

# BÀI 1 TỔNG QUAN VỀ HỆ ĐIỀU HÀNH

Bài học này cung cấp cho chúng ta một cái nhìn tổng quát về những nguyên lý cơ bản của hệ điều hành. Chúng ta bắt đầu với việc xem xét mục tiêu và các chức năng của hệ điều hành, sau đó khảo sát các dạng khác nhau của chúng cũng như xem xét quá trình phát triển qua từng giai đoạn. Các phần này được trình bày thông qua các nội dung như sau:

- Khái niệm về hệ điều hành
- Phân loại hệ điều hành
- Cấu trúc của hệ điều hành
- Lịch sử phát triển của hệ điều hành

Bài học này giúp chúng ta hiểu được hệ điều hành là gì, có cấu trúc ra sao. Hệ điều hành được phân loại theo những tiêu chuẩn nào. Quá trình phát triển của hệ điều hành phụ thuộc vào những yếu tố nào. Bài học này đòi hỏi những kiến thức về : kiến trúc máy tính.

.....

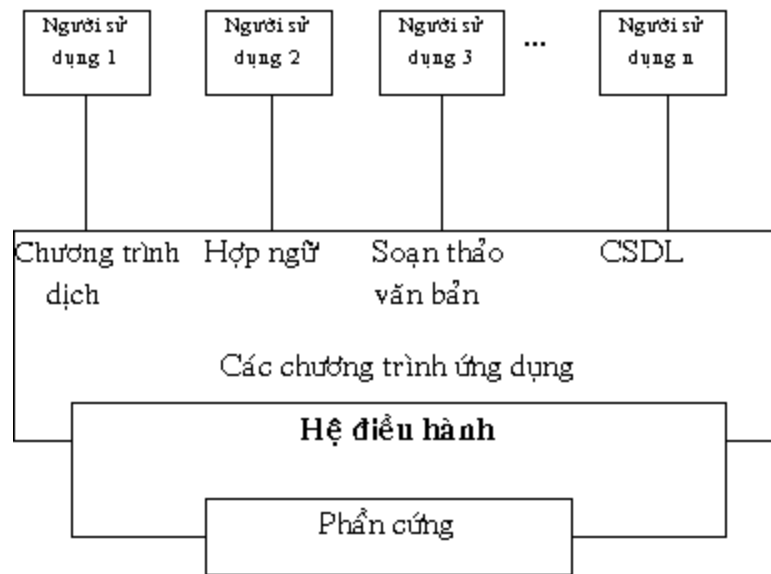
## I.KHÁI NIỆM VỀ HỆ ĐIỀU HÀNH

**Hệ điều hành** là một *chương trình* hay một *hệ chương trình* hoạt động giữa người sử dụng (user) và phần cứng của máy tính. Mục tiêu của hệ điều hành là cung cấp một môi trường để người sử dụng có thể thi hành các chương trình. Nó làm cho máy tính dễ sử dụng hơn, thuận lợi hơn và hiệu quả hơn.

Hệ điều hành là một phần quan trọng của hầu hết các hệ thống máy tính. Một hệ thống máy tính thường được chia làm bốn phần chính : phần cứng, hệ điều hành, các chương trình ứng dụng và người sử dụng.

**Phần cứng** bao gồm CPU, bộ nhớ, các thiết bị nhập xuất, đây là những tài nguyên của máy tính.

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



**Hình 1.1** Mô hình trừu tượng của hệ thống máy tính

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

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

## II. PHÂN LOẠI HỆ ĐIỀU HÀNH

### II.1 Hệ thống xử lý theo lô

#### • Bộ giám sát thường trực :

Khi một công việc chấm dứt, hệ thống sẽ thực hiện công việc kế tiếp mà không cần sự can thiệp của người lập trình, do đó thời gian thực hiện sẽ mau hơn. Một chương trình, còn gọi là bộ giám sát thường trực được thiết kế để giám sát việc thực hiện dãy các công việc một cách tự động, chương trình này luôn luôn thường trú trong bộ nhớ chính.

**Hệ điều hành theo lô** thực hiện các công việc lần lượt theo những chỉ thị định trước.

#### • CPU và thao tác nhập xuất :

CPU thường hay nhàn rỗi do tốc độ làm việc của các thiết bị nhập xuất (thường là thiết bị cơ) chậm hơn rất nhiều lần so với các thiết bị điện tử. Cho dù là một CPU chậm nhất, nó cũng nhanh hơn rất nhiều lần

so với thiết bị nhập xuất. Do đó phải có các phương pháp để đồng bộ hóa việc hoạt động của CPU và thao tác nhập xuất.

- **Xử lý off\_line :**

Xử lý off\_line là thay vì CPU phải đọc trực tiếp từ thiết bị nhập và xuất ra thiết bị xuất, hệ thống dùng một *bộ lưu trữ trung gian*. CPU chỉ thao tác với bộ phận này. Việc đọc hay xuất đều đến và từ bộ lưu trữ trung gian.

- **Spooling :**

*Spool (simultaneous peripheral operation on-line)* là đồng bộ hóa các thao tác bên ngoài on-line. Cơ chế này cho phép xử lý của CPU là on-line, sử dụng đĩa để lưu các dữ liệu nhập cũng như xuất.

## II.2 Hệ thống xử lý theo lô đa chương

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

Ý tưởng như sau : hệ điều hành lưu giữ một phần của các công việc ở nơi lưu trữ trong bộ nhớ . CPU sẽ lần lượt thực hiện các phần công việc này. Khi đang thực hiện, nếu có yêu cầu truy xuất thiết bị thì CPU không nghỉ mà thực hiện tiếp công việc thứ hai...

Với hệ đa chương hệ điều hành ra quyết định cho người sử dụng vì vậy, ***hệ điều hành đa chương*** rất tinh vi. Hệ phải xử lý các vấn đề lập lịch cho công việc, lập lịch cho bộ nhớ và cho cả CPU nữa.

## II.3 Hệ thống chia xẻ thời gian

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

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

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

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

## II.4 Hệ thống song song

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

Có nhiều nguyên nhân xây dựng dạng hệ thống này. Với sự gia tăng số lượng bộ xử lý, công việc được thực hiện nhanh chóng hơn, Nhưng không phải theo đúng tỉ lệ thời gian, nghĩa là có n bộ xử lý không có

nghĩa là sẽ thực hiện nhanh hơn n lần.

Hệ thống với máy nhiều bộ xử lý sẽ tối ưu hơn hệ thống có nhiều máy có một bộ xử lý vì các bộ xử lý chia sẻ các thiết bị ngoại vi, hệ thống lưu trữ, nguồn ... và rất thuận tiện cho nhiều chương trình cùng làm việc trên cùng một tập hợp dữ liệu.

Một lý do nữa là độ tin cậy. Các chức năng được xử lý trên nhiều bộ xử lý và sự hỏng hóc của một bộ xử lý sẽ không ảnh hưởng đến toàn bộ hệ thống.

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

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

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

## II.5 Hệ thống phân tán

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

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

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

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

## II.6 Hệ thống xử lý thời gian thực

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

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

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

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

### III. CẤU TRÚC CỦA HỆ ĐIỀU HÀNH

#### III.1 Các thành phần của hệ thống

##### Quản lý tiến trình

Một chương trình không thực hiện được gì cả nếu như nó không được CPU thi hành. Một **tiến trình** là một chương trình đang được thi hành, nhưng ý nghĩa của nó còn rộng hơn. Một công việc theo lô là một tiến trình. Một chương trình người dùng chia sẻ thời gian là một tiến trình, một công việc của hệ thống như soopling xuất ra máy in cũng là một tiến trình.

Một tiến trình phải sử dụng tài nguyên như thời gian sử dụng CPU, bộ nhớ, tập tin, các thiết bị nhập xuất để hoàn tất công việc của nó. Các tài nguyên này được cung cấp khi tiến trình được tạo hay trong quá trình thi hành. Khi tiến trình được tạo, nó sử dụng rất nhiều tài nguyên vật lý và luận lý. cũng như một số khởi tạo dữ liệu nhập. Ví dụ , khảo sát tiến trình hiển thị trạng thái của tập tin lên màn hình. Đầu vào của tiến trình là tên tập tin, và tiến trình sẽ thực hiện những chỉ thị thích hợp, thực hiện lời gọi hệ thống để nhận được những thông tin mong muốn và hiển thị nó lên màn hình. Khi tiến trình kết thúc, hệ điều hành sẽ tái tạo lại các tài nguyên có thể được dùng lại..

Một tiến trình là hoạt động (active) hoàn toàn-ngược lại với một tập tin trên đĩa là thụ động (passive)-với một bộ đếm chương trình cho biết lệnh kế tiếp được thi hành. Việc thi hành được thực hiện theo cơ chế tuần tự , CPU sẽ thi hành từ lệnh đầu đến lệnh cuối.

Một tiến trình được coi là một đơn vị làm việc của hệ thống. Một hệ thống có thể có nhiều tiến trình cùng lúc , trong đó một số tiến trình là của hệ điều hành, một số tiến trình là của người sử dụng. các tiến trình này có thể diễn ra đồng thời.

Vai trò của hệ điều hành trong việc quản lý tiến trình là :

- Tạo và hủy các tiến trình của người sử dụng và của hệ thống.
- Ngưng và thực hiện lại một tiến trình.
- Cung cấp cơ chế đồng bộ tiến trình.
- Cung cấp cách thông tin giữa các tiến trình.
- Cung cấp cơ chế kiểm soát deadlock(khái niệm này sẽ được trình bày trong chương II).

##### Quản lý bộ nhớ chính :

Trong hệ thống máy tính hiện đại, **bộ nhớ chính** là trung tâm của các thao tác, xử lý. Bộ nhớ chính có thể xem như một mảng kiểu byte hay kiểu word. Mỗi phân tử đều có địa chỉ. Đó là nơi lưu dữ liệu được CPU truy xuất một cách nhanh chóng so với các thiết bị nhập/xuất. CPU đọc những chỉ thị từ bộ nhớ

chính. Các thiết bị nhập/xuất cài đặt cơ chế DMA(xem chương IV) cũng đọc và ghi dữ liệu trong bộ nhớ chính. Thông thường bộ nhớ chính chứa các thiết bị mà CPU có thể định vị trực tiếp. Ví dụ CPU truy xuất dữ liệu từ đĩa, những dữ liệu này được chuyển vào bộ nhớ qua lời gọi hệ thống nhập/xuất.

Một chương trình muốn thi hành trước hết phải được ánh xạ thành địa chỉ tuyệt đối và nạp vào bộ nhớ chính. Khi chương trình thi hành, hệ thống truy xuất các chỉ thị và dữ liệu của chương trình trong bộ nhớ chính. Ngay cả khi tiến trình kết thúc, dữ liệu vẫn còn trong bộ nhớ cho đến khi một tiến trình khác được ghi chồng lên.

Để tối ưu hóa quá trình hoạt động của CPU và tốc độ của máy tính, một số tiến trình được lưu giữ trong bộ nhớ. Có rất nhiều kế hoạch quản trị bộ nhớ do có nhiều ứng dụng bộ nhớ khác nhau và hiệu quả của các thuật toán phụ thuộc vào tùy tình huống cụ thể. Lựa chọn một thuật toán cho một hệ thống được mô tả trước phụ thuộc vào nhiều yếu tố, đặc biệt là phần cứng của hệ thống.

Hệ điều hành có những vai trò như sau trong việc quản lý bộ nhớ chính :

- Lưu giữ thông tin về các vị trí trong bộ nhớ đã được sử dụng và ai sử dụng.
- Quyết định tiến trình nào được nạp vào bộ nhớ chính, khi bộ nhớ đã có thể dùng được.
- Cấp phát và thu hồi bộ nhớ khi cần thiết.

#### **Quản lý bộ nhớ phụ :**

Mục tiêu chính của hệ thống máy tính là thi hành chương trình. Những chương trình với dữ liệu truy xuất của chúng phải được đặt trong bộ nhớ chính trong suốt quá trình thi hành. Nhưng bộ nhớ chính quá nhỏ để có thể lưu giữ mọi dữ liệu và chương trình, ngoài ra dữ liệu sẽ mất khi không còn được cung cấp năng lượng. Hệ thống máy tính ngày nay cung cấp **hệ thống lưu trữ phụ**. Đa số các máy tính đều dùng đĩa để lưu trữ cả chương trình và dữ liệu. Hầu như tất cả chương trình : chương trình dịch, hợp ngữ, thủ tục, trình soạn thảo, định dạng... đều được lưu trữ trên đĩa cho tới khi nó được thực hiện, nạp vào trong bộ nhớ chính và cũng sử dụng đĩa để chứa dữ liệu và kết quả xử lý. Vì vậy một bộ quản lý hệ thống đĩa rất quan trọng cho hệ thống máy tính.

Vai trò của hệ điều hành trong việc quản lý đĩa :

- Quản lý vùng trống trên đĩa.
- Định vị lưu trữ.
- Lập lịch cho đĩa.

Vì hệ thống đĩa được sử dụng thường xuyên, nên nó phải được dùng hiệu quả. Tốc độ của toàn bộ hệ thống tùy thuộc rất nhiều vào tốc độ truy xuất đĩa.

#### **Quản lý hệ thống nhập xuất :**

Một trong những mục tiêu của hệ điều hành là *che dấu* những đặc thù của các thiết bị phần cứng đối với người sử dụng thay vào đó là một lớp thân thiện hơn, người sử dụng dễ thao tác hơn. Một hệ thống nhập/xuất bao gồm :

- Hệ thống buffer caching.
- Giao tiếp điều khiển thiết bị (device drivers) tổng quát.
- Bộ điều khiển cho các thiết bị phần cứng.

Chỉ có device driver mới hiểu đến cấu trúc đặc thù của thiết bị mà nó mô tả.

#### **Quản lý hệ thống tập tin :**

Hệ thống quản lý tập tin là thành phần rõ ràng nhất trong hệ điều hành. Máy tính có thể lưu trữ thông tin



trong nhiều dạng thiết bị vật lý khác nhau : băng từ, đĩa từ, , đĩa quang, ... Mỗi dạng có những đặc thù riêng về mặt tổ chức vật lý. Mỗi thiết bị có một bộ kiểm soát như bộ điều khiển đĩa (disk driver) và có những tính chất riêng. Những tính chất này là tốc độ, khả năng lưu trữ, tốc độ truyền dữ liệu và cách truy xuất.

Để cho việc sử dụng hệ thống máy tính thuận tiện, hệ điều hành cung cấp một cái nhìn logic đồng nhất về hệ thống lưu trữ thông tin. Hệ điều hành định nghĩa một đơn vị lưu trữ logic là tập tin. Hệ điều hành tạo một ánh xạ từ tập tin đến vùng thông tin trên đĩa và truy xuất những tập tin này thông qua thiết bị lưu trữ.

Một tập tin là một tập hợp những thông tin do người tạo ra nó xác định. Thông thường một tập tin đại diện cho một chương trình và dữ liệu. Dữ liệu của tập tin có thể là số, là ký tự, hay ký số. Tập tin thường có dạng tự do, như tập tin văn bản, nhị phân...(là tập tin chứa dãy các bit). (Xem bài VIII)

Vai trò của hệ điều hành trong việc quản lý tập tin :

- Tạo và xoá một tập tin.
- Tạo và xoá một thư mục.
- Hỗ trợ các thao tác trên tập tin và thư mục.
- Ánh xạ tập tin trên hệ thống lưu trữ phụ.
- Backup tập tin trên các thiết bị lưu trữ.

### Hệ thống bảo vệ :

Trong một hệ thống nhiều người sử dụng và cho phép nhiều tiến trình diễn ra đồng thời, các tiến trình phải được bảo vệ đối với những hoạt động khác. Do đó, hệ thống cung cấp cơ chế để đảm bảo rằng tập tin, bộ nhớ, CPU, và những tài nguyên khác chỉ được truy xuất bởi những tiến trình có quyền. Ví dụ, bộ nhớ đảm bảo rằng tiến trình chỉ được thi hành trong phạm vi địa chỉ của nó. Bộ thời gian đảm bảo rằng không có tiến trình nào độc chiếm CPU. Cuối cùng các thiết bị ngoại vi cũng được bảo vệ.

**Hệ thống bảo vệ** là một cơ chế kiểm soát quá trình truy xuất của chương trình, tiến trình, hoặc người sử dụng với tài nguyên của hệ thống. Cơ chế này cũng cung cấp cách thức để mô tả lại mức độ kiểm soát.

Hệ thống bảo vệ cũng làm tăng độ an toàn khi kiểm tra lỗi trong giao tiếp giữa những hệ thống nhỏ bên trong.

### Hệ thống cơ chế dòng lệnh :

Một trong những phần quan trọng của chương trình hệ thống trong một hệ điều hành là cơ chế **dòng lệnh**, đó là giao tiếp giữa người sử dụng và hệ điều hành. Một số hệ điều hành đặt cơ chế dòng lệnh bên trong hạt nhân, số khác như MS-DOS và UNIX thì xem hệ điều hành như là một chương trình đặt biệt, được thi hành khi các công việc bắt đầu hoặc khi người sử dụng login lần đầu tiên.

Các lệnh đưa vào hệ điều hành thông qua **bộ điều khiển lệnh**. Trong các hệ thống chia sẻ thời gian một chương trình có thể đọc và thông dịch các lệnh điều khiển được thực hiện một cách tự động. Chương trình này thường được gọi là bộ thông dịch điều khiển card, cơ chế dòng lệnh hoặc Shell. Chức năng của nó rất đơn giản đó là lấy lệnh kế tiếp và thi hành.

Mỗi hệ điều hành sẽ có những giao tiếp khác nhau, dạng đơn giản theo cơ chế dòng lệnh, dạng thân thiện với người sử dụng như giao diện của Macintosh có các biểu tượng, cửa sổ thao tác dùng chuột.

Các lệnh có quan hệ với việc tạo và quản lý các tiến trình, kiểm soát nhập xuất, quản lý bộ lưu trữ phụ, quản lý bộ nhớ chính, truy xuất hệ thống tập tin và cơ chế bảo vệ.

## IV. LỊCH SỬ PHÁT TRIỂN CÁC HỆ ĐIỀU HÀNH

### IV.1 Thế hệ 1 (1945 – 1955)

Vào khoảng giữa thập niên 1940, Howard Aiken ở Havard và John von Neumann ở Princeton, đã thành công trong việc xây dựng máy tính dùng ống chân không. Những máy này rất lớn với hơn 10000 ống chân không nhưng chậm hơn nhiều so với máy rẻ nhất ngày nay.

Mỗi máy được một nhóm thực hiện tất cả từ thiết kế, xây dựng lập trình, thao tác đến quản lý. Lập trình bằng ngôn ngữ máy tuyệt đối, thường là bằng cách dùng bảng điều khiển để thực hiện các chức năng cơ bản. Ngôn ngữ lập trình chưa được biết đến và hệ điều hành cũng chưa nghe đến.

Vào đầu thập niên 1950, phiếu đục lỗ ra đời và có thể viết chương trình trên phiếu thay cho dùng bảng điều khiển.

### IV.2 Thế hệ 2 (1955 – 1965)

Sự ra đời của thiết bị bán dẫn vào giữa thập niên 1950 làm thay đổi bức tranh tổng thể. Máy tính trở nên đủ tin cậy hơn. Nó được sản xuất và cung cấp cho các khách hàng. Lần đầu tiên có sự phân chia rõ ràng giữa người thiết kế, người xây dựng, người vận hành, người lập trình, và người bảo trì.

Để thực hiện một công việc (một chương trình hay một tập hợp các chương trình), lập trình viên trước hết viết chương trình trên giấy (bằng hợp ngữ hay FORTRAN) sau đó đục lỗ trên phiếu và cuối cùng đưa phiếu vào máy. Sau khi thực hiện xong nó sẽ xuất kết quả ra máy in.

**Hệ thống xử lý theo lô** ra đời, nó lưu các yêu cầu cần thực hiện lên băng từ, và hệ thống sẽ đọc và thi hành lần lượt. Sau đó, nó sẽ ghi kết quả lên băng từ xuất và cuối cùng người sử dụng sẽ đem băng từ xuất đi in.

Hệ thống xử lý theo lô hoạt động dưới sự điều khiển của một chương trình đặc biệt là tiền thân của hệ điều hành sau này. Ngôn ngữ lập trình sử dụng trong giai đoạn này chủ yếu là FORTRAN và hợp ngữ.

### IV.3 Thế hệ 3 (1965 – 1980)

Trong giai đoạn này, máy tính được sử dụng rộng rãi trong khoa học cũng như trong thương mại. Máy IBM 360 là máy tính đầu tiên sử dụng mạch tích hợp (IC). Từ đó kích thước và giá cả của các hệ thống máy giảm đáng kể và máy tính càng phổ biến hơn. Các thiết bị ngoại vi dành cho máy xuất hiện ngày càng nhiều và thao tác điều khiển bắt đầu phức tạp.

Hệ điều hành ra đời nhằm điều phối, kiểm soát hoạt động và giải quyết các yêu cầu tranh chấp thiết bị. Chương trình hệ điều hành dài cả triệu dòng hợp ngữ và do hàng ngàn lập trình viên thực hiện.

Sau đó, hệ điều hành ra đời khái niệm **đa chương**. CPU không phải chờ thực hiện các thao tác nhập xuất. Bộ nhớ được chia làm nhiều phần, mỗi phần có một công việc (job) khác nhau, khi một công việc chờ thực hiện nhập xuất CPU sẽ xử lý các công việc còn lại. Tuy nhiên khi có nhiều công việc cùng xuất hiện trong bộ nhớ, vấn đề là phải có một cơ chế bảo vệ tránh các công việc ảnh hưởng đến nhau. Hệ điều hành cũng cài đặt thuộc tính spool.

Giai đoạn này cũng đánh dấu sự ra đời của **hệ điều hành chia sẻ thời gian** như CTSS của MIT. Đồng thời các hệ điều hành lớn ra đời như MULTICS, UNIX và hệ thống các máy mini cũng xuất hiện như DEC PDP-1.

### IV.4 Thế hệ 4 (1980 - )

Giai đoạn này đánh dấu sự ra đời của máy tính cá nhân, đặc biệt là hệ thống IBM PC với hệ điều hành



MS-DOS và Windows sau này. Bên cạnh đó là sự phát triển mạnh của các hệ điều hành tựa Unix trên nhiều hệ máy khác nhau như Linux. Ngoài ra, từ đầu thập niên 90 cũng đánh dấu sự phát triển mạnh mẽ của *hệ điều hành mạng* và *hệ điều hành phân tán*.

### **Câu hỏi củng cố bài học**

1. Hệ điều hành là gì?
2. Có mấy loại hệ điều hành ? Việc phân loại này dựa trên những tiêu chuẩn nào ?
3. Nêu các thành phần chính của hệ điều hành và chức năng của mỗi thành phần này.
4. So sánh các cấu trúc khác nhau của hệ điều hành. Ưu khuyết điểm củ mỗi loại cấu trúc.
5. Quá trình phát triển của hệ điều hành phụ thuộc vào những yếu tố nào.

### **Bài tập**

1. Hệ điều hành là :
  - a. Một chương trình
  - b. Một chương trình hay hệ chương trình
  - c. Một thiết bị
  - d. ROM-BIOS
2. Một hệ điều hành bao gồm :
  - a. Hệ thống quản lý ttin, I/O
  - b. Hệ thống quản lý ttrình, bộ nhớ
  - c. a và b
  - d. a, b, c đều sai
3. Hệ điều hành MS-DOS có cấu trúc :
  - a. Đơn thể
  - b. Hạt nhân
  - c. Lớp
  - d. Máy ảo

## BÀI 2: CÁC MÔ HÌNH XỬ LÝ ĐỒNG HÀNH

Hầu hết các hệ điều hành hiện đại đều cho phép người dùng thi hành nhiều công việc đồng thời trên cùng một máy tính. Nhu cầu xử lý đồng hành (concurrency) này xuất phát từ đâu, và hệ điều hành cần phải tổ chức hỗ trợ như thế nào cho các môi trường đa nhiệm (multitask) như thế? Đó là nội dung chúng ta sẽ tìm hiểu trong bài này.

.....

### I. NHU CẦU XỬ LÝ ĐỒNG HÀNH

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

#### 🌟 Tăng hiệu suất sử dụng CPU

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

|     |    |     |    |     |
|-----|----|-----|----|-----|
| CPU | IO | CPU | IO | CPU |
|-----|----|-----|----|-----|

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

|     |    |     |    |     |
|-----|----|-----|----|-----|
| CPU | IO | CPU | IO | CPU |
|-----|----|-----|----|-----|

Tác vụ 1

|  |     |    |     |    |
|--|-----|----|-----|----|
|  | CPU | IO | CPU | IO |
|--|-----|----|-----|----|

Tác vụ 2

#### 🌟 Tăng tốc độ xử lý

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

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

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

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

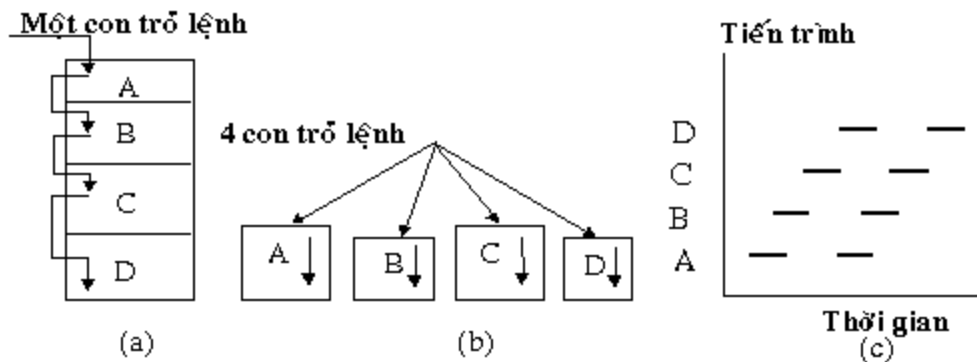
Để hỗ trợ sự đa chương, máy tính phải có khả năng thực hiện nhiều tác vụ đồng thời. Nhưng việc điều khiển nhiều hoạt động song song ở cấp độ phần cứng là rất khó khăn. Vì thế các nhà thiết kế hệ điều hành đề xuất một mô hình *song song giả lập* bằng cách chuyển đổi bộ xử lý qua lại giữa các chương

trình để duy trì hoạt động của nhiều chương trình cùng lúc, điều này tạo cảm giác có nhiều hoạt động được thực hiện đồng thời.

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

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

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



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

### III. KHÁI NIỆM TIỂU TRÌNH (THREAD) VÀ MÔ HÌNH ĐA TIỂU TRÌNH (MULTITHREAD)

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

**Ví dụ :** Một server quản lý tập tin thỉnh thoảng phải tự khóa để chờ các thao tác truy xuất đĩa hoàn tất. Nếu server có nhiều dòng xử lý, hệ thống có thể xử lý các yêu cầu mới trong khi một dòng xử lý bị khóa. Như vậy việc thực hiện chương trình sẽ có hiệu quả hơn. Điều này không thể đạt được bằng cách tạo hai tiến trình server riêng biệt vì cần phải chia sẻ cùng một vùng đệm, do vậy bắt buộc phải chia sẻ không gian địa chỉ.

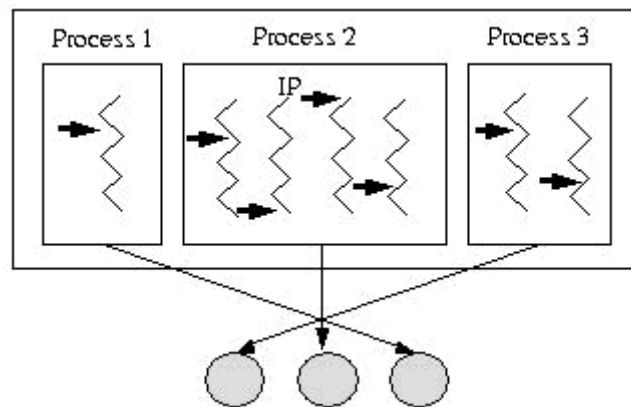
Chính vì các tình huống tương tự, người ta cần có một cơ chế xử lý mới cho phép có nhiều dòng xử lý trong cùng một tiến trình.

Ngày nay đã có nhiều hệ điều hành cung cấp một cơ chế như thế và gọi là *tiểu trình (threads)*.

### Nguyên lý chung :

*Một tiểu trình là một đơn vị xử lý cơ bản trong hệ thống . Mỗi tiểu trình xử lý tuần tự đoạn code của nó, sở hữu một con trỏ lệnh, tập các thanh ghi và một vùng nhớ stack riêng. Các tiểu trình chia sẻ CPU với nhau giống như cách chia sẻ giữa các tiến trình: một tiểu trình xử lý trong khi các tiểu trình khác chờ đến lượt. Một tiểu trình cũng có thể tạo lập các tiến trình con, và nhận các trạng thái khác nhau như một tiến trình thật sự. Một tiến trình có thể sở hữu nhiều tiểu trình.*

Các tiến trình tạo thành những thực thể độc lập. Mỗi tiến trình có một tập tài nguyên và một môi trường riêng (một con trỏ lệnh, một Stack , các thanh ghi và không gian địa chỉ ). Các tiến trình hoàn toàn độc lập với nhau, chỉ có thể liên lạc thông qua các cơ chế thông tin giữa các tiến trình mà hệ điều hành cung cấp. Ngược lại, các tiểu trình trong cùng một tiến trình lại chia sẻ một không gian địa chỉ chung , điều này có nghĩa là các tiểu trình có thể chia sẻ các biến toàn cục của tiến trình. Một tiểu trình có thể truy xuất đến cả các stack của những tiểu trình khác trong cùng tiến trình. Cấu trúc này không đề nghị một cơ chế bảo vệ nào, và điều này cũng không thật cần thiết vì các tiểu trình trong cùng một tiến trình thuộc về cùng một sở hữu chủ đã tạo ra chúng trong ý định cho phép chúng hợp tác với nhau.



Các tiểu trình trong cùng một tiến trình

### Phân bổ thông tin lưu trữ

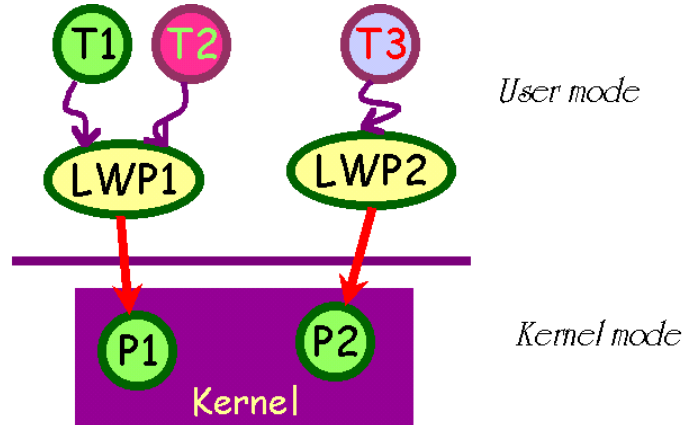
| Tiến trình             | Tiểu trình                   |
|------------------------|------------------------------|
| Không gian địa chỉ     | Con trỏ lệnh + các thanh ghi |
| Tài nguyên toàn cục    | Stack                        |
| Các thông tin thống kê | Tài nguyên cục bộ            |

Cấu trúc mô tả tiến trình và tiểu trình

### Kernel thread và user thread

Khái niệm tiểu trình có thể được cài đặt trong kernel của Hệ điều hành, khi đó đơn vị cơ sở sử dụng CPU để xử lý là tiểu trình, Hệ điều hành sẽ phân phối CPU cho các tiểu trình trong hệ thống. Tuy nhiên đối với một số hệ điều hành, khái niệm tiểu trình chỉ được hỗ trợ như một đối tượng người dùng, các thao tác tiểu trình được cung cấp kèm theo do một thư viện xử lý trong chế độ người dùng không đặc

quyền (user mode). Lúc này Hệ điều hành sẽ chỉ biết đến khái niệm tiến trình, do vậy cần cơ chế để liên kết các tiểu trình cùng một tiến trình với tiến trình cha trong kernel\_ đối tượng này đôi lúc được gọi là LWP (lightweight process).



#### IV. TÓM TẮT

Tiến trình là một chương trình đang hoạt động.

Để sử dụng hiệu quả CPU, sự đa chương cần được đưa vào hệ thống

Sự đa chương được tổ chức bằng cách lưu trữ nhiều tiến trình trong bộ nhớ tại một thời điểm, và điều phối CPU qua lại giữa các tiến trình trong hệ thống.

Mô hình đa tiểu trình cho phép mỗi tiến trình có thể tiến hành nhiều dòng xử lý đồng thời trong cùng một không gian địa chỉ nhằm thực hiện tác vụ hiệu quả hơn trong một số trường hợp.

#### +Củng cố bài học

Các câu hỏi cần trả lời được sau bài học này :

1. Tại sao các hệ điều hành hiện đại hỗ trợ môi trường đa nhiệm ?
2. Phân biệt multitask, multiprogramming và multiprocessing.
3. Khái niệm tiến trình được xây dựng nhằm mục đích gì ?
4. Sự khác biệt, mối quan hệ giữa tiến trình và tiểu trình ?

#### +Bài tập

**Bài 1.** Nhiều hệ điều hành không cho phép xử lý đồng hành. Thảo luận về các phức tạp phát sinh khi hệ điều hành cho phép đa nhiệm ?

**Bài 2.** Tìm một số ứng dụng thích hợp với mô hình đa tiến trình; và một số ứng dụng thích hợp với mô hình đa tiểu trình.

## Bài 3: QUẢN LÝ TIẾN TRÌNH

Trong bài này chúng ta sẽ tìm hiểu chức năng quản lý tiến trình của Hệ điều hành : làm thế nào để phân chia CPU cho các tiến trình ? Theo vết xử lý của tiến trình ? Và các thao tác trên tiến trình ?

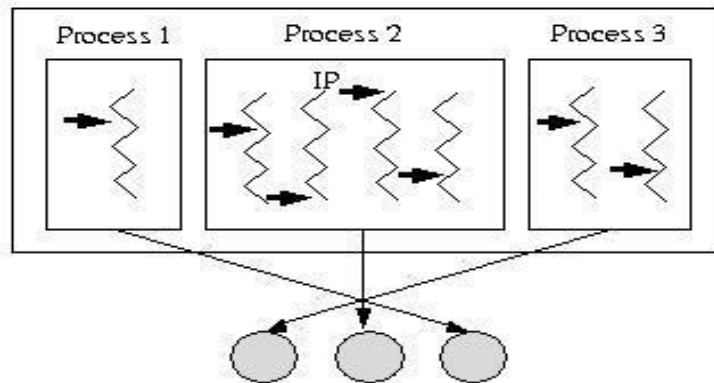
### I. TỔ CHỨC QUẢN LÝ TIẾN TRÌNH

#### I.1. Các trạng thái của tiến trình

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

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

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



Hình 2.2 Sơ đồ chuyển trạng thái giữa các tiến trình

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

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

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

#### I.2. Chế độ xử lý của tiến trình

Để đảm bảo hệ thống hoạt động đúng đắn, hệ điều hành cần phải được bảo vệ khỏi sự xâm phạm của các



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



Hình vẽ 2.3 Hai chế độ xử lý

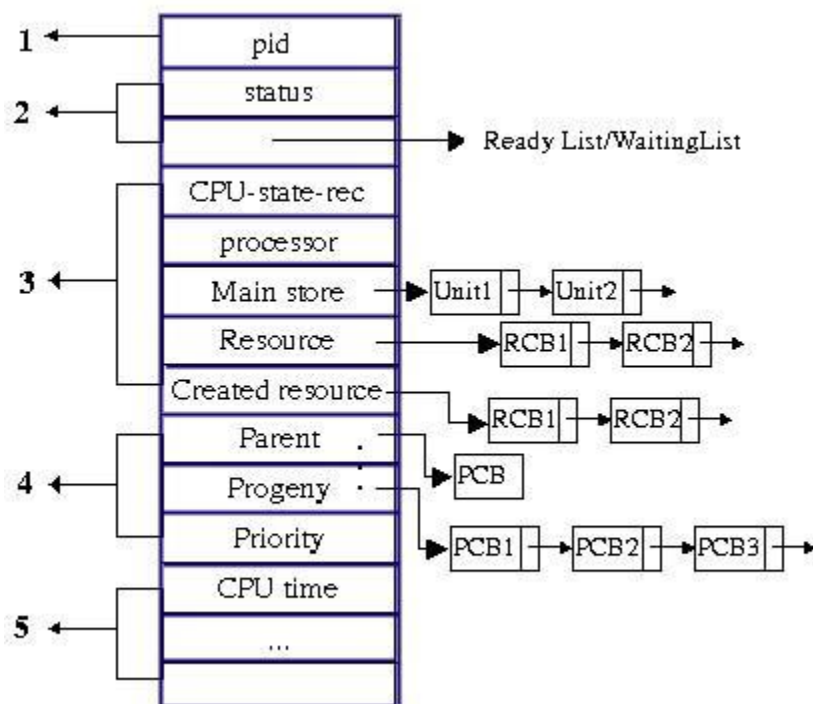
### ► I.3. Cấu trúc dữ liệu khối quản lý tiến trình

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

- **Định danh của tiến trình (1)** : giúp phân biệt các tiến trình
- **Trạng thái tiến trình (2)**: xác định hoạt động hiện hành của tiến trình.
- **Ngữ cảnh của tiến trình (3)**: mô tả các tài nguyên tiến trình đang trong quá trình, hoặc để phục vụ cho hoạt động hiện tại, hoặc để làm cơ sở phục hồi hoạt động cho tiến trình, bao gồm các thông tin về:
  - ▶ *Trạng thái CPU*: bao gồm nội dung các thanh ghi, quan trọng nhất là con trỏ lệnh IP lưu trữ địa chỉ câu lệnh kế tiếp tiến trình sẽ xử lý. Các thông tin này cần được lưu trữ khi xảy ra một ngắt, nhằm có thể cho phép phục hồi hoạt động của tiến trình đúng như trước khi bị ngắt.
  - ▶ *Bộ xử lý*: dùng cho máy có cấu hình nhiều CPU, xác định số hiệu CPU mà tiến trình đang sử dụng.
  - ▶ *Bộ nhớ chính*: danh sách các khối nhớ được cấp cho tiến trình.
  - ▶ *Tài nguyên sử dụng*: danh sách các tài nguyên hệ thống mà tiến trình đang sử dụng.
  - ▶ *Tài nguyên tạo lập*: danh sách các tài nguyên được tiến trình tạo lập.
- **Thông tin giao tiếp (4)**: phản ánh các thông tin về quan hệ của tiến trình với các tiến trình khác trong hệ thống :
  - ▶ *Tiến trình cha*: tiến trình tạo lập tiến trình này .
  - ▶ *Tiến trình con*: các tiến trình do tiến trình này tạo lập .

► **Độ ưu tiên** : giúp bộ điều phối có thông tin để lựa chọn tiến trình được cấp CPU.

• **Thông tin thống kê (5)**: đây là những thông tin thống kê về hoạt động của tiến trình, như thời gian đã sử dụng CPU, thời gian chờ. Các thông tin này có thể có ích cho công việc đánh giá tình hình hệ thống và dự đoán các tình huống tương lai.



**Hình vẽ 2.4** Khối mô tả tiến trình

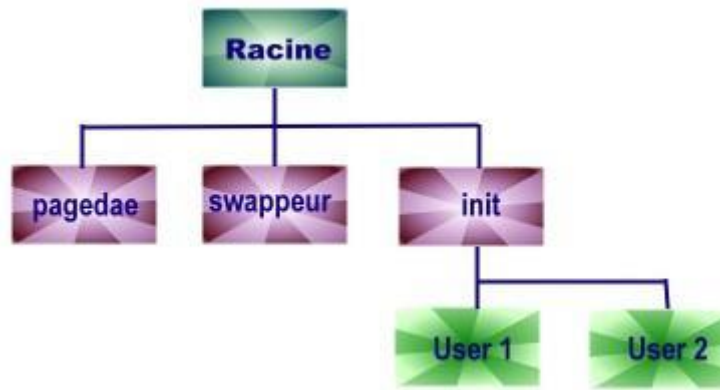
#### ► I.4. Thao tác trên tiến trình

Hệ điều hành cung cấp các thao tác chủ yếu sau đây trên một tiến trình :

- tạo lập tiến trình (create)
- kết thúc tiến trình (destroy)
- tạm dừng tiến trình (suspend)
- tái kích hoạt tiến trình (resume)
- thay đổi độ ưu tiên tiến trình

##### \* I.4.1. Tạo lập tiến trình

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



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

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

- ▶định danh cho tiến trình mới phát sinh
- ▶đưa tiến trình vào danh sách quản lý của hệ thống
- ▶xác định độ ưu tiên cho tiến trình
- ▶tạo PCB cho tiến trình
- ▶cấp phát các tài nguyên ban đầu cho tiến trình

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

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

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

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

#### **\*I.4.2. Kết thúc tiến trình**

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

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

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

#### **I.5. Cấp phát tài nguyên cho tiến trình**

Khi có nhiều người sử dụng đồng thời làm việc trong hệ thống, hệ điều hành cần phải cấp phát các tài nguyên theo yêu cầu cho mỗi người sử dụng. Do tài nguyên hệ thống thường rất giới hạn và có khi không thể chia sẻ, nên hiếm khi tất cả các yêu cầu tài nguyên đồng thời đều được thỏa mãn. Vì thế cần phải nghiên cứu một phương pháp để chia sẻ một số tài nguyên hữu hạn giữa nhiều tiến trình người

dùng đồng thời. Hệ điều hành quản lý nhiều loại tài nguyên khác nhau (CPU, bộ nhớ chính, các thiết bị ngoại vi ...), với mỗi loại cần có một cơ chế cấp phát và các chiến lược cấp phát hiệu quả. Mỗi tài nguyên được biểu diễn thông qua một cấu trúc dữ liệu, khác nhau về chi tiết cho từng loại tài nguyên, nhưng cơ bản chứa đựng các thông tin sau :

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



**Hình 2.6** Khối quản lý tài nguyên

Các mục tiêu của kỹ thuật cấp phát :

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

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

## II. ĐIỀU PHỐI TIẾN TRÌNH

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

Để thực hiện được mục tiêu này, hệ điều hành phải lựa chọn tiến trình được xử lý tiếp theo. Bộ điều phối

sẽ sử dụng một giải thuật điều phối thích hợp để thực hiện nhiệm vụ này. Một thành phần khác của hệ điều hành cũng tiềm ẩn trong công tác điều phối là *bộ phân phối* (dispatcher). Bộ phân phối sẽ chịu trách nhiệm chuyển đổi ngữ cảnh và trao CPU cho tiến trình được chọn bởi bộ điều phối để xử lý.

## II.1. Giới thiệu

### \*II.1.1. Mục tiêu điều phối

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

**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 quả (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.

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

### \*II.1.2. Các đặc điểm của tiến trình

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

**a) Tính hướng xuất / nhập của tiến trình ( I/O-boundedness):**

Khi một tiến trình nhận được CPU, chủ yếu nó chỉ sử dụng CPU đến khi phát sinh một yêu cầu nhập xuất ? Hoạt động của các tiến trình như thế thường bao gồm nhiều lượt sử dụng CPU , mỗi lượt trong một thời gian khá ngắn.

**b) Tính hướng xử lý của tiến trình ( CPU-boundedness):**

Khi một tiến trình nhận được CPU, nó có khuynh hướng sử dụng CPU đến khi hết thời gian dành cho nó ? Hoạt động của các tiến trình như thế thường bao gồm một số ít lượt sử dụng CPU , nhưng mỗi lượt trong một thời gian đủ dài.

**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 theo một số tiêu chuẩn đánh giá nào đó, một cách hợp lý, các tiến trình quan trọng hơn ( có độ ưu tiên cao hơn) cần được ưu tiên hơn.

**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.

**\*II.1.3. Điều phối không độc quyền và điều phối độc quyền (preemptive/nopreemptive)**

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

**☞Điều phối độc quyền :** Nguyên lý điều phối *độc quyền* cho phép một tiến trình khi nhận được CPU sẽ có quyền độc chiếm CPU đến khi hoàn tất xử lý hoặc tự nguyện giải phóng CPU. Khi đó quyết định điều phối CPU sẽ xảy ra trong các tình huống sau:

- Khi tiến trình chuyển từ trạng thái đang xử lý (running) sang trạng thái bị khóa blocked ( ví dụ chờ một thao tác nhập xuất hay chờ một tiến trình con kết thúc...).
- Khi tiến trình kết thúc.

Các giải thuật độc quyền thường đơn giản và dễ cài đặt. Tuy nhiên chúng thường không thích hợp với các hệ thống tổng quát nhiều người dùng, vì nếu cho phép một tiến trình có quyền xử lý bao lâu tùy ý, có nghĩa là tiến trình này có thể giữ CPU một thời gian không xác định, có thể ngăn cản những tiến trình còn lại trong hệ thống có một cơ hội để xử lý.

**☞Điều phối không độc quyền :** Ngược với nguyên lý độc quyền, điều phối theo nguyên lý *không độc quyền* cho phép tạm dừng hoạt động của một tiến trình đang sẵn sàng xử lý. Khi một tiến trình nhận được CPU, nó vẫn được sử dụng CPU đến khi hoàn tất hoặc tự nguyện giải phóng CPU, nhưng một tiến trình khác có độ ưu tiên có thể dành quyền sử dụng CPU của tiến trình ban đầu. Như vậy là tiến trình có thể bị tạm dừng hoạt động bất cứ lúc nào mà không được báo trước, để tiến trình khác xử lý. Các quyết định điều phối xảy ra khi :

- Khi tiến trình chuyển từ trạng thái đang xử lý (running) sang trạng thái bị khóa blocked ( ví dụ chờ một thao tác nhập xuất hay chờ một tiến trình con kết thúc...).
- Khi tiến trình chuyển từ trạng thái đang xử lý (running) sang trạng thái ready ( ví dụ xảy ra một ngắt).
- Khi tiến trình chuyển từ trạng thái chờ (blocked) sang trạng thái ready ( ví dụ một thao tác nhập/xuất hoàn tất).
- Khi tiến trình kết thúc.

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



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

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

## II.2. Tổ chức điều phối

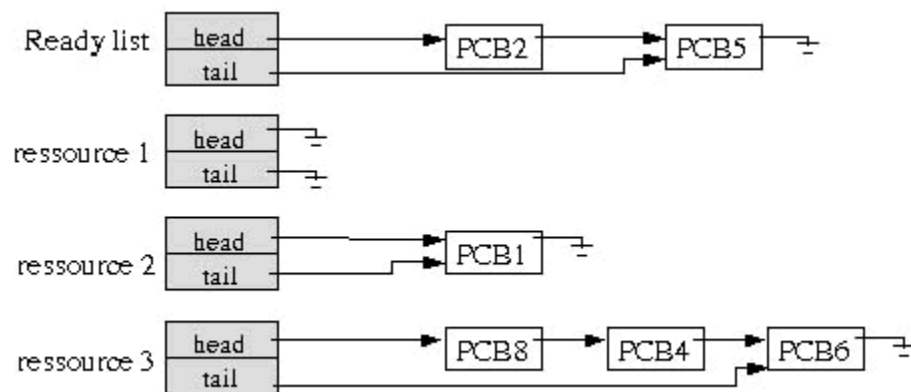
### \*II.2.1. Các danh sách sử dụng trong quá trình điều phối.

Hệ điều hành sử dụng hai loại danh sách để thực hiện điều phối các tiến trình là *danh sách sẵn sàng* (ready list) và *danh sách chờ đợi* (waiting list).

Khi một tiến trình bắt đầu đi vào hệ thống, nó được chèn vào danh sách các tác vụ (job list). Danh sách này bao gồm tất cả các tiến trình của hệ thống. Nhưng chỉ các tiến trình đang thường trú trong bộ nhớ chính và ở trạng thái sẵn sàng tiếp nhận CPU để hoạt động mới được đưa vào *danh sách sẵn sàng*.

Bộ điều phối sẽ chọn một tiến trình trong danh sách sẵn sàng và cấp CPU cho tiến trình đó. Tiến trình được cấp CPU sẽ thực hiện xử lý, và có thể chuyển sang trạng thái chờ khi xảy ra các sự kiện như đợi một thao tác nhập/xuất hoàn tất, yêu cầu tài nguyên chưa được thỏa mãn, được yêu cầu tạm dừng ... Khi đó tiến trình sẽ được chuyển sang một danh sách chờ đợi.

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



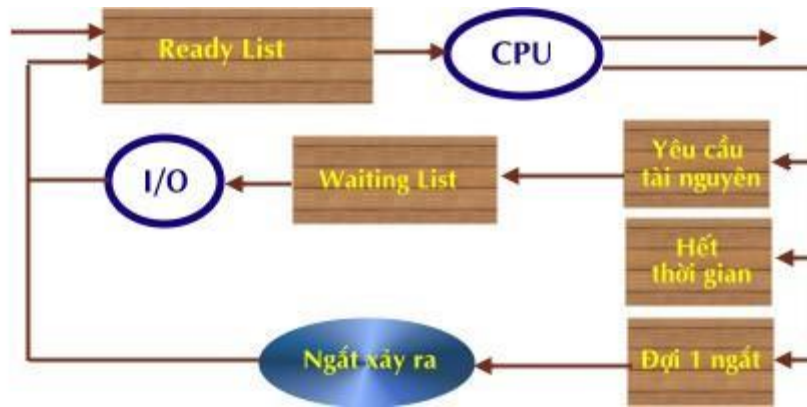
Hình 2.9 Các danh sách điều phối

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

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

- ▶ Tiến trình phát sinh một yêu cầu một tài nguyên mà hệ thống chưa thể đáp ứng, khi đó tiến trình sẽ được chuyển sang danh sách các tiến trình đang chờ tài nguyên tương ứng.
- ▶ Tiến trình có thể bị bắt buộc tạm dừng xử lý do một ngắt xảy ra, khi đó tiến trình được đưa trở

lại vào danh sách sẵn sàng để chờ được cấp CPU cho lượt tiếp theo.



