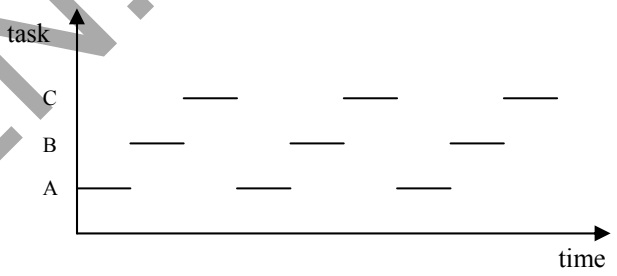
+ ***Cấp phát thiết bị (Allocation of devices)***: tình trạng thiết bị rảnh hay không rảnh, thiết bị đã cấp cho công việc nào, công việc nào cần đưa vào hàng đợi để chờ. Thiết bị nào có thể dùng chung và tối đa bao nhiêu công việc sử dụng chung thiết bị cùng lúc, thiết bị nào không thể dùng chung,… và phải tránh bị tắc nghẽn (các công việc chờ vô hạn để được cấp tài nguyên).

+ ***Cung cấp các hàm xử lý nhập/xuất (I/O routines)*:** Các hàm nhập/xuất sẽ che dấu sự phức tạp và đa dạng của các thiết bị nhập/xuất, quản lý việc sử dụng chung các thiết bị nhập/xuất.

1.3.3 Hệ thống xử lý đa nhiệm (Multitasking Systems)

Hệ thống xử lý đa nhiệm là hệ thống mở rộng của hệ thống xử lý đa chương. Đối với hệ điều hành trong hệ thống xử lý đa nhiệm, việc chuyển đổi công việc không chờ công việc đang thực thi có yêu cầu nhập/xuất, mà khi công việc đang thực thi hết thời gian qui định sử dụng CPU thì việc chuyển đổi công việc cũng sẽ xảy ra. Mỗi công việc được thực hiện luân phiên qua cơ chế chuyển đổi CPU, thời gian mỗi lần chuyển đổi diễn ra rất nhanh nên người sử dụng có cảm giác là các công việc đang được thi hành cùng lúc. Hệ thống xử lý đa nhiệm còn gọi là hệ thống chia xẻ thời gian (Time-Sharing Systems).

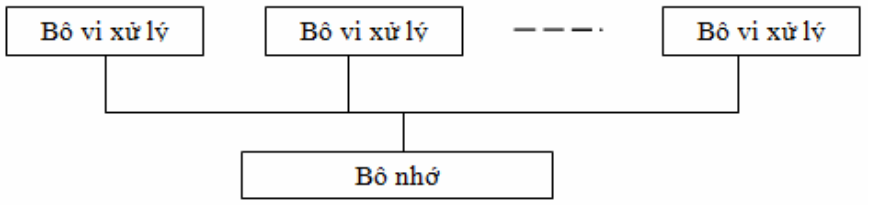
Ví dụ hệ thống có môt CPU và hiện có ba công việc A, B, C trong bộ nhớ. Ba công việc này sẽ được thực hiện luân phiên: công việc A thực hiện trong khoảng thời gian q (quantum) thì tạm ngừng, đến lượt công việc B thực hiện trong khoảng thời gian q, rồi đến lượt công việc C. Sau đó lại đến lượt A, … lặp lại việc thực thi các công việc cho đến khi tất cả các công việc hoàn tất.



Hình 1.7: các công việc A,B,C sử dụng cpu luân phiên trong hệ thống xử lý đa nhiệm

1.3.4 Hệ thống đa xử lý (Multiprocessor Systems)

Máy tính có nhiều bộ xử lý cùng chia xẻ hệ thống đường truyền dữ liệu, đồng hồ, bộ nhớ và các thiết bị ngoại vi. Mỗi CPU sẽ thực hiện một công việc và khi đó các công việc sẽ thực sự diễn ra đồng thời. Hệ thống đa xử lý còn gọi là hệ thống xử lý song song (Parallel Systems).



Hình 1.7: mô hình hệ thống đa xử lý: có nhiều cpu nhưng sử dụng chung bộ nhớ

*** Ưu điểm của hệ thống đa xử lý**

+ Sự hỏng hóc của một bộ xử lý sẽ không ảnh hưởng đến toàn bộ hệ thống.

+ Hệ thống sẽ thực hiện rất nhanh do thực hiện các công việc đồng thời trên các bộ xử lý khác nhau

+ Việc liên lạc giữa các công việc dễ dàng bằng cách sử dụng bộ nhớ dùng chung.

*** Phân loại hệ thống đa xử lý**

+ Hệ thống đa xử lý đối xứng (Symmetric MultiProcessing (SMP)): mỗi bộ xử lý chạy với một bản sao của hệ điều hành và các bộ xử lý là ngang cấp. Các hệ điều hành hiện nay đều hỗ trợ SMP.

+ Hệ thống đa xử lý bất đối xứng (Asymmetric multiprocessing): Có một bộ xử lý chính (master processor) kiểm soát, phân việc cho các bộ xử lý khác (slave processors).

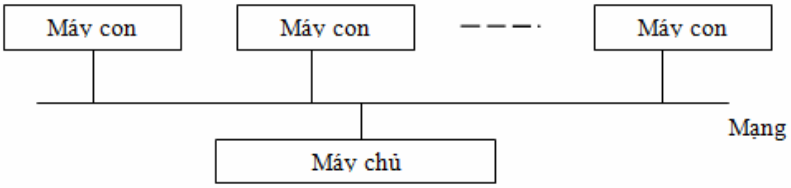
1.3.5 Hệ thống xử lý phân tán (Distributed Operating Systems)

Tương tự như hệ thống đa xử lý nhưng mỗi bộ xử lý có bộ nhớ riêng. Các bộ xử lý liên lạc với nhau thông qua các đường truyền dẫn mạng. Mạng LAN, WAN với hệ điều hành Windows, UNIX chính là các hệ thống xử lý phân tán.

*** Phân loại hệ thống xử lý phân tán:** có hai loại

+ Peer-to-peer: hệ thống mạng ngang hàng, các máy tính ngang cấp, không có máy nào đóng vai trò quản lý tài nguyên dùng chung.

+ Client-server: có một máy đóng vai trò quản lý các tài nguyên dùng chung gọi là máy server (máy chủ), các máy khác gọi là máy client (máy khách). Client muốn sử dụng tài nguyên dùng chung phải được server cấp quyền. Mô hình hệ thống client-server:



Hình 1.8: mô hình hệ thống xử lý phân tán

*** Ưu điểm của hệ thống xử lý phân tán**

+ Dùng chung tài nguyên: máy in, tập tin …

+ Tăng tốc độ tính toán: phân chia công việc để tính toán trên nhiều vị trí khác nhau

+ An toàn: Nếu một vị trí bị hỏng, các vị trí khác vẫn tiếp tục làm việc.

+ Truyền thông tin dễ dàng: download/upload file, gởi/nhận mail,…

1.3.6 Hệ thống xử lý thời gian thực (Real-Time Systems)

Hệ thống sẽ cho kết quả chính xác trong khoảng thời gian nhanh nhất. Hệ thống thường dùng cho những ứng dụng chuyên dụng như là hệ thống điều khiển trong công nghiệp.

*** Các loại hệ thống xử lý thời gian thực**

+ Hệ thống xử lý thời gian thực cứng (Hard real-time): các công việc được hoàn tất đúng thời điểm qui định.

+ Hệ thống xử lý thời gian thực mềm (Soft real-time): mỗi công việc có một độ ưu tiên riêng và sẽ được thi hành theo độ ưu tiên.

1.3.7 Hệ thống nhúng (Embedded Systems)

Hệ điều hành được nhúng trong các thiết bị gia dụng, các máy trò chơi,... Do các thiết bị gia dụng có bộ nhớ ít, bộ xử lý tốc độ thấp, kích thước màn hình nhỏ nên hệ điều hành này cần đơn giản, nhỏ gọn, có tính đặc trưng cho từng thiết bị. Ví dụ hệ điều hành dùng cho máy PDAs (Personal Digital Assistants), Mobil phones,… Hệ thống nhúng còn được gọi là hệ thống cầm tay (Handheld Systems).

1.4 CÁC DỊCH VỤ CỦA HỆ ĐIỀU HÀNH

Hệ điều hành thông thường cần cung cấp các dịch vụ sau:

- Quản lý tiến trình
- Quản lý bộ nhớ chính (RAM)
- Quản lý bộ nhớ phụ (DISK)
- Quản lý hệ thống nhập xuất
- Quản lý hệ thống tập tin
- Bảo vệ hệ thống
- Hệ thống dòng lệnh
- Quản lý mạng
- Các lời gọi hệ thống (system calls).

1.4.1 Dịch vụ quản lý tiến trình (Process Management)

Tiến trình là một chương trình đang thi hành. Trong bộ nhớ, tại một thời điểm có thể có nhiều tiến trình, một số tiến trình là của hệ điều hành, một số tiến trình là của người sử dụng. Khi tiến trình được tạo ra hoặc đang thi hành sẽ được hệ điều hành cung cấp các tài nguyên để tiến trình hoạt động như là CPU, bộ nhớ, tập tin, các thiết bị nhập/xuất… Khi tiến trình kết thúc, hệ điều hành sẽ thu hồi lại các tài nguyên đã cấp phát. Một tiến trình khi thực thi lại có thể tạo ra các tiến trình con và hình thành cây tiến trình.

*** Các chức năng của dịch vụ quản lý tiến trình**

+ Tạo và hủy các tiến trình của người sử dụng và của hệ điều hành.

+ Tạm ngưng và thực hiện lại một tiến trình.

+ Cung cấp cơ chế đồng bộ các tiến trình.

+ Cung cấp cơ chế liên lạc giữa các tiến trình.

+ Cung cấp cơ chế kiểm soát tắc nghẽn.

1.4.2 Dịch vụ quản lý bộ nhớ chính (Main Memory Management)

Tại một thời điểm, trong bộ nhớ chính có thể có nhiều tiến trình, hệ điều hành cần phải quản lý phần bộ nhớ đã cấp cho mỗi tiến trình để tránh xung đột.

*** Các chức năng của dịch vụ quản lý bộ nhớ chính**

+ Lưu giữ thông tin về các vị trí trong bộ nhớ đã sử dụng và tiến trình nào đang sử dụng.

+ Quyết định chọn tiến trình để nạp vào bộ nhớ chính khi bộ nhớ chính có chỗ trống.

+ Cấp phát bộ nhớ cho tiến trình và thu hồi bộ nhớ khi tiến trình thực thi xong.

1.4.3 Dịch vụ quản lý bộ nhớ phụ (Secondary Management)

Để lưu trữ dữ liệu lâu dài, dữ liệu cần lưu trên đĩa dạng tập tin, ngoài ra đĩa còn lưu giữ các tiến trình khi bộ nhớ RAM không còn đủ, vùng nhớ này gọi là bộ nhớ ảo.

*** Các chức năng của dịch vụ quản lý bộ nhớ phụ**

+ Quản lý vùng trống trên đĩa (Free space management)

+ Xác định vị trí cất giữ dữ liệu (Storage allocation).

+ Lập lịch cho đĩa (Disk scheduling).

1.4.4 Dịch vụ quản lý hệ thống nhập/xuất (I/O System Management)

Hệ điều hành cần che dấu những đặc thù của các thiết bị phần cứng, bằng cách cung cấp các chức năng xử lý nhập xuất đơn giản, không phụ thuộc vào chi tiết của mỗi loại thiết bị.

1.4.5 Dịch vụ quản lý hệ thống tập tin (File Management)

Máy tính có thể lưu trữ thông tin trong nhiều dạng thiết bị vật lý khác nhau như băng từ, đĩa từ, đĩa quang, ... Mỗi dạng có có khả năng lưu trữ, tốc độ truyền dữ liệu và cách truy xuất khác nhau. Hệ điều hành cần đồng nhất cách truy xuất hệ thống lưu trữ, định nghĩa một đơn vị lưu trữ là tập tin.

*** Các chức năng của dịch vụ quản lý hệ thống tập tin**

+ Hỗ trợ các thao tác trên tập tin và thư mục (tạo/xem/xoá/sao chép/di chuyển/đổi tên).

+ Ánh xạ tập tin trên hệ thống lưu trữ phụ.

+ Sao lưu tập tin trên các thiết bị lưu trữ.

1.4.6 Dịch vụ bảo vệ hệ thống (Protection System)

Hệ điều hành cần cung cấp cơ chế để đảm bảo rằng tài nguyên chỉ được truy xuất bởi những tiến trình có quyền. Ví dụ đảm bảo rằng tiến trình chỉ được thi hành trong phạm vi địa chỉ của nó hoặc đảm bảo rằng không có tiến trình nào độc chiếm CPU…

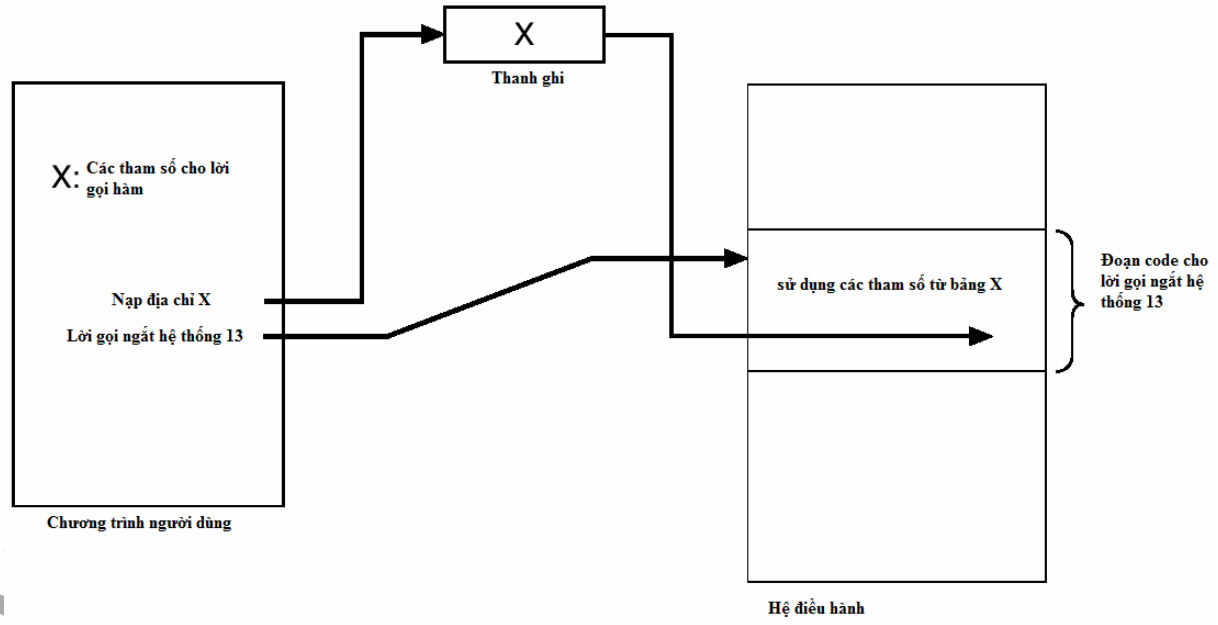
1.4.7 Lời gọi hệ thống (system call)

Lời gọi hệ thống là tập lệnh do hệ điều hành cung cấp dùng để giao tiếp giữa tiến trình của người dùng và hệ điều hành, lời gọi hệ thống còn gọi là ngắt. Các lời gọi hệ thống có thể được chia thành các loại như là tập lệnh quản lý tiến trình, tập lệnh quản lý tập tin, tập lệnh quản lý thiết bị, tập lệnh dùng để liên lạc giữa các tiến trình. Mỗi lời gọi hệ thống có một số hiệu duy nhất dùng để phân biệt lời gọi này với lời gọi khác. Các địa chỉ nơi chứa mã lệnh của các ngắt (lời gọi hệ thống) được lưu trong một bảng gọi là bảng vectơ ngắt.

Khi tiến trình dùng lời gọi hệ thống, cần cung cấp tham số cho lời gọi hệ thống. Có ba phương pháp mà tiến trình dùng để chuyển tham số cho hệ điều hành:

- Chuyển tham số vào thanh ghi
- Lưu trữ tham số trong một bảng trong bộ nhớ và ghi địa chỉ bảng vào thanh ghi
- Lưu trữ tham số vào stack và tham số được lấy ra bởi hệ điều hành .

Ví dụ chuyển địa chỉ bảng X (bảng chứa các tham số) vào thanh ghi, gọi ngắt 13. Ngắt 13 là lời gọi hệ thống do hệ điều hành cung cấp.



Hình 1.9: truyền tham số dạng bảng cho ngắt 13

Quản lý tiến trình	
Lời gọi hàm	Mô tả
Pid=fork()	Tạo một tiến trình con giống tiến trình cha
Pid=waitpid(pid, &statloc, options)	Đợi một tiến trình con kết thúc
Exit(status)	Kết thúc việc thực thi tiến trình và trả về trạng thái

Quản lý Tập tin	
Lời gọi hàm	Mô tả
Fd=open(file,how,…)	Mở một file để đọc, ghi hoặc cả hai
S=close(fd)	Đóng một file đã mở trước đó
N=read(fd,buffer,nbytes)	Đọc dữ liệu từ file vào vùng đệm
N= write(fd,buffer,nbytes)	Ghi dữ liệu từ buffer vào file
Position=lseek(fd,offset,whence)	Di chuyển con trỏ file
S=stat(name,&buf)	Lấy thông tin trạng thái của file

Quản lý Hệ thống file và thư mục	
Lời gọi hàm	Mô tả
S=mkdir(name,mode)	Tạo thư mục mới
S=rmdir(name)	Xóa thư mục rỗng
S=link(name1,name2)	Tạo một đối tượng mới name2 trỏ vào đối tượng name1 trước đó
S=unlink(name)	Xóa đối tượng thư mục
S=mount(special,name,flag)	Kích hoạt hệ thống file
S=unmount(special)	Ngừng kích hoạt hệ thống file

Hình 1.10: Một số lời gọi hệ thống

1.4.8 Hệ thống thông dịch dòng lệnh (Command-Interpreter System)

Là tập lệnh cơ bản cùng trình thông dịch lệnh để người sử dụng giao tiếp với hệ điều hành. Các lệnh cơ bản như lệnh quản lý tiến trình, quản lý nhập xuất, quản lý bộ nhớ chính, quản lý bộ nhớ phụ, quản lý tập tin và các lệnh bảo vệ hệ thống… Các lệnh trong hệ thống thông dịch dòng lệnh thực ra cũng sẽ gọi các các lời gọi hệ thống.

1.4.9 Quản lý mạng (Networking)

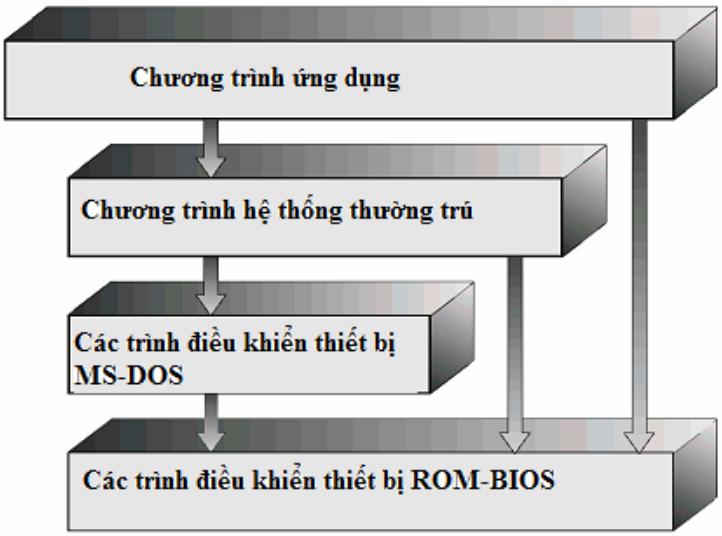
Cung cấp các chức năng phân quyền, chia xẻ tài nguyên mạng, liên lạc giữa các tiến trình trên mạng, …

1.5 CẤU TRÚC HỆ ĐIỀU HÀNH

1.5.1 Cấu trúc đơn giản

Hệ điều hành không được chia thành những lớp (phần) rõ rệt, một lớp có thể gọi hàm thuộc bất kỳ lớp nào khác. Hệ điều hành này đơn giản, dễ thiết kế, dễ cài đặt nhưng khó bảo vệ, khó mở rộng, và khó nâng cấp. Ví dụ hệ điều hành MSDOS là hệ điều hành có cấu trúc đơn giản: chương trình

ứng dụng có thể truy xuất trực tiếp các hàm nhập/xuất trong ROM BIOS để ghi trực tiếp lên màn hình hay bộ điều khiển đĩa.



Hình 1.11: Cấu trúc của hệ điều hành MS-DOS (cấu trúc đơn giản)

Hệ điều hành UNIX phiên bản đầu tiên cũng có cấu trúc đơn giản và được chia thành hai phần: phần system calls và phần kernel. Phần kernel cung cấp tất cả các dịch vụ của hệ điều hành. Các phần có thể gọi lẫn nhau.

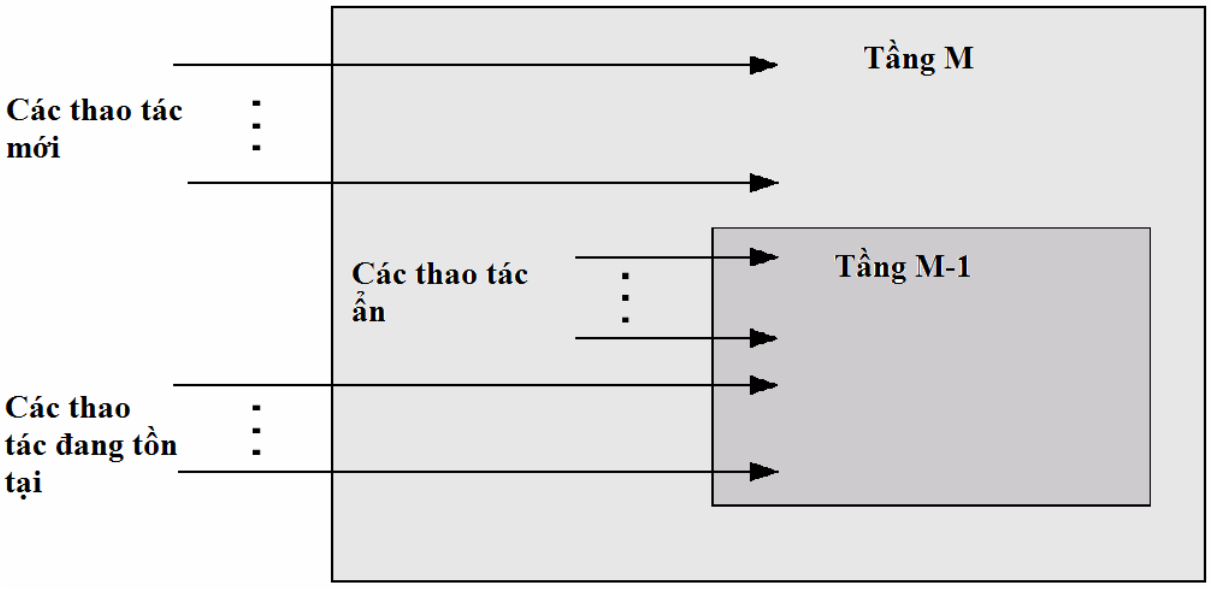
Các người dùng		
Các shells và lệnh Các trình biên dịch và thông dịch Các thư viện hệ thống		
Giao tiếp lời gọi hệ thống đến Kernel		
Hệ thống nhập xuất Trình điều khiển thiết bị đầu cuối	Hệ thống file Trình điều khiển đĩa và băng từ	Lập lịch CPU Phân trang Bộ nhớ ảo
Giao tiếp Kernel đến phần cứng		
Các bộ điều khiển đầu cuối Các đầu cuối	Các bộ điều khiển thiết bị Các thiết bị và băng từ	Các bộ điều khiển bộ nhớ Bộ nhớ vật lý

Hình 1.12: cấu trúc của hệ điều hành UNIX phiên bản đầu tiên (cấu trúc đơn giản)

1.5.2 Cấu trúc phân lớp

Hệ điều hành được chia thành nhiều lớp, mỗi lớp được xây dựng dựa vào những lớp thấp hơn. Lớp dưới cùng là phần cứng, lớp trên cùng là lớp giao tiếp với người sử dụng. Mỗi lớp chỉ sử dụng những hàm do lớp dưới cung cấp. Hạt nhân ở lớp kế lớp phần cứng, dùng các lệnh của phần cứng để tạo các lời gọi hệ thống.

Xem mô hình phân lớp ở hình 1.12: Lớp M thừa kế một số hàm của lớp M-1 và có thể có thêm một số hàm của riêng mình. Những hàm mà lớp M-1 đặt thuộc tính ẩn thì lớp M không được thừa kế.



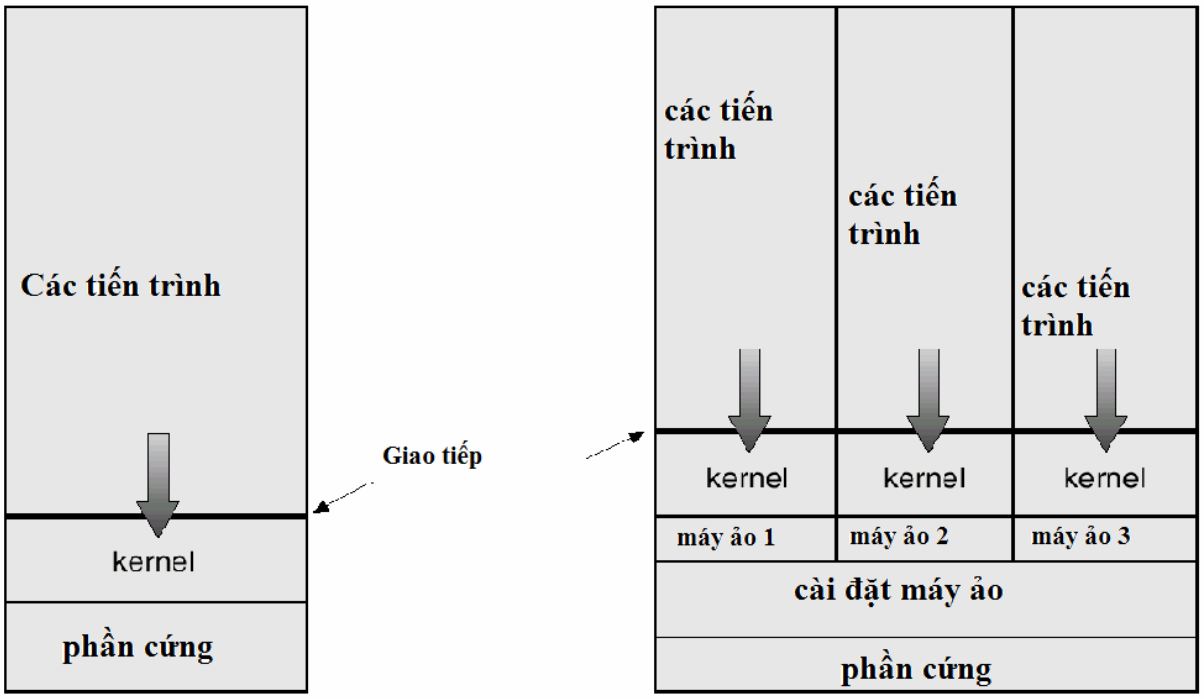
Hình 1.13: mô hình cấu trúc phân lớp

Tầng	Chức năng
5	Thao tác
4	Chương trình người dung
3	Quản lý Xuất/Nhập
2	Truyền thông Thao tác-Tiến trình
1	Quản lý bộ nhớ
0	Đa chương

Hình 1.14: cấu trúc phân lớp của hệ điều hành THE

1.5.3 Cấu trúc máy ảo

Với hệ điều hành máy ảo, một máy được giả lập thành nhiều máy, tài nguyên của hệ thống như là CPU, bộ nhớ, đĩa,… được chia xẻ để tạo các máy ảo. Mỗi máy ảo được cô lập với máy ảo khác nên tài nguyên dùng chung được bảo vệ nhưng cũng dẫn đến việc không được chia xẻ tài nguyên trực tiếp.



Hình 1.15: Mô hình cấu trúc máy ảo

1.5.4 Cấu trúc Client-Server

Hệ điều hành được chia thành nhiều phần (gọi là các tiến trình server), mỗi tiến trình thực hiện một dịch vụ như là dịch vụ quản lý tập tin, quản lý tiến trình, quản lý bộ nhớ, … Các tiến trình yêu cầu (gọi là tiến trình client) sẽ gửi yêu cầu đến một tiến trình server, tiến trình server thực hiện và gửi kết quả trở lại cho tiến trình client. Hạt nhân chỉ có nhiệm vụ kiểm soát quá trình liên lạc giữa các tiến trình client và server.

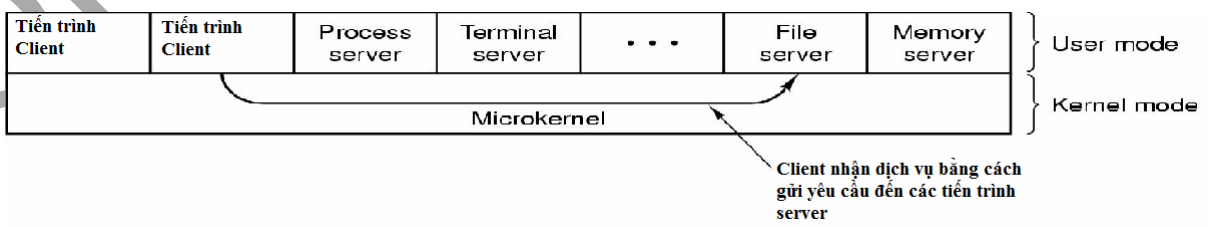
*** Ưu điểm của cấu trúc client-server**

+ Hạt nhân rất nhỏ, chỉ gồm các lệnh cơ bản, nên dễ bảo vệ, dễ nâng cấp.

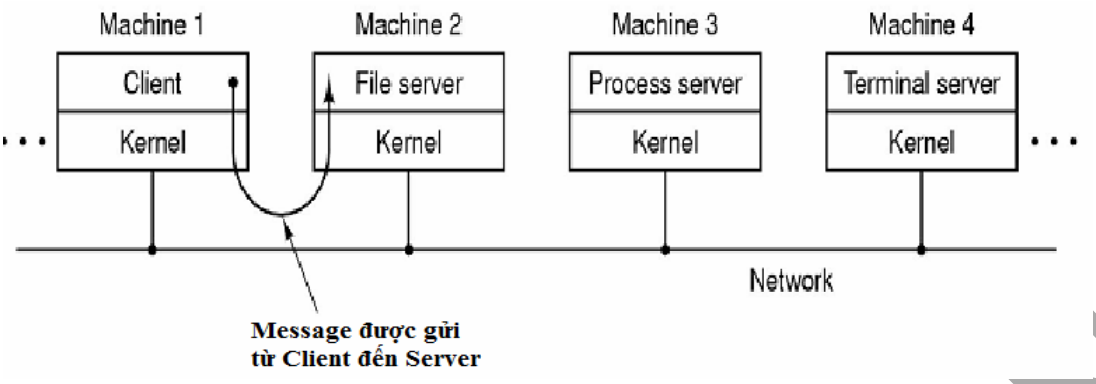
+ Mỗi dịch vụ của hệ điều hành do một tiến trình server đảm nhận, các tiến trình này độc lập với nhau nên khi một tiến trình server bị lỗi, hệ thống vẫn hoạt động.

+ Các tiến trình server được thực hiện ở chế độ người dùng (user-mode), không phải ở chế độ hạt nhân (kernel-mode), nên không truy xuất trực tiếp phần cứng.

+ Cấu trúc client-server rất thích hợp với mô hình hệ thống phân tán. Các tiến trình server có thể thực thi ở các máy khác nhau.



Hình 1.17: Mô hình hệ điều hành client-server trên một máy



Hình 1.18: Mô hình hệ điều hành client-server trên nhiều máy

1. 6 NGUYÊN LÝ THIẾT KẾ HỆ ĐIỀU HÀNH

+ Hệ điều hành cần dễ viết, dễ sửa lỗi, dễ nâng cấp (nên viết hệ điều hành bằng ngôn ngữ cấp cao vì dễ viết và dễ sửa lỗi hơn là viết bằng ngôn ngữ assembly).

+ Hệ điều hành cần dễ cài đặt, dễ bảo trì, không có lỗi và hiệu qủa.

+ Hệ điều hành cần dễ sử dụng, dễ học, an toàn, có độ tin cậy cao và thực hiện nhanh.

+ Hệ điều hành cần có tính khả chuyển cao (thực hiện được trên một nhóm các phần cứng khác nhau).

+ Hệ điều hành cần có chương trình SYSGEN (System Generation) thu thập thông tin liên quan đến phần cứng để thiết lập cấu hình hệ điều hành cho phù hợp với mỗi máy tính.

TÓM TẮT

Một hệ thống máy tính gồm có phần cứng, hệ điều hành và các chương trình ứng dụng. Hệ điều hành giúp cho việc sử dụng máy tính hiệu quả, đơn giản hơn. Hệ điều hành có nhiều loại nhưng thông dụng là loại hệ điều hành đa nhiệm, phân tán. Hệ điều hành cung cấp các dịch vụ cơ bản như dịch vụ quản lý tiến trình, dịch vụ quản lý bộ nhớ, dịch vụ quản lý tập tin, dịch vụ quản lý nhập/xuất,… và một tập các lời gọi hệ thống (ngắt). Hệ điều hành cần thiết kế sao cho dễ sửa lỗi, dễ cài đặt, dễ bảo trì, không có lỗi, dễ sử dụng, dễ học, độ tin cậy cao, thực hiện nhanh và có tính khả chuyển cao.

CÂU HỎI – BÀI TẬP

1. Nêu mục đích chung của hệ điều hành
2. Phân biệt hệ thống đa chương và hệ thống đa nhiệm
3. Nêu các vấn đề mà hệ thống đa chương/đa nhiệm cần giải quyết
3. Phân biệt hệ thống đa nhiệm và hệ thống đa xử lý
4. Phân biệt hệ thống đa xử lý và hệ thống xử lý phân tán
5. Nêu mục đích của hệ thống bộ nhớ đệm (buffer-caching system) trong hệ thống nhập/xuất
6. Chương trình điều khiển thiết bị (Drivers for specific hardware devices) do hệ điều hành cung cấp hay do hãng sản xuất thiết bị cung cấp?
7. Chương trình giao tiếp với chương trình điều khiển thiết bị (general device-driver interface) do hệ điều hành hay do hãng sản xuất hay do ngôn ngữ lập trình hay do người lập trình cung cấp?

8. Phần nhân (kernel) của hệ điều hành MS-DOS gồm những chương trình nào?

9. Nêu khuyết điểm của hệ điều hành có cấu trúc đơn giản.

TÀI LIỆU THAM KHẢO

[1]. Gary J. Nutt, University of Colorado. Centralized And Distributed Operating Systems. Second Edition, 2000.

[2]. Robert Switzer. Operating Systems, A Practical Approach. Prentice-Hall International, Inc. 1993.

[3]. Andrew S. Tanenbaum. Modern Operating Systems. Prentice-Hall International, Inc. Second Edition, 2001.

[4]. Abraham Silberschatz & Peter Baer Galvin. Operating System concepts. John Wiley & Sons, Inc. Fifth Edition, 1999.

[5]. H. M. Deitel. Operating Systems. Addison-Wesley Inc. Second Edition, 1999.

[6] Cẩm nang lập trình hệ thống cho máy vi tính IBM-PC tập 1 và 2, tác giả Michael Tischer.

[7]. Trần Hạnh Nhi & Lê Khắc Nhiên Ân & Hoàng Kiếm. Giáo trình hệ điều hành (tập 1 & 2). ĐHKHTN 2000.

CHƯƠNG 2

QUẢN LÝ NHẬP/XUẤT VÀ QUẢN LÝ HỆ THỐNG TẬP TIN

Chương "QUẢN LÝ NHẬP/XUẤT VÀ QUẢN LÝ HỆ THỐNG TẬP TIN" sẽ giới thiệu và giải thích các vấn đề sau:

2.1. Quản lý nhập/xuất

2.1.1 Phân loại và đặc tính của thiết bị nhập/xuất

2.1.2 Bộ điều khiển thiết bị nhập/xuất

2.1.3 Các chương trình thực hiện nhập/xuất và tổ chức hệ thống nhập/xuất

2.1.4 Cơ chế nhập/xuất và cơ chế DMA

2.1.5 Các thuật toán lập lịch di chuyển đầu đọc

2.1.6 Hệ số đan xen và ram disk

2.2 Quản lý hệ thống tập tin

2.2.1 Các khái niệm về đĩa cứng, tập tin, thư mục, bảng thư mục

2.2.2 Các phương pháp cài đặt hệ thống tập tin.

2.2.3 Phương pháp quản lý danh sách các khối trống

2.2.4 Phương pháp quản lý sự an toàn của hệ thống tập tin

2.2.5 Giới thiệu một số hệ thống tập tin: MSDOS/Windows, UNIX.

2.1. QUẢN LÝ NHẬP/XUẤT

2.1.1 Phân loại và đặc tính của thiết bị nhập/xuất

*** Phân loại thiết bị nhập/xuất:**

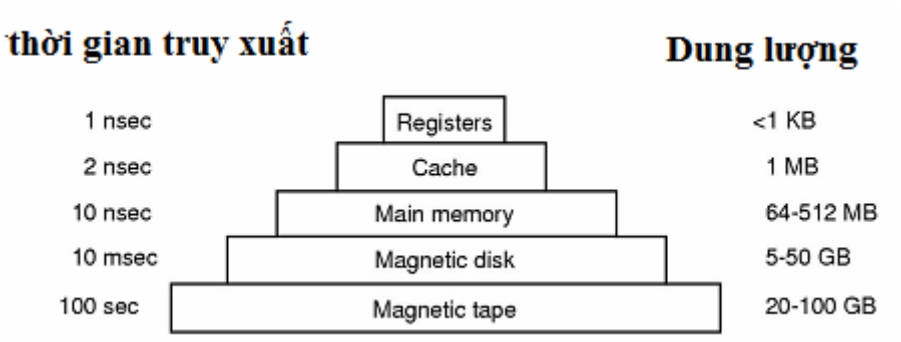
+ Thiết bị khối: thông tin được đọc/ghi theo từng khối có kích thước cố định và có địa chỉ xác định, ví dụ đĩa là một thiết bị khối.

+ Thiết bị tuần tự: thông tin được gửi/nhận theo dãy tuần tự các bit, không có địa chỉ, ví dụ màn hình, bàn phím, máy in, card mạng, chuột là thiết bị tuần tự.

+ Thiết bị khác: có một số các thiết bị không phù hợp với hai loại trên, ví dụ đồng hồ không là thiết bị khối, cũng không là thiết bị tuần tự.

*** Đặc tính của thiết bị nhập/xuất:**

+ Tốc độ truyền dữ liệu: ví dụ bàn phím : 0.01 KB/s, chuột 0.02 KB/s ...
+ Dung lượng lưu trữ, thời gian truy xuất một đơn vị dữ liệu.
+ Công dụng: dùng để nhập hay xuất
+ Đơn vị truyền dữ liệu: truyền theo khối hoặc ký tự
+ Biểu diễn dữ liệu: điều này tùy thuộc vào từng thiết bị cụ thể.
+ Tình trạng lỗi: nguyên nhân gây ra lỗi, cách mà thiết bị báo lỗi...

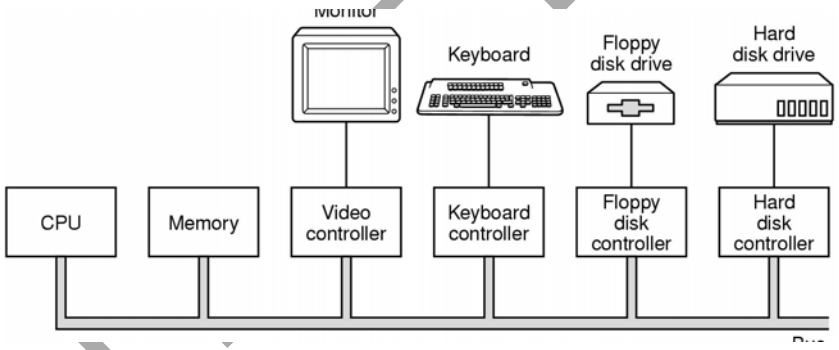


Hình 2.1: thời gian truy xuất và dung lượng của một số thiết bị nhập/xuất

2.1.2 Bộ điều khiển thiết bị nhập/xuất (I/O controller)

Là phần cứng điều khiển trực tiếp thiết bị nhập/xuất, CPU không thể truy xuất trực tiếp thiết bị nhập/xuất mà phải thông qua bộ điều khiển thiết bị, dùng hệ thống đường truyền gọi là bus. Ví dụ bộ điều khiển màn hình đọc các ký tự cần hiển thị trong bộ nhớ và điều khiển các tia của CRT (ống phóng điện tử của màn hình) để xuất ký tự trên màn hình. Thiết bị nhập/xuất và bộ điều khiển phải tuân theo cùng chuẩn giao tiếp như chuẩn ANSI, IEEE hay ISO…

Bộ điều khiển thiết bị nhập/xuất có thể điều khiển được nhiều thiết bị, ví dụ một bộ điều khiển màn hình (video controller) có thể điều khiển nhiều màn hình.



Hình 2.2: CPU truy xuất các thiết bị nhập/xuất thông qua bộ điều khiển thiết bị

Mỗi bộ điều khiển có một số thanh ghi để liên lạc với CPU, các thanh ghi này được gán một địa chỉ xác định như là một phần của bộ nhớ chính, gọi là ánh xạ bộ nhớ nhập/xuất. Ví dụ:

Bộ điều khiển nhập/xuất	Địa chỉ nhập/xuất (địa chỉ của các thanh ghi)	Vectơ ngắt
Đồng hồ	040 - 043	8
Bàn phím	060 - 063	9
RS232 phụ	2F8 - 2FF	11
Đĩa cứng	320 - 32F	13

Máy in	378 - 37F	15
Màn hình mono	380 - 3BF	-
Màn hình màu	3D0 - 3DF	-
Đĩa mềm	3F0 - 3F7	14
RS232 chính	3F8 - 3FF	12

Hình 2.3: bảng địa chỉ các thanh ghi của một số bộ điều khiển nhập/xuất.

CPU thực hiện nhập/xuất bằng cách ghi lệnh, cùng các tham số lên các thanh ghi của bộ điều khiển, sau đó CPU sẽ thực hiện công việc khác. Khi bộ điều khiển thực hiện xong, sẽ phát sinh một ngắt để báo hiệu cho CPU biết và đến lấy kết quả (kết quả cũng được bộ điều khiển lưu trong các thanh ghi).

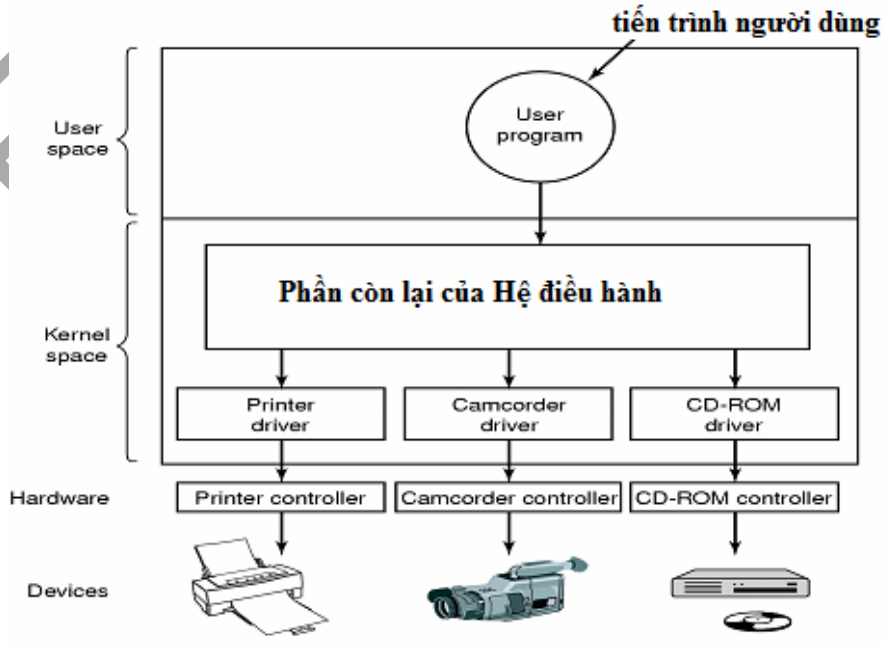
2.1.3 Các chương trình thực hiện nhập/xuất và tổ chức hệ thống nhập/xuất

*** Các chương trình thực hiện nhập/xuất:**

+ Chương trình nhập/xuất của người dùng (user program): thực hiện các lời gọi đến chương trình nhập/xuất độc lập thiết bị.

+ Chương trình nhập/xuất độc lập thiết bị: còn gọi là lời gọi hệ thống nhập/xuất hoặc ngắt nhập/xuất, do hệ điều hành cung cấp, chương trình nhập/xuất độc lập thiết bị cung cấp một giao tiếp đồng nhất cho chương trình nhập/xuất của người dùng.

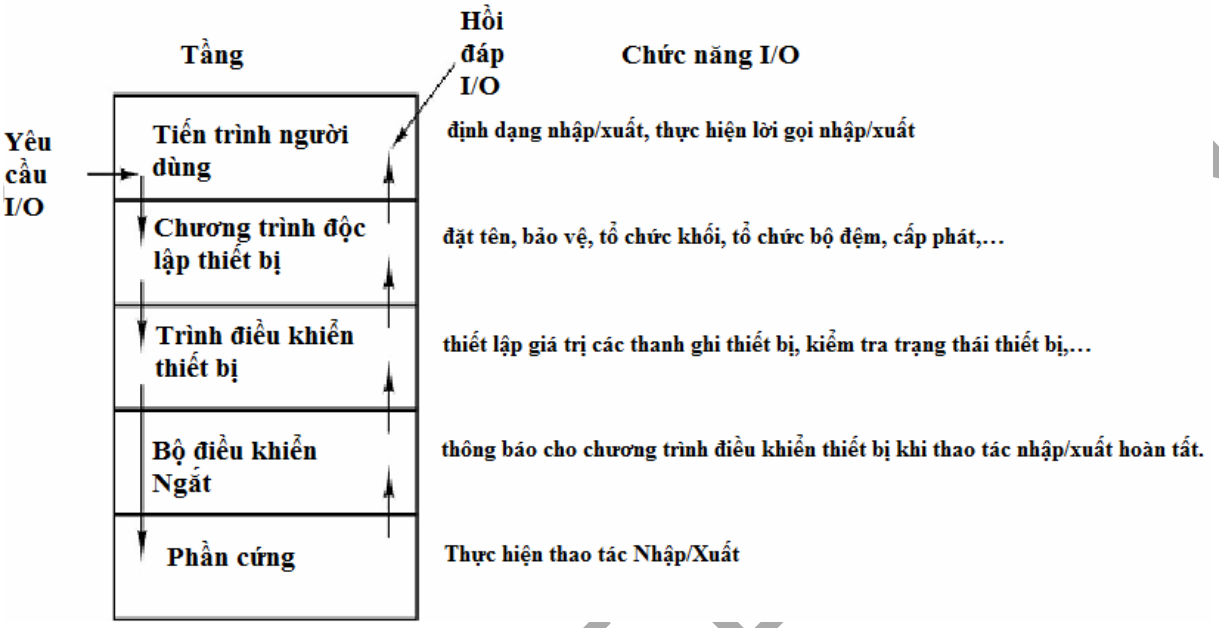
+ Chương trình điều khiển thiết bị (Device drivers): do hệ điều hành hoặc nhà sản xuất thiết bị cung cấp, chương trình này phụ thuộc vào thiết bị, sẽ nhận những yêu cầu nhập/xuất của chương trình nhập/xuất độc lập thiết bị. Nếu device driver đang bận, yêu cầu đó sẽ được đưa vào hàng đợi, ngược lại nó sẽ thực hiện ngay yêu cầu, bằng cách chuyển lệnh vào thanh ghi của bộ điều khiển thiết bị (I/O controller).



Hình 2.4: Sự giao tiếp giữa các chương trình thực hiện nhập/xuất

*** Tổ chức hệ thống nhập/xuất**

Hệ thống quản lý nhập/xuất được phân chia thành 5 lớp là: tiến trình người dùng (user processes), chương trình nhập/xuất độc lập thiết bị (device-independent software), chương trình điều khiển thiết bị (device drivers), chương trình kiểm soát ngắt (interrupt handlers), phần cứng (hardware). Mỗi lớp có chức năng riêng và có thể giao tiếp với lớp khác.



Hình 2.5: mô hình phân lớp của hệ thống quản lý nhập/xuất

+ Tiến trình người dùng (user processes): định dạng nhập/xuất, thực hiện lời gọi nhập/xuất.

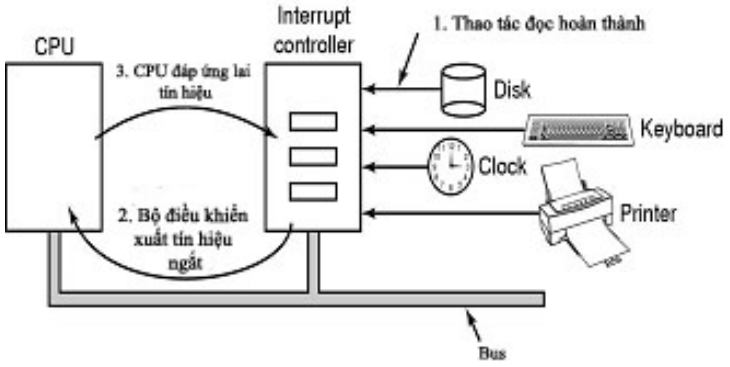
+ Chương trình nhập/xuất độc lập thiết bị (Device-independent software): đặt tên, bảo vệ, tổ chức khối, tổ chức bộ đệm, cấp phát,…

+ Chương trình điều khiển thiết bị (Device driver): thiết lập giá trị các thanh ghi thiết bị, kiểm tra trạng thái thiết bị,…

+ Chương trình kiểm soát ngắt (Interrupt handlers): thông báo cho chương trình điều khiển thiết bị khi thao tác nhập/xuất hoàn tất.

+ Phần cứng nhập/xuất (I/O Hardware): thực hiện thao tác nhập/xuất

Ví dụ tiến trình người dùng (user processes) muốn đọc một khối dữ liệu trên đĩa, sẽ gởi yêu cầu nhập/xuất đến chương trình nhập/xuất độc lập thiết bị (device-independent software), chương trình này sẽ tìm kiếm khối trong bộ đệm nhập/xuất, nếu khối cần đọc chưa có trong bộ đệm, nó sẽ gọi chương trình điều khiển thiết bị (device driver). Chương trình điều khiển thiết bị gửi yêu cầu đến đĩa cứng và tiến trình người dùng sẽ tạm ngưng cho đến khi thao tác đọc đĩa hoàn tất, đĩa sẽ phát sinh một ngắt thông báo đã đọc xong, gởi tín hiệu ngắt cho chương trình kiểm soát ngắt. Chương trình kiểm soát ngắt ghi nhận trạng thái của thiết bị và đánh thức tiến trình của người dùng để tiếp tục thực hiện.



Hình 2.6: khi thao tác đọc đĩa hoàn tất, bộ điều khiển đĩa sẽ phát sinh một ngắt.

2.1.4 Cơ chế nhập/xuất và cơ chế DMA

*** Cơ chế nhập/xuất:**

- Bộ xử lý phát sinh một lệnh I/O đến các thiết bị I/O, sau đó chờ cho đến khi thao tác I/O hoàn tất rồi mới tiếp tục xử lý, hoặc:

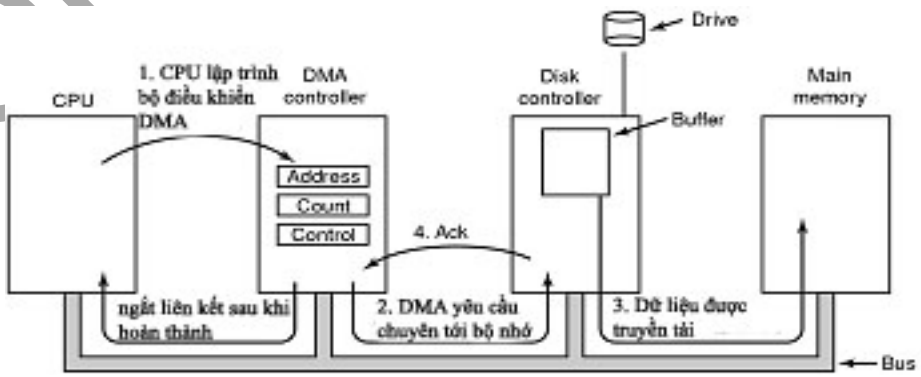
- Bộ xử lý phát sinh một lệnh I/O đến các thiết bị I/O, sau đó tiếp tục việc xử lý cho tới khi nhận được một ngắt từ thiết bị I/O báo là đã hoàn tất nhập/xuất, bộ xử lý tạm ngưng việc xử lý hiện tại để chuyển qua xử lý ngắt, hoặc:

- Sử dụng cơ chế DMA

*** Cơ chế DMA (Direct Memory Access):**

Xét quá trình đọc đĩa, CPU gửi cho bộ điều khiển đĩa (disk controller) lệnh đọc đĩa và các thông số như địa chỉ trên đĩa của khối, địa chỉ trong bộ nhớ RAM nơi sẽ cất khối đọc được, số byte cần đọc, sau đó CPU tiếp tục xử lý công việc khác. Bộ điều khiển sẽ đọc khối trên đĩa, từng bit cho tới khi toàn bộ khối được đưa vào buffer của bộ điều khiển (local buffer). Tiếp theo bộ điều khiển phát ra một ngắt để báo cho CPU biết là thao tác đọc đã hoàn tất. CPU đến lấy dữ liệu trong buffer chuyển vào bộ nhớ chính (RAM) bằng cách tạo một vòng lặp đọc lần lượt từng byte. Thao tác này làm lãng phí thời gian của CPU.

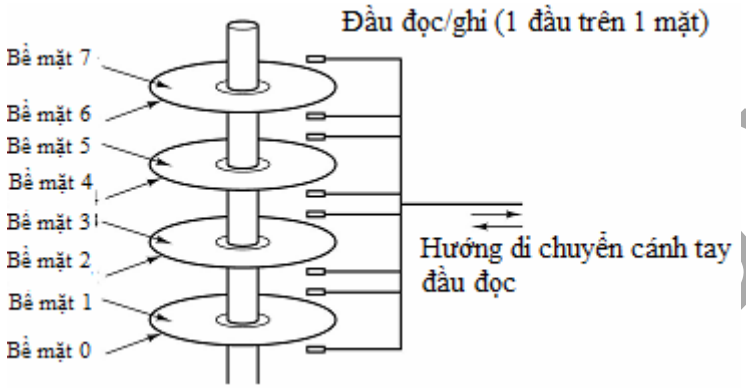
Để tối ưu, bộ điều khiển thường được cung cấp thêm khả năng truy xuất bộ nhớ trực tiếp (DMA). Nghĩa là sau khi bộ điều khiển đã đọc toàn bộ dữ liệu từ thiết bị vào buffer của nó, bộ điều khiển chuyển byte đầu tiên vào bộ nhớ chính tại địa chỉ được mô tả bởi địa chỉ bộ nhớ DMA. Sau đó nó tăng địa chỉ DMA và giảm số bytes phải chuyển. Quá trình này lặp lại cho tới khi số bytes phải chuyển bằng 0, và bộ điều khiển tạo một ngắt. Như vậy bộ điều khiển tự chuyển khối vào trong bộ nhớ chính.



Hình 2.7: Cơ chế DMA

2.1.5 Các thuật toán lập lịch di chuyển đầu đọc

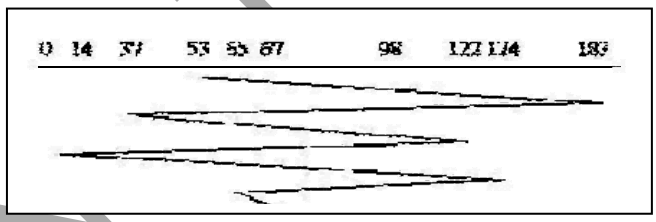
Để truy xuất các khối trên đĩa, trước tiên phải di chuyển đầu đọc đến track thích hợp, thao tác này gọi là seek và thời gian để hoàn tất thao tác này gọi là seek time. Một khi đã đến đúng track, còn phải chờ cho đến khi khối cần thiết đến dưới đầu đọc, thời gian chờ này gọi là latency time. Cuối cùng là chuyển dữ liệu từ đĩa vào bộ nhớ chính, thời gian này gọi là transfer time. Tổng thời gian cho dịch vụ đĩa chính là tổng của ba khoảng thời gian trên (seek time + latency time + transfer time). Trong đó seek time và latency time là mất nhiều thời gian nhất, do đó để giảm thiểu thời gian truy xuất, hệ điều hành cần đưa ra các thuật toán lập lịch dời đầu đọc sao cho tối ưu.



Hình 2.8: mô hình đĩa cứng

2.1.5.1 Thuật toán FCFS (first-come, first-served)

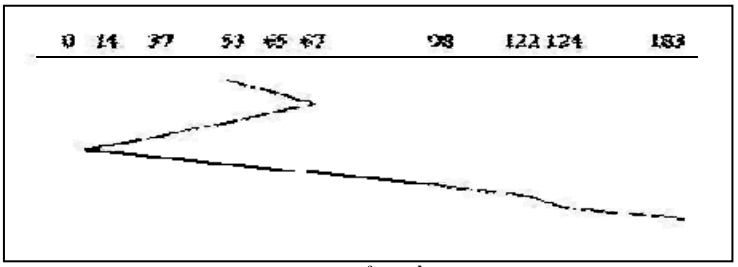
Thuật toán sẽ dời đầu đọc theo thứ tự đúng với thứ tự các khối cần đọc, thuật toán dễ lập trình nhưng chưa tốt. Ví dụ cần phải đọc các khối theo thứ tự như sau: 98, 183, 37, 122, 14, 124, 65, và 67. Giả sử hiện tại đầu đọc đang ở vị trí 53, thuật toán sẽ dời đầu đọc lần lượt đi qua các khối 53, 98, 183, 37, 122, 14, 124, 65, và 67



Hình 2.9: Các bước di chuyển đầu đọc theo thuật toán FCFS

2.1.5.2 Thuật toán SSTF (shortest-seek-time-first)

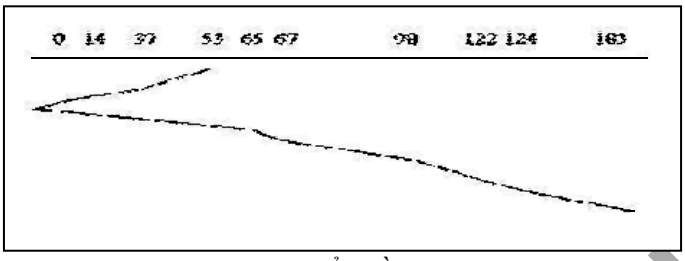
Thuật toán sẽ di chuyển đầu đọc lần lượt đến các khối cần đọc theo vị trí gần với vị trí hiện hành của đầu đọc nhất. Ví dụ cần đọc các khối như sau: 98, 183, 37, 122, 14, 124, 65, và 67. Giả sử hiện tại đầu đọc đang ở vị trí 53, thuật toán sẽ dời đầu đọc lần lượt đi qua các khối 53, 65, 67, 37, 14, 98, 122, 124 và 183.



Hình 2.10: Các bước di chuyển đầu đọc theo thuật toán SSTF

2.1.5.3 Thuật toán SCAN

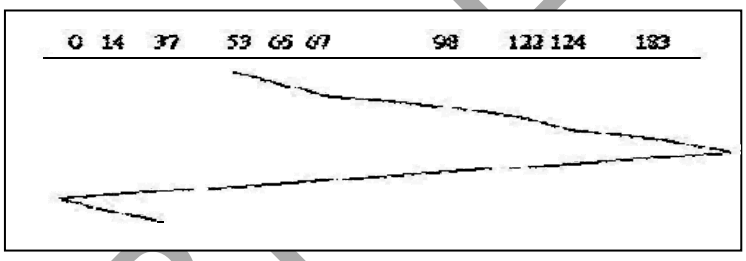
Thuật toán sẽ di chuyển đầu đọc về một phía của đĩa và từ đó di chuyển qua phía kia. Ví dụ cần đọc các khối như sau: 98, 183, 37, 122, 14, 124, 65, và 67. Giả sử hiện tại đầu đọc đang ở vị trí 53, đầu đọc lần lượt đi qua các khối: 53, 37, 14, 65, 67, 98, 122, 124 và 183



Hình 2.11: Các bước di chuyển đầu đọc theo thuật toán SCAN

2.1.5.4 Thuật toán C-SCAN

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



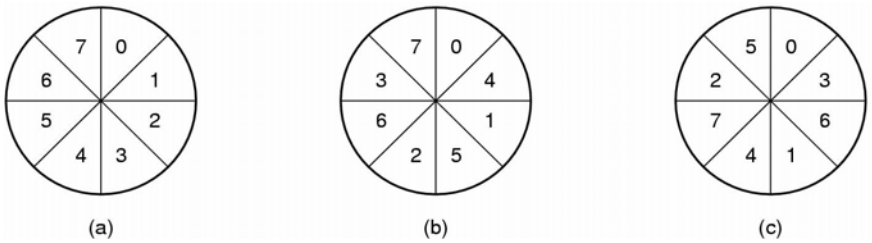
Hình 2.12: Các bước di chuyển đầu đọc theo thuật toán C-SCAN

Thuật toán SCAN và C-SCAN thích hợp cho những hệ thống phải truy xuất dữ liệu khối lượng lớn. Thuật toán lập lịch phụ thuộc vào số khối và kiểu khối cần truy xuất, ví dụ nếu số khối cần truy xuất là liên tục thì FCFS là thuật toán tốt.

2.1.6 Hệ số đan xen và Ram Disks

*** Hệ số đan xen (Interleave)**

Bộ điều khiển đĩa phải thực hiện hai chức năng là đọc/ghi dữ liệu và chuyển dữ liệu vào hệ thống. Để đồng bộ hai chức năng này, các sector được đánh số sao cho các sector có số hiệu liên tiếp nhau không nằm kế bên nhau mà có một khoảng cách, khoảng cách này được xác định bởi quá trình format đĩa và gọi là hệ số đan xen.



Hình 2.13: (a) interleave=0, (b) interleave=1, (c) interleave=2

Ví dụ giả sử hệ thống có 17 sector/track, và interleave = 4 thì các sector được bố trí theo thứ tự như sau: 1, 14, 10, 6, 2, 15, 11, 7, 3, 16, 12, 8, 4, 17, 13, 9, 5

Cách đọc lần lượt như sau :

Lần 1: 1, 2, 3, 4, 5

Lần 2: 6, 7, 8, 9, 10

Lần 3: 11, 12, 13, 14, 15

Lần 4: 16, 17

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

*** Ram disk:**

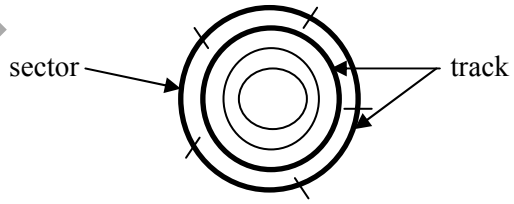
Hệ điều hành có thể dùng một phần của bộ nhớ chính để lưu trữ các khối đĩa, phần bộ nhớ này gọi là Ram Disk . Ram Disk cũng được chia làm nhiều khối, mỗi khối có kích thước bằng kích thước của khối trên đĩa. Khi driver nhận được lệnh đọc/ghi khối, sẽ tìm trong bộ nhớ Ram Disk vị trí của khối, và thực hiện việc đọc/ ghi trong đó thay vì từ đĩa . RAM disk có ưu điểm là cho phép truy xuất nhanh, không phải chờ quay hay tìm kiếm, thích hợp cho việc lưu trữ những chương trình hay dữ liệu được truy xuất thường xuyên.

2.2 QUẢN LÝ HỆ THỐNG TẬP TIN

2.2.1 Đĩa cứng, tập tin, thư mục

2.2.1.1 Đĩa cứng (hard disk)

Đĩa cứng được định dạng thành các vòng tròn đồng tâm gọi là rãnh (track), mỗi rãnh được chia thành nhiều phần bằng nhau gọi là cung (sector). Một khối (cluster) gồm một hoặc nhiều cung và dữ liệu được đọc/ghi theo đơn vị khối. Việc sử dụng đơn vị khối để tăng hiệu quả trong việc đọc/ghi và giảm chi phí quản lý số địa chỉ trên đĩa. Ngoài ra khi đĩa cứng lớn, có thể chia thành nhiều phân vùng (partition), mỗi phân vùng gồm một số từ trụ (cyclinder) liên tiếp. Một từ trụ là tập hợp các rãnh cùng bán kính.



Hình 2.14: mô hình tổ chức đĩa

2.2.1.2 File (tập tin)

File là một tập hợp thông tin được đặt tên và lưu trữ trên đĩa. File là đơn vị lưu trữ thông tin nhỏ nhất trên đĩa của hệ điều hành. File có thể lưu trữ chương trình hay dữ liệu, file có thể là dãy tuần tự các byte không cấu trúc hoặc có cấu trúc dòng (kết thúc bằng kí tự enter), hoặc cấu trúc mẫu tin có chiều dài cố định hay thay đổi. Cấu trúc file do hệ điều hành hoặc chương trình qui định. File có thể truy xuất tuần tự (đọc các byte theo thứ tự từ đầu file), hoặc truy xuất ngẫu nhiên (đọc/ghi tại một vị trí bất kỳ trong file).

*** Thuộc tính file (file attributes)**

Thuộc tính của file là các thông tin liên quan đến file, số thuộc tính của file tùy theo hệ điều hành, nhưng thường thì file có các thuộc tính sau:

+ Tên file (file name)

Tên file dùng để phân biệt file này với file khác, UNIX phân biệt tên file chữ thường với tên file chữ hoa nhưng WINDOWS thì không phân biệt. Trong UNIX tên file có thể có nhiều phân cách (ví dụ prog.c.Z), WINDOWS chỉ có một phân cách. Hệ điều hành dùng phần mở rộng để nhận dạng kiểu của file và các thao tác có thể thực hiện trên kiểu file đó, ví dụ phần mở rộng là *.exe, *.com thì hệ điều hành hiểu là file kiểu nhị phân có thể thực thi nhưng nếu không thực thi được thì là do cấu trúc file không đúng qui định của file *.exe, *.com.

+ Kiểu file (file type)

Có hai loại file là file văn bản và file nhị phân. File văn bản chứa các dòng văn bản cuối dòng có ký tự xuống dòng (kí tự enter). File nhị phân gồm dãy các byte, có cấu trúc tùy theo chương trình tạo ra file. Ví dụ file .com, .exe, .wav, .bmp,…. hệ điều hành chỉ thực thi được file .com, .exe nếu nó có cấu trúc đúng qui định.

+ Vị trí file: Danh sách các khối (cluster) trên đĩa đã cấp cho file.

+ Kích thước file: Kích thước hiện thời, kích thước tối đa của file tính bằng bytes/words/blocks,…

+ Ngày giờ tạo file, người tạo file: Ngày, giờ tạo file; ngày, giờ cập nhật file gần nhất; ngày, giờ sử dụng file sau cùng, người tạo file.

+ Loai file: file ẩn, chỉ đọc, hệ thống, lưu trữ, file/thư mục

+ Bảo vệ file: Dùng account (username, password), owner/creator hoặc dùng quyền đã cấp cho user, group trên file hay thư mục, gồm các quyền sau: quyền đọc (Read), ghi (Write), xóa (Delete), thực thi (Execute), liệt kê (List), thêm (Append)...

Thuộc tính	**Ý nghĩa**
Protection	Ai có thể truy xuất file và bằng cách nào
Password	Mật khẩu cần có để truy xuất file
Creator	ID của người tạo file
Owner	Người sở hữu hiện tại
Read-only flag	0: cho phép đọc/ghi; 1: chỉ cho phép đọc
Hidden flag	0: bình thường; 1: không xuất hiện khi liệt kê
System flag	0: file bình thường, 1: file hệ thống
Archive flag	0: đã backup, 1: cần được backup.
ASCII/binary flag	0: file dùng mã ASCII, 1: file dùng mã nhị phân
Random Access flag	0: chỉ cho phép truy xuất tuần tự; 1: cho phép truy xuất ngẫu
Temporary flag	0: bình thường, 1: file bị hủy khi tiến trình kết thúc
Lock flags	0 : không khóa, khác 0: khóa
Record length	Số bytes trong một mẫu tin
Key position	offset của khóa trong từng mẫu tin
Key length	Số byte của khóa trong mẫu tin
Creation time	Ngày giờ tạo file
Time of last access	Ngày giờ truy xuất file gần nhất
Time of last change	Ngày giờ cập nhật nội dung file gần nhất
Current size	Số byte của file
Maximum size	Số bytes tối đa cho phép của file

Hình 2.15: Một số thuộc tính của file/thư mục

*** Các thao tác trên file**

Hệ điều hành cần cung cấp các các lời gọi hệ thống (system call) xử lý file như là: tạo/xóa/mở/đóng/đọc/ghi/thêm/dời con trỏ file/lấy thuộc tính/đặt thuộc tính/đổi tên file,…

+ Tạo file (create): Ghi một mục chứa thông tin file vào cấu trúc thư mục và tìm một khối trống cấp cho file.

+ Xóa file (delete): Tìm tên file trong cấu trúc thư mục, giải phóng tất cả khối đĩa mà file chiếm giữ, xoá mục tương ứng trong cấu trúc thư mục.

+ Mở file (open): Khi lần đầu tiên file được truy xuất, thông tin về file được đọc từ cấu trúc thư mục và lưu vào bảng open-files trong bộ nhớ. Nếu file chưa đóng, thì những lần truy xuất sau sẽ không phải tìm thông tin về file trong cấu trúc thư mục nữa mà lấy trong bảng open-files. Thao tác mở file sẽ trả về chỉ mục trong bảng open-files

+ Đóng file (close): Ghi mục tương ứng trong bảng open-files vào cấu trúc thư mục, và hủy mục này trong bảng open-files.

+ Đọc file (read): Tìm tên file trong cấu trúc thư mục, biết được vị trí lưu trữ file, đọc file vào bộ nhớ, dùng một con trỏ đọc (read pointer) để ghi nhận vị trí cho lần đọc kế tiếp.

+ Ghi file (write): Tìm tên file trong cấu trúc thư mục, lấy được số hiệu khối nhớ đầu tiên cấp phát cho file, ghi dữ liệu của file vào vị trí này, dùng một con trỏ ghi (write pointer) để ghi nhận vị trí cho lần ghi kế tiếp.

+ Ghi thêm vào cuối (append)

+ Lấy thuộc tính (get attribute)

+ Đặt thuộc tính (set attribute)

+ Đổi tên file (rename)

2.2.1.3 Cấu trúc thư mục (Directory Structure)

*** Khái niệm**

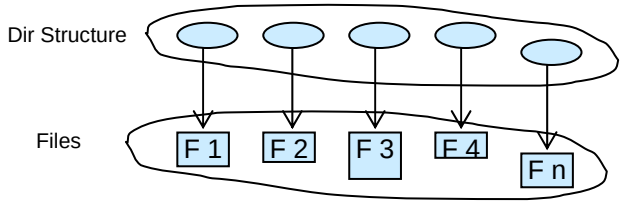
Cấu trúc thư mục là cấu trúc dùng để quản lý tất cả các file trên đĩa. Cấu trúc thư mục cũng được ghi trên đĩa và gồm nhiều mục (Directory Entry), mỗi mục lưu thông tin của một file. Thông thường thông tin file gồm có thuộc tính file và danh sách các số hiệu khối đĩa lưu trữ file. Khi một file được truy xuất, hệ điều hành tìm tên file trong cấu trúc thư mục, nếu tìm thấy sẽ lấy thuộc tính và các số hiệu khối lưu trữ file đưa vào một bảng trong bộ nhớ (bảng open-files), các lần sau khi truy xuất file hoặc thay đổi thông tin file thì không cần truy xuất đĩa mà truy xuất bảng open-files trong bộ nhớ chính và sẽ nhanh hơn nhiều. Hệ điều hành cũng cần cung cấp các lời gọi hệ thống để thao tác trên thư mục tương tự như đối với file. Các thao tác trên thư mục có thể là: Create, Delete, Open, Close, Read, Rename, Link, Unlink,…

mục 0	mục 1	….	mục n

Cấu trúc thư mục

TenFile	ThuocTinh	ViTriLuuTru

Một mục trong cấu trúc thư mục

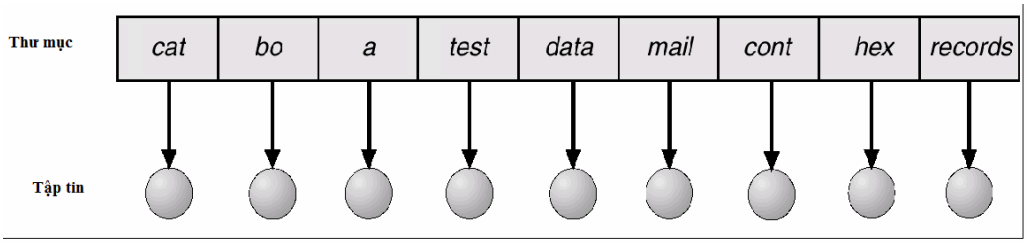


Hình 2.16: mô hình cấu trúc thư mục, mỗi mục trong cấu trúc thư mục quản lý một file hoặc thư mục con

*** Cấu trúc thư mục**

+ Thư mục một cấp: (Single Level Directory)

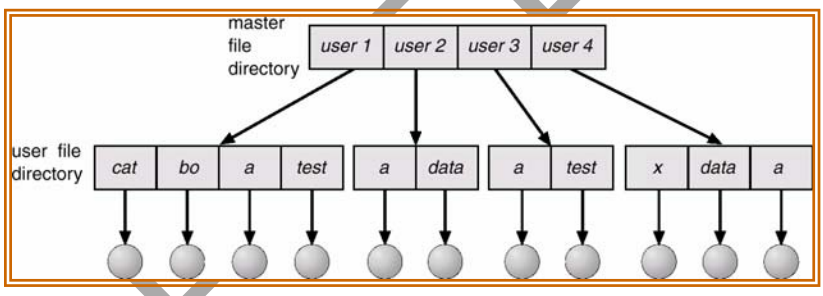
Cấu trúc thư mục sẽ chứa thuộc tính của tất cả các file của tất cả người dùng. Cấu trúc này đơn giản nhưng gây ra khó khăn do đặt tên file không được trùng nhau và người sử dụng không thể phân nhóm cho file cũng như tìm kiếm chậm khi số lương file nhiều.



