NÔNG MINH NGỌC (Chủ biên), NGUYỄN VĂN HUY

GIÁO TRÌNH

NGUYÊN LÝ HỆ ĐIỀU HÀNH

NHÀ XUẤT BẢN ĐẠI HỌC THÁI NGUYÊN
NĂM 2016

MÃ SỐ: 01 - 67 / ĐHTN - 2016

MỤC LỤC

Chương 1

TỔNG QUAN VỀ HỆ ĐIỀU HÀNH

Chương I cung cấp một cái nhìn tổng quát về những nguyên lý cơ bản của hệ điều hành. Bắt đầu từ việc xem xét mục tiêu và các chức năng của hệ điều này, sau đó đến việc khảo sát các dạng khác nhau cũng như xem xét quá trình phát triển qua từng giai đoạn của chúng. Các phần này được trình bày trong các nội dung 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

Chương I giúp 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?. Chương I đòi hỏi những kiến thức cơ bản về kiến trúc máy tính.

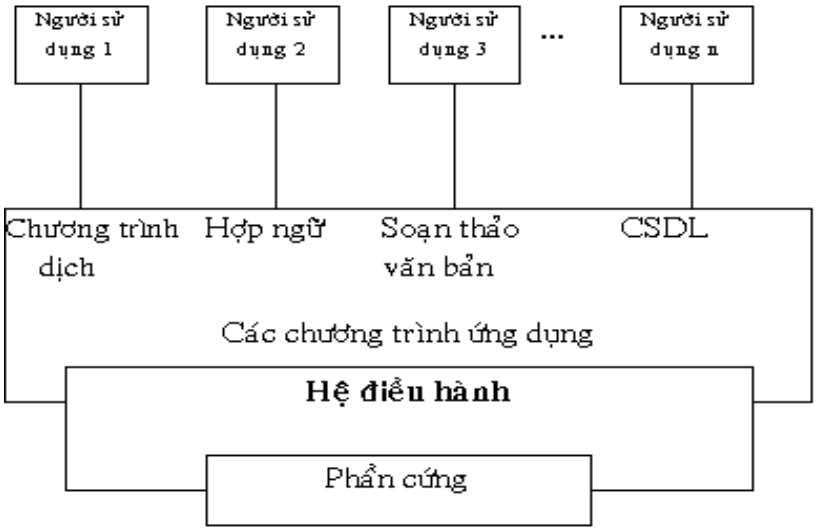
1.1. Khái niệm về hệ điều hành

Hệ điều hành là một *chương trình* hay một *hệ chương trình* hoạt động giữa người sử dụng (user) và phần cứng của máy tính. Mục tiêu của hệ điều hành là cung cấp môi trường để người sử dụng có thể thi hành các chương trình giúp 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 được coi là bộ phân phối tài nguyên của máy tính bao gồm: thời gian sử dụng CPU, vùng bộ nhớ, vùng lưu trữ tập tin, thiết bị nhập xuất v.v… do các chương trình yêu cầu nhằm giải quyết những vấn đề do người dùng đặt ra.

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

Tuy nhiên, nhìn chung chưa có định nghĩa nào là hoàn hảo về hệ điều hành. Hệ điều hành tồn tại để giải quyết các vấn đề sử dụng hệ thống máy tính. Mục tiêu cơ bản của nó là giúp cho việc thi hành các chương trình dễ dàng hơn. Mục tiêu thứ hai là hỗ trợ cho các thao tác trên hệ thống máy tính hiệu

quả hơn. Mục tiêu này đặc biệt quan trọng trong những hệ thống nhiều người dùng và trong những hệ thống lớn (phần cứng + quy mô sử dụng). Tuy nhiên hai mục tiêu này cũng có phần tương phản vì vậy lý thuyết về hệ điều hành tập trung vào việc tối ưu hóa việc sử dụng tài nguyên của máy tính.

1.2. Phân loại hệ điều hành

1.2.1. Hệ thống xử lý theo lô

** Bộ giám sát thường trực:*

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

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

** CPU và thao tác nhập xuất:*

CPU thường hay nhàn rỗi do tốc độ làm việc của các thiết bị nhập xuất (thường là thiết bị cơ) chậm hơn rất nhiều lần so với các thiết bị điện tử. Cho dù là một CPU chậm nhất, nó cũng nhanh hơn rất nhiều lần so với thiết bị nhập xuất. Do đó phải có các phương pháp để đồng bộ hóa việc hoạt động của CPU và thao tác nhập xuất.

** Xử lý off_line:*

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

** Spooling:*

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

1.2.2. Hệ thống xử lý theo lô đa chương

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

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

1.2.3. Hệ thống chia sẻ thời gian

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

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

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

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

1.2.4. Hệ thống song song

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

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

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

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

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

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

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

1.2.5. Hệ thống phân tán

Hệ thống này cũng tương tự như hệ thống chia sẻ thời gian nhưng các bộ xử lý không chia sẻ bộ nhớ và đồng hồ, thay vào đó mỗi bộ xử lý có bộ nhớ

cục bộ riêng. Các bộ xử lý thông tin với nhau thông qua các đường truyền thông như những bus tốc độ cao hay đường dây điện thoại.

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

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

* *Chia sẻ tài nguyên*: Một người sử dụng A có thể sử dụng máy in laser của người sử dụng B và người sử dụng B có thể truy xuất những tập tin của A. Tổng quát, chia sẻ tài nguyên trong hệ thống phân tán cung cấp một cơ chế để chia sẻ tập tin ở vị trí xa, xử lý thông tin trong một cơ sở dữ liệu phân tán, in ấn tại một vị trí xa, sử dụng những thiết bị ở xa 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.

1.2.6. Hệ thống xử lý thời gian thực

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

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

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

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

1.3. Cấu trúc của hệ điều hành

1.3.1. Các thành phần của hệ thống

1.3.1.1. 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).

1.3.1.2. 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 phần tử đều có địa chỉ. Đó là nơi lưu dữ liệu được CPU truy xuất một cách nhanh chóng so với các thiết bị nhập/xuất. CPU đọc những chỉ thị từ bộ nhớ chính. Các thiết bị nhập/xuất cài đặt cơ chế DMA (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.

1.3.1.3. 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 tuỳ thuộc rất nhiều vào tốc độ truy xuất đĩa.

1.3.1.4. 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 thao tác hơn.

Một hệ thống nhập/xuất bao gồm:

* Hệ thống buffer caching.

* Giao tiếp

1.4. Lịch sử phát triển các hệ điều hành

1.4.1. 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.

1.4.2. 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.

1.4.3. 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 thiế 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 xẻ 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.

1.4.4. 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*.

Chương 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 chính trong chương này.

2.1. 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:

2.1.1. Tăng hiệu suất sử dụng CPU

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

CPU	**IO**	**CPU**	**IO**	**CPU**

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

CPU	**IO**	**CPU**	**IO**	**CPU**

Tác vụ 1

	CPU	**IO**	**CPU**	**IO**

Tác vụ 2

2.1.2. Tăng tốc độ xử lý

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

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

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

2.2. Khái niệm tiến trình (Process) và mô hình đa tiến trình (Multiprocess)

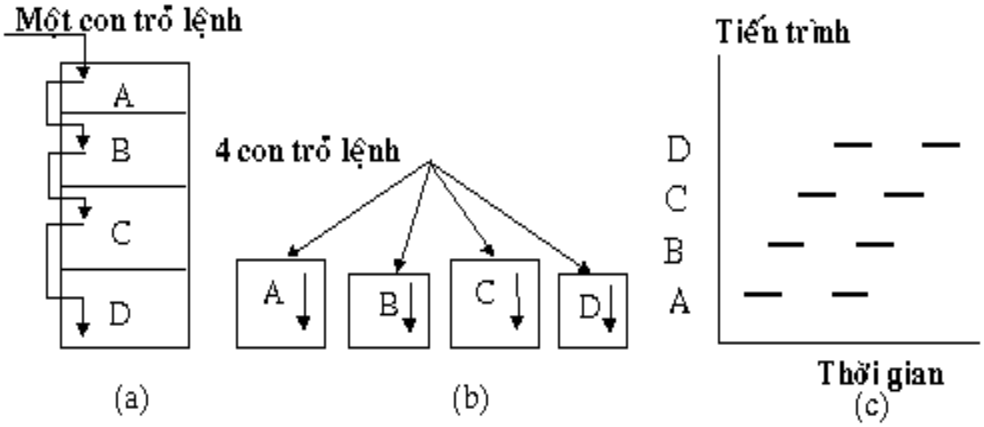
Để hỗ trợ sự đa chương, máy tính phải có khả năng thực hiện nhiều tác vụ đồng thời. Nhưng việc điều khiển nhiều hoạt động song song ở cấp độ phần cứng là rất khó khăn. Vì thế, các nhà thiết kế hệ điều hành đề xuất một mô hình *song song giả lập* bằng cách chuyển đổi bộ xử lý qua lại giữa các chương trình 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, sỡ hữu một con trỏ lệnh, tập các thanh ghi và các biến. Để hoàn thành tác vụ của mình, một tiến trình có thể cần đến một số tài nguyên – như CPU, bộ nhớ chính, các tập tin và thiết bị nhập/xuất.

Cần phân biệt hai khái niệm *chương trình* và *tiến trình*. Một chương trình là một thực thể thụ động, chứa đựng các chỉ thị điều khiển máy tính để tiến hành một tác vụ nào đó; khi cho thực hiện các chỉ thị này, chương trình chuyển thành tiến trình, 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 độc lập

(c) Tại một thời điểm chỉ có một chương trình hoạt động

2.3. Khái niệm tiểu trình (Thread) và mô hình đa tiểu trình

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

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

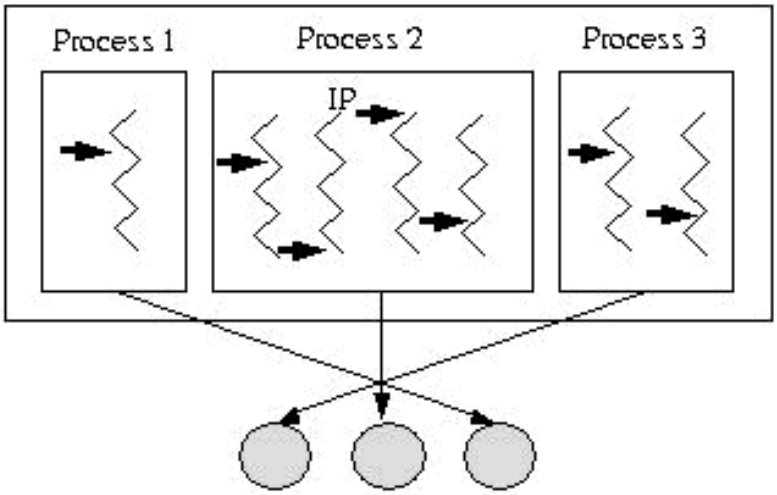
Chính vì các tình huống tương tự, người ta cần có một cơ chế xử lý mới cho phép có nhiều dòng xử lý trong cùng một tiến trình.

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

2.3.1. 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 sỡ hữu chủ đã tạo ra chúng trong ý định cho phép chúng hợp tác với nhau.



Hình 2.2. *Các tiểu trình trong cùng một tiểu trình*

2.3.2. Phân bổ thông tin lưu trữ

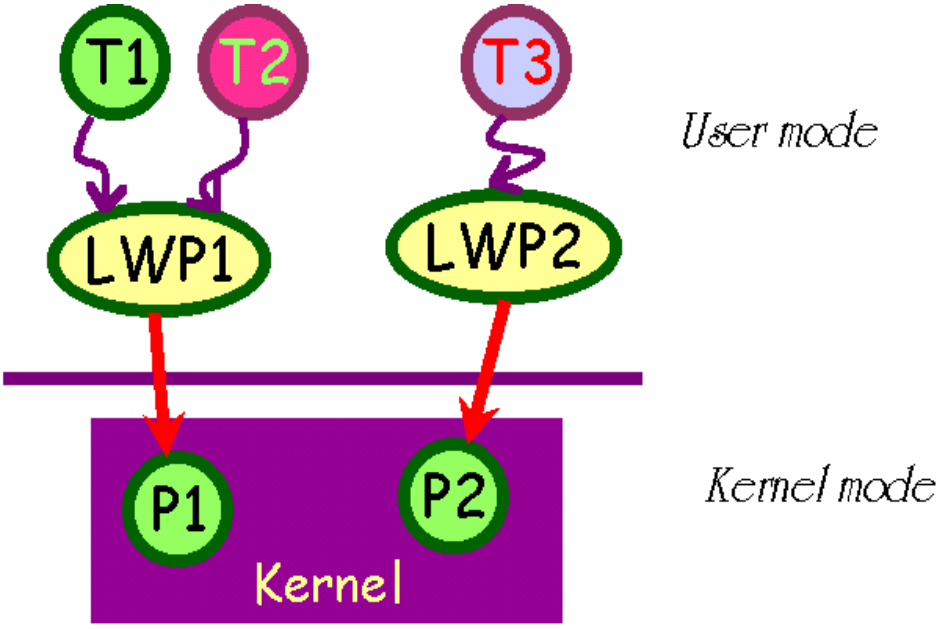
Tiến trình
Không gian địa chỉ Tài nguyên toàn cục Các thông tin thống kê

Tiểu trình
Con trỏ lệnh + các thanh ghi Stack Tài nguyên cục bộ

Cấu trúc mô tả tiến trình và tiểu trình

2.3.3. Kernel thread và user thread

Khái niệm tiến 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 bộ thư viện xử lý trong chế độ người dùng không đặc quyền (user mode). Lúc này Hệ điều hành sẽ chỉ biết đến khái niệm tiến trình, do vậy cần có 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).



Hình 2.3.

2.4. 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 qủa 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.

Chương 3

QUẢN LÝ TIẾN TRÌNH

Chương này triển khai các nội dung về 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? Các thao tác trên tiến trình?

3.1. Tổ chức quản lý tiến trình

3.1.1. Các trạng thái của tiến trình

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

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

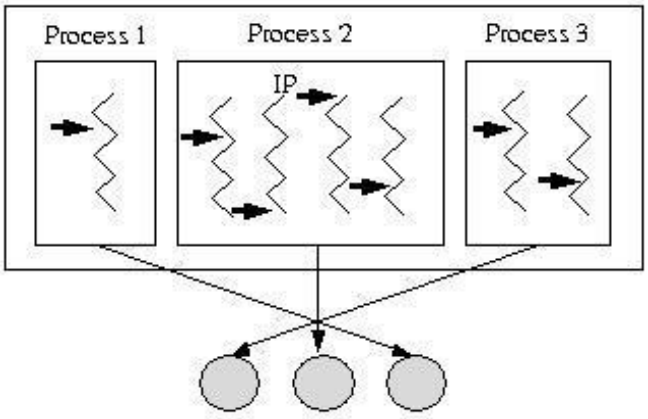
***** *Mới tạo*: tiến trình đang được tạo lập.

***** *Running*: các chỉ thị của tiến trình đang được xử lý.

***** *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 3.1. *Sơ đồ chuyển trạng thái giữa các tiến trình*

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

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

***** Tiến trình mới tạo được đưa vào hệ thống.

***** Bộ điều phối cấp phát cho tiến trình một khoảng thời gian sử dụng CPU.

***** Tiến trình kết thúc.

***** Tiến trình yêu cầu một tài nguyên nhưng chưa được đáp ứng vì tài nguyên chưa sẵn sàng để cấp phát tại thời điểm đó ; hoặc tiến trình phải chờ một sự kiện hay thao tác nhập/xuất.

***** Bộ điều phối chọn một tiến trình khác để cho xử lý.

***** Tài nguyên mà tiến trình yêu cầu trở nên sẵn sàng để cấp phát; hay sự kiện hoặc thao tác nhập/xuất tiến trình đang đợi hoàn tất.

3.1.2. Chế độ xử lý của tiến trình

Để đảm bảo hệ thống hoạt động đúng đắn, hệ điều hành cần phải được bảo vệ khỏi sự xâm phạm của các tiến trình. Bản thân các tiến trình và dữ liệu cũng cần được bảo vệ để tránh các ảnh hưởng sai lạc lẫn nhau. Một cách tiếp

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

Hình 3.2. *Hai chế độ xử lý*

3.1.3. Cấu trúc dữ liệu khối quản lý tiến trình

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

***** *Định danh của tiến trình (1):* giúp phân biệt các tiến trình.

***** *Trạng thái tiến trình (2):* xác định hoạt động hiện hành của tiến trình.

* *Ngữ cảnh của tiến trình (3)*: mô tả các tài nguyên tiến trình đang trong quá trình, hoặc để phục vụ cho hoạt động hiện tại, hoặc để làm cơ sở phục hồi hoạt động cho tiến trình, bao gồm các thông tin về:

- *Trạng thái CPU*: bao gồm nội dung các thanh ghi, quan trọng nhất là con trỏ lệnh IP lưu trữ địa chỉ câu lệnh kế tiếp tiến trình sẽ xử lý. Các thông tin

này cần được lưu trữ khi xảy ra một ngắt, nhằm có thể cho phép phục hồi hoạt động của tiến trình đúng như trước khi bị ngắt.

- *Bộ xử lý*: dùng cho máy có cấu hình nhiều CPU, xác định số hiệu CPU mà tiến trình đang sử dụng.

- *Bộ nhớ chính*: danh sách các khối nhớ được cấp cho tiến trình.

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

- *Tài nguyên tạo lập*: danh sách các tài nguyên được tiến trình tạo lập.

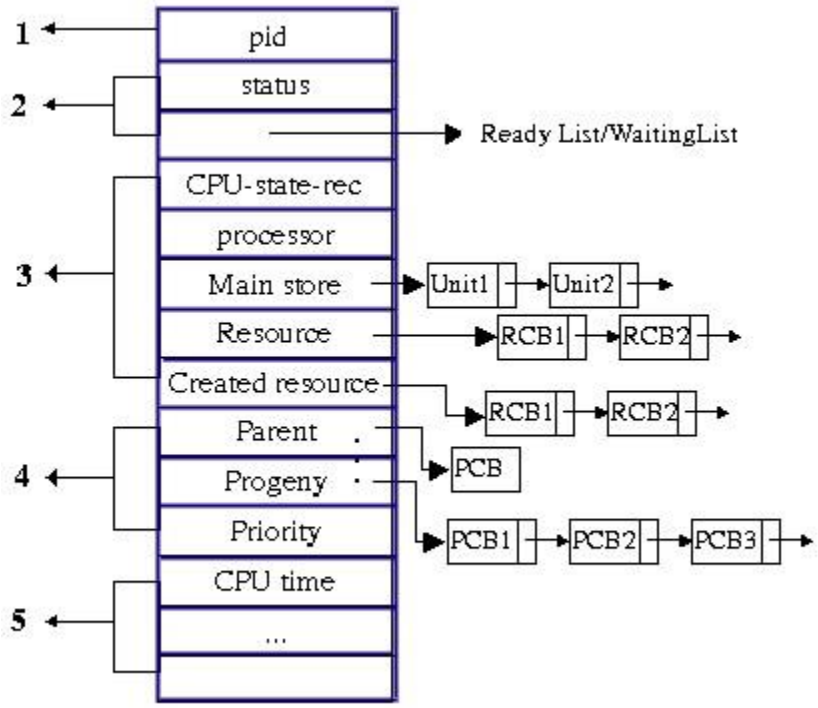
***** *Thông tin giao tiếp (4)*: phản ánh các thông tin về quan hệ của tiến trình với các tiến trình khác trong hệ thống :

- *Tiến trình cha*: 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 3.3. *Khối mô tả tiến trình*

3.1.4. Thao tác trên tiến trình

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

- Tạo lập tiến trình (create)

- Kết thúc tiến trình (destroy)

- Tạm dừng tiến trình (suspend)

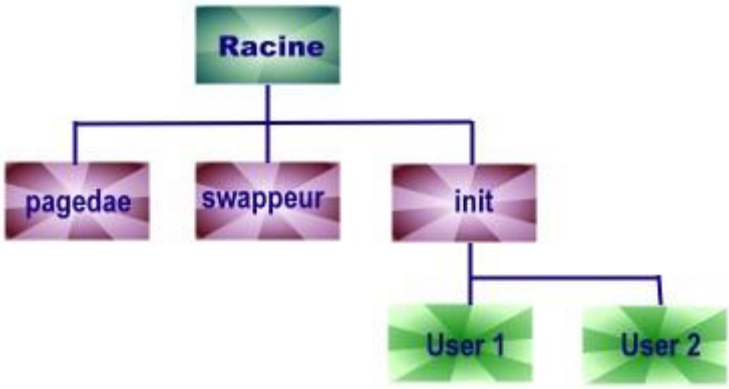
- Tái kích hoạt tiến trình (resume)

- Thay đổi độ ưu tiên tiến trình

3.1.4.1. Tạo lập tiến trình

Trong quá trình xử lý, một tiến trình có thể tạo lập nhiều tiến trình mới bằng cách sử dụng một lời gọi hệ thống tương ứng. Tiến trình gọi lời gọi hệ thống để tạo tiến trình mới sẽ được gọi là tiến trình *cha*, tiến trình được tạo gọi

là tiến trình *con*. Mỗi tiến trình con đến lượt nó lại có thể tạo các tiến trình mới…, quá trình này tiếp tục sẽ tạo ra một *cây tiến trình*.