**Hình 2.10** Sơ đồ chuyển đổi giữa các danh sách điều phối

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

### \*II.2.2. Các cấp độ điều phối

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

#### a) Điều phối tác vụ

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

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

#### b) Điều phối tiến trình

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

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

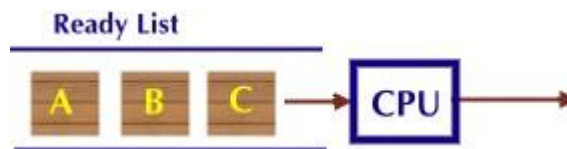


**Hình 2.11** Cấp độ điều phối trung gian

## II.3. Các chiến lược điều phối

### \*II.3.1. Chiến lược FIFO

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



**Hình 2.12** Điều phối FIFO

🔴 Ví dụ :

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

Thứ tự cấp phát CPU cho các tiến trình là :

| P1 | P2 | P3    |
|----|----|-------|
| 0  | 24 | 27 30 |

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

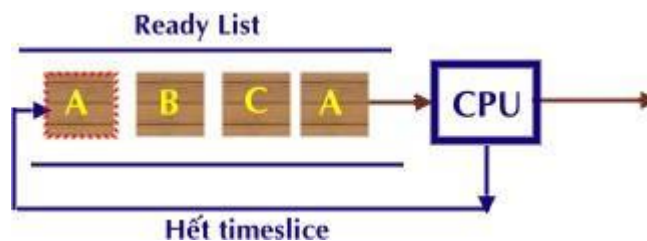
☀ **Thảo luận** : Thời gian chờ trung bình không đạt cực tiểu, và biến đổi đáng kể đối với các giá trị về thời gian yêu cầu xử lý và thứ tự khác nhau của các tiến trình trong danh sách sẵn sàng. Có thể xảy ra hiện tượng tích lũy thời gian chờ, khi các tất cả các tiến trình (có thể có yêu cầu thời gian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

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

### \*II.3.2. Chiến lược phân phối xoay vòng (Round Robin)

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

🎮 Ví dụ :



**Hình 2.13** Điều phối Round Robin

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

Nếu sử dụng quantum là 4 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  $(0+6+3+5)/3 = 4.66$  miliseconds.

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

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

### \*II.3.3. Điều phối với độ ưu tiên

✚ **Nguyên tắc** : Mỗi tiến trình được gán cho một độ ưu tiên tương ứng, tiến trình có độ ưu tiên cao nhất sẽ được chọn để cấp phát CPU đầu tiên. Độ ưu tiên có thể được định nghĩa nội tại hay nhờ vào các yếu tố bên ngoài. Độ ưu tiên nội tại sử dụng các đại lượng có thể đo lường để tính toán độ ưu tiên của tiến trình, ví dụ các giới hạn thời gian, nhu cầu bộ nhớ... Độ ưu tiên cũng có thể được gán từ bên ngoài dựa vào các tiêu chuẩn do hệ điều hành như tầm quan trọng của tiến trình, loại người sử dụng 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 điều phối với độ ưu tiên và không độc quyền sẽ thu hồi CPU từ tiến trình hiện hành để cấp phát cho tiến trình mới nếu độ ưu tiên của tiến trình này cao hơn tiến trình hiện hành. Một giải thuật độc quyền sẽ chỉ đơn giản chen tiến trình mới vào danh sách sẵn sàng, và tiến trình hiện hành vẫn tiếp tục xử lý hết thời gian dành cho nó.

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

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

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

|    |    |       |
|----|----|-------|
| P1 | P2 | P3    |
| 0  | 24 | 27 30 |

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

|    |    |    |      |
|----|----|----|------|
| P1 | P2 | P3 | P1   |
| 0  | 1  | 4  | 7 30 |

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

trình có độ ưu tiên cao thứ nhì, sẽ xảy ra sự chuyển đổi quyền sử dụng CPU. Quá trình này gọi là sự ‘lão hóa’ (*aging*) tiến trình.

#### \*II.3.4. Chiến lược công việc ngắn nhất (*Shortest-job-first SJF*)

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

🎥 Ví dụ :

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

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

|    |    |    |       |
|----|----|----|-------|
| P1 | P4 | P3 | P2    |
| 0  | 6  | 8  | 12 20 |

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

|    |    |    |    |       |
|----|----|----|----|-------|
| P1 | P4 | P1 | P3 | P2    |
| 0  | 3  | 5  | 8  | 12 20 |

🌟 **Thảo luận** : Giải thuật này cho phép đạt được thời gian chờ trung bình cực tiểu. Khó khăn thực sự của giải thuật SJF là không thể biết được thời gian yêu cầu xử lý còn lại của tiến trình ? Chỉ có thể dự đoán giá trị này theo cách tiếp cận sau : gọi  $t_n$  là độ dài của thời gian xử lý lần thứ  $n$ ,  $\tau_{n+1}$  là giá trị dự đoán cho lần xử lý tiếp theo. Với hy vọng giá trị dự đoán sẽ gần giống với các giá trị trước đó, có thể sử dụng công thức:

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

Trong công thức này,  $t_n$  chứa đựng thông tin gần nhất ;  $\tau_n$  chứa đựng các thông tin quá khứ được tích



lũy. Tham số  $\alpha$  ( $0 \leq \alpha \leq 1$ ) kiểm soát trọng số của hiện tại gần hay quá khứ ảnh hưởng đến công thức dự đoán.

### \*II.3.5. Chiến lược điều phối với nhiều mức độ ưu tiên

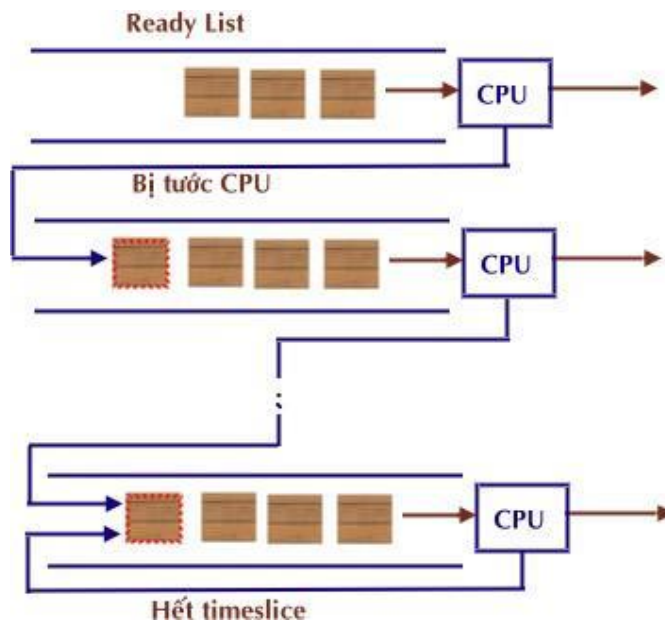
✚ **Nguyên tắc** : Ý tưởng chính của giải thuật là phân lớp các tiến trình tùy theo độ ưu tiên của chúng để có cách thức điều phối thích hợp cho từng nhóm. Danh sách sẵn sàng được phân tách thành các danh sách riêng biệt theo cấp độ ưu tiên, mỗi danh sách bao gồm các tiến trình có cùng độ ưu tiên và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



**Hình 2.14** Điều phối nhiều cấp ưu tiên

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

- Số lượng các cấp ưu tiên
- Giải thuật điều phối cho từng danh sách ứng với một cấp ưu tiên.
- Phương pháp xác định thời điểm di chuyển một tiến trình lên danh sách có độ ưu tiên cao hơn.
- Phương pháp xác định thời điểm di chuyển một tiến trình lên danh sách có độ ưu tiên thấp hơn.
- Phương pháp sử dụng để xác định một tiến trình mới được đưa vào hệ thống sẽ thuộc danh sách ứng với độ ưu tiên nào.



**Hình 2.15** Điều phối Multilevel Feedback

#### \*II.3.6. Chiến lược điều phối Xổ số (Lottery)

🔍 **Nguyên tắc** : Ý tưởng chính của giải thuật là phát hành một số vé số và phân phối cho các tiến trình trong hệ thống. Khi đến thời điểm ra quyết định điều phối, sẽ tiến hành chọn 1 vé "trúng giải", tiến trình nào sở hữu vé này sẽ được nhận CPU

🔍 **Thảo luận** : Giải thuật Lottery cung cấp một giải pháp đơn giản nhưng bảo đảm tính công bằng cho thuật toán điều phối với chi phí thấp để cập nhật độ ưu tiên cho các tiến trình :

### III. TÓM TẮT

- ▶ Trong suốt chu trình sống, tiến trình chuyển đổi qua lại giữa các trạng thái ready, running và blocked.
- ▶ Bộ điều phối của hệ điều hành chịu trách nhiệm áp dụng một giải thuật điều phối thích hợp để chọn tiến trình thích hợp được sử dụng CPU, và bộ phân phối sẽ chuyển giao CPU cho tiến trình này.
- ▶ Các giải thuật điều phối thông dụng : FIFO, RoundRobin, điều phối với độ ưu tiên, SJF, Multilevel Feedback

#### 📌 Câu hỏi củng cố bài học

Các câu hỏi cần trả lời được sau bài học này :

1. Thông tin lưu trữ trong PCB và TCB ?
2. Tổ chức điều phối tiến trình ?
3. Phân tích ưu, khuyết của các chiến lược điều phối


#### 📌 Bài tập

🔍 **Bài 1.** Xét tập các tiến trình sau (với thời gian yêu cầu CPU và độ ưu tiên kèm theo) :

| Tiến trình     | Thời điểm vào RL | Thời gian CPU | Độ ưu tiên |
|----------------|------------------|---------------|------------|
| P <sub>1</sub> | 0                | 10            | 3          |
| P <sub>2</sub> | 1                | 1             | 1          |
| P <sub>3</sub> | 2                | 2             | 3          |
| P <sub>4</sub> | 3                | 1             | 4          |
| P <sub>5</sub> | 4                | 5             | 2          |

Giả sử các tiến trình cùng được đưa vào hệ thống tại thời điểm 0

- Cho biết kết quả điều phối hoạt động của các tiến trình trên theo thuật toán FIFO; SJF; điều phối theo độ ưu tiên độc quyền (độ ưu tiên  $1 > 2 > \dots$ ); và RR (quantum=2).
- Cho biết thời gian lưu lại trong hệ thống (turnaround time) của từng tiến trình trong từng thuật toán điều phối ở câu a.
- Cho biết thời gian chờ trong hệ thống (waiting time) của từng tiến trình trong từng thuật toán điều phối ở câu a.
- Thuật toán điều phối nào trong các thuật toán ở câu a cho thời gian chờ trung bình là cực tiểu ?

 **Bài 2.** Giả sử có các tiến trình sau trong hệ thống :

| Tiến trình     | Thời điểm vào RL | Thời gian CPU |
|----------------|------------------|---------------|
| P <sub>1</sub> | 0.0              | 8             |
| P <sub>2</sub> | 0.4              | 4             |
| P <sub>3</sub> | 1.0              | 1             |

Sử dụng nguyên tắc điều phối độc quyền và các thông tin có được tại thời điểm ra quyết định để trả lời các câu hỏi sau đây :

- Cho biết thời gian lưu lại trung bình trong hệ thống (turnaround time) của các tiến trình trong thuật toán điều phối FIFO.
- Cho biết thời gian lưu lại trung bình trong hệ thống (turnaround time) của các tiến trình trong thuật toán điều phối SJF.

c) Thuật toán SJF dự định cải tiến sự thực hiện của hệ thống, nhưng lưu ý chúng ta phải chọn điều phối  $P_1$  tại thời điểm 0 vì không biết rằng sẽ có hai tiến trình ngắn hơn vào hệ thống sau đó. Thử tính thời gian lưu lại trung bình trong hệ thống nếu để CPU nhàn rỗi trong 1 đơn vị thời gian đầu tiên và sau đó sử dụng SJF để điều phối. Lưu ý  $P_1$  và  $P_2$  sẽ phải chờ trong suốt thời gian nhàn rỗi này, do vậy thời gian chờ của chúng tăng lên. Thuật toán điều phối này được biết đến như điều phối dựa trên thông tin về tương lai.

**❓ Bài 3.** Phân biệt sự khác nhau trong cách tiếp cận để ưu tiên cho tiến trình ngắn trong các thuật toán điều phối sau :

- a) FIFO.
- b) RR
- c) Điều phối với độ ưu tiên đa cấp

**❓ Bài 4.** Cho biết hai ưu điểm chính của mô hình đa tiểu trình so với đa tiến trình. Mô tả một ứng dụng thích hợp với mô hình đa tiểu trình và một ứng dụng khác không thích hợp.

**❓ Bài 5.** Mô tả các xử lý hệ điều hành phải thực hiện khi chuyển đổi ngữ cảnh giữa :

- a) các tiến trình
- b) các tiểu trình

**❓ Bài 6.** Xác định thời lượng quantum  $q$  là một nhiệm vụ khó khăn. Giả sử chi phí trung bình cho một lần chuyển đổi ngữ cảnh là  $s$ , và thời gian trung bình một tiến trình hướng nhập xuất sử dụng CPU trước khi phát sinh một yêu cầu nhập xuất là  $t$  ( $t \gg s$ ). Thảo luận các tác động đến sự thực hiện của hệ thống khi chọn  $q$  theo các quy tắc sau :

- a)  $q$  bất định
- b)  $q$  lớn hơn 0 1 ít
- c)  $q = s$
- d)  $s < q < t$
- e)  $q = t$
- f)  $q > t$

**❓ Bài 7.** Giả sử một hệ điều hành áp dụng giải thuật điều phối multilevel feedback với 5 mức ưu tiên (giảm dần). Thời lượng quantum dành cho hàng đợi cấp 1 là 0,5s. Mỗi hàng đợi cấp thấp hơn sẽ có thời lượng quantum dài gấp đôi hàng đợi ứng với mức ưu tiên cao hơn nó. Một tiến trình khi vào hệ thống sẽ được đưa vào hàng đợi mức cao nhất, và chuyển dần xuống các hàng đợi bên dưới sau mỗi lượt sử dụng CPU. Một tiến trình chỉ có thể bị thu hồi CPU khi đã sử dụng hết thời lượng quantum dành cho nó. Hệ thống có thể thực hiện các tác vụ xử lý theo lô hoặc tương tác, và mỗi tác vụ lại có thể hướng xử lý hay hướng nhập xuất.

- a) Giải thích tại sao hệ thống này hoạt động không hiệu quả ?
- b) Cần phải thay đổi (tối thiểu) như thế nào để hệ thống điều phối các tác vụ với những bản chất khác biệt như thế tốt hơn ?

## BÀI 4 : LIÊN LẠC GIỮA CÁC TIẾN TRÌNH & VẤN ĐỀ ĐỒNG BỘ HOÁ

Các tiến trình trên nguyên tắc là hoàn toàn độc lập, nhưng thực tế có thể như thế không ? Trong bài này chúng ta sẽ tìm hiểu lý do các tiến trình có nhu cầu liên lạc, các cơ chế hỗ trợ việc liên lạc này cũng như những vấn đề đặt ra khi các tiến trình trao đổi thông tin với nhau.

### I. LIÊN LẠC GIỮA CÁC TIẾN TRÌNH

#### I.1. Nhu cầu liên lạc giữa các tiến trình

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

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

#### I.2. Các vấn đề nảy sinh trong việc liên lạc giữa các tiến trình

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

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

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

### II. Các Cơ Chế Thông Tin Liên Lạc

#### II.1. Tín hiệu (Signal)

- **Giới thiệu:** Tín hiệu là một cơ chế phần mềm tương tự như các ngắt cứng tác động đến các tiến trình.

Một tín hiệu được sử dụng để thông báo cho tiến trình về một sự kiện nào đó xảy ra. Có nhiều tín hiệu được định nghĩa, mỗi một tín hiệu có một ý nghĩa tương ứng với một sự kiện đặc trưng.

► Ví dụ : Một số tín hiệu của UNIX

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

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

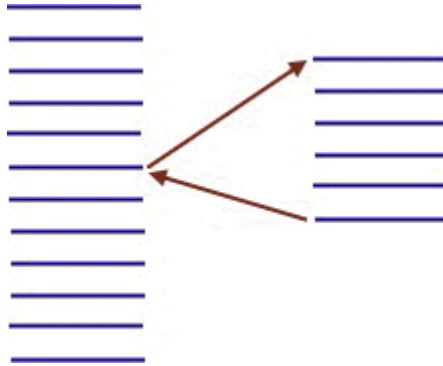
Các tín hiệu được gửi đi bởi :

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

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



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



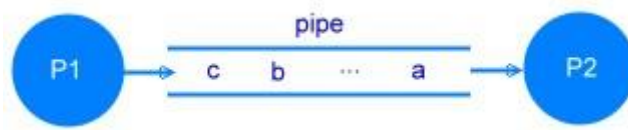
**Hình 3.1** Liên lạc bằng tín hiệu

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

## II.2. Pipe

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

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



**Hình 3.2** Liên lạc qua pipe

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

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

✚ **Thảo luận:** Liên lạc bằng pipe là một cơ chế liên lạc *một chiều (unidirectional)*, nghĩa là một tiến trình kết nối với một pipe chỉ có thể thực hiện một trong hai thao tác đọc hoặc ghi, nhưng không thể thực hiện cả hai. Một số hệ điều hành cho phép thiết lập hai pipe giữa một cặp tiến trình để tạo liên lạc hai chiều. Trong những hệ thống đó, có nguy cơ xảy ra tình trạng *tắc nghẽn (deadlock)* : một pipe bị giới

hạn về kích thước, do vậy nếu cả hai pipe nối kết hai tiến trình đều đầy(hoặc đều trống) và cả hai tiến trình đều muốn ghi (hay đọc) dữ liệu vào pipe(mỗi tiến trình ghi dữ liệu vào một pipe), chúng sẽ cùng bị khóa và chờ lẫn nhau mãi mãi !

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

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

### II.3. Vùng nhớ chia sẻ

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

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



**Hình 3.3** Liên lạc qua vùng nhớ chia sẻ

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

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

### II.4. Trao đổi thông điệp (Message)

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

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

Nếu hai tiến trình P và Q muốn liên lạc với nhau, cần phải thiết lập một môi liên kết giữa hai tiến trình, sau đó P, Q sử dụng các hàm IPC thích hợp để trao đổi thông điệp, cuối cùng khi sự liên lạc chấm dứt môi liên kết giữa hai tiến trình sẽ bị hủy. Có nhiều cách thức để thực hiện sự liên kết giữa hai tiến trình và cài đặt các theo tác send /receive tương ứng : liên lạc trực tiếp hay gián tiếp, liên lạc đồng bộ hoặc

không đồng bộ , kích thước thông điệp là cố định hay không ... Nếu các tiến trình liên lạc theo kiểu liên kết tường minh, các hàm Send và Receive sẽ được cài đặt với tham số :

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

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

## II.5. Sockets

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

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

Các thuộc tính của socket:

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

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

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

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

- hai tiến trình liên lạc với nhau không kết nối trực tiếp
- mỗi thông điệp phải kèm theo địa chỉ người nhận.

Hình thức liên lạc này có đặc điểm được :

- người gửi không chắc chắn thông điệp của họ được gửi đến người nhận,
- một thông điệp có thể được gửi nhiều lần,
- hai thông điệp được gửi theo một thứ tự nào đó có thể đến tay người nhận theo một thứ tự khác.

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

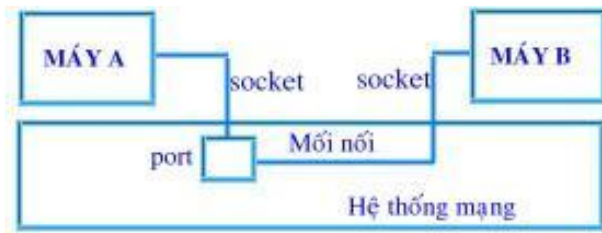
### **b) Liên lạc trong chế độ nối kết:**

Một liên kết được thành lập giữa hai tiến trình. Trước khi mỗi liên kết này được thiết lập, một trong hai tiến trình phải đợi có một tiến trình khác yêu cầu kết nối. Có thể sử dụng socket để liên lạc theo mô hình client-serveur. Trong mô hình này, server sử dụng lời gọi hệ thống listen và accept để nối kết với client, sau đó, client và server có thể trao đổi thông tin bằng cách sử dụng các primitive send và receive.

- Hủy một socket

▶ Ví dụ :

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



**Hình 3.4** Các socket và port trong mối nối TCP.

Hình 3.4 minh họa một cách giao tiếp giữa hai máy tính trong nghi thức truyền thông TCP. Máy A tạo ra một socket và kết buộc (bind) socket này với một port X (tức là một số nguyên dương có ý nghĩa cục bộ trong máy A), trong khi đó máy B tạo một socket khác và móc vào (connect) port X trong máy A.

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

## **III. Nhu cầu đồng bộ hóa (synchronisation)**

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

### **III.1. Yêu cầu độc quyền truy xuất (Mutual exclusion)**

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

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

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

### III.2. Yêu cầu phối hợp (Synchronization)

Nhìn chung, mối tương quan về tốc độ thực hiện của hai tiến trình trong hệ thống là không thể biết trước, vì điều này phụ thuộc vào nhiều yếu tố động như tần suất xảy ra các ngắt của từng tiến trình, thời gian tiến trình được cấp phát bộ xử lý... Có thể nói rằng các tiến trình hoạt động không đồng bộ với nhau. Nhưng có những tình huống các tiến trình cần hợp tác trong việc hoàn thành tác vụ, khi đó cần phải đồng bộ hóa hoạt động của các tiến trình, ví dụ một tiến trình chỉ có thể xử lý nếu một tiến trình khác đã kết thúc một công việc nào đó ...

### III.3. Bài toán đồng bộ hoá

#### \*III.3.1. Vấn đề tranh đoạt điều khiển (race condition)

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

```
if (taikhoan - tienrut >=0)
    taikhoan = taikhoan - tienrut;
else
    error (« không thể rút tiền ! »);
```

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

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

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

#### \*III.3.2. Miền găng (critical section)

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

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

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

của mỗi tiến trình tạo thành một miền găng.

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

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

- Không có hai tiến trình cùng ở trong miền găng cùng lúc.
- Không có giả thiết nào đặt ra cho sự liên hệ về tốc độ của các tiến trình, cũng như về số lượng bộ xử lý trong hệ thống.
- Một tiến trình tạm dừng bên ngoài miền găng không được ngăn cản các tiến trình khác vào miền găng.
- Không có tiến trình nào phải chờ vô hạn để được vào miền găng.

#### IV. Tóm tắt

▶ Một số tiến trình trong hệ thống có nhu cầu trao đổi thông tin để phối hợp hoạt động, do mỗi tiến trình có một không gian địa chỉ độc lập nên việc liên lạc chỉ có thể thực hiện thông qua các cơ chế do hệ điều hành cung cấp.

▶ Một số cơ chế trao đổi thông tin giữa các tiến trình :

- *Tín hiệu* : thông báo sự xảy ra của một sự kiện
- *Pipe* : truyền dữ liệu không cấu trúc
- *Vùng nhớ chia sẻ* : cho phép nhiều tiến trình truy cập đến cùng một vùng nhớ
- *Trao đổi thông điệp* : truyền dữ liệu có cấu trúc, có thể vận dụng trong các hệ phân tán
- *Socket* : chuẩn hoá việc liên lạc giữa các hệ thống khác biệt

▶ Khi các tiến trình trao đổi thông tin, chia sẻ tài nguyên chung, cần phải đồng bộ hoá hoạt động của chúng chủ yếu do yêu cầu độc quyền truy xuất hoặc phối hợp hoạt động.

▶ Miền găng là đoạn lệnh trong chương trình có khả năng phát sinh mâu thuẫn truy xuất. Để không xảy ra mâu thuẫn truy xuất, cần đảm bảo tại một thời điểm chỉ có một tiến trình được vào miền găng.

#### Củng cố bài học

Các câu hỏi cần trả lời được sau bài học này :

1. Các cơ chế trao đổi thông tin : tình huống sử dụng, ưu, khuyết ?
2. Các yêu cầu đồng bộ hoá ?

#### Bài tập

Phân tích các bài toán sau đây và xác định những yêu cầu đồng bộ hoá, miền găng :

##### Bài 1. Bài toán Tạo phân tử H<sub>2</sub>O

Đồng bộ hoạt động của một phòng thí nghiệm sử dụng nhiều tiến trình đồng hành sau để tạo các phân tử H<sub>2</sub>O:

```
MakeH () // Mỗi tiến trình MakeH tạo 1 nguyên tử H
{
    Make-Hydro ();
}
MakeO () // Mỗi tiến trình MakeO tạo 1 nguyên tử O
{
    Make-Oxy ();
}
```

```

MakeWater() /* Tiến trình MakeWater hoạt động đồng hành
với các tiến trình MakeH, MakeO, chờ có đủ 2 H và 1 O để tạo H2O */
{
    while (T)
        Make-Water(); //Tạo 1 phân tử H2O
}

```

## Bài 2. Bài toán Cây cầu cũ

Để tránh sự ùn ứ, người ta chỉ 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()** 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 3 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.*/
{
    RuntoBridge(); // Đi về phía cầu
    ArriveBridge(direction) ;
    PassBridge(); // Qua cầu
    Exit Bridge() ;
    RunfromBridge(); // Đã qua cầu
}

```

## Bài 3. Bài toán Qua sông

Để vượt qua sông, các nhân viên Microsoft 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 :

- Không chấp nhận 3 nhân viên Microsoft và 1 Linux hacker trên cùng một chiếc thuyền.
- Ngược lại, không chấp nhận 3 Linux hacker và 1 nhân viên Microsoft trên cùng một chiếc thuyền.
- Tất cả các trường hợp kết hợp khác đều hợp pháp.
- 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 lượng 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  
}
```

#### Bài 4. Bài toán Điều phối hành khách xe bus

Hãy tưởng tượng bạn chịu trách nhiệm kiểm soát hành khách lên xe bus tại một trạm dừng.

Mỗi xe bus có đủ chỗ cho 10 hành khách. Trong đó 4 chỗ chỉ dành cho khách ngồi xe lăn, 6 chỗ còn lại chỉ dành cho khách bình thường.

Công việc của bạn là cho khách lên xe theo đúng qui định chỗ, khi xe đầy khách 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 của bạn cho 1 chiếc xe bus được mô tả qua tiến trình

**GetPassengers()**; hoạt động của mỗi hành khách tùy loại được mô tả lần lượt bằng tiến trình **WheelPassenger()** và **NonWheelPassenger()** sau đây, hãy sửa chữa các đoạn code, sử dụng cơ chế semaphore để thực hiện các nguyên tắc đồng bộ hoá cần thiết.

```
GetPassenger()
```

```
{  
    ArriveTerminal(); // tiếp nhận một xe vào bến  
    OpenDoor(); // mở cửa xe, thủ tục này xem như đã có  
    for (int i=0; i<4; i++) // tiếp nhận các hành khách ngồi xe lăn  
    {  
        ArrangeSeat(); // đưa 1 khách vào chỗ  
    }  
    for (int i=0; i<6; i++) // tiếp nhận các hành khách bình thường  
    {  
        ArrangeSeat(); // đưa 1 khách vào chỗ  
    }  
    CloseDoor(); // đóng cửa xe, thủ tục này xem như đã có  
    DepartTerminal(); // cho một xe rời bến  
}
```

```
WheelPassenger()
```

```
{
```

```

        ArriveTerminal(); // đến bến
        GetOnBus(); // lên xe
    }
    NonWheelPassenger()
    {
        ArriveTerminal(); // đến bến
        GetOnBus(); // lên xe
    }

```

### Bài 5. Bài toán sản xuất thiết bị xe hơi

Hãng Pontiac có 2 bộ phận hoạt động song song :

- Bộ phận sản xuất 1 khung xe :

```

        MakeChassis() { // tạo khung xe
            Produce_chassis();
        }

```

- Bộ phận sản xuất 1 bánh xe :

```

        MakeTires() { // 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 :

- o Sản xuất một khung xe,
- o cần có đủ 4 bánh xe cho 1 khung xe được sản xuất ra, sau đó mới tiếp tục sản xuất khung xe khác...


## BÀI 5 : CÁC GIẢI PHÁP ĐỒNG BỘ HOÁ

Bài học này sẽ giới thiệu các giải pháp cụ thể để xử lý bài toán đồng bộ hoá. Có nhiều giải pháp để thực hiện việc truy xuất miền găng, các giải pháp này được phân biệt thành hai lớp tùy theo cách tiếp cận trong xử lý của tiến trình bị khóa : các giải pháp « busy waiting » và các giải pháp « sleep and wakeup ».

### I. GIẢI PHÁP « BUSY WAITING »


#### I.1. Các giải pháp phần mềm

##### I.1.1. Sử dụng các biến cờ hiệu:


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

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

**Hình 3.5** Cấu trúc một chương trình sử dụng biến khóa để đồng bộ

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

##### I.1.2. Sử dụng việc kiểm tra luân phiên :

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

```

while (TRUE) {

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

}

```

(a) Cấu trúc tiến trình A

```

while (TRUE) {

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

}

```

(b) Cấu trúc tiến trình B

**Hình 3.6** Cấu trúc các tiến trình trong giải pháp kiểm tra luân phiên

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

### \*1.1.3. Giải pháp của Peterson

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

```

int turn; // đến phiên ai

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

```

Nếu  $interesse[i] = TRUE$  có nghĩa là tiến trình  $P_i$  muốn vào miền găng. Khởi đầu,  $interesse[0]=interesse[1]=FALSE$  và giá trị của  $turn$  được khởi động là 0 hay 1. Để có thể vào được miền găng, trước tiên tiến trình  $P_i$  đặt giá trị  $interesse[i]=TRUE$  ( xác định rằng tiến trình muốn vào miền găng), sau đó đặt  $turn=j$  (đề nghị thử tiến trình khác vào miền găng). Nếu tiến trình  $P_j$  không quan

tâm đến việc vào miền găng ( $interesse[j]=FALSE$ ), thì  $P_i$  có thể vào miền găng, nếu không,  $P_i$  phải chờ đến khi  $interesse[j]=FALSE$ . Khi tiến trình  $P_i$  rời khỏi miền găng, nó đặt lại giá trị cho  $interesse[i]=FALSE$ .

```
while (TRUE) {

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

}
```

**Hình 3.7** Cấu trúc tiến trình  $P_i$  trong giải pháp Peterson

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

## 📖 I.2. Các giải pháp phân cứng

### \*I.2.1. Cắm ngắt:

✚ Tiếp cận: cho phép tiến trình cấm tất cả các ngắt trước khi vào miền găng, và phục hồi ngắt khi ra khỏi miền găng. Khi đó, ngắt đồng hồ cũng không xảy ra, do vậy hệ thống không thể tạm dừng hoạt động của tiến trình đang xử lý để cấp phát CPU cho tiến trình khác, nhờ đó tiến trình hiện hành yên tâm thao tác trên miền găng mà không sợ bị tiến trình nào khác tranh chấp.

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

### \*I.2.2. Chỉ thị TSL (Test-and-Set):

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

```
Test-and-Setlock (boolean target)
{
```

```

Test-and-Setlock = target;
target = TRUE;

}

```

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

```

while (TRUE) {

    while (Test-and-Setlock(lock));
    critical-section ();
    lock = FALSE;
    Noncritical-section ();

}

```

**Hình 3.8** Cấu trúc một chương trình trong giải pháp TSL

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

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

## II. CÁC GIẢI PHÁP « SLEEP AND WAKEUP »

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

*SLEEP* là một lời gọi hệ thống có tác dụng tạm dừng hoạt động của tiến trình (blocked) gọi nó và chờ đến khi được một tiến trình khác « đánh thức ». Lời gọi hệ thống *WAKEUP* nhận một tham số duy nhất : tiến trình sẽ được tái kích hoạt (đặt về trạng thái ready).

Ý tưởng sử dụng *SLEEP* và *WAKEUP* như sau : khi một tiến trình chưa đủ điều kiện vào miền găng, nó gọi *SLEEP* để tự khóa đến khi có một tiến trình khác gọi *WAKEUP* để giải phóng cho nó. Một tiến trình

gọi *WAKEUP* khi ra khỏi miền găng để đánh thức một tiến trình đang chờ, tạo cơ hội cho tiến trình này vào miền găng :

```
int busy;    // 1 nếu miền găng đang bị chiếm, nếu không là 0
int blocked; // đếm số lượng tiến trình đang bị khóa

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

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


**Hình 3.9** Cấu trúc chương trình trong giải pháp SLEEP and WAKEUP

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

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

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

## II.1. Semaphore

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

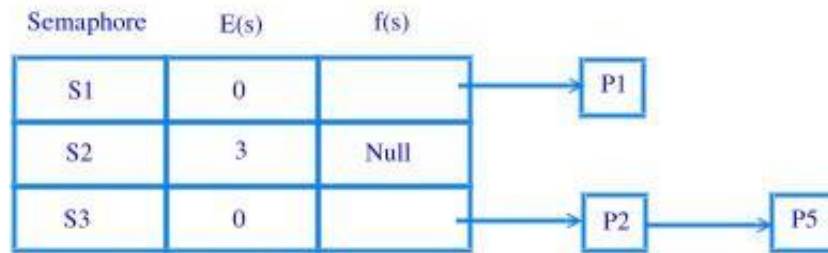
- Một giá trị nguyên dương  $e(s)$
- Một hàng đợi  $f(s)$  lưu danh sách các tiến trình đang bị khóa (chờ) trên semaphore *s*



- Chỉ có hai thao tác được định nghĩa trên semaphore

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

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



**Hình 3.10** Semaphore  $s$

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

**Down (s)** :

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

**Up (s)** :


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

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

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


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

các tiến trình.

 **Tổ chức truy xuất độc quyền với Semaphores:** khái niệm *semaphore* cho phép bảo đảm nhiều tiến trình cùng truy xuất đến miền găng mà không có sự mâu thuẫn truy xuất.  $n$  tiến trình cùng sử dụng một semaphore  $s$ ,  $e(s)$  được khởi gán là 1. Để thực hiện đồng bộ hóa, tất cả các tiến trình cần phải áp dụng cùng cấu trúc chương trình sau đây:

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

**Hình 3.11** Cấu trúc một chương trình trong giải pháp semaphore

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


**P1 :**

```
while (TRUE) {  
    job1 ();  
    Up(s); //đánh thức P2  
}
```

**P2 :**

```
while (TRUE) {  
    Down(s); // chờ P1  
    job2 ();  
}
```

**Hình 3.12** Cấu trúc chương trình trong giải pháp semaphore

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

Ví dụ :

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

}

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

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

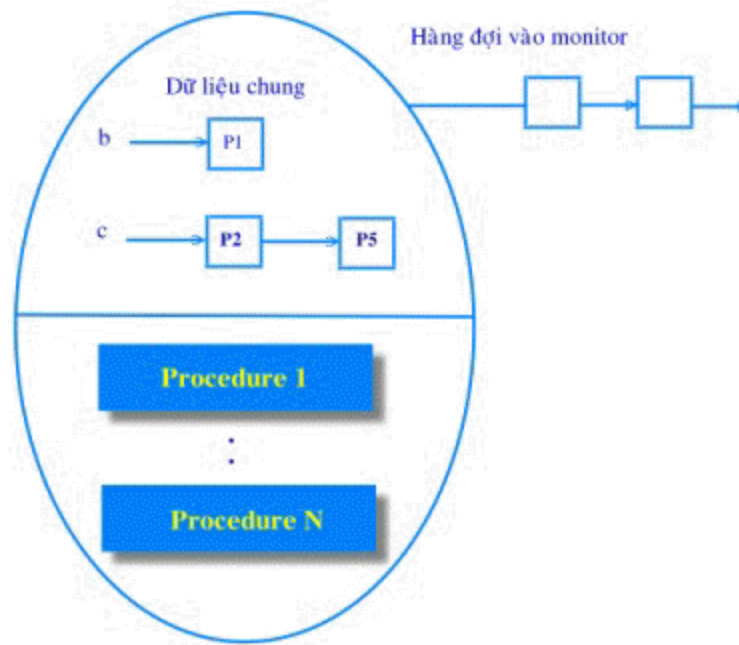
## II.2. Monitors

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

- Các biến và cấu trúc dữ liệu bên trong monitor chỉ có thể được thao tác bởi các thủ tục định nghĩa bên trong monitor đó. (*encapsulation*).
- Tại một thời điểm, chỉ có một tiến trình duy nhất được hoạt động bên trong một monitor (*mutual exclusive*).
- Trong một monitor, có thể định nghĩa các *biến điều kiện* và hai thao tác kèm theo là **wait** và **signal** như sau : gọi *c* là biến điều kiện được định nghĩa trong monitor:

▶ **wait(c)** : chuyển trạng thái tiến trình gọi sang blocked , và đặt tiến trình này vào hàng đợi trên biến điều kiện *c*.

▶ **signal(c)** : nếu có một tiến trình đang bị khóa trong hàng đợi của *c*, tái kích hoạt tiến trình đó, và tiến trình gọi sẽ rời khỏi monitor.



**Hình 3.13** Monitor và các biến điều kiện

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

• **Wait( $c$ ) :**

```
status(P) = blocked;  
enter(P, f(c));
```

**Signal( $c$ ) :**

```
if (f(c) != NULL) {  
  
    exit(Q, f(c)); //Q là tiến trình chờ trên c  
    status(Q) = ready;  
    enter(Q, ready-list);  
  
}
```

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

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

```

procedure Action1() ;
{

}

....

procedure Actionn() ;
{

}

end monitor;

```

**Hình 3.14** Cấu trúc một monitor

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

```

while (TRUE) {

    Noncritical-section ();
    <monitor>.Actioni; //critical-section();
    Noncritical-section ();

}

```

**Hình 3.15** Cấu trúc tiến trình P<sub>i</sub> trong giải pháp monitor

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

### II.3. Trao đổi thông điệp

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

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

✚ Sử dụng: Có nhiều cách thức để thực hiện việc truy xuất độc quyền bằng cơ chế trao đổi thông điệp. Đây là một mô hình đơn giản: một tiến trình kiểm soát việc sử dụng tài nguyên và nhiều tiến trình khác yêu cầu tài nguyên này. Tiến trình có yêu cầu tài nguyên sẽ gửi một thông điệp đến tiến trình kiểm soát và sau đó chuyển sang trạng thái blocked cho đến khi nhận được một thông điệp chấp nhận cho truy

xuất từ tiến trình kiểm soát tài nguyên. Khi sử dụng xong tài nguyên, tiến trình gửi một thông điệp khác đến tiến trình kiểm soát để báo kết thúc truy xuất. Về phần tiến trình kiểm soát, khi nhận được thông điệp yêu cầu tài nguyên, nó sẽ chờ đến khi tài nguyên sẵn sàng để cấp phát thì gửi một thông điệp đến tiến trình đang bị khóa trên tài nguyên đó để đánh thức tiến trình này.

```
while (TRUE) {

    Send(process controller, request message);
    Receive(process controller, accept message);
    critical-section ();
    Send(process controller, end message);
    Noncritical-section ();

}
```

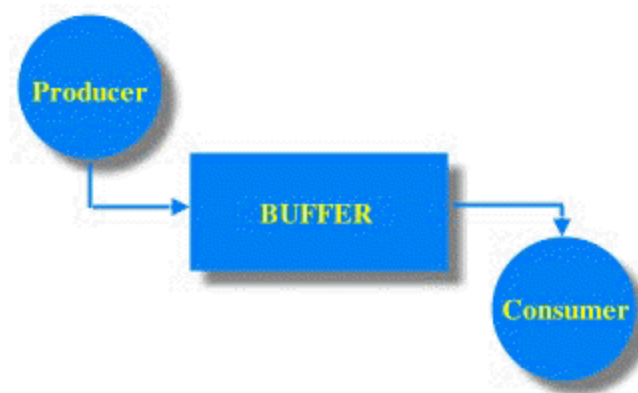
**Hình 3.16** Cấu trúc tiến trình yêu cầu tài nguyên trong giải pháp message

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

### III. CÁC VẤN ĐỀ CỐ ĐIỂN CỦA ĐỒNG BỘ HOÁ

#### III.1. Vấn đề Người sản xuất – Người tiêu thụ (Producer-Consumer)

📖 Vấn đề: hai tiến trình cùng chia sẻ một bộ đệm có kích thước giới hạn. Một trong hai tiến trình đóng vai trò người sản xuất – tạo ra dữ liệu và đặt dữ liệu vào bộ đệm- và tiến trình kia đóng vai trò người tiêu thụ – lấy dữ liệu từ bộ đệm ra để xử lý.



**Hình 3.17** Producer và Consumer

Để đồng bộ hóa hoạt động của hai tiến trình sản xuất tiêu thụ cần tuân thủ các quy định sau :

- Tiến trình sản xuất (producer) không được ghi dữ liệu vào bộ đệm đã đầy.(*synchronisation*)
- Tiến trình tiêu thụ (consumer) không được đọc dữ liệu từ bộ đệm đang trống.(*synchronisation*)
- Hai tiến trình sản xuất và tiêu thụ không được thao tác trên bộ đệm cùng lúc . (*exclusion mutuelle*)

## 🚧Giải pháp:

### \*III.1.1. Semaphore

Sử dụng ba semaphore : **full**, đếm số chỗ đã có dữ liệu trong bộ đệm; **empty**, đếm số chỗ còn trống trong bộ đệm; và **mutex**, kiểm tra việc Producer và Consumer không truy xuất đồng thời đến bộ đệm.

```

BufferSize = 3;                // số chỗ trong bộ đệm

semaphore mutex = 1;           // kiểm soát truy xuất độc quyền
semaphore empty = BufferSize;   // số chỗ trống
semaphore full = 0;             // số chỗ đầy

Producer()
{
    int item;
    while (TRUE) {
        produce_item(&item);    // tạo dữ liệu mới
        down(&empty);            // giảm số chỗ trống
        down(&mutex);            // báo hiệu vào miền găng
        enter_item(item);        // đặt dữ liệu vào bộ đệm
        up(&mutex);              // ra khỏi miền găng
        up(&full);               // tăng số chỗ đầy
    }
}

Consumer()
{
    int item;

    while (TRUE) {
        down(&full);             // giảm số chỗ đầy
        down(&mutex);            // báo hiệu vào miền găng
        remove_item(&item);      // lấy dữ liệu từ bộ đệm
        up(&mutex);              // ra khỏi miền găng
        up(&empty);              // tăng số chỗ trống
        consume_item(item);      // xử lý dữ liệu
    }
}

```

### \*III.1.2. Monitor

Định nghĩa một monitor **ProducerConsumer** với hai thủ tục **enter** và **remove** thao tác trên bộ đệm. Xử lý của các thủ tục này phụ thuộc vào các biến điều kiện **full** và **empty**.

```

monitor ProducerConsumer
    condition full, empty;

```



```

    int count;

    procedure enter();
    {
        if (count == N)
            wait(full);          // nếu bộ đệm đầy, phải chờ
        enter_item(item);        // đặt dữ liệu vào bộ đệm
        count++;                 // tăng số chỗ đầy
        if (count == 1)          // nếu bộ đệm không trống
            signal(empty);       // thì kích hoạt Consumer
    }
    procedure remove();
    {
        if (count == 0)
            wait(empty);         // nếu bộ đệm trống, chờ
        remove_item(&item);      // lấy dữ liệu từ bộ đệm
        count--;                 // giảm số chỗ đầy
        if (count == N-1)        // nếu bộ đệm không đầy
            signal(full);        // thì kích hoạt Producer
    }
    count = 0;
    end monitor;

    Producer();
    {
        while (TRUE)
        {
            produce_item(&item);
            ProducerConsumer.enter;
        }
    }
    Consumer();
    {
        while (TRUE)
        {
            ProducerConsumer.remove;
            consume_item(item);
        }
    }
}

```

### \*III.1.3. Trao đổi thông điệp

Thông điệp *empty* hàm ý có một chỗ trống trong bộ đệm. Tiến trình Consumer bắt đầu công việc bằng cách gửi 4 thông điệp *empty* đến Producer. Tiến trình Producer tạo ra một dữ liệu mới và chờ đến khi nhận được một thông điệp *empty* thì gửi ngược lại cho Consumer một thông điệp chứa dữ liệu. Tiến trình Consumer chờ nhận thông điệp chứa dữ liệu, và sau khi xử lý xong dữ liệu này, Consumer sẽ lại gửi một thông điệp *empty* đến Producer, ...

```

BufferSize = 4;

Producteur()
{
    int item;
    message m;                // thông điệp

```

```

while (TRUE) {
    produce_item(&item);
    receive(consumer, &m);           // chờ thông điệp empty
    create_message(&m, item);        // tạo thông điệp dữ liệu
    send(consumer, &m);              // gửi dữ liệu đến Consumer
}

Consumer()
{
    int item;
    message m;

    for(0 to N)
        send(producer, &m); // gửi N thông điệp empty
    while (TRUE) {
        receive(producer, &m); // chờ thông điệp dữ liệu
        remove_item(&m, &item); // lấy dữ liệu từ thông điệp
        send(producer, &m);      // gửi thông điệp empty
        consumer_item(item);     // xử lý dữ liệu
    }
}

```

## III.2. Mô hình Readers-Writers

**Vấn đề** : Nhiều tiến trình đồng thời sử dụng một cơ sở dữ liệu. Các tiến trình chỉ cần lấy nội dung của cơ sở dữ liệu được gọi là các tiến trình Reader, nhưng một số tiến trình khác lại có nhu cầu sửa đổi, cập nhật dữ liệu trong cơ sở dữ liệu chung này, chúng được gọi là các tiến trình Writer. Các quy định đồng bộ hóa việc truy xuất cơ sở dữ liệu cần tuân thủ là :

- Không cho phép một tiến trình Writer cập nhật dữ liệu trong cơ sở dữ liệu khi các tiến trình Reader khác đang truy xuất nội dung cơ sở dữ liệu.. (*synchronisation*)
- Tại một thời điểm , chỉ cho phép một tiến trình Writer được sửa đổi nội dung cơ sở dữ liệu. (*mutuelle exclusion*).

**Giải pháp:**

### \*III.2.1. Semaphore

Sử dụng một biến chung **rc** để ghi nhớ số lượng các tiến trình Reader muốn truy xuất cơ sở dữ liệu. Hai semaphore cũng được sử dụng : **mutex**, kiểm soát sự truy cập đến **rc**; và **db**, kiểm tra sự truy xuất độc quyền đến cơ sở dữ liệu.

```

semaphore mutex = 1; // Kiểm tra truy xuất rc
semaphore db = 1;    // Kiểm tra truy xuất cơ sở dữ liệu
int rc;              // Số lượng tiến trình Reader
Reader()
{
    while (TRUE) {

```

```

    down(&mutex);          // giành quyền truy xuất rc
    rc = rc + 1;            // thêm một tiến trình Reader
    if (rc == 1)           // nếu là Reader đầu tiên thì
        down(&db);        // cấm Writer truy xuất dữ liệu
    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)          // nếu là Reader cuối cùng thì
        up(&db);          // cho phép Writer truy xuất db
    up(&mutex);            // chấm dứt truy xuất rc
    use_data_read();
}
}

Writer()
{
    while (TRUE) {
        create_data();
        down(&db);        // giành quyền truy xuất db
        write_database(); // cập nhật dữ liệu
        up(&db);          // chấm dứt truy xuất db
    }
}

```

### **\*III.2.2. Monitor**

Sử dụng một biến chung **rc** để ghi nhớ số lượng các tiến trình Reader muốn truy xuất cơ sở dữ liệu. 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
    condition OKWrite, OKRead;
    int rc = 0;
    Boolean busy = false;

    procedure BeginRead()
    {
        if (busy) // nếu db đang bận, chờ
            wait(OKRead);
        rc++;    // thêm một Reader
        signal(OKRead);
    }
    procedure FinishRead()
    {
        rc--;    // bớt một Reader
        if (rc == 0) // nếu là Reader cuối cùng
            signal(OKWrite); // thì cho phép Writer
                           // truy xuất db
    }
    procedure BeginWrite()
    {
        if (busy || rc != 0) // nếu db đang bận, hay một
            wait(OKWrite); // Reader đang đọc db, chờ
    }

```

```

    busy = true;
}
procedure FinishWrite()
{
    busy = false;
    If (OKRead.Queue)
        signal(OKRead);
    else
        signal(OKWrite);
}
Reader()
{
    while (TRUE)
    {
        ReaderWriter.BeginRead();
        Read_database();
        ReaderWriter.FinishRead();
    }
}
Writer()
{
    while (TRUE)
    {
        create_data(&info);
        ReaderWriter.BeginWrite();
        Write_database();
        ReaderWriter.FinishWrite();
    }
}

```

### **\*III.2.3. Trao đổi thông điệp**

Cần có một tiến trình server điều khiển việc truy xuất cơ sở dữ liệu.

Các tiến trình Writer và Reader gọi các thông điệp yêu cầu truy xuất đến server và nhận từ server các thông điệp hồi đáp tương ứng .