Hình 2.17: cấu trúc thư mục một cấp

+ Thư mục hai cấp: (Two Level Directory)

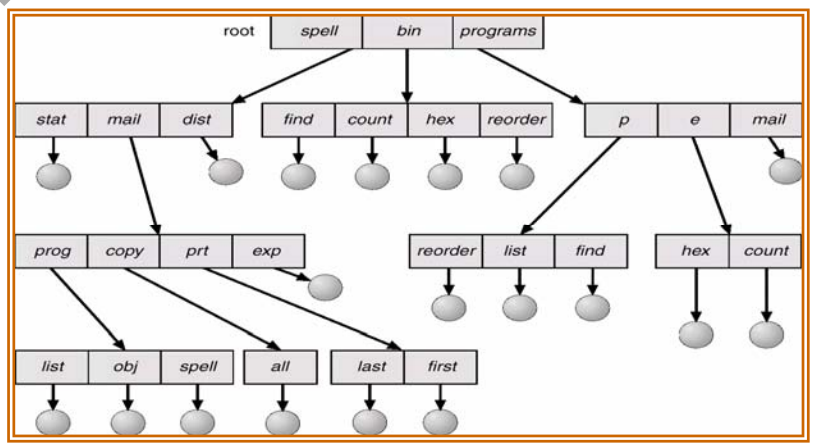
Sự bất tiện chủ yếu của thư mục một cấp là nhầm lẫn về tên file giữa những người dùng. Giải pháp là tạo thư mục riêng (user file directory:UFD) cho mỗi user. Mỗi UFD có cấu trúc giống nhau nhưng chỉ quản lý file của một user, như vậy các user có thể có file cùng tên nhưng user chỉ được truy xuất các file trong thư mục của user đó. Giải pháp này tìm kiếm file nhanh hơn, nhưng vẫn không có khả năng nhóm file. Trong hình 2.5, user1 muốn truy xuất file test thì sử dụng đường dẫn: user1/test.



Hình 2.18: cấu trúc thư mục hai cấp

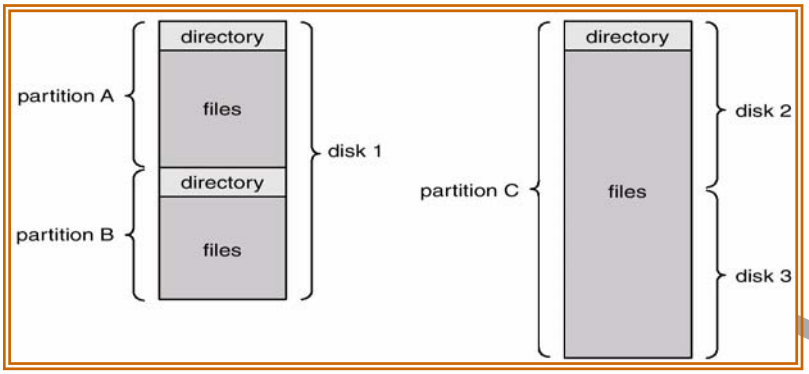
+ Thư mục đa cấp: (Tree-Structured Directory)

Có một thư mục gọi là thư mục gốc (root directory) và trong đó có các thư mục con. Mỗi user có thể tạo những thư mục con riêng, trong mỗi thư mục con chứa file và có thể chứa thư mục con khác. Cấu trúc này cho phép user có thể truy xuất đến file của user khác thông qua quyền được cấp và cho phép tìm kiếm hiệu quả hơn, cũng như có khả năng phân nhóm file.



Hình 2.19: cấu trúc thư mục đa cấp

Hệ điều hành có thể chia đĩa cứng thành nhiều phân vùng (partition), mỗi phân vùng gồm nhiều từ trụ (cyclinder) liên tiếp, hoặc tập hợp nhiều đĩa cứng thành một phân vùng. Mỗi phân vùng sẽ có cấu trúc thư mục riêng để quản lý các tập tin trong phân vùng đó.



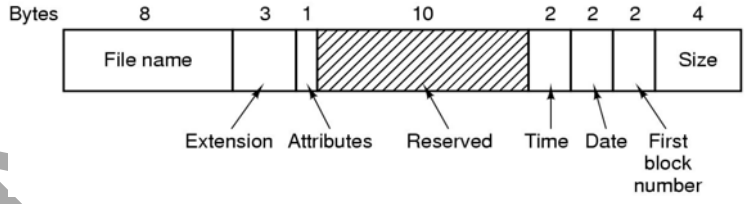
Hình 2.20: tổ chức phân vùng đĩa

*** Bảng thư mục (Directory Table)**

Bảng thư mục là một dạng cài đặt bằng dãy một chiều của cấu trúc thư mục (Directory Structure). Bảng thư mục có nhiều mục (directory entry), mỗi mục lưu trữ thông tin của một file/thư mục, thông tin file gồm thuộc tính của file và địa chỉ trên đĩa của toàn bộ file hoặc số hiệu của khối đầu tiên chứa file hoặc là số I-node của file. Mỗi đĩa có một bảng thư mục gọi là bảng thư mục gốc, cài đặt ở phần đầu của đĩa và có thể có nhiều bảng thư mục con.

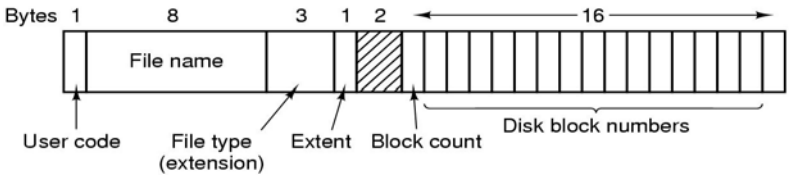
Ví dụ:

+ Mỗi mục trong bảng thư mục của hệ điều hành MSDOS/WINDOWS (FAT) chỉ lưu số hiệu khối đầu tiên của mỗi file/thư mục. Khi đó để biết số hiệu các khối còn lại của file/thư mục, hệ điều hành sẽ dùng bảng cấp phát file (bảng FAT).



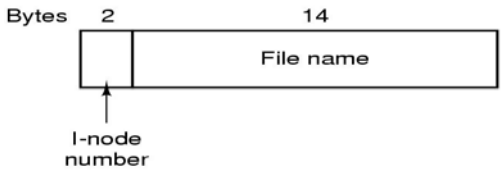
Hình 2.21: cấu trúc một mục trong bảng thư mục của MSDOS/WINDOWS (FAT)

+ Mỗi mục trong bảng thư mục của hệ điều hành CP/M chứa tất cả các số hiệu khối chứa file/thư mục, khi đó không cần dùng bảng cấp phát file (bảng FAT)



Hình 2.22: cấu trúc một mục trong bảng thư mục của CP/M

+ Mỗi mục trong bảng thư mục của hệ điều hành UNIX lưu số hiệu I-node



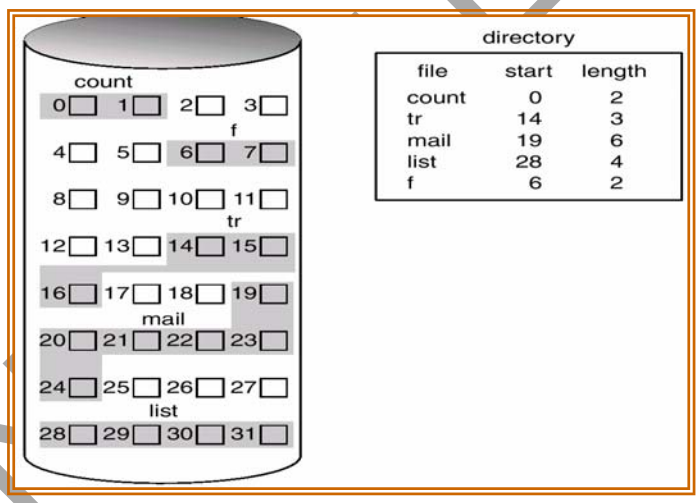
Hình 2.23: cấu trúc một mục trong bảng thư mục của UNIX

2.2.2 Cài đặt hệ thống quản lý file

Việc cài đặt hệ thống quản lý file phụ thuộc vào việc hệ điều hành chọn phương pháp cấp phát khối nhớ cho file là liên tục hay không liên tục.

2.2.2.1 Cấp phát khối nhớ liên tục

Khi cấp phát khối nhớ liên tục, để cài đặt hệ thống quản lý file, hệ điều hành chỉ cần dùng bảng thư mục, mỗi mục trong bảng thư mục ngoài những thuộc tính thông thường của file, cần có thêm thông tin về số hiệu khối bắt đầu (start) và số khối đã cấp cho file (length). Phương pháp này dễ cài đặt, dễ thao tác vì toàn bộ file được đọc từ các khối liên tiếp trên đĩa không cần định vị lại, nhưng có khuyết điểm là file không thể phát triển và có thể gây ra lãng phí do sự phân mảnh trên đĩa.



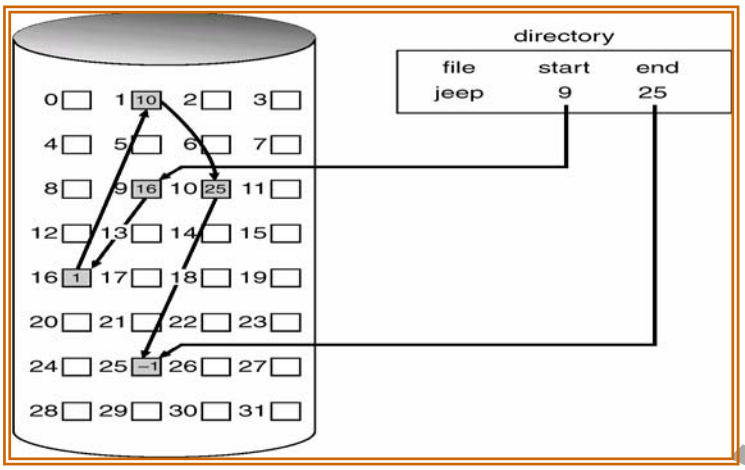
Hình 2.24: Bảng thư mục trong mô hình cấp phát liên tục

2.2.2.2 Cấp phát khối nhớ không liên tục

Khi cấp phát khối nhớ liên tục, để cài đặt hệ thống quản lý file, hệ điều hành sử dụng bảng thư mục và cần phải sử dụng thêm một trong các cấu trúc sau: danh sách liên kết, bảng chỉ mục, bảng cấp phát file, bảng I-Nodes.

2.2.2.2.1 Danh sách liên kết:

Mỗi mục trong bảng thư mục chứa số hiệu của khối đầu tiên (start) và số hiệu của khối kết thúc (end), mỗi khối trên đĩa dành một số byte đầu hoặc cuối (thường là 4 bytes) để lưu số hiệu khối kế tiếp của file, phần còn lại của khối sẽ lưu dữ liệu của file. Phương pháp này không bị lãng phí do phân mảnh ngoại vi, nhưng truy xuất ngẫu nhiên chậm và rất khó bảo vệ các số hiệu khối của File.



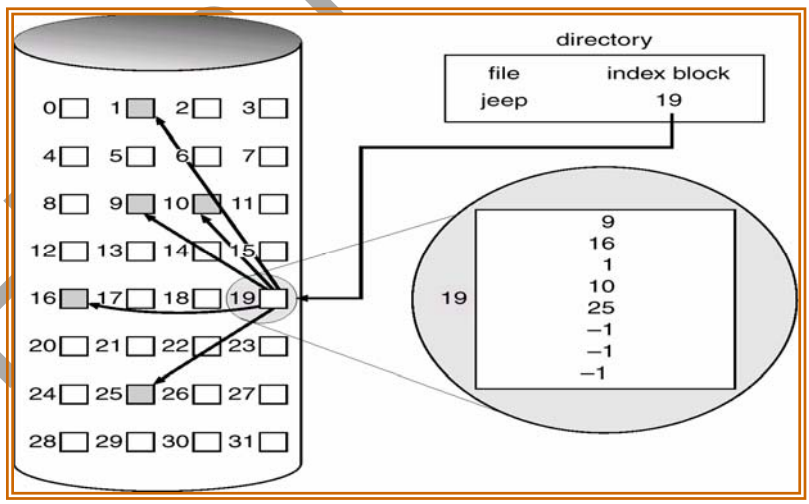
Hình 2.25: mô hình cấp phát không liên tục, sử dụng danh sách liên kết

2.2.2.2.2 Bảng chỉ mục (index table):

Mỗi file có một bảng chỉ mục chiếm một hoặc vài khối, bảng chỉ mục chứa tất cả các số hiệu khối của một file. Khi đó một mục trong bảng thư mục sẽ lưu số hiệu khối chứa bảng chỉ mục của file. Phương pháp này dễ bảo vệ các bảng chỉ mục, nghĩa là bảo vệ được các số hiệu khối của file nhưng tốn nhiều khối nhớ để lưu các bảng chỉ mục.

Ví dụ đĩa cứng có dung lượng 32 GB, kích thước 1 khối là 512 Bytes. Hỏi kích thước một bảng chỉ mục là bao nhiêu nếu muốn quản lý file có kích thước lớn nhất là 256K?

32 GB=2^5 x 2^{10} x 2^{10} KB = 2^{25} KB = 2^{16} khối => có 2^{16} địa chỉ trên đĩa => mỗi phần tử bảng chỉ mục cần 2 byte. File kích thước 256KB = 256 x 1024 byte = 512 khối => bảng chỉ mục cần có 512 phần tử => một bảng chỉ mục chiếm hai khối!

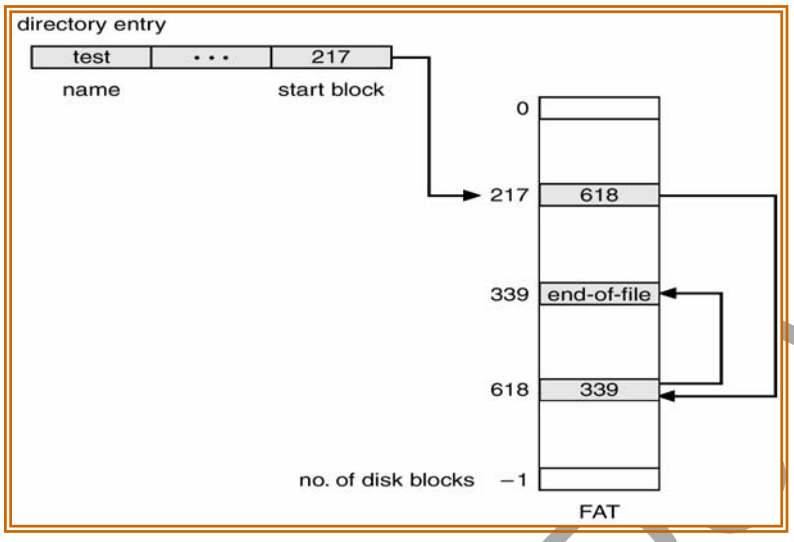


Hình 2.26: mô hình cấp phát không liên tục, sử dụng bảng chỉ mục

2.2.2.2.3 Bảng cấp phát file (FAT: File Allocation Table)

Nếu mỗi mục trong bảng thư mục chỉ chứa số hiệu của khối đầu tiên, thì số hiệu các khối còn lại của file sẽ được lưu trong một bảng gọi là bảng cấp phát file (bảng FAT). Phương pháp này dễ bảo vệ các số hiệu khối đã cấp cho file, truy xuất file ngẫu nhiên dễ dàng hơn, kích thước file dễ mở rộng nhưng bảng FAT bị giới hạn bởi kích thước bộ nhớ dành cho nó. Đây chính là cách mà hệ điều hành MSDOS, OS/2, Windows (FAT) sử dụng để quản lý File.

Ví dụ: Giả sử file test.txt được lưu trữ lần lượt ở 3 khối trên đĩa có số hiệu là: 217, 618, 339. Số hiệu khối đầu là 217 ghi trong một mục của bảng thư mục, các số hiệu khối tiếp theo là 618, 339 ghi trong bảng cấp phát file.



Hình 2.27: mô hình cấp phát không liên tục, sử dụng bảng FAT

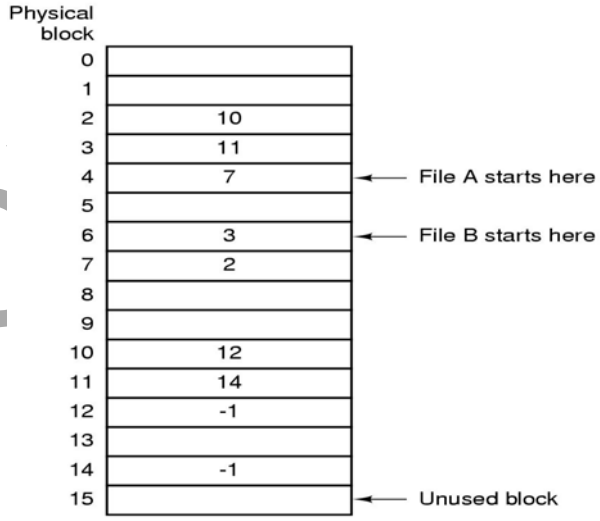
Ví dụ: Giả sử file A và file B được cấp phát gồm các khối theo thứ tự sau: file A: 4, 7, 2, 10, 12 ; file B: 6, 3, 11, 14. Khi đó trong bảng thư mục sẽ có một mục lưu tên file A và số hiệu khối đầu của file A là 4. Tương tự có một mục lưu tên file B và số hiệu khối đầu của file B là 6. Các số hiệu khối tiếp theo của file A và file B lưu trong bảng cấp phát file (fat)

Bảng thư mục:

…	(A, 4)	…	(B, 6)	…

Entry của A Entry của B

Bảng FAT :

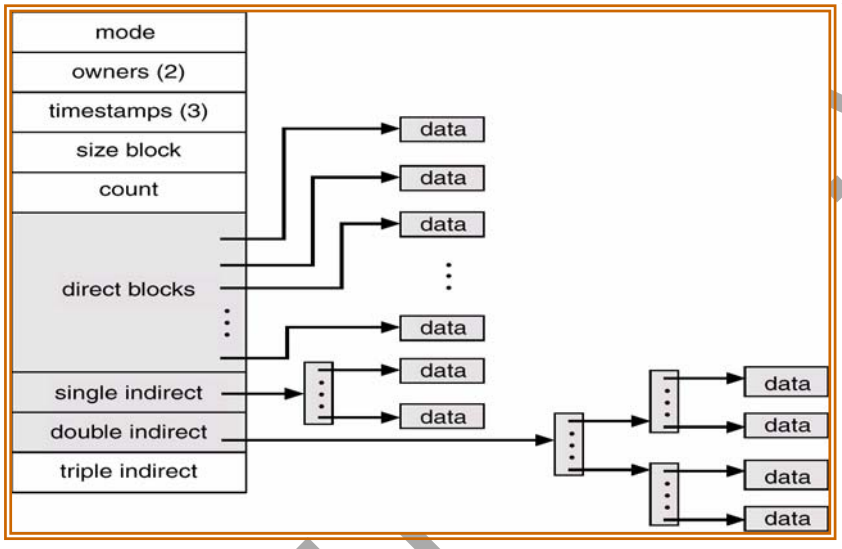


Hình 2.28: mô hình cấp phát không liên tục, sử dụng bảng FAT, có hai file

2.2.2.2.4 Bảng I-nodes:

Mỗi file được quản lý bằng một cấu trúc gọi là I-node, mỗi I-node gồm hai phần: phần thứ nhất lưu trữ thuộc tính file, phần thứ hai gồm 13 phần tử: 10 phần tử đầu chứa 10 số hiệu khối đầu tiên của file, phần tử thứ 11 chứa số hiệu khối chứa bảng single, phần tử thứ 12 chứa số hiệu khối chứa bảng double, phần tử thứ 13 chứa số hiệu khối chứa bảng triple. Trong đó mỗi phần tử của bảng triple chứa số hiệu khối chứa bảng double, mỗi phần tử của bảng double chứa số hiệu khối chứa bảng single và mỗi phần tử của bảng single chứa số hiệu khối dữ liệu tiếp theo cấp cho file.

Ghi chú: Hệ điều hành Unix sử dụng cấu trúc này và số phần tử của phần thứ hai, không nhất thiết là 13 mà có thể thay đổi tùy phiên bản của UNIX



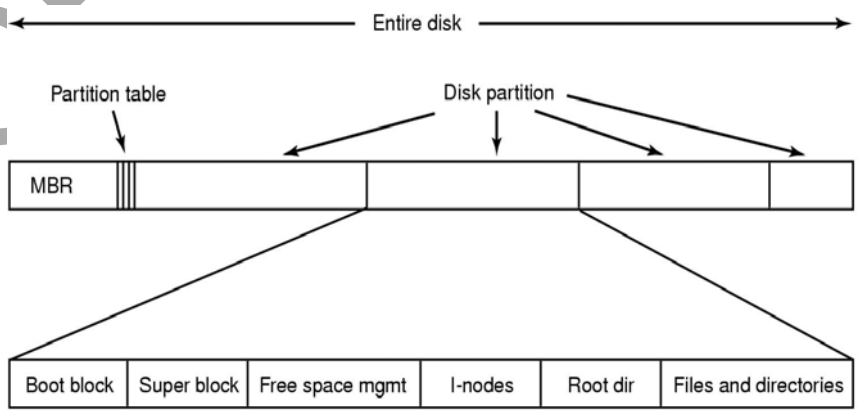
Hình 2.29: cấu trúc một I-node trong bảng I-nodes

**** Phương pháp tổ chức quản lý đĩa bằng I-nodes:***

+ MBR(Master Boot Record): là sector đầu tiên chứa thông tin về đĩa.

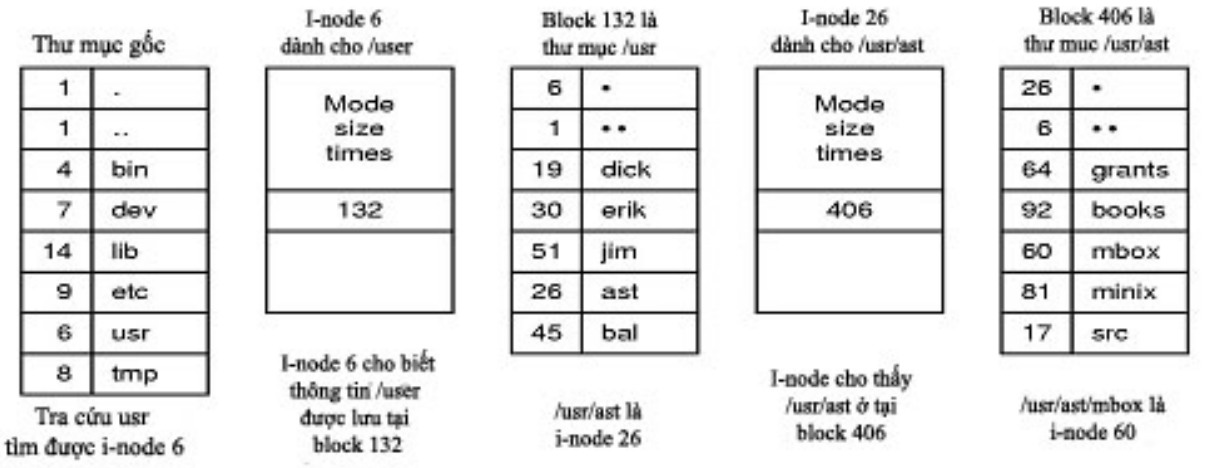
+ Partion Table: bảng phân vùng chứa các thông tin về mỗi phân vùng.

Một phân vùng được tổ chức thành các phần sau: boot block, super block, free space mgmt, i-nodes, root dir, files and directories. Trong đó I-nodes là bảng I-nodes gồm nhiều mục, mỗi mục gọi là một i-node, chứa một cấu trúc i-node ghi thông tin của một file. Mỗi mục của bảng thư mục gốc (root dir) ghi tên file và số hiệu i-nodes của file. Phương pháp này tương đối linh động và hiệu quả khi quản lý những hệ thống file lớn.



Hình 2.30: mô hình cấp phát không liên tục, sử dụng bảng I-nodes tổng quát

Ví dụ hệ điều hành muốn đọc file /usr/ast/mbox, trước tiên tìm tên thư mục "usr" trong bảng thư mục gốc (root dir), và nhận thấy thư mục "usr" có i-nodes là 6. Tiếp theo truy xuất phần tử i-node thứ 6 trong bảng i-nodes, lấy được số hiệu khối đầu của "usr" là 132, khối này sẽ chứa bảng thư mục con (sub dir) của "usr". Tiếp theo tìm tên thư mục "ast" trong bảng thư mục con của "usr", và tìm được thư mục "ast" có i-nodes là 26. Tiếp theo truy xuất phần tử i-node thứ 26 trong bảng i-nodes, lấy được số hiệu khối đầu của "ast" là 406, khối này sẽ chứa bảng thư mục con của "ast". Tiếp theo tìm tên file "mbox" trong bảng thư mục con này, và tìm được file "mbox" có i-nodes là 60. Truy xuất phần tử i-node thứ 60 trong bảng i-nodes, lấy được các số hiệu khối của file "mbox".



Hình 2.31: mô hình cấp phát không liên tục, sử dụng bảng I-nodes chi tiết

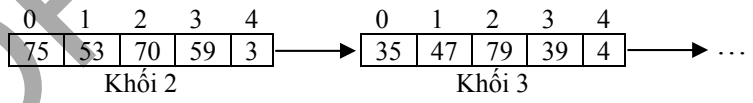
2.2.3. Quản lý các khối trống

Có hai phương pháp mà hệ điều hành thường sử dụng để quản lý các khối trống là sử dụng danh sách liên kết hoặc vector bit.

2.2.3.1 Danh sách liên kết

Mỗi nút trong danh sách liên kết (dslk) là một khối chứa một bảng gồm các số hiệu khối trống, phần tử cuối của bảng lưu số hiệu khối tiếp theo trong danh sách.

Ví dụ: Giả sử khối đầu tiên trong dslk là khối 2. Trong khối 2 lưu các số 75, 53, 70, 59 là các số hiệu khối trống, các khối 3 là khối chứa bảng các số hiệu khối trống tiếp theo…Hệ điều hành chỉ cần biết số hiệu khối đầu tiên của danh sách liên kết.



Ví dụ: Một đĩa 20M, dùng khối có kích thước 1 K. Để quản lý đĩa này, nếu đĩa hoàn toàn trống thì DSLK cần bao nhiêu khối (số nút tối đa của dslk)?

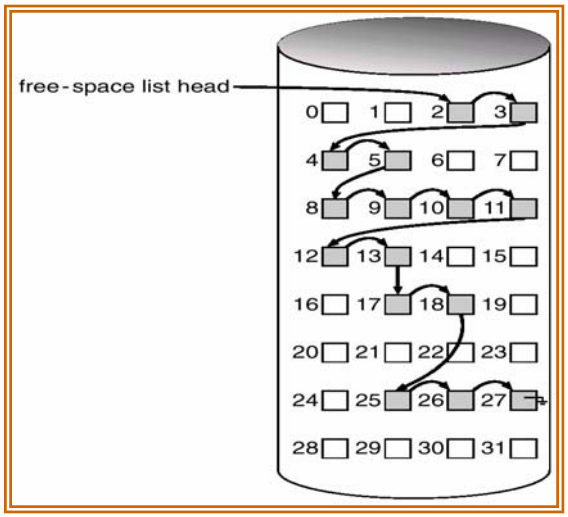
Giải:

20M= 20x 2^{10} khối ~ 2^{15} khối => cần dùng 16 bit=2 byte để lưu một số hiệu khối

=> 1 khối=1024 byte lưu được 511 số hiệu khối trống

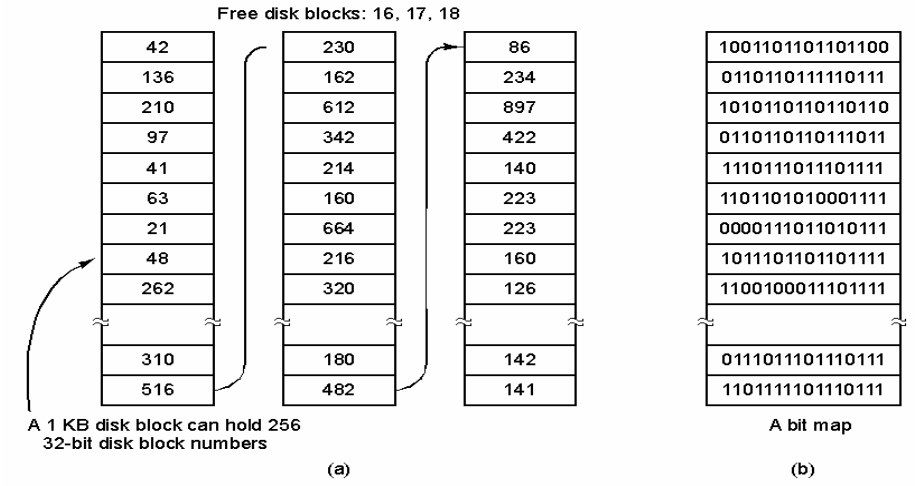
=> để quản lý đĩa có 20M hoàn toàn trống, dslk cần 20x 2^{10}/ 511 ~ 40 khối !

Nhận xét: Tốn khá nhiều khối nhớ cho dslk nếu đĩa hoàn toàn trống, nhưng sẽ ít tốn khối nhớ cho dslk nếu đĩa gần đầy.



Hình 2.32: Quản lý danh sách các khối trống sử dụng danh sách liên kết.

Ví dụ: Giả sử khối đĩa 1KB và một sô hiệu khối đĩa là 32 bit thì một khối đĩa có thể ghi được 256 -1 số hiệu khối đĩa trống



Hình 2.33: Quản lý danh sách các khối trống sử dụng danh sách liên kết và vector bit.

2.2.3.2 Dùng vector bit (dãy bít)

Bit thứ i = 1 là khối thứ i trống, bit thứ i = 0 là khối thứ i đã sử dụng. Vector bit được lưu trên một hoặc nhiều khối đĩa, khi cần sẽ đọc vào bộ nhớ để xử lý nhanh. Vector bit ít tốn khối nhớ hơn là dslk nhưng kích thước vector bit là cố định và hệ điều hành cần đồng bộ vector bit trong bộ nhớ và vector bit trên đĩa.

Ví dụ: xét lại ví dụ trên, nếu dùng vector bit, hãy tính kích thước vector bit.

HD: Đĩa 20M có 20×2^{10} khối nên kích thước vector bit là 20×2^{10} bit = 20×2^{10} /8/ 2^{10} KB ~ 3 khối.

2.2.4. Quản lý sự an toàn của hệ thống tập tin

2.2.4.1 Quản lý khối bị hỏng

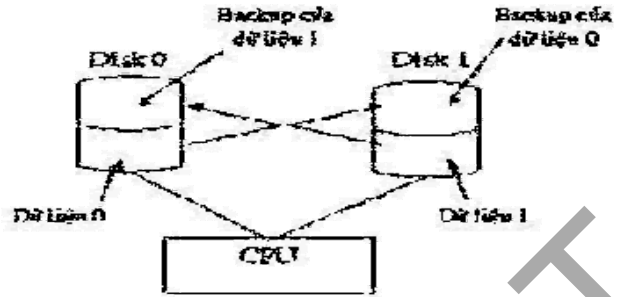
+ Dùng phần mềm: dùng một file chứa các danh sách các khối hỏng.

+ Dùng phần cứng: dùng một sector trên đĩa để lưu giữ danh sách các khối bị hỏng.

Khi bộ kiểm soát đĩa thực hiện lần đầu tiên, nó đọc danh sách khối bị hỏng vào bộ nhớ, từ đó không cho truy cập những khối hỏng.

2.2.4.2 Sao lưu file (Backup)

File trên đĩa mềm được sao lưu bằng cách chép lại toàn bộ qua một đĩa khác. Dữ liệu trên đĩa cứng nhỏ thì được sao lưu trên các băng từ. Đối với đĩa cứng lớn, việc sao lưu có thể tiến hành như sau: chia đĩa cứng làm hai phần một phần chứa dữ liệu gốc và một phần chứa bản sao. Mỗi khi thực hiện sao lưu, phần dữ liệu gốc sẽ được chép sang phần sao lưu.



Hình 2.34: sao lưu dữ liệu

2.2.5 Giới thiệu một số hệ thống tập tin

2.2.5.1 Hệ thống tập tin của MSDOS/Windows (FAT)

Hệ điều hành MSDOS hoặc Windows (sử dụng hệ thống FAT) sẽ quản lý hệ thống tập tin thông qua ba cấu trúc sau: Boot Sector, bảng FAT, bảng ROOT DIR.

+ Boot sector

Ở sector đầu tiên, track 0, side 0 của đĩa mềm, đối với đĩa cứng thì vị trí này là bảng partition, rồi mới tới boot sector của partition thứ nhất, đối với các partition khác, boot sector là sector đầu tiên. Boot sector chứa bảng tham số đĩa BPB (Bios Parameter Block) và chứa đoạn mã boot dùng để nạp các file hệ thống, boot sector hợp lệ phải có giá trị AA55 ở offset 1FE. Trên máy IBM PC sau khi thực hiện thao tác POST (Power On Self Test), ROM BIOS tìm boot sector hợp lệ, đọc boot sector vào địa chỉ 0X7C00, gán CS=0000h, IP=7C00h và cho thực thi lệnh đầu tiên trong boot sector (lệnh JMP). Boot sector có cấu trúc như sau:

Offset(hex)	Size(byte)	Content	Giải thích
00	3	JMP	Lệnh nhảy đến đầu đoạn mã boot
03	8	Version	Phiên bản hệ điều hành
0B	2	SectSiz	số byte /một sector, thường là 512 (đây là bắt đầu của BPB)
0D	1	ClustSize	số sector / một cluster (khác 0)
0E	2	ResSecs	số sector dành riêng trước bảng FAT, tính luôn boot sector (đối với FAT12, FAT16 =1, FAT32=20h)
10	1	FatCnt	số bảng FAT (thường là 2)
11	2	RootSiz	Số mục trong bảng ROOT DIR
13	2	TotSecs	Tổng số sector trên đĩa hay trên một partition (<=65535)
15	1	MediaDescriptor	Byte nhận dạng đĩa (F8:đĩa cứng, F0:1.44 MB)
16	2	FatSize	Số sector /một bảng FAT (<=65535). FAT32=0
18	2	TrkSecs	số sector / một track
1A	2	HeadCnt	số đầu đọc
1C	4	HidnSec	số sector ẩn (ở giữa bảng partition và partion)
20	4	BigTotSecs	Tổng số sector trên đĩa hay trên một partition (>65535)
24	4	BigFatSize	Số sector /một bảng FAT (>65535)
…	…	…	…
1D			(kết thúc BPB)
1E			đầu đoạn mã boot
…			…
1FF			cuối đoạn mã boot

Hình 2.35: Cấu trúc boot sector.

+ Bảng FAT

Sau boot sector là bảng FAT (File Allocation Table), thường có hai bảng FAT để an toàn. Mỗi mục của FAT quản lý một khối (cluster) dữ liệu (khối dữ liệu được đánh số bắt đầu từ 2). Hai mục đầu tiên của bảng FAT chứa thông tin mô tả loại đĩa. Kích thước khối được lưu trong boot sector thông thường từ 1 đến 8 sector. Có ba loại FAT là FAT 12 và FAT 16, FAT 32. FAT 12 (mỗi mục trong bảng FAT12 có kích thước là 12 bit) có thể quản lý được 2^{12} = 4096 khối, FAT 16 có thể quản lý 2^{16} = 64 K khối, FAT 32 có thể quản lý 2^{32} =4G khối trên một partition. Các giá trị có thể có trong một entry của bảng FAT.

(0)000	Cluster còn trống
(0)002 - (F)FEF	Cluster chứa dữ liệu của File, giá trị này là số hiệu cluster kế.
(F)FF0 - (F)FF6	Dành riêng, không dùng

(F)FF7	Cluster hỏng
(F)FF8 - (F)FFF	Cluster cuối cùng của chuỗi (kết thúc file)

Hình 2.36: Cấu trúc một mục (entry) trong bảng fat.

+ Bảng ROOT DIR (bảng thư mục gốc)

Bảng ROOT DIR nằm ngay sau FAT, và mỗi mục của bảng DIR là 32 byte. Khi hệ thống mở một File/ thư mục, MS-DOS tìm tên File/thư mục trong bảng ROOT DIR, lấy số hiệu khối đầu đã cấp cho file/thư mục và tìm các số hiệu khối tiếp theo trong bảng FAT.

Boot sector	Bảng FAT1	Bảng FAT2	Bảng DIR	DATA

Kích thước	Thuộc tính
8 byte	Tên File
3 byte	Phần mở rộng
1 byte	Thuộc tính : A – D – V – S - H - R
10 byte	Dành riêng để sử dụng sau này.
2 byte	Giờ : 5 bit cho giờ, 6 bit cho phút, 5 bit cho giây (thieu 1, nên lưu đơn vị 2 giây)
2 byte	Ngày : 7 bit cho năm (từ 1980), 4 bit cho tháng, 5 bit cho ngày.
2 byte	Số hiệu khối đầu tiên của file
4 byte	Kích thước File

Hình 2.37: Cấu trúc một mục trong bảng ROOT DIR/ SUB DIR

Giá trị của byte thuộc tính:

1 : File chỉ đọc (Read Only)

2 : File ẩn (Hidden)

4 : File hệ thống (System)

8 : nhãn đĩa (Volume)

16 : thư mục con (Directory)

32 : File chưa được sao lưu (Archive)

Ví dụ: Xét đĩa 1.4MB, được format dưới hệ điều hành MS-DOS/WINDOWS (FAT12) gồm có 2880 sector (1.4MB=>1.4*1024*1024/512=2880 sector), và 1 khối (cluster) = 1 sector nên mỗi mục của bảng FAT chỉ cần 12 bit. Sector đầu tiên là boot sector, bao gồm bảng tham số vật lý của đĩa và chương trình khởi động của hệ điều hành. 18 sector tiếp theo là FAT (FAT12), gồm 2 bảng, mỗi bảng 9 sector (có 3072 mục, nếu 8 sector thì không đủ để quản lý 2880 khối). Ba bytes đầu tiên của mỗi bảng FAT lưu số hiệu loại đĩa (240, 255, 255). 14 sector kế tiếp chứa bảng thư mục gốc (bảng ROOT DIR). Các sector còn lại dùng để lưu dữ liệu, cluster đánh số từ 2.

Số sector	1	9	9	14	Còn lại
Lưu trữ	Boot sector	FAT12	FAT12	DIR	DATA

Cấu trúc một mục trong bảng ROOT DIR

Byte	0-7	8-10	11	12-21	22-23	24-25	26-27	28-31
Lưu trữ	Tên File	Phần mở rộng	ADVSHR	Dành riêng	Giờ	Ngày	Số hiệu khối đầu	Kích thước File

Bảng DIR= 14 sector = 7168 byte = 224 entry (mỗi entry 32 byte), 1 sector=16 entry

Đĩa 1.44MB ở thư mục gốc có tối đa 224 file hoặc thư mục con. Nếu file có kích thước file =0 thì số hiệu khối đầu = 0. Kí tự đầu của tên file có giá trị là 0 (trống), dấu chấm (dành riêng cho DOS), 0xE5 (file đã bị xóa: khi xóa file, DOS sẽ điền kí tự 0xE5 vào kí tự đầu của file).

+ Cấu trúc bảng FAT

Byte	0,1,2	3,4,5	…	4606,4607
Lưu trữ	e0,e1 (số hiệu đĩa)	e2,e3	…	e3070, e3071

Bảng FAT12 = 9 sector = 4608 bytes = 3072 entry (mỗi entry 12 bit)

- Đọc/ghi FAT 12

Ví dụ: ghi vào entry 2 giá trị F2Ah và entry 3 giá trị BC5h. Do byte 0,1,2 lưu số hiệu đĩa, nên entry 2 sẽ được ghi vào byte thứ 3,4 và entry 3 sẽ được ghi vào byte thứ 4,5 . Cách ghi như sau:

Byte	0	1	2	3	4	5
Giá trị	(số hiệu đĩa	)		2A	5F	BC

```
unsigned char fat[512*9]; //mảng chứa bảng fat đọc từ đĩa
void WriteFat (unsigned new_fat, unsigned k) // ghi giá trị new_fat vào entry thứ k = 2,3,...,3071
{
        unsigned i=k*3/2; //entry k sẽ được ghi vào byte thứ i và i+1
        if (k%2==0)// k chẵn
        {       //đặt 8 bít cuối của new_fat vào byte thứ i
                fat[i] = new_fat&0x0FF;
                //đặt 4 bít cao của new_fat vào 4 bít cuối của byte thứ i+1
                fat[i+1] = (fat[i+1]&0xF0) | (new_fat>>8);
        }
        else //k lẻ
```

```
        {       //đặt 8 bít cao vào vào byte thứ i+1
                fat[i+1]=new_fat>>4;
                //đặt 4 bít thấp vào 4 bít cao của byte thứ i
                fat[i]=((new_fat&0xF)<<4)|(fat[i]&0x0F);
        }
}
```

Ví dụ: giả sử new_fat = 1AB, luu vao entry k=7 ->luu vao byte i=10 và i+1=11

9	10	11	
AB	CD	EF	(trước khi ghi)
AB	BD	1A	(sau khi ghi)

```
unsigned ReadFat (unsigned k) // Đọc giá trị của entry thứ k (k>=2)
{
        unsigned i=k*3/2;       //entry thứ k ở byte thứ i, i+1
        unsigned val_fat=(fat[i+1]<<8) | fat[i] ;  // hoặc val_fat=fat[i+1]*256+fat[i];
        if  (val_fat%2==0)  val_fat=val_fat & 0x0FFF;
        else  val_fat=val_fat>>4;
        return val_fat;
}
```

- Đọc/ghi giờ, phút, giây

time (2 byte)= 5 bit giờ, 6 bit phút, 5 bit giây (thieu 1, nên lưu đơn vị 2 giây)

- Ghi : time=(h<<11)|(m<<5)|(s>>1)
- Đọc: h=(time>>11); m=(time>>5)&0x3F; s=(time&0x1F)<<1;

- Đọc/Ghi ngày, tháng, năm

date (2 byte)= 7 bit cho năm (từ 1980), 4 bit cho tháng, 5 bit cho ngày.

- Ghi : y-=1980 ; date=(y<<9)|(m<<5)|d ;
- Đọc : y=(date>>9) + 1980 ; m=(date>>5)&0xF ; d=date&0x1F ;

2.2.5.2 Hệ thống tập tin của Windows NT

Sử dụng hệ thống NTFS, kích thước File tối đa trong NTFS là 232 cluster. Cấu trúc volume của NTFS như sau: partition boot sector, Master File Table, các file hệ thống, vùng dữ liệu.

+ ***Partition boot sector***: là sector khởi động của partition (<= 16 sector) gồm các thông tin về cấu trúc của volume, cấu trúc của hệ thống File và mã nguồn khởi động.

+ ***Master File Table (MFT):*** lưu các thông tin về tất cả file và thư mục trên volume NTFS này cũng như danh sách các khối trống. MFT được tổ chức thành nhiều dòng, mỗi dòng lưu những

thuộc tính cho một file hoặc một thư mục trên volume. Nếu kích thước file nhỏ thì toàn bộ nội dung của file được lưu trong dòng này.

+ ***Các file hệ th***ống: có kích thước khoảng 1Mb bao gồm:

. MFT2: bản sao của MFT

. Log file: thông tin dùng cho việc phục hồi.

. Cluster bitmap: biểu diễn thông tin lưu trữ của các cluster

. Bảng định nghĩa thuộc tính: định nghĩa các kiểu thuộc tính hỗ trợ cho volume.

Kiểu thuộc tính	Mô tả
Thông tin chuẩn	Bao gồm các thuộc tính truy xuất (chỉ đọc, đọc/ghi,...), nhãn thời gian, chỉ số liên kết
Danh sách thuộc tính	sử dụng khi tất cả thuộc tính vượt quá 1 dòng của MFT
Tên File	
Mô tả an toàn	thông tin về người sở hữu và truy cập
Dữ liệu	
Chỉ mục gốc	dùng cho thư mục
Chỉ mục định vị	dùng cho thư mục
thông tin volume	như tên version và tên volume
Bitmap	hiện trạng các dòng trong MFT

Hình 2.38: Các kiểu thuộc tính của File và thư mục trong Windows NTFS

2.5.3. Hệ thống file của UNIX

Hệ thống File của UNIX thông thường được cài đặt trên đĩa gồm các khối theo thứ tự sau: khối boot, khối đặc biệt, I-nodes, các khối dữ liệu.

+ ***Khối boot***: chứa mã khởi động của hệ thống.

+ ***Khối super block*** : chứa thông tin về toàn bộ hệ thống file, bao gồm:

- Kích thước của toàn bộ hệ thống file.
- Địa chỉ của khối dữ liệu đầu tiên.
- Số lượng khối trống và danh sách khối trống.
- Số lượng I-node trống và danh sách I-node trống.
- Ngày super block được cập nhật sau cùng.
- Tên của hệ thống file.

Nếu khối này bị hỏng, hệ thống file sẽ không truy cập được. Có rất nhiều trình ứng dụng sử dụng thông tin lưu trữ trong super block, vì vậy một bản sao super block được đặt trong RAM để tăng

tốc độ truy xuất đĩa. Việc cập nhật super block sẽ được thực hiện ngay trong RAM và sau đó mới ghi xuống đĩa. Các I-node được đánh số từ 1, mỗi I-node có độ dài là 64 byte và mô tả cho một file duy nhất, chứa thuộc tính và địa chỉ các khối lưu trữ trên đĩa của file. Tất cả file và thư mục được lưu trữ ở các khối dữ liệu.

TÓM TẮT

+ Thiết bị nhập/xuất có thể phân thành các loại như là thiết bị khối, thiết bị tuần tự, thiết bị khác. Đặc tính của thiết bị nhập/xuất là tốc độ truyền dữ liệu, công dụng, đơn vị truyền dữ liệu, cách biểu diễn dữ liệu, tình trạng lỗi

+ Mỗi thiết bị nhập/xuất cần có bộ điều khiển thiết bị, thiết bị và bộ điều khiển phải tuân theo cùng chuẩn giao tiếp. CPU không thể truy xuất trực tiếp thiết bị nhập/xuất mà phải thông qua bộ điều khiển thiết bị, mỗi bộ điều khiển có một số thanh ghi để liên lạc với CPU

+ Các chương trình thực hiện nhập/xuất gồm có chương trình nhập/xuất của người dùng, chương trình nhập/xuất độc lập thiết bị, chương trình điều khiển thiết bị.

+ Hệ thống quản lý nhập/xuất được tổ chức thành 5 lớp: chương trình nhập/xuất của người dùng, chương trình nhập/xuất độc lập thiết bị, chương trình điều khiển thiết bị, chương trình kiểm soát ngắt, phần cứng. Mỗi lớp có chức năng riêng và có thể giao tiếp với lớp khác.

+ Cơ chế nhập/xuất: bộ xử lý phát sinh một lệnh I/O đến các thiết bị I/O, sau đó chờ cho đến khi thao tác I/O hoàn tất rồi mới tiếp tục xử lý, hoặc bộ xử lý phát sinh một lệnh I/O đến các thiết bị I/O, sau đó tiếp tục việc xử lý cho tới khi nhận được một ngắt từ thiết bị I/O báo là đã hoàn tất nhập/xuất, bộ xử lý tạm ngưng việc xử lý hiện tại để chuyển qua xử lý ngắt, hoặc sử dụng cơ chế DMA

+ Cơ chế DMA: bộ điều khiển thường được cung cấp thêm khả năng truy xuất bộ nhớ trực tiếp (DMA). Nghĩa là sau khi bộ điều khiển đã đọc toàn bộ dữ liệu từ thiết bị vào buffer của nó, bộ điều khiển tự chuyển dữ liệu vào trong bộ nhớ chính.

+ Cơ chế truy xuất đĩa: đầu đọc được di chuyển đến track thích hợp, và chờ cho đến khi khối cần đọc đến dưới đầu đọc, đọc dữ liệu từ đĩa vào bộ nhớ chính, hệ điều hành cần đưa ra các thuật toán lập lịch dời đầu đọc sao cho tối ưu. Các thuật toán lập lịch di chuyển đầu đọc: FCFS, SSTF, SCAN, C-SCAN

+ Hệ số đan xen: Trên đĩa các sector số hiệu liên tiếp nhau không nằm kế bên nhau mà có một khoảng cách nhất định, khoảng cách này được xác định bởi quá trình format đĩa và gọi là hệ số đan xen. Mục đích của hệ số đan xen là để đồng bộ hai thao tác đọc/ghi dữ liệu và chuyển dữ liệu vào hệ thống

+ RAM disk là một phần của bộ nhớ chính để lưu trữ các khối đĩa. RAM disk truy xuất rất nhanh, thích hợp cho việc lưu trữ những chương trình hay dữ liệu được truy xuất thường xuyên.

+ Đĩa cứng được tổ chức thành các vòng tròn đồng tâm gọi là rãnh (track), mỗi rãnh được chia thành nhiều phần bằng nhau gọi là cung (sector). Một khối (cluster) gồm 1 hoặc nhiều cung và dữ liệu được đọc/ghi theo đơn vị khối.

+ File là một tập hợp các thông tin được đặt tên và lưu trữ trên đĩa. File có thể lưu trữ chương trình hay dữ liệu, file có thể là dãy tuần tự các byte không cấu trúc hoặc có cấu trúc dòng, hoặc cấu trúc mẫu tin có chiều dài cố định hay thay đổi. File có thể truy xuất tuần tự, hoặc truy xuất ngẫu nhiên.

+ Thuộc tính của file là thông tin của file, thường có các thuộc tính sau: tên file, kiểu file, vị trí file, kích thước file, ngày giờ tạo file, người tạo file, loai file, bảo vệ file.

+ Các thao tác trên file gồm có tạo/xóa/mở/đóng/đọc/ghi/thêm/dời con trỏ file/lấy thuộc tính/đặt thuộc tính/đổi tên file…

+ Cấu trúc thư mục là cấu trúc dùng để quản lý tất cả các file trên đĩa. Cấu trúc thư mục cũng được ghi trên đĩa và gồm nhiều mục, mỗi mục lưu thông tin của một file.

+ Cấu trúc thư mục có các loại sau: thư mục một cấp, thư mục hai cấp, thư mục đa cấp.

+ Hệ điều hành có thể chia đĩa cứng thành nhiều phân vùng hoặc tập hợp nhiều đĩa cứng thành một phân vùng. Mỗi phân vùng sẽ có cấu trúc thư mục riêng để quản lý các tập tin trong phân vùng đó.

+ Các thao tác trên thư mục tương tự như đối với file và gồm có: Create, Delete, Open, Close, Read, Rename, Link, Unlink,…

+ Bảng thư mục là một dạng cài đặt của cấu trúc thư mục. Bảng thư mục có nhiều mục, mỗi mục lưu trữ thông tin của một file, thông tin file gồm thuộc tính của file và địa chỉ trên đĩa của toàn bộ file hoặc số hiệu của khối đầu tiên chứa file hoặc là số I-node của file. Mỗi đĩa có một bảng thư mục gọi là bảng thư mục gốc, cài đặt ở phần đầu của đĩa và có thể có nhiều bảng thư mục con.

+ Có thể cài đặt hệ thống quản lý file theo phương pháp cấp phát khối nhớ cho file là liên tục hay không liên tục.

+ Để cài đặt hệ thống quản lý file theo phương pháp cấp phát khối nhớ cho file là liên tục, chỉ cần dùng bảng thư mục, mỗi mục trong bảng thư mục ngoài những thuộc tính thông thường của file, cần có thêm thông tin về số hiệu khối bắt đầu (start) và số khối đã cấp cho file (length).

+ Để cài đặt hệ thống quản lý file theo phương pháp cấp phát khối nhớ cho file là không liên tục, sử dụng bảng thư mục và sử dụng thêm một trong các cấu trúc sau: danh sách liên kết/bảng chỉ mục/bảng cấp phát file/cấu trúc I-Nodes. - Cấu trúc danh sách liên kết: mỗi mục trong bảng thư mục chứa số hiệu của khối đầu tiên và số hiệu của khối kết thúc , mỗi khối trên đĩa dành một số byte đầu để lưu số hiệu khối kế tiếp của file, phần còn lại của khối sẽ lưu dữ liệu của file. - Bảng chỉ mục: mỗi file có một bảng chỉ mục chiếm một hoặc vài khối, bảng chỉ mục chứa tất cả các số hiệu khối của một file. Một mục trong bảng thư mục sẽ lưu số hiệu khối chứa bảng chỉ mục của file. - Bảng cấp phát file: nếu mỗi mục trong bảng thư mục chỉ chứa số hiệu của khối đầu tiên, thì số hiệu các khối còn lại của file sẽ được lưu trong một bảng gọi là bảng cấp phát file. - Cấu trúc I-node: mỗi file được quản lý bằng một cấu trúc gọi là cấu trúc I-node, mỗi I-node gồm hai phần: phần thứ nhất lưu trữ thuộc tính file, phần thứ hai gồm 13 phần tử: 10 phần tử đầu chứa 10 số hiệu khối đầu tiên của file, phần tử thứ 11 chứa số hiệu khối chứa bảng single, phần tử thứ 12 chứa số hiệu khối chứa bảng double, phần tử thứ 13 chứa số hiệu khối chứa bảng triple. Trong đó mỗi phần tử của bảng triple chứa số hiệu khối chứa bảng double, mỗi phần tử của bảng double chứa số hiệu khối chứa bảng single và mỗi phần tử của bảng single chứa số hiệu khối dữ liệu tiếp theo cấp cho file.

+ Để quản lý các khối trống phương pháp thường dùng là danh sách liên kết hoặc vector bit.

Danh sách liên kết: mỗi nút là một khối chứa một bảng gồm các số hiệu khối trống, phần tử cuối của bảng lưu số hiệu khối tiếp theo trong danh sách. Vector bit: Bit thứ i = 1 là khối thứ i trống, = 0 là đã sử dụng. Vector bit được lưu trên một hoặc nhiều khối đĩa, khi cần sẽ đọc vào bộ nhớ để xử lý nhanh.

+ Quản lý khối bị hỏng có thể dùng phần mềm: dùng một file chứa các danh sách các khối hỏng. Hoặc dùng phần cứng: dùng một sector trên đĩa để lưu giữ danh sách các khối bị hỏng.

CÂU HỎI VÀ BÀI TẬP

1. Nêu các loại thiết bị nhập/xuất
2. Trình bày đặc tính của thiết bị nhập/xuất
3. Bộ điều khiển thiết bị nhập/xuất (I/O controller) là gì?
4. Nêu các chương trình thực hiện nhập/xuất
5. Nêu cách tổ chức hệ thống nhập/xuất
6. Nêu cơ chế nhập/xuất
6. Trình bày cơ chế DMA
7. Nêu cơ chế truy xuất đĩa
8. Trình bày các thuật toán lập lịch di chuyển đầu đọc
9. Trình bày hệ số đan xen
10. Nêu khái niệm Ram Disks
11. Tại sao dữ liệu không lưu trữ tại vị trí bất kỳ trên đĩa mà lại lưu trữ trên các rãnh (track)?
12. Nêu lý do tại sao cần phân chia đĩa cứng thành những phân vùng (partiton). Thông tin về các phân vùng được quản lý, lưu trữ như thế nào? Trên hệ điều hành MS-DOS hoặc windows, mỗi phân vùng có cần một bảng thư mục riêng không? tại sao?
13. Mục đích của bảng open-files, bảng open-files lưu trữ những thông tin nào?
14. Mục đích của bảng thư mục? phân biệt bảng thư mục gốc và bảng thư mục con.
15. Tại sao hệ điều hành CP/M không cần bảng cấp phát file (FAT)?
16. Nêu ưu/khuyết điểm của việc cấp phát các khối nhớ liên tục cho file.
17. Nêu ưu/khuyết điểm của việc cấp phát các khối nhớ không liên tục cho file.
18. Nêu ưu/khuyết điểm hệ thống quản lý file dùng bảng thư mục và bảng cấp phát file.
19. Nêu ưu/khuyết điểm hệ thống quản lý file dùng cấu trúc I-nodes.
20. Tại sao trong hệ điều hành MS-DOS và hệ điều hành WINDOWS sử dụng FAT, số file hoặc thư mục con trong thư mục gốc bị hạn chế, trong khi số file hoặc thư mục trong thư mụccon lại không bị hạn chế?
21. Cho dãy byte của FAT12 như sau (bắt đầu từ đầu):

240	255	255	0	64	0	9	112	255	255	143	0	255	255	255

Cho biết những phần tử nào của FAT có giá trị đặc biệt, ý nghĩa của phần tử đó.

Nếu sửa lại phần tử 5 là FF0 thì dãy byte của FAT12 này có nội dung như thế nào ?

22. Biết giá trị(dưới dạng thập phân) trong một buffer (mỗi phần tử 1 byte) lưu nội dung của FAT12 như sau (bắt đầu từ phần tử 0):

240	255	255	255	79	0	5	240	255	247	255	255

Cho biết giá trị của từng phần tử trong FAT (dưới dạng số thập phân)

23. Chép 1 File kích thước là 3220 bytes lên một đĩa 1.44Mb còn trống nhưng bị hỏng ở sector logic 33. Cho biết giá trị từng byte của Fat (thập phân) từ byte 0 đến byte 14 .

24. Giả sử một đĩa mềm có 2 side, mỗi side có 128 track, mỗi track có 18 sector. Thư mục gốc của đĩa có tối đa là 251 File (hoặc thư mục). Một cluster = 2 sector. Đĩa sử dụng Fat 12. Hỏi muốn truy xuất cluster 10 thì phải đọc những sector nào ?

25. Hiện trạng của FAT12 và RDET (mỗi entry chỉ gồm tên File và cluster đầu tiên)của một đĩa như sau :

240	255	255	247	79	0	6	0	0	255	159	0	10	240	255	255	127	255

	VD TXT 3	LT DOC 7	THO DAT 8	

Cho biết hiện trạng của FAT12 và RDET sau khi xoá File vd.txt và chép vào File bt.cpp có kích thước 1025 bytes (giả sử 1 cluster = 1 sector)

26. Một File được lưu trên đĩa tại những khối theo thứ tự sau :

20, 32, 34, 39, 52, 63, 75, 29, 37, 38, 47, 49, 56, 68, 79, 81, 92, 106, 157, 159, 160, 162, 163, 267, 269, 271, 277, 278, 279, 380, 381, 482, 489, 490, 499.

Vẽ I_node của File này, giả sử mỗi khối chỉ chứa được 3 phần tử.

27. Viết các lệnh nội trú và ngoại trú của MSDOS bằng ngôn ngữ C và chỉ được sử dụng hai hàm đọc/ghi sector sau:

int absread(int drive, int nsects, long lsect, void *buffer);

int abswrite(int drive, int nsects, long lsect, void *buffer);

TÀI LIỆU THAM KHẢO

[1]. Gary J. Nutt, University of Colorado. Centralized And Distributed Operating Systems. Second Edition, 2000.

[2]. Robert Switzer. Operating Systems, A Practical Approach. Prentice-Hall International, Inc. 1993.

[3]. Andrew S. Tanenbaum. Modern Operating Systems. Prentice-Hall International, Inc. Second Edition, 2001.

[4]. Abraham Silberschatz & Peter Baer Galvin. Operating System concepts. John Wiley & Sons, Inc. Fifth Edition, 1999.

[5]. H. M. Deitel. Operating Systems. Addison-Wesley Inc. Second Edition, 1999.

[6] Cẩm nang lập trình hệ thống cho máy vi tính IBM-PC tập 1 và 2, tác giả Michael Tischer.

[7]. Trần Hạnh Nhi & Lê Khắc Nhiên Ân & Hoàng Kiếm. Giáo trình hệ điều hành (tập 1 & 2). ĐHKHTN 2000.

CHƯƠNG 3
QUẢN LÝ TIẾN TRÌNH

Chương "QUẢN LÝ TIẾN TRÌNH" sẽ giới thiệu và giải thích các vấn đề sau:

3.1 Các khái niệm về tiến trình

3.2 Điều phối các tiến trình

3.3 Liên lạc giữa các tiến trình

3.4 Đồng bộ các tiến trình

3.5 Tính trạng tắc nghẽn (deadlock)

3.1. CÁC KHÁI NIỆM VỀ TIẾN TRÌNH

3.1.1 Tiến trình (Process)

Tiến trình là một chương trình đang xử lý, mỗi tiến trình có một không gian địa chỉ, một con trỏ lệnh, một tập các thanh ghi và stack riêng. Tiến trình có thể cần đến một số tài nguyên như CPU, bộ nhớ chính, các tập tin và thiết bị nhập/xuất. Hệ điều hành sử dụng bộ điều phối (scheduler) để quyết định thời điểm cần dừng hoạt động của tiến trình đang xử lý và lựa chọn tiến trình tiếp theo cần thực hiện. Trong hệ thống có những tiến trình của hệ điều hành và tiến trình của người dùng.

*** Mục đích cho nhiều tiến trình hoạt động đồng thời:**

a/ Tăng hiệu suất sử dụng CPU (tăng mức độ đa chương):

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

CPU	IO	CPU	IO	CPU

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

Tiến trình 1:

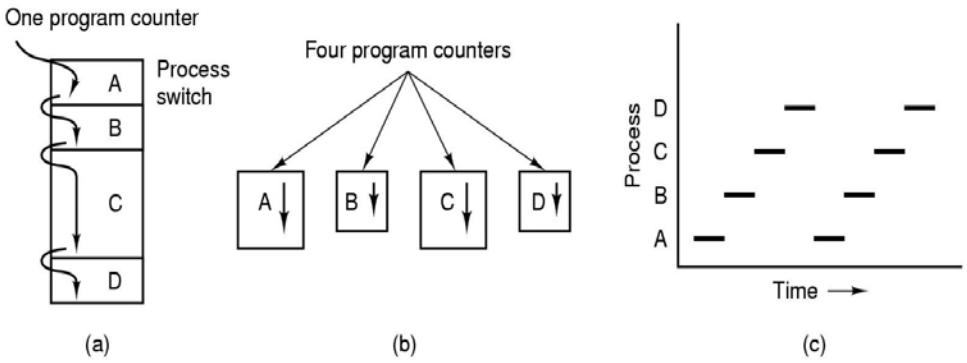
CPU	IO	CPU	IO	CPU

Tiến trình 2:

	CPU	IO	CPU	IO

b/ Tăng mức độ đa nhiệm

Cho mỗi tiến trình thực thi luân phiên trong một thời gian rất ngắn, tạo cảm giác là hệ thống có nhiều tiến trình thực thi đồng thời.



Hình 3.1: a) A,B,C,D thực thi tuần tự chỉ cần sử dụng một con trỏ lệnh. b) A,B,C,D thực thi đồng thời bằng cách chia xẻ CPU và sử dụng bốn con trỏ lệnh.

c/ Tăng tốc độ xử lý:

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

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

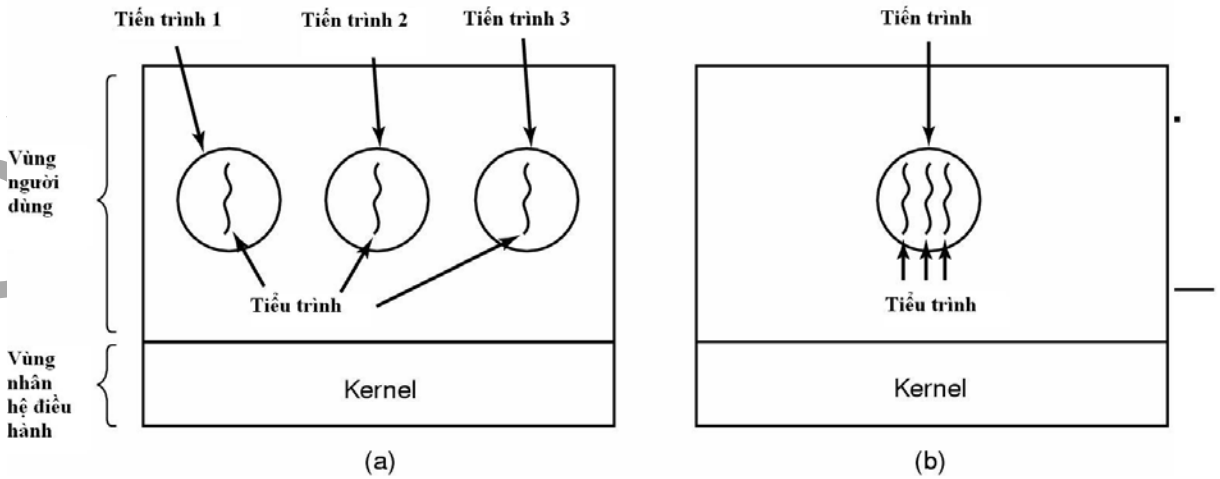
3.1.2 Tiểu trình (thread)

3.1.2.1 Khái niệm tiểu trình

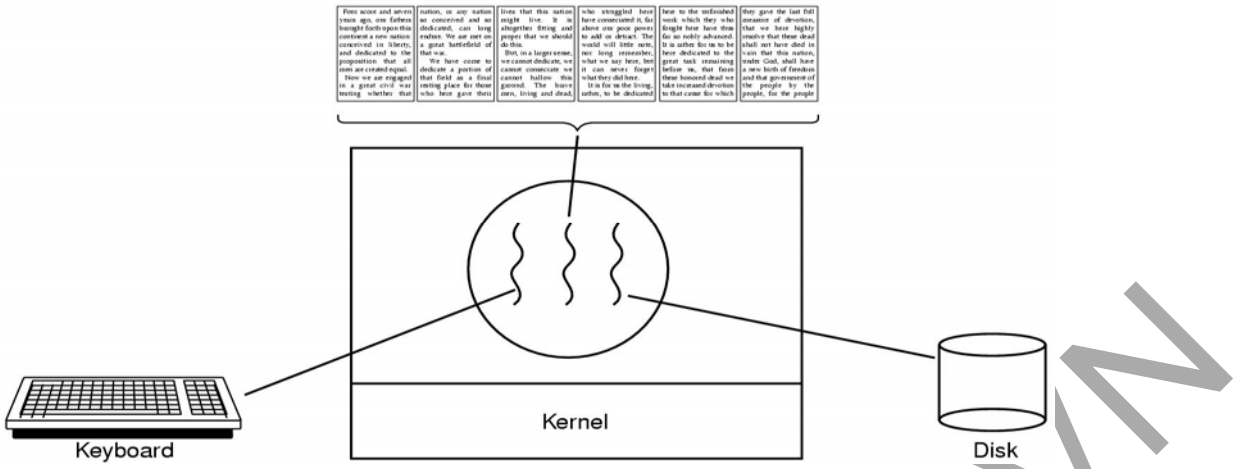
Một tiến trình có thể tạo nhiều tiểu trình, mỗi tiểu trình thực hiện một chức năng nào đó và thực thi đồng thời cũng bằng cách chia sẻ CPU. Các tiểu trình trong cùng một tiến trình dùng chung không gian địa chỉ tiến trình nhưng có con trỏ lệnh, tập các thanh ghi và stack riêng. Một tiểu trình cũng có thể tạo lập các tiến trình con, và nhận các trạng thái khác nhau như một tiến trình.

3.1.2.2 Liên lạc giữa các tiểu trình

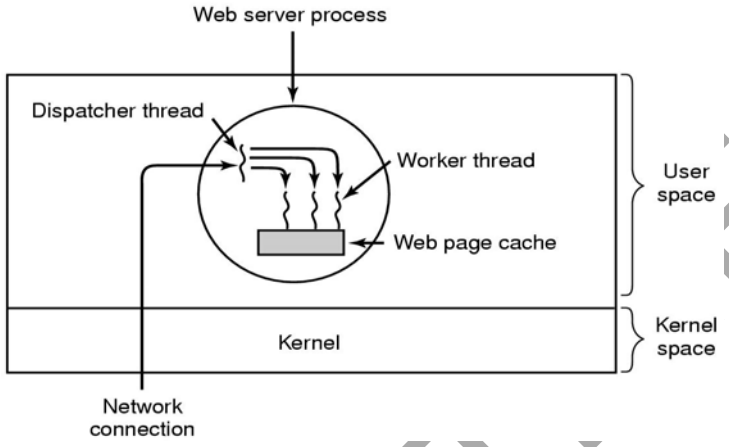
Các tiến trình chỉ có thể liên lạc với nhau thông qua các cơ chế do hệ điều hành cung cấp. Các tiểu trình liên lạc với nhau dễ dàng thông qua các biến toàn cục của tiến trình. Các tiểu trình có thể do hệ điều hành quản lý hoặc hệ điều hành và tiến trình cùng phối hợp quản lý.



Hình 3.2: a) ba tiến trình thực thi đồng thời, mỗi tiến trình chỉ có một tiểu trình. b) một tiến trình có ba tiểu trình, việc hoạt động đồng thời của các tiểu trình là do tiến trình quản lý.



Hình 3.3: một chương trình xử lý văn bản có ba thread: một thread nhận các kí tự nhập từ bàn phím, một thread hiện văn bản, một thread ghi văn bản vào đĩa.



Hình 3.4: web server có hai thread: worker thread và dispatcher thread, việc hoạt động đồng thời của các tiểu trình là do tiến trình quản lý.

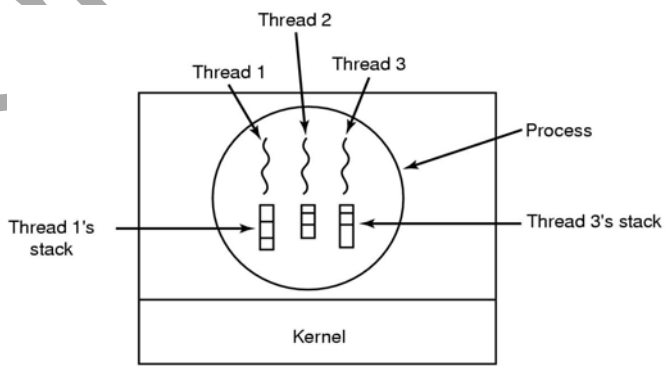
(a) đoạn mã cho dispatcher thread (b) đoạn mã cho worker thread

```
while (TRUE) {
  get_next_request(&buf);
  handoff_work(&buf);
}
```

(a)

```
while (TRUE) {
  wait_for_work(&buf)
  look_for_page_in_cache(&buf, &page);
  if (page_not_in_cache(&page)
      read_page_from_disk(&buf, &page);
  return_page(&page);
}
```

(b)



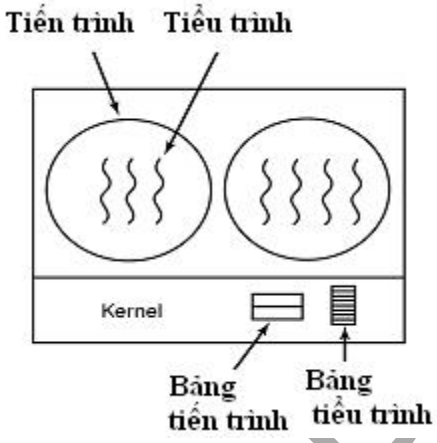
Hình 3.5: một process có ba thread, mỗi thread sẽ có stack riêng.