Hình 3.4. *Một cây tiến trình trong hệ thống UNIX*

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

- Định danh cho tiến trình mới phát sinh

- Đưa tiến trình vào danh sách quản lý của hệ thống

- Xác định độ ưu tiên cho tiến trình

- Tạo PCB cho tiến trình

- Cấp phát các tài nguyên ban đầu cho tiến trình

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

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

- Tiến trình cha tiếp tục xử lý đồng hành với tiến trình con.

- Tiến trình cha chờ đến khi một tiến trình con nào đó, hoặc tất cả các tiến trình con kết thúc xử lý.

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

3.1.4.2. Kết thúc tiến trình

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

- Thu hồi các tài nguyên hệ thống đã cấp phát cho tiến trình

- Hủy tiến trình khỏi tất cả các danh sách quản lý của hệ thống

- Hủy bỏ PCB của tiến trình

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

3.1.5. Cấp phát tài nguyên cho tiến trình

Khi có nhiều người sử dụng đồng thời làm việc trong hệ thống, hệ điều hành cần phải cấp phát các tài nguyên theo yêu cầu cho mỗi người sử dụng. Do tài nguyên hệ thống thường rất giới hạn và có khi không thể chia sẻ, nên hiếm khi tất cả các yêu cầu tài nguyên đồng thời đều được thỏa mãn. Vì thế cần phải nghiên cứu một phương pháp để chia sẻ một số tài nguyên hữu hạn giữa nhiều tiến trình người dùng đồng thời. Hệ điều hành quản lý nhiều loại tài nguyên khác nhau (CPU, bộ nhớ chính, các thiết bị ngoại vi…), với mỗi loại cần có một cơ chế cấp phát và các chiến lược cấp phát hiệu 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 3.5. *Khối quản lý tài nguyên*

Các mục tiêu của kỹ thuật cấp phát:

* Bảo đảm một số lượng hợp lệ các tiến trình truy xuất đồng thời đến các tài nguyên không chia sẻ được.

* Cấp phát tài nguyên cho tiến trình có yêu cầu trong một khoảng thời gian trì hoãn có thể chấp nhận được.

* Tối ưu hóa sự sử dụng tài nguyên.

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

3.2. Điều phối tiến trình

Trong môi trường đa nhiệm, có thể xảy ra tình huống nhiều tiến trình đồng thời sẵn sàng để xử lý. Mục tiêu của các hệ phân chia thời gian (time-

sharing) là chuyển đổi CPU qua lại giữa các tiến trình một cách thường xuyên để nhiều người sử dụng có thể tương tác cùng lúc với từng chương trình trong quá trình xử lý.

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

3.2.1. Giới thiệu

3.2.1.1. Mục tiêu 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.

3.2.1.2. Các đặc điểm của tiến trình

Điều phối hoạt động của các tiến trình là một vấn đề rất phức tạp, đòi hỏi hệ điều hành khi giải quyết phải xem xét nhiều yếu tố khác nhau để có thể đạt

được những mục tiêu đề ra. Một số đặc tính của tiến trình cần được quan tâm như tiêu chuẩn điều phối:

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.

3.2.1.3. Điều phối không độc quyền và điều phối độc quyền (preemptive/nopreemptive)

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

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

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

- Khi tiến trình kết thúc.

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

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

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

- Khi tiến trình chuyển từ trạng thái đang xử lý (running) sang trạng thái ready (ví dụ xảy ra một ngắt).

- Khi tiến trình chuyển từ trạng thái chờ (blocked) sang trạng thái ready (ví dụ một thao tác nhập/xuất hoàn tất).

- Khi tiến trình kết thúc.

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

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

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

3.2.2. Tổ chức điều phối

3.2.2.1. Các danh sách sử dụng trong quá trình điều phối

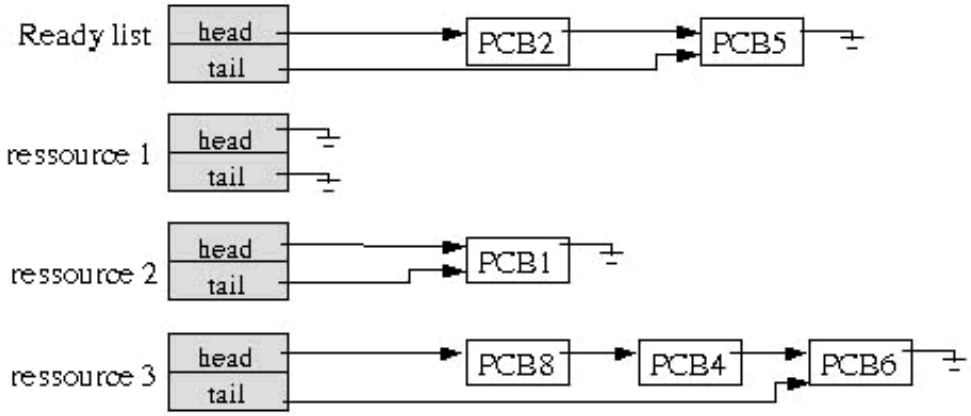
Hệ điều hành sử dụng hai loại danh sách để thực hiện điều phối các tiến trình là *danh sách sẵn sàng (ready list)* và *danh sách chờ đợi (waiting list)*.

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.



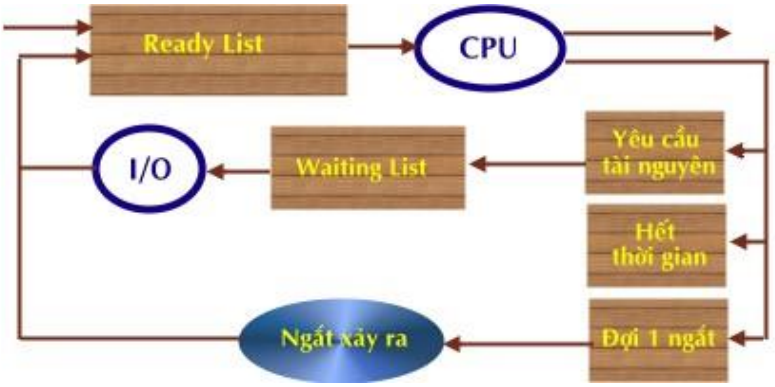
Hình 3.6. *Các danh sách điều phối*

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

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

- Tiến trình phát sinh một yêu cầu một tài nguyên mà hệ thống chưa thể đáp ứng, khi đó tiến trình sẽ được chuyển sang danh sách các tiến trình đang chờ tài nguyên tương ứng.

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



Hình 3.7. *Sơ đồ chuyển đổi giữa các danh sách điều phối*

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

3.2.2.2. Các cấp độ 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 nhiệm của hệ thống (số lượng tiến trình trong bộ nhớ chính). Khi hệ thống tạo lập một tiến trình, hay có một tiến trình kết thúc xử lý thì chức năng điều phối tác vụ mới được kích hoạt. Vì mức độ đa chương tương đối ổn định nên chức năng điều phối tác vụ có tần suất hoạt động thấp .

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

lựa chọn các tiến trình để nạp vào bộ nhớ sao cho hệ thống là sự pha trộn hợp lý giữa các tiến trình *hướng nhập xuất* và các tiến trình *hướng xử lý*

b) 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.

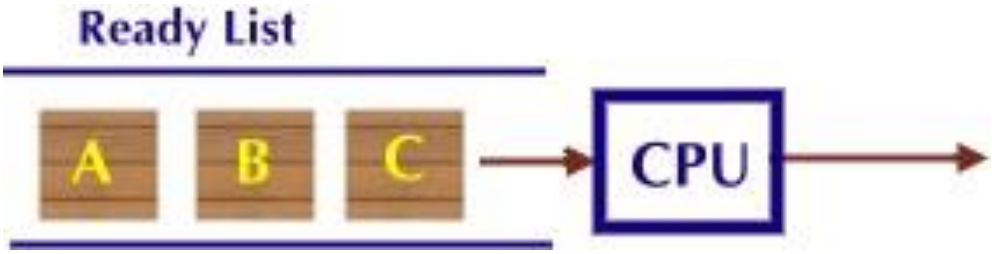
Trong nhiều hệ điều hành, có thể không có bộ điều phối tác vụ hoặc tách biệt rất ít đối với bộ điều phối tiến trình. Một vài hệ điều hành lại đưa ra một cấp độ điều phối trung gian kết hợp cả hai cấp độ điều phối tác vụ và tiến trình.

Hình 3.8. *Cấp độ điều phối trung gian*

3.2.3. Các chiến lược điều phối

3.2.3.1. Chiến lược FIFO

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



Hình 3.9. *Điều phối FIFO*

Ví dụ:

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

Thứ tự cấp phát CPU cho các tiến trình là:

P1	P2	P3
0	‘24	27 30

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

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

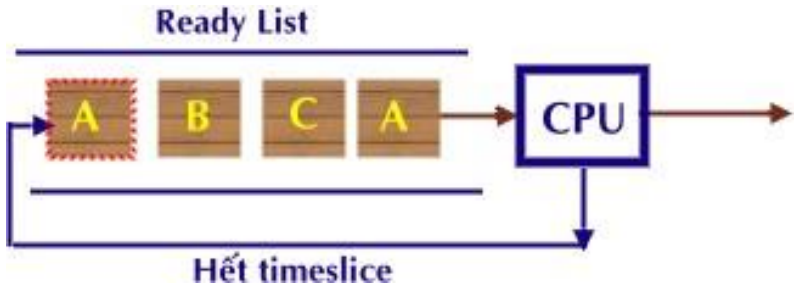
Giải thuật này đặc biệt không phù hợp với các hệ phân chia thời gian, trong các hệ này, cần cho phép mỗi tiến trình được cấp phát CPU đều đặn trong từng khoảng thời gian.

3.2.3.2. 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 3.10. *Điều phối Round Robin*

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

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

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

Thời gian chờ đợi trung bình sẽ là (0+6+3+5)/3 = 4.66 milisecondes.

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

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

3.2.3.3. 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 chèn tiến trình mới vào danh sách sẵn sàng, và tiến trình hiện hành vẫn tiếp tục xử lý hết thời gian dành cho nó.

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

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

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

P1	P2	P3
0	‘24	27 30

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

P1	P2	P3	P1
0	‘1	4	7 30

***** *Thảo luận*: Tình trạng "đói CPU" (starvation) là một vấn đề chính yếu của các giải thuật sử dụng độ ưu tiên. Các giải thuật này có thể để các tiến trình có độ ưu tiên thấp chờ đợi CPU vô hạn. Để ngăn cản các tiến trình có độ ưu tiên cao chiếm dụng CPU vô thời hạn, bộ điều phối sẽ giảm dần độ ưu tiên của các tiến trình này sau mỗi ngắt đồng hồ. Nếu độ ưu tiên của tiến trình này giảm xuống thấp hơn tiến trình có độ ưu tiên cao thứ nhì, sẽ xảy ra sự chuyển đổi quyền sử dụng CPU. Quá trình này gọi là sự "lão hóa" *(aging)* tiến trình.

3.2.3.4. Chiến lược công việc ngắn nhất (Shortest-job-first SJF)

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

Ví dụ:

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

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

P1	P4	P3	P2
0	6	8	12 20

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

P1	P4	P1	P3	P2
0	3	5	8	12 20

***** *Thảo luận*: Giải thuật này cho phép đạt được thời gian chờ trung bình cực tiểu. Khó khăn thực sự của giải thuật SJF là không thể biết được thời gian yêu cầu xử lý còn lại của tiến trình ? Chỉ có thể dự đoán giá trị này theo cách tiếp cận sau : gọi *tn* là độ dài của thời gian xử lý lần thứ *n*, τ_{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:

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

Trong công thức này,tn chứa đựng thông tin gần nhất; τ_n chứa đựng các thông tin quá khứ được tích lũy. Tham số α ($0 \le \alpha \le 1$) kiểm soát trọng số của hiện tại gần hay quá khứ ảnh hưởng đến công thức dự đón.

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

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

Hình 3.11. *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:

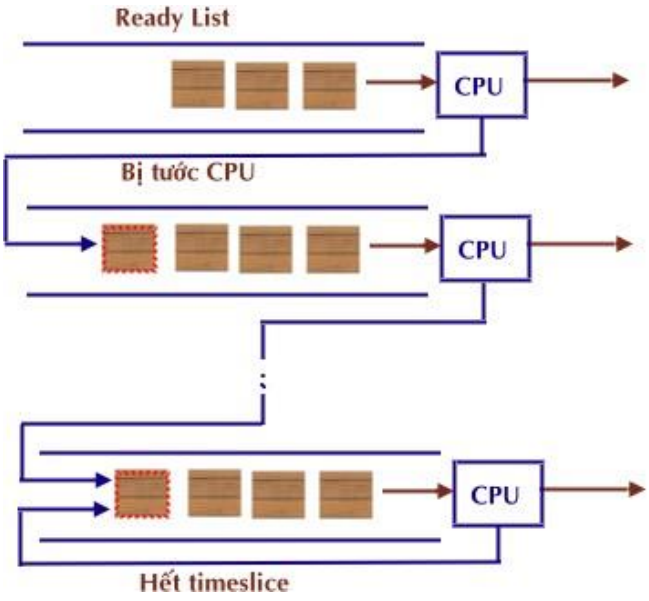
- Số lượng các cấp ưu tiên.

- 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 tiên nào.



Hình 3.12 *Điều phối Multilevel Feedback*

3.2.3.6. 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 sỡ 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:

3.3. 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_1	0	10	3
P_2	1	1	1
P_3	2	2	3
P_4	3	1	4
P_5	4	5	2

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

a) 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 > ...); và RR (quantum=2).

b) Cho biết thời gian lưu lại trong hệ thống (turnaround time) của từng tiến trình trong từng thuật toán điều phối ở câu a.

c) Cho biết thời gian chờ trong hệ thống (waiting time) của từng tiến trình trong từng thuật toán điều phối ở câu a.

d)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_1	0.0	8
P_2	0.4	4
P_3	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:

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

b) 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 ệ 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 vớ 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>>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?

Chương 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? Chương này sẽ giới thiệu nội dung về 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.

4.1. Liên lạc giữa các tiến trình

4.1.1. Nhu cầu liên lạc giữa các tiến trình

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

- *Chia sẻ thông tin*: nhiều tiến trình có thể cùng quan tâm đến những dữ liệu nào đó, do vậy hệ điều hành cần cung cấp một môi trường cho phép sự truy cập đồng thời đến các dữ liệu chung.

- *Hợp tác hoàn thành tác vụ*: đôi khi để đạt được một sự xử lý nhanh chóng, người ta phân chia một tác vụ thành các công việc nhỏ có thể tiến hành song song. Thường thì các công việc nhỏ này cần hợp tác với nhau để cùng hoàn thành tác vụ ban đầu, ví dụ dữ liệu kết xuất của tiến trình này lại là dữ liệu nhập cho tiến trình khác… Trong các trường hợp đó, hệ điều hành cần cung cấp cơ chế để các tiến trình có thể trao đổi thông tin với nhau.

4.1.2. Các vấn đề nảy sinh trong việc liên lạc giữa các tiến trình

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

- *Liên kết tường minh hay tiềm ẩn (explicit naming/implicit naming)*: tiến trình có cần phải biết tiến trình nào đang trao đổi hay chia sẻ thông tin với nó? Mối liên kết được gọi là tường minh khi được thiết lập rõ ràng, trực tiếp giữa các tiến trình, và là tiềm ẩn khi các tiến trình liên lạc với nhau thông qua một quy ước ngầm nào đó.

- *Liên lạc theo chế độ đồng bộ hay không đồng bộ (blocking / non-blocking)*: khi một tiến trình trao đổi thông tin với một tiến trình khác, các tiến trình có cần phải đợi cho thao tác liên lạc hoàn tất rồi mới tiếp tục các xử lý khác? Các tiến trình liên lạc theo cơ chế đồng bộ sẽ chờ nhau hoàn tất việc liên lạc, còn các tiến trình liên lạc theo cơ chế nonblocking thì không.

- *Liên lạc giữa các tiến trình trong hệ thống tập trung và hệ thống phân tán*: cơ chế liên lạc giữa các tiến trình trong cùng một máy tính có sự khác biệt với việc liên lạc giữa các tiến trình giữa những máy tính khác nhau?

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

4.2. Các cơ chế thông tin liên lạc

4.2.1. Tín hiệu (Signal)

***** *Giới thiệu:* Tín hiệu là một cơ chế phần mềm tương tự như các ngắt cứng tác động đến các tiến trình. Một tín hiệu được sử dụng để thông báo cho tiến trình về một sự kiện nào đó xảy ra. Có nhiều tín hiệu được định nghĩa, mỗi một tín hiệu có một ý nghĩa tương ứng với một sự kiện đặc trưng.

* Ví dụ : Một số tín hiệu của UNIX

Tín hiệu	Mô tả
SIGINT	Người dùng nhấn phím DEL để ngắt xử lý tiến trình
SIGQUIT	Yêu cầu thoát xử lý

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

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

Các tín hiệu được gởi đi bởi:

- Phần cứng (ví dụ lỗi do các phép tính số học).

- Hạt nhân hệ điều hành gởi đến một tiến trình (ví dụ lưu ý tiến trình khi có một thiết bị nhập/xuất tự do).

- Một tiến trình gởi đến một tiến trình khác (ví dụ tiến trình cha yêu cầu một tiến trình con kết thúc).

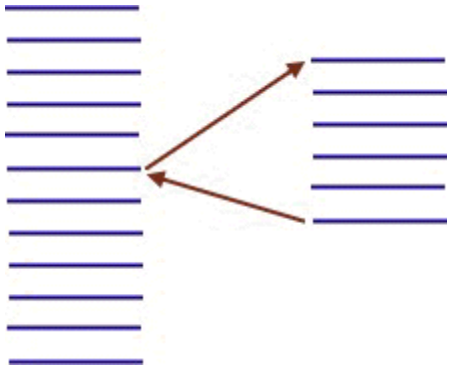
- Người dùng (ví dụ nhấn phím Ctl-C để ngắt xử lý của tiến trình)

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

- Bỏ qua tín hiệu.

- Xử lý tín hiệu theo kiểu mặc định.

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



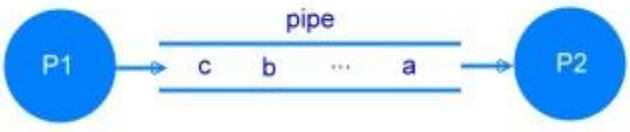
Hình 4.1. *Liên lạc bằng tín hiệu*

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

4.2.2. Pipe

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

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



Hình 4.2. *Liên lạc qua pipe*

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

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

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

***** *Thảo luận:* Liên lạc bằng pipe là một cơ chế liên lạc *một chiều (unidirectional)*, nghĩa là một tiến trình kết nối với một pipe chỉ có thể thực hiện một trong hai thao tác đọc hoặc ghi, nhưng không thể thực hiện cả hai. Một số hệ điều hành cho phép thiết lập hai pipe giữa một cặp tiến trình để tạo liên lạc hai chiều. Trong những hệ thống đó, có nguy cơ xảy ra tình trạng *tắc nghẽn* (deadlock): một pipe bị giới hạn về kích thước, do vậy nếu cả hai pipe nối kết hai tiến trình đều đầy (hoặc đều trống) và cả hai tiến trình đều muốn ghi (hay đọc) dữ liệu vào pipe (mỗi tiến trình ghi dữ liệu vào một pipe), chúng sẽ cùng bị khóa và chờ lẫn nhau mãi mãi.

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

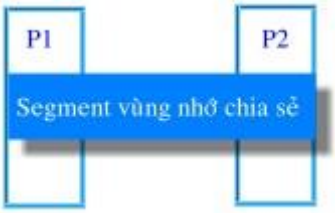
Ngoài ra, một giới hạn của hình thức liên lạc này là chỉ cho phép kết nối hai tiến trình có quan hệ cha-con, và trên cùng một máy tính.