```

Reader()
{
    while (TRUE) {
        send (server, RequestRead);
        receive (server, value);
        print(value); }
}

Writer()
{
    while (TRUE) {
        create_data(&value);
        send (server, RequestWrite,value);
        receive (server,OKWrite); }
}

```

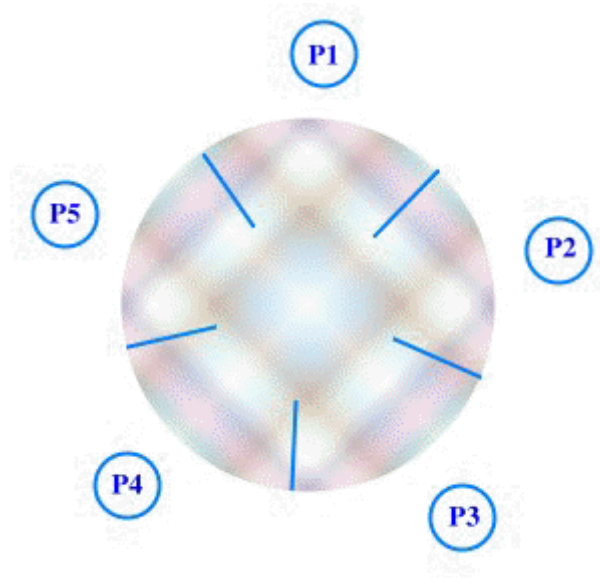
## IV. TẮC NGHẼN (DEADLOCK)

### IV.1. Định nghĩa:

Một tập hợp các tiến trình được định nghĩa ở trong tình trạng *tắc nghẽn* khi mỗi tiến trình trong tập hợp đều chờ đợi một sự kiện mà chỉ có một tiến trình khác trong tập hợp mới có thể phát sinh được.

Nói cách khác, mỗi tiến trình trong tập hợp đều chờ được cấp phát một tài nguyên hiện đang bị một tiến trình khác cũng ở trạng thái blocked chiếm giữ. Như vậy 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 !

**Vấn đề Bữa ăn tối của các triết gia :** 5 nhà triết học cùng ngồi ăn tối với món spaghetti nổi tiếng. Mỗi nhà triết học cần dùng 2 cái nĩa để có thể ăn spaghetti. Nhưng trên bàn chỉ có tổng cộng 5 cái nĩa để xen kẽ với 5 cái đĩa. Mỗi nhà triết học sẽ suy ngẫm các triết lý của mình đến khi cảm thấy đói thì dự định lần lượt cầm 1 cái nĩa bên trái và 1 cái nĩa bên phải để ăn. Nếu cả 5 nhà triết học đều cầm cái nĩa bên trái cùng lúc, thì sẽ không có ai có được cái nĩa bên phải để có thể bắt đầu thưởng thức spaghetti. Đây chính là tình trạng *tắc nghẽn*.



Hình 3.18 Bữa ăn tối của các triết gia

### IV.2. Điều kiện xuất hiện tắc nghẽn

Coffman, Elphick và Shoshani đã đưa ra 4 điều kiện cần có thể làm xuất hiện tắc nghẽn:

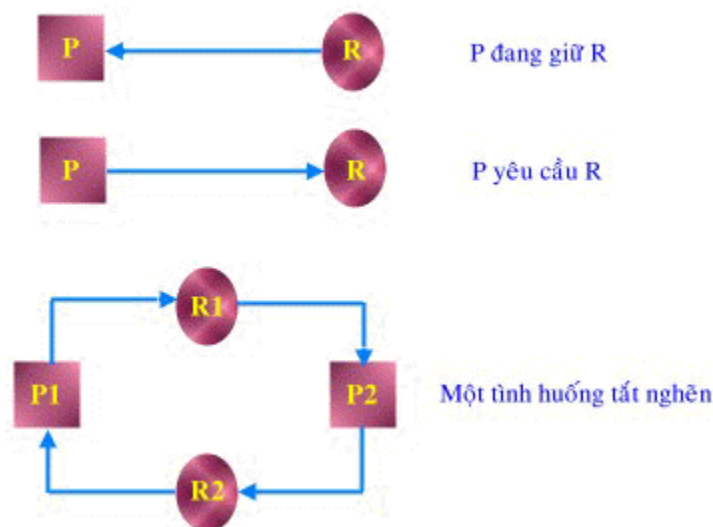
- Có sử dụng tài nguyên không thể chia sẻ (Mutual exclusion): Mỗi thời điểm, một tài nguyên không thể chia sẻ được hệ thống cấp phát chỉ cho một tiến trình, khi tiến trình sử dụng xong tài nguyên này, hệ thống mới thu hồi và cấp phát tài nguyên cho tiến trình khác.

- Sự chiếm giữ và yêu cầu thêm tài nguyên (Wait for): Các tiến trình tiếp tục chiếm giữ các tài nguyên đã cấp phát cho nó trong khi chờ được cấp phát thêm một số tài nguyên mới.
- Không thu hồi tài nguyên từ tiến trình đang giữ chúng (No preemption): Tài nguyên không thể được thu hồi từ tiến trình đang chiếm giữ chúng trước khi tiến trình này sử dụng chúng xong.
- Tồn tại một chu kỳ trong đồ thị cấp phát tài nguyên (Circular wait): có ít nhất hai tiến trình chờ đợi lẫn nhau : tiến trình này chờ được cấp phát tài nguyên đang bị tiến trình kia chiếm giữ và ngược lại.

Khi có đủ 4 điều kiện này, thì tắc nghẽn xảy ra. Nếu thiếu một trong 4 điều kiện trên thì không có tắc nghẽn.

### IV.3. Đồ thị cấp phát tài nguyên

Có thể sử dụng một đồ thị để mô hình hóa việc cấp phát tài nguyên. Đồ thị này có 2 loại nút : các tiến trình được biểu diễn bằng hình tròn, và mỗi tài nguyên được hiển thị bằng hình vuông



Hình 3.19 Đồ thị cấp phát tài nguyên

### IV.4. Các phương pháp xử lý tắc nghẽn

Chủ yếu có ba hướng tiếp cận để xử lý tắc nghẽn :

- Sử dụng một nghi thức (protocol) để bảo đảm rằng hệ thống không bao giờ xảy ra tắc nghẽn.
- Cho phép xảy ra tắc nghẽn và tìm cách sửa chữa tắc nghẽn.
- Hoàn toàn bỏ qua việc xử lý tắc nghẽn, xem như hệ thống không bao giờ xảy ra tắc nghẽn.

### IV.5. Ngăn chặn tắc nghẽn

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

- Tài nguyên không thể chia sẻ : nhìn chung gần như không thể tránh được điều kiện này vì bản chất tài nguyên gần như cố định. Tuy nhiên đối với một số tài nguyên về kết xuất, người ta có thể dùng các cơ chế spooling để biến đổi thành tài nguyên có thể chia sẻ.

- Sự chiếm giữ và yêu cầu thêm tài nguyên: phải bảo đảm rằng mỗi khi tiến trình yêu cầu thêm một tài nguyên thì nó không chiếm giữ các tài nguyên khác. Có thể áp đặt một trong hai cơ chế truy xuất sau :

- ▶ Tiến trình phải yêu cầu tất cả các tài nguyên cần thiết trước khi bắt đầu xử lý .

=> phương pháp này có khó khăn là tiến trình khó có thể ước lượng chính xác tài nguyên cần sử dụng vì có thể nhu cầu phụ thuộc vào quá trình xử lý . Ngoài ra nếu tiến trình chiếm giữ sẵn các tài nguyên chưa cần sử dụng ngay thì việc sử dụng tài nguyên sẽ kém hiệu quả.

- ▶ 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 làm phát sinh các khó khăn trong việc bảo vệ tính toàn vẹn dữ liệu của hệ thống.

- Không thu hồi tài nguyên: 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 khoá. Tuy nhiên với một số loại tài nguyên, việc thu hồi sẽ rất khó khăn vì vi phạm sự toàn vẹn dữ liệu .

- Tồn tại một chu kỳ: tránh tạo chu kỳ trong đồ thị bằng cách cấp phát tài nguyên theo một sự phân cấp như sau :

gọi  $R = \{R_1, R_2, \dots, R_m\}$  là tập các loại tài nguyên.

Các loại tài nguyên được phân cấp từ 1-N.

Ví dụ :  $F(\text{đĩa}) = 2, F(\text{máy in}) = 12$

Các tiến trình khi yêu cầu tài nguyên phải tuân thủ quy định : khi tiến trình đang chiếm giữ tài nguyên  $R_i$  thì chỉ có thể yêu cầu các tài nguyên  $R_j$  nếu  $F(R_j) > F(R_i)$ .

#### IV.6. Tránh tắc nghẽn

Ngăn cản tắc nghẽn là một mối bận tâm lớn khi sử dụng tài nguyên. Tránh tắc nghẽn là loại bỏ tất cả các cơ hội có thể dẫn đến tắc nghẽn trong tương lai. Cần phải sử dụng những cơ chế phức tạp để thực hiện ý định này.

#### Một số khái niệm cơ sở

- **Trạng thái an toàn** : trạng thái A là an toàn nếu hệ thống có thể thỏa mãn các nhu cầu tài



nguyên (cho đến tối đa) của mỗi tiến trình theo một thứ tự nào đó mà vẫn ngăn chặn được tắc nghẽn.

• **Một chuỗi cấp phát an toàn:** một thứ tự của các tiến trình  $\langle P_1, P_2, \dots, P_n \rangle$  là an toàn đối với tình trạng cấp phát hiện hành nếu với mỗi tiến trình  $P_i$  nhu cầu tài nguyên của  $P_i$  có thể được thỏa mãn với các tài nguyên còn tự do của hệ thống, cộng với các tài nguyên đang bị chiếm giữ bởi các tiến trình  $P_j$  khác, với  $j < i$ .

Một trạng thái an toàn không thể là trạng thái tắc nghẽn. Ngược lại một trạng thái không an toàn có thể dẫn đến tình trạng tắc nghẽn.

👉 **Chiến lược cấp phát :** chỉ thỏa mãn yêu cầu tài nguyên của tiến trình khi trạng thái kết quả là an toàn!

👉 **Giải thuật xác định trạng thái an toàn**

Cần sử dụng các cấu trúc dữ liệu sau :

```
int Available[NumResources];
/* Available[r]= số lượng các thể hiện còn tự do của tài nguyên r */
int Max[NumProcs, NumResources];
/* Max[p,r]= nhu cầu tối đa của tiến trình p về tài nguyên r */
int Allocation[NumProcs, NumResources];
/* Allocation[p,r] = số lượng tài nguyên r thực sự cấp phát cho p */
int Need[NumProcs, NumResources];
/* Need[p,r] = Max[p,r] - Allocation[p,r] */
```

1. Giả sử có các mảng

```
int Work[NumProcs, NumResources] = Available;
int Finish[NumProcs] = false;
```

2. Tìm  $i$  sao cho

```
Finish[i] == false
Need[i] <= Work[i]
```

Nếu không có  $i$  như thế, đến bước 4.

3.     Work = Work + Allocation[i];  
       Finish[i] = true;  
       Đến bước 2

4. Nếu Finish[i] == true với mọi  $i$ , thì hệ thống ở trạng thái an toàn.

👉 Ví dụ : Giả sử tình trạng hiện hành của hệ thống được mô tả như sau :

| Max |    |    | Allocation |    |    | Available |    |    |
|-----|----|----|------------|----|----|-----------|----|----|
| R1  | R2 | R3 | R1         | R2 | R3 | R1        | R2 | R3 |
| P1  | 3  | 2  | 2          | 1  | 0  | 0         |    |    |
| P2  | 6  | 1  | 3          | 2  | 1  | 1         |    |    |

|    |   |   |   |   |   |   |  |
|----|---|---|---|---|---|---|--|
| P3 | 3 | 1 | 4 | 2 | 1 | 1 |  |
| P4 | 4 | 2 | 2 | 0 | 0 | 2 |  |

Nếu tiến trình P2 yêu cầu 4 cho R1, 1 cho R3. hãy cho biết yêu cầu này có thể đáp ứng mà bảo đảm không xảy ra tình trạng deadlock hay không ? Nhận thấy Available[1] =4, Available[3] =2 đủ để thỏa mãn yêu cầu của P2, ta có

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

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

| Need |    |    | Allocation |    |    | Available |    |    |
|------|----|----|------------|----|----|-----------|----|----|
| R1   | R2 | R3 | R1         | R2 | R3 | R1        | R2 | R3 |
| P1   | 0  | 0  | 0          | 0  | 0  | 0         |    |    |

|    |   |   |   |   |   |   |  |
|----|---|---|---|---|---|---|--|
| P2 | 0 | 0 | 0 | 0 | 0 | 0 |  |
| P3 | 1 | 0 | 3 | 2 | 1 | 1 |  |
| P4 | 4 | 2 | 0 | 0 | 0 | 2 |  |

| Need |    |    | Allocation |    |    | Available |    |    |
|------|----|----|------------|----|----|-----------|----|----|
| R1   | R2 | R3 | R1         | R2 | R3 | R1        | R2 | R3 |
| P1   | 0  | 0  | 0          | 0  | 0  | 0         |    |    |
| P2   | 0  | 0  | 0          | 0  | 0  | 0         |    |    |
| P3   | 0  | 0  | 0          | 0  | 0  | 0         |    |    |
| P4   | 4  | 2  | 0          | 0  | 0  | 2         |    |    |

| Need |    |    | Allocation |    |    | Available |    |    |
|------|----|----|------------|----|----|-----------|----|----|
| R1   | R2 | R3 | R1         | R2 | R3 | R1        | R2 | R3 |
| P1   | 0  | 0  | 0          | 0  | 0  | 0         |    |    |
| P2   | 0  | 0  | 0          | 0  | 0  | 0         |    |    |
| P3   | 0  | 0  | 0          | 0  | 0  | 0         |    |    |
| P4   | 0  | 0  | 0          | 0  | 0  | 0         |    |    |

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

### ✚ Giải thuật yêu cầu tài nguyên

Giả sử tiến trình  $P_i$  yêu cầu  $k$  thể hiện của tài nguyên  $r$ .

1. Nếu  $k \leq \text{Need}[i]$ , đến bước 2  
Ngược lại, xảy ra tình huống lỗi

2. Nếu  $k \leq \text{Available}[i]$ , đến bước 3

Ngược lại,  $P_i$  phải chờ

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

```
Available[i] = Available[i] - k;  
Allocation[i] = Allocation[i] + k;  
Need[i] = Need[i] - k;
```

Nếu trạng thái kết quả là an toàn, lúc này các tài nguyên trên sẽ được cấp phát thật sự cho  $P_i$

Ngược lại,  $P_i$  phải chờ

#### IV.7. Phát hiện tắc nghẽn

Cần sử dụng các cấu trúc dữ liệu sau :

```
int Available[NumResources];  
// Available[r] = số lượng các thể hiện còn tự do của tài nguyên r  
int Allocation[NumProcs, NumResources];  
  
// Allocation[p,r] = số lượng tài nguyên r thực sự cấp phát cho p  
  
int Request[NumProcs, NumResources];  
// Request[p,r] = số lượng tài nguyên r tiến trình p yêu cầu thêm
```

#### Giải thuật phát hiện tắc nghẽn

1. `int Work[NumResources] = Available;`  
`int Finish[NumProcs];`  
  
`for (i = 0; i < NumProcs; i++)`  
`Finish[i] = (Allocation[i] == 0);`
2. Tìm  $i$  sao cho  
`Finish[i] == false`  
`Request[i] <= Work`  
Nếu không có  $i$  như thế, đến bước 4.
3. `Work = Work + Allocation[i];`  
`Finish[i] = true;`  
Đến bước 2
4. Nếu `Finish[i] == true` với mọi  $i$ ,  
thì hệ thống không có tắc nghẽn  
Nếu `Finish[i] == false` với một số giá trị  $i$ ,  
thì các tiến trình mà `Finish[i] == false` sẽ ở trong  
tình trạng tắc nghẽn.

## II.8. Hiệu chỉnh tắc nghẽn

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

### Đình chỉ hoạt động của các tiến trình liên quan

Cách tiếp cận này dựa trên việc thu hồi lại các tài nguyên của những tiến trình bị kết thúc. Có thể sử dụng một trong hai phương pháp sau :

- Đình chỉ tất cả các tiến trình trong tình trạng tắc nghẽn
- Đình chỉ từng tiến trình liên quan cho đến khi không còn chu trình gây tắc nghẽn : để chọn được tiến trình thích hợp bị đình chỉ, 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 yêu cầu...

### 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. 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: 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 không xảy ra tắc nghẽn.
- Tình trạng « đói tài nguyên »: làm sao bảo đảm rằng không có một tiến trình luôn luôn bị thu hồi tài nguyên ?

## V. TÓM TẮT

- ▶ Các giải pháp đồng bộ hoá do lập trình viên xây dựng không được ưa chuộng vì phải tiêu thụ CPU trong thời gian chờ vào miền bận (« busy waiting »), và khó mở rộng. Thay vào đó, lập trình viên có thể sử dụng các cơ chế đồng bộ do hệ điều hành hay trình biên dịch trợ giúp như semaphore, monitor, trao đổi thông điệp .
- ▶ Tắc nghẽn là tình trạng xảy ra trong một tập các tiến trình nếu có hai hay nhiều hơn các tiến trình chờ đợi vô hạn một sự kiện chỉ có thể được phát sinh bởi một tiến trình cũng đang chờ khác trong tập các tiến trình này.
- ▶ Có 3 hướng tiếp cận chính trong xử lý tắc nghẽn :
  - *Phòng tránh tắc nghẽn* : tuân thủ một vài nghi thức bảo đảm hệ thống không bao giờ lâm vào trạng thái tắc nghẽn.

- *Phát hiện tắc nghẽn* : khi có tắc nghẽn xảy ra, phát hiện các tiến trình liên quan và tìm cách phục hồi.


- *Bỏ qua tắc nghẽn* : xem như hệ thống không bao giờ lâm vào trạng thái tắc nghẽn.

## **Củng cố bài học**

Các câu hỏi cần trả lời được sau bài học này :

1. Phân biệt nhóm giải pháp busy waiting và Sleep&Wakeup
2. Phân biệt cách sử dụng semaphore, monitor và message để đồng bộ hoá.
3. Mô hình giải quyết nhu cầu độc quyền truy xuất và mô hình giải quyết nhu cầu phối hợp hoạt động.

## **Bài tập**

 **Bài 1.** Xét giải pháp phần mềm do Dekker đề nghị để tổ chức truy xuất độc quyền cho hai tiến trình . Hai tiến trình  $P_0, P_1$  chia sẻ các biến sau :

```
var flag : array [0..1] of boolean; (khởi động là false)
turn : 0..1;
```

Cấu trúc một tiến trình  $P_i$  (  $i=0$  hay 1, và  $j$  là tiến trình còn lại ) như sau :

```
repeat
flag[i] := true;
while flag[j] do
    if turn = j then
begin
flag[i] := false;
while turn = j do ;
flag[i] := true;
end;
critical_section();
turn := j;
flag[i] := false;
non_critical_section();
until false;
```

Giải pháp này có phải là một giải pháp đúng thỏa mãn 4 yêu cầu không ?

 **Bài 2.** Xét giải pháp phần mềm do Eisenberg và McGuire đề nghị để tổ chức truy xuất độc quyền cho  $N$  tiến trình . Các tiến trình chia sẻ các biến sau :


```
var flag : array [0..N-1] of (idle, want-in, in-cs);
turn : 0..N-1;
```

Tất cả các phần tử của mảng *flag* được khởi động là *idle*, *turn* được khởi gán một trong những giá trị từ 0..N-1

Cấu trúc một tiến trình Pi như sau :


```
repeat
repeat
flag[i] := want-in;
j := turn;
while j<>i do
if flag[j]<> idle then j:= turn
else j:= j+1 mod n;
flag[i]:= in-cs;
j:=0;
while ( j<N) and ( j = i or flag[j] <> in-cs) do j:=j+1;
until ( j>=N) and ( turn =i or flag[turn] = idle);
turn := i;
critical_section();
j:= turn + 1 mod N;
while (flag[j]= idle) do j := j+1 mod N;
turn := j;
flag[i]:= idle;
non_critical_section();
until false;
```

Giải pháp này có phải là một giải pháp đúng thỏa mãn 4 yêu cầu không ?

 **Bài 3.** 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 ?

 **Bài 4.** Giả sử một máy tính không có chỉ thị TSL, nhưng có chỉ thị Swap có khả năng hoán đổi nội dung của hai từ nhớ chỉ bằng một thao tác không thể phân chia :

```
procedure Swap() var a,b: boolean);
var temp : boolean;
begin
temp := a;
a:= b;
b:= temp;
end;
```

Sử dụng chỉ thị này có thể tổ chức truy xuất độc quyền không ? Nếu có xây dựng cấu trúc chương trình

tương ứng.

**❓ Bài 5.** Chứng tỏ rằng nếu các primitive Down và Up trên semaphore không thực hiện một cách không thể phân chia, thì sự truy xuất độc quyền sẽ bị vi phạm.

**❓ Bài 6.** Sử dụng semaphore để cài đặt cơ chế monitor.

**❓ Bài 7.** Xét hai tiến trình sau :

```
process A
{ while (TRUE)
  na = na +1;
}
process B
{ while (TRUE)
  nb = nb +1;
}
```

a) Đồng bộ hoá xử lý của hai tiến trình trên, sử dụng hai semaphore tổng quát, sao cho tại bất kỳ thời điểm nào cũng có  $nb < na \leq nb + 10$

b) Nếu giảm điều kiện chỉ là  $na \leq nb + 10$ , 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?

**❓ Bài 8.** 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 ?

**❓ Bài 9.** 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  $A_1$  và  $B_1$  đều hoàn tất trước khi  $A_2$  hay  $B_2$  bắt đầu .

**❓ Bài 10.** Tổng quát hoá câu hỏi 8) 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 ( $2 \leq k \leq 100$ ),  $A_k$  chỉ có thể bắt



đầu khi  $B_{(k-1)}$  đã kết thúc, và  $B_k$  chỉ có thể bắt đầu khi  $A_{(k-1)}$  đã kết thúc.

**❓ Bài 11.** Sử dụng semaphore để viết lại chương trình sau theo mô hình xử lý đồng hành:

```
w := x1 * x2
v := x3 * x4
y := v * x5
z := v * x6
y := w * y
z := w * z
ans := y + z
```

**❓ Bài 12.** Xây dựng một giải pháp ( sử dụng semaphore ) để giải quyết vấn đề Readers\_Writers trong đó :

- a) Readers được ưu tiên ( khi không có ai truy xuất database, Reader được ưu tiên truy cập database ngay, Writer phải đợi tất cả các Reader truy xuất xong mới được vào database)
- b) Writers được ưu tiên ( khi không có ai truy xuất database, Writer được ưu tiên truy cập database ngay, Reader phải đợi tất cả các Write truy xuất xong mới được vào database)
- c) Công bằng cho Reader, Writer ( khi không có ai truy xuất database, Writer hoặc Reader có cơ hội ngang nhau để truy cập database)

**❓ Bài 13.** Dining Philosophers : Giả sử hành vi của một triết gia thứ  $i$  trong bữa ăn tối được mô tả như sau :

```
#define N 5
void philosopher( int i)
{ while (TRUE)
{ think(); // Suy nghĩ
take_fork(i); // lấy nĩa bên trái
take_fork((i+1)%N); // lấy nĩa bên phải
eat(); // yum-yum, spaghetti
put_fork(i); // đặt nĩa bên trái lên bàn lại
put_fork((i+1)%N); // đặt nĩa bên phải lên bàn lại
}
}
```

- a) Lưu ý là trên bàn chỉ có 5 cái nĩa, và nếu có 2 triết gia cùng muốn lấy một cái nĩa, thì chỉ một người được quyền lấy cái nĩa đó. Sử dụng semaphore để tổ chức độc quyền truy xuất đến các cái nĩa cho đoạn chương trình trên ( Gợi ý : dùng mỗi semaphore phản ánh tình trạng sử dụng của mỗi cái nĩa)
- b) Liệu giải pháp của câu a) có là một giải pháp tốt cho bài toán Dining philosopher? Nếu không, cho biết các tình huống lỗi sẽ xảy ra, và đề nghị phương pháp cải tiến.

**❓ Bài 14.** Xét một giải pháp 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];
void philosopher( int i) // i : xác định triết gia nào (0..N-1)
{
while (TRUE)
{ thínhk(); // 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
}
}
void take_forks ( int i) // i : xác định triết gia nào (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
}
}
}
void put_forks ( int i) // i : xác định triết gia nào (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
}
}
}
void test ( int i) // i : xác định triết gia nào (0..N-1)
{
if(state[i]==HUNGRY && state[LEFT]!=EATING
&& state[RIGHT]!= EATING
{
state[i] = EATING;
up(s[i]);
}
}
}

```

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?

 **Bài 15.** Xây dựng giải pháp monitor cho bài toán Dining Philosophers.

**❓ Bài 16. Barber problem** : Một cửa hiệu cắt tóc có một thợ, một ghế cắt tóc và N ghế cho khách đợi. Nếu không có khách hàng, anh 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 phải ngồi chờ nếu có ghế đợi trống, hoặc rời khỏi tiệm nếu đã có N người đợi. 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.

```
/* Semaphore to protect critical sections */
Semaphore mutex = 1;

/* Semaphore for the number of waiting customers.
 * This lets the barber go to sleep when there are no customers */
Semaphore customers = 0;

/* Number of waiting customers in the barber shop */
/* Just used to turn away customers when there are too many already. */
int waiting_customers = 0

/* Semaphore on which to wait for a haircut */
Semaphore haircut = 0;

/* Customer calls this function to try to get their hair cut
 * it returns true if the hair gets cut. */
int customer()
{
    /* protect access to shared variables with semaphore mutex */
    wait( mutex );

    /* Make sure there is an empty chair */
    if( waiting_customers >= 5 ){
        signal( mutex );
        return 0;
    }
    /* there is now a new waiting customer */
    waiting_customers += 1;
    signal( mutex );
    /* Wake the barber if the shop was empty */
    signal( customers );
    /* Wait for a haircut from the barber */
    wait( haircut );
    return 1;
}

/* Barber loops within this function */
void barber()
{
    while( 1 ){
        /* Go to sleep if there are no customers */
        wait( customers );

        // protect access to shared variables with semaphore mutex
        wait( mutex );

        /* take customer out of a chair */
```

```

waiting_customers -= 1;

signal( mutex );

/* cut hair of the current customer */
cut_hair();

/* Let the customer go. */
signal( haircut );
}
}

```

❓ **Bài 17.** Giải quyết bài toán Barber trong trường hợp tiệm có nhiều thợ .

❓ **Bài 18.** Xét trạng thái hệ thống :

| Max |    |    | Allocation |    |    | Available |    |    |
|-----|----|----|------------|----|----|-----------|----|----|
| R1  | R2 | R3 | R1         | R2 | R3 | R1        | R2 | R3 |
| P1  | 3  | 2  | 2          | 1  | 0  | 0         |    |    |
| 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 ?

❓ **Bài 19.** Xét trạng thái hệ thống :

| Max |   |   |   | Allocation |   |   |   | Available |   |   |   |
|-----|---|---|---|------------|---|---|---|-----------|---|---|---|
| A   | B | C | D | A          | B | C | D | A         | B | C | D |
| P1  | 0 | 0 | 1 | 2          | 0 | 0 | 1 | 2         |   |   |   |
| 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 thống 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?

## BÀI 6 : QUẢN LÝ BỘ NHỚ

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

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

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

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

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

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

### I. Bối cảnh

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

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

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

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

☞ Thời điểm biên dịch: nếu tại thời điểm biên dịch, có thể biết vị trí mà tiến trình sẽ thường trú trong bộ nhớ, trình biên dịch có thể phát sinh ngay mã với các địa chỉ tuyệt đối. Tuy nhiên, nếu về sau có sự thay đổi vị trí thường trú lúc đầu của chương trình, cần phải biên dịch lại chương trình.

☞ Thời điểm nạp : nếu tại thời điểm biên dịch, chưa thể biết vị trí mà tiến trình sẽ thường trú

trong bộ nhớ, trình biên dịch cần phát sinh mã tương đối (translatable). Sự liên kết địa chỉ được trì hoãn đến thời điểm chương trình được nạp vào bộ nhớ, lúc này các địa chỉ tương đối sẽ được chuyển thành địa chỉ tuyệt đối do đã biết vị trí bắt đầu lưu trữ tiến trình. Khi có sự thay đổi vị trí lưu trữ, chỉ cần nạp lại chương trình để tính toán lại các địa chỉ tuyệt đối, mà không cần biên dịch lại.

☞ Thời điểm xử lý : nếu có nhu cầu di chuyển tiến trình từ vùng nhớ này sang vùng nhớ khác trong quá trình tiến trình xử lý, thì thời điểm kết buộc địa chỉ phải trì hoãn đến tận thời điểm xử lý. Để thực hiện kết buộc địa chỉ vào thời điểm xử lý, cần sử dụng cơ chế phân cứng đặc biệt.

## II. Không gian địa chỉ và không gian vật lý

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

☞ *Địa chỉ logic* – còn gọi là *địa chỉ ảo* , là tất cả các địa chỉ do bộ xử lý tạo ra.

☞ *Địa chỉ vật lý* - là địa chỉ thực tế mà trình quản lý bộ nhớ nhìn thấy và thao tác.

☞ *Không gian địa chỉ* – là tập hợp tất cả các địa chỉ ảo phát sinh bởi một chương trình.

☞ *Không gian vật lý* – là tập hợp tất cả các địa chỉ vật lý tương ứng với các địa chỉ ảo.

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

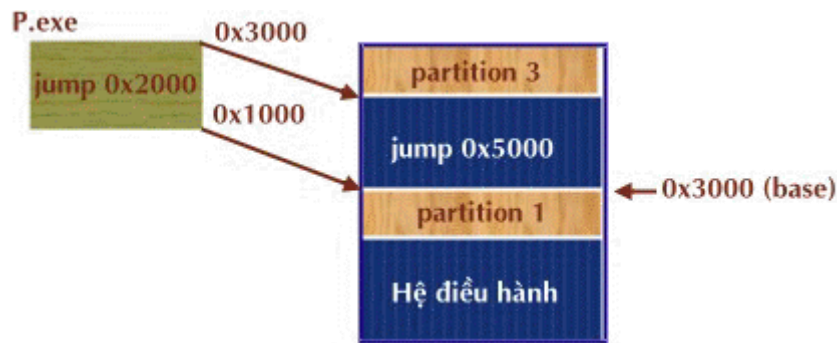
MMU (*memory-management unit*) là một cơ chế phân cứng được sử dụng để thực hiện chuyển đổi địa chỉ ảo thành địa chỉ vật lý vào thời điểm xử lý.

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

## III. Cấp phát liên tục

### III.1. Mô hình Linker Loader

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

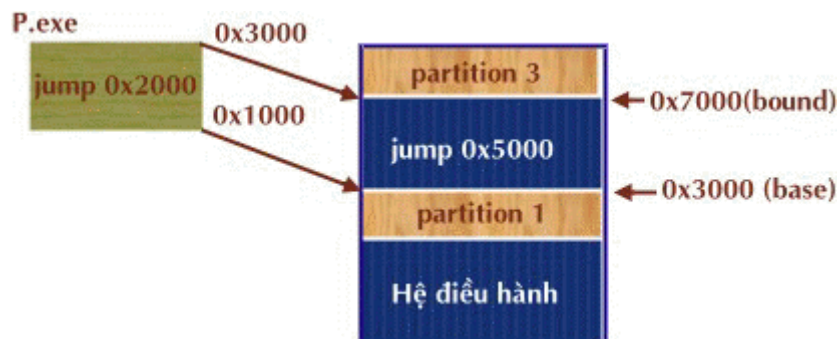


#### ✚ Thảo luận:

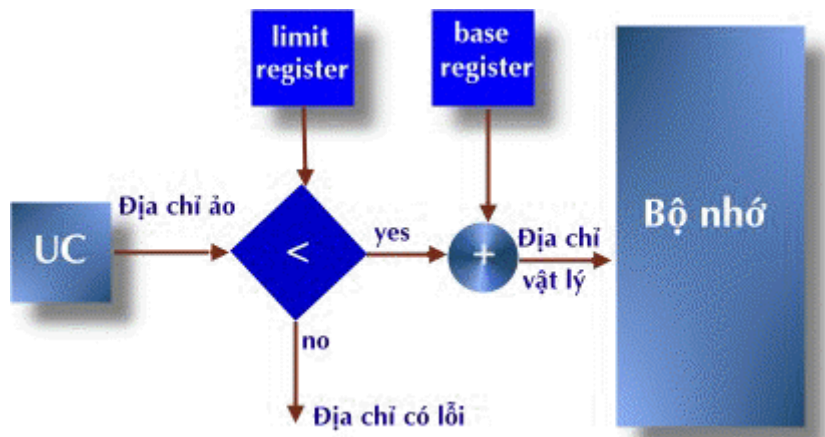
- Thời điểm kết thúc địa chỉ là thời điểm nạp, do vậy sau khi nạp không thể dời chuyển tiến trình trong bộ nhớ.
- Không có khả năng kiểm soát địa chỉ các tiến trình truy cập, do vậy không có sự bảo vệ.

### III.2. Mô hình Base & Bound

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



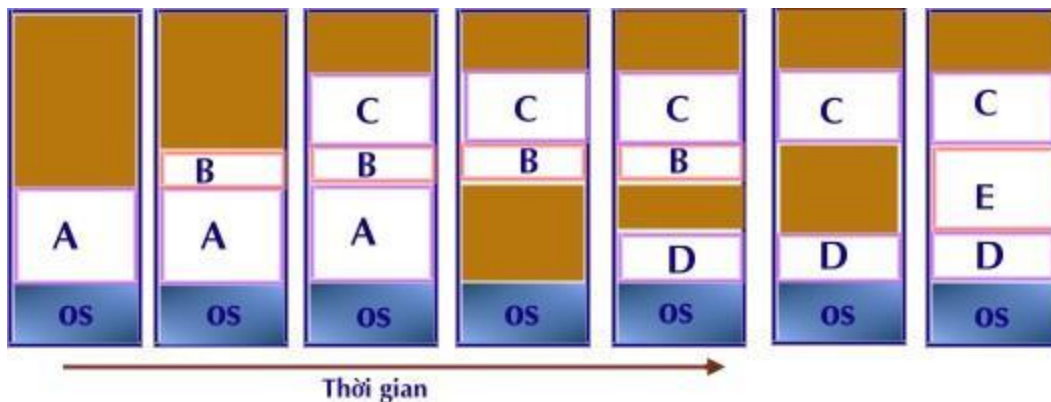




### Hai thanh ghi hỗ trợ chuyển đổi địa chỉ

#### ✚ Thảo luận:

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



### Phân mảnh ngoại vi

- Vấn đề nảy sinh khi kích thước của tiến trình tăng trưởng trong quá trình xử lý mà không còn vùng nhớ trống gần kề để mở rộng vùng nhớ cho tiến trình. Có hai cách giải quyết:
- ▶ Dời chỗ tiến trình : di chuyển tiến trình đến một vùng nhớ khác đủ lớn để thỏa mãn nhu cầu tăng trưởng của tiến trình.

▶ Cấp phát dư vùng nhớ cho tiến trình : cấp phát dự phòng cho tiến trình một vùng nhớ lớn hơn yêu cầu ban đầu của tiến trình.

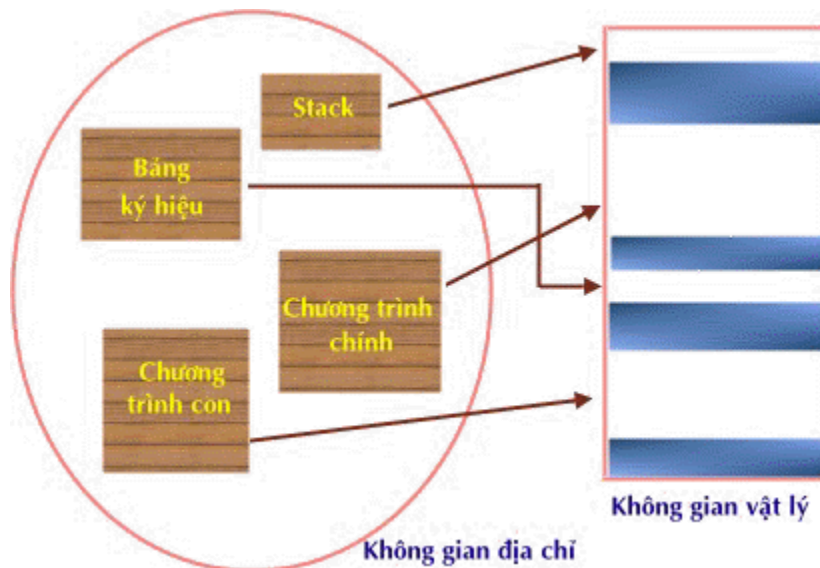
⚠ Một tiến trình cần được nạp vào bộ nhớ để xử lý. Trong các phương thức tổ chức trên đây, một tiến trình luôn được lưu trữ trong bộ nhớ suốt quá trình xử lý của nó. Tuy nhiên, trong trường hợp tiến trình bị khóa, hoặc tiến trình sử dụng hết thời gian CPU dành cho nó, nó có thể được chuyển tạm thời ra bộ nhớ phụ và sau này được nạp trở lại vào bộ nhớ chính để tiếp tục xử lý.

⚠ Các cách tổ chức bộ nhớ trên đây đều phải chịu đựng tình trạng bộ nhớ bị phân mảnh vì chúng đều tiếp cận theo kiểu cấp phát một vùng nhớ liên tục cho tiến trình. Như đã thảo luận, có thể sử dụng kỹ thuật dồn bộ nhớ để loại bỏ sự phân mảnh ngoại vi, nhưng chi phí thực hiện rất cao. Một giải pháp khác hữu hiệu hơn là cho phép không gian địa chỉ vật lý của tiến trình không liên tục, nghĩa là có thể cấp phát cho tiến trình những vùng nhớ tự do bất kỳ, không cần liên tục.

## IV. Cấp phát không liên tục

### IV.1. Phân đoạn (Segmentation)

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



Hình 4.17 Mô hình phân đoạn bộ nhớ

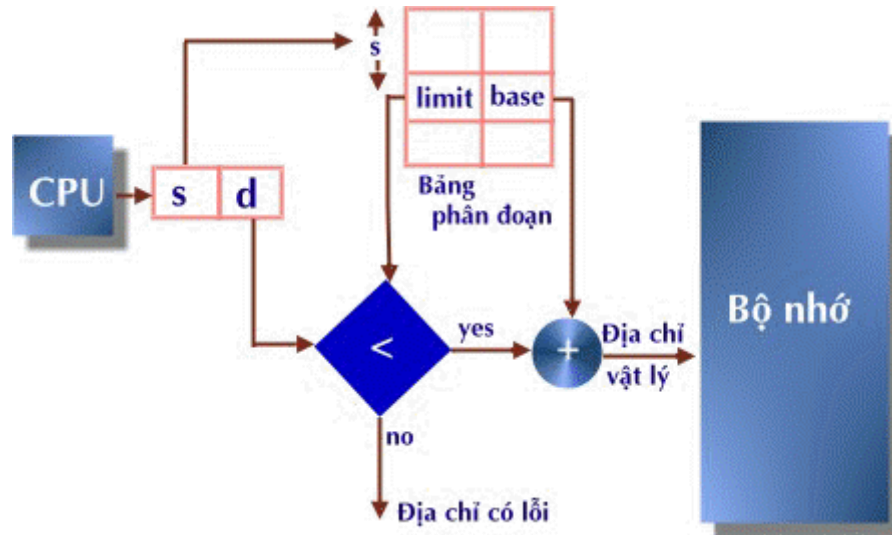
#### ✚ Cơ chế MMU trong kỹ thuật phân đoạn:

Cần phải xây dựng một ánh xạ để chuyển đổi các địa chỉ 2 chiều được người dùng định nghĩa thành địa chỉ vật lý một chiều. Sự chuyển đổi này được thực hiện qua một *bảng phân đoạn*. Mỗi thành phần trong bảng phân đoạn bao gồm một *thanh ghi nền* và một *thanh ghi giới hạn*. Thanh ghi nền lưu trữ địa chỉ vật lý nơi bắt đầu phân đoạn trong bộ nhớ, trong khi thanh ghi giới hạn đặc tả chiều dài của phân đoạn.

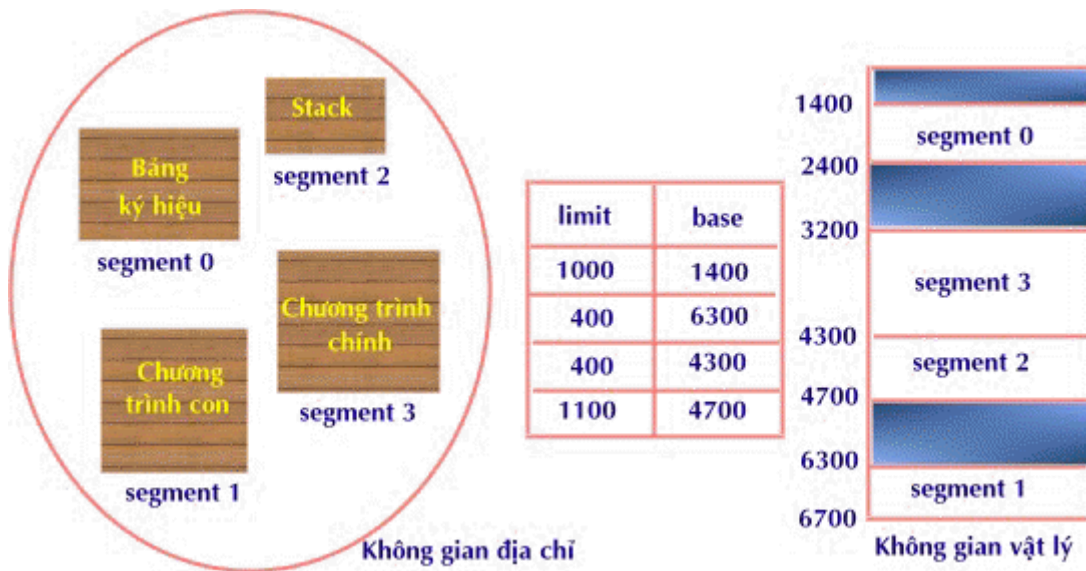
#### ✚ Chuyển đổi địa chỉ:

Mỗi địa chỉ ảo là một bộ  $\langle s, d \rangle$  :

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



**Hình 4.18** Cơ chế phần cứng hỗ trợ kỹ thuật phân đoạn

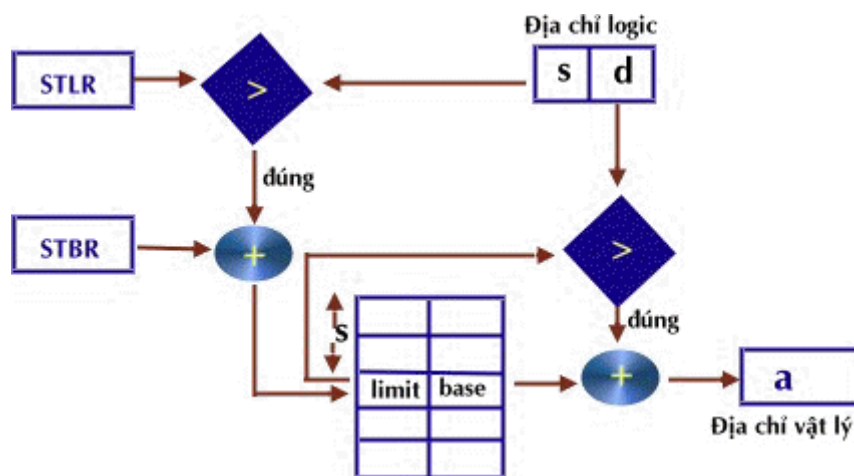


**Hình 4.19** Hệ thống phân đoạn

#### 🔧 Cài đặt bảng phân đoạn:

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

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

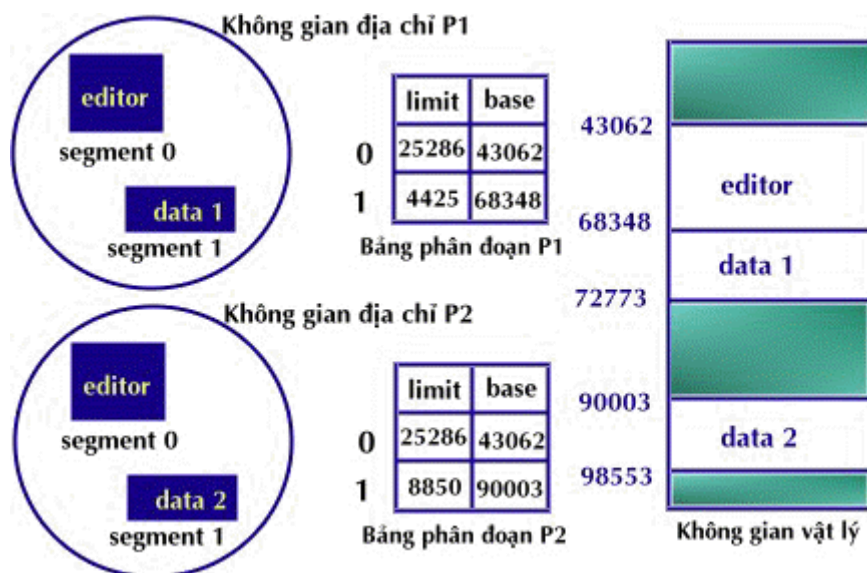


**Hình 4.20** Sử dụng STBR, STLR và bảng phân đoạn

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

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

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

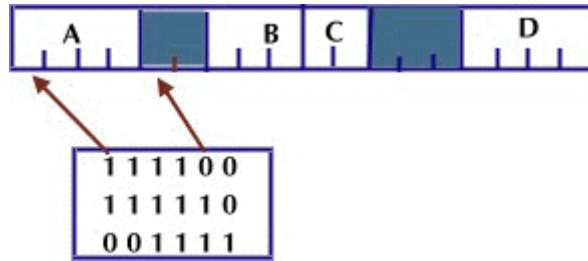


**Hình 4.21** Chia sẻ code trong hệ phân đoạn

**Thảo luận:**

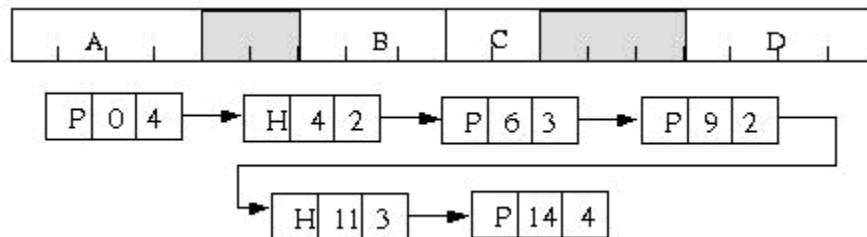
• Phải giải quyết vấn đề cấp phát động: làm thế nào để thỏa mãn một yêu cầu vùng nhớ kích thước  $N$  ? Cần phải chọn vùng nhớ nào trong danh sách vùng nhớ tự do để cấp phát ? Như vậy cần phải ghi nhớ hiện trạng bộ nhớ để có thể cấp phát đúng. Có hai phương pháp quản lý chủ yếu :

► Quản lý bằng một bảng các bit : bộ nhớ được chia thành các đơn vị cấp phát, mỗi đơn vị được phản ánh bằng một bit trong bảng các bit, một bit nhận giá trị 0 nếu đơn vị bộ nhớ tương ứng đang tự do, và nhận giá trị 1 nếu đơn vị tương ứng đã được cấp phát cho một tiến trình. Khi cần nạp một tiến trình có kích thước  $k$  đơn vị, cần phải tìm trong bảng các bit một dãy con  $k$  bit nhận giá trị 0. Đây là một giải pháp đơn giản, nhưng thực hiện chậm nên ít được sử dụng.



**Hình 4.5** Quản lý bộ nhớ bằng bảng các bit

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



**Hình 4.6** Quản lý bộ nhớ bằng danh sách

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

**First-fit**: cấp phát phân đoạn tự do đầu tiên đủ lớn.

**Best-fit**: cấp phát phân đoạn tự do nhỏ nhất nhưng đủ lớn để thỏa mãn nhu cầu.

**Worst-fit** : cấp phát phân đoạn tự do lớn nhất.

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

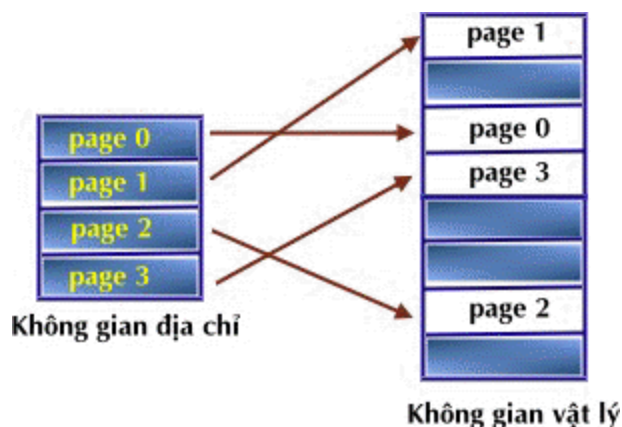
## IV.2. Phân trang ( Paging)

**Ý tưởng:**

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



gọi là *trang* (*page*). Khi cần nạp một tiến trình để xử lý, các trang của tiến trình sẽ được nạp vào những khung trang còn trống. Một tiến trình kích thước  $N$  trang sẽ yêu cầu  $N$  khung trang tự do.