Trong bảng dưới đây, tất cả các thread trong cùng process dùng chung các mục ở cột 1, nhưng mỗi thread sẽ có riêng các mục ở cột 2

Trong cùng tiến trình	Trong mỗi tiểu trình
Không gian địa chỉ	Bộ đếm chương trình
Các biến toàn cục	Các thanh ghi
Các tập tin mở	Ngăn xếp
Các tiến trình con	Trạng thái
Các cảnh báo	
Các tín hiệu và các bộ xử lý tín hiệu	
Thông tin tài khoản	

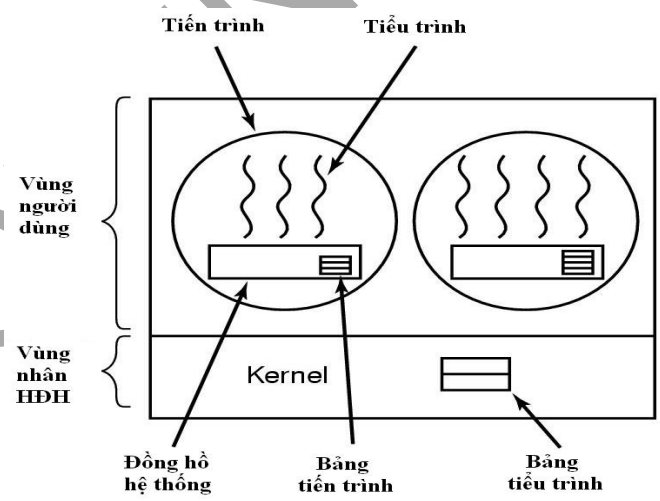
3.1.2.3 Cài đặt tiểu trình (Threads)

a/ Cài đặt trong Kernel-space : bảng quản lý thread lưu trữ ở phần kernel và việc điều phối các thread là do hệ điều hành chịu trách nhiệm.



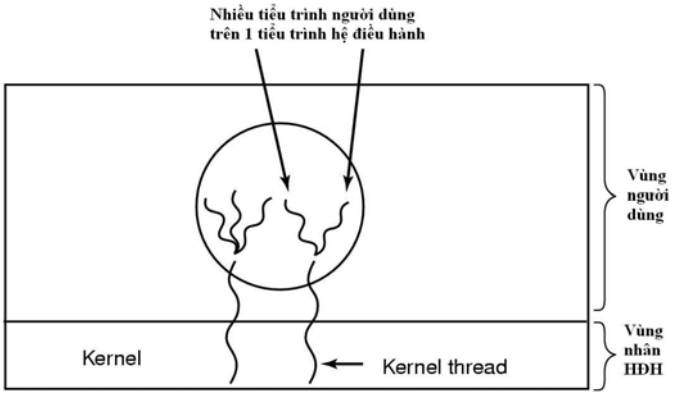
Hình 3.6: hệ điều hành chịu trách nhiệm điều phối các tiểu trình

b/ Cài đặt trong User-space: bảng quản lý thread lưu trữ ở phần user-space và việc điều phối các thread là do tiến trình chịu trách nhiệm.



Hình 3.7: tiến trình chịu trách nhiệm điều phối các tiểu trình thuộc tiến trình đó

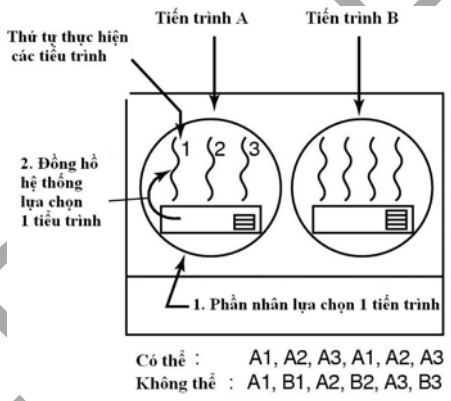
c/ Cài đặt trong Kernel-space và User-space: Một số thread mức User được cài đặt bằng một thread mức kernel.



Hình 3.8: một thread của hệ điều hành quản lý một số thread của tiến trình.

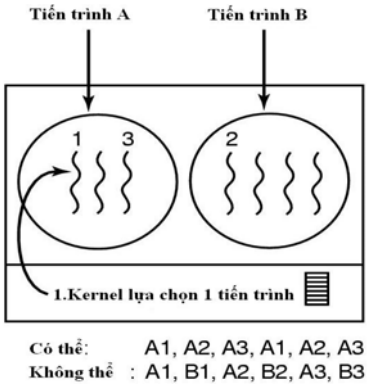
Ví dụ: giả sử quantum của process=50 msec, quantum của thread=5 msec và giả sử tiến trình A có ba thread, tiến trình B có 4 thread.

- Nếu việc điều phối thread được thực hiện mức user-space thì thứ tự điều phối có thể là A1, A2, A3, A1, A2, A3 nhưng không thể là A1, B1, A2, B2, A3, B3; vì khi tiến trình A được cho thực thi với quantum=50 và mỗi thread được thực thi với quantum=5 thì không thể A1 đến B1 được do thread của tiến trình nào tiến trình đó quản lý và tiến trình A chưa hết quantum nên thread của tiến trình B không thể thực hiện.



Hình 3.9: điều phối thread ở mức user, một thứ tự điều phối có thể và không thể

- Nếu việc điều phối thread được thực hiện mức kernel-space thì thứ tự điều phối A1 đến B1 là có thể vì các thread do hệ điều hành quản lý



Hình 3.10: điều phối thread ở mức kernel, một thứ tự điều phối có thể và không thể.

3.1.3 Các trạng thái của tiến trình

Việc chuyển trạng thái của tiến trình là do bộ điều phối (scheduler) thực hiện và tại một thời điểm, tiến trình có thể nhận một trong các trạng thái sau đây :

a/ New: tiến trình mới được tạo đang ở trong bộ nhớ tạm trên đĩa cứng.

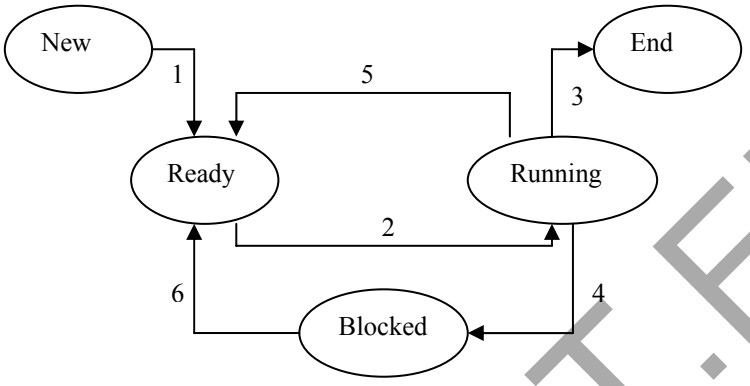
b/ Ready: tiến trình trong bộ nhớ và chờ được cấp phát CPU.

c/ Running: tiến trình trong bộ nhớ đang thực thi.

d/ Blocked (wait): tiến trình trong bộ nhớ chờ được cấp phát tài nguyên, hoặc chờ thao tác nhập/xuất hoàn tất hoặc chờ một sự kiện nào đó.

e/ End: tiến trình trong bộ nhớ hoàn tất xử lý.

3.1.3.1 Sơ đồ chuyển trạng thái tiến trình



Hình 3.11: sơ đồ chuyển trạng thái giữa các tiến trình.

Tại một thời điểm, chỉ có một tiến trình có thể nhận trạng thái running trên một bộ xử lý nào đó. Trong khi đó, có thể có nhiều tiến trình ở trạng thái blocked hay ready. Các cung chuyển tiếp trong sơ đồ trạng thái biểu diễn sáu sự chuyển trạng thái có thể xảy ra trong các điều kiện sau :

- Cung 1: Tiến trình mới tạo, nếu bộ nhớ còn trống, sẽ được đưa vào bộ nhớ và sẵn sàng nhận CPU, khi đó tiến trình từ trạng thái New được chuyển sang trạng thái Ready.

- Cung 2: Bộ điều phối cấp phát cho tiến trình một khoảng thời gian sử dụng CPU và cho tiến trình thực hiện, khi đó tiến trình từ trạng thái Ready được chuyển sang trạng thái Running.

- Cung 3: Khi tiến trình kết thúc việc thực hiện, khi đó tiến trình từ trạng thái Running được chuyển sang trạng thái End.

- Cung 4: Khi tiến trình yêu cầu một tài nguyên nhưng chưa được đáp ứng vì tài nguyên chưa sẵn sàng hoặc tiến trình chờ thao tác nhập/xuất hoàn tất hoặc tiến trình chờ một sự kiện nào đó, khi đó tiến trình được chuyển từ trạng thái Running sang trạng thái Blocked.

- Cung 5: Khi tiến trình tạm dừng vì hết thời gian sử dụng CPU, bộ điều phối sẽ chọn một tiến trình khác để cho xử lý, khi đó tiến trình được chuyển từ trạng thái Running sang trạng thái Ready.

- Cung 6: Khi tài nguyên mà tiến trình yêu cầu trở nên sẵn sàng để cấp phát ; hay sự kiện hoặc thao tác nhập/xuất mà tiến trình đang đợi đã hoàn tất, khi đó bộ tiến trình được chuyển từ trạng thái Blocked sang trạng thái Ready.

3.1.3.2 Các chế độ xử lý của tiến trình

+ Tập lệnh của CPU được phân chia thành tập lệnh đặc quyền (các lệnh nếu sử dụng không chính xác, có thể ảnh hưởng xấu đến hệ thống) và tập lệnh không đặc quyền (không ảnh hưởng tới hệ thống). Phần cứng chỉ cho phép các lệnh đặc quyền được thực hiện trong chế độ đặc quyền.

+ Thông thường chỉ có hệ điều hành hoạt động trong chế độ đặc quyền, các tiến trình của người dùng sẽ hoạt động trong chế độ không đặc quyền.

3.1.4 Khối quản lý tiến trình (Process Control Block: PCB)

Hệ điều hành quản lý các tiến trình thông qua bảng tiến trình (process table), mỗi mục trong bảng gọi là PCB (khối quản lý tiến trình), PCB lưu thông tin về một tiến trình gồm có các thông tin sau:

a/ Định danh tiến trình: mã số tiến trình, giúp phân biệt tiến trình này với tiên trình khác

b/ Trạng thái tiến trình: xác định hoạt động hiện hành của tiến trình.

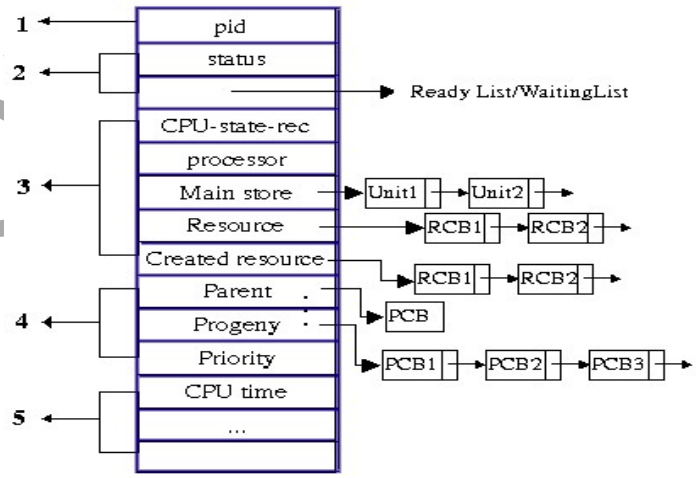
c/ Ngữ cảnh tiến trình: mô tả các tài nguyên tiến trình đang sử dụng, dùng để phục vụ cho hoạt động hiện tại, hoặc để làm cơ sở phục hồi hoạt động cho tiến trình. Ngữ cảnh tiến trình bao gồm các thông tin sau:

- Trạng thái CPU: bao gồm nội dung các thanh ghi, quan trọng nhất là con trỏ lệnh IP lưu trữ địa chỉ lệnh kế tiếp mà tiến trình sẽ thực hiện. Các thông tin này cần được lưu trữ khi xảy ra một ngắt, nhằm có thể cho phép phục hồi hoạt động của tiến trình đúng như trước khi bị ngắt.
- Số hiệu bộ xử lý: xác định số hiệu CPU mà tiến trình đang sử dụng.
- Bộ nhớ chính: danh sách các khối nhớ được cấp cho tiến trình.
- Tài nguyên sử dụng: danh sách các tài nguyên hệ thống mà tiến trình đang sử dụng.
- Tài nguyên tạo lập: danh sách các tài nguyên được tiến trình tạo lập.

d/ Thông tin giao tiếp: phản ánh các thông tin về quan hệ của tiến trình với các tiến trình khác trong hệ thống gồm có các thông tin sau:

- Tiến trình cha: tiến trình tạo lập tiến trình này.
- Tiến trình con: các tiến trình do tiến trình này tạo lập.
- Độ ưu tiên : giúp bộ điều phối có thông tin để lựa chọn tiến trình được cấp CPU.

e/ Thông tin thống kê: đây là những thông tin thống kê về hoạt động của tiến trình, như thời gian đã sử dụng CPU, thời gian chờ. Các thông tin này có thể có ích cho công việc đánh giá tình hình hệ thống và dự đoán các tình huống tương lai.



Hình 3.12: Cấu trúc của khối quản lý tiến trình (PCB)

Có thể liệt kê thông tin trong PCB theo chức năng quản lý như sau:

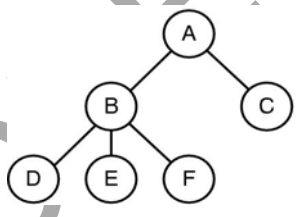
Quản lý tiến trình	**Quản lý bộ nhớ**	**Quản lý tập tin**
Các thanh ghi	Con trỏ tới đoạn văn bản	Thư mục gốc
Bộ đếm chương trình	Con trỏ tới đoạn dữ liệu	Thư mục làm việc
Trạng thái chương trình	Con trỏ tới đoạn stack	Các mô tả tập tin
Con trỏ Stack		ID người dùng
Tình trạng của tiến trình		ID nhóm
Độ ưu tiên		
Các tham số điều phối		
ID của tiến trình		
Tiến trình cha		
Nhóm tiến trình		
Các tín hiệu		
Thời điểm bắt đầu tiến trình		
Thời gian CPU sử dụng		
Thời gian CPU của tiến trình con		
Thời gian lần cảnh báo kế tiếp		

Hình 3.13: thông tin trong khối PCB được liệt kê theo chức năng quản lý

3.1.5 Các thao tác trên tiến trình

a/ Tạo lập tiến trình (create)

Trong quá trình xử lý, một tiến trình có thể tạo lập nhiều tiến trình mới bằng cách sử dụng một lời gọi hệ thống tương ứng. Tiến trình gọi lời gọi hệ thống để tạo tiến trình mới sẽ được gọi là tiến trình cha, tiến trình được tạo gọi là tiến trình con. Mỗi tiến trình con đến lượt nó lại có thể tạo các tiến trình mới…quá trình này tiếp tục sẽ tạo ra một cây tiến trình (trong Windows không có khái niệm cây tiến trình, mọi tiến trình là ngang cấp). Khi một tiến trình tạo lập một tiến trình con, tiến trình con có thể sẽ được hệ điều hành trực tiếp cấp phát tài nguyên hoặc được tiến trình cha cho thừa hưởng một số tài nguyên ban đầu. Khi tiến trình cha tạo tiến trình con, tiến trình cha có thể xử lý theo một trong hai khả năng sau: tiến trình cha tiếp tục xử lý đồng hành với tiến trình con, hoặc tiến trình cha chờ đến khi một tiến trình con nào đó, hoặc tất cả các tiến trình con kết thúc xử lý. Ví dụ: tiến trình A tạo hai tiến trình con B và C, B tạo ba tiến trình con D, E, F.



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

+ Các công việc cần thực hiện khi tạo lập tiến trình:

- Định danh cho tiến trình mới phát sinh
- Đưa tiến trình vào danh sách quản lý của hệ thống
- Xác định độ ưu tiên cho tiến trình

- Cấp phát các tài nguyên ban đầu cho tiến trình
- Tạo PCB lưu trữ thông tin tiến trình

+ Các thời điểm tiến trình được tạo ra :

Tiến trình được tạo ra vào một trong các thời điểm sau:

- Thời điểm khởi tạo hệ thống (System initialization)
- Thời điểm thực thi lời gọi tạo tiến trình
- Thời điểm người sử dụng yêu cầu tạo tiến trình mới
- Thời điểm khởi đầu một công việc theo lô (batch job)

b/ Kết thúc tiến trình (destroy)

Một tiến trình kết thúc xử lý khi nó hoàn tất lệnh cuối cùng và sử dụng một lời gọi hệ thống để yêu cầu hệ điều hành hủy bỏ nó. Một tiến trình có thể yêu cầu hệ điều hành kết thúc xử lý của một tiến trình khác.

+ Khi một tiến trình kết thúc hệ điều hành cần thực hiện các công việc sau:

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

Hầu hết các hệ điều hành không cho phép các tiến trình con tiếp tục tồn tại nếu tiến trình cha đã kết thúc. Trong những hệ thống như thế, hệ điều hành sẽ tự động phát sinh một loạt các thao tác kết thúc tiến trình con. Tiến trình có thể tự kết thúc bình thường (Normal exit) do đã thực thi xong hoặc có lỗi và tự kết thúc (Error exit) hoặc có lỗi nặng và bị hệ điều hành kết thúc (Fatal exit) hoặc bị kết thúc bởi tiến trình khác (Killed by another process).

c/ Tạm dừng tiến trình (suspend)

d/ Tái kích hoạt tiến trình (resume)

e/ Thay đổi độ ưu tiên tiến trình (change priority)

3.1.6 Khối quản lý tài nguyên (Resource Control Block: RCB)

Mỗi tài nguyên được hệ điều hành quản lý bằng một cấu trúc gọi là khối quản lý tài nguyên RCB. RCB khác nhau về chi tiết cho từng loại tài nguyên, nhưng cơ bản có các thông tin sau:

a/ Định danh tài nguyên: dùng để phân biệt tài nguyên này với tài nguyên khác.

b/ Trạng thái tài nguyên: mô tả chi tiết trạng thái tài nguyên, phần nào của tài nguyên đã cấp phát cho tiến trình, phần nào còn có thể sử dụng.

c/ Hàng đợi trên tài nguyên: danh sách các tiến trình đang chờ được cấp phát tài nguyên tương ứng.

d/ Bộ cấp phát tài nguyên: là đoạn mã đảm nhiệm việc cấp phát tài nguyên. Một số tài nguyên đòi hỏi các giải thuật đặc biệt (như CPU, bộ nhớ chính, hệ thống tập tin), trong khi những tài nguyên khác (như các thiết bị nhập/xuất) có thể cần các giải thuật cấp phát và giải phóng tổng quát hơn.

RCB	Ý nghĩa
Định danh tài nguyên	rid
Trạng thái tài nguyên	Danh sách các phần của tài nguyên có thể sử dụng
Hàng đợi	Danh sách các tiến trình đang đợi tài nguyên
Bộ cấp phát	Con trỏ đến bộ cấp phát tài nguyên

Hình 3.14: thông tin trong khối RCB

+ Mục tiêu của bộ cấp phát tài nguyên :

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

3.2 ĐIỀU PHỐI TIẾN TRÌNH

Hệ điều hành điều phối tiến trình thông qua bộ điều phối (scheduler) và bộ phân phối (dispatcher). Bộ điều phối sử dụng một giải thuật thích hợp để lựa chọn tiến trình được xử lý tiếp theo. Bộ phân phối chịu trách nhiệm cập nhật ngữ cảnh của tiến trình bị tạm ngưng và trao CPU cho tiến trình được chọn bởi bộ điều phối để tiến trình thực thi.

3.2.1 Mục tiêu của bộ điều phối

a/ Sự công bằng (Fairness): Các tiến trình chia sẻ CPU một cách công bằng, không có tiến trình nào phải chờ đợi vô hạn để được cấp phát CPU.

b/ Tính hiệu qủa (Efficiency): Hệ thống phải tận dụng được CPU 100% thời gian.

c/ Thời gian đáp ứng hợp lý (Response time): Cực tiểu hoá thời gian hồi đáp cho các tương tác của người sử dụng.

d/ Thời gian lưu lại trong hệ thống (Turnaround Time): Cực tiểu hóa thời gian hoàn tất các tác vụ xử lý theo lô.

e/ Thông lượng tối đa (Throughput): Cực đại hóa số công việc được xử lý trong một đơn vị thời gian.

Thường hệ điều hành khó thể thỏa mãn tất cả các mục tiêu kể trên mà chỉ có thể dung hòa. Để việc điều phối có hiệu qủa, hệ điều hành cần quan tâm đến đặc tính của tiến trình.

3.2.2 Các đặc tính của tiến trình

a/ Tính hướng nhập/xuất(I/O-boundedness):

Tiến trình khi thực thi, chủ yếu thực hiện thao tác nhập xuất, rất ít lệnh xử lý. Tiến trình có khuynh hướng không sử dụng CPU đến hết thời gian dành cho nó. Hoạt động của các tiến trình như thế thường bao gồm nhiều lượt sử dụng CPU, mỗi lượt trong một thời gian khá ngắn.

b/ Tính hướng xử lý(CPU-boundedness):

Tiến trình khi thực thi, chủ yếu thực hiện thao tác xử lý, rất ít thao tác nhập/xuất. Tiến trình có khuynh hướng sử dụng CPU đến khi hết thời gian dành cho nó. Hoạt động của các tiến trình như thế thường bao gồm một số ít lượt sử dụng CPU, nhưng mỗi lượt trong một thời gian đủ dài.

c/ Tiến trình tương tác hay xử lý theo lô :

Người sử dụng theo kiểu tương tác thường yêu cầu được hồi đáp tức thời đối với các yêu cầu của họ, trong khi các tiến trình của tác vụ được xử lý theo lô nói chung có thể trì hoãn trong một thời gian chấp nhận được.

d/ Độ ưu tiên của tiến trình

Các tiến trình có thể được phân cấp ưu tiên theo một số tiêu chuẩn nào đó. Các tiến trình có độ ưu tiên cao cần thực hiện trước.

e/ Thời gian đã sử dụng CPU của tiến trình

Một số quan điểm ưu tiên chọn những tiến trình đã sử dụng CPU nhiều thời gian nhất vì hy vọng chúng sẽ cần ít thời gian nhất để hoàn tất và rời khỏi hệ thống . Tuy nhiên cũng có quan điểm cho rằng các tiến trình nhận được CPU trong ít thời gian là những tiến trình đã phải chờ lâu nhất, do vậy ưu tiên chọn chúng.

f/ Thời gian còn lại tiến trình cần để hoàn tất

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

Khi thực hiện điều phối, cần quyết định thời điểm chuyển đổi CPU giữa các tiến trình, hệ điều hành có thể dựa vào các nguyên lý sau:

3.2.3 Các nguyên lý điều phối

3.2.3.1 Điều phối độc quyền (preemptive):

Tiến trình khi nhận được CPU sẽ được độc chiếm CPU đến khi hoàn tất xử lý hoặc tự nguyện giải phóng CPU. Các giải thuật độc quyền thường đơn giản và dễ cài đặt nhưng không thích hợp với các hệ thống nhiều người dùng, vì nếu cho phép một tiến trình có quyền xử lý bao lâu tùy ý, tiến trình này có thể giữ CPU một thời gian không xác định, có thể ngăn cản những tiến trình còn lại trong hệ thống có một cơ hội để xử lý. Điều phối độc quyền cũng có thể xảy ra tình trạng các tác vụ cần thời gian xử lý ngắn phải chờ tác vụ xử lý với thời gian rất dài hoàn tất.

3.2.3.2 Điều phối không độc quyền (nopreemptive):

Khi một tiến trình nhận được CPU, nó vẫn được sử dụng CPU đến khi hoàn tất hoặc tự nguyện giải phóng CPU, nhưng nếu xuất hiện một tiến trình khác có độ ưu tiên cao hơn thì hệ điều hành sẽ cho tiến trình có độ ưu tiên cao hơn dành quyền sử dụng CPU của tiến trình ban đầu.

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

Đối với các hệ thống tương tác, các hệ thời gian thực (real time), cần điều phối không độc quyền để các tiến trình quan trọng có cơ hội hồi đáp kịp thời. Tuy nhiên thực hiện điều phối không độc quyền đòi hỏi những cơ chế phức tạp trong việc phân định độ ưu tiên, và phát sinh thêm chi phí khi chuyển đổi CPU qua lại giữa các tiến trình. Vấn đề đặt ra cho hệ điều hành là thời điểm nào cần thực hiện điều phối.

+ Thời điểm thực hiện điều phối

Hệ điều hành thực hiện việc điều phối tiến trình khi có một trong các tình huống sau:

a/ Tiến trình chuyển từ trạng thái running sang trạng thái blocked: ví dụ chờ một thao tác nhập xuất hay chờ một tiến trình con kết thúc…

b/ Tiến trình chuyển từ trạng thái running sang trạng thái ready: ví dụ xảy ra một ngắt.

c/ Tiến trình chuyển từ trạng thái blocked sang trạng thái ready: ví dụ một thao tác nhập/xuất hoàn tất.

d/ Tiến trình kết thúc.

e/ Tiến trình có độ ưu tiên cao hơn xuất hiện: chỉ áp dụng đối với điều phối không độc quyền

3.2.4 Tổ chức điều phối

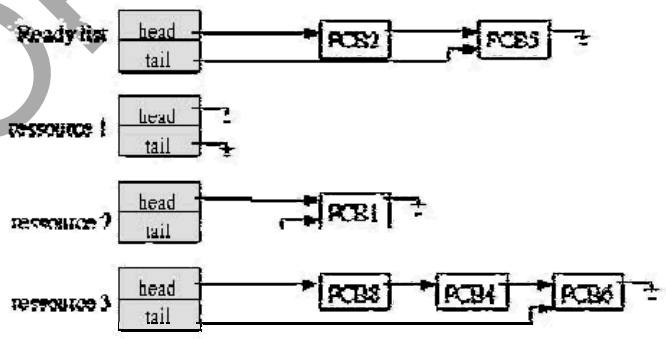
3.2.4.1 Các danh sách điều phối

Để thực hiện điều phối, hệ điều hành sử dụng ba loại danh sách là: danh sách tác vụ (job list), danh sách sẵn sàng (ready list), danh sách chờ đợi (waiting list).

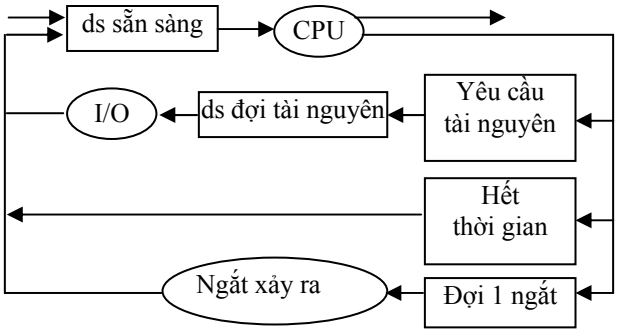
Khi một tiến trình được tạo, PCB của tiến trình sẽ được chèn vào danh sách tác vụ (job list). Khi bộ nhớ đủ chỗ, một tiến trình trong danh sách tác vụ được chọn, nạp từ đĩa vào bộ nhớ và PCB của tiến trình đó được chuyển sang danh sách sẵn sàng (ready list). Bộ điều phối sẽ chọn một tiến trình trong danh sách sẵn sàng và cấp CPU cho tiến trình đó. Tiến trình được cấp CPU sẽ thi hành, và sẽ chuyển sang danh sách chờ đợi (waiting list) khi xảy ra các sự kiện ví dụ như đợi một thao tác nhập/xuất hoàn tất hoặc yêu cầu tài nguyên mà chưa được thỏa mãn hoặc được yêu cầu tạm dừng ...

Tiến trình đang thi hành có thể bị bắt buộc tạm dừng xử lý do một ngắt xảy ra, khi đó tiến trình được đưa trở lại vào danh sách sẵn sàng để chờ được cấp CPU cho lượt tiếp theo.

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



Hình 3.15: Mỗi tài nguyên có một hàng đợi lưu danh sách các tiến trình đang đợi tài nguyên.



Hình 3.16: khi có yêu cầu tài nguyên mà chưa được đáp ứng, tiến trình được đưa vào hàng đợi tài nguyên.

3.2.4.2 Các loại điều phối

a) Điều phối tác vụ (job scheduling)

Là lựa chọn tác vụ nào được đưa vào bộ nhớ chính để thực hiện. Chức năng điều phối tác vụ quyết định mức độ đa chương của hệ thống (số lượng tiến trình trong bộ nhớ chính). Khi hệ thống tạo lập một tiến trình, hay có một tiến trình kết thúc xử lý thì chức năng điều phối tác vụ mới được kích hoạt. Vì mức độ đa chương tương đối ổn định nên chức năng điều phối tác vụ có tần suất hoạt động thấp . Để cân bằng hoạt động của CPU và các thiết bị ngoại vi, bộ điều phối tác vụ nên lựa chọn các tiến trình để nạp vào bộ nhớ sao cho là sự pha trộn hợp lý giữa các tiến trình hướng nhập xuất và các tiến trình hướng xử lý.

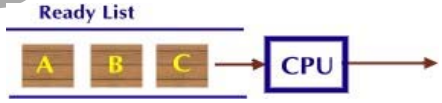
b) Điều phối tiến trình (process scheduling)

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

3.2.4.3 Các thuật toán điều phối

a/ Thuật toán FIFO

CPU được cấp phát cho tiến trình đầu tiên trong danh sách sẵn sàng, tiến trình này là tiến trình được đưa vào hệ thống sớm nhất. Đây là thuật toán điều phối theo nguyên tắc độc quyền.



Hình 3.17: mô hình điều phối theo FIFO

Ví dụ: Hệ thống lần lượt có ba tiến trình P1, P2, P3 vào ready list. Thời điểm vào RL và thời gian xử lý của mỗi tiến trình cho trong bảng sau:

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

Theo thuật toán FIFO, thứ tự cấp phát CPU cho các tiến trình là :

P1	P2	P3	
0	24	27	30

Thời gian chờ đợi được xử lý là 0 đối với P1, (24 -1) với P2 và (27-2) với P3. Thời gian chờ trung bình là (0+23+25)/3 = 16 miliseconds.

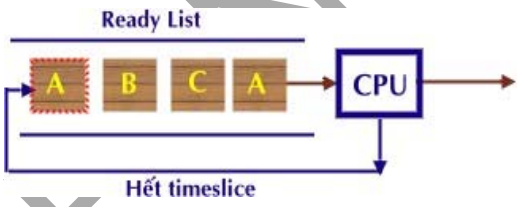
Nhận xét:

- Có thể một tiến trình có thời gian xử lý ngắn phải chờ một tiến trình có thời gian xử lý dài thực thi xong.

- Thời gian chờ trung bình phụ thuộc vào thứ tự của các tiến trình trong danh sách sẵn sàng.

b/ Thuật toán phân phối xoay vòng (Round Robin)

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



Hình 3.18: mô hình điều phối theo round robin

Ví dụ:

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

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

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

Thời gian chờ đợi trung bình sẽ là (6+3+5)/3 = 4.66 milisecondes.

Nhận xét:

- Nếu có n tiến trình trong danh sách sẵn sàng và sử dụng quantum q, thì mỗi tiến trình sẽ không phải đợi quá (n-1)q đơn vị thời gian trước khi nhận được CPU cho lượt kế tiếp.

- Nếu thời lượng quantum quá bé sẽ phát sinh nhiều sự chuyển đổi giữa các tiến trình và khiến cho việc sử dụng CPU kém hiệu qủa. Nhưng nếu sử dụng quantum quá lớn sẽ làm giảm khả năng tương tác của hệ thống.

c/ Thuật toán độ ưu tiên

Mỗi tiến trình được gán cho một độ ưu tiên, tiến trình có độ ưu tiên cao nhất sẽ được chọn để cấp phát CPU đầu tiên. Độ ưu tiên có thể được định nghĩa nhờ vào các yếu tố bên trong hay bên ngoài. Yếu tố bên trong như là giới hạn thời gian, nhu cầu bộ nhớ…Yếu tố bên ngoài như là tầm quan trọng của tiến trình, loại người sử dụng sở hữu tiến trình…

Giải thuật điều phối với độ ưu tiên có thể theo nguyên tắc độc quyền hay không độc quyền. Khi một tiến trình được đưa vào danh sách các tiến trình sẵn sàng, độ ưu tiên của nó được so sánh với độ ưu tiên của tiến trình hiện hành đang xử lý.

Giải thuật không độc quyền sẽ thu hồi CPU từ tiến trình hiện hành để cấp phát cho tiến trình mới nếu độ ưu tiên của tiến trình mới cao hơn tiến trình hiện hành. Giải thuật độc quyền sẽ chỉ đơn giản chèn tiến trình mới vào danh sách sẵn sàng theo thứ tự độ ưu tiên, và tiến trình hiện hành vẫn tiếp tục xử lý hết thời gian dành cho nó.

Ví dụ : giả sử độ ưu tiên 1 > độ ưu tiên 2> độ ưu tiên 3

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

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

P1	P2	P3
0	24	27 30

Thời gian chờ đợi trung bình sẽ là (0+23+25)/3 =16 milisecondes.

Nếu sử dụng thuật giải độ ưu tiên không độc quyền, thứ tự cấp phát CPU như sau :

P1	P2	P3	P1
0	1	4	7 30

Thời gian chờ đợi trung bình sẽ là (6+0+2)/3 = 2.7 milisecondes.

Nhận xét:

- Tình trạng 'đói CPU' (starvation) là một vấn đề chính yếu của các giải thuật sử dụng độ ưu tiên. Các giải thuật này có thể để các tiến trình có độ ưu tiên thấp chờ đợi CPU vô hạn!

- Để ngăn cản các tiến trình có độ ưu tiên cao chiếm dụng CPU vô thời hạn, bộ điều phối sẽ giảm dần độ ưu tiên của các tiến trình này sau mỗi ngắt đồng hồ (khoảng 100ms). Nếu độ ưu tiên của tiến trình này giảm xuống thấp hơn tiến trình có độ ưu tiên cao thứ nhì, sẽ xảy ra sự chuyển đổi quyền sử dụng CPU. Quá trình này gọi là sự 'lão hóa' (aging) tiến trình.

d/ Thuật toán công việc ngắn nhất (Shortest-job-first SJF)

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

Ví dụ :

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

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

P1	P4	P3	P2
0	6	8	12 20

Thời gian chờ đợi trung bình sẽ là (0+11+6+3)/4 = 5 milisecondes.

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

P1	P4	P1	P3	P2
0	3	5	8	12 20

Thời gian chờ đợi trung bình sẽ là (2+11+6+0)/3 = 6.33 milisecondes.

Nhận xét:

Giải thuật này cho phép đạt được thời gian chờ trung bình cực tiểu. Khó khăn thực sự của giải thuật SJF là thường không thể biết được thời gian yêu cầu xử lý còn lại của tiến trình.

e/ Thuật toán nhiều mức độ ưu tiên

Danh sách sẵn sàng được chia thành nhiều danh sách, mỗi danh sách gồm các tiến trình có cùng độ ưu tiên và được áp dụng một giải thuật điều phối riêng. Ngoài ra, cần có một giải thuật điều phối giữa các danh sách, thường giải thuật này là giải thuật không độc quyền và sử dụng độ ưu tiên cố định. Một tiến trình thuộc về một danh sách nào đó chỉ được cấp phát CPU khi các danh sách có cấp độ ưu tiên cao hơn đã trống.

Hình 3.19: mô hình điều phối theo nhiều mức ưu tiên

Nhận xét:

- Có thể dẫn đến tình trạng "đói CPU" cho các tiến trình thuộc về những danh sách có độ ưu tiên thấp. Do vậy có thể xây dựng giải thuật điều phối nhiều cấp ưu tiên và xoay vòng (Multilevel Feedback). Giải thuật này sẽ chuyển dần một tiến trình từ danh sách có độ ưu tiên cao xuống danh sách có độ ưu tiên thấp hơn sau một khoảng thời gian nào đó. Cũng vậy, một tiến trình chờ quá lâu trong các danh sách có độ ưu tiên thấp cũng được chuyển dần lên các danh sách có độ ưu tiên cao hơn.

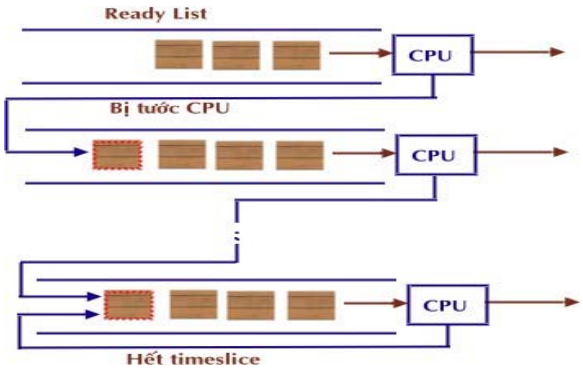
- Khi xây dựng giải thuật điều phối nhiều cấp ưu tiên và xoay vòng cần quan tâm các vấn đề sau :

- Số lượng các cấp ưu tiên

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

- Phương pháp xác định thời điểm di chuyển một tiến trình lên danh sách có độ ưu tiên cao hơn và phương pháp xác định thời điểm di chuyển một tiến trình xuống danh sách có độ ưu tiên thấp hơn.

- Phương pháp xác định một tiến trình mới được đưa vào hệ thống sẽ thuộc danh sách có độ tiên nào.



Hình 3.20: mô hình điều phối theo nhiều mức ưu tiên xoay vòng.

+ Chiến lược điều phối xổ số (Lottery)

Phát một vé số cho mỗi tiến trình khi vào hệ thống. Khi đến thời điểm ra quyết định điều phối, sẽ chọn 1 vé "trúng giải", tiến trình nào sỡ hữu vé này sẽ được nhận CPU. Đây là giải thuật độc quyền.

Nhận xét: Giải thuật Lottery đơn giản chi phí thấp, bảo đảm tính công bằng cho các tiến trình.

3.3. LIÊN LẠC GIỮA CÁC TIẾN TRÌNH

Một tiến trình không bị ảnh hưởng bởi một tiến trình khác gọi là tiến trình độc lập, ngược lại gọi là tiến trình hợp tác (cooperating process). Lý do các tiến trình hợp tác, liên lạc với nhau là để chia sẻ thông tin như dùng chung file, bộ nhớ,… hoặc hợp tác hoàn thành công việc. Hệ điều hành cần cung cấp cơ chế để các tiến trình liên lạc với nhau, và thông thường có các cơ chế liên lạc sau:

3.3.1 Liên lạc bằng tín hiệu (Signal)

Một tín hiệu được sử dụng để thông báo cho tiến trình về một sự kiện nào đó xảy ra. Với mỗi tín hiệu sẽ có một hàm xử lý tín hiệu (signal handler) do phần cứng hoặc hệ điều hành cung cấp. Ví dụ một số tín hiệu của hệ điều hành UNIX:

Tín hiệu	Mô tả
SIGINT	Người dùng nhấn phím Ctl-C để ngắt xử lý tiến trình
SIGILL	Tiến trình xử lý một chỉ thị bất hợp lệ
SIGKILL	Yêu cầu kết thúc một tiến trình
SIGFPT	Lỗi chia cho 0
SIGSEGV	Tiến trình truy xuất đến một địa chỉ bất hợp lệ
SIGCLD	Tiến trình con kết thúc

Hình 3.21: một số tín hiệu của hệ điều hành UNIX

+ Tín hiệu được gởi đi bởi:

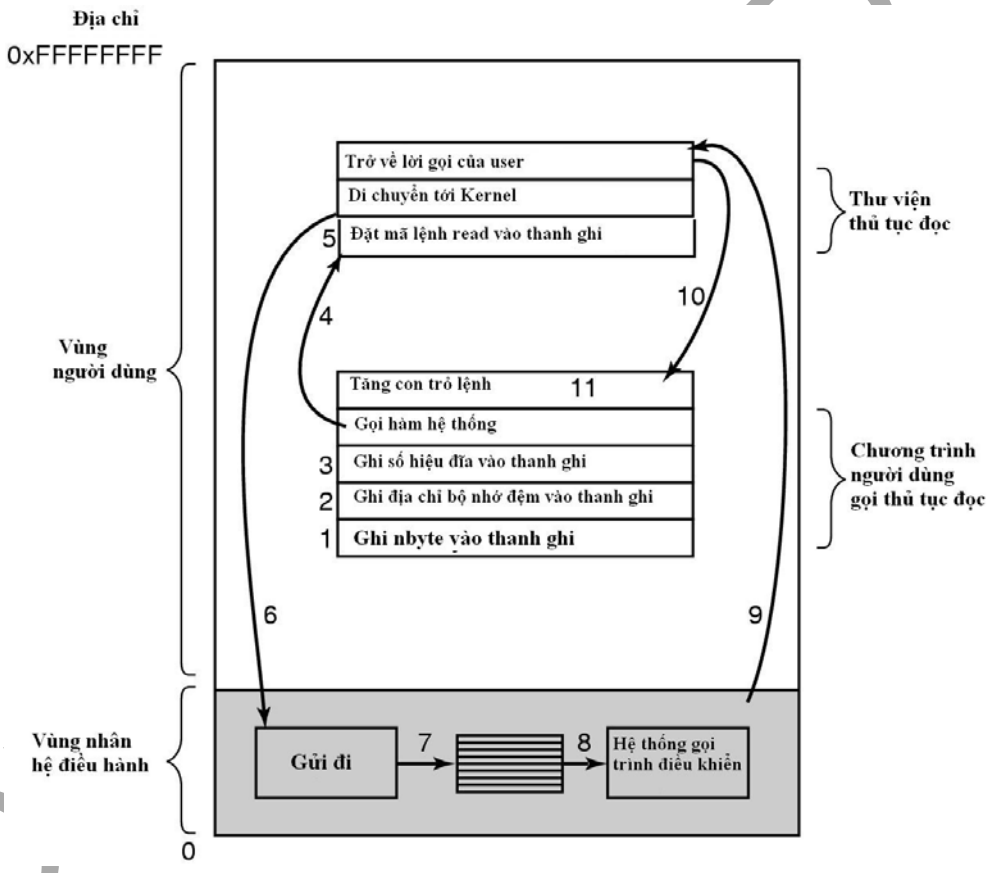
- Phần cứng: Ví dụ lỗi do các phép tính số học
- Hệ điều hành: Ví dụ một tiến trình nào đó truy xuất đến một địa chỉ bất hợp lệ.
- Tiến trình: Ví dụ tiến trình cha yêu cầu một tiến trình con kết thúc
- Người sử dụng: Ví dụ NSD nhấn phím Ctl-C để ngắt xử lý của tiến trình.

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

- Xử lý tín hiệu bằng cách gọi hàm xử lý tín hiệu.
- Xử lý theo cách riêng của tiến trình.
- Bỏ qua tín hiệu.

Ví dụ có 11 bước khi thực hiện lời gọi hệ thống: read(fd,buffer,nbyte)

ghi số byte cần đọc (nbytes) vào thanh ghi; ghi địa chỉ vùng nhớ chứa dữ liệu (buffer) vào thanh ghi; ghi số hiệu đĩa vào thanh ghi (fd=0 là đĩa mềm,…); gọi hàm hệ thống read;…



Hình 3.22: các bước thực hiện lời gọi hệ thống read.

Nhận xét

- Tiến trình nhận tín hiệu không thể xác định trước thời điểm nhận tín hiệu.
- Các tiến trình chỉ có thể thông báo cho nhau về một sự kiện, không thể trao đổi dữ liệu

Quản lý tiến trình

Hàm	Ý nghĩa
pid = fork()	Tạo một tiến trình con giống hết tiến trình cha
pid = waitpid(pid,&statloc, options)	Đợi một tiến trình con để ngừng thực thi
s = execve(name, argv, environp)	Thay thế 1 "ảnh nhân tiến trình" (process' core image)
exit(status)	Ngừng thực thi tiến trình và nhận về trạng thái

Quản trị tập tin

Hàm	Ý nghĩa
fd = open(file, how, …)	Mở 1 file để đọc hoặc ghi hoặc cả hai
s = close(fd)	Đóng 1 file đang mở
n = read(fd, buffer, nbytes)	Đọc dữ liệu từ 1 file vào 1 bộ đệm (buffer)
n = write(fd, buffer, nbytes)	Ghi dữ liệu từ 1 bộ đệm vào 1 file
position = lseek(fd, offset, whence)	Di chuyển con trỏ của file
s = stat(name, &buf)	Lấy thông tin trạng thái của 1 file

Quản trị hệ thống thư mục và tập tin

Hàm	Ý nghĩa
s = mkdir(name, mode)	Tạo thư mục mới
s = rmdir(name)	Xóa một thư mục rỗng
s = link(name1, name2)	Tạo một đề mục mới gọi là name2, chỉ tới name1.
s = unlink(name)	Xóa một đề mục
s = mount(special, name, tag)	Khởi tạo hệ thống tập tin
s = unmount(special)	Đóng hệ thống tập tin

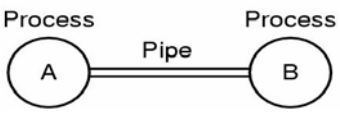
Các hàm khác

Hàm	Ý nghĩa
s = chdir(dirname)	Đổi thư mục làm việc
s = chmod(name, mode)	Đổi các bit bảo vệ của 1 file
s = kill(pid, signal)	Gửi 1 tín hiệu tới 1 tiến trình
seconds = time(&seconds)	Lấy thời gian đã trôi qua từ ngày 1/1/1097

Hình 3.23: một số lời gọi hệ thống thông dụng

3.3.2 Liên lạc bằng đường ống (Pipe)

Đường ống là một kênh liên lạc trực tiếp giữa hai tiến trình, dữ liệu xuất của tiến trình này được chuyển đến làm dữ liệu nhập cho tiến trình kia dưới dạng một dòng các byte. Thứ tự dữ liệu truyền qua pipe được bảo toàn theo nguyên tắc FIFO. Một tiến trình chỉ có thể sử dụng một pipe do nó tạo ra hay kế thừa từ tiến trình cha.



Hình 3.24: mô hình liên lạc bằng đường ống.

Hệ điều hành cần cung cấp các hàm (lời gọi hệ thống) read/write cho các tiến trình thực hiện thao tác đọc/ghi dữ liệu trong pipe.

+ Hệ điều hành đồng bộ hóa việc truy xuất pipe trong tình huống sau:

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

Nhận xét:

Một tiến trình kết nối với một pipe chỉ có thể thực hiện một trong hai thao tác đọc hoặc ghi.

Pipe cho phép truyền dữ liệu không cấu trúc.

Pipe chỉ để liên lạc giữa hai tiến trình có quan hệ cha-con, và trên cùng một máy tính.

3.3.3 Liên lạc qua vùng nhớ chia sẻ (shared memory)

Nhiều tiến trình cùng có thể truy xuất đến một vùng nhớ dùng chung, dữ liệu mà các tiến trình muốn gởi cho nhau, chỉ cần đặt vào vùng nhớ này. Vùng nhớ chia sẻ độc lập với các tiến trình, khi một tiến trình nào đó muốn truy xuất đến vùng nhớ dùng chung, tiến trình phải gắn kết vùng nhớ chung vào không gian địa chỉ riêng của tiến trình, và thao tác trên vùng nhớ chung giống như vùng nhớ của tiến trình.

Hình 3.25: Mô hình liên lạc bằng vùng nhớ chia xẻ

Nhận xét:

- Vùng nhớ chia sẻ là phương pháp nhanh nhất để trao đổi dữ liệu giữa các tiến trình.
- Vùng nhớ chia sẻ cần được bảo vệ bằng những cơ chế đồng bộ hóa.
- Vùng nhớ chia sẻ không thể áp dụng hiệu quả trong các hệ phân tán

3.3.4 Liên lạc bằng thông điệp (Message)

Hai tiến trình muốn liên lạc với nhau, cần thiết lập một mối liên kết giữa hai tiến trình, sau đó sử dụng các hàm send, receive do hệ điều hành cung cấp để trao đổi thông điệp. Khi sự liên lạc chấm dứt mối liên kết giữa hai tiến trình sẽ bị hủy.

*** Có hai cách liên lạc bằng thông điệp:**

a/ Liên lạc gián tiếp (indirect communication)

Hai tiến trình chỉ có thể liên lạc nếu có hộp thư/cổng (mailbox/port) dùng chung. Mỗi cổng (port) có một số hiệu duy nhất để phân biệt. Thông điệp sẽ được gởi và nhận thông qua cổng dùng chung.

- Send(A, message): gởi một thông điệp tới port A
- Receive(A, message): nhận một thông điệp từ port A

Tính chất: Một liên kết được thiết lập giữa hai tiến trình nếu và chỉ nếu chúng dùng chung port. Một liên kết có thể phục vụ nhiều tiến trình.

b/ Liên lạc trực tiếp (direct communication)

Send(P, message) : gởi một thông điệp đến tiến trình P

Receive(Q,message) : nhận một thông điệp từ tiến trình Q

Tính chất: Một liên kết duy nhất, hai chiều được thiết lập tự động giữa hai tiến trình P,Q và liên kết này chỉ dùng cho P và Q.

Ví dụ: Bài toán nhà sản xuất - người tiêu thụ (producer-consumer)

Hai tiến trình nsx, ntt thực thi đồng thời. Nsx sản xuất một sản phẩm, ntt tiêu thụ sản phẩm đó, nếu chưa có sản phẩm thì ntt chờ.

```
void nsx()
{       while(1)
        {       tạo_sp();
                send(ntt,sp); //gởi sp cho ntt
        }
}
void ntt()
{       while(1)
        {       receive(nsx,sp); //ntt chờ nhận sp
                tiêu_thụ(sp);
        }
}
```

Nhận xét:

Các tiến trình có thể trao đổi dữ liệu ở dạng có cấu trúc. Liên kết trực tiếp như trên còn gọi là liên kết trực tiếp đối xứng (symmetric), ta có thể có liên kết trực tiếp không đối xứng (asymmetric) như sau:

- Send(P, message) : gởi một thông điệp đến tiến trình P
- Receive(id,message): nhận một thông điệp từ tiến trình bất kỳ có mã số id.

3.3.5 Liên lạc qua socket

Socket là kênh liên lạc hai chiều. Hai tiến trình muốn liên lạc với nhau, mỗi tiến trình cần tạo một socket riêng, mỗi socket được kết buộc với một cổng khác nhau . Các thao tác đọc/ghi lên socket

chính là sự trao đổi dữ liệu giữa hai tiến trình. Cơ chế socket có thể sử dụng để chuẩn hoá mối liên lạc giữa các tiến trình vốn không liên hệ với nhau, và có thể hoạt động trong những hệ thống khác nhau và trong môi trường phân tán.

*** Có hai cách liên lạc qua socket:**

a/ Liên lạc kiểu thư tín (socket đóng vai trò bưu cục):

Hai tiến trình không cần kết nối, "tiến trình gởi" ghi dữ liệu vào socket của mình, dữ liệu sẽ được chuyển cho socket của "tiến trình nhận", "tiến trình nhận" sẽ nhận dữ liệu bằng cách đọc dữ liệu từ socket của "tiến trình nhận". Dữ liệu được gởi theo từng gói có chứa thông tin IP của máy nhận và port của tiến trình nhận (port dùng để phân biệt các tiến trình trên cùng một máy)

Nhận xét:

. "tiến trình gởi" không chắc chắn thông điệp được gởi đến "tiến trình nhận"

Hai thông điệp được gởi theo một thứ tự nào đó có thể đến "tiến trình nhận" theo một thứ tự khác.

. Một tiến trình sau khi đã tạo một socket có thể sử dụng nó để liên lạc với nhiều tiến trình khác nhau.

. Giao thức UDP sử dụng kiểu liên lạc này

b/ Liên lạc kiểu điện thoại (socket đóng vai trò tổng đài):

Hai tiến trình cần kết nối trước khi truyền/nhận dữ liệu và kết nối được duy trì suốt quá trình truyền nhận dữ liệu.

Nhận xét:

. Dữ liệu truyền nhận bảo đảm chính xác và đúng thứ tự gởi, nếu sai sẽ được gởi lại.

. Giao thức TCP sử dụng kiểu liên lạc này

3.4. ĐỒNG BỘ CÁC TIẾN TRÌNH

3.4.1. Yêu cầu đồng bộ

Đồng bộ các tiến trình là bảo đảm các tiến trình xử lý song song không tác động sai lệch đến nhau. Việc đồng bộ các tiến trình là do các yêu cầu sau:

a/ Yêu cầu độc quyền truy xuất (Mutual exclusion): tại một thời điểm, chỉ có một tiến trình được quyền truy xuất một tài nguyên không thể chia sẻ.

b/ Yêu cầu phối hợp (Synchronization): các tiến trình cần hợp tác với nhau để hoàn thành công việc. Ví dụ chương trình in xuất kí tự vào buffer, kí tự được lấy và in bởi chương trình điều khiển máy in (printer driver). Hai tiến trình này phải phối hợp với nhau như là chương trình in không được xuất kí tự vào buffer khi buffer đầy mà phải chờ printer driver lấy bớt dữ liệu.

Từ hai yêu cầu trên, ta có hai "bài toán đồng bộ" cần giải quyết đó là bài toán "độc quyền truy xuất" (hay còn gọi là "bài toán miền găng") và bài toán "phối hợp thực hiện".

3.4.2. Miền găng (critical section)

Đoạn mã của một tiến trình có khả năng xảy ra lỗi khi truy xuất tài nguyên dùng chung (biến, tập tin,…) được gọi là miền găng.

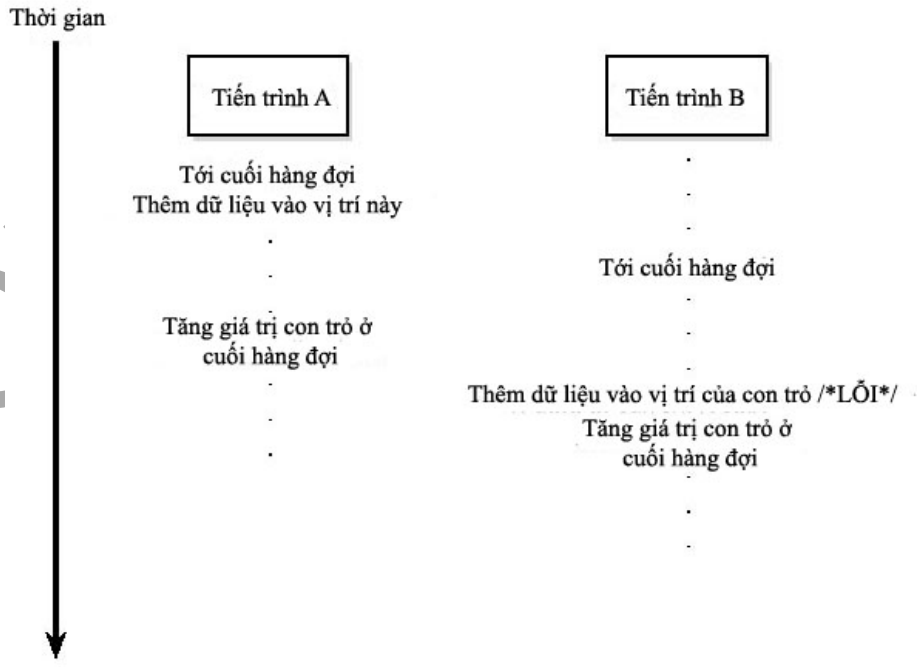
Ví dụ giả sử có hai tiến trình P1 và P2 sử dụng vùng nhớ dùng chung lưu trữ biến taikhoan. Mỗi tiến trình muốn rút một khoản tiền là tienrut từ tài khoản bằng đoạn mã sau:

if (taikhoan >= tienrut) taikhoan = taikhoan - tienrut;
else Thong bao “khong the rut tien ! “;

Giả sử tài khoản hiện còn 800, P1 muốn rút 500 và P2 muốn rút 400, có thể xảy ra tình huống lỗi như sau: giả sử P1 sau khi đã kiểm tra điều kiện (800 >=500) là đúng, P1 hết thời gian sử dụng CPU, hệ điều hành cấp phát CPU cho P2, P2 kiểm tra điều kiện (800>=400) cũng vẫn đúng, biến taikhoan được cập nhật lại là 400. Khi P1 được tiếp tục xử lý, nó sẽ không kiểm tra lại điều kiện (taikhoan>= tienrut) mà thực hiện rút tiền, giá trị của biến taikhoan sẽ cập nhật thành -100 !

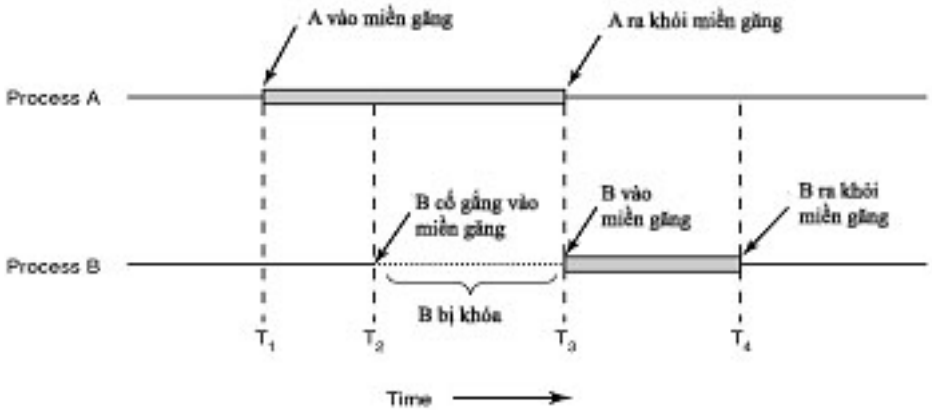
Đoạn mã:
if (taikhoan >= tienrut) taikhoan = taikhoan - tienrut;
gọi là một miền găng.

Ví dụ hai tiến trình chạy trên hai bộ xử lý, cùng ghi dữ liệu vào một hàng đợi dùng chung, sẽ xảy ra lỗi trong trường hợp sau: tiến trình A thêm dữ liệu vào hàng đợi tại vị trí tail (tail là vị trí cất dữ liệu) nhưng chưa kịp tăng tail thì hết quantum và đến lượt tiến trình B xử lý, tiến trình B thêm dữ liệu tại vị trí tail và như vậy sẽ ghi chồng lên dữ liệu do tiến trình A vừa ghi, lỗi xảy ra!



Hình 3.26: hai tiến trình A và B dùng chung một hàng đợi, có thể xảy ra lỗi.

Có thể giải quyết lỗi nếu bảo đảm tại một thời điểm chỉ có một tiến trình được xử lý lệnh trong miền găng, nghĩa là tại một thời điểm chỉ có một tiến trình truy xuất tài nguyên dùng chung.



Hình 3.27: Khi A trong miền găng thì nếu B muốn vào miền găng , B sẽ bị khoá chờ A ra khỏi miền găng, B mới được vào miền găng.

*** Khi giải quyết bài toán miền găng cần chú ý 4 điều kiện sau:**

a/ Không có hai tiến trình cùng ở trong miền găng cùng lúc.

b/ Không có giả thiết về tốc độ của các tiến trình, cũng như về số lượng bộ xử lý.

c/ Một tiến trình bên ngoài miền găng không được ngăn cản các tiến trình khác vào miền găng.

d/ Không có tiến trình nào phải chờ vô hạn để được vào miền găng.

3.4.3 Các giải pháp đồng bộ

Có 5 nhóm giải pháp để giải quyết "bài toán đồng bộ" là: Busy Waiting, Sleep And Wakeup, Semaphore, Monitor, Message.

3.4.3.1 Nhóm giải pháp Busy Waiting (bận thì đợi)

Nhóm giải pháp "Busy Waiting" lại chia thành hai loại: các giải pháp "sử dùng phần mềm" và các giải pháp "sử dùng phần cứng".

A/ Giải pháp phần mềm:

Việc đồng bộ là do chương trình thực hiện, giải pháp này là khó vì lập trình viên không lường trước được mọi tình huống có thể xảy ra. Thực vậy, chúng ta thử xét một số thuật toán đồng bộ sai sau đây:

a) Thuật toán 1: Sử dụng biến cờ hiệu, thuật toán mong đợi dùng cho nhiều tiến trình nhưng vẫn còn trường hợp sai. Ý tưởng như sau:

Các tiến trình dùng chung biến cờ hiệu lock , với ý nghĩa lock=0 là không có tiến trình trong miền găng, lock=1 là có một tiến trình trong miền găng. lock được gán trị ban đầu là 0. Một tiến trình muốn vào miền găng trước tiên kiểm tra giá trị của biến lock. Nếu lock = 0, tiến trình đặt lại giá trị cho lock = 1 và đi vào miền găng. Nếu lock đang có giá trị 1, tiến trình phải chờ bên ngoài miền găng cho đến khi lock có giá trị 0.

```
lock=0; //biến dùng chung cho mọi tiến trình
while (1) //một tiến trình có thể truy xuất miền găng nhiều lần
{       // nếu lock=1 là có tiến trình nào đó trong miền găng, dùng vòng lặp while để đợi
        while (lock == 1);
        //nếu lock=0 thoát khỏi while, trước khi vào mg gán lock=1 để không cho các tiến trình
        khác vào mg (độc quyền vào mg).
        lock = 1;
        critical-section (); //đoạn mã truy xuất dữ liệu dùng chung mà có thể gây ra lỗi.
        lock = 0; //cho phép các tiến trình khác vào mg.
        noncritical-section(); //đoạn mã không phải mg
}
```

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

b) Thuật toán 2: Sử dụng biến luân phiên, thuật toán dùng cho hai tiến trình nhưng vẫn còn trường hợp sai. Ý tưởng như sau:

Hai tiến trình A, B sử dụng chung biến turn với ý nghĩa sau: turn = 0, tiến trình A được vào miền găng, turn=1 thì B được vào miền găng. Turn được gán trị ban đầu là 0, tức là A được vào trước.

Nếu turn = 0, tiến trình A được vào miền găng. Nếu turn = 1, tiến trình A đi vào một vòng lặp chờ đến khi turn nhận giá trị 0 thì A được vào miền găng. Khi tiến trình A rời khỏi miền găng, nó đặt giá trị turn về 1 để cho phép tiến trình B đi vào miền găng.

```
// tiến trình A
while (1)
{
        while (turn == 1); // neu turn=1 thi A vao vong while de doi
        critical-section (); //turn=0 thi A duoc vao mien gang
        turn = 1; //gan turn=1 de cho B vao mg
        Noncritical-section ();
}
// tiến trình B
while (1)
{
        while (turn == 0); //neu turn=0 thi B vao vong while de doi
        critical-section (); //turn=1 thi B duoc vao mien gang
```

```
        turn = 0; //gan turn=0 de cho A vao mg
        Noncritical-section ();
}
```

Nhận xét: Hai tiến trình chắc chắn không thể vào miền găng cùng lúc, vì tại một thời điểm turn chỉ có một gía trị. Nhưng có thể vi phạm điều kiện thứ ba là một tiến trình có thể bị ngăn chặn vào miền găng bởi một tiến trình khác không ở trong miền găng. Thực vậy giả sử tiến trình A đang ở trong phần Noncritical-section(), thì B không thể vào miền găng hai lần liên tiếp. Như vậy, giải pháp này phụ thuộc vào tốc độ thực hiện của hai tiến trình, nó vi phạm cả điều kiện thứ hai.

c) Thuật toán Peterson: đây là giải pháp đúng và dùng cho hai tiến trình P0 và P1. Ý tưởng như sau:

Hai tiến trình dùng chung hai biến turn và flag[2] (kiểu int). Gán trị ban đầu flag [0]=flag [1]=FALSE và giá trị của turn được khởi động là 0 hay 1. Nếu flag [i] = TRUE (i=0,1) có nghĩa là tiến trình Pi muốn vào miền găng và turn=i là đến lượt Pi.

Để có thể vào được miền găng, trước tiên tiến trình Pi đặt giá trị flag [i]=TRUE để thông báo rằng tiến trình Pi muốn vào miền găng, sau đó đặt turn=j để thử đề nghị tiến trình Pj vào miền găng. Nếu tiến trình Pj không quan tâm đến việc vào miền găng nghĩa là flag [j] = FALSE, thì Pi có thể vào miền găng, nếu flag [j]=TRUE thì Pi phải chờ đến khi flag [j]=FALSE. Khi tiến trình Pi rời khỏi miền găng, nó đặt lại giá trị cho flag [i] là FALSE.

```
// tiến trình P0 (i=0)
while (TRUE)
{
        flag [0]= TRUE;//P0 thông báo là P0 muon vao mg
        turn = 1; //thu de nghi P1 vao
        while (turn == 1 && flag [1]==TRUE); //neu P1 muon vao thi P0 chờ
        critical_section();
        flag [0] = FALSE; //P0 ra ngoai mg
        noncritical_section ();
}
// tiến trình P1 (i=1)
while (TRUE)
{
        flag [1]= TRUE; //P1 thông báo là P1 muon vao mg
        turn = 0;//thử de nghi P0 vao
        while (turn == 0 && flag [0]==TRUE); //neu P0 muon vao thi P1 chờ
        critical_section();
        flag [1] = FALSE;//P1 ra ngoai mg
        Noncritical_section ();
}
```

Nhận xét: Nếu cả hai tiến trình đều muốn vào miền găng thì flag [0] = flag [1] =TRUE nhưng giá trị của turn tại một thời điểm chỉ có thể hoặc là 0 hoặc là 1, do vậy chỉ có một tiến trình được vào miền găng và dễ dàng kiểm tra là giải pháp cũng thỏa các điều kiện còn lại.

B/ Các giải pháp phần cứng:

a) Cấm ngắt:

Tiến trình cấm tất cả các ngắt trước khi vào miền găng, và phục hồi ngắt khi ra khỏi miền găng.

Khi đó ngắt đồng hồ cũng không thể xảy ra, do vậy hệ thống không thể tạm dừng hoạt động của tiến trình đang xử lý để cấp phát CPU cho tiến trình khác, nhờ đó tiến trình hiện hành yên tâm thao tác trên miền găng mà không sợ bị tiến trình nào khác tranh chấp, tức là hệ điều hành độc quyền truy xuất miền găng.

Nhận xét:

+ Dễ cài đặt nhưng cấm tất cả các ngắt là nguy hiểm .

+ Nếu hệ thống có nhiều bộ xử lý, lệnh cấm ngắt chỉ có tác dụng trên bộ xử lý đang xử lý tiến trình, còn các tiến trình hoạt động trên các bộ xử lý khác vẫn có thể truy xuất đến miền găng.

b) Sử dụng lệnh TSL (Test and Set Lock):

Đa số phần cứng cung cấp lệnh TSL cho phép kiểm tra và cập nhật một vùng nhớ trong một thao tác độc quyền. Nếu có hai lệnh TSL xử lý đồng thời trên hai CPU khác nhau thì chúng sẽ được xử lý tuần tự. Lệnh TSL có cấu trúc như sau:

```
boolean Test_And_Set_Lock (boolean lock)
{
        boolean temp=lock;
        lock = TRUE;
        return temp; //trả về giá trị ban đầu của biến lock
}
```

Có thể cài đặt giải pháp truy xuất độc quyền với TSL bằng cách sử dụng thêm một biến lock dùng chung được khởi gán là FALSE. Tiến trình phải kiểm tra giá trị của biến lock trước khi vào miền găng, nếu lock = FALSE, tiến trình có thể vào miền găng.

```
boolean lock=FALSE; //biến dùng chung
while (TRUE)
{
        while (Test_And_Set_Lock(lock));
        critical_section ();
        lock = FALSE;
        noncritical_section ();
}
```

Nhận xét: Nhóm giải pháp "busy and waiting" đều phải thực hiện một vòng lặp để kiểm tra xem có được vào miền găng hay không nên tiến trình đang chờ vẫn chiếm dụng CPU. Do đó cần tránh các giải pháp " busy waiting " nếu có thể.

3.4.3.2. Nhóm giải pháp "SLEEP and WAKEUP " (ngủ và đánh thức)

a) Sử dụng lệnh SLEEP VÀ WAKEUP

Hệ điều hành cung cấp hai lệnh SLEEP VÀ WAKEUP. Nếu tiến trình gọi lệnh SLEEP, hệ điều hành sẽ chuyển tiến trình sang "danh sách sẵn sàng", lấy lại CPU cấp cho tiến trình khác. Nếu tiến trình gọi lệnh WAKEUP, hệ điều hành sẽ chọn một tiến trình trong ready list, cho thực hiện tiếp. Khi một tiến trình chưa đủ điều kiện vào miền găng, nó gọi SLEEP để tự khóa, đến khi có một tiến trình khác gọi WAKEUP để giải phóng cho nó. Một tiến trình gọi WAKEUP khi ra khỏi miền găng để đánh thức một tiến trình đang chờ, tạo cơ hội cho tiến trình này vào miền găng.

Việc sử dụng hai lệnh SLEEP VÀ WAKEUP thực không đơn giản, rất dễ bị lỗi. Thực vậy, ta xét một chương trình giải quyết bài toán miền găng như sau:

```
//busy và blocked là hai biến dùng chung.
int busy=FALSE;   // TRUE là có tiến trình trong miền găng, FALSE là không có tiến trình trong miền găng.
int blocked=0;   // đếm số lượng tiến trình đang bị khóa
while (TRUE) //để cho một tiến trình có thể vào mg nhiều lần
{
        if (busy)
        {
                blocked = blocked + 1;
                sleep();
        }
        else      busy = TRUE;
        critical-section ();
        busy = FALSE;
        if  (blocked>0)
        {
                wakeup(); //đánh thức một tiến trình đang chờ
                blocked = blocked - 1;
        }
        Noncritical-section ();
}
```

Nhận xét:

- Có thể vi phạm điều kiện thứ 1 là có hai tiến trình trong miền găng cùng lúc. Thực vậy, giả sử tiến trình A kiểm tra biến busy, thấy busy=FALSE, nhưng chưa kịp gán busy=TRUE thì đến lượt tiến trình B. B thấy busy=FALSE, B gán busy=TRUE và vào miền găng. Trong khi B chưa ra khỏi miền găng thì đến lượt A, A gán busy=TRUE và vào miền găng!

- Có thể vi phạm điều kiện thứ ba là một tiến trình có thể bị ngăn chặn vào miền găng bởi một tiến trình khác không ở trong miền găng. Ví dụ giả sử tiến trình A vào miền găng, và trước khi nó rời khỏi miền găng thì tiến trình B được kích hoạt. Tiến trình B thử vào miền găng nhưng nó nhận

thấy A đang ở trong đó, do vậy B tăng giá trị biến blocked và chuẩn bị gọi SLEEP để tự khoá. Tuy nhiên trước khi B có thể thực hiện SLEEP, tiến trình A lại được tái kích hoạt và ra khỏi miền găng. Khi ra khỏi miền găng A nhận thấy có một tiến trình đang chờ (blocked=1) nên gọi WAKEUP và giảm giá trị của blocked. Khi đó tín hiệu WAKEUP sẽ lạc mất do tiến trình B chưa thật sự " ngủ " để nhận tín hiệu đánh thức ! Khi tiến trình B được tiếp tục xử lý, nó mới goi SLEEP và tự ngủ vĩnh viễn !

Lỗi này xảy ra do việc kiểm tra biến busy và việc gọi SLEEP là những hành động tách biệt, có thể bị ngắt giữa chừng trong quá trình xử lý. Để tránh những tình huống tương tự, hệ điều hành cung cấp những cơ chế đồng bộ hóa như Semaphore, Monitor dựa trên ý tưởng của chiến lược "SLEEP and WAKEUP" nhưng việc kiểm tra điều kiện vào miền găng và việc chờ xây dựng thành một hành động độc quyền, giúp việc giải quyết bài toán miền găng an toàn, hiệu qủa hơn.

b) Sử dụng cấu trúc Semaphore:

Semaphore là cấu trúc được Dijkstra đề xuất vào 1965, semaphore s là một biến có các thuộc tính sau:

- Một giá trị nguyên dương e
- Một hàng đợi f lưu danh sách các tiến trình đang chờ trên semaphore s
- Có hai thao tác được định nghĩa trên semaphore s:

Down(s): e=e-1. Nếu e < 0 thì tiến trình phải chờ trong f, ngược lại tiến trình tiếp tục.

Up(s): e=e+1. Nếu e<=0 thì chọn một tiến trình trong f cho tiếp tục thực hiện (đánh thức).

Gọi P là tiến trình thực hiện thao tác Down(s) hay Up(s):

```
Down(s)
{       e = e - 1;
        if (e < 0)
        {       status(P)= blocked; //chuyển P sang trạng thái bị khoá (chờ)
                enter(P,f);             //cho P vào hàng đợi f
        }
}
Up(s)
{       e = e + 1;
        if  (e<= 0 )
        {
                exit(Q,f);  //lấy một tt Q ra khỏi hàng đợi f theo một thuật toán nào đó (FIFO,…)
                status (Q) = ready; //chuyển Q sang trạng thái sẵn sàng
                enter(Q,ready-list); //đưa Q vào danh sách sẵn sàng của hệ thống
        }
}
```

Nhận xét:

- Hệ điều hành cần cài đặt các thao tác Down, Up là độc quyền. Để cài đặt sự độc quyền có thể dùng kỹ thuật cấm ngắt (1 cpu) hoặc các giải pháp phần mềm, hoặc lệnh TSL (nhiều cpu). Nếu dùng giải pháp phần mềm thì giải pháp semaphore vẫn là giải pháp "busy and waiting" nhưng tách vòng lặp chờ ra khỏi chương trình.
- Hàng đợi có thể cài đặt là một con trỏ trỏ tới danh sách các khối PCB của các tiến trình đang chờ trên s, khi đó semaphore có dạng:

```
class semaphore
{
        int e;
        PCB * f; //ds  riêng của semaphore
        public:
        down();
        up();
};
```

|e| = số tiến trình đang chờ trên f.

Có thể dùng semaphore để giải quyết bài toán miền găng hay đồng bộ các tiến trình.