4.2.3. Vùng nhớ chia sẻ

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

Với phương thức này, các tiến trình chia sẻ một vùng nhớ vật lý thông qua trung gian không gian địa chỉ của chúng. Một vùng nhớ chia sẻ tồn tại độc lập với các tiến trình, và khi một tiến trình muốn truy xuất đến vùng nhớ này,

tiến trình phải kết gắn vùng nhớ chung đó vào không gian địa chỉ riêng của từng tiến trình, và thao tác trên đó như một vùng nhớ riêng của mình.



Hình 4.3. *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 thờighi dữ liệu vào vùng nhớ chung? Rõ ràng vùng nhớ chia sẻ cần được bảo vệ bằng những cơ chế đồng bộ hóa thích hợp.

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

4.2.4. Trao đổi thông điệp (Message)

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

- *Send* (message): gởi một thông điệp

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

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

hay không… Nếu các tiến trình liên lạc theo kiểu liên kết tường minh, các hàm Send và Receive sẽ được cài đặt với tham số:

- *Send* (destination, message) : gởi một thông điệp đến *destination*

- *Receive* (source, message) : nhận một thông điệp từ *source*

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

4.2.5. Sockets

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

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

Các thuộc tính của socket:

- Domaine: định nghĩa dạng thức địa chỉ và các nghi thức sử dụng. Có nhiều domaines, ví dụ UNIX, INTERNET, XEROX_NS, ...

- Type: định nghĩa các đặc điểm liên lạc:

a) Sự tin cậy

b) Sự bảo toàn thứ tự dữ liệu

c) Lặp lại dữ liệu

d) Chế độ nối kết

e) Bảo toàn giới hạn thông điệp

f) Khả năng 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 gởi không chắc chắn thông điệp của học được gởi đến người nhận,

- Một thông điệp có thể được gởi nhiều lần;

- Hai thông điệp đượ gởi theo một thứ tự nào đó có thể đến tay người nhận theo một thứ tự khác.

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

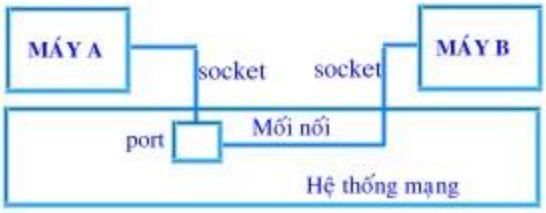
b) Liên lạc trong chế độ nối kết:

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

- Hủy một socket

Ví dụ :

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



Hình 4.4. *Các socket và port trong mối nối TCP.*

Hình 4.4 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 nầy với một port X (tức là một số nguyên dương có ý nghĩa cục bộ trong máy A), trong khi đó máy B tạo một socket khác và móc vào (connect) port X trong máy A.

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

4.3. Nhu cầu đồng bộ hóa (synchronisation)

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

4.3.1. Yêu cầu độc quyền truy xuất (Mutual exclusion)

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

- Đặc tính cấu tạo phần cứng của tài nguyên không cho phép chia sẻ.

- Nếu nhiều tiến trình sử dụng tài nguyên đồng thời, có nguy cơ xảy ra các kết quả không dự đoán được do hoạt động của các tiến trình trên tài nguyên ảnh hưởng lẫn nhau.

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

4.3.2. Yêu cầu phối hợp (Synchronization)

Nhìn chung, mối tương quan về tốc độ thực hiện của hai tiến trình trong hệ thống là không thể biết trước, vì điều này phụ thuộc vào nhiều yếu tố động như tần suất xảy ra các ngắt của từng tiến trình, thời gian tiến trình được cấp phát bộ xử lý… Có thể nói rằng các tiến trình hoạt động không đồng bộ với nhau. 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…

4.3.3. Bài toán đồng bộ hoá

4.3.3.1. Vấn đề tranh đoạt điều khiển (race condition)

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

```
if (taikhoan - tienrut >=0)
    taikhoan = taikhoan - tienrut;
else
    error(« khong the rut tien ! »);
```

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

- Sau khi đã kiểm tra điều kiện `(taikhoan - tienrut >=0)` và nhận kết quả là 300, P_1 hết thời gian xử lý mà hệ thống cho phép, hệ điều hành cấp phát CPU cho P_2.

- P_2 kiểm tra cùng điều kiện trên, nhận được kết quả là 400 (do P_1 vẫn chưa rút tiền) và rút 400. Giá trị của *taikhoan* được cập nhật lại là 400.

- Khi P_1 được tái kích hoạt và tiếp tục xử lý, nó sẽ không kiểm tra lại điều kiện `(taikhoan - tienrut >=0)` –vì đã kiểm tra trong lượt xử lý trước- mà thực hiện rút tiền. Giá trị của *taikhoan* sẽ lại được cập nhật thành -100. Tình huống lỗi xảy ra.

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

4.3.3.2. Miền găng (critical section)

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

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

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

của mỗi tiến trình tạo thành một miền găng.

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

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

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

- Không có giả thiết nào đặt ra cho sự liên hệ về tốc độ của các tiến trình, cũng như về số lượng bộ xử lý trong hệ thống.

- Một tiến trình tạm dừng bên ngoài miền găng không được ngăn cản các tiến trình khác vào miền găng.

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

4.4. 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 viêc 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 hoán 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_2O

Đồng bộ hoạt động của một phòng thí nghiệm sử dụng nhiều tiến trình đồng hành sau để tạo các phân tử H_2O:

```
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 sụp đổ, người ta chỉ có cho phép tối đa 3 xe lưu thông đồng thời qua một cây cầu rất cũ. Hãy xây dựng thủ tục **ArriveBridge(int direction)** và **ExitBridge()** 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 Microsof và các Linux hacker cùng sử dụng một bến sông và phải chia sẻ một số thuyền đặc biệt. Mỗi chiếc thuyền này chỉ cho phép chở 1 lần 4 người, và phải có đủ 4 người mới khởi hành được. Để bảo đảm an toàn cho cả 2 phía, cần tuân thủ các luật sau:

a. Không chấp nhận 3 nhân viên Microsoft và 1 Linux hacker trên cùng một chiếc thuyền.

b. Ngược lại, không chấp nhận 3 Linux hacker và 1 nhân viên Microsoft trên cùng một chiếc thuyền.

c. Tất cả các trường hợp kết hợp khác đều hợp pháp.

d. Thuyền chỉ khởihành khi đã có đủ 4 hành khách.

Cần xây dựng 2 thủ tục **HackerArrives()** và **EmployeeArrives()** được gọi tương ứng bởi 1 hacker hoặc 1 nhân viên khi họ đến bờ sông để kiểm tra điều kiện có cho phép họ xuống thuyền không? Các thủ tục này sẽ sắp xếp những người thích hợp có thể lên thuyền. Những người đã được lên thuyền khi

thuyền chưa đầy sẽ phải chờ đến khi người thứ 4 xuống thuyền mới có thể khởi hành qua sông. (Không quan tâm đến số lương thuyền hay việc thuyền qua sông rồi trở lại… Xem như luôn có thuyền để sắp xếp theo các yêu cầu hợp lệ).

Giả sử hoạt động của mỗi hacker được mô tả bằng một tiến trình **Hacker()** sau đây:

```
Hacker()
{
RuntoRiver(); // Đi đến bờ sông
HackerArrives (); // Kiểm tra điều kiện xuống thuyền
CrossRiver(); // Khởi hành qua sông
}
```

và hoạt động của mỗi nhân viên được mô tả bằng một tiến trình **Employee()** sau đây:

```
Employee()
  {
RuntoRiver(); // Đi đến bờ sông
EmployeeArrives (); // Kiểm tra điều kiện xuống thuyền
CrossRiver(); // Khởi hành qua sông
}
```

Chương 5

CÁC GIẢI PHÁP ĐỒNG BỘ HOÁ

Chương 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".

5.1. Giải pháp “busy waiting”

5.1.1. Các giải pháp phần mềm

5.1.1.1. 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 5.1. *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.

5.1.1.2. 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 5.2. *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 nhặt đến lượt tiến trình nào được vào miền găng. Do đó nó có thể ngăn chặn được tình trạng hai tiến trình cùng vào miền găng, nhưng lại có thể vi phạm

điều kiện thứ ba: một tiến trình có thể bị ngăn chặn vào miền găng bởi một tiến trình khác không ở trong miền găng. Giả sử tiến trình B ra khỏi miền găng rất nhanh chóng. Cả hai tiến trình đều ở ngoài miền găng, và turn = 0. Tiến trình A vào miền găng và ra khỏi nhanh chóng, đặt lại giá trị của turn 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.

5.1.1.3. 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 Pi muốn vào miền găng. Khởi đầu, *interesse[0]=interesse[1]=FALSE* và giá trị của est được khởi động là 0 hay 1. Để có thể vào được miền găng, trước tiên tiến trình Pi đặt giá trị *interesse[i]*=TRUE (xác định rằng tiến trình muốn vào miền găng), sau đó đặt turn=*j* (đề nghị thử tiến trình khác vào miền găng). Nếu tiến trình Pj không quan tâm đến việc vào miền găng (*interesse[j]=FALSE*), thì Pi có thể vào miền găng, nếu không, Pi phải chờ đến khi *interesse[j]=FALSE*. Khi tiến trình Pi rời khỏi miền găng, nó đặt lại giá trị cho *interesse[i]*= FALSE.

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

```
        Noncritical-section ();
        }
```

Hình 5.3. *Cấu trúc tiến trình Pi trong giải pháp Peterson*

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

5.1.2. Các giải pháp phần cứng

5.1.2.1. Cấm 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.

5.1.2.2. Chỉ thị TSL (Test-and-Set)

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

```
Test-and-Setlock(boolean target)
{
Test-and-Setlock = target;
target = TRUE;
```

```
}
```

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

```
while (TRUE) {
while (Test-and-Setlock(lock));
critical-section ();
lock = FALSE;
Noncritical-section ();
}
```

Hình 5.4. *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*".

5.2. Các giải pháp “SLEEP and WAKEUP”

Để loại bỏ các bất tiện của giải pháp "busy waiting", chúng ta có thể tiếp cận theo hướng cho một tiến trình chưa đủ điều kiện vào miền găng chuyển sang trạng thái blocked, từ bỏ quyền sử dụng CPU. Để thực hiện điều này, cần phải sử dụng các thủ tục do hệ điều hành cung cấp để thay đổi trạng thái tiến

trình. Hai thủ tục cơ bản *SLEEP* và *WAKEUP* thường được sử dụng để phục vụ mục đích này.

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

Ý tưởng sử dụng SLEEP và WAKEUP như sau: khi một tiến trình chưa đủ điều kiện vào miền găng, nó gọi *SLEEP* để tự khóa đến khi có một tiến trình khác gọi *WAKEUP* để giải phóng cho nó. Một tiến trình gọi *WAKEUP* khi ra khỏi miền găng để đánh thức một tiến trình đang chờ, tạo cơ hội cho tiến trình này vào miền găng:

```
int busy;  // 1 nếu miền găng đang bị chiếm, nếu không là 0
int blocked;  // đếm số lượng tiến trình đang bị khóa
while (TRUE) {
if (busy){
    blocked = blocked + 1;
    sleep();
}
else busy = 1;
critical-section ();
busy = 0;
if(blocked){
wakeup(process);
    blocked = blocked - 1;
}
Noncritical-section ();
}
```

Hình 5.5. *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 goi *SLEEP* và tự khó vĩnh viễn.

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

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

5.2.1. Semaphore

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

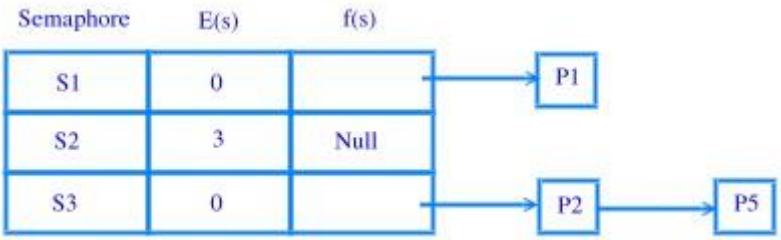
- Một giá trị nguyên dương *e(s).*

- Một hàng đợi *f(s)* lưu danh sách các tiến trình đang bị khóa (chờ) trên semaphore *s.*

- Chỉ có hai thao tác được định nghĩa trên semaphore.

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

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



Hình 5.6. *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 5.7. *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 5.8. *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.

5.2.2. Monitors

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

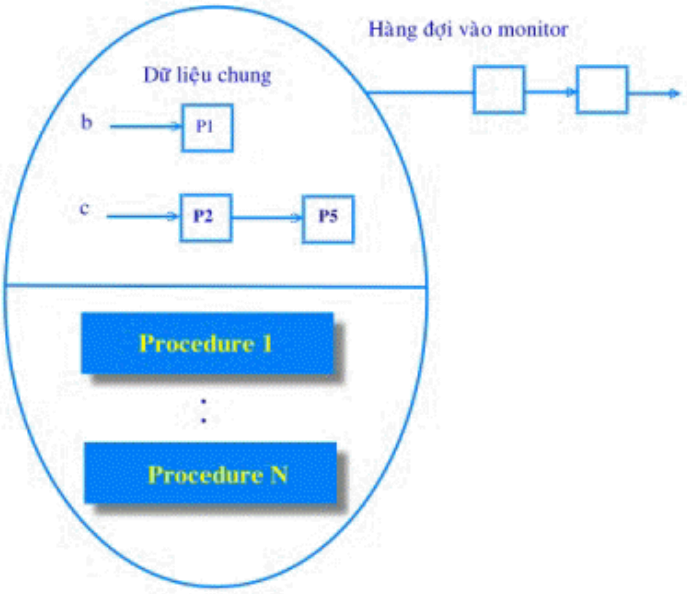
- Các biến và cấu trúc dữ liệu bên trong monitor chỉ có thể được thao tác bởi các thủ tục định nghĩa bên trong monitor đó. (*encapsulation*).

- Tại một thời điểm, chỉ có một tiến trình duy nhất được hoạt động bên trong một monitor (*mutual exclusive*).

- Trong một monitor, có thể định nghĩa các *biến điều kiện* và hai thao tác kèm theo là **`Wait`** và **`Signal`** như sau: gọi ***c*** là biến điều kiện được định nghĩa trong monitor:

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

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



Hình 5.9. *Monitor và các biến điều kiện*

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

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

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

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

```
procedure Action1();
{
}                    ....
procedure Actionn();
{
} end monitor;
```

Hình 5.10. *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 5.11. *Cấu trúc tiến trình Pi 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.

5.2.3. Trao đổi thông điệp

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

- *Send(destination, message):* 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 gởi một thông điệp khác đến tiến trình kiểm soát để báo kết thúc truy xuất. Về phần tiến trình kiểm soát, khi nhận được thông điệp yêu cầu tài nguyên, nó sẽ chờ đến khi tài nguyên sẵn sàng để cấp phát thì gởi một thông điệp đến tiến trình đang bị khóa trên tài nguyên đó để đánh thức tiến trình này.

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

Hình 3.16. *Cấu trúc tiến trình*

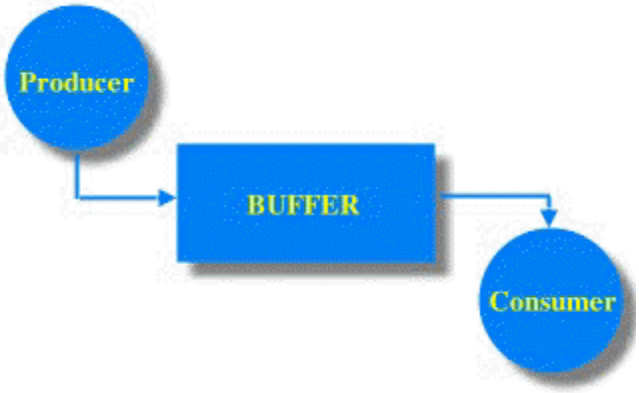
yêu cầu tài nguyên trong giải pháp message

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.

5.3. Vấn đề đồng bộ hoá

5.3.1. 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 5.12 *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:

5.3.1.1. 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
 }
}
```

5.3.1.2. 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);
   }
  }
```

5.3.1.3. 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 gởi 4 thông điệp *empty* đấng 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);                 //   gởi   dữ   liệu   đến
Consumer
   }
}

Consumer()
{
    int item;
    message m;

    for(0 to N)
             send(producer, &m);  // gởi 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);        // gởi thông điệp empty
      consumer_item(item);        // xử lý dữ liệu
    }
}
```

5.3.2. 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 sỡ 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*:

5.3.2.1. 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
  }
}
```

5.3.2.2. 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();
      }
   }
```