**Hình 4.8** Mô hình bộ nhớ phân trang

#### ✚ Cơ chế MMU trong kỹ thuật phân trang:

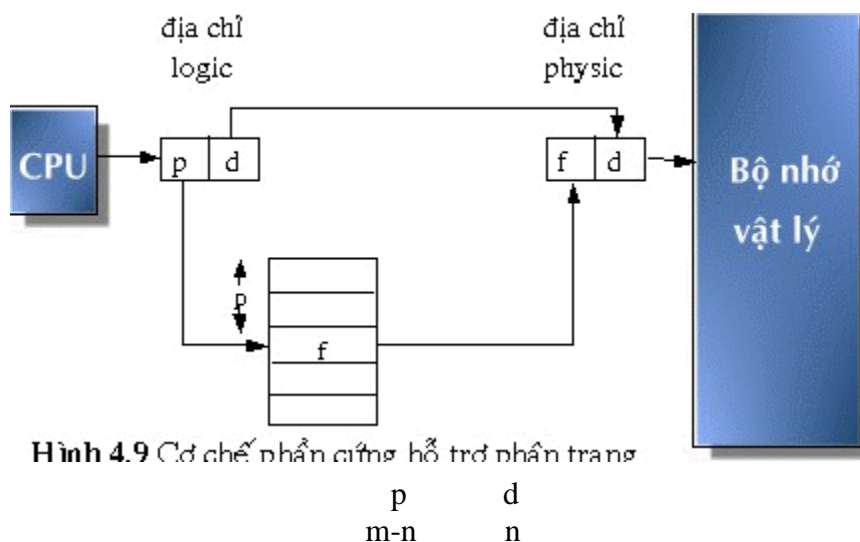
Cơ chế phần cứng hỗ trợ thực hiện chuyển đổi địa chỉ trong cơ chế phân trang là bảng trang (*pages table*). Mỗi phần tử trong bảng trang cho biết các địa chỉ bắt đầu của vị trí lưu trữ trang tương ứng trong bộ nhớ vật lý (số hiệu khung trang trong bộ nhớ vật lý đang chứa trang).

#### ✚ Chuyển đổi địa chỉ:

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

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

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

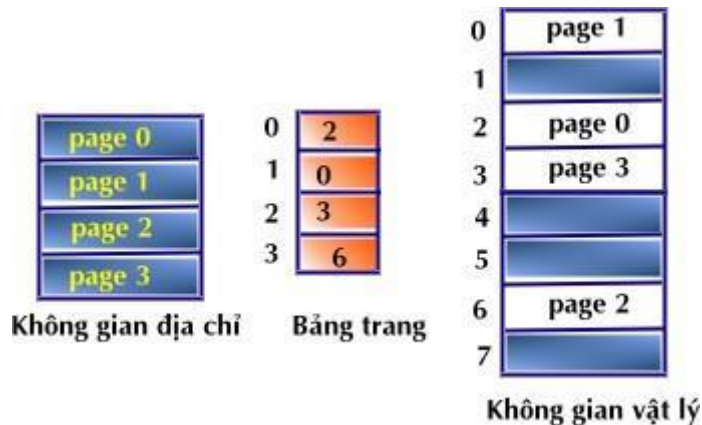


**Hình 4.9** Cơ chế phần cứng hỗ trợ phân trang

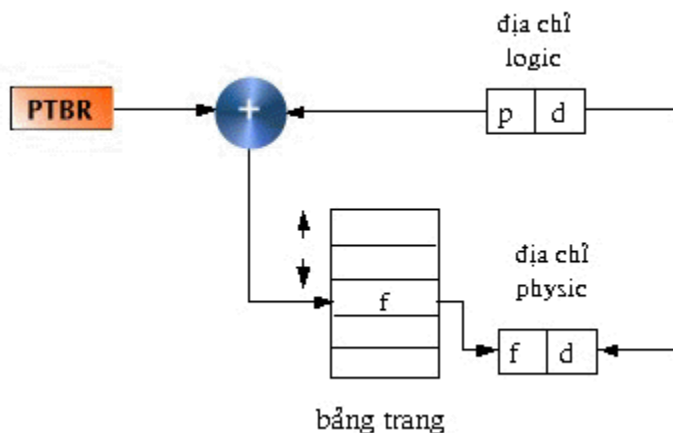
### 🔗 Cài đặt bảng trang:

Trong trường hợp đơn giản nhất, bảng trang một tập các thanh ghi được sử dụng để cài đặt bảng trang. Tuy nhiên việc sử dụng thanh ghi chỉ phù hợp với các bảng trang có kích thước nhỏ, nếu bảng trang có kích thước lớn, nó phải được lưu trữ trong bộ nhớ chính, và sử dụng một thanh ghi để lưu địa chỉ bắt đầu lưu trữ bảng trang (PTBR).

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



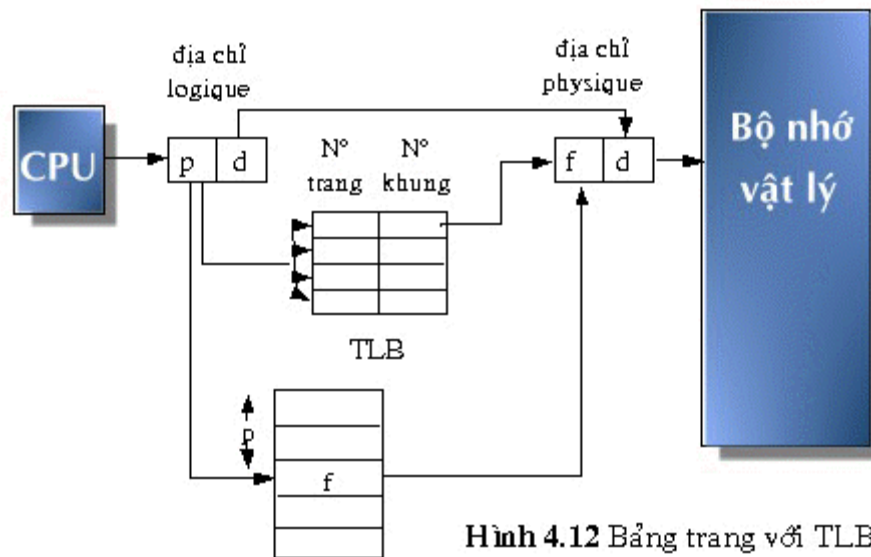
Hình 4.10 Mô hình bộ nhớ phân trang



Hình 4.11 Sử dụng thanh ghi nền trở đến bảng trang

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

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

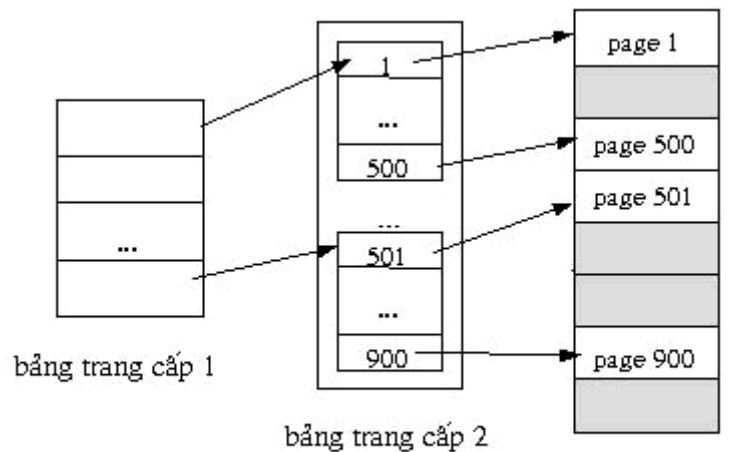


Hình 4.12 Bảng trang với TLBs

### ✚ Tổ chức bảng trang:

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

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



Hình 4.13 Bảng trang nhị cấp

Bộ nhớ vật lý

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

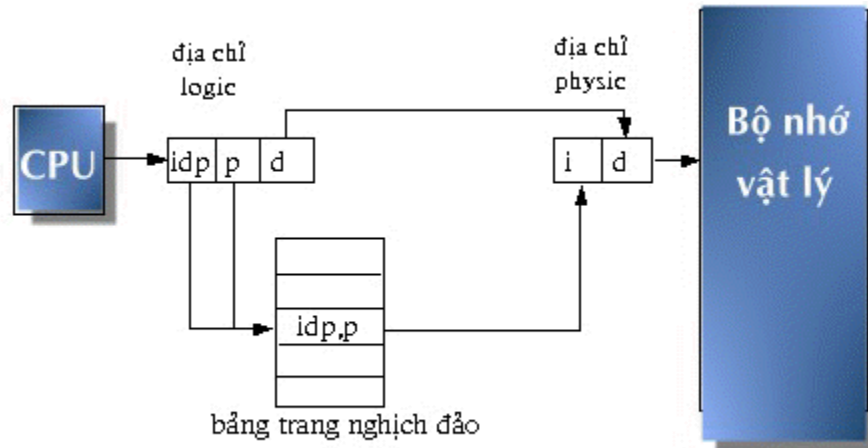
Trong đó :

**idp** là định danh của tiến trình



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

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



**Hình 4.14** Bảng trang nghịch đảo

#### ✚ Bảo vệ:

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

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

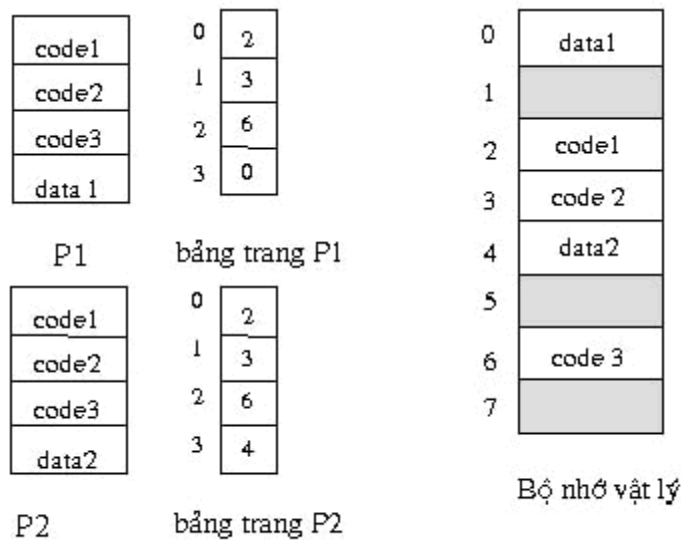
- *Hợp lệ*: trang tương ứng thuộc về không gian địa chỉ của tiến trình.
- *Bất hợp lệ*: trang tương ứng không nằm trong không gian địa chỉ của tiến trình, điều này có nghĩa tiến trình đã truy xuất đến một địa chỉ không được phép.

|                        |                      |
|------------------------|----------------------|
| số hiệu<br>khung trang | bit<br>valid-invalid |
|------------------------|----------------------|

**Hình 4.15** Cấu trúc một phần tử trong bảng trang

#### ✚ Chia sẻ bộ nhớ trong cơ chế phân trang:

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



**Hình 4.16** Chia sẻ các trang trong hệ phân trang

#### ✚ Thảo luận:

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

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

⚠ Kỹ thuật phân đoạn 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 trong bộ nhớ vật lý. Điều này làm rắc rối vấn đề hơn rất nhiều so với việc cấp phát các trang có kích thước tĩnh. Một giải pháp dung hoà là kết hợp cả hai kỹ thuật phân trang và phân đoạn : chúng ta tiến hành *phân trang các phân đoạn*.

### IV.3. Phân đoạn kết hợp phân trang (Paged segmentation)

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

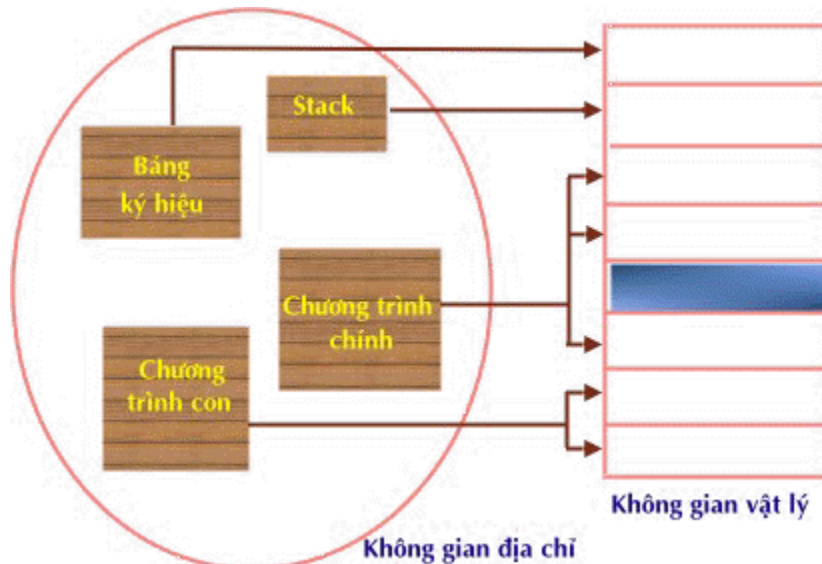
✚ **Cơ chế MMU trong kỹ thuật phân đoạn kết hợp phân trang:**

Để hỗ trợ kỹ thuật phân đoạn, cần có một *bảng phân đoạn*, nhưng giờ đây mỗi phân đoạn cần có một *bảng trang* phân biệt.

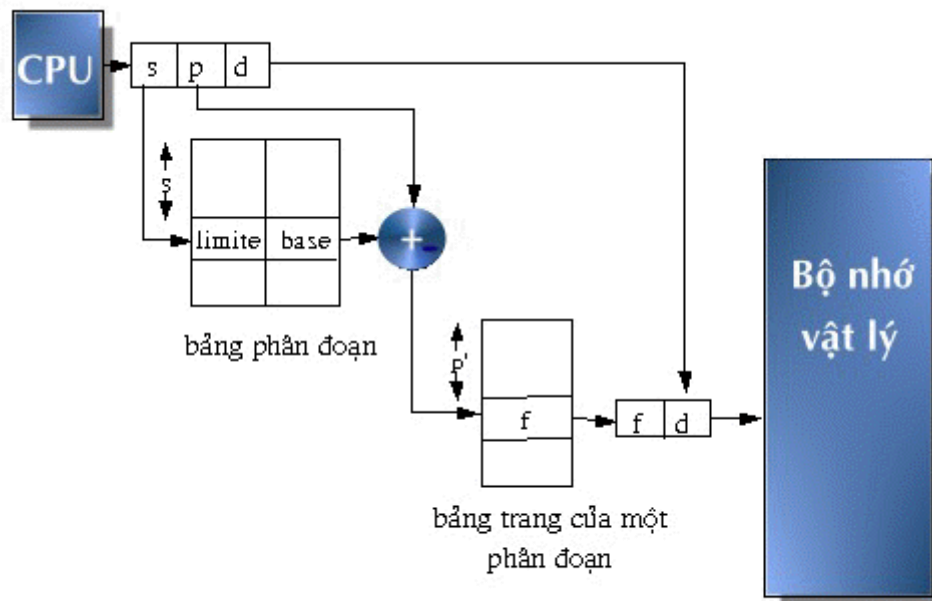
✚ **Chuyển đổi địa chỉ:**

Mỗi địa chỉ logic là một bộ ba:  $\langle s, p, d \rangle$

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



Hình 4.22 Mô hình phân đoạn kết hợp phân trang



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

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

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

## V. Tóm tắt

► Có nhiều cách tiếp cận khác nhau để tổ chức quản lý bộ nhớ, nhưng tựu chung mong đạt đến các mục tiêu sau :

- Có thể đáp ứng được đầy đủ các nhu cầu bộ nhớ của chương trình với một bộ nhớ vật lý giới hạn
- Quá trình chuyển đổi địa chỉ, tổ chức cấp phát bộ nhớ là trong suốt với người dùng, và có khả năng tái định vị.
- Tận dụng hiệu quả bộ nhớ ( ít có vùng nhớ không sử dụng được)
- Bộ nhớ được bảo vệ tốt
- Có khả năng chia sẻ bộ nhớ giữa các tiến trình

► Một số cách tiếp cận tổ chức bộ nhớ chính

- *Cấp phát liên tục* : có thể cấp phát các vùng nhớ liên tục cho các tiến trình trong những phân vùng có kích thước cố định hay biến động. Điểm yếu của cách tiếp cận này là kích thước các chương trình có thể được xử lý bị giới hạn bởi các kích thước của khối nhớ liên tục có thể sử dụng. Các hiện tượng phân mảnh ngoại vi, nội vi đều có thể xuất hiện
- *Cấp phát không liên tục* : có thể cấp phát các vùng nhớ không liên tục cho một tiến trình. Hai kỹ thuật




thường được áp dụng là phân trang và phân đoạn. Kỹ thuật phân trang cho phép loại bỏ hiện tượng phân mảnh ngoại vi, kỹ thuật phân đoạn loại bỏ hiện tượng phân mảnh nội vi, nhưng phải giải quyết vấn đề cấp phát động.

## **Củng cố bài học**






Các câu hỏi cần trả lời được sau bài học này :

1. Nhiệm vụ quản lý bộ nhớ bao gồm các công việc nào ? Giai đoạn nào do hệ điều hành thực hiện , giai đoạn nào cần sự trợ giúp của phần cứng?
2. Các khái niệm : phân mảnh nội vi, phân mảnh ngoại vi, bài toán cấp phát động, địa chỉ logic, địa chỉ physic
3. Phân tích ưu khuyết của các mô hình tổ chức bộ nhớ.

## **Bài Tập**

-  **Bài 1.** Giải thích sự khác biệt giữa địa chỉ logic và địa chỉ physic?
-  **Bài 2.** Giải thích sự khác biệt giữa hiện tượng phân mảnh nội vi và ngoại vi?
-  **Bài 3.** Giả sử bộ nhớ chính được phân thành các phân vùng có kích thước là 600K, 500K, 200K, 300K ( theo thứ tự ), cho biết các tiến trình có kích thước 212K, 417K, 112K và 426K ( theo thứ tự ) sẽ được cấp phát bộ nhớ như thế nào, nếu sử dụng :
- a) Thuật toán First fit
  - b) Thuật toán Best fit
  - c) Thuật toán Worst fit

Thuật toán nào cho phép sử dụng bộ nhớ hiệu quả nhất trong trường hợp trên ?

-  **Bài 4.** Xét một hệ thống trong đó một chương trình khi được nạp vào bộ nhớ sẽ phân biệt hoàn toàn phân đoạn code và phân đoạn data. Giả sử CPU sẽ xác định được khi nào cần truy xuất lệnh hay dữ liệu , và phải truy xuất ở đâu. Khi đó mỗi chương trình sẽ được cung cấp 2 bộ thanh ghi base-limit : một cho phân đoạn code, và một cho phân đoạn data. Bộ thanh ghi base-limit của phân đoạn code tự động được đặt thuộc tính readonly. Thảo luận các ưu và khuyết điểm của hệ thống này.
-  **Bài 5.** Tại sao kích thước trang luôn là lũy thừa của 2 ?
-  **Bài 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 ?
-  **Bài 7.** Tại sao trong hệ thống sử dụng kỹ thuật phân trang, một tiến trình không thể truy xuất đến vùng nhớ không được cấp cho nó ? Làm cách nào hệ điều hành có thể cho phép sự truy xuất này xảy ra ? Hệ điều hành có nên cho phép điều đó không ? Tại sao ?
-  **Bài 8.** 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 200nanoseconds, thì mất bao nhiêu thời gian cho một thao tác truy xuất bộ nhớ trong hệ thống này ?
  - b) Nếu sử dụng TLBs với hit-ratio ( tỉ lệ tìm thấy) 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)

**❓ Bài 9.** Nếu cho phép hai phần tử trong bảng trang cùng lưu trữ một số hiệu khung trang trong bộ nhớ thì sẽ có hiệu quả gì ? Giải thích làm cách nào hiệu quả này có thể được sử dụng để giảm thời gian cần khi sao chép một khối lượng lớn vùng nhớ từ vị trí này sang vị trí khác. Khi đó nếu sửa nội dung một trang thì sẽ tác động đến trang còn lại thế nào?

**❓ Bài 10.** Vì sao đôi lúc người ta kết hợp hai kỹ thuật phân trang và phân đoạn ?

**❓ Bài 11.** Mô tả cơ chế cho phép một phân đoạn có thể thuộc về không gian địa chỉ của hai tiến trình.

**❓ Bài 12.** Giải thích vì sao chia sẻ một module trong kỹ thuật phân đoạn lại dễ hơn trong kỹ thuật phân trang?

**❓ Bài 13.** Xét bảng phân đoạn sau đây :

| Segment | Base | Length |
|---------|------|--------|
| 0       | 219  | 600    |
| 1       | 2300 | 14     |
| 2       | 90   | 100    |
| 3       | 1327 | 580    |
| 4       | 1952 | 96     |

Cho biết địa chỉ vật lý tương ứng với các địa chỉ logique sau đây :

- a. 0,430
- b. 1,10
- c. 2,500
- d. 3,400
- e. 4,112

## BÀI 7: BỘ NHỚ ẢO

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

### I. Dẫn nhập

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

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

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

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

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

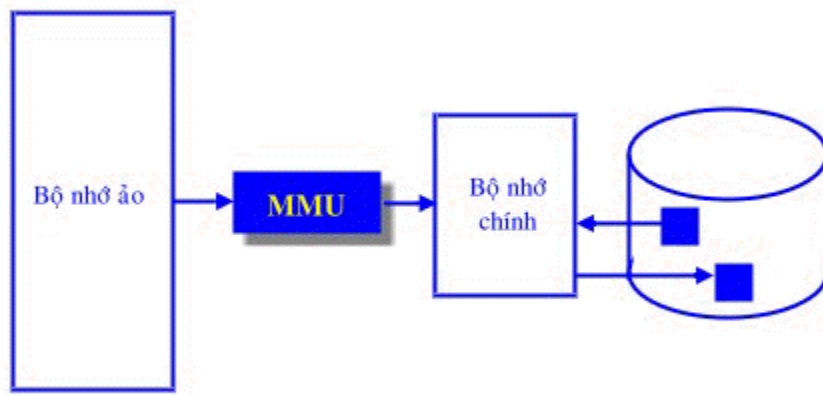
#### I.1. Định nghĩa

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

#### Thảo luận:

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





**Hình 2.24** Bộ nhớ ảo

## **I.2. Cài đặt bộ nhớ ảo**

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

### **Phân trang theo yêu cầu ( demand paging)**

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

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

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

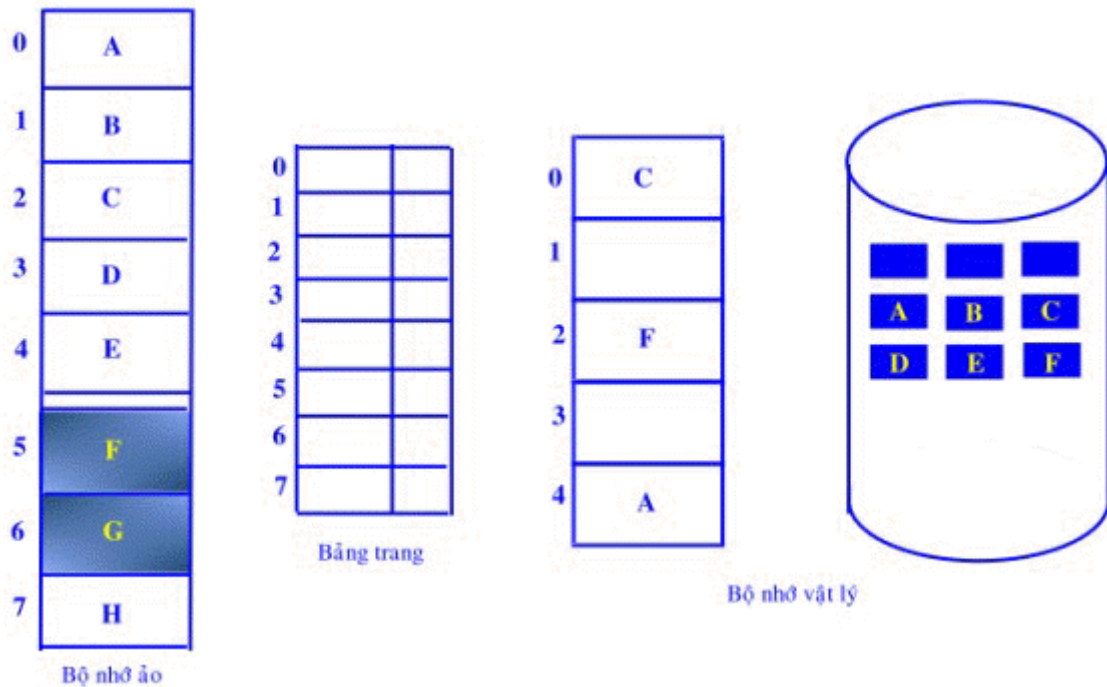
Một phần tử trong bảng trang mô tả cho một trang không nằm trong bộ nhớ chính, sẽ được đánh dấu *invalid* và chứa địa chỉ của trang trên bộ nhớ phụ.

### **Cơ chế phần cứng :**

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

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





**Hình 2.24** Bảng trang với một số trang trên bộ nhớ phụ

### ❖ Lỗi trang

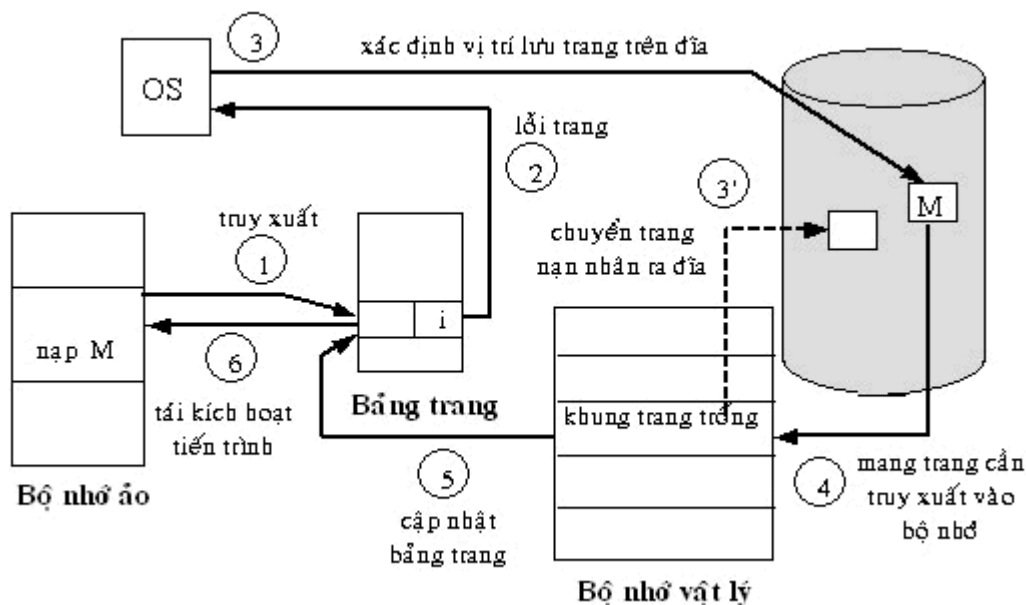
Truy xuất đến một trang được đánh dấu bất hợp lệ sẽ làm phát sinh một *lỗi trang* (*page fault*). Khi dò tìm trong bảng trang để lấy các thông tin cần thiết cho việc chuyển đổi địa chỉ, nếu nhận thấy trang đang được yêu cầu truy xuất là bất hợp lệ, cơ chế phần cứng sẽ phát sinh một ngắt để báo cho hệ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

- Kiểm tra truy xuất đến bộ nhớ là hợp lệ hay bất hợp lệ
- Nếu truy xuất bất hợp lệ : kết thúc tiến trình
- Ngược lại : đến bước 3
- Tìm vị trí chứa trang muốn truy xuất trên đĩa.
- Tìm một khung trang trống trong bộ nhớ chính :

▶ Nếu tìm thấy : đến bước 5

▶ Nếu không còn khung trang trống, chọn một khung trang « nạn nhân » và chuyển trang « nạn nhân » ra bộ nhớ phụ (lưu nội dung của trang đang chiếm giữ khung trang này lên đĩa), cập nhật bảng trang tương ứng rồi đến bước 5

- Chuyển trang muốn truy xuất từ bộ nhớ phụ vào bộ nhớ chính : nạp trang cần truy xuất vào khung trang trống đã chọn (hay vừa mới làm trống) ; cập nhật nội dung bảng trang, bảng khung trang tương ứng.
- Tái kích hoạt tiến trình người sử dụng.



**Hình 2.26** Các giai đoạn xử lý lỗi trang

## II. Thay thế trang

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

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

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

**Hình 4.27** Cấu trúc một phần tử trong bảng trang

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

### II.1. Sự thi hành phân trang theo yêu cầu

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

Giả sử  $p$  là xác suất xảy ra một lỗi trang ( $0 \leq p \leq 1$ ):

$p = 0$  : không có lỗi trang nào  
 $p = 1$  : mỗi truy xuất sẽ phát sinh một lỗi trang

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

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

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

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

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

## II.2. Các thuật toán thay thế trang

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

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

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

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

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

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

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

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

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

### \*II.2.1. Thuật toán FIFO

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

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

|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 7 | 0 | 1 | 2 | 0 | 3 | 0 | 4 | 2 | 3 | 0 | 3 | 2 | 1 | 2 | 0 | 1 | 7 | 0 | 1 |
| 7 | 7 | 7 | 2 | 2 | 2 | 2 | 4 | 4 | 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 7 | 7 | 7 |
|   | 0 | 0 | 0 | 0 | 3 | 3 | 3 | 2 | 2 | 2 | 2 | 2 | 1 | 1 | 1 | 1 | 1 | 0 | 0 |
|   |   | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 3 | 3 | 3 | 3 | 3 | 2 | 2 | 2 | 2 | 2 | 1 |

|   |   |   |   |  |   |   |   |   |   |   |  |  |   |   |  |  |   |   |   |
|---|---|---|---|--|---|---|---|---|---|---|--|--|---|---|--|--|---|---|---|
| * | * | * | * |  | * | * | * | * | * | * |  |  | * | * |  |  | * | * | * |
|---|---|---|---|--|---|---|---|---|---|---|--|--|---|---|--|--|---|---|---|

Ghi chú : \* : có lỗi trang

• Thảo luận:

▶ Để áp dụng thuật toán FIFO, thực tế không nhất thiết phải ghi nhận thời điểm mỗi trang được nạp vào bộ nhớ, mà chỉ cần tổ chức quản lý các trang trong bộ nhớ trong một danh sách FIFO, khi đó trang đầu danh sách sẽ được chọn để thay thế.

▶ Thuật toán thay thế trang FIFO dễ hiểu, dễ cài đặt. Tuy nhiên khi thực hiện không phải lúc nào cũng có kết quả tốt : trang được chọn để thay thế có thể là trang chứa nhiều dữ liệu cần thiết, thường xuyên được sử dụng nên được nạp sớm, do vậy khi bị chuyển ra bộ nhớ phụ sẽ nhanh chóng gây ra lỗi trang.

▶ Số lượng lỗi trang xảy ra sẽ tăng lên khi số lượng khung trang sử dụng tăng. Hiện tượng này gọi là *ngịch lý Belady*.

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

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

|   |   |   |   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 1 | 2 | 5 | 1 | 2 | 3 | 4 | 5 |
| 1 | 1 | 1 | 4 | 4 | 4 | 5 | 5 | 5 | 5 | 5 | 5 |
|   | 2 | 2 | 2 | 1 | 1 | 1 | 1 | 1 | 3 | 3 | 3 |
|   |   | 3 | 3 | 3 | 2 | 2 | 2 | 2 | 2 | 4 | 4 |
| * | * | * | * | * | * | * |   |   | * | * |   |

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

|   |   |   |   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 1 | 2 | 5 | 1 | 2 | 3 | 4 | 5 |
| 1 | 1 | 1 | 1 | 1 | 1 | 5 | 5 | 5 | 5 | 4 | 4 |
|   | 2 | 2 | 2 | 2 | 2 | 2 | 1 | 1 | 1 | 1 | 5 |
|   |   | 3 | 3 | 3 | 3 | 3 | 3 | 2 | 2 | 2 | 2 |
|   |   |   | 4 | 4 | 4 | 4 | 4 | 4 | 3 | 3 | 3 |
| * | * | * | * |   |   | * | * | * | * | * | * |

**\*II.2.2. Thuật toán tối ưu**

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

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

|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 7 | 0 | 1 | 2 | 0 | 3 | 0 | 4 | 2 | 3 | 0 | 3 | 2 | 1 | 2 | 0 | 1 | 7 | 0 | 1 |
| 7 | 7 | 7 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 7 | 7 | 7 |
|   | 0 | 0 | 0 | 0 | 0 | 0 | 4 | 4 | 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|   |   | 1 | 1 | 1 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| * | * | * | * |   | * |   | * |   |   | * |   |   | * |   |   |   | * |   |   |

- Thảo luận:

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

### ***\*II.2.3. Thuật toán « Lâu nhất chưa sử dụng » ( Least-recently-used LRU)***

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

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

|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 7 | 0 | 1 | 2 | 0 | 3 | 0 | 4 | 2 | 3 | 0 | 3 | 2 | 1 | 2 | 0 | 1 | 7 | 0 | 1 |
| 7 | 7 | 7 | 2 | 2 | 2 | 2 | 4 | 4 | 4 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
|   | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 3 | 3 | 3 | 3 | 3 | 3 | 0 | 0 | 0 | 0 | 0 |
|   |   | 1 | 1 | 1 | 3 | 3 | 3 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 7 | 7 | 7 |
| * | * | * | * |   | * |   | * | * | * | * |   |   | * |   | * |   | * |   |   |

- Thảo luận:

► Thuật toán FIFO sử dụng thời điểm nạp để chọn trang thay thế, thuật toán tối ưu lại dùng thời điểm trang sẽ được sử dụng, vì thời điểm này không thể xác định trước nên thuật toán LRU phải dùng thời điểm cuối cùng trang được truy xuất – dùng quá khứ gần để dự đoán tương lai.

► Thuật toán này đòi hỏi phải được cơ chế phần cứng hỗ trợ để xác định một thứ tự cho các trang theo thời điểm truy xuất cuối cùng. Có thể cài đặt theo một trong hai cách :

#### **Sử dụng bộ đếm:**

- thêm vào cấu trúc của mỗi phần tử trong bảng trang một trường ghi nhận thời điểm truy xuất mới nhất, và thêm vào cấu trúc của CPU một bộ đếm.
- mỗi lần có sự truy xuất bộ nhớ, giá trị của counter tăng lên 1.

- Mỗi lần thực hiện truy xuất đến một trang, giá trị của counter được ghi nhận vào trường thời điểm truy xuất mới nhất của phần tử tương ứng với trang trong bảng trang.
- thay thế trang có giá trị trường thời điểm truy xuất mới nhất là nhỏ nhất.

#### **Sử dụng stack:**

- tổ chức một stack lưu trữ các số hiệu trang
- mỗi khi thực hiện một truy xuất đến một trang, số hiệu của trang sẽ được xóa khỏi vị trí hiện hành trong stack và đưa lên đầu stack.
- trang ở đỉnh stack là trang được truy xuất gần nhất, và trang ở đáy stack là trang lâu nhất chưa được sử dụng.

#### **\*II.2.4. Các thuật toán xấp xỉ LRU**

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

- một bit reference, được khởi gán là 0, được gán với một phần tử trong bảng trang.
- bit reference của một trang được phần cứng đặt giá trị 1 mỗi lần trang tương ứng được truy cập, và được phần cứng gán trở về 0 sau từng chu kỳ qui định trước.
- Sau từng chu kỳ qui định trước, kiểm tra giá trị của các bit reference, có thể xác định được trang nào đã được truy xuất đến và trang nào không, sau khi đã kiểm tra xong, các bit reference được phần cứng gán trở về 0.
- với bit reference, có thể biết được trang nào đã được truy xuất, nhưng không biết được thứ tự truy xuất. Thông tin không đầy đủ này dẫn đến nhiều thuật toán xấp xỉ LRU khác nhau.

|               |                   |           |               |
|---------------|-------------------|-----------|---------------|
| số hiệu trang | bit valid-invalid | dirty bit | bit reference |
|---------------|-------------------|-----------|---------------|

**Hình 4.28** Cấu trúc một phần tử trong bảng trang

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

- **Tiếp cận:** Có thể thu thập thêm nhiều thông tin về thứ tự truy xuất hơn bằng cách lưu trữ các bit references sau từng khoảng thời gian đều đặn:
- với mỗi trang, sử dụng thêm 8 bit lịch sử (history) trong bảng trang
- sau từng khoảng thời gian nhất định (thường là 100 milliseconds), một ngắt đồng hồ được phát sinh, và quyền điều khiển được chuyển cho hệ điều hành. Hệ điều hành đặt bit reference của mỗi trang vào bit cao nhất trong 8 bit phụ trợ của trang đó bằng cách đẩy các bit khác sang phải 1 vị trí, bỏ luôn bit thấp nhất.
- như vậy 8 bit thêm vào này sẽ lưu trữ tình hình truy xuất đến trang trong 8 chu kỳ cuối cùng.
- nếu giá trị của 8 bit là 00000000, thì trang tương ứng đã không được dùng đến suốt 8 chu kỳ cuối cùng, ngược lại nếu nó được dùng đến ít nhất 1 lần trong mỗi chu kỳ, thì 8 bit phụ trợ sẽ là 11111111. Một trang mà 8 bit phụ trợ có giá trị 11000100 sẽ được truy xuất gần thời điểm hiện tại hơn trang có 8 bit phụ trợ là 01110111.
- nếu xét 8 bit phụ trợ này như một số nguyên không dấu, thì trang LRU là trang có số phụ trợ nhỏ nhất.
- **Ví dụ :**

|              |   |   |   |   |   |   |   |   |   |   |
|--------------|---|---|---|---|---|---|---|---|---|---|
|              |   |   |   |   |   |   |   |   |   |   |
|              | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 |
| HR =11000100 |   |   |   |   |   |   |   |   |   |   |
| HR =11100010 |   |   |   |   |   |   |   |   |   |   |
| HR =01110001 |   |   |   |   |   |   |   |   |   |   |

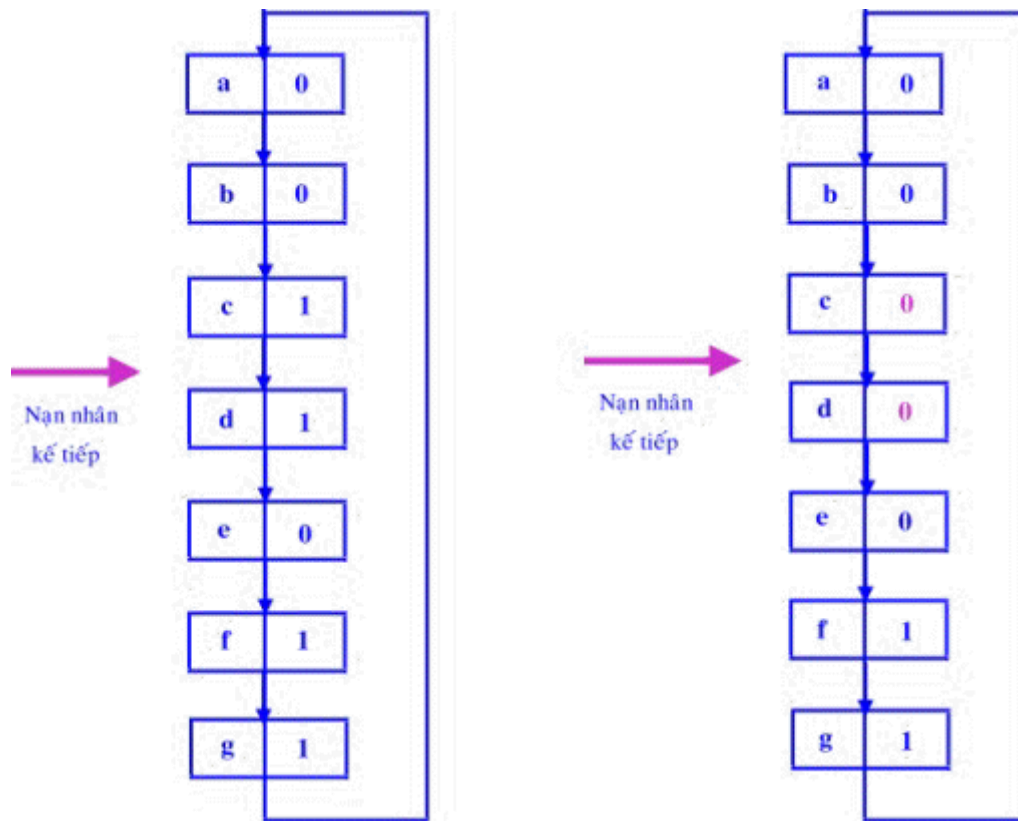
- Thảo luận: Số lượng các bit lịch sử có thể thay đổi tùy theo phần cứng, và phải được chọn sao cho việc cập nhật là nhanh nhất có thể.

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

- Tiếp cận: Sử dụng một bit reference duy nhất. Thuật toán cơ sở vẫn là FIFO, tuy nhiên khi chọn được một trang theo tiêu chuẩn FIFO, kiểm tra bit reference của trang đó :
  - Nếu giá trị của bit reference là 0, thay thế trang đã chọn.
  - Ngược lại, cho trang này một cơ hội thứ hai, và chọn trang FIFO tiếp theo.
  - Khi một trang được cho cơ hội thứ hai, giá trị của bit reference được đặt lại là 0, và thời điểm vào Ready List được cập nhật lại là thời điểm hiện tại.
  - Một trang đã được cho cơ hội thứ hai sẽ không bị thay thế trước khi hệ thống đã thay thế hết những trang khác. Hơn nữa, nếu trang thường xuyên được sử dụng, bit reference của nó sẽ duy trì được giá trị 1, và trang hầu như không bao giờ bị thay thế.

- Thảo luận:

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



**Hình 2.29** Thuật toán thay thế trang <<ơ hội thứ hai >>

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

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

**d) Các thuật toán thống kê**

- Tiếp cận: sử dụng một biến đếm lưu trữ số lần truy xuất đến một trang, và phát triển hai thuật toán sau :
- *Thuật toán LFU*: thay thế trang có giá trị biến đếm nhỏ nhất, nghĩa là trang ít được sử dụng nhất.



• **Thuật toán MFU:** thay thế trang có giá trị biến đếm lớn nhất, nghĩa là trang được sử dụng nhiều nhất (most frequently used).

### III. Cấp phát khung trang

Vấn đề đặt ra là làm thế nào để cấp phát một vùng nhớ tự do có kích thước cố định cho các tiến trình khác nhau?

Trong trường hợp đơn giản nhất của bộ nhớ ảo là hệ đơn nhiệm, có thể cấp phát cho tiến trình duy nhất của người dùng tất cả các khung trang trống.

Vấn đề nảy sinh khi kết hợp kỹ thuật phân trang theo yêu cầu với sự đa chương : cần phải duy trì nhiều tiến trình trong bộ nhớ cùng lúc, vậy mỗi tiến trình sẽ được cấp bao nhiêu khung trang.

#### ☞ Số khung trang tối thiểu:

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. 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 có đủ các khung trang để nạp tất cả các trang mà một chỉ thị duy nhất có thể truy xuất.

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.

#### ☞ Các thuật toán cấp phát khung trang

Có hai hướng tiếp cận:

##### ▶ Cấp phát cố định:

Cấp phát công bằng: nếu có  $m$  khung trang và  $n$  tiến trình, mỗi tiến trình được cấp  $m/n$  khung trang.

Cấp phát theo tỷ lệ: tùy vào kích thước của tiến trình để cấp phát số khung trang :

$s_i$  = kích thước của bộ nhớ ảo cho tiến trình  $p_i$

$S = \sum s_i$

$m$  = số lượng tổng cộng khung trang có thể sử dụng

Cấp phát  $a_i$  khung trang cho tiến trình  $p_i$  :  $a_i = (s_i / S) m$

▶ Cấp phát theo độ ưu tiên : sử dụng ý tưởng cấp phát theo tỷ lệ, nhưng số lượng khung trang cấp cho tiến trình phụ thuộc vào độ ưu tiên của tiến trình, hơn là phụ thuộc kích thước tiến trình:

Nếu tiến trình  $p_i$  phát sinh một lỗi trang, chọn một trong các khung trang của nó để thay thế, hoặc chọn một khung trang của tiến trình khác với độ ưu tiên thấp hơn để thay thế.

#### ☞ Thay thế trang toàn cục hay cục bộ

Có thể phân các thuật toán thay thế trang thành hai lớp chính:

▶ Thay thế toàn cục: khi lỗi trang xảy ra với một tiến trình , chọn trang « nạn nhân » từ tập tất cả các khung trang trong hệ thống, bất kể khung trang đó đang được cấp phát cho một tiến trình khác.

▶ Thay thế cục bộ: yêu cầu chỉ được chọn trang thay thế trong tập các khung trang được cấp cho tiến trình phát sinh lỗi trang.

Một khuyết điểm của thuật toán thay thế toàn cục là các tiến trình không thể kiểm soát được tỷ lệ phát sinh lỗi trang của mình. Vì thế, tuy thuật toán thay thế toàn cục nhìn chung cho phép hệ thống có nhiều

khả năng xử lý hơn, nhưng nó có thể dẫn hệ thống đến tình trạng *trì trệ toàn bộ* (thrashing).

### III.1. Trì trệ toàn bộ hệ thống (Thrashing)

Nếu một tiến trình không có đủ các khung trang để chứa những trang cần thiết cho xử lý, thì nó sẽ thường xuyên phát sinh các lỗi trang, và vì thế phải dùng đến rất nhiều thời gian sử dụng CPU để thực hiện thay thế trang. Một hoạt động phân trang như thế được gọi là *sự trì trệ* (thrashing). Một tiến trình lâm vào trạng thái trì trệ nếu nó sử dụng nhiều thời gian để thay thế trang hơn là để xử lý !

Hiện tượng trì trệ này ảnh hưởng nghiêm trọng đến hoạt động hệ thống, xét tình huống sau :

- Hệ điều hành giám sát việc sử dụng CPU.
- Nếu hiệu suất sử dụng CPU quá thấp, hệ điều hành sẽ nâng mức độ đa chương bằng cách đưa thêm một tiến trình mới vào hệ thống.
- Hệ thống có thể sử dụng thuật toán thay thế toàn cục để chọn các trang nạn nhân thuộc một tiến trình bất kỳ để có chỗ nạp tiến trình mới, có thể sẽ thay thế cả các trang của tiến trình đang xử lý hiện hành.
- Khi có nhiều tiến trình trong hệ thống hơn, thì một tiến trình sẽ được cấp ít khung trang hơn, và do đó phát sinh nhiều lỗi trang hơn.
- Khi các tiến trình phát sinh nhiều lỗi trang, chúng phải trải qua nhiều thời gian chờ các thao tác thay thế trang hoàn tất, lúc đó hiệu suất sử dụng CPU lại giảm
- Hệ điều hành lại quay trở lại bước 1...

Theo kịch bản trên đây, hệ thống sẽ lâm vào tình trạng luẩn quẩn của việc giải phóng các trang để cấp phát thêm khung trang cho một tiến trình, và các tiến trình khác lại thiếu khung trang...và các tiến trình không thể tiếp tục xử lý. Đây chính là tình trạng *trì trệ toàn bộ* hệ thống. Khi tình trạng trì trệ này xảy ra, hệ thống gần như mất khả năng xử lý, tốc độ phát sinh lỗi trang tăng cao khủng khiếp, không công việc nào có thể kết thúc vì tất cả các tiến trình đều bận rộn với việc phân trang !

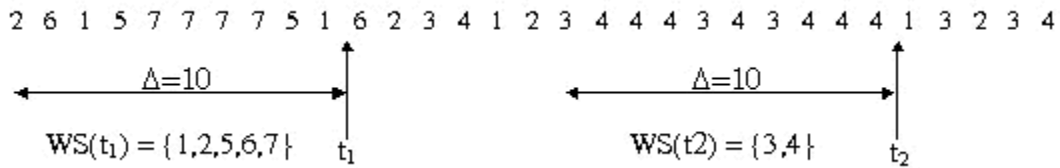
Để ngăn cản tình trạng trì trệ này xảy ra, cần phải cấp cho tiến trình *đủ* các khung trang cần thiết để hoạt động. Vấn đề cần giải quyết là làm sao biết được tiến trình cần bao nhiêu trang?

**Mô hình cục bộ ( Locality ) :** theo lý thuyết cục bộ, thì khi một tiến trình xử lý, nó có khuynh hướng di chuyển từ nhóm trang cục bộ này đến nhóm trang cục bộ khác . Một nhóm trang cục bộ là một tập các trang đang được tiến trình dùng đến trong một khoảng thời gian. Một chương trình thường bao gồm nhiều nhóm trang cục bộ khác nhau và chúng có thể giao nhau.

#### \*III.1.1. Mô hình « tập làm việc » (working set)

##### ➤ Tiếp cận :

Mô hình working set đặt cơ sở trên lý thuyết cục bộ. Mô hình này sử dụng một tham số  $\Delta$ , để định nghĩa một cửa sổ cho *working set*. Giả sử khảo sát  $\Delta$  đơn vị thời gian (lần truy xuất trang) cuối cùng, tập các trang được tiến trình truy xuất đến trong  $\Delta$  lần truy cập cuối cùng này được gọi là *working set* của tiến trình tại thời điểm hiện tại. Nếu một trang đang được tiến trình truy xuất đến, nó sẽ nằm trong *working set*, nếu nó không được sử dụng nữa, nó sẽ bị loại ra khỏi *working set* của tiến trình sau  $\Delta$  đơn vị thời gian kể từ lần truy xuất cuối cùng đến nó. Như vậy *working set* chính là một sự xấp xỉ của khái niệm nhóm trang cục bộ.