*** Giải quyết bài toán miền găng bằng Semaphores:**

Dùng một semaphore s, e được khởi gán là 1. Tất cả các tiến trình áp dụng cùng cấu trúc chương trình sau:

```
semaphore s=1; //nghĩa là e của s=1
while (1)
{
        Down(s);
        critical-section ();
        Up(s);
        Noncritical-section ();
}
```

*** Giải quyết bài toán đồng bộ bằng Semaphores:**

Ví dụ có hai tiến trình đồng hành P1 và P2, P1 thực hiện công việc 1, P2 thực hiện công việc 2. Giả sử muốn cv1 làm trước rồi mới làm cv2, ta có thể cho hai tiến trình dùng chung một semaphore s, khởi gán e(s)= 0:

```
semaphore s=0; //dùng chung cho hai tiến trình
P1:
{
        job1();
        Up(s); //đánh thức P2
}
```

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

Nhận xét: Nhờ lệnh down, up là độc quyền, semaphore đã giải quyết được vấn đề tín hiệu "đánh thức" bị thất lạc. Tuy nhiên sử dụng semaphore cũng không đơn giản, chương trình dễ bị lỗi mà không dự đoán được. Ta xét một số tình huống gây ra lỗi sau:

- Nếu đặt Down và Up sai vị trí hoặc thiếu thì có thể bị sai. Ví dụ nếu P1 để Up() lên trước lệnh job1() thì nếu P1 thực hiện trước P2, có thể job1 chưa thực hiện xong mà job2 được thực hiện.

Xét ví dụ khác

```
e(s)=1;
while (1)
{
        Down(s); critical-section (); Noncritical-section ();
}
```

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

- Sử dụng semaphore có thể gây ra tình trạng tắc nghẽn. Ví dụ có hai tiến trình P1, P2 sử dụng chung hai semaphore s1=s2=1

```
P1:
{
        down(s1); down(s2);
        ….
        up(s1); up(s2);
}
P2:
{
        down(s2); down(s1);
        ….
        up(s2); up(s1);
}
```

Nếu thứ tự thực hiện như sau: P1: down(s1), P2: down(s2) , P1: down(s2), P2: down(s1) khi đó s1=s2=-1 nên P1,P2 đều chờ mãi

- Sử dụng semaphore có thể gây ra tình trạng đói CPU khi giải thuật chọn tiến trình đang đợi là giải thuật LIFO.

Semaphore xây dựng như trên gọi là semaphore đếm (counting semaphore) giá trị e không giới hạn. Semaphore nhị phân (binary semaphore) có e=0,1 dễ cài đặt hơn vì được sự hỗ trợ của phần cứng. Semaphore đếm có thể cài đặt bằng semaphore nhị phân như sau:

```
//các biến dùng chung
binary semaphore s1=1, s2=0;
int c=giá trị ban đầu e của semaphore đếm;
down() //down of counting semaphore
{
        down(s1); //down of binary semaphore
        c--;
        if (c<0) down(s2); //down of binary semaphore
        up(s1); //up of binary semaphore
}
up() //up of counting semaphore
{
        down(s1); //down of binary semaphore
        c++;
        if (c<=0) up(s2); //up of binary semaphore
        up(s1);
}
```

down(s1); up(s1); để đảm bảo độc quyền truy xuất miền găng .

c) Sử dụng cấu trúc Monitors

Để có thể dễ viết đúng các chương trình đồng bộ hóa hơn, Hoare(1974) và Brinch & Hansen (1975) đã đề nghị một cơ chế cao hơn được cung cấp bởi ngôn ngữ lập trình là monitor.

Monitor là một cấu trúc đặc biệt (lớp) bao gồm các phương thức độc quyền (chính là các critical-section) và các biến có tính chất sau :

+ Các biến trong monitor chỉ có thể được truy xuất bởi các phương thức trong monitor, đây chính là các biến được dùng chung cho các tiến trình.

+ Tại một thời điểm, chỉ có một tiến trình duy nhất được hoạt động bên trong một monitor.

+ Trong monitor có thể khai báo các biến điều kiện (thuộc lớp condition) dùng để đồng bộ việc sử dụng các biến trong monitor. Việc sử dụng bao nhiêu biến điều kiện là do người lập trình quyết định. Biến điều kiện có hai lệnh Wait và Signal:

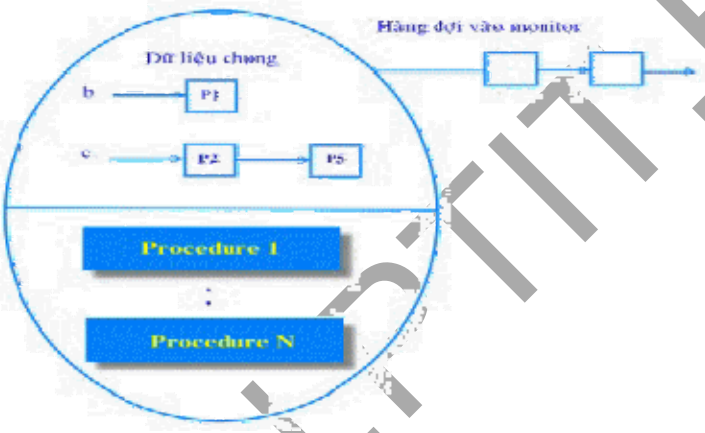
- Wait(c): chuyển trạng thái tiến trình gọi sang chờ (blocked) và đặt tiến trình này vào hàng đợi trên biến điều kiện c.

- Signal(c): nếu có một tiến trình đang chờ trong hàng đợi của c, tái kích hoạt tiến trình đó và tiến trình gọi sẽ rời khỏi monitor. Nếu không có tiến trình nào đang chờ trong hàng đợi của c thì lệnh signal(c) bị bỏ qua.

```
Wait(c)
{       status(P)= blocked;     //chuyển P sang trạng thái chờ
        enter(P,f(c));          //đặt P vào hàng đợi f(c) của biến điều kiện c
}
Signal(c)
{
        if (f(c) != NULL)
        {
                exit(Q,f(c));           //Lấy tiến trình  Q đang chờ trên c
                statusQ) = ready;       //chuyển Q sang trạng thái sẵn sàng
                enter(Q,ready-list);    //đưa Q vào danh sách sẵn sàng.
        }
}
```

+ Mỗi biến kiểu monitor có một hàng đợi toàn cục lưu các tiến trình đang chờ được sử dụng monitor.

+ Biến kiểu monitor dùng chung cho các tiến trình dùng chung tài nguyên.



Hình 3.28: mô hình cấu trúc monitor

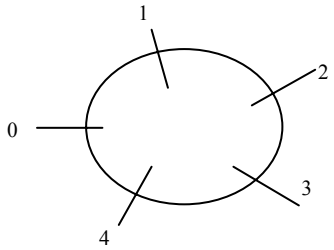
```
monitor <tên monitor > //khai báo monitor dùng chung cho các tiến trình
{
        <cac bien dung chung>;
        <các biến điều kiện>;
        <cac phuong thuc độc quyền>;
}
//tiến trình Pi:
while (1) //cấu trúc tiến trình thứ i
{
        Noncritical-section ();
        <ten monitor>.Phươngthức_i;  //thực hiện công việc độc quyền thứ i
        Noncritical-section ();
}
```

Nhận xét:

- Việc truy xuất độc quyền được bảo đảm bởi trình biên dịch mà không do lập trình viên, do vậy nguy cơ thực hiện đồng bộ hóa sai giảm rất nhiều.
- Giải pháp monitor đòi hỏi ngôn ngữ lập trình đang sử dụng có kiểu dữ liệu là monitor, hiện các ngôn ngữ này chưa có nhiều.
- Giải pháp "busy and waiting" không phải thực hiện việc chuyển đổi ngữ cảnh trong khi giải pháp "sleep and wakeup" sẽ tốn thời gian cho việc này.

Ví dụ: Bài toán 5 triết gia ăn tối.

Có 5 triết gia ăn tối món mì ống, ngồi xung quanh một bàn tròn, trước mặt mỗi người có một chiếc đũa. Biết rằng muốn ăn được phải cần hai chiếc đũa, nếu mỗi người đều lấy chiếc đũa trước mặt mình cùng lúc thì sẽ xảy ra trường hợp là không triết gia nào ăn được (deadlock). Hãy đồng bộ việc ăn tối của 5 triết gia sao cho tất cả đều ăn được.



Hình 3.29: bài toán năm triết gia ăn tối

```
monitor philosopher
{       enum {thinking, hungry, eating} state[5];// cac bien dung chung cho các triết gia
        condition self[5];      //cac bien dieu kien dung de dong bo viec an toi
        //cac pt doc quyen (cac mien gang), viec doc quyen do nnlt ho tro.
        void init();//phương thức khoi tao
        void test(int i); //phương thức kiểm tra điều kiện trước khi cho triết gia thứ i ăn
        void pickup(int i);       //phương thức lấy đũa
        void putdown(int i); //phương thức trả đũa
}
void philosopher()//phương thức khởi tạo (constructor)
{       //gán trạng thái ban đầu cho các triết gia là "đang suy nghĩ"
        for (int i = 0; i < 5; i++) state[i] = thinking;
}
void test(int i)
{       //nếu tg_i đói và các tg bên trái, bên phải không đang ăn thì cho tg_i ăn
        if ( (state[i] == hungry) && (state[(i + 4) % 5] != eating) &&(state[(i + 1) % 5] != eating))
```

```
        {
                self[i].signal();//đánh thức tg_i, nếu tg_i đang chờ
                state[i] = eating; //ghi nhận tg_i đang ăn
        }
}
void pickup(int i)
{
        state[i] = hungry; //ghi nhận tg_i đói
        test(i); //kiểm tra đk trước khi cho tg_i ăn
        if (state[i] != eating)  self[i].wait(); //doi tai nguyen
}
void putdown(int i)
{
        state[i] = thinking; //ghi nhận tg_i đang suy nghĩ
        test((i+4) % 5); //kt tg bên phải, nếu hợp lệ thì cho tg này ăn
        test((i+1) % 5);//kt tg bên  trái nếu hợp lệ thì cho tg này ăn
}

// Cấu trúc tiến trình Pi  thực hiện viec ăn của triết gia thứ  i
philosopher pp; //bien monitor dung chung
Pi:
while (1)
{
        Noncritical-section ();
        pp.pickup(i); //pickup là miền găng và được truy xuất độc quyền
        eat(); //triet gia an
        pp.putdown(i);//putdown là miền găng và được truy xuất độc quyền
        Noncritical-section ();
}
```

d) Sử dụng thông điệp:

Có một tiến trình kiểm soát việc sử dụng tài nguyên và nhiều tiến trình khác yêu cầu tài nguyên. Tiến trình cần tài nguyên sẽ gởi một thông điệp yêu cầu tài nguyên đến tiến trình kiểm soát, sau đó tự chuyển sang trạng thái blocked (chờ trong hàng đợi tài nguyên). Tiến trình kiểm soát , khi nhận được thông điệp yêu cầu tài nguyên, đợi đến khi tài nguyên sẵn sàng thì gởi một thông điệp đến một tiến trình đang đợi tài nguyên để đánh thức và cho sử dụng tài nguyên. Khi sử dụng xong tài nguyên , tiến trình sử dụng tài nguyên gởi một thông điệp khác đến tiến trình kiểm soát để báo kết thúc truy xuất tài nguyên.

//Cấu trúc tiến trình yêu cầu tài nguyên trong giải pháp message

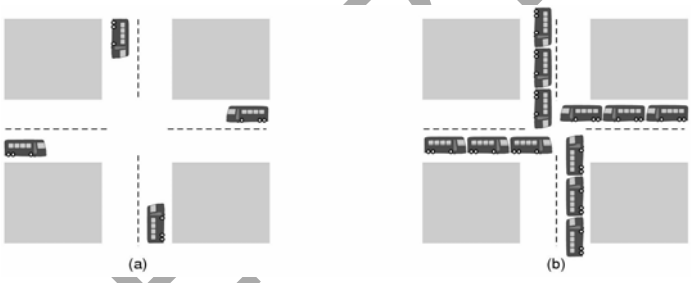
```
while (1)
{
        Send(process controler, request message); //goi td yc tn va chuyen sang trang thai blocked
        Receive(process controler, accept message); //nhan td chap nhan su dung tn
        critical-section (); //doc quyen su dung tai nguyen dung chung
        Send(process controler, end message); //goi td ket thuc su dung tn.
        Noncritical-section ();
}
```

Nhận xét: Semaphore và monitor có thể giải quyết được vấn đề truy xuất độc quyền trên các máy tính có một hoặc nhiều bộ xử lý chia sẻ một vùng nhớ chung. Nhưng không thuận lợi trong các hệ thống phân tán, khi mà mỗi bộ xử lý sở hữu một bộ nhớ riêng biệt và liên lạc thông qua mạng. Trong những hệ thống phân tán, cơ chế trao đổi thông điệp sẽ đơn giản hơn và được dùng để giải quyết bài toán đồng bộ hóa.

3.5 TÌNH TRẠNG TẮC NGHẼN (DEADLOCKS)

3.5.1 Khái niệm tắc nghẽn:

Một tập hợp các tiến trình gọi là ở tình trạng tắc nghẽn nếu mỗi tiến trình trong tập hợp đều chờ đợi tài nguyên mà tiến trình khác trong tập hợp đang chiếm giữ. Ví dụ tại một thời diểm, tiến trình 1 đang giữ tài nguyên R1, yêu cầu R2 và tiến trình 2 đang giữ tài nguyên R2, yêu cầu R1, như vây yêu cầu về tài nguyên không thể đáp ứng cho cả hai tiến trình. Khi đó không có tiến trình nào có thể tiếp tục xử lý, cũng như giải phóng tài nguyên cho tiến trình khác sử dụng, tất cả các tiến trình trong tập hợp đều bị khóa vĩnh viễn!



Hình 3.30: (a) một tình trạng tắc nghẽn tiềm ẩn; (b) một tình trạng tắc nghẽn thực sự.

ví dụ: Bữa ăn tối của các triết gia. Có 5 nhà triết gia cùng ngồi ăn tối. Mỗi nhà triết gia cần dùng 2 cái đũa để có thể ăn. Nhưng trên bàn chỉ có tổng cộng 5 cái đũa, nếu cả 5 người đều cầm cái đũa bên trái cùng lúc, thì sẽ không có ai có được cái đũa bên phải để có thể bắt đầu ăn . Tình trạng này gọi là tình trạng tắc nghẽn.

Tài nguyên có thể là tài nguyên vật lý (máy in, bộ nhớ, cpu, đĩa, …) hoặc tài nguyên logic (file, semaphore, monitor,…). Tài nguyên lại phân thành hai loại: loại tài nguyên có thể lấy lại từ một tiến trình đang chiếm giữ mà không ảnh hưởng đến tiến trình đang chiếm giữ và loại tài nguyên không thể thu hồi lại từ tiến trình đang chiếm giữ.

3.5.2 Điều kiện xuất hiện tắc nghẽn

Hệ thống sẽ xuất hiện tắc nghẽn khi và chỉ khi có đủ 4 điều kiện sau:

+ **Điều kiện 1**: Có sử dụng tài nguyên không thể chia sẻ

Mỗi thời điểm, một tài nguyên không thể chia sẻ được hệ thống cấp phát chỉ cho một tiến trình , khi tiến trình sử dụng xong tài nguyên này, hệ thống mới thu hồi và cấp phát tài nguyên cho tiến trình khác.

+ **Điều kiện 2**: Sự chiếm giữ và yêu cầu thêm tài nguyên không thể chia sẻ

Có tiến trình chiếm giữ các tài nguyên trong khi lại chờ được cấp phát thêm tài nguyên bị chiếm giữ bởi tiến trình khác.

+ **Điều kiện 3**: Không thu hồi tài nguyên từ tiến trình đang giữ chúng

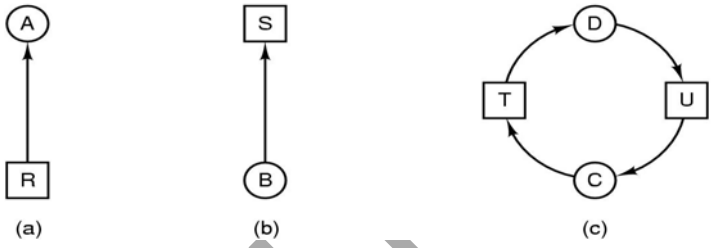
Tài nguyên không thể được thu hồi từ tiến trình đang chiếm giữ chúng trước khi tiến trình này sử dụng chúng xong.

+ **Điều kiện 4**: Tồn tại một chu trình trong đồ thị cấp phát tài nguyên

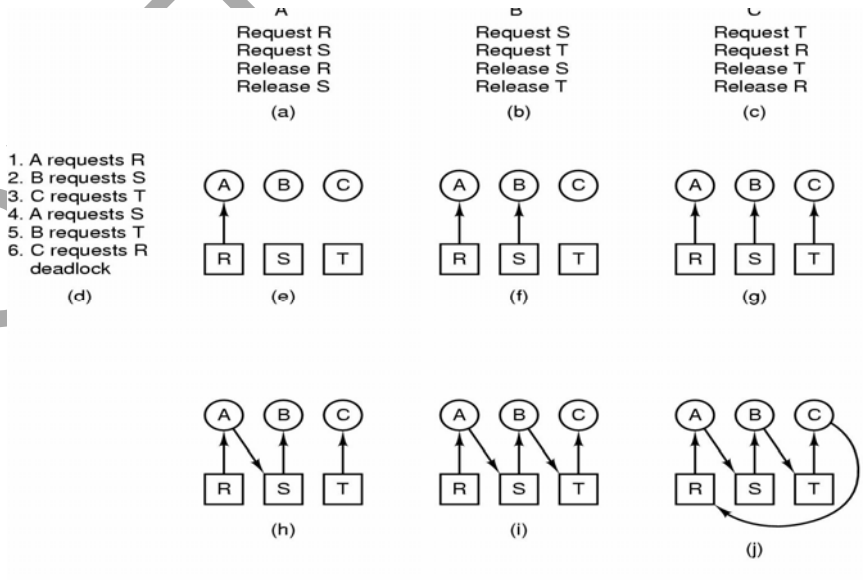
Có ít nhất hai tiến trình chờ đợi lẫn nhau: tiến trình này chờ được cấp phát tài nguyên đang bị tiến trình khác chiếm giữ và ngược lại.

3.5.3 Đồ thị cấp phát tài nguyên:

hình tròn là tiến trình, hình vuông là tài nguyên. Đối với tiến trình, mũi tên đi ra là chiếm giữ tài nguyên, mũi tên vào là yêu cầu tài nguyên. Ví dụ tiến trình A đang giữ tài nguyên R, tiến trình B yêu cầu tài nguyên S. Tiến trình C giữ U, yêu cầu T, tiến trình D giữ T, yêu cầu U. Tập hợp tiến trình {C,D} gọi là ở tình trạng tắc nghẽn.



Hình 3.31: mô hình đồ thị cấp phát tài nguyên



Hình 3.32: một ví dụ về tắc nghẽn

3.5.3 Các phương pháp xử lý tắc nghẽn và ngăn chặn tắc nghẽn

**** Các phương pháp xử lý tắc nghẽn***

+ Sử dụng một thuật toán cấp phát tài nguyên nào đó mà bảo đảm không bao giờ xảy ra tắc nghẽn.

+ Hoặc cho phép xảy ra tắc nghẽn và tìm cách sữa chữa tắc nghẽn.

+ Hoặc bỏ qua việc xử lý tắc nghẽn, xem như hệ thống không bao giờ xảy ra tắc nghẽn. Thường áp dụng phương pháp này khi hệ thống rất ít khi bị tắc nghẽn và chi phí kiểm tra tắc nghẽn cao (UNIX và WINDOWS sử dụng phương pháp này)

**** Ngăn chặn tắc nghẽn***

Để không xảy ra tắc nghẽn, cần bảo đảm tối thiểu một trong 4 điều kiện đã nêu ở trên không xảy ra:

+ Điều kiện 1 gần như không thể tránh được điều kiện này vì bản chất tài nguyên gần như cố định.

+ Để điều kiện 2 không xảy ra, thì có thể áp dụng một trong hai nguyên tắc sau :

- Tiến trình phải yêu cầu tất cả các tài nguyên cần thiết trước khi cho bắt đầu xử lý. Phương pháp này gặp khó khăn là hệ điều hành khó có thể biết trước các tài nguyên tiến trình cần sử dụng vì nhu cầu tài nguyên còn phụ thuộc vào quá trình tiến trình thực hiện. Ngoài ra nếu cho tiến trình chiếm giữ sẵn các tài nguyên chưa cần sử dụng ngay thì việc sử dụng tài nguyên sẽ kém hiệu quả.

- Khi tiến trình yêu cầu một tài nguyên mới và bị từ chối, nó phải giải phóng các tài nguyên đang chiếm giữ , sau đó lại được cấp phát trở lại cùng lần với tài nguyên mới. Phương pháp này sẽ gặp khó khăn trong việc bảo vệ tính toàn vẹn dữ liệu của hệ thống.

+ Để điều kiện 3 không xảy ra, hệ điều hành cần cho phép hệ thống được thu hồi tài nguyên từ các tiến trình bị khoá và cấp phát trở lại cho tiến trình khi nó thoát khỏi tình trạng bị khóa. Tuy nhiên với một số loại tài nguyên, việc thu hồi sẽ rất khó khăn vì vi phạm sự toàn vẹn dữ liệu.

+ Để điều kiện 4 không xảy ra, có thể cấp phát tài nguyên theo một sự phân cấp như sau :

Gọi R = {R1, R2,...,Rm} là tập các loại tài nguyên. Các loại tài nguyên được đánh số thứ tự . Ví dụ : F(đĩa) = 2, F(máy in) = 12,…

Khi tiến trình đang chiếm giữ tài nguyên Ri thì chỉ có thể yêu cầu các tài nguyên Rj nếu F(Rj) > F(Ri).

Ta có thể tránh tắc nghẽn khi cấp phát tài nguyên bằng cách sử dụng giải thuật cấp phát tài nguyên như sau:

3.5.4 Giải thuật cấp phát tài nguyên (giải thuật banker)

3.5.4.1 Giải thuật xác định trạng thái an toàn

Nếu hệ thống có thể thỏa mãn các nhu cầu tài nguyên tối đa của mỗi tiến trình theo một thứ tự cấp phát nào đó mà không bị tắc nghẽn thì gọi là hệ thống ở trạng thái là an toàn. Hệ thống ở trạng thái không an toàn có thể dẫn đến tình trạng tắc nghẽn. Ta có giải thuật xác định trạng thái an toàn như sau:

```
int NumResources;//số tài nguyên, một tài nguyên có thể có nhiều thể hiện (instance)
int NumProcs;//số tiến trình trong hệ thống
int Available[NumResources]; // Available[r]= số lượng các thể hiện còn tự do của tài nguyên r
int Max[NumProcs, NumResources]; //Max[p,r]= nhu cầu tối đa của tiến trình p về tài nguyên r
```

int Allocation[NumProcs, NumResources];// Allocation[p,r] = số tài nguyên r đã cấp phát cho tiến trình p

int Need[NumProcs, NumResources]; // Need[p,r] = Max[p,r] - Allocation[p,r]= số tài nguyên r mà tiến trình p còn cần sử dụng

int Finish[NumProcs] = false; //Finish[p]=true là tiến trình p đã thực thi xong;

B1.Tìm tiến trình i thoả các điều kiện sau:

- Tiến trình i chưa thực thi xong:

  Finish[i] = false

- Mọi nhu cầu về tài nguyên của tiến trình i đều có thể đáp ứng:

  Need[i,j] <= Available[j], với mọi tài nguyên j

Nếu không có tiến trình i như thế thì đến bước 3, nếu có xuống bước 2

B2. Cấp phát mọi tài nguyên mà tiến trình i cần

- Cấp phát đủ tài nguyên cho tiến trình i.

  Allocation[i,j]= Allocation[i,j]+Need[i,j]; $\forall$ j

  need[i,j]=0 ; $\forall$ j

  Available[j]= Available[j] - Need[i,j];

- Đánh dấu tiến trình i thực hiện xong

  Finish[i] = true;

- Thu hồi lại tất cả tài nguyên đã cấp cho tiến trình i, cập nhật lại số tài nguyên j khả dụng

  Available[j]= Available[j] + Allocation[i,j];

- Quay lại bước 1

B3. Nếu Finish[i] = true với mọi i, thì hệ thống ở trạng thái an toàn, ngược lại là không an toàn.

3.5.4.2 Giải thuật Banker

Khi có tiến trình yêu cầu các tài nguyên, hệ điều hành thử cấp phát, sau đó xác định hệ thống có an toàn không (dùng giải thuật xác định trạng thái an toàn). Nếu hệ thống an toàn thì cấp phát thực sự các tài nguyên mà tiến trình yêu cầu, ngược lại tiến trình phải đợi.

Giả sử tiến trình Pi yêu cầu kr thể hiện của tài nguyên r. Giải thuật cấp phát được thực hiện như sau:

B1. Nếu kr <= Need[i,r] với mọi r, đến bước 2, ngược lại, xảy ra tình huống lỗi

B2. Nếu kr <= Available[r] với mọi r, đến bước 3 , ngược lại Pi phải chờ

B3. Giả sử hệ thống đã cấp phát cho Pi các tài nguyên mà nó yêu cầu và cập nhật tình trạng hệ thống như sau:

  Available[r]=Available[r]-kr; với mọi r
  Allocation[i,r] =Allocation[i,r]+ kr; với mọi r
  Need[i,r] = Need[i,r] - kr; với mọi r

B4: Kiểm tra trạng thái an toàn của hệ thống.

Dùng giải thuật "xác định trạng thái an toàn" để xác định trạng thái của hệ thống sau khi đã thử cấp tài nguyên cho Pi. Nếu trạng thái là an toàn thì các tài nguyên sẽ được cấp phát thật sự cho Pi. Ngược lại, Pi phải chờ.

Ví dụ giả sử tình trạng hiện hành của hệ thống được mô tả như trong bảng dưới đây. Cột Max là nhu cầu tối đa về mỗi tài nguyên của mỗi tiến trình, Allocation là số lượng của mỗi loại tài nguyên đã cấp cho mỗi tiến trình, Available là số lượng của mỗi loại tài nguyên còn có thể sử dụng.

Nếu tiến trình P2 yêu cầu 4 R1, 1 R3. hãy cho biết yêu cầu này có thể đáp ứng mà không xảy ra deadlock hay không?

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

Áp dụng Giải thuật banker

+ B0: Tính Need là nhu cầu còn lại về mỗi tài nguyên j của mỗi tiến trình i:

Need[i,j]=Max[i,j]-Allocation[i,j]

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

+ B1+B2: yêu cầu tài nguyên của P2 thoả đk ở B1, B2.

+ B3: Thử cấp phát cho P2, cập nhật tình trạng hệ thống

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

+ B4: Kiểm tra trạng thái an toàn của hệ thống (dùng giải thuật "xác định trạng thái an toàn"). Lần lượt chọn tiến trình để thử cấp phát:

- Chọn P2, thử cấp phát, g/s P2 thực thi xong thu hồi:

Available[j]= Available[j] + Allocation[i,j];

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

+ Chọn P1

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

P4	4	2	0	0	0	2			

+ Chọn P3:

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

+ Chọn P4:

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

Mọi tiến trình đã được cấp phát tài nguyên với yêu cầu cao nhất, nên trạng thái của hệ thống là an toàn, do đó có thể cấp phát các tài nguyên theo yêu cầu của P2.

3.5.5 Giải thuật phát hiện tắc nghẽn

Nếu khi tiến trình yêu cầu tài nguyên mà hệ thống cứ cấp phát thì tình trạng tắc nghẽn có thể xảy ra, khi đó hệ thống cần cung cấp giải thuật phát hiện tắc nghẽn và giải thuật phục hồi tình trạng trước khi tắc nghẽn. Ta có hai loại tài nguyên, mỗi loại sẽ có giải thuật phát hiện tắc nghẽn tương ứng.

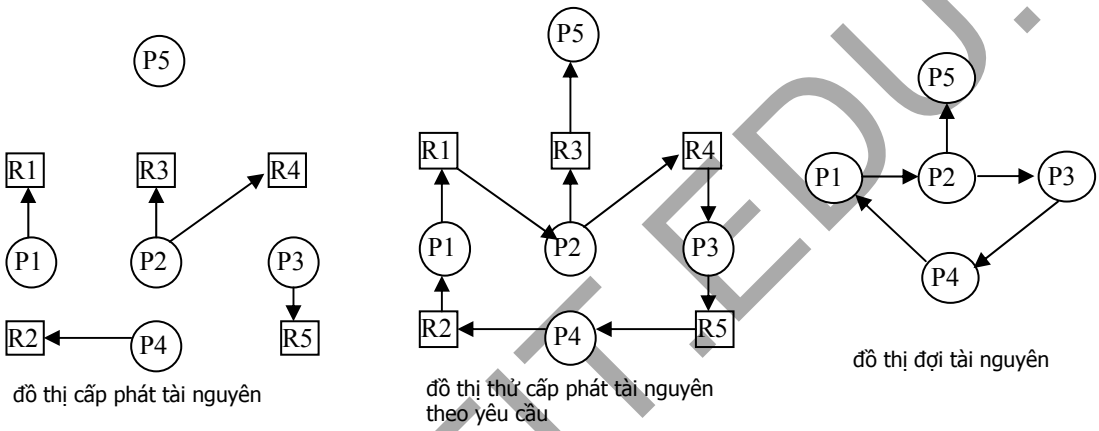
a/ Tài nguyên chỉ có một thể hiện

Dùng đồ thị đợi tài nguyên (wait-for graph), đồ thị này được xây dựng từ đồ thị cấp phát tài nguyên (resource allocation graph) bằng cách bỏ những đỉnh biểu diễn loại tài nguyên, khi đó một cạnh từ PI tới PJ nghĩa là PJ đang đợi PI giải phóng một tài nguyên mà PJ cần. Hệ thống bị tắc nghẽn nếu và chỉ nếu đồ thị đợi tài nguyên có chu trình, do đó để phát hiện tắc nghẽn ta chỉ cần dùng một giải thuật phát hiện chu trình (xem lý thuyết đồ thị).

Ví dụ: Các tiến trình đang chiếm giữ và đồng thời yêu cầu các tài nguyên như cho trong bảng dưới đây. Hỏi hệ thống có bị tắc nghẽn không?

Tiến trình	Chiếm giữ	Yêu cầu
P1	R1	R2
P2	R3, R4	R1
P3	R5	R4
P4	R2	R5
P5		R3

HD:Xây dựng đồ thị đợi tài nguyên:



Do đồ thị có chu trình nên hệ thống bị tắc nghẽn.

b/ Tài nguyên có nhiều thể hiện

Đồ thị đợi tài nguyên không thể áp dụng cho trường hợp này. Ta có thể áp dụng giải thuật sau để phát hiện tắc nghẽn (khá giống giải thuật cấp phát tài nguyên) :

Bước 1: Chọn Pi đầu tiên sao cho có yêu cầu tài nguyên có thể được đáp ứng, nếu không có thì hệ thống bị tắc nghẽn, ngược lại xuống Bước 2

Bước 2: Thử cấp phát tài nguyên cho Pi và kiểm tra trạng thái hệ thống, nếu hệ thống an toàn thì tới Bước 3, ngược lại thì quay lên Bước 1 tìm Pi kế tiếp.

Bước 3: Cấp phát tài nguyên cho Pi. Nếu tất cả Pi được đáp ứng thì hệ thống không bị tắc nghẽn, ngược lại quay lại Bước 1.

Giải thuật phát hiện tắc nghẽn có thể được gọi mỗi khi một yêu cầu cấp phát tài nguyên không được đáp ứng ngay, khi đó có thể xác định được tập hợp các tiến trình bị tắc nghẽn và cũng xác định được tiến trình gây ra tắc nghẽn nhưng sẽ tốn nhiều thời gian khi gọi giải thuật quá nhiều lần như vậy. Cách khác là gọi giải thuật theo một chu kỳ định trước, ví dụ 1 giờ gọi một lần hoặc gọi khi hiệu suất sử dụng CPU dưới 40%

3.5.6 Hiệu chỉnh tắc nghẽn

Khi đã phát hiện được tắc nghẽn, có hai lựa chọn chính để hiệu chỉnh tắc nghẽn :

a/ Hủy tiến trình trong tình trạng tắc nghẽn:

Hủy tất cả các tiến trình trong tình trạng tắc nghẽn hay hủy từng tiến trình liên quan cho đến khi không còn chu trình gây tắc nghẽn (tiến trình bị hủy sẽ bị thu hồi tất cả tài nguyên đã được cấp phát). Để chọn được tiến trình thích hợp bị hủy, phải dựa vào các yếu tố như độ ưu tiên, thời gian đã xử lý, số lượng tài nguyên đang chiếm giữ , số lượng tài nguyên còn yêu cầu thêm...

b/ Thu hồi tài nguyên:

Có thể hiệu chỉnh tắc nghẽn bằng cách thu hồi một số tài nguyên từ các tiến trình và cấp phát các tài nguyên này cho những tiến trình khác cho đến khi loại bỏ được chu trình tắc nghẽn. Khi thu hồi tài nguyên cần giải quyết 3 vấn đề sau:

+ Chọn lựa một nạn nhân: tiến trình nào sẽ bị thu hồi tài nguyên? và thu hồi những tài nguyên nào?

+ Trở lại trạng thái trước tắc nghẽn (rollback): khi thu hồi tài nguyên của một tiến trình, cần phải phục hồi trạng thái của tiến trình trở lại trạng thái gần nhất trước đó mà chưa bị tắc nghẽn.

+ Tình trạng « đói tài nguyên »: làm sao bảo đảm rằng không có một tiến trình nào luôn luôn bị thu hồi tài nguyên?

TÓM TẮT

+ Tiến trình là một chương trình đang xử lý, mỗi tiến trình có một không gian địa chỉ, một con trỏ lệnh, một tập các thanh ghi và stack riêng. Các tiến trình chỉ có thể liên lạc với nhau thông qua các cơ chế do hệ điều hành cung cấp.

+ Mục đích cho nhiều tiến trình hoạt động đồng thời là để tăng hiệu suất sử dụng CPU, tăng mức độ đa nhiệm, tăng tốc độ xử lý.

+ Một tiến trình có thể tạo nhiều tiểu trình, mỗi tiểu trình thực hiện một chức năng nào đó và thực thi đồng thời cũng bằng cách chia sẻ CPU. Các tiểu trình trong cùng một tiến trình dùng chung không gian địa chỉ tiến trình nhưng có con trỏ lệnh, tập các thanh ghi và stack riêng. Các tiểu trình liên lạc với nhau thông qua các biến toàn cục của tiến trình.

+ Các trạng thái của tiến trình : New, Ready, Running, Blocked, End

+ Tập lệnh của CPU được phân thành tập lệnh đặc quyền (các lệnh nếu sử dụng không chính xác, có thể ảnh hưởng xấu đến hệ thống) và tập lệnh không đặc quyền (không ảnh hưởng tới hệ thống). Phần cứng chỉ cho phép các lệnh đặc quyền được thực hiện trong chế độ đặc quyền. Hệ điều hành hoạt động trong chế độ đặc quyền, các tiến trình của người dùng sẽ hoạt động trong chế độ không đặc quyền

+ Hệ điều hành quản lý các tiến trình thông qua bảng tiến trình, mỗi mục trong bảng gọi là khối quản lý tiến trình lưu thông tin về một tiến trình gồm có: định danh của tiến trình , trạng thái tiến trình, ngữ cảnh của tiến trình, thông tin giao tiếp, thông tin thống kê.

+ Các thao tác trên tiến trình: tạo tiến trình, kết thúc tiến trình, tạm dừng tiến trình, tái kích hoạt tiến trình, thay đổi độ ưu tiên tiến trình.

+ Mỗi tài nguyên được quản lý bằng một cấu trúc gọi là khối quản lý tài nguyên chứa các thông tin sau : định danh tài nguyên, trạng thái tài nguyên, hàng đợi trên một tài nguyên, bộ cấp phát.

+ Một số đặc tính của tiến trình: tính hướng nhập/xuất, tính hướng xử lý, tiến trình tương tác hay xử lý theo lô, độ ưu tiên của tiến trình, thời gian đã sử dụng CPU của tiến trình, thời gian còn lại tiến trình cần để hoàn tất.

+ Tiến trình được điều phối thông qua bộ điều phối và bộ phân phối. Bộ điều phối sử dụng một giải thuật thích hợp để lựa chọn tiến trình được xử lý tiếp theo. Bộ phân phối chịu trách nhiệm cập nhật ngữ cảnh của tiến trình bị tạm ngưng và trao CPU cho tiến trình được chọn bởi bộ điều phối để tiến trình thực thi.

+ Mục tiêu của bộ điều phối: sự công bằng, tính hiệu qủa, thời gian đáp ứng hợp lý, thời gian lưu lại trong hệ thống, thông lượng tối đa. Các nguyên lý điều phối: điều phối độc quyền, điều phối không độc quyền.

+ Thời điểm thực hiện điều phối: tiến trình chuyển từ trạng thái running sang trạng thái blocked, hoặc tiến trình chuyển từ trạng thái running sang trạng thái ready, hoặc tiến trình chuyển từ trạng thái blocked sang trạng thái ready, hoặc tiến trình kết thúc, hoặc tiến trình có độ ưu tiên cao hơn xuất hiện.

+ Để thực hiện điều phối, hệ điều hành sử dụng ba loại danh sách là: danh sách tác vụ , danh sách sẵn sàng, danh sách chờ đợi. Các thuật toán điều phối: FIFO, Round Robin, độ ưu tiên, SJF, nhiều mức độ ưu tiên,điều phối xổ số.

+ Cơ chế liên lạc giữa các tiến trình: liên lạc bằng tín hiệu, liên lạc bằng đường ống, liên lạc qua vùng nhớ chia sẻ, liên lạc bằng thông điệp, liên lạc qua socket.

+ Đồng bộ các tiến trình, miền găng. Các giải pháp đồng bộ: semaphore, monitor, trao đổi thông điệp.

+ Tình trạng tắc nghẽn, điều kiện xuất hiện tắc nghẽn, các phương pháp xử lý tắc nghẽn. Giải thuật cấp phát tài nguyên.

CÂU HỎI VÀ BÀI TẬP

1. Xét giải pháp phần mềm do Dekker đề nghị để tổ chức truy xuất độc quyền cho hai tiến trình. Hai tiến trình P0, P1 chia sẻ các biến sau :

```
int turn; // đến phiên i hay j (i,j=0..1)
int flag [2]; // khởi động là FALSE
//cấu trúc tiến trình Pi
While (TRUE)
{
        flag[i] = TRUE;
        while (flag[j])
                if (turn == j)
                {
                        flag[i]= FALSE;
                        while (turn == j) ;
                        flag[i]= TRUE;
                }
```

```
        critical_section();
        turn= j; flag[i]= FALSE;
        non_critical_section();
}
```

Giải pháp này có thỏa mãn 4 yêu cầu của bài toán miền găng không ?

2. Xét giải pháp đồng bộ hoá sau:

```
while (TRUE)
{
        int j = 1-i;
        flag[i]= TRUE; turn = i;
        while (turn == j && flag[j]==TRUE);
        critical-section ();
        flag[i] = FALSE;
        Noncritical-section ();
}
```

Đây có phải là một giải pháp bảo đảm được độc quyền truy xuất không ?

3. Giả sử một máy tính không có lệnh TSL, nhưng có lệnh swap hoán đổi nội dung của hai từ nhớ bằng một thao tác độc quyền :

```
void swap(int &a, int &b)
{
        int temp=a;
        a= b; b= temp;
}
```

Sử dụng lệnh này có thể tổ chức truy xuất độc quyền không ? Nếu có xây dựng cấu trúc chương trình tương ứng.

4. Trong giải pháp peterson bỏ biến turn có được không?

5. Phát triển giải pháp Peterson cho nhiều tiến trình

6. Chứng tỏ rằng nếu các lệnh Down và Up trên semaphore không thực hiện một cách không thể phân chia, thì không thể dùng semaphore để giải quyết bài toán miền găng.

7. Xét hai tiến trình xử lý đoạn chương trình sau:

process P1 { A1 ; A2 }

process P2 { B1 ; B2 }

Đồng bộ hoá hoạt động của hai tiến trình này sao cho cả A1 và B1 đều hoàn tất trước khi A2 hay B2 bắt đầu .

8. Tổng quát hoá bai 6: cho các tiến trình xử lý đoạn chương trình sau:

process P1 { for (i = 1; i <= 100; i ++) Ai ;}

process P2 { for (j = 1; j <= 100; j ++) Bj ;}

Đồng bộ hoá hoạt động của hai tiến trình này sao cho cả với k bất kỳ (2 <= k <= 100), Ak chỉ có thể bắt đầu khi B(k-1) đã kết thúc, và Bk chỉ có thể bắt đầu khi A(k-1) đã kết thúc.

9. Sử dụng semaphore để viết lại chương trình sau theo mô hình xử lý đồng hành:

w = x1 * x2 (1) v = x3 * x4 (2) y = v * x5 (3) z = v * x6 (4)

y = w * y (5) z = w * z (6) ans = y + z (7)

(x1,x2,x3,x4,x5,x6 là các hằng số)

10. Viết lại bài 9 dùng monitor

11. Xét hai tiến trình sau:

process A

{ while (1) na = na +1;}

process B

{ while (1) nb = nb +1;}

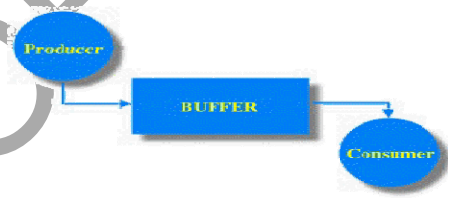
a) Đồng bộ hoá xử lý của hai tiến trình trên, sao cho tại bất kỳ thời điểm nào cũng có nb < na <= nb +10

b) Nếu giảm điều kiện chỉ là na <= nb +10, giải pháp của bạn sẽ được sửa chữa như thế nào ?

c) Giải pháp của bạn có còn đúng nếu có nhiều tiến trình loại A và B cùng thực hiện?

12. Bài toán Người sản xuất – Người tiêu thụ (Producer-Consumer)

Hai tiến trình cùng chia sẻ một bộ đệm có kích thước giới hạn. Một tiến trình tạo dữ liệu, đặt dữ liệu vào bộ đệm (người sản xuất) và một tiến trình lấy dữ liệu từ bộ đệm để xử lý (người tiêu thụ).



Hai tiến trình cần thoả các điều kiện sau :

- ĐK1: Tiến trình sản xuất không được ghi dữ liệu vào bộ đệm đã đầy.
- ĐK2: Tiến trình tiêu thụ không được đọc dữ liệu từ bộ đệm đang trống.
- ĐK3: Hai tiến trình cùng loại hoặc khác loại đều không được truy xuất bộ đệm cùng lúc.

13. Bài toán Readers-Writers

Khi cho phép nhiều tiến trình truy xuất cơ sở dữ liệu dùng chung các hệ quản trị CSDL cần đảm bảo các điều kiện sau :

- Đk1: khi có reader thì không có writer nhưng có thể có các reader khác tx dl
- Đk2: Khi có writer thì không có writer hoặc reader nào khác tx dl.

(Các điều kiện này cần có để đảm bảo tính nhất quán của dữ liệu.)

14. Bài toán Tạo phân tử H2O

Đồng bộ hoạt động của một phòng thí nghiệm sử dụng nhiều tiến trình đồng hành sau để tạo các phân tử H2O:

```
MakeH()
{
        while (true)
        Make-Hydro(); // tạo 1 nguyên tử H
}
MakeO()
{
        while (true)
                Make-Oxy(); //tạo 1 nguyên tử O
}
/* Tiến trình MakeWater hoạt động đồng hành  với các tiến trình MakeH, MakeO, chờ có đủ 2 H và 1 O để tạo H2O */
MakeWater()
{
   while (True)
        Make-Water(); //Tạo 1 phân tử H2O
}
```

15. Xét một giải pháp semaphore đúng cho bài toán Dining philosophers :

```
#define N               5
#define LEFT            (i-1)%N
#define RIGHT           (i+1)%N
#define THINKING        0
#define HUNGRY          1
#define EATING          2
int             state[N];
semaphore       mutex = 1;
semaphore       s[N];//gan tri ban dau =0
//tiến trình mô phỏng triết gia thứ i
```

```
void philosopher( int i) // i là triết gia thứ i : 0..N-1
{
	while (TRUE)
	{
		think(); // Suy nghĩ
		take_forks(i); // yêu cầu đến khi có đủ 2 nĩa
		eat(); // yum-yum, spaghetti
		put_forks(i); // đặt cả 2 nĩa lên bàn lại
	}
}
//kiểm tra điều kiện được ăn
void test ( int i) // i là triết gia thứ i : 0..N-1
{
	if(state[i]==HUNGRY && state[LEFT]!=EATING && state[RIGHT]!= EATING)
	{
		state[i] = EATING;
		up(s[i]);
	}
}
//yêu cầu lấy 2 nĩa
void take_forks ( int i) // i là triết gia thứ i : 0..N-1
{
	while (TRUE)
	{
		down(mutex); // vào miền găng
		state[i] = HUNGRY; // ghi nhận triết gia i đã đói
		test(i); // cố gắng lấy 2 nĩa
		up(mutex); // ra khỏi miền găng
		down(s[i]); // chờ nếu không có đủ 2 nĩa
	}
}
//đặt 2 nĩa xuống
void put_forks ( int i) // i là triết gia thứ i : 0..N-1
{
	while (TRUE)
	{
		down(mutex); // vào miền găng
		state[i] = THINKING; // ghi nhận triết gia i ăn xong
```

```
            test(LEFT); // kiểm tra người bên trái đã có thể ăn?
            test(RIGHT); // kiểm tra người bên phải đã có thể ăn?
            up(mutex); // ra khỏi miền găng
        }
}
```

a) Tại sao phải đặt state[i] = HUNGRY trong take_forks ?

b) Giả sử trong put_forks, lệnh gán state[i] = THINKING được thực hiện sau hai lệnh test(LEFT), test(RIGHT). Điều này ảnh hưởng thế nào đến giải pháp cho 3 triết gia? Cho 100 triết gia?

16. Một cửa hiệu cắt tóc có một thợ, một ghế cắt tóc và N ghế cho khách đợi. Nếu không có khách, thợ cắt tóc sẽ ngồi vào ghế cắt tóc và ngủ thiếp đi. Khi một khách hàng vào tiệm, anh ta phải đánh thức người thợ. Nếu một khách hàng vào tiệm khi người thợ đang bận cắt tóc cho khách hàng khác, người mới vào sẽ ngồi chờ nếu có ghế đợi trống, hoặc rời khỏi tiệm nếu đã có N người đợi (hết ghế). Xây dựng một giải pháp với semaphore để thực hiện đồng bộ hoá hoạt động của thợ và khách hàng trong cửa hiệu cắt tóc này.

17. Bài toán Cây cầu cũ

Người ta chỉ có cho phép tối đa 3 xe lưu thông đồng thời qua một cây cầu rất cũ. Hãy xây dựng thủ tục ArriveBridge(int direction) và ExitBridge() để kiểm soát giao thông trên cầu sao cho :

- Tại mỗi thời điểm, chỉ cho phép tối đa 3 xe lưu thông trên cầu.
- Tại mỗi thời điểm, chỉ cho phép tối đa 2 xe lưu thông cùng hướng trên cầu.

Mỗi chiếc xe khi đến đầu cầu sẽ gọi ArriveBridge(direction) để kiểm tra điều kiện lên cầu, và khi đã qua cầu được sẽ gọi ExitBridge() để báo hiệu kết thúc. Giả sử hoạt động của mỗi chiếc xe được mô tả bằng một tiến trình Car() sau đây:

```
Car(int direction) /* direction xác định hướng di chuyển của mỗi chiếc xe.*/
{
        ArriveBridge(direction); //tới cầu
        OnBridge(); //lên cầu
        ExitBridge();// Qua cầu
}
```

18. Bài toán Qua sông

Để vượt qua sông, các nhân viên Microsof và các Linux hacker cùng sử dụng một bến sông và phải chia sẻ một số thuyền đặc biệt. Mỗi chiếc thuyền này chỉ cho phép chở 1 lần 4 người, và phải có đủ 4 người mới khởi hành được. Để bảo đảm an toàn cho cả 2 phía, cần tuân thủ các luật sau :

a. Không chấp nhận 3 nhân viên Microsoft và 1 Linux hacker trên cùng một chiếc thuyền.

b. Ngược lại, không chấp nhận 3 Linux hacker và 1 nhân viên Microsoft trên cùng một chiếc thuyền.

c. Tất cả các trường hợp kết hợp khác đều hợp pháp.

d. Thuyền chỉ khởi hành khi đã có đủ 4 hành khách.

Cần xây dựng 2 thủ tục HackerArrives() và EmployeeArrives() được gọi tương ứng bởi 1 hacker hoặc 1 nhân viên khi họ đến bờ sông để kiểm tra điều kiện có cho phép họ xuống thuyền không ? Các thủ tục này sẽ sắp xếp những người thích hợp có thể lên thuyền. Những người đã được lên thuyền khi thuyền chưa đầy sẽ phải chờ đến khi người thứ 4 xuống thuyền mới có thể khởi hành qua sông. (Không quan tâm đến số lương thuyền hay việc thuyền qua sông rồi trở lại…Xem như luôn có thuyền để sắp xếp theo các yêu cầu hợp lệ)

Giả sử hoạt động của mỗi hacker được mô tả bằng một tiến trình Hacker() sau đây:

```
Hacker()
{
        RuntoRiver(); // Đi đến bờ sông
        HackerArrives (); // Kiểm tra điều kiện xuống thuyền
        CrossRiver(); // Khởi hành qua sông
}
```

và hoạt động của mỗi nhân viên được mô tả bằng một tiến trình Employee() sau đây:

```
Employee()
{
        RuntoRiver(); // Đi đến bờ sông
        EmployeeArrives (); // Kiểm tra điều kiện xuống thuyền
        CrossRiver(); // Khởi hành qua sông
}
```

19. Bài toán Điều phối hành khách xe bus tại một trạm dừng

Mỗi xe bus có 10 chỗ, 4 chỗ dành cho khách ngồi xe lăn, 6 chỗ dành cho khách bình thường, khi xe đầy khách thì sẽ khởi hành. Có thể có nhiều xe và nhiều hành khách vào bến cùng lúc, nguyên tắc điều phối sẽ xếp khách vào đầy một xe, cho xe này khởi hành rồi mới điều phối cho xe khác.

Giả sử hoạt động điều phối khách cho 1 chiếc xe bus được mô tả qua tiến trình GetPassengers(); hoạt động của mỗi loại hành khách được mô tả bằng tiến trình WheelPassenger() và NonWheelPassenger(). Hãy sửa chữa các đoạn code, sử dụng semaphore để đồng bộ hoá .

```
GetPassenger() //chương trình điều phối khách cho 1 xe
{
        ArriveTerminal(); // tiếp nhận một xe vào bến
        OpenDoor(); // mở cửa xe
        for (int i=0; i<4; i++) // tiếp nhận các khách ngồi xe lăn
        {
                ArrangeSeat(); // đưa 1 khách ngồi xe lăn vào chỗ
        }
        for (int i=0; i<6; i++) // tiếp nhận các khách bình thường
```

```
        {
                ArrangeSeat(); // đưa 1 khách bình thường vào chỗ
        }
        CloseDoor(); // đóng cửa xe
        DepartTerminal(); // cho một xe rời bến
}
WheelPassenger() //chương trình tạo khách ngồi xe lăn
{
        ArriveTerminal(); // đến bến
        GetOnBus(); // lên xe
}
NonWheelPassenger() // chương trình tạo khách bình thường
{
        ArriveTerminal(); // đến bến
        GetOnBus(); // lên xe
}
```

20. Nhà máy sản xuất thiết bị xe hơi, có 2 bộ phận hoạt động song song

- Bộ phận sản xuất 1 khung xe :

```
MakeChassis()
{
        Produce_chassis();// tạo khung xe
}
```

- Bộ phận sản xuất 1 bánh xe :

```
Make_Tires()
{
        // tạo bánh xe và gắn vào khung xe
        Produce_tire();
        Put_tire_to_Chassis();
}
```

Hãy đồng bộ hoạt động trong việc sản xuất xe hơi theo nguyên tắc sau :

- Sản xuất một khung xe, trước khi tạo bánh xe.
- Cần có đủ 4 bánh xe cho 1 khung xe được sản xuất ra, sau đó mới tiếp tục sản xuất khung xe khác…

21. Thuật toán các triết gia ăn tối sau đúng hay sai?

```
semaphore s[5]; //có các giá ban trị đầu bằng 1
//tiến trình  triết gia thứ i:
{
```

```
        down(s[i]);             //lấy đũa
        down(s[(i+1)%5]);       //lấy đũa của người bên cạnh
        eat();
        up(s[i]);               //bỏ đũa
        up(s[(i+1)%5]);         //trả đũa cho người bên cạnh
}
```

22. Xét trạng thái hệ thống:

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

Nếu tiến trình P2 yêu cầu 4 cho R1, 1 cho R3. hãy cho biết yêu cầu này có thể đáp ứng mà bảo đảm không xảy ra tình trạng deadlock hay không ?

23. Xét trạng thái hệ thống sau:

	Max				Allocation				Available			
	A	B	C	D	A	B	C	D	A	B	C	D
P1	0	0	1	2	0	0	1	2	1	5	2	0
P2	1	7	5	0	1	0	0	0				
P3	2	3	5	6	1	3	5	4				
P4	0	6	5	2	0	6	3	2				
P5	0	6	5	6	0	0	1	4				

a) Cho biết nội dung của bảng Need.

b) Hệ thông có ở trạng thái an toàn không?

c) Nếu tiến trình P2 có yêu cầu tài nguyên (0,4,2,0), yêu cầu này có được đáp ứng tức thời không?

TÀI LIỆU THAM KHẢO

[1]. Gary J. Nutt, University of Colorado. Centralized And Distributed Operating Systems. Second Edition, 2000.

[2]. Robert Switzer. Operating Systems, A Practical Approach. Prentice-Hall International, Inc. 1993.

[3]. Andrew S. Tanenbaum. Modern Operating Systems. Prentice-Hall International, Inc. Second Edition, 2001.

[4]. Abraham Silberschatz & Peter Baer Galvin. Operating System concepts. John Wiley & Sons, Inc. Fifth Edition, 1999.

[5]. H. M. Deitel. Operating Systems. Addison-Wesley Inc. Second Edition, 1999.

[6]. Trần Hạnh Nhi & Lê Khắc Nhiên Ân & Hoàng Kiếm. Giáo trình hệ điều hành (tập 1 & 2). ĐHKHTN 2000.

CHƯƠNG 4
QUẢN LÝ BỘ NHỚ

Chương "QUẢN LÝ BỘ NHỚ" sẽ giới thiệu và giải thích các vấn đề sau:

4.1 Các vấn đề phát sinh khi quản lý bộ nhớ.

4.2 Các mô hình cấp phát bộ nhớ.

4.3 Bộ nhớ ảo

4.1 CÁC VẤN ĐỀ PHÁT SINH KHI QUẢN LÝ BỘ NHỚ

+ Chuyển đổi địa chỉ tương đối trong chương trình thành địa chỉ thực trong bộ nhớ chính.

+ Quản lý bộ nhớ đã cấp phát và chưa cấp phát.

+ Các kỹ thuật cấp phát bộ nhớ sao cho:

- Ngăn chặn các tiến trình xâm phạm đến vùng nhớ đã được cấp phát cho tiến trình khác.

- Cho phép nhiều tiến trình có thể dùng chung một phần bộ nhớ của nhau.

- Mở rộng bộ nhớ để có thể lưu trữ được nhiều tiến trình đồng thời.

4.1.1 Chuyển đổi địa chỉ tương đối sang tuyệt đối

Các địa chỉ trong chương trình thực thi (dạng exe) là địa chỉ tương đối, và cần được chuyển đổi các địa chỉ này thành các địa chỉ tuyệt đối trong bộ nhớ chính. Việc chuyển đổi có thể xảy ra vào một trong những thời điểm sau:

+ Thời điểm biên dịch (compile time):

Nếu tại thời điểm biên dịch, có thể biết vị trí mà tiến trình sẽ được nạp vào trong bộ nhớ, trình biên dịch có thể phát sinh ngay mã với các địa chỉ tuyệt đối. Tuy nhiên, nếu về sau có sự thay đổi vị trí của chương trình, cần phải biên dịch lại chương trình. Ví dụ các chương trình .com chạy trên hệ điều hành MS-DOS có mã tuyệt đối ngay khi biên dịch.

+ Thời điểm nạp (load time):

Nếu tại thời điểm biên dịch, chưa thể biết vị trí mà tiến trình sẽ được nạp vào trong bộ nhớ, trình biên dịch chỉ phát sinh mã tương đối. Khi nạp chương trình vào bộ nhớ, hệ điều hành sẽ chuyển các địa chỉ tương đối thành địa chỉ tuyệt đối do đã biết vị trí bắt đầu lưu trữ tiến trình. Khi có sự thay đổi vị trí lưu trữ, cần nạp lại chương trình để thực hiện lại việc chuyển đổi địa chỉ, không cần biên dịch lại chương trình.

+ Thời điểm xử lý (execution time):

Nếu có nhu cầu di chuyển tiến trình từ vùng nhớ này sang vùng nhớ khác trong quá trình tiến trình xử lý, thì việc chuyển đổi địa chỉ sẽ được thực hiện vào lúc tiến trình thực thi. Chức năng chuyển đổi địa chỉ do phần cứng cung cấp gọi là MMU (memory management unit). Các hệ điều hành thường dùng việc chuyển đổi theo cách này.

4.1.2 Không gian địa chỉ ảo và không gian địa chỉ vật lý

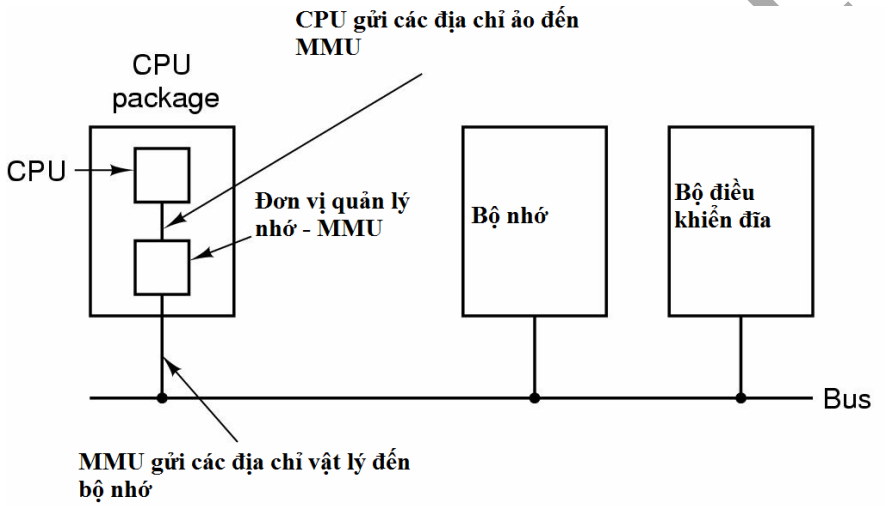
+ ***Địa chỉ ảo (địa chỉ logic):*** là địa chỉ do bộ xử lý (CPU) tạo ra.

+ ***Địa chỉ vật lý (địa chỉ physic):*** là địa chỉ thực trong bộ nhớ chính, địa chỉ vật lý còn gọi là địa chỉ tuyệt đối/địa chỉ thực.

+ ***Không gian địa chỉ ảo của tiến trình***: là tập hợp tất cả các địa chỉ ảo của một tiến trình.

+ ***Không gian địa chỉ vật lý của tiến trìn***h: là tập hợp tất cả các địa chỉ vật lý tương ứng với các địa chỉ ảo.

Khi chương trình nạp vào bộ nhớ các địa chỉ tương đối trong chương trình được CPU chuyển thành địa chỉ ảo, khi thực thi, địa chỉ ảo được hệ điều hành kết hợp với phần cứng MMU chuyển thành địa chỉ vật lý .Tóm lại chỉ có khái niệm địa chỉ ảo nếu việc chuyển đổi địa chỉ xảy ra vào thời điểm xử lý, khi đó tiến trình chỉ thao tác trên các địa chỉ ảo, địa chỉ vật lý chỉ được xác định khi thực hiện truy xuất bộ nhớ vật lý.



Hình 4.1: CPU gởi địa chỉ ảo tới MMU, MMU chuyển địa chỉ ảo thành địa chỉ vật lý

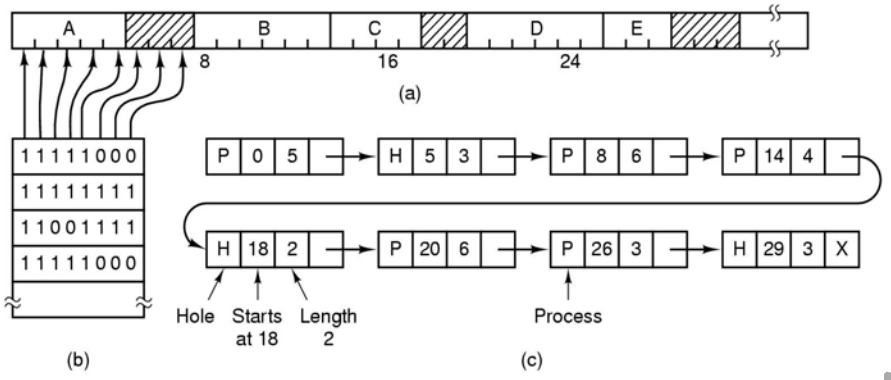
4.1.3 Quản lý bộ nhớ đã cấp phát và chưa cấp phát

Hệ điều hành cần lưu trữ thông tin về phần bộ nhớ đã cấp phát và phần bộ nhớ chưa cấp phát. Nếu đã cấp phát thì cấp cho tiến trình nào. Khi cần cấp phát bộ nhớ cho một tiến trình thì làm sao tìm được phần bộ nhớ trống thích hợp nhanh chóng và khi bộ nhớ bị phân mảnh thì cần dồn bộ nhớ lại để tận dụng bộ nhớ và để tiến trình thực thi nhanh hơn.

4.1.3.1 Các phương pháp quản lý việc cấp phát bộ nhớ:

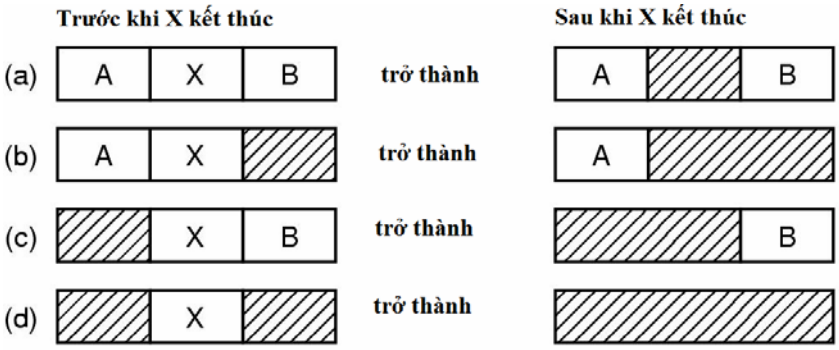
a/ Sử dụng dãy bit : bít thứ i bằng 1 là khối thứ i đã cấp phát, bằng 0 là chưa cấp phát.

b/ Sử dụng danh sách liên kết: mỗi nút của danh sách liên kết lưu thông tin một vùng nhớ chứa tiến trình (P) hay vùng nhớ trống giữa hai tiến trình (H).



Hình 4.2: quản lý việc cấp phát bộ nhớ bằng dãy bit hoặc danh sách liên kết

Trước khi tiến trình X kết thúc, có 4 trường hợp có thể xảy ra và khi tiến trình X kết thúc, hệ điều hành cần gom những nút trống gần nhau.



Hình 4.3: các trường hợp có thể xảy ra trước khi tiến trình X kết thúc

4.1.3.2 Các thuật toán chọn một đoạn trống:

+ ***First-fit***: chọn đoạn trống đầu tiên đủ lớn.

+ ***Best-fit***: chọn đoạn trống nhỏ nhất nhưng đủ lớn để thỏa mãn nhu cầu.

+ ***Worst-fit*** : chọn đoạn trống lớn nhất.

4.2 CÁC MÔ HÌNH CẤP PHÁT BỘ NHỚ

Có hai mô hình dùng để cấp phát bộ nhớ cho một tiến trình là:

+ Cấp phát liên tục: tiến trình được nạp vào một vùng nhớ liên tục.

+ Cấp phát không liên tục: tiến trình được nạp vào một vùng nhớ không liên tục

4.2.1 Mô hình cấp phát liên tục

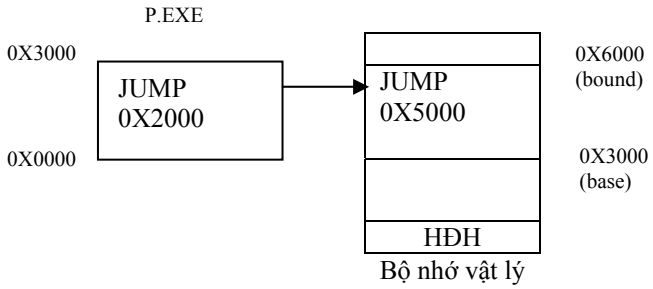
Có hai mô hình cấp phát bộ nhớ liên tục là mô hình Linker-Loader hoặc mô hình Base & Limit.

4.2.1.1 Mô hình Linker_Loader:

Chương trình được nạp vào một vùng nhớ liên tục đủ lớn để chứa toàn bộ chương trình. Hệ điều hành sẽ chuyển các địa chỉ tương đối về địa chỉ tuyệt đối (địa chỉ vật lý) ngay khi nạp chương trình, theo công thức:

địa chỉ tuyệt đối = địa chỉ bắt đầu nạp tiến trình + địa chỉ tương đối.

Ví dụ: xét chương trình P.EXE có lệnh Jump 0X200, . Giả sử chương trình được nạp tại địa chỉ 0X300, khi đó địa chỉ tương đối 0X200 sẽ được chuyển thành địa chỉ vật lý là 0X300+0X200=0X500



Hình 4.4: Một ví dụ về chuyển đổi địa chỉ tương đối thành địa chỉ vật lý trong mô hình linker-loader

Chương trình khi nạp vào bộ nhớ cho thực thi thì gọi là tiến trình, vậy trường hợp này các địa chỉ trong tiến trình là địa chỉ tuyệt đối, còn địa chỉ trong chương trình là địa chỉ tương đối.

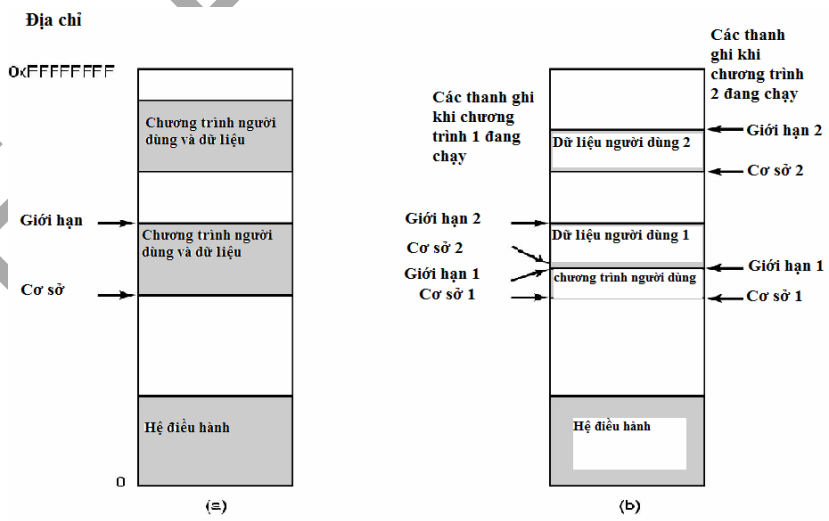
Nhận xét:

+ Vì việc chuyển đổi địa chỉ chỉ thực hiện vào lúc nạp nên sau khi nạp không thể di chuyển tiến trình trong bộ nhớ

+ Do không có cơ chế kiểm soát địa chỉ mà tiến trình truy cập, nên không thể bảo vệ một tiến trình bị một tiến trình khác truy xuất bộ nhớ của tiến trình một cách trái phép.

4.2.1.2 Mô hình Base & Limit

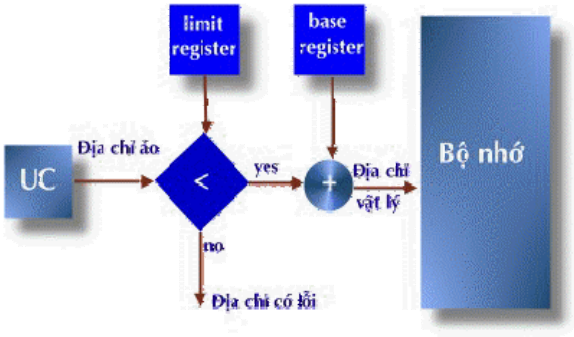
Giống như mô hình Linker-Loader nhưng phần cứng cần cung cấp hai thanh ghi, một thanh ghi nền (base register) và một thanh ghi giới hạn (limit register). Khi một tiến trình được cấp phát vùng nhớ, hệ điều hành cất vào thanh ghi nền địa chỉ bắt đầu của vùng nhớ cấp phát cho tiến trình, và cất vào thanh ghi giới hạn kích thước của tiến trình.



Hình 4.5: một ví dụ về mô hình base&limit

Khi tiến trình thực thi, mỗi địa chỉ ảo (địa chỉ ảo cũng chính là địa chỉ tương đối) sẽ được MMU so sánh với thanh ghi giới hạn để bảo đảm tiến trình không truy xuất ngoài phạm vi vùng nhớ

được cấp cho nó. Sau đó địa chỉ ảo được cộng với giá trị trong thanh ghi nền để cho ra địa chỉ tuyệt đối trong bộ nhớ.



Hình 4.6: cơ chế MMU trong mô hình base&limit

Nhận xét:

+ Có thể di chuyển các chương trình trong bộ nhớ vì do tiến trình được nạp ở dạng địa chỉ ảo, khi tiến trình được di chuyển đến một vị trí mới, hệ điều hành chỉ cần nạp lại giá trị cho thanh ghi nền, và việc chuyển đổi địa chỉ được MMU thực hiện vào thời điểm xử lý.

+ Có thể có hiện tượng phân mảnh ngoại vi (external fragmentation): tổng vùng nhớ trống đủ để thoả mãn yêu cầu, nhưng các vùng nhớ này lại không liên tục nên không đủ để cấp cho một tiến trình khác. Có thể áp dụng kỹ thuật "dồn bộ nhớ " (memory compaction) để kết hợp các mảnh bộ nhớ nhỏ rời rạc thành một vùng nhớ lớn liên tục, tuy nhiên kỹ thuật này đòi hỏi nhiều thời gian xử lý. Ví dụ về sự phân mảnh ngoại vi của bộ nhớ, các tiến trình liên tục vào ra bộ nhớ, sau một thời gian sẽ để lại các vùng nhớ nhỏ mà không thể chứa bất kỳ tiến trình nào.

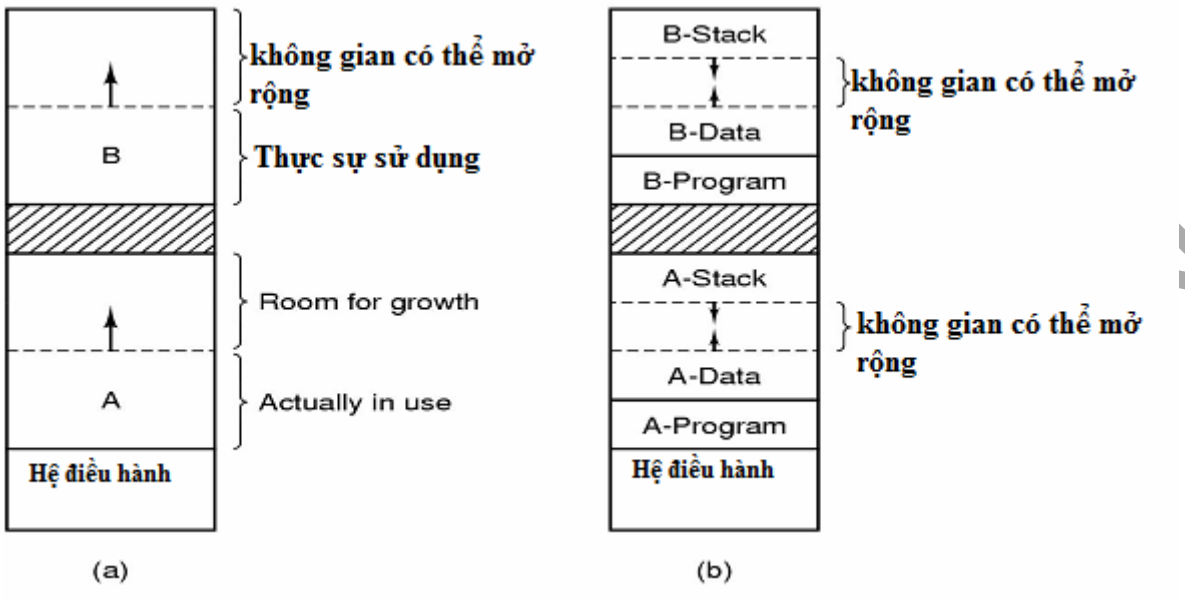
<table>
<tr><td rowspan="3"></td><td rowspan="3"></td><td></td><td>D</td><td>D</td><td>D</td><td>D</td><td>D</td></tr>
<tr><td rowspan="2">C</td><td rowspan="2">C</td><td rowspan="2"></td><td></td><td></td><td></td></tr>
<tr><td>E</td><td>E</td><td>E</td></tr>
<tr><td></td><td>B</td><td>B</td><td>B</td><td>B</td><td>B</td><td>B</td><td>B</td></tr>
<tr><td>A</td><td>A</td><td>A</td><td>A</td><td>A</td><td>A</td><td rowspan="2"></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td>F</td></tr>
<tr><td>OS</td><td>OS</td><td>OS</td><td>OS</td><td>OS</td><td>OS</td><td>OS</td><td>OS</td></tr>
</table>

Hình 4.7: một ví dụ về sự phân mảnh ngoại vi trong mô hình cấp phát liên tục

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

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

+ ***Cấp phát dư vùng nhớ cho tiến trình*** : cấp phát dự phòng cho tiến trình một vùng nhớ lớn hơn yêu cầu ban đầu của tiến trình.



