

Thông tin về giáo trình

- Tên giáo trình: Nguyên Lý Hệ Điều Hành
- Ứng với mã học phần: CSC3312 Số tín chỉ: 2
- Các kiến thức học phần tiên quyết: Kiến trúc máy tính, Ngôn ngữ lập trình bậc cao.

Mục tiêu của giáo trình

- Kiến thức: giúp sinh viên nắm rõ kiến trúc của hệ điều hành cũng như những vấn đề mà một hệ điều hành gặp phải cùng với các phương pháp giải quyết.
- Kỹ năng: sinh viên có khả năng tự đọc tài liệu tham khảo và tham gia thảo luận trên lớp.

Tóm tắt nội dung giáo trình

Giáo trình này được xây dựng đáp ứng với học phần Nguyên lý hệ điều hành trong chương trình Tín chỉ chuyên ngành CNTT, giúp sinh viên nắm rõ nguyên tắc hoạt động, tổ chức, quản lý phần cứng máy tính cùng với các phương pháp giải quyết khi xảy ra xung đột. Tổ chức quản lý và cấp phát tài nguyên, tối ưu hoá hệ thống theo yêu cầu. Từ đó, sinh viên sẽ hiểu rõ và ứng dụng trong xây dựng và phát triển ứng dụng phần mềm cũng như phần cứng.

Giáo trình *Nguyên lý Hệ điều hành* bao gồm 4 chương. Chương 1 trình bày những kiến thức tổng quan về Hệ điều hành. Chương 2 tìm hiểu về nguyên lý hoạt động của các tiến trình và vai trò của hệ điều hành trong việc quản lý các tiến trình, đặc biệt trong môi trường đơn Vi xử lý. Nội dung của chương 3 là các kỹ thuật nạp chương trình vào bộ nhớ và kỹ thuật bộ nhớ ảo. Chương 4 giới thiệu về tổ chức hệ thống lưu trữ bao gồm: hệ thống tập tin, tổ chức lưu trữ trên bộ nhớ phụ, ...

Mục lục

Thông tin về giáo trình.....	1
Mục tiêu của giáo trình	1
Tóm tắt nội dung giáo trình.....	1
Mục lục	2
Chương 1. TỔNG QUAN.....	5
1.1. Hệ điều hành là gì?	5
1.1.1. Tầm nhìn người dùng.....	6
1.1.2. Tầm nhìn hệ thống.....	6
1.1.3. Mục tiêu hệ thống	7
1.1.4. Một số định nghĩa Hệ điều hành.....	8
1.2. Phân loại hệ điều hành	8
1.2.1. Hệ điều hành theo lô đơn giản	8
1.2.2. Hệ điều hành theo lô đa chương	9
1.2.3. Hệ điều hành chia sẻ thời gian.....	9
1.2.4. Hệ điều hành mạng	10
1.3. Một số khái niệm của hệ điều hành	10
1.4. Các thành phần của hệ điều hành	11
Một số câu hỏi ôn tập	12
Chương 2 QUẢN LÝ TIẾN TRÌNH.....	13
2.1 Tổng quan về tiến trình	13
2.1.1 Khái niệm về tiến trình	13
2.1.2. Trạng thái tiến trình	15
2.1.3. Mô hình tiến trình	16
2.1.4. Khối điều khiển tiến trình (PCB).....	17
2.1.5. Các thao tác điều khiển tiến trình	18
2.2 Lập lịch tiến trình	20
2.2.1 Các khái niệm cơ bản.....	20
2.2.1.1 Chu kỳ CPU - I/O.....	21
2.2.1.2 Trình lập lịch CPU (CPU Scheduler)	21
2.2.1.3 Trình điều vận (Dispatcher)	23
2.2.2 Tiêu chuẩn lập lịch tiến trình	23
2.2.3 Các giải thuật lập lịch	24
2.2.3.1 Giải thuật FCFS (First-come, First-served).....	25
2.2.3.2 Giải thuật phân phối xoay vòng RR (Round Robin).....	26
2.2.3.3 Giải thuật công việc ngắn nhất SJF (Shortest Job Fist).....	27
2.2.3.4 Giải thuật theo độ ưu tiên	28
2.2.3.5 Chiến lược nhiều cấp độ ưu tiên.....	29
2.3 Đồng bộ hóa tiến trình.....	31
2.3.1 Tài nguyên căng và đoạn căng	32

2.3.1.1	Tài nguyên găng (Critical Resource)	32
2.3.1.2	Đoạn găng (Critical Section).....	35
2.3.1.3	Yêu cầu của công tác điều độ qua đoạn găng.....	35
2.3.2	Điều độ tiến trình qua đoạn găng.....	36
2.3.2.1	Các giải pháp “Busy waiting”	36
2.3.2.2	Các giải pháp “Sleep and Wakeup”	41
2.3.2.3	Giải pháp dùng Monitors.....	46
2.4	Tắc nghẽn (Deadlock) và chống tắc nghẽn	50
2.4.1	Mô hình hệ thống.....	51
2.4.2	Mô tả DeadLock	51
2.4.2.1	Điều kiện xảy ra tắc nghẽn	51
2.4.2.2	Biểu đồ phân phối tài nguyên.....	52
2.4.3	Các phương thức xử lý tắc nghẽn	53
2.4.4	Ngăn chặn tắc nghẽn (Deadlock Prevention)	54
2.4.5	Tránh khỏi tắc nghẽn	55
2.4.5.1	Trạng thái an toàn (Safe State).	55
2.4.5.2	Giải thuật đồ thị phân phối tài nguyên	57
2.4.5.3	Giải thuật chủ nhà băng (Banker’s Algorithm)	58
2.4.6	Phát hiện tắc nghẽn (Deadlock Detection)	61
2.4.6.1	Mỗi loại tài nguyên có 1 cá thể	62
2.4.6.2	Mỗi loại tài nguyên có nhiều cá thể	62
2.4.6.3	Ví dụ minh họa.....	63
2.4.6.4	Cách sử dụng giải thuật phát hiện deadlock.....	64
2.4.7	Phục hồi hệ thống sau tắc nghẽn.....	64
2.4.7.1	Dừng tiến trình	64
2.4.7.2	Ưu tiên trước tài nguyên.....	65
	Một số câu hỏi và bài tập	65
	Chương 3 QUẢN LÝ BỘ NHỚ.....	68
3.1	Bộ nhớ chính.....	68
3.1.1	Tổng quan	68
3.1.1.1	Liên kết địa chỉ	68
3.1.1.2	Không gian địa chỉ logic và không gian vật lý.....	70
3.1.1.3	Nạp động (Dynamic Loading).....	71
3.1.1.4	Liên kết động.....	71
3.1.2	Các kỹ thuật cấp phát bộ nhớ.....	71
3.1.2.1	Các yêu cầu khi tổ chức lưu trữ tiến trình	71
3.1.2.2	Cấp phát bộ nhớ liên tục (Contiguous Allocation).....	73
3.1.2.3	Cấp phát bộ nhớ không liên tục.....	80
3.2	Kỹ thuật bộ nhớ ảo (Virtual Memory).....	86
3.2.1	Tổng quan	86
3.2.2	Phân trang theo yêu cầu	88

3.2.3	Lỗi trang (page fault)	90
3.2.4	Chiến lược nạp trang.....	92
3.2.5	Chiến lược thay thế trang (Page Replacement)	92
3.2.5.1.	Chiến lược thay thế trang FIFO.....	93
3.2.5.2.	Chiến lược thay thế trang tối ưu (Optimal)	94
3.2.5.3.	Chiến lược thay thế trang LRU	95
3.2.5.4.	Chiến lược thay thế trang xấp xỉ LRU	96
3.2.6	Trì trệ hệ thống	98
	Một số câu hỏi và bài tập	99
	Chương 4. QUẢN LÝ LƯU TRỮ	101
4.1.	Tổ chức hệ thống tập tin.....	101
4.1.1.	Tập tin	101
4.1.2.	Các phương thức truy cập.....	102
4.1.2.1	Các phương pháp lưu trữ file	102
4.1.2.2	Các phương pháp tổ chức, truy nhập file	105
4.1.3.	Cấu trúc thư mục.....	107
4.1.3.1.	Hệ thống thư mục theo cấp bậc	108
4.1.3.2.	Hệ thống thư mục theo dạng đồ thị không chứa chu trình	110
4.1.3.3.	Hệ thống thư mục theo dạng đồ thị tổng quát	112
4.1.4.	Bảo vệ	114
4.1.4.1.	Kiểm soát truy xuất	114
4.1.4.2.	Quản lý khối bị hỏng	115
4.1.4.3.	Backup.....	115
4.1.4.4.	Tính không đổi của hệ thống tập tin.....	115
4.1.4.5.	Các tiếp cận bảo vệ khác	116
4.1.5.	Tập tin chia sẻ.....	116
4.1.5.1.	Nhiều người dùng.....	116
4.1.5.2.	Hệ thống tập tin ở xa	117
4.2.	Cài đặt hệ thống tập tin	117
4.2.1.	Cấu trúc hệ thống tập tin.....	117
4.2.2.	Cài đặt hệ thống tập tin	119
4.2.3.	Hệ thống tập tin ảo.....	122
4.2.4.	Cài đặt thư mục.....	124
4.2.4.1.	Danh sách tuyến tính	124
4.2.4.2.	Bảng băm.....	124
4.2.5.	Các phương pháp cấp phát.....	125
4.2.5.1.	Cấp phát liên tục.....	125
4.2.5.2.	Cấp phát liên kết.....	127
4.2.5.3.	Cấp phát chỉ mục.....	130
4.2.6.	Quản lý không gian trống	132
4.2.6.1.	Bit vector	132

4.2.6.2. Danh sách liên kết	133
4.3. Hệ thống nhập/xuất.....	134
4.3.1. Các khái niệm cơ bản.....	134
4.3.2. Phần cứng nhập/xuất.....	134
4.3.2.1. Thăm dò.....	135
4.3.2.2. Ngắt	136
4.3.2.3. Truy xuất bộ nhớ trực tiếp.....	137
4.3.3. Giao diện nhập/xuất ứng dụng.....	137
4.3.4. Hệ thống con nhập/xuất của nhân (kernel I/O subsystem).....	138
4.3.4.1. Định thời biểu nhập/xuất.....	139
4.3.4.2. Vùng đệm	139
4.3.4.3. Vùng lưu trữ	139
4.3.4.4. Vùng chứa và đặt trước thiết bị.....	140
4.3.4.5. Quản lý lỗi.....	140
4.3.5. Chuyển đổi yêu cầu nhập/xuất từ thao tác phần cứng	140
Một số từ khóa tiếng Anh.....	143
Tài liệu tham khảo.....	144


Chương 1. TỔNG QUAN


Hệ điều hành là một chương trình quản lý phần cứng máy tính. Nó cung cấp nền tảng cho các chương trình ứng dụng và đóng vai trò trung gian giao tiếp giữa người dùng máy tính và phần cứng của máy tính đó. Hệ điều hành thiết lập cho các tác vụ này rất đa dạng. Một vài hệ điều hành thiết kế tiện dụng trong khi một số khác thiết kế hiệu quả hoặc kết hợp cả hai.

Để hiểu hệ điều hành là gì, trước hết chúng ta phải hiểu chúng được phát triển như thế nào. Trong chương này chúng ta điểm lại sự phát triển của hệ điều hành từ những hệ thử nghiệm đầu tiên tới những hệ đa chương và chia sẻ thời gian. Thông qua những giai đoạn khác nhau chúng ta sẽ thấy cách thức mà những thành phần của hệ điều hành được cải tiến như những giải pháp tự nhiên cho những vấn đề trong những hệ thống máy tính ban đầu. Xem xét những lý do phía sau sự phát triển của hệ điều hành cho chúng ta một đánh giá về những tác vụ gì hệ điều hành làm và cách hệ điều hành thực hiện chúng.

1.1. Hệ điều hành là gì?

Một hệ điều hành là một thành phần quan trọng của mọi hệ thống máy tính. Một hệ thống máy tính có thể được chia thành bốn thành phần: phần cứng, hệ điều hành, các chương trình ứng dụng và người dùng.

 **Phần cứng (Hardware):** bao gồm bộ xử lý trung tâm (CPU), bộ nhớ, thiết bị xuất/nhập,... cung cấp tài nguyên cơ bản cho hệ thống.

 **Các chương trình ứng dụng (application programs):** trình biên dịch (compiler), trình soạn thảo văn bản (text editor), hệ cơ sở dữ liệu (database system), trình duyệt Web,... định nghĩa cách mà trong đó các tài nguyên được sử dụng để giải quyết yêu cầu của người dùng.

🚦 **Người dùng (user):** có nhiều loại người dùng khác nhau, thực hiện những yêu cầu khác nhau, do đó sẽ có nhiều ứng dụng khác nhau.

🚦 **Hệ điều hành (operating system):** hay còn gọi là chương trình hệ thống, điều khiển và hợp tác việc sử dụng phần cứng giữa những chương trình ứng dụng khác nhau cho những người dùng khác nhau. Hệ điều hành có thể được khám phá từ hai phía: người dùng và hệ thống.

1.1.1. Tầm nhìn người dùng

Tầm nhìn người dùng của máy tính rất đa dạng bởi giao diện được dùng. Hầu hết những người dùng máy tính ngồi trước máy tính cá nhân gồm có màn hình, bàn phím, chuột và bộ xử lý hệ thống (system unit). Một hệ thống như thế được thiết kế cho một người dùng độc quyền sử dụng tài nguyên của nó để tối ưu hoá công việc mà người dùng đang thực hiện. Trong trường hợp này, hệ điều hành được thiết kế dễ dàng cho việc sử dụng với sự quan tâm về năng lực nhưng không quan tới việc sử dụng tài nguyên. Năng lực thực hiện là quan trọng với người dùng nhưng không là vấn đề nếu hầu hết hệ thống đang rồi, chờ tốc độ xuất/nhập chậm từ phía người dùng.

Vài người dùng ngồi tại thiết bị đầu cuối (terminal) được nối kết tới máy tính lớn (mainframe) hay máy tính tầm trung (minicomputer). Những người khác đang truy xuất cùng máy tính thông qua các thiết bị đầu cuối khác. Những người dùng này chia sẻ các tài nguyên và có thể trao đổi thông tin. Hệ điều hành được thiết kế để tối ưu hoá việc sử dụng tài nguyên-để đảm bảo rằng tất cả thời gian sẵn dùng của CPU, bộ nhớ và thiết bị xuất nhập được sử dụng hữu hiệu và không cá nhân người dùng sử dụng độc quyền tài nguyên hơn là chia sẻ công bằng.

Những người dùng khác ngồi tại trạm làm việc, được nối kết tới mạng của các trạm làm việc khác và máy chủ. Những người dùng này có tài nguyên tận hiến là trạm làm việc của mình nhưng họ cũng chia sẻ các tài nguyên trên mạng và các máy chủ- tập tin, tính toán và các máy phục vụ in. Do đó, hệ điều hành của họ được thiết kế để thoả hiệp giữa khả năng sử dụng cá nhân và việc tận dụng tài nguyên.

1.1.2. Tầm nhìn hệ thống

Từ quan điểm của máy tính, hệ điều hành là chương trình gắn gũi với phần cứng. Chúng ta có thể thấy một hệ điều hành như bộ cấp phát tài nguyên. Hệ thống máy tính có nhiều tài nguyên - phần cứng và phần mềm - mà có thể được yêu cầu để giải quyết một vấn đề: thời gian CPU, không gian bộ nhớ, không gian lưu trữ tập tin, các thiết bị xuất/nhập,... Hệ điều hành hoạt động như bộ quản lý tài nguyên. Đương đầu với một lượng lớn các yêu cầu có thể xung đột về tài nguyên, hệ điều hành phải quyết định cách cấp phát tài nguyên tới những chương trình cụ thể và người dùng để có thể điều hành hệ thống máy tính hữu hiệu và công bằng.

Một tầm nhìn khác của hệ điều hành nhấn mạnh sự cần thiết để điều khiển các thiết bị xuất/nhập khác nhau và chương trình người dùng. Một hệ điều hành là một chương trình điều khiển. Chương trình điều khiển quản lý sự thực thi của các chương trình người dùng để ngăn chặn lỗi và việc sử dụng không hợp lý máy tính. Nó đặc biệt quan tâm với những thao tác và điều khiển các thiết bị nhập/xuất.

Nhìn chung, không có định nghĩa hoàn toàn đầy đủ về hệ điều hành. Các hệ điều hành tồn tại vì chúng là cách hợp lý để giải quyết vấn đề tạo ra một hệ thống máy tính có thể sử dụng. Mục tiêu cơ bản của hệ thống máy tính là thực thi chương trình người dùng và giải quyết vấn đề người dùng dễ dàng hơn. Hướng đến mục tiêu này, phần cứng máy tính được xây dựng. Tuy nhiên, chỉ đơn thuần là phần cứng thì không dễ sử dụng và phát triển các chương trình ứng dụng. Các chương trình khác nhau này đòi hỏi những thao tác chung nào đó, chẳng hạn như điều khiển thiết bị xuất/nhập. Sau đó, những chức năng chung về điều khiển và cấp phát tài nguyên được đặt lại với nhau vào một bộ phận phần mềm gọi là hệ điều hành.

Cũng không có định nghĩa bao quát nào được chấp nhận để xác định phần gì thuộc về hệ điều hành, phần gì không. Một quan điểm đơn giản là mọi thứ liên quan khi chúng ta ra lệnh hệ điều hành nên được xem xét. Tuy nhiên, những yêu cầu về bộ nhớ và những đặc điểm bên trong rất khác nhau trong từng hệ thống. Một định nghĩa bao quát hơn về hệ điều hành là một chương trình chạy liên tục trên máy tính (thường gọi là nhân - kernel), những chương trình còn lại thuộc về chương trình ứng dụng.

1.1.3. Mục tiêu hệ thống

Định nghĩa những gì hệ điều hành làm thì dễ hơn xác định hệ điều hành là gì. **Mục đích chính của hệ điều hành là dễ dàng sử dụng.** Vì sự tồn tại của hệ điều hành hỗ trợ nhiều cho máy tính trong việc đáp ứng các ứng dụng của người dùng. Tầm nhìn này đặc biệt rõ ràng hơn khi nhìn hệ điều hành trên các máy tính cá nhân.

Mục tiêu thứ hai của hệ điều hành là điều hành hữu hiệu hệ thống máy tính. Mục tiêu này đặc biệt quan trọng cho các hệ thống lớn, được chia sẻ, nhiều người dùng. Những hệ thống tiêu biểu này khá đắt, khai thác hiệu quả nhất các hệ thống này luôn là điều mong muốn. Tuy nhiên, hai mục tiêu tiện dụng và hữu hiệu đôi khi mâu thuẫn nhau. Trong quá khứ, xem xét tính hữu hiệu thường quan trọng hơn tính tiện dụng. Do đó, lý thuyết hệ điều hành tập trung nhiều vào việc tối ưu hoá sử dụng tài nguyên tính toán. Hệ điều hành cũng phát triển dần theo thời gian. Thí dụ, UNIX bắt đầu với bàn phím và máy in như giao diện của nó giới hạn tính tiện dụng đối với người dùng. Qua thời gian, phần cứng thay đổi và UNIX được gắn vào phần cứng mới với giao diện thân thiện với người dùng hơn. Nhiều giao diện người dùng đồ hoạ GUIs (graphical user interfaces) được bổ sung cho phép tiện dụng hơn với người dùng trong khi vẫn quan tâm tính hiệu quả.

Thiết kế hệ điều hành là một công việc phức tạp. Người thiết kế gặp phải nhiều sự thỏa hiệp trong thiết kế và cài đặt. Nhiều người tham gia không chỉ mang đến hệ điều hành những lợi điểm mà còn liên tục xem xét và nâng cấp.

Để thấy rõ những hệ điều hành là gì và những gì hệ điều hành làm, chúng ta xem xét cách chúng phát triển trong bốn mươi lăm năm qua. Bằng cách lần theo sự tiến triển, chúng ta có thể xác định những thành phần của hệ điều hành và thấy cách thức và lý do hệ điều hành phát triển như chúng có.

Hệ điều hành và kiến trúc máy tính có mối quan hệ khăng khít nhau. Để dễ dàng sử dụng phần cứng, hệ điều hành được phát triển. Sau đó, các người dùng hệ điều hành đề nghị những chuyển đổi trong thiết kế phần cứng để đơn giản chúng.

Nhìn lại lịch sử gần gũi này, chú trọng cách giải quyết những vấn đề về hệ điều hành để giới thiệu những đặc điểm phần cứng.

1.1.4. Một số định nghĩa Hệ điều hành

Định nghĩa 1: Hệ điều hành là một hệ thống mô hình hoá, mô phỏng hoạt động của máy tính, của người sử dụng và của lập trình viên, hoạt động trong chế độ đối thoại nhằm tạo môi trường khai thác thuận lợi hệ thống máy tính và quản lý tối ưu tài nguyên của hệ thống.

Định nghĩa 2: Hệ điều hành là hệ thống chương trình với các chức năng giám sát, điều khiển việc thực hiện các chương trình của người sử dụng, quản lý và phân chia tài nguyên cho nhiều chương trình người sử dụng đồng thời sao cho việc khai thác chức năng của hệ thống máy tính của người sử dụng là thuận lợi và hiệu quả nhất.

Định nghĩa 3: Hệ điều hành là một chương trình đóng vai trò như là giao diện giữa người sử dụng và phần cứng máy tính, nó điều khiển việc thực hiện của tất cả các loại chương trình. Định nghĩa này rất gần với các hệ điều hành đang sử dụng trên các máy tính hiện nay.

1.2. Phân loại hệ điều hành

1.2.1. Hệ điều hành theo lô đơn giản

Những máy tính thời kỳ đầu là những máy cực lớn chạy từ một thiết bị cuối (console). Những thiết bị nhập thường là những bộ đọc thẻ và các ổ đĩa băng từ. Các thiết bị xuất thông thường thường là những máy in dòng (line printers), các ổ đĩa từ và các phiếu đục lỗ. Người dùng không giao tiếp trực tiếp với các hệ thống máy tính. Thay vào đó, người dùng chuẩn bị một công việc- chứa chương trình, dữ liệu và các thông tin điều khiển về tính tự nhiên của công việc- và gửi nó đến người điều hành máy tính. Công việc này thường được thực hiện trong các phiếu đục lỗ. Tại những thời điểm sau đó (sau vài phút, giờ hay ngày), dữ liệu xuất xuất hiện. Dữ liệu xuất chứa kết quả chương trình cũng như kết xuất bộ nhớ cuối cùng và nội dung các thanh ghi cho việc gỡ rối.

Hệ điều hành trong các máy tính thời kỳ đầu này tương đối đơn giản. Tác vụ chính là chuyển điều khiển tự động từ một công việc này sang công việc khác. Hệ điều hành luôn được thường trú trong bộ nhớ.

Trong môi trường thực thi này, CPU luôn rảnh vì tốc độ của các thiết bị xuất/nhập dạng cơ thực chất chậm hơn tốc độ của các thiết bị điện. Ngay cả một CPU chậm (với hàng ngàn chỉ thị lệnh được thực thi trên giây) cũng chỉ làm việc trong vài phần trăm giây. Thêm vào đó, một bộ đọc thẻ nhanh có thể đọc 1200 thẻ trong thời gian 1 phút (hay 20 thẻ trên giây). Do đó, sự khác biệt giữa tốc độ CPU và thiết bị xuất/nhập của nó có thể là 3 lần hay nhiều hơn. Dĩ nhiên theo thời gian, sự tiến bộ trong công nghệ dẫn đến sự ra đời những thiết bị nhập/xuất nhanh hơn. Tuy nhiên, tốc độ CPU tăng tới một tỷ lệ lớn hơn vì thế vấn đề không những không được giải quyết mà còn làm tăng.

1.2.2. Hệ điều hành theo lô đa chương

Một khía cạnh quan trọng nhất của định thời công việc là khả năng đa chương. Thông thường, một người dùng giữ CPU hay các thiết bị xuất/nhập luôn bận. Đa chương gia tăng khả năng sử dụng CPU bằng cách tổ chức các công việc để CPU luôn có một công việc để thực thi.

Ý tưởng của kỹ thuật đa chương có thể minh họa như sau: Hệ điều hành giữ nhiều tác vụ trong bộ nhớ tại một thời điểm. Tập hợp các tác vụ này là tập con của các tác vụ được giữ trong vùng tác vụ -bởi vì số lượng các tác vụ có thể được giữ cùng lúc trong bộ nhớ thường nhỏ hơn số tác vụ có thể có trong vùng đệm. Hệ điều hành sẽ lấy và bắt đầu thực thi một trong các tác vụ có trong bộ nhớ. Giả sử, tác vụ đang thực hiện phải chờ một vài tác vụ như một thao tác xuất/nhập để hoàn thành. Trong hệ thống đơn chương, CPU sẽ chờ ở trạng thái rỗi. Trong hệ thống đa chương, hệ điều hành sẽ chuyển sang thực thi một tác vụ khác. Khi tác vụ đầu tiên kết thúc việc chờ, nó sẽ nhận CPU trở lại. Chỉ cần có ít nhất một tác vụ cần thực thi, CPU sẽ không bao giờ ở trạng thái rỗi.

Đa chương là một trường hợp đầu tiên khi hệ điều hành phải thực hiện quyết định cho những người dùng. Do đó, hệ điều hành đa chương tương đối tinh vi. Tất cả tác vụ đưa vào hệ thống được giữ trong vùng tác vụ. Vùng này chứa tất cả tiến trình định vị trên đĩa chờ được cấp phát bộ nhớ chính. Nếu nhiều tác vụ sẵn sàng được mang vào bộ nhớ và nếu không đủ không gian cho tất cả thì hệ điều hành phải chọn một trong số chúng. Khi hệ điều hành chọn một tác vụ từ vùng tác vụ, nó nạp tác vụ đó vào bộ nhớ để thực thi. Có nhiều chương trình trong bộ nhớ tại cùng thời điểm yêu cầu phải có sự quản lý bộ nhớ. Ngoài ra, nếu nhiều tác vụ sẵn sàng chạy cùng thời điểm, hệ thống phải chọn một trong số chúng. Việc thực hiện quyết định này là định thời CPU (lập lịch CPU).

1.2.3. Hệ điều hành chia sẻ thời gian

Hệ thống theo lô đa chương cung cấp một môi trường nơi mà nhiều tài nguyên khác nhau (chẳng hạn như CPU, bộ nhớ, các thiết bị ngoại vi) được sử dụng hiệu quả. Tuy nhiên, nó không cung cấp giao tiếp người dùng với hệ thống máy tính. Chia sẻ thời gian (hay đa nhiệm) là sự mở rộng logic của đa chương. CPU thực thi nhiều tác vụ bằng cách chuyển đổi qua lại giữa chúng, và những chuyển đổi này có thể xảy ra quá thường xuyên để người dùng có thể giao tiếp với mỗi chương trình trong khi chạy.

Một hệ thống máy tính giao tiếp (interactive computer) hay thực hành (hands-on computer system) cung cấp giao tiếp trực tuyến giữa người dùng và hệ thống. Người dùng cho những chỉ thị tới hệ điều hành hay trực tiếp tới một chương trình, sử dụng bàn phím hay chuột và chờ nhận kết quả tức thì. Do đó, thời gian đáp ứng thường là rất ngắn-diễn hình trong phạm vi 1 giây hay ít hơn.

Một hệ thống chia sẻ thời gian cho phép nhiều người dùng chia sẻ CPU cùng một thời điểm. Vì mỗi hoạt động hay lệnh trong hệ chia sẻ thời gian được phục vụ ngắn, chỉ một ít thời gian CPU được yêu cầu cho mỗi người dùng. Khi hệ thống nhanh chóng chuyển từ một người dùng này sang người dùng tiếp theo, mỗi người

dùng được cho cảm giác rằng toàn bộ hệ thống máy tính được cung cấp cho mình, nhưng thật sự máy tính đó đang được chia sẻ cho nhiều người dùng.

Một hệ điều hành chia sẻ thời gian sử dụng lập lịch CPU và đa chương để cung cấp mỗi người dùng với một phần nhỏ của máy tính chia sẻ thời gian. Mỗi người dùng có ít nhất một chương trình riêng trong bộ nhớ. Một chương trình được nạp vào trong bộ nhớ và thực thi thường được gọi là một tiến trình. Khi một tiến trình thực thi, nó có thể thực thi chỉ tại một khoảng thời gian ngắn hay cần thực hiện xuất/nhập. Vì giao tiếp xuất/nhập chủ yếu chạy ở “tốc độ người dùng”, nó có thể mất một khoảng thời gian dài để hoàn thành. Ví dụ, dữ liệu nhập có thể bị giới hạn bởi tốc độ nhập của người dùng; 7 ký tự trên giây là nhanh đối với người dùng, nhưng quá chậm so với máy tính. Vì vậy, thay vì để CPU rảnh khi người dùng nhập dữ liệu, hệ điều hành sẽ nhanh chóng chuyển CPU tới một tiến trình khác.

Hệ điều hành chia sẻ thời gian phức tạp hơn nhiều so với hệ điều hành đa chương. Trong cả hai dạng, nhiều tác vụ được giữ cùng lúc trong bộ nhớ, vì thế hệ thống phải có cơ chế quản lý bộ nhớ và bảo vệ. Để đạt được thời gian đáp ứng hợp lý, các tác vụ có thể được hoán vị vào/ra bộ nhớ chính. Một phương pháp chung để đạt mục tiêu này là sử dụng bộ nhớ ảo, là kỹ thuật cho phép việc thực thi của một tác vụ có thể không hoàn toàn ở trong bộ nhớ. Ưu điểm chính của cơ chế bộ nhớ ảo là các chương trình có thể lớn hơn bộ nhớ vật lý. Ngoài ra, nó trừu tượng hoá bộ nhớ chính thành mảng lưu trữ lớn và đồng nhất, chia bộ nhớ logic như được thấy bởi người dùng từ bộ nhớ vật lý. Sự sắp xếp này giải phóng lập trình viên khỏi việc quan tâm đến giới hạn lưu trữ của bộ nhớ.

Ý tưởng chia thời được giới thiệu trong những năm 1960, nhưng vì hệ chia thời là phức tạp và rất đắt để xây dựng, chúng không phổ biến cho tới những năm 1970. Mặc dù xử lý theo lô vẫn được thực hiện nhưng hầu hết hệ thống ngày nay là chia sẻ thời gian. Do đó, đa chương và chia sẻ thời gian là những chủ đề trung tâm của hệ điều hành hiện đại và cũng là chủ đề trọng tâm của giáo trình này.

1.2.4. Hệ điều hành mạng

Là các hệ điều hành dung để điều khiển sự hoạt động của mạng máy tính. Vì vậy, ngoài các chức năng cơ bản của một hệ điều hành, các hệ điều hành mạng còn phải thực hiện việc chia sẻ và bảo vệ tài nguyên của mạng.

Các mạng với sự đa dạng về giao thức được dùng, khoảng cách giữa các nút và phương tiện truyền. TCP/IP là giao thức mạng phổ biến nhất mặc dù ATM và các giao thức khác được sử dụng rộng rãi. Tương tự, hệ điều hành hỗ trợ sự đa dạng về giao thức. Hầu hết các hệ điều hành hỗ trợ TCP/IP, gồm Windows và UNIX. Một số hệ điều hành khác hỗ trợ các giao thức riêng phù hợp với yêu cầu của chúng. Đối với một hệ điều hành, một giao thức mạng chỉ cần một thiết bị giao diện – ví dụ: một card mạng - với một trình điều khiển thiết bị để quản lý nó và một phần mềm để đóng gói dữ liệu trong giao thức giao tiếp để gửi nó và mở gói để nhận nó.

1.3. Một số khái niệm của hệ điều hành

Tiến trình (Process): Tiến trình là một bộ phận của chương trình đang thực hiện. Tiến trình là đơn vị làm việc cơ bản của hệ thống, trong hệ thống có thể tồn

tại nhiều tiến trình cùng hoạt động, trong đó có cả tiến trình của hệ điều hành và tiến trình của chương trình người sử dụng. Các tiến trình này có thể hoạt động đồng thời với nhau.

Tiểu trình (Thread): Tiểu trình cũng là đơn vị xử lý cơ bản trong hệ thống, nó cũng xử lý tuần tự đoạn code của nó, nó cũng sở hữu một con trỏ lệnh, một tập các thanh ghi và một vùng nhớ stack riêng và các tiểu trình cũng chia sẻ thời gian xử lý của processor như các tiến trình.

Bộ xử lý lệnh (Shell): Shell là một bộ phận hay một tiến trình đặc biệt của hệ điều hành, nó có nhiệm vụ nhận lệnh của người sử dụng, phân tích lệnh và phát sinh tiến trình mới để thực hiện yêu cầu của lệnh, tiến trình mới này được gọi là tiến trình đáp ứng yêu cầu.

Sự phân lớp hệ thống (System Layering): Như đã biết, hệ điều hành là một hệ thống các chương trình bao quanh máy tính thực (vật lý) nhằm tạo ra một máy tính mở rộng (logic) đơn giản và dễ sử dụng hơn. Theo đó, khi khai thác máy tính người sử dụng chỉ cần tác động vào lớp vỏ bọc bên ngoài của máy tính, mọi sự giao tiếp giữa lớp vỏ bọc này với các chi tiết phần cứng bên trong đều do hệ điều hành thực hiện.

Tài nguyên hệ thống (System Resources): là những tài nguyên tồn tại về mặt vật lý tại một thời điểm nhất định hoặc tại mọi thời điểm, và nó có khả năng tác động đến hiệu suất của hệ thống máy tính. Một cách tổng quát, có thể chia tài nguyên hệ thống thành 2 loại cơ bản:

- Tài nguyên không gian: là các không gian lưu trữ của hệ thống như đĩa, bộ nhớ chính, quan trọng nhất là không gian bộ nhớ chính, nơi lưu trữ các chương trình đang được CPU thực hiện.
- Tài nguyên thời gian: chính là thời gian thực hiện lệnh của processor và thời gian truy xuất dữ liệu trên bộ nhớ.

Lời gọi hệ thống (System Calls):

Để tạo môi trường giao tiếp giữa chương trình của người sử dụng và hệ điều hành, hệ điều hành đưa ra các lời gọi hệ thống. Chương trình của người sử dụng dùng các lời gọi hệ thống để liên lạc với hệ điều hành và yêu cầu các dịch vụ từ hệ điều hành.

Mỗi lời gọi hệ thống tương ứng với một thủ tục trong thư viện của hệ điều hành, do đó chương trình của người sử dụng có thể gọi thủ tục để thực hiện một lời gọi hệ thống. Lời gọi hệ thống còn được thiết dưới dạng các câu lệnh trong các ngôn ngữ lập trình cấp thấp. Lệnh gọi ngắt trong hợp ngữ (Int), và thủ tục gọi hàm API trong windows được xem là một lời gọi hệ thống.

1.4. Các thành phần của hệ điều hành

Thành phần quản lý tiến trình:

- Tạo lập, hủy bỏ tiến trình.
- Tạm dừng, tái kích hoạt tiến trình.

- Tạo cơ chế thông tin liên lạc giữa các tiến trình.
- Tạo cơ chế đồng bộ hóa giữa các tiến trình.

Thành phần quản lý bộ nhớ chính:

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

Thành phần quản lý xuất/nhập.

Thành phần quản lý bộ nhớ phụ (đĩa):

- Quản lý không gian trống trên đĩa.
- Định vị lưu trữ thông tin trên đĩa.
- Lập lịch cho vấn đề ghi/ đọc thông tin trên đĩa của đầu từ.

Thành phần quản lý tập tin

Thành phần thông dịch lệnh

Thành phần bảo vệ hệ thống

Một số câu hỏi ôn tập

Câu 1. Cho biết hai chức năng chính của Hệ điều hành.

Câu 2. Ưu điểm của hệ điều hành chia sẻ thời gian?

Câu 3. Phân biệt hai khái niệm Shell và lời gọi hệ thống (System Call)

Câu 4. Các loại tài nguyên nào được gọi là tài nguyên thời gian?

Câu 5. Tìm hiểu chi tiết các thành phần của hệ điều hành.

Câu 6. Ưu điểm chính đối với người thiết kế hệ điều hành khi sử dụng cấu trúc máy ảo? Đối với người dùng thì ưu điểm chính là gì?

Câu 7. Tìm hiểu chi tiết các cấu trúc tổng quát của hệ điều hành? Cụ thể với hệ điều hành Linux.


Chương 2 QUẢN LÝ TIẾN TRÌNH

Trong hệ điều hành đa nhiệm hiện nay, cho phép nhiều chương trình được nạp vào bộ nhớ và được thực thi đồng hành. Điều này dẫn đến yêu cầu sự điều khiển mạnh mẽ hơn và phân chia nhiều hơn giữa các tiến trình. Tiến trình là một đơn vị công việc trong một hệ điều hành phân chia thời gian hiện đại.

Mặc dù quan tâm chủ yếu của hệ điều hành là thực thi chương trình người sử dụng, nhưng nó cũng quan tâm đến các tác vụ khác. Do đó, một hệ thống chứa tập hợp các tiến trình: tiến trình hệ điều hành thực thi mã hệ thống, tiến trình người sử dụng thực thi mã người sử dụng. Tất cả tiến trình này có tiềm năng thực thi đồng hành, với một CPU (hay nhiều CPU) được đa hợp giữa chúng. Bằng cách chuyển đổi CPU giữa các tiến trình, hệ điều hành có thể làm cho máy tính hoạt động với năng suất cao hơn. Điều này có nghĩa là các tiến trình này luân phiên giữa các trạng thái: sử dụng processor và đợi thực hiện vào/ra hay một vài sự kiện nào đó xảy ra. Nội dung của chương này sẽ giúp người đọc nắm được những kiến thức liên quan đến quản lý tiến trình trong các hệ điều hành hiện nay, như: các khái niệm cơ bản về tiến trình, lập lịch (điều phối) tiến trình, đồng bộ tiến trình, xử lý tắc nghẽn,

2.1 Tổng quan về tiến trình


2.1.1 Khái niệm về tiến trình

 **Tiến trình (Process):** Tiến trình là một chương trình đang thi hành, đơn vị thực hiện tiến trình là processer. Nói cách khác, một tiến trình không chỉ là một chương trình, nó còn bao gồm hoạt động hiện hành như được hiện diện bởi giá trị của bộ đếm chương trình và nội dung các thanh ghi của bộ xử lý. Ngoài ra, một tiến trình thường chứa ngăn xếp tiến trình, dữ liệu tạm thời (như các tham số, các địa chỉ trở về, các biến cục bộ) và phần dữ liệu chứa các biến toàn cục.

Một số khái niệm khác:

- Tiến trình là một chương trình do một processor logic thực hiện (Saltzer).
- Tiến trình là một quá trình chuyển từ trạng thái này sang trạng thái khác dưới tác động của hàm hành động, xuất phát từ một trạng thái ban đầu nào đó (Horning & Rendell).

Chúng ta nhấn mạnh rằng, một chương trình không phải là một tiến trình; một chương trình là một thực thể thụ động, như nội dung của các tập tin được lưu trên đĩa, trái lại một tiến trình là một thực thể chủ động, với một bộ đếm chương trình xác định chỉ thị lệnh tiếp theo sẽ thực thi và tập hợp tài nguyên có liên quan.


 **Tiểu trình:** Thông thường mỗi tiến trình có một không gian địa chỉ và một dòng xử lý. Nhưng trong thực tế có một số ứng dụng cần nhiều dòng xử lý cùng chia sẻ một không gian địa chỉ tiến trình, các dòng xử lý này có thể hoạt động song song với nhau như các tiến trình độc lập trên hệ thống. Để thực hiện được điều này các hệ điều hành hiện nay đưa ra một cơ chế thực thi (các chỉ thị trong chương trình) mới, được gọi là tiểu trình.

Tiểu trình là một đơn vị xử lý cơ bản trong hệ thống, nó hoàn toàn tương tự như

tiến trình. Tức là nó cũng phải xử lý tuần tự các chỉ thị máy của nó, nó cũng sở hữu con trỏ lệnh, một tập các thanh ghi, và một không gian stack riêng.

Một tiến trình đơn có thể bao gồm nhiều tiểu trình. Các tiểu trình trong một tiến trình chia sẻ một không gian địa chỉ chung, nhờ đó mà các tiểu trình có thể chia sẻ các biến toàn cục của tiến trình và có thể truy xuất lên các vùng nhớ stack của nhau.

Các tiểu trình chia sẻ thời gian xử lý của processor giống như cách của tiến trình, nhờ đó mà các tiểu trình có thể hoạt động song song (giả) với nhau. Trong quá trình thực thi của tiểu trình nó cũng có thể tạo ra các tiến trình con của nó.

 **Đa tiểu trình trong đơn tiến trình:** Điểm đáng chú ý nhất của mô hình tiểu trình là: có nhiều tiểu trình trong phạm vi một tiến trình đơn. Các tiến trình đơn này có thể hoạt động trên các hệ thống multiprocessor hoặc uniprocessor. Các hệ điều hành khác nhau có cách tiếp cận mô hình tiểu trình khác nhau. Ở đây chúng ta tiếp cận mô hình tiểu trình từ mô hình tác vụ (Task), đây là các tiếp cận của windows NT và các hệ điều hành đa nhiệm khác. Trong các hệ điều hành này tác vụ được định nghĩa như là một đơn vị của sự bảo vệ hay đơn vị cấp phát tài nguyên. Trong hệ thống tồn tại một không gian địa chỉ ảo để lưu giữ tác vụ và một cơ chế bảo vệ sự truy cập đến các file, các tài nguyên Vào/Ra và các tiến trình khác (trong các thao tác truyền thông liên tiến trình).

Trong phạm vi một tác vụ, có thể có một hoặc nhiều tiểu trình, mỗi tiểu trình bao gồm: Một trạng thái thực thi tiểu trình (running, ready,...). Một lưu trữ về ngữ cảnh của processor khi tiểu trình ở trạng thái not running (một cách để xem tiểu trình như một bộ đếm chương trình độc lập hoạt động trong phạm vi tác vụ). Các thông tin thống kê về việc sử dụng các biến cục bộ của tiểu trình. Một stack thực thi. Truy xuất đến bộ nhớ và tài nguyên của tác vụ, được chia sẻ với tất cả các tiểu trình khác trong tác vụ.

Trong các ứng dụng server, chẳng hạn như ứng dụng file server trên mạng cục bộ, khi có một yêu cầu hình thành một file mới, thì một tiểu trình mới được hình thành từ chương trình quản lý file. Vì một server sẽ phải điều khiển nhiều yêu cầu, có thể đồng thời, nên phải có nhiều tiểu trình được tạo ra và được giải phóng trong, có thể đồng thời, một khoảng thời gian ngắn. Nếu server là một hệ thống multiprocessor thì các tiểu trình trong cùng một tác vụ có thể thực hiện đồng thời trên các processor khác nhau, do đó hiệu suất của hệ thống tăng lên. Sự hình thành các tiểu trình này cũng thật sự hữu ích trên các hệ thống uniprocessor, trong trường hợp một chương trình phải thực hiện nhiều chức năng khác nhau. Hiệu quả của việc sử dụng tiểu trình được thấy rõ trong các ứng dụng cần có sự truyền thông giữa các tiến trình hoặc các chương trình khác nhau.

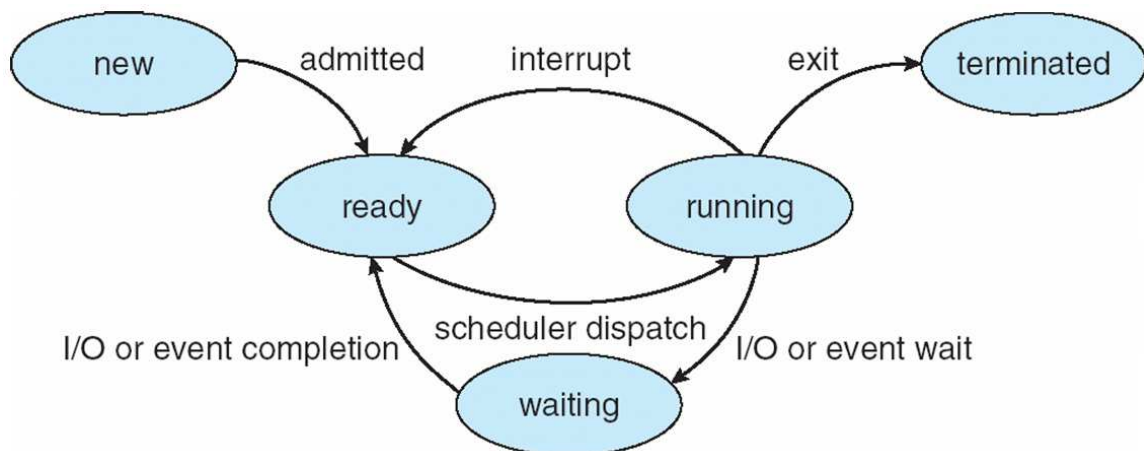
Các thao tác lập lịch và điều phối tiến trình của hệ điều hành thực hiện trên cơ sở tiểu trình. Nhưng nếu có một thao tác nào đó ảnh hưởng đến tất cả các tiểu trình trong tác vụ thì hệ điều hành phải tác động vào tác vụ.

Vì tất cả các tiểu trình trong một tác vụ chia sẻ cùng một không gian địa chỉ, nên tất cả các tiểu trình phải được đưa vào trạng thái suspend tại cùng thời điểm. Tương tự, khi một tác vụ kết thúc thì sẽ kết thúc tất cả các tiểu trình trong tác vụ

đó. Trạng thái suspend sẽ được giải thích ngay sau đây.

2.1.2. Trạng thái tiến trình

Khi một tiến trình thực thi, nó thay đổi trạng thái. Trạng thái của tiến trình được định nghĩa bởi các hoạt động hiện hành của tiến trình đó. Mỗi tiến trình có thể ở một trong những trạng thái sau (hình 2.1):



Hình 2.1. Lược đồ chuyển trạng thái của tiến trình

- New: Tiến trình đang được tạo ra.
- Running: Các chỉ thị đang được thực thi (đang sở hữu vi xử lý (VXL)).
- Waiting (hay blocked): Tiến trình đang đợi một (hoặc một vài) sự kiện xảy ra (như hoàn thành việc nhập/xuất hay nhận tín hiệu).
- Ready: Tiến trình đang đợi để được cấp phát vi xử lý.
- Terminated: Tiến trình hoàn thành việc thực thi

Tại một thời điểm xác định chỉ có duy nhất một tiến trình ở trạng thái Running (trên bất kỳ bộ vi xử lý nào), nhưng có thể có nhiều tiến trình ở trạng thái Ready và Waiting. Các tiến trình ở trạng thái Ready và Blocked được chứa trong các hàng đợi (Queue) riêng.

Quá trình chuyển trạng thái diễn ra như sau:

1. (Admitted) Tiến trình được khởi tạo, được đưa vào hệ thống, được cấp phát đầy đủ tài nguyên (chỉ thiếu vi xử lý).
2. (Scheduler dispatch) Tiến trình được cấp VXL để bắt đầu thực hiện.
3. (Exit) Tiến trình hoàn thành xử lý và kết thúc.
4. (Interrupt) Tiến trình bị bộ điều phối tiến trình thu hồi VXL, do hết thời gian được quyền sử dụng VXL, để cấp phát cho tiến trình khác.
5. (I/O or event wait) Tiến trình đang chờ một sự kiện nào đó xảy ra hay đang chờ một thao tác vào/ra kết thúc hay tài nguyên mà tiến trình yêu cầu chưa được hệ điều hành đáp ứng.

6. (I/O or event completion) Sự kiện mà tiến trình chờ đã xảy ra, thao tác vào/ra mà tiến trình đợi đã kết thúc, hay tài nguyên mà tiến trình yêu cầu đã được hệ điều hành đáp ứng.

Có nhiều lý do để một tiến trình đang ở trạng thái running chuyển sang trạng thái blocked, do đó đa số các hệ điều hành đều thiết kế một hệ thống hàng đợi gồm nhiều hàng đợi, mỗi hàng đợi dùng để chứa những tiến trình đang đợi cùng một sự kiện nào đó.

✓ Trong môi trường hệ điều hành đa nhiệm thì việc tổ chức các Queue để lưu các tiến trình chưa thể hoạt động là cần thiết, nhưng nếu tồn tại quá nhiều tiến trình trong Queue, hay chính xác hơn trong bộ nhớ chính, sẽ dẫn đến tình trạng lãng phí bộ nhớ, không còn đủ bộ nhớ để nạp các tiến trình khác khi cần thiết. Mặt khác nếu các tiến trình trong Queue đang chiếm giữ tài nguyên của hệ thống, mà những tài nguyên này lại là những tài nguyên các tiến trình khác đang cần, điều này dẫn đến tình trạng sử dụng tài nguyên không hợp lý, làm cho hệ thống thiếu tài nguyên (thực chất là thừa) trầm trọng và có thể làm cho hệ thống tắc nghẽn. Với những lý do trên các hệ điều hành đa nhiệm thiết kế thêm một trạng thái tiến trình mới, đó là trạng thái Nonresident (bảng 2.1).

Bảng 2.1. Bảng chuyển đổi trạng thái với trạng thái Nonresident

	READY	RUN	BLOCKED	NONRESIDENT
READY		1		5
RUN	2		3	
BLOCKED	4			6

Trạng thái Nonresident là trạng thái của một tiến trình khi nó đang được lưu trữ trên bộ nhớ phụ, hay chính xác hơn đây là các tiến trình đang ở trong trạng thái blocked và/hoặc ready bị hệ điều hành chuyển ra đĩa để thu hồi lại không gian nhớ đã cấp cho tiến trình hoặc thu hồi lại tài nguyên đã cấp cho tiến trình để cấp cho một tiến trình khác đang rất cần được nạp vào bộ nhớ tại thời điểm hiện tại.