**Hình 2.30** Mô hình working set

Một thuộc tính rất quan trọng của working set là kích thước của nó. Nếu tính toán kích thước working set,  $WSS_i$ , cho mỗi tiến trình trong hệ thống, thì có thể xem như :

$$D = \sum WSS_i$$

với  $D$  là tổng số khung trang yêu cầu cho toàn hệ thống. Mỗi tiến trình sử dụng các trang trong working set của nó, nghĩa là tiến trình  $i$  yêu cầu  $WSS_i$  khung trang. Nếu tổng số trang yêu cầu vượt quá tổng số trang có thể sử dụng trong hệ thống ( $D > m$ ), thì sẽ xảy ra tình trạng trì trệ toàn bộ.

#### 🔗 Sử dụng:

Hệ điều hành giám sát working set của mỗi tiến trình và cấp phát cho tiến trình tối thiểu các khung trang để chứa đủ working set của nó. Như vậy một tiến trình mới chỉ có thể được nạp vào hệ thống khi có đủ khung trang tự do cho working set của nó. Nếu tổng số khung trang yêu cầu của các tiến trình trong hệ thống vượt quá các khung trang có thể sử dụng, hệ điều hành chọn một tiến trình để tạm dừng, giải phóng bớt các khung trang cho các tiến trình khác hoàn tất.

#### 🔗 Thảo luận:

- Chiến lược working set đã loại trừ được tình trạng trì trệ trong khi vẫn đảm bảo mức độ đa chương của hệ thống là cao nhất có thể, cho phép sử dụng tối ưu CPU.
- Điểm khó khăn của mô hình này là theo vết của các working set của tiến trình trong từng thời điểm. Có thể xấp xỉ mô hình working set với một ngắt đồng hồ sau từng chu kỳ nhất định và một bit reference:
  - ▶ phát sinh một ngắt đồng hồ sau từng  $T$  lần truy xuất bộ nhớ.
  - ▶ khi xảy ra một ngắt đồng hồ, kiểm tra các trang có bit reference là 1, các trang này được xem như thuộc về working set.
- Một hệ thống sử dụng kỹ thuật phân trang theo yêu cầu thuần túy (một trang không bao giờ được nạp trước khi có yêu cầu truy xuất) để lộ một đặc điểm khá bất lợi : một số lượng lớn lỗi trang xảy ra khi khởi động tiến trình. Tình trạng này là hậu quả của khuynh hướng đạt tới việc đưa nhóm trang cục bộ vào bộ nhớ. Tình trạng này cũng có thể xảy ra khi một tiến trình bị chuyển tạm thời ra bộ nhớ phụ, khi được tái kích hoạt, tất cả các trang của tiến trình đã được chuyển lên đĩa phải được mang trở lại vào bộ nhớ, và một loạt lỗi trang lại xảy ra. Để ngăn cản tình hình lỗi trang xảy ra quá nhiều tại thời điểm khởi động tiến trình, có thể sử dụng kỹ thuật tiên phân trang (prepaging) : nạp vào bộ nhớ một lần tất cả các trang trong working set của tiến trình.

### III.2. Tần suất xảy ra lỗi trang

**Tiếp cận:** Tần suất lỗi trang rất cao khiến tình trạng trì trệ hệ thống có thể xảy ra.

- Khi tần suất lỗi trang quá cao, tiến trình cần thêm một số khung trang.
- Khi tần suất lỗi trang quá thấp, tiến trình có thể sở hữu nhiều khung trang hơn mức cần thiết.

Có thể thiết lập một giá trị chặn trên và chặn dưới cho tần suất xảy ra lỗi trang, và trực tiếp ước lượng và kiểm soát tần suất lỗi trang để ngăn chặn tình trạng trì trệ xảy ra :

- Nếu tần suất lỗi trang vượt quá chặn trên, cấp cho tiến trình thêm một khung trang
- Nếu tần suất lỗi trang thấp hơn chặn dưới, thu hồi bớt một khung trang từ tiến trình

## IV. Tóm tắt

► Các kỹ thuật hỗ trợ các mô hình tổ chức bộ nhớ hiện đại :

• *Swapping* : sử dụng thêm bộ nhớ phụ để lưu trữ tạm các tiến trình đang bị khóa, nhờ vậy có thể tăng mức độ đa chương của hệ thống với cấu hình máy có dung lượng bộ nhớ chính thấp.

• *Bộ nhớ ảo* : sử dụng kỹ thuật phân trang theo yêu cầu, kết hợp thêm kỹ thuật swapping để mở rộng bộ nhớ chính. Tách biệt không gian địa chỉ và không gian vật lý, nhờ đó có thể xử lý các chương trình có kích thước lớn hơn bộ nhớ vật lý thật sự

► Khi cài đặt bộ nhớ ảo, phải sử dụng một thuật toán thay thế trang thích hợp để chọn các trang bị chuyển tạm thời ra bộ nhớ phụ, dành chỗ trong bộ nhớ chính cho trang mới. Các thuật toán thay thế thường sử dụng là FIFO, LRU và các thuật toán xấp xỉ LRU, các thuật toán thống kê NFU, MFU...

► Khi mức độ đa chương tăng cao đến một chừng mực nào đó, hệ thống có thể lâm vào tình trạng trì trệ do tất cả các tiến trình đều thiếu khung trang. Có thể áp dụng mô hình working set để dành cho mỗi tiến trình đủ các khung trang cần thiết tại một thời điểm, từ đó có thể ngăn chặn tình trạng trì trệ xảy ra.

## 📌Củng cố bài học

Các câu hỏi cần trả lời được sau bài học này :

1. Bộ nhớ ảo là gì ?
2. Sự thật đằng sau ảo giác: giới hạn của bộ nhớ ảo ? Chi phí thực hiện?
3. Các vấn đề của bộ nhớ ảo : thay thế trang, cấp phát khung trang ?
4. Mô hình working set : khái niệm, cách tính trong thực tế, sử dụng ?

## 📌Bài Tập

🔍 **Bài 1.** Khi nào thì xảy ra lỗi trang ? Mô tả xử lý của hệ điều hành khi có lỗi trang.

🔍 **Bài 2.** Giả sử có một chuỗi truy xuất bộ nhớ có chiều dài  $p$  với  $n$  số hiệu trang khác nhau xuất hiện trong chuỗi. Giả sử hệ thống sử dụng  $m$  khung trang ( khởi động trống). Với một thuật toán thay thế trang bất kỳ :

- Cho biết số lượng tối thiểu các lỗi trang xảy ra ?
- Cho biết số lượng tối đa các lỗi trang xảy ra ?

🔍 **Bài 3.** 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, và cho offset. Cho biết kích thước một trang trong hệ thống, và địa chỉ ảo có bao nhiêu trang ?

🔍 **Bài 4.** Giả sử địa chỉ ảo 32-bit được phân tách thành 4 trường  $a, b, c, d$ . 3 trường đầu tiên được dùng

cho bảng trang tam cấp, trường thứ 4 dành cho offset. Số lượng trang có phụ thuộc vào cả kích thước 4 trường này không ? Nếu không, những trường nào ảnh hưởng đến số lượng trang, và những trường nào không ?

**❓ Bài 5.** 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)? Trong bảng trang nghịch đảo ?

**❓ Bài 6.** Một máy tính cung cấp cho người dùng một không gian địa chỉ ảo  $2^{32}$  bytes. Máy tính này có bộ nhớ vật lý  $2^{18}$  bytes. Bộ nhớ ảo được thực hiện với kỹ thuật phân trang, kích thước trang là 4096 bytes. Một tiến trình của người dùng phát sinh địa chỉ ảo 11123456. Giải thích cách hệ thống chuyển đổi địa chỉ ảo này thành địa chỉ vật lý tương ứng. Phân biệt các thao tác phần mềm và phần cứng.

**❓ Bài 7.** 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 100nanoseconds. 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 access time) không vượt quá 200nanoseconds ?

**❓ Bài 8.** Xét các thuật toán thay thế trang sau đây. Xếp thứ tự chúng dựa theo tỷ lệ phát sinh lỗi trang của chúng. Phân biệt các thuật toán chịu đựng nghịch lý Belady và các thuật toán không bị nghịch lý này ảnh hưởng.

- a) LRU
- b) FIFO
- c) Chiến lược thay thế tối ưu
- d) Cơ hội thứ hai

**❓ Bài 9.** Một máy tính có 4 khung trang. Thời điểm nạp, thời điểm truy cập cuối cùng, và các bit reference (R), modify (M) của mỗi trang trong bộ nhớ được cho trong bảng sau :

| Trang | Nạp | Truy cập cuối | R | M |
|-------|-----|---------------|---|---|
| 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"

**❓ Bài 10.** Xét mảng hai chiều A:

```
var A: array [1 ..100, 1..100] of integer;
```

Với A[1][1] được lưu trữ tại vị trí 200, trong bộ nhớ tổ chức theo kỹ thuật phân trang với kích thước trang là 200. Một tiến trình trong trang 0 (chiếm vị trí từ 0 đến 199) sẽ thao tác ma trận này ; như vậy mỗi chỉ thị sẽ được nạp từ trang 0. Với 3 khung trang, có bao nhiêu lỗi trang sẽ phát sinh khi thực hiện vòng lặp sau đây để khởi động mảng, sử dụng thuật toán thay thế LRU , và giả sử khung trang 1 chứa tiến trình, hai khung trang còn lại được khởi động ở trạng thái trống :

```
a. for j:= 1 to 100 do
    for i :=1 to 100 do A[i][j]:= 0;

b. for i :=1 to 100 do
    for j:=1 to 100 do A[i][j]:= 0;
```

**❓ Bài 11.** Xét chuỗi truy xuất bộ nhớ sau:

1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6

Có bao nhiêu lỗi trang xảy ra khi sử dụng các thuật toán thay thế sau đây, giả sử có 1, 2, 3, 4, 5, 6, 7 khung trang ?

- a) LRU
- b) FIFO
- c) Chiến lược tối ưu

**❓ Bài 12.** Trong một hệ thống sử dụng kỹ thuật phân trang theo yêu cầu, xét hai đoạn chương trình sau đây:

```
const N = 1024*1024
var A,B : array [1..N] of integer;
[Program 1]
  for i:=1 to N do
    A[i]:=i;
  for i:=1 to N do
    B[A[i]]:=random(N) ;
[Program 2]
  for i:=1 to N do
    A[i]:= random(N);
  for i:=1 to N do
    B[A[i]]:=i;
```

**❓ Bài 13.** Giả sử có một máy tính đồ chơi sử dụng 7-bit địa chỉ. Kích thước một trang là 8 bytes, và 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 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 ?

**❓ Bài 14.** 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 ?

## BÀI 8 HỆ THỐNG QUẢN LÝ TẬP TIN

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

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

Bài học này giúp chúng ta hiểu được tập tin là gì, cách thức tổ chức và quản lý tập tin như thế nào. Từ đó giúp chúng ta hiểu được các cơ chế cài đặt hệ thống tập tin trên các hệ điều hành.

Bài học này đòi hỏi những kiến thức về : các thao tác với tập tin, một số tính chất của tập tin ở góc độ người sử dụng và những kiến thức về cấu trúc dữ liệu cũng như về kiến trúc máy tính phần cấu trúc và tổ chức lưu trữ của đĩa.

### I. CÁC KHÁI NIỆM CƠ BẢN

#### I.1 Bộ nhớ ngoài

Máy tính phải sử dụng thiết bị có khả năng lưu trữ trong thời gian dài (long-term) vì :

- o Phải chứa những lượng thông tin rất lớn (giữ vé máy bay, ngân hàng...)
- o Thông tin phải được lưu giữ một thời gian dài trước khi xử lý
- o Nhiều tiến trình có thể truy cập thông tin cùng lúc.

Giải pháp là sử dụng các thiết bị lưu trữ bên ngoài gọi là bộ nhớ ngoài.

#### I.2 Tập tin và thư mục

##### • *Tập tin*

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

##### • *Thư mục*

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



coi như là tập tin.

### I.3 Hệ thống quản lý tập tin

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

## II. MÔ HÌNH TỔ CHỨC VÀ QUẢN LÝ CÁC TẬP TIN

### II.1 Mô hình

 Tập tin :

*\*Tên tập tin :*

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

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

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

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

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

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

*\*Cấu trúc của tập tin :*

Gồm 3 loại :

- Dãy tuần tự các byte không cấu trúc : hệ điều hành không biết nội dung của tập tin:MS-DOS và UNIX sử dụng loại này.

- Dãy các record có chiều dài cố định.
- Cấu trúc cây : gồm cây của những record, không cần thiết có cùng độ dài, mỗi record có một trường khóa giúp cho việc tìm kiếm nhanh hơn.

### \*Kiểu tập tin :

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

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

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

**Tập tin nhị phân** : có cấu trúc khác tập tin văn bản. Mặc dù về mặt kỹ thuật , tập tin nhị phân gồm dãy các byte , nhưng hệ điều hành chỉ thực thi tập tin đó nếu nó có cấu trúc đúng. Ví dụ một tập tin nhị phân thi hành được của UNIX. Thường thường nó bao gồm năm thành phần : header, text, data, relocation bits, symbol table. Header bắt đầu bởi byte nhận diện cho biết đó là tập tin thi hành. Sau đó là 16 bit cho biết kích thước các thành phần của tập tin, địa chỉ bắt đầu thực hiện và một số bit cờ. Sau header là dữ liệu và text của tập tin. Nó được nạp vào bộ nhớ và định vị lại bởi những bit relocation. Bảng symbol được dùng để debug.

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

|                       |  |                      |              |
|-----------------------|--|----------------------|--------------|
| Số hiệu               |  |                      | Tên          |
| Kích thước<br>Text    |  | Header               | Đơn thể      |
| Kích thước<br>dữ liệu |  | Đối                  | Ngày         |
| Kích thước<br>BSS     |  | Tượng                | Người sở hữu |
| Kích thước<br>ST      |  | Đơn                  | Bảo vệ       |
| Đầu vào               |  | Thế                  | Kích thước   |
|                       |  | Header               |              |
| Cờ                    |  |                      |              |
| Vùng Text             |  | Đối tượng<br>Đơn thể |              |
| Vùng dữ liệu          |  | Header               |              |
| Bit<br>Định vị        |  | Đối tượng            |              |
| Bảng<br>Đối tượng     |  | Đơn thể              |              |

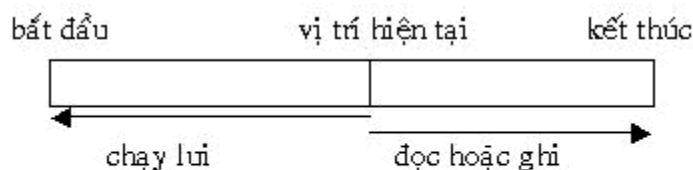
**Hình 8.1** Cấu trúc tập tin nhị phân trong UNIX

**\*Truy xuất tập tin :**

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

Kiểu truy xuất tập tin đơn giản nhất là truy xuất tuần tự . Tiến trình đọc tất cả các byte trong tập tin theo thứ tự từ đầu. Các trình soạn thảo hay trình biên dịch cũng truy xuất tập tin theo cách này. Hai thao tác chủ yếu trên tập tin là đọc và ghi. Thao tác đọc sẽ đọc một mẫu tin tiếp theo trên tập tin và tự động tăng con trỏ tập tin. Thao tác ghi cũng tương tự như vậy. Tập tin có thể tự khởi động lại từ vị trí đầu tiên và trong một số hệ thống tập tin cho phép di chuyển con trỏ tập tin đi tới hoặc đi lui n mẫu tin.

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



**Hình 8.2** Truy xuất tuần tự trên tập tin

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

#### **\*Thuộc tính tập tin :**

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

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

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

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

**Hình 8.3** Một số thuộc tính thông dụng của tập tin

📁 **Thư mục :**

### **\*HỆ THỐNG THƯ MỤC THEO CẤP BẠC :**

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

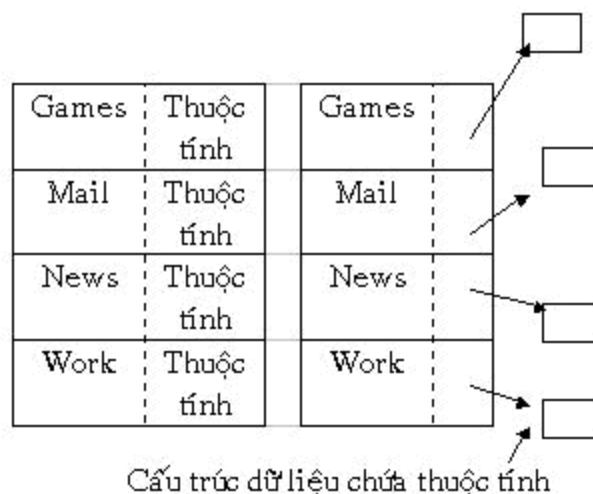
Khi một tập tin được mở, hệ điều hành tìm trên thư mục của nó cho tới khi tìm thấy tên của tập tin được mở. Sau đó nó sẽ xác định thuộc tính cũng như địa chỉ lưu trữ trên đĩa và đưa vào một bảng trong bộ nhớ. Những truy xuất sau đó thực hiện trong bộ nhớ chính.

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

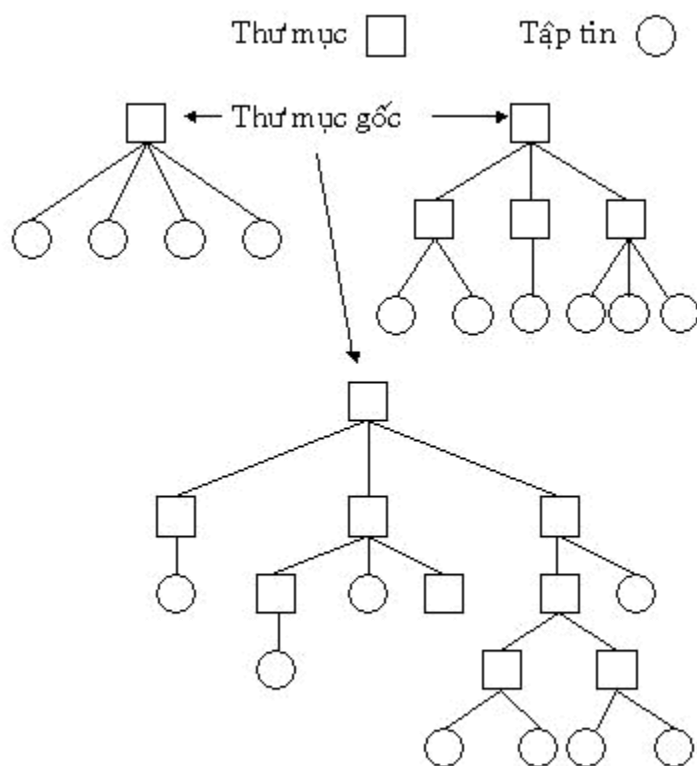
Cách thứ hai là có một thư mục gốc và trong đó có nhiều thư mục con, trong mỗi thư mục con chứa tập tin của người sử dụng (còn gọi là thư mục hai cấp), cách này tránh được trường hợp xung đột tên nhưng cũng còn khó khăn với người dùng có nhiều tập tin. Người sử dụng luôn muốn nhóm các ứng dụng lại một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

Từ đó, hệ thống thư mục theo cấp bậc (còn gọi là cây thư mục) được hình thành với mô hình một thư mục có thể chứa tập tin hoặc một thư mục con và cứ tiếp tục như vậy hình thành cây thư mục như trong các hệ điều hành DOS, Windows, v. v...

Ngoài ra, trong một số hệ điều hành nhiều người dùng, hệ thống còn xây dựng các hình thức khác của cấu trúc thư mục như cấu trúc thư mục theo đồ thị có chu trình và cấu trúc thư mục theo đồ thị tổng quát. Các cấu trúc này cho phép các người dùng trong hệ thống có thể liên kết với nhau thông qua các thư mục chia sẻ.



**Hình 8.4** Hai dạng cấu trúc thư mục



**Hình 8.5** Hệ thống thư mục theo cấp bậc.

### \*ĐƯỜNG DẪN :

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

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

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

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

## II.2 Các chức năng

### Tập tin :

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

## **Thư mục :**

- **Tạo** : một thư mục được tạo, nó rỗng, ngoại trừ "." và ".." được đặt tự động bởi hệ thống.
- **Xóa** : xóa một thư mục, chỉ có thư mục rỗng mới bị xóa, thư mục chứa "." và ".." coi như là thư mục rỗng.
- **Mở thư mục** : thư mục có thể được đọc. Ví dụ để liệt kê tất cả tập tin trong một thư mục, chương trình liệt kê mở thư mục và đọc ra tên của tất cả tập tin chứa trong đó. Trước khi thư mục được đọc, nó phải được mở ra trước.
- **Đóng thư mục** : khi một thư mục đã được đọc xong, phải đóng thư mục để giải phóng vùng nhớ.
- **Đọc thư mục** : Lệnh này trả về entry tiếp theo trong thư mục đã mở. Thông thường có thể đọc thư mục bằng lời gọi hệ thống READ, lệnh đọc thư mục luôn luôn trả về một entry dưới dạng



chuẩn .

- *Đổi tên* :cũng như tập tin, thư mục cũng có thể được đổi tên.
- *Liên kết* :kỹ thuật này cho phép một tập tin có thể xuất hiện trong nhiều thư mục khác nhau. Khi có yêu cầu, một liên kết sẽ được tạo giữa tập tin và một đường dẫn được cung cấp.
- *Bỏ liên kết* :Nếu tập tin chỉ còn liên kết với một thư mục, nó sẽ bị loại bỏ hoàn toàn khỏi hệ thống, nếu nhiều thì nó bị giảm chỉ số liên kết.

### **Câu hỏi kiểm tra kiến thức**

1. Tập tin là gì ? Thư mục là gì ? Tại sao phải quản lý tập tin và thư mục ?
2. Tập tin có những đặc tính gì ? Những đặc tính nào là quan trọng ? Tại sao ?
3. Nêu các chức năng của tập tin và thư mục.

## BÀI 9 CÁC PHƯƠNG PHÁP CÀI ĐẶT HỆ THỐNG QUẢN LÝ TẬP TIN

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

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

Bài học này giúp chúng ta nắm đặc điểm cũng như ưu và khuyết điểm của các phương pháp tổ chức quản lý tập tin trên đĩa và một số vấn đề liên quan khác nhờ đó có thể hiểu được cách các hệ điều hành cụ thể quản lý tập tin như thế nào.

Bài học này đòi hỏi những kiến thức về : mô hình tổ chức các tập tin và thư mục cũng và một số cấu trúc dữ liệu.

## I. BẢNG QUẢN LÝ THƯ MỤC, TẬP TIN

## I.1 Khái niệm

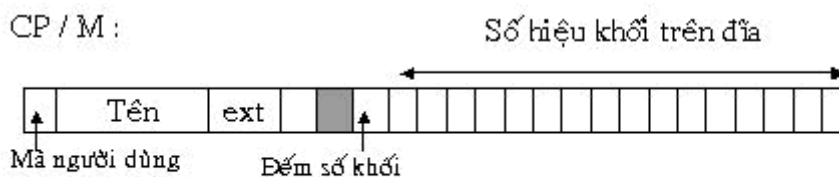
Trước khi tập tin được đọc, tập tin phải được mở, để mở tập tin hệ thống phải biết đường dẫn do người sử dụng cung cấp và được định vị trong cấu trúc đầu vào thư mục (directory entry).

Directory entry cung cấp các thông tin cần thiết để tìm kiếm các khối. Tùy thuộc vào mỗi hệ thống, thông tin là địa chỉ trên đĩa của toàn bộ tập tin, số hiệu của khối đầu tiên, hoặc là số I-node.

## II.2 Cài đặt

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

Ví dụ quản lý thư mục trong CP/M :



Hình 9.1

## II. BẢNG PHÂN PHỐI VÙNG NHỚ

### II.1 Khái niệm

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

### II.2 Các phương pháp

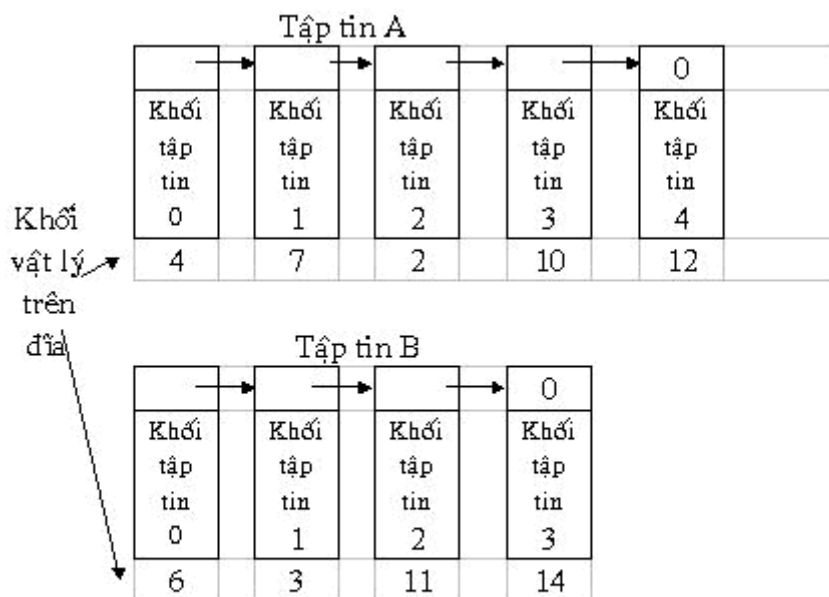
#### \*Định vị liên tiếp :

Lưu trữ tập tin trên dãy các khối liên tiếp.

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

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

#### \*Định vị bằng danh sách liên kết :



Hình 9.2 Định vị bằng danh sách liên kết

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

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

#### \*Danh sách liên kết sử dụng index :

## Khối vật lý

|    |    |                           |
|----|----|---------------------------|
| 0  |    |                           |
| 1  |    |                           |
| 2  | 10 |                           |
| 3  | 11 |                           |
| 4  | 7  | ← Tập tin A bắt đầu ở đây |
| 5  |    |                           |
| 6  | 3  | ← Tập tin B bắt đầu ở đây |
| 7  | 2  |                           |
| 8  |    |                           |
| 9  |    |                           |
| 10 | 12 |                           |
| 11 | 14 |                           |
| 12 | 0  |                           |
| 13 |    |                           |
| 14 | 0  |                           |
| 15 |    | ← Khối chưa sử dụng       |

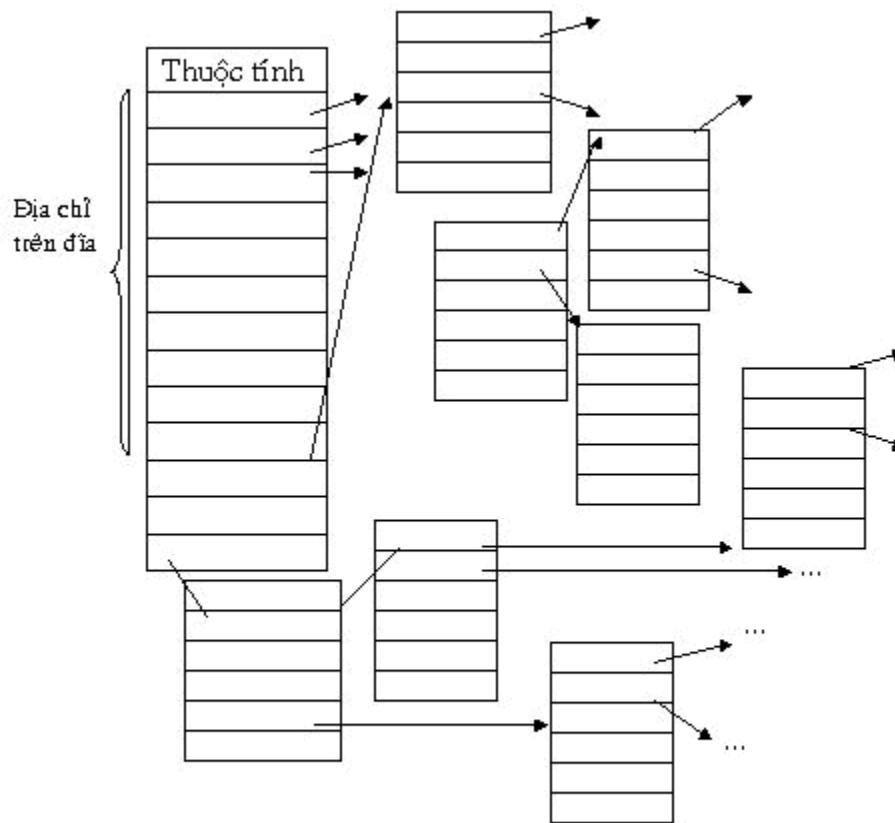
**Hình 9.3** Bảng chỉ mục của danh sách liên kết

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

### \*I-nodes :

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

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



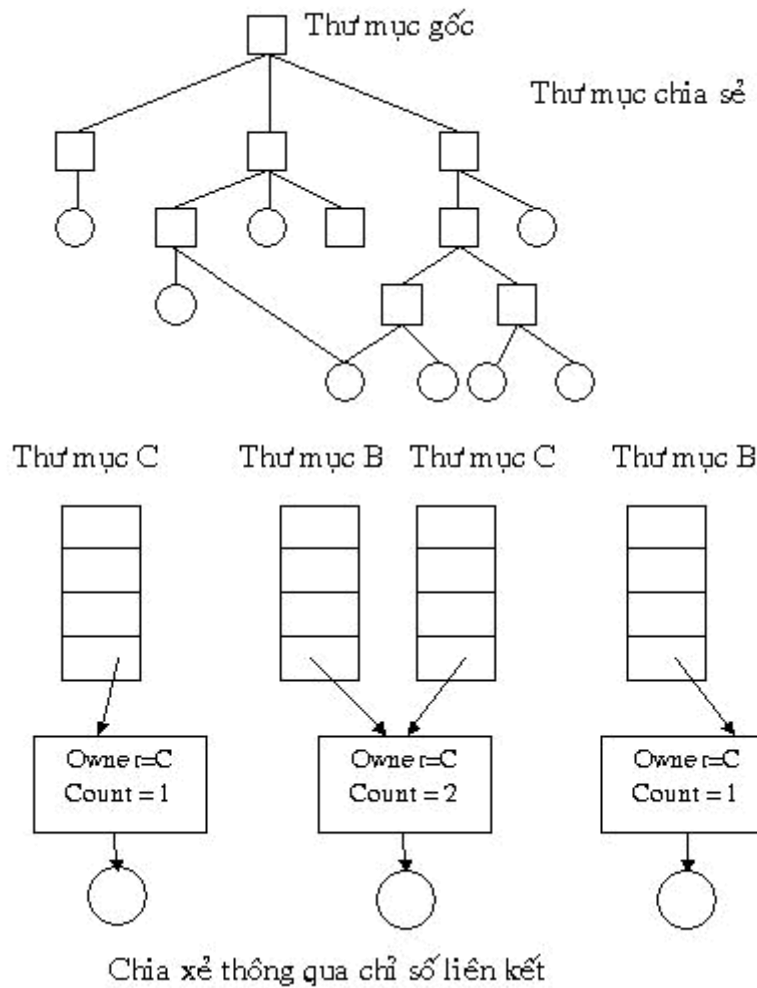
Hình 9.4 Cấu trúc của I-node

### III. TẬP TIN CHIA SẺ

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

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

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



Hình 9.5

## IV. QUẢN LÝ ĐĨA

Tập tin được lưu trữ trên đĩa, do đó việc quản trị đĩa là hết sức quan trọng trong việc cài đặt hệ thống tập tin. Có hai phương pháp lưu trữ : một là chứa tuần tự trên n byte liên tiếp, hai là tập tin được chia làm thành từng khối. Cách thứ nhất không hiệu quả khi truy xuất những tập tin có kích thước lớn, do đó hầu hết các hệ thống tập tin đều dùng khối có kích thước cố định.

### IV.1 Kích thước khối

Một vấn đề đặt ra là kích thước khối phải bằng bao nhiêu. Điều này phụ thuộc vào tổ chức của đĩa như số sector, số track, số cylinder. Nếu dùng một cylinder cho một khối cho một tập tin thì theo tính toán sẽ lãng phí đến 97% dung lượng đĩa. Nên thông thường mỗi tập tin thường được lưu trên một số khối. Ví dụ một đĩa có 32768 byte trên một track, thời gian quay là 16.67 msec, thời gian tìm kiếm trung bình là 30 msec thì thời gian tính bằng msec để đọc một khối kích thước k byte là :

$$30 + 8.3 + (k/32768) \times 16.67$$

Từ đó thống kê được kích thước khối thích hợp phải  $< 2K$  .

Thông thường kích thước khối là 512, 1K hay 2K.

## IV.2 Lưu giữa các khối trống

Có hai phương pháp. Một là sử dụng danh sách liên kết của khối đĩa. Mỗi khối chứa một số các địa chỉ các khối trống. Ví dụ một khối có kích thước 1 K có thể lưu trữ được 511 địa chỉ 16 bit. Một đĩa 20M cần khoảng 40 khối. Hai là, sử dụng bitmap. Một đĩa n khối sẽ được ánh xạ thành n bit với giá trị 1 là còn trống, giá trị 0 là đã lưu dữ liệu. Như vậy một đĩa 20M cần 20K bit để lưu trữ nghĩa là chỉ có khoảng 3 khối. Phương pháp thứ hai này thường được sử dụng hơn.

|     |     |     |                  |
|-----|-----|-----|------------------|
| 42  | 230 | 86  | 1001101101101100 |
| 136 | 162 | 234 | 0110110111110111 |
| 210 | 612 | 897 | 1010110110110110 |
| 97  | 342 | 422 | 0110110110111011 |
| 41  | 214 | 140 | 1110111011101111 |
| 63  | 160 | 223 | 1101101010001111 |
| 21  | 664 | 223 | 0000111011010111 |
| 48  | 216 | 160 | 1011101101101111 |
| 262 | 320 | 126 | 1100100011101111 |
|     |     |     |                  |
| 310 | 180 | 142 | 0111011101110111 |
| 516 | 482 | 141 | 1101111101110111 |

Danh sách liên kết

Bit map

Hình 9.6 Hai phương pháp lưu giữ khối trống

## V. ĐỘ AN TOÀN CỦA HỆ THỐNG TẬP TIN

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

### V.1 Quản lý khối bị hỏng

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

Có hai giải pháp : phần mềm và phần cứng.

Phần cứng là dùng một sector trên đĩa để lưu giữ danh sách các khối bị hỏng. Khi bộ kiểm soát trực hiện lần đầu tiên, nó đọc những khối bị hỏng và dùng một khối thừa để lưu giữ. Từ đó không cho truy cập những khối hỏng nữa.

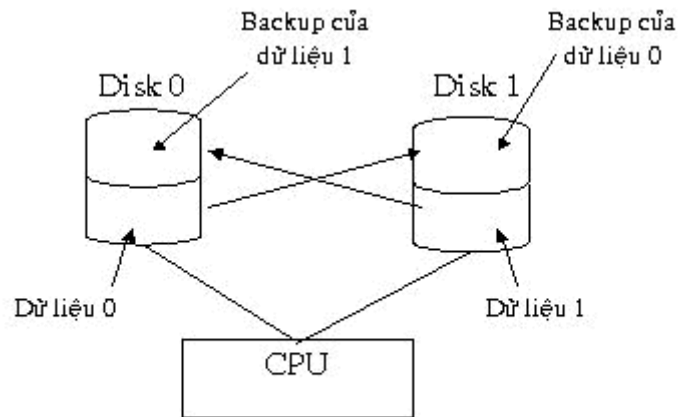
Phần mềm là hệ thống tập tin xây dựng một tập tin chứa các khối hỏng. Kỹ thuật này loại trừ chúng ra khỏi danh sách các khối trống, do đó nó sẽ không được cấp phát cho tập tin.

### V.2 Backup

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

Tập tin trên đĩa mềm được backup bằng cách chép lại toàn bộ qua một đĩa khác. Dữ liệu trên đĩa cứng nhỏ thì được backup trên các băng từ.

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



Hình 9.7 Backup

### V.3 Tính không đổi của hệ thống tập tin

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



|         |                        |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |                 |
|---------|------------------------|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|-----------------|
| Số khối | 0                      | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |                 |
|         | 1                      | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 0  | 1  | 1  | 1  | 0  | 0  | Khối đã sử dụng |
|         | 0                      | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1  | 0  | 0  | 0  | 1  | 1  | Khối trống      |
|         | Trạng thái bình thường |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |                 |
|         | 0                      | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |                 |
|         | 1                      | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 0  | 1  | 1  | 1  | 0  | 0  | Khối đã sử dụng |
|         | 0                      | 0 | 1 | 0 | 2 | 0 | 0 | 0 | 0 | 1 | 1  | 0  | 0  | 0  | 1  | 1  | Khối trống      |
|         | Mất khối               |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |                 |
|         | 0                      | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |                 |
|         | 1                      | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 0  | 1  | 1  | 1  | 0  | 0  | Khối đã sử dụng |
|         | 0                      | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1  | 0  | 0  | 0  | 1  | 1  | Khối trống      |
|         | Chồng khối trống       |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |                 |
|         | 0                      | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |                 |
|         | 1                      | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 0  | 1  | 1  | 1  | 0  | 0  | Khối đã sử dụng |
|         | 0                      | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1  | 0  | 0  | 0  | 1  | 1  | Khối trống      |
|         | Chồng khối dữ liệu     |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |                 |

Hình 9.8 Trạng thái của hệ thống tập tin

### Câu hỏi kiểm tra kiến thức

1. Vai trò của bảng thư mục tập tin
2. So sánh các phương pháp cài đặt bảng phân phối vùng nhớ.
3. Tập tin chia sẻ là gì ?
4. Vì sao phải lưu ý đến độ an toàn của hệ thống tập tin ?

### Bài tập

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 tập tin (hoặc thư mục), mỗi entry có kích thước 32 bytes. Một cluster = 2 sector. Đĩa sử dụng phương pháp định bằng bảng chỉ mục mỗi phần tử trong bảng có kích thước 12 bits. Hỏi muốn truy xuất cluster 10 thì phải đọc những sector nào ?

## BÀI 10 GIỚI THIỆU MỘT SỐ HỆ THỐNG TẬP TIN

Trong bài học này chúng ta sẽ tìm hiểu các phương pháp tổ chức quản lý tập tin của một số hệ điều hành sau:

- MS-DOS
- Windows 95
- Windows NT
- Unix

Bài học này giúp chúng ta hiểu được cách một số hệ điều hành thông dụng quản lý tập tin như thế nào. Bài học này đòi hỏi những kiến thức từ hai bài học trước.

### I. MS-DOS

#### I.1 Đặc điểm

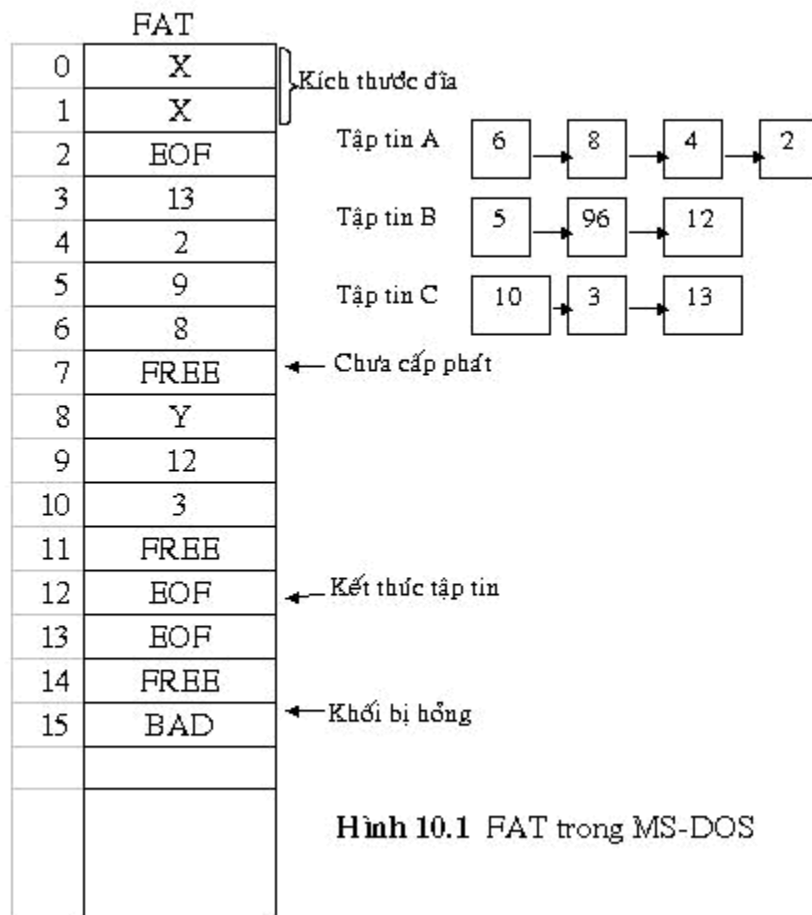
Hệ thống tập tin của MS-DOS bắt nguồn từ hệ thống tập tin của hệ điều hành CP/M. Nó có những đặc điểm như sau :

- Hệ thống cây thư mục.
- Khái niệm thư mục hiện hành.
- Đường dẫn tương đối và đường dẫn tuyệt đối.
- Thư mục "." và "..".
- Có tập tin thiết bị và tập tin khối.
- Tên tập tin 8+3.
- Đường dẫn \.
- Không phân biệt chữ thường và chữ hoa.
- Không có khái niệm người sở hữu.
- Không có khái niệm nhóm và bảo vệ.
- Không có liên kết.
- Không có mount hệ thống tập tin.
- Có thuộc tính của tập tin.

#### I.2 Cài đặt

Cài đặt trên đĩa mềm cũng tương tự như trên đĩa cứng, nhưng trên đĩa cứng phức tạp hơn. Phần này khảo sát trên đĩa cứng. Lúc đó, hệ điều hành MS-DOS được cài đặt trên một partition. Sector đầu tiên của *partition* là **bootsector**.

Sau bootsector là **FAT** (File Allocation Table), lưu giữ tất cả không gian trên đĩa theo phương pháp danh sách liên kết có chỉ mục. Thông thường có từ hai FAT trở lên để phòng hờ. Mỗi entry của FAT quản lý một khối (còn gọi là cluster được đánh số bắt đầu từ 2) trên đĩa. Kích thước khối được lưu trong bootsector thông thường từ 1 đến 8 sector. Có hai loại FAT là FAT 12 và FAT 16. FAT 12 có thể quản lý được 4096 khối còn FAT 16 có thể quản lý 64 K khối trên một partition.



Giá trị trong mỗi phần tử (entry) có ý nghĩa như sau :

|                 |                                      |
|-----------------|--------------------------------------|
| 0               | Cluster còn trống                    |
| (0)002 - (F)FEF | Cluster chứa dữ liệu của các tập tin |
| (F)FF0 - (F)FF6 | Dành riêng, không dùng               |
| (F)FF7          | Cluster hỏng                         |
| (F)FF8 - (F)FFF | Cluster cuối cùng của chuỗi          |

Có một ánh xạ một một giữa entry và khối ngoại trừ hai entry đầu tiên, dùng cho đĩa.

Khi hệ thống mở một tập tin, MS-DOS tìm trong bảng mô tả tập tin trong PSP, sau đó kiểm tra tên tập tin xem có phải là con, lpt, ... tiếp theo kiểm tra các đường dẫn để xác định vị trí trong bảng thư mục.

|                                    |
|------------------------------------|
| Tên tập tin (8bytes)               |
| Phần mở rộng (3bytes)              |
| Thuộc tính (1 byte)<br>A-D-V-S-H-R |
| Dành riêng (10bytes)               |
| Giờ (2bytes)                       |
| Ngày (2bytes)                      |
| Khối đầu tiên (2bytes)             |
| Kích thước tập tin (4bytes)        |

**Hình 10.2** Một entry thư mục trong MS-DOS

Bảng thư mục nằm ngay sau FAT, và mỗi entry là 32 byte. Mười một byte đầu tiên mô tả tên và phần mở rộng(không lưu trữ dấu chấm phân cách). Sau đó là byte thuộc tính, với giá trị :

- 1 : tập tin chỉ đọc
- 2 : tập tin ẩn
- 4 : tập tin hệ thống
- 8 : nhãn đĩa
- 16 : thư mục con
- 32 : tập tin chưa backup

Byte thuộc tính có thể được đọc ghi trong quá trình sử dụng. Tiếp theo là 10 byte trống dành riêng sử dụng sau này. Sau đó là 4 byte lưu trữ giờ, ngày với 6 bit cho giây, 4 bit cho giờ, 5 bit cho ngày, 4 bit cho tháng và 7 bit cho năm (từ 1980). Hai byte kế tiếp chứa số hiệu của khối đầu tiên (khối trong MS-DOS còn được gọi là cluster) và bốn byte sau cùng lưu trữ kích thước của tập tin.

**Ví dụ :**

Trên đĩa 1.44Mb, được format dưới hệ điều hành MS-DOS gồm có 2880 sector:

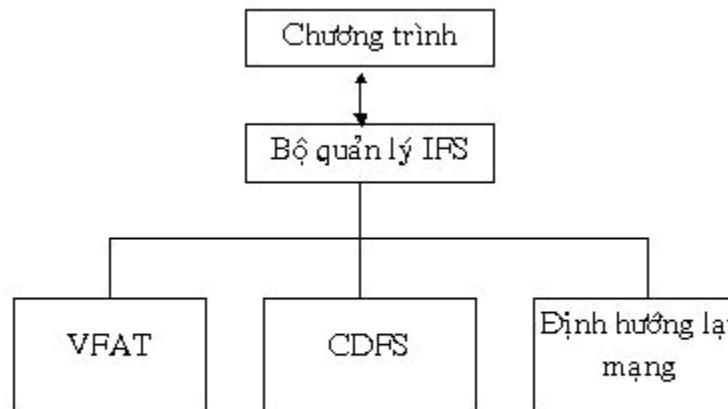
- Sector đầu tiên là bootsector, bao gồm bảng tham số vật lý của đĩa và chương trình khởi động của hệ điều hành (nếu có).
- 18 sector tiếp theo là FAT (FAT12), gồm 2 bảng, mỗi bảng 9 sector. Ba bytes đầu tiên của FAT lưu số hiệu loại đĩa.(240, 255, 255).
- 14 sector kế tiếp chứa bảng thư mục còn gọi là root directory entry table(RDET)
- Các sector còn lại dùng để lưu dữ liệu (1 cluser = 1 sector).

## II.WINDOWS95

### II.1 Bộ quản lý cài đặt hệ thống tập tin (IFS)

Hệ thống tập tin của Windows 95 là 32-bit và cho phép những hệ thống tập tin khác sử dụng được trên hệ thống này. Nó cũng làm cho máy tính nhanh hơn và linh hoạt hơn, có nghĩa là bạn có nhiều vùng hơn để cô lập xử lý các vấn đề.

Bộ quản lý IFS quản lý các thao tác bên trong của hệ thống tập tin được cài đặt. Các thành phần của IFS bao gồm IFSHLP.SYS và IFSMGR.VXD.



**Hình 10.3** Cấu trúc của bộ quản lý hệ thống tập tin được cài đặt

Trong Windows 95, hệ thống tập tin là một thành phần của ring 0 của hệ điều hành. Sau đây là các bước cài đặt của hệ thống tập tin trong Windows 95 :

- VFAT- Bảng định vị file ảo cho truy cập file 32-bit.
- CDFS- hệ thống tập tin của CD-ROM (thay thế MSCDEX)
- Bộ định hướng lại-Sử dụng cho truy xuất mạng.

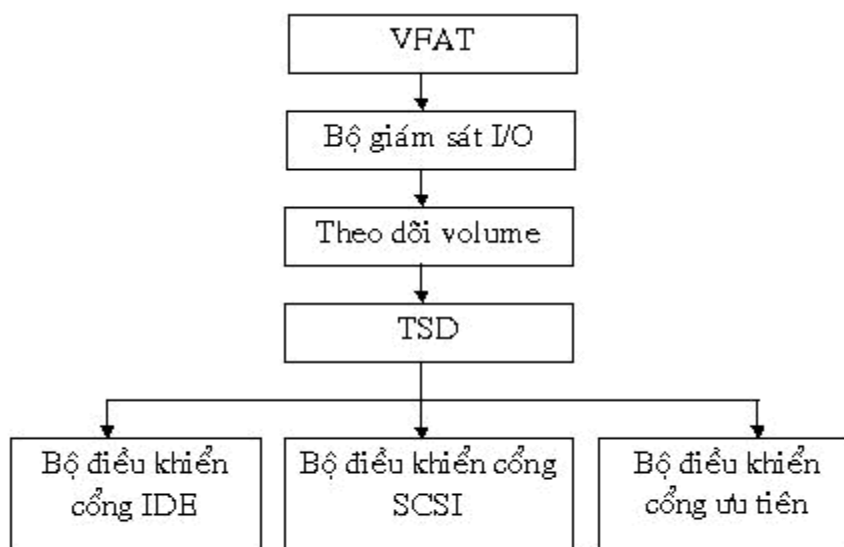
Người sử dụng cũng có thể cài đặt hệ thống tập tin khác. Ví dụ hệ thống tập tin cài đặt trên hệ thống Windows 95 có thể xử lý trên những hệ thống tập tin trên những hệ điều hành khác như Macintosh hay UNIX.

Bộ quản lý IFS quản lý vận chuyển nhập xuất tập tin cho chế độ bảo vệ của bộ định hướng lại, mode bảo vệ của server, VFAT, CDFS, và hệ thống tập tin của MS-DOS. Những hệ thống khác có thể được thêm vào trong tương lai.