Hình 4.8: dành chỗ trống để tiến trình có thể phát triển trong mô hình cấp phát liên tục

+ Tiến trình luôn được lưu trữ trong bộ nhớ suốt quá trình xử lý của nó nên tính đa chương của hệ điều hành sẽ bị hạn chế bởi kích thước bộ nhớ và kích thước của các tiến trình trong bộ nhớ. Cách giải quyết là khi tiến trình bị khóa (đợi tài nguyên, đợi một sự kiện,…) hoặc tiến trình sử dụng hết thời gian CPU dành cho nó, nó có thể được chuyển tạm thời ra bộ nhớ phụ (đĩa,…) và sau này được nạp trở lại vào bộ nhớ chính để tiếp tục xử lý (kỹ thuật swapping).

Để tránh tình trạng bộ nhớ bị phân mảnh vì do phải cấp phát một vùng nhớ liên tục cho tiến trình, hệ điều hành có thể cấp phát cho tiến trình những vùng nhớ tự do bất kỳ, không cần liên tục.

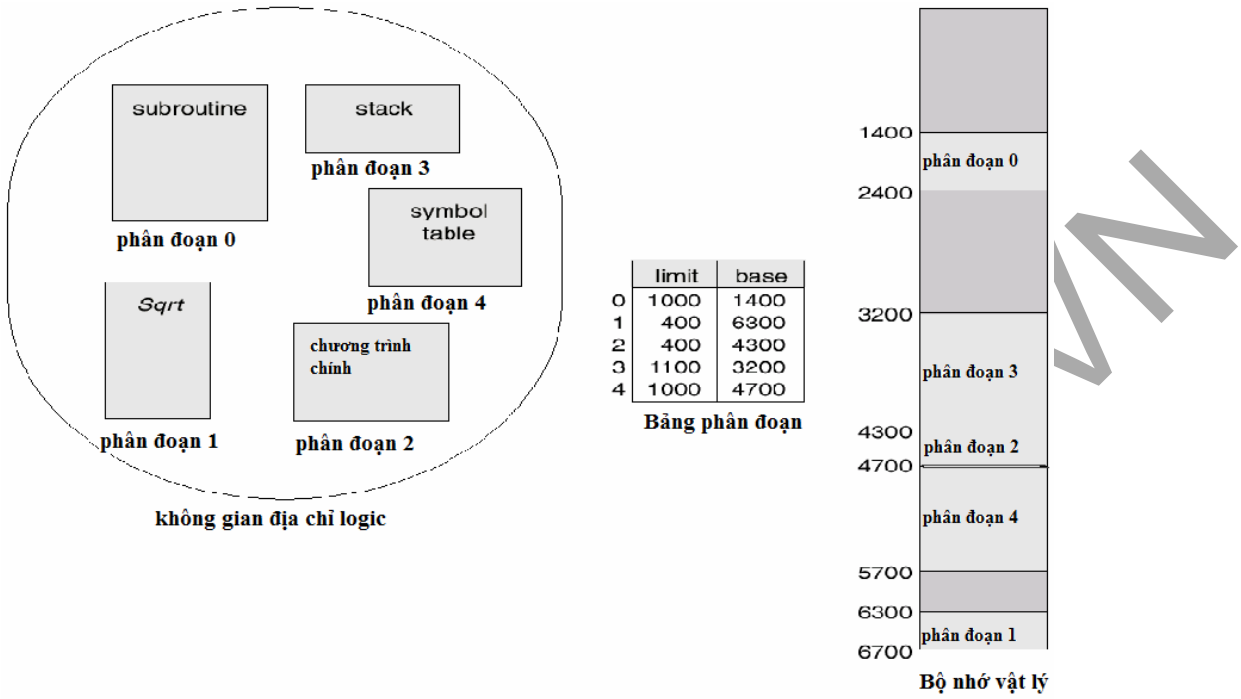
4.2.2 Mô hình cấp phát không liên tục

Có ba mô hình cấp phát bộ nhớ không liên tục là mô hình phân đoạn, mô hình phân trang và mô hình phân đoạn kết hợp phân trang.

4.2.2.1 Mô hình phân đoạn (Segmentation)

Một chương trình được người lập trình chia thành nhiều phân đoạn, mỗi phân đoạn có ngữ nghĩa khác nhau và hệ điều hành có thể nạp các phân đoạn vào bộ nhớ tại các vị trí không liên tục.

Ví dụ: chương trình chia làm 5 phân đoạn (segment), mỗi phân đoạn được nạp vào vùng nhớ trống có thể không liên tục.



Hình 4.9: mô hình phân đoạn trong kỹ thuật cấp phát bộ nhớ không liên tục

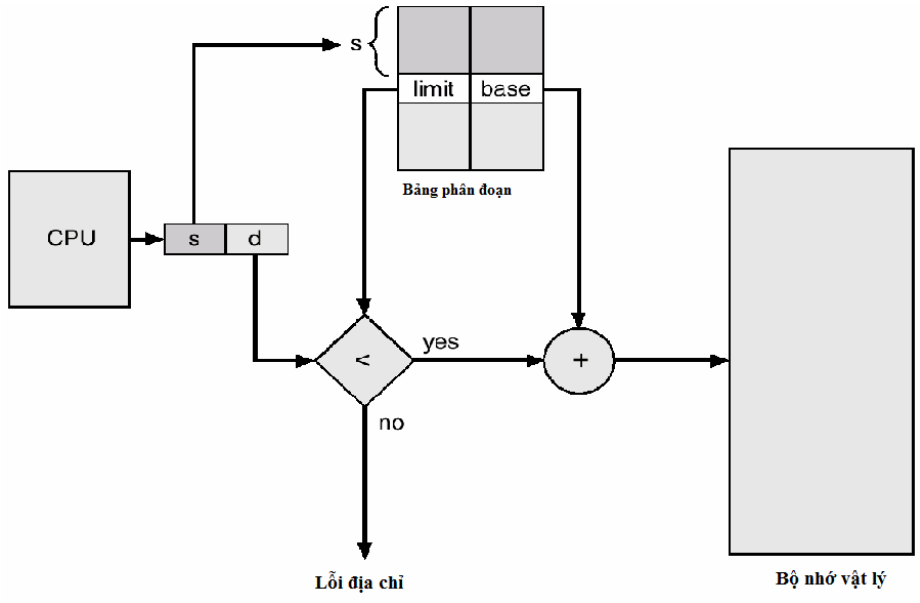
*** Cơ chế MMU trong kỹ thuật phân đoạn:**

Khi chương trình được nạp vào bộ nhớ, MMU ghi các vị trí lưu trữ và kích thước các phân đoạn vào bảng phân đoạn còn CPU làm nhiệm vụ chuyển đổi tất cả các địa chỉ tương đối trong chương trình thành địa chỉ ảo.

Phần tử thứ s trong bảng phân đoạn gồm hai phần (base, limit), base là địa chỉ vật lý bắt đầu phân đoạn s, limit là chiều dài của phân đoạn s. Mỗi địa chỉ ảo gồm hai phần (s,d) với s là số hiệu phân đoạn , d là địa chỉ tương đối trong phân đoạn s.

Để chuyển địa chỉ ảo (s,d) thành địa chỉ vật lý, MMU truy xuất phần tử thứ s trong bảng phân đoạn, lấy được giá trị limit và base của phân đoạn s, sau đó kiểm tra điều kiện (d<limit), nếu sai thì thông báo lỗi " truy xuất địa chỉ không hợp lệ", nếu đúng thì tính địa chỉ vật lý theo công thức: đcvl =base + d.

Theo ví dụ trên, giả sử tiến trình truy xuất địa chỉ ảo (s,d)=(4,1500) thì MMU sẽ thông báo lỗi!. Nếu tiến trình truy xuất địa chỉ ảo (4,100) thì MMU sẽ chuyển thành địa chỉ vật lý là 4700+100=4800.

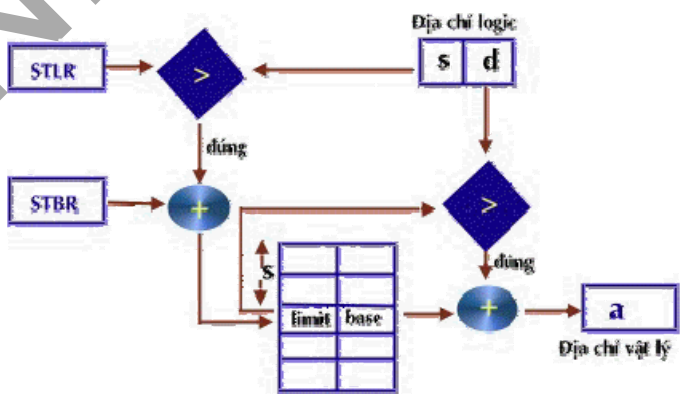


Hình 4.10: cơ chế MMU trong mô hình phân đoạn

*** Cài đặt bảng phân đoạn:**

Có thể sử dụng các thanh ghi để lưu trữ bảng phân đoạn nếu có ít phân đoạn. Nếu chương trình có nhiều phân đoạn, bảng phân đoạn phải được lưu trong bộ nhớ chính. Phần cứng cần cung cấp một thanh ghi nền STBR (Segment Table Base Register) để lưu địa chỉ bắt đầu của bảng phân đoạn và một thanh ghi STLR lưu số phân đoạn (Segment Table Limit Register) mà chương trình sử dụng.

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



Hình 4.11: cơ chế MMU trong mô hình phân đoạn. sử dụng thanh ghi STLR và STBR

*** Bảo vệ phân đoạn**

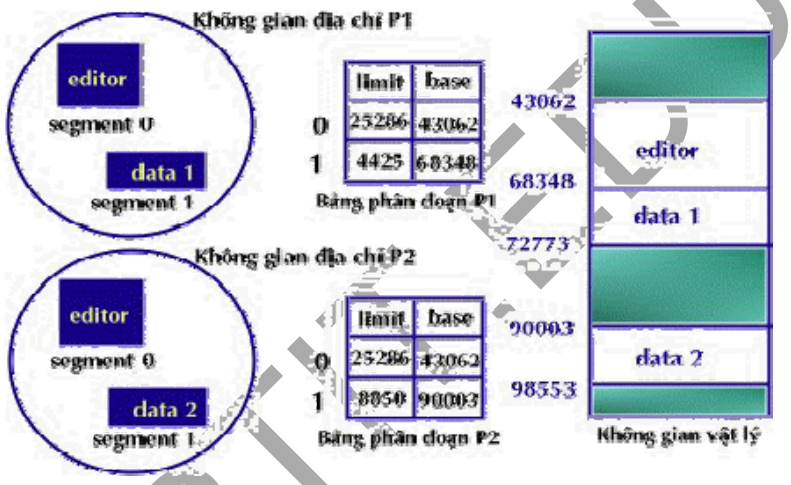
Vì mỗi phân đoạn do người lập trình xác định và người lập trình biết được một phân đoạn chứa những gì bên trong, do vậy họ có thể chỉ định các thuộc tính bảo vệ thích hợp cho mỗi phân đoạn. Khi đó mỗi phần tử của bảng phân đoạn cần có thêm một thành phần gọi là thuộc tính bảo vệ. MMU sẽ kiểm tra giá trị của thuộc tính này để ngăn chặn các thao tác xử lý bất hợp lệ đến phân đoạn. Giá trị của thuộc tính có thể là R (chỉ đọc), X (thực thi), W (ghi),…

Limit	Base	Attribute

Hình 4.12: Cấu trúc một phần tử trong bảng phân đoạn có sử dụng thuộc tính bảo vệ

*** Chia sẻ phân đoạn**

Muốn hai tiến trình dùng chung một phân đoạn nào đó, MMU chỉ cần gán hai phần tử trong hai bảng phân đoạn của hai tiến trình cùng giá trị.



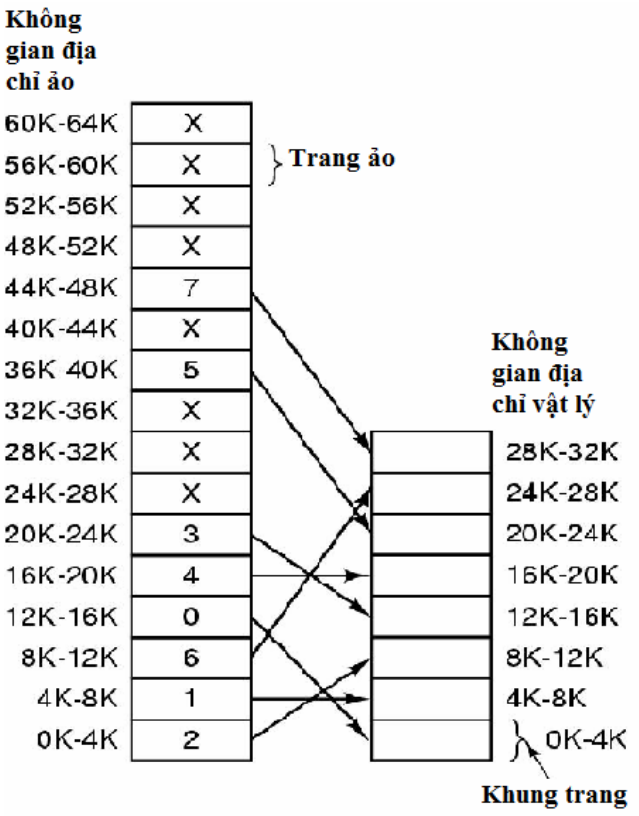
Hình 4.13: hai tiến trình P1,P2 dùng chung phân đoạn 0 (phân đoạn editor)

+ Nhận xét

Trong hệ thống sử dụng kỹ thuật phân đoạn , hiện tượng phân mảnh ngoại vi vẫn xảy ra khi các khối nhớ trống đều quá nhỏ, không đủ để chứa một phân đoạn. Ưu điểm của kỹ thuật phân đoạn là mã chương trình và dữ liệu được tách riêng thành những không gian địa chỉ độc lập nên dễ dàng bảo vệ mã chương trình và dễ dàng dùng chung dữ liệu hoặc hàm.

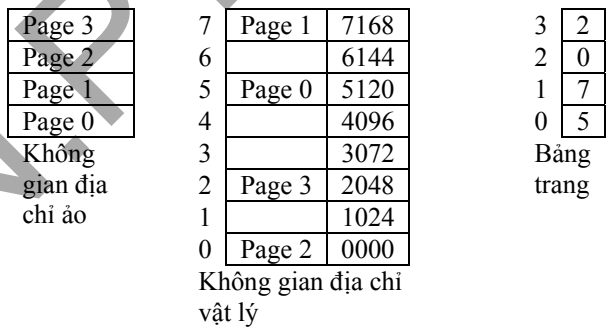
4.2.2.2 Mô hình phân trang (Paging)

Bộ nhớ vật lý được chia thành các khối có kích thước cố định và bằng nhau gọi là khung trang (page frame). Không gian địa chỉ ảo cũng được chia thành các khối có cùng kích thước với khung trang và gọi là trang (page). Khi một tiến trình được đưa vào bộ nhớ để xử lý, các trang của tiến trình sẽ được cất vào những khung trang còn trống, như vậy một tiến trình kích thước N trang sẽ cần N khung trang trống.



Hình 4.14: không gian địa chỉ ảo được chia thành nhiều trang và lưu vào các khung trang

Ví dụ mỗi khung trang 1KB, một tiến trình 3.5KB sẽ được chia làm 4 trang. Gỉa sử trang 0 được cất ở khung trang 5, trang 1 ở khung trang 7,…



Hình 4.15: sử dụng bảng trang để lưu các số hiệu khung trang chứa trang.

*** Cấu trúc địa chỉ ảo:**

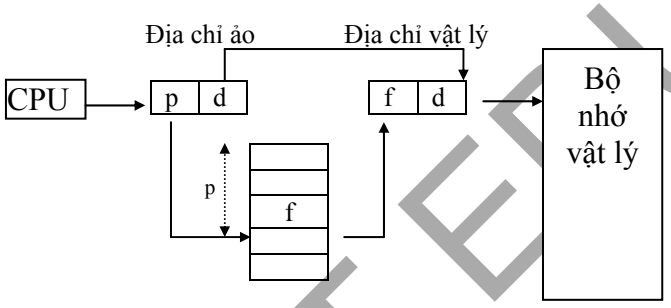
Để dễ dàng phân tích địa chỉ ảo thành số hiệu trang và địa chỉ tương đối, phần cứng qui định kích thước của trang là lũy thừa của 2^n (9<=n<= 13). Nếu kích thước của không gian địa chỉ ảo là 2^m (CPU dùng địa chỉ ảo m bít) và kích thước trang là 2^n thì m-n bit cao của địa chỉ ảo sẽ biểu diễn số hiệu trang, và n bit thấp biểu diễn địa chỉ tương đối trong trang. Khi đó mỗi địa chỉ ảo m bit sẽ có dạng (p,d) với p chiếm m-n bit và p là số hiệu trang, d chiếm n bit và là địa chỉ tương đối trong trang p.

địa chỉ ảo dạng (p,d) có m bit	
p là số hiệu trang và chiếm m-n bit cao	d là địa chỉ tương đối trong trang và chiếm n bít thấp

Hình 4.16: cấu trúc địa chỉ ảo gồm hai phần: các bit cao lưu số hiệu trang, các bit thấp lưu địa chỉ tương đối trong trang.

*** Cơ chế MMU trong mô hình phân trang:**

Khi chương trình được nạp vào bộ nhớ, MMU ghi nhận lại số hiệu khung trang chứa trang vào bảng trang (pages table), còn CPU làm nhiệm vụ chuyển đổi tất cả các địa chỉ tương đối trong chương trình thành địa chỉ ảo. Phần tử thứ p trong bảng trang lưu số hiệu khung trang trong bộ nhớ vật lý đang chứa trang p. Để chuyển địa chỉ ảo (p,d) thành địa chỉ vật lý, MMU truy xuất phần tử thứ p trong bảng trang, lấy được giá trị f là số hiệu khung trang chứa trang p và từ đó tính được địa chỉ vật lý = vị trí bắt đầu của khung trang f + d.



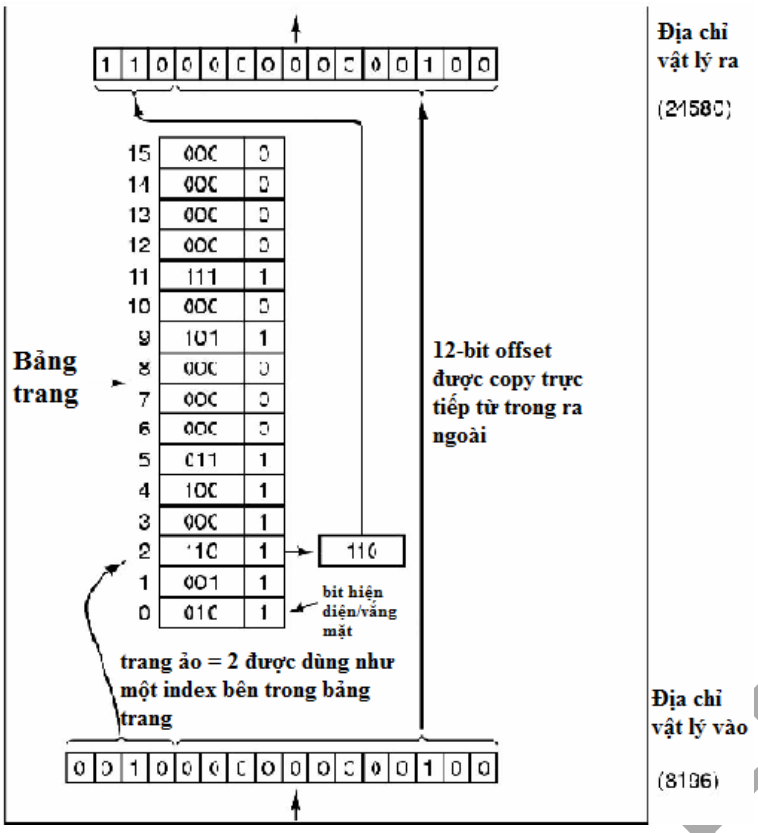
Hình 4.17: cơ chế MMU trong mô hình phân trang

Theo ví dụ trên, giả sử tiến trình truy xuất địa chỉ ảo (p,d) = (3,500), MMU sẽ truy xuất phần tử thứ 3 trong bảng trang và biết được trang 3 ở khung trang 2 và chuyển địa chỉ ảo thành địa chỉ vật lý là $2x\ 2^{10} + 500 = 2548$ ($2x\ 2^{10} = 2048$ là địa chỉ bắt đầu của khung trang 2).

Trong thực tế, việc chuyển đổi địa chỉ ảo (p,d) được MMU thực hiện như sau: MMU truy xuất phần tử thứ p trong bảng trang, lấy được giá trị f là số hiệu khung trang chứa trang p và tính địa chỉ vật lý bằng cách chép d vào n bit thấp của địa chỉ vật lý và chép f vào (m-n) bit cao của địa chỉ vật lý.

Ví dụ: Một hệ thống có địa chỉ ảo 16 bit dạng (p,d) với p có 4 bít, d có 12 bít (hệ thống có 16 trang, mỗi trang 4 KB) . Bít Present/absent =1 nghĩa là trang hiện ở trong bộ nhớ và =0 là ở bộ nhớ phụ.

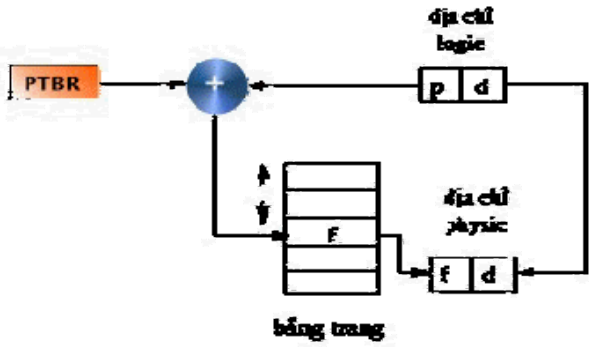
Xét địa chỉ ảo $8196_{10} = 0010.0000.0000.0100_2$ => $p = 0010_2 = 2_{10}$, $d = 0000.0000.0100_2 = 4_{10}$. Do trang p=2 ở khung trang $f=110_2 = 6_{10}$, nên địa chỉ vật lý là $0110.0000.0000.0100_2 = 6x2^{12} + 4 = 24580$



Hình 4.18: cơ chế chuyển đổi địa chỉ của MMU

*** Cài đặt bảng trang**

Nếu bảng trang có kích thước nhỏ có thể dùng một tập các thanh ghi để cài đặt bảng trang. Nếu bảng trang có kích thước lớn, cần phải được lưu trữ trong bộ nhớ chính, và phần cứng cung cấp một thanh ghi PTBR (Page Table Base Register) lưu địa chỉ bắt đầu của bảng trang và thanh ghi PTLR (Page Table Limit Register) lưu số phần tử trong bảng trang.Với một địa chỉ logic (p,d), trước tiên số hiệu trang p được kiểm tra tính hợp lệ (p<PTLR). Kế tiếp, cộng giá trị p với PTBR (PTBR+s) để có được địa chỉ của phần tử thứ p trong bảng phân đoạn và MMU truy xuất phần tử thứ p trong bảng trang, lấy được giá trị f là số hiệu khung trang chứa trang p và từ đó tính được địa chỉ vật lý = vị trí bắt đầu của khung trang f + d.

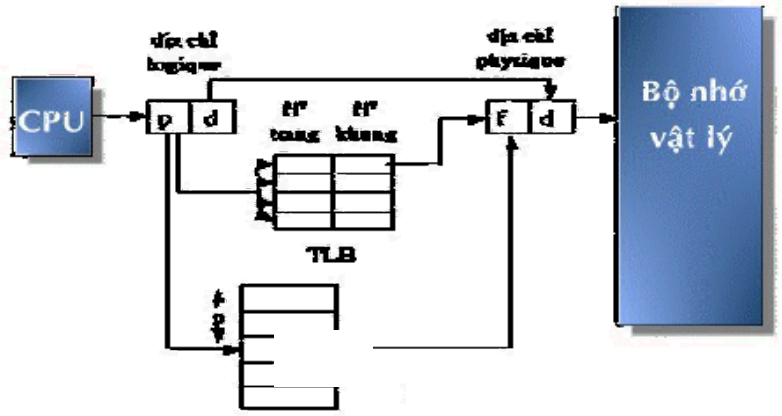


Hình 4.19: cơ chế MMU trong mô hình phân trang, sử dụng hai thanh ghi PTLR và PTBR

*** Bộ nhớ kết hợp (Translation Lookaside Buffers:TLBs)**

Trong kỹ thuật phân trang, mỗi lần truy xuất đến dữ liệu hay chỉ thị đều cần hai lần truy xuất bộ nhớ: một cho truy xuất đến bảng trang để tìm số hiệu khung trang và một cho bản thân dữ liệu. Có thể giảm bớt việc truy xuất bộ nhớ hai lần bằng cách sử dụng thêm bộ nhớ kết hợp (TLBs). Bộ nhớ kết hợp có tốc độ truy xuất rất nhanh và cho phép tìm kiếm song song. Mỗi thanh ghi trong bộ nhớ kết hợp gồm một từ khóa và một giá trị, khi đưa đến bộ nhớ kết hợp một từ khoá cần tìm, từ khoá này sẽ được so sánh cùng lúc với các từ khóa trong bộ nhớ kết hợp để tìm ra giá trị tương ứng.

Trong kỹ thuật phân trang, TLBs được sử dụng để lưu trữ các số hiệu trang được truy cập gần hiện tại nhất. Khi tiến trình truy xuất một địa chỉ ảo, số hiệu trang của địa chỉ sẽ được so sánh với các số hiệu trang trong TLBs, nếu tìm thấy thì sẽ xác định được ngay số hiệu khung trang tương ứng, nếu không có thì mới cần tìm kiếm trong bảng trang.



Hình 4.20: cơ chế MMU trong mô hình phân trang có sử dụng bộ nhớ kết hợp.

Hợp lệ	Trang ảo	Cập nhật	Bảo vệ	Khung trang
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

Hình 4.21: một ví dụ về bảng trang có thuộc tính bảo vệ (protection) và thuộc tính cập nhật (modified): 1 là mời đươc cập nhật, 0 là chưa cập nhật.

Ví dụ một hệ thống máy tính 32 bit, có kích thước 1 khung trang là 4K. Hỏi hệ thống quản lý được tiến trình kích thước tối đa là bao nhiêu?

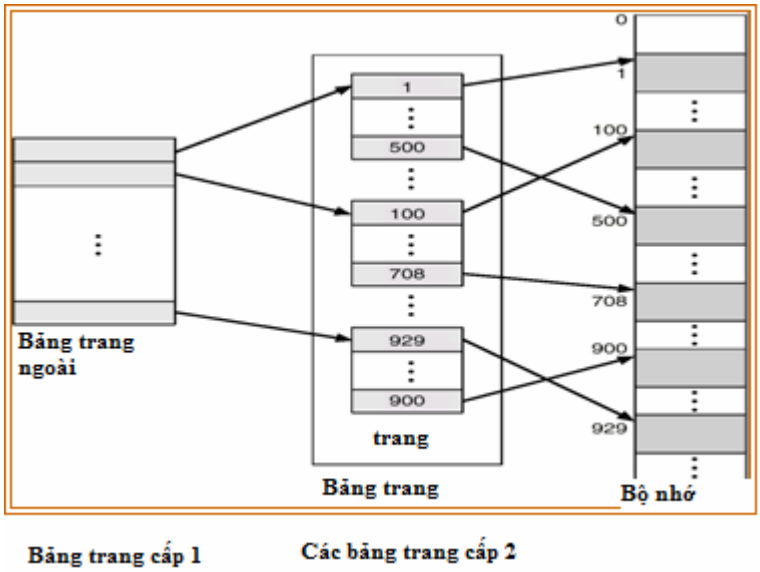
HD: Máy tính 32 bit => địa chỉ ảo (p,d) có 32 bit => số bít của p + số bít của d = 32, mà 1 trang 4K=2^{12} bytes => d có 12 bit =>p có 20 bit => 1 bảng trang có 2^{20} phần tử => hệ thống quản lý được tiến trình có tối đa 2^{20} trang => kích thước tiến trình lớn nhất là 2^{20} x 2^{12} byte = 2^{32} byte =4 GB. Nhận xét: Máy tính n bit quản lý được tiến trình kích thước lớn nhất là 2^n byte.

*** Tổ chức bảng trang**

Thông thường hệ điều hành cấp cho mỗi tiến trình một bảng trang và phải dùng bảng trang kích thước đủ lớn để quản lý tiến trình lớn nhất nên rất tốn bộ nhớ. Có ba giải pháp cho vấn đề này là sử dụng phân trang đa cấp hoặc bảng trang băm hoặc bảng trang nghịch đảo.

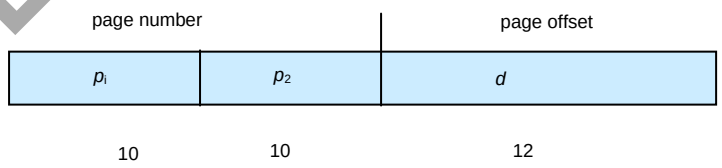
a/ Phân trang đa cấp

Bản thân bảng trang cũng sẽ được phân trang. Xét trường hợp phân trang nhị cấp, khi đó bảng trang cấp 1 lưu số hiệu khung trang chứa bảng trang cấp 2, các bảng trang cấp 2 lưu số hiệu khung trang tiến trình sử dụng. Thông thường trong phân trang đa cấp mỗi bảng trang chiếm 1 khung trang, riêng bảng trang cấp 1 có thể lưu trữ trên đĩa và có thể có kích thước lớn hơn 1 khung trang.



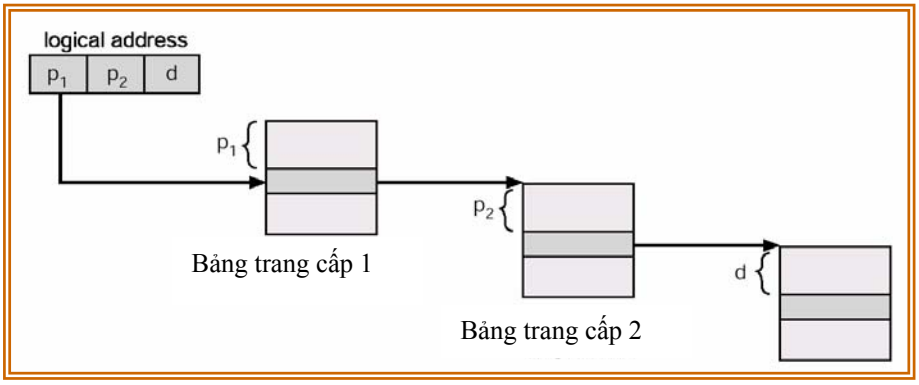
Hình 4.22: một ví dụ về phân trang nhị cấp.

Nếu một máy tính 32 bít, với kích thước trang 4K thì địa chỉ logic có thể biểu diễn như sau: dùng p=20 bít lưu số hiệu trang, d=12 bít lưu vị trí tương đối trong trang. Nếu dùng bảng trang nhị cấp thì p được chia ra thành p1, p2 (p=p1+p2): p1=10 bít lưu chỉ mục của bảng trang cấp 1, p2=10 bít lưu chỉ mục của bảng trang cấp 2 (việc phân chia p1, p2 là bao nhiêu bít thì do phần cứng qui định). Ta có: BTC1[p1] lưu số hiệu khung trang chứa bảng trang cấp 2, BTC2[p2] lưu số hiệu khung trang chứa trang của tiến trình.



p1 chỉ mục của bảng trang cấp một. p2 chỉ mục của bảng trang cấp 2

Hình 4.23: cấu trúc của một địa chỉ ảo trong phân trang nhị cấp

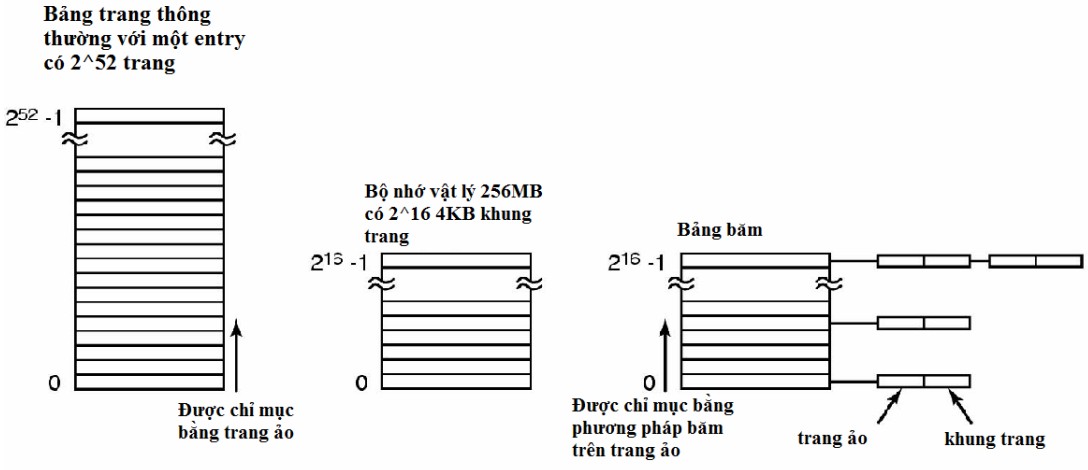


Hình 4.24: cơ chế chuyển đổi địa chỉ trong bảng trang nhị cấp.

b/ Bảng trang băm

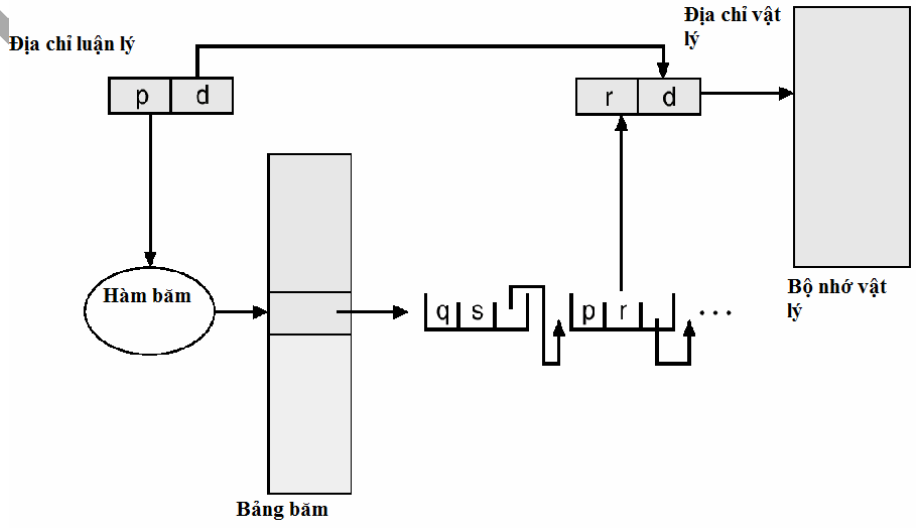
Khi không gian địa chỉ ảo lớn (> 32 bít) thường hệ điều hành dùng bảng băm để lưu trữ bảng trang. Gỉa sử trang p, lưu ở khung trang r, thì thông tin này được lưu trữ như sau: p được băm và lưu trữ trong một danh sách xung đột tương ứng của bảng băm.

Ví dụ: Một máy tính 64 bít, có RAM 256MB, kích thước 1 khung trang là 4KB. Bảng trang thông thường phải có 2^{52} mục, nếu dùng bảng trang băm có thể sử dụng bảng có số mục bằng số khung trang vật lý là 2^{16} ($<<2^{32}$) với hàm băm là hasfunc(p)=p mod 2^{16}.



Hình 4.25: sử dụng bảng băm để lưu trữ bảng trang

Khi tìm khung trang chứa trang p, hệ điều hành dùng hàm băm tìm ra danh sách chứa trang p, sau đó tìm tuyến tính trên danh sách này để tìm ra khung trang chứa trang p.



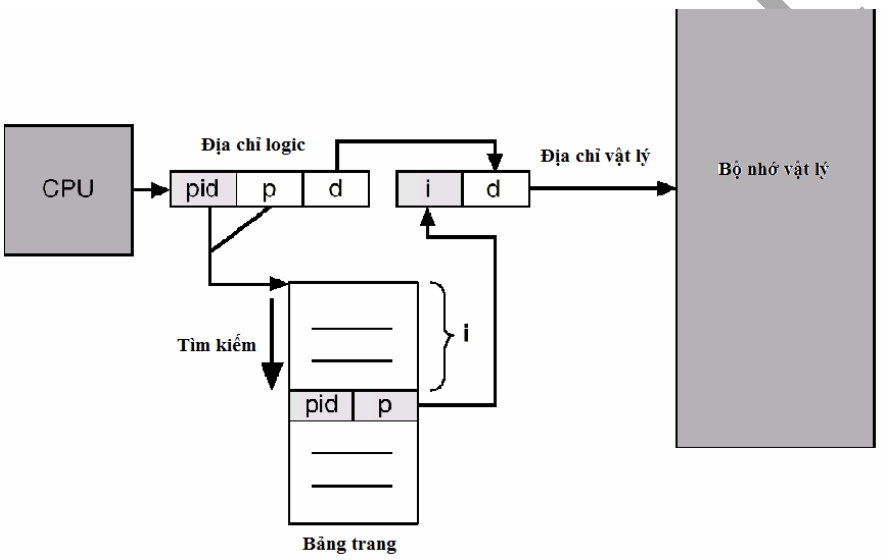
Hình 4.26: cơ chế chuyển đổi địa chỉ khi sử dụng bảng trang băm

c/ Bảng trang nghịch đảo

Hệ điều hành có thể dùng một bảng trang duy nhất để quản lý bộ nhớ của tất cả các tiến trình và gọi là bảng trang nghịch đảo. Mỗi phần tử của bảng trang nghịch đảo là cặp (pid, p), pid là mã số của tiến trình, p là số hiệu trang và mỗi địa chỉ ảo là một bộ ba (pid, p, d).

Khi một truy xuất bộ nhớ được phát sinh, một phần địa chỉ ảo là (pid, p) được đưa đến cho trình quản lý bộ nhớ để tìm phần tử tương ứng trong bảng trang nghịch đảo, nếu tìm thấy tại phần tử thứ i, thì i chính là số hiệu khung trang chứa trang p và địa chỉ vật lý tương ứng là (i,d). Trong các trường hợp khác, xem như đã truy xuất một địa chỉ bất hợp lệ.

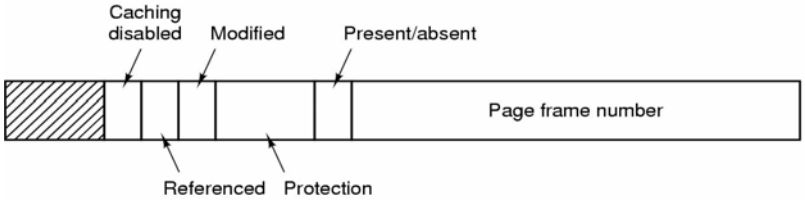
Nhận xét: số phần tử trong bảng trang nghịch đảo bằng với số khung trang vật lý



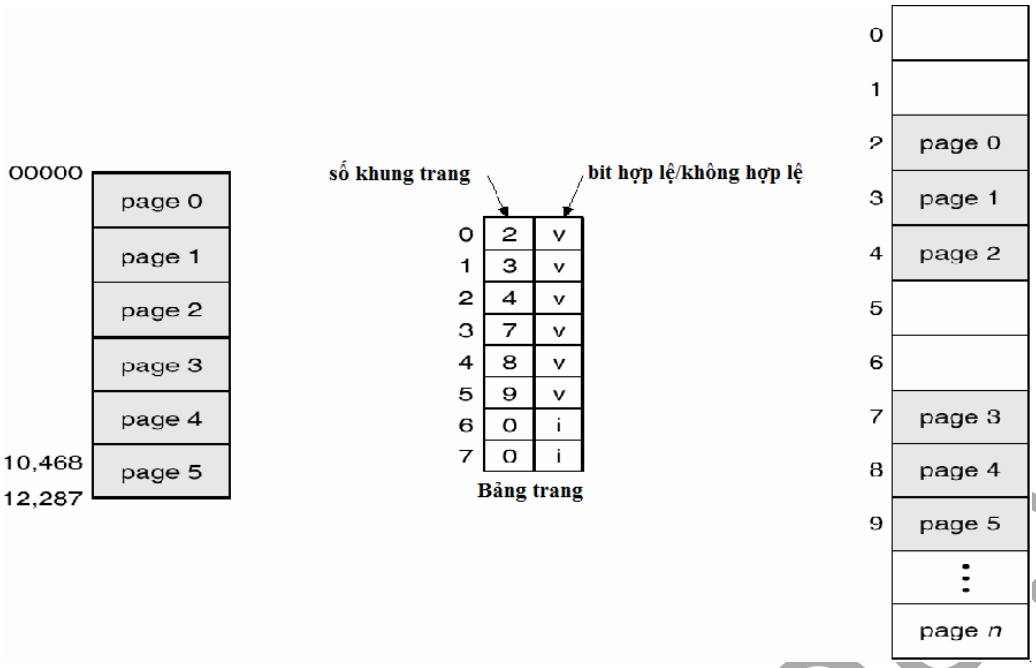
Hình 4.27: cơ chế chuyển đổi địa chỉ khi sử dụng bảng trang nghịch đảo

*** Bảo vệ trang**

Cơ chế bảo vệ trong hệ thống phân trang được thực hiện với các bit bảo vệ (protection) được lưu trong mỗi phần tử của bảng trang, vì mỗi truy xuất đến bộ nhớ đều phải tham khảo đến bảng trang để phát sinh địa chỉ vật lý, khi đó, hệ thống có thể kiểm tra các thao tác truy xuất trên khung trang tương ứng có hợp lệ với thuộc tính bảo vệ của nó không. Ngoài ra, có thể thêm một số bít khác với các mục đích khác nhau.



Hình 4.28: cấu trúc tổng quát của một phần tử trong bảng trang

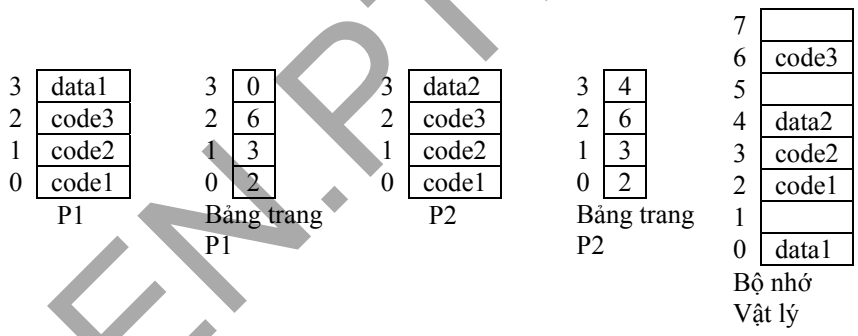


Hình 4.29: một ví dụ về mô hình phân trang

*** Chia xẻ bộ nhớ:**

Trong kỹ thuật phân trang, hệ điều hành cũng có thể cho phép các tiến trình dùng chung một số khung trang, bằng cách ghi cùng số hiệu khung trang vào bảng trang của mỗi tiến trình.

ví dụ: Hai tiến trình P1, P2 sinh ra từ chương trình word.exe có thể dùng chung đoạn code (gồm 3 trang) nhưng mỗi tiến trình có đoạn data riêng.



Hình 4.30: hai tiến trình P1,P2 dùng chung ba trang 0,1,2

Nhận xét:

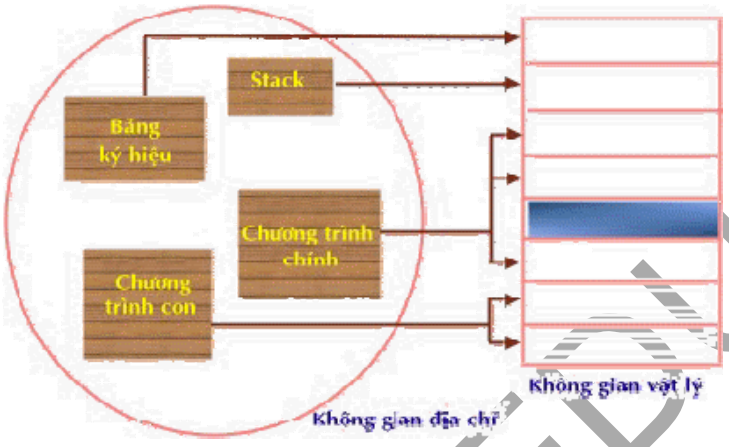
+ Kỹ thuật phân trang loại bỏ được hiện tượng phân mảnh ngoại vi vì mỗi khung trang đều có thể được cấp phát cho một tiến trình nào đó có yêu cầu. Tuy nhiên hiện tượng phân mảnh nội vi vẫn có thể xảy ra khi kích thước của tiến trình không đúng bằng bội số của kích thước một trang, khi đó trang cuối cùng sẽ không được sử dụng hết.

+ Sự phân trang không phản ánh đúng cách thức người sử dụng cảm nhận về bộ nhớ. Kỹ thuật phân đoạn thỏa mãn được nhu cầu thể hiện cấu trúc logic của chương trình nhưng nó dẫn đến tình huống phải cấp phát các khối nhớ có kích thước khác nhau cho các phân đoạn. Điều này làm rắc rối vấn đề hơn rất nhiều so với việc cấp phát các trang có kích thước cố định và bằng nhau. Một

giải pháp dung hoà là kết hợp cả hai kỹ thuật phân trang và phân đoạn: chúng ta tiến hành phân trang các phân đoạn.

4.2.2.3 Mô hình phân đoạn kết hợp phân trang (Paged segmentation)

Một tiến trình gồm nhiều phân đoạn, mỗi phân đoạn được chia thành nhiều trang, lưu trữ vào các khung trang có thể không liên tục.

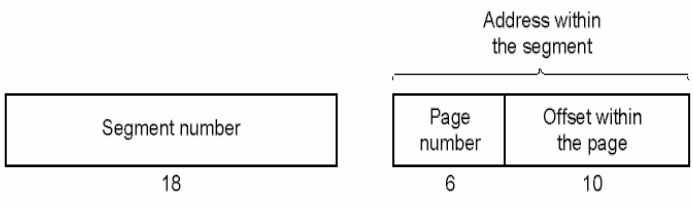


Hình 4.31: mô hình phân đoạn kết hợp phân trang

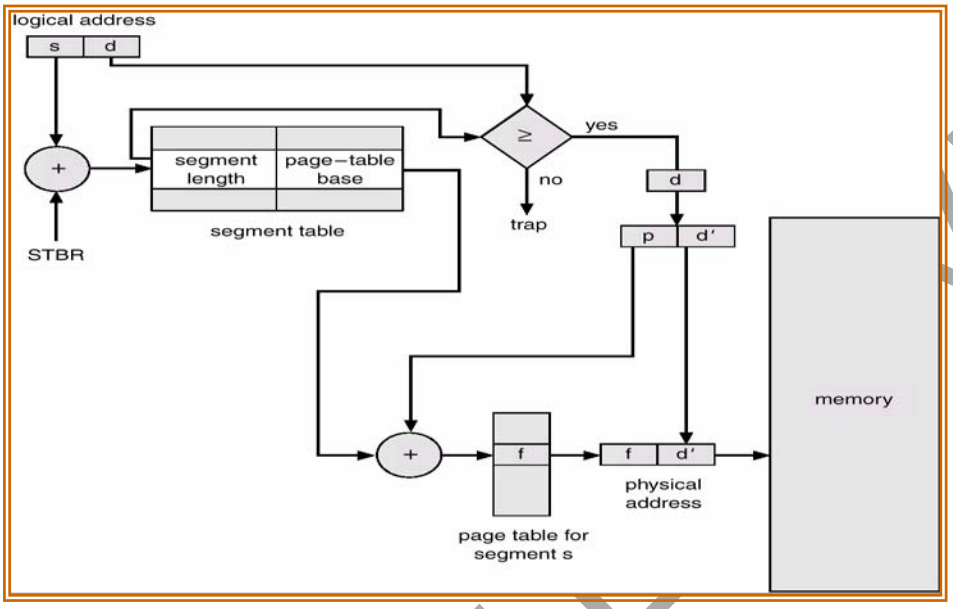
*** Cơ chế MMU trong mô hình phân đoạn kết hợp phân trang**

Mỗi địa chỉ logic là một bộ (s,d) với s là số hiệu phân đoạn, d là địa chỉ tương đối trong phân đoạn. Tách d thành p và d' (số bít của d = số bít của p + số bít của d') với p là chỉ số trang, d' là địa chỉ tương đối trong trang (số bít của d' do phần cứng qui định).

Để chuyển các địa chỉ ảo 2 chiều thành địa chỉ vật lý một chiều, MMU dùng một bảng phân đoạn, mỗi phân đoạn cần có một bảng phân trang tương ứng. Mỗi phần tử trong bảng phân đoạn gồm hai phần (base,limit), base lưu địa chỉ vật lý nơi bắt đầu của bảng trang của phân đoạn này, limit lưu chiều dài của phân đoạn. Hệ thống cần cung cấp một thanh ghi STBR lưu vị trí bắt đầu của bảng phân đoạn, khi tiến trình truy xuất một địa chỉ logic (s,d)=(s,p,d'), MMU lấy STBR cộng với s để truy xuất phần tử thứ s trong bảng phân đoạn. Phần tử thứ s của bảng phân đoạn lưu hai gía trị (segment length, page-table base): segment length là kích thước phân đoạn, page-table base là vị trí lưu trữ bảng trang tương ứng với phân đoạn s. Nếu segment length <d thì thông báo "truy xuất địa chỉ không hợp lệ" ngược lại phân tích d thành p và d' và cộng page-table base với p để truy xuất phần tử thứ p trong bảng phân trang lấy được giá trị f là số hiệu khung trang chứa trang p. Sau đó cộng f với d' sẽ cho địa chỉ vật lý tương ứng.



Hình 4.32: cấu trúc một phần tử của bảng trang trong mô hình phân đoạn kết hợp phân trang.



Hình 4.33: cơ chế chuyển đổi địa chỉ trong mô hình phân đoạn kết hợp phân trang

Comparison field					Is this entry used?
Segment number	Virtual page	Page frame	Protection	Age	
4	1	7	Read/write	13	1
6	0	2	Read only	10	1
12	3	1	Read/write	2	1
					0
2	1	0	Execute only	7	1
2	2	12	Execute only	9	1

Hình 4.34: hệ điều hành MULTICS dùng phân đoạn kết hợp phân trang và bộ nhớ kết hợp

Nhận xét:

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

+ Kích thước tiến trình bị giới hạn bởi kích thước của bộ nhớ vật lý.

+ Khó nâng cao mức độ đa chương của hệ thống.

4.3 BỘ NHỚ ẢO

Bộ nhớ ảo là kỹ thuật dùng bộ nhớ phụ lưu trữ tiến trình, các phần của tiến trình được chuyển vào-ra giữa bộ nhớ chính và bộ nhớ phụ để cho phép thực thi một tiến trình mà không cần nạp

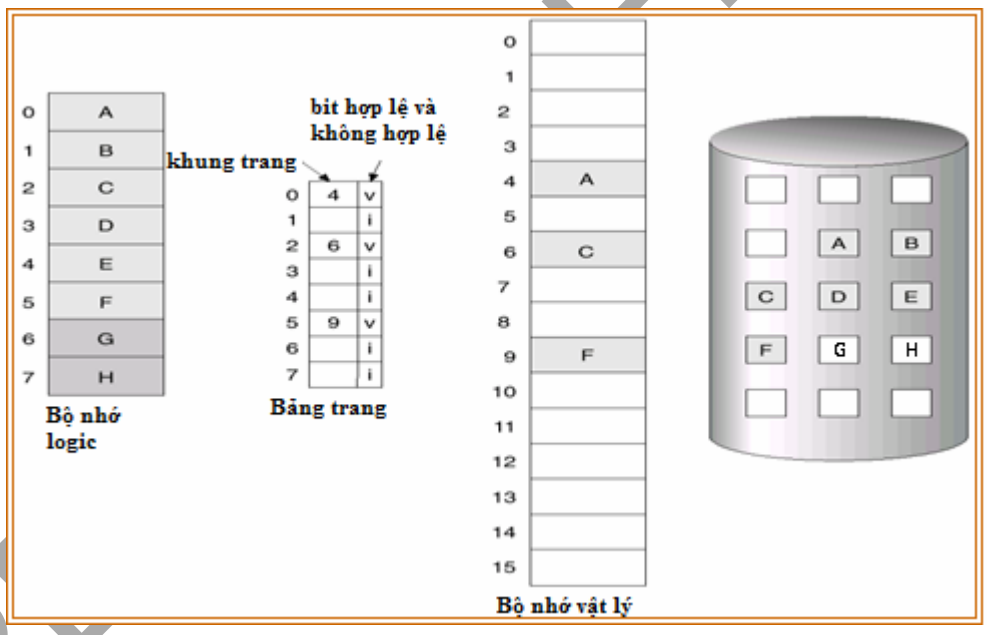
toàn bộ vào bộ nhớ vật lý. Với kỹ thuật bộ nhớ ảo, hệ điều hành sẽ tăng được mức độ đa chương của hệ thống, có thể thực thi được những chương trình kích thước rất lớn so với kích thước bộ nhớ vật lý và người lập trình không cần quan tâm máy tính có đủ RAM để thực thi chương trình hay không. Có hai phương pháp cài đặt kỹ thuật bộ nhớ ảo đó là phân trang theo yêu cầu (Demand paging) hoặc phân đoạn theo yêu cầu (Demand segmentation)

4.3.1 Phân trang theo yêu cầu (Demand paging)

Một tiến trình được chia thành nhiều trang, thường trú trên bộ nhớ phụ (thường là đĩa cứng) và một trang chỉ được nạp vào bộ nhớ chính khi có yêu cầu. Vùng không gian đĩa dùng để lưu trữ tạm các trang gọi là không gian swapping.

4.3.1.1 Cấu trúc một phần tử trong bảng trang

Mỗi phần tử trong bảng trang sẽ gồm hai trường: Một trường chứa bit "kiểm tra" có giá trị 1 (valid) là trang đang ở trong bộ nhớ chính , 0 (invalid) là trang đang được lưu trên bộ nhớ phụ hoặc trang không thuộc tiến trình, khởi đầu tất cả bit kiểm tra trong bảng trang đều bằng 0. Một trường chứa số hiệu khung trang (nếu bit kiểm tra là valid) hoặc chứa địa chỉ của trang trên bộ nhớ phụ (nếu bit kiểm tra là invalid).



Hình 4.35: mô hình phân trang theo yêu cầu

4.3.1.2 Chuyển địa chỉ ảo (p,d) thành địa chỉ vật lý

+ ***Bước 1***: MMU truy xuất phần tử thứ p trong bảng trang để lấy các thông tin cần thiết cho việc chuyển đổi địa chỉ.

+ ***Bước 2***: Nếu phần tử thứ p có bit kiểm tra bằng 1 (valid), thì trang đang yêu cầu truy xuất hợp lệ, tức là có sẵn trong bộ nhớ, khi đó việc chuyển đổi địa chỉ ảo thành địa chỉ vật lý xảy ra bình thường, địa chỉ vật lý = số hiệu khung trang * kích thước của một khung + d. Nếu phần tử thứ p có bit kiểm tra bằng 0 (invalid), thì MMU sẽ phát sinh một ngắt để báo cho hệ điều hành có "lỗi

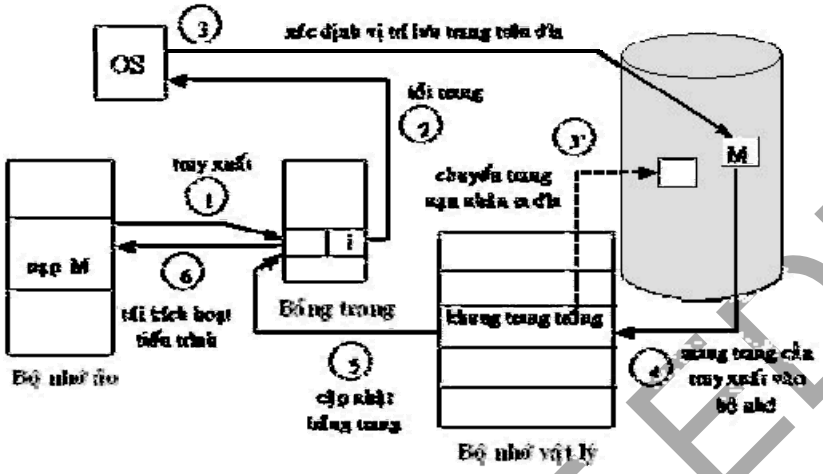
trang". Khi đó hệ điều hành sẽ kiểm tra trang truy xuất có thuộc tiến trình không, nếu trang không thuộc tiến trình thì hệ điều hành kết thúc tiến trình, ngược lại chuyển đến bước 3

+ ***Bước 3***: Tìm vị trí trên đĩa chứa trang muốn truy xuất và tìm một khung trang trống trong bộ nhớ chính: nếu có khung trang trống thì đến bước 4, nếu không còn khung trang trống, chọn một khung trang "nạn nhân" (victim) và chuyển trang "nạn nhân " ra bộ nhớ phụ , rồi đến bước 4

+ ***Bước 4***: Chuyển trang muốn truy xuất từ bộ nhớ phụ vào khung trang trống đã chọn.

+ ***Bước 5***: Cập nhật nội dung bảng trang.

+ ***Bước 6***: Tái kích hoạt tiến trình người sử dụng.



Hình 4.36: cơ chế chuyển đổi địa chỉ trong mô hình phân trang theo yêu cầu

4.3.1.3 Thay thế trang

Nếu không có khung trang trống, thì mỗi khi xảy ra lỗi trang cần phải thực hiện hai thao tác chuyển trang : chuyển một trang ra bộ nhớ phụ và nạp một trang khác vào bộ nhớ chính. Có thể giảm bớt số lần chuyển trang bằng cách sử dụng thêm một bit "cập nhật" (dirty bit). Giá trị của bit được phần cứng đặt là 1 nếu nội dung trang có bị sửa đổi. Khi cần thay thế một trang, nếu bit cập nhật có giá trị là 1 thì trang này cần được lưu lại trên đĩa, ngược lại, nếu bit cập nhật là 0, nghĩa là trang không bị thay đổi, thì không cần lưu trữ trang trở lại đĩa.

số hiệu khung trang chứa trang hoặc địa chỉ trên đĩa của trang	bit nhận diện trang có trong bộ nhớ (bit valid-invalid)	bit nhận diện trang có thay đổi (bit dirty)

Hình 4.37: cấu trúc một phần tử của bảng trang trong kỹ thuật phân trang theo yêu cầu.

4.3.1.4 Thời gian thực hiện một yêu cầu truy xuất bộ nhớ

Gọi xác suất xảy ra lỗi trang là p: $0 \le p \le 1.0$, nếu p = 0 nghĩa là không có lỗi trang , nếu p = 1 nghĩa là mỗi lần truy xuất đều xảy ra lỗi. Memory access (ma) là thời gian một lần truy xuất bộ nhớ. Effective Access Time (EAT) là thời gian thực hiện một yêu cầu truy xuất bộ nhớ. Page fault overhead (pfo) là thời gian xử lý một lỗi trang. Swap page in (spi) là thời gian chuyển trang từ đĩa vào bộ nhớ. Swap page out (spo) là thời gian chuyển trang ra đĩa (swap page out có thể bằng 0). Restart overhead (ro) là thời gian tái khởi động lại việc truy xuất bộ nhớ.

Ta có:

EAT = (1 – p) x ma+ p (pfo+ [spo]+ spi+ ro)

Ví dụ:

Giả sử thời gian một lần truy xuất bộ nhớ là 1 micro second và giả sử 40% trang được chọn đã thay đổi nội dung và thời gian hoán chuyển trang ra/vào là 10 mili second . Tính ETA. Ta có:

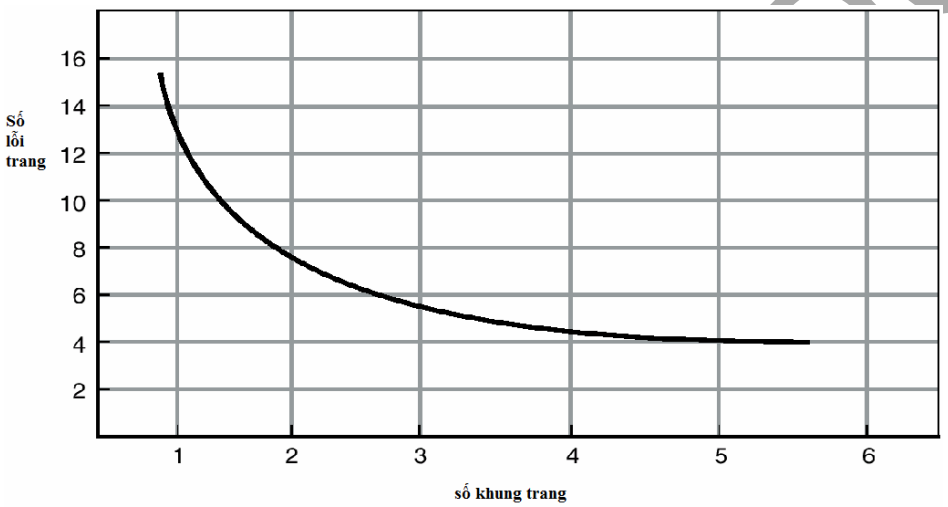
ma=1 micro second

spo= spin = 10 milisecond = 10000 micro second

=> EAT = (1 – p) + p (pfo+10000*0.4+10000+ro) micro second

4.3.1.5 Các thuật toán chọn trang nạn nhân

Mục tiêu của các thuật tóan là chọn trang «nạn nhân» là trang mà sau khi thay thế sẽ gây ra ít lỗi trang nhất. Thông thường số lỗi trang tỉ lệ nghịch với số khung trang dành cho tiến trình (số khung trang tăng thì số lỗi trang giảm).



Hình 4.38: biểu đồ minh họa số lỗi trang sẽ giảm khi số khung trang dành cho tiến trình tăng

Có thể đánh giá thuật toán bằng cách xét một chuỗi các trang mà tiến trình sẽ lần lượt truy xuất với số khung trang cấp cho tiến trình đã biết và tính số lỗi trang phát sinh. Ví dụ tiến trình truy xuất các địa chỉ theo thứ tự : 0100, 0432, 0101, 0611. Nếu kích thước của một trang là 100 bytes thì có thể viết lại chuỗi địa chỉ thành chuỗi trang mà tiến trình truy xuất là 1, 4, 1, 6.

4.3.1.5.1 Thuật toán FIFO (First In First Out)

Trang ở trong bộ nhớ lâu nhất sẽ được chọn làm trang nạn nhân (vào trước ra trước)

Ví dụ: Một tiến trình được cấp 3 khung trang, ban đầu cả 3 khung đều trống, tiến trình lần lượt truy xuất tới các trang theo thứ tự sau: 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1. Tính số lỗi trang khi áp dụng thuật toán FIFO để chọn trang nạn nhân.

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

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

Kí hiệu * là có lỗi trang và có 15 lỗi trang

Nhận xét:

Không cần ghi nhận thời điểm trang được nạp vào bộ nhớ, mà chỉ cần quản lý các trang trong bộ nhớ bằng một danh sách FIFO, khi đó nếu có lỗi trang thì trang truy xuất được đưa vào cuối danh sách và nếu hết khung trang thì trang đầu danh sách sẽ được chọn làm trang nạn nhân.

Thuật toán FIFO đơn giản, dễ cài đặt, nhưng nếu trang được chọn là trang thường xuyên được sử dụng, thì khi bị chuyển ra bộ nhớ phụ sẽ nhanh chóng gây ra lỗi trang.

Số lượng lỗi trang có thể tăng lên khi số lượng khung trang sử dụng tăng, hiện tượng này gọi là nghịch lý Belady.

Ví dụ: Xét tiến trình truy xuất chuỗi trang theo thứ tự sau: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

+ Nếu sử dụng 3 khung trang, sẽ có 9 lỗi trang

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

+ Nếu sử dụng 4 khung trang, sẽ có 10 lỗi trang

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

4.3.1.5.2 Thuật toán tối ưu (Optimal Page Replacement Algorithm)

Chọn trang lâu được sử dụng nhất trong tương lai. Ví dụ:

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

Nhận xét:

Có 9 lỗi trang, FIFO 15 lỗi => tốt hơn FIFO nhiều.

Thuật toán này bảo đảm số lượng lỗi trang phát sinh là thấp nhất, nó cũng không bị nghịch lý Belady, tuy nhiên đây là một thuật toán khó cài đặt vì thường không thể biết trước chuỗi truy xuất của tiến trình.

Đối với các hệ điều hành cho thiết bị gia dụng, thường chỉ có một số tiến trình cố định thực thi nên có thể cho tiến trình chạy trước một lần, ghi nhận lại chuỗi trang truy xuất, các lần thực thi sau đó có thể sử dụng thuật toán tối ưu để chọn trang nạn nhân.

4.3.1.5.3 Thuật toán LRU (Least-recently-used)

Thuật toán FIFO sử dụng thời điểm nạp trang để chọn trang thay thế, thuật toán tối ưu dùng thời điểm trang sẽ được sử dụng gần nhất trong tương lai. Vì thời điểm này thường khó xác định trước nên thuật toán LRU sẽ dùng thời điểm cuối cùng trang được truy xuất (dùng quá khứ gần để dự đoán tương lai gần). Với mỗi trang, ghi nhận thời điểm cuối cùng trang được truy cập, trang được chọn để thay thế sẽ là trang lâu nhất chưa được truy xuất vì với suy nghĩ là trang này có khả năng ít được sử dụng nhất.

Ví dụ:

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

Nhận xét:

Có 12 lỗi trang, FIFO 15 lỗi =>LRU tốt hơn FIFO.

OPT và LRU có số lỗi trang không đổi khi nghịch đảo chuỗi địa chỉ truy xuất.

*** Cài đặt thuật toán LRU: có thể dùng hai kỹ thuật sau**

+ Sử dụng bộ đếm:

Thêm vào cấu trúc của mỗi phần tử trong bảng trang một trường ghi nhận "thời điểm truy xuất gần nhất", và thêm vào cấu trúc của CPU một thanh ghi đếm (counter). Mỗi lần thực hiện truy xuất đến một trang, giá trị của counter tăng lên 1 và ghi giá trị counter vào trường "thời điểm truy xuất gần nhất" của phần tử tương ứng với trang trong bảng trang. Khi đó trang "nạn nhân" là trang có giá trị trường "thời điểm truy xuất gần nhất" là nhỏ nhất.

số hiệu khung trang chứa trang hoặc địa chỉ trang trên đĩa	bit valid - invalid	bit dirty	thời điểm truy xuất gần nhất

+ ***Sử dụng danh sách liên kết :***

Dùng một một dslk lưu trữ các số hiệu trang, trang ở cuối danh sách là trang được truy xuất gần nhất, và trang ở đầu danh sách là trang lâu nhất chưa được sử dụng. Nếu có lỗi trang và nếu có khung trang trống thì thêm nút chứa số hiệu trang đang truy xuất vào cuối danh sách, nếu không có khung trống thì trang được chọn làm trang nạn nhân sẽ là trang ở đầu danh sách, khi đó hủy nút đầu và thêm nút chứa số hiệu trang đang truy xuất vào cuối danh sách. Nếu không có lỗi trang thì chuyển nút chứa số hiệu trang hiện hành xuống cuối danh sách.

4.3.1.5.4 Các thuật toán xấp xỉ LRU

Có ít hệ thống được cung cấp đủ các phần cứng hỗ trợ để cài đặt thuật toán LRU thật sự. Tuy nhiên, nhiều hệ thống được trang bị thêm một bit tham khảo (reference). Mỗi phần tử trong bảng trang có thêm bit reference được khởi gán là 0 bởi hđh và được phần cứng gán là 1 mỗi lần trang tương ứng được truy cập. Sau mỗi chu kỳ qui định trước, phần cứng kiểm tra giá trị của các bit reference để xác định được trang nào đã được truy xuất đến và trang nào không, sau khi đã kiểm tra xong, các bit reference được phần cứng gán trở về 0. Với bit reference, có thể biết được trang nào đã được truy xuất, nhưng không biết được thứ tự truy xuất của các trang. Thông tin không đầy đủ này dẫn đến nhiều thuật toán xấp xỉ LRU khác nhau.

số hiệu khung trang chứa trang hoặc địa chỉ trang trên đĩa	bit valid-invalid	bit dirty	bit reference

Hình 4.39: cấu trúc một phần tử của bảng trang trong thuật toán xấp xỉ LRU

a/ Thuật toán với các bít history

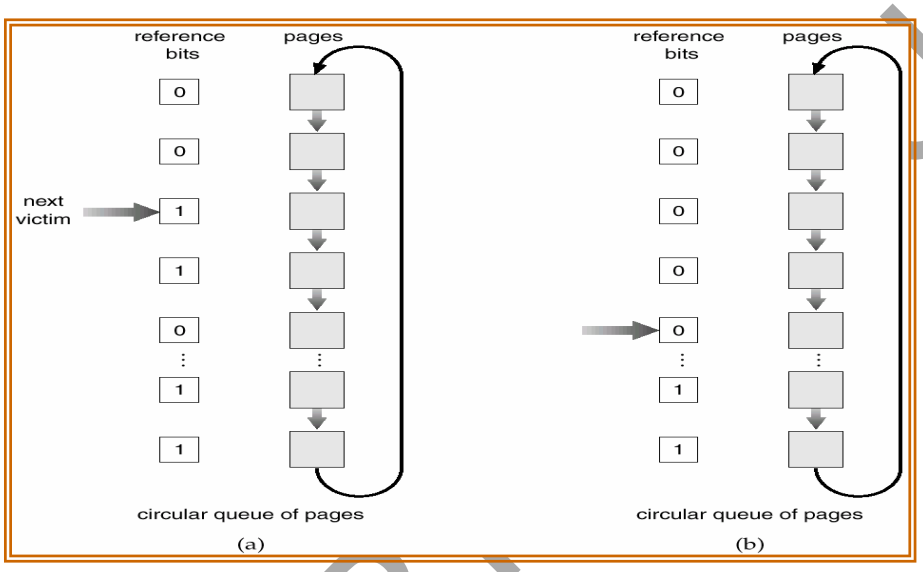
Mỗi trang sử dụng thêm 8 bit lịch sử (history). Sau từng khoảng thời gian nhất định (thường là 100 milliseconds), một ngắt đồng hồ được phát sinh và quyền điều khiển được chuyển cho hệ điều hành. Hệ điều hành sẽ cập nhật các bít history của mỗi trang bằng cách dịch các bit history sang phải 1 vị trí để loại bỏ bit thấp nhất và đặt bit reference của mỗi trang vào bit cao nhất trong 8 bit history của trang đó. 8 bit history sẽ lưu trữ tình hình truy xuất đến trang trong 8 chu kỳ cuối cùng.

Nếu 8 bit history là 00000000 thì trang tương ứng có khả năng không được dùng trong 8 chu kỳ cuối, nếu 8 bit history là 11111111 thì trang tương ứng được dùng đến ít nhất 1 lần trong mỗi 8 chu kỳ cuối. Nếu xét 8 bit history như một số nguyên không dấu thì trang "nạn nhân" là trang có

giá trị history nhỏ nhất. Số lượng các bit history có thể thay đổi tùy theo phần cứng, số bít history nhiều thì việc chọn trang "nạn nhân" sẽ chính xác hơn.

b/ Thuật toán cơ hội thứ hai

Tìm một trang theo nguyên tắc FIFO, rồi kiểm tra bit reference của trang đó. Nếu bit reference là 0, chọn trang này, nếu bit reference là 1 thì gán lại là 0 rồi tìm trang FIFO tiếp theo (cho trang này một cơ hội thứ hai). Một trang đã được cho cơ hội thứ hai sẽ không bị thay thế cho tới khi tất cả những trang khác được thay thế hoặc được cho cơ hội thứ hai. Nếu trang thường xuyên được sử dụng, bit reference của nó sẽ duy trì được giá trị 1 và trang hầu như không bao giờ bị thay thế. Nếu tất cả các bít reference là 1 thì thuật toán trở thành FIFO. Thuật toán có thể cài đặt bằng dslk vòng.



Hình 4.40: cơ chế chọn trang nạn nhân trong thuật toán chọn trang nạn nhân thứ hai.