2.1.3. Mô hình tiến trình

Trong mô hình tiến trình hệ điều hành chia chương trình thành nhiều tiến trình, khởi tạo và đưa vào hệ thống nhiều tiến trình của một chương trình hoặc của nhiều chương trình khác nhau, cấp phát đầy đủ tài nguyên (trừ processor) cho tiến trình và đưa các tiến trình sang trạng thái sẵn sàng. Hệ điều hành bắt đầu cấp processor cho một tiến trình trong số các tiến trình ở trạng thái sẵn sàng để tiến trình này hoạt động, sau một khoảng thời gian nào đó hệ điều hành thu hồi processor của tiến trình này để cấp cho một tiến trình sẵn sàng khác, sau đó hệ điều hành lại thu hồi processor từ tiến trình mà nó vừa cấp để cấp cho tiến trình khác, có thể là tiến trình mà trước đây bị hệ điều hành thu hồi processor khi nó chưa kết thúc, và cứ như thế cho đến khi tất cả các tiến trình mà hệ điều hành khởi tạo đều hoạt động và kết thúc được. Điều đáng chú ý trong mô hình tiến trình này là khoảng thời gian chuyển

processor từ tiến trình này sang tiến trình khác hay khoảng thời gian giữa hai lần được cấp phát processor của một tiến trình là rất nhỏ nên các tiến trình có cảm giác luôn được sở hữu processor (logic) hay hệ thống có cảm giác các tiến trình/ chương trình hoạt động song song nhau. Hiện tượng này được gọi là sự song song giả.

Rõ ràng với mô hình tiến trình hệ thống có được 2 điều lợi:

- Tiết kiệm được bộ nhớ: vì không phải nạp tất cả chương trình vào bộ nhớ mà chỉ nạp các tiến trình cần thiết nhất, sau đó tùy theo yêu cầu mà có thể nạp tiếp các tiến trình khác.
- Cho phép các chương trình hoạt động song song nên tốc độ xử lý của toàn hệ thống tăng lên và khai thác tối đa thời gian xử lý của processor.

Việc chọn thời điểm dừng của tiến trình đang hoạt động (đang chiếm giữ processor) để thu hồi processor chuyển cho tiến trình khác hay việc chọn tiến trình tiếp theo nào trong số các tiến trình đang ở trạng thái sẵn sàng để cấp processor là những vấn đề khá phức tạp đòi hỏi hệ điều hành phải có một cơ chế điều phối thích hợp thì mới có thể tạo ra được hiệu ứng song song giả và sử dụng tối ưu thời gian xử lý của processor. Bộ phận thực hiện chức năng này của hệ điều hành được gọi là bộ điều phối (dispatcher) tiến trình.

2.1.4. Khởi điều khiển tiến trình (PCB)

Để quản lý các tiến trình và tài nguyên trong hệ thống, hệ điều hành phải có các thông tin về trạng thái hiện thời của mỗi tiến trình và tài nguyên. Trong trường hợp này hệ điều hành xây dựng và duy trì các bảng thông tin về mỗi đối tượng (memory, devices, file, process) mà nó quản lý, đó là các bảng: memory table cho đối tượng bộ nhớ, I/O table cho đối tượng thiết bị vào/ra, file table cho đối tượng tập tin, process table cho đối tượng tiến trình. Memory table được sử dụng để theo dõi cả bộ nhớ thực lẫn bộ nhớ ảo, nó phải bao gồm các thông tin sau: Không gian bộ nhớ chính dành cho tiến trình. Không gian bộ nhớ phụ dành cho tiến trình. Các thuộc tính bảo vệ bộ nhớ chính và bộ nhớ ảo. Các thông tin cần thiết để quản lý bộ nhớ ảo. Ở đây chúng tôi điểm qua một vài thông tin về memory table, là để lưu ý với các bạn rằng: nhiệm vụ quản lý tiến trình và quản lý bộ nhớ của hệ điều hành có quan hệ chéo với nhau, bộ phận quản lý tiến trình cần phải có các thông tin về bộ nhớ để điều khiển sự hoạt động của tiến trình, ngược lại bộ phận quản lý bộ nhớ phải có các thông tin về tiến trình để tổ chức nạp tiến trình vào bộ nhớ, ... Điều này cũng đúng với các bộ phận quản lý Vào/ ra và quản lý tập tin. Trong phần trình bày sau đây chúng tôi chỉ đề cập đến Process Table của hệ điều hành.

Để quản lý và điều khiển được một tiến trình, thì hệ điều hành phải biết được vị trí nạp tiến trình trong bộ nhớ chính, phải biết được các thuộc tính của tiến trình cần thiết cho việc quản lý tiến trình của nó:

➤ **Định vị của tiến trình (process location):** định vị của tiến trình phụ thuộc vào chiến lược quản lý bộ nhớ đang sử dụng. Trong trường hợp đơn giản nhất, tiến trình, hay chính xác hơn là hình ảnh tiến trình, được lưu giữa tại các khối nhớ liên tục trên bộ nhớ phụ (thường là đĩa), để tiến trình thực hiện được thì tiến trình phải được nạp vào bộ nhớ chính. Do đó, hệ điều hành cần phải biết định vị của mỗi tiến

trình trên đĩa và cho mỗi tiến trình đó trên bộ nhớ chính. Trong một số chiến lược quản lý bộ nhớ, hệ điều hành chỉ cần nạp một phần tiến trình vào bộ nhớ chính, phần còn lại vẫn nằm trên đĩa. Hay tiến trình đang ở trên bộ nhớ chính thì có một phần bị swap-out ra lại đĩa, phần còn lại vẫn còn nằm ở bộ nhớ chính. Trong các trường hợp này hệ điều hành phải theo dõi tiến trình để biết phần nào của tiến trình là đang ở trong bộ nhớ chính, phần nào của tiến trình là còn ở trên đĩa.

Đa số các hệ điều hành hiện nay đều sử dụng chiến lược quản lý bộ nhớ mà trong đó không gian địa chỉ của tiến trình là một tập các block, các block này có thể không liên tiếp nhau. Tùy theo chiến lược bộ nhớ sử dụng mà các block này có thể có chiều dài cố định (chiến lược phân phân trang bộ nhớ) hay thay đổi (chiến lược phân đoạn bộ nhớ) hay kết hợp cả hai. Hệ điều hành cho phép không nạp tất cả các trang (page) và/hoặc các đoạn (segment) của tiến trình vào bộ nhớ. Do đó, process table phải được duy trì bởi hệ điều hành và phải cho biết vị trí của mỗi trang/ đoạn tiến trình trên hệ thống. Những điều trên đây sẽ được làm rõ ở phần chiến lược cấp phát bộ nhớ trong chương *Quản lý bộ nhớ* của tài liệu này.

➤ **Các thuộc tính của tiến trình:** Trong các hệ thống đa chương, thông tin về mỗi tiến trình là rất cần cho công tác quản lý tiến trình của hệ điều hành, các thông tin này có thể thường trú trong khối quản lý tiến trình (PCB: process control block). Các hệ điều hành khác nhau sẽ có cách tổ chức PCB khác nhau, ở đây chúng ta khảo sát một trường hợp chung nhất. Các thông tin trong PCB có thể được chia thành ba nhóm chính:

- Định danh tiến trình (PID: process identification): mỗi tiến trình được gán một định danh duy nhất để phân biệt với các tiến trình khác trong hệ thống. Định danh của tiến trình có thể xuất hiện trong memory table, I/O table. Khi tiến trình này truyền thông với tiến trình khác thì định danh tiến trình được sử dụng để hệ điều hành xác định tiến trình đích. Khi tiến trình cho phép tạo ra tiến trình khác thì định danh được sử dụng để chỉ đến tiến trình cha và tiến trình con của mỗi tiến trình. Tóm lại, các định danh có thể lưu trữ trong PCB bao gồm: định danh của tiến trình này, định danh của tiến trình tạo ra tiến trình này, định danh của người sử dụng.

- Thông tin trạng thái processor (processor state information): bao gồm các thanh ghi User-visible, các thanh ghi trạng thái và điều khiển, các con trỏ stack.

- Thông tin điều khiển tiến trình (process control information): bao gồm thông tin trạng thái và lập lịch, cấu trúc dữ liệu, truyền thông liên tiến trình, quyền truy cập tiến trình, quản lý bộ nhớ, tài nguyên khởi tạo và tài nguyên sinh ra.

PCB là một trong những cấu trúc dữ liệu trung tâm và quan trọng của hệ điều hành. Mỗi PCB chứa tất cả các thông tin về tiến trình mà nó rất cần cho hệ điều hành. Có nhiều modun thành phần trong hệ điều hành có thể read và/hoặc modified PCB như: lập lịch tiến trình, cấp phát tài nguyên cho tiến trình, ngắt tiến trình, vv. Có thể nói các thiết lập trong PCB định nghĩa trạng thái của hệ điều hành.

2.1.5. Các thao tác điều khiển tiến trình

➤ **Khi khởi tạo tiến trình hệ điều hành thực hiện các thao tác sau:**

- Hệ điều hành gán PID cho tiến trình mới và đưa tiến trình vào danh sách quản lý của hệ thống, tức là, dùng một entry trong PCB để chứa các thông tin liên quan đến tiến trình mới tạo ra này.

- Cấp phát không gian bộ nhớ cho tiến trình. Ở đây hệ điều hành cần phải xác định được kích thước của tiến trình, bao gồm code, data và stack. Giá trị kích thước này có thể được gán mặt định dựa theo loại của tiến trình hoặc được gán theo yêu cầu của người sử dụng khi có một công việc (job) được tạo. Nếu một tiến trình được sinh ra bởi một tiến trình khác, thì tiến trình cha có thể chuyển kích thước của nó đến hệ điều hành trong yêu cầu tạo tiến trình.

- Khởi tạo các thông tin cần thiết cho khối điều khiển tiến trình như các PID của tiến trình cha (nếu có), thông tin trạng thái tiến trình, độ ưu tiên của tiến trình, thông tin ngữ cảnh của processor (bộ đến chương trình và các thanh ghi khác), vv.

- Cung cấp đầy đủ các tài nguyên cần thiết nhất, trừ processor, để tiến trình có thể vào trạng thái ready được hoặc bắt đầu hoạt động được.

- Đưa tiến trình vào một danh sách tiến trình nào đó: ready list, suspend list, waiting list, vv, sao cho phù hợp với chiến lược điều phối tiến trình hiện tại của bộ phận điều phối tiến trình của hệ điều hành.

Khi một tiến trình tạo lập một tiến trình con, tiến trình con có thể được cấp phát tài nguyên bởi chính hệ điều hành, hoặc được tiến trình cha cho thừa hưởng một số tài nguyên ban đầu của nó.

➤ **Khi kết thúc tiến trình hệ điều hành thực hiện các thao tác sau:** Khi tiến trình kết thúc xử lý, hoàn thành chỉ thị cuối cùng, hệ điều hành sẽ thực hiện các thao tác sau đây:

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

Hầu hết các hệ điều hành đều không cho phép tiến trình con hoạt động khi tiến trình cha đã kết thúc. Trong những trường hợp như thế hệ điều hành sẽ chủ động việc kết thúc tiến trình con khi tiến trình cha vừa kết thúc.

➤ **Khi thay đổi trạng thái tiến trình hệ điều hành thực hiện các bước sau:** Khi một tiến trình đang ở trạng thái running bị chuyển sang trạng thái khác (ready, blocked, ...) thì hệ điều hành phải tạo ra sự thay đổi trong môi trường làm việc của nó. Sau đây là các bước mà hệ điều hành phải thực hiện đầy đủ khi thay đổi trạng thái tiến trình:

- Lưu (save) ngữ cảnh của processor, bao gồm thanh ghi bộ đếm chương trình (PC: program counter) và các thanh ghi khác.
- Cập nhật PCB của tiến trình, sao cho phù hợp với trạng thái mới của tiến trình, bao gồm trạng thái mới của tiến trình, các thông tin tính toán, vv.

- Di chuyển PCB của tiến trình đến một hàng đợi thích hợp, để đáp ứng được các yêu cầu của công tác điều phối tiến trình.
- Chọn một tiến trình khác để cho phép nó thực hiện.
- Cập nhật PCB của tiến trình vừa được chọn thực hiện ở trên, chủ yếu là thay đổi trạng thái của tiến trình đến trạng thái running.
- Cập nhật các thông tin liên quan đến quản lý bộ nhớ. Bước này phụ thuộc vào các yêu cầu chuyển đổi địa chỉ bộ nhớ đang được sử dụng.
- Khôi phục (Restore) lại ngữ cảnh của processor và thay đổi giá trị của bộ đếm chương trình và các thanh ghi khác sao cho phù hợp với tiến trình được chọn ở trên, để tiến trình này có thể bắt đầu hoạt động được.

Như vậy, khi hệ điều hành chuyển một tiến trình từ trạng thái running (đang chạy) sang một trạng thái nào đó (tạm dừng) thì hệ điều hành phải lưu trữ các thông tin cần thiết, nhất là Program Count, để sau này hệ điều hành có thể cho tiến trình tiếp tục hoạt động trở (tái kích hoạt) lại được. Đồng thời hệ điều hành phải chọn một tiến trình nào đó đang ở trạng thái ready để cho tiến trình này chạy (chuyển tiến trình sang trạng thái running). Tại đây, trong các thao tác phải thực hiện, hệ điều hành phải thực hiện việc thay đổi giá trị của PC, thay đổi ngữ cảnh processor, để PC chỉ đến địa chỉ của chỉ thị đầu tiên của tiến trình running mới này trong bộ nhớ. Đây cũng chính là bản chất của việc thực hiện các tiến trình trong các hệ thống uniprocessor.

2.2 Lập lịch tiến trình

Lập lịch tiến trình (hay điều phối tiến trình) là cơ sở của các hệ điều hành đa nhiệm. Bằng cách chuyển đổi CPU giữa các tiến trình, hệ điều hành có thể làm máy tính hoạt động nhiều hơn (giúp đạt được sự sử dụng CPU tối đa). Bộ phận điều phối tiến trình có nhiệm vụ xem xét và quyết định khi nào thì dừng tiến trình hiện tại để thu hồi processor và chuyển processor cho tiến trình khác, và khi đã có được processor thì chọn tiến trình nào trong số các tiến trình ở trạng thái ready để cấp processor cho nó. Trong chương này, chúng ta giới thiệu các khái niệm cơ bản và trình bày các giải thuật lập lịch CPU khác nhau. Chúng ta cũng xem xét vấn đề chọn một giải thuật cho một hệ thống xác định.

2.2.1 Các khái niệm cơ bản

Mục tiêu của đa chương là có nhiều tiến trình chạy cùng thời điểm để tối ưu hóa việc sử dụng CPU. Trong hệ thống đơn vi xử lý, chỉ một tiến trình có thể chạy tại một thời điểm; bất kỳ tiến trình nào khác đều phải chờ cho đến khi CPU rảnh và có thể được lập lịch lại.

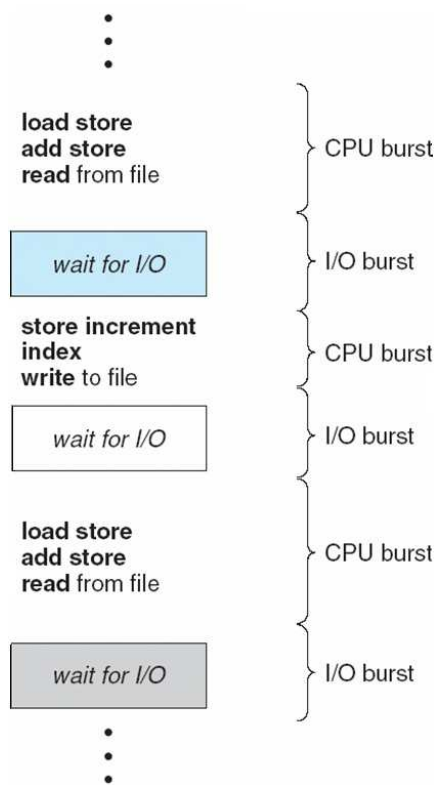
Ý tưởng của đa chương là tương đối đơn giản. Một tiến trình được thực thi có thể phải chờ yêu cầu nhập/xuất hoàn thành để kết thúc công việc của nó. Trong một hệ thống máy tính đơn giản thì CPU sẽ rảnh (lúc tiến trình thao tác nhập xuất); tất cả thời gian chờ này là lãng phí. Với hệ thống đa chương, chúng ta cố gắng dùng thời gian này để CPU có thể phục vụ cho các tiến trình khác. Nhiều tiến trình được lưu giữ trong bộ nhớ tại cùng thời điểm. Khi một tiến trình phải chờ, hệ điều hành

lấy CPU từ tiến trình này và cấp cho quá trình khác.

Lập lịch là chức năng cơ bản của hệ điều hành. Hầu hết tài nguyên máy tính được lập lịch trước khi dùng. Dĩ nhiên, CPU là một trong những tài nguyên máy tính ưu tiên. Do đó, lập lịch là trọng tâm trong việc thiết kế hệ điều hành.

2.2.1.1 Chu kỳ CPU - I/O

Sự thành công của việc lập lịch CPU phụ thuộc vào thuộc tính được xem xét sau đây của tiến trình. Việc thực thi tiến trình gồm một chu kỳ (cycle) thực thi của CPU và chờ nhập/xuất. Các tiến trình chuyển đổi giữa hai trạng thái này. Sự thực thi tiến trình bắt đầu với một chu kỳ CPU (*CPU burst*), theo sau bởi một chu kỳ nhập/xuất (*I/O burst*), sau đó một chu kỳ CPU khác, sau đó lại tới một chu kỳ nhập/xuất,... Sau cùng, chu kỳ CPU cuối cùng sẽ kết thúc với một yêu cầu hệ thống để kết thúc việc thực thi (hình 2.2). Một chương trình hướng nhập/xuất (I/O-bound) thường có nhiều chu kỳ CPU ngắn. Một chương trình hướng xử lý (CPU-bound) có thể có một nhiều chu kỳ CPU dài. Sự phân bổ này có thể giúp chúng ta chọn giải thuật định thời CPU hợp lý.



Hình 2.2. Chu kỳ CPU – I/O

2.2.1.2 Trình lập lịch CPU (CPU Scheduler)

Mỗi khi CPU rỗi, hệ điều hành cần chọn trong số các tiến trình sẵn sàng thực hiện trong bộ nhớ (ready queue), cấp phát CPU cho một trong số đó. Tiến trình được thực hiện bởi trình lập lịch ngắn kỳ (short-term scheduler, CPU scheduler).

Hàng đợi sẵn sàng (ready queue) không nhất thiết là hàng đợi vào trước, ra trước (FIFO). Xem xét một số giải thuật lập lịch khác nhau, một hàng đợi sẵn sàng

có thể được cài đặt như một hàng đợi FIFO, một hàng đợi ưu tiên, một cây, hay đơn giản là một danh sách liên kết không thứ tự. Tuy nhiên, về khái niệm tất cả các tiến trình trong hàng đợi sẵn sàng được xếp hàng chờ cơ hội để thực hiện trên CPU. Các mẫu tin trong hàng đợi thường là khối điều khiển tiến trình của tiến trình đó.

Các quyết định lập lịch CPU có thể xảy ra khi 1 tiến trình:

1. Chuyển từ trạng thái hoạt động sang trạng thái chờ (ví dụ: I/O request)
2. Chuyển từ trạng thái hoạt động sang trạng thái sẵn sàng (vd: khi một ngắt xuất hiện)
3. Chuyển từ trạng thái đợi sang trạng thái sẵn sàng (ví dụ: I/O hoàn thành)
4. Kết thúc

Lập lịch CPU khi 1 và 4 là *không được ưu tiên trước (non-preemptive)*: độc quyền

- ⌚ Không có sự lựa chọn: phải chọn 1 tiến trình mới để thực hiện.
- ⌚ Khi 1 tiến trình được phân phối CPU, nó sẽ sử dụng CPU cho đến khi nó giải phóng CPU bằng cách kết thúc hoặc chuyển sang trạng thái chờ.
- ⌚ Các tiến trình sẵn sàng nhường điều khiển của CPU

Các quyết định lập lịch xảy ra khi: Tiến trình chuyển trạng thái từ Running sang Blocked hoặc khi tiến trình kết thúc.

Lập lịch 2 và 3 là *được ưu tiên trước (preemptive)*: không độc quyền

- ⌚ Khi 2: tiến trình “đá” bật CPU ra. Cần phải chọn tiến trình kế tiếp.
- ⌚ Khi 3: tiến trình có thể “đá” bật tiến trình khác ra khỏi CPU.

Các quyết định lập lịch xảy ra khi: Tiến trình chuyển trạng thái hoặc khi tiến trình kết thúc.

➤ **Các đặc điểm của tiến trình:** Khi tổ chức lập lịch tiến trình, trình lập lịch CPU của hệ điều hành thường dựa vào các đặc điểm của tiến trình. Sau đây là một số đặc điểm của tiến trình:

- Tiến trình thiên hướng Vào/Ra: Là các tiến trình cần nhiều thời gian hơn cho việc thực hiện các thao tác xuất/nhập dữ liệu, so với thời gian mà tiến trình cần để thực hiện các chỉ thị trong nó, được gọi là các tiến trình thiên hướng Vào/Ra.
- Tiến trình thiên hướng xử lý: Ngược lại với trên, đây là các tiến trình cần nhiều thời gian hơn cho việc thực hiện các chỉ thị trong nó, so với thời gian mà tiến trình để thực hiện các thao tác Vào/Ra.
- Tiến trình tương tác hay xử lý theo lô: Tiến trình cần phải trả lại kết quả tức thời (như trong hệ điều hành tương tác) hay kết thúc xử lý mới trả về kết quả (như trong hệ điều hành xử lý theo lô).
- Độ ưu tiên của tiến trình: Mỗi tiến trình được gán một độ ưu tiên nhất định, độ ưu tiên của tiến trình có thể được phát sinh tự động bởi hệ thống hoặc được gán tường minh trong chương trình của người sử dụng. Độ ưu tiên của tiến

trình có hai loại: Thứ nhất, độ ưu tiên tĩnh: là độ ưu tiên gán trước cho tiến trình và không thay đổi trong suốt thời gian sống của tiến trình. Thứ hai, độ ưu tiên động: là độ ưu tiên được gán cho tiến trình trong quá trình hoạt động của nó, hệ điều hành sẽ gán lại độ ưu tiên cho tiến trình khi môi trường xử lý của tiến trình bị thay đổi. Khi môi trường xử lý của tiến trình bị thay đổi hệ điều hành phải thay đổi độ ưu tiên của tiến trình cho phù hợp với tình trạng hiện tại của hệ thống và công tác điều phối tiến trình của hệ điều hành.

- Thời gian sử dụng processor của tiến trình: Tiến trình cần bao nhiêu khoảng thời gian của processor để hoàn thành xử lý.
- Thời gian còn lại tiến trình cần processor: Tiến trình còn cần bao nhiêu khoảng thời gian của processor nữa để hoàn thành xử lý.

Trình lập lịch tiến trình thường dựa vào đặc điểm của tiến trình để thực hiện lập lịch ở mức tác vụ, hay điều phối tác vụ. Điều phối tác vụ được phải thực hiện trước điều phối tiến trình. Ở mức này hệ điều hành thực hiện việc chọn tác vụ để đưa vào hệ thống. Khi có một tiến trình được tạo lập hoặc khi có một tiến trình kết thúc xử lý thì bộ phận điều phối tác vụ được kích hoạt. Điều phối tác vụ quyết định sự đa chương của hệ thống và hiệu quả cũng như mục tiêu của điều phối của bộ phận điều phối tiến trình. Ví dụ, để khi thác tối đa thời gian xử lý của processor thì bộ phận điều phối tác vụ phải đưa vào hệ thống số lượng các tiến trình tính hướng Vào/Ra cân đối với số lượng các tiến trình tính hướng xử lý, các tiến trình này thuộc những tác vụ nào. Nếu trong hệ thống có quá nhiều tiến trình tính hướng Vào/Ra thì sẽ lãng phí thời gian xử lý của processor. Nếu trong hệ thống có quá nhiều tiến trình tính hướng xử lý thì processor không thể đáp ứng và có thể các tiến trình phải đợi lâu trong hệ thống, dẫn đến hiệu quả tương tác sẽ thấp.

2.2.1.3 Trình điều vận (Dispatcher)

Một thành phần khác liên quan đến chức năng lập lịch CPU là trình điều vận (dispatcher). Trình điều vận là một module có nhiệm vụ trao quyền điều khiển CPU tới tiến trình được chọn bởi bộ lập lịch (CPU scheduler). Các bước thực hiện:

- 🕒 Chuyển ngữ cảnh
- 🕒 Chuyển sang user mode
- 🕒 Nhảy tới vị trí thích hợp trong chương trình của người sử dụng để khởi động lại chương trình đó.

Trễ điều vận (Dispatcher latency) – thời gian cần thiết để trình điều vận dừng một tiến trình và khởi động một tiến trình khác hoạt động.

2.2.2 Tiêu chuẩn lập lịch tiến trình

Trình lập lịch tiến trình của hệ điều hành phải đạt được các mục tiêu sau đây trong công tác lập lịch của nó.

- Tính hiệu quả (**CPU utilization**): Tận dụng được tối đa thời gian xử lý của processor, tức là giữ cho CPU càng bận càng tốt (*Max*).
- Thông lượng tối đa (**Throughtput**): Chính sách lập lịch phải cố gắng

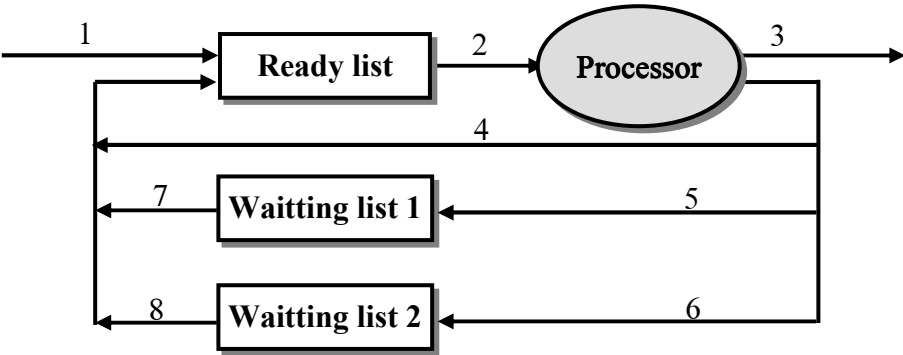
để cực đại được số lượng tiến trình hoàn thành trên một đơn vị thời gian. Mục tiêu này ít phụ thuộc vào chính sách điều phối mà phụ thuộc nhiều vào thời gian thực hiện trung bình của các tiến trình (*Max*).

- Thời gian lưu lại trong hệ thống (Turnaround time): Đây là khoảng thời gian từ khi tiến trình được đưa ra đến khi được hoàn thành. Bao gồm thời gian chờ được đưa vào bộ nhớ + thời gian chờ trong ready queue + thời gian thực hiện bởi CPU + thời gian thực hiện vào-ra (*Min*).
- Waiting time – lượng thời gian mà tiến trình chờ đợi trong ready queue (*Min*).
- Thời gian đáp ứng hợp lý (**Response time**): Đối với các tiến trình tương tác, đây là khoảng thời gian từ khi tiến trình đưa ra yêu cầu cho đến khi nhận được sự hồi đáp. Một tiến trình đáp ứng yêu cầu của người sử dụng, phải nhận được thông tin hồi đáp từ yêu cầu của nó thì nó mới có thể trả lời người sử dụng. Do đó, theo người sử dụng thì bộ phận điều phối phải cực tiểu hoá thời gian hồi đáp của các tiến trình, có như vậy thì tính tương tác của tiến trình mới tăng lên (*Min*).

2.2.3 Các giải thuật lập lịch

Để tổ chức điều phối tiến trình hệ điều hành sử dụng hai danh sách: Danh sách sẵn sàng (Ready list) dùng để chứa các tiến trình ở trạng thái sẵn sàng. Danh sách đợi (Waiting list) dùng để chứa các tiến trình đang đợi để được bổ sung vào danh sách sẵn sàng.

Chỉ có những tiến trình trong ready list mới được chọn để cấp processor. Các tiến trình bị chuyển về trạng thái blocked sẽ được bổ sung vào waiting list. Hệ thống chỉ có duy nhất một ready list, nhưng có thể tồn tại nhiều waiting list. Thông thường hệ điều hành thiết kế nhiều waiting list, mỗi waiting list dùng để chứa các tiến trình đang đợi được cấp phát một tài nguyên hay một sự kiện riêng biệt nào đó. Hình 2.3 minh họa cho việc chuyển tiến trình giữa các danh sách:



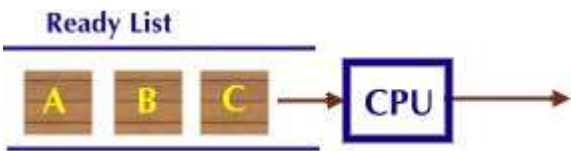
Hình 2.3 Sơ đồ chuyển tiến trình vào các danh sách

- Trong đó:
1. Tiến trình trong hệ thống được cấp đầy đủ tài nguyên chỉ thiếu processor.
 2. Tiến trình được bộ điều phối chọn ra để cấp processor để bắt đầu xử lý.

- 3. Tiến trình kết thúc xử lý và trả lại processor cho hệ điều hành.
- 4. Tiến trình hết thời gian được quyền sử dụng processor (time-out), bị bộ điều phối tiến trình thu hồi lại processor.
- 5. Tiến trình bị khóa (blocked) do yêu cầu tài nguyên nhưng chưa được hệ điều hành cấp phát. Khi đó tiến trình được đưa vào danh sách các tiến trình đợi tài nguyên (waiting list 1).
- 6. Tiến trình bị khóa (blocked) do đang đợi một sự kiện nào đó xảy ra. Khi đó tiến trình được bộ điều phối đưa vào danh sách các tiến trình đợi tài nguyên (waiting list 2).
- 7. Tài nguyên mà tiến trình yêu cầu đã được hệ điều hành cấp phát. Khi đó tiến trình được bộ điều phối chuyển sang danh sách các tiến trình ở trạng thái sẵn sàng (ready list) để chờ được cấp processor để được hoạt động.
- 8. Sự kiện mà tiến trình chờ đã xảy ra. Khi đó tiến trình được bộ điều phối chuyển sang danh sách các tiến trình ở trạng thái sẵn sàng (ready list) để chờ được cấp processor.

2.2.3.1 Giải thuật FCFS (First-come, First-served)

Với cơ chế này, tiến trình yêu cầu CPU trước được cấp phát CPU trước. Việc cài đặt chính sách FCFS được quản lý dễ dàng với hàng đợi FIFO. Khi một tiến trình đi vào hàng đợi sẵn sàng, PCB của nó được liên kết tới đuôi của hàng đợi. Khi CPU rảnh, nó được cấp phát tới một tiến trình tại đầu hàng đợi. Sau đó, tiến trình đang chạy được lấy ra khỏi hàng đợi. Mã của giải thuật FCFS đơn giản để viết và hiểu.



Ví dụ: Nếu hệ điều hành cần cấp CPU cho 3 tiến trình P₁, P₂, P₃, với thời điểm vào ready list (RL) và khoảng thời gian mỗi tiến trình cần CPU (được tính theo đơn vị ms) được mô tả trong bảng sau:

Tiến trình	Thời điểm vào (RL)	Thời gian xử lý (Burst time)
P ₁	0	24
P ₂	1	3
P ₃	2	3

Thứ tự cấp phát CPU cho các tiến trình diễn ra như sau:

Tiến trình:	P ₁	P ₂	P ₃
Thời điểm:	0	24	27

Vậy thời gian chờ của tiến trình P₁ là 0, của P₂ là 23 (24 - 0), của P₃ là 25 (24 + 3 - 2). Và thời gian chờ đợi trung bình của các tiến trình là:

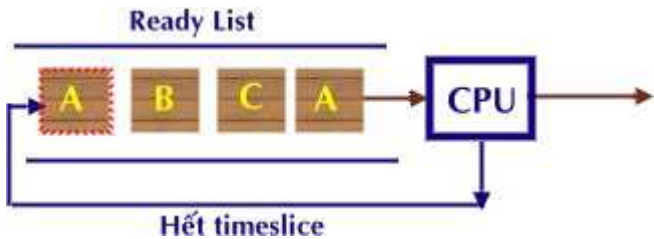
$$(0 + 23 + 25)/3 = 16 \text{ (ms)}.$$

Nhận xét: 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.

2.2.3.2 Giải thuật phân phối xoay vòng RR (Round Robin)

Trong chiến lược này, 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: Nếu hệ điều hành cần cấp processor cho 3 tiến trình P₁, P₂, P₃ với thời điểm vào ready list và khoảng thời gian mỗi tiến trình cần processor được mô tả trong bảng sau:

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 ms, thứ tự cấp phát CPU sẽ là :

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

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

Như vậy RR có thời gian chờ đợi trung bình nhỏ hơn so với FIFO

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

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

2.2.3.3 Giải thuật công việc ngắn nhất SJF (Shortest Job Fist)

Một tiếp cận khác đối với việc lập lịch CPU là giải thuật công việc ngắn nhất (shortest-job-first-SJF). Giải thuật này gắn với mỗi tiến trình là thời gian sử dụng CPU *tiếp sau của nó*. Thời gian này được sử dụng để lập lịch các tiến trình với thời gian ngắn nhất. Nếu hai tiến trình có cùng chiều dài chu kỳ CPU kế tiếp, giải thuật FCFS được dùng. Chú ý rằng thuật ngữ phù hợp hơn là chu kỳ CPU kế tiếp ngắn nhất (shortest next CPU burst) vì lập lịch được thực hiện bằng cách xem xét chiều dài của chu kỳ CPU kế tiếp của tiến trình hơn là toàn bộ chiều dài của nó.

Giải thuật này có thể cài đặt theo cơ chế độc quyền (non-preemptive) hoặc không độc quyền. Sự chọn lựa xảy ra khi có một tiến trình mới được đưa vào danh sách sẵn sàng trong khi một tiến trình khác đang xử lý. Tiến trình mới có thể sở hữu một yêu cầu thời gian sử dụng CPU cho lần tiếp theo (CPU-burst) ngắn hơn thời gian còn lại mà tiến trình hiện hành cần xử lý. Giải thuật SJF không độc quyền sẽ dừng hoạt động của tiến trình hiện hành, trong khi giải thuật độc quyền sẽ cho phép tiến trình hiện hành tiếp tục xử lý.

Ví dụ:

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

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

P1	P4	P3	P2
0	6	8	12

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

👉 *Nhận xét* : Giải thuật này cho phép đạt được thời gian chờ trung bình cực tiểu. Khó khăn thực sự của giải thuật SJF là không thể biết chính xác thời gian sử dụng CPU tiếp sau của tiến trình nhưng có thể đoán giá trị xấp xỉ của nó dựa vào thời gian sử dụng CPU trước đó và sử dụng công thức đệ quy:

- gọi t_n là độ dài của thời gian xử lý lần thứ n ,
- τ_{n+1} là giá trị dự đoán cho lần xử lý CPU tiếp theo.
- $\alpha, 0 \leq \alpha \leq 1$
- τ_0 là một hằng số

Với hy vọng giá trị dự đoán sẽ gần giống với các giá trị trước đó, có thể sử dụng công thức:

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

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

- $\alpha = 0$: $\tau_{n+1} = \tau_n = \tau_0$
 - Thời gian thực tế sử dụng CPU gần đây không có tác dụng gì cả.
- $\alpha = 1$: $\tau_{n+1} = \alpha t_n$
 - Chỉ tính đến thời gian sử dụng CPU ngay trước đó.

2.2.3.4 Giải thuật theo độ ưu tiên

Giải thuật SJF là trường hợp đặc biệt của giải thuật lập lịch theo độ ưu tiên (*priority-scheduling algorithm*). Độ ưu tiên được gán với mỗi tiến trình và CPU được cấp phát cho tiến trình có độ ưu tiên cao nhất, tại thời điểm hiện tại. Tiến trình có độ ưu tiên bằng nhau được lập lịch trong thứ tự FCFS.

Giải thuật SJF là giải thuật ưu tiên đơn giản ở đó độ ưu tiên p là nghịch đảo với chu kỳ CPU được dự đoán tiếp theo. Chu kỳ CPU lớn hơn có độ ưu tiên thấp hơn và ngược lại.

Các độ ưu tiên thường nằm trong dãy số cố định, chẳng hạn 0 tới 7 hay 0 tới 4,095. Tuy nhiên, không có sự thoả thuận chung về 0 là độ ưu tiên thấp nhất hay cao nhất. Một vài hệ thống dùng số thấp để biểu diễn độ ưu tiên thấp; ngược lại các hệ thống khác dùng các số thấp cho độ ưu tiên cao. Trong tài liệu này chúng ta quy định các chỉ số thấp biểu diễn độ ưu tiên cao.

Độ ưu tiên có thể được định nghĩa bên trong hay bên ngoài. Độ ưu tiên được định nghĩa bên trong thường dùng định lượng hoặc nhiều định lượng có thể đo được để tính toán độ ưu tiên của một tiến trình. Ví dụ, các giới hạn thời gian, các yêu cầu bộ nhớ, số lượng tập tin đang mở và tỉ lệ của chu kỳ nhập/xuất trung bình với tỉ lệ của chu kỳ CPU trung bình. Các độ ưu tiên bên ngoài được thiết lập bởi các tiêu chuẩn bên ngoài đối với hệ điều hành như tính quan trọng của tiến trình, chi phí thực hiện tiến trình, ...

Lập lịch theo độ ưu tiên có thể thực hiện theo cơ chế độc quyền hay không độc quyền. Khi một tiến trình đến hàng đợi sẵn sàng, độ ưu tiên của nó được so sánh với độ ưu tiên của tiến trình hiện đang thực hiện. Giải thuật lập lịch theo độ ưu tiên không độc quyền sẽ chiếm dụng CPU nếu độ ưu tiên của tiến trình mới đến cao hơn độ ưu tiên của tiến trình đang thực thi. Giải thuật lập lịch theo độ ưu tiên độc quyền sẽ đơn giản đặt tiến trình mới tại đầu hàng đợi sẵn sàng.

Vấn đề chính với giải thuật lập lịch theo độ ưu tiên là nghẽn không hạn định (*indefinite blocking*) hay “đói” CPU (*starvation*). Một tiến trình ở trạng thái sẵn sàng nhưng thiếu CPU có thể xem như bị nghẽn-chờ đợi CPU. Giải thuật lập lịch theo độ ưu tiên có thể làm dẫn đến nhiều tiến trình có độ ưu tiên thấp chờ CPU không hạn định.

Một giải pháp cho vấn đề nghẽn không hạn định ở trên là sử dụng giải pháp **Aging**. Aging là kỹ thuật tăng dần độ ưu tiên của tiến trình chờ trong hệ thống một thời gian dài. Ví dụ, nếu các độ ưu tiên nằm trong dãy từ 127 (thấp) đến 0 (cao), chúng ta giảm giá trị độ ưu tiên của tiến trình đang chờ xuống 1 sau mỗi 15 phút. Cuối cùng, thậm chí một tiến trình với độ ưu tiên khởi đầu 127 sẽ đạt độ ưu tiên cao nhất trong hệ thống và sẽ được thực thi. Thật vậy, một tiến trình sẽ mất không quá 32 giờ để đạt được độ ưu tiên từ 127 tới 0.

Ở đây hệ điều hành thường tổ chức gán độ ưu tiên cho tiến trình theo nguyên tắc kết hợp giữ gán tĩnh và gán động. Khi khởi tạo tiến trình được gán độ ưu tiên tĩnh, sau đó phụ thuộc vào môi trường hoạt động của tiến trình và công tác điều phối tiến trình của bộ phận điều phối mà hệ điều hành có thể thay đổi độ ưu tiên của tiến trình.

2.2.3.5 Chiến lược nhiều cấp độ ưu tiên

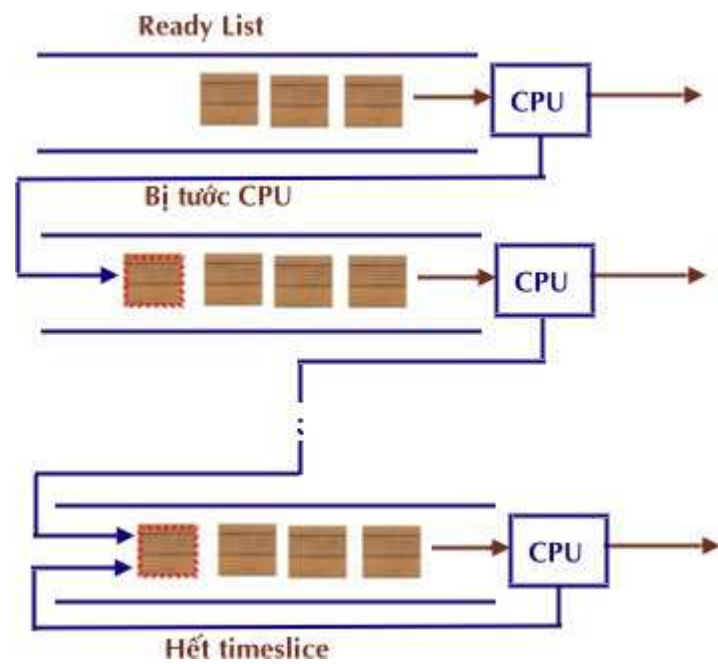
Ý tưởng chính của giải thuật là phân lớp các tiến trình tùy theo độ ưu tiên của chúng để có cách thức lập lịch thích hợp cho từng nhóm. Danh sách sẵn sàng được phân tách thành các danh sách riêng biệt theo cấp độ ưu tiên, mỗi danh sách bao gồm các tiến trình có cùng độ ưu tiên và được áp dụng một giải thuật lập lịch thích hợp để điều phối. Ngoài ra, còn có một giải thuật lập lịch 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.



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 lập lịch, 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 lập lịch nhiều cấp ưu tiên và xoay vòng cần quyết định các tham số:

- Số lượng các cấp ưu tiên
- Giải thuật lập lịch 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.



2.3 Đồng bộ hóa tiến trình

Xét ví dụ với 2 tiến trình P1 và P2 hoạt động trong hệ thống các tiến trình hợp tác tuần tự, thực hiện bất đồng bộ và cùng chia sẻ một đoạn code (và dữ liệu) sau:

P1:

P2:

```
{
    shared int x ;
    x = 10;
    while (1)
    {
        x = x - 1 ;
        x = x + 1 ;
        if (x != 10 )
            printf (" x is % d",x)
    }
}
```

```
{
    shared int x;
    x = 10;
    while (1)
    {
        x = x - 1;
        x = x + 1;
        if ( x != 10)
            printf ("x is % d" ,x)
    }
}
```

- Để in ra x là 10. Ta xen kẽ giữa các xử lý tiến trình. Để thực hiện công việc này thì khi kiểm tra giá trị của x để in ta thực hiện xen kẽ việc tăng giá trị x lên bằng một tiến trình khác. Do đó khi kiểm tra giá trị x lúc thực thi thì x không bằng 10 nhưng ta vẫn in ra x là 10 khi đọc từ bộ nhớ.

```
P1:  x = x - 1           // x = 9
P1:  x = x + 1          // x = 10
```

```
P2:  x = x - 1           // x = 9
P1:  if (x!=10)          // true
P2:  x = x + 1           // x = 10
P1:  printf (" x is %d", x)  // x = 10
```

- Theo trên, ta sẽ có thể in ra dòng thông báo “x is 10”. Bây giờ, đề bài lại yêu cầu như sau: Thực hiện đồng bộ hoá 2 tiến trình P1 và P2 để bảo đảm chắc chắn rằng lệnh printf () sẽ không bao giờ được thực thi. Để giải quyết bài toán này, chúng ta sẽ xem xét các khái niệm tiếp sau đây.

2.3.1 Tài nguyên căng và đoạn căng

2.3.1.1 Tài nguyên căng (Critical Resource)

Trong môi trường hệ điều hành đa nhiệm - đa chương – đa người sử dụng, việc chia sẻ tài nguyên cho các tiến trình của người sử dụng dùng chung là cần thiết, nhưng nếu hệ điều hành không tổ chức tốt việc sử dụng tài nguyên dùng chung của các tiến trình hoạt động đồng thời, thì không những không mang lại hiệu quả khai thác tài nguyên của hệ thống mà còn làm hỏng dữ liệu của các ứng dụng. Và nguy hiểm hơn là việc hỏng dữ liệu này có thể hệ điều hành và ứng dụng không thể phát hiện được. Việc hỏng dữ liệu của ứng dụng có thể làm sai lệch ý nghĩa thiết kế của nó. Đây là điều mà cả hệ điều hành và người lập trình đều không mong muốn.

Các tiến trình hoạt động đồng thời thường cạnh tranh với nhau trong việc sử dụng tài nguyên dùng chung. Hai tiến trình hoạt động đồng thời cùng ghi vào một không gian nhớ chung (một biến chung) trên bộ nhớ hay hai tiến trình đồng thời cùng ghi dữ liệu vào một file chia sẻ, đó là những biểu hiện của sự cạnh tranh về việc sử dụng tài nguyên dùng chung của các tiến trình. Để các tiến trình hoạt động đồng thời không cạnh tranh hay xung đột với nhau khi sử dụng tài nguyên dùng chung hệ điều hành phải tổ chức cho các tiến trình này được độc quyền truy xuất/ sử dụng trên các tài nguyên dùng chung này.

Những tài nguyên được hệ điều hành chia sẻ cho nhiều tiến trình hoạt động đồng thời dùng chung, mà có nguy cơ dẫn đến sự tranh chấp giữa các tiến trình này khi sử dụng chúng, được gọi là tài nguyên căng. Tài nguyên căng có thể là tài nguyên phần cứng hoặc tài nguyên phần mềm, có thể là tài nguyên phân chia được hoặc không phân chia được, nhưng đa số thường là tài nguyên phân chia được như là: các biến chung, các file chia sẻ.

Ví dụ 1: Giả sử có một chương trình, trong đó có hai tiến trình P1 và P2 hoạt động đồng thời với nhau. Tiến trình P1 phải tăng biến Count lên 1 đơn vị, tiến trình P2 phải tăng biến Count lên 1 đơn vị, với mục đích tăng Count lên được 2 đơn vị.

Chương trình có thể thực hiện như sau:

1. Tiến trình P1 ghi nội dung biến toàn cục Count vào biến cục bộ L1
2. Tiến trình P2 ghi nội dung biến toàn cục Count vào biến cục bộ L2
3. Tiến trình P1 thực hiện L1:= L1 + 1 và Count := L1
4. Tiến trình P2 thực hiện L2:= L2 + 1 và Count := L2

Như vậy thoát nhìn ta thấy rằng chắc chắn Count đã tăng được 2 đơn vị, nhưng trong thực tế có thể Count chỉ tăng được 1 đơn vị. Bởi vì, nếu P1 và P2 đồng thời nhận giá trị của Count (giả sử ban đầu Count = 4) vào L1 và L2, sau đó P1 tăng L1 lên 1 và P2 tăng L2 lên 1 (L1 = 5, L2 = 5), rồi sau đó cả P1 và P2 đồng thời ghi giá trị biến L của nó vào lại Count, thì Count chỉ tăng được 1 đơn vị, Count = 5. Đây là điều mà chương trình không mong muốn nhưng cả chương trình và hệ điều hành đều khó có thể phát hiện được.

Nguyên nhân ở trên là do 2 tiến trình P1 và P2 đồng thời truy xuất biến Count, cả khi nhận giá trị của count, lẫn khi ghi giá trị vào Count. Trong trường hợp này nếu hệ điều hành không cho phép hai tiến trình P1 và P2 đồng thời truy xuất Count, hoặc hệ điều hành cho phép mỗi tiến trình được độc quyền truy xuất Count trong đoạn code sau, thì lỗi trên sẽ không xảy ra.

P1: Begin	P2: Begin
L1 := Count;	L2 := Count;
L1 := L1 + 1;	L2 := L2 + 1;
Count := L1;	Count := L2;
End;	End;

Trong trường hợp này tài nguyên găng là biến count.

Ví dụ 2: Giả sử có một ứng dụng Kế toán, hoạt động trong môi trường đa nhiệm, đa người sử dụng. Mỗi người sử dụng trong môi trường này khi cần thực hiện thao tác rút tiền từ trong tài khoản chung thì phải khởi tạo một tiến trình, tạm gọi là tiến trình rút tiền, tiến trình rút tiền chỉ có thể thực hiện được thao tác rút tiền khi số tiền cần rút nhỏ hơn số tiền còn lại trong tài khoản chung. Trong môi trường này có thể có nhiều người sử dụng đồng thời thực hiện thao tác rút tiền từ tài khoản chung của hệ thống.

Như vậy các tiến trình rút tiền, giả sử có hai tiến trình rút tiền P1 và P1, có thể hoạt động đồng thời với nhau và cùng chia sẻ không gian nhớ lưu trữ biến **Tài khoản**, cho biết số tiền còn trong tài khoản dùng chung của hệ thống. Và mỗi tiến trình rút tiền khi muốn rút một khoảng tiền từ tài khoản (**Tiền rút**) thì phải thực hiện kiểm tra **Tài khoản** sau đó mới thực hiện việc rút tiền. Tức là mỗi tiến trình rút tiền, khi cần rút tiền đều phải thực hiện đoạn code sau đây:

IF (Tài khoản - Tiền rút >= 0)	{kiểm tra tài khoản}
Tài khoản := Tài khoản - Tiền rút	{thực hiện rút tiền}
Else	
Thông báo lỗi	{không thể rút tiền}
EndIf;	

Nếu tại một thời điểm nào đó:

- Trong tài khoản còn 800 ngàn đồng (**Tài khoản = 800**).

- Tiến trình rút tiền P1 cần rút 500 ngàn đồng (**Tiền rút = 500**).
- Tiến trình rút tiền P2 cần rút 400 ngàn đồng (**Tiền rút = 400**).
- Tiến trình P1 và P2 đồng thời rút tiền.