5.3.2.3. 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 gửi 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); }
}
```

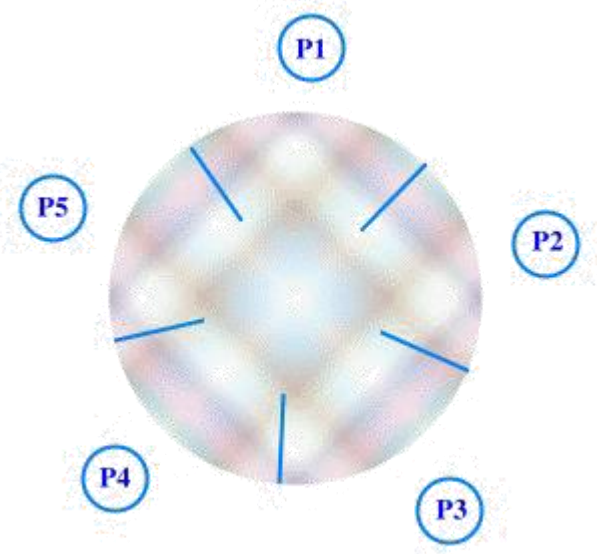
5.4. Tắc nghẽn (Deadlock)

5.4.1. 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 dĩa để có thể ăn spaghetti. Nhưng trên bàn chỉ có tổng cộng 5 cái dĩa 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 dĩa bên trái và 1 cái dĩa bên phải để ăn. Nếu cả 5 nhà triết học đều cầm cái dĩa bên trái cùng lúc, thì sẽ không có ai có được cái dĩa bên phải để có thể bắt đầu thưởng thức spaghetti. Đây chính là tình trạng *tắc nghẽn*.



Hình 5.13. *Bữa ăn tối của các triết gia*

5.4.2. 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.

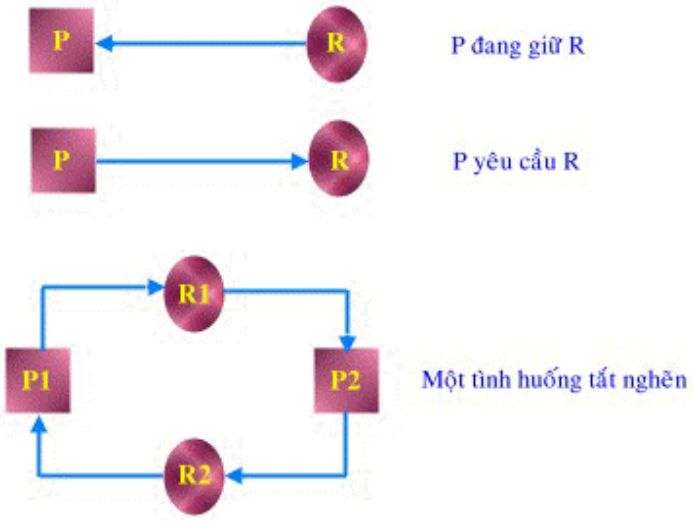
* Không thu hồi tài nguyên từ tiến trình đang giữ chúng (No preemption): Tài nguyên không thể được thu hồi từ tiến trình đang chiếm giữ chúng trước khi tiến trình này sử dụng chúng xong.

* 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, tắc nghẽn sẽ xảy ra. Nếu thiếu một trong 4 điều kiện trên thì không có tắc nghẽn.

5.4.3. 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 5.14 *Đồ thị cấp phát tài nguyên*

5.4.4. Các phương pháp xử lý tắc nghẽn

Chủ yếu có ba hương 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 sữa chữa tắc nghẽn.

- Hoàn toàn bỏ qua việc xử lý tắc nghẽn, xem như hệ thống không bao giờ xảy ra tắc nghẽn.

5.4.5. 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 khóa. 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,...,Rm} là tập các loại tài nguyên.

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

Ví dụ : F(đĩa) = 2, F(máy in) = 12

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

5.4.6. 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 <P_1, P_2,...,Pn> là an toàn đối với tình trạng cấp phát hiện hành nếu với mỗi tiến trình Pi nhu cầu tài nguyên của Pi 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 Pj 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 thõa mãn yêu cầu của P2, ta có:

Need			Allocation			Available		
R1	R2	R3	R1	R2	R3	R1	R2	R3
P1	2	2	2	1	0	0		

P2	0	0	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 Pi yêu cầu *k* thể hiện của tài nguyên *r*.

1. Nếu k <= Need[i], đến bước 2

   Ngược lại, xảy ra tình huống lỗi

2. Nếu k <= Available[i],đến bước 3

   Ngược lại, Pi phải chờ

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

Available[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 Pi.

Ngược lại, Pi phải chờ.

5.4.7. 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.

5.4.8. 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?

5.5. 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 găng ("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**

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 giaỉ quyết nhu cầu phối hợp hoạt động.

*** Bài tập**

Bài 1. Xét giải pháp phần mềm do Dekker đề nghị để tổ chức truy xất 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 Pi (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 xất 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 <= nb +10*

b) Nếu giảm điều kiện chỉ là *na <= nb +10*, giải pháp của bạn sẽ được sửa chữa như thế nào?

c) Giải pháp của bạn có còn đúng nếu có nhiều tiến trình loại A và B cùng thực hiện?

Bài 8. 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 ≤ k ≤ 100), `Ak` chỉ có thể bắt đầu khi `B(k-1)` đã kết thúc, và `Bk` chỉ có thể bắt đầu khi `A(k-1)` đã kết thúc.

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 databas, 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 dĩa, và nếu có 2 triết gia cùng muốn lấy một cái dĩa, thì chỉ một người được quyền lấy cái dĩa đó. Sử dụng semaphore để tổ chức độc quyền truy xuất đến các cái dĩa cho đoạn chương trình trên (Gợi ý: dùng mỗi semaphore phản ánh tình trạng sử dụng của mỗi cái dĩa)

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)
{ thinhk(); // 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. *Baber 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 kh1ch 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 Baber 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 thông có ở trạng thái an toàn không?

c) Nếu tiến trình P2 có yêu cầu tài nguyên (0,4,2,0), yêu cầu này có được đáp ứng tức thời không?

Chương 6

QUẢN LÝ BỘ NHỚ

Chương này sẽ giới thiệu 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 phân tích ưu, khuyết điểm để 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 logic (symbolic) trong chương trình thành một địa chỉ thực (vật lý) trong bộ nhớ chính?

- *Quản lý bộ nhớ vật lý*: làm cách nào để mở rộng bộ nhớ có sẵn nhằm lưu trữ được nhiều tiến trình đồng thời?

- *Chia sẻ thông tin*: làm thế nào để cho phép hai tiến trình có thể chia sẻ thông tin trong bộ nhớ?

- *Bảo vệ:* làm thế nào để ngăn chặn các tiến trình xâm phạm đến vùng nhớ được cấp phát cho tiến trình khác?

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

6.1. 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 phần cứng đặc biệt.

6.2. 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 qủa là đưa ra khái niệm không gian địa chỉ được xây dựng trên không gian nhớ vật lý, việc tách rời hai không gian này giúp hệ điều hành dễ dàng xây dựng các cơ chế và chiến lược quản lý bộ nhớ hữu hiệu:

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

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

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

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

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

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

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

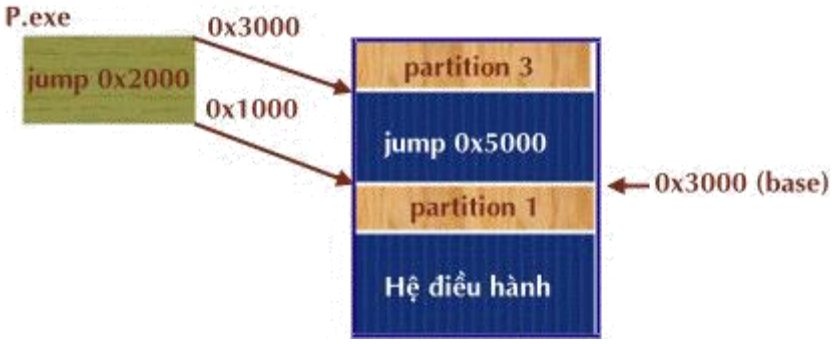
6.3. Cấp phát liên tục

6.3.1. Mô hình Linker_Loader

* *Ý tưởng*: Tiến trình được nạp vào một vùng nhớ liên tục đủ lớn để chứa toàn bộ tiến trình. Tại thời điểm biên dịch các địa chỉ bên trong tiến trình vẫn là

địa chỉ tương đối. Tại thời điểm nạp, Hệ điều hành sẽ trả về địa chỉ bắt đầu nạp tiến trình, và tính toán để chuyển các địa chỉ tương đối về địa chỉ tuyệt đối trong bộ nhớ vật lý theo công thức

địa chỉ vật lý = địa chỉ bắt đầu + địa chỉ tương đối.



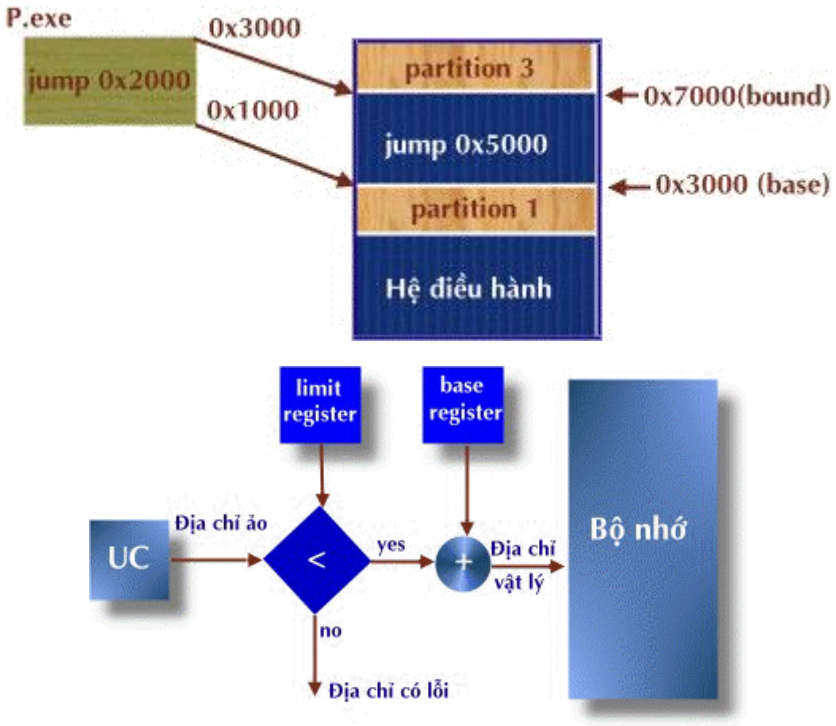
** Thảo luận:*

- Thời điểm kết buộc địa chỉ là thời điểm nạp, do vậy sau khi nạp không thể dịch chuyển tiến trình trong bộ nhớ.

- Không có khả năng kiểm soát địa chỉ các tiến trình truy cập, do vậy không có sự bảo vệ.

6.3.2. Mô hình Base &Bound

** Ý tưởng*: Tiến trình được nạp vào một vùng nhớ liên tục đủ lớn để chứa toàn bộ tiến trình. Tại thời điểm biên dịch các địa chỉ bên trong tiến trình vẫn là địa chỉ tương đối. Tuy nhiên bổ túc vào cấu trúc phần cứng của máy tính một thanh ghi cơ sở (*base register*) và một thanh ghi giới hạn (*bound register*). Khi một tiến trình được cấp phát vùng nhớ, nạp vào thanh ghi cơ sở 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 cơ sở 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.



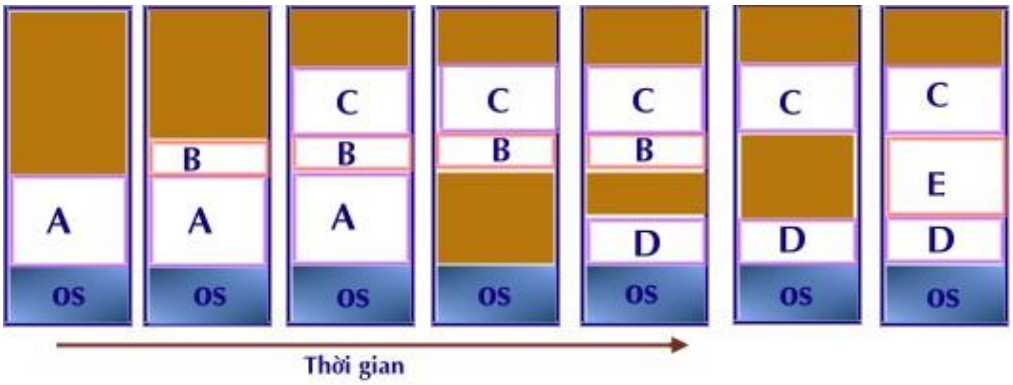
Hình 6.1. *Hai thanh ghi hỗ trợ chuyển đổi địa chỉ*

** Thảo luận:*

- Một ưu điểm của việc sử dụng thanh ghi cơ sở là có thể di chuyển các chương trình trong bộ nhớ sau khi chúng bắt đầu được xử lý, mỗi khi tiến trình được di chuyển đến một vị trí mới, chỉ cần nạp lại giá trị cho thanh ghi cơ sở, các địa chỉ tuyệt đối sẽ được phát sinh lại mà không cần cập nhật các địa chỉ tương đối trong chương trình.

- Có 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 thoả mãn yêu cầu, nhưng các vùng nhớ này lại không liên tục. Người ta có thể áp dụng kỹ thuật "dồn bộ nhớ" (*memory compaction*) để kết hợp các mảnh bộ nhớ nhỏ rời rạc thành một vùng nhớ lớn liên tục. Tuy nhiên, kỹ thuật này đòi hỏi nhiều thời gian xử lý, ngoài ra, sự kết buộc địa chỉ

phải thực hiện vào thời điểm xử lý, vì các tiến trình có thể bị di chuyển trong quá trình dồn bộ nhớ.



Hình 6.2. *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:

+ Di chuyển tiến trình_: di chuyển tiến trình đến một vùng nhớ khác đủ lớn để thỏa mãn nhu cầu tăng trưởng kích thước của tiến trình.

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

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

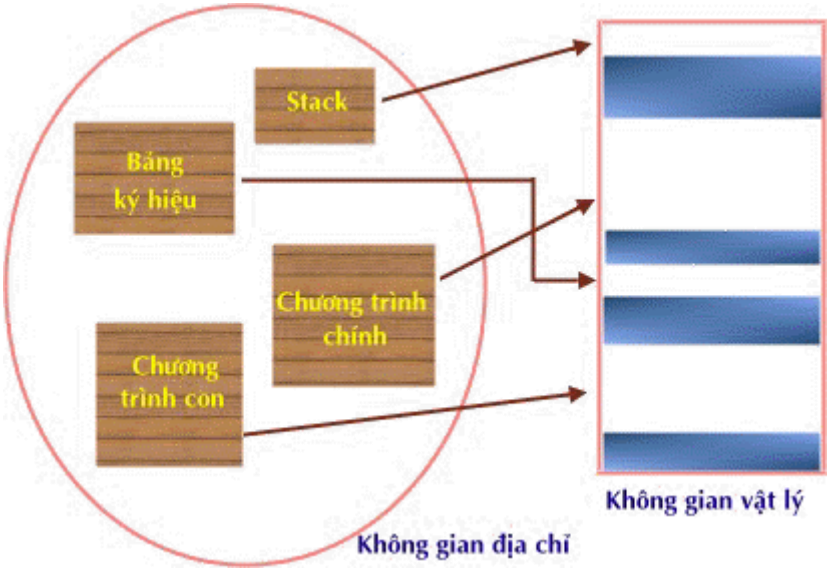
⚠ Các cách tổ chức bộ nhớ trên đây đều phải chịu đựng tình trạng bộ nhớ bị phân mảnh vì chúng đều tiếp cận theo kiểu cấp phát một vùng nhớ liên tục cho tiến trình. Như đã thảo luận, có thể sử dụng kỹ thuật dồn bộ nhớ để loại bỏ sự phân mảnh ngoại vi, nhưng chi phí thực hiện rất cao. Một giải pháp khác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

tục, nghĩa là có thể cấp phát cho tiến trình những vùng nhớ tự do bất kỳ, không cần liên tục.

6.4. Cấp phát không liên tục

6.4.1. Phân đoạn (Segmentation)

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



Hình 6.3. *Mô hình phân đoạn bộ nhớ*

* *Cơ chế MMU trong kỹ thuật phân đoạn:*

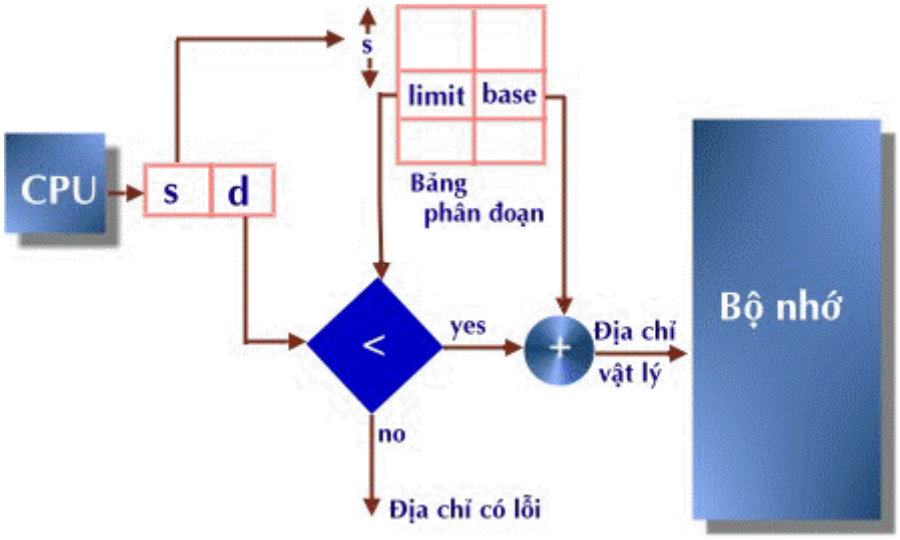