Ví dụ:

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

Xem danh sách liên kết vòng của ví dụ trên

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
7(1)	7(1)	7(1)	0(0)	1(0)	2(1)	2(1)	3(0)	0(0)	4(1)	2(0)	2(0)	0(1)	3(0)	3(0)	1(0)	2(1)	0(0)	1(0)	7(1)

	0(1)	0(1)	1(0)	2(1)	0(1)	3(1)	0(0)	4(1)	2(1)	3(0)	0(1)	3(1)	2(0)	1(0)	2(1)	0(1)	1(0)	7(1)	0(1)
		1(1)	2(1)	0(1)	3(1)	0(1)	4(1)	2(1)	3(1)	0(1)	3(1)	2(1)	1(0)	2(1)	0(1)	1(1)	7(1)	0(1)	1(1)

c/ Thuật toán cơ hội thứ hai nâng cao (Not Recently Used Page Replacement Algorithm: NRU)

Xem các bit reference và dirty bit như một cặp có thứ tự và tạo thành 4 lớp sau :

- Lớp 1 (0,0): gồm những trang có (ref,dirty)=(0,0). Những trang thuộc lớp này không được truy xuất gần đây và không bị sửa đổi, đây là những trang tốt nhất để thay thế.

- Lớp 2 (0,1): trang không truy xuất gần đây nhưng đã bị sửa đổi. Trường hợp này không thật tốt, vì trang cần được lưu trữ lại trước khi thay thế.

- Lớp 3 (1,0): trang được truy xuất gần đây, nhưng không bị sửa đổi. Trang có thể nhanh chóng được tiếp tục được sử dụng.

- Lớp 4 (1,1): trang được truy xuất gần đây, và bị sửa đổi. Trang có thể nhanh chóng được tiếp tục được sử dụng và trước khi thay thế cần phải được lưu trữ lại.

Lớp 1 có độ ưu tiên thấp nhất, và lớp 4 có độ ưu tiên cao nhất. Một trang sẽ thuộc về một trong bốn lớp trên và trang được chọn làm trang "nạn nhân" là trang đầu tiên tìm thấy trong lớp có độ ưu tiên thấp nhất.

4.3.1.5.5 Các thuật toán thống kê

Sử dụng một biến đếm lưu số lần truy xuất đến một trang.

+ Thuật toán LFU (least frequently used): Thay thế trang có giá trị biến đếm nhỏ nhất, nghĩa là trang ít được sử dụng nhất.

+ Thuật toán MFU (most frequently used): Thay thế trang có giá trị biến đếm lớn nhất, nghĩa là trang được sử dụng nhiều nhất.

4.3.2 Cấp phát số lượng khung trang và thay thế trang

Với mỗi tiến trình, cần phải cấp phát một số khung trang tối thiểu nào đó để tiến trình có thể hoạt động. Số khung trang tối thiểu này được quy định bởi kiến trúc của của một chỉ thị. Ví dụ máy IBM 370 để lệnh MOVE có thể thực hiện tối thiểu phải có hai trang: một trang from , một trang to. Khi một lỗi trang xảy ra trước khi chỉ thị hiện hành hoàn tất, chỉ thị đó cần được tái khởi động, lúc đó cần phải có đủ các khung trang để nạp tất cả các trang mà một chỉ thị cần sử dụng. Số khung trang tối thiểu được qui định bởi kiến trúc máy tính, trong khi số khung trang tối đa được xác định bởi dung lượng bộ nhớ vật lý có thể sử dụng.

4.3.2.1 Cấp phát số lượng khung trang

Có ba cách cấp phát số lượng khung trang là: cấp phát ngang bằng, cấp phát theo tỉ lệ kích thước, cấp phát theo tỉ lệ độ ưu tiên.

a/ Cấp phát ngang bằng:

Nếu có m khung trang và n tiến trình, mỗi tiến trình được cấp m/n khung trang. Cấp phát này đơn giản nhưng không hiệu quả.

b/ Cấp phát theo tỷ lệ kích thước

Tùy vào kích thước của tiến trình để cấp phát số khung trang. Gọi s_i là kích thước của tiến trình p_i

$S = \sum s_i$ là tổng kích thước của tất cả tiến trình.

m = số lượng khung trang có thể sử dụng

Khi đó tiến trình pi sẽ được cấp phát ai khung trang

S --> m khung

si --> ai =?

ai= si/ S x m

ví dụ: có hai tiến trình, tiến trình 1= 10K, tiến trình 2=127K và có 62 khung trang trống. Khi đó có thể cấp cho

tiến trình 1: 10/137 x 62 ~ 4 khung

tiến trình 2: 127/137 x62 ~ 57 khung

c/ Cấp phát theo tỷ lệ độ ưu tiên:

Số lượng khung trang cấp cho tiến trình phụ thuộc vào độ ưu tiên của tiến trình. Tiến trình có độ ưu tiên cao sẽ được cấp nhiều khung hơn để tăng tốc độ thực hiện.

4.3.2.2 Thay thế trang

Có hai cách thay thế trang: thay thế toàn cục và thay thế cục bộ

a/ Thay thế toàn cục

Chọn trang " nạn nhân " từ tập tất cả các khung trang trong hệ thống, khung trang đó có thể đang được cấp phát cho một tiến trình khác. Ví dụ có thể chọn trang của tiến trình có độ ưu tiên thấp hơn làm trang nạn nhân. Thuật toán thay thế toàn cục cho phép hệ thống có nhiều khả năng lựa chọn hơn, số khung trang cấp cho một tiến trình có thể thay đổi, nhưng các tiến trình không thể kiểm soát được tỷ lệ phát sinh lỗi trang của mình.

b/ Thay thế cục bộ

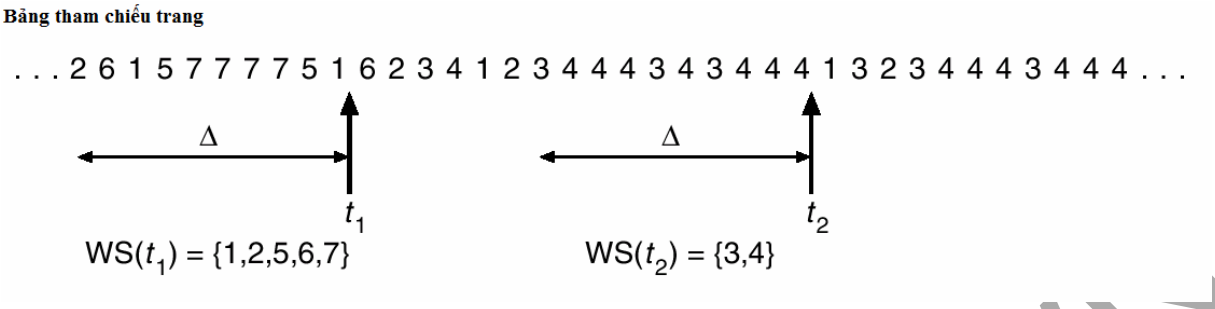
Chỉ chọn trang thay thế trong tập các khung trang được cấp cho tiến trình phát sinh lỗi trang, khi đó số khung trang cấp cho một tiến trình sẽ không thay đổi

4.3.3 Hệ thống trì trệ (thrashing)

Khi tiến trình không có đủ các khung trang để chứa những trang cần thiết cho việc xử lý, thì nó sẽ thường xuyên phát sinh các lỗi trang, vì thế phải dùng đến rất nhiều thời gian sử dụng CPU để thực hiện thay thế trang. Hệ điều hành thấy hiệu quả sử dụng CPU thấp sẽ tăng mức độ đa chương, dẫn đến trì trệ toàn bộ hệ thống. Để ngăn cản tình trạng trì trệ này xảy ra, cần phải cấp cho tiến trình đủ các khung trang cần thiết để hoạt động. Vấn đề là làm sao biết được mỗi tiến trình cần bao nhiêu trang?

4.3.3.1 Mô hình tập làm việc (working set)

Tập làm việc của tiến trình tại thời điểm t là tập các trang được tiến trình truy xuất đến trong Δ lần truy cập cuối cùng tính tại thời điểm t.



Hình 4.41: mô hình tập làm việc

Gọi WSSi (Δ , t) là số phần tử của tập working set của tiến trình Pi tại thời điểm t.

m là số khung trang trống.

D = ∑WSSi là tổng số khung trang yêu cầu cho toàn hệ thống .

Hệ điều hành giám sát working set của mỗi tiến trình Pi và tại thời điểm t sẽ cấp phát cho tiến trình Pi số khung trang bằng với số phần tử trong tập làm việc (WSSi)(Δ, t-1).

Nếu tổng số khung trang yêu cầu của các tiến trình trong hệ thống vượt quá các khung trang có thể sử dụng (D>m), thì sẽ xảy ra tình trạng hệ thống trì trệ. Khi đó hệ điều hành chọn một tiến trình để tạm dừng, giải phóng các khung trang của tiến trình đã chọn để các tiến trình khác có đủ khung trang hoàn tất công việc.

4.3.3.2 Cấu trúc chương trình

Số lỗi trang có khi phụ thuộc vào ngôn ngữ lập trình, nên khi lập trình ta cần chú ý để chương trình có thể thực hiện nhanh hơn.

Ví dụ: xét ct sau:

```
int  a[128][128];
for (i=0; i<128; i++)
for (j=0; j<128; j++)
                a[i][j]=0;
```

Giả sử trang có kích thước 128 bytes và tiến trình được cấp 2 khung trang: khung trang thứ nhất chứa mã tiến trình, khung trang còn lại được khởi động ở trạng thái trống . Trong Pascal, C mảng lưu theo hàng, mỗi hàng chiếm 1 trang bộ nhớ, nên số lỗi trang phát sinh là 128. Nhưng trong Fortran mảng lưu theo cột, do đó số lỗi trang sẽ là 128x128=1638.

TÓM TẮT

+ Các vấn đề cần phải giải quyết khi quản lý bộ nhớ là việc chuyển đổi địa chỉ tương đối thành địa chỉ thực, quản lý bộ nhớ đã cấp phát và chưa cấp phát, các kỹ thuật cấp phát bộ nhớ.

+ Việc chuyển đổi địa chỉ tương đối thành địa chỉ thực có thể xảy ra vào một trong những thời điểm sau: thời điểm biên dịch, thời điểm nạp, thời điểm xử lý.

+ Địa chỉ ảo là địa chỉ do bộ xử lý sinh ra, địa chỉ vật lý là địa chỉ thực trong bộ nhớ. Khi chương trình nạp vào bộ nhớ các địa chỉ tương đối trong chương trình được CPU chuyển thành địa chỉ ảo, khi thực thi, địa chỉ ảo được hệ điều hành kết hợp với phần cứng MMU chuyển thành địa chỉ vật lý .

+ Có hai phương pháp quản lý việc cấp phát bộ nhớ là sử dụng một dãy bit hoặc sử dụng một danh sách liên kết, mỗi nút của danh sách liên kết lưu thông tin một vùng nhớ chứa tiến trình hay vùng nhớ trống giữa hai tiến trình.

+ Để chọn một đoạn trống có thể sử dụng một trong các thuật toán sau :First-fit, Best-fit, Worst-fit

+ Có hai kỹ thuật dùng để cấp phát bộ nhớ cho một tiến trình là

Cấp phát liên tục: tiến trình được nạp vào một vùng nhớ liên tục.

Cấp phát không liên tục: tiến trình được nạp vào một vùng nhớ không liên tục

+ Có ba mô hình cấp phát bộ nhớ liên tục là mô hình Linker-Loader hoặc mô hình Base & Limit. Mô hình Linker-Loader: chương trình được nạp vào một vùng nhớ liên tục đủ lớn để chứa toàn bộ chương trình, hệ điều hành sẽ chuyển các địa chỉ tương đối về địa chỉ tuyệt đối ngay khi nạp chương trình. Mô hình Base & Limit giống như mô hình Linker-Loader nhưng phần cứng cần cung cấp hai thanh ghi, một thanh ghi nền và một thanh ghi giới hạn . Khi một tiến trình được cấp phát vùng nhớ, hệ điều hành cất vào thanh ghi nền địa chỉ bắt đầu của vùng nhớ cấp phát cho tiến trình, và cất vào thanh ghi giới hạn kích thước của tiến trình.

+ Có ba mô hình cấp phát bộ nhớ không liên tục là mô hình phân đoạn, mô hình phân trang và mô hình phân đoạn kết hợp phân trang.

- Mô hình phân đoạn: một chương trình được người lập trình chia thành nhiều phân đoạn, mỗi phân đoạn có ngữ nghĩa khác nhau và hệ điều hành có thể nạp các phân đoạn vào bộ nhớ tại các vị trí không liên tục và ghi các vị trí các phân đoạn vào bảng phân đoạn, đồng thời chuyển các địa chỉ tương đối trong chương trình thành các địa chỉ ảo. Mỗi địa chỉ ảo gồm hai phần (s,d): s là số hiệu phân đoạn , d là địa chỉ tương đối trong phân đoạn s. Mỗi phần tử trong bảng phân đoạn gồm hai phần (base, limit): base là địa chỉ vật lý bắt đầu phân đoạn, limit là chiều dài của phân đoạn.

- Mô hình phân trang: Bộ nhớ vật lý được chia thành các khối có kích thước cố định và bằng nhau gọi là khung trang. Tiến trình cũng được chia thành các khối có cùng kích thước với khung trang và gọi là trang. Khi chương trình được nạp vào bộ nhớ, MMU ghi nhận lại số hiệu khung trang chứa trang vào bảng trang , CPU chuyển địa chỉ tương đối trong chương trình thành địa chỉ ảo. Mỗi địa chỉ ảo có dạng (p,d): p là số hiệu trang, d là địa chỉ tương đối trong trang p. Mỗi phần tử trong bảng trang lưu số hiệu khung trang chứa trang.

- Mô hình phân đoạn kết hợp phân trang:

Một tiến trình gồm nhiều phân đoạn, mỗi phân đoạn được chia thành nhiều trang, lưu trữ vào các khung trang có thể không liên tục.

+ Bộ nhớ ảo là kỹ thuật dùng bộ nhớ phụ lưu trữ tiến trình, các phần của tiến trình được chuyển vào-ra giữa bộ nhớ chính và bộ nhớ phụ để cho phép thực thi một tiến trình mà không cần nạp

toàn bộ vào bộ nhớ vật lý. Có hai phương pháp cài đặt kỹ thuật bộ nhớ ảo đó là phân trang theo yêu cầu hoặc phân đoạn theo yêu cầu

- Phân trang theo yêu cầu:

Một tiến trình được chia thành nhiều trang, thường trú trên đĩa cứng và một trang chỉ được nạp vào bộ nhớ chính khi có yêu cầu. Nếu khi nạp trang mà không còn khung trang trống, chọn một khung trang "nạn nhân" và chuyển trang "nạn nhân " ra bộ nhớ phụ , rồi chuyển trang muốn truy xuất từ bộ nhớ phụ vào khung trang trống đã chọn.

+ Các thuật toán chọn trang nạn nhân: FIFO, tối ưu, LRU, các thuật toán xấp xỉ LRU

CÂU HỎI VÀ BÀI TẬP

1. Giả sử có một hệ thống sử dụng kỹ thuật phân trang theo yêu cầu. Bảng trang được lưu trữ trong các thanh ghi. Để xử lý một lỗi trang tốn 8 miliseconds nếu có sẵn một khung trang trống, hoặc trang bị thay thế không bị sửa đổi nội dung, và tốn 20 miliseconds nếu trang bị thay thế bị sửa đổi nội dung. Mỗi truy xuất bộ nhớ tốn 100 nanoseconds. Giả sử trang bị thay thế có xác suất bị sửa đổi là 70%. Tỷ lệ phát sinh lỗi trang phải là bao nhiêu để có thể duy trì thời gian truy xuất bộ nhớ (effective acess time) không vượt quá 200 nanoseconds?

2. Xét chương trình C sau :

```
int A [100][100] ;
for (i=0; i<100; i++)
for (j=0; j<100; j++) A[i][j]= 0;
```

Giả sử tiến trình được cấp 3 khung trang với kích thước một khung trang là 200 bytes, mã tiến trình luôn chiếm khung trang 1, khung trang 2 và 3 để lưu mảng A và khởi đầu khung 2, 3 là rỗng. Hỏi tiến trình có bao nhiêu lỗi trang khi sử dụng thuật toán thay thế LRU. Xét chương trình C sau với câu hỏi tương tự như trên

```
int A [100][100] ;
for (j=0; j<100; j++)
for (i=0; i<100; i++) A[i][j]= 0;
```

3. Trong một hệ thống sử dụng kỹ thuật phân trang theo yêu cầu, kích thước mỗi trang là 2K , xét đoạn chương trình C sau đây:

```
int n = 3*1024; int A[n], B[n];
for (i=0; i<n;i++) A[i]=i;
for (i=0 ;i<n;i++) B[A[i]]=i;
```

a) Nếu số khung cấp cho tiến trình là không hạn chế và giả sử khung trang thứ nhất luôn dùng để chứa tiến trình, các khung trang còn lại được khởi động ở trạng thái trống thì tiến trình có bao nhiêu lỗi trang.

b) Nếu số khung cấp cho tiến trình là 2 khung và giả sử khung trang thứ nhất luôn dùng để chứa tiến trình, khung trang thứ hai được khởi động ở trạng thái trống thì tiến trình có bao nhiêu lỗi trang.

4. Một máy tính có 4 khung trang. Thời điểm nạp, thời điểm truy cập cuối cùng, và các bit Reference (R), Dirty (D) của mỗi trang trong bộ nhớ được cho trong bảng sau :

Trang	Thời điểm nạp	Thời điểm truy cập cuối cùng	R	D
0	126	279	0	0
1	230	260	1	0
2	120	272	1	1
3	160	280	1	1

Trang nào sẽ được chọn thay thế theo :

a) thuật toán NRU

b) thuật toán FIFO

c) thuật toán LRU

d) thuật toán " cơ hội thứ 2"

5. Tính kích thước dãy bít dùng để quản lý RAM 512 MB, giả sử địa chỉ đánh theo byte.

6. Xét một không gian địa chỉ có 8 trang, mỗi trang có kích thước 1K, ánh xạ vào bộ nhớ vật lý có 32 khung trang.

a) Địa chỉ logic gồm bao nhiêu bit ?

b) Địa chỉ physic gồm bao nhiêu bit ?

7. Xét một hệ thống sử dụng kỹ thuật phân trang, với bảng trang được lưu trữ trong bộ nhớ chính.

a) Nếu thời gian cho một lần truy xuất bộ nhớ bình thường là 200 nanoseconds, thì mất bao nhiêu thời gian cho một thao tác truy xuất bộ nhớ trong hệ thống này ?

b) Nếu sử dụng TLBs với tỉ lệ tìm thấy (hit-ratio) là 75%, thời gian để tìm trong TLBs xem như bằng 0, tính thời gian truy xuất bộ nhớ trong hệ thống (effective memory reference time)

8. Xét bảng phân đoạn sau:

Segment	Base	Length
1	2300	14
2	90	100

Cho biết địa chỉ vật lý tương ứng với các địa chỉ ảo sau đây :

a. (1,10)

b. (2,500)

9. Một máy tính 32-bit địa chỉ, sử dụng một bảng trang nhị cấp. Địa chỉ ảo được phân bổ như sau: 9 bit dành cho bảng trang cấp 1, 11 bit cho bảng trang cấp 2, còn lại dành cho offset. Cho biết kích thước một trang trong hệ thống, và không gian địa chỉ ảo có bao nhiêu trang ?

10. Một máy tính có 48-bit địa chỉ ảo, và 32-bit địa chỉ vật lý, kích thước một trang là 8K. Có bao nhiêu phần tử trong một bảng trang thông thường và trong bảng trang nghịch đảo?

11. Giả sử có một máy tính đồ chơi sử dụng 7-bit địa chỉ, hệ thống sử dụng một bảng trang nhị cấp, dùng 2-bit làm chỉ mục đến bảng trang cấp 1, 2-bit làm chỉ mục đến bảng trang cấp 2. Xét một tiến trình sử dụng các địa chỉ ảo trong những phạm vi sau : 0..15, 21..29, 94..106, và 115..127.

a) Vẽ chi tiết toàn bộ bảng trang cho tiến trình này

b) Phải cấp phát cho tiến trình bao nhiêu khung trang, giả sử tất cả đều nằm trong bộ nhớ chính?

c) Bao nhiêu bytes ứng với các vùng phân mảnh nội vi trong tiến trình này?

d) Cần bao nhiêu bộ nhớ cho bảng trang của tiến trình này?

12. Giả sử có một máy tính sử dụng 16-bit địa chỉ. Bộ nhớ ảo được thực hiện với kỹ thuật phân đoạn kết hợp phân trang, kích thước tối đa của một phân đoạn là 4096 bytes. Bộ nhớ vật lý được phân thành các khung trang có kích thước 512 bytes.

a) Thể hiện cách địa chỉ ảo được phân tích để phản ánh segment, page, offset

b) Xét một tiến trình sử dụng các miền địa chỉ sau, xác định số hiệu segment và số hiệu page tương ứng trong segment mà chương trình truy cập đến :

350..1039, 3046..3904, 7100..9450, 33056..39200, 61230..63500

c) Bao nhiêu bytes ứng với các vùng phân mảnh nội vi trong tiến trình này?

d) Cần bao nhiêu bộ nhớ cho bảng phân đoạn và bảng trang của tiến trình này ?

TÀI LIỆU THAM KHẢO

[1]. Gary J. Nutt, University of Colorado. Centralized And Distributed Operating Systems. Second Edition, 2000.

[2]. Robert Switzer. Operating Systems, A Practical Approach. Prentice-Hall International, Inc. 1993.

[3]. Andrew S. Tanenbaum. Modern Operating Systems. Prentice-Hall International, Inc. Second Edition, 2001.

[4]. Abraham Silberschatz & Peter Baer Galvin. Operating System concepts. John Wiley & Sons, Inc. Fifth Edition, 1999.

[5]. H. M. Deitel. Operating Systems. Addison-Wesley Inc. Second Edition, 1999.

[6]. Trần Hạnh Nhi & Lê Khắc Nhiên Ân & Hoàng Kiếm. Giáo trình hệ điều hành (tập 1 & 2). ĐHKHTN 2000.

CHƯƠNG 5
QUẢN LÝ PROCESSOR

Chương “QUẢN LÝ PROCESSOR" sẽ giới thiệu và giải thích các vấn đề sau:

5.1 Processor Vật lý và Processor logic

5.2 Ngắt và xử lý ngắt

5.3 Xử lý ngắt trong IBM-PC

5.1. PROCESSOR VẬT LÝ VÀ PROCESSOR LOGIC

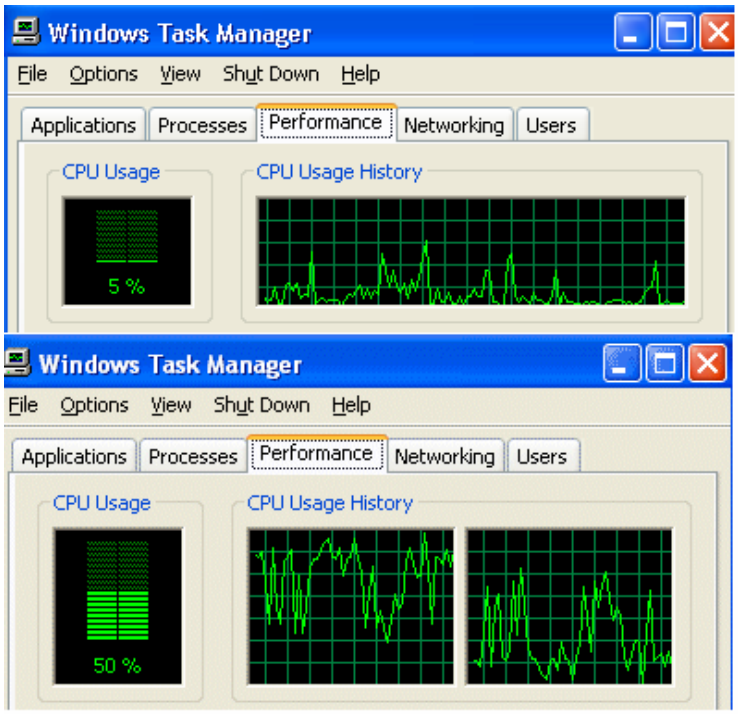
Trong lĩnh vực công nghệ thông tin, khái niệm bộ xử lý (processor) thường được gọi là đơn vị xử lý trung tâm (CPU-Central Processing Unit). CPU là một thành phần bên trong máy tính thực hiện công việc biên dịch các chỉ thị của máy tính và xử lý các dữ liệu bên trong các chương trình. Cùng với đơn vị lưu trữ chính, thiết bị nhập/xuất, CPU là thành phần cơ bản nhất không thể thiếu trong bất kỳ hệ thống máy tính nào.

Để tăng tốc độ cho một hệ thống máy tính, ngoài việc sử dụng nhiều CPU, còn có một giải pháp khác – hyper threading (siêu phân luồng). Với kỹ thuật này, hệ điều hành sẽ nhìn thấy một processor vật lý thành hai processor logic. Processor vật lý là một thành phần phần cứng thực sự, thực hiện các thao tác do hệ điều hành và các chương trình đang thực thi trên máy tính ra lệnh. Trong khi đó, processor logic được tạo ra bởi các chương trình, tài nguyên hệ điều hành, hoặc là sự mô phỏng của processor vật lý. Một lý do khác để tạo ra processor logic là để tận dụng tối đa nguồn tài nguyên bên trong bộ xử lý đồng thời cho phép thực hiện tính toán cho nhiều tiểu trình hơn trong cùng một thời điểm.

Hình 5.1. Processor vật lý và processor logic

Giả sử hệ thống có một CPU vật lý. Nếu CPU này không hỗ trợ kỹ thuật siêu phân luồng thì hệ điều hành chỉ nhìn thấy hệ thống có một CPU (vật lý) duy nhất. Ngược lại, nếu CPU này hỗ trợ kỹ thuật siêu phân luồng thì hệ điều hành sẽ cho rằng hệ thống này có hai CPU (logic). Hình 5.2 minh họa điều này khi thực hiện trên hai hệ thống được giả sử như trên. Khi đó, ở hình (b) ta sẽ

thấy có hai ô hình chữ nhật rời nhau để đo Performance cho từng CPU (logic). Trong khi đó, ở hình (a) chỉ có một CPU (vật lý) được đo Performance.



(a)

Hệ thống không hỗ trợ kỹ thuật siêu phân luồng. Hệ điều hành chỉ nhìn thấy 1 CPU (vật lý).

(b)

Hệ thống hỗ trợ kỹ thuật siêu phân luồng. Hệ điều hành nhìn thấy 2 CPU (logic). Nhưng thực chất nó chỉ có 1 CPU (vật lý).

Hình 5.2. Minh họa Performance của hệ thống có 1 CPU vật lý và hệ thống có 2 CPU logic

5.2. NGẮT VÀ XỬ LÝ NGẮT

Ngắt là sự xảy ra của một điều kiện, sự kiện làm cho treo tạm thời chương trình trong khi đó điều kiện, sự kiện này được phục vụ bởi chương trình khác. Một hệ thống được điều khiển bằng ngắt cho ảo giác là nó thực hiện nhiều công việc đồng thời. Dĩ nhiên, CPU mỗi lần không thể thực thi nhiều hơn một lệnh; nhưng nó có thể tạm treo việc thực thi một chương trình để thực thi chương trình khác, rồi quay về chương trình thứ nhất. Cụ thể hơn, khi ngắt xảy ra, thì:

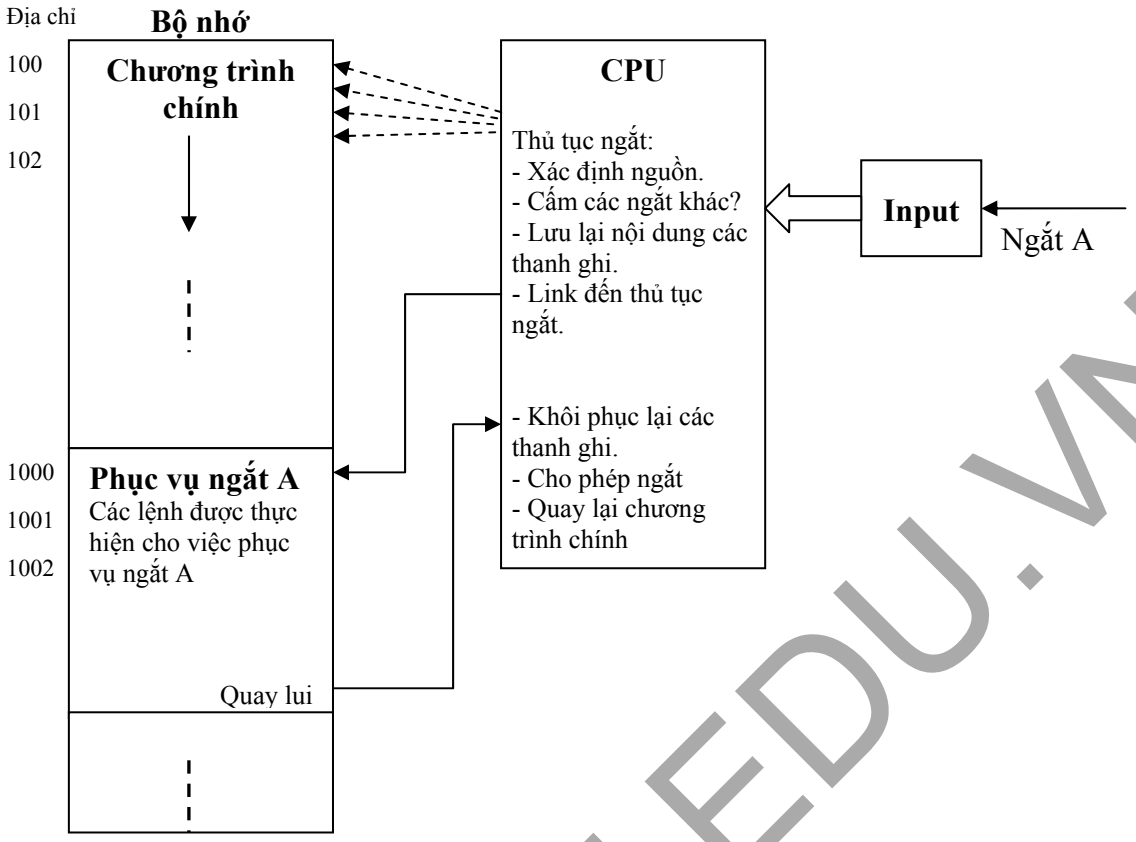
+ Hệ điều hành lấy được sự điều khiển.

+ Hệ điều hành lưu lại trạng thái của tiến trình bị ngắt. Trong nhiều hệ thống, các thông tin này được lưu trong khối điều khiển tiến trình của tiến trình bị ngắt.

+ Hệ điều hành phân tích ngắt và chuyển điều khiển đến một thủ tục tương ứng để thực hiện ngắt.

+ Thủ tục thực hiện ngắt tiến hành ngắt.

+ Tiến trình bị ngắt được thực thi.



Hình 5.3 Thủ tục ngắt

Ngắt có thể được nạp từ một tiến trình đang thực thi, từ một sự cố có liên quan hoặc không liên quan đến tiến trình đang thực thi. Như ví dụ trong hình 5.11, CPU nhảy đến địa chỉ 1000 và thực thi chương trình ở đó. Và ở cuối thủ tục ngắt, một lệnh được khởi tạo để khôi phục lại trạng thái của các thanh ghi bị treo trong chương trình chính, nhờ đó nó cũng giúp khôi phục lại việc điều khiển cho chương trình gốc.

Cấu trúc ngắt có thể khác nhau giữa các hệ thống, nhưng nhìn chung chúng đều thực hiện cùng một thủ tục ngắt như nhau. Dưới đây là một số yêu cầu mà một hệ thống ngắt phải đáp ứng được trước khi thực hiện ngắt:

+ Định nghĩa một tập các sự kiện có khả năng gây ra ngắt.

+ Có phương tiện để ghi lại các tình huống (ngữ cảnh) khi một ngắt xảy ra, thường là một hoặc một vài bit cờ (flag).

+ CPU phải thực hiện việc kiểm tra trạng thái ngắt theo thời gian định kỳ. Việc thiết lập và kiểm tra trạng thái các cờ ngắt thường được xử lý tự động bởi phần cứng với rất ít hoặc không có sự tham gia của CPU.

+ Hệ thống phải kiểm tra để xác định xem yêu cầu ngắt xuất phát từ đâu, và để quyết định có nên cấp CPU để đáp ứng yêu cầu ngắt đó hay không.

+ Xác định vị trí bên trong CPU, để lưu thông tin về nguyên nhân gây ra ngắt.

+ Nếu yêu cầu ngắt cần được đáp ứng, thì CPU phải nhảy đến chương trình hoặc thủ tục phục vụ ngắt tương ứng.

+ Bên trong hệ điều hành, các ngắt thường được sử dụng để truy xuất các dịch vụ của kernel (nhân của hệ điều hành). Việc thâm nhập vào kernel của một hệ điều hành có liên quan mật thiết

đến quá trình xử lý ngắt, một vài khả năng đáp ứng ngắt là yêu cầu tất yếu của kernel trong một hệ điều hành. Có ba loại ngắt được sử dụng trong một hệ điều hành như sau:

a/ Ngắt giám sát (supervisor call interupt): là một loại ngắt đặc biệt, xảy ra khi một tiến trình phát ra một chỉ thị yêu cầu một thủ tục bên trong hệ điều hành. Sau đó, hệ điều hành có thể được xem như là một tập các chương trình hệ thống được liên kết với nhau để thực thi các tín hiệu ngắt.

b/ Ngắt nội (internal interupt): hay còn gọi là ngắt mềm, được tạo ra bởi các sự kiện nào đó, bên trong bộ xử lý đang thực thi các sự kiện đó, chẳng hạn như khi một chương trình thực hiện sai chức năng (ví dụ như chương trình cố gắng chia một số cho 0).

c/ Ngắt ngoại (external interupt): hay còn gọi là ngắt cứng, được tạo ra bên ngoài bộ xử lý đang xảy ra ngắt, thường là bởi các bộ xử lý khác hoặc các thiết bị nhập/xuất trong hệ thống.

Mỗi loại ngắt có một thủ tục xử lý ngắt (*interupt handler*) để xử lý các yêu cầu ngắt tương ứng. Khi một thủ tục xử lý ngắt đoạt được quyền điều khiển CPU, như minh họa trong hình 5.11, thông thường nó sẽ cấm tất cả các chỉ thị từ các ngắt khác cho đến khi nó đạt đến "điểm an toàn". Điểm an toàn là vị trí mà thông tin trạng thái ngắt có thể được lưu và sẽ được khôi phục lại sau khi yêu cầu ngắt đã được đáp ứng.

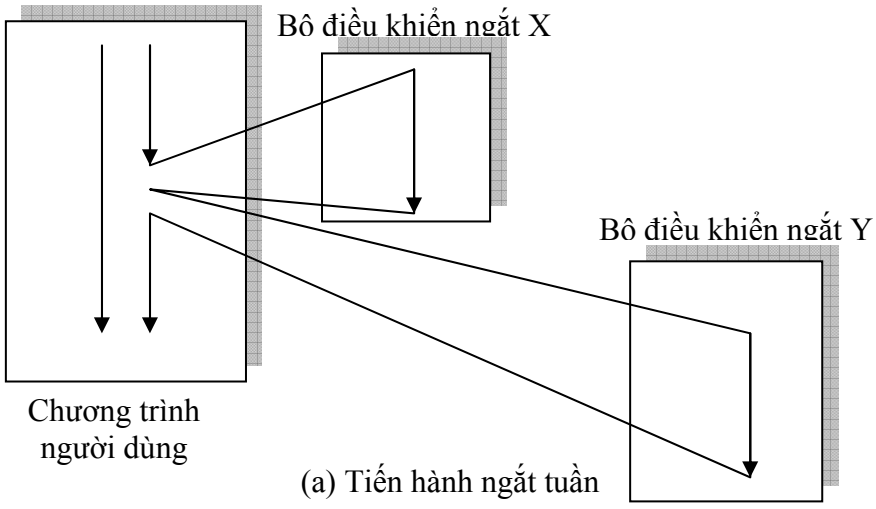
5.2.1. Đồng hồ ngắt

Để ngăn chặn những người sử dụng chiếm độc quyền hệ thống (chủ động hoặc bị động), hệ điều hành sẽ có các cơ chế để đòi lại CPU. Hệ điều hành cài một đồng hồ ngắt để ngắt các hoạt động chiếm giữ CPU trong một thời gian xác định. Khi CPU được trao cho một tiến trình, tiến trình đó được quyền điều khiển CPU cho đến khi nó chủ động trả lại CPU hoặc khi đồng hồ ngắt thực hiện ngắt. Nếu người sử dụng vẫn còn đang thực thi chương trình, và đồng hồ ngắt xảy ra việc ngắt giờ, thì việc ngắt này sẽ lấy quyền xử lý về lại cho hệ điều hành. Sau đó hệ điều hành sẽ quyết định tiến trình nào sẽ được nhận CPU tiếp theo.

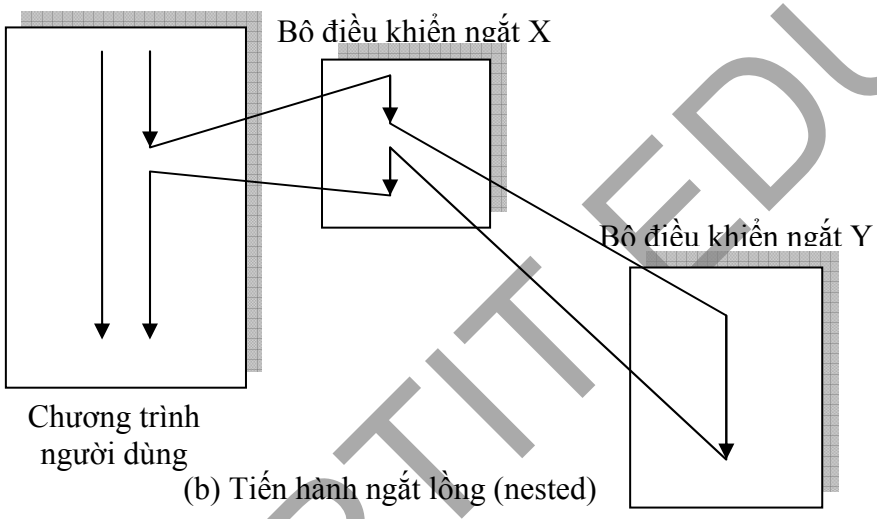
Đồng hồ ngắt giúp đảm bảo được thời gian hồi đáp hợp lý cho hệ thống đa người dùng, ngăn cản hệ thống bị treo khi có một người sử dụng đang thực thi một vòng lặp vô hạn.

5.2.2. Đa ngắt

Những vấn đề chúng ta vừa nêu trên được áp dụng cho sự xuất hiện của một ngắt tại một thời điểm. Tuy nhiên, trong thực tế, một lúc có thể xảy ra nhiều ngắt. Chẳng hạn như, một chương trình có thể nhận dữ liệu từ bên ngoài vào và in ra kết quả tương ứng. Mỗi lần máy in chỉ thực hiện được một ngắt cho đến khi kết thúc hoạt động in. Bộ điều khiển bus sẽ thực hiện một ngắt mỗi khi có một đơn vị dữ liệu đến. Như vậy, ngắt điều khiển bus cho phép nhận dữ liệu sẽ xảy ra trong khi ngắt phục vụ in ấn đang thi hành.



(a) Tiến hành ngắt tuần



(b) Tiến hành ngắt lồng (nested)

Hình 5.4. Quy trình điều khiển đa ngắt

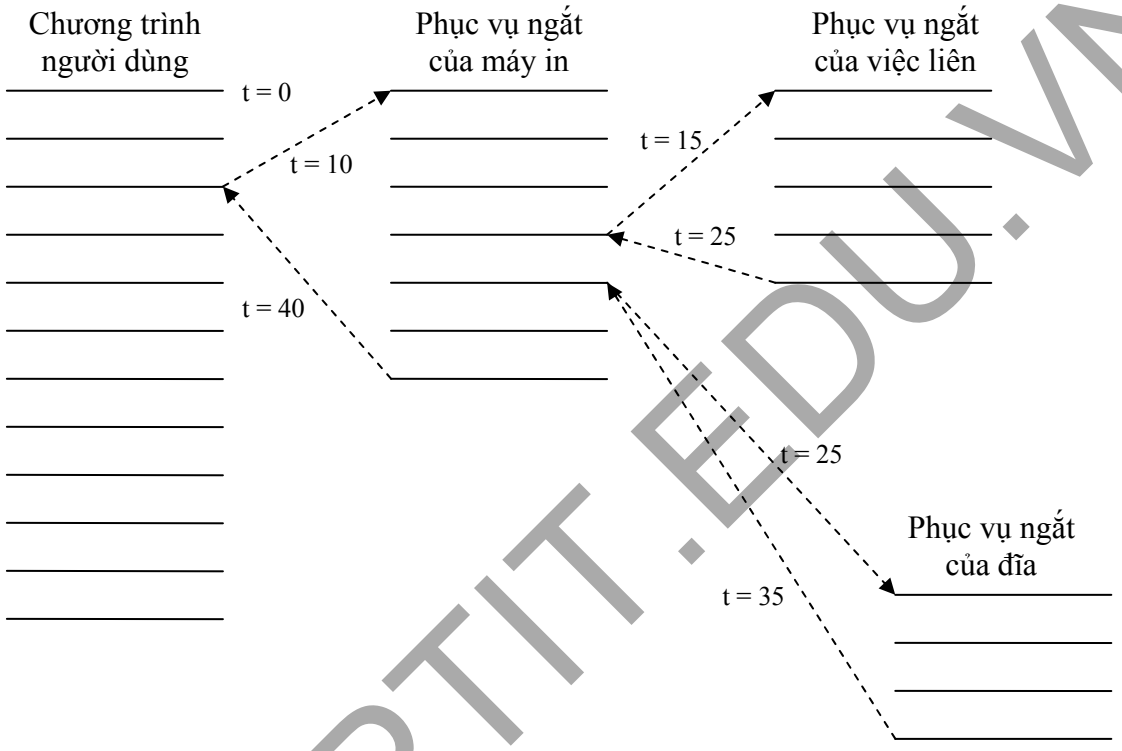
*** Có hai cách tiếp cận để giải quyết vấn đề đa ngắt:**

Cấm các ngắt hoạt động trong khi có một ngắt đang tiến hành. Nghĩa là CPU có thể và sẽ bỏ qua các tín hiệu yêu cầu ngắt. Nếu một ngắt xảy ra trong thời gian này thì nó sẽ bị treo và sẽ được kiểm tra sau khi bộ xử lý cho phép ngắt. Do đó, khi một chương trình của người sử dụng đang thực thi và một ngắt xảy ra thì các ngắt khác sẽ bị cấm ngay lập tức. Sau khi thủ tục phục vụ ngắt hoàn tất thì các ngắt được cho phép trước khi tái kích hoạt chương trình của người sử dụng và bộ xử lý sẽ kiểm tra xem có ngắt nào xảy ra nữa không. Cách tiếp cận này dễ và đơn giản vì các ngắt được điều khiển theo một thứ tự nghiêm ngặt. Minh hoạ trong hình 5.4(a) cho ta thấy cách xử lý ngắt trong hệ thống đa ngắt theo cách tiếp cận này. Ngắt X được phục vụ xong thì mới tời lượt ngắt Y được phục vụ.

Xác định thứ tự ưu tiên cho các ngắt và cho phép ngắt có thứ tự ưu tiên thấp hơn tự ngắt. Nghĩa là, trong khi một ngắt đang được phục vụ, nếu có ngắt khác xảy đến, thì hệ thống phải tiến hành kiểm tra thứ tự yêu tiên của các ngắt. Nếu ngắt đang được phục vụ có độ ưu tiên cao hơn thì ngắt đến sau sẽ bị bỏ qua. Ngược lại, nếu ngắt đang được phục vụ có độ ưu tiên thấp hơn, thì nó phải thực hiện viêc "tự ngắt" để cho phép ngắt đến sau thực hiện trước. Minh họa trong hình 5.4(b). Ngắt A

đang được phục vụ, giả sử ngắt B đến sau nhưng có độ ưu tiên cao hơn ngắt A, thì ngắt A phải tự ngắt để trao quyền điều khiển CPU cho ngắt B.

Trở lại ví dụ ban đầu, giả sử một hệ thống có ba thiết bị nhập/xuất: một máy in, một ổ đĩa, và một đường liên lạc với các mức ưu tiên tăng dần tương ứng là 2, 4, và 5. Hình 5.5 mô tả một chuỗi các thao tác có thể xảy ra.



Hình 5.5. Một ví dụ về xử lý ngắt có độ ưu tiên

Một chương trình của người sử dụng bắt đầu tại thời điểm t=0. Tại thời điểm t=10, một ngắt cho máy in xảy ra; các thông tin của người sử dụng được lưu trong ngăn xếp của hệ thống và việc thực thi vẫn tiếp tục tại thủ tục phục vụ ngắt của máy in. Trong khi thủ tục này vẫn đang thực thi thì tại thời điểm t=15 một ngắt liên lạc xảy ra. Vì đường liên lạc có mức ưu tiên cao hơn máy in nên việc ngắt cho nó được ưu tiên trước Thủ tục phục vụ ngắt của máy in bị ngắt, trạng thái của nó lại được đưa vào ngăn xếp và việc thực thi vẫn được tiếp tục với thủ tục phục vụ ngắt của sự liên lạc. Trong khi thủ tục này đang thực thi thì một ngắt cho đĩa xảy ra (t=20). Vì ngắt này có mức ưu tiên thấp hơn nên nó phải chờ và thủ tục phục vụ ngắt của việc liên lạc vẫn hoạt động cho đến khi nó hoàn tất.

Khi thủ tục phục vụ ngắt cho việc liên lạc kết thúc (t=25) thì trạng thái của bộ xử lý trước được khôi phục (trạng thái thực thi thủ tục phục vụ ngắt của máy in). Tuy nhiên, trước khi một lệnh trong thủ tục đó được thực thi thì bộ xử lý ưu tiên cho việc ngắt đĩa có mức ưu tiên cao hơn và sự điều khiển được chuyển cho thủ tục phục vụ ngắt của đĩa. Chỉ khi thủ tục này chấm dứt (t=35) thì thủ tục phục vụ ngắt của máy in mới được tái kích hoạt. Khi thủ tục đó kết thúc (t=40) thì sự điều khiển trở về chương trình của người sử dụng.

5.3. Xử lý ngắt trong IBM-PC

Trong phần này, chúng ta sẽ tìm hiểu một số ngắt cứng quan trọng và việc xử lý chúng như thế nào trong IBM-PC. Các thiết kế đầu tiên của IBM-PC dựa trên bộ xử lý 8088 và sử dụng bộ điều khiển ngắt 8259 cho phép tạo ra 8 tín hiệu ngắt đồng thời.

Ngắt 00h: chia 0

8088 có hai lệnh chia (DIV và IDIV), chúng cho phép chia các số 16 bit và 32 bit cho các số 8 và 16 bit. Theo qui tắc toán học, thì phép chia cho 0 là không hợp lệ. Bởi vậy, 8088 cấm các phép chia có thương số là 0. Nếu xuất hiện phép chia cho 0, bộ bộ xử lý sẽ gọi ngắt 00h để thực hiện một thủ tục xử lý tương ứng (hiển thị thông báo "Division by Zero").

Ngắt 01h: thực hiện từng lệnh

CPU gọi ngắt này khi bit TRAP của thanh ghi cờ bằng 1. Ngắt sẽ được gọi sau mỗi lệnh máy. Ngắt này cho phép người sử dụng theo dõi việc thực hiện từng lệnh của một chương trình ngôn ngữ máy. Để không xảy ra việc gọi đệ quy ngắt này, bộ xử lý sẽ đặt bit TRAP bằng 0 khi vào ngắt này. Nó cất thanh ghi cờ vào ngăn xếp, có nghĩa là cả bit TRAP.

Nếu thủ tục xử lý ngắt được kết thúc bằng lệnh IRET, thì thanh ghi cờ (chứa bit TRAP) được tự động khôi phục từ ngăn xếp. sau khi thực hiện xong lệnh tiếp theo, ngắt 01h lại được gọi. sau khi người lập trình đã nhận đủ thông tin về chương trình, thì bit TRAP có thể bị xóa. Tuy nhiên, chương trình đang được kiểm tra không hề biết rằng nó đang được thực hiện từng lệnh, và nó không có lệnh nào để xóa bit TRAP trong thanh ghi cờ. Tuy nhiên, ngắt 01h rất ít khi được sử dụng bởi chương trình.

Ngắt 02h: NMI (ngắt không che)

Ngắt không che (Non-Maskable Interupt) được gọi như vậy bởi lẻ người dùng không thể ngăn cản nó. Người dùng có thể ngăn cản các ngắt khác bằng lệnh CLI, nhưng đối với ngắt 02h thì không thể. NMI thông báo cho người sử dụng biết rằng có lỗi trong RAM. Lỗi này có thể xuất hiện do hỏng một chip RAM nào đấy.Vì chip RAM hỏng có thể làm này sinh những ảnh hưởng nghiêm trọng cho hệ thống, nên ngắt này có độ ưu tiên cao nhất.

Ngắt 03h: Điểm dừng

Ngắt này được sử dụng trong các chương trình tiện ích. Khác với các ngắt khác, là các ngắt được gọi bởi một lệnh máy dài 2 bytes, ngắt này được gọi bởi một lệnh máy dài 1 byte. Ngắt này rất hữu ích để kiểm tra một chương trình chạy tới một điểm nào đó. Ngắt 03h dừng việc chạy chương trình, và cho phép người sử dụng kiểm tra nội dung các thanh ghi của bộ xử lý.

Ngắt 04h: Lỗi tràn

Ngắt này có thể được gọi bởi một lệnh. Đó là lệnh INTO (Interupt of Overflow), lệnh này chỉ gọi ngắt 04h khi bit tràn của thanh ghi cờ bằng 1. Điều này có thể xảy ra sau một thao tác toán học, nếu như kết quả của thao tác này không chứa nổi trong một số bit đã đặt.

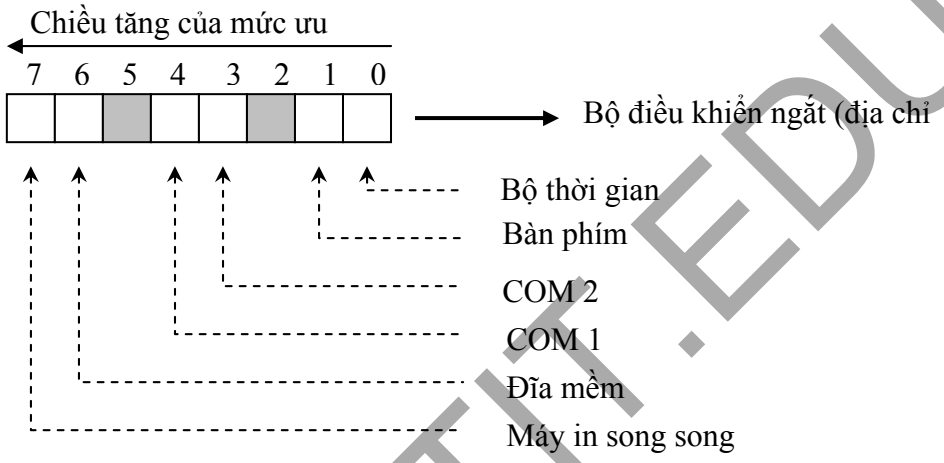
Ngắt 05h: Copy cứng

Khi ấn phím PrtScr, ngắt này sẽ được gọi. Ngắt này gửi toàn bộ nội dung màn hình ra máy in.

Ngắt 06h-07h: không sử dụng (dành cho các ứng dụng về sau).

Ngắt 08h -0Fh: Các ngắt này do bộ điều khiển ngắt 8259 tạo ra. Bộ điều khiển ngắt này nhận tất cả các yêu cầu ngắt của hệ thống. Nó xác định mức ưu tiên của các yêu cầu ngắt. Có đến 8 nguồn sinh ngắt (thiết bị) có thể được nối tới 8259, mỗi thiết bị được gán một mức ưu tiên. Thông qua các bit của thanh ghi cờ, CPU có thể cấm tất cả các ngắt của 8259 (trừ ngắt 02h và NMI).

Ta cũng có thể cấm ngắt từ một thiết bị nào đó, bằng cách đặt bit ngắt tương ứng của thanh ghi che ngắt thuộc 8259 bằng 1. Thanh ghi này được truy nhập thông qua cổng 21h. 8 bit của thanh ghi này tương ứng với 8 thiết bị sinh ngắt. Bit 0 tương ứng với thiết bị số 0, bit 7 tương ứng với thiết bị số 7. Nếu bit có giá trị 0, thì CPU sẽ nhận được ngắt do thiết bị tương ứng sinh ra. Nếu ngắt bằng 1, yêu cầu ngắt sẽ bị bỏ qua. Nếu đồng thời có nhiều yêu cầu ngắt, yêu cầu ngắt số 0 sẽ có mức ưu tiên cao nhất, yêu cầu ngắt số 7 có mức ưu tiên thấp nhất. Nếu yêu cầu ngắt mức cao được xử lý xong, thì yêu cầu ngắt tiếp theo sẽ được 8259 chuyển tới CPU. Hình 6.14 mô tả các yêu cầu ngắt của các thiết bị và mức ưu tiên của chúng trên PC. Trong hình này, các bit 2 và 5 không được sử dụng. Bit 2 được sử dụng bởi bộ điều khiển ngắt thứ hai (trong các máy AT). Bit 5 chưa dùng do trước đây nó được IBM dành riêng cho cổng song song LPT2, nhưng vì có quá ít người dùng sử dụng cổng LPT2 nên IBM không tiếp tục phát triển ngắt này nữa (tuy nhiên, card âm thanh có thể sẽ sử dụng nó). Do vậy, sau này trên các dòng máy XT thì bit 5 được dùng để phục vụ cho ngắt của đĩa cứng.



Hình 5.6. Các yêu cầu ngắt và mức yêu tiên trên PC

Ngắt 08h: bộ thời gian (bit 0)

Cứ sau 65.536 nhịp tín hiệu (khoảng 18,2 lần/giây), ngắt 08h lại được gọi. bộ điều khiển ngắt 8259 sẽ chuyển yêu cầu ngắt này tới CPU. Vì sự xuất hiện ngắt này không phụ thuộc tần số nhịp của hệ thống, nên nó có thể dùng để đo thời gian. Sau 18,2 lần gọi ngắt, thì có nghĩa là đã trôi qua một giây.

Ngắt 09h: ngắt bàn phím (bit 1)

Bàn phím được hỗ trợ bởi bộ xử lý Intel 8048(PC/XT) hoặc 8042 (AT). Bộ xử lý này kiểm soát và ghi nhận phím bị ấn, nhả hoặc bị ấn và giữ liên tục. Nó gửi tín hiệu yêu cầu ngắt tới 8259, bộ này lại yêu cầu CPU gọi ngắt số 09h (trong trường hợp đồng thời đang có một ngắt khác có mức ưu tiên cao hơn).

Ngắt 0Ah -0Ch: thay đổi

Các ngắt này thay đổi tùy theo từng máy.

Ngắt 0Dh: đĩa cứng (bit 5)

Ngắt này được gọi khi kết thúc các thao tác đọc, ghi đĩa cứng.

Ngắt 0Eh: đĩa mềm (bit 6)

Ngắt này được gọi khi kết thúc các thao tác đọc, ghi đĩa mềm hoặc khi có lỗi xuất hiện.

Ngắt 0Fh: máy in (bit 7)

Máy in song song gọi ngắt này thông qua 8259 khi nó cần sự phục vụ của CPU.

5.3.1. Thanh ghi ngắt

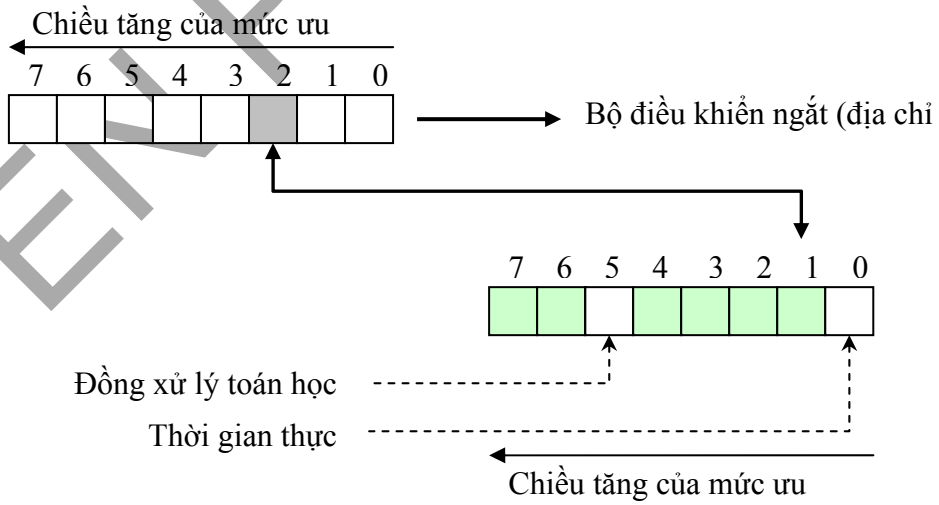
8259 biết được khi nào một yêu cầu ngắt đã được xử lý xong là do thanh ghi chỉ thị ngắt ở địa chỉ 20h. Khi một ngắt của 8259 kết thúc, nó thực hiện lệnh OUT giá trị 20h (EOI = End Of Interupt) tới cổng 20h. Lệnh này thông báo cho 8259 biết rằng việc xử lý ngắt đã kết thúc, và 8259 có thể chấp nhận yêu cầu ngắt tiếp theo.

Việc định nghĩa các bit cuả thanh ghi che ngắt là khác nhau, tùy loại PC. Có thể coi rằng, thiết bị tương ứng bit 0 của thanh ghi she ngắt sẽ gọi ngắt 08h. Thiết bị tương ứng bit 1 sẽ gọi ngắt 09h … Ngắt 0Fh, ngắt cuối cùng của 8259, sẽ được kích hoạt bởi thiết bị tương ứng bit 7 của thanh ghi che ngắt. Tổng quát, 8 ngắt trên có các ký hiệu là IRQ0, IRQ1, … IRQ có nghĩa là Interupt Request (yêu cầu ngắt).

5.3.2. Các bộ điều khiển ngắt của máy AT

AT có hai bộ điều khiển ngắt 8259, và nó có thể xử lý tới 16 yêu cầu ngắt (nguồn sinh ngắt). Các ngắt của bộ điều khiển ngắt thứ hai có tên là IRQ8, …, IRQ15. Nếu xuất hiện một yêu cầu ngắt của bộ điều khiển ngắt thứ 2, thì nó được coi như là yêu cầu ngắt số 2 của bộ điều khiển ngắt thứ nhất. Bởi vậy các yêu cầu ngắt của bộ điều khiển ngắt thứ hai có độ ưu tiên cao hơn các yêu cầu ngắt từ 3 đến 7 của bộ điều khiển ngắt thứ nhất.

Ta cũng có thể che các yêu cầu ngắt của bộ điều khiển ngắt thứ hai, bằng cách thao tác các bit của thanh ghi che ngắt thuộc 8259 thứ hai. Thanh ghi này có địa chỉ cổng A0h. ta phải gửi lệnh EOI tới cả hai bộ điều khiển ngắt sau khi kết thúc việc xử lý một yêu cầu ngắt của bộ điều khiển ngắt thứ hai. Đó là vì hai bộ điều khiển này được nối với nhau, mỗi yêu cầu ngắt của bộ thứ hai đều làm xuất hiện yêu cầu ngắt số 2 của bộ thứ nhất. Các hình sau đây cho thấy các yêu cầu ngắt của các thiết bị và mức ưu tiên của chúng.



Hình 5.7. Các yêu cầu ngắt và mức yêu tiên trên máy AT

5.3.3. Các ngắt của AT

Vì có thêm bộ xử lý ngắt thứ hai, nên AT có nhiềuu ngắt cứng hơn PC và XT. Bộ điều khiển ngắt thứ hai có thể gọi các ngắt từ 70h tới 77h. Thiết bị tương ứng yêu cầu ngắt số 0 sẽ gọi ngắt 70h, thiết bị tương ứng yêu cầu ngắt số 1 gọi ngắt 71h, …

Chỉ có các ngắt 70h và 75h là được bộ điều khiển ngắt gọi, bởi lẽ chỉ có hai thiết bị nối với 8259 này. Tuy nhiên, các vector ngắt 71h-74h và 76h-77h có thể được đổi hướng cho các mục đích khác. Hình 5.15 mô tả chi tiết một số yêu cầu ngắt và mức độ ưu tiên của chúng trên máy AT.

Ngắt 70h: đồng hồ thời gian thực

Ngắt 70h được gọi khi đến thời điểm báo chuông, thời gian vừa được cập nhật, hoặc vừa xảy ra một ngắt chu kỳ. ngắt này thường được một thủ tục của BIOS xử lý.

Ngắt 75h: bộ đồng xử lý tóan học

Ngắt 75h thông báo cho CPU biết là bộ đồng xử lý toán học yêu cầu phục vụ (thì dụ, nó đã thực hiện xong một phép tính nào đó).

TÓM TẮT

Trong lĩnh vực công nghệ thông tin, khái niệm bộ xử lý (processor) thường được gọi là đơn vị xử lý trung tâm (CPU-Central Processing Unit). CPU là một thành phần bên trong máy tính thực hiện công việc biên dịch các chỉ thị của máy tính và xử lý các dữ liệu bên trong các chương trình. Cùng với đơn vị lưu trữ chính, thiết bị nhập/xuất, CPU là thành phần cơ bản nhất không thể thiếu trong bất kỳ hệ thống máy tính nào.

Để tăng tốc độ cho một hệ thống máy tính, ngoài việc sử dụng nhiều CPU, còn có một giải pháp khác – hyper threading (siêu phân luồng). Với kỹ thuật này, hệ điều hành sẽ nhìn thấy một processor vật lý thành hai processor logic. Processor vật lý là một thành phần phần cứng thực sự, thực hiện các thao tác do hệ điều hành và các chương trình đang thực thi trên máy tính ra lệnh. Trong khi đó, processor logic được tạo ra bởi các chương trình, tài nguyên hệ điều hành, hoặc là sự mô phỏng của processor vật lý. Một lý do khác để tạo ra processor logic là để tận dụng tối đa nguồn tài nguyên bên trong bộ xử lý đồng thời cho phép thực hiện tính toán cho nhiều tiểu trình hơn trong cùng một thời điểm.

CÂU HỎI VÀ BÀI TẬP

1. Phân biệt giữa CPU vật lý và CPU logic. Mục tiêu của CPU logic là gì?

2. Kỹ thuật siêu phân luồng và Dual core có tương tự nhau không? Hãy phân biệt bản chất của chúng.

3. Tại sao ngắt được xem là mức điều khiển cao nhất trong một bộ xử lý?

4. Định nghĩa và phân loại một vài loại ngắt cứng, ngắt mềm. Chỉ ra mục đích của ngắt mềm.

5. Điểm an tòan trong quá trình thực thi một chương trình là gì? Đồng hồ ngắt có mục đích gì trong hệ thống?

TÀI LIỆU THAM KHẢO

[1]. Gary J. Nutt, University of Colorado. Centralized And Distributed Operating Systems. Second Edition, 2000.

[2]. Robert Switzer. Operating Systems, A Practical Approach. Prentice-Hall International, Inc. 1993.

[3]. Andrew S. Tanenbaum. Modern Operating Systems. Prentice-Hall International, Inc. Second Edition, 2001.

CHƯƠNG 6
HỆ ĐIỀU HÀNH NHIỀU BỘ VI XỬ LÝ

Chương "HỆ ĐIỀU HÀNH NHIỀU BỘ VI XỬ LÝ" sẽ giới thiệu và giải thích các vấn đề sau:

6.1 Cấu hình nhiều processor

6.2 Các loại hệ điều hành hỗ trợ nhiều bộ vi xử lý

6.3 Đồng bộ trong hệ thống đa xử lý

6.4 Điều phối trong hệ thống đa xử lý

6.1. CẤU HÌNH NHIỀU PROCESSOR

Mặc dù tốc độ máy tính ngày càng được cải thiện nhờ vào các công nghệ mới, nhưng nhu cầu của con người vẫn chưa được thỏa mãn. Một trong các cách tiếp cận nhằm mục đích tăng tốc độ của máy tính đó là sử dụng nhiều bộ xử lý trên một máy. Mỗi bộ xử lý chỉ cần hoạt động ở tốc độ bình thường cũng đã cung cấp cho hệ thống khả năng tính toán tốt hơn rất nhiều so với hệ thống chỉ có một bộ xử lý. Hệ thống có nhiều bộ xử lý có thể chia làm 3 loại chính như sau:

+ Đa xử lý dùng bộ nhớ chia sẻ.

+ Đa xử lý dùng bộ nhớ riêng.

+ Đa xử lý phân tán.

Đối với mô hình thứ nhất, các CPU truyền thông với nhau để thực hiện một hoặc một số công việc nào đó thông qua việc sử dụng chung một bộ nhớ (bộ nhớ chia sẻ). Trong mô hình này, các CPU đều có quyền như nhau để truy xuất vào bộ nhớ vật lý. Đối với mô hình thứ hai, hệ thống gồm nhiều cặp CPU-bộ nhớ được kết nối với nhau thông qua các đường kết nối tốc độ cao. Trong mô hình này, bộ nhớ là cục bộ đối với mỗi CPU và chỉ có thể được truy xuất bởi CPU đó. Còn trong mô hình thứ ba, hệ thống cũng gồm nhiều cặp CPU-bộ nhớ, nhưng chúng kết nối với nhau thông qua mạng diện rộng, chẳng hạn như Internet, và hình thành nên một hệ thống phân tán. Trong mô hình này, việc truyền thông giữa các cặp CPU-bộ nhớ này cũng sử dụng cách chuyển thông điệp (message passing), giống như trong mô hình thứ hai. Tuy nhiên, sự khác nhau giữa hai hệ thống này đó là độ trễ. Độ trễ để truyền thông điệp giữa các cặp CPU-bộ nhớ trong mô hình thứ hai là thấp hơn rất nhiều so với độ trễ trong mô hình thứ 3.

Một hệ thống đa xử lý dùng bộ nhớ chia sẻ (shared-memory multiprocessor, hoặc đôi khi người ta chỉ gọi multiprocessor) là một hệ thống có hai hoặc nhiều hơn hai CPU cùng chia sẻ một bộ nhớ RAM. Một chương trình chạy trên bất kỳ CPU nào cũng đều có khả năng nhìn thấy cùng một không gian địa chỉ ảo như nhau (nói cách khác, chúng được phân trang như trong hệ thống có một bộ xử lý). Tuy nhiên, điều khác biệt trong hệ thống đa xử lý thể hiện ở chỗ, một CPU có thể ghi vào một từ nhớ nào đó một giá trị là *a* nhưng khi đọc ra có thể sẽ mang giá trị khác *a* (bởi vì một CPU khác đã làm thay đổi giá trị này). Điều này tạo nên đặc tính cơ bản của việc truyền thông giữa các tiến trình với nhau trong hệ thống có nhiều bộ xử lý – một CPU ghi dữ liệu vào trong bộ nhớ và một CPU khác sẽ đọc để lấy dữ liệu đó ra. Nói chung, hệ điều hành dùng cho hệ thống đa xử lý cũng tương tự như hệ điều hành trong hệ thống đơn xử lý. Nó cũng xử lý các lời gọi hệ thống, thực hiện việc quản lý bộ nhớ, cung cấp cơ chế quản lý tập tin cũng như các cơ chế quản lý vào ra. Tuy nhiên, có một số vấn đề mới mà chúng ta cần quan tâm khi nghiên cứu một hệ điều

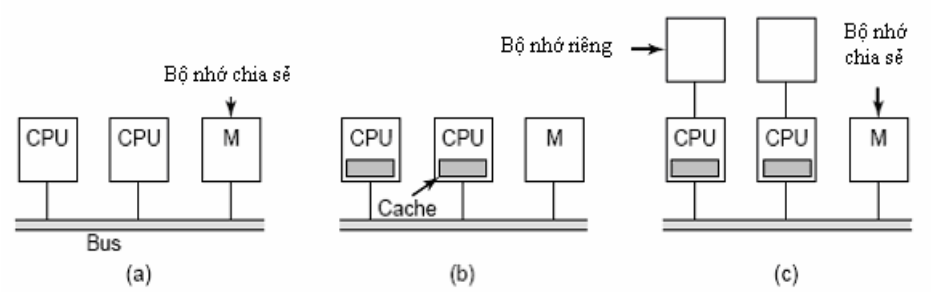
hành dùng trong hệ thống đa xử lý. Chẳng hạn như: việc quản lý tài nguyên, việc đồng bộ tiến trình, cũng như việc điều phối tiến trình trong nhiều bộ xử lý khác nhau.

Trong giới hạn của giáo trình này, chúng tôi chỉ cung cấp cho bạn đọc chi tiết về cấu hình phần cứng cũng như các vấn đề liên quan về hệ điều hành cho hệ thống đa xử lý dùng bộ nhớ chia sẻ. Phần tương tự cho hai hệ thống còn lại, bạn đọc có thể tham khảo thêm trong các tài liệu khác về hệ điều hành khác.

Mặc dù các hệ thống có nhiều bộ xử lý đều cho phép mọi CPU có thể truy xuất đến bộ nhớ của hệ thống, nhưng một vài hệ thống có thêm một đặc tính nữa, đó là nó cho phép mọi từ nhớ có thể được đọc ra với cùng một tốc độ. Những hệ thống này được gọi là hệ thống đa xử lý UMA (Uniform Memory Access Multiprocessor). Ngược lại, hệ thống nào không có khả năng trên thì được gọi là NUMA (NonUniform Memory Access Multiprocessor). Vì sao có sự khác biệt này chúng ta sẽ tìm hiểu ở phần sau, còn bây giờ chúng ta sẽ lần lượt tìm hiểu từng loại hệ thống một.

6.1.1. Hệ thống đa xử lý UMA dùng mô hình Bus

Các hệ thống có nhiều bộ xử lý đơn giản nhất đều dựa trên một bus chung, được minh họa trong hình 6-1(a). Hai hoặc nhiều CPU và một hoặc nhiều bộ nhớ, tất cả sử dụng một tuyến *bus* để truyền thông. Khi một CPU muốn đọc một từ nhớ, trước tiên nó phải kiểm tra xem *bus* có rỗi không. Nếu trạng thái *bus* là rỗi, CPU sẽ gởi địa chỉ của từ nhớ mà nó muốn đọc dữ liệu lên trên *bus*, kiểm tra một vài tín hiệu điều khiển, và đợi cho đến khi bộ nhớ đặt từ nhớ được yêu cầu lên lên *bus*.



Hình 6.1: Các hệ thống đa xử lý dùng mô hình Bus: (a) Không có cache. (b) Có cache. (c) Có cache và các bộ nhớ riêng.

Còn nếu *bus* bận khi một CPU muốn đọc hoặc ghi bộ nhớ, CPU phải đợi đến khi *bus* trở về trạng thái rỗi. Đối với hệ thống có 2 hoặc 3 CPU, việc cạnh tranh *bus* là có thể quản lý được. Tuy nhiên nếu hệ thống có số lượng CPU lớn hơn (ví dụ 32, hoặc 64 CPU), điều này là không thể. Hệ thống bị hạn chế hoàn toàn bởi băng thông cho phép của *bus*, và vì vậy hầu hết các CPU sẽ rỗi trong phần lớn thời gian.

Giải pháp cho vấn đề này là thêm bộ nhớ *cache* cho mỗi CPU như chỉ ra trong hình 6-1(b). Bộ nhớ *cache* có thể nằm bên trong *chip* của CPU, nằm kế bên, nằm trên bo mạch của CPU, hoặc được kết hợp từ các cách trên. Điều này thực sự làm giảm bớt tải cho *bus* chung. Và vì vậy hệ thống có thể hỗ trợ nhiều CPU hơn. Nói chung, *caching* không thực hiện trên một đơn vị nhớ riêng nào, mà nó dựa trên khối các byte (thường là các khối 32-byte hoặc 64-byte). Khi một từ nhớ được tham chiếu đến, thì toàn bộ khối (*block)* chứa từ nhớ đó sẽ được nạp vào trong *cache* của CPU yêu cầu.

Mỗi *block* dữ liệu trong *cache* hoặc là *read-only* (trong trường hợp nó hiện diện trong nhiều *cache* ở cùng một thời điểm), hoặc là *read-write* (trong trường hợp nó không hiện diện trong các *cache* khác). Nếu một CPU cố gắng ghi một từ nhớ đang hiện diện bên trong một hoặc một vài *cache* khác, một tín hiệu sẽ được phần cứng phát lên trên *bus* để thông báo cho các *cache* khác biết việc ghi này. Nếu các *cache* này có một bản sao (giống như bản gốc trong bộ nhớ), thì chúng sẽ chỉ hủy bỏ những bản sao này và cho phép CPU nạp *block* dữ liệu từ bộ nhớ vào *cache*. Đối với một vài *cache* có một bản sao đã được thay đổi, thì nó phải hoặc là được ghi ngược lại ra bộ nhớ trước khi việc ghi được thực hiện, hoặc được truyền trực tiếp đến CPU có nhu cầu ghi thông qua *bus*.