### II.2 VFAT

VFAT là hệ thống tập tin FAT MS-DOS ảo 32 bit cung cấp truy xuất 32 bit cho Windows 95.

VFAT.VXD là driver điều khiển quá trình ảo hóa và sử dụng mã 32 bit cho tất cả các truy xuất tập tin.



**Hình 10.4** Tổ chức VFAT

VFAT chỉ cung cấp truy xuất ảo cho những volume đĩa cứng có các thành phần truy xuất đĩa 32 bit được cài đặt. Những dạng volume khác sẽ có cài đặt hệ thống tập tin cho chính nó. Ví dụ hệ thống tập tin của CD-ROM là CDFS.

VFAT ảo hóa đĩa và sử dụng mã 32 bit để truy xuất tập tin.

Bộ quản trị nhập/xuất được cài đặt từ Win 311 là \*KHỎIDEV. Bộ quản trị nhập/xuất của Windows 95 cung cấp \*KHỎIDEV những dịch vụ cho những driver FastDisk cũ. Ngoài ra nó có những chức năng sau :

- Đăng ký driver.
- Gửi và lập hàng đợi cho yêu cầu nhập/xuất
- Gửi những thông báo đến driver khi cần thiết.
- Cung cấp những dịch vụ cho driver để định vị bộ nhớ và hoàn tất yêu cầu nhập/xuất.

Theo dõi volume luôn hiện hữu khi có một thiết bị thông tin có thể được loại bỏ. Nó có trách nhiệm đảm bảo rằng thông tin đúng với thiết bị cũng như là kiểm tra và báo cáo những thông tin không thích hợp được loại bỏ hay chèn vào.

Nó thực hiện theo hai cách :

- Đối với đĩa không bảo vệ, theo dõi volume sẽ ghi một ID duy nhất vào đầu FAT của đĩa. ID này khác với số serial của volume.
- Trên đĩa có bảo vệ, theo dõi volume lưu trữ nhãn đĩa, số serial và khối tham số của BIOS.

**Bộ điều khiển mô tả kiểu (TSD)**

TSD làm việc với những thiết bị được mô tả. Ví dụ, đĩa mềm và cứng là một kiểu điều khiển nhưng đĩa CD là kiểu khác. TSD làm cho các yêu cầu nhập/xuất có hiệu lực, chuyển đổi những yêu cầu logic thành yêu cầu vật lý, và thông báo khi yêu cầu đã hoàn tất. Có thể xem TSD như một bộ dịch giữa bộ điều khiển vật lý và bộ quản trị nhập/xuất.

## II.3 VCACHE

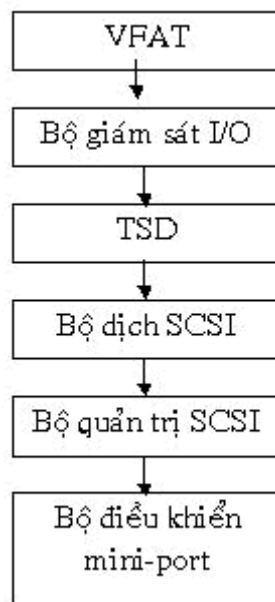
Vcache là vùng bộ nhớ mode bảo vệ được sử dụng bởi các bộ điều khiển hệ thống tập tin ở chế độ bảo vệ (ngoại trừ CDFS) : VFAT, VREDIR, NWREDIR. VCACHE được cài đặt tương tự như Win 3.11. Bộ điều khiển này thay thế cho phần mềm SMARTDrive disk cache 16-bit ở mode thực của MS-DOS và Win3.1. Đặc điểm của VCACHE là thuật toán thông minh hơn SMARTDrive trong lưu trữ thông tin nhập và xuất từ bộ điều khiển đĩa. VCACHE cũng quản lý vùng lưu trữ cho CDFS và NWREDIR 32-bit.

Việc sử dụng VCACHE là phụ thuộc với thiết bị. Ví dụ VCACHE dùng để truy xuất đĩa cứng khác với VCACHE truy xuất CD-ROM. Tất cả bộ điều khiển hệ thống tập tin của Windows 95 trừ CDFS đều sử dụng mode bảo vệ để đọc buffer. CDFS cung cấp cơ chế riêng. VFAT dùng VCACHE để giảm bớt việc ghi.

Bộ điều khiển cổng được thiết kế để cung cấp những truy xuất cho adapter.

## II.4 SCSI

Trong Windows 95, lớp SCSI là trung gian giữa lớp TSD và bộ điều khiển cổng. Có ba lớp SCSI được mô tả dưới đây:



Hình 10.5

### \*Bộ dịch SCSI :

Bộ dịch SCSI làm việc với tất cả những thiết bị SCSI như đĩa cứng, CD-ROM. Bộ dịch chịu trách nhiệm xây dựng khối mô tả lệnh SCSI cho những lớp của thiết bị SCSI và thực hiện tìm lỗi ở cấp thiết bị.

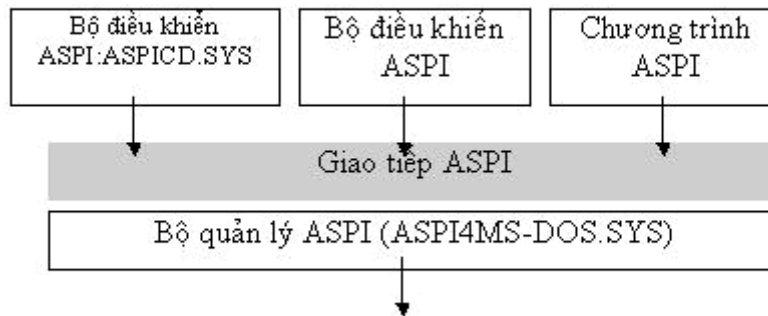
### \*Bộ quản trị SCSI :

Bộ quản trị SCSI quản lý việc giao tiếp giữa bộ dịch SCSI và bộ điều khiển miniport. Bộ điều khiển cổng SCSI khởi động bộ điều khiển miniport, chuyển đổi dạng yêu cầu nhập/xuất, thực hiện những thao tác giao tiếp với bộ điều khiển miniport. Khi liên kết với nó, bộ quản trị SCSI cung cấp cùng chức năng như Windows 95 chuẩn hoặc bộ điều khiển Fast Disk cũng như quan tâm đến những lớp cấp cao hơn.

### \*Bộ điều khiển miniport :

Làm việc với tập hợp những adapter SCSI được mô tả. Bộ điều khiển phụ thuộc vào những thủ tục lớp bên dưới để khởi động adapter, quản lý ngắt, chuyển những yêu cầu nhập/xuất cho thiết bị, và thực hiện những khôi phục lỗi ở mức adapter. Khi kết hợp với bộ quản lý SCSI, nó cung cấp cùng những chức năng như bộ điều khiển công chuẩn của Windows 95.

### ASPI trong Windows95



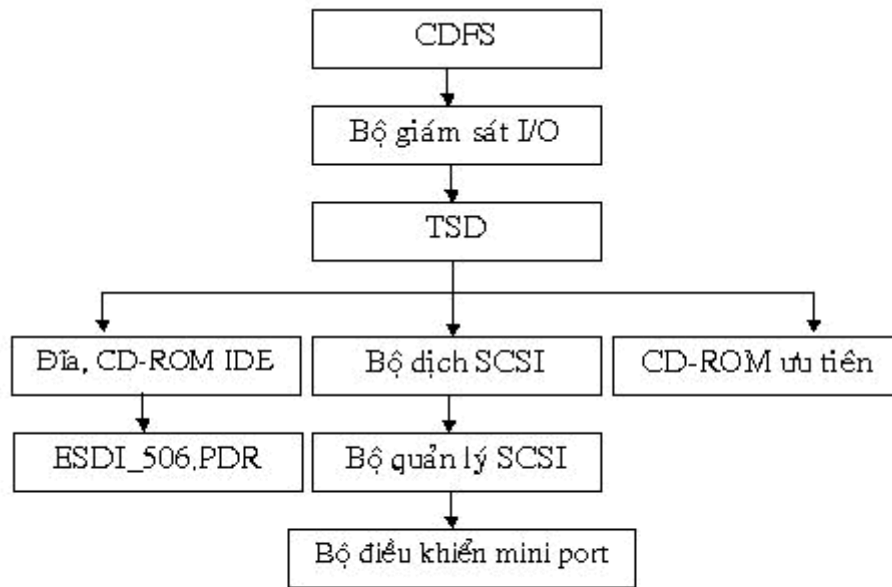
**Hình 10.6** Kiến trúc ASPI

Bộ ảnh xạ chương trình giao tiếp SCSI cao cấp (ASPI) của Windows 95 là APIX.VXD, cung cấp hỗ trợ mode bảo vệ cho những thiết bị và chương trình cần giao tiếp ASPI. Bộ quản lý ASPI cung cấp những giao tiếp giữa bộ điều khiển thiết bị và adapter chuẩn và thiết bị SCSI được nối trên adapter chủ. Bộ điều khiển ASPI gọi bộ quản trị ASPI. Bộ quản trị ASPI chuyển lời gọi cho CDB (Command Descriptor Khối) gọi tới những thành phần SCSI. Bộ quản trị ASPI cần thiết cho những trường hợp sau đây :

- Nhiều adapter chủ.
- Đĩa cứng SCSI với SCSI ID khác 0 hay 1.
- SCSI tape, máy in, máy vẽ, máy quét.

## II.5 CDFS





**Hình 10.7** Kiến trúc của CDFS

CDFS thay thế cho VFAT trong điều khiển thiết bị CD-ROM. Chức năng của CDFS tương tự như VFAT cho đĩa cứng. Các thành phần khác đều tương thích với version của CD-ROM. Một yêu cầu nhập/xuất tập tin trên CD-ROM được thực hiện bởi một trong bốn cách sau

- Bộ điều khiển IDE hỗ trợ mode bảo vệ : ESDI\_506.PDR.
- Bộ điều khiển SCSI hỗ trợ bộ điều khiển miniport mode bảo vệ.
- Bộ điều khiển ưu tiên hỗ trợ những bộ điều khiển ở mode bảo vệ được liệt kê trong tập tin ADAPTER.INF.
- Bộ điều khiển thiết bị CD-ROM ở mode thực sử dụng FAT MS-DOS và MSCDEX như hệ thống tập tin mở rộng CD-ROM cho FAT.

CDFS sử dụng bộ lưu trữ chia sẻ với VCACHE.

### **Hỗ trợ tên tập tin dài :(LFN)**

Windows 95 cho phép đặt tên tập tin dài không còn bị giới hạn bởi 8.3 nữa. Tuy nhiên, mỗi lần tạo(LFN), một tên 8.3 được tự động gán cho nó.

Một LFN có thể có tới 256 ký tự bao gồm luôn cả khoảng trắng. Đường dẫn có thể lên đến 260 ký tự. Việc gán tên 8.3 cho LFN theo quy tắc sau :

- Bỏ tất cả những ký tự đặc biệt sau : \ ? : \* “ < > |
- Lấy 6 ký tự đầu tiên của LFN thêm dấu ~ và một số bắt đầu từ 1 đến 9, nếu không đủ thì chỉ lấy 5 ký tự với số từ 10 đến 99 v.v...
- Đối với phần mở rộng, sử dụng 3 ký tự hợp lệ đầu tiên sau dấu chấm cuối cùng. Nếu không có dấu chấm thì không có phần mở rộng.

Khi sao chép tập tin dưới MS-DOS, LFN sẽ mất đi, chỉ còn lại tên 8.3 mà thôi. Nếu tập tin được tạo dưới MS-DOS thì LFN cũng chính là tên đó. Cũng có thể sử dụng LFN trong ứng dụng MS-DOS nhưng

khi đó, tên tập tin phải được đặt trong nháy kép. LFN sử dụng vùng dành riêng của FAT. Chương trình dùng phần dành riêng của FAT để tìm kiếm thông tin LFN.

### III. WINDOWS NT

Hệ điều hành WindowsNT hỗ trợ nhiều loại hệ thống tập tin bao gồm FAT trên MS-DOS và Windows95 và OS/2. Tuy nhiên nó cũng có hệ thống tập tin riêng, đó là NTFS.

#### III.1 Đặc điểm của NTFS

NTFS là một hệ thống tập tin mạnh và linh động, những đặc điểm nổi bật là :

- Khả năng phục hồi
- An toàn
- Quản lý được đĩa dung lượng lớn và kích thước tập tin lớn
- Quản lý hiệu quả.

#### III.2 Cấu trúc tập tin và volume của NTFS

NTFS sử dụng những khái niệm sau : Sector, cluster, volume

**Cluster** là đơn vị định vị cơ bản trong NTFS. Kích thước tập tin tối đa trong NTFS là  $2^{32}$  cluster, tương đương  $2^{48}$  bytes. Sự tương ứng giữa kích thước volume và cluster như hình sau :

| Kích thước volume | Số sector/cluster | Kích thước Cluster |
|-------------------|-------------------|--------------------|
| ≤ 512Mb           | 1                 | 512 bytes          |
| 512Mb – 1Gb       | 2                 | 1K                 |
| 1Gb – 2Gb         | 4                 | 2K                 |
| 2Gb – 4Gb         | 8                 | 4K                 |
| 4Gb – 8Gb         | 16                | 8K                 |
| 8Gb – 16Gb        | 32                | 16K                |
| 16Gb – 32Gb       | 64                | 32K                |
| > 32Gb            | 128               | 64K                |

**Hình 10.8** Windows NTFS Partition và kích thước cluster.

Cấu trúc volume của NTFS :

|                       |                   |                      |                  |
|-----------------------|-------------------|----------------------|------------------|
| Partition Boot Sector | Master File Table | Các tập tin hệ thống | Vùng các tập tin |
|-----------------------|-------------------|----------------------|------------------|

**Hình 10.9** Tổng quan volume NTFS

Bao gồm bốn vùng. Vùng thứ nhất là các sector khởi động của partition (có thể đến 16 sectors) bao gồm các thông tin về cấu trúc của volume, cấu trúc của hệ thống tập tin cũng như những thông tin và mã nguồn khởi động. Vùng tiếp theo là bảng Master File (MFT) lưu các thông tin về tất cả tập tin và thư mục trên volume NTFS này cũng như thông tin về các vùng trống. Sau vùng MFT là vùng các tập tin hệ thống có kích khoảng 1Mb bao gồm :

- MFT2 : bản sao của MFT

- Log file : thông tin về các giao tác dùng cho việc phục hồi.
- Cluster bitmap : biểu diễn thông tin lưu trữ của các cluster
- Bảng định nghĩa thuộc tính : định nghĩa các kiểu thuộc tính hỗ trợ cho volume đó.

MFT được tổ chức thành nhiều dòng. Mỗi dòng mô tả cho một tập tin hoặc một thư mục trên volume. Nếu kích thước tập tin nhỏ thì toàn bộ nội dung của tập tin được lưu trong dòng này. mỗi dòng cũng lưu những thuộc tính cho tập tin hay thư mục mà nó quản lý.

| 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 tập tin          |  |
| 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 10.10** Các kiểu thuộc tính của tập tin và thư mục của Windows NTFS

## IV. UNIX

### IV.1 Hệ thống tập tin của Unix :

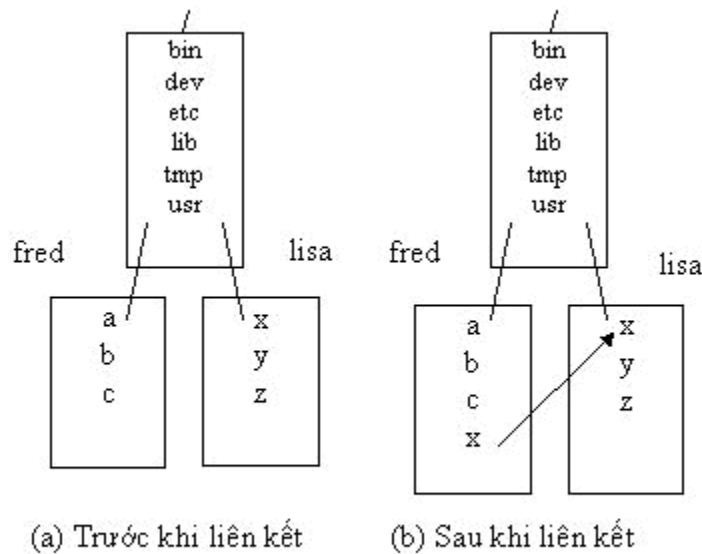
Một tập tin được mở với lời gọi hệ thống OPEN, với tham số đầu tiên cho biết đường dẫn và tên tập tin , tham số thứ hai cho biết tập tin được mở để đọc, ghi hay vừa đọc vừa ghi. Hệ thống kiểm tra xem tập tin có tồn tại không. Nếu có, nó kiểm tra bit quyền để xem có được quyền truy cập không, nếu có hệ thống sẽ trả về một số dương nhỏ gọi là biến mô tả tập tin cho nơi gọi. Nếu không nó sẽ trả về -1.

Khi một tiến trình bắt đầu, nó luôn có ba giá trị của biến mô tả tập tin : 0 cho nhập chuẩn, 1 cho xuất chuẩn và 2 cho lỗi chuẩn. Tập tin được mở đầu tiên sẽ có giá trị là 3 và sau đó là 4 ... Khi tập tin đóng, biến mô tả tập tin cũng được giải phóng.

Có hai cách mô tả tên tập tin trong UNIX. Cách thứ nhất là dùng đường dẫn tuyệt đối, tập tin được truy

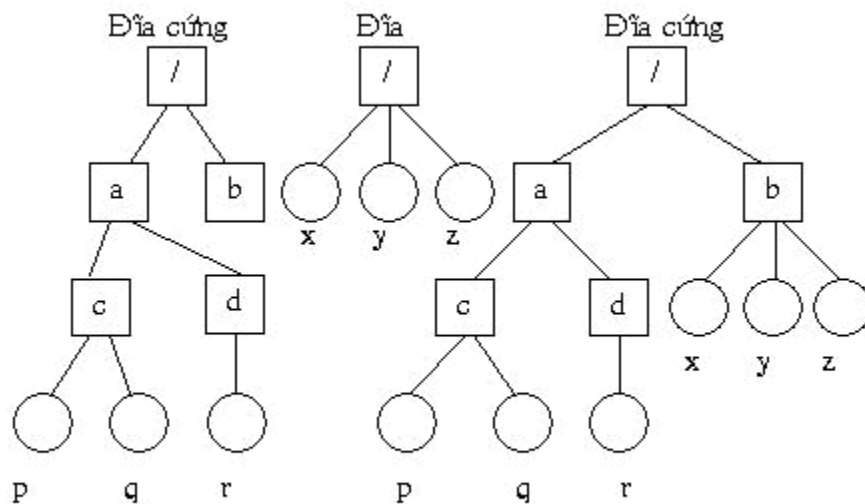
cập từ thư mục gốc. Thứ hai là dùng khái niệm thư mục làm việc hay thư mục hiện hành trong đường dẫn tương đối.

UNIX cung cấp đặc tính LINK, cho phép nhiều người sử dụng cùng dùng chung một tập tin, hay còn gọi là chia sẻ tập tin. Ví dụ như hình sau, fred và lisa cùng làm việc trong cùng một đề án, họ cần truy cập tập tin lẫn nhau. Giả sử fred cần truy cập tập tin x của lisa, anh ta sẽ tạo một entry mới trong thư mục của anh ta và sau đó có thể dùng x với nghĩa là /usr/lisa/x.



**Hình 10.11** Liên kết trong UNIX

Ngoài ra UNIX cho phép một đĩa có thể được mount thành một thành phần của hệ thống cây thư mục của một đĩa khác.



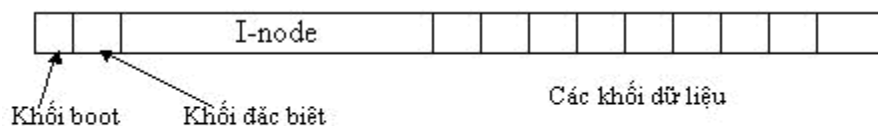
**Hình 10.12** Mount trong UNIX

Một đặc tính thú vị khác của hệ thống tập tin của UNIX là khóa (locking). Trong một số ứng dụng, một

số tiến trình có thể sử dụng cùng một tập tin cùng lúc. Có hai loại khóa là chia sẻ hay loại trừ. Nếu tập tin đã chứa khóa chia sẻ thì có thể đặt thêm một khóa chia sẻ nữa, nhưng không thể đặt một khóa loại trừ nhưng nếu đã được đặt khóa loại trừ thì không thể đặt thêm khóa nữa. Vùng khóa có thể được ghi chồng.

## IV.2 Cài đặt hệ thống tập tin của Unix

Hệ thống tập tin của UNIX thông thường được cài đặt trên đĩa như ở hình sau :



**Hình 10.13** Tổ chức hệ thống tập tin của Unix

Khối 0 thường chứa mã khởi động của hệ thống.

Khối 1 gọi là khối đặc biệt (super block), nó lưu giữ các thông tin quan trọng về toàn bộ hệ thống tập tin, bao gồm:

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

Nếu khối này bị hỏng, hệ thống tập tin 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 của hệ thống tập tin gốc 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.

Sau khối đặc biệt là các I-node, được đánh số từ một cho tới tối đa. Mỗi I-node có độ dài là 64 byte và mô tả cho một tập tin duy nhất (chứa thuộc tính và địa chỉ khối lưu trữ trên đĩa của tập tin).

Sau phần I-node là các khối dữ liệu. Tất cả tập tin và thư mục đều được lưu trữ ở đây.

Một entry của directory có 16 byte, trong đó 14 byte là tên của tập tin và 2 byte là địa chỉ của I-node. Để mở một tập tin trong thư mục làm việc, hệ thống chỉ đọc thư mục, so sánh tên được tìm thấy trong mỗi entry cho đến khi tìm được, từ đó xác định được chỉ số I-node và đưa vào bộ nhớ để truy xuất.

Tập tin được tạo hay tăng kích thước bằng cách sử dụng thêm các khối từ danh sách các khối còn trống. Ngược lại, khối được giải phóng sẽ trả về danh sách khối trống khi xóa tập tin. Super block sẽ chứa địa chỉ của 50 khối trống. Trong đó địa chỉ cuối cùng chứa địa chỉ của một khối chứa địa chỉ của 50 khối trống kế tiếp và cứ tiếp tục như thế. Unix sử dụng khối trống trong super block trước. Khi khối trống cuối cùng trong super block được sử dụng, 50 khối trống kế tiếp sẽ được đọc vào trong super block. Ngược lại, khi một khối được giải phóng, địa chỉ của nó sẽ được thêm vào danh sách của super block. Khi đã đủ 50 địa chỉ trong super block, khối trống kế tiếp sẽ được dùng để lưu trữ 50 địa chỉ khối trống đang đặt trong super block thay cho super block.

## Bài tập

### Bài 1 :

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 ?

### Bài 2 :

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)

### Bài 3 :

Chép 1 tập tin 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 .

### Bài 4 :

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 tập tin (hoặc thư mục). Một cluster = 2 sector. Đĩa sử dụng Fat 12. Hỏi muốn truy xuất cluster 10 thì phải đọc những sector nào ?

### Bài 5 :

Hiện trạng của FAT12 và RDET (mỗi entry chỉ gồm tên tập tin 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 xóa tập tin vd.txt và chép vào tập tin bt.cpp có kích thước 1025 bytes ( giả sử 1 cluster = 1 sector)

### Bài 6 :

Một tập tin 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.

Về I\_node của tập tin này, giả sử mỗi khối chỉ chứa được 3 phần tử.

## BÀI 11 HỆ THỐNG QUẢN LÝ NHẬP/XUẤT

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

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

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

Bài học này đòi hỏi những kiến thức về : kiến trúc máy tính, cơ chế ngắt trên máy tính.

### I.KHÁI NIỆM VỀ HỆ THỐNG QUẢN LÝ NHẬP/XUẤT

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

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

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

## II. PHẦN CỨNG NHẬP/XUẤT

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

### II.1 Thiết bị I/O

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

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

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

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

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

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

Thiết bị truyền thông : như modem...

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

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

### II.2 Tổ chức của chức năng I/O

Có ba cách để thực hiện I/O :

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



Các bước tiến hóa của chức năng I/O :

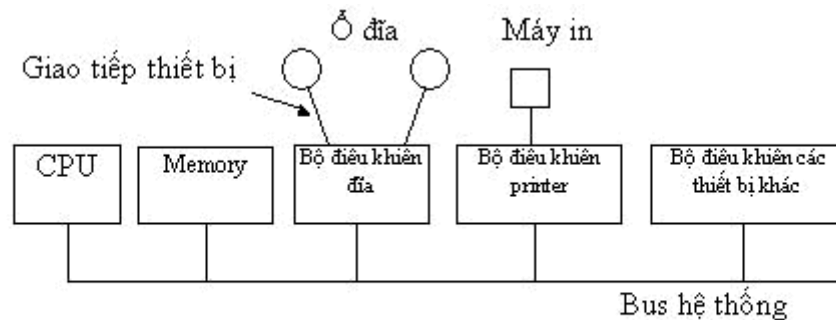
- Bộ xử lý kiểm soát trực tiếp các thiết bị ngoại vi.
- Hệ thống có thêm bộ điều khiển thiết bị. Bộ xử lý sử dụng cách thực hiện nhập xuất thứ nhất. Theo cách này bộ xử lý được tách rời khỏi các mô tả chi tiết của các thiết bị ngoại vi.
- Bộ xử lý sử dụng thêm cơ chế ngắt.
- Sử dụng cơ chế DMA, bộ xử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

### II.3 Bộ điều khiển thiết bị

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

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

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



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

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

Công việc của bộ điều khiển là chuyển đổi dãy các bit tuần tự trong một khối các byte và thực hiện sửa chữa nếu cần thiết. Thông thường khối các byte được tổ chức thành từng bit và đặt trong buffer của bộ điều khiển. Sau khi thực hiện checksum nội dung của buffer sẽ được chuyển vào bộ nhớ chính. Ví dụ : bộ điều khiển cho màn hình đọc các byte của ký tự để hiển thị trong bộ nhớ và tổ chức các tín hiệu để điều khiển các tia của CRT để xuất trên màn ảnh bằng cách quét các tia dọc và ngang. Nếu không có bộ điều khiển, lập trình viên hệ điều hành phải tạo thêm chương trình điều khiển tín hiệu analog cho đèn hình. Với bộ điều khiển , hệ điều hành chỉ cần khởi động chúng với một số tham số như số ký tự trên một dòng, số dòng trên màn hình và bộ điều khiển sẽ thực hiện điều khiển các tia.

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

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

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

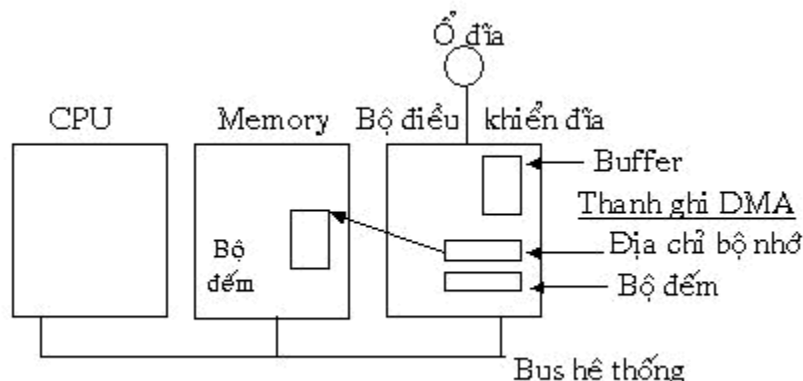
#### II.4 DMA (Direct Memory Access)

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

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

Sau khi bộ điều khiển đã đọc toàn bộ dữ liệu từ thiết bị vào buffer của nó và kiểm tra checksum. Bộ

điều khiển chuyển byte đầu tiên vào bộ nhớ chính tại địa chỉ được mô tả bởi địa chỉ bộ nhớ DMA. Sau đó nó tăng địa chỉ DMA và giảm số bytes phải chuyển. Quá trình này lặp cho tới khi số bytes phải chuyển bằng 0, và bộ điều khiển tạo một ngắt. Như vậy không cần phải copy khối vào trong bộ nhớ, nó đã hiện hữu trong bộ nhớ.



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

### III. PHẦN MỀM NHẬP/XUẤT

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

#### III.1 Kiểm soát ngắt

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

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

#### III.2 Điều khiển thiết bị (device drivers)

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

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

Chức năng của device drivers là nhận những yêu cầu trừu tượng từ phần mềm nhập/xuất độc lập thiết bị ở lớp trên, và giám sát yêu cầu này thực hiện. Nếu driver đang rảnh, nó sẽ thực hiện ngay yêu cầu, ngược lại, yêu cầu đó sẽ được đưa vào hàng đợi.

Ví dụ, bước đầu tiên của yêu cầu nhập/xuất đĩa là chuyển từ trừu tượng thành cụ thể. Driver của đĩa phải biết khối nào cần đọc, kiểm tra sự hoạt động của motor đĩa, xác định vị trí của đầu đọc đã đúng chưa v.v...

Nghĩa là device drivers phải xác định được những thao tác nào của bộ điều khiển phải thi hành và theo trình tự nào. Một khi đã xác định được chỉ thị cho bộ điều khiển, nó bắt đầu thực hiện bằng cách chuyển lệnh vào thanh ghi của bộ điều khiển thiết bị. Bộ điều khiển có thể nhận một hay nhiều chỉ thị liên tiếp và sau đó tự nó thực hiện không cần sự trợ giúp của hệ điều hành. Trong khi lệnh thực hiện. Có hai trường hợp xảy ra : Một là device drivers phải chờ cho tới khi bộ điều khiển thực hiện xong bằng cách tự khóa lại cho tới khi một ngắt phát sinh mở khóa cho nó. Hai là, hệ điều hành chấm dứt mà không chờ, vì vậy driver không cần thiết phải khóa.

Sau khi hệ điều hành hoàn tất việc kiểm tra lỗi và nếu mọi thứ đều ổn driver sẽ chuyển dữ liệu cho phần mềm độc lập thiết bị. Cuối cùng nó sẽ trả về thông tin về trạng thái hay lỗi cho nơi gọi và nếu có một yêu cầu khác ở hàng đợi, nó sẽ thực hiện tiếp, nếu không nó sẽ khóa lại chờ đến yêu cầu tiếp theo.

### III.3 Phần mềm nhập/xuất độc lập thiết bị

Mặc dù một số phần mềm nhập/xuất mô tả thiết bị nhưng phần lớn chúng là độc lập với thiết bị. Ranh giới chính xác giữa drivers và phần mềm độc lập thiết bị là độc lập về mặt hệ thống, bởi vì một số hàm mà được thi hành theo kiểu độc lập thiết bị có thể được thi hành trên drivers vì lý do hiệu quả hay những lý do khác nào đó.

|  |
|--|
| Giao tiếp đồng nhất cho device drivers   |
| Đặt tên thiết bị                         |
| Bảo vệ thiết bị                          |
| Cung cấp khối độc lập thiết bị           |
| Tổ chức buffer                           |
| Định vị lưu trữ trên thiết bị khối       |
| Cấp phát và giải phóng thiết bị tận hiến |
| Báo lỗi                                  |

Chức năng cơ bản của phần mềm nhập/xuất độc lập thiết bị là những chức năng chung cho tất cả các thiết bị và cung cấp một giao tiếp đồng nhất cho phần mềm phạm vi người sử dụng.

Trước tiên nó phải có chức năng tạo một ánh xạ giữa thiết bị và một tên hình thức. Ví dụ đối với UNIX, tên /dev/tty0 dành riêng để mô tả I-node cho một file đặc biệt, và I-node này chứa chứa số thiết bị chính, được dùng để xác định driver thích hợp và số thiết bị phụ, được dùng để xác định các tham số cho driver

để cho biết là đọc hay ghi.

Thứ hai là bảo vệ thiết bị, là cho phép hay không cho phép người sử dụng truy xuất thiết bị. Các hệ điều hành có thể có hay không có chức năng này.

Thứ ba là cung cấp khối dữ liệu độc lập thiết bị vì ví dụ những đĩa khác nhau sẽ có kích thước sector khác nhau và điều này sẽ gây khó khăn cho các phần mềm người sử dụng ở lớp trên. Chức năng này cung cấp các khối dữ liệu logic độc lập với kích thước sector vật lý.

Thứ tư là cung cấp buffer để hỗ trợ cho đồng bộ hóa quá trình hoạt động của hệ thống. Ví dụ buffer cho bàn phím.

Thứ năm là định vị lưu trữ trên các thiết bị khối.

Thứ sáu là cấp phát và giải phóng các thiết bị tận hiến.

Cuối cùng là thông báo lỗi cho lớp bên trên từ các lỗi do device driver báo về.

### III.4 Phần mềm nhập/xuất phạm vi người sử dụng

Hầu hết các phần mềm nhập/xuất đều ở bên trong của hệ điều hành và một phần nhỏ của chúng chứa các thư viện liên kết với chương trình của người sử dụng ngay cả những chương trình thi hành bên ngoài hạt nhân.

Lời gọi hệ thống, bao gồm lời gọi hệ thống nhập/xuất thường được thực hiện bởi các hàm thư viện. Ví dụ khi trong chương trình C có lệnh

```
count = write(fd, buffer, nbytes) ;
```

Hàm thư viện write được dịch và liên kết dưới dạng nhị phân và nằm trong bộ nhớ khi thi hành. Tập hợp tất cả những hàm thư viện này rõ ràng là một phần của hệ thống nhập/xuất.

Không phải tất cả các phần mềm nhập/xuất đều chứa hàm thư viện, có một loại quan trọng khác gọi là hệ thống spooling dùng để khai thác tối đa thiết bị nhập/xuất trong hệ thống đa chương.

Các hàm thư viện chuyển các tham số thích hợp cho lời gọi hệ thống và hàm thư viện thực hiện việc định dạng cho nhập và xuất như lệnh printf trong C. Thư viện nhập/xuất chuẩn chứa một số hàm có chức năng nhập/xuất và tất cả chạy như chương trình người dùng.

Chức năng của spooling là tránh trường hợp một tiến trình đang truy xuất thiết bị, chiếm giữ thiết bị nhưng sau đó không làm gì cả trong một khoảng thời gian và như vậy các tiến trình khác bị ảnh hưởng vì không thể truy xuất thiết bị đó. Một ví dụ của spooling device là line printer. Spooling còn được sử dụng trong hệ thống mạng như hệ thống e-mail chẳng hạn.


## IV. TÓM TẮT

I/O là một phần quan trọng và không thể thiếu được của hệ điều hành, nhưng thường không được quan tâm đúng mức. Trong bài này, chúng ta đã khảo sát phần cứng I/O, mối liên hệ giữa thiết bị I/O với bộ kiểm soát thiết bị. Sau đó chúng ta khảo sát bốn cấp của phần mềm I/O : thủ tục ngắt, điều khiển thiết bị, phần mềm độc lập thiết bị và những thư viện I/O trong môi trường người dùng. Điều khiển thiết bị quản lý tất cả những mô tả chi tiết của một hoặc nhiều thiết bị. Phần mềm độc lập thiết bị xây dựng các vùng đệm và định vị, đây là những việc chung cho nhiều thiết bị.

### Câu hỏi kiểm tra kiến thức

1. Tổ chức của hệ thống quản lý I/O như thế nào ?
2. Có bao nhiêu loại thiết bị I/O ? Với mỗi loại cho ví dụ cụ thể.
3. Vai trò của bộ điều khiển thiết bị là gì ?
4. Cơ chế hoạt động DMA như thế nào ?
5. Có thể thực hiện I/O mà không sử dụng điều khiển thiết bị không? Tại sao?
6. Nêu vai trò của phần mềm độc lập thiết bị.


### Câu hỏi trắc nghiệm

 1. Hệ thống quản lý I/O của hệ điều hành là :

- |   |             |
|---|-------------|
| a. Một cơ chế trên mọi thiết bị tin học | c. Bàn phím |
| b. Một phần của hệ điều hành            | d. Máy in   |

 2. Hệ thống quản lý I/O của một hệ điều hành bao gồm :

- |                                  |                     |
|----------------------------------|---------------------|
| a. Phần cứng I/O và phần mềm I/O | c. Kiểm soát ngắt   |
| b. DMA                           | d. a, b, c đều sai. |

 3. DMA là viết tắt của :

- |                              |                            |
|------------------------------|----------------------------|
| a. Direct Management Access  | c. Direct Memory Access    |
| b. Direct Memory Application | d. Direct Mainboard Access |

 4. Phần mềm độc lập thiết bị :

- |                                    |   |
|------------------------------------|---|
| a. Do Microsoft viết ra            | c. Là một phần của hệ thống quản lý I/O |
| b. Là một phần của trình biên dịch | d. a, b, c đều sai.                     |

## BÀI 12 GIỚI THIỆU MỘT SỐ HỆ THỐNG I/O

*Cơ chế quản lý nhập/xuất(I/O) của hệ điều hành được minh họa cụ thể qua việc điều khiển các thiết bị I/O cụ thể. Trong bài này chúng ta tìm hiểu một số hệ thống I/O sau:*

- Hệ thống nhập xuất đĩa
- Hệ thống nhập xuất chuẩn
- Cài đặt đồng hồ

Qua bài học này, chúng ta hiểu được cơ chế quản lý nhập/xuất của hệ điều hành được thể hiện cụ thể trên một số thiết bị I/O. Chúng ta cũng nắm được cơ chế tương tác giữa hệ điều hành với các thiết bị đó và trên hết chúng ta thấy được vai trò của độc lập thiết bị.

Bài học này đòi hỏi những kiến thức về : kiến trúc máy tính, hệ thống quản lý I/O của hệ điều hành.

### I. HỆ THỐNG I/O ĐĨA

Hầu như tất cả các máy tính đều có đĩa để lưu trữ thông tin. Đĩa có ba ưu điểm chính hơn sử dụng bộ nhớ chính để lưu trữ :

- Dung lượng lưu trữ lớn hơn rất nhiều.
- Giá trên một bit rẻ hơn.
- Thông tin không bị mất đi khi không còn cung cấp điện.

#### I.1 Phân cứng đĩa

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

Một đặc điểm thiết bị cài đặt quan trọng cho driver của đĩa là khả năng của bộ điều khiển thực hiện tìm kiếm trên hai hay nhiều driver cùng lúc gọi là tìm kiếm chồng. Trong khi bộ điều khiển và phần mềm đợi việc tìm kiếm hoàn tất trên một đĩa, bộ điều khiển có thể khởi động việc tìm kiếm trên đĩa khác. Các bộ điều khiển không thể cùng lúc đọc hoặc ghi trên hai driver vì khả năng này có thể làm giảm thời gian truy xuất trung bình.

#### I.2 Các thuật toán đọc đĩa

Tất cả mọi công việc đều phụ thuộc vào việc nạp chương trình và nhập xuất tập tin, do đó điều quan trọng là dịch vụ đĩa phải càng nhanh càng tốt. Hệ điều hành có thể tổ chức dịch vụ truy xuất đĩa tốt hơn bằng cách lập lịch yêu cầu truy xuất đĩa.

Tốc độ đĩa bao gồm ba phần. Để truy xuất các khối trên đĩa, trước tiên phải di chuyển đầu đọc đến track hay cylinder thích hợp, thao tác này gọi là seek và thời gian để hoàn tất gọi là *seek time*. Một khi đã đến đúng track, còn phải chờ cho đến khi khối cần thiết đến dưới đầu đọc. Thời gian chờ này gọi là *latency time*. Cuối cùng là vận chuyển dữ liệu giữa đĩa và bộ nhớ chính gọi là *transfer time*. Tổng thời gian cho dịch vụ đĩa chính là tổng của ba khoảng thời gian trên. Trong đó *seek time* và *latency time* là mất nhiều thời gian nhất, do đó để giảm thiểu thời gian truy xuất hệ điều hành đưa ra các thuật toán lập lịch truy xuất.

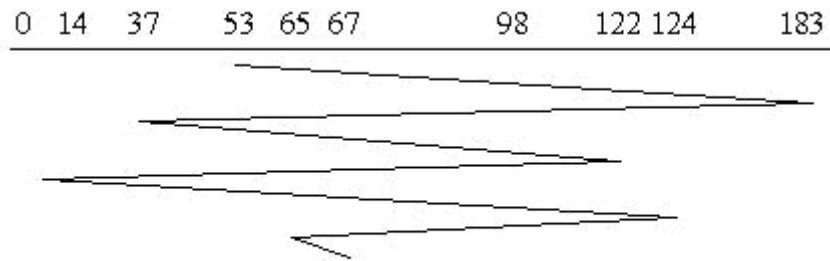
#### \*Lập lịch FCFS :

Phương pháp lập lịch đơn giản nhất là FCFS(first-come,first-served). Thuật toán này rất dễ lập trình

nhưng không cung cấp được một dịch vụ tốt. Ví dụ : cần phải đọc các khối theo thứ tự như sau :

98, 183, 37, 122, 14, 124, 65, và 67

Giả sử hiện tại đầu đọc đang ở vị trí 53. Như vậy đầu đọc lần lượt đi qua các khối 53, 98, 183, 37, 122, 14, 124, 65, và 67 như hình sau :



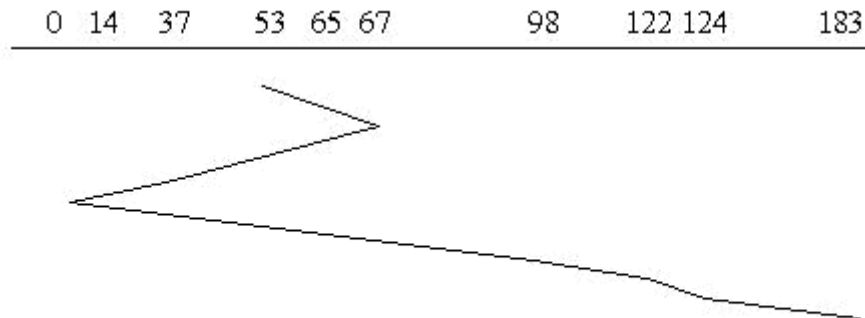
**Hình 12.1** Phương pháp FCFS

#### **\*Lập lịch SSTF (shortest-seek-time-first)**

Thuật toán này sẽ di chuyển đầu đọc đến các khối cần thiết theo vị trí lần lượt gần với vị trí hiện hành của đầu đọc nhất. Ví dụ : cần đọc các khối như sau :

98, 183, 37, 122, 14, 124, 65, và 67

Giả sử hiện tại đầu đọc đang ở vị trí 53. Như vậy đầu đọc lần lượt đi qua các khối 53, 65, 67, 37, 14, 98, 122, 124 và 183 như hình sau :



**Hình 12.2** Phương pháp SSTF

Với ví dụ này, thuật toán SSTF làm giảm số khối mà đầu đọc phải di chuyển là 208 khối.

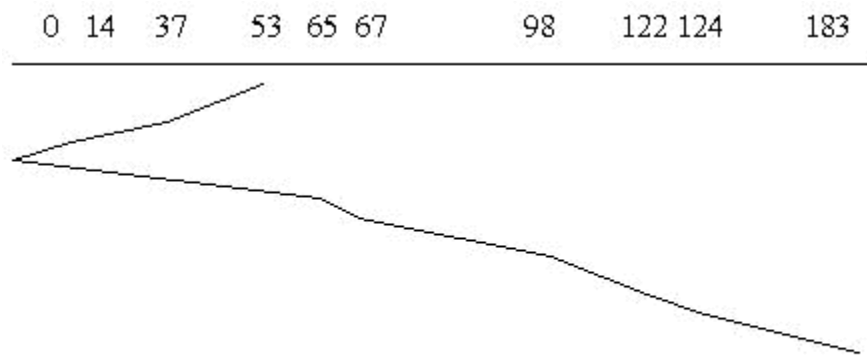
#### **\*Lập lịch SCAN**

Theo thuật toán này, đầu đọc sẽ di chuyển về một phía của đĩa và từ đó di chuyển qua phía kia. Ví dụ : cần đọc các khối như sau :

98, 183, 37, 122, 14, 124, 65, và 67

Giả sử hiện tại đầu đọc đang ở vị trí 53. Như vậy đầu đọc lần lượt đi qua các khối 53, 37, 14, 0, 65, 67, 98, 122, 124 và 183 như hình sau :



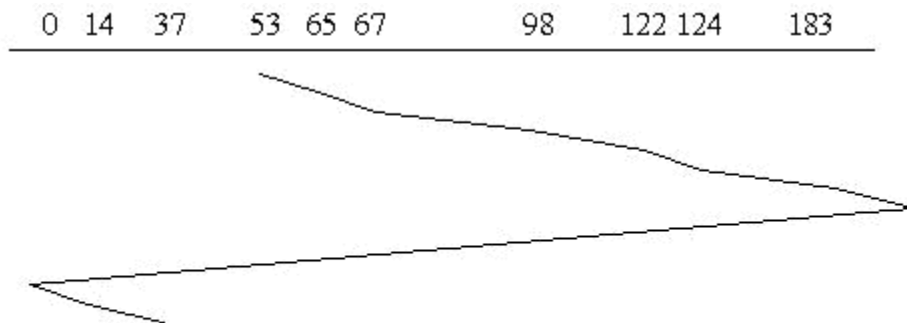


**Hình 12.3** Phương pháp SCAN

Thuật toán này còn được gọi là thuật toán thang máy. Hình ảnh thuật toán giống như hình ảnh của một người quét tuyết, hay quét lá.

**\*Lập lịch C-SCAN**

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



**Hình 12.4** Phương pháp C-SCAN

**\*Lập lịch LOOK:**

Nhận xét rằng cả hai thuật toán lập lịch SCAN và C-SCAN luôn luôn chuyển đầu đọc của đĩa từ đầu này sang đầu kia. Nhưng thông thường thì đầu đọc chỉ chuyển đến khối xa nhất ở mỗi hướng chứ không đến cuối. Do đó SCAN và C-SCAN được chỉnh theo thực tế và gọi là lập lịch LOOK. Như hình sau :



**Hình 12.5** Phương pháp LOOK

### \*Lựa chọn thuật toán lập lịch :

Với những thuật toán lập lịch, vấn đề là phải lựa chọn thuật toán nào cho hệ thống. Thuật toán SSTF thì rất thông thường. Thuật toán SCAN và C-SCAN thích hợp cho những hệ thống phải truy xuất dữ liệu khối lượng lớn. Với bất kỳ thuật toán lập lịch nào, điều quan trọng là khối lượng về số và kiểu khối cần truy xuất. Ví dụ , nếu số khối cần truy xuất là liên tục thì FCFS là thuật toán tốt.

### I.3 Quản lý lỗi

Đĩa là đối tượng mà khi truy xuất có thể gây nhiều lỗi. Một trong số các lỗi thường gặp là :

- *Lỗi lập trình* : yêu cầu đọc các sector không tồn tại.

Lỗi lập trình xảy ra khi yêu cầu bộ điều khiển tìm kiếm cylinder không tồn tại, đọc sector không tồn tại, dùng đầu đọc không tồn tại, hoặc vận chuyển vào và ra bộ nhớ không tồn tại. Hầu hết các bộ điều khiển kiểm tra các tham số và sẽ báo lỗi nếu không thích hợp.

- *Lỗi checksum tạm thời* : gây ra bởi bụi trên đầu đọc.

Bụi tồn tại giữa đầu đọc và bề mặt đĩa sẽ gây ra lỗi đọc. Nếu lỗi tồn tại, khối có thể bị đánh dấu hỏng bởi phần mềm.

- *Lỗi checksum thường trực* : đĩa bị hư vật lý trên các khối.

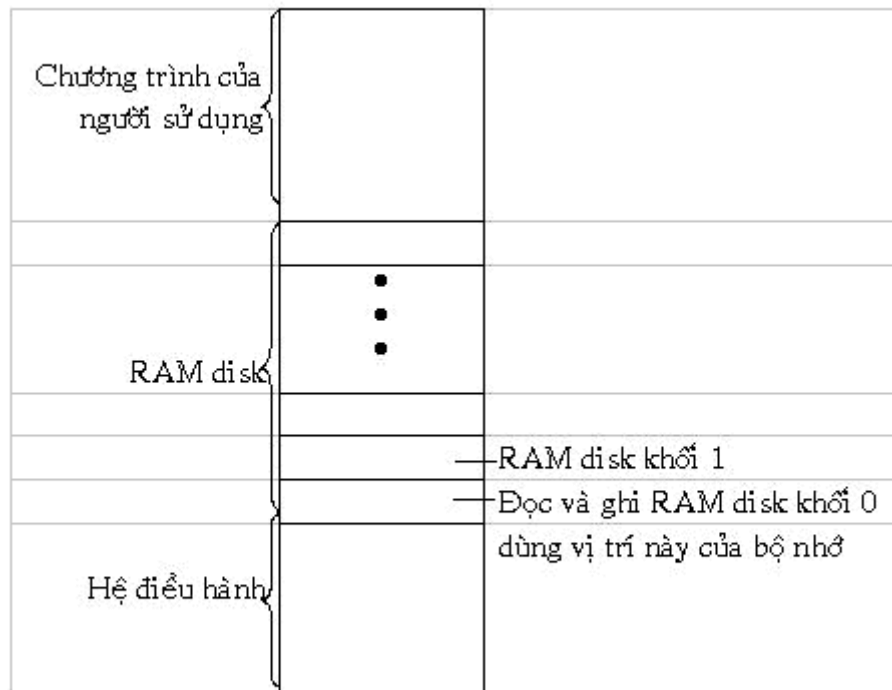
- *Lỗi tìm kiếm* : ví dụ đầu đọc đến cylinder 7 trong khi đó phải đọc 6.