Cần phải xây dựng một ánh xạ để chuyển đổi các địa chỉ 2 chiều được người dùng định nghĩa thành địa chỉ vật lý một chiều. Sự chuyển đổi này được thực hiện qua một *bảng phân đoạn*. Mỗi thành phần trong bảng phân đoạn bao gồm một *thanh ghi cơ sở* và một *thanh ghi giới hạn*. Thanh ghi cơ sở lưu trữ địa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

* *Chuyển đổi địa chỉ:*

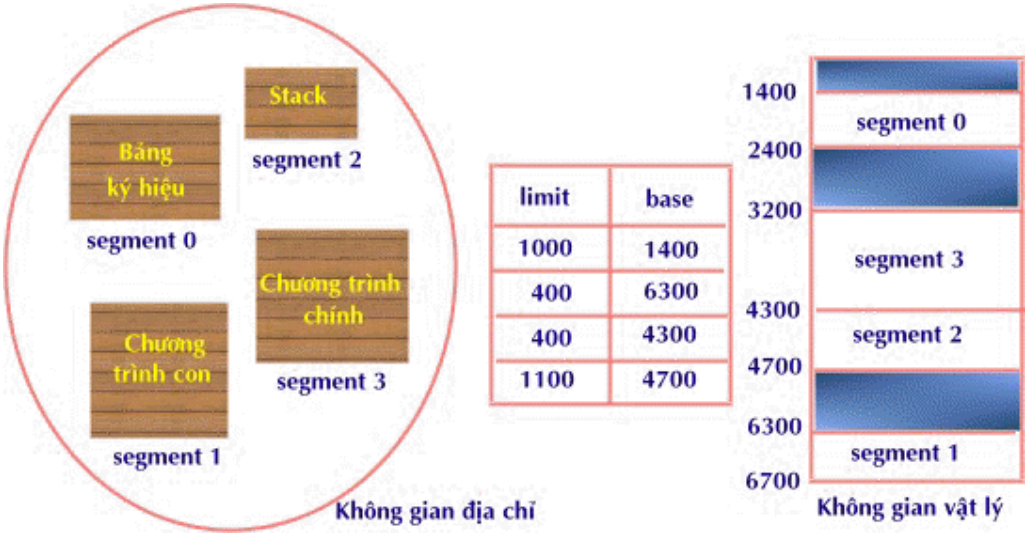
Mỗi địa chỉ ảo là một bộ <*s*,*d*> *:*

- *Số hiệu phân đoạn s*: được sử dụng như chỉ mục đến bảng phân đoạn

- *Địa chỉ tương đối d*: có giá trị trong khoảng từ 0 đến giới hạn chiều dài của phân đoạn. Nếu địa chỉ tương đối hợp lệ, nó sẽ được cộng với giá trị chứa trong thanh ghi cơ sở để phát sinh địa chỉ vật lý tương ứng.



Hình 6.4. *Cơ chế phần cứng hổ trợ kĩ thuật phân đoạn*

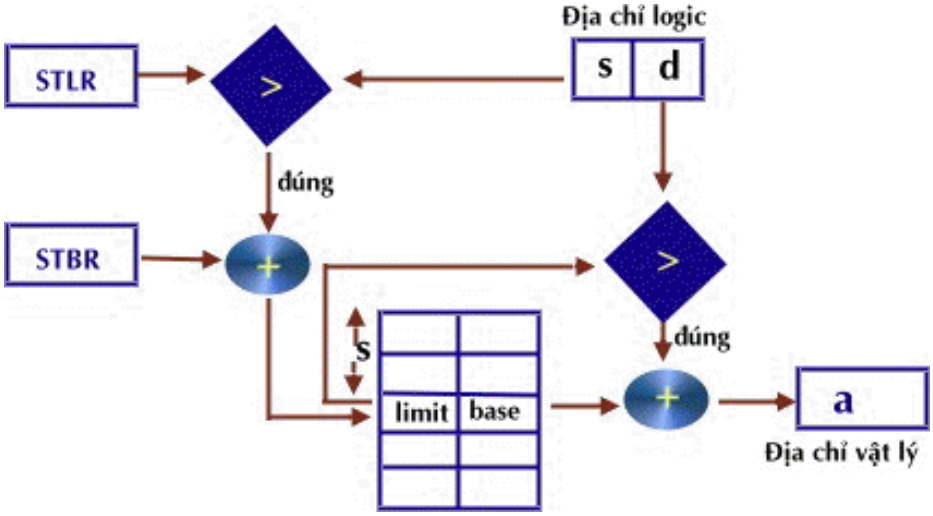


Hình 6.5. *Hệ thống phân đoạn*

** 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. Việt quảng lý bảng phân đoạn thông qua *thanh ghi cơ sở của bảng phân đoạn* (STBR) chỉ đến địa chỉ bắt đầu của bảng phân đoạn. Vì số lượng phân đoạn sử dụng trong một chương trình biến động, cần sử dụng thêm một *thanh ghi đặc tả kích thước bảng phân đoạn* (STLR).

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

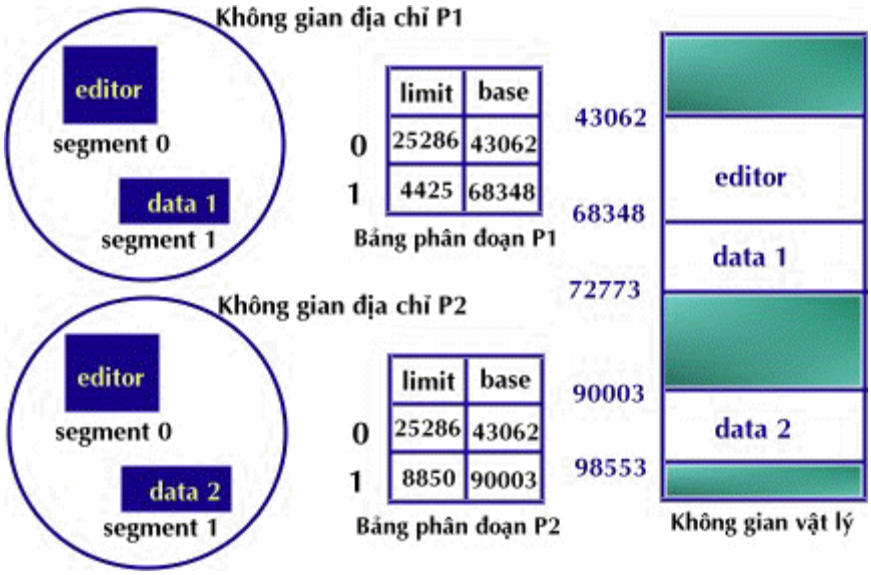


Hình 6.6. *Sử dụng STBR, STLR và bảng phân đoạn*

** 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 mục đích được người dùng xác định, người sử dụng có thể biết được một phân đoạn chứa đựng những gì bên trong, do vậy họ có thể đặc tả các thuộc tính bảo vệ thích hợp cho từng phân đoạn.

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

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



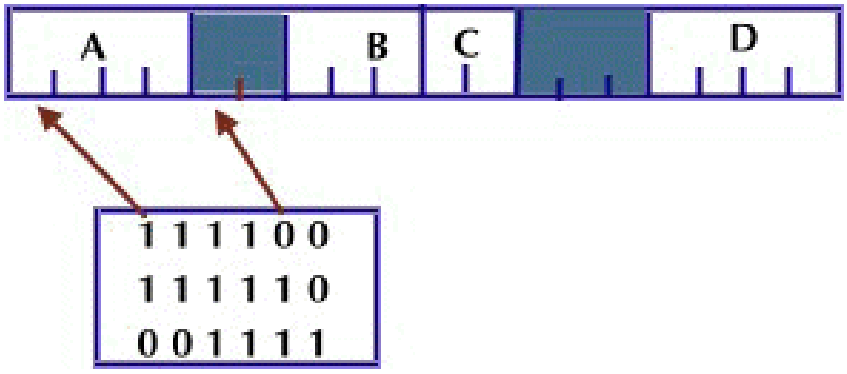
Hình 6.7. *Chia sẻ code trong hệ phân đoạn*

* *Thảo luận*:

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

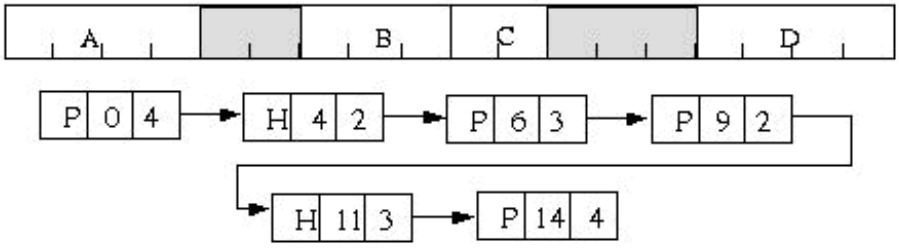
+ *Quản lý bằng một bảng các bit*: Bộ nhớ được chia thành các đơn vị cấp phát, mỗi đơn vị được phản ánh bằng một bit trong bảng các bit, một bit nhận

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



Hình 6.8. *Quản lý bộ nhớ bằng bảng các bit*

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



Hình 6.9. *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.

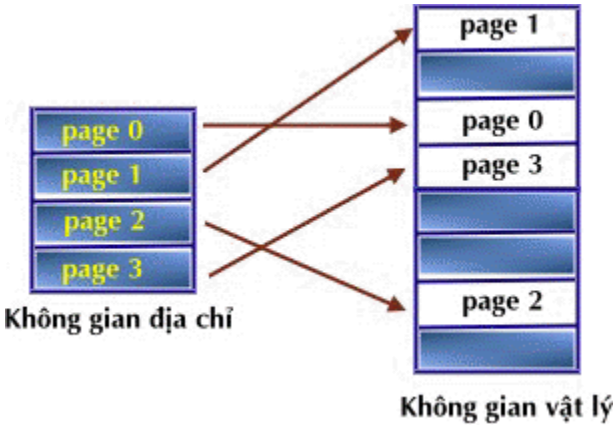
Worst-fit: cấp phát phân đoạn tự do lớn nhất.

- 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.

6.4.2. Phân trang (Paging)

** Ý tưởng:*

Phân bộ nhớ vật lý thành các khối (block) có kích thước cố định và bằng nhau, gọi là *khung trang (page frame)*. Không gian địa chỉ cũng được chia thành các khối có cùng kích thước với khung trang, và được gọi là *trang* (*page*). Khi cần nạp một tiến trình để xử lý, các trang của tiến trình sẽ được nạp vào những khung trang còn trống. Một tiến trình kích thước N trang sẽ yêu cầu N khung trang tự do.



Hình 6.10. *Mô hình bộ nhớ phân trang*

** Cơ chế MMU trong kỹ thuật phân trang:*

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

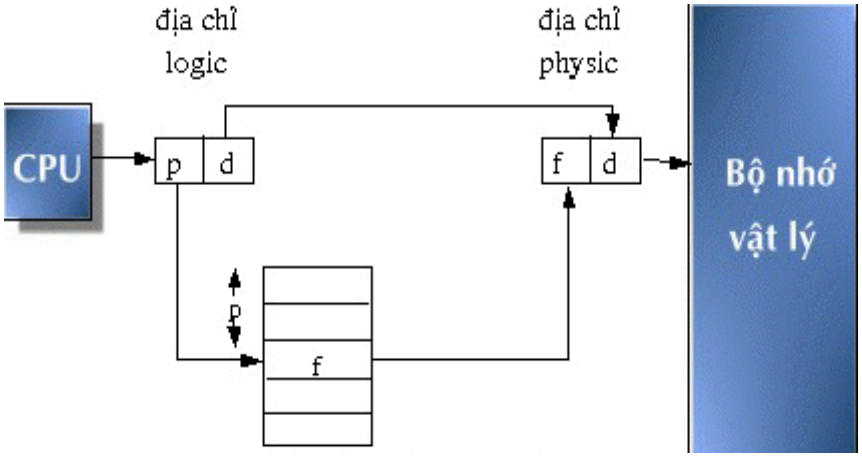
** Chuyển đổi địa chỉ:*

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

- *Số hiệu trang (p)*: sử dụng như chỉ mục đến phần tử tương ứng trong bảng trang.

- *Địa chỉ tương đối trong trang (d)*: kết hợp với địa chỉ bắt đầu của trang để tạo ra địa chỉ vật lý mà trình quản lý bộ nhớ sử dụng.

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

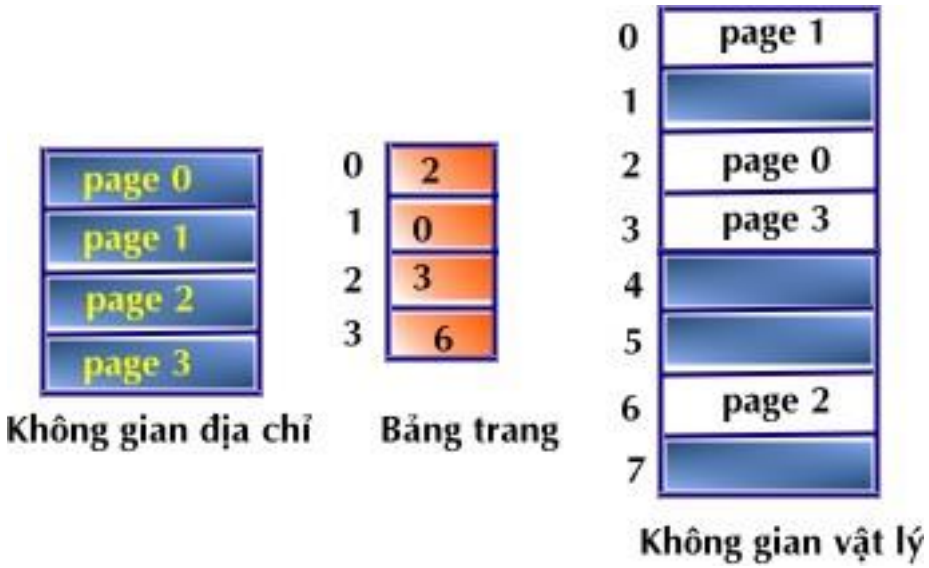


Hình 6.11. *Cơ chế phần cứng hỗ trợ phân trang*

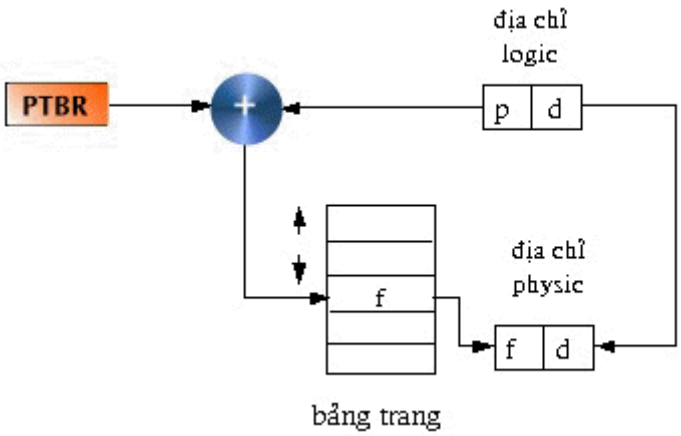
** Cài đặt bảng trang:*

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

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



Hình 6.12. *Mô hình bộ nhớ phân trang*

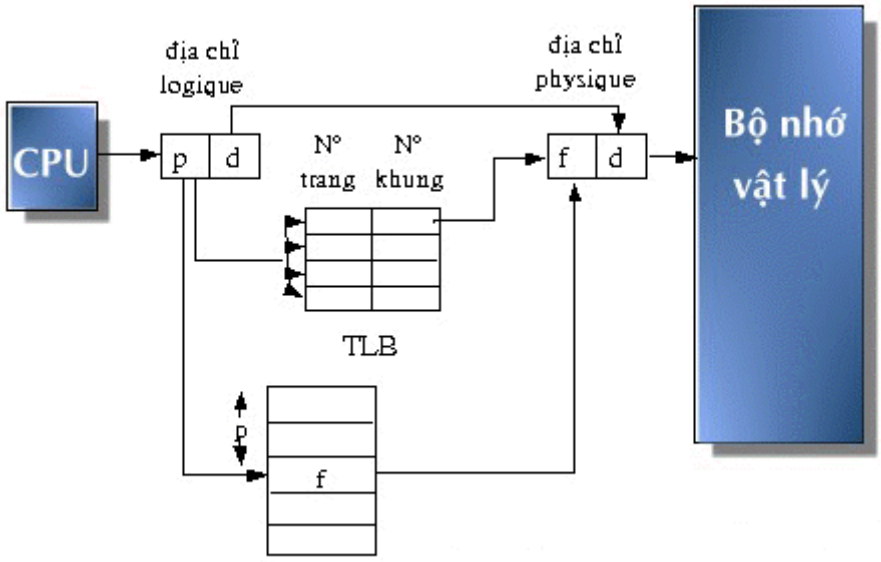


Hình 6.13. *Sử dụng thanh ghi cơ sở trỏ đến bảng trang*

⚠ Có thể né tránh bớt việc truy xuất bộ nhớ hai lần bằng cách sử dụng thêm một vùng nhớ đặc biệt, với tốc độ truy xuất nhanh và cho phép tìm kiếm song song, vùng nhớ cache nhỏ này thường được gọi là bộ nhớ kết hợp (TLBs). Mỗi thanh ghi trong bộ nhớ kết hợp gồm một từ khóa và một giá trị, khi đưa đến

bộ nhớ kết hợp một đối tượng cần tìm, đối tượng này sẽ được so sánh cùng lúc với các từ khóa trong bộ nhớ kết hợp để tìm ra phần tử tương ứng. Nhờ đặc tính này mà việc tìm kiếm trên bộ nhớ kết hợp được thực hiện rất nhanh, nhưng chi phí phần cứng lại cao.

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

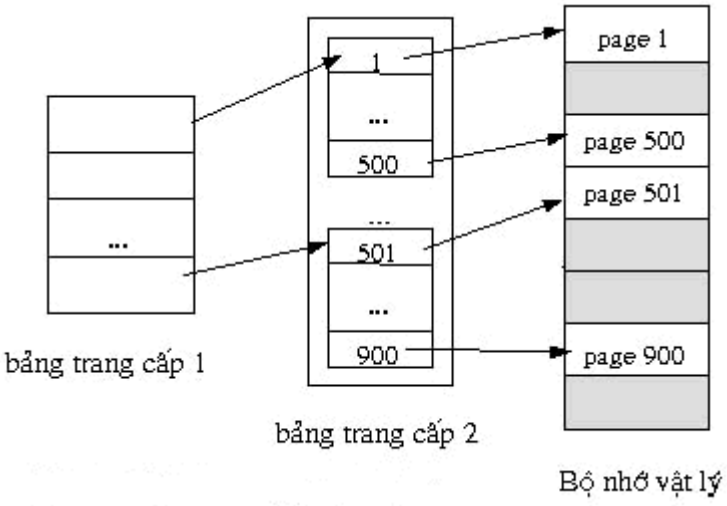


Hình 6.14. *Bảng trang với TLBs*

** 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.



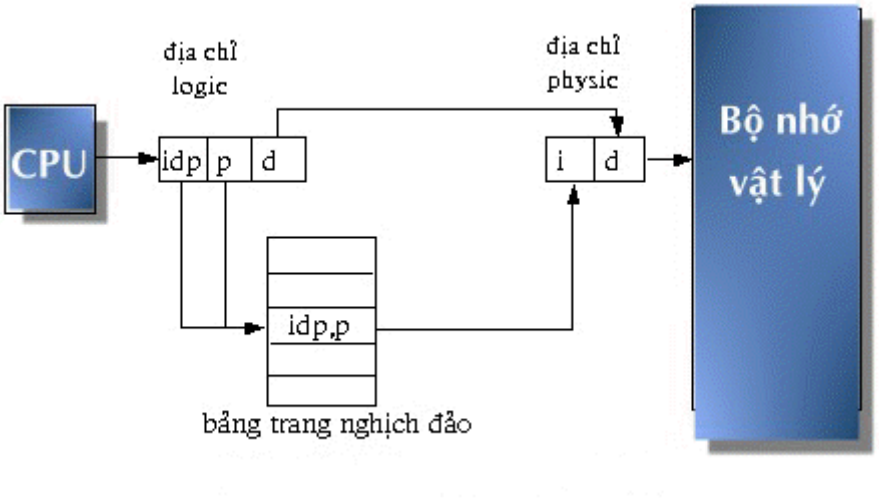
Hình 6.15. *Bảng trang nhị cấp*

- Bảng trang nghịch đảo:

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

Trong đó: idp là định danh của tiến trình, p là số hiệu trang và d là địa chỉ tương đối trong trang.

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



Hình 6.16. *Bảng trang nghịch đảo*

** Bảo vệ:*

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

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

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

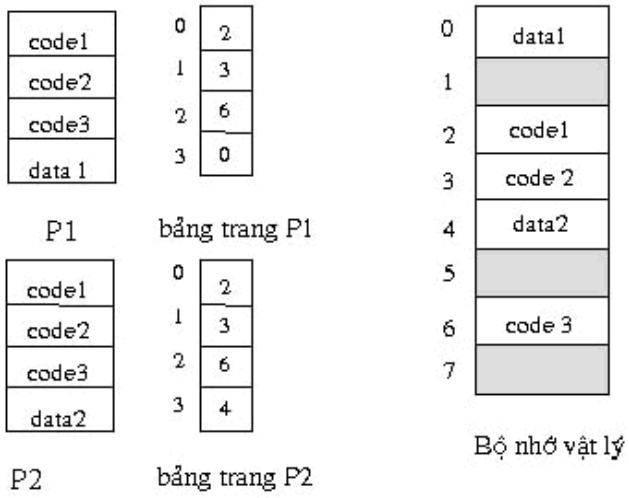
- *Bất hợp lệ*: trang tương ứng không nằm trong không gian địa chỉ của tiến trình, điều này có nghĩa tiến trình đã truy xuất đến một địa chỉ không được phép.

số hiệu khung trang	bit valid-invalid

Hình 6.17 *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 6.18. *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*.

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

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

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

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

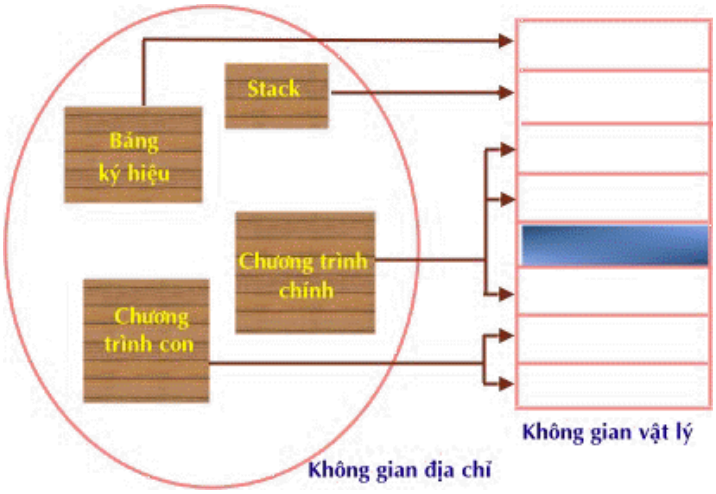
* *Chuyển đổi địa chỉ:*

Mỗi địa chỉ logic là một bộ ba: <*s,p,d*>

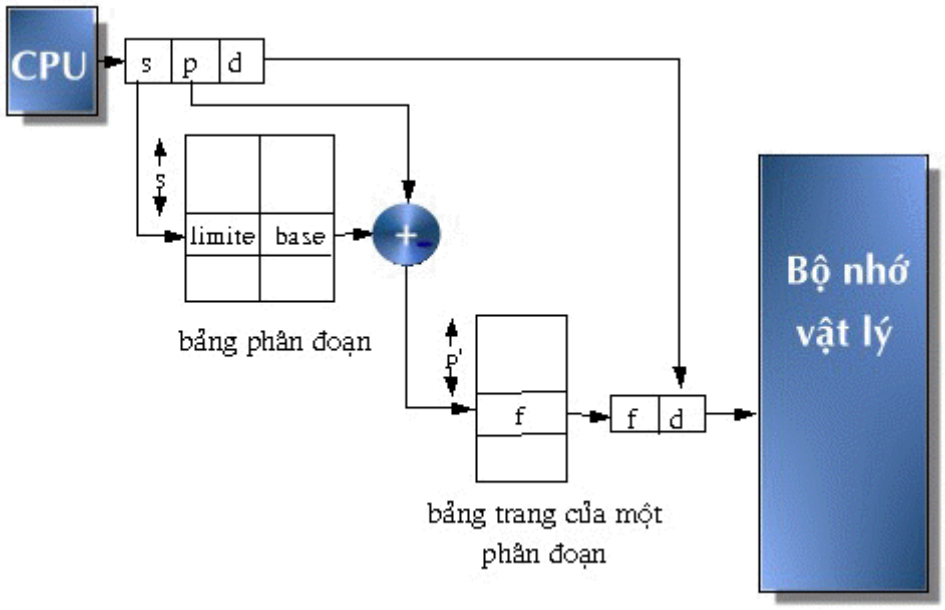
- *Số hiệu phân đoạn (s)*: sử dụng như chỉ mục đến phần tử tương ứng trong bảng phân đoạn.

- *Số hiệu trang (p)*: sử dụng như chỉ mục đến phần tử tương ứng trong bảng trang của phân đoạn.

- *Địa chỉ tương đối trong trang (d):* kết hợp với địa chỉ bắt đầu của trang để tạo ra địa chỉ vật lý mà trình quản lý bộ nhớ sử dụng.



Hình 6.19. *Mô hình phân đoạn kết hợp phân trang*



Hình 6.20. *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.

6.5. Tóm tắt

- Có nhiều cách tiếp cận khác nhau để tổ chức quãn lý bộ nhớ, nhưng đều có điểm 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 dược 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 (vật lý)?

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 qủa 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 qủa gì? Giải thích làm cách nào hiệu qủa 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

Chương 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 nội dung chương này.

7.1. Giới thiệu

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).*

7.1.1. Định nghĩa

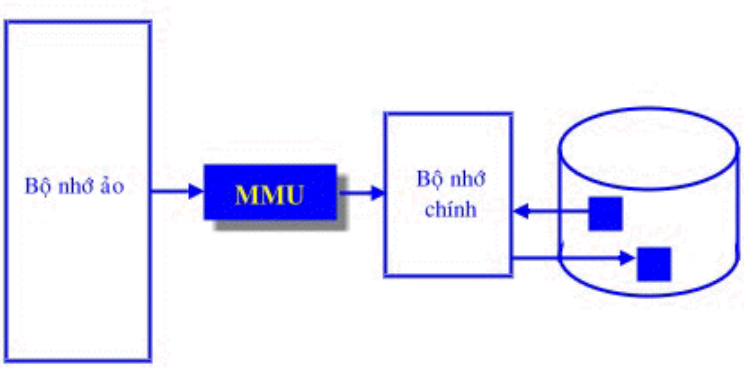
Bộ nhớ ảo là một kỹ thuật cho phép xử lý một tiến trình không được nạp toàn bộ vào bộ nhớ vật lý. Bộ nhớ ảo mô hình hoá bộ nhớ như một bảng lưu trữ rất lớn và đồng nhất, tách biệt hẳn khái niệm không gian địa chỉ và không gian vật lý. Người sử dụng chỉ nhìn thấy và làm việc trong không gian địa chỉ ảo,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

** Thảo luận:*

- Cần kết hợp kỹ thuật *swapping* đển chuyển các phần của chương trình vào-ra giữa bộ nhớ chính và bộ nhớ phụ khi cần thiết.

- Nhờ việc tách biệt bộ nhớ ảo và bộ nhớ vật lý, có thể tổ chức một bộ nhớ ảo có kích thước lớn hơn bộ nhớ vật lý.

- Bộ nhớ ảo cho phép giảm nhẹ công việc của lập trình viên vì họ không cần bận tâm đến giới hạn của vùng nhớ vật lý, cũng như không cần tổ chức chương trình theo cấu trúc overlays.



Hình 7.1. *Bộ nhớ ảo*

7.1.2. Cài đặt bộ nhớ ảo

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

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

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

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

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

- *invalid*: hoặc trang bất hợp lệ (không thuộc về không gian địa chỉ của tiến trình) hoặc trang hợp lệ nhưng đang được lưu trên bộ nhớ phụ.

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

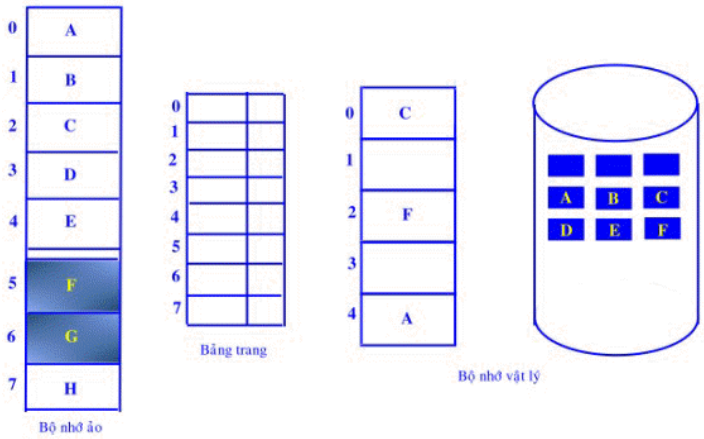
** Cơ chế phần cứng:*

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

- *Bảng trang*: Cấu trúc bảng trang phải cho phép phản ánh tình trạng của một trang là đang nằm trong bộ nhớ chính hay bộ nhớ phụ.

- *Bộ nhớ phụ*: Bộ nhớ phụ lưu trữ những trang không được nạp vào bộ nhớ chính. Bộ nhớ phụ thường được sử dụng là đĩa, và vùng không gian đĩa

dùng để lưu trữ tạm các trang trong kỹ thuật swapping được gọi là *không gian swapping.*



Bảng 7.2. *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.

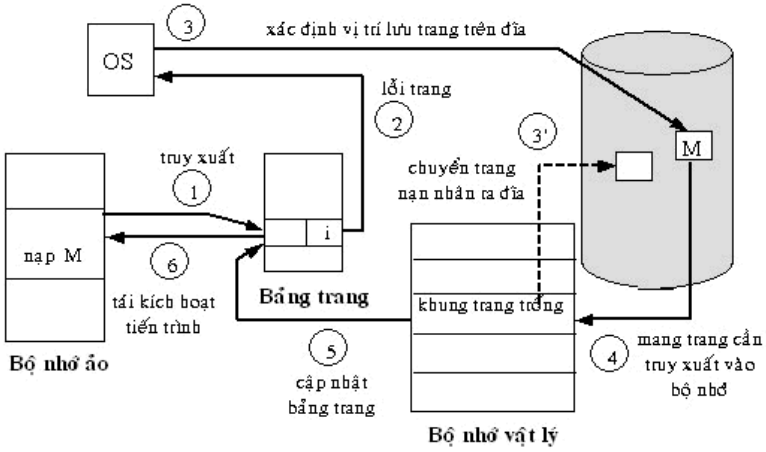
- Tìm một khung trang trống trong bộ nhớ chính:

+ 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 7.3. *Các giai đoạn xử lý lỗi trang*

7.2. Thay thế trang

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

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

số hiệu trang	valid-invalid bit	dirty bit

Hình 7.4. *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.

7.2.1. Sự thi hành phân trang theo yêu cầu

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

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*.

7.2.2. Các thuật toán thay thế trang

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

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

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

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

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

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

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

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

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

7.2.2.1. 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 qủa tốt : trang được chọn để thay thế có thể là trang chức nhiều dữ liệu cần thiết, thường xuyên được sử dụng nên được nạp sớm, do vậy khi bị chuyển ra bộ nhớ phụ sẽ nhanh chóng gây ra lỗi trang.

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

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

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

1	2	3	4	1	2	5	1	2	3	4	5
1	1	1	4	4	4	5	5	5	5	5	5
	2	2	2	1	1	1	1	1	3	3	3
		3	3	3	2	2	2	2	2	4	4
*	*	*	*	*	*	*			*	*	

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

1	2	3	4	1	2	5	1	2	3	4	5
1	1	1	1	1	1	5	5	5	5	4	4
	2	2	2	2	2	2	1	1	1	1	5
		3	3	3	3	3	3	2	2	2	2
			4	4	4	4	4	4	3	3	3
*	*	*	*			*	*	*	*	*	*

7.2.2.2. 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.

7.2.2.3. 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.

7.2.2.4. 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 reference bit, được khởi gán là 0, được gắn với một phần tử trong bảng trang.

- Reference bit 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 quy định trước.

- Sau từng chu kỳ quy định trước, kiểm tra giá trị của các reference bit, có thể xác định được trang nào đã được truy xuất đến và trang nào không, sau khi đã kiểm tra xong, các reference bit được phần cứng gán trở về 0.

- Với reference bit, 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	valid-invalid bit	dirty bit	reference bit bit

Hình 7.5. *Cấu trúc một phần tử trong bảng trang*

a) Thuật toán với các reference bit phụ trợ

* Tiếp cận: Có thể thu thập thêm nhiều thông tin về thứ tự truy xuất hơn bằng cách lưu trữ các reference bits 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 millisecondes), một ngắt đồng hồ được phát sinh, và quyền điều khiển được chuyển cho hệ điều hành. Hệ điều hành đặt reference bit 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 reference bit duy nhất. Thuật toán cơ sở vẫn là FIFO, tuy nhiên khi chọn được một trang theo tiêu chuẩn FIFO, kiểm tra reference bit của trang đó :

- Nếu giá trị của reference bit là 0, thay thế trang đã chọn.

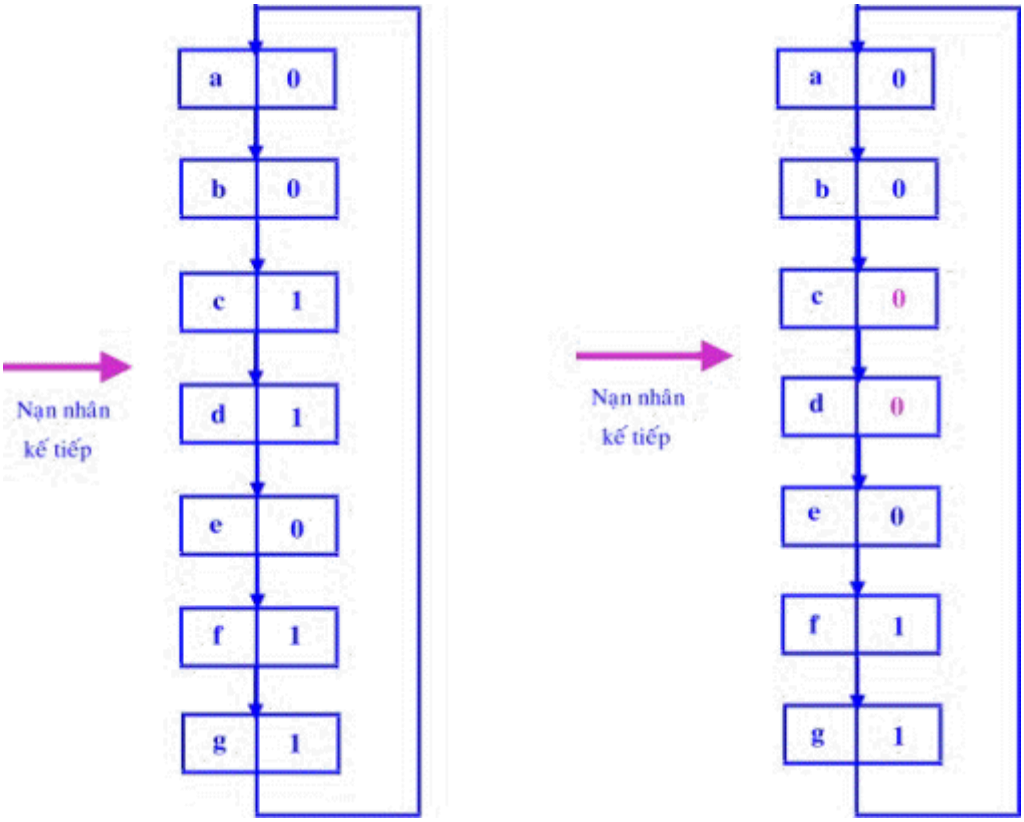
- Ngược lại, cho trang này một cơ hội thứ hai, và chọn trang FIFO tiếp theo.

- Khi một trang được cho cơ hội thứ hai, giá trị của reference bit 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, reference bit 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 7.6. *Thuật toán thay thế trang <<cơ hội thứ hai >>*

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

* Tiếp cận: xem các reference bit và dirty bit như một cặp có thứ tự.

- Với hai bit này, có thể có 4 tổ hợp tạo thành 4 lớp sau:

(0,0) không truy xuất, không sửa đổi: đây là trang tốt nhất để thay thế.

(0,1) không truy xuất gần đây, nhưng đã bị sửa đổi: trường hợp này không thật tốt, vì trang cần được lưu trữ lại trước khi thay thế.

(1,0) được truy xuất gần đây, nhưng không bị sửa đổi: trang có thể nhanh chóng được tiếp tục được sử dụng.

(1,1) được truy xuất gần đây, và bị sửa đổi: trang có thể nhanh chóng được tiếp tục được sử dụng, và trước khi thay thế cần phải được lưu trữ lại.

- Lớp 1 có độ ưu tiên thấp nhất, và lớp 4 có độ ưu tiên cao nhất.

- Một trang sẽ thuộc về một trong bốn lớp trên, tuỳ vào reference bit 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).

7.3. 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:

si = kích thước của bộ nhớ ảo cho tiến trình *pi*

$S = \Sigma\ si$

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

Cấp phát *ai* khung trang cho tiến trình *pi* : $ai = (si / S)\ m$

- *Cấp phát theo độ ưu tiên*: sử dụng ý tưởng cấp phát theo tỷ lệ, nhưng 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 *pi* 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*).

7.3.1. 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.

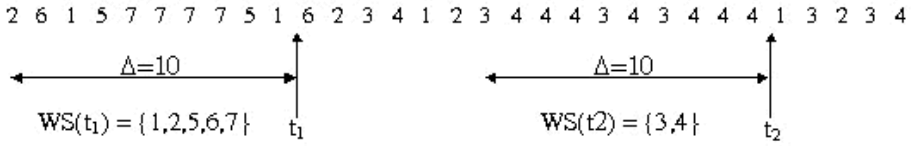
Để ngăn cản tình trạng trì trệ này xảy ra, cần phải cấp cho tiến trình *đủ* các khung trang cần thiết để hoạt động. Vấn đề cần giải quyết là làm sao biết được tiến trình cần bao nhiêu trang?

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.

7.3.1.1. 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 Δ , để định nghĩa một cửa sổ cho *working set*. Giả sử khảo sát Δ đơn vị thời gian (lần truy xuất trang) cuối cùng, tập các trang được tiến trình truy xuất đến trong Δ 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 Δ 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 7.7. *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, WSSi, cho mỗi tiến trình trong hệ thống, thì có thể xem như:

$$D = \Sigma WSSi$$

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 *WSSi* 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 reference bit:

+ 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 reference bit 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 tiền phân trang (prepaging): nạp vào bộ nhớ một lần tất cả các trang trong working set của tiến trình.

7.3.2. 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 sỡ 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 trang 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.

7.4. 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 sử đổi là 70%. Tỷ lệ phát sinh lỗi trang phải là bao nhiêu để có thể duy trì thời gian truy xuất bộ nhớ (effective acess time) không vượt quá 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 reference bit (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?

Chương 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. Chương này giới thiệu 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.

8.1. Các khái niệm cơ bản

8.1.1. Bộ nhớ ngoài

Máy tính phải sử dụng thiết bị có khả năng lưu trữ trong thời gian dài (long-term) vì:

Phải chứa những lượng thông tin rất lớn (giữ vé máy bay, ngân hàng...).

Thông tin phải được lưu giữ một thời gian dài trước khi xử lý.

Nhiều tiến trình có thể truy cập thông tin cùng lúc.

Giải pháp là sử dụng các thiết bị lưu trữ bên ngoài gọi là bộ nhớ ngoài.

8.1.2. Tập tin và thư mục

- Tập tin:

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

- *Thư mục:*

Để lưu trữ dãy các tập tin, hệ thống quản lý tập tin cung cấp thư mục, mà trong nhiều hệ thống có thể coi như là tập tin.

8.1.3. Hệ thống quản lý tập tin

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

8.2. Mô hình tổ chức và quản lý các tập tin

8.2.1. Mô hình

a,Tập tin:

** Tên tập tin:*

Tập tin là một cơ chế trừu tượng và để quản lý mỗi đối tượng phải có một tên. Khi tiến trình tạo một tập tin, nó sẽ đặt một tên, khi tiến trình kết thúc tập tin vẫn tồn tại và có thể được truy xuất bởi các tiến trình khác với tên tập tin đó.

Cách đặt tên tập tin của mỗi hệ điều hành là khác nhau, đa số các hệ điều hành cho phép sử dụng 8 chữ cái để đặt tên tập tin như ctdl, caycb, tamhghau v.v…, thường thường thì các ký tự số và ký tự đặc biệt cũng được sử dụng như baitap2,…

Hệ thống tập tin có thể có hay không phân biệt chữ thường và chữ hoa. Ví dụ: UNIX phân biệt chữ thường và hoa còn MS-DOS thì không phân biệt.

Nhiều hệ thống tập tin hỗ trợ tên tập tin gồm 2 phần được phân cách bởi dấu ‘.’ mà phần sau được gọi là phần mở rộng. Ví dụ: vidu.txt. Trong MS-

DOS tên tập tin có từ 1 đến 8 ký tư, phần mở rộng có từ 1 đến 3 ký tự. Trong UNIX có thể có nhiều phân cách như prog.c.Z.

Một số kiểu mở rộng thông thường là:

.bak, .bas, .bin, .c, .dat, .doc, .ftn, .hlp, .lib, .obj, .pas, .tex, .txt.

Trên thực tế phần mở rộng có hữu ích trong một số trường hợp, ví dụ như có những trình dịch C chỉ nhận biết các tập tin có phần mở rộng là .C

** Cấu trúc của tập tin:*

Gồm 3 loại:

- Dãy tuần tự các byte không cấu trúc : hệ điều hành không biết nội dung của tập tin: MS-DOS và UNIX sử dụng loại này.

- Dãy các record có chiều dài cố định.

- Cấu trúc cây: gồm cây của những record, không cần thiết có cùng độ dài, mỗi record có một trường khóa giúp cho việc tìm kiếm nhanh hơn.