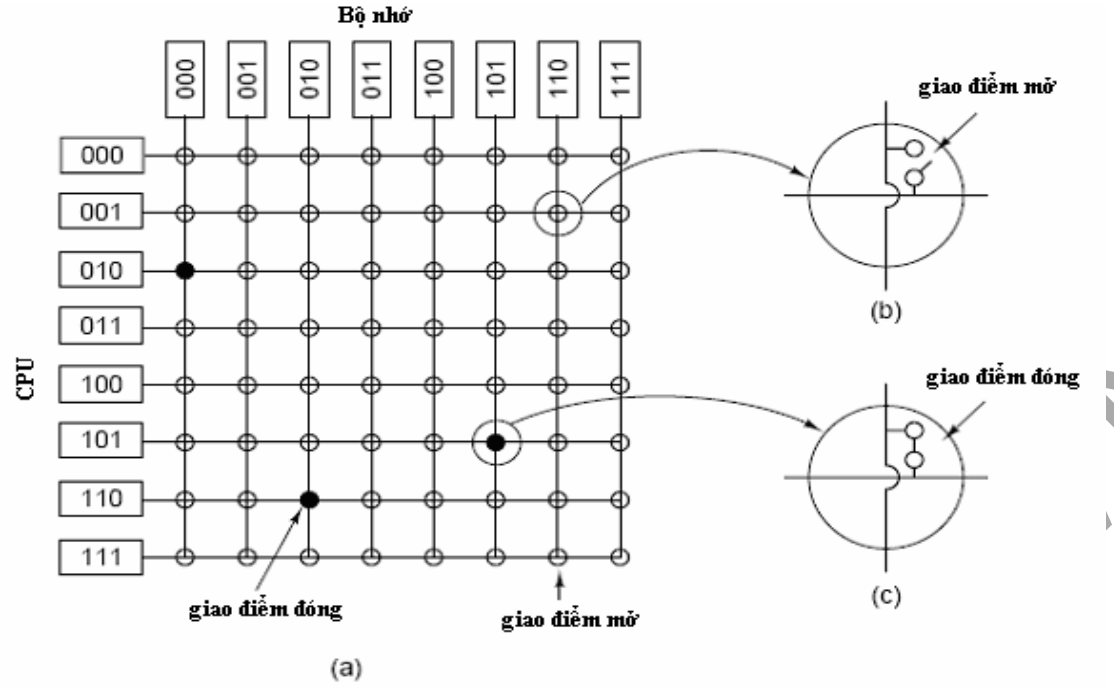
Một khả năng khác có thể được thiết kế như trong hình 6-1(c). Trong mô hình này, mỗi CPU không chỉ có *cache* mà còn có bộ nhớ riêng được truy xuất thông qua một *bus* riêng. Để sử dụng tối ưu cấu hình này, trình biên dịch nên đặt tất cả các chương trình text, chuỗi, hằng số và những dữ liệu chỉ đọc, ngăn xếp, và các biến cục bộ vào trong các bộ nhớ riêng này. Bộ nhớ chia sẻ dùng chung chỉ được sử dụng cho các biến chia sẻ. Trong hầu hết các trường hợp, việc làm này sẽ giảm đáng kể lưu lượng cho *bus* chung, tuy nhiên điều này cũng đòi hỏi sự tích cực hợp tác từ trình biên dịch.

6.1.2. Hệ thống đa xử lý UMA dùng mô hình chuyển mạch chéo (Crossbar Switch)

Ngay cả khi hệ thống được hỗ trợ nhiều CPU với bộ nhớ *cache*, thì việc sử dụng một tuyến *bus* duy nhất cũng chỉ cho phép tối đa 16 hoặc 32 CPU trong một hệ thống đa xử lý UMA. Nhằm năng cao hơn nữa khả năng đáp ứng cho hệ thống, cần thay đổi cách kết nối các CPU. Một cách kết nối đơn giản giữa *n* CPU và *k* bộ nhớ để hình thành một mô hình kết nối chéo được thể hiện trong hình 6-2. Mô hình này đã được ứng dụng cách đây nhiều thập kỷ trong các tổng đài chuyển mạch điện thoại để kết nối một nhóm các *line* vào và một tập các *line* ra. Trạng thái của mỗi giao điểm (*crosspoint*), điểm giao nhau giữa đường ngang (*line* vào) và đường dọc (*line* ra), là đóng hay mở tùy thuộc vào trạng thái kết nối hay không kết nối của đường ngang và đường dọc này. Trong hình 6-2(a), 3 *crosspoint* đóng đồng thời, cho phép 3 kết nối giữa CPU và bộ nhớ được hình thành cùng lúc. Đó là các cặp (001, 000), (101, 101), và (110, 010). Đương nhiên là nhiều sự kết hợp khác cũng đều có khả năng như vậy. Mô hình này có thể hỗ trợ tối đa *n*x*k* sự kết hợp có thể có giữa *n* CPU và *k* bộ nhớ.

Một trong những đặc điểm nổi bật của mô hình này là nó đảm bảo hệ thống không bị nghẽn. Nghĩa là sẽ không có trường hợp một CPU nào đó không có bộ nhớ để làm việc chỉ vì một vài điểm *crosspoint* đã bị sử dụng. Ngoài ra, hệ thống cũng không cần phải lập ra kế hoạch phân phối tài nguyên cho CPU trước. Ngay cả khi những kết nối bất kỳ giữa CPU và bộ nhớ đã được thiết lập, hệ thống đều có khả năng cho phép thực hiện kết nối các CPU và bộ nhớ còn lại với nhau.

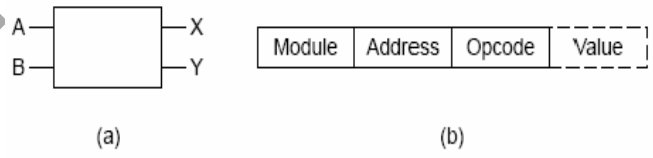
Một trong những yếu điểm lớn nhất của mô hình này là số lượng *crosspoint* rất lớn khi số CPU và bộ nhớ tăng lên. Với 1000 CPU và 1000 bộ nhớ thì hệ thống sẽ có 1000000 *crosspoint*. Một mô hình kết nối lớn như thế là không khả thi. Tuy nhiên, đối với các hệ thống với kích thước nhỏ hơn, thì đây là một mô hình tuyệt vời.



Hình 6.2. (a) Chuyển mạch chéo 8x8. (b) Giao điểm mở. (c) Giao điểm đóng.

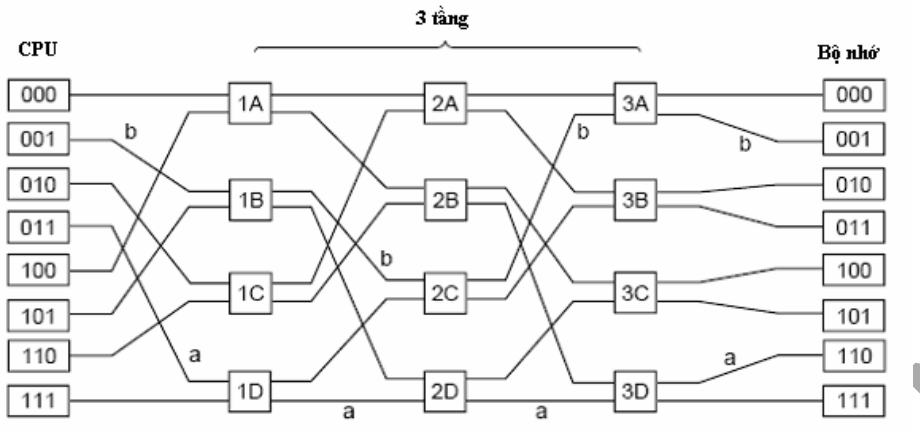
6.1.3. Hệ thống đa xử lý UMA dùng mô hình mạng chuyển mạch đa tầng (Multistage Switching Network)

Một thiết kế hoàn toàn khác cho hệ thống đa xử lý dựa trên chuyển mạch *2x2* được trình bày trong hình 6-3(a). Chuyển mạch này có 2 ngỏ vào và 2 ngỏ ra. Các *message* có thể đến bất kỳ một trong hai ngỏ vào và được chuyển ra theo một trong hai ngỏ ra. Theo đó, mỗi *message* sẽ gồm 4 phần, như trong hình 6-3(b). Trong hình này, trường *Module* cho biết vùng nhớ nào được sử dụng. Trường *Address* cho biết địa chỉ nào trong vùng nhớ đó. Trường *Opcode* cho biết hoạt động gì sẽ được thực hiện (đọc (READ) hay ghi (WRITE)). Và trường *Value* là trường tùy chọn, cho biết toán hạng nào sẽ được dùng vào việc đọc hoặc ghi (chẳng hạn như một từ 32-bit sẽ được đọc hoặc ghi). Chuyển mạch (*switch*) sẽ kiểm tra trường *Module* và dùng nó để xác định xem *message* nên đi ra ngỏ nào, X hay Y.



Hình 6.3. (a) Chuyển mạch 2x2. (b) Định dạng của *Message*.

Các chuyển mạch *2x2* có thể được sắp xếp theo nhiều cách để tạo nên một mạng chuyển mạch đa tầng (multistage switching network). Một kiến trúc điển hình cho loại này được trình bày trong hình 6-4. Ở đây, 8 CPU được kết nối với 8 bộ nhớ sử dụng 12 switch. Một cách tổng quát, với *n* CPU và *n* bộ nhớ, chúng ta cần $log_2 n$ tầng (stage) với *n/2* switch cho mỗi tầng. Nghĩa là, tổng cộng hệ thống cần $(n/1)log_2 n$ switch. Điều này rõ ràng là tốt hơn nhiều so với hệ thống đa xử lý UMA dùng chuyển mạch chéo, cần tới n^2 crosspoint, đặc biệt là khi *n* mang giá trị lớn.



Hình 6.3. Mạng chuyển mạch Omega

Xét mạng chuyển mạch Omega như trong hình 6-4, giả sử CPU 011 muốn đọc một từ nhớ (*word*) từ bộ nhớ 110. CPU gởi *message* READ đến chuyển mạch 1D chứa giá trị 110 trong trường *Module*. Switch lấy bit đầu tiên (bên trái nhất) của 110 và dùng nó cho việc định tuyến. Nếu bit này có giá trị 0, switch sẽ chọn lên ngỏ ra phía trên, ngược lại, nếu bit này có giá trị 1, thì switch sẽ chọn tuyến bên dưới. Như vậy, trong trường hợp này, bit đầu tiên có giá trị 1, nên *message* được đưa đến ngỏ ra bên dưới để đi đến switch 2D.

Tất cả các switch ở tầng thứ 2, bao gồm switch 2D, bit thứ 2 (từ trái sang) sẽ được dùng vào việc định tuyến. Và trong trường hợp này, bit thứ 2 có giá trị 1, nên *message* cũng được chuyển đến ngỏ ra bên dưới đến switch 3D. Tại đây, bit thứ 3 từ trái sang sẽ được kiểm tra, và vì nó mang giá trị 0 nên *message* sẽ được chuyển đến ngỏ ra bên trên và đi đến bộ nhớ 110. Kết quả là *message* sẽ đi theo con đường được đánh dấu bằng ký tự *a* trong hình 6-4.

Giả sử tại cùng thời điểm diễn ra những việc trên, CPU 001 muốn ghi một *word* đến bộ nhớ 001. Một tiến tình tương tự như vậy cũng xảy ra, ở đó, *message* được định tuyến thông qua các cổng theo thứ tự như sau: *message* đến ngỏ vào trên của switch 1B và đi ra ở ngỏ ra trên của 1B, sau đó đến switch 2C, và đi ra ở ngỏ ra trên của 2C để đến switch 3A, sau cùng thì *message* sẽ đi ra ở ngỏ ra dưới của switch 3A để đến bộ nhớ 001. Kết quả là message sẽ đi theo con đường được đánh dấu bằng ký tự *b* trong hình 6-4. Bởi vì 2 yêu cầu này sử dụng các switch, kết nối và bộ nhớ khác nhau, nên không xảy ra bất kỳ sự đụng độ nào, chúng có thể thực hiện công việc một cách đồng thời.

Tuy nhiên, điều gì sẽ xảy ra nếu CPU 000 đồng thời muốn truy xuất bộ nhớ 000. Yêu cầu của nó sẽ đụng độ với nhu cầu của CPU 001 tại switch 3A. Một trong hai yêu cầu này phải đợi. Không giống như cơ chế chuyển mạch chéo, mạng Omega là một mạng có khả năng xảy ra nghẽn. Không phải mọi yêu cầu đều có thể được xử lý đồng thời. Đụng độ có thể xảy ra do việc sử dụng chung kết nối, switch hoặc bộ nhớ mà các yêu cầu truy xuất đến.

Từ hệ thống này, người ta mong đợi có một hệ thống được cải tiến hơn bằng cách cho phép các *word* liên tục được lưu trong các bộ nhớ khác nhau. Điều này cho phép hệ thống truy xuất đến bộ nhớ nhanh hơn. Ngòai ra, tình trạng nghẽn mạng cũng có thể được khắc phục bằng cách cung cấp nhiều đường đi từ một CPU này đến một bộ nhớ bất kỳ, khi đó tốc độ truy xuất bộ nhớ cũng được cải thiện đáng kể.

6.1.4. Hệ thống đa xử lý NUMA

Các hệ thống đa xử lý UMA dùng *bus* thường bị giới hạn tối đa khoảng vài tá CPU, còn các hệ thống dùng chuyển mạch chéo hoặc chuyển mạch đa tầng thì cần nhiều phần cứng hỗ trợ. Để cho phép một hệ thống có thể hỗ trợ tốt với trên 100 CPU, người ta đưa ra một cách tiếp cận khác. Với cách tiếp cận này, việc truy xuất bộ nhớ cục bộ sẽ nhanh hơn việc truy xuất bộ nhớ ở xa. Như vậy, các chương trình hỗ trợ UMA sẽ chạy tốt trên các máy hỗ trợ NUMA mà không có sự thay đổi nào. Trong khi đó, các chương chương trình được hỗ trợ NUMA sẽ giảm hiệu suất thực thi khi chạy trên các máy hỗ trợ UMA ở cùng một tốc độ đồng hồ.

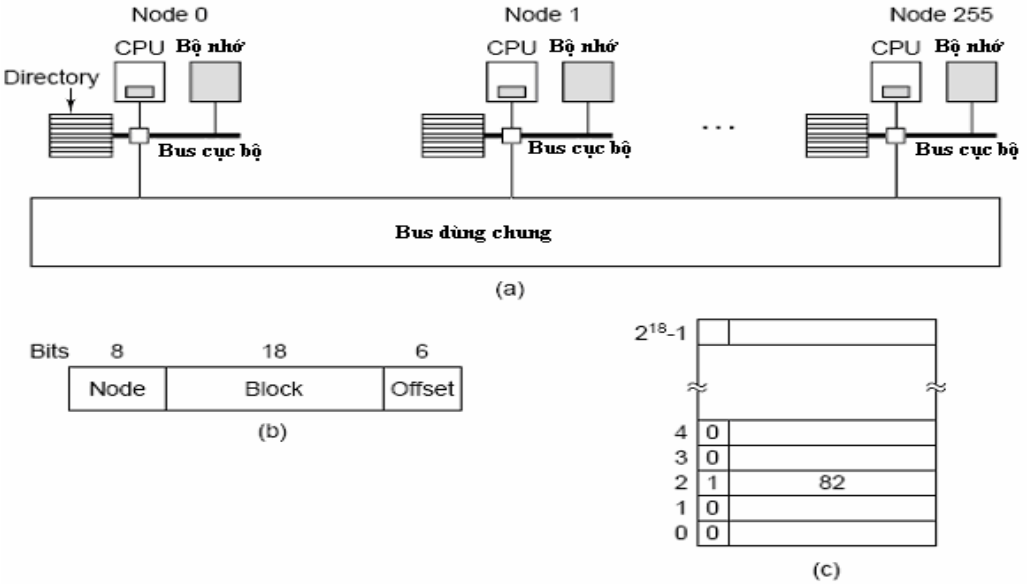
Các máy NUMA có 3 đặc điểm chính có thể phân biệt với các hệ thống đa xử lý khác, đó là:

Có một không gian địa chỉ duy nhất có thể nhìn thấy bởi tất cả các CPU.

Truy xuất bộ nhớ ở xa thông qua hai lệnh LOAD và STORE.

Truy xuất bộ nhớ ở xa chậm hơn truy xuất bộ nhớ cục bộ.

Khi thời gian truy xuất bộ nhớ ở xa có sự khác biệt lớn so với thời gian truy xuất bộ nhớ cục bộ (bởi vì không có cơ chế *caching*) thì hệ thống được gọi là NC-NUMA (NonCaching NUMA). Ngược lại, khi có các bộ nhớ *cache*, thì hệ thống được gọi là CC-NUMA (Cache Coherent NUMA).



Hình 6.4. (a) Hệ thống đa xử lý sử dụng Directory – 256 node. (b) Phân chia địa chỉ ô nhớ 32-bit thành các trường. (c) Cấu trúc Directory của node 36.

Cách tiếp cận phổ biến nhất để xây dựng một hệ thống đa xử lý CC-NUMA hiện nay là sử dụng một cơ sở dữ liệu để lưu vị trí của các khối cache và trạng thái của chúng. Khi một khối cache được tham chiếu, cơ sở dữ liệu được yêu cầu được truy vấn để tìm ra vị trí và trạng thái của nó là nguyên bản hay đã bị sửa đổi. Vì cơ sở dữ liệu này phải được truy vấn bởi mọi chỉ thị lệnh tham chiếu đến bộ nhớ, nên nó phải được lưu giữ trong một thiết bị phần cứng đặc biệt hỗ trợ tốc độ truy xuất cực nhanh.

Để làm rõ hơn ý tưởng của hệ thống này, chúng ta xét ví dụ đơn giản được mô tả như trong hình 6-5. Một hệ thống gồm 256 node, mỗi node gồm một CPU và 16MB bộ nhớ RAM được kết nối đến CPU thông qua một *bus* cục bộ. Tổng bộ nhớ là 2^{32} byte, được chia làm 2^{26} khối cache, mỗi khối 64 byte. Bộ nhớ được định vị cố định tại mỗi node, với 0-16MB cho node 0, 16MB–32MB

cho node 1 … Các node được kết nối với nhau như trong hình 6-5(a). Ngoài ra, mỗi node cũng lưu giữ các thực thể (*entry*) trong thư mục (directory) cho 2^{18} khối cache 64-byte (hình thành bộ nhớ 2^{24} byte) tương ứng.

Để thấy rõ cơ chế làm việc của hệ thống này, thực hiện theo vết lệnh LOAD từ CPU 20 như sau. Đầu tiên, CPU sẽ phát chỉ thị lệnh đến đơn vị quản lý bộ nhớ của nó (MMU), đơn vị này sẽ chuyển chỉ thỉ lệnh đó sang một địa chỉ vật lý (giả sử là 0x24000108). MMU tiếp tục chia địa chỉ này thành 3 phần như trong hình 6-5(b). Giả sử 3 phần này lần lượt có giá trị là node 36, khối cache 4 và offset 8. Như vậy, MMU nhận thấy rằng, *word* được tham chiếu là từ node 36, không phải node 20, do vậy nó gởi một *message* yêu cầu đến node 36, là node quản lý khối cache 4, hỏi xem khối 4 có được cache hay không, nếu có thì nó được cache ở đâu.

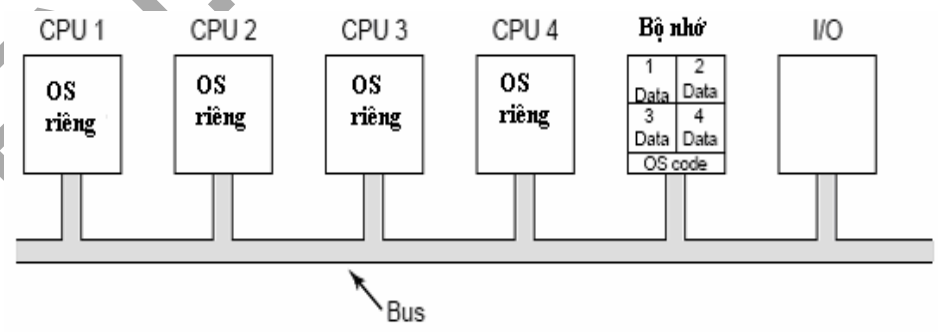
Khi yêu cầu này đến node 36, nó sẽ được chuyển đến phần cứng *Directory*. Phần cứng này sẽ dò trong bảng gồm 2^{18} thực thể của nó để tìm ra thực thể 4. Từ hình 6-5(c), chúng ta thấy rằng khối 4 không được cache, do vậy phần cứng sẽ nạp dòng 4 từ bộ nhớ RAM cục bộ và gởi ngược lại node 20, đồng thời cập nhật *Directory* ở thực thể 4 và chỉ ra rằng khối này bây giờ được cache ở node 20.

Bây giờ, chúng ta xem xét một yêu cầu khác, lần này node 20 hỏi về khối cache 2 của node 36. Từ hình 6-5(c), chúng ta thấy rằng khối này được cache tại node 82. Như vậy, phần cứng phải cập nhật thực thể 2 trong *Directory* để chỉ ra rằng khối này đang được cache ở node 20 đồng thời vô hiệu hóa *cache* của nó.

6.2. CÁC LOẠI HỆ ĐIỀU HÀNH HỖ TRỢ NHIỀU BỘ XỬ LÝ

Trong phần này chúng ta chuyển từ phần cứng sang tìm hiểu về phần mềm, cụ thể là chúng ta sẽ tìm hiểu về các hệ điều hành hỗ trợ cho các hệ thống có nhiều bộ xử lý. Có rất nhiều loại, tuy nhiên, ở đây chúng ta sẽ tìm hiểu 3 trong số các loại đó.

6.2.1. Mỗi CPU có riêng một hệ điều hành



Hình 6.5. Phân chia bộ nhớ cho các CPU trong hệ thống đa xử lý, nhưng cùng chia sẻ chung tập lệnh của hệ điều hành. Dữ liệu cũng được lưu trữ riêng cho từng CPU.

Cách đơn giản nhất để tổ chức một hệ điều hành hỗ trợ nhiều bộ xử lý là phân chia cố định bộ nhớ thành nhiều phần tương ứng với số lượng CPU mà hệ thống hỗ trợ. Mỗi CPU được cấp một bộ nhớ riêng và sở hữu một bản sao riêng của hệ điều hành. Kết quả là, *n* CPU sau đó sẽ hoạt động như là *n* máy tính độc lập. Một mô hình tối ưu như được trình bày trong hình 6-6, ở đó, hệ thống

cho phép các CPU chia sẻ *code* của hệ điều hành trong khi dữ liệu thì được lưu trữ riêng tại các vùng nhớ đã dành riêng cho chúng.

Sơ đồ này vẫn tốt hơn trường hợp hệ thống có nhiều máy tính tách biệt bởi vì nó cho phép các CPU có thể chia sẻ một tập các tài nguyên đĩa và các thiết bị nhập/xuất khác, đồng thời nó cũng cho phép bộ nhớ được chia sẻ một cách linh hoạt hơn. Thí dụ, nếu một ngày đẹp trời nào đó, một chương trình có kích thước lớn bất thường cần được thực thi, thì một trong các CPU vẫn có thể được cung cấp một phần bộ nhớ đủ lớn để thực thi chương trình đó. Ngòai ra, các tiến trình còn có thể truyền thông với nhau một cách hiệu quả, chẳng hạn như một *producer* có thể ghi dữ liệu vào bộ nhớ đồng thời một *consumer* lấy dữ liệu đó ra từ nơi mà *producer* ghi vào. Tuy nhiên, thiết kế này vẫn cho thấy một số nhược điểm sau:

Thứ nhất, khi một tiến trình tạo một lời gọi hệ thống, thì lời gọi hệ thống này sẽ được thực thi trên chính CPU của tiến trình đó sử dụng các cấu trúc dữ liệu trong các bảng của cùng hệ điều hành dành CPU đó.

Thứ hai, vì mỗi hệ điều hành đều có một tập các tiến trình được điều phối bởi chính nó. Cho nên, sẽ không có việc chia sẻ tiến trình ở đây. Nếu một user làm việc với CPU 1 thì tất cả các tiến trình của user này chỉ chạy trên CPU 1. Kết quả là, CPU1 quá tải trong khi các CPU khác thì rảnh rỗi.

Thứ ba, không có việc chia sẻ trang nhớ ở đây. Chẳng hạn như, trong khi CPU 1 có nhiều trang nhớ dư thừa, thì CPU 2 vẫn phải thực hiện phân trang liên tục. Không có cách nào để CPU 2 có thể mượn một vài trang nhớ từ CPU 1 bởi vì bộ nhớ đã được chia cố định.

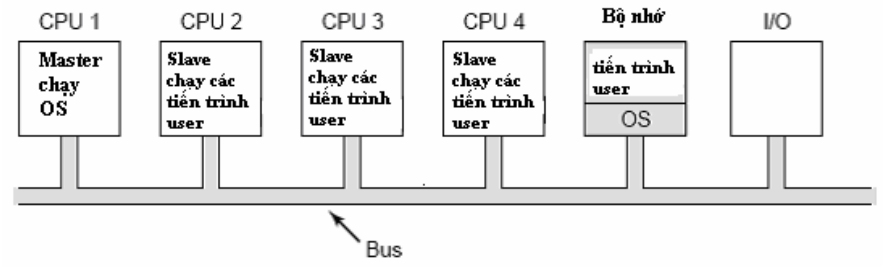
Thứ tư và cũng là nhược điểm lớn nhất. Nếu mỗi hệ điều hành của từng CPU lưu giữ một vùng nhớ *cache* của các khối đĩa mới sử dụng gần đây, thì mỗi hệ điều hành sẽ thao tác trên khối dữ liệu này một cách độc lập với các hệ điều hành khác. Vì vậy, có thể sẽ xảy ra trường hợp là các khối đĩa này trở thành một phần riêng và chỉ bị thay đổi bởi một CPU tương ứng tại một thời điểm. Điều này dẫn đến những kết quả mâu thuẫn nhau. Chỉ có một cách duy nhất để loại bỏ vấn đề này là loại bỏ các vùng nhớ *cache*. Điều này không có gì khó, nhưng vấn đề là nó sẽ làm giảm đáng kể hiệu suất làm việc của hệ thống.

Vì những lý do đó mà mô hình này không còn được sử dụng nữa. Một mô hình thứ hai được đề cập trong phần tiếp theo là hệ điều hành hỗ trợ nhiều bộ xử lý hoạt động theo cơ chế Chủ-Tớ (Master-Slave).

6.2.2. Hệ điều hành cho nhiều bộ xử lý hoạt động theo cơ chế Chủ-Tớ (Master-Slave)

Trong mô hình này, được trình bày trong hình 6-7, một bản sao của hệ điều hành được lưu giữ trên CPU 1, các CPU khác không có tính năng này. Khi đó, tất cả các lời gọi hệ thống đều được chuyển đến CPU 1 để được xử lý ở đây. Ngoài ra, CPU 1 cũng có thể chạy các tiến trình người dùng nếu nó có dư thời gian. Mô hình này được gọi là "Chủ-Tớ" với CPU 1 là "chủ" còn các CPU khác đóng vai trò là "tớ".

Mô hình Chủ-Tớ này giải quyết hầu hết các vấn đề trong mô hình thứ nhất. Có một cấu trúc dữ liệu duy nhất (thí dụ như một danh sách hoặc một tập các danh sách được sắp thứ tự ưu tiên) để theo vết các tiến trình sẵn sàng. Khi một CPU muốn đi vào trạng thái rỗi, nó sẽ yêu cầu hệ điều hành gán cho nó một tiến trình để thực thi. Nếu được gán thì nó tiếp tục làm việc, nếu không nó mới đi vào trạng thái rỗi. Như vậy, sẽ không bao giờ xảy ra trường hợp một CPU rỗi trong khi một CPU khác quá tải. Tương tự thế, các trang nhớ cũng có thể được phân phối cho tất cả các tiến trình một cách linh động. Ngoài ra, mô hình này chỉ hỗ trợ một vùng nhớ *cache* nên sẽ không bao giờ xảy ra việc có những kết quả mâu thuẫn nhau.

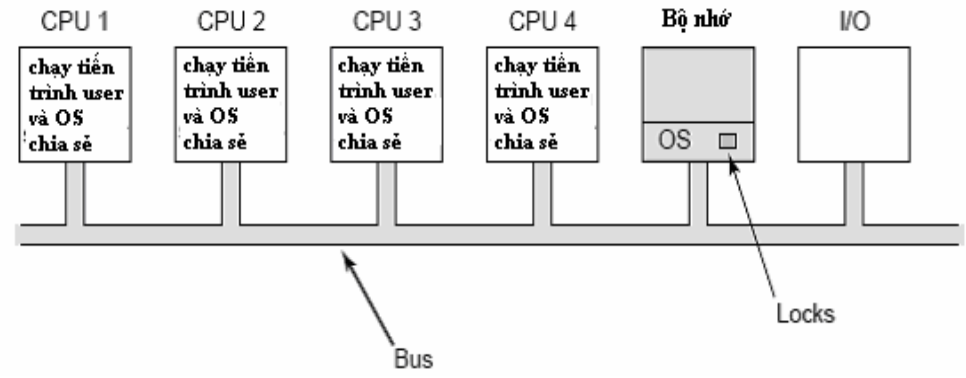


Hình 6.6. Mô hình hệ điều hành Chủ-Tớ trong hệ thống đa xử lý

Vấn đề trong mô hình này là hệ thống có thể xảy ra tình trạng nghẽn cổ chai tại CPU "chủ" nếu có quá nhiều CPU trong hệ thống. Nghĩa là CPU "chủ" phải giải quyết tất cả các lời gọi hệ thống từ các CPU "tớ". Giả sử có 10% tổng thời gian được dùng vào việc xử lý các lời gọi hệ thống thì với hệ thống có 10 CPU nó sẽ làm tràn ngập CPU "chủ". Còn nếu hệ thống có 20 CPU thì nó sẽ bị quá tải. Vì vậy mô hình này là đơn giản và chỉ có thể làm việc được cho các hệ thống đa xử lý với số lượng nhỏ các CPU.

6.2.3. Hệ điều hành cho hệ thống có nhiều bộ xử lý đối xứng (Symmetric multiprocessors)

Trong mô hình này, chỉ có một bản sao của hệ điều hành trong bộ nhớ, nhưng bất kỳ CPU nào cũng có khả năng sử dụng nó. Khi một lời gọi hệ thống được tạo ra cho một CPU nào đó, thì CPU này sẽ thực hiện truy xuất đến *kernel* của hệ điều hành và tiến hành xử lý lời gọi hệ thống đó. Mô hình này được thể hiện trong hình 6-8.



Hình 6.7. Mô hình hệ điều hành cho hệ thống đa xử lý đối xứng

Mô hình này làm cân bằng quá trình xử lý và phân phối bộ nhớ một cách linh hoạt trong hệ thống vì nó chỉ hỗ trợ duy nhất một tập các bảng của hệ điều hành trên cùng một phần cứng. Ngoài ra, nó còn loại bỏ được vấn đề nghẽn cổ chai trong mô hình "chủ-tớ" vì nó không có CPU nào đóng vai trò "chủ" trong hệ thống cả. Tuy nhiên nó vẫn có vấn đề của riêng nó. Nếu có hai hoặc nhiều CPU đang thực thi cùng lúc các đoạn *code* của hệ điều hành, vấn đề nghiêm trọng sẽ xảy ra. Điều gì sẽ xảy ra nếu có hai CPU chọn cùng tiến trình để thực thi và yêu cầu cùng trang nhớ rỗi? Cách

đơn giản nhất để giải quyết vấn đề này là sử dụng biến *mutex* để cho phép khi nào thì một CPU được đi vào miền găng để thực thi tiến trình và sử dụng trang nhớ mà nó cần. Khi một CPU muốn thực thi đoạn *code* của hệ điều hành, trước tiên nó phải giành được *mutex*. Nếu *mutex* bị khóa, nó phải đợi. Theo cách này, bất kỳ CPU nào cũng có thể thực thi *code* của hệ điều hành, nhưng chỉ tại một thời điểm chỉ có một CPU được thực thi.

Tuy nhiên hiệu suất thực hiện của mô hình này cũng không khá hơn mô hình "chủ-tớ" nhiều. Giả sử rằng 10% tổng thời gian được dành cho hệ điều hành xử lý tương tranh, nếu hệ thống hỗ trợ 20 CPU thì cần phải có một hàng đợi CPU khá dài để các CPU lần lượt được phục vụ. Tuy thế, trong mô hình này, điều này được cải thiện dễ dàng hơn. Vì rằng, nhiều phần trong một hệ điều hành là độc lập với một vài phần khác. Chẳng hạn như, sẽ chẳng có vấn đề gì nếu một CPU đang thực thi việc điều phối trong khi một CPU thứ hai khác đang giải quyết một lời gọi hệ thống về tập tin và một CPU thứ ba thì lại đang xử lý vấn đề lỗi trang.

Từ nhận xét này, chúng ta thấy rõ rằng hệ thống có thể được chia thành nhiều miền găng độc lập, các miền găng này không thực hiện bất kỳ một sự tương tác nào với nhau. Mỗi miền găng được bảo vệ bởi một biến *mutex* của nó, và như vậy chỉ có một CPU có thể thực thi *code* của hệ điều hành tại một thời điểm. Tương tự như trong trường hợp một bảng nào đó trong hệ điều hành có thể được sử dụng bởi nhiều miền găng, và mỗi bảng như thế cũng cần có một biến *mutex* của chính nó để quản lý việc tương tranh. Theo cách này, mỗi miền găng có thể được thực thi bởi một CPU tại một thời điểm và mỗi bảng (được bảo vệ bởi miền găng) cũng có thể được truy xuất chỉ bởi một CPU ở một thời điểm.

Hầu hết các hệ thống có nhiều bộ xử lý đều sử dụng mô hình này. Cái khó trong cách tiếp cận này không nằm ở chỗ viết code như thế nào cho khác với một hệ điều hành thông thường trước đây. Mà cái khó ở đây là việc chia nó thành nhiều miền găng để có thể thực thi *code* của hệ điều hành một cách đồng thời bởi nhiều CPU mà không bị ảnh hưởng bởi bất kỳ một CPU nào khác. Bên cạnh đó, tất cả các bảng được sử dụng bởi hai hoặc nhiều miền găng phải được bảo vệ bằng một biến *mutex* và các đoạn *code* đang sử dụng bảng này cũng phải sử dụng biến *muxtex* một cách phù hợp.

Ngoài ra, vấn đề "khóa chết" (deadlocks) cũng phải được quan tâm. Nếu cả hai miền găng đều cần dùng bảng A và bảng B, và một trong hai miền găng này yêu cầu bảng A trước trong khi miền găng kia lại yêu cầu bảng B trước. Vì vậy, trước sau gì thì deadlock cũng sẽ xảy ra và không miền găng nào biết được lý do tại sao. Theo lý thuyết thì tất cả các bảng nên được gán các giá trị nguyên, và các miền găng cũng cần giành được các bảng theo thứ tự tăng dần. Chiến lược này sẽ tránh deadlocks nhưng nó đòi hỏi người lập trình phải suy nghĩ rất cẩn thận để chọn ra bảng nào mà mỗi miền găng cần giành được theo đúng thứ tự. Khi *code* đã được phát triển sau một thời gian thực thi, một miền găng có thể cần một bảng mới mà nó chưa bao giờ cần trước đây. Nếu một lập trình viên chưa có kinh nghiệm và chưa hiểu rỏ về hoạt động logic của hệ thống thì sẽ dễ dàng thực hiện việc chọn ra một *mutex* cho bảng mà miền găng cần và sau đó giải phòng nó đi khi không còn cần đến nữa. Điều này dễ dẫn đến deadlocks. Vì vậy, việc sử dụng biến *mutex* đã không dễ thì việc giữ nó làm việc đúng trong suốt quá trình hoạt động của hệ thống là điều cực hơn đối với các lập trình viên.

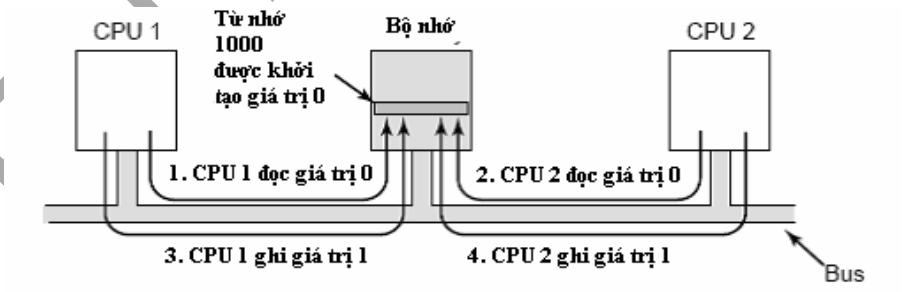
6.3. ĐỒNG BỘ TRONG HỆ THỐNG ĐA XỬ LÝ

Các CPU trong một hệ thống cần được đồng bộ thường xuyên. Chúng ta đã thấy điều này thông qua việc các miền găng và bảng được bảo vệ bởi các biến *mutex*. Bây giờ chúng ta tìm hiểu xem việc đồng bộ thực sự được thực hiện như thế nào trong một hệ thống có nhiều bộ xử lý.

Cũng cần nhắc lại rằng, đồng bộ là tính chất cơ bản trong bất kỳ một hệ thống máy tính nào. Nếu một tiến trình trên một hệ thống có một bộ xử lý tạo một lời gọi hệ thống yêu cầu truy xuất một vài miền găng hay bảng trong *kernel* của hệ điều hành, thì *code* trong *kernel* chỉ có thể thực hiện cấm các ngắt trước khi cho phép truy xuất bảng. Lời gọi hệ thống sau đó thực thi công việc của nó và cũng biết rằng nó có thể sẽ kết thúc mà không có sự truy xuất bảng của bất kỳ tiến trình nào khác. Trên một hệ thống nhiều bộ xử lý, việc cấm ngắt chỉ gây ảnh hưởng đối với một CPU nào đó thôi. Các CPU khác tiếp tục làm việc và có thể thực hiện việc truy xuất bảng trong *kernel* của hệ điều hành. Kết quả là, một giao thức sử dụng *mutex* thích hợp phải được sử dụng bởi tất cả các CPU để đảm bảo rằng việc ngăn chặn truy xuất đồng thời làm việc tốt.

Phần chính của một giao thức dùng biến *mutex* là một chỉ thị lệnh. Chỉ thị này cho phép một từ nhớ được kiểm tra và thiết đặt lại giá trị. Chúng ta đã thấy cách mà giao thức TSL (Test and Set Lock) được sử dụng trong chương 3 để cài đặt các miền găng như thế nào. Như chúng ta đã thấy trước đây, những gì mà một chỉ thị làm đó là đọc một từ nhớ ra và ghi nó vào một thanh ghi. Đồng thời, nó ghi một giá trị 1 (hoặc một giá trị khác 0 nào đó) vào từ nhớ. Tất nhiên, nó phải tốn hai vòng truy xuất *bus* riêng biệt để thực hiện việc đọc và ghi bộ nhớ. Trên một hệ thống có một bộ xử lý, miễn là một chỉ thị không bị ngắt giữa chừng, giao thức TSL luôn làm việc đúng chức năng của nó.

Bây giờ, chúng ta tìm hiểu vấn đề này trên một hệ thống có nhiều bộ xử lý. Trong hình 6-9, từ nhớ 1000 được dùng như một biến *lock* có giá trị khởi tạo là 0. Ở bước 1, CPU 1 đọc ra từ nhớ và lấy được giá trị 0. Ở bước 2, trước khi CPU 1 có cơ hội để ghi ngược lại từ nhớ một giá trị mới mang giá trị 1, thì CPU 2 truy xuất vào và cũng đọc ra từ nhớ có giá trị 0. Ở bước 3, CPU 1 ghi giá trị 1 vào từ nhớ. Bước 4, CPU 2 cũng ghi giá trị 1 vào từ nhớ. Vì cả hai đều lấy giá trị 0 từ chỉ thị TSL, nên cả hai đều có quyền truy xuất đến miền găng và vấn đề giải quyết tương tranh bị thất bại.



Hình 6.8: Chỉ thị lệnh TSL có thể bị lỗi nếu *Bus* không thể bị khóa. Bốn bước trong hình minh họa dãy các sự kiện gây ra lỗi.

Để ngăn chặn vấn đề này, chỉ thị TSL đầu tiên phải khóa *bus*, ngăn chặn các CPU khác truy xuất *bus*. Sau đó mới cho phép cả hai CPU truy xuất đến bộ nhớ. Và sau cùng mới giải phóng *bus*. Chỉ thị này chỉ có thể được cài đặt trên một *bus* có hỗ trợ giao thức phần cứng đặc biệt. Các *bus* hiện

nay đều có khả năng này, còn trước đây, việc cài đặt chỉ thị dạng này là không thực hiện được. Đây cũng là lý do tại sao Peterson đã đưa ra giao thức đồng bộ bằng phần mềm.

Nếu TSL được cài đặt và sử dụng đúng, nó đảm bảo giải quyết tốt vấn đề tương tranh. Tuy nhiên, phương pháp này vẫn có hạn chế nhất định. Vì biến *lock* sẽ bị chiếm dụng theo cơ chế quay tròn (spinning). Nghĩa là các CPU phải lần lượt thực hiện vòng lặp kiểm tra biến này và chiếm dụng *bus* một cách chặt chẽ. Điều này không chỉ làm lãng phí thời gian yêu cầu CPU mà còn gây ra một lượng tải lớn cho *bus* hoặc bộ nhớ, làm giảm nghiêm trọng tốc độ làm việc của các CPU khác.

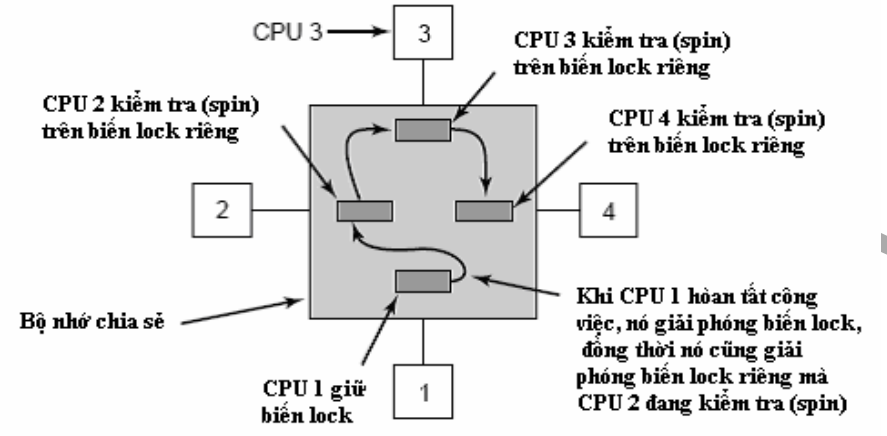
Thoáng qua, ta có thể nghĩ rằng nếu dùng cơ chế *caching* sẽ giải quyết được vấn đề tranh giành *bus*, nhưng thực sự là không. Về lý thuyết, khi CPU cần đọc giá trị của biến *lock*, nó nên có một bản sao trong *cache* của mình. Miễn là không có CPU nào khác cố gắng sử dụng *lock*, CPU sẽ loại bỏ hết *cache* của nó. Khi CPU sở hữu *lock* ghi giá trị 1 vào, cơ chế *caching* sẽ tự động vô hiệu tất cả các bản sao của CPU đó trong các *cache* ở xa, và các *cache* này sẽ nạp lại một giá trị hợp lệ mới. Nhưng vấn đề là ở chỗ, *cache* làm việc với các *block* 32 hoặc 64 byte. Thông thường, các từ nhớ xung quanh *lock* là rất cần cho CPU đang nắm giữ *lock* đó. Vì chỉ thị TSL là một lệnh ghi, nên nó cần thực hiện truy xuất có chọn lọc đến khối cache đang chứa *lock* đó. Bởi vậy, mọi chỉ thị TSL đều làm vô hiệu khối cache bên trong *cache* của CPU đang lưu giữ *lock* đó và lấy ra một bản sao riêng có chọn lọc cho từng yêu cầu của CPU. Ngay khi CPU đang lưu giữ *lock* truy xuất đến một từ nhớ kế bên *lock* đó, khối cache được chuyển đến CPU đó. Kết quả là, toàn bộ khối cache chứa *lock* đó được chuyển qua lại liên tục giữa CPU giữ *lock* và CPU yêu cầu *lock*, tạo ra nhiều lưu lượng bus hơn việc đọc một biến *lock* bình thường.

Nếu chúng ta có thể loại bỏ được tất cả các chỉ thị ghi do TSL đưa ra, thì có thể giảm đáng kể vấn đề *cache thrashing*. Để đạt được điều này, CPU đầu tiên sẽ thực hiện việc đọc chỉ để kiểm tra xem *lock* có rỗi hay không. Chỉ nếu *lock* rỗi, thì CPU mới thực hiện một TSL để thực sự chiếm lấy *lock*. Kết quả của sự thay đổi nhỏ này thể hiện ở chỗ hầu hết các thăm dò (*poll*) bây giờ sẽ chỉ là đọc mà không ghi. Nếu CPU giữ *lock* đang chỉ đọc các biến trong cùng khối cache, thì mỗi CPU có một bản sao của khối cache trong chế độ chỉ đọc, loại bỏ tất cả các chuyển đổi khối cache. Khi *lock* cuối cùng rỗi, CPU sở hữu nó thực hiện chỉ thị ghi, cho phép việc truy xuất dành riêng, vì vậy nó làm vô hiệu tất cả các bản sao khác trong các *cache* ở xa. Trong lần đọc tiếp theo, khối cache sẽ được nạp lại. Nếu hai hoặc nhiều CPU đang cạnh tranh vì muốn chiếm dụng cùng một *lock*, có thể xảy ra trường hợp cả hai CPU sẽ cùng đồng thời thấy *lock* này rỗi, và cả hai thực hiện chỉ thị TSL để chiếm lấy *lock* đó đồng thời. Sẽ chỉ có một trong hai thành công, vì vậy sẽ không có bất kỳ điều kiện cạnh tranh nào ở đây vì việc giành được *lock* được thực hiện bởi chỉ thị TSL và chỉ thị này là không thể chia nhỏ được nữa.

Một cách khác để giảm lưu lượng bus là sử dụng giải thuật *Ethernet binary exponential backoff* do Anderson đưa ra năm 1990. Thay vì phải thăm dò liên tục, một vòng lặp trễ (delay loop) sẽ được thêm vào giữa các biến thăm dò. Ban đầu, *delay* là một chỉ thị lệnh. Nếu *lock* vẫn bận, *delay* được nhân đôi thành hai chỉ thị lệnh, sau đó là 4 chỉ thị lệnh … đến một con số tối đa nào đó. Nếu giá trị tối đa này thấp sẽ cho phép thực hiện hồi đáp nhanh khi *lock* được giải phóng, nhưng nó lại làm lãng phí nhiều vòng bus trong vấn đề *cache thrashing*. Còn nếu giá trị tối đa này cao thì sẽ làm giảm *cache thrashing* nhưng việc thông tin về trạng thái *lock* bị chậm đi. *Binary Exponential Backoff* có thể được sử dụng trong cả hai trường hợp có hoặc không có các chỉ thị đọc kiểm tra trước chỉ thị TSL thực sự.

Một ý tưởng hay hơn đó là cho phép mỗi CPU chiếm lấy một *mutex* (biến lock riêng của nó) để kiểm tra như minh họa trong hình 6-10. Biến *lock* này nên nằm trong một khối cache chưa sử

dụng để tránh xung đột. Khi CPU đang giữ *lock* thóat ra khỏi miền găng, nó giải phóng biến *lock* riêng mà CPU đầu tiên trên danh sách đang kiểm tra (trên *cache* của nó). CPU này sau đó đi vào miền găng. Khi nó được thực thi, nó giải phóng *lock* mà CPU trước nó sử dụng ... Mặc dù giao thức này là tương đối phức tạp (để tránh việc hai CPU cùng đồng thời gắn chúng vào cuối danh sách), nó thật sự là giao thức hiệu quả và giải quyết được vấn đề đói tài nguyên.



Hình 6.9. Sử dụng nhiều biến *lock* để tránh *cache thrashing*

6.4. ĐIỀU PHỐI TRONG HỆ THỐNG ĐA XỬ LÝ

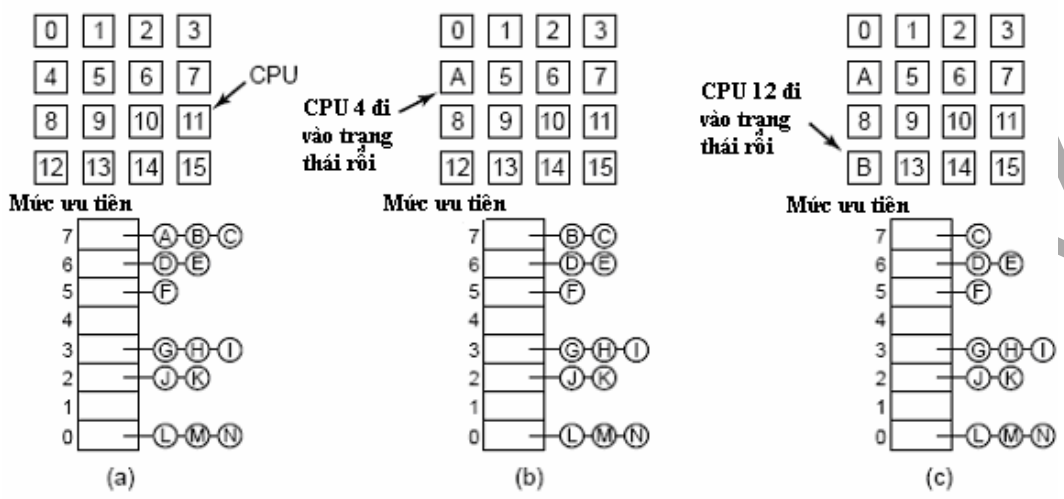
Trên hệ thống có một bộ xử lý, điều phối là một hướng. Chỉ một câu hỏi cần phải trả lời (lặp đi lặp lại) là: ‘‘Tiến trình nào sẽ được chạy tiếp theo?’’. Trên một hệ thống có nhiều bộ xử lý, điều phối là 2 hướng. Bộ điều phối phải quyết định tiến trình nào chạy với CPU nào. Điều này làm cho việc điều phối trên hệ thống có nhiều bộ xử lý phức tạp hơn nhiều.

Một yếu tố phức tạp khác là trong nhiều hệ thống, các tiến trình không liên quan gì với nhau trong khi các tiến trình khác lại thuộc cùng một nhóm. Một ví dụ cho trường hợp đầu là hệ thống chia sẻ thời gian, ở đó, các user khởi tạo các tiến trình một cách độc lập. Các tiến trình không liên quan gì với nhau và mỗi tiến trình có thể được điều phối mà không cần quan tâm đến các tiến trình khác. Một ví dụ cho trường hợp thứ 2 thường xuất hiện trong các môi trường phát triển phần mềm. Các hệ thống lớn thường bao gồm nhiều tập tin header, bao gồm tập tin macro, tập tin định nghĩa, và tập tin khai báo các biến. Các tập tin này sẽ được sử dụng trong các tập tin chính của chương trình. Khi một tập tin header thay đổi, tất cả các tập tin chính có dùng tập tin header này phải được biên dịch lại. Thí dụ, chương trình *make* thường được sử dụng để quản lý những việc dạng này. Khi *make* được gọi, nó bắt đầu kết hợp chỉ những tập tin chính, những tập tin này là cần thiết phải được biên dịch lại do những thay đổi của tập tin header mà chúng sử dụng. Phiên bản cũ của *make* làm việc một cách tuần tự, nhưng phiên bản mới được thiết kế cho hệ điều hành có nhiều bộ xử lý có thể bắt đầu các công việc biên dịch tại cùng một lúc. Nếu có 10 công việc cần biên dịch, nó phải thực hiện nhóm các tiến trình này lại và thực hiện điều phối chúng một cách đồng thời.

6.4.1. Điều phối theo phương pháp chia sẻ thời gian (Time sharing)

Đầu tiên, hãy tìm hiểu việc điều phối trong trường hợp các tiến trình độc lập nhau. Giải thuật điều phối đơn giản nhất để giải quyết các tiến trình (hoặc tiểu trình) độc lập nhau là dùng một cấu trúc dữ liệu cho toàn hệ thống cho các tiến trình sẵn sàng, có thể là một danh sách, nhưng có thể là một tập các danh sách cho các tiến trình với các độ ưu tiên khác nhau như được chỉ ra trong hình 7-

11(a). Ở đây, 16 CPU hiện đang ở trạng thái bận, và một tập được ưu tiên gồm 14 tiến trình đang đợi để chạy. CPU đầu tiên phải kết thúc công việc hiện tại của nó (hoặc tiến trình của nó phải bị khóa) là CPU 4. Sau đó CPU này khóa các hàng đợi điều phối và chọn tiến trình có độ ưu tiên cao nhất, A, như chỉ ra trong hình 7-11(b). Tiếp theo, CPU 12 đi vào trạng thái rỗi và chọn tiến trình B, như minh họa trong hình 7-11(c). Miễn là các tiến trình hoàn toàn độc lập nhau, thì việc điều phối theo kiểu này là một chọn lựa hợp lý.



Hình 6.10. Sử dụng một cấu trúc dữ liệu duy nhất để thực hiện qúa trình điều phối cho hệ thống đa xử lý.

Ưu thế của cách tiếp cận này là sử dụng chung một cấu trúc dữ liệu để điều phối. Điều này cho phép cân bằng tải, nghĩa là sẽ không có trường hợp một CPU nào đó rỗi trong khi các CPU khác thì làm việc quá tải. Hai hạn chế trong trong tiếp cận này là sự cạnh tranh cấu trúc dữ liệu điều phối khi số lượng CPU tăng lên và lượng *overhead* lớn khi thực hiện chuyển đổi ngữ cảnh trong trường hợp một tiến trình bị khóa để đợi thao tác nhập xuất.

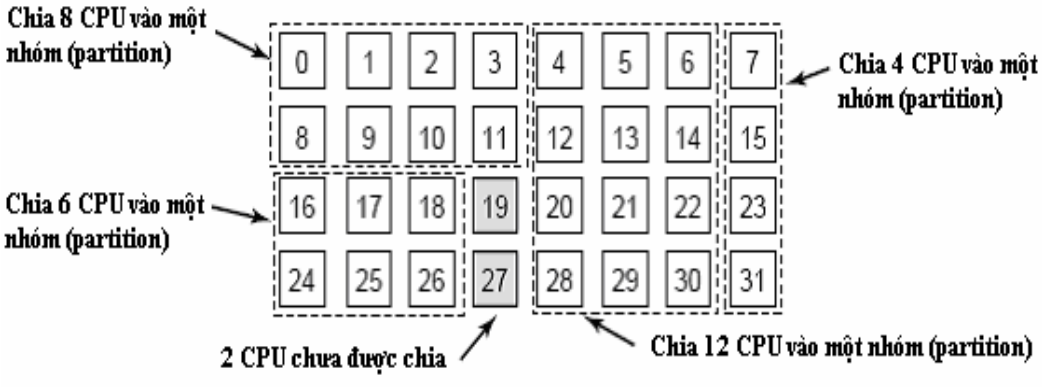
Một vấn đề khác cũng đóng một vai trò quan trọng trong điều phối là trong khi hầu hết các CPU có năng lực như nhau, thì một vài CPU lại có năng lực cao hơn. Đặc biệt, khi tiến trình A đã thực thi trong một thời gian dài trên CPU *k*, *cache* của CPU *k* sẽ chứa đầy các *block* của A. Nếu A tiến hành thực thi trở lại liền, nó có thể sẽ thực hiện tốt hơn nếu chạy với CPU *k*. Có nhiều *block* được nạp trước lên *cache* sẽ giúp hệ thống đáp ứng nhanh hơn cho từng tiến trình. Nhờ vậy, TLB có thể cũng chứa các trang đúng, giảm thiểu các loại lỗi TLB.

Một số hệ thống có nhiều bộ xử lý thực hiện điều phối theo cách gọi là *affinity scheduling*. Ý tưởng chính của cách điều phối này cho phép một tiến trình (chạy trên cùng một CPU mà nó đã chạy trước đó) sử dụng giải thuật điều phối hai mức. Khi một tiến trình được tạo ra, nó được gán cho một CPU. Việc gán các tiến trình cho các CPU được thực hiện ở mức cao. Kết quả là mỗi CPU chiếm giữ một tập các tiến trình của chính nó. Việc điều phối thực sự cho các tiến trình diễn ra ở mức thấp. Điều được thực hiện bởi từng CPU riêng lẻ sử dụng các quyền ưu tiên hoặc dựa vào một vài yếu tố khác. Bằng cách giữ một tiến trình trên cùng một CPU, *cache* được tận dụng tối đa. Ngoài ra, nếu một CPU không có tiến trình nào để thực thi, nó sẽ lấy một tiến trình nào đó từ một CPU khác hơn là phải đi vào trạng thái rỗi. Cơ chế điều phối hai mức này có ba lợi điểm. Đầu tiên, nó phân bố tải đều trên các CPU hiện có. Thứ hai, *cache* được tận dụng tối đa. Thứ ba, bằng cách cho phép mỗi CPU có một danh sách các tiến trình sẵn sàng của chính nó, việc tranh chấp danh sách sẵn sàng được giảm thiểu.

6.4.2. Điều phối theo phương pháp chia sẻ không gian (Space Sharing)

Một cách tiếp cận phổ biến khác cho vấn đề điều phối trên hệ thống đa xử lý khi các tiến trình có liên quan với nhau theo một cách nào đó là *space sharing*. Chẳng hạn chương trình *make* trong ví dụ trên cũng là một trường hợp. Hoặc như, một công việc gồm nhiều tiến trình liên quan với nhau, hoặc một tiến trình gồm nhiều tiểu trình làm những việc tương tự nhau. Việc điều phối cho nhiều tiểu trình ở cùng thời điểm cho nhiều CPU gọi là *space sharing*.

Thuật tóan *space sharing* đơn giản nhất làm việc như sau. Giả sử tất cả các tiểu trình thuộc cùng một nhóm được tạo tại cùng một thời điểm. Vào lúc chúng được tạo, bộ điều phối kiểm tra xem có nhiều CPU rỗi cho các tiểu trình này không! Nếu có, mỗi tiểu trình được cấp cho một CPU tương ứng và tất cả đều bắt đầu. Nếu không đủ CPU, không có tiểu trình nào được thực hiện và chúng phải đợi cho đến khi có đủ CPU cho mọi tiểu trình. Mỗi tiểu trình sẽ nằm giữ CPU của nó cho đến khi kết thúc. Nếu một tiểu trình bị khóa do chờ thao tác nhập/xuất, nó vẫn tiếp tục giữ CPU cho đến khi tiểu trình quay trở lại. Khi có một nhóm các tiểu trình khác xuất hiện, thuật tóan được lặp lại. Tập các CPU được chia cố định thành các nhóm (partition), mỗi nhóm thực thi các tiểu trình của cùng một tiến trình. Trong hình 7-12, có các nhóm gồm 4, 6, 8, 12 CPU và 2 CPU chưa được gán. Sau một thời gian thực thi, kích thước của mỗi nhóm sẽ thay đổi do một vài tiến trình kết thúc và các tiến trình khác phát sinh.



Hình 6.11. Một tập gồm 32 CPU được chia thành 4 nhóm và 2 CPU chưa được chia

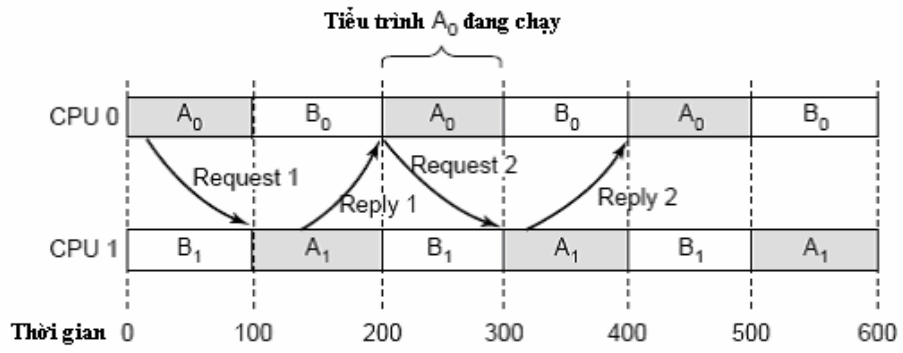
Theo định kỳ, các quyết định điều phối phải được tạo ra. Trong các hệ thống có một bộ xử lý, *shortest-job-first* là thuật tóan được sử dụng nhiều nhất để điều phối cho nhóm. Thuật tóan tương tự trong hệ thống đa xử lý là chọn ra tiến trình đang cần số lượng nhỏ nhất các chu kỳ CPU. Tuy nhiên, trong thực tế, thông tin này ít khi có sẵn. Vì vậy giải thuật này rất khó để thực hiện. Thực vậy, giải thuật *first-come first-served* rất khó được thay thế. Một cách tiếp cận khác hỗ trợ tốt cho cơ chế xử lý song song là dùng một server chính để theo dõi dấu vết của các tiến trình đang chạy, muốn chạy và những yêu cầu về CPU của chúng. Định kỳ, mỗi CPU thăm dò server chính để hỏi xem có bao nhiêu CPU mà nó có thể dùng được. Sau đó, nó sẽ điều chỉnh số lượng tiến trình hoặc tiểu trình để phù hợp với số lượng cho phép. Ví dụ, một Web Server có 1, 2, 5, 10, 20 hoặc một số lượng nào đó các tiểu trình đang chạy song song. Nếu hiện thời nó có 10 tiểu trình đang chạy, và bất ngờ nó cần nhiều CPU hơn nữa, nó sẽ yêu cầu giảm xuống còn 5 tiểu trình (bỏ đi 5 tiểu trình), sau khi 5 tiểu trình này thực hiện xong, nó yêu cầu thóat chứ không tiếp tục công việc mới. Sơ đồ

này cho phép kích thước của mỗi nhóm được thay đổi một cách linh động để phù hợp với công việc hiện hành, hiệu quả hơn hệ thống cố định như cách tiếp cận trong hình 7-12.

6.4.3. Gang Scheduling

Một ưu điểm của *space sharing* là loại bỏ được *overhead* do việc chuyển đổi ngữ cảnh. Tuy nhiên, một nhược điểm của nó là thời gian bị lãng phí khi một CPU bị khóa, nó không có gì để làm cho đến khi nó được giải phóng. Do vậy, nhiều nhà nghiên cứu đã tìm kiếm giải pháp để thực hiện điều phối cho phép kết hợp cả thời gian và không gian), đặc biệt đối với các tiến trình tạo ra nhiều tiểu trình.

Để tìm hiểu vấn đề gì xảy ra khi các tiểu trình của một tiến trình (hoặc các tiến trình của một công việc) được điều phối độc lập, xét một hệ thống với các tiểu trình A0 và A1 thuộc cùng tiến trình A và các tiểu trình B0 và B1 thuộc cùng tiến trình B. Các tiểu trình A0 và B0 được chia sẻ thời gian trên CPU 0; tiểu trình A1 và B1 được chia sẻ thời gian trên CPU 1. Các tiểu trình A0 và A1 cần truyền thông thường xuyên. Giả sử A0 gửi cho A1 một *message*, sau đó A1 gửi ngược lại cho A0 một hồi đáp, công việc này diễn ra lặp đi lặp lại. Giả sử rằng A0 và B1 bắt đầu trước như được chỉ ra trong hình 7-13.



Hình 6.12. Truyền thông giữa hai tiểu trình của cùng một tiến trình A

Trong khe thời gian 0 (0 ms), A0 gửi cho A1 một yêu cầu (request), nhưng A1 không thể nhận được cho đến khi nó chạy ở khe thời gian 1 (100ms). Nó gửi hồi đáp (reply) ngay lặp tức, nhưng A0 không nhận được hồi đáp này liền đến khi nó chạy lại ở 200 ms sau đó. Kết quả là một cặp *request/reply* sẽ được hoàn thành trong vòng 200 ms. Cách tiếp cận này rõ ràng là không hiệu quả.

Giải pháp cho vấn đề này là *gang scheduling*. Cơ chế điều phối này gồm ba phần:

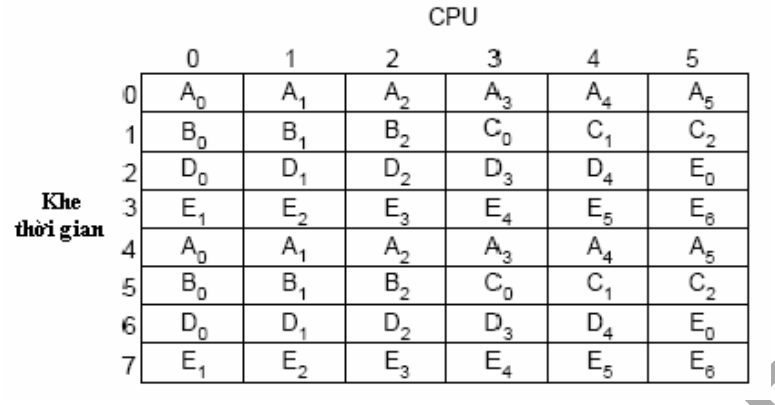
Các tiểu trình liên quan được điều phối như là một nhóm.

Tất cả các thành viên của một nhóm thực thi đồng thời, trên các CPU chia sẻ thời gian.

Tất cả các thành viên bắt đầu và kết thúc các khe thời gian cùng nhau.

Bản chất của *gang scheduling* là cho phép tất cả các CPU được điều phối đồng thời. Nghĩa là, thời gian được chia thành các phần (quantum) rời rạc như trong hình 7-14. Ở đầu mỗi *quantum* mới, tất cả các CPU được điều phối lại, với một tiểu trình mới đang được bắt đầu trên mỗi *quantum*. Bắt đầu của mỗi quantum tiếp theo, một sự kiện điều phối khác xảy ra. Trong khoảng giữa, việc điều phối không được thực hiện. Nếu một tiểu trình bị khóa, CPU của nó trở về trạng thái rỗi cho đến khi kết thúc *quantum*.

Một ví dụ cho thấy *gang scheduling* làm việc như thế nào được minh họa trong hình 7-14. Ở đây, chúng ta có một hệ thống đa xử lý với 6 CPU đang được sử dụng bởi 5 tiến trình từ A đến E với tổng cộng 24 tiểu trình sẵn sàng. Trong suốt khe thời gian 0, các tiểu trình A0 đến A6 được điều phối để chạy. Trong suốt khe thời gian 1, các tiểu trình B0, B1, B2, C0, C1 và C2 được điều phối để chạy. Trong suốt khe thời gian 2, 5 tiểu trình của D và E0 được điều phối để chạy. Sáu tiểu trình còn lại của tiến trình E được điều phối để chạy trong khe thời gian 3. Sau đó chu kỳ được lặp lại, với khe thời gian 4 giống như khe thời gian 0 ...



Hình 6.13. Một ví dụ minh họa cơ chế làm việc của *Gang Scheduling*

Ý tưởng của *gang scheduling* là cho tất cả các tiểu trình của một tiến trình chạy cùng nhau, để nếu một trong chúng gửi yêu cầu cho một tiểu trình khác, nó sẽ nhận được gần như là lập tức và có thể hồi đáp trở lại cũng hầu như tức thời. Trong hình 7-14, vì tất cả các tiểu trình thuộc tiến trình A đang chạy cùng nhau, trong cùng một *quantum*, nên chúng có thể gửi và nhận một số lượng rất lớn các *message* trong cùng *quantum* đó, và vì vậy nó loại bỏ được vấn đề đã chỉ ra trong hình 6-13.

TÓM TẮT

Mặc dù tốc độ máy tính ngày càng được cải thiện nhờ vào các công nghệ mới, nhưng nhu cầu của con người vẫn chưa được thỏa mãn. Một trong các cách tiếp cận nhằm mục đích tăng tốc độ của máy tính đó là sử dụng nhiều bộ xử lý trên một máy. Mỗi bộ xử lý chỉ cần hoạt động ở tốc độ bình thường cũng đã cung cấp cho hệ thống khả năng tính toán tốt hơn rất nhiều so với hệ thống chỉ có một bộ xử lý. Hệ thống có nhiều bộ xử lý có thể chia làm 3 loại chính như sau:

+ Đa xử lý dùng bộ nhớ chia sẻ.

+ Đa xử lý dùng bộ nhớ riêng.

+ Đa xử lý phân tán.

CÂU HỎI VÀ BÀI TẬP

1. Phân biệt giữa đa chương và đa xử lý. Các loại cấu hình đa xử lý: dùng bộ nhớ chia sẻ, dùng bộ nhớ riêng, hệ thống phân tán.

2. Các cấu hình phần cứng của một hệ thống có nhiều bộ xử lý: UMA dùng bus, UMA dùng crossbar switch, UMA dùng multistage switch network.

3. Cho biết ưu và nhược điểm của việc sử dụng bus chung trong hệ thống đa xử lý.

4. Giới hạn của mô hình chuyển mạch chéo (crossbar switch).

5. Ưu và nhược của mô hình mạng chuyển mạch đa tầng (multistage switch network) so với chuyển mạch chép (crossbar switch).

6. Phân biệt hai hệ thống UMA và NUMA. Ưu nhược của từng hệ thống.

7. Các hệ điều hành hỗ trợ cho hệ thống đa xử lý: mỗi CPU có một hệ điều hành, Hệ điều hành cho nhiều bộ xử lý hoạt động theo cơ chế Chủ-Tớ (Master-Slave), Hệ điều hành cho hệ thống có nhiều bộ xử lý đối xứng (Symmetric Multiprocessors). Ưu nhược của từng mô hình.

8. Đồng bộ trong hệ thống đa xử lý có gì khác so với đồng bộ trong hệ thống chỉ có một bộ xử lý?

9. Điều phối trong hệ thống đa xử lý cần quan tâm đến những vấn đề khi so với điều phối trong hệ thống có một bộ xử lý.

10. Các phương pháp điều phối trong hệ thống đa xứ lý: chia sẻ thời gian, chia sẻ không gian, gang scheduling. Đánh giá từng phương pháp.

11. Trong một hệ thống có nhiều bộ xử lý, nếu có hai CPU cố gắng truy xuất cùng một từ nhớ vào cùng một thời điểm. Điều gì sẽ xảy ra?

12. Trên một hệ thống máy tính hoạt động với tốc độ 200 MIPS (triệu lệnh trên giây). Giả sử, trong mỗi lệnh, một CPU đều có nhu cầu truy xuất bộ nhớ. Và mỗi tham chiếu đến bộ nhớ yêu cầu một vòng truy xuất bus. Vậy, sẽ cần bao nhiêu CPU để tiêu tốn toàn bộ 400MHz bus. Câu hỏi tương tự nhưng trong trường hợp có sử dụng cơ chế caching, với tỉ lệ thành công (ti lệ mà các từ nhớ tham chiếu đến tồn tại trong cache) của các cache là 90%. Nếu hệ thống cần cho phép 32 CPU cùng chia sẻ bus (nhưng không làm quá tải bus), thì tỉ lệ thành công sẽ là bao nhiêu?

13. Xét mô hình mạng Omega như trong hình 6-5, nếu đường nối giữa switch 2A và switch 3B bị đứt. Vậy thì những CPU nào sẽ bị ngắt tham chiếu đến các bộ nhớ nào?

14. Trong mô hình "Chủ-Tớ" (tham khảo hình 6-7), việc xử lý tín hiệu được thực hiện như thế nào? (ví dụ như có một tín hiệu được tạo ra từ bàn phím).

15. Trong mô hình đối xứng (tham khảo hình 6-8), khi một lời gọi hệ thống được sinh ra, một vấn đề phải được giải quyết ngay sau khi trap xảy ra (điều này không xảy ra trong hệ mô hình "chủ-tớ"). Bản chất của vấn đề đó là gì? Nó được giải quyết thể nào?

16. Tại sao vấn đề đồng bộ là thực sự cần thiết trong hệ thống có nhiều bộ xử lý?

17. Đối với hệ điều hành hỗ trợ hệ thống đa xử lý đối xứng, các miền găng (trong code của hệ điều hành) hoặc các biến mutex (trong cấu trúc dữ liệu của hệ điều hành) có thực sự cần thiết để tránh việc tương tranh?

18. Khi chỉ thị lênh TSL được sử dụng cho việc đồng bộ trong hệ thống có nhiều bộ xử lý, khối cache đang giữ biến mutex sẽ chuyển đi chuyển về giữa CPU giữ lock và CPU yêu cầu lock nếu cả hai đang tham chiếu đến block đó. Để giảm lưu lượng bus, CPU yêu cầu lock sẽ thực thi một chỉ thị lệnh TSL trong 50 vòng bus, còn CPU giữ lock luôn tham chiếu đến khối cache giữa 2 chỉ thị lệnh liên tiếp. Nếu một khối cache gồm 16 từ nhớ 32-bit, mỗi từ nhớ yêu cầu một vòng bus để truyền, và bus hoạt động ở tốc độ 400 MHz. Tỉ lệ phần trăm của băng thông bus đã bị lãng phí do việc duy chuyển khối cache tới lui là bao nhiêu?

19. Trong cơ chế đồng bộ cho hệ thống có nhiều bộ xử lý, một thuật toán gọi là binary exponential backoff được sử dụng chen vào giữa lần thực hiện cơ chế TSL để thăm dò biến lock. Đồng thời, một tham số khác gọi là độ trễ tối đa (maximum delay) cũng được sử dụng giữa các lần thăm dò. Vậy, thuật tóan này có làm việc đúng không nếu không có tham số độ trễ tối đa này?