Thì theo nguyên tắc điều trên không thể xảy ra, vì tổng số tiền mà hai tiến trình cần rút lớn hơn số tiền còn lại trong tài khoản ($500 + 400 > 800$). Nhưng trong môi trường đa nhiệm, đa người sử dụng nếu hệ điều hành không giám sát tốt việc sử dụng tài nguyên dùng chung của các tiến trình hoạt động đồng thời thì điều trên vẫn có thể xảy ra. tức là, cả hai tiến trình P1 và P2 đều thành công trong thao tác rút tiền, mà ứng dụng cũng như hệ điều hành không hề phát hiện. Bởi vì, quá trình rút tiền của các tiến trình P1 và P2 có thể diễn ra như sau:

1. P1 được cấp processor để thực hiện việc rút tiền: P1 thực hiện kiểm tra tài khoản: **Tài khoản - Tiền rút = $800 - 500 = 300 > 0$** , P1 ghi nhận điều này và chuẩn bị rút tiền.
2. Nhưng khi P1 chưa kịp rút tiền thì bị hệ điều hành thu hồi lại processor, và hệ điều hành cấp processor cho P2. P1 được chuyển sang trạng thái ready.
3. P2 nhận được processor, được chuyển sang trạng thái running, nó bắt đầu thực hiện việc rút tiền như sau: kiểm tra tài khoản: **Tài khoản - Tiền rút = $800 - 400 = 500 \geq 0$** , P2 ghi nhận điều này và thực hiện rút tiền:

$$\text{Tài khoản} = \text{Tài khoản} - \text{Tiền rút} = 800 - 400 = 400.$$

4. P2 hoàn thành nhiệm vụ rút tiền, nó kết thúc xử lý và trả lại processor cho hệ điều hành. Hệ điều hành cấp lại processor cho P1, tái kích hoạt lại P1 để nó tiếp tục thao tác rút tiền.
5. Khi được hoạt động trở lại P1 thực hiện ngay việc rút tiền mà không thực hiện việc kiểm tra tài khoản (vì đã kiểm tra trước đó):

$$\text{Tài khoản} = \text{Tài khoản} - \text{Tiền rút} = 400 - 500 = -100.$$

6. P1 hoàn thành nhiệm vụ rút tiền và kết thúc tiến trình.

Như vậy cả 2 tiến trình P1 và P2 đều hoàn thành việc rút tiền, không thông báo lỗi, mà không gặp bất kỳ một lỗi hay một trở ngại nào. Nhưng đây là một lỗi nghiêm trọng đối với ứng dụng, vì không thể rút một khoảng tiền lớn hơn số tiền còn lại trong tài khoản, hay **Tài khoản** không thể nhận giá trị âm.

Nguyên nhân của lỗi này không phải là do hai tiến trình P1 và P2 đồng thời truy xuất biến Tài khoản, mà do hai thao tác: kiểm tra tài khoản và thực hiện rút tiền, của các tiến trình này bị tách rời nhau. Nếu hệ điều hành làm cho hai thao tác này không tách rời nhau thì lỗi này sẽ không xảy ra.

Trong trường hợp này tài nguyên găng là biến Tài khoản.

Qua các ví dụ trên ta thấy rằng trong các hệ thống đa chương, đa người sử dụng thường xảy ra hiện tượng, nhiều tiến trình đồng thời cùng đọc/ghi dữ liệu vào một vùng nhớ, nơi chứa các biến của chương trình, và nếu không có sự can thiệp của hệ

điều hành thì có thể gây hậu quả nghiêm trọng cho ứng dụng và cho cả hệ thống. Để ngăn chặn các tình huống trên hệ điều hành phải thiết lập cơ chế độc quyền truy xuất trên tài nguyên dùng chung. Tức là, tại mỗi thời điểm chỉ có một tiến trình duy nhất được phép truy xuất trên các tài nguyên dùng chung. Nếu có nhiều tiến trình hoạt động đồng thời cùng yêu cầu truy xuất tài nguyên dùng chung thì chỉ có một tiến trình được chấp nhận truy xuất, các tiến trình khác phải xếp hàng chờ để được truy xuất sau.

Chúng ta cũng thấy rằng nguyên nhân tiềm ẩn của sự xung đột giữa các tiến trình hoạt động đồng thời khi sử dụng tài nguyên găng là: các tiến trình này hoạt động đồng thời với nhau một cách hoàn toàn độc lập và không trao đổi thông tin với nhau nhưng sự thực thi của các tiến trình này lại ảnh hưởng đến nhau. Trường hợp lỗi trong ví dụ 3 ở trên minh chứng cho điều này.

2.3.1.2 Đoạn găng (Critical Section)

Đoạn code trong các tiến trình đồng thời, có tác động đến các tài nguyên có thể trở thành tài nguyên găng được gọi là đoạn găng hay miền găng. Tức là, các đoạn code trong các chương trình dùng để truy cập đến các vùng nhớ chia sẻ, các tập tin chia sẻ được gọi là các đoạn găng.

Trong ví dụ 2 ở trên, đoạn code sau đây là đoạn găng:

```
{ IF (Tài khoản - Tiền rút >= 0)
    Tài khoản := Tài khoản - Tiền rút }
```

Trong ví dụ 1 ở trên có hai đoạn găng là:

```
{ L1 := Count và Count := L1 }.
```

Để hạn chế các lỗi có thể xảy ra do sử dụng tài nguyên găng, hệ điều hành phải điều khiển các tiến trình sao cho, tại một thời điểm chỉ có một tiến trình nằm trong đoạn găng, nếu có nhiều tiến trình cùng muốn vào (thực hiện) đoạn găng thì chỉ có một tiến trình được vào, các tiến trình khác phải chờ, một tiến trình khi ra khỏi (kết thúc) đoạn găng phải báo cho hệ điều hành và/hoặc các tiến trình khác biết để các tiến trình này vào đoạn găng, vv. Các công tác điều khiển tiến trình thực hiện đoạn găng của hệ điều hành được gọi là *điều độ tiến trình qua đoạn găng*. Để công tác điều độ tiến trình qua đoạn găng được thành công, thì cần phải có sự phối hợp giữa vi xử lý, hệ điều hành và người lập trình. Vi xử lý đưa ra các chỉ thị, hệ điều hành cung cấp các công cụ để người lập trình xây dựng các sơ đồ điều độ hợp lý, để đảm bảo sự độc quyền trong việc sử dụng tài nguyên găng của các tiến trình.

2.3.1.3 Yêu cầu của công tác điều độ qua đoạn găng

Một giải pháp giải quyết tốt bài toán điều độ qua đoạn găng (hay bài toán đồng bộ hoá) cần thoả mãn 4 điều kiện sau:

- (1). Mutual Exclusion: Không có hai tiến trình nằm trong đoạn găng cùng lúc.
- (2). Progress: Một tiến trình tạm dừng bên ngoài đoạn găng không được ngăn cản các tiến trình khác vào đoạn găng.

- (3). Bounded Waiting: Không có tiến trình nào phải chờ vô hạn để được vào đoạn găng.
- (4). Không có giả thiết nào đặt ra cho sự liên hệ về tốc độ của các tiến trình, cũng như về số lượng bộ vi xử lý trong hệ thống.

Trong phần tiếp theo sẽ xem xét các giải pháp thoả 4 yêu cầu này.

2.3.2 Điều độ tiến trình qua đoạn găng

Có nhiều giải pháp để thực hiện việc điều độ tiến trình qua đoạn găng. Các giải pháp này, tùy thuộc vào cách tiếp cận trong xử lý của tiến trình bị khoá, được phân biệt thành hai lớp: busy waiting và sleep and wakeup.

2.3.2.1 Các giải pháp “Busy waiting”

Để đơn giản chúng ta xét với các giải pháp áp dụng chỉ với 2 tiến trình cùng yêu cầu vào đoạn găng đồng thời, ký hiệu tương ứng là P_0 và P_1 . Để thuận tiện, ta quy ước, khi xét tiến trình hiện tại tại P_i , thì tiến trình còn lại được hiểu là P_j , và $j = 1 - i$.

a) Giải thuật 1 (biến khoá chung): Ý tưởng của giải thuật này là 2 tiến trình cùng chia sẻ một biến số nguyên chung, (ký hiệu là **turn**), được khởi tạo bằng 0 (hay 1). Nếu $turn == 0$, thì tiến trình P_1 được phép đi vào đoạn găng. Sơ đồ điều độ của giải thuật 1 như sau:

```
do {  
    while (turn != i);  
    Critical_section();  
    turn = j;  
    Non_Critical_section();  
}  
While (1);
```

Sơ đồ điều độ dùng biến khoá chung này đơn giản, dễ xây dựng nhưng vẫn xuất hiện hiện tượng chờ đợi tích cực, khi chờ cho đến khi $turn = i$ (While (turn != i);). Hiện tượng chờ đợi tích cực gây lãng phí thời gian của processor.

Nếu một tiến trình trong đoạn găng không thể ra khỏi đoạn găng, thì các tiến trình chờ ngoài đoạn găng có thể chờ đợi vô hạn (vì turn không được đặt lại = j).

b) Giải thuật 2 (dùng biến khoá riêng): Vấn đề này sinh của giải thuật 1 đó là nó không giữ lại đủ thông tin về trạng thái của mỗi tiến trình; để giải quyết vấn đề này, giải thuật 2 thay thế biến turn bằng mảng sau:

```
Boolean    flag[2];
```

Các phần tử của mảng được khởi tạo bằng false. Nếu $false[i]=true$, có nghĩa là tiến trình P_i sẵn sàng vào đoạn găng. Sơ đồ điều độ của giải thuật này như sau:

```

do {
    flag[i] = true;
    while (flag[j]);
        Critical_section();
    flag[i] = false;
        Non_Critical_section();
}
While (1);

```

Trong giải thuật này, tiến trình P_i đầu tiên thiết lập $\text{flag}[i]=\text{true}$ để xác nhận nó sẵn sàng vào miền găng; sau đó nó sẽ kiểm tra tiến trình P_j xem P_j có trong đoạn găng hay không; nếu P_j đã ở trong đoạn găng thì nó sẽ chờ cho đến khi P_j ra khỏi đoạn găng ($\text{flag}[j]=\text{false}$). Khi đó, P_i sẽ đi vào đoạn găng. Khi ra khỏi đoạn găng, P_i phải đặt lại $\text{flag}[i]=\text{false}$ để cho phép tiến trình khác đang chờ ở ngoài có thể đi vào đoạn găng.

Giải thuật 2 cơ bản khắc phục được nhược điểm của giải thuật 1 nhưng nó vẫn tồn tại một vấn đề như sau:

- Yêu cầu vào đoạn găng của các tiến trình có thể không bao giờ được thỏa mãn. Để minh họa, chúng ta xét trường hợp sau:
 - Tại thời điểm t_0 : P_0 giữ VXL và thực hiện thiết lập $\text{flag}[0]=\text{true}$.
 - Tại thời điểm t_1 : P_0 bị thu hồi VXL và giao cho P_1 . Lúc này P_1 cũng thực hiện thiết lập $\text{flag}[1]=\text{true}$
 - Rõ ràng, sau này P_0 và P_1 sẽ lặp mãi mãi trong câu lệnh while tương ứng của chúng.

c) Giải thuật 3: Giải thuật này được thực hiện bằng cách kết hợp 2 giải thuật 1 và 2 ở trên. Khi đó, các tiến trình chia sẻ 2 biến:

```

Boolean    flag[2];
Int        turn;

```

Khởi tạo $\text{flag}[0] = \text{flag}[1] = \text{false}$ và giá trị turn là không xác định (là 0 hoặc 1). Sơ đồ điều độ tiến trình P_i ($i=0$ hay 1) được biểu diễn như sau:

```

do {
    flag[i] = true;
    while (flag[j])
        if (turn == j)
        {
            flag[i]=false;
            while turn==j do;
            flag[i]=true;
        }
    Critical_section();
    turn = j;
    flag[i] = false;
    Non_Critical_section();
}
While (1);

```

Để đi vào đoạn găng, tiến trình P_i trước tiên đặt $\text{flag}[i]=\text{true}$ và sau đó đặt $\text{turn}=j$. Lưu ý rằng giá trị cuối cùng của turn quyết định tiến trình nào trong 2 tiến trình được phép đi vào đoạn găng trước.

Để chứng tỏ giải pháp này là đúng, ta xét các trường hợp thoả mãn các yêu cầu điều độ ở trên:

- Yêu cầu độc quyền truy xuất: khi cả 2 tiến trình đồng thời quan tâm đến việc vào đoạn găng ($\text{flag}[i]=\text{true}$ và $\text{flag}[j]=\text{true}$) thì chỉ có 1 tiến trình được vào đoạn găng tùy theo giá trị của turn .
 - Giả sử tại thời điểm t_0 tiến trình P_0 chuẩn bị vào đoạn găng. Khi đó nó đặt $\text{flag}[0]=\text{true}$ và turn lúc này bằng 0; vòng lặp $\text{while } (\text{flag}[1]) = \text{false}$ nên P_0 vào đoạn găng.
 - Tiếp theo tại thời điểm t_1 , tiến trình P_1 cũng yêu cầu vào đoạn găng. Khi đó, nó đặt $\text{flag}[1]=\text{true}$; trong vòng lặp $\text{while } (\text{flag}[0]) = \text{true}$, và điều kiện $(\text{turn}==0)=\text{true}$ nên P_1 sẽ thực hiện lệnh $\text{flag}[1]=\text{false}$ và chờ ngoài đoạn găng (trong vòng lặp $\text{while } (\text{turn}==0)$).
- Yêu cầu Progress và Bound Waiting:
 - Giả sử tại thời điểm t_3 , P_0 ra khỏi đoạn găng, nó sẽ thực hiện đặt $\text{turn}=1$ và $\text{flag}[0]=\text{false}$. Lúc này P_1 có thể thoát ra khỏi vòng lặp $\text{while } (\text{turn}==0)$; thiết lập $\text{flag}[1]=\text{true}$ để xác nhận và đi vào đoạn găng. Lúc này, nếu P_0 muốn vào đoạn găng thì sẽ phải đợi ở vòng lặp $\text{while } (\text{turn}==1)$;

d) Giải pháp phân cứng đồng bộ hoá

Như các khía cạnh khác của phần mềm, các đặc điểm phần cứng có thể làm các tác vụ lập trình dễ hơn và cải tiến tính hiệu quả của hệ thống. Trong phần này, chúng ta trình bày một số chỉ thị phần cứng đơn giản sẵn dùng trên nhiều hệ thống và trình bày cách chúng được dùng hiệu quả trong việc giải quyết vấn đề điều độ qua đoạn găng.

Vấn đề đoạn găng có thể được giải quyết đơn giản trong môi trường chỉ có một bộ xử lý nếu chúng ta cấm các ngắt xảy ra khi một biến chia sẻ đang được thay đổi. Tuy nhiên, giải pháp này là không khả thi trong một môi trường có nhiều bộ xử lý. Vô hiệu hoá các ngắt trên đa bộ xử lý có thể mất nhiều thời gian khi một thông điệp muốn truyền qua tất cả bộ xử lý. Việc truyền thông điệp này bị trì hoãn khi đi vào đoạn găng và tính hiệu quả của hệ thống bị giảm.

Do đó nhiều máy cung cấp các chỉ thị phần cứng cho phép chúng ta kiểm tra hay thay đổi nội dung của một từ (word) hay để thay đổi nội dung của hai từ tuân theo tính nguyên tử (atomically)-như là một đơn vị không thể ngắt. Chúng ta có thể sử dụng các chỉ thị đặc biệt này để giải quyết vấn đề đoạn găng trong một cách tương đối đơn giản.

Chỉ thị TestAndSet có thể được định nghĩa như sau:

```
boolean TestAndSet (boolean *target)
{
    boolean rv = *target;
    *target = TRUE;
    return rv;
}
```

Đặc điểm quan trọng của chỉ thị này là việc thực thi có tính nguyên tử. Do đó, nếu hai chỉ thị TestAndSet được thực thi cùng một lúc (mỗi chỉ thị trên một CPU khác nhau), thì chúng sẽ được thực thi tuần tự trong thứ tự bất kỳ.

Nếu một máy hỗ trợ chỉ thị TestAndSet thì chúng ta có thể yêu cầu độc quyền truy xuất bằng cách khai báo một biến khoá (lock) kiểu boolean và được khởi tạo bằng false. Cấu trúc của quá trình P_i được biểu diễn như sau:

```
do {
    while ( TestAndSet (&lock ))
        ; // do nothing
    // critical_section
    lock = FALSE;
    // non_critical_section
```

```
} while (TRUE);
```

Chỉ thị Swap thao tác trên nội dung của hai từ; tương tự như chỉ thị TestAndSet, nó được thực thi theo tính nguyên tử; và được định nghĩa như sau:

```
void Swap (boolean *a, boolean *b)
{
    boolean temp = *a;
    *a = *b;
    *b = temp;
}
```

Nếu một máy hỗ trợ chỉ thị Swap, thì việc yêu cầu độc quyền có thể được cung cấp như sau:

- Một biến toàn cục *lock* kiểu boolean được khai báo và được khởi tạo bằng false.
- Ngoài ra, mỗi tiến trình cũng có một biến boolean cục bộ *key*. Cấu trúc của tiến trình *Pi* được biểu diễn như sau:

```
do {
    key = TRUE;
    while ( key == TRUE)
        Swap (&lock, &key );
        // critical_section
    lock = FALSE;
        // non_critical_section
} while (TRUE);
```

Tuy nhiên, các giải thuật ở trên không thoả mãn yêu cầu chờ đợi có giới hạn (Bounded Waiting). Giải thuật sau sẽ thoả mãn tất cả các yêu cầu của bài toán đồng bộ hoá:

```
do {
    waiting[i] = TRUE;
    key = TRUE;
```



```

while (waiting[i] && key)
    key = TestAndSet(&lock);
waiting[i] = FALSE;
    // critical section
j = (i + 1) % n;
while ((j != i) && !waiting[j])
    j = (j + 1) % n;
if (j == i)
    lock = FALSE;
else
    waiting[j] = FALSE;
    // remainder section
} while (TRUE);

```

Những giải pháp xét ở trên đều phải thực hiện một vòng lặp để kiểm tra liệu tiến trình có được phép vào đoạn găng hay không. Nếu điều kiện chưa thoả, 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 đoạn găng vì vậy được gọi là các giải pháp chờ đợi bận “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 chờ đợi bận.

2.3.2.2 Các giải pháp “Sleep and Wakeup”

Các giải pháp trên tồn tại hiện tượng chờ đợi tích cực, gây lãng phí thời gian xử lý của VXL. Điều này có thể khắc phục bằng một nguyên tắc rất cơ bản: nếu một tiến trình khi chưa đủ điều kiện vào đoạn găng thì được chuyển ngay sang trạng thái blocked để nó trả lại processor cho hệ thống, cấp cho tiến trình khác. Để thực hiện được điều này cần phải có sự hỗ trợ của hệ điều hành và các ngôn ngữ lập trình để các tiến trình có thể chuyển trạng thái của nó. Hai thủ tục **Sleep** và **Wakeup** được hệ điều hành cung cấp để sử dụng cho mục đích này:

- Khi tiến trình chưa đủ điều kiện vào đoạn găng nó sẽ thực hiện một lời gọi hệ thống để gọi Sleep để chuyển nó sang trạng thái blocked, và tiến trình được gọi này đưa vào hàng đợi để đợi cho đến khi có một tiến trình khác gọi thủ tục Wakeup để giải phóng nó ra khỏi hàng đợi và có thể đưa nó vào đoạn găng.

- Một tiến trình khi ra khỏi đoạn găng phải gọi Wakeup để đánh thức một tiến trình trong hàng đợi blocked ra để tạo điều kiện cho tiến trình này vào đoạn găng.

Cấu trúc chương trình trong giải pháp Sleep and Wakeup có thể được biểu diễn

như sau:

```
int busy; // 1 nếu miền tương trực đang bị chiếm
int blocked; // đếm số lượng quá trình đang bị khoá
do{
    if (busy) {
        blocked = blocked + 1;
        sleep();
    }
    else busy = 1;
}while (1);
    Critical_section();
    busy = 0;
    if (blocked){
        wakeup(process);
        blocked = blocked - 1;
    }
    Non_Critical_section();
```

Như vậy giải pháp này được áp dụng trên nhóm các tiến trình hoạt động đồng thời có trao đổi thông tin với nhau, và các tiến trình phải hợp tác với nhau để hoàn thành nhiệm vụ. Các tiến trình này liên lạc với nhau bằng cách gọi tín hiệu cho nhau. Một tiến trình trong hệ thống này có thể bị buộc phải dừng (bị blocked) cho đến khi nhận được một tín hiệu nào đó từ tiến trình bên kia, đó là tiến trình hợp tác với nó.

Việc sử dụng SLEEP và WAKEUP ở trên vẫn có thể xảy ra một vài tình huống như sau: giả sử tiến trình A vào đoạn găng, và trước khi nó rời đoạn găng thì tiến trình B được kích hoạt. Tiến trình B thử vào đoạn găng nhưng nó nhận thấy A đang ở trong đó, vì vậy B tăng giá trị biến blocked lên 1 và chuẩn bị gọi SLEEP để tự khoá. Tuy nhiên, trước khi B có thể thực hiện SLEEP, tiến trình A được kích hoạt trở lại và ra khỏi đoạn găng. Khi ra khỏi đoạn găng, tiến trình A nhận thấy có một tiến trình đang chờ (blocked=1) nên gọi WAKEUP và giảm giá trị blocked xuống 1. Khi đó tín hiệu WAKEUP sẽ lạc mất do tiến trình B chưa thật sự “ngủ” để nhận tín hiệu đánh thức! Khi tiến trình B được tiếp tục xử lý, nó mới gọi SLEEP và tự khoá vĩnh viễn!

Vấn đề ghi nhận được là tình trạng lỗi này xảy ra do việc kiểm tra trạng thái đoạn găng và việc gọi SLEEP hay WAKEUP là những hành động tách biệt, có thể bị ngắt nửa chừng trong quá trình xử lý, do đó có khi tín hiệu WAKEUP gửi đến một tiến trình chưa bị ngهن sẽ lạc mất. Để tránh những tình huống tương tự, hệ điều hành cung cấp những cơ chế đồng bộ hoá dựa trên ý tưởng của chiến lược

“SLEEP and WAKEUP” nhưng được xây dựng bao gồm cả phương tiện kiểm tra điều kiện vào đoạn găng giúp sử dụng an toàn.

Sau đây là các giải pháp sử dụng ý tưởng của Sleep và Wakeup.

a) Giải pháp dùng Semaphore (đèn báo)

Giải pháp này được Dijkstra đề xuất vào năm 1965. Một semaphore S là một biến số nguyên (integer) được truy xuất chỉ thông qua hai thao tác nguyên tử (atomic): wait() và signal(). Định nghĩa cơ bản của wait() và signal() là:

```
wait (S) {
    while S <= 0
        ; // no-op
    S--;
}
//
signal (S) {
    S++;
}
```

Những thay đổi đối với giá trị integer của semaphore trong các thao tác wait() và signal() phải được thực thi không bị phân chia. Nghĩa là khi một tiến trình thay đổi giá trị semaphore, không có tiến trình nào cùng một lúc có thể thay đổi cùng biến semaphore đó. Ngoài ra, trong trường hợp của biến wait(S), kiểm tra giá trị integer của S ($S \leq 0$) và có thể thay đổi của nó (S--) cũng phải được thực thi mà không bị ngắt.

Có thể sử dụng Semaphore ở 2 loại: Counting Semaphore, trong đó vùng giá trị của semaphore là không bị hạn chế; và Binary Semaphore, trong đó vùng giá trị của semaphore chỉ có thể là 0 hoặc 1. Để đơn giản, chúng ta chỉ xét với Binary Semaphore để giải quyết vấn đề đoạn găng với N tiến trình. N tiến trình chia sẻ cùng một biến semaphore, mutex (mutual exclusion) được khởi tạo bằng 1. Mỗi tiến trình P_i được tổ chức điều độ như sau:

```
Semaphore mutex; // initialized to 1
do {
    wait (mutex);
    // Critical_Section
    signal (mutex);
    // Non_Critical_Section
```

```
} while (TRUE);
```

Tương tự như các giải pháp điều độ qua đoạn găng ở phần trên, giải pháp semaphore ở đây đều đòi hỏi sự chờ đợi bận (Busy waiting). Để giải quyết vấn đề này, chúng ta có thể điều chỉnh định nghĩa của các thao tác wait() và signal() của semaphore. Khi một tiến trình thực thi thao tác wait() và nhận thấy rằng nếu giá trị của semaphore không dương, nó phải chờ. Tuy nhiên, thay vì chờ đợi bận, tiến trình có thể khoá chính nó (bằng thao tác block). Thao tác khoá này đặt tiến trình vào một hàng đợi gắn liền với semaphore và trạng thái tiến trình được chuyển sang trạng thái chờ.

Một tiến trình bị khoá chờ trên biến semaphore phải được khởi động lại khi tiến trình khác thực thi thao tác signal(). Tiến trình được khởi động lại bởi thao tác wakeup và chuyển tiến trình từ trạng thái chờ sang trạng thái sẵn sàng. Sau đó, tiến trình này được đặt vào hàng đợi sẵn sàng.

Hai thao tác wait() và signal() được định nghĩa lại như sau:

- Thao tác wait:

```
wait(semaphore *S) {
    S->value--;
    if (S->value < 0) {
        add this process to S->list;
        block();
    }
}
```

- Thao tác signal:

```
signal(semaphore *S) {
    S->value++;
    if (S->value <= 0) {
        remove a process P from S->list;
        wakeup(P);
    }
}
```

Các ví dụ sau đây thay cho sự giải thích về sơ đồ điều độ ở trên:

Ví dụ 1: Sự thực hiện của hai tiến trình P1 và P2 trong sơ đồ điều độ trên

P thực hiện	Wait/Signal	S	Trạng thái của P1/P2
		1	
1. P1	Wait(S)	0	P1 hoạt động
2. P2	Wait(S)	-1	P2 chờ
3. P1	Signal(S)	0	P2 hoạt động
4. P1	Wait(S)	-1	P1 chờ
5. P2	Signal(S)	0	P1 hoạt động

Ví dụ 2: Nếu trong hệ thống có 6 tiến trình hoạt động đồng thời, cùng sử dụng tài nguyên găng, tài nguyên găng này chỉ cho phép một tiến trình truy xuất đến nó tại một thời điểm. Tức là hệ điều hành phải tổ chức truy xuất độc quyền trên tài nguyên găng này. Thứ tự yêu cầu sử dụng tài nguyên găng của các tiến trình, cùng với thời gian mà tiến trình cần processor khi nó ở trong đoạn găng (cần tài nguyên găng) và độ ưu tiên của các tiến trình, được mô tả như sau:

- Có 6 tiến trình yêu cầu sử dụng tài nguyên găng tương ứng với S lần lượt là:

A B C D E F
- Độ ưu tiên của các tiến trình là (5 là độ ưu tiên cao nhất):

1 1 2 4 2 5
- Thời gian các tiến trình cần sử dụng tài nguyên găng là:

4 2 2 2 1 1

Nếu dùng sơ đồ điều độ semaphore ở trên để tổ chức điều độ cho 6 tiến trình này thì ta có được bảng mô tả sự thực hiện của các tiến trình A, B, C, D, E, F như sau:

T	Wait/ Signal	Tiến trình thực hiện	S	Tiến trình hoạt động	Các tiến trình trong hàng đợi
0	-	-	1	-	-
1	Wait	A	0	A	-
2	Wait	B	-1	A	B
3	Wait	C	-2	A	C B
4	Wait	D	-3	A	D C B
5	Signal	A	-2	D	C B

6	Wait	E	-3	D	C E B
7	Signal	D	-2	C	E B
8	Wait	F	-3	C	F E B
9	Signal	C	-2	F	E B
10	Signal	F	-1	E	B
11	Signal	E	0	B	-
12	Signal	B	1	-	-

Bảng trên lưu ý với chúng ta hai điều. Thứ nhất, trị tuyệt đối của S cho biết số lượng các tiến trình trong hàng đợi F(S). Thứ hai, tiến trình chưa được vào đoạn găng thì được đưa vào hàng đợi và tiến trình ra khỏi đoạn găng sẽ đánh thức tiến trình có độ ưu tiên cao nhất trong hàng đợi để đưa nó vào đoạn găng. Tiến trình được đưa vào hàng đợi sau nhưng có độ ưu tiên cao hơn sẽ được đưa vào đoạn găng trước các tiến trình được đưa vào hàng đợi trước nó.

2.3.2.3 Giải pháp dùng Monitors

Để có thể dễ viết đúng các chương trình đồng bộ hoá hơn, Hoare (1974) và Brinch & Hansen (1975) đề nghị một cơ chế đồng bộ hoá cấp cao hơn được cung cấp bởi ngôn ngữ lập trình là monitor. Một monitor được mô tả bởi một tập hợp của các toán tử được định nghĩa bởi người lập trình. Biểu diễn kiểu của một monitor bao gồm việc khai báo các biến mà giá trị của nó xác định trạng thái của một thể hiện kiểu, cũng như thân của thủ tục hay hàm mà cài đặt các thao tác trên kiểu đó. Cú pháp của monitor được hiển thị trong hình dưới đây:

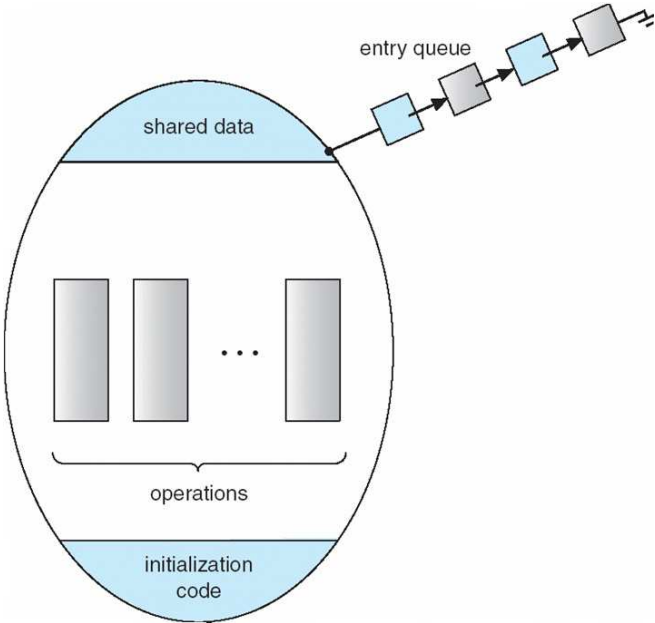
```
monitor monitor-name
{
    // shared variable declarations
    procedure P1 (...) { .... }
    ...

    procedure Pn (...) {.....}

    Initialization code ( ....) { ... }
    ...
}
```

Biểu diễn kiểu monitor không thể được dùng trực tiếp bởi các tiến trình khác nhau. Do đó, một thủ tục được định nghĩa bên trong một monitor chỉ có thể truy xuất những biến được khai báo cục bộ bên trong monitor đó và các tham số chính thức của nó. Tương tự, những biến cục bộ của monitor có thể được truy xuất chỉ bởi những thủ tục cục bộ.

Xây dựng monitor phải đảm bảo rằng chỉ một tiến trình tại một thời điểm có thể được kích hoạt trong monitor. Do đó, người lập trình không cần viết mã ràng buộc đồng bộ hoá như hình 2.4 sau:



Hình 2.4 Giải pháp dùng Monitor

Tuy nhiên, xây dựng monitor như được định nghĩa là không đủ mạnh để mô hình hoá các cơ chế đồng bộ. Vì vậy, chúng ta cần định nghĩa các cơ chế đồng bộ hoá bổ sung. Những cơ chế này được cung cấp bởi construct *condition*. Người lập trình có thể định nghĩa một hay nhiều biến kiểu *condition*:

condition x, y;

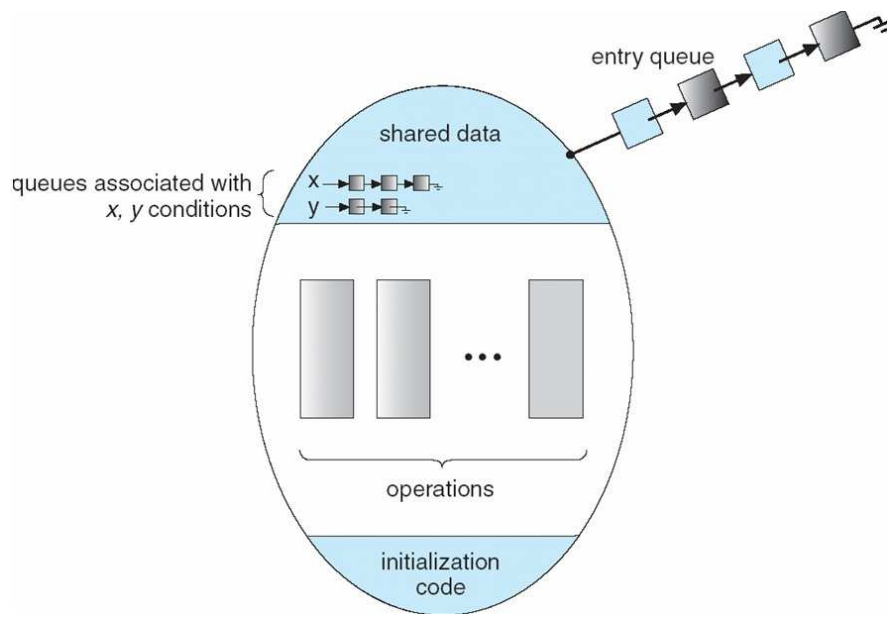
Thao tác kèm theo là Wait và Signal, chỉ có wait và signal được tác động đến các biến điều kiện.

- x.wait (): khi một tiến trình gọi wait, thì wait sẽ chuyển tiến trình gọi sang trạng thái blocked, và đặt tiến trình này vào hàng đợi trên biến điều kiện x.
- x.signal (): khi một tiến trình gọi signal, thì signal sẽ kiểm tra trong hàng đợi của x có tiến trình nào hay không, nếu có thì tái kích hoạt tiến trình đó, và tiến trình gọi signal sẽ rời khỏi monitor.

Xét với trường hợp như hình 2.5 sau:

Giả sử rằng, khi thao tác x.signal() được gọi bởi một tiến trình P thì có một tiến trình Q gắn với biến điều kiện x bị tạm dừng. Rõ ràng, nếu tiến trình Q được phép

thực thi tiếp thì tiến trình P phải dừng. Hai khả năng có thể xảy ra:



Hình 2.5 Monitor với các biến điều kiện

- P chờ cho đến khi Q rời khỏi monitor hoặc chờ điều kiện khác.
- Q chờ cho đến khi P rời monitor hoặc chờ điều kiện khác.

Bây giờ chúng ta xem xét cài đặt cơ chế monitor dùng semaphores. Đối với mỗi monitor, một biến semaphore mutex (được khởi tạo 1) được cung cấp. Một tiến trình phải thực thi wait(mutex) trước khi đi vào monitor và phải thực thi signal(mutex) sau khi rời monitor.

Vì tiến trình đang báo hiệu phải chờ cho đến khi tiến trình được bắt đầu lại rời hay chờ, một biến semaphore bổ sung next được giới thiệu, khởi tạo 0 trên tiến trình báo hiệu có thể tự tạm dừng. Một biến số nguyên next_count cũng sẽ được cung cấp để đếm số lượng tiến trình bị tạm dừng trên next.

```
semaphore mutex; // (initially = 1)
semaphore next;  // (initially = 0)
int next-count = 0;
```

Do đó, mỗi thủ tục *F* sẽ được thay thế bởi:

```
wait(mutex);
...
body of F;
...
if (next_count > 0)
```



```
        signal(next)
    else
        signal(mutex);
```

Mutual exclusion trong monitor được đảm bảo.

Đối với mỗi biến điều kiện x, chúng ta có:

```
semaphore x_sem; // (initially = 0)
int x-count = 0;
```

Thao tác x.wait có thể được cài đặt như sau:

```
x-count++;
if (next_count > 0)
    signal(next);
else
    signal(mutex);
wait(x_sem);
x-count--;
```

Thao tác x.signal có thể được cài đặt như sau:

```
if (x-count > 0) {
    next_count++;
    signal(x_sem);
    wait(next);
    next_count--;
}
```

Chúng ta sẽ xem xét vấn đề thứ tự bắt đầu lại của tiến trình trong monitor. Nếu nhiều tiến trình bị trì hoãn trên biến điều kiện x và thao tác x.signal được thực thi bởi một vài tiến trình thì thứ tự các tiến trình bị trì hoãn được thực thi trở lại như thế nào? Một giải pháp đơn giản là dùng phương pháp FCFS, tức là tiến trình chờ lâu nhất sẽ tiếp tục được thực thi trước. Tuy nhiên, trong nhiều trường hợp, cơ chế điều phối như vậy là không đủ mạnh. Vì vậy, một cấu trúc *conditional-wait* có thể được sử dụng với dạng như sau:

```
x.wait(c);
```

ở đây c là một biểu thức số nguyên được xác định giá trị khi thao tác wait được thực thi. Giá trị c, được gọi là giá trị ưu tiên, được lưu với tên tiến trình tạm dừng. Khi x.signal được thực thi, tiến trình với giá trị ưu tiên nhỏ nhất (có độ ưu tiên cao nhất) được tiếp tục thi hành.

Để biểu diễn cơ chế mới này, chúng ta xem xét monitor được biểu diễn như hình (...), điều khiển việc cấp phát của một tài nguyên đơn giữa các tiến trình cạnh tranh. Mỗi tiến trình khi yêu cầu cấp phát tài nguyên cho nó, phải xác định thời gian tối đa nó hoạch định để sử dụng tài nguyên. Khi đó, monitor sẽ cấp phát tài nguyên tới tiến trình có yêu cầu thời gian cấp phát ngắn nhất.

```
monitor ResourceAllocator
{
    boolean busy;
    condition x;
    void acquire(int time) {
        if (busy)
            x.wait(time);
        busy = TRUE;
    }
    void release() {
        busy = FALSE;
        x.signal();
    }
    initialization code() {
        busy = FALSE;
    }
}
```

Một tiến trình cần truy xuất tài nguyên phải chú ý thứ tự sau:

```
R.acquire(t);
...
<truy xuất tài nguyên>
...
R.release();
```

ở đây R là thể hiện của kiểu ResourceAllocation.

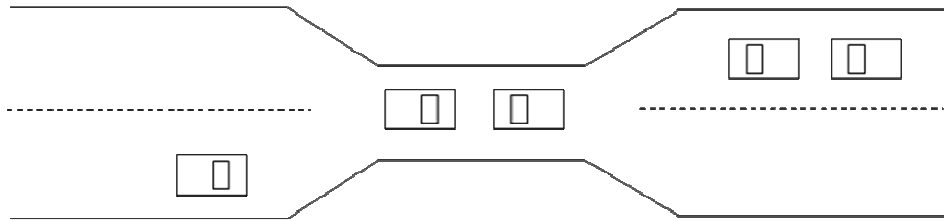
Phần đọc thêm và thảo luận: Các bài toán đồng bộ hoá kinh điển

2.4 Tắc nghẽn (Deadlock) và chống tắc nghẽn

Trong môi trường multiprogramming, một số tiến trình có thể tranh nhau một số tài nguyên hạn chế (một ổ đĩa, một record trong cơ sở dữ liệu, hay một không

gian địa chỉ trên bộ nhớ chính). Một tiến trình yêu cầu các tài nguyên, nếu tài nguyên không thể đáp ứng tại thời điểm đó thì tiến trình sẽ chuyển sang trạng thái chờ. Các tiến trình chờ có thể sẽ không bao giờ thay đổi lại trạng thái được vì các tài nguyên mà nó yêu cầu bị giữ bởi các tiến trình chờ khác.

⇒ ví dụ: tắc nghẽn trên cầu



2.4.1 Mô hình hệ thống

- Các loại tài nguyên R_1, R_2, \dots, R_m
 - Các chu kỳ CPU, không gian bộ nhớ, các tệp, các thiết bị vào-ra
 - Mỗi loại tài nguyên R_i có W_i cá thể (instance).
 - Ví dụ: hệ thống có 2 CPU, có 5 máy in
- ⇒ có thể đáp ứng yêu cầu của nhiều tiến trình hơn
- Mỗi tiến trình sử dụng tài nguyên theo các bước sau:
 - ✓ yêu cầu tài nguyên (request): nếu yêu cầu không được giải quyết ngay (như khi tài nguyên đang được tiến trình khác sử dụng) thì tiến trình yêu cầu phải đợi cho đến khi nhận được tài nguyên.
 - ✓ sử dụng tài nguyên (use)
 - ✓ giải phóng tài nguyên (release)

2.4.2 Mô tả DeadLock

2.4.2.1 Điều kiện xảy ra tắc nghẽn

Deadlock có thể xảy ra nếu 4 điều kiện sau đồng thời tồn tại:

- **Ngăn chặn lẫn nhau** (mutual excution): tại một thời điểm, chỉ một tiến trình có thể sử dụng một tài nguyên.
- **Giữ và đợi** (hold and wait): một tiến trình đang giữ ít nhất một tài nguyên và đợi để nhận được tài nguyên khác đang được giữ bởi tiến trình khác.
- **Không có ưu tiên** (No preemption): một tài nguyên chỉ có thể được tiến trình (tự nguyện!) giải phóng khi nó đã hoàn thành công việc.
- **Chờ đợi vòng tròn** (Circular wait): tồn tại một tập các tiến trình chờ đợi $\{P_0, P_1, \dots, P_n, P_0\}$
 - ✓ P_0 đang đợi tài nguyên bị giữ bởi P_1 ,



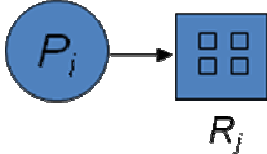
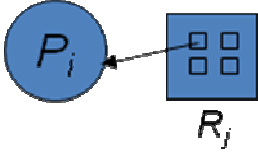
- ✓ P_1 đang đợi tài nguyên bị giữ bởi P_2, \dots
- ✓ P_{n-1} đang đợi tài nguyên bị giữ bởi P_n
- ✓ và P_n đang đợi tài nguyên bị giữ bởi P_0 .

Ba điều kiện đầu là điều kiện cần chứ không phải là điều kiện đủ để xảy ra tắc nghẽn. Điều kiện thứ tư là kết quả tất yếu từ ba điều kiện đầu.

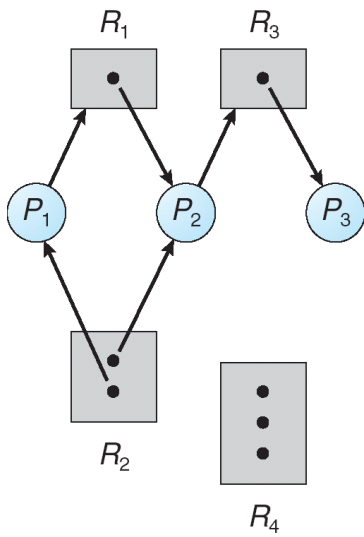
2.4.2.2 Biểu đồ phân phối tài nguyên

Đồ thị phân phối tài nguyên được biểu diễn bởi một tập các đỉnh V và tập các cạnh E như sau:

- Tập đỉnh V được chia thành 2 loại:
 - $P = \{P_1, P_2, \dots, P_n\}$, tập tất cả các tiến trình trong hệ thống.
 - $R = \{R_1, R_2, \dots, R_m\}$, tập các loại tài nguyên trong hệ thống.
- ✓ Mỗi cá thể là một hình vuông bên trong
- cạnh yêu cầu – cạnh có hướng $P_i \rightarrow R_j$. (tiến trình P_i đang đợi nhận một hay nhiều cá thể của tài nguyên R_j).
- cạnh chỉ định – cạnh có hướng $R_j \rightarrow P_i$. (tiến trình P_i đang giữ một hay nhiều cá thể của tài nguyên R_j).

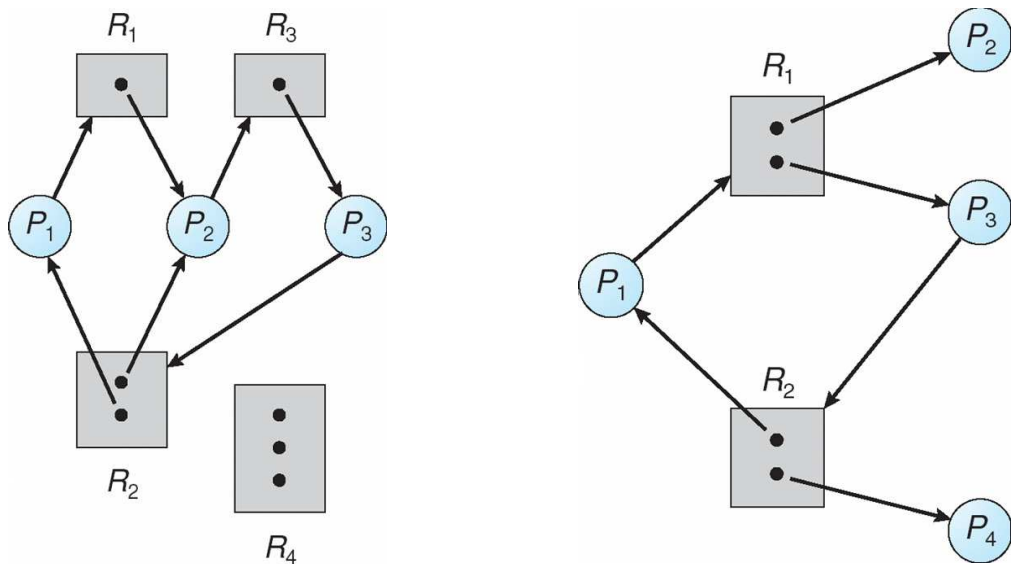
Tiến trình P_i	Tài nguyên R_j có 4 cá thể	P_i yêu cầu cá thể của R_j	P_i đang giữ 1 cá thể của R_j
	 R_j	 R_j	 R_j

Ta xét một ví dụ về đồ thị phân phối tài nguyên như hình 2.6 sau:



Hình 2.6 Một ví dụ về đồ thị phân phối tài nguyên

Đồ thị ở trên không tạo thành một chu trình, tức là các tiến trình có thể có đủ tài nguyên để thực hiện và kết thúc (P_3 có thể kết thúc, trả tài nguyên cho P_2 để P_2 có thể kết thúc, dẫn đến P_1 kết thúc). Điều này có nghĩa là sẽ không có tiến trình nào bị deadlock. Tuy nhiên, nếu đồ thị có chu trình thì *có thể* tồn tại deadlock, như hình sau:



Hình 2.7. Đồ thị phân phối tài nguyên có chu trình

Trường hợp có deadlock Không deadlock: P_4 và P_2 có thể kết thúc, khiến P_1 và P_3 kết thúc được

Như vậy, với đồ thị phân phối tài nguyên có thể kết luận như sau:

- Nếu đồ thị không chu trình:
 - » Không xảy ra deadlock
- Nếu đồ thị có chu trình \Rightarrow
 - » Nếu mỗi loại tài nguyên chỉ một cá thể thì **chắc chắn** xảy ra deadlock.
 - » Nếu mỗi loại tài nguyên có một vài cá thể thì deadlock **có thể** xảy ra.

2.4.3 Các phương thức xử lý tắc nghẽn

Một số phương thức xử lý tắc nghẽn có thể được thực hiện như sau:

- Sử dụng một phương thức để ngăn ngừa hoặc tránh xa, đảm bảo rằng hệ thống sẽ không bao giờ đi vào trạng thái deadlock.
- Cho phép hệ thống đi vào trạng thái deadlock rồi khôi phục lại.
- Bỏ qua vấn đề này và xem như deadlock không bao giờ xuất hiện trong hệ thống. Giải pháp này được sử dụng trong hầu hết các hệ điều hành, bao gồm cả UNIX.

2.4.4 Ngăn chặn tắc nghẽn (Deadlock Prevention)

Ngăn chặn tắc nghẽn là thiết kế một hệ thống sao cho hiện tượng tắc nghẽn bị loại trừ. Các phương thức ngăn chặn tắc nghẽn đều tập trung giải quyết bốn điều kiện gây ra tắc nghẽn, đảm bảo ít nhất một trong 4 điều kiện không thể xuất hiện:

- Đối với điều kiện độc quyền (**Mutual Exclusion**): đảm bảo là hệ thống không có các tài nguyên không thể chia sẻ. Ví dụ, một máy in không thể được chia sẻ cùng lúc bởi nhiều tiến trình. Ngược lại, các tài nguyên có thể chia sẻ không đòi hỏi truy xuất đối với điều kiện độc quyền và do đó không thể liên quan đến deadlock (ví dụ: những tập tin chỉ đọc là một loại tài nguyên có thể chia sẻ. Nếu nhiều tiến trình cố gắng mở một tập tin chỉ đọc tại cùng một thời điểm thì chúng có thể được gán truy xuất cùng lúc tập tin. Một tiến trình không bao giờ phải chờ tài nguyên có thể chia sẻ. Tuy nhiên, thường chúng ta không thể ngăn chặn deadlock bằng cách từ chối điều kiện độc quyền: một số tài nguyên về thực chất không thể chia sẻ.

- Đối với điều kiện giữ và đợi (**Hold and Wait**): Điều kiện này có thể ngăn chặn bằng cách phải đảm bảo rằng mỗi khi tiến trình yêu cầu một tài nguyên, nó không giữ bất kỳ tài nguyên nào khác. Một phương thức thực hiện đó là: (1) đòi hỏi tiến trình yêu cầu và được phân phối tất cả các tài nguyên của nó trước khi nó bắt đầu thực hiện, hoặc chỉ cho phép tiến trình yêu cầu các tài nguyên khi không có tài nguyên nào cả. Một phương thức khác (2) là cho phép một tiến trình yêu cầu tài nguyên chỉ khi tiến trình này không có tài nguyên nào. Một tiến trình có thể yêu cầu một số tài nguyên và dùng chúng. Tuy nhiên, trước khi nó có thể yêu cầu bất kỳ tài nguyên bổ sung nào, nó phải giải phóng tất cả tài nguyên mà nó hiện đang được cấp phát.