** Kiểu tập tin :*

Nếu hệ điều hành nhận biết được loại tập tin, nó có thể thao tác một cách hợp lý trên tập tin đó. Các hệ điều hành hỗ trợ cho nhiều loại tập tin khác nhau bao gồm các kiểu như : tập tin thường, thư mục, tập tin có ký tự đặc biệt, tập tin khối.

- *Tập tin thường*: là tập tin text hay tập tin nhị phân chứa thông tin của người sử dụng.

- *Thư mục*: là những tập tin hệ thống dùng để lưu giữ cấu trúc của hệ thống tập tin.

- *Tập tin có ký tự đặc biệt*: liên quan đến nhập xuất thông qua các thiết bị nhập xuất tuần tự như màn hình, máy in, mạng.

- *Tập tin khối*: dùng để truy xuất trên thiết bị đĩa.

- Tập tin thường được chia làm hai loại là tập tin văn bản và tập tin nhị phân.

Tập tin văn bản chứa các dòng văn bản cuối dòng có ký hiệu enter. Mỗi dòng có độ dài có thể khác nhau. Ưu điểm của kiểu tập tin này là nó có thể hiển thị, in hay soạn thảo với một editor thông thường. Đa số các chương trình dùng tập tin văn bản để nhập xuất, nó cũng dễ dàng làm đầu vào và đầu ra cho cơ chế pipeline.

Tập tin nhị phân có cấu trúc khác tập tin văn bản. Mặc dù về mặt kỹ thuật , tập tin nhị phân gồm dãy các byte, nhưng hệ điều hành chỉ thực thi tập tin đó nếu nó có cấu trúc đúng. Ví dụ một một tập tin nhị phân thi hành được của UNIX. Thường thường nó bao gồm năm thành phần: header, text, data, relocation bits, symbol table. Header bắt đầu bởi byte nhận diện cho biết đó là tập tin thi hành. Sau đó là 16 bit cho biết kích thước các thành phần của tập tin, địa chỉ bắt đầu thực hiện và một số bit cờ. Sau header là dữ liệu và text của tập tin. Nó được nạp vào bộ nhớ và định vị lại bởi những bit relocation. Bảng symbol được dùng để debug.

Một ví dụ khác là tập tin nhị phân kiểu archive. Nó chứa các thư viện đã được dịch nhưng chưa được liên kết. Bao gồm một header cho biết tên, ngày tạo, người sở hữu, mã bảo vệ, và kích thước…

Hình 8.1. *Cấu trúc tập tin nhị phân trong UNIX*

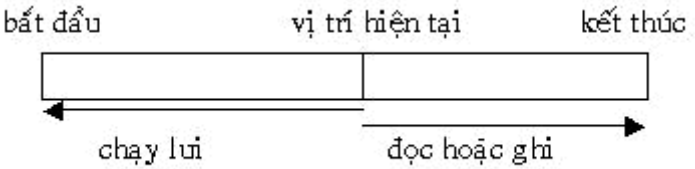
**** Truy xuất tập tin:***

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 dự trên kiểu truy xuất này như truy xuất theo chỉ mục...

**** Thuộc tính tập tin:***

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

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

Tên thuộc tính	**Ý nghĩa**
Bảo vệ	Ai có thể truy xuất được và bằng cách nào
Mật khẩu	Mật khẩu cần thiết để truy xuất tập tin
Người tạo	Id của người tạo tập tin

Người sở hữu	Người sở hữu hiện tại
Chỉ đọc	0 là đọc ghi, 1 là chỉ đọc
Aản	0 là bình thường, 1 là không hiển thị khi liệt kê
Hệ thống	0 là bình thường, 1 là tập tin hệ thống
Lưu trữ	0 đã được backup, 1 cần backup
ASCII/binary	0 là tập tin văn bản, 1 là tập tin nhị phân
Truy xuất ngẫu nhiên	0 truy xuất tuần tự, 1 là truy xuất ngẫu nhiên
Temp	0 là bình thường, 1 là bị xóa khi tiến trình kết thúc
Khóa	0 là không khóa, khác 0 là khóa
Độ dài của record	Số byte trong một record
Vị trí khóa	Offset của khóa trong mỗi record
Giờ tạo	Ngày và giờ tạo tập tin
Thời gian truy cập cuối cùng	Ngày và giờ truy xuất tập tin gần nhất
Thời gian thay đổi cuối cùng	Ngày và giờ thay đổi tập tin gần nhất
Kích thước hiện thời	Số byte của tập tin
Kích thước tối đa.	Số byte tối đa của tập tin

Bảng 8.1. *Một số thuộc tính thông dụng của tập tin*

b, Thư mục:

** Hệ thống thư mục theo cấp bậc:*

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

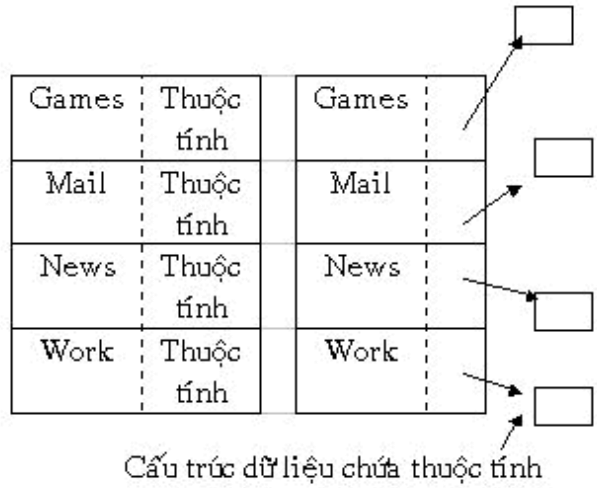
Khi một tập tin được mở, hệ điều hành tìm trên thư mục của nó cho tới khi tìm thấy tên của tập tin được mở. Sau đó nó sẽ xác định thuộc tính cũng như địa chỉ lưu trữ trên đĩa và đưa vào một bảng trong bộ nhớ. Những truy xuất sau đó thực hiện trong bộ nhớ chính.

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

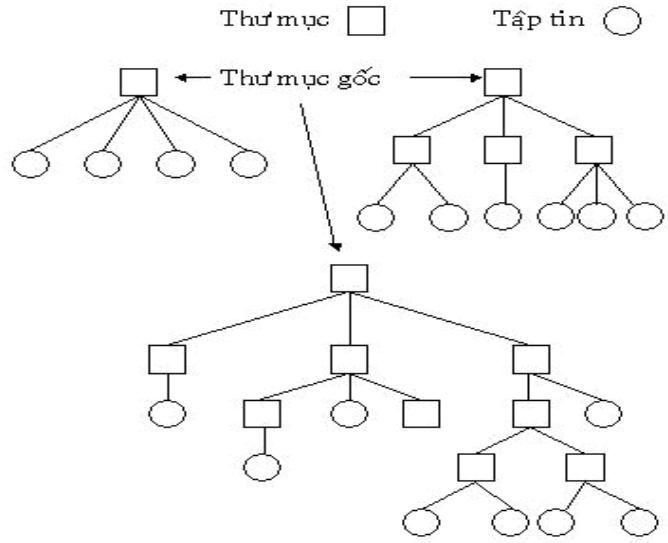
Cách thứ hai là có một thư mục gốc và trong đó có nhiều thư mục con, trong mỗi thư mục con chứa tập tin của người sử dụng (còn gọi là thư mục hai cấp), cách này tránh được trường hợp xung đột tên nhưng cũng còn khó khăn với người dùng có nhiều tập tin. Người sử dụng luôn muốn nhóm các ứng dụng lại một cách logic.

Từ đó, hệ thống thư mục theo cấp bậc (còn gọi là cây thư mục) được hình thành với mô hình một thư mục có thể chứa tập tin hoặc một thư mục con và cứ tiếp tục như vậy hình thành cây thư mục như trong các hệ điều hành DOS, Windows, v. v...

Ngoài ra, trong một số hệ điều hành nhiều người dùng, hệ thống còn xây dựng các hình thức khác của cấu trúc thư mục như cấu trúc thư mục theo đồ thị có chu trình và cấu trúc thư mục theo đồ thị tổng quát. Các cấu trúc này cho phép các người dùng trong hệ thống có thể liên kết với nhau thông qua các thư mục chia sẻ.



Hình 8.3. *Hai dạng cấu trúc thư mục*



Hình 8.4. *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.

8.2.2. 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 xoá tập tin sau một khoảng thời gian n ngày.

- *Mở*: Trước khi sử dụng một tập tin, tiến trình phải mở nó. Mục tiêu của mở là cho phép hệ thống thiết lập một số thuộc tính và địa chỉ đĩa trong bộ nhớ để tăng tốc độ truy xuất.

- *Đóng*: Khi chấm dứt truy xuất, thuộc tính và địa chỉ trên đĩa không cần dùng nữa, tập tin được đóng lại để giải phóng vùng nhớ. Một số hệ thống hạn chế tối đa số tập tin mở trong một tiến trình.

- *Đọc*: Đọc dữ liệu từ tập tin tại vị trí hiện thời của đầu đọc, nơi gọi sẽ cho biết cần bao nhiêu dữ liệu và vị trí của buffer lưu trữ nó.

- *Ghi*: Ghi dữ liệu lên tập tin từ vị trí hiện thời của đầu đọc. Nếu là cuối tập tin,kích thước tập tin sẽ tăng lên, nếu đang ở giữa tập tin, dữ liệu sẽ bị ghi chồng lên.

- *Thêm*: Gần giống như WRITE nhưng dữ liệu luôn được ghi vào cuối tập tin.

- *Tìm*: Dùng để truy xuất tập tin ngẫu nhiên. Khi xuất hiện lời gọi hệ thống, vị trí con trỏ đang ở vị trí hiện hành được di chuyển tới vị trí cần thiết. Sau đó dữ liệu sẽ được đọc ghi tại vị trí này.

- *Lấy thuộc tính*: Lấy thuộc tính của tập tin cho tiến trình

- *Thiết lập thuộc tính*: Thay đổi thuộc tính của tập tin sau một thời gian sử dụng.

- *Đổi tên*: Thay đổi tên của tập tin đã tồn tại.

** Thư mục:*

- *Tạo*: Một thư mục được tạo, nó rỗng, ngoại trừ "." và ".." được đặt tự động bởi hệ thống.

- *Xóa*: Xoá một thư mục, chỉ có thư mục rỗng mới bị xóa, tư mục chứa "." và ".." coi như là thư mục rỗng.

- *Mở thư mục*: Thư mục có thể được đọc. Ví dụ để liệt kê tất cả tập tin trong một thư mục, chương trình liệt kê mở thư mục và đọc ra tên của tất cả tập tin chứa trong đó. Trước khi thư mục được đọc, nó phải được mở ra trước.

- *Đóng thư mục*: Khi một thư mục đã được đọc xong, phải đóng thư mục để giải phóng vùng nhớ.

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

- *Đổi tên*: Cũng như tập tin, thư mục cũng có thể được đổi tên.

- *Liên kết*: Kỹ thuật này cho phép một tập tin có thể xuất hiện trong nhiều thư mục khác nhau. Khi có yêu cầu, một liên kết sẽ được tạo giữa tập tin và một đường dẫn được cung cấp.

- *Bỏ liên kết*: Nếu tập tin chỉ còn liên kết với một thư mục, nó sẽ bị loại bỏ hoàn toàn khỏi hệ thống, nếu nhiều thì nó bị giảm chỉ số liên kết.

*** 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.

Chương 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. Chương này sẽ giới thiệu 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.

9.1. Bảng quản lý thư mục, tập tin

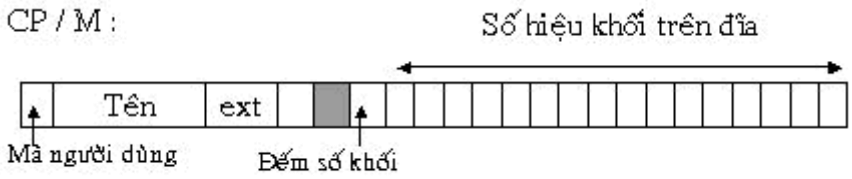
9.1.1. Khái niệm

Trước khi tập tin được đọc, tập tin phải được mở, để mở tập tin hệ thống phải biết đường dẫn do người sử dụng cung cấp và được định vị trong cấu trúc đầu vào thư mục (directory entry). Directory entry cung cấp các thông tin cần thiết để tìm kiếm các khối. Tuỳ thuộc vào mỗi hệ thống, thông tin là địa chỉ trên đĩa của toàn bộ tập tin, số hiệu của khối đầu tiên, hoặc là số I-node.

9.1.2. Cài đặt

Bảng này thường được cài đặt ở phần đầu của đĩa. Bảng là dãy các phần tử có kích thước xác định, mỗi phần tử được gọi là một entry. Mỗi entry sẽ lưu thông tin về tên, thuộc tính, vị trí lưu trữ... của một tập tin hay thư mục.

Ví dụ quản lý thư mục trong CP/M:



Hình 9.1. (thiếu nội dung)

9.2. Bảng phân phối vùng nhớ

9.2.1. 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.

9.2.2 Các phương pháp

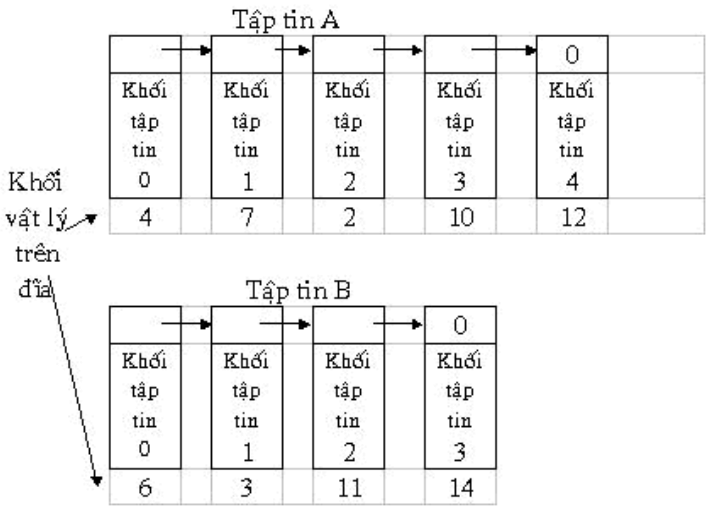
** Định vị liên tiếp*

Lưu trữ tập tin trên dãy các khối liên tiếp.

Phương pháp này có 2 ưu điểm: thứ nhất, 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:*

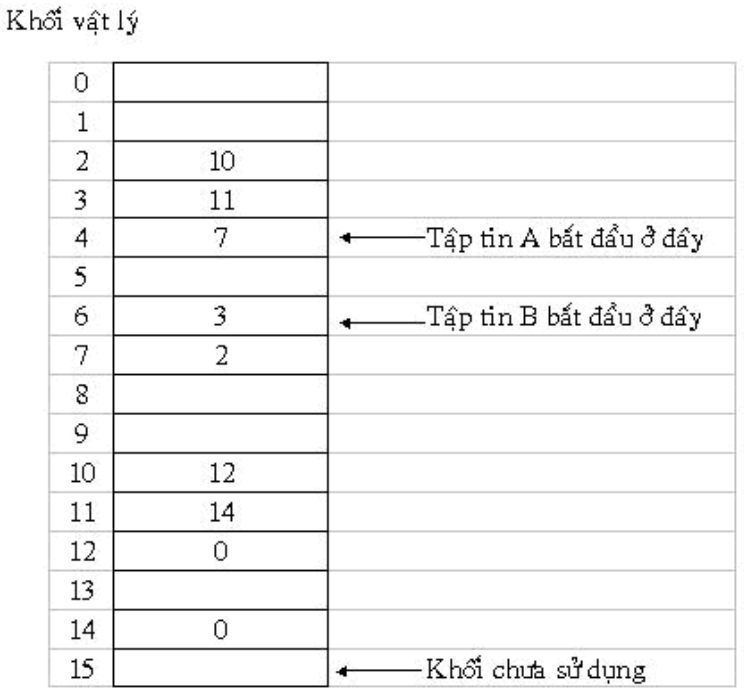


Hình 9.2. *Định vị bằng danh sách liên kết*

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:



Hình 9.3. *Bảng chỉ mục của danh sách*

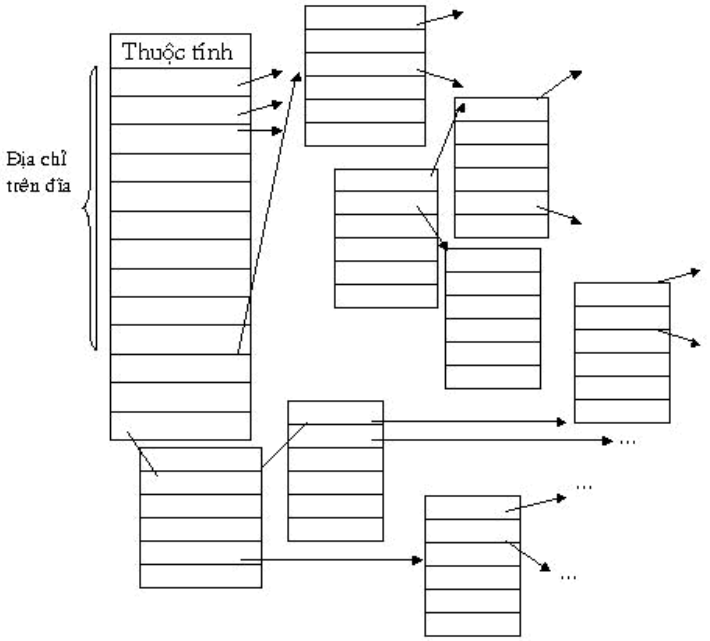
Tương tự như hai nhưng thay vì dùng con trỏ thì dùng một bảng index. Khi đó toàn bộ khối chỉ chứa dữ liệu. Truy xuất ngẫu nhiên sẽ dễ dàng hơn. Kích thước tập tin được mở rộng hơn. Hạn chế là bản này bị giới hạn bởi kích thước bộ nhớ.