20. Giả sử rằng chỉ thị lệnh TSL không còn được sử dụng cho việc đồng bộ trong hệ thống có nhiều bộ xử lý nữa. Thay vào đó, một chỉ thị lệnh khác gọi là SWP sẽ được sử dụng, chỉ thị này tự động trao đổi qua lại nội dung của thanh ghi với từ nhớ trong bộ nhớ. Vậy, chỉ thị SWP này có thể được sử dụng để thực hiện đồng bộ cho hệ thống đa xử lý không? Nếu có, nó được sử dụng như thế nào? Nếu không, vì sao?

21. Giả sử rằng mỗi chỉ thị lệnh được thực thi bởi CPU tốn 5 nsec. Sau khi một chỉ thị lệnh kết thúc, bất kỳ vòng bus nào (cần cho TSL) cũng đều được thực hiện. Mỗi vòng bus tốn thêm 10 nsec. Nếu một tiến trình cố gắng đi vào miền găng sử dụng TSL loop, tỉ lệ phấn trăm của băng thông bus bị tiêu hao sẽ là bao nhiêu? Giả sử rằng cơ chế caching vẫn hoạt động bình thường để cho phép việc nạp chỉ thị lệnh bên trong loop không tốn vòng bus nào.

22. Cơ chế Affinity Scheduling làm giảm cache miss (từ nhớ cần tham chiếu không có sẵn trong cache). Vậy nó có giảm TLB miss không? Và các lỗi trang thì thế nào?

TÀI LIỆU THAM KHẢO

[1]. Gary J. Nutt, University of Colorado. Centralized And Distributed Operating Systems. Second Edition, 2000.

[2]. Robert Switzer. Operating Systems, A Practical Approach. Prentice-Hall International, Inc. 1993.

[3]. Andrew S. Tanenbaum. Modern Operating Systems. Prentice-Hall International, Inc. Second Edition, 2001.

[4]. Abraham Silberschatz & Peter Baer Galvin. Operating System concepts. John Wiley & Sons, Inc. Fifth Edition, 1999.

[5]. H. M. Deitel. Operating Systems. Addison-Wesley Inc. Second Edition, 1999.

[6]. Trần Hạnh Nhi & Lê Khắc Nhiên Ân & Hoàng Kiếm. Giáo trình hệ điều hành (tập 1 & 2). ĐHKHTN 2000.

HƯỚNG DẪN CÂU HỎI VÀ BÀI TẬP

CHƯƠNG 1: GIỚI THIỆU VỀ HỆ ĐIỀU HÀNH

1. Mục đích chung của hệ điều hành

Giúp người dùng sử dụng máy tính đơn giản, hiệu quả.

2. Các thành phần của một hệ thống máy tính

Phần cứng, hệ điều hành, chương trình ứng dụng/chương trình hệ thống, người sử dụng

3. Các thành phần cơ bản của hệ điều hành

Bộ cấp phát tài nguyên, chương trình kiểm soát và phần nhân.

4. Các loại hệ điều hành

Hệ điều hành xử lý theo lô, hệ điều hành xử lý đa chương, hệ điều hành xử lý đa nhiệm, hệ điều hành đa xử lý, hệ điều hành lý phân tán, hệ điều hành xử lý thời gian thực, hệ điều hành nhúng.

5. Phân biệt hệ điều hành đa chương và hệ điều hành đa nhiệm

Giống nhau: tại một thời điểm có thể có nhiều công việc trong bộ nhớ, sử dụng chung một CPU

Khác nhau:

Đa chương: việc chuyển đổi công việc xảy ra khi công việc đang thực thi có yêu cầu nhập/xuất.

Đa nhiệm: việc chuyển đổi công việc xảy ra khi công việc đang thực thi hết thời gian qui định sử dụng CPU hoặc có yêu cầu nhập/xuất.

6. Các vấn đề mà hệ điều hành đa chương/đa nhiệm cần giải quyết

Lập lịch CPU, Quản lý bộ nhớ, Cấp phát thiết bị, Cung cấp các hàm xử lý nhập/xuất.

7. Phân biệt hệ điều hành đa nhiệm và hệ điều hành đa xử lý

Giống nhau: tại một thời điểm có thể có nhiều công việc trong bộ nhớ

Khác nhau:

Đa nhiệm: các công việc dùng chung một CPU, các công việc sẽ thực hiện luân phiên, không đồng thời.

Đa xử lý: máy tính có nhiều CPU, mỗi CPU sẽ thực hiện một công việc, các công việc sẽ thực sự diễn ra đồng thời.

8. Nêu các ưu điểm chính của hệ điều hành đa xử lý

+ Sự hỏng hóc của một bộ xử lý sẽ không ảnh hưởng đến toàn bộ hệ thống.

+ Hệ thống sẽ thực hiện rất nhanh do thực hiện các công việc đồng thời trên các bộ xử lý khác nhau, việc liên lạc giữa các công việc rất dễ dàng bằng cách sử dụng bộ nhớ dùng chung.

9. Nêu các loại hệ thống đa xử lý

Có hai loại hệ thống đa xử lý:

+ Hệ thống đa xử lý đối xứng (Symmetric MultiProcessing (SMP)): mỗi bộ xử lý chạy với một bản sao của hệ điều hành và các bộ xử lý là ngang cấp.

+ Hệ thống đa xử lý bất đối xứng (Asymmetric multiprocessing): Có một bộ xử lý chính (master processor) kiểm soát, phân việc cho các bộ xử lý khác (slave processors).

10. Phân biệt hệ điều hành đa xử lý và hệ điều hành xử lý phân tán

Giống nhau: tại một thời điểm có thể có nhiều công việc trong bộ nhớ, mỗi công việc có thể được thực hiện trên các CPU khác nhau.

Khác nhau:

Đa xử lý: các CPU dùng chung đường truyền dữ liệu, đồng hồ, bộ nhớ và các công việc liên lạc với nhau qua bộ nhớ dùng chung.

Xử lý phân tán: các CPU có đường truyền dữ liệu, đồng hồ, bộ nhớ riêng và các công việc liên lạc với nhau qua đường truyền mạng.

11. Nêu các loại hệ thống xử lý phân tán

Có hai loại: client-server hoặc peer-to-peer.

+ Peer-to-peer: hệ thống mạng ngang hàng, các máy tính ngang cấp, không có máy nào đóng vai trò quản lý tài nguyên dùng chung.

+ Client-server: có một máy đóng vai trò quản lý các tài nguyên dùng chung gọi là máy server (máy chủ), các máy khác gọi là máy client (máy khách). Client muốn sử dụng tài nguyên dùng chung phải được server cấp quyền.

12. Nêu các ưu điểm chính của hệ điều hành xử lý phân tán

+ Chia xẻ tài nguyên : máy in, tập tin …

+ Tăng tốc độ tính toán : phân chia công việc để tính toán trên nhiều vị trí khác nhau

+ An toàn : Nếu một vị trí bị hỏng, các vị trí khác vẫn tiếp tục làm việc.

+ Truyền thông tin dễ dàng: download/upload file, gởi/nhận mail,…

13. Trình bày hệ thống xử lý thời gian thực

Hệ thống phải cho kết quả chính xác trong khoảng thời gian nhanh nhất. Hệ thống thường dùng cho những ứng dụng chuyên dụng như là hệ thống điều khiển trong công nghiệp. Có hai loại hệ thống xử lý thời gian thực :

+ Hệ thống xử lý thời gian thực cứng (Hard real-time): các công việc được hoàn tất đúng thời điểm qui định.

+ Hệ thống xử lý thời gian thực mềm (Soft real-time): mỗi công việc có một độ ưu tiên riêng và sẽ được thi hành theo độ ưu tiên.

14. Trình bày hệ thống nhúng

Hệ điều hành được nhúng trong các thiết bị gia dụng, các máy trò chơi,... Do các thiết bị gia dụng có bộ nhớ ít, bộ xử lý tốc độ thấp, kích thước màn hình nhỏ nên hệ điều hành này cần đơn giản, nhỏ gọn, có tính đặc trưng cho từng thiết bị. Ví dụ hệ điều hành dùng cho máy PDAs (Personal Digital Assistants), Mobil phones,… Hệ thống nhúng còn được gọi là hệ thống cầm tay (Handheld Systems).

15. Nêu các dịch vụ của hệ điều hành

Quản lý tiến trình, Quản lý bộ nhớ chính, Quản lý bộ nhớ phụ, Quản lý hệ thống nhập xuất, Quản lý hệ thống tập tin, Các lời gọi hệ thống , Bảo vệ hệ thống, Hệ thống dòng lệnh, Quản lý mạng.

16. Trình bày khái niệm tiến trình

Tiến trình là một chương trình đang thi hành. Trong bộ nhớ, tại một thời điểm có thể có nhiều tiến trình, một số tiến trình là của hệ điều hành, một số tiến trình là của người sử dụng. Khi tiến trình được tạo ra hoặc đang thi hành sẽ được hệ điều hành cung cấp các tài nguyên như CPU, bộ nhớ, tập tin, các thiết bị nhập/xuất… Khi tiến trình kết thúc, hệ điều hành sẽ thu hồi lại các tài nguyên đã cấp phát. Một tiến trình khi thực thi lại có thể tạo ra các tiến trình con và hình thành cây tiến trình.

17. Trình bày các chức năng của dịch vụ quản lý tiến trình

Tạo và hủy các tiến trình của người sử dụng và của hệ điều hành.

Tạm ngưng và thực hiện lại một tiến trình.

Cung cấp cơ chế đồng bộ các tiến trình.

Cung cấp cơ chế liên lạc giữa các tiến trình.

Cung cấp cơ chế kiểm soát tắc nghẽn.

18. Trình bày các chức năng của dịch vụ quản lý bộ nhớ chính

Lưu giữ thông tin về các vị trí trong bộ nhớ đã sử dụng và tiến trình nào đang sử dụng.

Quyết định tiến trình nào sẽ được nạp vào bộ nhớ chính khi bộ nhớ chính có chỗ trống.

Cấp phát bộ nhớ cho tiến trình và thu hồi bộ nhớ khi tiến trình thực thi xong.

19. Trình bày các chức năng của dịch vụ quản lý bộ nhớ phụ

Quản lý vùng trống trên đĩa

Xác định vị trí cất giữ dữ liệu

Lập lịch cho đĩa

20. Nêu các thành phần của một hệ thống nhập/xuất

Một hệ thống nhập/xuất gồm có các thành phần sau:

Hệ thống bộ nhớ đệm

Các chương trình điều khiển thiết bị.

Các chương trình giao tiếp với các chương trình điều khiển thiết bị .

21. Trình bày các chức năng của dịch vụ quản lý hệ thống tập tin

Hỗ trợ các thao tác trên tập tin và thư mục (tạo/xem/xoá/sao chép/di chuyển/đổi tên).

Ánh xạ tập tin trên hệ thống lưu trữ phụ.

Sao lưu tập tin trên các thiết bị lưu trữ.

22. Trình bày dịch vụ lời gọi hệ thống (ngắt)

Lời gọi hệ thống là lệnh do hệ điều hành cung cấp dùng để giao tiếp giữa tiến trình của người dùng và hệ điều hành, lời gọi hệ thống còn gọi là ngắt. Các lời gọi hệ thống có thể được chia thành các loại như là tập lệnh quản lý tiến trình, tập lệnh quản lý tập tin, tập lệnh quản lý thiết bị, tập lệnh dùng để liên lạc giữa các tiến trình. Mỗi lời gọi hệ thống có một số hiệu duy nhất dùng để phân biệt lời gọi này với lời gọi khác. Các địa chỉ nơi chứa mã lệnh của các ngắt được lưu trong một bảng gọi là bảng vectơ ngắt

23. Trình bày hệ thống thông dịch dòng lệnh

Là tập lệnh cơ bản cùng trình thông dịch lệnh để người sử dụng giao tiếp với hệ điều hành. Các lệnh cơ bản như lệnh quản lý tiến trình, quản lý nhập xuất, quản lý bộ nhớ chính, quản lý bộ nhớ phụ, quản lý tập tin và các lệnh bảo vệ hệ thống… Các lệnh trong hệ thống thông dịch dòng lệnh thực ra sẽ gọi các các lời gọi hệ thống.

24. Nêu các cấu trúc của hệ điều hành

Cấu trúc đơn giản, Cấu trúc phân lớp, Cấu trúc máy ảo, Cấu trúc Client-Server,

25. Nêu các mục tiêu thiết kế hệ điều hành

Dễ viết, dễ sửa lỗi, dễ nâng cấp (nên viết hệ điều hành bằng ngôn ngữ cấp cao vì dễ viết và dễ sửa lỗi hơn là viết bằng ngôn ngữ assembly).

Dễ cài đặt, bảo trì, không có lỗi và hiệu qủa.

Dễ sử dụng, dễ học, an toàn, độ tin cậy cao và thực hiện nhanh.

Có tính khả chuyển cao.

CHƯƠNG 2: QUẢN LÝ NHẬP/XUẤT VÀ QUẢN LÝ TẬP TIN

1. Nêu các loại thiết bị nhập/xuất

Thiết bị khối, Thiết bị tuần tự, Thiết bị khác

2. Trình bày đặc tính của thiết bị nhập/xuất

Tốc độ truyền dữ liệu, công dụng, đơn vị truyền dữ liệu, biểu diễn dữ liệu, tình trạng lỗi.

3. Bộ điều khiển thiết bị nhập/xuất (I/O controller) là gì?

CPU không thể truy xuất trực tiếp thiết bị nhập/xuất mà phải thông qua bộ điều khiển thiết bị, dùng hệ thống đường truyền gọi là bus. Thiết bị và bộ điều khiển phải tuân theo cùng chuẩn giao tiếp như chuẩn ANSI, IEEE hay ISO…

4. Nêu các chương trình thực hiện nhập/xuất

Chương trình nhập/xuất của người dùng, chương trình nhập/xuất độc lập thiết bị, chương trình điều khiển thiết bị.

5. Nêu cách tổ chức hệ thống nhập/xuất

Hệ thống quản lý nhập/xuất được tổ chức thành 5 lớp: tiến trình người dùng, chương trình nhập/xuất độc lập thiết bị, chương trình điều khiển thiết bị, chương trình kiểm soát ngắt, phần cứng. Mỗi lớp có chức năng riêng và có thể giao tiếp với lớp khác.

6. Nêu cơ chế nhập/xuất

Bộ xử lý phát sinh một lệnh I/O đến các thiết bị I/O, sau đó chờ cho đến khi thao tác I/O hoàn tất rồi mới tiếp tục xử lý, hoặc Bộ xử lý phát sinh một lệnh I/O đến các thiết bị I/O, sau đó tiếp tục việc xử lý cho tới khi nhận được một ngắt từ thiết bị I/O báo là đã hoàn tất nhập/xuất, bộ xử lý tạm ngưng việc xử lý hiện tại để chuyển qua xử lý ngắt, hoặc Sử dụng cơ chế DMA.

7. Trình bày cơ chế DMA

Xét quá trình đọc đĩa, CPU gửi cho bộ điều khiển đĩa (disk controller) lệnh đọc đĩa, sau đó CPU tiếp tục xử lý công việc khác. Bộ điều khiển sẽ đọc khối trên đĩa, tiếp theo bộ điều khiển phát ra một ngắt để báo cho CPU biết là thao tác đọc đã hoàn tất. CPU đến lấy dữ liệu trong buffer chuyển vào bộ nhớ, thao tác này làm lãng phí thời gian của CPU. Để tối ưu, bộ điều khiển được cung cấp thêm khả năng truy xuất bộ nhớ trực tiếp (DMA), nghĩa là bộ điều khiển tự chuyển khối đã đọc vào trong bộ nhớ chính.

8. Nêu cơ chế truy xuất đĩa

Di chuyển đầu đọc đến track thích hợp, chờ cho đến khi khối cần đọc đến dưới đầu đọc, chuyển dữ liệu giữa đĩa và bộ nhớ chính. Để giảm thiểu thời gian truy xuất đĩa, hệ điều hành cần đưa ra các thuật toán lập lịch dời đầu đọc sao cho tối ưu

9. Trình bày các thuật toán lập lịch di chuyển đầu đọc

Thuật toán FCFS, SSTF, SCAN, C-SCAN

10. Trình bày hệ số đan xen

Bộ điều khiển đĩa phải thực hiện hai chức năng là đọc/ghi dữ liệu và chuyển dữ liệu vào hệ thống. Để đồng bộ hai chức năng này, các sector được đánh số sao cho các sector có số hiệu liên tiếp nhau không nằm kế bên nhau mà có một khoảng cách, khoảng cách này được xác định bởi quá trình format đĩa và gọi là hệ số đan xen.

11. Nêu khái niệm Ram Disks

Hệ điều hành có thể dùng một phần của bộ nhớ chính để lưu trữ các khối đĩa, phần bộ nhớ này gọi là Ram Disk . Ram Disk cũng được chia làm nhiều khối, mỗi khối có kích thước bằng kích thước của khối trên đĩa. Khi driver nhận được lệnh đọc/ghi khối, sẽ tìm trong bộ nhớ Ram Disk vị trí của khối, và thực hiện việc đọc/ ghi trong đó thay vì từ đĩa . RAM disk có ưu điểm là cho phép truy xuất nhanh, không phải chờ quay hay tìm kiếm, thích hợp cho việc lưu trữ những chương trình hay dữ liệu được truy xuất thường xuyên.

12. Nêu cách tổ chức lưu trữ dữ liệu trên đĩa cứng

Đĩa cứng được định dạng thành các vòng tròn đồng tâm gọi là rãnh (track), mỗi rãnh được chia thành nhiều phần bằng nhau gọi là cung (sector). Một khối (cluster) gồm 1 hoặc nhiều cung và dữ liệu được đọc/ghi theo đơn vị khối.

13. Trình bày khái niệm file

File là một tập hợp các thông tin được đặt tên và lưu trữ trên đĩa. File có thể lưu trữ chương trình hay dữ liệu, file có thể là dãy tuần tự các byte không cấu trúc hoặc có cấu trúc dòng, hoặc cấu trúc mẫu tin có chiều dài cố định hay thay đổi. Cấu trúc file do hệ điều hành hoặc chương trình qui định. File có thể truy xuất tuần tự hoặc truy xuất ngẫu nhiên.

14. Nêu các thuộc tính file

Tên file, kiểu file, vị trí file, kích thước file, ngày giờ tạo file, người tạo file, loai file, bảo vệ file.

15. Trình bày các thao tác trên file

Tạo/xóa/mở/đóng/đọc/ghi/thêm/dời con trỏ file/lấy thuộc tính/đặt thuộc tính/đổi tên file…

16. Thế nào là cấu trúc thư mục?

Là cấu trúc dùng để quản lý tất cả các file trên đĩa. Cấu trúc thư mục cũng được ghi trên đĩa và gồm nhiều mục, mỗi mục lưu thông tin của một file.

17. Nêu các loại cấu trúc thư mục

Thư mục một cấp, thư mục hai cấp, thư mục đa cấp

18. Trình bày khái niệm phân vùng

Hệ điều hành có thể chia đĩa cứng thành nhiều phân vùng hoặc tập hợp nhiều đĩa cứng thành một phân vùng. Mỗi phân vùng sẽ có cấu trúc thư mục riêng để quản lý các tập tin trong phân vùng đó

19. Nêu các thao tác trên thư mục

Create, Delete, Open, Close, Read, Rename, Link, Unlink,…

20. Trình bày bảng thư mục

Là một dạng cài đặt của cấu trúc thư mục. Bảng thư mục có nhiều mục, mỗi mục lưu trữ thông tin của một file, thông tin file gồm thuộc tính của file và địa chỉ trên đĩa của toàn bộ file hoặc số hiệu của khối đầu tiên chứa file hoặc là số I-node của file. Mỗi đĩa có một bảng thư mục gọi là bảng thư mục gốc, cài đặt ở phần đầu của đĩa và có thể có nhiều bảng thư mục con.

21. Nêu cách cài đặt hệ thống quản lý file theo phương pháp cấp phát khối nhớ cho file liên tục

Chỉ cần dùng bảng thư mục, mỗi mục trong bảng thư mục ngoài những thuộc tính thông thường của file, cần có thêm thông tin về số hiệu khối bắt đầu (start) và số khối đã cấp cho file (length).

22. Nêu cách cài đặt hệ thống quản lý file theo phương pháp cấp phát khối nhớ cho file là không liên tục

Sử dụng bảng thư mục và sử dụng thêm một trong các cấu trúc sau: danh sách liên kết/bảng chỉ mục/bảng cấp phát file/cấu trúc I-Nodes. - Cấu trúc danh sách liên kết: mỗi mục trong bảng thư mục chứa số hiệu của khối đầu tiên và số hiệu của khối kết thúc , mỗi khối trên đĩa dành một số byte đầu để lưu số hiệu khối kế tiếp của file, phần còn lại của khối sẽ lưu dữ liệu của file. - Bảng chỉ mục: mỗi file có một bảng chỉ mục chiếm một hoặc vài khối, bảng chỉ mục chứa tất cả các số hiệu khối của một file. Một mục trong bảng thư mục sẽ lưu số hiệu khối chứa bảng chỉ mục của file. - Bảng cấp phát file: nếu mỗi mục trong bảng thư mục chỉ chứa số hiệu của khối đầu tiên, thì số hiệu các khối còn lại của file sẽ được lưu trong một bảng gọi là bảng cấp phát file. - Cấu trúc I-node: mỗi file được quản lý bằng một cấu trúc gọi là cấu trúc I-node, mỗi I-node gồm hai phần: phần thứ nhất lưu trữ thuộc tính file, phần thứ hai gồm 13 phần tử: 10 phần tử đầu chứa 10 số hiệu khối đầu tiên của file, phần tử thứ 11 chứa số hiệu khối chứa bảng single, phần tử thứ 12 chứa số hiệu khối chứa bảng double, phần tử thứ 13 chứa số hiệu khối chứa bảng triple. Trong đó mỗi phần tử của bảng triple chứa số hiệu khối chứa bảng double, mỗi phần tử của bảng double chứa số hiệu khối chứa bảng single và mỗi phần tử của bảng single chứa số hiệu khối dữ liệu tiếp theo cấp cho file.

23. Trình bày phương pháp quản lý các khối trống

Dùng là danh sách liên kết hoặc vector bit. Danh sách liên kết: mỗi nút là một khối chứa một bảng gồm các số hiệu khối trống, phần tử cuối của bảng lưu số hiệu khối tiếp theo trong danh sách. Vector bit: Bit thứ i = 1 là khối thứ i trống, = 0 là đã sử dụng. Vector bit được lưu trên một hoặc nhiều khối đĩa, khi cần sẽ đọc vào bộ nhớ để xử lý nhanh.

24. Nêu phương pháp quản lý các khối hỏng

Có thể dùng phần mềm: dùng một file chứa các danh sách các khối hỏng. Hoặc dùng phần cứng: dùng một sector trên đĩa để lưu giữ danh sách các khối bị hỏng.

CHƯƠNG 3: QUẢN LÝ TIẾN TRÌNH

1. Xét giải pháp phần mềm do Dekker đề nghị để tổ chức truy xuất độc quyền cho hai tiến trình. Hai tiến trình P0, P1 chia sẻ các biến sau :

int turn; // đến phiên i hay j (i,j=0..1)

int flag [2]; // khởi động là FALSE

Cấu trúc tiến trình Pi

```
While (TRUE)
{
        flag[i] = TRUE;
        while (flag[j])
                if (turn == j)
                {
                        flag[i]= FALSE;
                        while (turn == j) ;
                        flag[i]= TRUE;
                }
        critical_section();
        turn= j; flag[i]= FALSE;
        non_critical_section();
}
```

Giải pháp này có thỏa mãn 4 yêu cầu của bài toán miền găng không ?

HD:

```
Cấu trúc tiến trình Pi
While (TRUE)
{
        flag[i] = TRUE;
        while (flag[j])
                if (turn == j)
                {
                        flag[i]= FALSE;
                        while (turn == j) ;
                        flag[i]= TRUE;
                }
        critical_section();
        turn= j; flag[i]= FALSE;
        non_critical_section();
}
```

```
Cấu trúc tiến trình Pj
While (TRUE)
{
        flag[j] = TRUE;
        while (flag[i])
                if (turn == i)
                {
                        flag[j]= FALSE;
                        while (turn == i) ;
                        flag[j]= TRUE;
                }
        critical_section();
        turn= i; flag[j]= FALSE;
        non_critical_section();
}
```

Tại thời điểm Pi kiểm tra flag[j], nếu flag[j]=TRUE thì Pi chờ, nếu flag[j]=FALSE thì Pi sẽ vào miền găng, Pj không thể vào miền găng vì khi đó flag[i]=TRUE. Vậy thoả yêu cầu thứ 1.

Khi Pi ở ngoài miền găng, nếu flag[i]=FALSE thì Pj vào được miền găng, nếu flag[i]=TRUE và turn=i thì Pj gán flag[j]=FLASE và chờ ngoài miền găng, khi đó Pi vào trong miền găng,... Vậy thoả đk 3

Dễ thấy giải pháp thoả đk 2, 4.

2. Xét giải pháp đồng bộ hoá sau:

```
while (TRUE)
```

```
{
        int j = 1-i;
        flag[i]= TRUE; turn = i;
        while (turn == j && flag[j]==TRUE);
        critical-section ();
        flag[i] = FALSE;
        Noncritical-section ();
}
```

Đây có phải là một giải pháp bảo đảm được độc quyền truy xuất không ?

HD:

Pi:	Pj:
while (TRUE)	while (TRUE)
{	{
int j = 1-i;	int i = 1-j;
flag[i]= TRUE; turn = i;//j	flag[j]= TRUE; turn = j;//i
while (turn == j && flag[j]==TRUE);	while (turn == i && flag[i]==TRUE);
critical-section ();	critical-section ();
flag[i] = FALSE;	flag[j] = FALSE;
Noncritical-section ();	Noncritical-section ();
}	}

Giả sử Pi muốn vào miền găng, nó đặt turn=i. Khi kiểm tra điều kiện (turn == j && flag[j]==TRUE), đk sai nhưng chưa kịp vào miền găng thì đến lượt Pj. Pj đặt turn=j và kiểm tra điều kiện (turn == i && flag[i]==TRUE), đk sai, Pj vào miền găng nhưng chưa ra khỏi miền găng thì lại tới lượt Pi, Pi không kiểm tra đk nữa mà vào miền găng, vậy giải pháp trên không đảm bảo được độc quyền truy xuất.

3. Giả sử một máy tính không có lệnh TSL, nhưng có lệnh swap hoán đổi nội dung của hai từ nhớ bằng một thao tác độc quyền :

```
void swap(int &a, int &b)
{
        int temp=a;
        a= b; b= temp;
}
```

Sử dụng lệnh này có thể tổ chức truy xuất độc quyền không ? Nếu có xây dựng cấu trúc chương trình tương ứng.

HD: Dùng biến chung lock, gán trị ban đầu là 0

Chương trình P:

```
int lock=0; //biến dùng chung
```

```
While (1)
{
        int key=1;//bien cục bộ
        do{
                swap(lock,key);
        }while (key);
        critical_section();
        lock=0;
        noncritical_section();
}
```

4. Trong giải pháp peterson bỏ biến turn có được không?

HD: khi đó cấu trúc tiến trình Pi sẽ như sau:

```
while (TRUE)
{
        int j = 1-i; // j là tiến trình còn lại
        flag [i]= TRUE;
        while (flag [j]==TRUE);
        critical_section_Pi ();
        flag [i] = FALSE;
        Noncritical_section_Pi ();
}
```

và xét trường hợp là nếu flag[i]=flag[j]=true thì cả hai tiến trình sẽ đợi vô hạn nên vi phạm đk 3,4

5. Phát triển giải pháp Peterson cho nhiều tiến trình

HD:

Tiến trình Pi sẽ cấp cho một số number[i] là số lớn nhất trong các số đã cấp trước đó +1, tiến trình có số nhỏ nhất sẽ được cho vào miền găng trong lượt kế tiếp. Nếu có hai tiến trình Pi và Pj có cùng số thì xét nếu i<j sẽ chọn Pi. Qui ước: (a,b)<(c,d) nếu a<c hay nếu a=c và b<d

Các biến dùng chung number[i] khởi đầu =0, choosing[i]=false;

Cấu trúc tiến trình Pi:

```
int choosing[n], number[n]; //các biến dùng chung
while (1)
{
        choosing[i]=true;// Pi muốn vào miền găng
        number[i]=max(number[0],…,number[n-1])+1;//đề nghị tt j có number[j] nho nhat vao mg
        choosing[i]=false; //để Pi không đợi chính Pi
        for (j=0;j<n;j++)
        {
```

```
            while (choosing[j]);//nếu có Pj (i!=j)  nào đó muốn vào miền găng thì Pi sẽ chờ
            //neu co Pj <Pi thì Pi chờ
            while ((number[j]!=0) && (number[j],j)<(number[i],i));
        }
        critical_section();
        number[i]=0;
        non_critical_section();
}
```

6. Chứng tỏ rằng nếu các lệnh Down và Up trên semaphore không thực hiện một cách không thể phân chia, thì không thể dùng semaphore để giải quyết bài toán miền găng.

HD:

Xét lại đoạn ct giải quyết miền găng bằng semaphore

```
e(s)= 1;
while (TRUE)
{
        Down(s)
        critical-section ();
        Up(s)
        Noncritical-section ();
}
```

tt1: down(s): e=0

tt2: down(s): e=-1

Khi kiểm tra đk (e<0) thoả nên cả hai cùng chờ ngoài miền găng.

7. Một biến X được chia sẻ bởi hai tiến trình cùng thực hiện đoạn code sau:

```
do{
        X = X +1;
        if ( X == 20) X = 0;
}while ( TRUE );
```

Bắt đầu với giá trị X = 0, chứng tỏ rằng giá trị X có thể vượt quá 20. Cần sửa chữa đoạn chương trình trên như thế nào để bảo đảm X không vượt quá 20 ?

HD:

Giả sử khi X=19

tt1: tăng X lên 20, ngừng

tt2: tăng X lên 21, vượt quá 20.

Cách sửa chữa: Dùng một semaphore s, e(s)=1

```
do
{
```

```
        down(s);
        X = X +1;
        if ( X == 20) X = 0;
        up(s);
}while ( TRUE );
```

8. Xét hai tiến trình xử lý đoạn chương trình sau:

process P1 { A1 ; A2 }

process P2 { B1 ; B2 }

Đồng bộ hoá hoạt động của hai tiến trình này sao cho cả A1 và B1 đều hoàn tất trước khi A2 hay B2 bắt đầu .

HD:

e(s1)=1; e(s2)=1;

P1	P2
{	{
down(s2); A1(); up(s1);	down(s1); B1(); up(s2);
down(s2); A2(); up(s1);	down(s1); B2(); up(s2);
}	}

9. Tổng quát hoá bai 6: cho các tiến trình xử lý đoạn chương trình sau:

process P1 { for (i = 1; i <= 100; i ++) Ai ;}

process P2 { for (j = 1; j <= 100; j ++) Bj ;}

Đồng bộ hoá hoạt động của hai tiến trình này sao cho cả với k bất kỳ (2 <= k <= 100), Ak chỉ có thể bắt đầu khi B(k-1) đã kết thúc, và Bk chỉ có thể bắt đầu khi A(k-1) đã kết thúc.

HD:

e(s1)=1; e(s2)=1;

P1	P2
{	{
for(i=1; i<=100; i++)	for(j=1; j<=100; j++)
{	{
down(s2); Ai(); up(s1);	down(s1); Bj(); up(s2);
}	}
}	}

10. Sử dụng semaphore để viết lại chương trình sau theo mô hình xử lý đồng hành:

w = x1 * x2 (1) v = x3 * x4 (2) y = v * x5 (3) z = v * x6 (4)

y = w * y (5) z = w * z (6) ans = y + z (7)

(x1,x2,x3,x4,x5,x6 là các hằng số)

HD:

semaphore s3=s4=s5=s6=s7=0;

```
P1:{w=x1*x2; up(s5);up(s6);}
P2:{v=x3*x4; up(s3); up(s4);}
P3:{down(s3); y = v * x5; up(s5);}
P4:{down(s4); z = v * x6; up(s6);}
P5:{down(s5); down(s5); y = w * y; up(s7);}
P6:{down(s6);down(s6); z = w * z; up(s7);}
P7:{down(s7);down(s7); ans = y + z;}
```

Viết lại bài 10 dùng monitor

HD:

```
monitor cal
{
        int v,w,y,z,ans;
        int v1,w1,y1,z1,y2,z2;
        condition s3,s4,s5w,s5y,s6w,s6z,s7y,s7z;
        F1(); F2(); F3(); F4(); F5(); F6(); F7();
        void init(){v1=w1=y1=z1=y2=z2=0;}
};
cal::F1()
{
        w=x1*x2; w1=1;
        s5w.signal(); s6w.signal();
}
cal::F2()
{
        v=x3*x4; v1=1;
        s3.signal(); s4.signal();
}
cal::F3()
{
        if (v1==0) s3.wait();
        y = v * x5; y1=1;
        s5y.signal();
}
cal::F4()
{
        if (v1==0) s4.wait();
        z = v * x6; z1=1;
```

```
        s6z.signal();
}
cal::F5()
{
        if (w1==0) s5w.wait();
        if (y1==0) s5y.wait();
        y = w * y; y2=1;
        s7y.signal();
}
cal::F6()
{
        if (w1==0) s6w.wait();
        if (z1==0) s6z.wait();
        z = w * z; z2=1;
        s7z.signal();
}
cal::F7()
{
        if (y2==0) s7y.wait();
        if (z2==0) s7z.wait();
        ans = y + z;
}
```

Các chương trình đồng hành:

P1:{cal.F1();} P2:{cal.F2();} P3:{cal.F3();} P4:{cal.F4();}

P5:{cal.F5();}P6:{cal.F6();} P7:{cal.F7();}

10. Xét hai tiến trình sau:

process A

{ while (1) na = na +1;}

process B

{ while (1) nb = nb +1;}

a) Đồng bộ hoá xử lý của hai tiến trình trên, sao cho tại bất kỳ thời điểm nào cũng có nb < na <= nb +10

b)Nếu giảm điều kiện chỉ là na <= nb +10, giải pháp của bạn sẽ được sửa chữa như thế nào ?

c)Giải pháp của bạn có còn đúng nếu có nhiều tiến trình loại A và B cùng thực hiện?

HD:

a) nb < na <= nb +10

semaphore s1=1,s2=1;

int na,nb;//bien dung chung na,nb được gán tri ban dau thoa dieu kien nb < na <= nb +10

```
process A
{
        while (1)
        {
                down(s1);
                if (na < nb +10) na = na +1;
                up(s1);
        }
}
process B
{
        while (1)
        {
                down(s2);
                if (nb +1< na) nb = nb +1;
                up(s2);
        }
}
```

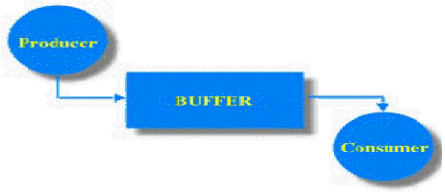
b) na <= nb +10

```
semaphore s1=1,s2=1;
int na,nb;//bien dung chung na,nb gan tri ban dau thoa dieu kien na <= nb +10
process A
{
        while (1)
        {
                down(s1);
                if (na < nb +10) na = na +1;
                up(s1);
        }
}
process B
{
        while (1)
        {
                nb = nb +1;
        }
}
```

c) Đúng

11. Bài toán Người sản xuất – Người tiêu thụ (Producer-Consumer)

Hai tiến trình cùng chia sẻ một bộ đệm có kích thước giới hạn. Một tiến trình tạo dữ liệu, đặt dữ liệu vào bộ đệm (người sản xuất) và một tiến trình lấy dữ liệu từ bộ đệm để xử lý (người tiêu thụ).



Hai tiến trình cần thoả các điều kiện sau :

- ĐK1: Tiến trình sản xuất không được ghi dữ liệu vào bộ đệm đã đầy.
- ĐK2: Tiến trình tiêu thụ không được đọc dữ liệu từ bộ đệm đang trống.
- ĐK3: Hai tiến trình cùng loại hoặc khác loại đều không được truy xuất bộ đệm cùng lúc.

+ Cách 1: dùng sleep and wakeup

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

void producer(void)
{
     int item;

     while (TRUE) {                             /* repeat forever */
          item = produce_item( );               /* generate next item */
          if (count == N) sleep( );             /* 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? */
     }
}


void consumer(void)
{
     int item;

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

Nhận xét: chỉ áp dụng cho một producer và một consumer và chỉ thoả dk1 và dk2. Hãy mở rộng để áp dụng cho nhiều producer, nhiều consumer và thoả cả 3 đk

+ Cách 2: dùng Semaphore

Sử dụng ba semaphore :

- empty: đếm số chỗ còn trống trong bộ đệm. (dùng để giải quyết ĐK1)
- full: đếm số chỗ đã có dữ liệu trong bộ đệm(dùng để giải quyết ĐK2)
- mutex: kiểm tra việc không truy xuất đồng thời.(dùng để giải quyết ĐK3)

```
BufferSize = 3;                    // số chỗ trong bộ đệm
semaphore mutex = 1;       // kiểm soát truy xuất độc quyền buffer (các tt chờ khi mutex=-1)
semaphore empty = BufferSize;   // số chỗ trống (tt sx chờ khi empty<0)
semaphore full = 0;                // số chỗ có dữ liệu (tt tt chờ khi full<0)

Producer() //nhà sản xuất
{
        int item;
        while (TRUE)
        {
                produce_item(item); // tạo dữ liệu mới
                down(empty); // giảm số chỗ trống, đầy thi chờ (empty<0 thì chờ)
                down(mutex); // độc quyền truy xuất buffer (miền găng )
                enter_item(item); // đặt dữ liệu vào bộ đệm
                up(mutex); // ra khỏi miền găng
                up(full); // tăng số chỗ đầy, danh thuc ntt
        }
}
Consumer() //người tiêu thụ
{
        int item;
        while (TRUE)
        {
                down(full); // giảm số chỗ đầy, neu bo dem trong thi cho (full<0 thi chờ)
                down(mutex); // độc quyền vào miền găng
                remove_item(item); // lấy dữ liệu từ bộ đệm
                up(mutex);  // ra khỏi miền găng
                up(empty);  // tăng số chỗ trống, danh thuc nsx
                consume_item(item); // xử lý dữ liệu
        }
}
```

Nhận xét: áp dụng cho nhiều producer và nhiều consumer và thoả cả 3 đk.

Cách 3: dùng Monitor

```
monitor ProducerConsumer //xây dựng monitor
{
   int count=0;  //đếm số chỗ có dữ liệu
   condition full, empty; //nsx chờ trên full, ntt chờ trên empty
```

```
  void enter() //nsx dùng phương thức này để đặt dl vào buffer
  {
        if (count == N)   wait(full);       // nếu bộ đệm đầy, nxs phải chờ (đk1)
        enter_item(item);                   // đặt dữ liệu vào bộ đệm
        count ++;                           // tăng số chỗ đầy
        if (count ==1)  signal(empty);  // nếu bộ đệm có dl thì đánh thức ntt neu co
  }
  void remove() //ntt sẽ dùng phương thức này để lấy dl từ buffer
  {
        if (count == 0)  wait(empty)     // nếu bộ đệm  trống, ntt phải chờ (đk2)
        remove_item(&item);              // lấy dữ liệu từ bộ đệm
        count --;                        // giảm số chỗ có dl
        if (count == N-1) signal(full);  // nếu bộ đệm không đầy thì kích hoạt nsx neu co
  }
}
Producer()       //Tiến trình sản xuất
 {
        while (TRUE)   {
                produce_item(item);  //sx dl
                ProducerConsumer.enter(); //dat dl vao buffer
        }
 }
 Consumer() //tiến trình tiêu thụ
  {
        while (TRUE)   {
                ProducerConsumer.remove();//lay dl tu buffer
                consume_item(item); //tiêu thụ dl
        }
  }
```

Nhận xét:

G/s N=2, và có 2 ntt đang đợi trên empty, khi đó nsx 1 xuất hiện, tăng count=1, đánh thức ntt1, và tiếp đó nsx 2 xuất hiện, tăng count=2, nhưng không đánh thức ntt2, ntt2 đợi mãi…

Do đó cần bỏ đk khi gọi signal() trong enter() và remove. Khi đó sẽ giải quyết triệt để bài toán

12. Bài toán Readers-Writers

Khi cho phép nhiều tiến trình truy xuất cơ sở dữ liệu dùng chung các hệ quản trị CSDL cần đảm bảo các điều kiện sau :

- Đk1: khi có reader thì không có writer nhưng có thể có các reader khác tx dl
- Đk2: Khi có writer thì không có writer hoặc reader nào khác tx dl.

(Các điều kiện này cần có để đảm bảo tính nhất quán của dữ liệu.)

Cách 1: dùng Semaphore

Sử dụng biến chung rc (reader count) để ghi nhớ số tiến trình Reader và sử dụng hai semaphore:

- mutex: kiểm soát sự truy xuất độc quyền rc
- db: kiểm tra sự truy xuất độc quyền đến dữ liệu.

```
int rc=0;                    // Số tiến trình Reader
semaphore mutex = 1;   // Kiểm tra truy xuất rc
semaphore db = 1;        // Kiểm tra truy xuất dữ liệu
Reader()
{
        while (TRUE)
        {
                down(mutex);  // giành quyền truy xuất rc
                rc = rc + 1;       // thêm một tiến trình Reader
        // nếu là Reader đầu tiên thì cấm Writer tx dữ liệu hoặc chờ nếu có writer đang cập nhật dl

                if (rc == 1)  down(db);
                up(mutex);        // chấm dứt truy xuất rc
                read_database(); // đọc dữ liệu
                down(mutex);    // giành quyền truy xuất rc
                rc = rc - 1;        // bớt một tiến trình Reader
                if (rc == 0)   up(db); // nếu là Reader cuối cùng thì cho Writer truy xuất dl
                up(mutex);  // chấm dứt truy xuất rc
                use_data_read();  // su dung du lieu
        }
}
Writer()
{
        while (TRUE)
        {
                create_data();
                down(db); // không có writer hoặc reader nào khác được tx
                //hoac khi co reader đang đọc dl thì writer chờ
                write_database();          // cập nhật dữ liệu
                up(db);  // chấm dứt truy xuất db
        }
```

}

Nhận xét:

Thuật toán chỉ đúng khi có ít nhất một Reader thực thi trước mọi Writer. Nếu không sẽ có nhiều trường hợp sai, ví dụ nếu writer thực hiện trước và đang cập nhật dl thì chỉ cấm được reader 1, không cấm được reader 2.

Thuật toán này gọi là thuật tóan ưu tiên Readers: Reader thực hiện trước writer và Writer phải đợi tất cả các Reader truy xuất xong mới được vào database. Hãy xây dựng thuật toán ưu tiên Writer

Cách 2: Monitor

Sử dụng biến chung rc để ghi nhớ số các tiến trình Reader. Một tiến trình Writer phải chuyển sang trạng thái chờ nếu rc > 0. Khi ra khỏi miền găng, tiến trình Reader cuối cùng sẽ đánh thức tiến trình Writer đang bị khóa.

```
monitor ReaderWriter
{
        int         rc = 0; //khac 0 la co Reader dang doc
        int   busy = 0; //=1 la co writer dang ghi
        condition OKWrite, OKRead;
         void BeginRead() //truoc khi doc, reader goi phương thức nay de kiem tra dk duoc đọc
          {
                if (busy) wait(OKRead);  // nếu có 1 writer đang ghi thì reader chờ trên OKRead
                rc++;                    // thêm một Reader
                signal(OKRead);          //đánh thức một reader chờ trên OKRead
        }
         void FinishRead()
        {
                rc--; // bớt một Reader
                if (rc == 0)  signal(OKWrite);//nếu là Reader cuối cùng thì cho Writer truy xuất db
        }
         void BeginWrite()
        {
                if (busy || rc) // nếu có 1 writer đang ghi, hay có Reader đang đọc
                wait(OKWrite); //thì cho writer chờ trên OKWrite
                busy = 1;          //khong cho cac reader va writer khac truy xuat db
        }
        void FinishWrite()
        {
                busy = 0; // cho cac reader va writer khac truy xuat db
                if (!Empty(OKRead.Queue)) signal(OKRead);//neu co reader dang doi thi cho đọc
```

```
                else    signal(OKWrite); //nguoc lai thi cho một writer ghi
        }
}
Reader()
{
        while (TRUE)
        {
                ReaderWriter.BeginRead(); //bat dau doc
                Read_Database();        //doc dl
                ReaderWriter.FinishRead(); //da doc xong
        }
}
 Writer()
 {
        while (TRUE)
        {
                Create_data(info);      //tao dl
                ReaderWriter.BeginWrite(); //bat dau ghi
                Write_database(info); //ghi
                ReaderWriter.FinishWrite();//da ghi xong
        }
 }
```

Ghi chú:

- OKRead.Queue là hàng đợi của biến điều kiện OKRead.
- Thuật toán vẫn đúng khi Writer thực thi trước Reader.
- Câu hỏi: trong BeginRead () bỏ lệnh signal(OKRead) được không?

13. Bài toán Tạo phân tử H2O

Đồng bộ hoạt động của một phòng thí nghiệm sử dụng nhiều tiến trình đồng hành sau để tạo các phân tử H2O:

```
MakeH()
{
        while (true)
        Make-Hydro(); // tạo 1 nguyên tử H
}
MakeO()
{
        while (true)
```

```
        Make-Oxy(); //tạo 1 nguyên tử O
}
/* Tiến trình MakeWater hoạt động đồng hành  với các tiến trình MakeH, MakeO, chờ có đủ 2 H và 1 O để tạo H2O */
MakeWater()
{
   while (True)
        Make-Water(); //Tạo 1 phân tử H2O
}
```

HD:

```
Semaphore s1=0, s2=0;
MakeH()         // tạo 1 nguyên tử H
{
        while (1)
        {
                Make-Hydro(); up(s1);
        }
}
MakeO()         //tạo 1 nguyên tử O
{
        while(1)
        {
                Make-Oxy(); up(s2);
        }
}
MakeWater()
        {
        while (1)
        {
                down(s1); down(s1); down(s2);
                Make-Water(); //Tạo 1 phân tử H2O
        }
}
```

14. Xét một giải pháp semaphore đúng cho bài toán Dining philosophers :

```
#define N               5
#define LEFT            (i-1)%N
#define RIGHT           (i+1)%N
```

```
#define THINKING    0
#define HUNGRY      1
#define EATING      2
int             state[N];
semaphore       mutex = 1;
semaphore       s[N];//gan tri ban dau =0
//tiến trình mô phỏng triết gia thứ i
void philosopher( int i) // i là triết gia thứ i : 0..N-1
{
        while (TRUE)
        {
                think(); // Suy nghĩ
                take_forks(i); // yêu cầu đến khi có đủ 2 nĩa
                eat(); // yum-yum, spaghetti
                put_forks(i); // đặt cả 2 nĩa lên bàn lại
        }
}
//kiểm tra điều kiện được ăn
void test ( int i) // i là triết gia thứ i : 0..N-1
{
        if(state[i]==HUNGRY && state[LEFT]!=EATING && state[RIGHT]!= EATING)
        {
                state[i] = EATING;
                up(s[i]);
        }
}
//yêu cầu lấy 2 nĩa
void take_forks ( int i) // i là triết gia thứ i : 0..N-1
{
        while (TRUE)
        {
                down(mutex); // vào miền găng
                state[i] = HUNGRY; // ghi nhận triết gia i đã đói
                test(i); // cố gắng lấy 2 nĩa
                up(mutex); // ra khỏi miền găng
                down(s[i]); // chờ nếu không có đủ 2 nĩa
        }
}
```

```
//đặt 2 nĩa xuống
void put_forks ( int i) // i là triết gia thứ i : 0..N-1
{
        while (TRUE)
        {
                down(mutex); // vào miền găng
                state[i] = THINKING; // ghi nhận triết gia i ăn xong
                test(LEFT); // kiểm tra người bên trái đã có thể ăn?
                test(RIGHT); // kiểm tra người bên phải đã có thể ăn?
                up(mutex); // ra khỏi miền găng
        }
}
```

a)Tại sao phải đặt state[i] = HUNGRY trong take_forks ?

b)Giả sử trong put_forks, lệnh gán state[i] = THINKING được thực hiện sau hai lệnh test(LEFT), test(RIGHT). Điều này ảnh hưởng thế nào đến giải pháp cho 3 triết gia? Cho 100 triết gia?

15. Một cửa hiệu cắt tóc có một thợ, một ghế cắt tóc và N ghế cho khách đợi. Nếu không có khách, thợ cắt tóc sẽ ngồi vào ghế cắt tóc và ngủ thiếp đi. Khi một khách hàng vào tiệm, anh ta phải đánh thức người thợ. Nếu một khách hàng vào tiệm khi người thợ đang bận cắt tóc cho khách hàng khác, người mới vào sẽ ngồi chờ nếu có ghế đợi trống, hoặc rời khỏi tiệm nếu đã có N người đợi (hết ghế). Xây dựng một giải pháp với semaphore để thực hiện đồng bộ hoá hoạt động của thợ và khách hàng trong cửa hiệu cắt tóc này.

HD:

```
Semaphore mutex = 1;
Semaphore customers = 0;
Semaphore haircut = 0;
int waiting = 0
void customer() //khách đến cắt tóc
{
        down( mutex );
        if( waiting == N )        //nếu số khách đợi = N thì rời khỏi tiệm
        {
                up( mutex );   return ;
        }
        waiting ++;  //tăng số khách đợi
        up( mutex );
        up(customers);              //đánh thức barber nếu đang ngủ
        down(haircut);  //đang cắt nhưng chưa xong (chờ trên ghế cắt tóc)
```

```
}
void barber() //thợ cắt tóc
{
        while( 1 ) //cắt liên tục, hết khách này đến khách khác
        {
                down( customers );    //nếu không có khách, barber sẽ ngủ
                down( mutex );
                waiting --;  //giảm 1 khách đợi
                up(mutex);
                cut_hair();
                up(haircut);     //cho khách đã cắt tóc xong rời khỏi tiệm
        }
}
```

16. Bài toán Cây cầu cũ

Người ta chỉ có cho phép tối đa 3 xe lưu thông đồng thời qua một cây cầu rất cũ. Hãy xây dựng thủ tục ArriveBridge(int direction) và ExitBridge() để kiểm soát giao thông trên cầu sao cho :

- Tại mỗi thời điểm, chỉ cho phép tối đa 3 xe lưu thông trên cầu.
- Tại mỗi thời điểm, chỉ cho phép tối đa 2 xe lưu thông cùng hướng trên cầu.

Mỗi chiếc xe khi đến đầu cầu sẽ gọi ArriveBridge(direction) để kiểm tra điều kiện lên cầu, và khi đã qua cầu được sẽ gọi ExitBridge() để báo hiệu kết thúc. Giả sử hoạt động của mỗi chiếc xe được mô tả bằng một tiến trình Car() sau đây:

```
Car(int direction) /* direction xác định hướng di chuyển của mỗi chiếc xe.*/
{
        ArriveBridge(direction); //tới cầu
        OnBridge(); //lên cầu
        ExitBridge();// Qua cầu
}
HD:
semaphore ncar=3, ncar1=2, ncar2=2;
ArriveBridge(direction)
{
        down(ncar);
        if (direction==1) down(ncar1);
        else down(ncar2);
}
Exit Bridge(direction)
{
```

```
        up(ncar);
        if (direction==1) up(ncar1);
        else up(ncar2);
}
```

17. Bài toán Qua sông

Để vượt qua sông, các nhân viên Microsof và các Linux hacker cùng sử dụng một bến sông và phải chia sẻ một số thuyền đặc biệt. Mỗi chiếc thuyền này chỉ cho phép chở 1 lần 4 người, và phải có đủ 4 người mới khởi hành được. Để bảo đảm an toàn cho cả 2 phía, cần tuân thủ các luật sau :

a. Không chấp nhận 3 nhân viên Microsoft và 1 Linux hacker trên cùng một chiếc thuyền.

b. Ngược lại, không chấp nhận 3 Linux hacker và 1 nhân viên Microsoft trên cùng một chiếc thuyền.

c. Tất cả các trường hợp kết hợp khác đều hợp pháp.

d. Thuyền chỉ khởi hành khi đã có đủ 4 hành khách.

Cần xây dựng 2 thủ tục HackerArrives() và EmployeeArrives() được gọi tương ứng bởi 1 hacker hoặc 1 nhân viên khi họ đến bờ sông để kiểm tra điều kiện có cho phép họ xuống thuyền không ? Các thủ tục này sẽ sắp xếp những người thích hợp có thể lên thuyền. Những người đã được lên thuyền khi thuyền chưa đầy sẽ phải chờ đến khi người thứ 4 xuống thuyền mới có thể khởi hành qua sông. (Không quan tâm đến số lương thuyền hay việc thuyền qua sông rồi trở lại…Xem như luôn có thuyền để sắp xếp theo các yêu cầu hợp lệ)

Giả sử hoạt động của mỗi hacker được mô tả bằng một tiến trình Hacker() sau đây:

```
Hacker()
{
        RuntoRiver(); // Đi đến bờ sông
        HackerArrives (); // Kiểm tra điều kiện xuống thuyền
        CrossRiver(); // Khởi hành qua sông
}
```

và hoạt động của mỗi nhân viên được mô tả bằng một tiến trình Employee() sau đây:

```
Employee()
{
        RuntoRiver(); // Đi đến bờ sông
        EmployeeArrives (); // Kiểm tra điều kiện xuống thuyền
        CrossRiver(); // Khởi hành qua sông
}
```

HD:

```
int n=0;//so nguoi tren thuyen
semaphore h=2, e=2, s=0;
HackerArrives()
{
        down(h); //cho hacker cho tren bo neu da co 2 hacker tren thuyen
```

```
        down(mutex);
        n++; //cho hacker xuong thuyen
        if (n<4)
        {       up(mutex);
                down(s); //neu thuyen chua du 4 nguoi thi cho tren thuyen
        }
        else //neu du 4 thi danh thuc nhung nguoi dang cho tren thuyen de qua song
        {
                n=0;up(mutex);
                up(s);up(s);up(s);
        }
        up(h); // danh thuc cac hacker cho tren bo cho xuong thuyen
}

EmployeeArrives()
{
        down(e); //cho Employee cho tren bo neu da co 2 Employee tren thuyen
        down(mutex);
        n++; //cho Employee xuong thuyen
        if (n<4)
        {       up(mutex);
                down(s); //neu thuyen chua du 4 nguoi thi cho tren thuyen
        }
        else //neu du 4 thi danh thuc nhung nguoi dang cho tren thuyen de qua song
        {
                n=0; up(mutex);
                up(s);up(s);up(s);
        }
        up(e);// danh thuc cac Employee cho tren bo cho xuong thuyen
}
```

18. Bài toán Điều phối hành khách xe bus tại một trạm dừng

Mỗi xe bus có 10 chỗ, 4 chỗ dành cho khách ngồi xe lăn, 6 chỗ dành cho khách bình thường, khi xe đầy khách thì sẽ khởi hành. Có thể có nhiều xe và nhiều hành khách vào bến cùng lúc, nguyên tắc điều phối sẽ xếp khách vào đầy một xe, cho xe này khởi hành rồi mới điều phối cho xe khác.

Giả sử hoạt động điều phối khách cho 1 chiếc xe bus được mô tả qua tiến trình GetPassengers(); hoạt động của mỗi loại hành khách được mô tả bằng tiến trình WheelPassenger() và NonWheelPassenger(). Hãy sửa chữa các đoạn code, sử dụng semaphore để đồng bộ hoá .

```
GetPassenger() //chương trình điều phối khách cho 1 xe
{
        ArriveTerminal(); // tiếp nhận một xe vào bến
        OpenDoor(); // mở cửa xe
        for (int i=0; i<4; i++) // tiếp nhận các khách ngồi xe lăn
        {
                ArrangeSeat(); // đưa 1 khách ngồi xe lăn vào chỗ
        }
        for (int i=0; i<6; i++) // tiếp nhận các khách bình thường
        {
                ArrangeSeat(); // đưa 1 khách bình thường vào chỗ
        }
        CloseDoor(); // đóng cửa xe
        DepartTerminal(); // cho một xe rời bến
}
WheelPassenger() //chương trình tạo khách ngồi xe lăn
{
        ArriveTerminal(); // đến bến
        GetOnBus(); // lên xe
}
NonWheelPassenger() // chương trình tạo khách bình thường
{
        ArriveTerminal(); // đến bến
        GetOnBus(); // lên xe
}
```

HD:

```
semaphore bus=1,wheel=0, nonwheel=0;
GetPassenger()
{
        down(bus);
        ArriveTerminal(); // tiếp nhận một xe vào bến
        OpenDoor(); // mở cửa xe
        for (int i=0; i<4; i++) // tiếp nhận các hành khách ngồi xe lăn
        {
                down(wheel);
                ArrangeSeat(); // đưa 1 khách xe lăn vào chỗ
        }
        for (int i=0; i<6; i++) // tiếp nhận các hành khách bình thường
```

```
        {
                down(nonwheel);
                ArrangeSeat(); // đưa 1 khách binh thuong vào chỗ
        }
        CloseDoor(); // đóng cửa xe
        DepartTerminal(); // cho một xe rời bến
        up(bus);
}

WheelPassenger()
{
        while(1)
        {
                ArriveTerminal(); // đến bến
                up(wheel);
        }
}
NonWheelPassenger()
{
        while(1)
        {
                ArriveTerminal(); // đến bến
                up(nonwheel);
        }
}
```

NX: khách bình thường nếu tới trước vẫn không được lên xe, phải chờ khách xe lăn

21. Nhà máy sản xuất thiết bị xe hơi, có 2 bộ phận hoạt động song song

- Bộ phận sản xuất 1 khung xe :

```
MakeChassis()
{
        Produce_chassis();// tạo khung xe
}
```

- Bộ phận sản xuất 1 bánh xe :

```
Make_Tires()
{
// tạo bánh xe và gắn vào khung xe
        Produce_tire();
```

```
        Put_tire_to_Chassis();
}
```

Hãy đồng bộ hoạt động trong việc sản xuất xe hơi theo nguyên tắc sau :

- Sản xuất một khung xe, trước khi tạo bánh xe.
- Cần có đủ 4 bánh xe cho 1 khung xe được sản xuất ra, sau đó mới tiếp tục sản xuất khung xe khác…

HD:

```
semaphore chassis=0,tire=0;
MakeChassis()
{
        while(1)
        {
                // tạo khung xe
                Produce_chassis(); up(chassis);
                down(tire); down(tire); down(tire); down(tire);
        }
}
MakeTires()
{
        while(1)
        {
                down(chassis);
                for (i=0;i<4;i++)
                {
                        Produce_tire();
                        Put_tire_to_Chassis(); up(tire);
                }
        }
}
```

19. Thuật toán các triết gia ăn tối sau đúng hay sai?

```
semaphore s[5]; //có các giá ban trị đầu bằng 1
pi
{
        down(s[i]);             //lấy đũa
        down(s[(i+1)%5]);       //lấy đũa của người bên cạnh
        eat();
```

```
        up(s[i]);               //bỏ đũa
        up(s[(i+1)%5]);         //trả đũa cho người bên cạnh
}
```

HD:
nếu tất cả Pi đều thực hiện được một lệnh đầu tiên down(s[i]), và sau đó các Pi đều thực hiện lệnh thứ 2, khi đó tất cả s[i]<0 nên các Pi sẽ chờ nhau vô hạn!

20. Xét trạng thái hệ thống:

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

Nếu tiến trình P2 yêu cầu 4 cho R1, 1 cho R3. hãy cho biết yêu cầu này có thể đáp ứng mà bảo đảm không xảy ra tình trạng deadlock hay không ?

21. Xét trạng thái hệ thống sau:

	Max				Allocation				Available			
	A	B	C	D	A	B	C	D	A	B	C	D
P1	0	0	1	2	0	0	1	2	1	5	2	0
P2	1	7	5	0	1	0	0	0				
P3	2	3	5	6	1	3	5	4				
P4	0	6	5	2	0	6	3	2				
P5	0	6	5	6	0	0	1	4				

a) Cho biết nội dung của bảng Need.

b) Hệ thông có ở trạng thái an toàn không?

c) Nếu tiến trình P2 có yêu cầu tài nguyên (0,4,2,0), yêu cầu này có được đáp ứng tức thời không?

CHƯƠNG 4: QUẢN LÝ BỘ NHỚ

1. Giả sử có một hệ thống sử dụng kỹ thuật phân trang theo yêu cầu. Bảng trang được lưu trữ trong các thanh ghi. Để xử lý một lỗi trang tốn 8 miliseconds nếu có sẵn một khung trang trống, hoặc trang bị thay thế không bị sửa đổi nội dung, và tốn 20 miliseconds nếu trang bị thay thế bị sửa đổi nội dung. Mỗi truy xuất bộ nhớ tốn 100 nanoseconds. Giả sử trang bị thay thế có xác suất bị sửa đổi là 70%. Tỷ lệ phát sinh lỗi trang phải là bao nhiêu để có thể duy trì thời gian truy xuất bộ nhớ (effective acess time) không vượt quá 200 nanoseconds?

HD:

ma=100 nanoseconds

EAT = (1-p)ma + p (page fault time) = (1-p)100+p(20*0.7+0.3*8)*1000000 <=200

=>p<= a

2. Xét chương trình C sau :

```
int A [100][100] ;
for (i=0; i<100; i++)
for (j=0; j<100; j++) A[i][j]= 0;
```

Giả sử tiến trình được cấp 3 khung trang với kích thước một khung trang là 200 bytes, mã tiến trình luôn chiếm khung trang 1, khung trang 2 và 3 để lưu mảng A và khởi đầu khung 2, 3 là rỗng. Hỏi tiến trình có bao nhiêu lỗi trang khi sử dụng thuật toán thay thế LRU.

Xét chương trình C sau với câu hỏi tương tự câu a

```
int A [100][100] ;
for (j=0; j<100; j++)
for (i=0; i<100; i++) A[i][j]= 0;
```

HD:

a) mỗi dòng một trang nên có 100 lỗi

b) 10000 lỗi

3. Trong một hệ thống sử dụng kỹ thuật phân trang theo yêu cầu, kích thước mỗi trang là 2K , xét đoạn chương trình C sau đây:

```
int n = 3*1024; int A[n], B[n];
for (i=0; i<n;i++) A[i]=i;
for (i=0 ;i<n;i++) B[A[i]]=i;
```

a) Nếu số khung cấp cho tiến trình là không hạn chế và giả sử khung trang thứ nhất luôn dùng để chứa tiến trình, các khung trang còn lại được khởi động ở trạng thái trống thì tiến trình có bao nhiêu lỗi trang.

b) Nếu số khung cấp cho tiến trình là 2 khung và giả sử khung trang thứ nhất luôn dùng để chứa tiến trình, khung trang thứ hai được khởi động ở trạng thái trống thì tiến trình có bao nhiêu lỗi trang.

HD:

a) Mảng A có kích thước là 6K => cần 3 trang, mỗi trang chứa 1024 phần tử liên tiếp.

Cứ truy xuất 1024 pt của mảng A sẽ sinh ra một lỗi trang => vòng for thứ 1 sinh ra 3 lỗi trang. Lý luận tương tự vòng for thứ 2 sinh ra 3 lỗi trang. Vậy tiến trình có 6 lỗi trang

b) Số lỗi trang là: 3+2*3*1024= 6147 lỗi trang

4. Một máy tính có 4 khung trang. Thời điểm nạp, thời điểm truy cập cuối cùng, và các bit Reference (R), Dirty (D) của mỗi trang trong bộ nhớ được cho trong bảng sau :

Trang	Thời điểm nạp	Thời điểm truy cập cuối cùng	R	D
0	126	279	0	0
1	230	260	1	0
2	120	272	1	1
3	160	280	1	1

Trang nào sẽ được chọn thay thế theo :

a) thuật toán NRU

b) thuật toán FIFO

c) thuật toán LRU

d) thuật toán " cơ hội thứ 2"

HD:

trang 0

trang 2

trang 1

trang 0

5. Tính kích thước dãy bít dùng để quản lý RAM 512 MB, giả sử địa chỉ đánh theo byte.

HD: 512 MB=512*1024*1024 bytes = 229 bytes. Mỗi một byte dùng 1 bit để quản lý => kích thước dãy bít sử dụng là 229/8/1024/1024 MB = 64 MB.

6. Xét một không gian địa chỉ có 8 trang, mỗi trang có kích thước 1K, ánh xạ vào bộ nhớ vật lý có 32 khung trang.

a) Địa chỉ logic gồm bao nhiêu bit ?

b) Địa chỉ physic gồm bao nhiêu bit ?

HD:

a) 8 trang => p= 3 bit. Kích thước trang 1 K =>d=10 bit=> đc logic= p+d=13 bit

b) đc vật lý có 32 khung => đcvl=32 K= 215 bytes => dcvl=15 bit

7. Xét một hệ thống sử dụng kỹ thuật phân trang, với bảng trang được lưu trữ trong bộ nhớ chính.

a) Nếu thời gian cho một lần truy xuất bộ nhớ bình thường là 200 nanoseconds, thì mất bao nhiêu thời gian cho một thao tác truy xuất bộ nhớ trong hệ thống này ?

b) Nếu sử dụng TLBs với tỉ lệ tìm thấy (hit-ratio) là 75%, thời gian để tìm trong TLBs xem như bằng 0, tính thời gian truy xuất bộ nhớ trong hệ thống (effective memory reference time)

HD:

a) 200+200=400 nanoseconds

b) 200*0.25+200= 250 nanoseconds

8. Xét bảng phân đoạn sau:

Segment	Base	Length
1	2300	14
2	90	100

Cho biết địa chỉ vật lý tương ứng với các địa chỉ ảo sau đây :

a. (1,10)

b. (2,500)

HD: đc logic=(s,d)

s=1, d=10, base(1)=2300 => đcvl=2300+10=2310

s=2, d=500, d>length(2)=100 => lỗi truy xuất đc không hợp lệ

9. Một máy tính 32-bit địa chỉ, sử dụng một bảng trang nhị cấp. Địa chỉ ảo được phân bổ như sau: 9 bit dành cho bảng trang cấp 1, 11 bit cho bảng trang cấp 2, còn lại dành cho offset. Cho biết kích thước một trang trong hệ thống, và không gian địa chỉ ảo có bao nhiêu trang ?

HD:

Đc ảo = (p1,p2,d) =32 bit

p1=9, p2=11 => d=12 bit => kt 1 trang= 212 bytes = 4 KB

Số trang trong kg đc ảo = 29 x 211 = 220 trang

10. Một máy tính có 48-bit địa chỉ ảo, và 32-bit địa chỉ vật lý, kích thước một trang là 8K. Có bao nhiêu phần tử trong một bảng trang thông thường và trong bảng trang nghịch đảo?

HD:

a) Bảng trang thông thường

kt trang = 8 K = 213 bytes => d=13 bit => p=35 bit

=> một bảng trang thông thường có 235 phần tử

b) Bảng trang nghịch đảo

1 khung trang = 8K =213 bytes

ktbnvl = 232 bytes => số khung trang của bnvl= 232/213 = 219 khung

=> số phần tử trong bảng trang nghịch đảo là 219

11. Giả sử có một máy tính đồ chơi sử dụng 7-bit địa chỉ, hệ thống sử dụng một bảng trang nhị cấp, dùng 2-bit làm chỉ mục đến bảng trang cấp 1, 2-bit làm chỉ mục đến bảng trang cấp 2. Xét một tiến trình sử dụng các địa chỉ ảo trong những phạm vi sau : 0..15, 21..29, 94..106, và 115..127.

a) Vẽ chi tiết toàn bộ bảng trang cho tiến trình này

b) Phải cấp phát cho tiến trình bao nhiêu khung trang, giả sử tất cả đều nằm trong bộ nhớ chính?

c) Bao nhiêu bytes ứng với các vùng phân mảnh nội vi trong tiến trình này?

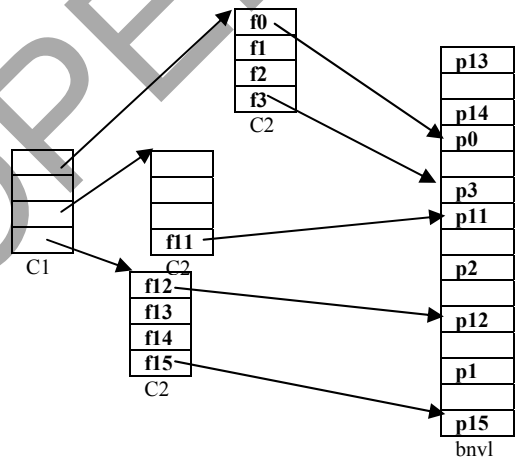
d) Cần bao nhiêu bộ nhớ cho bảng trang của tiến trình này?

HD:

đc ảo = (p1,p2,d)=7 bit

p1=p2=2 bit=> d=3 bit => kt trang=8 bytes

a) vẽ bảng trang : gọi fi là số hiệu khung trang chứa trang pi



0	00.00.000	(0,0,0)
15	00.01.111	(0,1,7)
21	00.10.101	(0,2,5)
29	00.11.101	(0,3,5)
94	10.11.110	(2,3,6)
106	11.01.010	(3,1,2)
115	11.10.011	(3,2,3)
127	11.11.111	(3,3,7)

Tiến trình sử dụng các địa chỉ ảo trong các phạm vi sau:
(0,0,0)..(0,1,7); (0,2,5)..(0,3,5);
(2,3,6)..(3,1,2); (3,2,3)..(3,3,7)

b) cấp 9 khung + 4 =13 khung

c) tính phân mảnh nội vi:

khung trang cấp cho trang 0 và trang 1 được sử dụng hết

khung trang cấp cho trang 2 dư 21-16 byte=5 byte đầu

tương tự tính phân mảnh cho các trang còn lại

kích thước bảng trang cho tiến trình này

d) dùng 1 bảng trang cấp 1, 3 bảng trang cấp 2 => kích thước bảng trang = 4 khung trang = 32 bytes

12. Giả sử có một máy tính sử dụng 16-bit địa chỉ. Bộ nhớ ảo được thực hiện với kỹ thuật phân đoạn kết hợp phân trang, kích thước tối đa của một phân đoạn là 4096 bytes. Bộ nhớ vật lý được phân thành các khung trang có kích thước 512 bytes.

a) Thể hiện cách địa chỉ ảo được phân tích để phản ánh segment, page, offset

b) Xét một tiến trình sử dụng các miền địa chỉ sau, xác định số hiệu segment và số hiệu page tương ứng trong segment mà chương trình truy cập đến :

350..1039, 3046..3904, 7100..9450, 33056..39200, 61230..63500

c) Bao nhiêu bytes ứng với các vùng phân mảnh nội vi trong tiến trình này?

d) Cần bao nhiêu bộ nhớ cho bảng phân đoạn và bảng trang của tiến trình này ?

HD:

a) đc ảo= (s,d)=(s,p,d')=16 bit, với d=p+d'

kt khung trang=512 => d'=9 bit

kích thước tối đa của 1 phân đoạn là 4096 bytes => một phân đoạn được chia tối đa thành 4096/512 = 8 trang =>p=3 bit =>d=12 bit => s=4 bit => dc ảo=(s,p,d')=(4,3,9)

b) Tiến trình sử dụng các miền địa chỉ ảo sau:

350	0000.000.101011110	(0,0,350)
1039	0000.010.000001111	(0,2,15)
3046	0000.101.111100110	(0,5,486)
3904	0000.111.101000000	(0,7,320)
7100	0001.101.110111100	(1,5,444)
9450	0010.010.011101010	(2,2,234)
33056	1000.000.100100000	(8,0,288)
39200	1001.100.100100000	(9,4,288)
61230	1110.111.100101110	(14,7,302)
63500	1111.100.000001100	(15,4,12)

(0,0,350)..(0,2,15);(0,5,486)..(0,7,320);
(1,5,444)..(2,2,234);
(8,0,288)..(9,4,288); (14,7,302)..(15,4,12)
=>
s=0: p=0,1,2,5,6,7
s=1: p=5,6,7
s=2: p=0,1,2
s=8: p=0,1,2,3,4,5,6,7
s=9: p=0,1,2,3,4
s=14: p=7
s=15: p=0,1,2,3,4

c) tính phân mảnh nội vi:

s=0,p=0 dư: 350 byte đầu

s=0, p=2 dư: 512-15=497 byte cuối

vv…

d) Tiến trình dùng 1 bảng phân đoạn => bộ nhớ dành cho bảng phân đoạn= 512 byte.

Tiến trình có 16 phân đoạn =>số bảng trang tiến trình sử dụng là 16=> bộ nhớ dành cho bảng trang= 16*512 (giả sử một bảng trang chiếm 1 khung trang)

TÀI LIỆU THAM KHẢO

[1]. Gary J. Nutt, University of Colorado. Centralized And Distributed Operating Systems. Second Edition, 2000.

[2]. Robert Switzer. Operating Systems, A Practical Approach. Prentice-Hall International, Inc. 1993.

[3]. Andrew S. Tanenbaum. Modern Operating Systems. Prentice-Hall International, Inc. Second Edition, 2001.

[4]. Abraham Silberschatz & Peter Baer Galvin. Operating System concepts. John Wiley & Sons, Inc. Fifth Edition, 1999.

[5]. H. M. Deitel. Operating Systems. Addison-Wesley Inc. Second Edition, 1999.

[6]. Trần Hạnh Nhi & Lê Khắc Nhiên Ân & Hoàng Kiếm. Giáo trình hệ điều hành (tập 1 & 2). ĐHKHTN 2000.

MỤC LỤC

- HẾT -