Để hiểu hơn sự khác nhau giữa hai phương thức trên, chúng ta xét một quá trình sao chép dữ liệu từ băng từ tới tập tin đĩa, sắp xếp tập tin đĩa và sau đó in kết quả ra máy in. Nếu tất cả tài nguyên phải được yêu cầu cùng một lúc (theo phương thức (1)) thì khởi đầu tiến trình phải yêu cầu băng từ, tập tin đĩa và máy in. Nó sẽ giữ máy in trong toàn thời gian thực thi của nó mặc dù nó cần máy in chỉ ở giai đoạn cuối.

Phương pháp thứ hai (2) cho phép tiến trình yêu cầu ban đầu chỉ băng từ và tập tin đĩa. Nó sao chép dữ liệu từ băng từ tới đĩa, rồi giải phóng cả hai băng từ và đĩa. Sau đó, tiến trình phải yêu cầu lại tập tin đĩa và máy in. Tiếp theo, sao chép tập tin đĩa tới máy in, giải phóng hai tài nguyên này và kết thúc.

- Đối với điều kiện không có ưu tiên (**No preemption**): Điều kiện này có thể ngăn chặn bằng cách: nếu một tiến trình đang giữ một số tài nguyên và yêu cầu tài nguyên khác mà chưa được phân phối ngay (nghĩa là, tiến trình phải chờ) thì nó phải giải phóng tất cả các tài nguyên mà nó đang chiếm giữ. Các tài nguyên này được ưu tiên thêm vào danh sách các tài nguyên mà tiến trình đang chờ. Tiến trình sẽ được khởi động lại chỉ khi nó có thể lấy lại các tài nguyên cũ của nó cũng như các tài nguyên mới mà nó đang yêu cầu.

Phương thức này thường được áp dụng tới các tài nguyên mà trạng thái của nó có thể được lưu lại dễ dàng và phục hồi lại sau đó, như các thanh ghi CPU và

không gian bộ nhớ. Nó thường không thể được áp dụng cho các tài nguyên như máy in và băng từ.

- Đối với điều kiện chờ đợi vòng tròn (**Circular Wait**): Điều kiện thứ tư và cũng là điều kiện cuối cùng cho deadlock là điều kiện tồn tại chu trình trong đồ thị cấp phát tài nguyên. Một cách để đảm bảo rằng điều kiện này không bao giờ xảy ra là áp dụng một thứ tự tuyệt đối cho tất cả các loại tài nguyên: mỗi loại được gán một số nguyên.

- Mỗi tiến trình yêu cầu các tài nguyên theo thứ tự tăng dần: chỉ có thể nhận được tài nguyên có trọng số cao hơn của bất kỳ tài nguyên nào nó đang chiếm giữ.
- Muốn có tài nguyên j , tiến trình phải giải phóng tất cả các tài nguyên có trọng số $i > j$ (nếu có).

2.4.5 Tránh khỏi tắc nghẽn

Một phương pháp khác để tránh deadlock là yêu cầu Hệ điều hành phải có một số thông tin ưu tiên về cách tài nguyên được yêu cầu. Ví dụ, trong một hệ thống với một ổ băng từ và một máy in, chúng ta có thể đảm bảo rằng tiến trình P sẽ yêu cầu ổ băng từ trước và sau đó đến máy in, trước khi giải phóng cả hai tài nguyên. Trái lại, tiến trình Q sẽ yêu cầu máy in trước và ổ băng từ sau. Với sự nhận biết về thứ tự hoàn thành của yêu cầu và giải phóng cho mỗi tiến trình, chúng ta có thể quyết định cho mỗi yêu cầu của tiến trình sẽ chờ hay không. Mỗi yêu cầu đòi hỏi hệ thống xem tài nguyên hiện có, tài nguyên hiện được cấp tới mỗi tiến trình, các yêu cầu và giải phóng tương lai của mỗi tiến trình, để yêu cầu của tiến trình hiện tại có thể được thoả mãn hay phải chờ, tránh khả năng xảy ra deadlock.

Các giải thuật khác nhau có sự khác nhau về lượng và loại thông tin được yêu cầu. Mô hình đơn giản và hữu ích nhất là yêu cầu mỗi tiến trình phải khai báo số lượng tài nguyên lớn nhất của mỗi loại mà nó có thể cần đến. Dựa vào các thông tin trước về số lượng tối đa tài nguyên của mỗi loại được yêu cầu cho mỗi tiến trình, có thể xây dựng một giải thuật để đảm bảo hệ thống sẽ không bao giờ đi vào trạng thái deadlock. Đây là giải thuật tiếp cận tránh deadlock. Giải thuật tránh deadlock luôn kiểm tra trạng thái phân phối tài nguyên để đảm bảo rằng sẽ không bao giờ có tình trạng chờ đợi vòng tròn. Trạng thái phân phối tài nguyên được xác định bởi số tài nguyên khả dụng và đã được phân phối cũng như sự yêu cầu tối đa từ các tiến trình.

2.4.5.1 Trạng thái an toàn (Safe State).

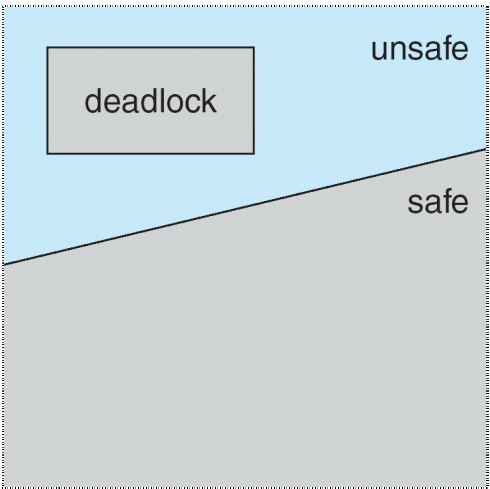
Một trạng thái là an toàn nếu hệ thống có thể phân phối các tài nguyên cho mỗi tiến trình mà vẫn tránh được deadlock. Khi một tiến trình yêu cầu một tài nguyên còn rồi, hệ thống phải quyết định liệu phân phối ngay lập tức có thể làm cho hệ thống mất an toàn hay không? Hệ thống ở trong trạng thái an toàn nếu tồn tại một chuỗi an toàn của tất cả các tiến trình.

Chuỗi $\langle P_1, P_2, \dots, P_n \rangle$ là thứ tự an toàn cho trạng thái phân phối hiện thời nếu với mỗi P_i , tài nguyên mà nó yêu cầu có thể được cung cấp bởi tài nguyên khả dụng hiện tại và các tài nguyên đang được giữ bởi P_j , với $j < i$. Trong trường hợp nếu tài

nguyên mà tiến trình P_i yêu cầu không sẵn dùng tức thời (đang bị các P_j giữ) thì nó có thể đợi cho đến khi tất cả các P_j kết thúc. Khi các P_j kết thúc, P_i có thể giành được các tài nguyên cần thiết, thực hiện công việc của nó, rồi trả lại các tài nguyên đó và kết thúc. Khi P_i kết thúc, P_{i+1} có thể giành được tài nguyên cần thiết.

Như vậy, một thực tế dễ dàng nhận thấy (hình 2.8), đó là:

- Nếu hệ thống ở trạng thái an toàn => không có deadlock.
- Nếu hệ thống ở trạng thái không an toàn => có thể có deadlock.



Hình 2.8. Trạng thái hệ thống

Để minh hoạ, chúng ta xét một hệ thống với 12 ổ băng từ và 3 tiến trình: P_0 , P_1 , P_2 . Tiến trình P_0 yêu cầu 10 ổ băng từ, tiến trình P_1 có thể cần 4 và tiến trình P_2 có thể cần tới 9 ổ băng từ. Giả sử rằng tại thời điểm t_0 , tiến trình P_0 giữ 5 ổ băng từ, tiến trình P_1 giữ 2 và tiến trình P_2 giữ 2 ổ băng từ. (Do đó, có 3 ổ băng từ còn khả dụng).

	Max	Allocation
P_0	10	5
P_1	4	2
P_2	9	2

Tại thời điểm t_0 , hệ thống ở trạng thái an toàn. Chuỗi tiến trình $\langle P_1, P_0, P_2 \rangle$ thỏa điều kiện an toàn vì tiến trình P_1 có thể được cấp phát tức thì tất cả các ổ đĩa từ và sau đó trả lại chúng (hệ thống có 5 ổ băng từ khả dụng), tiếp theo tiến trình P_0 có thể nhận tất cả ổ băng từ và trả lại chúng (hệ thống sẽ có 10 ổ băng từ khả dụng), và cuối cùng tiến trình P_2 có thể nhận tất cả ổ băng từ của nó và trả lại chúng (hệ thống sẽ có tất cả 12 ổ băng từ khả dụng).

Một hệ thống có thể đi từ trạng thái an toàn tới một trạng thái không an toàn. Giả sử rằng tại thời điểm t_1 , tiến trình P_2 yêu cầu và được cấp 1 ổ băng từ nữa. Hệ

thống không còn trong trạng thái an toàn. Tại thời điểm này, chỉ tiến trình P_1 có thể được cấp tất cả ổ băng từ của nó. Khi nó trả lại chúng, hệ thống chỉ còn 4 ổ băng từ khả dụng. Vì tiến trình P_0 yêu cầu 5 ổ băng từ nữa (yêu cầu tối đa 10, đã được cấp phát 5), nên tiến trình P_0 phải chờ. Tương tự, tiến trình P_2 có thể yêu cầu thêm 6 ổ băng từ và phải chờ, dẫn đến deadlock.

Lỗi xảy ra ở đây là do thoả mãn yêu cầu từ tiến trình P_2 thêm 1 ổ băng từ nữa. Nếu ta buộc P_2 phải chờ cho đến khi các tiến trình khác kết thúc và giải phóng tài nguyên của chúng thì hoàn toàn có thể tránh được deadlock.

Do vậy, để tránh khỏi deadlock, phải đảm bảo rằng hệ thống sẽ không bao giờ bước vào trạng thái không an toàn. Ban đầu, hệ thống ở trong trạng thái an toàn. Bất cứ khi nào một tiến trình yêu cầu một tài nguyên hiện có, hệ thống phải quyết định tài nguyên có thể được cấp phát tức thì hay tiến trình phải chờ. Yêu cầu được thoả mãn chỉ nếu việc cấp phát mà hệ thống vẫn ở trong trạng thái an toàn. Các giải thuật tránh tắc nghẽn có thể được sử dụng với các trường hợp sau:

- Mỗi loại tài nguyên có một cá thể: sử dụng giải thuật đồ thị phân phối tài nguyên (resource-allocation graph).
- Mỗi loại tài nguyên có nhiều cá thể: sử dụng giải thuật chủ nhà băng (banker's algorithm).

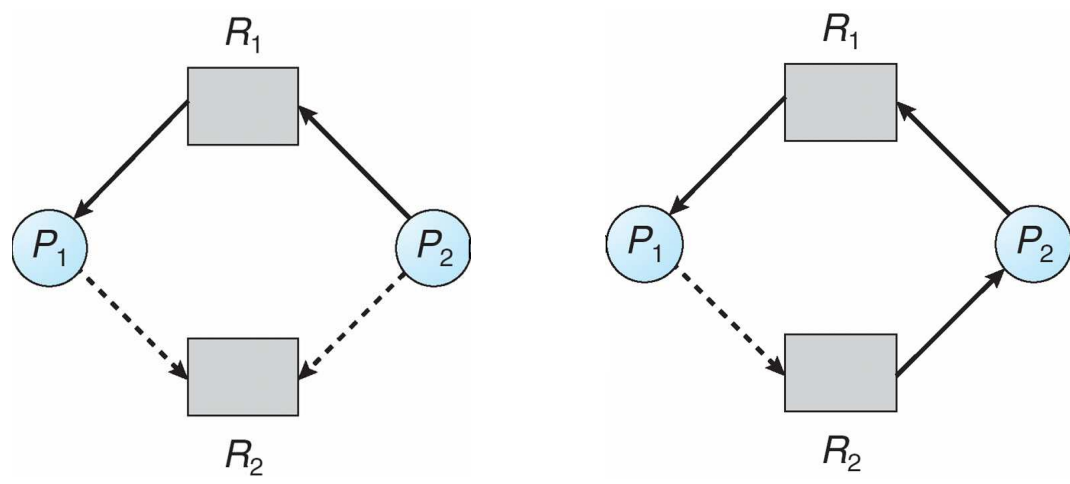
2.4.5.2 Giải thuật đồ thị phân phối tài nguyên

Dựa theo biểu đồ phân phối tài nguyên khai báo ở phần trên, ngoài các cạnh yêu cầu và cạnh chỉ định, ta định nghĩa thêm một loại cạnh mới, gọi là cạnh muốn yêu cầu (*claim edge*) $P_i \rightarrow R_j$ biểu thị tiến trình P_i có thể yêu cầu tài nguyên R_j vào một thời điểm trong tương lai; được biểu diễn bởi một đường đứt nét. Cạnh muốn yêu cầu biến thành cạnh yêu cầu (request edge) khi một tiến trình yêu cầu một tài nguyên. Tương tự, khi tài nguyên được một tiến trình giải phóng, cạnh yêu cầu lại thành cạnh muốn yêu cầu. Chú ý rằng, các tài nguyên phải được yêu cầu trước trong hệ thống. Nghĩa là, trước khi P_i bắt đầu thực thi, tất cả các cạnh muốn yêu cầu của nó phải xuất hiện trong đồ thị phân phối tài nguyên.

Giả sử rằng P_i yêu cầu tài nguyên R_j . Yêu cầu có thể được thoả mãn chỉ nếu chuyển cạnh muốn yêu cầu $P_i \rightarrow R_j$ tới cạnh chỉ định $R_j \rightarrow P_i$ không dẫn đến việc hình thành chu trình trong đồ thị phân phối tài nguyên. Chú ý rằng chúng ta kiểm tra tính an toàn bằng cách dùng giải thuật phát hiện chu trình. Một giải thuật để phát hiện một chu trình trong đồ thị này yêu cầu một thứ tự của n^2 thao tác, ở đây n là số tiến trình trong hệ thống.

Nếu không có chu trình tồn tại, thì việc cấp phát tài nguyên sẽ vẫn giữ hệ thống trong trạng thái an toàn. Nếu chu trình được tìm thấy thì việc cấp phát sẽ đặt hệ thống trong trạng thái không an toàn. Do đó, tiến trình P_i sẽ phải chờ yêu cầu của nó được thoả.

Để minh hoạ giải thuật này, chúng ta xét đồ thị cấp phân phối tài nguyên như hình 2.9 sau.



Hình 2.9. Đồ thị phân phối tài nguyên

Giả sử rằng P_2 yêu cầu R_2 . Dù R_2 vẫn đang tự do, ta vẫn không thể phân phối nó cho P_2 vì hoạt động này sẽ tạo ra một chu trình, dẫn đến hệ thống ở trong trạng thái không an toàn. Nếu P_1 yêu cầu R_2 và P_2 yêu cầu R_1 thì deadlock sẽ xảy ra.

2.4.5.3 Giải thuật chủ nhà băng (Banker's Algorithm)

Giải thuật đồ thị cấp phân phối tài nguyên ở trên không thể áp dụng cho hệ thống cấp phát tài nguyên với nhiều cá thể của mỗi loại tài nguyên. Giải thuật tránh deadlock mà chúng ta mô tả tiếp theo có thể áp dụng tới một hệ thống với nhiều cá thể của mỗi loại tài nguyên. Giải thuật này thường được gọi là giải thuật chủ nhà băng (Banker's algorithm). Giải thuật này có thể được sử dụng trong hệ thống nhà băng để đảm bảo rằng nhà băng không bao giờ phân phối quá số tiền khả dụng của nó đến mức mà nó có thể thỏa mãn mọi yêu cầu từ các khách hàng.

Khi một tiến trình mới đi vào hệ thống, nó phải khai báo số lượng tối đa cá thể của mỗi loại tài nguyên mà nó có thể cần đến. Số này có thể vượt quá tổng số tài nguyên trong hệ thống. Khi một người dùng yêu cầu tài nguyên, hệ thống phải xác định liệu sự phân phối có giữ hệ thống trong trạng thái an toàn hay không. Nếu trạng thái hệ thống sẽ là an toàn, tài nguyên sẽ được cấp; ngược lại tiến trình phải chờ cho tới khi một vài tiến trình giải phóng đủ tài nguyên.

a. Cấu trúc dữ liệu

Đặt n là số tiến trình trong hệ thống và m là số loại tài nguyên trong hệ thống. Chúng ta cần các cấu trúc dữ liệu sau:

- **Available:** vector độ dài m – số lượng tài nguyên khả dụng của mỗi loại:
 - Nếu $Available[j] = k$: có k cá thể của loại tài nguyên R_j là khả dụng.
- **Max:** ma trận $n \times m$ - xác định số lượng tối đa yêu cầu của mỗi tiến trình.
 - Nếu $Max[i,j] = k$: tiến trình P_i có thể yêu cầu tối đa k cá thể của loại tài nguyên R_j .

- **Allocation:** ma trận $n \times m$ - xác định số lượng tài nguyên mỗi loại hiện đang được phân phối cho mỗi tiến trình.
 - Nếu $Allocation[i,j] = k$: P_i hiện đang được phân phối k cá thể của tài nguyên R_j .
- **Need:** ma trận $n \times m$ - xác định số tài nguyên còn thiếu cho mỗi tiến trình.
 - $Need[i,j] = k$: P_i có thể cần k cá thể nữa của R_j ;
 - $Need[i,j] = Max[i,j] - Allocation[i,j]$
- Cấu trúc dữ liệu này biến đổi theo thời gian về kích thước và giá trị.

Để đơn giản việc trình bày của giải thuật Banker, chúng ta thiết lập vài ký hiệu. Gọi X và Y là các vector có chiều dài n . Chúng ta nói rằng $X \leq Y$ nếu và chỉ nếu $X[i] \leq Y[i]$ cho tất cả $i = 1, 2, \dots, n$. Thí dụ, nếu $X = (1, 7, 3, 2)$ và $Y = (0, 3, 2, 1)$ thì $Y \leq X$, $Y < X$ nếu $Y \leq X$ và $Y \neq X$.

Chúng ta có thể xem xét mỗi dòng trong ma trận $Allocation$ và $Need$ như là những vectors và tham chiếu tới chúng như $Allocation_i$ và $Need_i$ tương ứng. Vector $Allocation_i$ xác định tài nguyên hiện được cấp phát cho tiến trình P_i ; vector $Need_i$ xác định các tài nguyên cần bổ sung mà tiến trình P_i có thể vẫn yêu cầu để hoàn thành tác vụ của nó.

b. Giải thuật kiểm tra trạng thái an toàn (Safety Algorithm)

Ý tưởng: tìm một chuỗi an toàn. Nếu tìm được, trạng thái là an toàn, ngược lại trạng thái là không an toàn.

1. Gán $Work$ và $Finish$ là các vector có độ dài m và n , một cách tương ứng. Khởi tạo:

$Work := Available$

$Finish[i] := (Allocation_i == 0)$ với $i = 1, 2, \dots, n$

2. Tìm i thỏa mãn cả 2 điều kiện:

a) $Finish[i] = false$ và

b) $Need_i \leq Work$

Nếu không có i như vậy, nhảy đến bước 4.

3. $Work := Work + Allocation_i$

$Finish[i] := true$

nhảy đến bước 2

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

Giải thuật này có thể yêu cầu độ phức tạp $m.n^2$ thao tác để quyết định trạng thái là an toàn hay không.

- c. Giải thuật yêu cầu tài nguyên đối với tiến trình P_i*

Đặt ***Request_i*** là vector yêu cầu cho tiến trình P_i . Nếu $Request_i[j] = k$, thì tiến trình P_i muốn k cá thể của loại tài nguyên R_j . Khi một yêu cầu tài nguyên được thực hiện bởi tiến trình P_i , thì các bước sau được thực hiện:

- 1. Nếu $Request_i \leq Need_i$, chuyển sang bước 2. Trái lại, phát sinh trạng thái lỗi do tiến trình đã vượt quá yêu cầu tối đa của nó.
- 2. Nếu $Request_i \leq Available_i$, chuyển sang bước 3. Trái lại P_i phải đợi vì tài nguyên chưa sẵn sàng.
- 3. Phân phối *thử* các tài nguyên cho P_i bằng cách sửa trạng thái như sau:

$Available = Available - Request_i ;$
 $Allocation_i = Allocation_i + Request_i ;$
 $Need_i = Need_i - Request_i ;$

- Nếu kết quả trạng thái phân phối tài nguyên là an toàn, thì giao dịch được hoàn thành và tiến trình P_i được cấp phát tài nguyên của nó.
- Tuy nhiên, nếu trạng thái mới là không an toàn, thì P_i phải chờ và trạng thái phân phối tài nguyên cũ được phục hồi.

d. Ví dụ minh họa

Xét một hệ thống với 5 tiến trình từ P_0 tới P_4 , và 3 loại tài nguyên A, B, C. Loại tài nguyên A có 10 cá thể, loại tài nguyên B có 5 cá thể và loại tài nguyên C có 7 cá thể. Giả sử rằng tại thời điểm T_0 trạng thái hiện tại của hệ thống như sau:

	Allocation			Max			Available		
	A	B	C	A	B	C	A	B	C
P_0	0	1	0	7	5	3	3	3	2
P_1	2	0	0	3	2	2			
P_2	3	0	2	9	0	2			
P_3	2	1	1	2	2	2			
P_4	0	0	2	4	3	3			

Nội dung ma trận Need được tính từ Max-Allocation như sau:

	Need		
	A	B	C
P_0	7	4	3
P_1	1	2	2

P ₂	6	0	0
P ₃	0	1	1
P ₄	4	3	1

Hệ thống đang ở trạng thái an toàn vì chuỗi tiến trình <P₁, P₃, P₄, P₂, P₀> thỏa mãn các điều kiện an toàn.

Giả sử bây giờ P₁ yêu cầu thêm một cá thể loại A và hai cá thể loại C, vì vậy Request₁ = (1, 0, 2). Để quyết định yêu cầu này có thể được thỏa mãn tức thì hay không, trước tiên chúng ta phải thực hiện:

- Kiểm tra Request ≤ Need ↔ (1,0,2) ≤ (1,2,2) ⇒ true.
- Kiểm tra Request ≤ Available ↔ (1,0,2) ≤ (3,3,2) ⇒ true.
- *Thử đáp ứng yêu cầu*, hệ thống sẽ đến trạng thái mới sau:

	Allocation			Need			Available		
	A	B	C	A	B	C	A	B	C
P ₀	0	1	0	7	4	3	2	3	0
P ₁	3	0	2	0	2	0			
P ₂	3	0	2	6	0	0			
P ₃	2	1	1	0	1	1			
P ₄	0	0	2	4	3	1			

Việc thực hiện giải thuật kiểm tra trạng thái an toàn cho thấy chuỗi <P₁, P₃, P₄, P₀, P₂> vẫn thỏa mãn các yêu cầu an toàn, vì vậy có thể chấp nhận ngay yêu cầu từ P₁.

Tuy nhiên, chúng ta cũng thấy rằng, khi hệ thống ở trong trạng thái này, một yêu cầu (3, 3, 0) bởi P₄ không thể được thực hiện vì các tài nguyên là không sẵn dùng. Một yêu cầu khác là (0, 2, 0) bởi P₀ cũng không thể được cấp mặc dù tài nguyên là sẵn dùng vì trạng thái kết quả là không an toàn.

2.4.6 Phát hiện tắc nghẽn (Deadlock Detection)

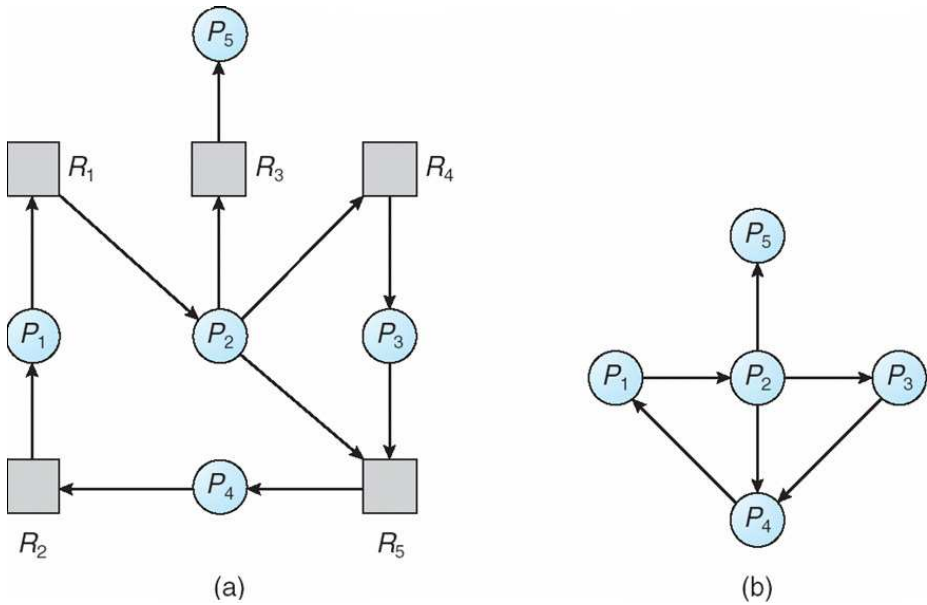
Nếu một hệ thống không thể thực hiện được việc ngăn ngừa hay tránh xa deadlock thì deadlock có thể xuất hiện. Trong môi trường này, hệ thống phải cung cấp:

- Giải thuật phát hiện deadlock
- Giải thuật phục hồi từ deadlock.

2.4.6.1 Mỗi loại tài nguyên có 1 cá thể

Khi tất cả tài nguyên chỉ có 1 cá thể, giải thuật phát hiện deadlock sử dụng 1 dạng biến thể của đồ thị phân phối tài nguyên, bằng cách bỏ đi các nút của loại tài nguyên và xoá các cạnh tương ứng, ta gọi là đồ thị *wait-for*.

- Các nút là các tiến trình.
- $P_i \rightarrow P_j$ nếu P_i đang đợi P_j .



Hình 2.10 Đồ thị phân phối tài nguyên (a) và Đồ thị wait-for tương ứng (b)

Như đã đề cập ở trước, deadlock tồn tại trong hệ thống nếu và chỉ nếu đồ thị *wait-for* chứa chu trình. Để phát hiện deadlock, hệ thống cần duy trì đồ thị chờ và định kỳ gọi giải thuật để tìm kiếm chu trình trong đồ thị.

Một giải thuật phát hiện chu trình trong đồ thị yêu cầu độ phức tạp n^2 phép toán, với n là số số đỉnh trong đồ thị.

2.4.6.2 Mỗi loại tài nguyên có nhiều cá thể

Lược đồ đồ thị *wait-for* không thể áp dụng đối với hệ thống phân phối tài nguyên với nhiều cá thể cho mỗi loại tài nguyên. Giải thuật phát hiện deadlock mà chúng ta mô tả sau đây có thể áp dụng cho hệ thống này. Giải thuật thực hiện nhiều cấu trúc dữ liệu thay đổi theo thời gian tương tự như trong giải thuật nhà băng ở trên.

- *Available*: vectơ độ dài m xác định số tài nguyên khả dụng của mỗi loại.
- *Allocation*: ma trận $n \times m$ xác định các tài nguyên của mỗi loại hiện đang được phân phối cho mỗi tiến trình.
- *Request*: ma trận $n \times m$ xác định yêu cầu hiện tại của mỗi tiến trình. Nếu $Request[i,j] = k$, thì tiến trình P_i đang yêu cầu k cá thể nữa của loại tài nguyên R_j .

Giải thuật phát hiện deadlock được mô tả như sau:

- 1. Gán *Work* và *Finish* là các vector độ dài *m* và *n*. Khởi tạo:
 - a) *Work*: = *Available*
 - b) *For i:=1 to n do If Allocation_i ≠ 0 then Finish[i] := false*
Else Finish[i] := True

2. Tìm chỉ số *i* thỏa mãn cả 2 điều kiện:

2.4 *Finish[i] = false*

2.5 *Request_i ≤ Work*

Nếu không tồn tại *i*, nhảy sang bước 4.

3. *Work := Work + Allocation_i*

Finish[i] := true

nhảy sang bước 2

4. Nếu *Finish[i] = true* với mọi *i* thì hệ thống không có deadlock

Nếu *Finish[i] = false*, với số các giá trị *i*, $1 \leq i \leq n$, thì *P_i* bị deadlock, hệ thống ở trong trạng thái deadlock.

Độ phức tạp tính toán của giải thuật là $O(m \times n^2)$.

2.4.6.3 Ví dụ minh họa

Để minh họa giải thuật ở trên, chúng ta xét hệ thống với 5 tiến trình P₀ đến P₄ và 3 loại tài nguyên A, B, C. Loại tài nguyên A có 7 cá thể, loại tài nguyên B có 2 cá thể và loại tài nguyên C có 6 cá thể. Giả sử rằng tại thời điểm T₀, chúng ta có trạng thái phân phối tài nguyên sau:

	Allocation			Request			Available		
	A	B	C	A	B	C	A	B	C
P ₀	0	1	0	0	0	0	0	0	0
P ₁	2	0	0	2	0	2			
P ₂	3	0	3	0	0	0			
P ₃	2	1	1	1	0	0			
P ₄	0	0	2	0	0	2			

Chúng ta khẳng định rằng hệ thống không ở trong trạng thái deadlock. Thật vậy, nếu chúng ta thực thi giải thuật, chúng ta sẽ tìm ra chuỗi thứ tự <P₀, P₂, P₃, P₁, P₄> cho kết quả *Finish[i] = true* với tất cả *i*. Vì vậy sẽ không có deadlock.

Bây giờ giả sử rằng tiến trình P₂ thực hiện yêu cầu thêm cá thể loại C. Ma trận

yêu cầu được hiệu chỉnh như sau:

	Need		
	A	B	C
P ₀	0	0	0
P ₁	2	0	2
P ₂	0	0	1
P ₃	1	0	0
P ₄	0	0	2

- Trạng thái của hệ thống?
 - Có thể phục hồi các tài nguyên bị giữ bởi tiến trình P_0 khi nó kết thúc, nhưng không đủ tài nguyên để hoàn thành các tiến trình khác.
 - Deadlock xuất hiện, gồm các tiến trình P_1, P_2, P_3 và P_4

2.4.6.4 Cách sử dụng giải thuật phát hiện deadlock

Thời điểm và mức thường xuyên cần đến giải thuật phụ thuộc vào hai yếu tố:

- Deadlock có khả năng thường xuyên xảy ra như thế nào?
- Có bao nhiêu tiến trình bị tác động khi deadlock xuất hiện.

Nếu giải thuật phát hiện deadlock ít được sử dụng, có thể có nhiều chu trình trong biểu đồ tài nguyên và do đó ta không thể tìm được những tiến trình nào “gây ra” deadlock.

Nếu phát hiện ra deadlock, ta cần khôi phục lại bằng một trong hai cách:

- Dừng các tiến trình.
- Buộc chúng phải giải phóng tài nguyên (ưu tiên trước).

2.4.7 Phục hồi hệ thống sau tắc nghẽn

Khi giải thuật phát hiện xác định rằng deadlock tồn tại, nhiều khả năng có thể được chọn lựa. Một khả năng là thông báo cho hệ điều hành deadlock xảy ra và để hệ điều hành giải quyết deadlock bằng thủ công. Một khả năng khác là để hệ thống tự động phục hồi. Có hai giải pháp cho việc phá vỡ deadlock. Giải pháp đơn giản là huỷ bỏ một hay nhiều tiến trình để phá vỡ việc tồn tại chu trình trong đồ thị phân phối tài nguyên. Giải pháp thứ hai là thu hồi lại một số tài nguyên từ một hay nhiều tiến trình bị deadlock.

2.4.7.1 Dừng tiến trình

Để phá vỡ deadlock bằng cách huỷ bỏ quá trình, chúng ta dùng một trong hai phương pháp. Trong cả hai phương pháp, hệ thống lấy lại tài nguyên được cấp phát

đối với tiến trình bị kết thúc.

- Hủy bỏ tất cả tiến trình bị deadlock: phương pháp này rõ ràng sẽ phá vỡ chu trình deadlock, nhưng chi phí cao; các tiến trình này có thể đã thực hiện các phép tính toán trong thời gian dài, và các kết quả của các tính toán từng phần này phải bị bỏ đi và có thể phải tính lại sau đó.
- Hủy bỏ một tiến trình tại thời điểm cho đến khi chu trình deadlock loại trừ: phương pháp này chịu một phần chi phí vì sau khi tiến quá trình bị hủy bỏ, một giải thuật phát hiện deadlock phải được nạp lên để xác định có tiến trình nào vẫn đang bị deadlock.

Việc hủy bỏ quá trình là không đơn giản. Nếu một tiến trình đang ở giữa giai đoạn cập nhật một tập tin, kết thúc nó sẽ dẫn đến tập tin đó ở trong trạng thái không phù hợp. Tương tự, nếu tiến trình đang ở giữa giai đoạn in dữ liệu ra máy in, hệ thống phải khởi động lại trạng thái đúng trước khi in công việc tiếp theo.

Nếu phương pháp kết thúc một phần được sử dụng thì với một tập hợp các tiến trình deadlock, chúng ta phải xác định tiến trình nào (hay các tiến trình nào) nên được kết thúc để phá vỡ deadlock. Việc xác định này là một quyết định chính sách tương tự như các vấn đề điều phối CPU. Nếu xét về tính kinh tế, rõ ràng ta nên hủy bỏ tiến trình nào mà sự kết thúc của nó sẽ chịu chi phí tối thiểu. Tuy nhiên, thuật ngữ chi phí tối thiểu là không chính xác. Có nhiều yếu tố có thể xác định tiến trình nào được lựa chọn, bao gồm:

- Theo mức ưu tiên của tiến trình.
- Theo thời gian tiến trình đã thực hiện, và thời gian cần thiết còn lại để hoàn thành.
- Theo tài nguyên tiến trình đã sử dụng.
- Theo tài nguyên tiến trình cần hoàn thành.
- Theo số tiến trình sẽ cần bị dừng.
- Tiến trình là tiến trình tương tác hay tiến trình bó.

2.4.7.2 Ưu tiên trước tài nguyên

- Chọn một số tiến trình nạn nhân dựa vào giá trị nhỏ nhất (mức ưu tiên, số tài nguyên đang dùng,...)
- Rollback – quay lại trạng thái an toàn trước, khởi động lại tiến trình ở trạng thái đó.
- Starvation – 1 tiến trình có thể luôn bị chọn làm nạn nhân khiến nó không thể kết thúc. Phải đảm bảo rằng một tiến trình được chọn làm nạn nhân chỉ trong khoảng thời gian ngắn.

Giải pháp: Thêm các rollback vào yếu tố giá trị.

Một số câu hỏi và bài tập

Câu 1. Giả sử rằng tại thời điểm thứ 5 không có tài nguyên hệ thống nào đang được sử dụng ngoại trừ vi xử lý và bộ nhớ chính. Bây giờ hãy theo dõi các sự kiện sau:

- Tại thời điểm thứ 5: Tiến trình P1 thực hiện một lệnh đọc từ đĩa Unit 3
- Tại thời điểm thứ 15: thời gian phân chia của P5 kết thúc.
- Tại thời điểm thứ 18 : tiến trình P7 thực hiện một lệnh viết vào đĩa Unit 3.
- Tại thời điểm thứ 20: tiến trình P3 thực hiện một lệnh để đọc từ đĩa Unit 2.
- Tại thời điểm thứ 24: tiến trình P5 thực hiện một lệnh để viết vào Unit3.
- Tại thời điểm thứ 28: tiến trình P5 bị đưa ra ngoài.
- Tại thời điểm thứ 33: một ngắt xuất hiện từ đĩa Unit 2: Quá trình đọc của P3 được cung cấp đầy đủ tài nguyên (quá trình đọc của P3 đầy đủ).
- Tại thời điểm thứ 36: một ngắt xuất hiện từ đĩa Unit 3: Quá trình đọc của P1 được cung cấp đầy đủ tài nguyên.
- Tại thời điểm 38: Tiến trình P8 chấm dứt.
- Tại thời điểm 40: Một ngắt xuất hiện từ Unit 3: Quá trình viết của P5 được cung cấp đầy đủ tài nguyên.
- Tại thời điểm thứ 44: tiến trình P5 được đưa vào trở lại.
- Tại thời điểm thứ 48: một ngắt xuất hiện từ đĩa Unit 3: Quá trình viết của P7 được cung cấp đầy đủ tài nguyên.

Yêu cầu: Tại mỗi thời điểm thứ 22, 37 và 47, hãy cho biết trạng thái của mỗi tiến trình. Nếu một tiến trình bị khóa (blocked), thì hãy cho biết thêm về sự kiện đã khóa nó (hay là sự kiện mà nó mong đợi).

Câu 2. Vẽ sơ đồ chuyển trạng thái tiến trình (mô hình tiến trình 3 trạng thái). Khi nào thì tiến trình bị chuyển từ trạng thái Running sang trạng thái Ready.

Câu 3. Xét một biến thể của thuật toán Round Robin (RR), nơi mà các mục trong hàng đợi sẵn sàng (ready queue) được trở đến bởi bộ điều phối tiến trình (PCB).

a. Điều gì sẽ xảy ra nếu cho hai con trỏ cùng trỏ tới một tiến trình trong hàng đợi sẵn sàng (ready queue)?

b. Những ưu điểm lớn của việc làm này là gì?

c. Làm sao để có thể biến đổi thuật toán RR cơ bản để đạt được kết quả tương tự như cách làm này mà không cần sử dụng đến con trỏ thứ hai?

Câu 4. Giả sử có hai tiến trình foo and bar hoạt động đồng thời và cùng chia sẻ đèn báo biến biến S và R (được khởi tạo bằng 1) và biến nguyên x (được khởi tạo bằng 0)

Void foo()

```
{ Do
    {
        Semwait (S);
        Semwait (R);
        X++;
        Semsignal (S);
        Semsignal ( R);
    }
    While (1)
}
```

```

Void bar()
{ Do
  {
    Semwait ( R);
    Semwait (S);
    X--;
    Semsignal (S);
    Semsignal ( R);
  }
  While ( 1)
}

```

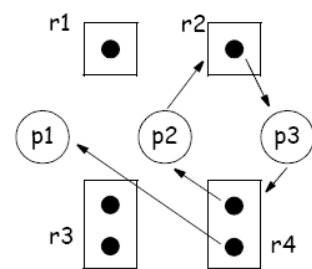
a) Nếu cả hai tiến trình hoạt động đồng thời thì có dẫn đến kết quả một hoặc cả hai tiến trình rơi vào trạng thái block mãi mãi không? Nếu có thì hãy chỉ ra câu lệnh mà ở đó một hoặc hai tiến trình sẽ rơi vào trạng thái block mãi mãi.

b) Nếu cả hai tiến trình hoạt động đồng thời thì có dẫn đến kết quả chỉ một tiến trình rơi vào trạng thái block mãi mãi không? Nếu có thì hãy chỉ ra câu lệnh mà ở đó một tiến trình sẽ rơi vào trạng thái block mãi mãi.

Câu 5. Thế nào là tài nguyên găng và đoạn găng ? Trình bày ý tưởng của giải pháp Semaphore điều độ tiến trình qua đoạn găng.

Câu 6. Tắc nghẽn (deadlock) là gì? Cho biết điều kiện để xảy ra tắc nghẽn?

Câu 7. Giả sử đồ thị phân phối tài nguyên tại thời điểm T0 như hình bên. Có thể xảy ra tắc nghẽn hay không? Giải thích.



Chương 3 QUẢN LÝ BỘ NHỚ

Trong chương này chúng ta sẽ thảo luận nhiều cách thức khác nhau để quản lý bộ nhớ. Các giải thuật quản lý bộ nhớ cơ bản là chiến lược phân trang và phân đoạn. Mỗi tiếp cận có lợi điểm và nhược của chính nó. Chọn phương pháp quản lý bộ nhớ cho một hệ thống xác định phụ thuộc vào nhiều yếu tố, đặc biệt trên thiết kế phần cứng của hệ thống. Chúng ta sẽ thấy nhiều giải thuật yêu cầu hỗ trợ phần cứng mặc dù các thiết kế gần đây đã tích hợp phần cứng và hệ điều hành.

3.1 Bộ nhớ chính

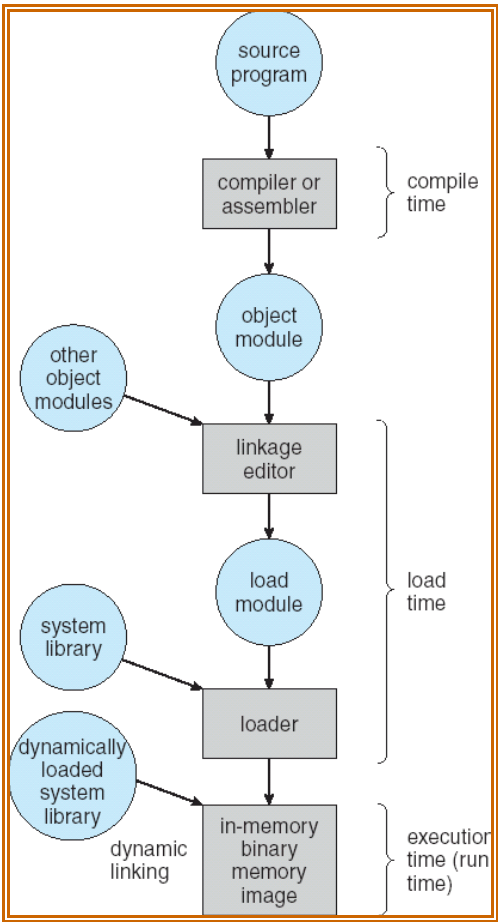
3.1.1 Tổng quan

3.1.1.1 Liên kết địa chỉ

Thông thường, một chương trình nằm trên đĩa như một tập tin có thể thực thi dạng nhị phân. Chương trình cần được nạp vào bộ nhớ chính để thi hành (sẽ được tổ chức theo cấu trúc của tiến trình tương ứng). Phụ thuộc vào việc quản lý bộ nhớ đang dùng, tiến trình có thể được di chuyển giữa đĩa và bộ nhớ chính trong khi thực thi. Tập hợp các tiến trình trên đĩa đang chờ được đưa vào bộ nhớ để thực thi hình thành một hàng đợi nhập (input queue).

Thao tác thông thường là lựa chọn một trong những tiến trình trong hàng đợi nhập và nạp tiến trình đó vào bộ nhớ chính. Khi một tiến trình được thực thi, nó truy xuất các chỉ thị và dữ liệu từ bộ nhớ chính (nói cách khác, CPU chỉ có thể truy xuất trực tiếp Main Memory). Khi tiến trình kết thúc, không gian bộ nhớ của nó được xác định là được giải phóng (cùng với các tài nguyên khác).

Hầu hết các hệ thống cho phép một tiến trình người dùng nằm ở bất cứ phần nào của bộ nhớ vật lý. Do đó, mặc dù không gian địa chỉ của máy tính bắt đầu tại 00000h, nhưng địa chỉ đầu tiên của tiến trình người dùng không nhất thiết phải ở tại 00000h. Sắp xếp này ảnh hưởng đến địa chỉ mà chương trình người dùng có thể sử dụng. Trong hầu hết các trường hợp, một chương trình người dùng sẽ đi qua một số bước- một vài trong chúng có thể là tùy chọn-trước khi được thực thi (hình 3.1). Các địa chỉ có thể được hiện diện trong những cách khác trong những bước này. Các địa chỉ trong chương trình nguồn thường là địa chỉ symbolic (*tượng trưng*). Trình biên dịch sẽ liên kết các địa chỉ tượng trưng tới các địa chỉ có thể tái định vị. Bộ nạp sẽ liên kết các địa chỉ có thể tái định vị tới địa chỉ tuyệt đối (chẳng hạn 74014h). Mỗi liên kết là một ánh xạ từ một không gian địa chỉ này tới một không gian địa chỉ khác.



Hình 3.1. Quá trình xử lý nhiều bước của chương trình người sử dụng

Có thể thực hiện việc liên kết (kết buộc) địa chỉ tại 1 trong 3 thời điểm sau:

- *Thời điểm biên dịch (Compile-time)*: nếu tại thời điểm biên dịch có thể xác định tiến trình nằm ở đâu trong bộ nhớ thì mã (địa chỉ) tuyệt đối có thể được phát sinh. Ví dụ, nếu biết trước tiến trình người dùng nằm tại vị trí R thì mã trình biên dịch được phát sinh sẽ bắt đầu tại vị trí đó và mở rộng từ đó. Nếu tại thời điểm sau đó, vị trí nạp thay đổi thì sẽ cần biên dịch lại chương trình. Các chương trình định dạng .COM của MS-DOS là mã tuyệt đối giới hạn tại thời điểm biên dịch.
- *Thời điểm nạp (Load-time)*: nếu tại thời điểm biên dịch chưa biết nơi tiến trình sẽ nằm ở đâu trong bộ nhớ chính thì trình biên dịch phải phát sinh mã có thể tái định vị (địa chỉ tương đối). Trong trường hợp này, liên kết cuối cùng được trì hoãn cho tới thời điểm nạp (khi nạp, biết vị trí bắt đầu sẽ tính lại địa chỉ tuyệt đối). Nếu địa chỉ nạp thay đổi, chúng ta chỉ cần nạp lại mã người dùng để hợp nhất giá trị được thay đổi này.
- *Thời điểm thực thi (Execution time)*: nếu tiến trình có thể phải được di chuyển trong thời gian thực thi từ một phân đoạn bộ nhớ này tới một phân

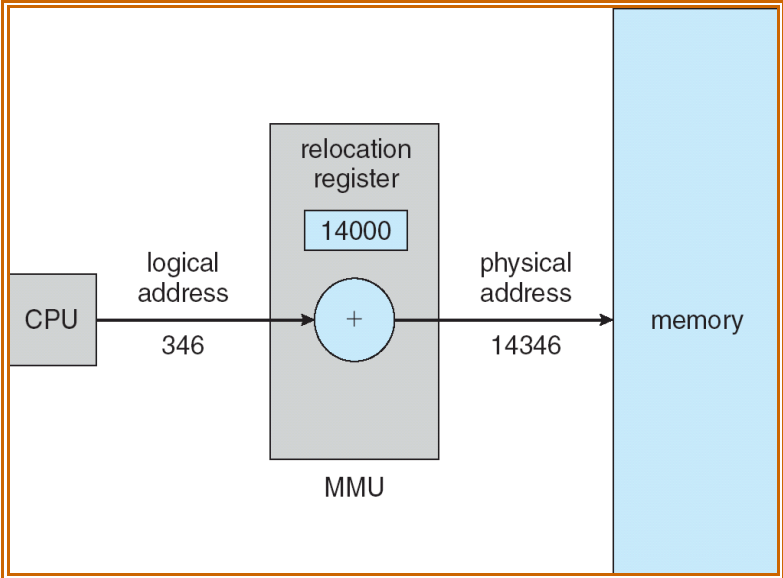
đoạn bộ nhớ khác thì việc liên kết phải bị trì hoãn cho tới thời gian thi hành (khi biên dịch, nạp chỉ phát sinh địa chỉ tương đối). Phần cứng đặc biệt phải sẵn dùng cho cơ chế này để thực hiện công việc (MMU). Hầu hết những hệ điều hành này dùng phương pháp này.

3.1.1.2 Không gian địa chỉ logic và không gian vật lý

Một địa chỉ được tạo ra bởi CPU thường được gọi là địa chỉ logic (logical address), ngược lại một địa chỉ mà trình quản lý bộ nhớ nhìn thấy và thao tác - nghĩa là, một địa chỉ được nạp vào thanh ghi địa chỉ bộ nhớ - thường được gọi là địa chỉ vật lý (physical address).

Các phương pháp liên kết địa chỉ thời điểm biên dịch và thời điểm nạp tạo ra địa chỉ logic và địa chỉ vật lý xác định. Tuy nhiên, cơ chế liên kết địa chỉ tại thời điểm thực thi dẫn đến sự khác nhau giữa địa chỉ logic và địa chỉ vật lý. Trong trường hợp này, chúng ta thường gọi địa chỉ logic như là địa chỉ ảo (virtual address). Tập hợp tất cả địa chỉ logic được tạo ra bởi chương trình là không gian địa chỉ logic; tập hợp tất cả địa chỉ vật lý tương ứng địa chỉ logic này là không gian địa chỉ vật lý. Do đó, trong cơ chế liên kết địa chỉ tại thời điểm thực thi, không gian địa chỉ logic và không gian địa chỉ vật lý là khác nhau.

Việc ánh xạ tại thời điểm thực thi từ địa chỉ ảo tới địa chỉ vật lý được thực hiện bởi một thiết bị phần cứng được gọi là đơn vị quản lý bộ nhớ MMU (memory-management unit). Chúng ta có thể lựa chọn các phương pháp khác nhau để thực hiện việc ánh xạ.



Hình 3.2 Định vị tự động sử dụng thanh ghi tái định vị

Xét ví dụ trong hình 3.2 ở trên, phương pháp này yêu cầu sự hỗ trợ phần cứng. Thanh ghi cơ sở bây giờ được gọi là thanh ghi tái định vị. Giá trị trong thanh ghi tái định vị được cộng vào mỗi địa chỉ được tạo ra bởi tiến trình người sử dụng tại thời

điểm nó được gọi tới bộ nhớ chính. Ví dụ, nếu giá trị cơ sở là 14000, một truy xuất tới địa chỉ logic 346 được ánh xạ tới vị trí 14346.