** I-nodes*

Một I-node bao gồm hai phần. Phần thứ nhất là thuộc tính của tập tin. Phần này lưu trữ các thông tin liên quan đến tập tin như kiểu, người sở hữu, kích thước, v.v... Phần thứ hai chứa địa chỉ của khối dữ liệu. Phần này chia làm hai phần nhỏ. Phần nhỏ thứ nhất bao gồm 10 phần tử, mỗi phần tử chứa địa chỉ khối dữ liệu của tập tin. Phần tử thứ 11 chứa địa chỉ gián tiếp cấp 1 (single

indirect), chứa địa chỉ của một khối, trong khối đó chứa một bảng có thể từ 2^{10} đến 2^{32} phần tử mà mỗi phần tử mới chứa địa chỉ của khối dữ liệu. Phần tử thứ 12 chứa địa chỉ gián tiếp cấp 2 (double indirect), chứa địa chỉ của bảng các khối single indirect. Phần tử thứ 13 chứa địa chỉ gián tiếp cấp 3 (double indirect), chứa địa chỉ của bảng các khối double indirect.

Cách tổ chức này tương đối linh động. Phương pháp này hiệu quả trong trường hợp sử dụng để quán lý những hệ thống tập tin lớn. Hệ điều hành sử dụng phương pháp này là Unix (Ví dụ: BSD Unix)



Hình 9.4. *Cấu trúc của I-node*

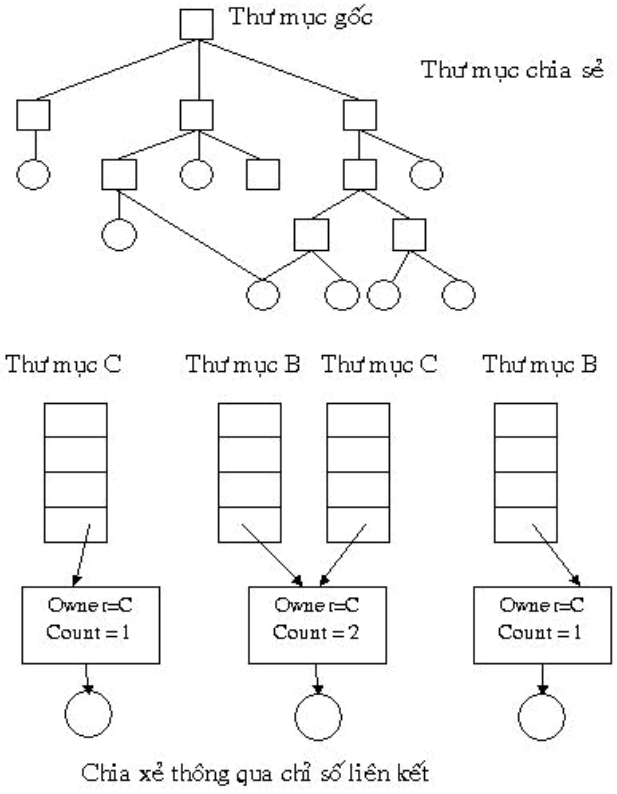
9.3. Tập tin chia sẻ

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 trỏ tới cấu trúc này. Một cách khác là hệ thống tạo một tập tin mới có kiểu LINK, tập tin mới này chỉ chứa đường dẫn của tập tin được liên kết, khi cần truy xuất sẽ dựa trên tập tin LINK để xác định

tập tin cần truy xuất, phương pháp này gọi là liên kết hình thức. Mỗi phương pháp đều có những ưu và khuyết điểm riêng.

Ở phương pháp thứ nhất hệ thống biết được có bao nhiêu thư mục liên kết với tập tin nhờ vào chỉ số liên kết. Ở phương pháp thứ hai khi loại bỏ liên kết hình thức, tập tin không bị ảnh hưởng.



Hình 9.5.Cấu trúc quản lý tập tin chia sẻ

9.4. Quản lý đĩa (bộ nhớ ngoài)

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.

9.4.1. 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:

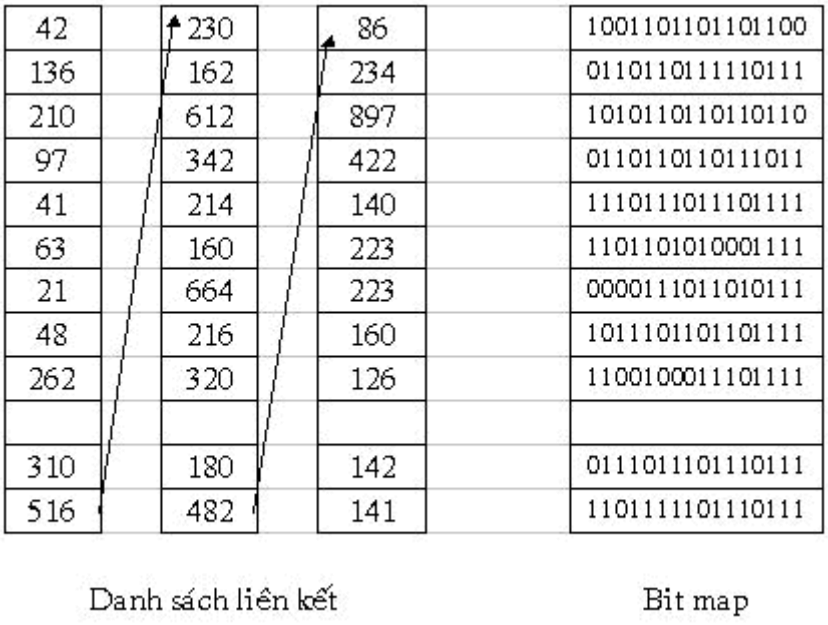
$$30 + 8.3 + (k/32768) \times 16.67$$

Từ đó thống kê được kích thước khối thích hợp phải < 2K .

Thông thường kích thuóc khối là 512, 1K hay 2K.

9.4.2. 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.



Hình 9.6. *Hai phương pháp lưu giữ khối trống*

9.5. 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 những hư hỏng do phần cứng gây ra, vì vậy chúng phải được cài đặt một số chức năng để bảo vệ.

9.5.1. Quản lý khối bị hỏng

Đĩa thường có những khối bị hỏng trong quá trình sử dụng đặc biệt đối với đĩa cứng vì khó kiểm tra được hết tất cả.

Có hai giải pháp: phần mềm và phần cứng.

Phần cứng là dùng một sector trên đĩa để lưu giữ danh sách các khối bị hỏng. Khi bộ kiểm soát tực hiện lần đầu tiên, nó đọc những khối bị hỏng và dùng một khối thừa để lưu giữ. Từ đó không cho truy cập những khối hỏng nữa.

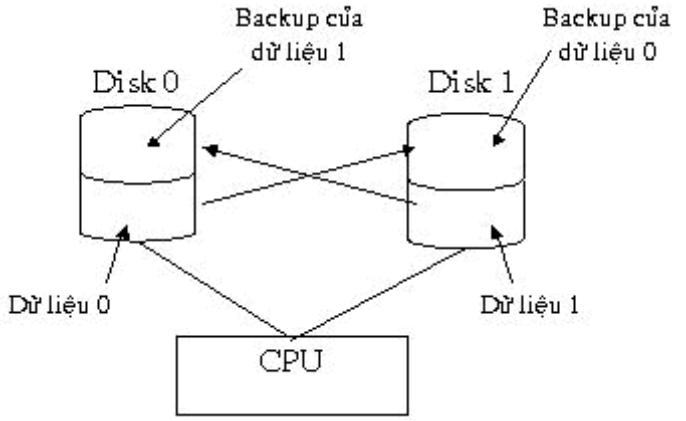
Phần mềm là hệ thống tập tin xây dựng một tập tin chứa các khối hỏng. Kỹ thuật này loại trừ chúng ra khỏi danh sách các khối trống, do đó nó sẽ không được cấp phát cho tập tin.

9.5.2. 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.

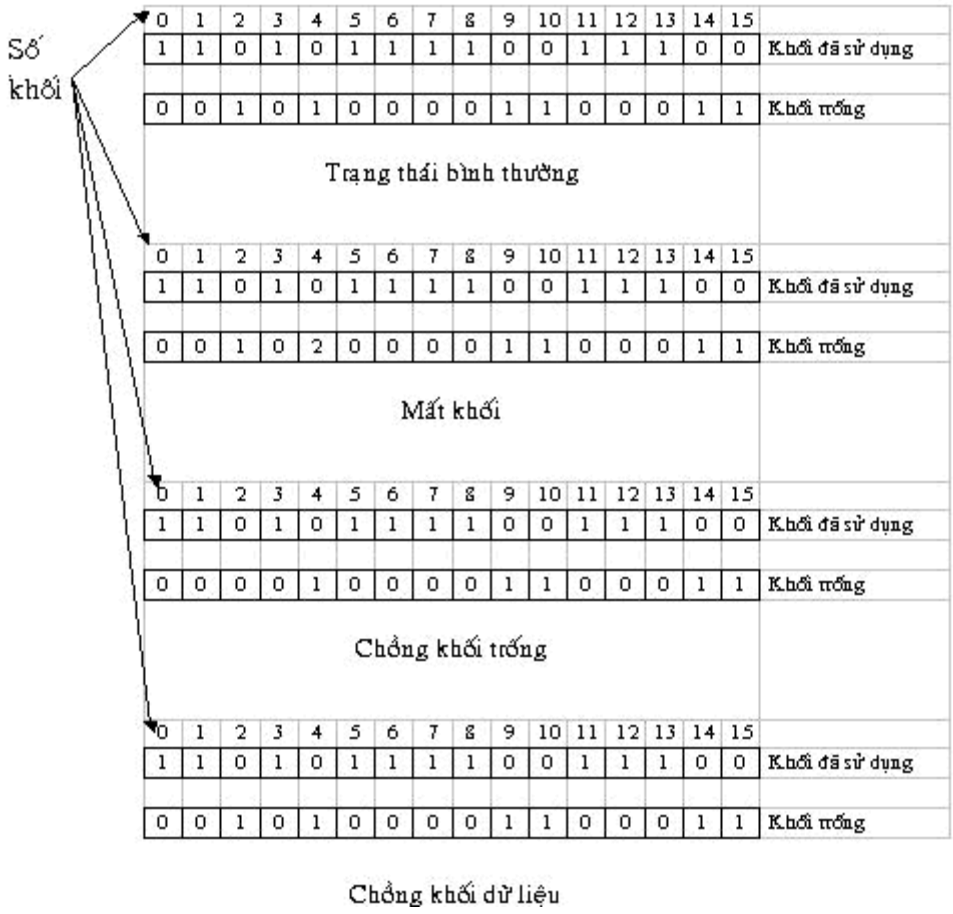
Đối với các đĩa cứng lớn, việc backup thường được tiến hành ngay trên nó. Một chiến lược dể cài đặt nhưng lãng phí một nữa đĩa là chia đĩa cứng làm hai phần một phần dữ liệu và một phần là backup. Mỗi tối, dữ liệu từ phần dữ liệu sẽ được chép sang phần backup.



Hình 9.7. *Backup*

9.5.3. 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 lân khởi động, hệ thống sẽ thực hiện việc kiểm tra trên hai phần khối và tập tin. Việc kiểm tra thực hiện, khi phát hiện ra lỗi sẽ tiến hành sữa chữa cho các trường hợp cụ thể:



Hình 9.8 *Trạng thái của hệ thống tập tin*

Chương 10

GIỚI THIỆU MỘT SỐ HỆ THỐNG TẬP TIN

Chương này giới thiệu 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.

10.1. MS-DOS

10.1.1. 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.

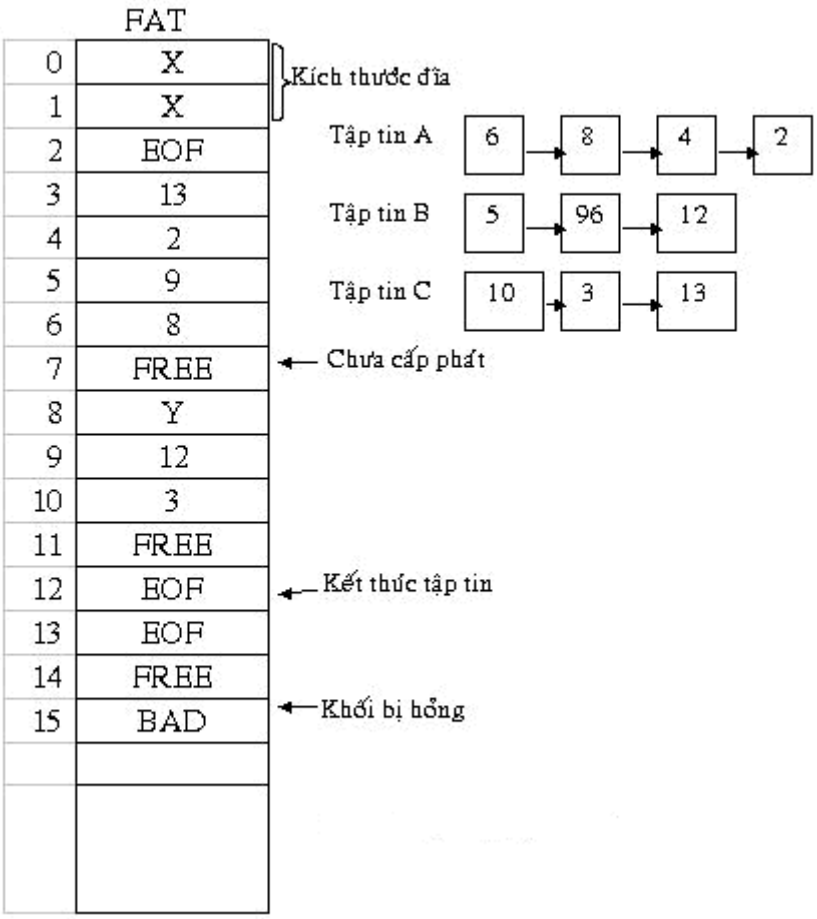
- Không có mount hệ thống tập tin.

- Có thuộc tính của tập tin.

10.1.2. Cài đặt

Cài đặt trên đĩa mềm cũng tương tự như trên đĩa cứng, những 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.



Hình 10.1. *FAT trong MS-DOS*

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) A-D-V-S-H-R
Dành riêng (10bytes)
Giờ (2bytes)
Ngày (2bytes)
Khối đầu tiên (2bytes)
Kích thước tập tin (4bytes)

Bảng 10.1 *Một entry của 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 dàng 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)

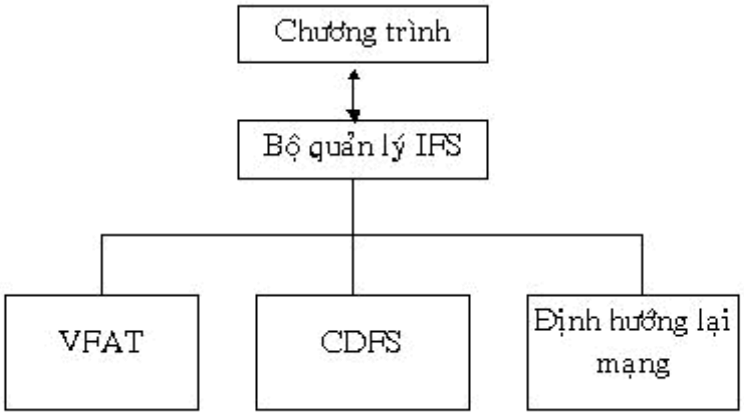
- Các sector còn lại dùng để lưu dữ liệu (1 cluser = 1 sector).

10.2. Windows95

10.2.1. 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 10.2. *Cấu trúc của bộ quản lý hệ thống thông 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)

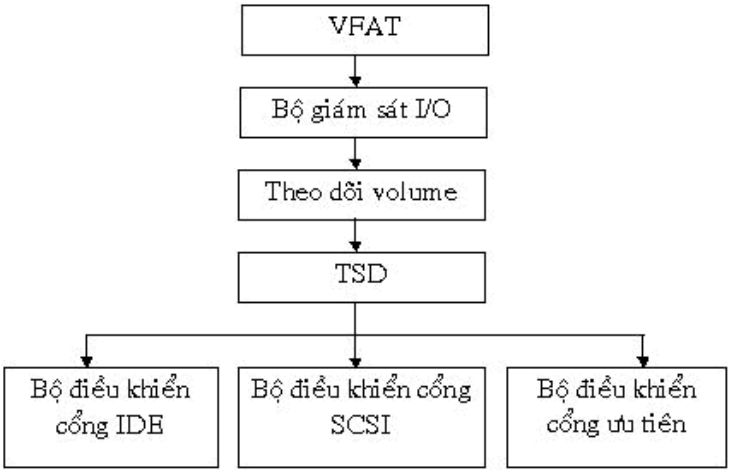
- Bộ định hướng lại-Sử dụng cho truy xuất mạng.

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.

10.2.2. 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 10.3. *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. 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 lam 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.

10.2.3. 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

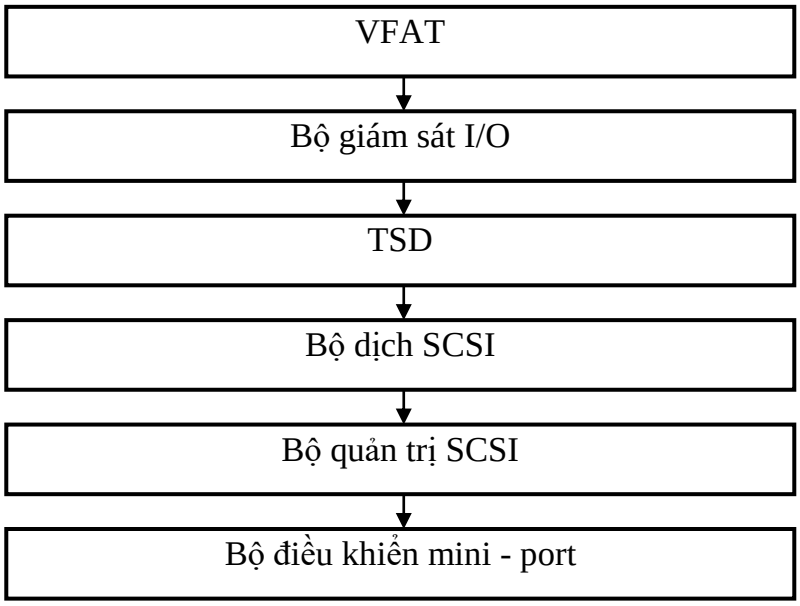
thay thế cho phần mềm SMARTDrive disk cache 16-bit ở mode thực của MS-DOS và Win3.1. Đặc điểm của VCACHE là thuật toán thông minh hơn SMARTDrive trong lưu trữ thông tin nhập và xuất từ bộ điều khiển đĩa.VCACHE cũng quản lý vùng lưu trữ cho CDFS và NWREDIR 32-bit.

Việc sử dụng VCACHE là phụ thuộc với thiết bị. Ví dụ VCACHE dùng để truy xuất đĩa cứng khác với VCACE 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.

10.2.4. 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:



** Bộ dịch SCSI:*

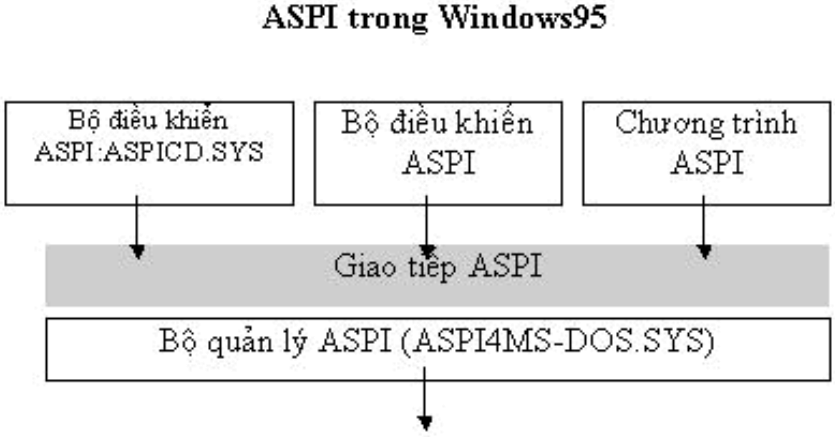
Bộ dịch SCSI làm việc với tất cả những thiết bị SCSI như đĩa cứng, CD-ROM. Bộ dịch chịu trách nhiệm xây dựng khối mô tả lệnh SCSI cho những lớp của thiết bị SCSI và thực hiện tìm lỗi ở cấp thiết bị.

** 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 mimiport, 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 cổng chuẩn của Windows 95.



Hình 10.4. *Kiến trúc ASPI*

Bộ ánh 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.

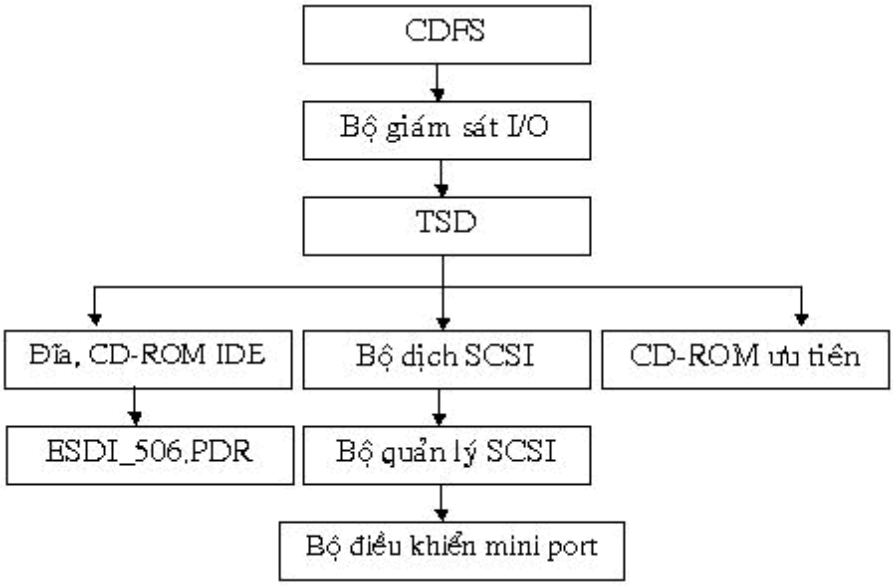
Bộ điều khiển ASPI gọi bộ quản trị ASPI. Bộ quản trị ASPI chuyển lời gọi cho CDB (Command Descriptor Khối) gọi tới những thành phần SCSI. Bộ quản trị ASPI cần thiết cho những trường hợp sau đây :

- Nhiều adapter chủ.

- Đĩa cứng SCSI với SCSI ID khác 0 hay 1.

- SCSI tape, máy in, máy vẽ, máy quét.

10.2.5. CDFS



Hình 10.5. *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.

10.3. WINDOWNS 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.

10.3.1. 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.

10.3.2. 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 10.6. *Windows NTFS Partition và kích thước cluster.*

Cấu trúc volume của NTFS :

Hình 10.7. *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 10.2. *Các kiểu thuộc tính của tập tin và thư mục của Windows NTFS*

10.4. UNIX

10.4.1. Hệ thống tập tin của Unix

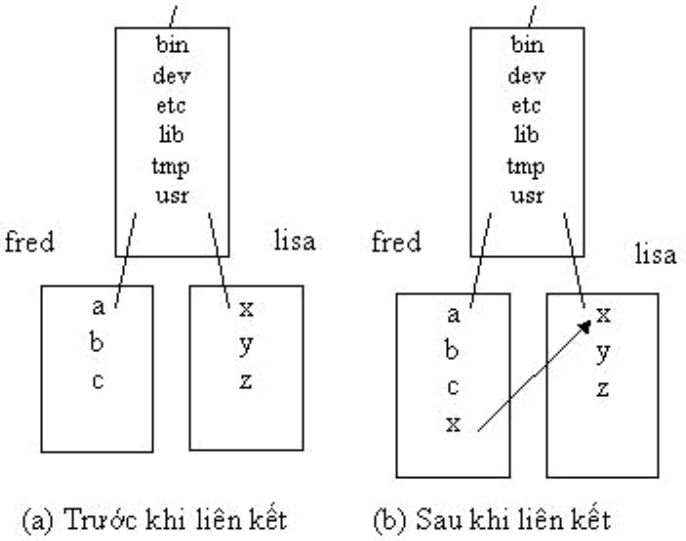
Một tập tin được mở với lời gọi hệ thống OPEN, với tham số đầu tiên cho biết đường dẫn và tên tập tin , tham số thứ hai cho biết tập tin được mở để đọc, ghi hay vừa đọc vừa ghi. Hệ thống kiểm tra xem tập tin có tồn tại không. Nếu có, nó kiểm tra bit quyền để xem có được quyền truy cập không, nếu có hệ 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.

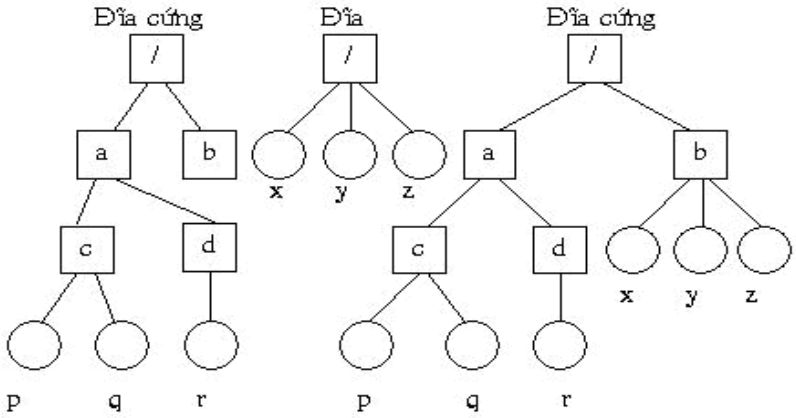
Có hai cách mô tả tên tập tin trong UNIX. Cách thứ nhất là dùng đường dẫn tuyệt đối, tập tin được truy cập từ thư mục gốc. Thứ hai là dùng khái niệm thư mục làm việc hay thư mục hiện hành trong đường dẫn tương đối.

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 10.8. *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.

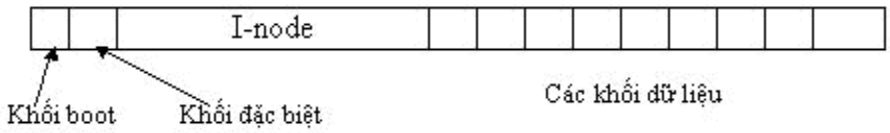


Hình 10.9. *Mount trong UNIX*

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 xẻ nữa, nhưng không thể đặt một khoá loại trừ nhưng nếu đã được đặt khóa loại trừ thì không thể đặt thêm khóa nữa. Vùng khóa có thể được ghi chồng.

10.4.2. 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 10.10. *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 xoá 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.

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

NHÀ XUẤT BẢN ĐẠI HỌC THÁI NGUYÊN
Địa chỉ: Phường Tân Thịnh - Thành phố Thái Nguyên - Tỉnh Thái Nguyên
Điện thoại: 0280 3840023; Fax: 0280 3840017
Website: nxb.tnu.edu.vn * E-mail: nxb.dhtn@gmail.com

NÔNG MINH NGỌC (Chủ biên), NGUYỄN VĂN HUY

GIÁO TRÌNH

NGUYÊN LÝ HỆ ĐIỀU HÀNH

Chịu trách nhiệm xuất bản:
PGS.TS. NGUYỄN ĐỨC HẠNH
Giám đốc - Tổng biên tập

Biên tập:	TRỊNH THANH ĐIỆP
Thiết kế bìa:	NGUYỄN VĂN HƯNG
Trình bày:	NGUYỄN THỊ THÙY DƯƠNG
Sửa bản in:	PHẠM VĂN VŨ

ISBN: 978-604-915-375-4

In 500 cuốn, khổ 17 x 24 cm, tại Công ty TNHH In và Thương mại Trường Xuân (Địa chỉ: Khu X1, Phạm Hùng, Từ Liêm, Hà Nội). Giấy phép xuất bản số 1928-2016/CXBIPH/01-67/ĐHTN. Quyết định xuất bản số: 104/QĐ-NXBĐHTN. In xong và nộp lưu chiểu quý II năm 2016.