- *Lỗi điều khiển* : bộ điều khiển từ chối thi hành lệnh.

### I.4 RAM Disks

Ý tưởng RAM disk khá đơn giản. Thiết bị khối là phân lưu trữ trung gian với hai lệnh : đọc một khối và ghi một khối. Thông thường những khối này được lưu trữ trên đĩa mềm hoặc đĩa cứng. RAM disk dùng một phần đã định vị trước của bộ nhớ chính để lưu trữ các khối. RAM disk có ưu điểm là cho phép truy xuất nhanh chóng (không phải chờ quay hay tìm kiếm). Như vậy nó thích hợp cho việc lưu trữ những chương trình hay dữ liệu được truy xuất thường xuyên.

## Bộ nhớ chính (RAM)



**Hình 12.6** RAM disks

Hình trên mô tả ý tưởng của RAM disk. Một RAM disk được chia làm nhiều khối, số lượng tùy thuộc vào dung lượng của vùng nhớ. Mỗi khối có cùng kích thước và vừa đúng bằng kích thước của khối thực sự trên đĩa. Khi driver nhận được chỉ thị là đọc hoặc ghi một khối, nó sẽ tìm trong bộ nhớ RAM disk vị trí của khối, và thực hiện việc đọc hay ghi trong đó thay vì từ đĩa mềm hay đĩa cứng.

### I.5 Interleave

Bộ điều khiển đọc ghi đĩa phải thực hiện hai chức năng là đọc/ghi dữ liệu và chuyển dữ liệu vào hệ thống. Để thực hiện được đồng bộ hai chức năng này, bộ điều khiển đọc đĩa cung cấp chức năng interleave. Trên đĩa các sector số hiệu liên tiếp nhau không nằm kế bên nhau mà có một khoảng cách nhất định, khoảng cách này được xác định bởi quá trình format đĩa. Ví dụ : giả sử hệ thống chỉ có 17 sector, và interleave được chọn là 4 thì các sector được bố trí theo thứ tự như sau :

1, 14, 10, 6, 2, 15, 11, 7, 3, 16, 12, 8, 4, 17, 13, 9, 5

Cách đọc lần lượt như sau :

Lần 1:

**1**, 14, 10, 6, **2**, 15, 11, 7, **3**, 16, 12, 8, **4**, 17, 13, 9, **5**

Lần 2:

1, 14, 10, **6**, 2, 15, 11, **7**, 3, 16, 12, **8**, 4, 17, 13, **9**, 5

Lần 3:

1, 14, **10**, 6, 2, 15, **11**, 7, 3, 16, **12**, 8, 4, 17, **13**, 9, 5

Lần 4:

1, **14**, 10, 6, 2, **15**, 11, 7, 3, **16**, 12, 8, 4, **17**, 13, 9, 5

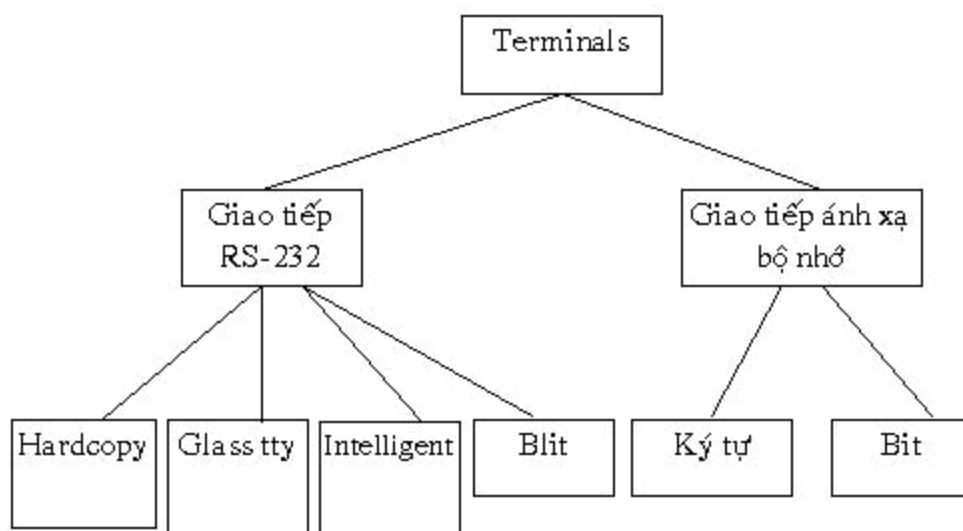
Như vậy sau bốn lần thứ tự các sector đọc được vẫn là từ 1 đến 17

## II. HỆ THỐNG I/O CHUẨN (TERMINALS)

Mọi máy tính đều liên lạc với một hay nhiều terminals. Terminals có rất nhiều dạng khác nhau. Bộ điều khiển terminals ấn dấu mọi sự khác biệt, vì vậy phần độc lập thiết bị của hệ điều hành và chương trình người sử dụng không cần thiết phải viết lại cho mỗi loại terminal.

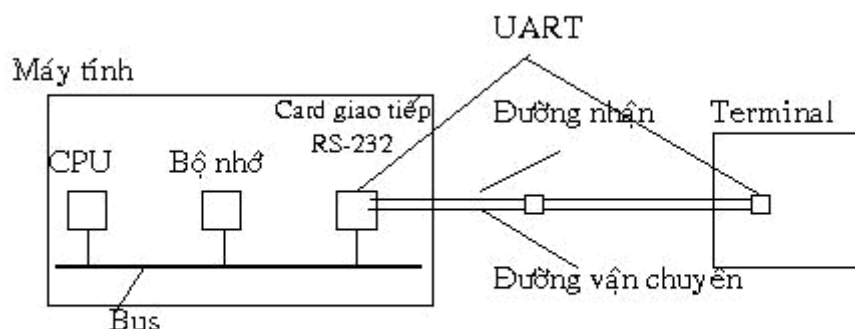
### II.1 Phần cứng terminal

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



Hình 12.7 Các loại terminals

Terminal RS-232 là những thiết bị bao gồm như bàn phím và màn hình. Đây là thiết bị giao tiếp tuần tự, mỗi lần một bit. Những terminals này dùng connector 25-pin, một pin dùng để chuyển dữ liệu, một pin dùng để nhận dữ liệu, một pin là nền, 22 pin còn lại có những chức năng khác nhau, hầu hết thường không dùng đến. Để gởi một ký tự cho terminal RS-232, máy tính mỗi lần chuyển một bit, ngoài ra có một bit bắt đầu, và sau đó có 1 hoặc 2 bit kết thúc để giới hạn một ký tự. Thường thường tốc độ vận chuyển là 1200, 2400, 4800, 9600...bps. Vì cả máy tính và terminal đều làm việc với ký tự mà phải liên lạc với nhau bằng bit nên hệ thống phải thiết kế bộ chuyển đổi gọi là UART. Bộ phận này được gắn vào các card giao tiếp của RS-232.

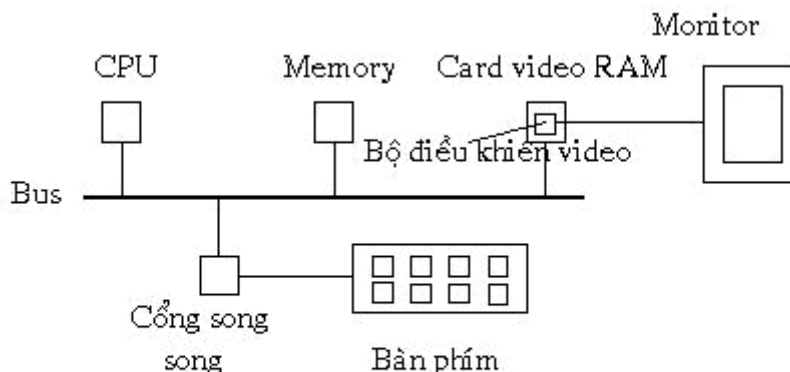


**Hình 12.8** Terminal RS-232

Để in một ký tự, bộ điều khiển terminal ghi một ký tự lên card giao tiếp, sau đó sẽ chuyển cho UART. Terminal RS-232 được chia làm nhiều loại. Dạng đơn giản nhất là terminal hardcopy(printing). Ví dụ các ký tự được nhập vào từ bàn phím và chuyển cho máy tính. Các ký tự từ máy tính xuất ra máy in. Dạng tương tự như vậy nhưng ký tự được xuất trên màn hình gọi là "glass ttys" do đó nó cũng có chức năng tương tự như trên. Terminals intelligent dùng trong máy tính nhỏ. Điểm khác biệt với loại trên dưới quan điểm hệ điều hành là nó sẽ gửi ký tự ASCII ESC sau những ký tự khác nhau dùng để chuyển cursor đến vị trí bất kỳ trên màn hình, chèn một dòng vào giữa màn hình. Blit là một terminal có bộ xử lý mạnh và một màn hình có 1024x800 điểm giao tiếp với máy tính bằng RS-232.

## **II.2 Terminal ánh xạ bộ nhớ**

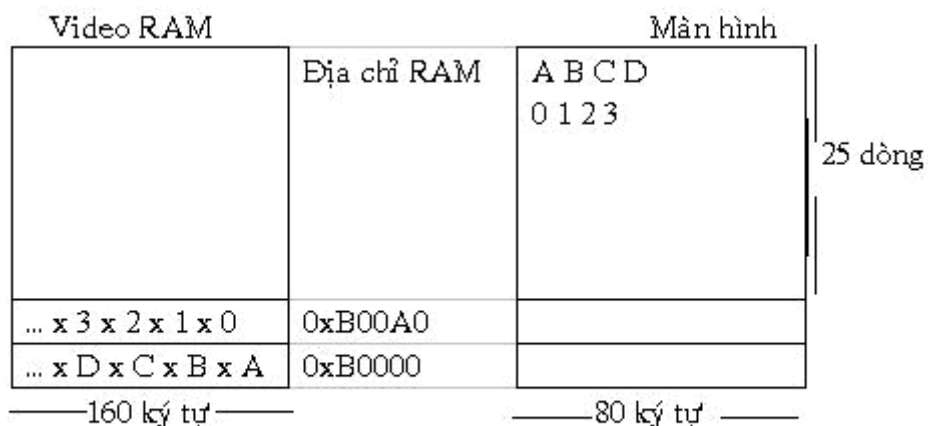
Dạng thứ hai của terminal là terminal ánh xạ bộ nhớ. Loại này không giao tiếp với máy tính qua đường serial. Nó là một phần của hệ thống máy tính. Terminal ánh xạ bộ nhớ giao tiếp bằng một bộ nhớ đặc biệt gọi là video RAM, là một phần của bộ nhớ chính được định vị bởi CPU.



**Hình 12.9** Terminal ánh xạ bộ nhớ

Trên card video RAM có một chip gọi là bộ điều khiển video. Chip này sẽ lấy thông tin từ video RAM và tạo ra tín hiệu video để điều khiển màn hình. Màn hình tạo những tia điện tử quét từ trên xuống dưới. Thường thường có khoảng từ 200 đến 1200 dòng, trên mỗi dòng có từ 200 đến 1200 điểm. Mỗi điểm được gọi là pixel. Bộ điều khiển tín hiệu sẽ xác định mỗi điểm là sáng hay tối. Màn hình màu sẽ có ba tia là đỏ, lục và xanh.

Một ví dụ về màn hình ảnh xạ ký tự trên máy IBM PC. Một phần bộ nhớ chính bắt đầu từ địa chỉ 0xB000 cho màn hình đơn sắc và 0xB800 cho màn hình màu. Mỗi ký tự trên màn hình chiếm hai bytes trong bộ nhớ. Byte thấp chứa giá trị ASCII của ký tự, byte cao chứa thuộc tính như màu sắc, nhấp nháy v.v... Màn hình 80x25 sẽ chiếm 4000 bytes bộ nhớ video RAM



### Hình 12.10 Ảnh xa màn hình

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

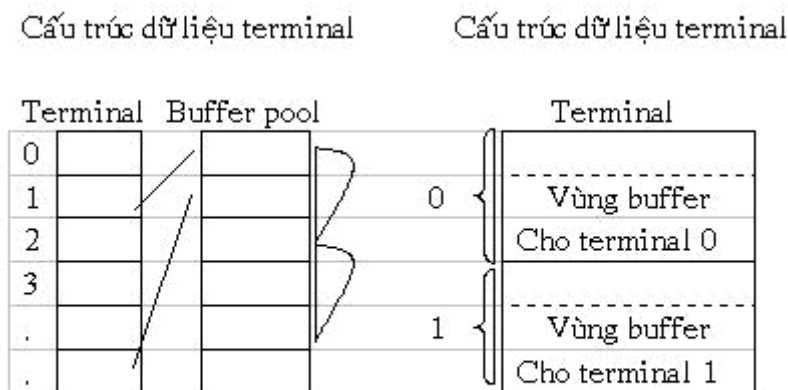
Terminal bit-map tương tự như vậy, ngoại trừ là mọi bit trong video RAM kiểm soát mỗi điểm trên màn hình. Màn hình có 1024x800 pixel cần dùng 100 K bộ nhớ nhưng khó thiết kế font và kích thước cho ký tự. Bàn phím giao tiếp thông qua cổng song song và giao tiếp RS-232. Mỗi khi gõ phím vào, CPU bị ngắt, bộ điều khiển bàn phím xác định kiểu ký tự được đọc từ cổng I/O. Đôi khi bàn phím chỉ cung cấp số hiệu phím, không phải mã ASCII. Trên IBM PC khi gõ phím A mã ký tự 30 được đưa vào thanh ghi I/O. Bộ điều khiển xác định ký tự là chữ hoa hay chữ thường hay là tổ hợp phím.

## II.3 Phần mềm nhập

Bàn phím và màn hình hầu như độc lập với thiết bị. Công việc cơ bản của bộ điều khiển bàn phím là tập hợp các dữ liệu nhập từ bàn phím và chuyển cho chương trình của người sử dụng. Khi có một phím được gõ, nó sẽ gây một ngắt, và bộ điều khiển yêu cầu ký tự trong suốt quá trình ngắt này. Nếu ngắt được gây ra bởi một lời gọi ngắt của một ngôn ngữ lập trình cấp thấp nó sẽ chuyển ký tự này cho chương trình đó. Nó sử dụng một buffer trong bộ nhớ chính và một thông điệp để báo cho bộ điều khiển biết đã có ký tự nhập. Một khi bộ điều khiển nhận một ký tự, nó sẽ bắt đầu xử lý. Nếu dưới dạng mã bàn phím, nó sẽ ánh xạ lại mã ASCII thật. Nếu terminal ở dạng cook, ký tự phải được lưu trữ cho tới khi nhận được hết dòng vì người sử dụng có thể xóa một phần nội dung của nó.

Có hai loại buffer thông thường. Dạng thứ nhất, bộ điều khiển chứa pool chính của buffer, mỗi buffer chứa 16 ký tự. Có một cấu trúc dữ liệu liên kết với nó, trong đó có chứa một con trỏ trỏ tới chuỗi trong

buffer. Khi ký tự chuyển cho chương trình, nó sẽ được loại khỏi buffer. Dạng thứ hai là buffer trực tiếp có cấu trúc dữ liệu vì nếu tổ chức theo dạng thứ nhất sẽ không đủ bộ nhớ. Hình sau cho biết sự khác biệt giữa hai cách như hình sau:



**Hình 12.11** Hai dạng cấu trúc dữ liệu terminal

Mặt dù màn hình và bàn phím là hai thiết bị logic riêng biệt, nhưng mọi người đều quen với việc gõ ký tự và xem nó xuất hiện trên màn hình. Một số terminal cho phép tự động hiển thị lên màn hình những gì vừa gõ hoặc chỉ là những dấu . khi gõ password. Một số terminal không hiển thị ký tự được gõ do đó phải dựa vào phần mềm để hiển thị input, xử lý này gọi là echoing.

Echoing phức tạp vì chương trình phải xuất lên màn hình khi người dùng gõ vào. Bộ điều khiển bàn phím phải kiểm soát không cho ghi chồng lên output của chương trình. Echoing cũng gặp khó khăn khi người nhập gõ nhiều hơn 80 ký tự trên màn hình 80 ký tự một dòng. Một vấn đề khác là xử lý tab. Bộ điều khiển phải tính toán vị trí hiện thời cursor sau đó tính toán để chuyển cho chương trình và cho echoing và tính toán bao nhiêu khoảng trống phải hiển thị. Vấn đề tiếp theo là phải xử lý carriage return và line feed để chuyển cursor qua đầu dòng mới. Việc xử lý này tùy thuộc vào các hệ điều hành khác nhau. Ngoài ra phải kiểm soát tổ hợp ký tự và những ký tự xóa, lùi, hay các phím chức năng.

## **II.4 Phần mềm xuất**

Phần mềm xuất thì đơn giản hơn nhập nhưng ở hai dạng thiết bị terminal RS-232 và ánh xạ bộ nhớ là khác nhau. Phương pháp thông thường của terminal RS-232 là có một buffer xuất cho mỗi loại terminal. Dạng buffer có thể là pool như buffer nhập hay là dạng tận hiến như input. Khi chương trình ghi lên terminal, trước tiên nó xuất lên buffer. Sau khi đã xuất lên buffer, ký tự đầu tiên được xuất, sau đó bộ điều khiển tạm dừng, khi có một ngắt phát sinh, ký tự tiếp theo sẽ được xuất, và cứ tiếp tục như vậy.

Với terminal ánh xạ bộ nhớ, vấn đề đơn giản hơn. Những ký tự được in được xuất một lần từ chương trình người dùng được xuất lên video RAM. Với một số ký tự sẽ được xử lý đặc biệt. Ví dụ : backspace, carriage return, line feed, và bell (CTRL-G). Bộ điều khiển ánh xạ bộ nhớ, lưu giữ trong phần mềm vị trí của video RAM, vì vậy những ký tự in được được xuất trên đó theo thứ tự, các ký tự đặc biệt cũng được cập nhật thích hợp.

Khi một line feed được xuất tại cuối dòng của màn hình, màn hình sẽ cuộn. Thường thường phần cứng cung cấp một số giúp đỡ ở đây. Hầu hết những bộ điều khiển màn hình chứa một thanh ghi xác định vị trí của video RAM để bắt đầu đặt các byte vào dòng đầu tiên của màn hình. Phần mềm soạn thảo màn hình phải có nhiều xử lý phức tạp hơn là chỉ xuống dòng. Để tương thích, một số bộ điều khiển terminal hỗ trợ một số xử lý, thông thường là :

- Di chuyển cursor lên, xuống, trái, phải của một vị trí.
- Di chuyển cursor đến vị trí x,y.
- Chèn một ký tự hay chèn một dòng.
- Xóa một ký tự hay một dòng.
- Cuộn màn hình lên hoặc xuống n dòng.
- Xóa màn hình từ vị trí cursor đến cuối dòng hoặc màn hình.
- Tạo tương phản, gạch dưới, nhấp nháy, hay mode thường.
- Tạo, hủy, di chuyển quản trị các cửa sổ.

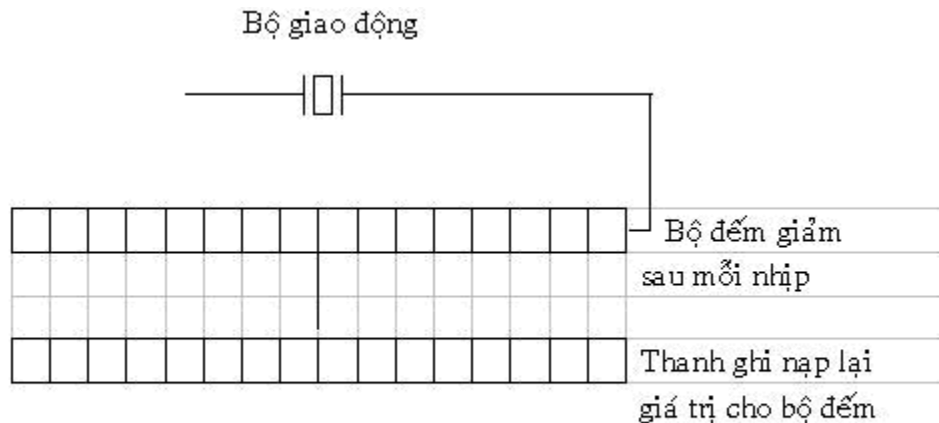
### III. CÀI ĐẶT ĐỒNG HỒ

Đồng hồ còn được gọi là timer, là bộ phận rất cần thiết cho các thao tác của những hệ thống chia sẻ vì nhiều nguyên nhân khác nhau. Nó kiểm soát thời gian trong ngày và không cho phép một tiến trình nào đó độc chiếm CPU trong khi tồn tại những tiến trình khác. Phần mềm đồng hồ có thể xem như là device driver mặc dù đồng hồ không phải là thiết bị khối như đĩa hay thiết bị tuần tự như bàn phím, màn hình.

#### III.1 Phần cứng đồng hồ

Trong máy tính thường sử dụng hai loại đồng hồ nhưng cả hai đều khác với đồng hồ người sử dụng thông thường. Dạng đơn giản sử dụng đồng hồ với điện thế 110v hay 220v, và tạo ra ngắt theo mỗi chu kỳ của hiệu điện thế, từ 50 đến 60 MHz.

Một dạng khác của đồng hồ được xây dựng dựa trên ba thành phần : bộ dao động bằng thạch anh, bộ đếm và bộ thanh ghi lưu trữ như hình vẽ. Dưới tác dụng của dòng điện, tinh thể thạch anh tạo ra dao động. Nhịp dao động rất chính xác theo thời gian, thường thường vào khoảng từ 5 đến 100 MHz tùy theo mỗi loại thạch anh. Tín hiệu này sẽ chuyển cho bộ đếm và bộ đếm sẽ thực hiện việc đếm lùi về 0. Khi bộ đếm có giá trị là 0, nó sẽ gây ra một ngắt CPU. Điều gì xảy ra tiếp theo là do hệ điều hành.



**Hình 12.12** Cấu trúc của đồng hồ

Dạng đồng hồ có thể lập trình có vài dạng thao tác. Thứ nhất là one-shot, khi đồng hồ khởi động, nó sẽ copy giá trị trong thanh ghi lưu trữ vào bộ đếm và sau đó giảm bộ đếm sau mỗi nhịp của thạch anh. Khi bộ đếm đến giá trị 0, nó sẽ gây ra một ngắt và dừng lại cho đến khi phần mềm khởi động lại nó. Thứ hai là square-wave, khi đến giá trị 0, nó sẽ gây ra một ngắt, bộ thanh ghi lưu trữ tự động nạp lại giá trị vào bộ đếm, và tiến trình sẽ được lập lại. Những ngắt phát sinh định kỳ này gọi là clock tick.



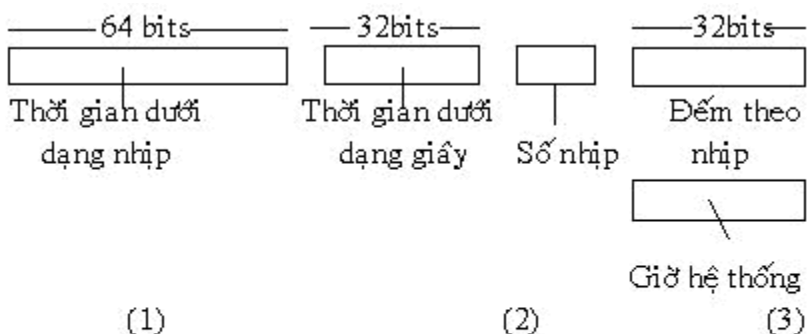
Ưu điểm của đồng hồ có thể lập trình là ngắt định kỳ được điều khiển bởi phần mềm. Nếu sử dụng tín hiệu thạch anh có tần số 1 MHz, bộ đếm sẽ có nhịp là mỗi micro giây. Với thanh ghi 16 bit, ngắt có thể được lập trình để xảy ra trong khoảng từ 1 đến 65535 msec.

### III.2 Phần mềm đồng hồ

Tất cả mọi việc mà phần cứng đồng hồ thực hiện tạo ra các ngắt theo từng khoảng thời gian đều đặn. Mọi điều khác đều được thực hiện bởi phần mềm đồng hồ, là driver đồng hồ. Công việc của driver đồng hồ trên mỗi hệ điều hành là khác nhau, nhưng thường bao gồm những chức năng chính như sau :

- Quản lý thời gian trong ngày.
- Không cho phép tiến trình chạy lâu hơn thời gian mà nó được phép.
- Kế toán việc sử dụng CPU.
- Cung cấp watchdog timer cho một phần của chính hệ thống đó.

☀ Chức năng đầu tiên của đồng hồ, quản lý thời gian trong ngày thì không khó. Chỉ cần tăng một bộ đếm sau mỗi nhịp của đồng hồ như đề cập ở trên. Vấn đề lưu ý ở đây là số lượng bit cho bộ counter. Với đồng hồ ở tần số 60 MHz, một bộ đếm 32 bit sẽ bị tràn sau hai năm. Do đó hệ thống không thể lưu trữ thời gian thực sự dưới dạng số nhịp từ 01/01/1970. Có ba cách giải quyết. Thứ nhất, dùng bộ đếm 64 bit, giải pháp này tốn kém. Thứ hai, lưu trữ dưới dạng giây thay vì nhịp vì  $2^{32}$  giây sẽ là 136 năm. Thứ ba, đếm theo nhịp, nhưng liên hệ với thời gian của hệ thống khi khởi động.



**Hình 12.13** Tổ chức lưu trữ của đồng hồ

☀ Chức năng thứ hai là không cho phép một tiến trình thực hiện quá lâu. Khi nào một tiến trình bắt đầu, bộ lập lịch sẽ khởi gán giá trị cho bộ đếm, mỗi ngắt đồng hồ sẽ giảm giá trị của bộ đếm, khi nào giá trị bằng 0, bộ điều khiển đồng hồ sẽ yêu cầu bộ lập lịch thiết lập giá trị cho một tiến trình khác.

☀ Chức năng thứ ba là kế toán việc sử dụng CPU. Cách thức chính xác nhất là sử dụng một bộ timer thứ hai, khác với timer hệ thống. Bộ timer thứ hai khởi động khi tiến trình bắt đầu và khi tiến trình kết thúc, timer này sẽ cho biết thời gian tiến trình đã thực hiện.

☀ Phần lớn hệ thống cần thiết thiết lập timer. Gọi là watchdog timer. Ví dụ, để sử dụng đĩa mềm, hệ thống phải khởi động motor và chờ khoảng 500msec đạt được tốc độ. Vì vậy, ý tưởng tốt là phải sử dụng watchdog timer để chờ cho thao tác I/O tiếp theo, vào khoảng 3 giây, không tắt motor.

### 📌 Câu hỏi kiểm tra kiến thức

1. So sánh các thuật toán đọc đĩa.

2. Lựa chọn các thuật toán đọc đĩa như thế nào ?
3. Nguyên nhân các lỗi khi truy xuất đĩa và cách khắc phục ?
4. RAM disks là gì ?
5. Vì sao có cơ chế Interleave ?
6. Đặc điểm của phần cứng terminal.
7. Terminal ánh xạ bộ nhớ dùng để làm gì ?
8. Vai trò của đồng hồ.

## Bài tập

Giả sử đĩa có 2 side, mỗi side có 1024 track, mỗi track có 32 sector. Tốc độ xoay của đĩa là 6000 vòng/phút. Thời gian di chuyển giữa các track là 100ms. Giả sử thời gian đọc và chuyển dữ liệu là không đáng kể. Cho biết để truy xuất tất cả sector logic sau phải tốn bao lâu :

34, 16, 120, 14, 86, 200, 79, 300, 8, 500, 170, 450, 1000, 380, 800

Biết :

$$\text{Sector} = \text{Seclog} / \text{SecTrk} + 1$$

$$\text{Side} = (\text{Seclog} / \text{SecTrk}) / \text{SideNo}$$

$$\text{Track} = (\text{Seclog} / (\text{SecTrk} * \text{SideNo}))$$

Với Seclog là sector logic, SideNo là số side, SecTrk là số sector trên 1 track

## BÀI 13 BẢO VỆ VÀ AN TOÀN HỆ THỐNG

*An toàn và bảo vệ hệ thống là chức năng không thể thiếu của các hệ điều hành hiện đại. Trong bài học này, chúng ta sẽ làm quen với các khái niệm về tổ chức an toàn hệ thống, cũng như các cơ chế bảo vệ hỗ trợ việc triển khai các chiến lược này.*

### I. MỤC TIÊU BẢO VỆ HỆ THỐNG (PROTECTION)

Mục tiêu của việc bảo vệ hệ thống là:

☞ **Bảo vệ chống lỗi của tiến trình** : khi có nhiều tiến trình cùng hoạt động, lỗi của một tiến trình j phải được ngăn chặn không cho lan truyền trên hệ thống làm ảnh hưởng đến các tiến trình khác. Đặc biệt, qua việc phát hiện các lỗi tiềm ẩn trong các thành phần của hệ thống có thể tăng cường độ tin cậy hệ thống ( reliability ).

☞ **Chống sự truy xuất bất hợp lệ** : Bảo đảm các bộ phận tiến trình sử dụng tài nguyên theo một cách thức hợp lệ được qui định cho nó trong việc khai thác các tài nguyên này .

Vai trò của bộ phận bảo vệ trong hệ thống là cung cấp một *cơ chế* để áp dụng các *chiến lược* quản trị việc sử dụng tài nguyên . Cần phân biệt khái niệm cơ chế và chiến lược:

☞ **Cơ chế** : xác định làm thế nào để thực hiện việc bảo vệ, có thể có các cơ chế phần mềm hoặc cơ chế phần cứng.

☞ **Chiến lược**: quyết định việc bảo vệ được áp dụng như thế nào : những đối tượng nào trong hệ thống cần được bảo vệ, và các thao tác thích hợp trên các đối tượng này

Để hệ thống có tính tương thích cao, cần phân tách các cơ chế và chiến lược được sử dụng trong hệ thống. Các chiến lược sử dụng tài nguyên là khác nhau tùy theo ứng dụng, và thường dễ thay đổi . Thông thường các chiến lược được lập trình viên vận dụng vào ứng dụng của mình để chống lỗi truy xuất bất hợp lệ đến các tài nguyên, trong khi đó hệ thống cung cấp các cơ chế giúp người sử dụng có thể thực hiện được chiến lược bảo vệ của mình.

### II. MIỀN BẢO VỆ (DOMAIN OF PROTECTION)

#### II.1. Khái niệm

Một hệ thống máy tính được xem như một tập các đối tượng (*objects*). Một đối tượng có thể là một bộ phận phần cứng ( CPU, bộ nhớ, ổ đĩa...) hay một thực thể phần mềm ( tập tin, chương trình, semaphore...). Mỗi đối tượng có một định danh duy nhất để phân biệt với các đối tượng khác trong hệ thống, và chỉ được truy xuất đến thông qua các thao tác được định nghĩa chặt chẽ và được qui định ngữ nghĩa rõ ràng. Các thao tác có thể thực hiện được trên một đối tượng được xác định cụ thể tùy vào đối tượng.

Để có thể kiểm soát được tình hình sử dụng tài nguyên trong hệ thống, hệ điều hành chỉ cho phép các tiến trình được truy xuất đến các tài nguyên mà nó có quyền sử dụng, hơn nữa tiến trình chỉ được truy xuất đến các tài nguyên cần thiết trong thời điểm hiện tại để nó hoàn thành tác vụ (nguyên lý **need-to-know**) nhằm hạn chế các lỗi truy xuất mà tiến trình có thể gây ra trong hệ thống.

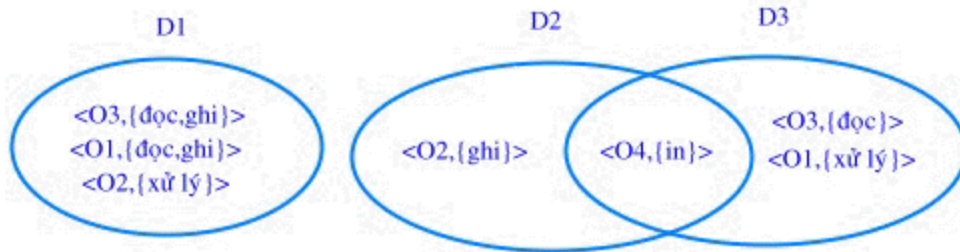
Mỗi tiến trình trong hệ thống đều hoạt động trong một miền bảo vệ (*protection domain*) nào đó. Một miền bảo vệ sẽ xác định các tài nguyên ( đối tượng) mà những tiến trình hoạt động trong miền bảo vệ này có thể sử dụng, và các thao tác hợp lệ các tiến trình này có thể thực hiện trên những tài nguyên đó.

Ví dụ : <File F, {read, write}>

## II.2. Cấu trúc của miền bảo vệ

Các khả năng thao tác trên một đối tượng được gọi là quyền truy xuất (*access right*). Một *miền bảo vệ* là một tập các quyền truy xuất, mỗi quyền truy xuất được định nghĩa bởi một bộ hai thứ tự  $\langle \text{đối tượng}, \{\text{quyền thao tác}\} \rangle$ .

Các miền bảo vệ khác nhau có thể giao nhau một số quyền truy xuất :



Hình vẽ 5.1 Hệ thống với 3 miền bảo vệ

Mối liên kết giữa một tiến trình và một miền bảo vệ có thể tĩnh hay động :

- **Liên kết tĩnh** : trong suốt thời gian sống của tiến trình, tiến trình chỉ hoạt động trong một miền bảo vệ . Trong trường hợp tiến trình trải qua các giai đoạn xử lý khác nhau, ở mỗi giai đoạn tiến trình có thể thao tác trên những tập tài nguyên khác nhau bằng các thao tác khác nhau. Tuy nhiên, nếu sử dụng liên kết tĩnh, rõ ràng là ngay từ đầu miền bảo vệ đã phải đặc tả tất cả các quyền truy xuất qua các giai đoạn cho tiến trình , điều này có thể khiến cho tiến trình có dư quyền trong một giai đoạn nào đó, và vi phạm nguyên lý need-to-know. Để có thể tôn trọng nguyên lý này, khi đó cần phải có khả năng cập nhật nội dung miền bảo vệ để có thể phản ánh các quyền tối thiểu của tiến trình trong miền bảo vệ tại một thời điểm!
- **Liên kết động** : cơ chế này cho phép tiến trình chuyển từ miền bảo vệ này sang miền bảo vệ khác trong suốt thời gian sống của nó. Để tiếp tục tuân theo nguyên lý **need-to-know**, thay vì sửa đổi nội dung của miền bảo vệ, có thể tạo ra các miền bảo vệ mới với nội dung thay đổi qua từng giai đoạn xử lý của tiến trình, và chuyển tiến trình sang hoạt động trong miền bảo vệ phù hợp theo từng thời điểm.

Một miền bảo vệ có thể được xây dựng cho:

- **Một người sử dụng** : trong trường hợp này, tập các đối tượng được phép truy xuất phụ thuộc vào định danh của người sử dụng, miền bảo vệ được chuyển khi thay đổi người sử dụng.
- **Một tiến trình** : trong trường hợp này, tập các đối tượng được phép truy xuất phụ thuộc vào định danh của tiến trình, miền bảo vệ được chuyển khi quyền điều khiển được chuyển sang tiến trình khác.
- **Một thủ tục** : trong trường hợp này, tập các đối tượng được phép truy xuất là các biến cục bộ được định nghĩa bên trong thủ tục, miền bảo vệ được chuyển khi thủ tục được gọi.

## III. MA TRẬN QUYỀN TRUY XUẤT ( ACCESS MATRIX)


Một cách trừu tượng, có thể biểu diễn mô hình bảo vệ trên đây như một ma trận quyền truy xuất ( access matrix). Các dòng của ma trận biểu diễn các miền bảo vệ và các cột tương ứng với các đối tượng trong hệ thống. Phần tử  $access[i,j]$  của ma trận xác định các quyền truy xuất mà một tiến trình hoạt động trong miền bảo vệ  $D_i$  có thể thao tác trên đối tượng  $O_j$ .

| <b>object</b><br><b>domain</b> | <b>F<sub>1</sub></b> | <b>F<sub>2</sub></b> | <b>F<sub>3</sub></b> | <b>Máy in</b> |
|--------------------------------|----------------------|----------------------|----------------------|---------------|
| <b>D<sub>1</sub></b>           | đọc                  |                      | đọc                  |               |
| <b>D<sub>2</sub></b>           |                      |                      |                      | in            |
| <b>D<sub>3</sub></b>           |                      | đọc                  | xử lý                |               |
| <b>D<sub>4</sub></b>           | đọc<br>ghi           |                      | đọc<br>ghi           |               |

**Hình 5.2** Ma trận quyền truy xuất

Cơ chế bảo vệ được cung cấp khi ma trận quyền truy xuất được cài đặt ( với đầy đủ các thuộc tính ngữ nghĩa đã mô tả trên lý thuyết), lúc này người sử dụng có thể áp dụng các chiến lược bảo vệ bằng cách đặc tả nội dung các phần tử tương ứng trong ma trận \_ xác định các quyền truy xuất ứng với từng miền bảo vệ , và cuối cùng, hệ điều hành sẽ quyết định cho phép tiến trình hoạt động trong miền bảo vệ thích hợp.


Ma trận quyền truy xuất cũng cung cấp một cơ chế thích hợp để định nghĩa và thực hiện một sự kiểm soát nghiêm ngặt cho cả phương thức liên kết tĩnh và động các tiến trình với các miền bảo vệ :

 Có thể kiểm soát việc chuyển đổi giữa các miền bảo vệ nếu quan niệm miền bảo vệ cũng là một đối tượng trong hệ thống, và bổ sung các cột mô tả cho nó trong ma trận quyền truy xuất.

Khi đó tiến trình được phép chuyển từ miền bảo vệ **Di** sang miền bảo vệ **Dj** nếu phần tử **access(i,j)** chứa đựng quyền « chuyển » ( switch).

| <b>object</b><br><b>domain</b> | <b>F<sub>1</sub></b> | <b>F<sub>2</sub></b> | <b>F<sub>3</sub></b> | <b>Máy in</b> | <b>D<sub>1</sub></b> | <b>D<sub>2</sub></b> | <b>D<sub>3</sub></b> | <b>D<sub>4</sub></b> |
|--------------------------------|----------------------|----------------------|----------------------|---------------|----------------------|----------------------|----------------------|----------------------|
| <b>D<sub>1</sub></b>           | đọc                  |                      | đọc                  |               |                      | chuyển               |                      |                      |
| <b>D<sub>2</sub></b>           |                      |                      |                      | in            |                      |                      | chuyển               | chuyển               |
| <b>D<sub>3</sub></b>           |                      | đọc                  | xử lý                |               |                      |                      |                      |                      |
| <b>D<sub>4</sub></b>           | đọc<br>ghi           |                      | đọc<br>ghi           |               | chuyển               |                      |                      |                      |

**Hình 5.3** Ma trận quyền truy xuất với domain là một đối tượng

 Có thể kiểm soát việc sửa đổi nội dung ma trận (thay đổi các quyền truy xuất trong một miền bảo vệ) nếu quan niệm bản thân ma trận cũng là một đối tượng.

Các thao tác sửa đổi nội dung ma trận được phép thực hiện bao gồm : sao chép quyền ( copy), chuyển

quyền ( transfer), quyền sở hữu (owner), và quyền kiểm soát (control)

- **Copy**: nếu một quyền truy xuất  $R$  trong  $access[i,j]$  được đánh dấu là  $R^*$  thì có thể sao chép nó sang một phần tử  $access[k,j]$  khác ( mở rộng quyền truy xuất  $R$  trên cùng đối tượng  $O_j$  nhưng trong miền bảo vệ  $D_k$  ).
- **Transfer** : nếu một quyền truy xuất  $R$  trong  $access[i,j]$  được đánh dấu là  $R^+$  thì có thể chuyển nó sang một phần tử  $access[k,j]$  khác ( chuyển quyền truy xuất  $R^+$  trên đối tượng  $O_j$  sang miền bảo vệ  $D_k$  ).
- **Owner** : nếu  $access[i,j]$  chứa quyền truy xuất *owner* thì tiến trình hoạt động trong miền bảo vệ  $Di$  có thể thêm hoặc xóa các quyền truy xuất trong bất kỳ phần tử nào trên cột  $j$  (có quyền thêm hay bớt các quyền truy xuất trên đối tượng  $O_j$  trong những miền bảo vệ khác).
- **Control** : nếu  $access[i,j]$  chứa quyền truy xuất *control* thì tiến trình hoạt động trong miền bảo vệ  $Di$  có thể xóa bất kỳ quyền truy xuất nào trong các phần tử trên dòng  $j$  (có quyền bỏ bớt các quyền truy xuất trong miền bảo vệ  $Dj$ ).

| object domain  | F <sub>1</sub> | F <sub>2</sub> | F <sub>3</sub> |
|----------------|----------------|----------------|----------------|
| D <sub>1</sub> | xử lý          |                | ghi+           |
| D <sub>2</sub> | xử lý          | đọc*           | xử lý          |
| D <sub>3</sub> | xử lý          |                |                |

(a)

| object domain  | F <sub>1</sub> | F <sub>2</sub> | F <sub>3</sub> |
|----------------|----------------|----------------|----------------|
| D <sub>1</sub> | xử lý          |                |                |
| D <sub>2</sub> | xử lý          | đọc*           | xử lý          |
| D <sub>3</sub> | xử lý          | đọc            | ghi+           |

(b)

**Hình 5.4** Ma trận quyền truy xuất với quyền *copy* , *transfer* (a) trước, (b) sau cập nhật

| object domain  | F <sub>1</sub> | F <sub>2</sub> | F <sub>3</sub> |
|----------------|----------------|----------------|----------------|
| D <sub>1</sub> | owner<br>xử lý |                | ghi            |
| D <sub>2</sub> |                | đọc*<br>owner  | đọc*<br>owner  |

|                      |       |  |      |
|----------------------|-------|--|------|
|                      |       |  | ghi* |
| <b>D<sub>3</sub></b> | xử lý |  |      |

(a)

| <b>object domain</b> | <b>F<sub>1</sub></b> | <b>F<sub>2</sub></b>  | <b>F<sub>3</sub></b>  |
|----------------------|----------------------|-----------------------|-----------------------|
| <b>D<sub>1</sub></b> | owner<br>xử lý       |                       |                       |
| <b>D<sub>2</sub></b> |                      | owner<br>đọc*<br>ghi* | đọc*<br>owner<br>ghi* |
| <b>D<sub>3</sub></b> |                      | ghi                   |                       |

(b)

**Hình 5.5** Ma trận quyền truy xuất với quyền *owner* (a) trước, (b) sau cập nhật

| <b>object domain</b> | <b>F<sub>1</sub></b> | <b>F<sub>2</sub></b> | <b>F<sub>3</sub></b> | <b>Máy in</b> | <b>D<sub>1</sub></b> | <b>D<sub>2</sub></b> | <b>D<sub>3</sub></b> | <b>D<sub>4</sub></b> |
|----------------------|----------------------|----------------------|----------------------|---------------|----------------------|----------------------|----------------------|----------------------|
| <b>D<sub>1</sub></b> | đọc                  |                      | đọc                  |               |                      | chuyển               |                      |                      |
| <b>D<sub>2</sub></b> |                      |                      |                      | in            |                      |                      | chuyển               | control<br>chuyển    |
| <b>D<sub>3</sub></b> |                      | đọc                  | xử lý                |               |                      |                      |                      |                      |
| <b>D<sub>4</sub></b> | ghi                  |                      | ghi                  |               | chuyển               |                      |                      |                      |

**Hình 5.6** Ma trận quyền truy xuất đã sửa đổi nội dung so với H5.3 nhờ quyền *control*

## IV. CÀI ĐẶT MA TRẬN QUYỀN TRUY XUẤT

### IV.1. Bảng toàn cục

Cách đơn giản nhất để cài đặt ma trận truy xuất là sử dụng một bảng bao gồm các bộ ba thứ tự  $\langle$  miền bảo vệ, đối tượng, các quyền truy xuất  $\rangle$ . Mỗi khi thực hiện thao tác  $M$  trên đối tượng  $O_j$  trong miền bảo vệ  $D_i$ , cần tìm trong bảng toàn cục một bộ ba  $\langle D_i, O_j, R_k \rangle$  mà  $M \in R_k$ . Nếu tìm thấy, thao tác  $M$  được phép thi hành, nếu không, xảy ra lỗi truy xuất.

### IV.2. Danh sách quyền truy xuất ( Access control list \_ ACL)

Có thể cài đặt mỗi cột trong ma trận quyền truy xuất như một danh sách quyền truy xuất đối với một đối tượng. Mỗi đối tượng trong hệ thống sẽ có một danh sách bao gồm các phần tử là các bộ hai thứ tự  $\langle$ miền bảo vệ, các quyền truy xuất $\rangle$ , danh sách này sẽ xác định các quyền truy xuất được qui định trong từng miền bảo vệ có thể tác động trên đối tượng. Mỗi khi thực hiện thao tác  $M$  trên đối tượng  $O_j$  trong

miền bảo vệ  $Di$ , cần tìm trong danh sách quyền truy xuất của đối tượng  $Oj$  một bộ hai  $\langle Di, Rk \rangle$  mà  $M \in Rk$ . Nếu tìm thấy, thao tác  $M$  được phép thi hành, nếu không, xảy ra lỗi truy xuất.

Ví dụ : Một miền bảo vệ trong hệ thống UNIX được xác định tương ứng với một người sử dụng (uid) trong một nhóm (gid) nào đó. Giả sử có 4 người dùng : A,B,C,D thuộc các nhóm tương ứng là system, staff, student, student. Khi đó các tập tin trong hệ thống có thể có các ACL như sau :

- File0 : ( A,\*,RWX)
- File1 : ( A,system,RWX)
- File2 : ( A,\*,RW-),(B,staff,R--),(D,\*,RW-)
- File3 : ( \*,student,R--)
- File4 : (C,\*,---),(\*,student,R--)

Thực tế, hệ thống tập tin trong UNIX được bảo vệ bằng cách mỗi tập tin được gán tương ứng 9 bit bảo vệ, từng 3 bit sẽ mô tả quyền truy xuất R(đọc), W(ghi) hay X(xử lý) của các tiến trình trên tập tin này theo thứ tự : tiến trình sở hữu các tiến trình cùng nhóm với tiến trình sở hữu, các tiến trình khác. Đây là một dạng ACL nhưng được nén thành 9 bit.

### IV.3. Danh sách tiềm năng của miền bảo vệ (Capability list – C\_List)

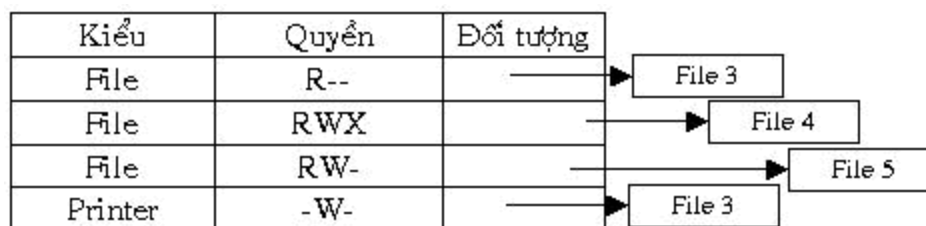
Mỗi dòng trong ma trận quyền truy xuất tương ứng với một miền bảo vệ sẽ được tổ chức thành một danh sách tiềm năng (*capabilities list*) :

- Một danh sách tiềm năng của một miền bảo vệ là một danh sách các đối tượng và các thao tác được quyền thực hiện trên đối tượng khi tiến trình hoạt động trong miền bảo vệ này.
- Một phần tử của C-List được gọi là một tiềm năng (capability) là một hình thức biểu diễn được định nghĩa một cách có cấu trúc cho một đối tượng trong hệ thống và các quyền truy xuất hợp lệ trên đối tượng này.

| kiểu đối tượng | quyền truy xuất | con trỏ đến đối tượng |
|----------------|-----------------|-----------------------|
|----------------|-----------------|-----------------------|

**Hình 5.7** Tiềm năng

Ví dụ :



- Tiến trình chỉ có thể thực hiện thao tác  $M$  trên đối tượng  $Oj$  trong miền bảo vệ  $Di$ , nếu trong  $C\_List$  của  $Di$  có chứa tiềm năng tương ứng của  $Oj$ .
- Danh sách tiềm năng được gán tương ứng với từng miền bảo vệ, thực chất nó cũng là một đối tượng được bảo vệ bởi hệ thống, và tiến trình của người sử dụng chỉ có thể truy xuất đến nó một cách gián tiếp để tránh làm sai lạc  $C\_List$ .
- Hệ điều hành cung cấp các thủ tục cho phép tạo lập, hủy bỏ và sửa đổi các tiềm năng của một đối tượng, và chỉ các tiến trình đóng vai trò server (thường là tiến trình hệ điều hành) mới có thể



sửa đổi nội dung C\_List.

#### IV.4. Cơ chế khóa và chìa

Đây là cách tiếp cận kết hợp giữa danh sách quyền truy xuất và danh sách khả năng. Mỗi đối tượng sở hữu một danh sách các mã nhị phân, được gọi là « khóa » (lock). Cũng như thế, mỗi miền bảo vệ sẽ sở hữu một danh sách mã nhị phân gọi là « chìa » (key). Một tiến trình hoạt động trong một miền bảo vệ chỉ có thể truy xuất đến một đối tượng nếu miền bảo vệ sở hữu một chìa tương ứng với một khóa trong danh sách của đối tượng.

Cũng như C\_List, danh sách « khóa » và « chìa » được hệ điều hành quản lý, người sử dụng không thể truy xuất trực tiếp đến chúng để thay đổi nội dung.