3.1.1.3 Nạp động (Dynamic Loading)

Trong các hệ điều hành trước đây, toàn bộ chương trình và dữ liệu của một tiến trình phải ở trong bộ nhớ vật lý để tiến trình thực thi. Kích thước của tiến trình bị giới hạn bởi kích thước của bộ nhớ vật lý. Để đạt được hiệu quả tốt hơn trong việc sử dụng không gian bộ nhớ chính, chúng ta có thể sử dụng phương pháp nạp động (dynamic loading). Với nạp động, một thủ tục (routine) không được nạp cho tới khi nó được gọi. Tất cả thủ tục được giữ trên đĩa trong định dạng nạp có thể tái định vị. Chương trình chính được nạp vào bộ nhớ và được thực thi. Khi một thủ tục cần gọi một thủ tục khác, thủ tục gọi trước hết kiểm tra để thấy thủ tục cần gọi đã được nạp hay chưa. Nếu chưa, bộ nạp liên kết có thể tái định vị để nạp thủ tục mong muốn vào bộ nhớ và cập nhật các bảng địa chỉ của chương trình để phản ánh sự thay đổi này. Sau đó, điều khiển này được chuyển đến thủ tục mới được nạp.

Thuận lợi của sự nạp động là ở đó một thủ tục không được dùng thì sẽ không bao giờ được nạp. Phương pháp này đặc biệt có ích khi lượng lớn mã được yêu cầu quản lý các trường hợp xảy ra không thường xuyên, chẳng hạn như các thủ tục lỗi. Trong trường hợp này, mặc dù kích thước toàn bộ chương trình có thể lớn, nhưng phần được dùng (và do đó được nạp) có thể nhỏ hơn nhiều.

Nạp động không yêu cầu hỗ trợ đặc biệt từ hệ điều hành. Nhiệm vụ của người sử dụng là thiết kế các chương trình của họ để đạt được sự thuận lợi đó. Tuy nhiên, hệ điều hành có thể giúp người lập trình bằng cách cung cấp các thủ tục thư viện để cài đặt nạp tự động.

3.1.1.4 Liên kết động

Khái niệm liên kết động là tương tự như khái niệm nạp động. Đặc điểm này thường được dùng với các thư viện hệ thống như các thư viện chương trình con của các ngôn ngữ. Với liên kết động, một **stub** là một đoạn mã hiển thị cách định vị chương trình con trong thư viện cư trú trong bộ nhớ chính hay cách nạp thư viện nếu chương trình con chưa hiện diện.

Khi stub này được thực thi, nó kiểm tra để thấy chương trình con được yêu cầu đã ở trong bộ nhớ hay chưa. Nếu chưa, chương trình này sẽ nạp chương trình con vào trong bộ nhớ. Dù là cách nào, stub thay thế chính nó với địa chỉ của chương trình con và thực thi chương trình con đó. Do đó, thời điểm tiếp theo phân đoạn mã đạt được, chương trình con trong thư viện được thực thi trực tiếp mà không gây ra bất kỳ chi phí cho việc liên kết động. Dưới cơ chế này, tất cả các tiến trình sử dụng một thư viện ngôn ngữ thực thi chỉ một bản sao của mã thư viện.

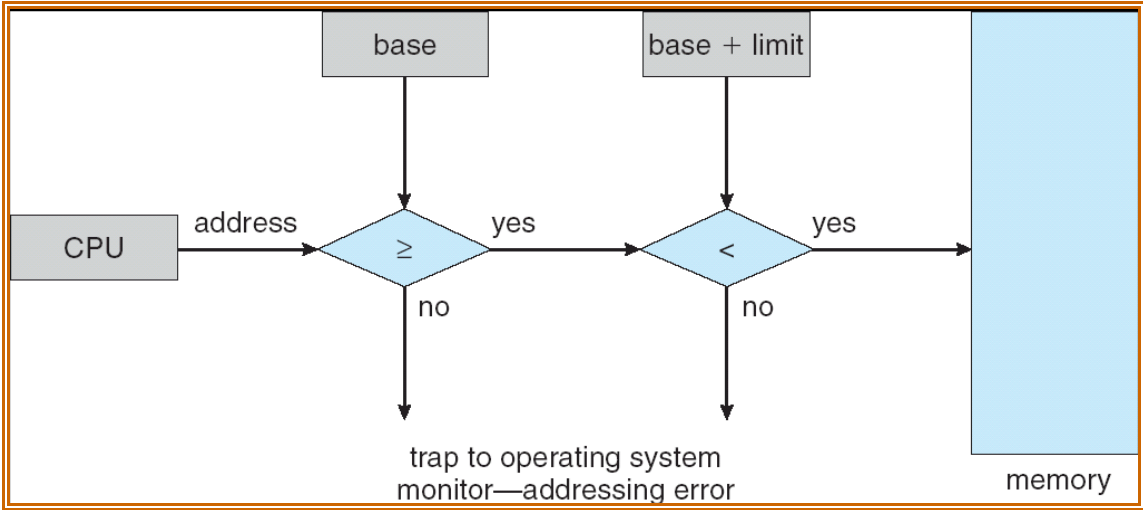
3.1.2 Các kỹ thuật cấp phát bộ nhớ

3.1.2.1 Các yêu cầu khi tổ chức lưu trữ tiến trình

a. Bảo vệ bộ nhớ (Protection)

Vấn đề bảo vệ bộ nhớ, tức là bảo vệ hệ điều hành từ tiến trình người dùng, và

bảo vệ các tiến trình từ một tiến trình khác (không cho phép tiến trình truy cập đến các vị trí nhớ đã cấp cho tiến trình khác khi chưa có phép). Tuy nhiên, không thể thực hiện việc kiểm tra hợp lệ tại thời điểm biên dịch hay nạp, vì chương trình có thể được tái định vị. Vì vậy, thực hiện kiểm tra tại thời điểm thi hành, và cần có sự hỗ trợ từ phần cứng (có thể bằng cách dùng các thanh ghi cơ sở (base) và thanh ghi giới hạn (limit), như hình 3.3).



Hình 3.3 Bảo vệ bộ nhớ với các thanh ghi base và limit

Thanh ghi base chứa giá trị địa chỉ vật lý nhỏ nhất; thanh ghi limit chứa vùng địa chỉ logic (mỗi địa chỉ logic phải nhỏ hơn thanh ghi limit). MMU ánh xạ địa chỉ (động) bằng cách cộng giá trị trong thanh ghi cơ sở. Địa chỉ được sử dụng để truy nhập bộ nhớ chính.

b. Sự tái định vị (Relocation): Trong các hệ thống đa chương, không gian bộ nhớ chính thường được chia sẻ cho nhiều tiến trình khác nhau và yêu cầu bộ nhớ của các tiến trình luôn lớn hơn không gian bộ nhớ vật lý mà hệ thống có được. Do đó, một chương trình đang hoạt động trên bộ nhớ cũng có thể bị đưa ra đĩa (swap-out) và nó sẽ được đưa vào lại (swap-in) bộ nhớ tại một thời điểm thích hợp nào đó sau này. Vấn đề đặt ra là khi đưa một chương trình vào lại bộ nhớ thì hệ điều hành phải định vị nó vào đúng vị trí mà nó đã được nạp trước đó. Để thực hiện được điều này hệ điều hành phải có các cơ chế để ghi lại tất cả các thông tin liên quan đến một chương trình bị swap-out, các thông tin này là cơ sở để hệ điều hành swap-in chương trình vào lại bộ nhớ chính và cho nó tiếp tục hoạt động. Hệ điều hành buộc phải swap-out một chương trình vì nó còn không gian bộ nhớ chính để nạp tiến trình khác, do đó sau khi swap-out một chương trình hệ điều hành phải tổ chức lại bộ nhớ để chuẩn bị nạp tiến trình vừa có yêu cầu. Các nhiệm vụ trên do bộ phần quản lý bộ nhớ của hệ điều hành thực hiện. Ngoài ra trong nhiệm vụ này hệ điều hành phải có khả năng chuyển đổi các địa chỉ bộ nhớ được ghi trong code của chương trình thành các địa chỉ vật lý thực tế trên bộ nhớ chính khi chương trình thực hiện các thao tác truy xuất trên bộ nhớ, bởi vì người lập trình không hề biết

trước hiện trạng của bộ nhớ chính và vị trí mà chương trình được nạp khi chương trình của họ hoạt động. Trong một số trường hợp khác các chương trình bị swap-out có thể được swap-in vào lại bộ nhớ tại vị trí khác với vị trí mà nó được nạp trước đó.

c. Chia sẻ bộ nhớ (Sharing): Bất kỳ một chiến lược nào được cài đặt đều phải có tính mềm dẻo để cho phép nhiều tiến trình có thể truy cập đến cùng một địa chỉ trên bộ nhớ chính. Ví dụ, khi có nhiều tiến trình cùng thực hiện một chương trình thì việc cho phép mỗi tiến trình cùng truy cập đến một bản copy của chương trình sẽ thuận lợi hơn khi cho phép mỗi tiến trình truy cập đến một bản copy sở hữu riêng. Các tiến trình đồng thực hiện (co-operating) trên một vài tác vụ có thể cần để chia sẻ truy cập đến cùng một cấu trúc dữ liệu. Hệ thống quản lý bộ nhớ phải điều khiển việc truy cập đến không gian bộ nhớ được chia sẻ mà không vi phạm đến các yêu cầu bảo vệ bộ nhớ. Ngoài ra, trong môi trường hệ điều hành đa nhiệm hệ điều hành phải chia sẻ không gian nhớ cho các tiến trình để hệ điều hành có thể nạp được nhiều tiến trình vào bộ nhớ để các tiến trình này có thể hoạt động đồng thời với nhau.

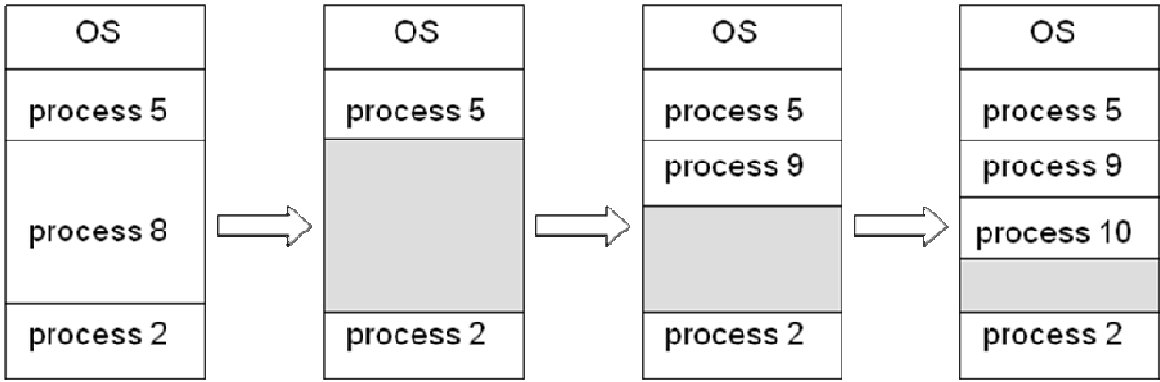
3.1.2.2 Cấp phát bộ nhớ liên tục (Contiguous Allocation)

Bộ nhớ chính thường được phân chia thành 2 phân vùng cung cấp cho cả hệ điều hành (vùng bộ nhớ thấp như bảng vector ngắt, ...) và các tiến trình người sử dụng (vùng nhớ cao hơn). Do đó, chúng ta cần cấp phát các vùng khác nhau của bộ nhớ chính theo các cách hiệu quả nhất có thể. Phần này chúng ta phân tích một phương pháp thông dụng, đó là cấp phát bộ nhớ liên tục.

Một trong những yêu cầu đặt ra là có càng nhiều tiến trình người sử dụng được định vị trong bộ nhớ chính tại cùng thời điểm càng tốt (hỗ trợ Multiprogramming). Do đó, chúng ta cần xem xét cách cấp phát bộ nhớ trống tới những tiến trình ở trong hàng đợi nhập đang chờ được đưa vào bộ nhớ chính. Trong cấp phát bộ nhớ liên tục, mỗi tiến trình được lưu giữ trong một vùng bộ nhớ liên tục. Vì vậy, phải cần một vùng nhớ liên tục, đủ lớn để chứa chương trình.

a. Kỹ thuật phân vùng cố định (Fixed Partitioning)

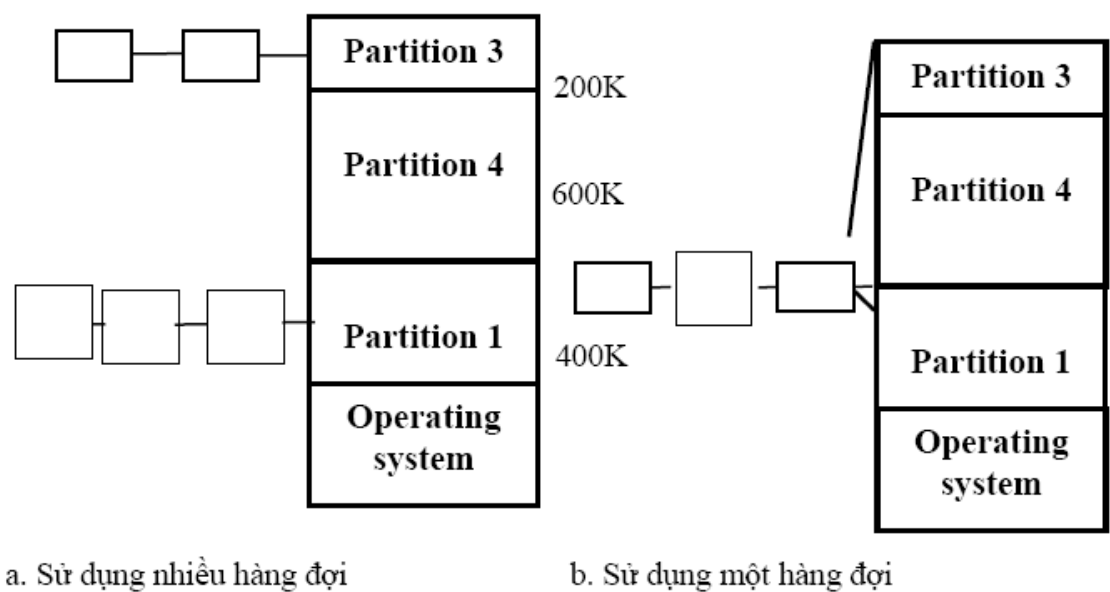
Một trong những phương pháp đơn giản nhất để cấp phát bộ nhớ là chia bộ nhớ thành những phân vùng có kích thước cố định. Mỗi phân vùng có thể chứa chính xác một tiến trình. Do đó, cấp độ đa chương được giới hạn bởi số lượng phân vùng (hình 3.4).



Hình 3.4. Cấp phát phân vùng kích thước cố định

Trong phương pháp đa phân vùng, khi một phân vùng rảnh, một tiến trình được chọn từ hàng đợi nhập và được nạp vào phân vùng trống. Khi tiến trình kết thúc, phân vùng trở nên sẵn dùng cho một tiến trình khác. Có hai tiếp cận để tổ chức hàng đợi:

- Sử dụng nhiều hàng đợi: mỗi phân vùng sẽ có một hàng đợi tương ứng (hình 3.5a). Khi một tiến trình mới được tạo ra, nó được đưa vào hàng đợi của phân vùng có kích thước nhỏ nhất thoả nhu cầu chứa nó. Cách tổ chức này có khuyết điểm: trong trường hợp các hàng đợi của một số phân vùng trống trong khi các hàng đợi của các phân vùng khác lại đầy, buộc các tiến trình trong những hàng đợi này phải chờ được cấp phát bộ nhớ.
- Sử dụng một hàng đợi: tất cả các tiến trình được đặt trong một hàng đợi duy nhất (hình 3.5b). Khi có một phân vùng trống, tiến trình đầu tiên trong hàng đợi có kích thước phù hợp sẽ được đặt vào phân vùng và cho thực hiện. Cần dùng 1 cấu trúc dữ liệu để theo dõi các partition tự do.

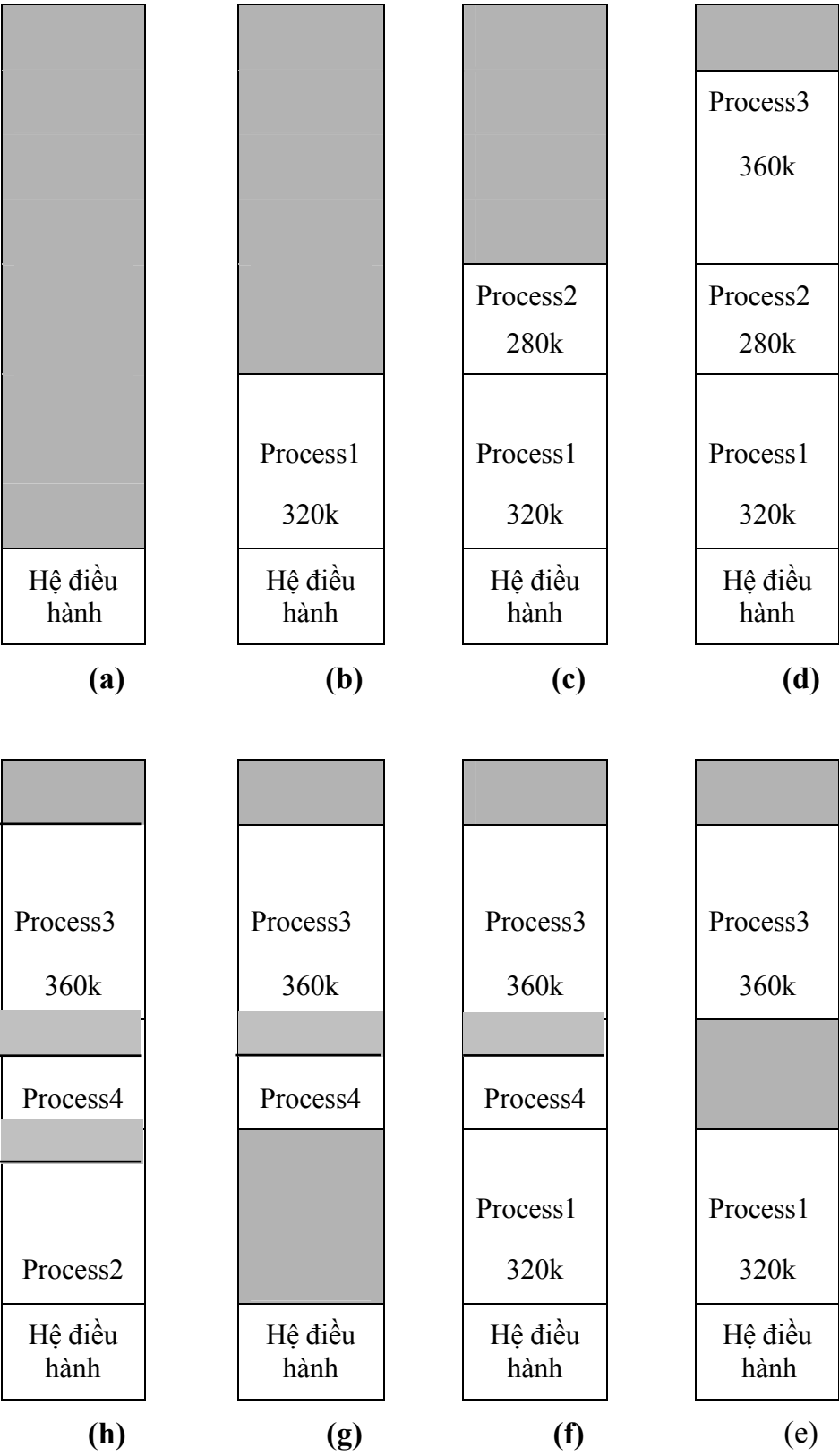


Hình 3.5. Cấp phát phân vùng có kích thước cố định sử dụng hàng đợi

b. Kỹ thuật phân vùng động (Dynamic Partitioning)

Trong kỹ thuật phân vùng động, bộ nhớ chính không được phân chia trước, các phân vùng trên bộ nhớ có kích thước tùy ý, sẽ hình thành trong quá trình nạp các tiến trình vào hệ thống. Hệ điều hành giữ một bảng biểu thị những phân vùng của bộ nhớ là sẵn dùng và phân cùng nào đang bận. Ban đầu, tất cả bộ nhớ là sẵn dùng cho tiến trình người dùng, và được xem như một khối lớn bộ nhớ sẵn dùng. Khi một tiến trình đến và yêu cầu bộ nhớ, hệ điều hành sẽ tìm kiếm một phân cùng trống đủ lớn cho tiến trình này. Nếu tìm thấy, hệ điều hành cấp cho nó không gian vừa đủ để chứa tiến trình, phần còn lại để sẵn sàng cấp cho tiến trình khác sau này.

Khi các tiến trình đi vào hệ thống, chúng được đặt vào hàng đợi nhập. Hệ điều hành xem xét yêu cầu bộ nhớ của mỗi tiến trình và lượng không gian bộ nhớ sẵn có để xác định các tiến trình nào được cấp phát bộ nhớ. Khi một tiến trình được cấp không gian, nó được nạp vào bộ nhớ và sau đó nó có thể cạnh tranh CPU. Khi một tiến trình kết thúc, nó giải phóng bộ nhớ của nó, sau đó hệ điều hành có thể cấp cho tiến trình khác từ hàng đợi nhập, ngay cả khi tiến trình này có kích thước nhỏ hơn kích thước của không gian nhớ trống đó (hình 3.6).



Hình 3.6: Kết quả của sự phân trang động với thứ tự nạp các tiến trình.

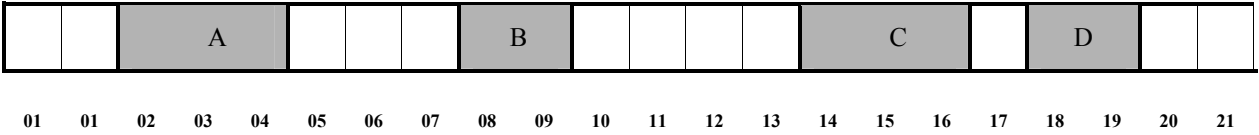
Hình 3.6 trên đây minh họa cho quá trình nạp/kết thúc các tiến trình theo thứ tự: nạp process1, nạp process2, nạp process3, kết thúc process2, nạp process4, kết thúc process1, nạp process2 vào lại, trong hệ thống phân vùng động. Như vậy dần dần

trong bộ nhớ hình thành nhiều không gian nhớ có kích thước nhỏ không đủ chứa các tiến trình nằm rải rác trên bộ nhớ chính, hiện tượng này được gọi là hiện tượng phân mảnh bên ngoài (external fragmentation). Để chống lại sự lãng phí bộ nhớ do phân mảnh, thỉnh thoảng hệ điều hành phải thực hiện việc sắp xếp lại bộ nhớ, để các không gian nhớ nhỏ rời rạc nằm liền kề lại với nhau tạo thành một khối nhớ có kích thước đủ lớn để chứa được một tiến trình nào đó. Việc làm này làm chậm tốc độ của hệ thống, hệ điều hành phải chi phí cao cho việc này, đặc biệt là việc tái định vị các tiến trình khi một tiến trình bị đưa ra khỏi bộ nhớ và được nạp vào lại bộ nhớ để tiếp tục hoạt động.

Tại bất cứ thời điểm nào, chúng ta có một danh sách kích thước khối nhớ trống và hàng đợi nhập. Hệ điều hành có thể xếp hàng đợi nhập dựa theo giải thuật định thời. Bộ nhớ được cấp phát tới các tiến trình cho đến khi các yêu cầu bộ nhớ của tiến trình kế tiếp không thể được thỏa mãn; tức là không có khối bộ nhớ trống (hole) đủ lớn để quản lý tiến trình đó. Khi đó, hệ điều hành có thể chờ cho đến khi có khối nhớ đủ lớn sẵn dùng hay nó có thể tìm tiếp (di chuyển xuống) trong hàng đợi nhập để xem có các yêu cầu bộ nhớ nhỏ hơn của các tiến trình khác mà có thể được thỏa hay không.

Thông thường, một tập hợp các khối có kích thước khác nhau được phân tán khắp bộ nhớ tại bất cứ thời điểm đang xét. Khi một tiến trình đến và yêu cầu bộ nhớ, hệ thống tìm trong tập hợp này một khối nhớ trống đủ lớn cho tiến trình này. Nếu khối nhớ trống quá lớn, nó có thể được chia thành hai phần: một phần được cấp cho tiến trình đến; phần còn lại được trả về lại trong tập hợp các khối nhớ (trống). Nếu khối nhớ mới nằm kề với các khối khác, các khối nằm kề này có thể được kết hợp lại để tạo thành một khối nhớ lớn hơn. Tại thời điểm này, hệ thống cần kiểm tra xem có tiến trình nào đang chờ để được cấp phát bộ nhớ (mà chưa được đáp ứng) và bộ nhớ trống mới hay bộ nhớ vừa được kết hợp lại có thể thỏa yêu cầu của bất kỳ tiến trình đang chờ này không.

Trong kỹ thuật phân vùng động này hệ điều hành phải đưa ra các cơ chế thích hợp để quản lý các khối nhớ đã cấp phát hay còn trống trên bộ nhớ. Hệ điều hành sử dụng 2 giải pháp chủ yếu cho vấn đề này là: Bản đồ bit và Danh sách liên kết. Trong cả 2 giải pháp này, hệ điều hành đều chia không gian nhớ thành các đơn vị cấp phát có kích thước bằng nhau, các đơn vị cấp phát liên tiếp nhau tạo thành một khối nhớ (block), hệ điều hành cấp phát các block này cho các tiến trình khi nạp tiến trình vào bộ nhớ.

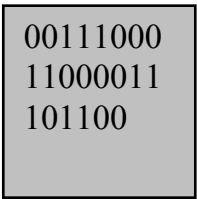


Hình 3.7a: Một đoạn nhớ bao gồm 22 đơn vị cấp phát, tạo thành 9 block, trong đó có 4 block đã cấp phát (tô đậm, kí hiệu là P) cho các tiến trình: A, B, C, D và 5 block chưa được cấp phát (đề trắng, kí hiệu là H).

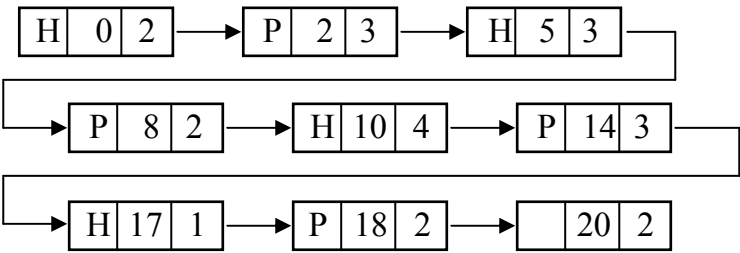
- **Quản lý bằng bản đồ bit:** mỗi đơn vị cấp phát được ánh xạ tới một bit trong bản đồ bit (hình 3.7a). Giá trị bit này xác định trạng thái của đơn vị bộ

nhớ đó: 0- đang tự do, 1 đã được cấp phát. Khi cần nạp một tiến trình có kích thước k đơn vị, hệ thống sẽ tìm trong bản đồ bit một dãy k bit có giá trị 0 (hình 3.7b). Kích thước của đơn vị cấp phát cũng là một vấn đề lớn trong thiết kế. Nếu kích thước đơn vị cấp phát nhỏ sẽ làm tăng kích thước của bản đồ bit. Ngược lại, nếu kích thước đơn vị cấp phát lớn có thể gây hao phí cho đơn vị cấp phát sau cùng. Đây là giải pháp đơn giản nhưng thực hiện chậm nên ít được sử dụng.

- **Quản lý bằng danh sách liên kết:** dùng một danh sách liên kết để quản lý các phân đoạn bộ nhớ đã cấp phát và phân đoạn tự do, một phân đoạn có thể là một tiến trình hay một vùng nhớ trống giữa hai tiến trình. Danh sách liên kết gồm nhiều phần tử liên tiếp. Mỗi phần tử gồm 1 bit đầu để xác định phân đoạn đó là khối trống (H) hay tiến trình (P), trường thứ hai cho biết thứ tự của đơn vị cấp phát đầu tiên trong block, trường thứ ba cho biết block gồm bao nhiêu đơn vị cấp phát. Hình 3.7c là danh sách liên kết của khối nhớ ở trên. Việc sắp xếp các phân đoạn theo địa chỉ hay theo kích thước tùy thuộc vào giải thuật quản lý bộ nhớ.



Hình 3.7b: quản lý các đơn vị cấp phát bằng bản đồ bit.



Hình 3.7c: quản lý các đơn vị cấp phát bằng danh sách liên kết.

Như vậy khi cần nạp một tiến trình vào bộ nhớ thì hệ điều hành phải dựa vào bản đồ bit hoặc danh sách liên kết để tìm ra một block có kích thước đủ để nạp tiến trình. Sau khi thực hiện một thao tác cấp phát hoặc sau khi đưa một tiến trình ra khỏi bộ nhớ thì hệ điều hành phải cập nhật lại bản đồ bit hoặc danh sách liên kết, điều này có thể làm giảm tốc độ thực hiện của hệ thống.

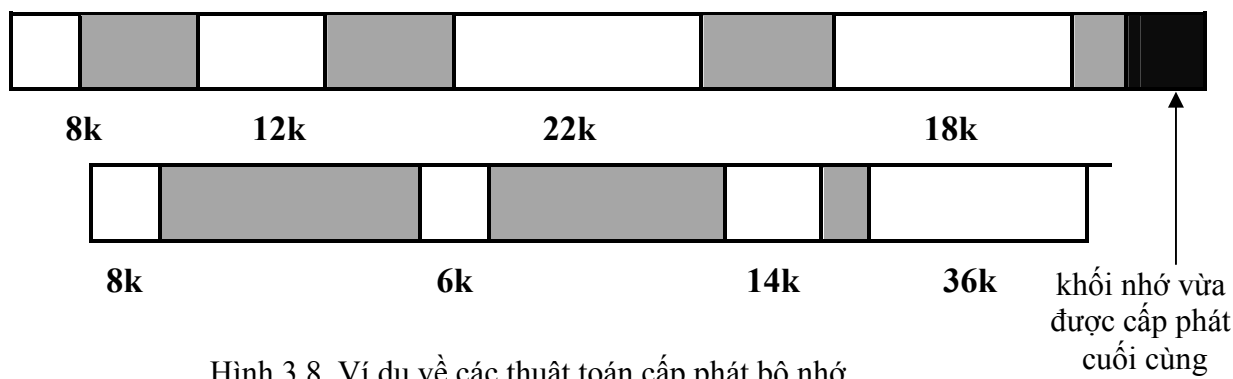
Danh sách liên kết có thể được sắp xếp theo thứ tự tăng dần hoặc giảm dần của kích thước hoặc địa chỉ, điều này giúp cho việc tìm khối nhớ trống có kích thước vừa đủ để nạp các tiến trình theo các thuật toán dưới đây sẽ đạt tốc độ nhanh hơn và hiệu quả cao hơn. Một số hệ điều hành tổ chức 2 danh sách liên kết riêng để theo dõi các đơn vị cấp phát trên bộ nhớ, một danh sách để theo dõi các block đã cấp phát và một danh sách để theo dõi các block còn trống. Cách này giúp việc tìm các khối nhớ trống nhanh hơn, chỉ tìm trên danh sách các khối nhớ trống, nhưng tốn thời gian nhiều hơn cho việc cập nhật danh sách sau mỗi thao tác cấp phát, vì phải thực hiện trên cả hai danh sách.

Khi có một tiến trình cần được nạp vào bộ nhớ mà trong bộ nhớ có nhiều hơn một khối nhớ trống có kích thước lớn hơn kích thước của tiến trình đó, thì hệ điều

hành phải quyết định chọn một khối nhớ trống phù hợp nào để nạp tiến trình sao cho việc lựa chọn này dẫn đến việc sử dụng bộ nhớ chính là hiệu quả nhất. Có 3 chiến lược mà hệ điều hành sử dụng trong trường hợp này, đó là: Best-fit, First-fit, và Next-fit. Cả 3 chiến lược này đều phải chọn một khối nhớ trống có kích thước bằng hoặc lớn hơn kích thước của tiến trình cần nạp vào, nhưng nó có các điểm khác nhau cơ bản sau đây:

- **Best-fit:** chọn khối nhớ tự do có kích thước nhỏ nhất đủ chứa tiến trình cần được nạp vào bộ nhớ.
- **First-fit:** trong trường hợp này hệ điều hành sẽ bắt đầu quét qua các khối nhớ trống bắt đầu từ khối nhớ trống đầu tiên trong bộ nhớ, và sẽ chọn khối nhớ trống đầu tiên có kích thước đủ lớn để nạp tiến trình.
- **Worst-Fit:** chọn partition tự do lớn nhất đủ chứa chương trình.

Các kết quả mô phỏng chứng tỏ rằng cả first-fit và best-fit là tốt hơn worst-fit về việc giảm thời gian và hiệu suất lưu trữ. Tuy nhiên, giữa first-fit và best-fit không thể xác định rõ chiến lược nào tốt hơn về hiệu suất lưu trữ, nhưng first-fit có tốc độ nhanh hơn.



Hình 3.8. Ví dụ về các thuật toán cấp phát bộ nhớ

Hình 3.8 cho thấy hiện tại trên bộ nhớ có các khối nhớ chưa được cấp phát theo thứ tự là: 8k, 12k, 22k, 18k, 8k, 6k, 14k, 36k. Trong trường hợp này nếu có một tiến trình có kích thước 16k cần được nạp vào bộ nhớ, thì hệ điều hành sẽ nạp nó vào:

- khối nhớ 22k nếu theo thuật toán First-fit
- khối nhớ 18k nếu theo thuật toán Best-fit
- khối nhớ 36k nếu theo thuật toán Worst-fit

Các hệ điều hành không cài đặt cố định trước một thuật toán nào, tùy vào trường hợp cụ thể mà nó chọn cấp phát theo một thuật toán nào đó, sao cho chi phí về việc cấp phát là thấp nhất và hạn chế được sự phân mảnh bộ nhớ sau này. Việc chọn thuật toán này thường phụ thuộc vào thứ tự swap và kích thước của tiến trình. Thuật toán First-fit được đánh giá là đơn giản, dễ cài đặt nhưng mang lại hiệu quả

cao nhất đặc biệt là về tốc độ cấp phát. Về hiệu quả thuật toán Next-fit không bằng First-fit, nhưng nó thường xuyên sử dụng được các khối nhớ trống ở cuối vùng nhớ, các khối nhớ ở vùng này thường có kích thước lớn nên có thể hạn chế được sự phân mảnh, theo ví dụ trên thì việc xuất hiện một khối nhớ trống 20k sau khi cấp một tiến trình 16k thì không thể gọi là phân mảnh được, nhưng nếu tiếp tục như thế thì dễ dẫn đến sự phân mảnh lớn ở cuối bộ nhớ. Thuật toán Best-fit, không như tên gọi của nó, đây là một thuật toán có hiệu suất thấp nhất, trong trường hợp này hệ điều hành phải duyệt qua tất cả các khối nhớ trống để tìm ra một khối nhớ có kích thước vừa đủ để chứa tiến trình vừa yêu cầu, điều này làm giảm tốc độ cấp phát của hệ điều hành. Mặt khác với việc chọn kích thước vừa đủ có thể dẫn đến sự phân mảnh lớn trên bộ nhớ, tức là có quá nhiều khối nhớ có kích thước quá nhỏ trên bộ nhớ, nhưng nếu xét về mặt lãng phí bộ nhớ tại thời điểm cấp phát thì thuật toán này làm lãng phí ít nhất. Tóm lại, khó có thể đánh giá về hiệu quả sử dụng của các thuật toán này, vì hiệu quả của nó được xét trong “tương lai” và trên nhiều khía cạnh khác nhau chứ không phải chỉ xét tại thời điểm cấp phát. Và hơn nữa trong bản thân các thuật toán này đã có các mâu thuẫn với nhau về hiệu quả sử dụng của nó.

3.1.2.3 Cấp phát bộ nhớ không liên tục

Trong cấp phát bộ nhớ không liên tục, các tiến trình được nạp vào bộ nhớ chính ở nhiều vùng nhớ không liên tục. Khi đó, không gian địa chỉ (KGDC) được phân chia thành nhiều partition (Segmentation, Paging) và không gian vật lý (KGVV) có thể được tổ chức như sau:

Các phân vùng cố định (Fixed partition): Paging

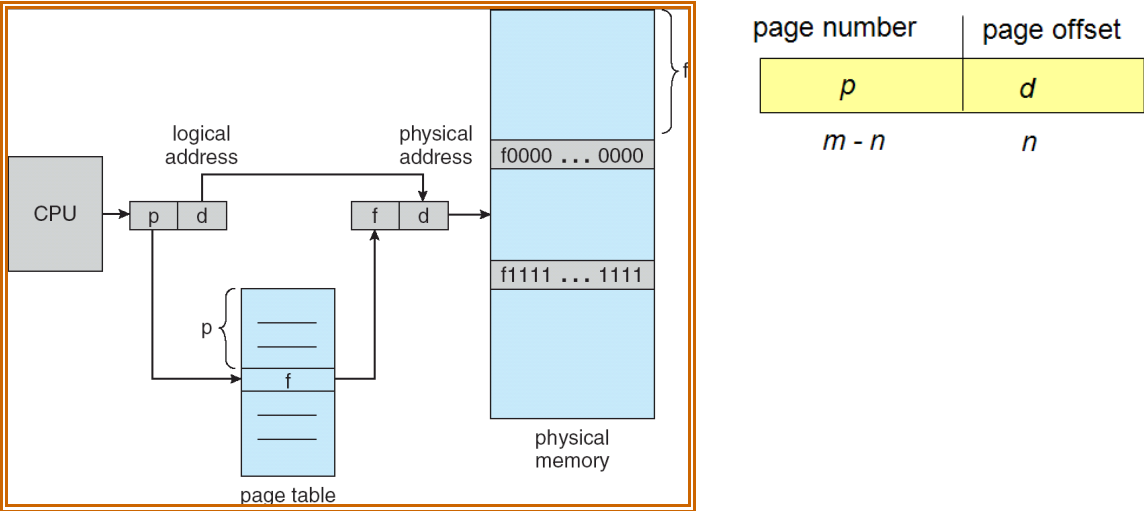
Các phân vùng có kích thước thay đổi (Variable partition: Segmentation)

a. Kỹ thuật phân trang (Paging)

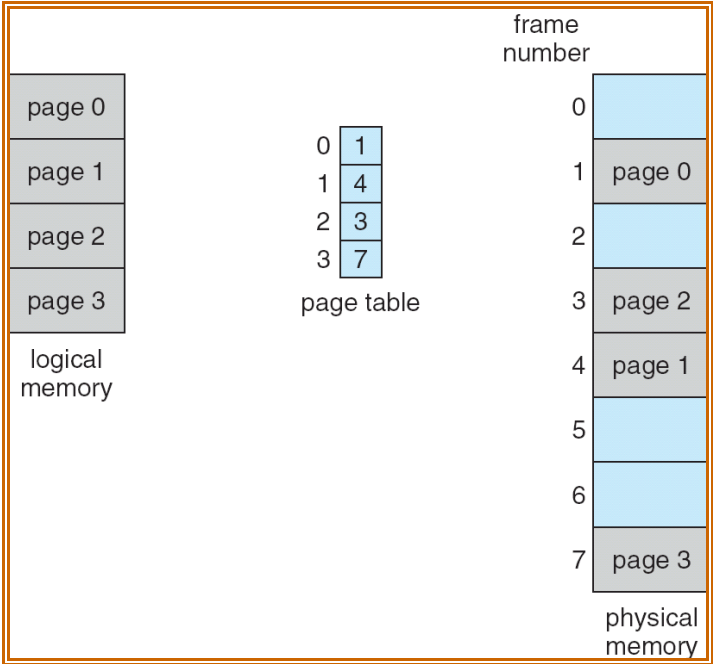
Phân trang là cơ chế quản lý bộ nhớ cho phép không gian địa chỉ vật lý của tiến trình là không liên tiếp nhau, có kích thước cố định bằng nhau, được đánh số địa chỉ bắt đầu từ 0 và được gọi là các khung trang (page frame). Không gian địa chỉ của các tiến trình cũng được chia thành các phần có kích thước bằng nhau và bằng kích thước của một khung trang, được gọi là các trang (page) của tiến trình. Khi một tiến trình được thực thi, các trang của nó được nạp vào các khung trang còn trống (có thể không liên tiếp nhau) từ bộ nhớ phụ. Khi đó, tại bộ nhớ phụ, cũng được chia thành các khối có kích thước cố định và có cùng kích thước như các khung trang. Nếu kích thước của tiến trình không phải là bội số của kích thước một khung trang thì sẽ xảy ra hiện tượng phân mảnh nội vi ở khung trang chứa trang cuối cùng của tiến trình. Ở đây không xảy ra hiện tượng phân mảnh ngoại vi. Trên bộ nhớ có thể tồn tại các trang của nhiều tiến trình khác nhau. Khi một tiến trình bị swap-out thì các khung trang mà tiến trình này chiếm giữ sẽ được giải phóng để hệ điều hành có thể nạp các trang tiến trình khác.

Hỗ trợ phần cứng cho kỹ thuật phân trang được hiển thị trong hình 3.9. Mỗi địa chỉ được tạo ra bởi CPU được chia thành hai phần: Số hiệu trang - page number (p) và địa chỉ tương đối trong trang - page offset (d). Page number được dùng như chỉ mục vào bảng trang (page table). Bảng trang chứa địa chỉ cơ sở của mỗi khung trang trong bộ nhớ vật lý. Địa chỉ cơ sở này sẽ được kết hợp với giá trị page offset

để xác định địa chỉ bộ nhớ vật lý mà nó được gởi đến đơn vị bộ nhớ. Mô hình phân trang bộ nhớ được hiển thị như hình 3.10.

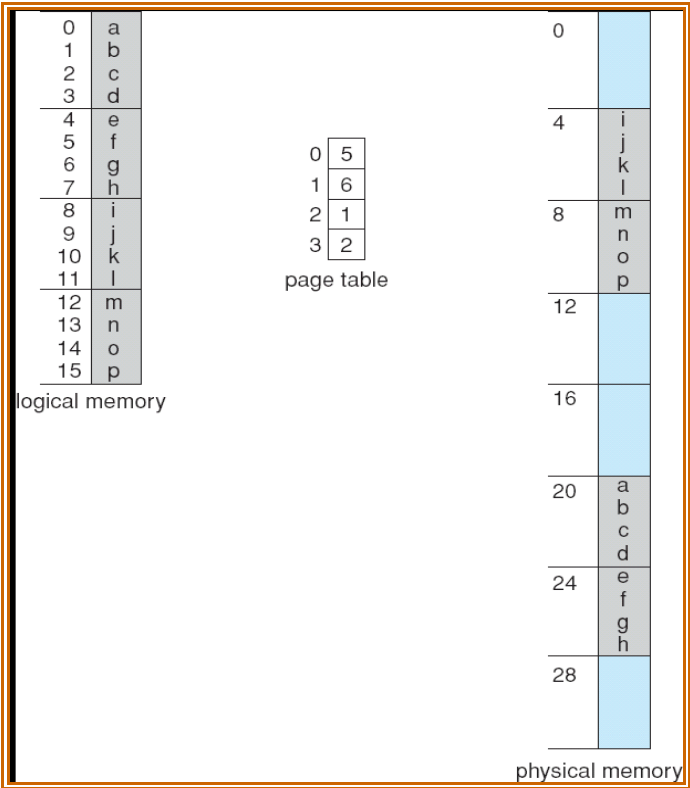


Hình 3.9. Kỹ thuật phân trang



Hình 3.10 Mô hình phân trang bộ nhớ của bộ nhớ logic và bộ nhớ vật lý

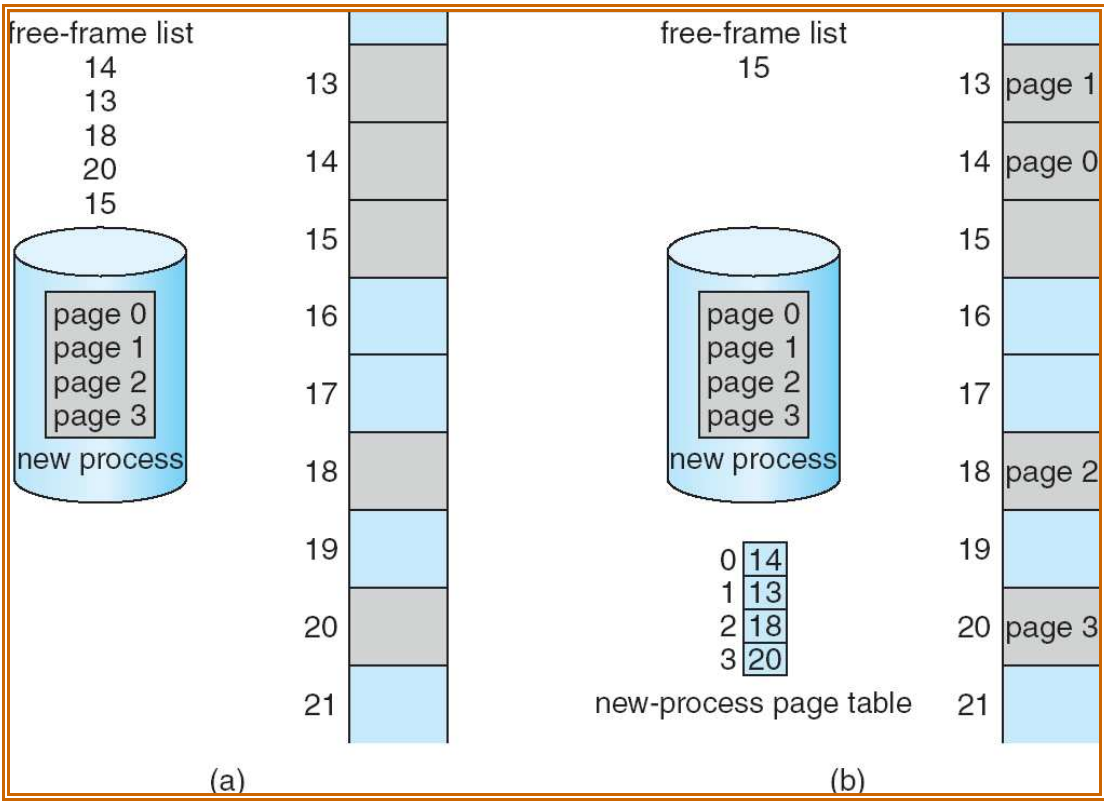
Xét một ví dụ trong hình 3.11, trong đó sử dụng kích thước trang 4 bytes và bộ nhớ vật lý 32 bytes (có 8 trang). Địa chỉ logic 0 là trang 0, địa chỉ tương đối 0. Chỉ mục trong bảng trang, chúng ta thấy rằng trang 0 ở trong khung trang 5. Do đó, địa chỉ logic 0 ánh xạ tới địa chỉ vật lý 20 $(= (5 \times 4) + 0)$. Địa chỉ logic 3 (trang 0, tương đối 3) ánh xạ tới địa chỉ vật lý 23 $(= (5 \times 4) + 3)$. Địa chỉ logic 4 ở trang 1, tương đối 0; dựa theo bảng trang, trang 1 được ánh xạ tới khung trang 6. Do đó, địa chỉ logic 4 ánh xạ tới địa chỉ 24 $(= (6 \times 4) + 0)$. Địa chỉ logic 13 ánh xạ tới địa chỉ vật lý 9.



Hình 3.11 Mô hình phân trang với bộ nhớ 32 bytes, mỗi trang có kích thước 4 bytes.

Khi sử dụng một cơ chế phân trang, không có sự phân mảnh ngoại vi. Tuy nhiên, vẫn có thể xuất hiện phân mảnh nội vi. Ví dụ, nếu các trang là 2048 bytes, một tiến trình 72,766 bytes sẽ cần 35 trang cộng với 1086 bytes. Nó được cấp phát 36 khung, do đó sẽ có phân mảnh nội vi là $2048 - 1086 = 962$ bytes. Trong trường hợp tồi nhất, một tiến trình cần n trang cộng với 1 byte, nó sẽ được cấp phát $n+1$ khung trang, dẫn đến sự phân mảnh nội vi gần như toàn bộ khung trang.

Khi một tiến trình đi vào hệ thống để thực thi, kích thước của nó, được diễn tả trong các trang, sẽ được xem xét. Mỗi trang của tiến trình cần được nạp trên một khung trang. Do đó, nếu tiến trình yêu cầu n trang, ít nhất phải có n khung phải sẵn dùng trong bộ nhớ. Nếu có n khung là sẵn dùng, chúng được cấp phát tới tiến trình này. Trang đầu tiên của tiến trình được nạp vào một trong những khung trang được cấp phát, và số khung trang tương ứng được đặt vào trong bảng trang của tiến trình này. Trang kế tiếp được nạp vào một khung trang khác, và số khung của nó được đặt vào trong bảng trang, ... (hình 3.12).



Hình 3.12. (a) Trước khi cấp phát ; (b) Sau khi cấp phát

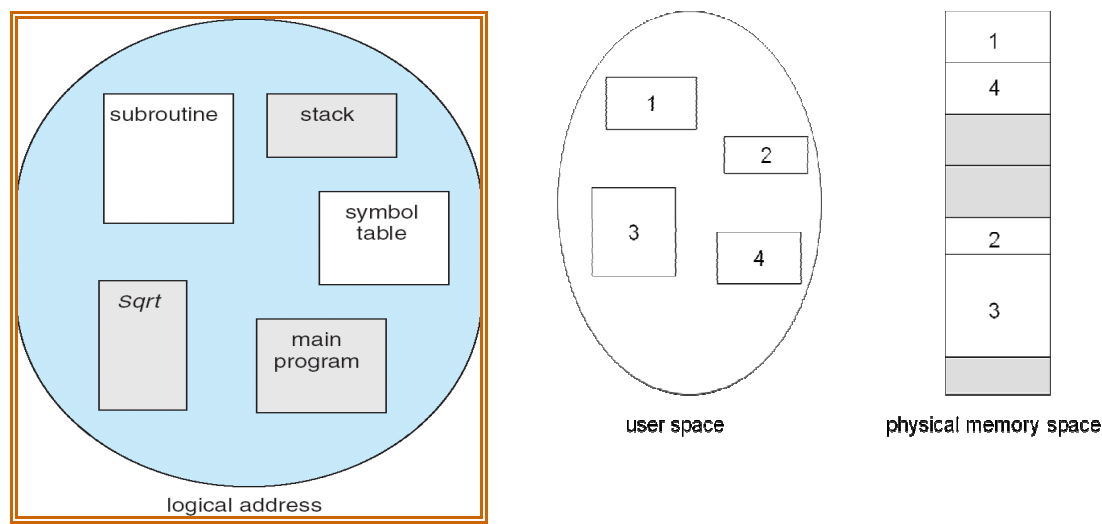
Một khía cạnh quan trọng của sự phân trang là sự phân chia rõ ràng giữa tầm nhìn bộ nhớ của người dùng và bộ nhớ vật lý thật sự. Chương trình người dùng nhìn bộ nhớ như một không gian liên tục, chứa chỉ một chương trình. Thực tế, chương trình người dùng được phân bố khắp bộ nhớ vật lý (không liên tục) cùng các tiến trình khác. Vấn đề tương thích giữa tầm nhìn bộ nhớ của người dùng và bộ nhớ vật lý thật sự được thực hiện bởi phần cứng chuyển đổi địa chỉ. Địa chỉ logic được chuyển đổi thành địa chỉ vật lý. Ánh xạ này được che giấu đối với chương trình người sử dụng và được điều khiển bởi hệ điều hành. Chú ý rằng như định nghĩa, tiến trình người dùng không thể truy xuất vùng bộ nhớ mà nó không sở hữu. Không có cách định địa chỉ bộ nhớ bên ngoài bảng trang của nó và bảng chứa chỉ những trang mà tiến trình sở hữu.

Trong kỹ thuật này hệ điều hành phải đưa ra các cơ chế thích hợp để theo dõi trạng thái của các khung trang (còn trống hay đã cấp phát) trên bộ nhớ và các khung trang đang chứa các trang của một tiến trình của các tiến trình khác nhau trên bộ nhớ. Hệ điều hành sử dụng một danh sách để ghi số hiệu của các khung trang còn trống trên bộ nhớ, hệ điều hành dựa vào danh sách này để tìm các khung trang trống trước khi quyết định nạp một tiến trình vào bộ nhớ, danh sách này được cập nhật ngay sau khi hệ điều hành nạp một tiến trình vào bộ nhớ, được kết thúc hoặc bị swap out ra bên ngoài.

Các vấn đề khác liên quan đến cơ chế phân trang (như cài đặt bảng trang, cơ chế bảo vệ trang, ...) sẽ được đề cập trong phần tiếp theo của chương này, là phần quản lý bộ nhớ ảo.

b. Kỹ thuật phân đoạn (Segmentation)

Phân đoạn là một cơ chế quản lý bộ nhớ hỗ trợ tầm nhìn bộ nhớ của người dùng. Không gian địa chỉ logic là tập hợp các phân đoạn. Mỗi phân đoạn bao gồm số hiệu phân đoạn và kích thước của nó. Không gian địa chỉ của các tiến trình kể cả các dữ liệu liên quan cũng được chia thành các đoạn khác nhau và không nhất thiết phải có kích thước bằng nhau, thông thường mỗi thành phần của một chương trình/tiến trình như: main program, procedure, function, method, object, local variables, global variables, common block, stack, symbol table, arrays, ..., là một đoạn (đơn vị logic).



Hình 3.13. Tầm nhìn chương trình người dùng

Để đơn giản việc cài đặt, các phân đoạn được đánh số và được tham chiếu tới bởi số hiệu phân đoạn. Do đó, địa chỉ logic thường là một bộ hai giá trị:

<số hiệu phân đoạn, offset>

Thông thường, chương trình người dùng được biên dịch, và trình biên dịch tự động tạo ra các phân đoạn phản ánh chương trình nhập. Chẳng hạn, một chương trình Pascal có thể tạo các phân đoạn riêng như sau:

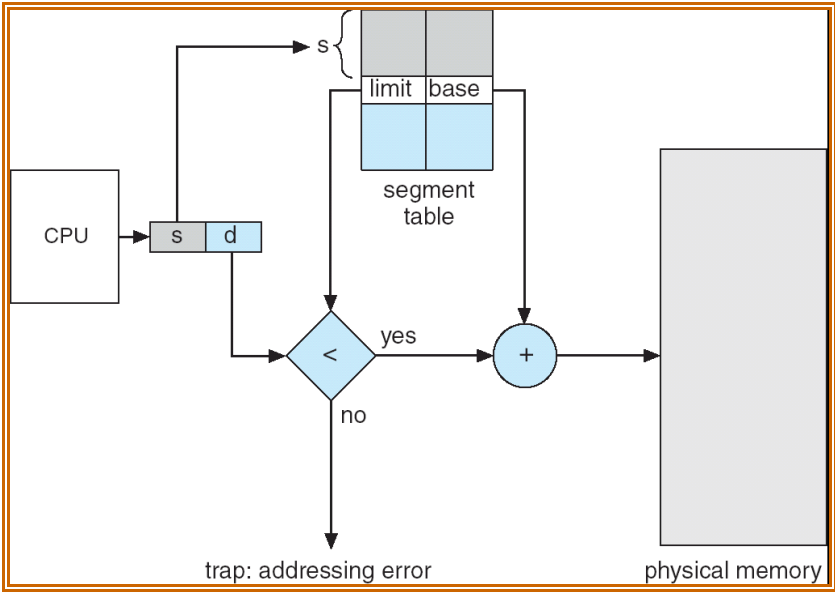
- 1) Các biến toàn cục;
- 2) Ngăn xếp gọi thủ tục, để lưu trữ các tham số và trả về các địa chỉ;
- 3) Phần mã của mỗi thủ tục hay hàm;
- 4) Các biến cục bộ của mỗi thủ tục và hàm

Một trình biên dịch có thể tạo một phân đoạn riêng cho mỗi khối chung. Các mảng có thể được gán các phân đoạn riêng. Bộ nạp có thể mang tất cả phân đoạn này và gán chúng số hiệu phân đoạn.

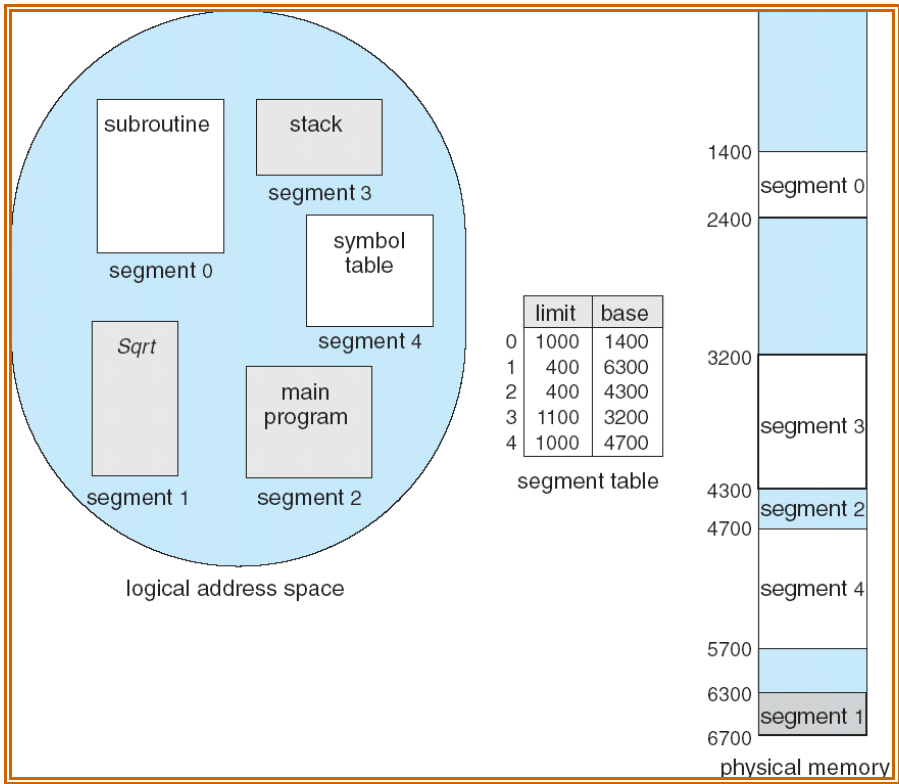
Khi một tiến trình được nạp vào bộ nhớ thì tất cả các đoạn của nó sẽ được nạp vào các phân đoạn còn trống khác nhau trên bộ nhớ. Các phân đoạn này có thể không liên tiếp nhau.

Để theo dõi các đoạn của các tiến trình khác nhau trên bộ nhớ, hệ điều hành sử dụng các bảng phân đoạn (Segment Table) tiến trình, thông thường một tiến trình có một bảng phân đoạn riêng. Mỗi phần tử trong bảng phân đoạn gồm tối thiểu 2 trường: trường thứ nhất cho biết địa chỉ cơ sở (base) của phân đoạn mà đoạn chương trình tương ứng được nạp, trường thứ hai cho biết độ dài/giới hạn (length/limit) của phân đoạn, trường này còn có tác dụng dùng để kiểm soát sự truy xuất bất hợp lệ của các tiến trình. Các bảng phân đoạn có thể được chứa trong các thanh ghi nếu có kích thước nhỏ, nếu kích thước bảng phân đoạn lớn thì nó được chứa trong bộ nhớ chính, khi đó hệ điều hành sẽ dùng một thanh ghi để lưu trữ địa chỉ bắt đầu nơi lưu trữ bảng phân đoạn, thanh ghi này được gọi là thanh ghi STBR: Segment table base register. Ngoài ra vì số lượng các đoạn của một chương trình/tiến trình có thể thay đổi nên hệ điều hành dùng thêm thanh ghi STLR:Segment table length register, để ghi kích thước hiện tại của bảng phân đoạn. Hệ điều hành cũng tổ chức một danh sách riêng để theo dõi các segment còn trống trên bộ nhớ (hình 3.14).

Xét một ví dụ trong trường hợp ở hình 3.15. Chúng ta có năm phân đoạn được đánh số từ 0 đến 4. Các phân đoạn được lưu trong bộ nhớ vật lý như được hiển thị. Bảng phân đoạn có một mục từ riêng cho mỗi phân đoạn, cho địa chỉ bắt đầu của phân đoạn trong bộ nhớ vật lý (hay nền) và chiều dài của phân đoạn đó (hay giới hạn). Thí dụ, phân đoạn 2 dài 400 bytes và bắt đầu tại vị trí 4300. Do đó, một tham chiếu byte 53 của phân đoạn 2 được ánh xạ tới vị trí $4300 + 53 = 4353$. Một tham chiếu tới phân đoạn 3, byte 852, được ánh xạ tới 3200 (giá trị nền của phân đoạn 3) $+852=4052$. Một tham chiếu tới byte 1222 của phân đoạn 0 dẫn đến một trap tới hệ điều hành, khi phân đoạn này chỉ dài 1000 bytes.



Hình 3.14. Phần cứng phân đoạn



Hình 3.15. Một ví dụ về sự phân đoạn

c. Kỹ thuật phân đoạn kết hợp với phân trang

Cả hai kỹ thuật phân đoạn và phân trang đều có những ưu điểm và nhược điểm riêng. Sự kết hợp hai phương pháp này có thể tận dụng tối đa ưu điểm của chúng, cũng như loại bỏ các nhược điểm. Các bộ vi xử lý phổ biến nhất hiện nay (như dòng Motorola 68000 được thiết kế dựa trên cơ sở không gian địa chỉ phẳng, hay họ Intel 80x86 và Petium dựa trên cơ sở phân đoạn) đều hướng tới mô hình bộ nhớ hợp nhất sự kết hợp của phân trang và phân đoạn. Sự kết hợp này được thể hiện tốt nhất bởi kết cấu của Intel 386.

Ấn bản IBM OS/2 32-bit là một hệ điều hành chạy trên đỉnh của kiến trúc Intel 386 (hay cao hơn). Intel 386 sử dụng kết hợp phân đoạn với phân trang cho việc quản lý bộ nhớ. Số lượng tối đa các phân đoạn trên tiến trình là 16KB và mỗi phân đoạn có thể lên đến 4GB. Kích thước trang là 4 KB. Mô tả đầy đủ về kiến trúc quản lý bộ nhớ của Intel 386 trở lên có thể tham khảo trong giáo trình Kiến trúc máy tính. Trong chương này, chúng tôi sẽ trình bày các ý tưởng quan trọng liên quan đến vấn đề quản lý bộ nhớ ảo (được xem xét chi tiết ở phần sau của chương này).

3.2 Kỹ thuật bộ nhớ ảo (Virtual Memory)

3.2.1 Tổng quan

Ở phần trên của chương này, chúng ta đã thảo luận các chiến lược quản lý bộ nhớ được dùng trong hệ thống máy tính. Tất cả những chiến lược này có cùng mục đích: giữ nhiều tiến trình trong bộ nhớ cùng một lúc để cho phép sự đa chương. Tuy nhiên, chúng có khuynh hướng yêu cầu toàn bộ tiến trình ở trong bộ nhớ trước

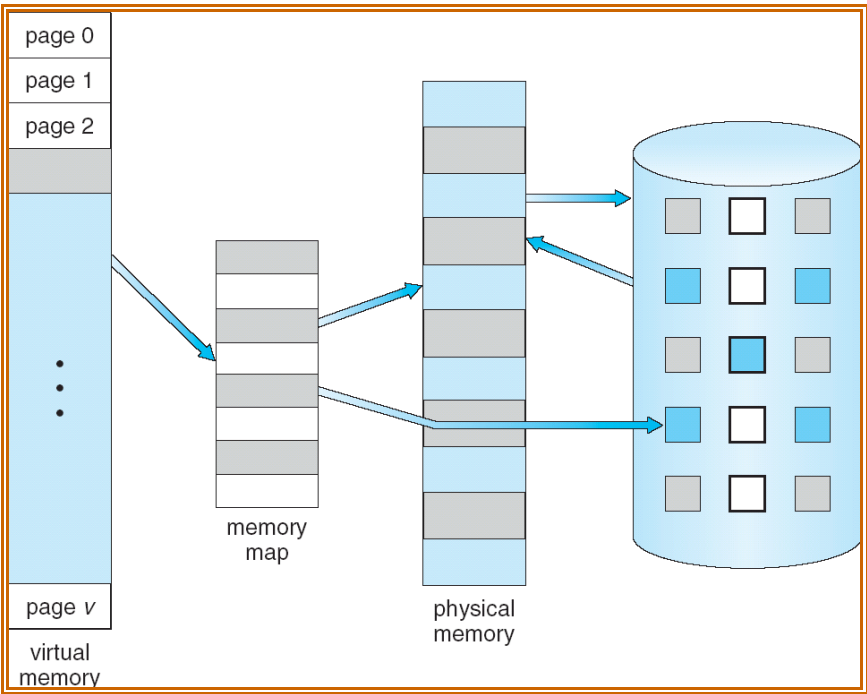
khi tiến trình có thể thực thi. Điều này có thể gây lãng phí bộ nhớ, vì không phải lúc nào tất cả các trang/đoạn của tiến trình cần thiết để tiến trình này có thể hoạt động được.

Khả năng thực thi chương trình với chỉ một phần của chương trình ở trong bộ nhớ có nhiều lợi điểm:

- Chương trình sẽ không còn bị ràng buộc bởi không gian bộ nhớ vật lý sẵn có. Người lập trình có thể thiết kế chương trình có không gian địa chỉ ảo rất lớn, đơn giản hoá tác vụ lập trình.
- Vì mỗi chương trình người dùng có thể lấy ít hơn bộ nhớ vật lý nên nhiều chương trình hơn có thể được thực thi tại một thời điểm. Điều này giúp gia tăng việc sử dụng CPU và thông lượng nhưng không tăng thời gian đáp ứng.
- Yêu cầu ít nhập/xuất hơn để nạp hay hoán vị mỗi chương trình người dùng trong bộ nhớ vì thế mỗi chương trình người dùng sẽ chạy nhanh hơn.

Bộ nhớ ảo là một kỹ thuật cho phép việc thực thi của tiến trình mà tiến trình đó có thể không được nạp hoàn toàn vào trong bộ nhớ. Một ưu điểm quan trọng của cơ chế này là cho phép các chương trình có thể lớn hơn bộ nhớ vật lý. Ngoài ra, bộ nhớ ảo “phóng đại” bộ nhớ chính thành bộ nhớ logic cực lớn khi được *hiển thị* bởi người dùng. Kỹ thuật này giải phóng người lập trình cho việc quan tâm đến giới hạn kích thước bộ nhớ. Bộ nhớ ảo cũng cho phép các tiến trình dễ dàng chia sẻ tập tin và không gian địa chỉ, cung cấp cơ chế hữu hiệu cho việc tạo tiến trình.

Bộ nhớ ảo là sự tách biệt bộ nhớ logic từ bộ nhớ vật lý. Việc tách biệt này cho phép không gian bộ nhớ ảo rất lớn được cung cấp cho người lập trình khi bộ nhớ vật lý sẵn có là nhỏ hơn rất nhiều (hình 3.16).



Hình 3.16. Lưu đồ minh họa giữa không gian bộ nhớ ảo và bộ nhớ vật lý

Bộ nhớ ảo thực hiện tác vụ lập trình dễ hơn nhiều vì người lập trình không cần lo lắng về dung lượng bộ nhớ vật lý sẵn có nữa hay về mã gì có thể được thay thế trong việc phủ lấp; thay vào đó, người lập trình có thể quan tâm vấn đề được lập trình. Trên những hệ thống hỗ trợ bộ nhớ ảo, việc phủ lấp hầu như biến mất.

Bộ nhớ ảo thường được cài đặt bởi kỹ thuật phân trang theo yêu cầu (demand paging). Nó cũng có thể được cài đặt trong cơ chế phân đoạn. Các hệ thống hiện nay thường cung cấp cơ chế kết hợp kỹ thuật phân đoạn và phân trang (tức là phân trang trên đoạn). Trong cơ chế này các phân đoạn được chia thành các trang. Do đó, tầm nhìn người dùng là phân đoạn, nhưng hệ điều hành có thể cài đặt tầm nhìn này với cơ chế phân trang theo yêu cầu. Phân đoạn theo yêu cầu cũng có thể được dùng để cung cấp bộ nhớ ảo. Các hệ thống máy tính của Burrough dùng phân đoạn theo yêu cầu. Tuy nhiên, các giải thuật thay thế đoạn phức tạp hơn các giải thuật thay thế trang vì các đoạn có kích thước thay đổi. Chúng ta sẽ không đề cập phân đoạn theo yêu cầu trong giáo trình này.

Để cài đặt được bộ nhớ ảo hệ điều hành cần phải có:

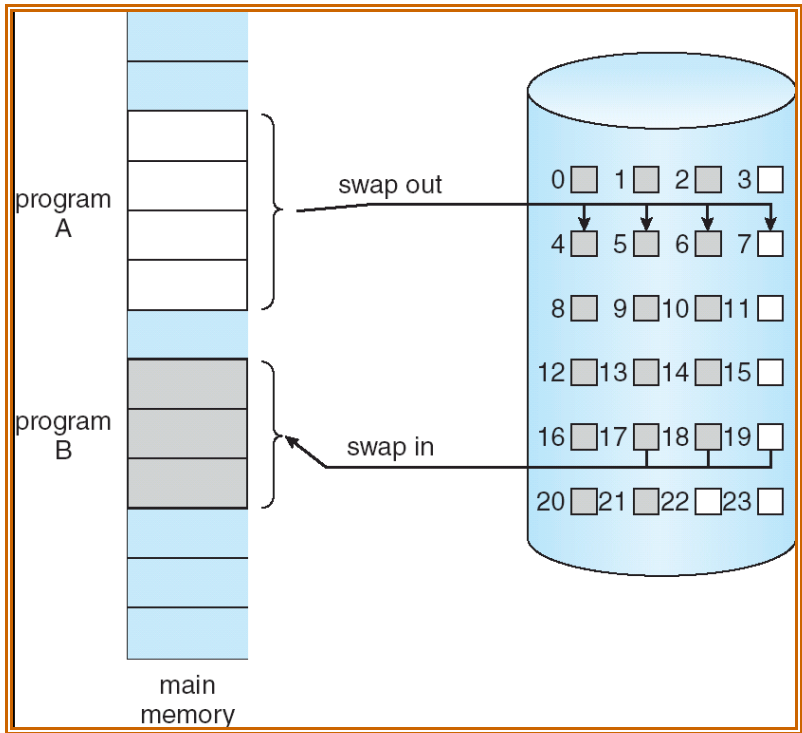
- Một lượng không gian bộ nhớ phụ (đĩa) cần thiết đủ để chứa các trang bị swap out, không gian đĩa này được gọi là không gian swap.
- Có cơ chế để theo dõi các trang của một tiến trình, của tất cả các tiến trình đang hoạt động trên bộ nhớ chính, là đang ở trên bộ nhớ chính hay ở trên bộ nhớ phụ.
- Dựa vào các tiêu chuẩn cụ thể để chọn một trang nào đó trong số các trang đang ở trên bộ nhớ chính để swap out trong trường hợp cần thiết. Các hệ điều hành đã đưa ra các thuật toán cụ thể để phục vụ cho mục đích này.

Việc sử dụng bộ nhớ ảo mang lại các lợi ích sau đây:

- Hệ điều hành có thể nạp được nhiều tiến trình hơn vào bộ nhớ, trên bộ nhớ tồn tại các trang của nhiều tiến trình khác nhau.
- Có thể nạp vào bộ nhớ một tiến trình có không gian địa chỉ lớn hơn tất cả không gian địa chỉ của bộ nhớ vật lý. Trong thực tế người lập trình có thể thực hiện việc này mà không cần sự hỗ trợ của hệ điều hành và phần cứng bằng cách thiết kế chương trình theo cấu trúc Overlay, việc làm này là quá khó đối với người lập trình. Với kỹ thuật bộ nhớ ảo người lập trình không cần quan tâm đến kích thước của chương trình và kích thước của bộ nhớ tại thời điểm nạp chương trình, tất cả mọi việc này đều do hệ điều hành và phần cứng thực hiện.

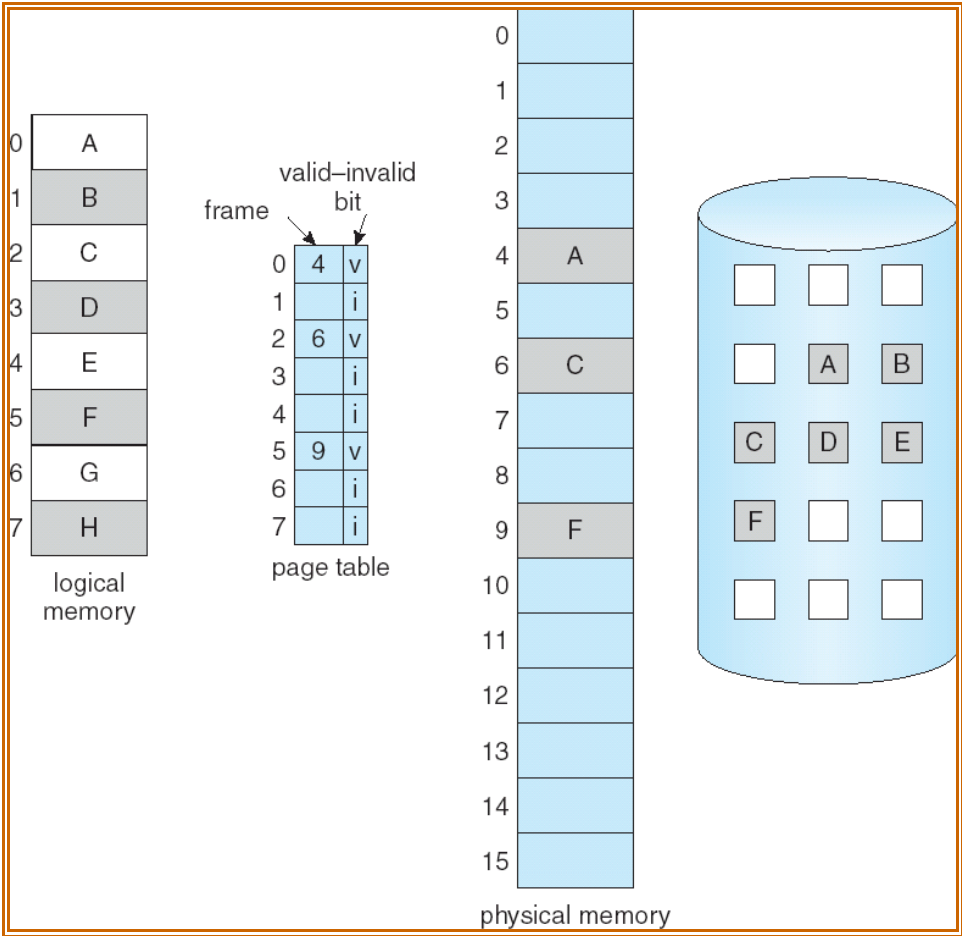
3.2.2 Phân trang theo yêu cầu

Một hệ thống phân trang theo yêu cầu tương tự một hệ thống phân trang với cơ chế swap (hình 3.17). Các tiến trình định vị trong bộ nhớ phụ (thường là đĩa). Khi chúng ta muốn thực thi một tiến trình, chúng ta chuyển nó vào bộ nhớ (swap in). Tuy nhiên, thay vì chuyển toàn bộ tiến trình vào trong bộ nhớ chính, chúng ta sử dụng một đơn vị chuyển đổi, gọi là “lazy swapper”. Đơn vị chuyển đổi sẽ không bao giờ chuyển một trang vào trong bộ nhớ chính trừ khi trang đó được yêu cầu.



Hình 3.17. Chuyển dời của các trang bộ nhớ đến không gian đĩa liên tục

Trong kỹ thuật phân trang đơn, mỗi tiến trình sở hữu một bảng trang riêng, khi tất cả các trang của tiến trình được nạp vào bộ nhớ chính thì bảng trang của tiến trình được tạo ra và cũng được nạp vào bộ nhớ (nếu lớn), mỗi phần tử trong bảng trang chỉ chứa số hiệu của khung trang mà trang tương ứng được nạp vào. Trong kỹ thuật bộ nhớ ảo cũng vậy, nhưng một phần tử trong bảng trang sẽ chứa nhiều thông tin phức tạp hơn. Bởi vì trong kỹ thuật bộ nhớ ảo chỉ có một vài page của tiến trình được nạp vào bộ nhớ chính, do đó cần phải có một bit để cho biết một page tương ứng của tiến trình là có hay không trên bộ nhớ chính và một bit cho biết page có bị thay đổi hay không so với lần nạp gần đây nhất. Cơ chế bit hợp lệ-không hợp lệ (valid-invalid) có thể được dùng cho mục đích này. Thời điểm khi bit được đặt “hợp lệ”, giá trị này biểu thị rằng trang được tham chiếu tới là hợp lệ và ở đang trong bộ nhớ chính. Nếu một bit được đặt “không hợp lệ”, giá trị này biểu thị rằng trang không hợp lệ (nghĩa là trang không ở trong không gian địa chỉ của tiến trình) hoặc hợp lệ nhưng hiện đang ở trên đĩa. Phần tử trong bảng trang cho trang không ở trong bộ nhớ đơn giản được đánh dấu không hợp lệ, hay chứa địa chỉ của trang trên đĩa (hình 3.18).

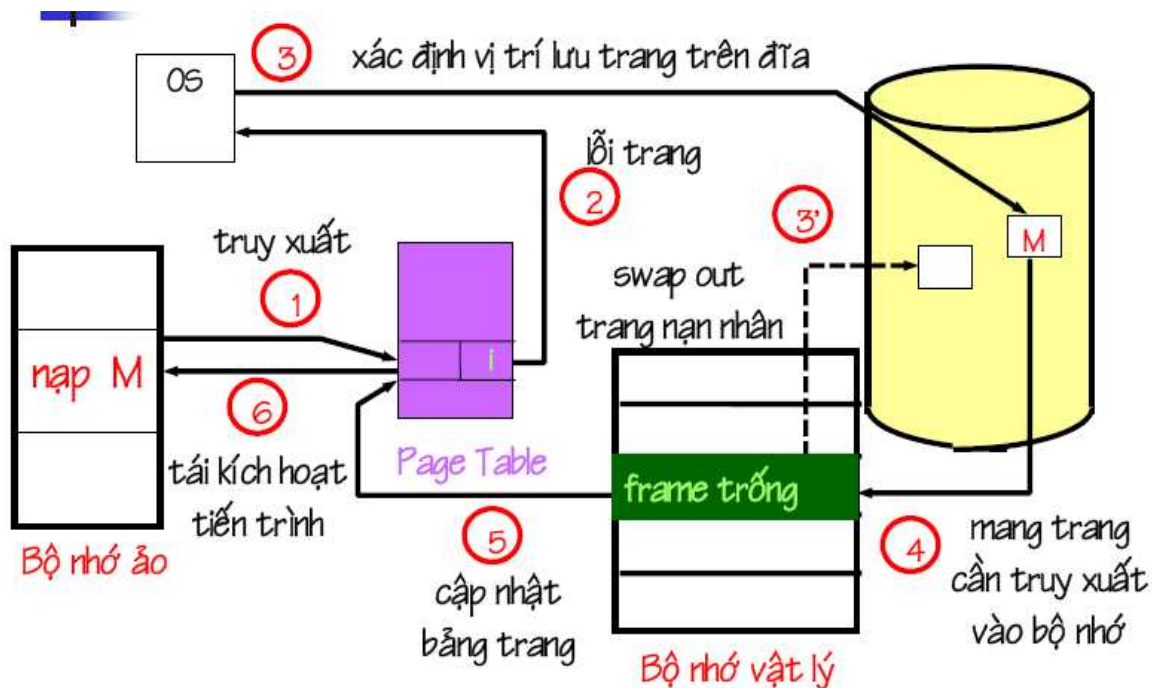


Hình 3.18. Bảng trang biểu thị một số trang không ở trong bộ nhớ chính

3.2.3 Lỗi trang (page fault)

Trong mô hình bộ nhớ ảo khi cần truy xuất đến một page của tiến trình thì trước hết hệ thống phải kiểm tra bit valid-invalid tại phần tử tương ứng với page cần truy xuất trong bảng trang để biết được page cần truy xuất đã được nạp vào bộ nhớ hay chưa. Trường hợp hệ thống cần truy xuất đến một page của tiến trình mà page đó đã được nạp vào bộ nhớ chính, được gọi là truy xuất hợp lệ (v: valid). Trường hợp hệ thống cần truy xuất đến một page của tiến trình mà page đó chưa được nạp vào bộ nhớ chính, được gọi là truy xuất bất hợp lệ (i: invalid). Khi hệ thống truy xuất đến một trang của tiến trình mà trang đó không thuộc phạm vi không gian địa chỉ của tiến trình cũng được gọi là truy xuất bất hợp lệ.

Khi hệ thống truy xuất đến một page được đánh dấu là bất hợp lệ thì sẽ phát sinh một lỗi trang. Như vậy lỗi trang là hiện tượng hệ thống cần truy xuất đến một page của tiến trình mà trang này chưa được nạp vào bộ nhớ, hay không thuộc không gian địa chỉ của tiến trình. Ở đây ta chỉ xét lỗi trang của trường hợp: Page cần truy xuất chưa được nạp vào bộ nhớ chính.



Hình 3.19. Các bước xử lý khi xảy ra lỗi trang

Khi nhận được tín hiệu lỗi trang, hệ điều hành phải tạm dừng tiến trình hiện tại để tiến hành việc xử lý lỗi trang, theo các bước như sau (hình 3.19):

1. Kiểm tra truy xuất đến bộ nhớ là hợp lệ hay bất hợp lệ
2. Nếu truy xuất bất hợp lệ: Kết thúc chương trình.

Ngược lại: đến bước 3

3. Tìm vị trí chứa trang muốn truy xuất trên đĩa.
4. Tìm một khung trang trống trong bộ nhớ chính:
 - a. Nếu tìm thấy: đến bước 5
 - b. Nếu không còn khung trang trống, chọn một khung trang nạn nhân để swap out, cập nhật bảng trang tương ứng rồi đến bước 5
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.
6. Tái kích hoạt tiến trình người sử dụng.

Vì chúng ta lưu trạng thái (thanh ghi, mã điều kiện, bộ đếm chỉ thị lệnh) của tiến trình bị ngắt khi lỗi trang xảy ra, nên hoàn toàn có thể tái khởi động lại tiến trình một cách chính xác. Trong cách thức này, chúng ta có thể thực thi một tiến trình mặc dù các trang của nó chưa được nạp vào trong bộ nhớ chính. Khi tiến trình cố gắng truy xuất các vị trí chưa ở trong bộ nhớ, phần cứng trap tới hệ điều hành

(lỗi trang). Hệ điều hành đọc trang được yêu cầu vào bộ nhớ và khởi động lại tiến trình như thể trang luôn ở trong bộ nhớ.

Các vấn đề liên quan đến việc xử lý lỗi trang có thể được đề cập như sau :

- Chọn trang nào để nạp vào bộ nhớ chính : vấn đề này liên quan đến chiến lược nạp trang.
- Chọn trang nạn nhân nào trong bộ nhớ chính để đưa ra bộ nhớ phụ : vấn đề này liên quan đến chiến lược thay thế trang.
- Cấp phát khung trang : vấn đề này liên quan đến chiến lược cấp phát khung trang.

3.2.4 Chiến lược nạp trang

Chiến lược này liên quan đến việc quyết định thời điểm nạp một hay nhiều trang vào bộ nhớ chính. Có 2 chiến lược thường được đề cập :

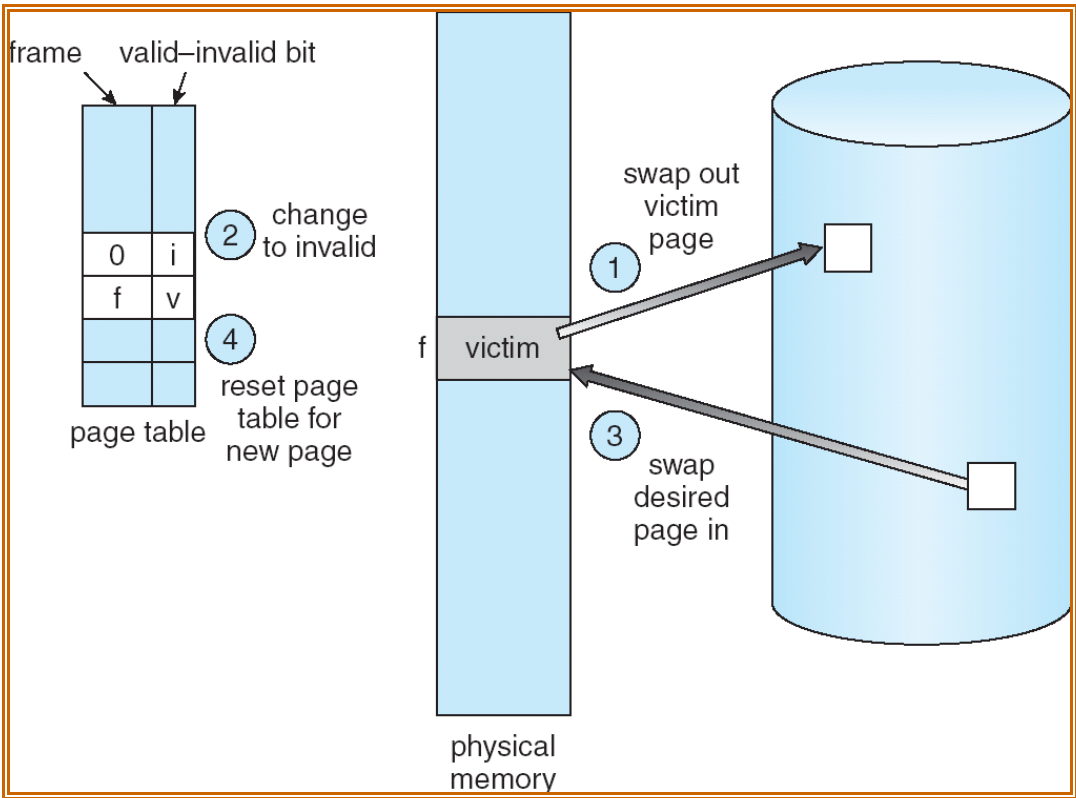
- Nạp trước (prepaging) : Nạp sẵn một số trang cần thiết vào bộ nhớ chính trước khi truy xuất chúng.
- Nạp sau (pure demand paging) : Chỉ nạp trang khi được yêu cầu truy xuất đến trang đó.

Chiến lược nạp sau có thể dẫn đến tần suất lỗi trang cao hơn nạp trước, tuy nhiên lại có thể tiết kiệm không gian nhớ. Vì vậy, hệ điều hành thường kết hợp cả 2 chiến lược này trong vấn đề nạp trang.

3.2.5 Chiến lược thay thế trang (Page Replacement)

Mục tiêu của chiến lược thay thế trang là đảm bảo sao cho tần suất xảy ra lỗi trang là thấp nhất. Quá trình thay thế trang có thể được thực hiện như sau (Hình 3.20):

- 1) Tìm vị trí trang mong muốn trên bộ nhớ phụ.
- 2) Tìm khung trang trống :
 - a. Nếu có khung trang trống, sử dụng nó để nạp trang từ bộ nhớ phụ.
 - b. Nếu không có khung trang trống, sử dụng một giải thuật thay thế trang để chọn khung trang chứa trang nạn nhân (trang sẽ được đưa ra bộ nhớ phụ).
 - c. Đưa trang nạn nhân (victim) ra bộ nhớ phụ, cập nhật bảng trang và bảng khung trang tương ứng.
- 3) Đọc trang mong muốn vào khung trang trống; thay đổi bảng trang và bảng khung trang.
- 4) Khởi động lại tiến trình.



Hình 3.20. Các bước thay thế trang

Có nhiều chiến lược thay thế trang khác nhau. Mỗi hệ điều hành có thể có cơ chế thay thế của chính nó. Việc lựa chọn một v thay thế trang như thế nào phải đảm bảo một tỉ lệ lỗi trang nhỏ nhất.

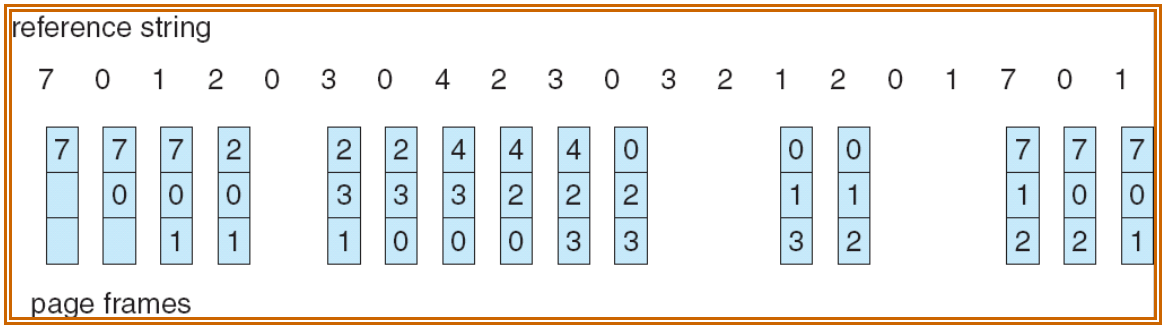
Chúng ta đánh giá các chiến lược bằng cách thực hiện chúng trên cùng một chuỗi các truy xuất bộ nhớ cụ thể với số khung trang giống nhau và sau đó tính số lượng lỗi trang. Chuỗi các truy xuất bộ nhớ thường được gọi ngắn gọn là chuỗi truy xuất. Chúng ta có thể phát sinh chuỗi truy xuất giả tạo (thí dụ, bằng bộ phát sinh số ngẫu nhiên), ví dụ như sau:

- Chuỗi truy xuất
 - 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1
 - 3 khung trang (frame).

3.2.5.1. Chiến lược thay thế trang FIFO

Đây được xem là chiến lược đơn giản nhất. Chiến lược này gắn với mỗi trang thời gian khi trang đó được đưa vào trong bộ nhớ. Khi một trang phải được thay thế, trang nạn nhân là trang được nạp vào lâu nhất (cũ nhất) trong hệ thống. Để thực hiện chiến lược này, một hàng đợi FIFO có thể được tạo ra để quản lý các trang trong bộ nhớ. Trang đầu tiên trong danh sách sẽ là trang nạn nhân.

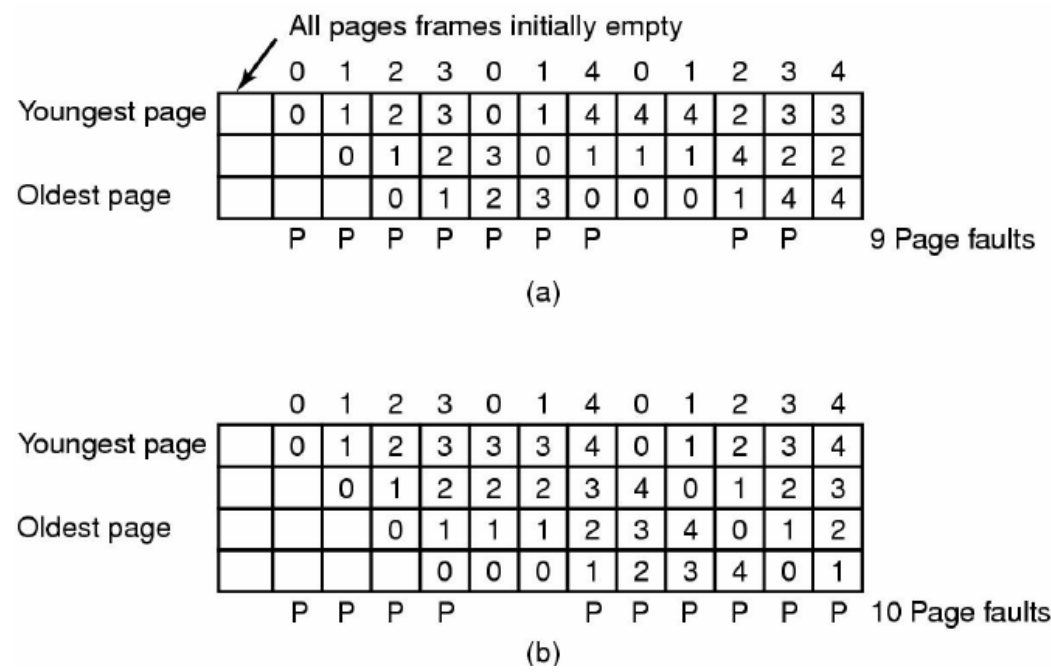
Ta xét ví dụ chiến lược FIFO với chuỗi truy xuất và số khung trang như trên (hình 3.21).



Hình 3.21. Chiến lược thay thế trang FIFO (15 lỗi trang)

Chiến lược thay thế trang FIFO là dễ hiểu và dễ lập trình. Tuy nhiên, nó có một vấn đề là không xét đến tính sử dụng, tức là có thể gặp trường hợp trang được sử dụng để thay thế lại có thể là trang chứa nhiều dữ liệu cần thiết, thường xuyên được sử dụng nên được nạp sớm, do vậy khi chuyển ra bộ nhớ phụ sẽ nhanh chóng gây ra lỗi trang.

Một vấn đề nữa có thể phát sinh khi sử dụng chiến lược FIFO, đó là càng sử dụng nhiều khung trang, lại càng phát sinh nhiều lỗi trang (hình 3.22). Vấn đề này được gọi là nghịch lý Belady (tỉ lệ lỗi trang có thể tăng khi số lượng khung trang được cấp phát tăng).

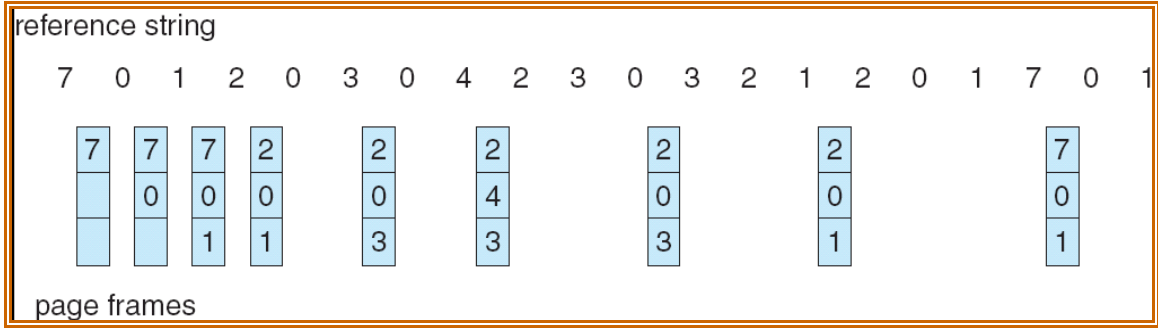


Hình 3.22. Chiến lược thay thế trang FIFO – nghịch lý Belady

3.2.5.2. Chiến lược thay thế trang tối ưu (Optimal)

Kết quả phát hiện sự nghịch lý của Belady là tìm ra một chiến lược thay thế trang tối ưu. Chiến lược thay thế trang tối ưu có tỉ lệ lỗi trang thấp nhất và sẽ không bao giờ gặp phải sự nghịch lý Belady. Chiến lược này đơn giản là thay thế trang (trang nạn nhân) mà nó không được dùng cho một khoảng thời gian lâu nhất trong

tương lai.

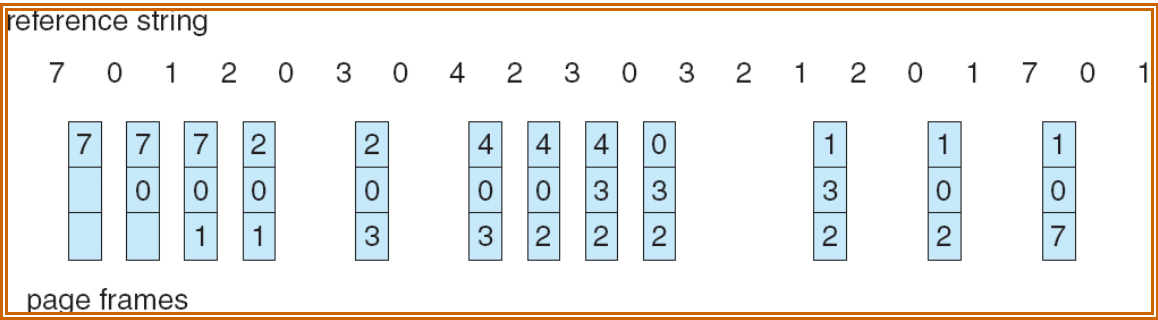


Hình 3.23. Chiến lược thay thế trang tối ưu (9 lỗi trang)

Tuy nhiên, chiến lược thay thế trang tối ưu là rất khó cài đặt vì nó yêu cầu kiến thức tương lai về chuỗi truy xuất. Vì vậy, chiến lược tối ưu là rất khó khả thi và thường chỉ được sử dụng chủ yếu trong nghiên cứu so sánh. Ví dụ, nó có thể có ích để biết rằng, mặc dù một chiến lược là không tối ưu nhưng tỉ lệ lỗi trang của nó nằm trong khoảng 12.3% (của chiến lược tối ưu) thường là chiến lược tồi, và trong khoảng 4.7% là trung bình...

3.2.5.3. Chiến lược thay thế trang LRU

Nếu chiến lược tối ưu là không khả thi, một chiến lược xấp xỉ tối ưu là có thể được xem xét. Sự khác biệt chủ yếu giữa chiến lược FIFO và Optimal là FIFO sử dụng thời gian khi trang được đưa vào bộ nhớ; chiến lược Optimal lại sử dụng thời gian khi trang sẽ được sử dụng. Nếu chúng ta sử dụng quá khứ gần đây như là một xấp xỉ của tương lai gần thì chúng ta sẽ thay thế trang mà nó không được dùng cho khoảng thời gian lâu nhất trong quá khứ (hình 3.24). Theo hướng tiếp cận này, chiến lược LRU là sử dụng trang nạn nhân là trang lâu nhất chưa sử dụng đến trong quá khứ. (least-recently-used (LRU)).