#### IV.5. Thu hồi quyền truy xuất

Trong một hệ thống bảo vệ động, đôi khi hệ điều hành cần thu hồi một số quyền truy xuất trên các đối tượng được chia sẻ giữa nhiều người sử dụng. Khi đó đặt ra một số vấn đề như sau :

- *Thu hồi tức khắc hay trì hoãn, trì hoãn đến khi nào ?*
- *Nếu loại bỏ một quyền truy xuất trên một đối tượng, thu hồi quyền này trên tất cả hay chỉ một số người sử dụng?*
- *Thu hồi một số quyền hay toàn bộ quyền trên một đối tượng ?*
- *Thu hồi tạm thời hay vĩnh viễn một quyền truy xuất ?*

Đối với các hệ thống sử dụng danh sách quyền truy xuất, việc thu hồi có thể thực hiện dễ dàng : tìm và hủy trên ACL quyền truy xuất cần thu hồi, như vậy việc thu hồi được thực hiện tức thời, có thể áp dụng cho tất cả hay một nhóm người dùng, thu hồi toàn bộ hay một phần, và thu hồi vĩnh viễn hay tạm thời đều được.

Tuy nhiên trong các hệ sử dụng C\_List, vấn đề thu hồi gặp khó khăn vì các tiềm năng được phân tán trên khắp các miền bảo vệ trong hệ thống, do vậy cần tìm ra chúng trước khi loại bỏ. Có thể giải quyết vấn đề này theo nhiều phương pháp :

- *Tái yêu cầu (Reacquisition):* loại bỏ các tiềm năng ra khỏi mỗi miền bảo vệ sau từng chu kỳ, nếu miền bảo vệ vẫn còn cần tiềm năng nào, nó sẽ tái yêu cầu tiềm năng đó lại.
- *Sử dụng các con trỏ đến tiềm năng (Back-pointers) :* với mỗi đối tượng, lưu trữ các con trỏ đến những tiềm năng tương ứng trên đối tượng này. Khi cần thu hồi quyền truy xuất nào trên đối tượng, lần theo các con trỏ để cập nhật tiềm năng tương ứng.
- *Sử dụng con trỏ gián tiếp (Indirection) :* các tiềm năng không trực tiếp trỏ đến các đối tượng, mà trỏ đến một bảng toàn cục do hệ điều hành quản lý. Khi cần thu hồi quyền, sẽ xóa phần tử tương ứng trong bảng này.
- *Khóa ( Key) :* nếu sử dụng cơ chế khóa và chìa, khi cần thu hồi quyền, chỉ cần thay đổi khóa và bắt buộc tiến trình hay người dùng yêu cầu chìa mới.

## V. AN TOÀN HỆ THỐNG (SECURITY)

Bảo vệ hệ thống (*protection*) là một cơ chế kiểm soát việc sử dụng tài nguyên của các tiến trình hay người sử dụng để đối phó với các tình huống lỗi có thể phát sinh từ trong hệ thống . Trong khi đó khái

niệm an toàn hệ thống (*security*) muốn đề cập đến mức độ tin cậy mà hệ thống duy trì khi phải đối phó không những với các vấn đề nội bộ, mà còn cả với những tác hại đến từ môi trường ngoài .

### V.1. Các vấn đề về an toàn hệ thống

Hệ thống được gọi là an toàn nếu các tài nguyên được sử dụng đúng như quy ước trong mọi hoàn cảnh. Kém may mắn là điều này hiếm khi đạt được trong thực tế ! Thông thường, an toàn bị vi phạm vì các nguyên nhân vô tình hay cố ý phá hoại. Việc chống đỡ các phá hoại cố ý là rất khó khăn và gần như không thể đạt hiệu quả hoàn toàn. Bảo đảm an toàn hệ thống ở cấp cao chống lại các tác hại từ môi trường ngoài như hoả hoạn, mất điện, phá hoại... cần được thực hiện ở 2 mức độ *vật lý* (trang bị các thiết bị an toàn cho vị trí đặt hệ thống...) và *nhân sự* (chọn lọc cẩn thận những nhân viên làm việc trong hệ thống...). Nếu an toàn môi trường được bảo đảm khá tốt, an toàn của hệ thống sẽ được duy trì tốt nhờ các cơ chế của hệ điều hành (với sự trợ giúp của phần cứng).

Lưu ý rằng nếu bảo vệ hệ thống có thể đạt độ tin cậy 100%, thì các cơ chế an toàn hệ thống được cung cấp chỉ với hy vọng ngăn chặn bớt các tình huống bất an hơn là đạt đến độ an toàn tuyệt đối.

### V.2. Kiểm định danh tính (Authentication)

Để đảm bảo an toàn, hệ điều hành cần giải quyết tốt vấn đề chủ yếu là *kiểm định danh tính* (*authentication*). Hoạt động của hệ thống bảo vệ phụ thuộc vào khả năng xác định các tiến trình đang xử lý. Khả năng này, đến lượt nó, lại phụ thuộc vào việc xác định được *người dùng* đang sử dụng hệ thống để có thể kiểm tra người dùng này được cho phép thao tác trên những tài nguyên nào.

Cách tiếp cận phổ biến nhất để giải quyết vấn đề là sử dụng *password* để kiểm định đúng danh tính của người dùng. Mỗi khi người dùng muốn sử dụng tài nguyên, hệ thống sẽ kiểm tra password của người dùng nhập vào với password được lưu trữ, nếu đúng, người dùng mới được cho phép sử dụng tài nguyên. Password có thể được để bảo vệ từng đối tượng trong hệ thống, thậm chí cùng một đối tượng sẽ có các password khác nhau ứng với những quyền truy xuất khác nhau.

Cơ chế password rất dễ hiểu và dễ sử dụng do vậy được sử dụng rộng rãi, tuy nhiên yếu điểm nghiêm trọng của phương pháp này là khả năng bảo mật password rất khó đạt được sự hoàn hảo, những tác nhân tiêu cực có thể đoán ra password của người khác nhờ nhiều cách thức khác nhau.

### V.3. Mối đe dọa từ các chương trình

Trong môi trường mà một chương trình được tạo lập bởi người này lại có thể được người khác sử dụng, có thể xảy ra các tình huống sử dụng không đúng, từ đó dẫn đến những hậu quả khó lường. Hai trường hợp điển hình là :

#### V.3.1. Ngựa thành Troy

Khi một người dùng A cho một chương trình do B viết hoạt động dưới danh nghĩa của mình ( trong miền bảo vệ được gán tương ứng cho người dùng A), chương trình này có thể trở thành một « con ngựa thành Troy » vì khi đó các đoạn lệnh trong chương trình có thể thao tác trên các tài nguyên với những quyền tương ứng của người A (mà có thể người B vốn bị cấm!), nhiều chương trình như thế đã « lợi dụng hoàn cảnh » để gây ra các tác hại đáng tiếc.

#### V.3.2. Cánh cửa nhỏ ( Trap-door)

Một mối đe dọa đặc biệt nguy hiểm và khó chống đỡ đến từ sự vô tình hay ý nghĩ bất chính của các lập trình viên. Khi xây dựng chương trình, các lập trình viên có thể để lại một « cánh cửa nhỏ » trong phần mềm mà chỉ có họ là có khả năng sử dụng , qua đó thâm nhập và phá hoại hệ thống ( ví dụ làm tròn các số lẻ trong những tài khoản, và thu lợi riêng từ phần dư này...). Vấn đề này rất khó đối phó vì cần phải

tiến hành phân tích chương trình nguồn để tìm ra chỗ sơ hở.

#### **V.4. Mối đe dọa từ hệ thống**

Hầu hết các hệ điều hành đều cung cấp phương tiện cho phép các tiến trình khi hoạt động có thể tạo ra (*spawn*) những tiến trình khác. Trong các môi trường như thế, tài nguyên hệ thống và các tập tin của người dùng có thể bị sử dụng sai lạc để gây tác hại. Hai phương pháp phổ biến để phá hoại hệ thống theo phương thức này là :

##### **\*V.4.1. Các chương trình « sâu bọ » ( Worm)**

Một chương trình « sâu bọ » là chương trình lợi dụng cơ chế phát sinh tiến trình của hệ thống để đánh bại chính hệ thống. Tiến trình « sâu bọ » có khả năng tự động phát sinh các phiên bản ngay cả trên môi trường mạng, lan tràn trên nhiều máy tính khác nhau, sau đó chiếm dụng các tài nguyên hệ thống và làm ngừng trệ hoàn toàn hoạt động của các tiến trình khác trên hệ thống mạng .

Ví dụ : chương trình của Robert Tappan Morris, 11/1988 lan truyền trên Internet

##### **\*V.4.2. Các chương trình Virus**

Virus là một dạng phá hoại nguy hiểm khác đối với các hệ thống thông tin. Khác với « sâu bọ » là những chương trình hoàn chỉnh, virus chỉ là những đoạn code có khả năng lây truyền vào các chương trình chính thống khác và từ đó tàn phá hệ thống. Virus thường hoành hành trên các máy đơn, và chủ yếu lây truyền giữa các máy qua việc trao đổi đĩa mềm.

#### **V.5. Giám sát các mối đe dọa**

Nhìn chung việc bảo đảm an toàn hệ thống là rất khó do có các yếu tố con người. Hệ điều hành chỉ có thể áp dụng một số kỹ thuật để giảm bớt khả năng bị phá hoại như ghi nhận các sự kiện như :

- cố gắng nhập nhiều lần password sai
- sử dụng các password dễ đoán
- các tiến trình với định danh nghi ngờ không được ủy quyền
- các tiến trình không được ủy quyền trong những thư mục hệ thống
- các chương trình kéo dài xử lý một cách đáng ngờ
- bảo vệ các tập tin và thư mục không hợp lý
- thay đổi kích thước của các chương trình hệ thống

Việc kiểm tra thường kỳ và ghi nhận các thông tin này giúp hệ thống phát hiện kịp thời các nguy cơ, và cho phép phân tích , dự đoán các cách đối phó về sau.

## **VI. TÓM TẮT**

► Các đối tượng trong hệ thống ( có thể là phần cứng như CPU, bộ nhớ chính ; hay phần mềm như tập tin, chương trình) cần phải được bảo vệ khỏi sự sử dụng sai trái. Điều này có thể thực hiện được bằng cách đặt các đối tượng vào những miền bảo vệ thích hợp với các quyền truy xuất được định nghĩa trên đó.

► Quyền truy xuất trên một đối tượng là thao tác được phép thực hiện trên đối tượng đó.

► Miền bảo vệ là tập các quyền truy xuất trên một số các đối tượng. Mỗi tiến trình sẽ hoạt động trong một miền bảo vệ tại một thời điểm.

▶Ma trận quyền truy xuất là mô hình tổng quát cho bài toán bảo vệ hệ thống. Ma trận này có thể được cài đặt như một mảng, các danh sách quyền truy xuất hoặc các danh sách tiềm năng.

▶Mô hình bảo vệ động cho phép tiến trình có thể chuyển đổi miền bảo vệ trong thời gian xử lý. Có thể tổ chức mô hình bảo vệ động nếu xem các miền bảo vệ và bản thân ma trận quyền truy xuất cũng là các đối tượng trong hệ thống.

▶An toàn hệ thống không chỉ quan tâm đến các nguy cơ nội bộ hệ thống, mà phải đối phó với cả các nguy cơ từ môi trường ngoài.

▶Vấn đề quan trọng nhất trong an toàn hệ thống là kiểm định danh tính của người dùng để xác định đúng các quyền tương ứng của họ. Phương pháp thông dụng nhất để làm điều này là kiểm tra password.

▶Ngoài ra hệ thống còn phải đối phó với các tác hại của những sự tấn công cố ý từ phía những chương trình « sâu bọ », virus, các loại khe hở trong chương trình...


▶Nhìn chung không thể đạt được mức độ an toàn hệ thống 100%, nhưng có thể áp dụng các biện pháp để hạn chế tối đa các nguy cơ phá hoại hệ thống.


## **Củng cố bài học**

Các câu hỏi cần trả lời được sau bài học này :


1. Các khái niệm : quyền truy xuất, miền bảo vệ, ma trận quyền truy xuất
2. Các cơ chế bảo vệ : ACL, Capabilities
3. Các chiến lược an toàn hệ thống : kiểm định danh tính.

## **Bài tập**

 **Bài 1.** Khả năng hạn chế truy xuất dựa trên nguyên lý « need-to-know » có thể giúp chống lại loại chương trình phá hoại nào ?


 **Bài 2.** Xét 4 tiến trình sau, mỗi tiến trình hoạt động trong một miền bảo vệ khác nhau được mô tả trong ma trận quyền truy xuất dưới đây. Trong miền bảo vệ nào tiến trình không thể đọc tập tin F1 ( mà không có sự giúp đỡ của các tiến trình khác) ?

|     | FF1       | DD1        | DD2 | DD3 | DD4       |
|-----|-----------|------------|-----|-----|-----------|
| DD1 | Ownerer - | -          | -   | -   | -         |
| DD2 | Readad* - | -          | -   | -   | -         |
| DD3 | -         | -          | -   | -   | Controlol |
| DD4 | -         | Switchch - | -   | -   | -         |

 **Bài 3.** Giả sử tiến trình A đang hoạt động trong miền bảo vệ D , thao tác nào A được phép thực hiện trong các cơ chế bảo vệ dưới đây :

- a) Sử dụng danh sách quyền truy xuất (access list system), thêm một phần tử vào danh sách quyền truy xuất của một đối tượng sở hữu bởi D
- b) Sử dụng danh sách tiềm năng (capability list system), thêm một phần tử vào danh sách tiềm

năng của D

 **Bài 4.** Giả sử một lớp thực hành UNIX được tổ chức như sau :

- Tất cả sinh viên trong lớp thuộc về một nhóm thực hành của lớp .
- Giáo viên hướng dẫn thực hành không thuộc về nhóm thực hành, nhưng lại sở hữu các tập tin trong thư mục của nhóm. Như vậy, quyền truy xuất của giáo viên hướng dẫn trên các tập tin này là quyền sở hữu, trong khi các sinh viên nhận được quyền truy xuất như là thành viên của nhóm.

Xét chiến lược bảo vệ sau : lớp được gọi là CLASS; bao gồm 2 sinh viên A và B. C không phải là sinh viên thuộc lớp CLASS. Giáo viên hướng dẫn là GVHD. Mỗi sinh viên có một tập tin riêng lưu bài tập gọi là HW, chỉ có sinh viên đó được quyền đọc, ghi. GVHD có một tập tin lưu kết quả đánh giá thực hành là GRADES, chỉ có GVHD đó được quyền đọc, ghi. GVHD có quyền đọc ghi hai tập tin đề cương SYLLABUS , và bài tập thực hành ASSIGNMENT ; tất cả sinh viên của lớp có quyền đọc ASSIGNMENT. Tất cả mọi người đều có quyền đọc SYLLABUS.

- a) Mô tả chiến lược bảo vệ này bằng ma trận quyền truy xuất.
- b) Mô tả chiến lược bảo vệ này bằng tập các danh sách tiềm năng cho mỗi người dùng .
- c) Mô tả chiến lược bảo vệ này bằng tập các danh sách quyền truy xuất cho mỗi tập tin
- d) Mô tả cách thức UNIX cho phép thực hiện chiến lược này.

# BÀI 14 : WINDOWS NT

## Mục tiêu

Giới thiệu cách tiếp cận của hệ điều hành WinNT trong thiết kế và cài đặt

## I. LỊCH SỬ

Vào khoảng cuối những năm 1980, IBM và Microsoft cố gắng thay thế MS-DOS với một hệ điều hành mới là OS/2. Tuy nhiên diễn tiến không được tốt đẹp như mong muốn : OS/2 ra đời chậm trễ, không có tính tương thích cao và không phổ biến. IBM và Microsoft đổ lỗi lẫn nhau để cuối cùng sự hợp tác đi đến đổ vỡ .

1988, Microsoft mời Dave Cutler (người phát triển VMS và VAX ở Digital ) làm thủ lĩnh để phát triển phần của họ trong dự án OS/2 thành một hệ điều hành hoàn toàn mới là Windows New Technology hay tắt là WinNT.

Thoạt trông thì giao diện thân thiện của WinNT giống như họ Windows 9x, nhưng thực sự thì phần lõi bên trong của hệ điều hành được xây dựng hoàn toàn khác biệt. WinNT kết hợp chặt chẽ các tính năng bảo vệ phần cứng và bảo đảm an toàn người dùng, là một hệ điều hành có độ tin cậy cao.

## II. MỤC TIÊU THIẾT KẾ

Microsoft công bố các mục tiêu thiết kế được đề ra cho WinNT bao gồm :

- ✚ 1. **Có khả năng mở rộng (extensibility)** : có thể nâng cấp dễ dàng nhờ vào kiến trúc phân lớp, thiết kế đơn thể
- ✚ 2. **Dễ mang chuyển (portability)** : có thể thi hành trên nhiều cấu hình phần cứng (bao gồm nhiều họ CPU khác nhau)
- ✚ 3. **Có độ tin cậy cao (reliability)** : có khả năng phục hồi lỗi, bảo đảm an toàn và bảo mật
- ✚ 4. **Có tính tương thích mạnh (compatibility)** : có thể thi hành các ứng dụng MS-DOS, 16-bit Win, Win32, OS/2 & POSIX
- ✚ 5. **Hiệu quả thực hiện tốt (performance)** : bảo đảm thực hiện hiệu quả với các cơ chế quản lý nội bộ, IPC, hỗ trợ mạng và hỗ trợ đa xử lý.

## III. CÁC THÀNH PHẦN HỆ THỐNG

- ✚ 1. **HAL ( hardware abstraction layer)** : Tầng cung cấp sự trừu tượng hoá phần cứng, che dấu các khác biệt của phần cứng nhằm đến khả năng dễ mang chuyển của WinNT
- ✚ 2. **Hạt nhân (kernel)** : là thành phần cốt lõi của WinNT. Chịu trách nhiệm điều phối tiến trình, cung cấp các cơ chế đồng bộ hoá, xử lý ngắt và xử lý ngoại lệ. Đây cũng là phần giao tiếp với HAL, chuyển các ngắt phát sinh thành những tập chuẩn được qui định trước. Hạt nhân được thiết kế hướng đối tượng., sử dụng 2 tập đối tượng :

- Các đối tượng điều phối và đồng bộ : events, mutants, mutexes, semaphores, threads và timers
- Các đối tượng kiểm soát : procedure calls, interrupts, power status, processes, profiles

Tầng điều hành (Executive): cung cấp các dịch vụ cơ bản mà tất cả các hệ thống con khác nhau đều có

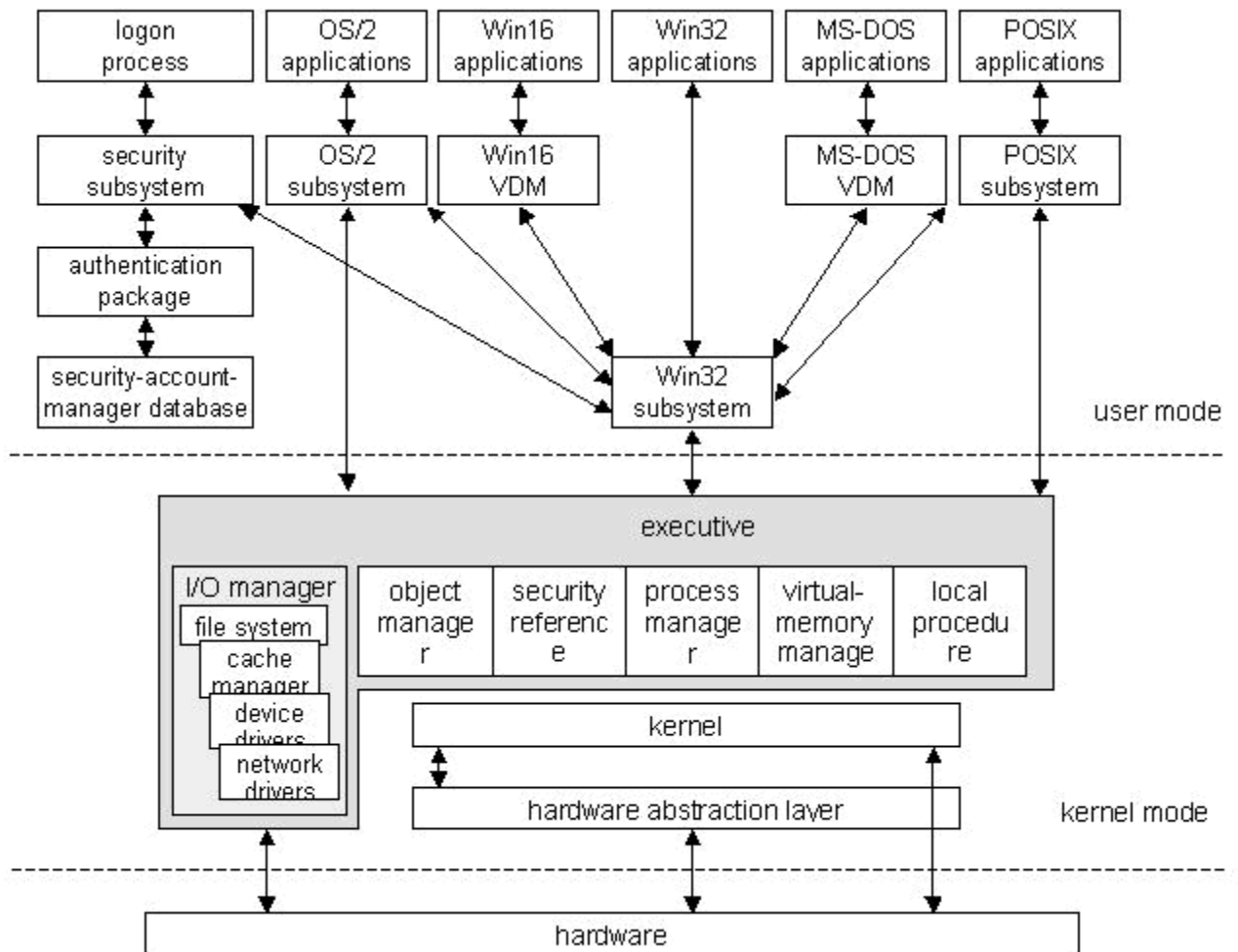
thể sử dụng . Bao gồm :

- **Quản Lý Đối Tượng** : cung cấp các phương thức chuẩn để thao tác đối tượng như *create, open, close, delete, query name, parse, security*
- **Quản Lý Tiến Trình**: cung cấp các phương thức tạo và hủy tiến trình, tiểu trình
- **Quản Lý Bộ Nhớ Ảo Và Quản Lý Nhập Xuất** : tích hợp chặt chẽ các chiến lược phân trang với điều phối đĩa
- **Hỗ Trợ Gọi Hàm Cục Bộ** : tối ưu hoá cơ chế trao đổi thông điệp trên cơ sở chia sẻ bộ nhớ chung
- **Hỗ Trợ An Toàn Và Bảo Mật** : cung cấp các dịch vụ kiểm tra , bảo vệ đối tượng trong thời gian thực thi.

🔗 3. **Các hệ thống con (subsystems)** : gồm những tiến trình hệ thống nhưng hoạt động trong chế độ user, chịu trách nhiệm tạo môi trường thuận lợi cho các ứng dụng thi hành. Đây là các lớp dịch vụ được xây dựng dựa trên nền tảng các dịch vụ gốc nhưng hướng về các ứng dụng vốn được phát triển trên cho các hệ điều hành khác. Mỗi hệ thống con cung cấp một API cho một hệ điều hành nào đó. Win32 là môi trường hoạt động chính, khi phát hiện một ứng dụng khác Win32, tiến trình tương ứng sẽ được chuyển sang hệ thống con thích hợp.

🔗 4. **Các hệ thống bảo mật và an toàn**

## IV. KIẾN TRÚC HỆ ĐIỀU HÀNH WINDOWSNT



## V. CÁC MODULE QUẢN LÝ CỦA WINDOWSNT

### V.1 Quản lý tiến trình

Một tiến trình trong WinNT có :

- Một không gian địa chỉ ảo
- PCB chứa các thông tin như độ ưu tiên của tiến trình và các thống kê điều phối
- Một hay nhiều tiểu trình

Một tiểu trình trong WinNT có :

- Dòng xử lý và các thông tin kèm theo
- Các trạng thái tiểu trình :
  - ready: chờ CPU
  - standby: là tiểu trình kế tiếp sẽ nhận CPU
  - running: đang xử lý
  - waiting: chờ tín hiệu (IO)



- transition: chờ tài nguyên
- terminated: kết thúc

Bộ điều phối sử dụng chiến lược điều phối ,multilevel feedback với 32 mức ưu tiên :

- Các tiểu trình real time (16-31) : độ ưu tiên tĩnh
- Các tiểu trình khác nhận độ ưu tiên động (0-15) được cập nhật tự động khi tiến trình thay đổi trạng thái, hay thay đổi tường minh do người dùng.

## V.2 Quản lý bộ nhớ ảo

Giả sử phần cứng hỗ trợ :

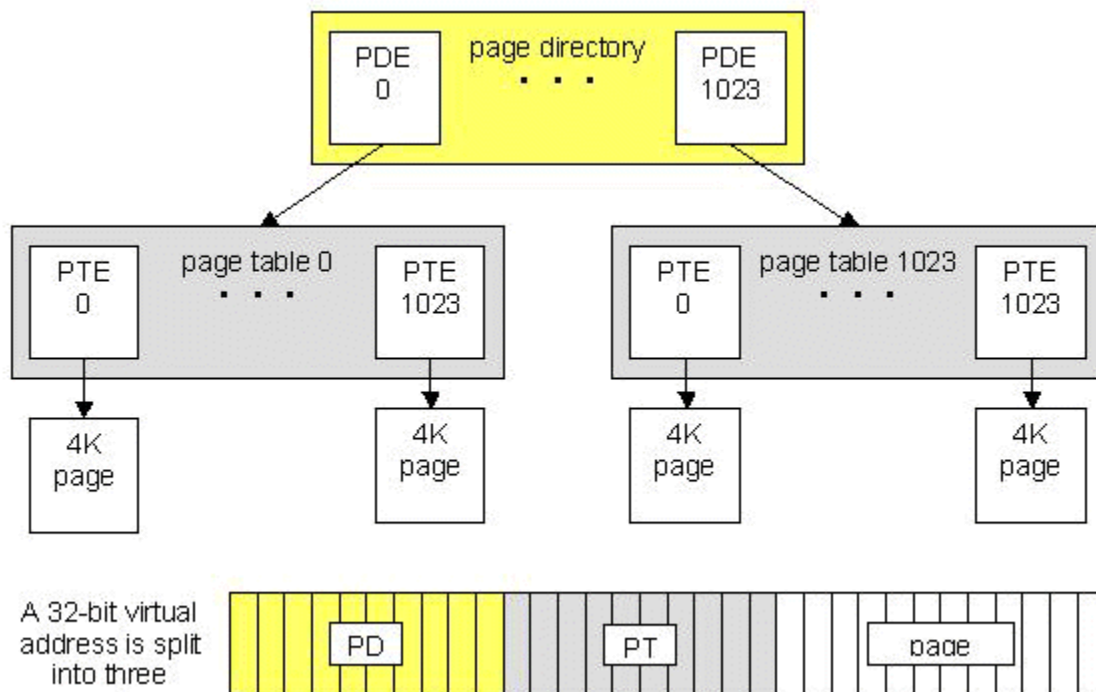
- Cơ chế chuyển đổi địa chỉ virtual sang physical
- Cơ chế phân trang

Sử dụng kỹ thuật phân trang với trang kích thước 4K

Sử dụng 32 bits địa chỉ => không gian địa chỉ ảo 4G

Cấu trúc dữ liệu hỗ trợ chuyển đổi địa chỉ:

- Một thư mục trang (page directory) gồm 1024 phần tử (PDE's) 4 bytes
- Mỗi PDE trỏ đến một bảng trang gồm 1024 phần tử 4 bytes (PTE's)
- Mỗi PTE trỏ đến một frame trong BNVL



Khi tiến trình khởi động, nó được cấp một workingset mặc định tối thiểu từ 30-50 trang (120-200K) . Sau đó chiến lược demand-paging được vận dụng để cấp phát các trang cho đến khi đạt workingset cực đại 1.4Mb. Khi cần thêm bộ nhớ, các trang yêu cầu thêm sẽ chỉ được nạp nếu bộ nhớ tự do còn lớn. Nếu không, trên máy cấu hình 1 CPU, 1 chiến lược thay thế trang xấp xỉ LRU được vận dụng để giải phóng

bớt các trang của những tiến trình đã sử dụng quá workingset tối thiểu.

### V.3 Hệ thống tập tin

WinNT hỗ trợ môi trường đa hệ thống tập tin với FA, HPFS và đặc biệt là hệ thống tập tin là NTFS với nhiều ưu điểm :

- Sử dụng địa chỉ 64-bit cluster : lo liệu được cho các đĩa cứng dung lượng lớn, ngay cả khi sử dụng cluster kích thước nhỏ (512)
- Có khả năng mở rộng kiểm soát an toàn cho mỗi tập tin
- Sử dụng tên file dài với UNICODE
- Ghi nhận lỗi trong các giao tác trong file \ \$LOGFILE
- Các cluster tự do được ghi nhận trong system file, \ \$BITMAP
- Các cluster hỏng được ghi nhận trong system file, \ \$BADCLUS
- Các cluster đang sử dụng được ghi nhận trong master file table

Mỗi file trong WinNT được mô tả bởi tối thiểu một mẫu tin trong Master file table (MFT)

- Kích thước mẫu tin trong MFT được đặc tả khi hệ thống được thiết lập, biến thiên từ 1K đến 4K.
- Tất cả các file đều được ánh xạ vào MFT, kể cả chính MFT

Mỗi mẫu tin trong MFT có một trường header và một hay nhiều attributes để mô tả cho tập tin tương ứng

- Mỗi attribute lại bao gồm header và data
- Nếu attribute nhỏ, nó được chứa ngay trong mẫu tin của MFT (ví dụ 'file' name, thông tin ngày giờ cập nhật...)
- Nếu attribute lớn (ví dụ nội dung file), số hiệu cluster chứa data sẽ được lưu trong mẫu tin.

### V.4 Quản lý nhập xuất

Mô hình IO của WinNT được thiết kế theo kiến trúc tầng cho phép các trình điều khiển thiết bị riêng biệt thực hiện ở các tầng logic khác nhau của việc quản lý IO. Các trình điều khiển có thể được tải vào hay gỡ bỏ khỏi hệ thống một cách linh động tùy theo nhu cầu của người dùng. Mô hình phân lớp cho phép chèn các trình điều khiển thiết bị mới một cách dễ dàng.

### V.5 Hỗ trợ mạng

NT hỗ trợ cả 2 mô hình kết nối mạng peer-to-peer và client-server

Một số protocol được hỗ trợ : SMB, NetBios, NetBeui, TCP/IP

### V.6 Hỗ trợ xử lý phân tán

NT cũng hỗ trợ một số cơ chế để thực hiện xử lý phân tán như: named pipe, mailslot, winsock và RPC

## Bài 15 **HỆ ĐIỀU HÀNH LINUX**

*Linux là một hệ điều hành tựa Unix rất mạnh, linh động, dễ mở rộng và hoàn toàn miễn phí. Nó được sử dụng hiệu quả trong nhiều lĩnh vực của tin học và những ứng dụng mới đang được xây dựng cho Linux mỗi ngày. Trong bài này chúng ta tìm hiểu hệ điều hành Linux với những nội dung sau:*

- Giới thiệu về hệ điều hành Linux
- Tổ chức hệ thống

Qua bài học này, chúng ta hiểu được những đặc điểm cơ bản và một số vấn đề về tổ chức hệ thống của hệ điều hành Linux. Từ đó chúng ta có một cái nhìn rộng hơn về các hệ điều hành và có thể lựa chọn được cho mình một môi trường làm việc và phát triển thuận lợi nhất.

Bài học này đòi hỏi những kiến thức tổng quát về cấu trúc và tổ chức của hệ điều hành.

### **I. GIỚI THIỆU**

Unix là một hệ điều hành thông dụng trên thế giới do được nhiều hệ thống hỗ trợ. Hệ điều hành này tồn tại trên hầu hết các kiểu máy tính kể cả máy tính cá nhân.

Linux là một hệ thống Unix tự do được hình thành từ một đồ án giải trí của một sinh viên người Phần Lan : Linus Torvalds, lấy nguồn cảm hứng từ Minix, một hệ thống Unix nhỏ do Andy Tanenbaum thiết kế. Tháng 10/1991 Linus Torvalds công bố version chính thức đầu tiên của Linux (version 0.02).

Ngày nay, Linux là một hệ thống 32-bits, multi-tasking, multi-user, mạng...nó được cài trên phần lớn máy PC (có thể chung với những hệ điều hành khác). Nó hỗ trợ đủ loại chương trình khác nhau như X Windows, TCP/IP, C/C++GNU và những công cụ khác của GNU, thư điện tử, những công cụ với LaTeX hoặc những tiện ích cho văn phòng (ApplixWare, StartOffice).

Sự thành công của Linux dựa vào nhiều yếu tố như :

Mã nguồn của hệ thống, cũng như của hạt nhân, các công cụ lập trình cũng như phát triển đều được phân phối tự do.

Linux tương thích với một số lớn các chuẩn của Unix ở mức độ mã nguồn, bao gồm những đặc tính của POSIX, system V, và BSD.

Một số lượng lớn các trình ứng dụng Unix miễn phí có thể sử dụng lại dưới Linux mà không cần mô tả lại.

Hệ thống Linux được thiết kế dựa trên bộ xử lý của Intel và tận dụng được tất cả những chức năng của chúng.

Linux là một hệ điều hành bao gồm hạt nhân, bộ thông dịch lệnh và nhiều trình tiện ích.

Hạt nhân quản lý những tài nguyên vật lý (như bộ xử lý, bộ nhớ, thiết bị ngoại vi) và logic (như tiến trình, tập tin...). Hạt nhân được tạo thành từ một tập hợp các thủ tục và hàm được thiết kế bằng ngôn ngữ C. Cấu trúc của hạt nhân là nguyên khối và lớp. Người sử dụng thao tác với Linux thông qua bộ thông dịch lệnh. Một trong những điểm đặc biệt của Linux là có nhiều shell khác nhau.

### **II. TỔ CHỨC HỆ THỐNG**

#### **II.1 Hệ thống tập tin**

Một trong những đặc điểm quan trọng của Linux là nó hỗ trợ nhiều hệ thống tập tin. Điều này làm cho nó rất linh động và có thể cùng tồn tại với nhiều hệ điều hành khác. Từ những phiên bản đầu tiên, Linux đã hỗ trợ 15 loại tập tin : ext, ext2, xia, minix, umsdos, msdos, vfat, proc, smb, ncp, iso9660, sysv, hpfs, affs và ufs.

Trong Linux, cũng như Unix, hệ thống tập tin được truy xuất thông qua một cấu trúc cây thừa kế đơn thay vì là các thiết bị xác định (như tên ổ đĩa hoặc số hiệu của ổ đĩa). Linux thêm những hệ thống tập tin mới vào cây này khi nó được thiết lập (mount). Tất cả hệ thống tập tin được thiết lập vào một thư mục và được gọi là thư mục thiết lập.

Khi một đĩa được khởi tạo, trên đĩa được chia thành nhiều partitions. Mỗi partition có một hệ thống tập tin. Hệ thống tập tin của Linux coi những thiết bị khối một cách đơn giản như là một tập hợp tuyến tính các khối mà không quan tâm đến tổ chức vật lý trên đĩa. Chương trình điều khiển thiết bị khối chịu trách nhiệm ánh xạ khối cần đọc thành vị trí vật lý cụ thể trên đĩa.

Trong hệ thống tập tin của Linux, người sử dụng không quan tâm đến những khác biệt của chúng trên những thiết bị vật lý khác nhau. Hệ thống tập tin không chỉ trên hệ thống cục bộ mà còn là những ổ đĩa từ xa trên mạng. Ví dụ một hệ thống tập tin của Linux :

|   |     |       |      |      |                 |     |
|---|-----|-------|------|------|-----------------|-----|
| A | E   | boot  | etc  | lib  | opt             | tmp |
| C | F   | cdrom | fd   | proc | root            | var |
| D | bin | dev   | home | mnt  | lost+found..... |     |

Hệ thống tập tin đầu tiên Linux hỗ trợ là Minix cho phép tập tin có tên tối đa là 14 ký tự và kích thước không vượt quá 64Mb. Hệ thống tập tin đặc thù đầu tiên của Linux là ‘hệ thống tập tin mở rộng’ (EXT) được giới thiệu vào tháng 4/1992, hệ thống này không hiệu quả nên sau đó vào năm 1993 ‘hệ thống tập tin mở rộng thế hệ 2’ được thêm vào.

Có một điểm quan trọng khi hệ thống tập tin EXT được đưa vào Linux. Hệ thống tập tin thật bị tách khỏi hệ điều hành và các dịch vụ hệ thống bằng một tầng giao tiếp gọi là hệ thống tập tin ảo VFS. VFS cho phép Linux hỗ trợ nhiều loại hệ thống tập tin khác nhau, mỗi loại được xem như là một giao tiếp phần mềm với VFS. Tất cả chi tiết của hệ thống tập tin được chuyển đổi sao cho chúng xuất hiện như nhau đối với phần còn lại của hạt nhân Linux và các chương trình thi hành trong hệ thống. Lớp VFS cho phép thiết lập một cách trong suốt nhiều hệ thống tập tin khác nhau cùng lúc. VFS được cài đặt sao cho việc truy xuất các tập tin của nó nhanh nhất có thể và nó cũng đảm bảo tính chính xác khi truy xuất.

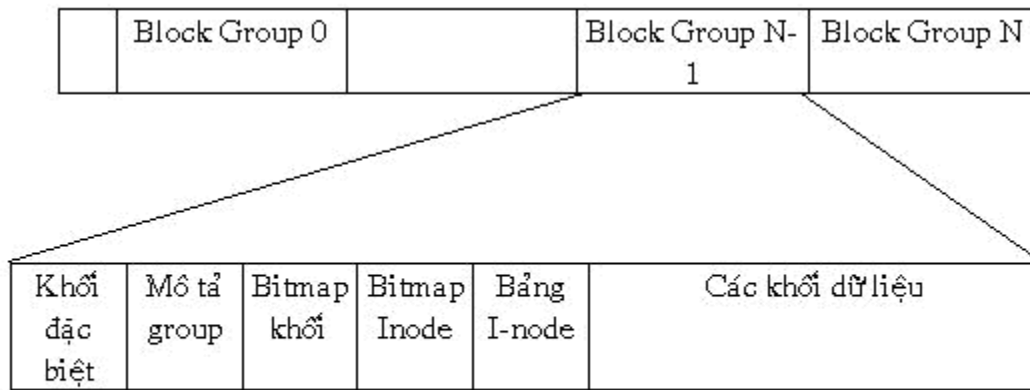
VFS đặt các thông tin của mỗi hệ thống tập tin vào bộ nhớ mỗi khi chúng được thiết lập và sử dụng. Các cấu trúc dữ liệu mô tả tập tin và thư mục truy xuất được tạo và hủy thường trực. Khi các khối được truy xuất, chúng được đặt vào vùng đệm. Vùng đệm không chỉ lưu giữ dữ liệu mà còn hỗ trợ quản lý giao tiếp không đồng bộ với bộ phận điều khiển thiết bị khối.

### ***‘hệ thống tập tin mở rộng thế hệ 2’ EXT2***

Là hệ thống tập tin thành công nhất của Linux. Hệ thống này, cũng như các hệ thống tập tin khác, được xây dựng trên cơ sở các khối dữ liệu. Các khối dữ liệu này có cùng kích thước và mọi kích thước tập tin được làm tròn thành một số nguyên các khối này.

EXT2 định nghĩa hệ thống tập tin bằng cách mô tả mỗi tập tin trong hệ thống bằng một cấu trúc I-node duy nhất và mỗi inode có một số để định danh. Các I-node này được đặt trong bảng I-node. Các thư mục trong EXT2 được xem như những tập tin đặc biệt chứa những con trỏ đến các i-node của các entry.

Hình 15.1 cho thấy tổng quan của hệ thống tập tin EXT2. Hệ thống này chia những partitions mà nó quản lý vào các Block Group.



**Hình 15.1** Tổng quan về hệ thống tập tin EXT2

## II.2 Điều khiển thiết bị

Như đã đề cập trong phần quản lý nhập xuất, một trong những mục tiêu chính của hệ điều hành là giúp cho người sử dụng độc lập với thiết bị. Trong Linux, các thiết bị phần cứng đều được xem như là các tập tin thông thường : chúng có thể được mở, đóng, đọc, ghi, và sử dụng cùng những lời gọi hệ thống để thao tác. Mọi thiết bị trong hệ thống đều được biểu diễn bởi các tập tin thiết bị, ví dụ đĩa IDE đầu tiên của hệ thống được biểu diễn bởi `/dev/hda`.

Linux ánh xạ tập tin thiết bị vào các lời gọi hệ thống. Linux hỗ trợ ba loại thiết bị là : ký tự, khối và mạng. Có nhiều chương trình điều khiển thiết bị khác nhau trong hạt nhân của Linux nhưng chúng cùng chia sẻ những thuộc tính chung :

Mã nguồn của hạt nhân : chương trình điều khiển thiết bị là một phần của hạt nhân và cũng như những đoạn mã nguồn khác nếu nó bị lỗi nó có thể đe dọa nghiêm trọng đến hệ thống.

Chương trình điều khiển thiết bị cung cấp một giao tiếp chuẩn với hạt nhân của Linux. Chúng sử dụng những dịch vụ chuẩn từ hạt nhân này.

Hầu hết các chương trình điều khiển thiết bị của Linux được nạp như những đơn thể của hạt nhân khi có yêu cầu và được gỡ ra khi nó không còn được sử dụng nữa.

Khi khởi động hệ thống mỗi chương trình điều khiển thiết bị được khởi tạo và nó sẽ tìm thiết bị phần cứng mà nó sẽ điều khiển.

Linux sử dụng cơ chế DMA và sử dụng một vector cấu trúc ***dma\_chan*** để quản lý các kênh DMA (mỗi kênh có một vector).

Mỗi lớp điều khiển thiết bị : ký tự, khối, mạng cung cấp những giao tiếp chung với hạt nhân. Khi một thiết bị được khởi tạo, thanh ghi điều khiển thiết bị bên trong hạt nhân của Linux được thêm một entry trong vector ***chrdevs*** của cấu trúc ***devive\_struct***. Một định danh(id) được xác định cho vector này và là cố định. Mỗi entry trong vector ***chrdevs***, một ***device\_struct*** gồm hai phần tử : một con trỏ đến tên của thanh ghi điều khiển thiết bị và một con trỏ tới khối của tập tin thao tác. Khối của tập tin thao tác này là địa chỉ của những thủ tục trong chương trình điều khiển thiết bị mà chúng kiểm soát những thao tác như mở, đóng, đọc, ghi. Nội dung của `/proc/devices` cho các thiết bị ký tự được lấy từ vector ***chrdevs***.

Các thiết bị khối được truy xuất như những tập tin. Linux quản lý các thanh ghi thiết bị khối bằng vector ***blkdevs***, và tương tự như vector ***chrdevs*** mỗi entry của nó là các cấu trúc ***device\_struct*** nhưng có hai loại

thiết bị khối : SCSI và IDE. Mỗi chương trình điều khiển thiết bị khối phải cung cấp một giao tiếp với vùng đệm cũng như những thao tác tập tin thông thường. Cấu trúc **blk\_dev\_struct** bao gồm địa chỉ của các thủ tục được yêu cầu và một con trỏ đến những cấu trúc dữ liệu cần thiết, mỗi cấu trúc này sẽ đại diện cho một yêu cầu từ vùng đệm cho những điều khiển đọc hoặc ghi một khối dữ liệu.

Trong Linux, mỗi thiết bị mạng là một thực thể có thể nhận hoặc gửi gói dữ liệu. Mỗi thiết bị mạng được biểu diễn bằng một cấu trúc **device**. Các chương trình điều khiển thiết bị mạng sẽ ghi nhận những thiết bị mà chúng sẽ điều khiển trong quá trình khởi tạo mạng lúc khởi động hệ thống. Cấu trúc **device** chứa những thông tin về các thiết bị và địa chỉ của các hàm hỗ trợ những nghi thức và dịch vụ mạng khác nhau. Những hàm này chủ yếu tập trung vào việc chuyển dữ liệu sử dụng các thiết bị mạng, các thiết bị sử dụng các cơ chế hỗ trợ mạng chuẩn để chuyển dữ liệu nhận được cho lớp nghi thức thích hợp. Tất cả các gói dữ liệu chuyển và nhận được biểu diễn bởi cấu trúc **sk\_buff**, đây là một cấu trúc linh động cho phép các tiêu đề (header) nghi thức mạng có thể dễ dàng thêm vào hoặc loại bỏ.

Cấu trúc **device** lưu những thông tin về thiết bị mạng : tên, thông tin về đường truyền, cờ trạng thái giao tiếp (mô tả các thuộc tính và khả năng của các thiết bị mạng), thông tin về nghi thức, hàng đợi gói tin (đây là hàng đợi của các gói **sk\_buff** chờ chuyển dữ liệu trên thiết bị mạng đó), các hàm hỗ trợ (mỗi thiết bị cung cấp một tập hợp các thủ tục chuẩn được gọi bởi lớp nghi thức).

### II.3 Quản lý tiến trình

Mỗi tiến trình trong Linux được biểu diễn bằng một cấu trúc dữ liệu **task\_struct** (task có nghĩa là tiến trình trong Linux). Linux sử dụng **task** vector để quản lý các con trỏ đến các **task\_struct**, mặc định là có 512 phần tử. Khi một tiến trình được tạo ra, một **task\_struct** mới được cấp phát trong bộ nhớ và được thêm vào vector **task**. Linux hỗ trợ hai loại tiến trình là loại bình thường và loại thời gian thực. Cấu trúc **task\_struct** gồm những trường như sau : trạng thái (thi hành, chờ, ngưng, lưng chừng), thông tin lập lịch, định danh, thông tin liên lạc giữa các tiến trình, liên kết, định thời gian, hệ thống tập tin, bộ nhớ ảo, ngữ cảnh.

Trong Linux cũng như Unix, chương trình và lệnh được thực hiện theo cơ chế thông dịch. Bộ thông dịch được gọi là shell. Linux hỗ trợ nhiều loại shell như sh, bash, tcsh. Tập tin thi hành có nhiều dạng, dạng được sử dụng thông dụng nhất trong Linux là EFL, ngoài ra Linux cũng có thể hiểu được nhiều dạng tập tin khác.

Các cơ chế thông tin liên lạc giữa các tiến trình được sử dụng là tín hiệu (SIGNALS), đường ống (PIPE), sockets, semaphore và bộ nhớ chia sẻ.

Có một tập hợp các tín hiệu được định nghĩa trước, các tín hiệu này có thể được phát sinh bởi hạt nhân hoặc những tiến trình khác trong bộ nhớ. Linux cài đặt các tín hiệu trong **task\_struct**. Không phải mọi tiến trình trong hệ thống có thể gửi tín hiệu đến mọi tiến trình khác. Tín hiệu được phát sinh bằng cách thiết lập một bit thích hợp trong trường signal của cấu trúc **task\_struct**. Tín hiệu không xuất hiện cho tiến trình ngay khi nó được tạo mà phải chờ đến khi tiến trình được thực hiện trở lại.

Trong Linux cơ chế đường ống cài đặt sử dụng hai cấu trúc file trỏ đến cùng inode VFS tạm thời xác định một trang vật lý trong bộ nhớ. Khi tiến trình ghi thực hiện việc ghi vào đường ống, các byte sẽ được chép vào trang dữ liệu chia sẻ. Linux phải đồng bộ quá trình truy xuất trong đường ống. Tiến trình ghi sử dụng các hàm thư viện ghi chuẩn.

Các cơ chế socket, semaphores và bộ nhớ chia sẻ của Linux gần giống với hệ thống Unix System V.

### II.4 Quản lý bộ nhớ

Linux hỗ trợ hệ thống chia sẻ bộ nhớ IPC (Inter Process Communication- thông tin liên lạc giữa các tiến

trình) của Unix System V.

Linux sử dụng bộ nhớ ảo và cơ chế phân trang. Trên hệ thống Alpha AXP một trang có kích thước 8Kb và trên hệ thống Intel x86 một trang có kích thước 4Kb. Chiến lược thay trang được áp dụng là LRU.

Linux có số cấp bảng trang tùy vào các nền khác nhau (Alpha 3, Intel x86 2). Bảng trang sẽ được chuyển thành các tiến trình đặc thù trên các nền khác nhau, điều này giúp cho việc thao tác trên bảng trang không lệ thuộc vào các nền này.

Linux sử dụng thuật toán Buddy để thực hiện việc cấp phát và thu hồi các khối của trang. ( 1 khối = 1, 2, hoặc 4 trang...)

### **Câu hỏi kiểm tra kiến thức**

1. Cho biết quá trình phát triển của hệ điều hành Linux.
2. Nêu những đặc điểm nổi bật của hệ điều hành Linux.
3. Hệ thống quản lý tập tin của Linux có gì đặc biệt ?
4. Linux quản lý nhập xuất như thế nào?
5. Theo anh(chị), trong tương lai hệ điều hành Linux có thể được sử dụng rộng rãi như Windows ngày nay không ? Tại sao
6. Nếu phải chọn một hệ điều hành để cài lên máy PC của mình, anh (chị) chọn hệ điều hành nào ? Tại sao ?