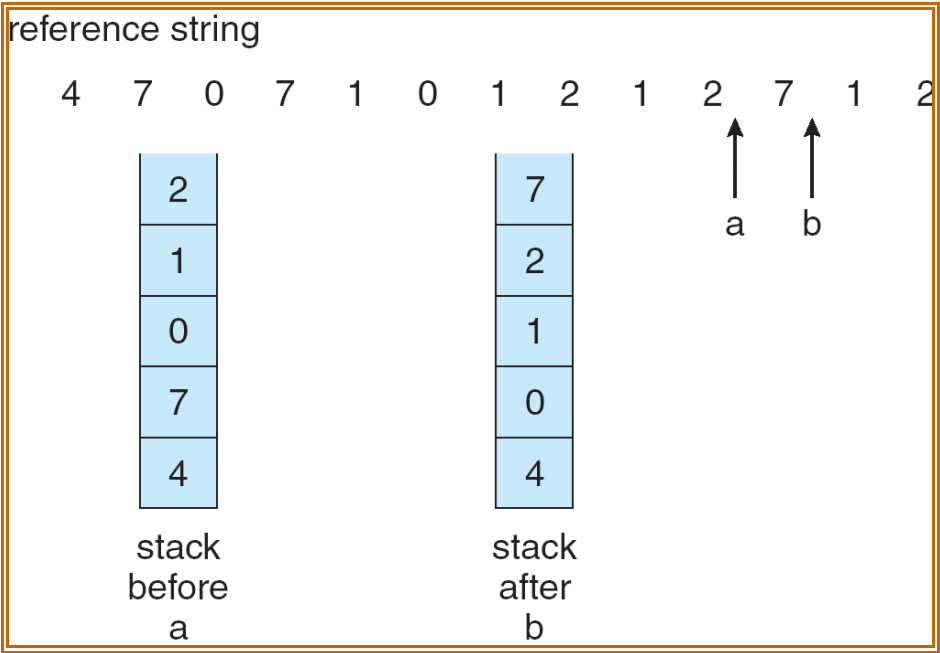


Hình 3.24. Chiến lược thay thế trang LRU (12 lỗi trang)

Chính sách LRU thường được dùng như giải thuật thay thế trang và được đánh giá là khá tốt. Vấn đề chính là cách cài đặt để thực hiện chiến lược thay thế trang LRU. Thường thì một chiến lược thay thế trang LRU có thể yêu cầu sự trợ giúp từ phần cứng. Có hai cách cài đặt khả thi, đó là:

- Sử dụng bộ đếm:

- Thêm trường **reference time** cho mỗi phần tử trong bảng trang
 - Thêm vào cấu trúc của CPU một bộ đếm **Counter**
 - Mỗi lần có sự truy xuất đến 1 trang trong bộ nhớ:
 - Giá trị của counter tăng lên 1
 - Giá trị của counter được ghi nhận vào reference time của trang tương ứng
 - Thay thế trang có reference time là min
- 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



Hình 3.25. Một ví dụ sử dụng Stack để ghi những trang được tham khảo gần nhất

3.2.5.4. Chiến lược thay thế trang xấp xỉ LRU

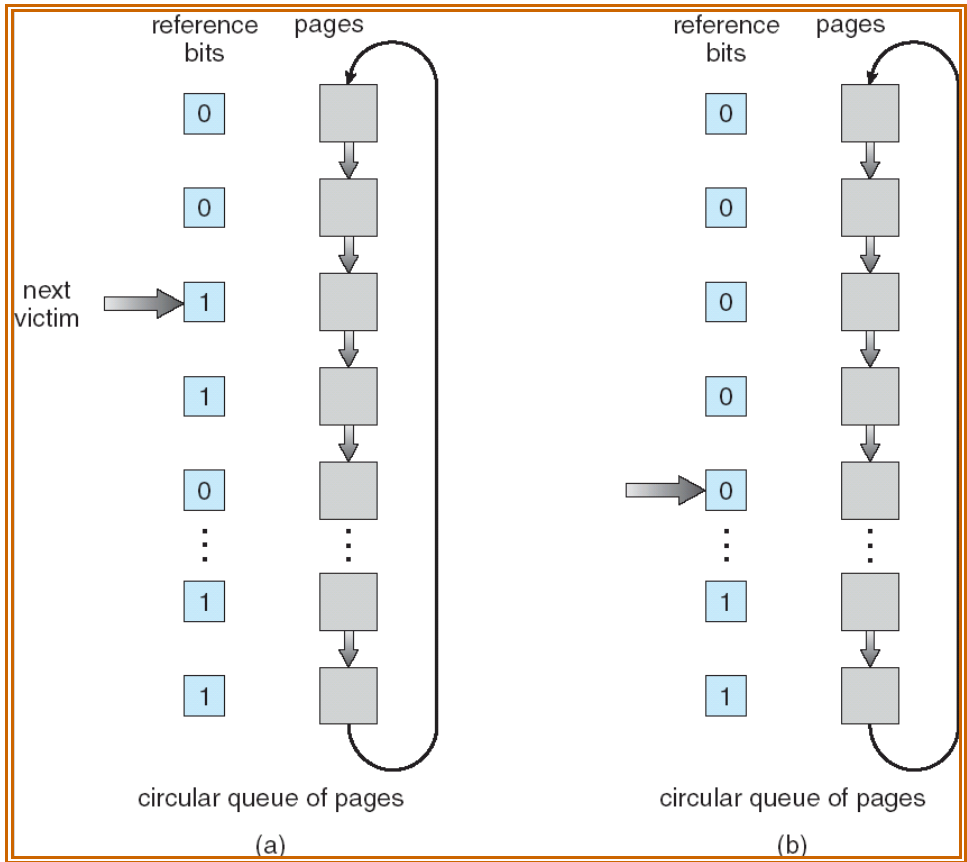
Rất ít hệ thống máy tính cung cấp đầy đủ sự hỗ trợ phần cứng cho chiến lược thay thế trang LRU. Một số hệ thống không cung cấp bất cứ sự hỗ trợ phần cứng nào và khi đó chiến lược thay thế trang khác (như FIFO) phải được sử dụng. Tuy nhiên, nhiều hệ thống cung cấp một vài hỗ trợ dưới dạng 1 bit tham khảo (reference):

- Gắn với một phần tử trong bảng trang.
- Được khởi gán bằng 0
- Được phần cứng đặt giá trị 1 mỗi lần trang tương ứng được truy cập
- Được phần cứng gán trở về 0 sau từng chu kỳ qui định trước

Tuy nhiên, bit reference chỉ giúp xác định những trang có truy cập, không xác định thứ tự truy cập. Một chiến lược khác thường được sử dụng là chiến lược thay thế trang cơ hội thứ 2 (Clock algorithm). Giải thuật thay thế trang cơ hội thứ hai cơ bản là giải thuật thay thế FIFO, được thực hiện như sau:

- Sử dụng một bit reference duy nhất.
- Chọn được trang nạn nhân theo FIFO
- Kiểm tra bit reference của trang đó:
 - Nếu reference = 0, đúng là trang nạn nhân.
 - Nếu reference = 1, cho trang này một cơ hội thứ 2:
 - Xoá bit reference = 0
 - thời điểm vào Ready List được cập nhật lại là thời điểm hiện tại
- Chọn trang FIFO tiếp theo...

Nhận xét: 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. Nếu trang thường xuyên được sử dụng, bit reference của nó sẽ duy trì được giá trị 1, và trang hầu như không bao giờ bị thay thế.



Hình 3.26. Chiến lược thay thế trang cơ hội thứ 2 – Second-chance (clock)

Một cách để cài đặt chiến lược cơ hội thứ hai là thực hiện như một hàng đợi vòng. Một con trỏ hiển thị trang nào được thay thế tiếp theo. Khi một khung trang được yêu cầu, con trỏ tăng cho tới khi nó tìm được trang với bit tham khảo 0. Khi nó tăng, nó xoá các bit tham khảo (hình 3.26). Một khi trang nạn nhân được tìm thấy, trang được thay thế và trang mới được chèn vào hàng đợi vòng trong vị trí đó. Chú ý rằng, trong trường hợp xấu nhất khi tất cả bit được đặt, con trỏ xoay vòng suốt toàn hàng đợi, cho mỗi trang một cơ hội thứ hai. Thay thế cơ hội thứ hai trở thành thay thế FIFO nếu tất cả bit được đặt.

3.2.6 Trì trệ hệ thống

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 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 được gọi là Thrashing nếu nó dành nhiều thời gian (bận rộn) với việc hoán đổi các trang vào và ra hơn là thời gian thực hiện (tất cả tiến trình đều bận rộn xử lý lỗi 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 tập làm việc (Working Set): tập hợp các trang tiến trình đang truy

xuất tại 1 thời điểm:

- Các pages được truy xuất trong Δ lần cuối cùng sẽ nằm trong working set của tiến trình.
- Δ : working set parameter.
- Kích thước của WS thay đổi theo thời gian tùy vào locality của tiến trình.
- $\Delta \equiv \text{working-set window} \equiv \text{số lần truy cập (ví dụ: 10,000 instruction)}$
- 2 6 1 5 7 7 7 5 1 6 2 3 4 1 2 3 4 4 4 3 4 3 4 4 4 1 3 2 3
 - $\Delta = 10$
 - $WS(t1) = \{1,2,5,6,7\}$, $WS(t2) = \{3,4\}$
- WSS_i (Working set of Process P_i) = tổng số trang được truy cập trong Δ lần gần đây nhất.
- $D = \sum WSS_i \equiv \text{Tổng các frame cần cho } N \text{ tiến trình trong hệ thống}$
- If $D > m \Rightarrow \text{Thrashing}$
 - If $D > m$, chọn một / một số tiến trình để đình chỉ tạm thời

Một số câu hỏi và bài tập

Câu 1. Giải thích sự khác biệt giữa địa chỉ logic và địa chỉ vật lý.

Câu 2. Giải thích sự khác biệt giữa hiện tượng phân mảnh nội vi và phân mảnh ngoại vi.

Câu 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:

- Thuật toán First-Fit
- Thuật toán Best-Fit
- Thuật toán Worst-Fit

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

Câu 4. Xét 1 không gian địa chỉ có 8 trang, mỗi trang có kích thước 1K, ánh xạ vào bộ nhớ vật lý có 32 khung trang.

- Địa chỉ logic bao gồm bao nhiêu bit
- Địa chỉ vật lý bao gồm bao nhiêu bit

Câu 5. Khi nào thì xảy ra lỗi trang? Mô tả xử lý của hệ điều hành khi có lỗi trang.

Câu 6. Một máy tính 32-bit địa chỉ, sử dụng một bảng trang 2 cấp. Địa chỉ ảo được phân bổ như sau: 9 bit dành cho bảng trang cấp 1, 11 bit dành cho bảng trang cấp 2

và 12 bit còn lại dành cho offset. Cho biết kích thước một trang trong hệ thống, và địa chỉ ảo có bao nhiêu trang.

Câu 7. Giải thích ý nghĩa của việc sử dụng bảng TLB trong quản lý bộ nhớ ảo. Cho biết các thông số sau:

- Thời gian truy xuất trong TLB là: $t_c=20ns$
- Tỉ lệ tìm thấy 1 số hiệu trang trong TLB là: 80%.
- Thời gian truy cập vào bộ nhớ chính là: $t_m=75ns$
- Thời gian chuyển trang khi trang bị lỗi trang là : $t_d=500000ns$.
- Tỉ lệ trang bị thay thế khi bị lỗi trang là: 50%.

Yêu cầu:

- a) Tính thời gian truy cập thực tế (EAT) nếu số trang bị lỗi là: 0%.
- b) Tính thời gian truy cập thực tế (EAT) nếu số trang bị lỗi là :10%

Câu 8. Xem đoạn chương trình sau:

```
#define Size 64
int A[ Size, Size ], B[ Size, Size ], C[ Size, Size ];
int register i, j;
for (j=0; j<Size; j++)
    for (i=0; i<Size; i++)
        C[ i, j ] = A[ i, j ] + B[ i, j ];
```

Giả sử rằng chương trình muốn chạy trên hệ thống cần phải sử dụng trang và mỗi trang có kích thước 1KB. Mỗi số nguyên dài 4B. Rõ ràng là mỗi mảng cần 16 trang. Như một ví dụ, $A[0,0]$ - $A[0,63]$, $A[1,0]$ - $A[1,63]$, $A[2,0]$ - $A[2,63]$, $A[3,0]$ - $A[0,63]$ sẽ được lưu trữ trong phần của trang dữ liệu. Một mẫu lưu trữ tương tự có thể bắt nguồn từ phần còn lại của mảng A, B, C. Giả sử rằng hệ thống chỉ có thể cấp bộ nhớ 4 trang cho tiến trình này. Một trong các trang sẽ được sử dụng bởi chương trình và 3 trang còn lại có thể sử dụng để lưu trữ dữ liệu. Ngoài ra, 2 thanh ghi chỉ mục đảm nhiệm i và j (vì thế, bộ nhớ khi cần thiết cũng không được truy xuất 2 biến này).

a. Thảo luận mức độ thường xuyên mà lỗi xảy ra (trong mỗi số bước để thực hiện $C[i,j] = A[i,j] + B[i,j]$).

b. Có thể điều chỉnh lại chương trình để làm giảm mức độ xảy ra lỗi là thấp nhất?

c. Mức độ xảy ra lỗi trang sẽ như thế nào sau khi điều chỉnh lại chương trình này.

Câu 9. Giả sử ban đầu hệ thống có 3 khung trang (frame) còn trống và hệ điều hành cần phải nạp một danh sách các trang sau đây vào bộ nhớ: 2, 3, 2, 1, 5, 2, 4, 5, 3, 2, 5, 2. Trong trường hợp xảy ra lỗi trang, hãy sử dụng 2 thuật toán FIFO, LRU để xử lý. Nhận xét kết quả.

Chương 4. QUẢN LÝ LƯU TRỮ

Đối với hầu hết người dùng, hệ thống tập tin là diện mạo dễ nhìn thấy nhất của hệ điều hành. Nó cung cấp cơ chế cho việc lưu trữ trực tuyến và truy xuất dữ liệu, chương trình của hệ điều hành và tất cả người dùng của hệ thống máy tính. Hệ thống tập tin chứa hai phần riêng biệt: tập hợp các tập tin (files), mỗi tập tin lưu trữ dữ liệu có liên quan và cấu trúc thư mục (directory structure) mà nó tổ chức và cung cấp thông tin về tất cả tập tin trong hệ thống. Một số hệ thống tập tin còn có thêm phần thứ ba, các phân khu (partitions) mà nó được dùng để tách rời tập hợp các thư mục logic và vật lý.

Trong chương này chúng ta xét các khía cạnh khác nhau của tập tin và cấu trúc thư mục. Chúng ta cũng thảo luận các cách để quản lý việc bảo vệ tập tin (file protection), cần thiết khi nhiều người dùng truy xuất các tập tin và chúng ta muốn kiểm soát ai và cách gì truy xuất tập tin. Cuối cùng, chúng ta thảo luận việc chia sẻ giữa nhiều tiến trình, người dùng, và máy tính.

4.1. Tổ chức hệ thống tập tin

4.1.1. Tập tin

Máy tính có thể lưu giữ thông tin trên các thiết bị lưu trữ khác nhau như đĩa từ, băng từ, đĩa quang... Để cho máy tính trở nên thuận tiện và dễ sử dụng, Hệ điều hành cung cấp một cách lưu giữ thông tin logic như nhau trên các thiết bị lưu giữ đó là file (tập tin). Tập tin được ánh xạ bởi hệ điều hành trên các thiết bị vật lý. Các thiết bị lưu trữ được dùng thường ổn định vì thế nội dung không bị mất khi mất điện hay khởi động lại hệ thống.

Tập tin là đơn vị lưu trữ thông tin của bộ nhớ ngoài. Từ quan điểm người dùng, một tập tin là phần nhỏ nhất của thiết bị lưu trữ phụ logic; nghĩa là dữ liệu không thể được viết tới thiết bị lưu trữ phụ trừ khi chúng ở trong một tập tin. Các tập tin dữ liệu có thể là số, chữ, ký tự số hay nhị phân. Các tập tin có thể có dạng bất kỳ như tập tin văn bản, hay có thể được định dạng không đổi. Thông thường, một tập tin là một chuỗi các bits, bytes, dòng hay mẫu tin,..được định nghĩa bởi người tạo ra nó. Do đó, khái niệm tập tin là cực kỳ tổng quát.

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.

Thuộc tính tập tin

Để tiện cho người dùng, một tập tin được đặt tên và được tham khảo bởi tên của nó. Một tên thường là một chuỗi các ký tự, thí dụ: example.c. Một số hệ thống có sự phân biệt giữa ký tự hoa và thường trong tên, ngược lại các hệ thống khác xem hai trường hợp đó là tương đương. Khi một tập tin được đặt tên, nó trở nên độc lập với quá trình, người dùng, và thậm chí với hệ thống tạo ra nó. Thí dụ, một người dùng có thể tạo tập tin example.c, ngược lại người dùng khác có thể sửa tập tin đó bằng cách xác định tên của nó. Người sở hữu tập tin có thể ghi tập tin tới đĩa mềm, gửi nó vào email hay chép nó qua mạng và có thể vẫn được gọi example.c trên hệ thống đích.

Một tập tin có một số thuộc tính khác mà chúng rất khác nhau từ một hệ điều hành này tới một hệ điều hành khác, nhưng điển hình chúng gồm:

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

Thông tin về tất cả tập tin được giữ trong cấu trúc thư mục (directory) nằm trong thiết bị lưu trữ phụ. Điển hình, mục từ thư mục chứa tên tập tin và định danh duy nhất của nó. Định danh lần lượt xác định thuộc tính tập tin khác. Trong hệ thống có nhiều tập tin, kích thước của chính thư mục có thể là Mbyte. Bởi vì thư mục giống tập tin, phải bền, chúng phải được lưu trữ trên thiết bị và mang vào bộ nhớ khi cần.

4.1.2. Các phương thức truy cập

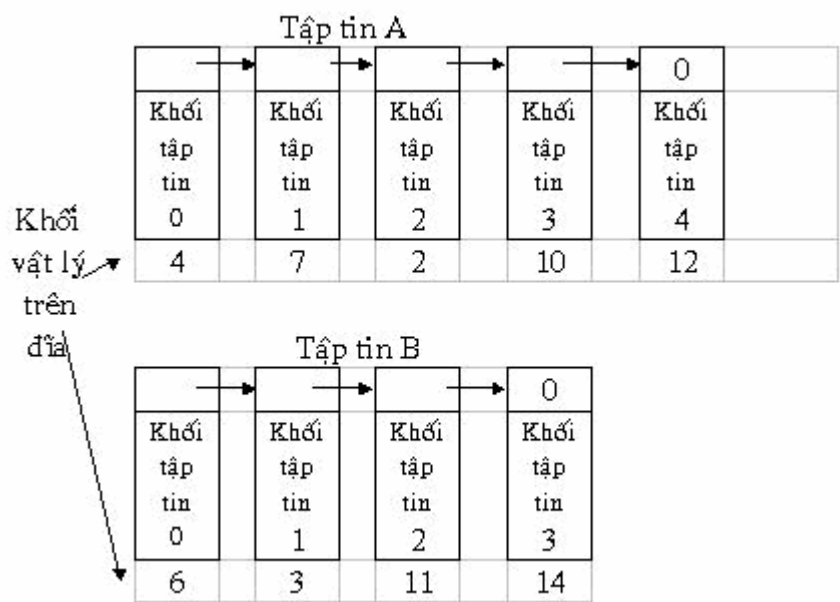
4.1.2.1 Các phương pháp lưu trữ file

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 4.1 Định vị bằng danh sách liên kết

Mọi khối đều được cấp phát, không bị lãng phí trong trường hợp phân mảnh và chỉ mục thư mục (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:

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.

Khối vật lý

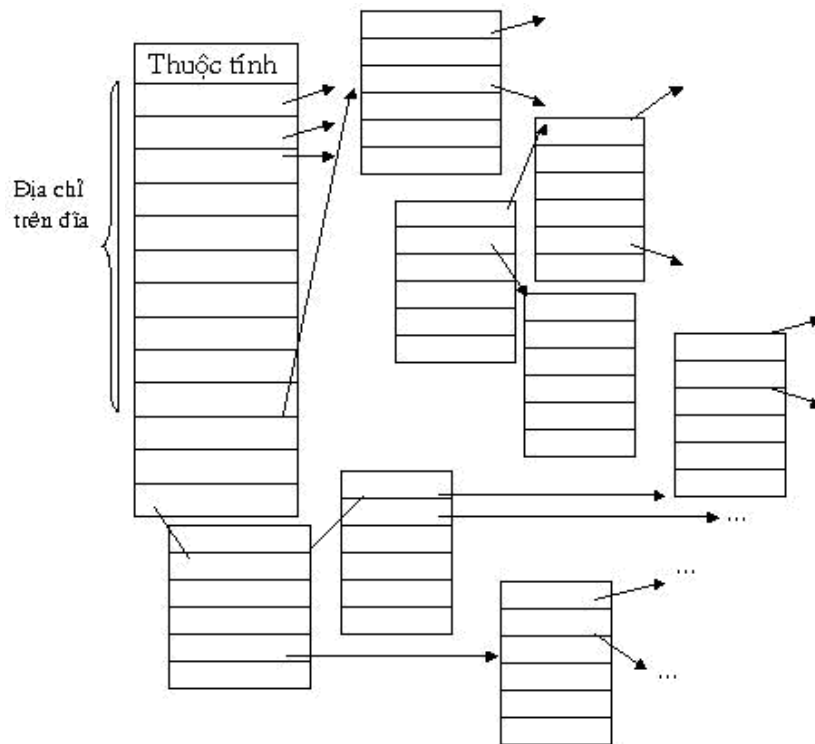
0		
1		
2	10	
3	11	
4	7	← Tập tin A bắt đầu ở đây
5		
6	3	← Tập tin B bắt đầu ở đây
7	2	
8		
9		
10	12	
11	14	
12	0	
13		
14	0	
15		← Khối chưa sử dụng

Hình 4.2 Bảng chỉ mục của danh sách liên kết

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 quản lý những hệ thống tập tin lớn. Hệ điều hành sử dụng phương pháp này là Unix (Ví dụ : BSD Unix)



Hình 4.3 Cấu trúc của I-node

4.1.2.2 Các phương pháp tổ chức, truy nhập file

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.

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.

Một kiểu truy xuất khác là **truy xuất trực tiếp**. Một tập tin có cấu trúc là các mẫu tin logic có kích thước bằng nhau, nó cho phép chương trình đọc hoặc ghi nhanh chóng mà không cần theo thứ tự. Kiểu truy xuất này dựa trên mô hình của đĩa. Đĩa cho phép truy xuất ngẫu nhiên bất kỳ khối dữ liệu nào của tập tin. Truy xuất trực tiếp được sử dụng trong trường hợp phải truy xuất một khối lượng thông tin lớn như trong cơ sở dữ liệu chẳng hạn. Ngoài ra còn có một số cách truy xuất khác dựa trên kiểu truy xuất này như truy xuất theo chỉ mục... Các tập tin truy xuất trực tiếp được dùng nhiều cho truy xuất tức thời tới một lượng lớn thông tin. Cơ sở dữ liệu thường là loại này. Khi một truy vấn tập trung một chủ đề cụ thể, chúng ta tính khối nào chứa câu trả lời và sau đó đọc khối đó trực tiếp để cung cấp thông tin mong muốn.

Không phải tất cả hệ điều hành đều hỗ trợ cả hai truy xuất tuần tự và trực tiếp cho tập tin. Một số hệ thống cho phép chỉ truy xuất tập tin tuần tự; một số khác cho phép chỉ truy xuất trực tiếp. Một số hệ điều hành yêu cầu một tập tin được định nghĩa như tuần tự hay trực tiếp khi nó được tạo ra; như tập tin có thể được truy xuất chỉ trong một cách không đổi với khai báo của nó. Tuy nhiên, chúng ta dễ dàng mô phỏng truy xuất tuần tự trên tập tin truy xuất trực tiếp. Nếu chúng ta giữ một biến cp để xác định vị trí hiện tại thì chúng ta có thể mô phỏng các thao tác tập tin tuần tự như được hiển thị trong hình 4.4. Mặc dù, không đủ và không gọn để mô phỏng một tập tin truy xuất trực tiếp trên một tập tin truy xuất tuần tự.

Các phương pháp truy xuất khác

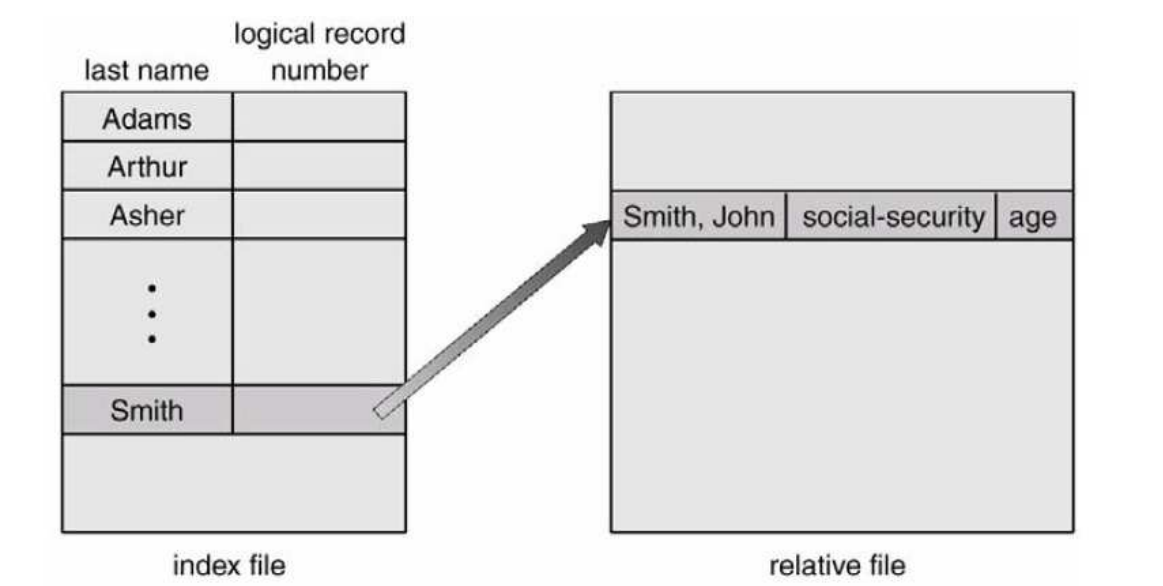
Các phương pháp truy xuất khác có thể được xây dựng trên cơ sở của phương pháp truy xuất trực tiếp. Các phương pháp khác thường liên quan đến việc xây dựng chỉ mục cho tập tin. Chỉ mục chứa các con trỏ chỉ tới các khối khác. Để tìm một mẫu tin trong tập tin, trước hết chúng ta tìm chỉ mục và sau đó dùng con trỏ để truy xuất tập tin trực tiếp và tìm mẫu tin mong muốn.

sequential access	implementation for direct access
<i>reset</i>	<i>cp = 0;</i>
<i>read next</i>	<i>read cp;</i> <i>cp = cp+1;</i>
<i>write next</i>	<i>write cp;</i> <i>cp = cp+1;</i>

Hình 4.4 Mô phỏng truy xuất tuần tự trên truy xuất trực tiếp

Với những tập tin lớn, chỉ mục tập tin có thể trở nên quá lớn để giữ trong bộ nhớ. Một giải pháp là tạo chỉ mục cho tập tin chỉ mục. Tập tin chỉ mục chính chứa

các con trỏ chỉ tới các tập tin chỉ mục thứ cấp mà nó chỉ tới các thành phần dữ liệu thật sự.



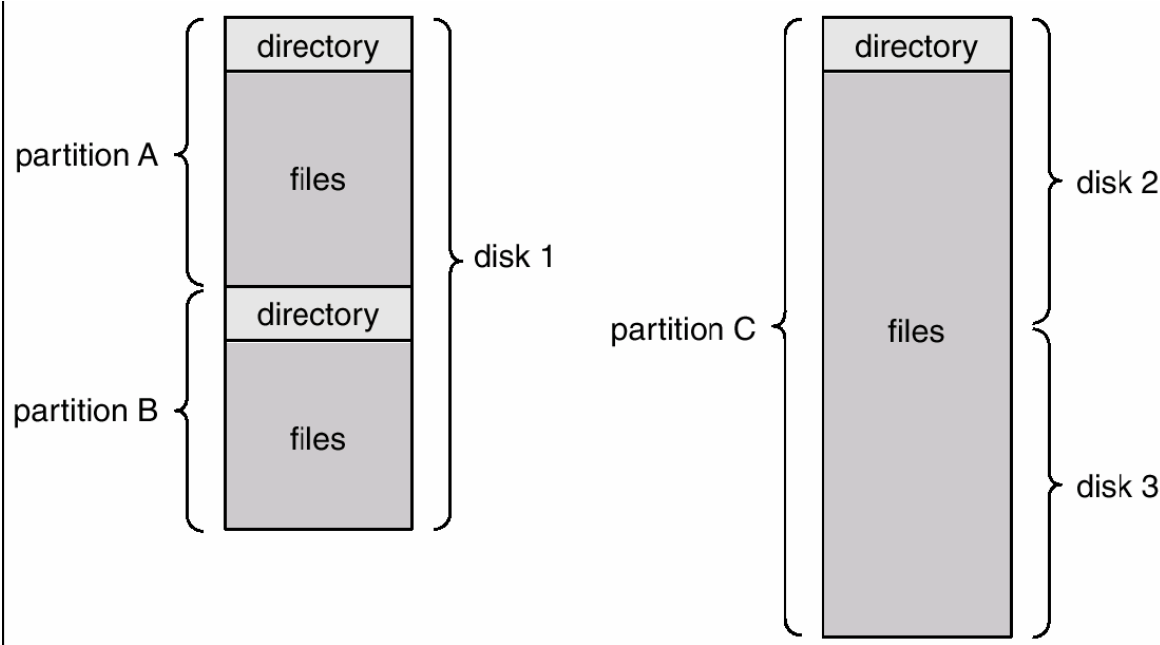
Hình 4.5 Ví dụ về chỉ mục và các tập tin liên quan

4.1.3. Cấu trúc thư mục

Các hệ thống tập tin của máy tính có thể rất lớn về số lượng. Một số hệ thống lưu trữ hàng triệu tập tin trên các terabytes đĩa. Để quản lý tất cả dữ liệu này, chúng ta cần tổ chức lại chúng. Việc tổ chức này thường được thực hiện hai phần.

Thứ nhất, đĩa được chia thành một hay nhiều phân khu (partition) hay phân vùng (volumes). Điển hình, mỗi đĩa trên hệ thống chứa ít nhất một phân khu. Phân khu này là cấu trúc cấp thấp mà các tập tin và thư mục định vị. Thỉnh thoảng các phân khu được dùng để cung cấp nhiều vùng riêng rẽ trong một đĩa, mỗi phân khu được xem như một thiết bị lưu trữ riêng, trái lại các hệ thống khác cho phép các phân khu có dung lượng lớn hơn một đĩa để nhóm các đĩa vào một cấu trúc luận lý và cấu trúc tập tin, và có thể bỏ qua hoàn toàn những vấn đề cấp phát không gian vật lý cho các tập tin. Cho lý do này, các phân khu có thể được xem như các đĩa ảo. Các phân khu cũng có thể lưu trữ nhiều hệ điều hành, cho phép hệ thống khởi động và chạy nhiều hơn một hệ điều hành.

Thứ hai, mỗi phân khu chứa thông tin về các tập tin trong nó. Thông tin này giữ trong những mục từ trong một thư mục thiết bị hay bảng mục lục phân vùng (volume table of contents). Thư mục thiết bị (được gọi đơn giản là thư mục) ghi thông tin-như tên, vị trí, kích thước và kiểu-đối với tất cả tập tin trên phân khu (như hình 4.6).



Hình 4.6 Tổ chức hệ thống tập tin điển hình

Thư mục có thể được hiển thị như một bảng danh biểu dịch tên tập tin thành các mục từ thư mục. Các thư mục có thể được tổ chức trong nhiều cách. Chúng ta muốn có thể chèn mục từ, xóa mục từ, tìm kiếm một mục từ và liệt kê tất cả mục từ trong thư mục.

Định nghĩa thư mục: Thư mục là nơi để lưu giữ tập các file.

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.

4.1.3.1. Hệ thống thư mục theo cấp bậc

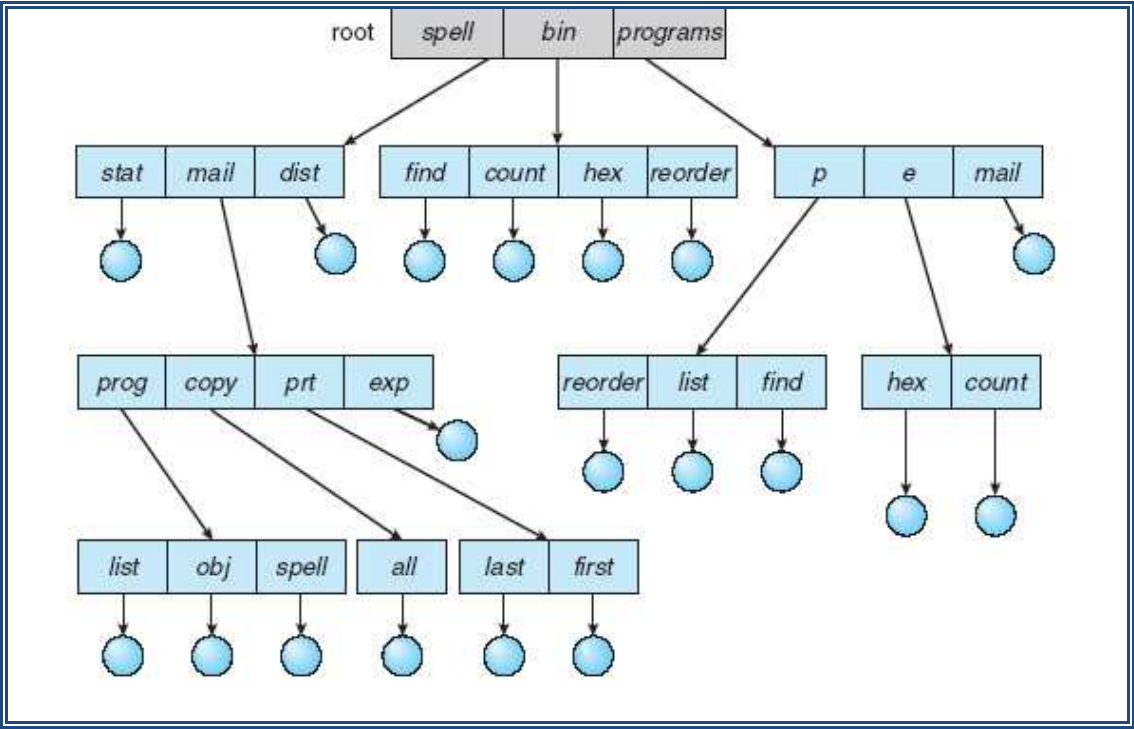
Một thư mục thường thường chứa một số *entry*, mỗi entry cho một tập tin. Mỗi entry chứa tên tập tin, thuộc tính và địa chỉ trên đĩa lưu dữ liệu hoặc một entry chỉ chứa tên tập tin và một con trỏ, trỏ tới một cấu trúc, trên đó có thuộc tính và vị trí lưu trữ của tập tin.

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 lặp là một vấn đề khó.

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

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



Hình 4.7 Hệ thống thư mục theo dạng cây

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.

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

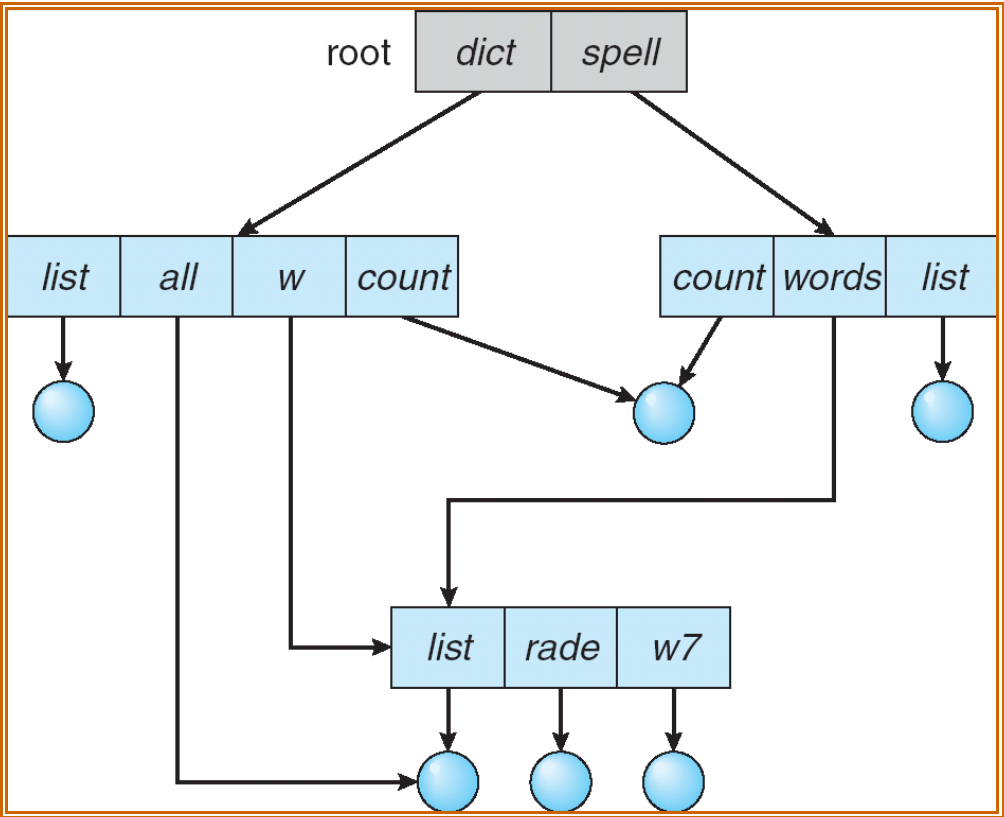
thông tập tin. Nhưng nếu một hàm thư viện thay đổi đường dẫn và sau đó không đổi lại thì sẽ có ảnh hưởng đến tiến trình.

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.

4.1.3.2. Hệ thống thư mục theo dạng đồ thị không chứa chu trình

Xét hai người lập trình đang làm việc trên một dự án chung. Các tập tin gắn với dự án đó có thể được lưu trong thư mục con, tách rời chúng từ các dự án khác và các tập tin của hai người lập trình. Nhưng vì cả hai người lập trình có trách nhiệm ngang nhau trong dự án, cả hai muốn thư mục con ở trong các thư mục của chính họ. Thư mục con nên được chia sẻ. Một thư mục hay tập tin sẽ tồn tại trong hệ thống tập tin trong hai (hay nhiều hơn) nơi tại một thời điểm.

Cấu trúc cây ngăn cản việc chia sẻ các tập tin và thư mục. Một đồ thị không chứa chu trình (acyclic graph) cho phép thư mục chia sẻ thư mục con và tập tin (Hình 4.8) Cùng tập tin và thư mục con có thể ở trong hai thư mục khác nhau. Một đồ thị không chứa chu trình là trường hợp tổng quát của cơ chế thư mục có cấu trúc cây.



Hình 4.8 Hệ thống thư mục theo cấu trúc đồ thị không chứa chu trình

Một tập tin (hay thư mục) được chia sẻ không giống như hai bản sao của một tập tin. Với hai bản sao, mỗi người lập trình có thể thích hiện thị bản sao hơn bản gốc, nhưng nếu một người lập trình thay đổi nội dung tập tin, những thay đổi sẽ không

xuất hiện trong bản sao của người còn lại. Với một tập tin được chia sẻ, chỉ một tập tin thực sự tồn tại vì thế bất cứ sự thay đổi được thực hiện bởi một người này lập tức nhìn thấy bởi người dùng khác. Việc chia sẻ là rất quan trọng cho các thư mục con; một tập tin mới được tạo bởi người này sẽ tự động xuất hiện trong tất cả thư mục con được chia sẻ.

Khi nhiều người đang làm việc như một nhóm, tất cả tập tin họ muốn chia sẻ có thể đặt vào một thư mục. Các UFD của tất cả thành viên trong nhóm chứa thư mục của tập tin được chia sẻ như một thư mục con. Ngay cả khi có một người dùng, tổ chức tập tin của người dùng này yêu cầu rằng một số tập tin được đặt vào các thư mục con khác nhau. Thí dụ, một chương trình được viết cho một dự án nên đặt trong thư mục của tất cả chương trình và trong thư mục cho dự án đó.

Các tập tin và thư mục con được chia sẻ có thể được cài đặt trong nhiều cách. Cách thông dụng nhất được UNIX dùng là tạo một mục từ thư mục được gọi là liên kết. Một liên kết là một con trỏ chỉ tới một tập tin hay thư mục con khác. Thí dụ, một liên kết có thể được cài đặt như tên đường dẫn tuyệt đối hay tương đối. Khi một tham chiếu tới tập tin được thực hiện, chúng ta tìm kiếm thư mục. Nếu mục từ thư mục được đánh dấu như một liên kết thì tên tập tin thật sự (hay thư mục) được cho. Chúng ta phân giải liên kết bằng cách sử dụng tên đường dẫn để định vị tập tin thật sự. Những liên kết được xác định dễ dàng bởi định dạng trong mục từ thư mục và được định rõ bằng các con trỏ gián tiếp. Hệ điều hành bỏ qua các liên kết này khi duyệt qua cây thư mục để lưu giữ cấu trúc không chứa chu trình của hệ thống.

Một tiếp cận khác để cài đặt các tập tin được chia sẻ là nhân bản tất cả thông tin về chúng trong cả hai thư mục chia sẻ. Do đó, cả hai mục từ là giống hệt nhau. Một liên kết rất khác từ mục từ thư mục gốc. Tuy nhiên, nhân bản mục từ thư mục làm cho bản gốc và bản sao không khác nhau. Một vấn đề chính với nhân bản mục từ thư mục là duy trì tính không đổi nếu tập tin bị sửa đổi.

Một cấu trúc thư mục đồ thị không chứa chu trình linh hoạt hơn cấu trúc cây đơn giản nhưng nó cũng phức tạp hơn. Một số vấn đề phải được xem xét cẩn thận. Một tập tin có nhiều tên đường dẫn tuyệt đối. Do đó, các tên tập tin khác nhau có thể tham chiếu tới cùng một tập tin. Trường hợp này là tương tự như vấn đề bí danh cho các ngôn ngữ lập trình. Nếu chúng ta đang cố gắng duyệt toàn bộ hệ thống tập tin-để tìm một tập tin, để tập hợp các thông tin thống kê trên tất cả tập tin, hay chép tất cả tập tin tới thiết bị lưu dự phòng-vấn đề này trở nên lớn vì chúng ta không muốn duyệt các cấu trúc chia sẻ nhiều hơn một lần.

Một vấn đề khác liên quan đến việc xoá. Không gian được cấp phát tới tập tin được chia sẻ bị thu hồi và sử dụng lại khi nào? một khả năng là xoá bỏ tập tin bất cứ khi nào người dùng xoá nó, nhưng hoạt động này để lại con trỏ chỉ tới một tập tin không tồn tại. Trong trường hợp xấu hơn, nếu các con trỏ tập tin còn lại chứa địa chỉ đĩa thật sự và không gian được dùng lại sau đó cho các tập tin khác, các con trỏ này có thể chỉ vào phần giữa của tập tin khác.

Trong một hệ thống mà việc chia sẻ được cài đặt bởi liên kết biểu tượng, trường hợp này dễ dàng quản lý hơn. Việc xoá một liên kết không cần tác động tập tin nguồn, chỉ liên kết bị xoá. Nếu chính tập tin bị xoá, không gian cho tập tin này

được thu hồi, để lại các liên kết chơi vơi. Chúng ta có thể tìm các liên kết này và xoá chúng, nhưng nếu không có danh sách các liên kết được nối kết, việc tìm kiếm này sẽ tốn rất nhiều chi phí. Một cách khác, chúng ta có thể để lại các liên kết này cho đến khi nó được truy xuất. Tại thời điểm đó, chúng ta xác định rằng tập tin của tên được cho bởi liên kết không tồn tại và có thể bị lỗi để phục hồi tên liên kết; truy xuất này được đối xử như bất cứ tên tập tin không hợp lệ khác. Trong trường hợp UNIX, các liên kết biểu tượng được để lại khi một tập tin bị xoá và nó cho người dùng nhận thấy rằng tập tin nguồn đã mất hay bị thay thế. Microsoft Windows (tất cả ấn bản) dùng cùng tiếp cận.

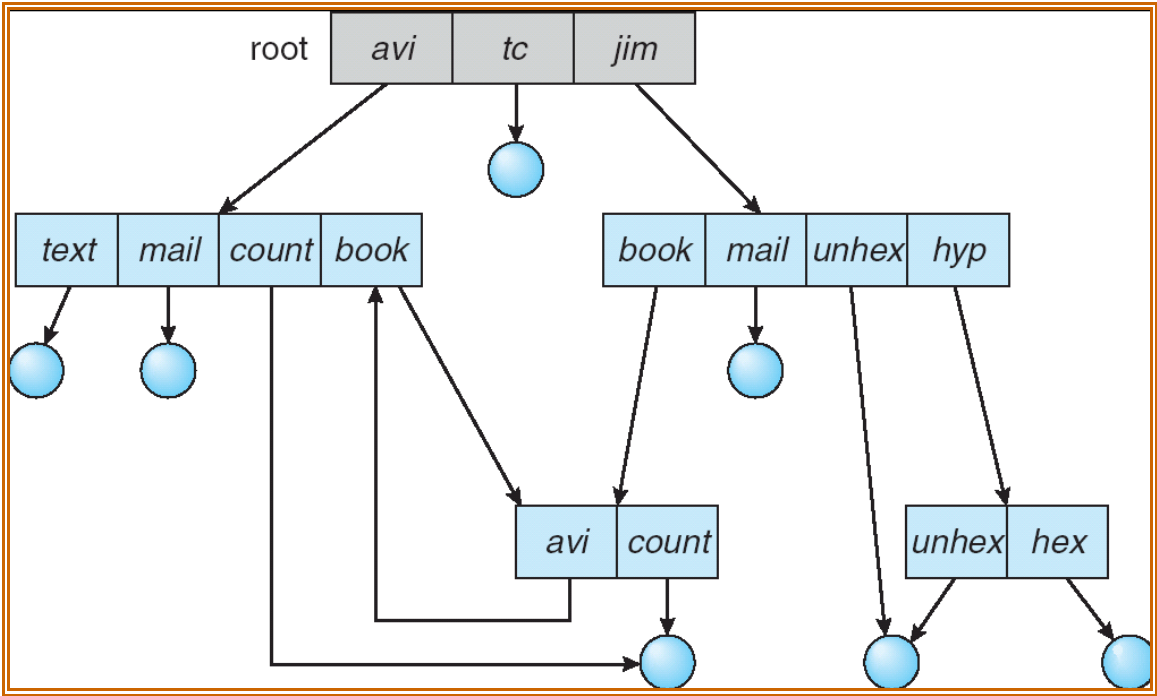
Một tiếp cận khác đối với việc xoá là giữ lại tập tin cho tới khi tất cả tham chiếu tới nó bị xoá. Để cài đặt tiếp cận này, chúng ta phải có một số cơ chế để xác định rằng tham chiếu cuối cùng tới tập tin bị xoá. Chúng ta giữ danh sách của tất cả tham chiếu tới một tập tin (các mục từ thư mục hay các liên kết biểu tượng). Khi một liên kết hay bản sao của mục từ thư mục được thiết lập, một mục từ mới được thêm tới danh sách tham chiếu tập tin. Khi một mục từ thư mục hay liên kết bị xoá, chúng ta gỡ bỏ mục từ của nó trên danh sách. Tập tin này bị xoá khi danh sách tham chiếu tập tin của nó là rỗng.

Trở ngại với tiếp cận này là kích thước của danh sách tham chiếu thay đổi và có thể rất lớn. Tuy nhiên, chúng ta thật sự không cần giữ toàn bộ danh sách—chúng ta chỉ cần giữ số đếm của số tham chiếu. Một liên kết mới hay mục từ thư mục mới sẽ tăng số đếm tham chiếu; xoá một liên kết hay mục từ sẽ giảm số đếm. Khi số đếm là 0, tập tin có thể được xoá; không còn tham chiếu nào tới nó. Hệ điều hành UNIX dùng tiếp cận này cho các liên kết không biểu tượng (hay liên kết cứng), giữ một số đếm tham chiếu trong khối thông tin tập tin (hay inode). Bằng cách ngăn cản hiệu quả nhiều tham chiếu tới các thư mục, chúng ta duy trì cấu trúc đồ thị không chứa chu trình.

Để tránh vấn đề này, một số hệ thống không cho phép thư mục hay liên kết được chia sẻ. Thí dụ, trong MS-DOS, cấu trúc thư mục là một cấu trúc cây hơn là đồ thị không chứa chu trình.

4.1.3.3. Hệ thống thư mục theo dạng đồ thị tổng quát

Một vấn đề lớn trong việc dùng cấu trúc đồ thị không chứa chu trình là đảm bảo rằng không có chu trình trong đồ thị. Nếu chúng ta bắt đầu với thư mục hai cấp và cho phép người dùng tạo thư mục con, một thư mục có cấu trúc cây tạo ra. Dễ thấy rằng thêm các tập tin và thư mục con mới tới một thư mục có cấu trúc cây đã có vẫn bảo đảm tính tự nhiên của cấu trúc cây. Tuy nhiên, khi chúng ta liên kết một thư mục cấu trúc cây đã có, cấu trúc cây bị phá vỡ hình thành một đồ thị đơn giản (hình 4.9).



Hình 4.9 Hệ thống thư mục theo cấu trúc đồ thị tổng quát

Một lợi điểm chính của đồ thị không chứa chu trình là tương đối đơn giản trong giải thuật duyệt đồ thị và xác định khi không có tham chiếu nữa tới tập tin. Chúng ta muốn tránh duyệt các phần được chia sẻ của đồ thị không chứa chu trình hai lần để tăng năng lực. Nếu chúng ta chỉ tìm một thư mục con được chia sẻ cho một tập tin xác định, chúng ta muốn tránh việc tìm kiếm thư mục con đó một lần nữa; tìm kiếm lần hai sẽ lãng phí thời gian.

Nếu các chu trình được cho phép tồn tại trong thư mục, chúng ta muốn tránh tìm kiếm bất cứ thành phần nào hai lần vì tính đúng đắn cũng như năng lực. Một giải thuật được thiết kế nghèo nàn dẫn tới vòng lặp vô tận trên các chu trình. Một giải pháp là giới hạn số lượng thư mục sẽ được truy xuất trong quá trình tìm kiếm.

Một vấn đề tương tự tồn tại khi chúng ta cố gắng xác định khi nào một tập tin có thể bị xoá. Như với cấu trúc thư mục đồ thị không chứa chu trình, một giá trị 0 trong số đếm tham chiếu có nghĩa là không còn tham chiếu nữa tới tập tin hay thư mục và tập tin có thể bị xoá. Tuy nhiên, khi có chu trình tồn tại, số đếm tham chiếu khác 0 ngay cả khi không còn tham chiếu tới thư mục hay tập tin. Sai sót này dẫn tới khả năng tham chiếu chính nó (hay chu trình) trong cấu trúc thư mục. Trong trường hợp này, chúng ta cần dùng cơ chế thu dọn rác (garbage collection) để xác định khi tham chiếu cuối cùng bị xoá và không gian đĩa có thể được cấp phát lại. Thu dọn rác liên quan đến việc duyệt toàn bộ hệ thống tập tin, đánh dấu mọi thứ có thể được truy xuất. Sau đó, duyệt lần hai tập hợp mọi thứ không được đánh dấu trên danh sách không gian trống. Tuy nhiên, thu dọn rác cho một đĩa dựa trên hệ thống tập tin rất mất thời gian và do đó hiếm khi được thực hiện.

Thu dọn rác cần thiết chỉ vì có chu trình trong đồ thị. Do đó, cấu trúc đồ thị không chứa chu trình dễ hơn nhiều. Khó khăn là tránh chu trình khi các liên kết

mới được thêm vào cấu trúc. Chúng ta biết như thế nào và khi nào một liên kết mới sẽ hình thành chu trình? Có nhiều giải thuật phát hiện các chu trình trong đồ thị; tuy nhiên chi phí tính toán cao, đặc biệt khi đồ thị ở trên đĩa lưu trữ. Một giải thuật đơn giản hơn trong trường hợp đặc biệt của thư mục và liên kết là bỏ qua liên kết khi duyệt qua thư mục. Chu trình có thể tránh được và không có chi phí thêm xảy ra.

4.1.4. Bảo vệ

Một hệ thống tập tin bị hỏng còn nguy hiểm hơn máy tính bị hỏng vì những hư hỏng trên thiết bị sẽ ít chi phí hơn là hệ thống tập tin vì nó ảnh hưởng đến các phần mềm trên đó. Hơn nữa hệ thống tập tin không thể chống lại được như hư hỏng do phần cứng gây ra (khả năng tin cậy) và những truy xuất không hợp lý, vì vậy chúng phải cài đặt một số chức năng để bảo vệ. Khả năng tin cậy thường được cung cấp bởi nhân bản các tập tin. Nhiều máy tính có các chương trình hệ thống tự động chép các tập tin trên đĩa tới băng từ tại những khoảng thời gian đều đặn để duy trì một bản sao.

4.1.4.1. Kiểm soát truy xuất

Tiếp cận thông dụng nhất đối với vấn đề bảo vệ là thực hiện truy xuất phụ thuộc định danh của người dùng. Những người dùng khác nhau cần các kiểu truy xuất khác nhau tới một tập tin hay thư mục. Cơ chế thông dụng nhất để cài đặt truy xuất phụ thuộc định danh là gắn với mỗi tập tin và thư mục một danh sách kiểm soát truy xuất (access-control list-ACL) xác định tên người dùng và kiểu truy xuất được phép cho mỗi người dùng. Khi một người dùng yêu cầu truy xuất tới một tập tin cụ thể, hệ điều hành kiểm tra danh sách truy xuất được gắn tới tập tin đó. Nếu người dùng đó được liệt kê cho truy xuất được yêu cầu, truy xuất được phép. Ngược lại, sự vi phạm bảo vệ xảy ra và công việc của người dùng đó bị từ chối truy xuất tới tập tin.

Tiếp cận này có lợi điểm của việc cho phép các phương pháp truy xuất phức tạp. Vấn đề chính với các danh sách truy xuất là chiều dài của nó. Nếu chúng ta muốn cho phép mọi người dùng đọc một tập tin, chúng ta phải liệt kê tất cả người dùng với truy xuất đọc. Kỹ thuật này có hai kết quả không mong muốn:

- Xây dựng một danh sách như thế có thể là một tác vụ dài dòng và không đáng, đặc biệt nếu chúng ta không biết trước danh sách người dùng trong hệ thống.
- Mục từ thư mục, trước đó có kích thước cố định, bây giờ có kích thước thay đổi, dẫn đến việc quản lý không gian phức tạp hơn

Những vấn đề này có thể được giải quyết bởi việc dùng ẩn bản cô đọng của danh sách truy xuất.

Để cô đọng chiều dài của danh sách kiểm soát truy xuất, nhiều hệ thống nhận thấy 3 sự phân cấp người dùng trong nối kết với mỗi tập tin:

- Người sở hữu (Owner): người dùng tạo ra tập tin đó
- Nhóm (Group): tập hợp người dùng đang chia sẻ tập tin và cần truy xuất tương tự là nhóm hay nhóm làm việc
- Người dùng khác (universe): tất cả người dùng còn lại trong hệ thống

Tiếp cận phổ biến gần đây nhất là kết hợp các danh sách kiểm soát truy xuất với người sở hữu, nhóm và cơ chế kiểm soát truy xuất được mô tả ở trên

Để cơ chế này làm việc hợp lý, các quyền và danh sách truy xuất phải được kiểm soát chặt chẽ. Kiểm soát này có thể đạt được trong nhiều cách. Thí dụ, trong hệ thống UNIX, các nhóm có thể được tạo và sửa đổi chỉ bởi người quản lý của tiện ích. Do đó, kiểm soát này đạt được thông qua giao tiếp người dùng.

Với việc phân cấp bảo vệ được giới hạn hơn, chỉ có ba trường được yêu cầu để xác định bảo vệ. Mỗi trường thường là một tập hợp các bit, mỗi trường cho phép hay ngăn chặn truy xuất được gắn với nó. Thí dụ, hệ thống UNIX định nghĩa 3 trường 3 bit-rwx, ở đây r kiểm soát truy xuất đọc, w kiểm soát truy xuất viết, x kiểm soát truy xuất thực thi. Một trường riêng rẽ được giữ cho người sở hữu, cho nhóm tập tin và cho tất cả người dùng khác. Trong cơ chế này, 9 bits trên tập tin được yêu cầu để ghi lại thông tin bảo vệ.

4.1.4.2. 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 tức 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.

4.1.4.3. Backup

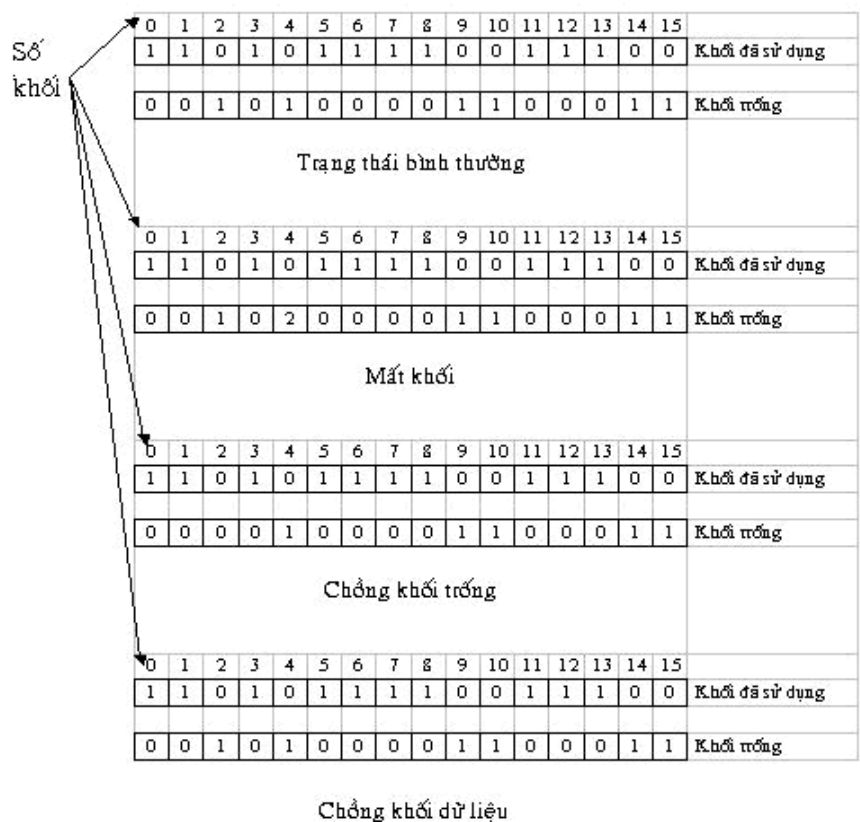
Mặc dù có các chiến lược quản lý các khối hỏng, nhưng một công việc hết sức quan trọng là phải backup tập tin thường xuyên.

Tập tin trên đĩa mềm được backup bằng cách chép lại toàn bộ qua một đĩa khác. Dữ liệu trên đĩa cứng nhỏ thì được backup trên các băng từ.

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

4.1.4.4. Tính không đổi của hệ thống tập tin

Một vấn đề nữa về độ an toàn là **tính không đổi**. Khi truy xuất một tập tin, trong quá trình thực hiện, nếu có xảy ra những sự cố làm hệ thống ngừng hoạt động đột ngột, lúc đó hàng loạt thông tin chưa được cập nhật lên đĩa. Vì vậy mỗi lần khởi động ,hệ thống sẽ thực hiện việc kiểm tra trên hai phần khối và tập tin. Việc kiểm tra thực hiện, khi phát hiện ra lỗi sẽ tiến hành sửa chữa cho các trường hợp cụ thể:



Hình 4.10 Trạng thái của hệ thống tập tin

4.1.4.5. Các tiếp cận bảo vệ khác

Một tiếp cận khác cho vấn đề bảo vệ là gắn mật khẩu với mỗi tập tin. Giống như truy xuất tới hệ thống máy tính thường được kiểm soát bởi một mật khẩu, truy xuất tới mỗi tập tin có thể được kiểm soát bởi một mật khẩu. Nếu các mật khẩu được chọn một cách ngẫu nhiên và thường được thay đổi thì cơ chế này có thể hiệu quả trong truy xuất có giới hạn tới tập tin cho những người dùng biết mật khẩu. Tuy nhiên, cơ chế này có nhiều nhược điểm. Thứ nhất, số lượng mật khẩu mà người dùng cần nhớ quá nhiều, làm cho cơ chế này không thực tế. Thứ hai, nếu chỉ một mật khẩu được dùng cho tất cả tập tin thì một khi nó bị phát hiện tất cả tập tin có thể truy xuất. Một số hệ thống cho phép người dùng gắn một mật khẩu tới một thư mục con hơn là với từng tập tin riêng rẽ để giải quyết vấn đề này. Thứ ba, thường chỉ một mật khẩu gắn với tất cả tập tin người dùng. Do đó, bảo vệ dựa trên cơ sở tất cả hay không có gì (all-or-nothing). Để cung cấp sự bảo vệ trên cấp độ chi tiết hơn chúng ta phải dùng nhiều mật khẩu.

4.1.5. Tập tin chia sẻ

4.1.5.1. Nhiều người dùng

Khi hệ điều hành cung cấp nhiều người dùng, các vấn đề chia sẻ tập tin, đặt tên tập tin, và bảo vệ tập tin trở nên quan trọng. Đối với một cấu trúc thư mục cho phép các tập tin được chia sẻ bởi nhiều người dùng, hệ thống phải dàn xếp việc chia sẻ tập tin. Mặc định, hệ thống có thể cho phép một người dùng truy xuất các tập tin

của người dùng khác hay nó yêu cầu rằng một người dùng gán quyền truy xuất cụ thể tới các tập tin.

Để cài đặt chia sẻ và bảo vệ, hệ thống phải duy trì nhiều thuộc tính tập tin và thư mục hơn trên hệ thống đơn người dùng. Mặc dù, có nhiều tiếp cận cho chủ đề này, hầu hết các hệ thống đưa ra khái niệm người sở hữu (owner) và nhóm (group) tập tin/thư mục. Người sở hữu là người dùng có thể thay đổi các thuộc tính, gán truy xuất, và có hầu hết điều khiển qua tập tin và thư mục. Thuộc tính nhóm của tập tin được dùng để định nghĩa tập hợp con các người dùng có thể chia sẻ truy xuất tới tập tin.

4.1.5.2. Hệ thống tập tin ở xa

Sự phát triển của mạng cho phép giao tiếp giữa các máy tính ở xa. Mạng cho phép chia sẻ các tài nguyên trải rộng trong một khu hay thậm chí khắp thế giới. Một tài nguyên quan trọng để chia sẻ là dữ liệu ở dạng tập tin. Thông quan sự phát triển mạng và công nghệ tập tin, phương pháp chia sẻ tập tin thay đổi. Trong phương pháp đầu tiên được cài đặt, người dùng truyền tập tin giữa các máy tính bằng chương trình gọi là ftp. Phương pháp quan trọng thứ hai là hệ thống tập tin phân tán (distributed file system-DFS) trong đó, các thư mục ở xa có thể được nhìn thấy từ máy cục bộ. Trong một số cách, phương pháp thứ ba, World Wide Web là một sự trở lại của phương pháp đầu tiên. Một trình duyệt được yêu cầu để đạt được truy xuất các tập tin từ xa và các thao tác riêng biệt được dùng để truyền tập tin.

4.2. Cài đặt hệ thống tập tin

Trong phần trước chúng ta thấy rằng, hệ thống tập tin cung cấp cơ chế cho việc lưu trữ trực tuyến (on-line storage) và truy xuất tới nội dung tập tin, gồm dữ liệu và chương trình. Hệ thống tập tin định vị vĩnh viễn trên thiết bị lưu trữ phụ. Các thiết bị này được thiết kế để quản lý lượng lớn thông tin không thay đổi.

Phần này tập trung chủ yếu với những vấn đề xoay quanh việc lưu trữ tập tin và truy xuất trên các thiết bị lưu trữ phụ. Chúng ta khám phá các cách để xây dựng cấu trúc sử dụng tập tin, cấp phát không gian đĩa và phục hồi không gian trống để ghi lại vị trí dữ liệu và để giao tiếp với các phần khác của hệ điều hành tới thiết bị lưu trữ phụ.

4.2.1. Cấu trúc hệ thống tập tin

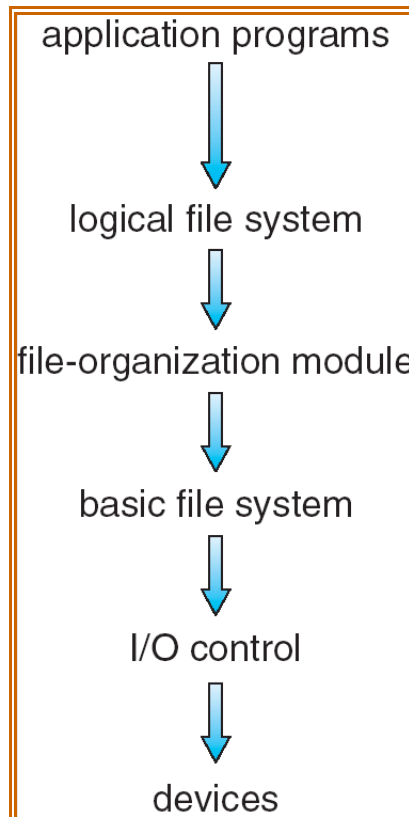
Đĩa cung cấp số lượng thiết bị lưu trữ phụ mà trên đó hệ thống tập tin được duy trì. Có hai đặc điểm làm đĩa trở thành phương tiện tiện dụng cho việc lưu trữ nhiều tập tin:

- Chúng có thể được viết lại bằng cách thay thế; có thể đọc một khối từ đĩa, sửa một khối và viết nó ngược trở lại đĩa trong cùng vị trí.
- Chúng có thể được truy xuất trực tiếp bất cứ khối thông tin nào trên đĩa.

Để cải tiến tính hiệu quả nhập/xuất, thay vì chuyển một byte tại một thời điểm, nhập/xuất chuyển giữa bộ nhớ và đĩa được thực hiện trong đơn vị khối. Mỗi khối là một hay nhiều cung từ (sector). Phụ thuộc ổ đĩa, các cung từ biến đổi từ 32 bytes tới 4096 bytes; thường là 512 bytes.

Để cung cấp việc truy xuất hiệu quả và tiện dụng tới đĩa, hệ điều hành áp đặt một hay nhiều hệ thống tập tin để cho phép dữ liệu được lưu trữ, định vị và truy xuất lại dễ dàng. Một hệ thống tập tin đặt ra hai vấn đề thiết kế rất khác nhau. Vấn đề đầu tiên là định nghĩa hệ thống tập tin nên quan tâm đến người dùng như thế nào. Tác vụ này liên quan đến việc định nghĩa một tập tin và thuộc tính của nó, các thao tác được phép trên một tập tin và các giải thuật và cấu trúc cho việc tổ chức tập tin. Vấn đề thứ hai là tạo giải thuật và cấu trúc dữ liệu để ánh xạ hệ thống tập tin logic vào các thiết bị lưu trữ phụ.

Hệ thống tập tin thường được tạo thành từ nhiều cấp khác nhau. Cấu trúc được hiển thị trong hình 4.11 là một thí dụ của thiết kế phân cấp. Mỗi cấp trong thiết kế dùng các đặc điểm của cấp thấp hơn để tạo các đặc điểm mới cho việc sử dụng bởi cấp cao hơn.



Hình 4.11 Hệ thống tập tin phân tầng

- Điều khiển nhập/xuất (I/O control): là cấp thấp nhất chứa các trình điều khiển thiết bị và các bộ quản lý ngắt để chuyển thông tin giữa bộ nhớ chính và hệ thống đĩa. Trình điều khiển thiết bị thường viết các mẫu bit xác định tới các vị trí trong bộ nhớ của bộ điều khiển nhập/xuất để báo với bộ điều khiển vị trí trên thiết bị nào và hoạt động gì xảy ra.

- Hệ thống tập tin cơ bản (basic file system) chỉ cần phát ra các lệnh thông thường tới các trình điều khiển thiết bị tương ứng để đọc và viết các khối vật lý trên đĩa. Mỗi khối vật lý được xác định bởi địa chỉ đĩa (thí dụ, đĩa 1, cylinder 73, track 2, sector 10).

- Module tổ chức tập tin (file-organization module) biết các tập tin và các khối luận lý cũng như các khối vật lý. Bằng cách biết kiểu cấp phát tập tin được dùng và vị trí của tập tin, module tổ chức tập tin có thể dịch các địa chỉ khối luận lý thành các địa chỉ khối vật lý cho hệ thống tập tin cơ bản để truyền. Các khối luận lý của mỗi tập tin được đánh số từ 0 (hay 1) tới N, ngược lại các khối vật lý chứa dữ liệu thường không khớp với các số luận lý vì thế một thao tác dịch được yêu cầu để định vị mỗi khối. Module tổ chức tập tin cũng chứa bộ quản lý không gian trống (free-space manager), mà nó ghi vết các khối không được cấp phát và cung cấp các khối này tới module tổ chức tập tin khi được yêu cầu.

- Hệ thống tập tin luận lý (logical file system) quản lý thông tin siêu dữ liệu (metadata). Metadata chứa tất cả cấu trúc hệ thống tập tin, ngoại trừ dữ liệu thật sự (hay nội dung của các tập tin). Hệ thống tập tin luận lý quản lý cấu trúc thư mục để cung cấp module tổ chức tập tin những thông tin yêu cầu sau đó, được cho tên tập tin ký hiệu. Nó duy trì cấu trúc tập tin bằng khối điều khiển tập tin. Một khối điều khiển tập tin (file control block-FCB) chứa thông tin về tập tin, gồm người sở hữu, quyền và vị trí của nội dung tập tin.

Nhiều hệ thống tập tin được cài đặt hiện nay. Hầu hết hệ điều hành hỗ trợ nhiều hơn một hệ thống tập tin. Mỗi hệ điều hành có hệ thống tập tin dựa trên cơ sở đĩa. UNIX dùng hệ thống tập tin UNIX (UNIX file system-UFS) như là cơ sở. Windows NT hỗ trợ các định dạng tập tin FAT, FAT32 và NTFS cũng như CD-ROM, DVD và các định dạng hệ thống tập tin đĩa mềm. Bằng cách dùng cấu trúc phân cấp cho việc cài đặt hệ thống tập tin, nên nhân bản mã là tối thiểu. Điều khiển nhập/xuất và mã hệ thống tập tin cơ bản có thể được dùng bởi nhiều hệ thống tập tin. Mỗi hệ thống tập tin có hệ thống tập tin luận lý và module tổ chức tập tin của chính nó.

4.2.2. Cài đặt hệ thống tập tin

Nhiều cấu trúc trên đĩa và trên bộ nhớ được dùng để cài đặt một hệ thống tập tin. Các cấu trúc này thay đổi dựa trên hệ điều hành và hệ thống tập tin nhưng có một số nguyên tắc chung được áp dụng. Trên đĩa, hệ thống tập tin chứa thông tin về cách khởi động hệ điều hành được lưu trữ ở đó, tổng số khối, số và vị trí của các khối trống, cấu trúc thư mục, các tập tin riêng biệt.

Các cấu trúc trên đĩa gồm:

- Khối điều khiển khởi động (boot control block) có thể chứa thông tin được yêu cầu bởi hệ thống để khởi động một hệ điều hành từ phân khu đó. Nếu đĩa không chứa hệ điều hành thì khối này là rỗng. Điển hình, nó là khối đầu tiên của đĩa. Trong UFS, khối này được gọi là khối khởi động; trong NTFS, nó là cung khởi động phân khu (partition boot sector).

- Khối điều khiển phân khu (partition control block) chứa chi tiết về phân khu, như số lượng khối trong phân khu, kích thước khối, bộ đếm khối trống và con trỏ khối trống, bộ đếm FCB trống và con trỏ FCB. Trong UFS khối này được gọi là siêu khối (superblock); trong NTFS, nó là bảng tập tin chính (Master File Table)

- Một cấu trúc tập tin được dùng để tổ chức các tập tin

• Một FCB chứa nhiều chi tiết tập tin gồm các quyền tập tin, người sở hữu, kích thước, và vị trí của các khối dữ liệu. Trong UFS khối này được gọi là inode. Trong NTFS, thông tin này được lưu trong Master File Table dùng cấu trúc cơ sở dữ liệu quan hệ với một dòng cho một tập tin.

Thông tin trong bộ nhớ được dùng cho việc quản lý hệ thống tập tin và cải tiến năng lực qua lưu trữ (caching). Các cấu trúc này có thể bao gồm:

- Bảng phân khu trong bộ nhớ chứa thông tin về mỗi phân khu được gắn vào.
- Cấu trúc thư mục trong bộ nhớ quản lý thông tin thư mục của những thư mục vừa được truy xuất. (đối với các thư mục nơi mà các phân khu được gắn vào, nó có thể chứa một con trỏ chỉ tới bảng phân khu.)
- Bảng tập tin đang mở của hệ thống (system-wide open-file table) chứa bản sao của FCB của mỗi tập tin đang mở cũng như các thông tin khác.
- Bảng tập tin đang mở trên quá trình (per-process open-file table) chứa con trỏ chỉ tới mục từ tương ứng trong bảng tập tin đang mở của hệ thống cũng như những thông tin khác.

Để tạo một tập tin mới, một chương trình ứng dụng gọi hệ thống tập tin luận lý. Hệ thống tập tin luận lý biết định dạng của các cấu trúc thư mục. Để tạo một tập tin mới, nó cấp phát một FCB mới, đọc thư mục tương ứng vào bộ nhớ, cập nhật nó với tên tập tin mới và FCB, và viết nó trở lại đĩa. Một FCB điển hình được hiển thị trong hình 4.12.

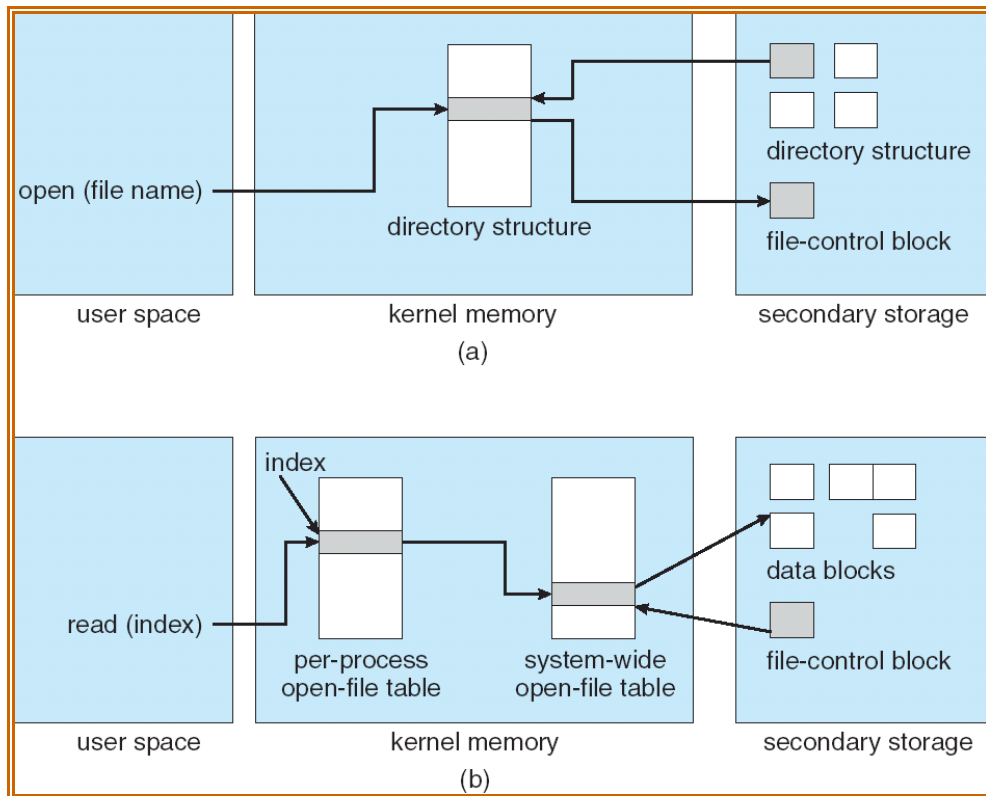
file permissions
file dates (create, access, write)
file owner, group, ACL
file size
file data blocks or pointers to file data blocks

Hình 4.11 Một khối điều khiển tập tin

Một số hệ điều hành như UNIX xem một thư mục như là một tập tin-một tập tin với một trường kiểu hiển thị rằng nó là một thư mục. Các hệ điều hành khác như Windows NT cài đặt các lời gọi hệ thống riêng cho tập tin và thư mục và xem các thư mục như các thực thể tách rời từ các tập tin. Đối với cấu trúc lớn hơn, hệ thống tập tin luận lý có thể gọi module tổ chức tập tin để ánh xạ nhập/xuất thư mục vào

số khối đĩa mà chúng được truyền trên cơ sở hệ thống tập tin và hệ thống điều khiển nhập/xuất. Module tổ chức tập tin cũng cấp phát các khối cho việc lưu trữ dữ liệu của tập tin.

Một tập tin được tạo, nó có thể được dùng cho nhập/xuất. Đầu tiên, nó phải được mở. Lời gọi open truyền tên tập tin tới hệ thống tập tin. Khi một tập tin được mở, cấu trúc thư mục thường được lưu vào bộ nhớ để tăng tốc độ các thao tác thư mục. Một khi tập tin được tìm thấy, FCB được chép vào bảng tập tin đang mở của hệ thống trong bộ nhớ. Bảng này không chỉ chứa FCB mà còn có các mục từ cho số đếm của số quá trình có mở tập tin.



Hình 4.12 Cấu trúc hệ thống trong bộ nhớ. (a) mở tập tin. (b) đọc tập tin.

Tiếp theo, một mục từ được tạo trong bảng tập tin đang mở trên quá trình, với một con trỏ chỉ tới mục từ trong bảng hệ thống tập tin đang mở của hệ thống và một số trường khác. Các trường khác này có thể chứa con trỏ chỉ tới vị trí hiện hành trong tập tin (cho các thao tác read hay write tiếp theo) và chế độ truy xuất trong tập tin được mở. Lời gọi open trả về một con trỏ chỉ tới mục từ tương ứng trong bảng hệ thống tập tin trên quá trình. Sau đó, tất cả thao tác tập tin được thực hiện bằng con trỏ này. Tên tập tin không phải là một phần của bảng tập tin đang mở, hệ thống không dùng nó một khi FCB tương ứng được định vị trên đĩa. Tên được cho đối với mục từ rất đa dạng. Các hệ thống UNIX chỉ tới nó như một bộ mô tả tập tin (file descriptor); Windows 2000 chỉ tới nó như một bộ quản lý tập tin (file handle). Do đó, với điều kiện là tập tin không đóng, tất cả các thao tác tập tin được thực hiện trên bảng tập tin đang mở.

Khi một quá trình đóng tập tin, mục từ trong bảng trên quá trình bị xoá và bộ đếm số lần mở của mục từ hệ thống giảm. Khi tất cả người dùng đóng tập tin, thông tin tập tin được cập nhật sẽ được chép trở lại tới cấu trúc thư mục dựa trên đĩa và mục từ bảng tập tin đang mở bị xoá.

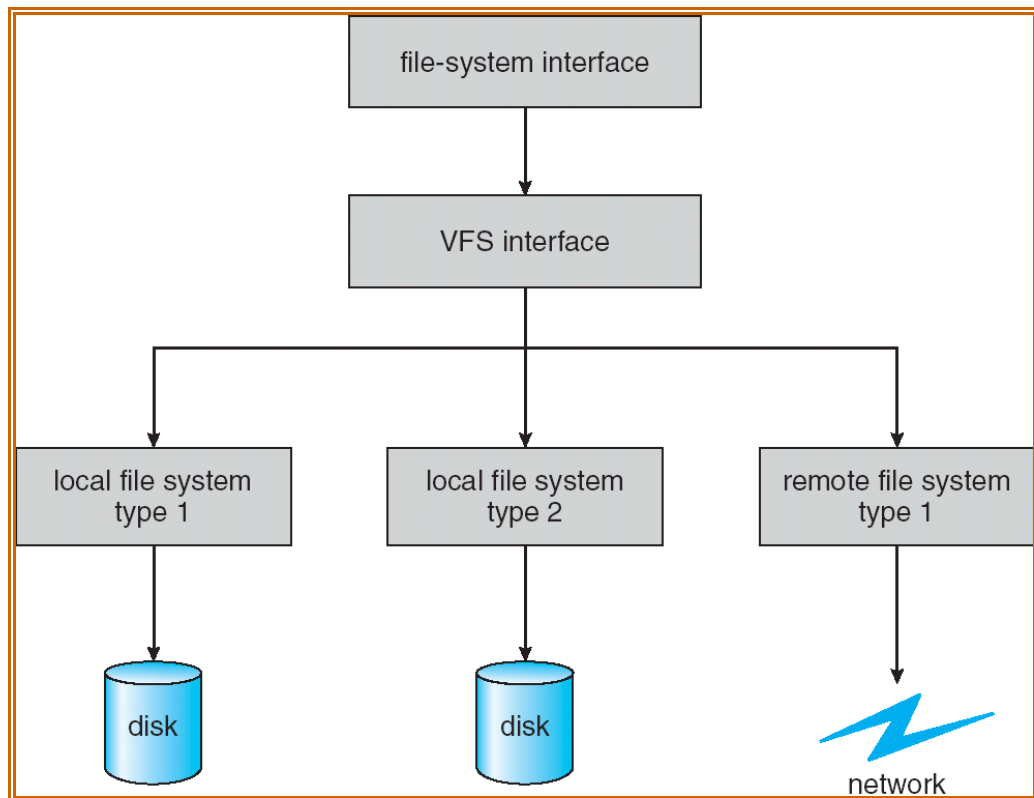
Trong thực tế, lời gọi hệ thống open đầu tiên tìm bảng tập tin đang mở hệ thống để thấy nếu tập tin được sử dụng rồi bởi một quá trình khác. Nếu nó là mục từ bảng tập tin đang mở trên quá trình được tạo để chỉ tới bảng tập tin đang mở hệ thống đã có. Giải thuật này có thể tiết kiệm chi phí khi các tập tin đã mở rồi.

Các cấu trúc điều hành của việc cài đặt hệ thống tập tin được tóm tắt như hình 4.12.

4.2.3. Hệ thống tập tin ảo

Một phương pháp nổi bật cho việc cài đặt nhiều loại hệ thống tập tin là viết các chương trình con thư mục và tập tin cho mỗi loại. Đúng hơn là hầu hết các hệ điều hành, gồm UNIX, dùng các kỹ thuật hướng đối tượng để đơn giản hóa, tổ chức, và module hóa việc cài đặt. Sử dụng các phương pháp này cho phép nhiều loại hệ thống tập tin khác nhau được cài đặt trong cùng cấu trúc, gồm các hệ thống tập tin mạng như NFS. Người dùng có thể truy xuất các tập tin được chứa trong nhiều hệ thống tập tin trên đĩa cục bộ, hay ngay cả trên các hệ thống tập tin sẵn dùng qua mạng.

Các cấu trúc dữ liệu và thủ tục được dùng để cô lập chức năng lời gọi hệ thống cơ bản từ các chi tiết cài đặt. Do đó, cài đặt hệ thống tập tin chứa ba tầng chính; nó được mô tả dưới dạng lưu đồ trong hình 4.13. Tầng đầu tiên là giao diện hệ thống tập tin dựa trên cơ sở lời gọi open, read, write, close và các bộ mô tả tập tin.



Hình 4.13 Hình ảnh dạng lưu đồ của hệ thống tập tin ảo.

Tầng thứ hai được gọi là hệ thống tập tin ảo (Virtual File System-VFS); nó phục vụ hai chức năng quan trọng:

1) Nó tách biệt các thao tác hệ thống tập tin giống nhau từ việc cài đặt bằng cách định nghĩa một giao diện VFS rõ ràng. Nhiều cài đặt cho giao diện VFS có thể cùng tồn tại trên cùng một máy, cho phép truy xuất trong suốt tới các loại hệ thống tập tin khác nhau được gắn cục bộ.

2) VFS dựa trên cấu trúc biểu diễn tập tin, được gọi là vnode, chứa một bộ gắn bằng số (numerical designator) cho tập tin duy nhất qua mạng. (Inode của UNIX là duy nhất chỉ trong một hệ thống tập tin đơn). Tính duy nhất qua mạng được yêu cầu để hỗ trợ các hệ thống tập tin mạng. Nhân duy trì một cấu trúc vnode cho mỗi nút hoạt động (tập tin hay thư mục).

Do đó, VFS có sự khác biệt các tập tin cục bộ với các tập tin ở xa, và các tập tin cục bộ được phân biệt dựa theo loại hệ thống tập tin của chúng.

VFS kích hoạt các thao tác đặc tả hệ thống tập tin để quản lý các yêu cầu cục bộ dựa theo các loại hệ thống tập tin và ngay cả các lời gọi các thủ tục giao thức NFS cho các yêu cầu ở xa. Quản lý tập tin được xây dựng từ vnode tương ứng và được truyền như các tham số tới các thủ tục này. Tầng cài đặt kiểu hệ thống tập tin, hay giao thức hệ thống tập tin ở xa, là tầng dưới cùng của kiến trúc.

4.2.4. Cài đặt thư mục

Chọn giải thuật cấp phát thư mục và quản lý thư mục có tác động lớn đến tính hiệu quả, năng lực, khả năng tin cậy của hệ thống tập tin. Do đó, chúng ta cần hiểu sự thoả hiệp liên quan trong các giải thuật này.

4.2.4.1. Danh sách tuyến tính

Phương pháp đơn giản nhất cho việc cài đặt thư mục là dùng một danh sách tuyến tính chứa tên tập tin với con trỏ chỉ tới các khối dữ liệu. một danh sách tuyến tính với các mục từ thư mục yêu cầu tìm kiếm tuyến tính để xác định một mục từ cụ thể. Phương pháp này đơn giản để lập trình nhưng mất nhiều thời gian để thực thi.

- Để tạo tập tin mới, trước tiên chúng ta phải tìm thư mục để đảm bảo rằng không có tập tin nào tồn tại với cùng một tên. Sau đó, chúng ta thêm một mục từ mới vào cuối thư mục.
- Để xoá một tập tin, chúng ta tìm kiếm thư mục cho tập tin được xác định bởi tên, sau đó giải phóng không gian được cấp phát tới nó.
- Để dùng lại mục từ thư mục, chúng ta có thể thực hiện một vài bước. Chúng ta có thể đánh dấu mục từ như không được dùng (bằng cách gán nó một tên đặc biệt, như một tên trống hay với một bit xác định trạng thái được dùng hoặc không được dùng trong mỗi mục từ), hay chúng ta có thể gán nó tới một danh sách của các mục từ thư mục trống. Một thay đổi thứ ba là chép mục từ cuối cùng trong thư mục vào vị trí trống và giảm chiều dài của thư mục. Một danh sách liên kết có thể được dùng để giảm thời gian xoá một tập tin.

Bất lợi thật sự của danh sách tuyến tính chứa các mục từ thư mục là tìm kiếm tuyến tính để tìm một tập tin. Thông tin thư mục được dùng thường xuyên và người dùng nhận thấy việc truy xuất tới tập tin là chậm. Để khắc phục nhược điểm này, nhiều hệ điều hành cài đặt một vùng lưu trữ phần mềm (software cache) để lưu hầu hết những thông tin thư mục được dùng gần nhất. Một chập dữ liệu được lưu trữ sẽ tránh đọc lại liên tục thông tin từ đĩa. Một danh sách được sắp xếp cho phép tìm kiếm nhị phân và giảm thời gian tìm kiếm trung bình. Tuy nhiên, yêu cầu mà một danh sách phải được sắp xếp có thể phức tạp việc tạo và xoá tập tin vì chúng ta phải di chuyển lượng thông tin liên tục để duy trì một thư mục được xếp thứ tự. Một cấu trúc dữ liệu cây tinh vi hơn như B-tree có thể giúp giải quyết vấn đề. Lợi điểm của danh sách được sắp xếp là liệt kê một thư mục có thứ tự mà không cần một bước sắp xếp riêng.

4.2.4.2. Bảng băm

Một cấu trúc dữ liệu khác thường được dùng cho một thư mục tập tin là bảng băm (hash table). Trong phương pháp này, một danh sách tuyến tính lưu trữ các mục từ thư mục nhưng một cấu trúc bảng băm cũng được dùng. Bảng băm lấy một giá trị được tính từ tên tập tin và trả về con trỏ chỉ tới tên tập tin trong danh sách tuyến tính. Do đó, nó có thể giảm rất lớn thời gian tìm kiếm thư mục. Chèn và xoá cũng tương đối đơn giản mặc dù có thể phát sinh đụng độ-những trường hợp có hai

tên tập tin được băm cùng vị trí. Khó khăn chính với một bảng băm là kích thước của nó thường cố định và phụ thuộc vào hàm băm trên kích thước đó.

Ví dụ: giả sử rằng chúng ta thực hiện một bảng băm thăm dò tuyến tính quản lý 64 mục từ. Hàm băm chuyển các tập tin thành các số nguyên từ 0 tới 63, thí dụ bằng cách dùng số dư của phép chia cho 64. Sau đó, nếu chúng ta cố tạo tập tin thứ 65, chúng ta phải mở rộng bảng băm từ 64 mục-tới 128 mục từ. Kết quả là chúng ta cần hàm băm mới phải ánh xạ tới dãy 0-127 và chúng ta phải sắp xếp lại các mục từ thư mục đã có để phản ánh giá trị hàm băm mới. Một cách khác, một bảng băm vòng có thể được dùng. Mỗi mục từ băm có thể là danh sách liên kết thay vì chỉ một giá trị riêng và chúng ta có thể giải quyết các đụng độ bằng cách thêm mục từ mới vào danh sách liên kết. Tìm kiếm có thể chậm vì tìm kiếm một tên có thể yêu cầu từ bước thông qua một danh sách liên kết của các mục từ bảng đụng độ; nhưng điều này vẫn nhanh hơn tìm kiếm tuyến tính qua toàn thư mục.

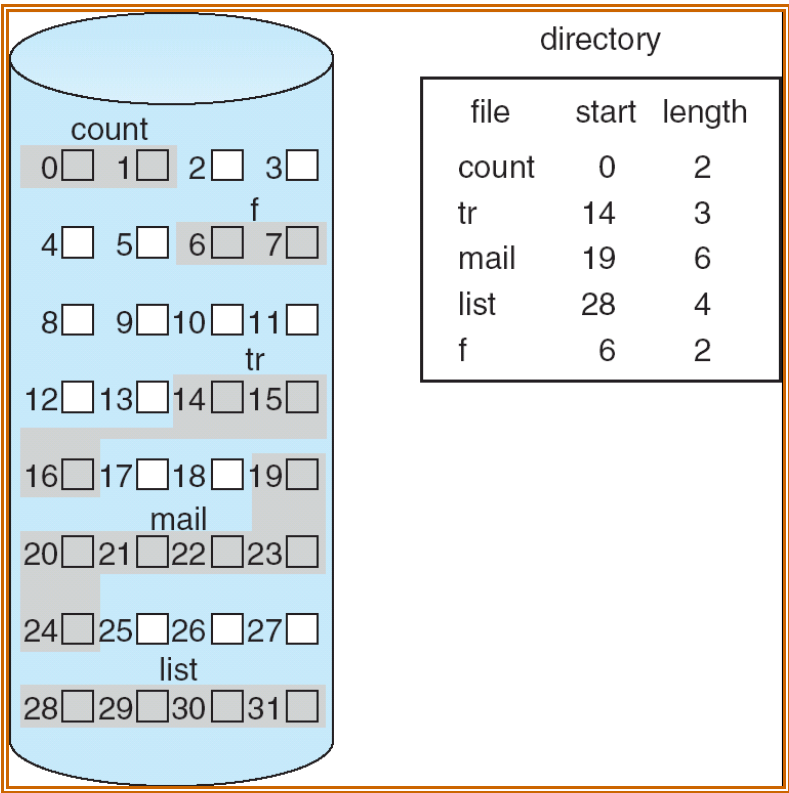
4.2.5. Các phương pháp cấp phát

Tính tự nhiên của truy xuất trực tiếp đĩa cho phép chúng ta khả năng linh hoạt trong việc cài đặt tập tin. Trong hầu hết mọi trường hợp, nhiều tập tin sẽ được lưu trên cùng đĩa. Vấn đề chính là không gian cấp phát tới các tập tin này như thế nào để mà không gian đĩa được sử dụng hiệu quả và các tập tin có thể được truy xuất nhanh chóng. Ba phương pháp quan trọng cho việc cấp phát không gian đĩa được sử dụng rộng rãi: cấp phát kê, liên kết và chỉ mục. Mỗi phương pháp có ưu và nhược điểm. Một số hệ thống hỗ trợ cả ba. Thông dụng hơn, một hệ thống sẽ dùng một phương pháp cụ thể cho tất cả tập tin.

4.2.5.1. Cấp phát liên tục

Phương pháp cấp phát liên tục yêu cầu mỗi tập tin chiếm một tập hợp các khối liên tục nhau trên đĩa. Các địa chỉ đĩa định nghĩa một thứ tự tuyến tính trên đĩa. Với thứ tự này, giả sử rằng chỉ một công việc đang truy xuất đĩa, truy xuất khối $b+1$ sau khi khối b không yêu cầu di chuyển trước. Khi di chuyển đầu đọc được yêu cầu (từ cung từ cuối cùng của cylinder tới cung từ đầu tiên của cylinder tiếp theo), nó chỉ di chuyển một rãnh (track). Do đó, số lượng tìm kiếm đĩa được yêu cầu cho truy xuất liên tục tới các tập tin được cấp phát là nhỏ nhất, như là thời gian tìm kiếm khi tìm kiếm cuối cùng được yêu cầu. Hệ điều hành IBM VM/CMS dùng cấp phát liên tục.

Cấp phát liên tục của một tập tin được định nghĩa bởi địa chỉ đĩa và chiều dài (tính bằng đơn vị khối) của khối đầu tiên. Nếu tập tin có n khối và bắt đầu tại khối b thì nó chiếm các khối $b, b+1, b+2, \dots, b+n-1$. Mục từ thư mục cho mỗi tập tin hiển thị địa chỉ của khối bắt đầu và chiều dài của vùng được cấp phát cho tập tin này (hình 4.14).



Hình 4.14 Không gian đĩa được cấp phát liên tục.

Truy xuất một tập tin được cấp phát liên tục rất dễ. Đối với truy xuất tuần tự, hệ thống tập tin nhớ địa chỉ đĩa của khối cuối cùng được tham chiếu và đọc khối tiếp theo. Để truy xuất khối i của tập tin mà bắt đầu tại khối b , chúng ta có thể lập tức truy xuất khối $b+i$. Do đó, cả truy xuất tuần tự và truy xuất trực tiếp có thể được hỗ trợ bởi cấp phát liên tục.

Tuy nhiên, cấp phát liên tục có một số vấn đề. Một khó khăn là tìm không gian cho một tập tin mới. Việc cài đặt của hệ thống quản lý không gian trống xác định tác vụ này được hoàn thành như thế nào. Bất cứ hệ thống quản lý nào có thể được dùng nhưng nhanh chậm khác nhau.

Vấn đề cấp phát không gian liên tục có thể được xem là vấn đề cấp phát lưu trữ động của ứng dụng để thỏa mãn yêu cầu kích thước n từ danh sách các lỗ trống. First fit và best fit là những chiến lược chung nhất được dùng để chọn lỗ trống từ tập hợp các lỗ trống sẵn dùng. Những mô phỏng biểu thị rằng cả hai first fit và best fit hiệu quả hơn worst fit về thời gian và sử dụng không gian lưu trữ. First fit hay best fit đều không phải là giải thuật tốt nhất nhưng thường thì first fit nhanh hơn best fit.

Các giải thuật này gặp phải vấn đề phân mảnh ngoài. Khi các tập tin được cấp phát và xoá, không gian đĩa trống bị chia thành những mảnh nhỏ. Phân mảnh ngoài tồn tại bất cứ khi nào không gian trống được chia thành những đoạn. Nó trở thành một vấn đề khi đoạn liên tục lớn nhất không đủ cho một yêu cầu; lưu trữ được phân thành nhiều mảnh, không mảnh nào đủ lớn để lưu dữ liệu. Phụ thuộc vào tổng

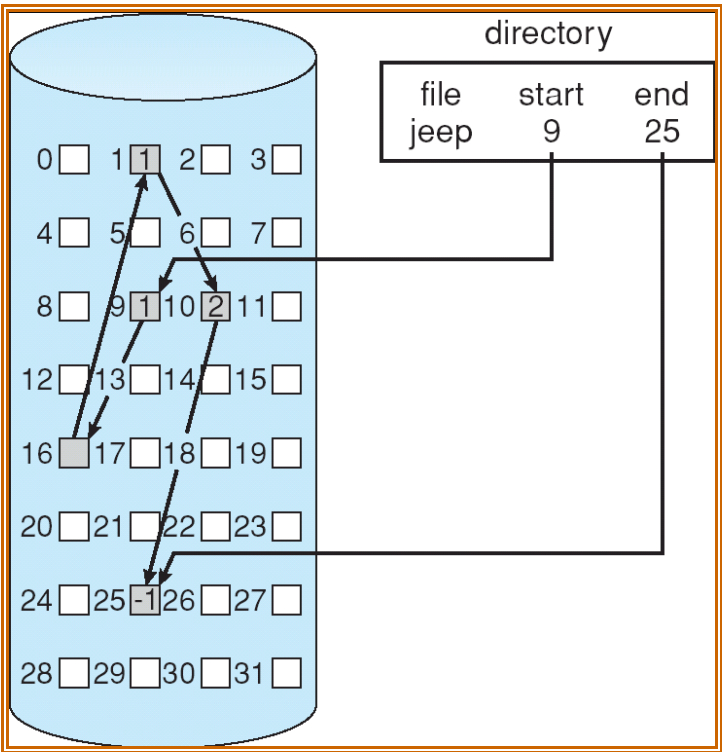
lượng lưu trữ đĩa và kích thước tập tin trung bình, phân mảnh ngoài có thể là vấn đề chính yếu hay phụ.

Một vấn đề khác với cấp phát liên tục là xác định bao nhiêu không gian được yêu cầu cho một tập tin. Khi một tập tin được tạo, toàn bộ không gian nó cần phải được tìm kiếm và được cấp phát. Người tạo (chương trình hay người) biết kích thước tập tin được tạo như thế nào? Trong một số trường hợp việc xác định này tương đối đơn giản (thí dụ chép một tập tin đã có); tuy nhiên, kích thước của tập tin xuất có thể khó để ước lượng.

Để tối thiểu các khó khăn này, một số hệ điều hành dùng một cơ chế cấp phát kè được hiệu chỉnh. Trong cơ chế này đoạn không gian kè được cấp phát trước và sau đó khi lượng không gian đó không đủ lớn, một đoạn không gian kè khác, một đoạn mở rộng (extent), được thêm vào cấp phát ban đầu. Sau đó, vị trí của các khối tập tin được ghi lại như một vị trí và một bộ đếm khối cộng với một liên kết tới khối đầu tiên của đoạn mở rộng tiếp theo. Trên một số hệ thống, người sở hữu tập tin có thể đặt kích thước đoạn mở rộng, nhưng việc đặt này có thể không hiệu quả nếu người sở hữu không đúng. Phân mảnh trong vẫn còn là vấn đề nếu đoạn mở rộng quá lớn và phân mảnh ngoài có thể là vấn đề khi các đoạn mở rộng có kích thước khác nhau được cấp phát và thu hồi.

4.2.5.2. Cấp phát liên kết

Cấp phát liên kết giải quyết vấn đề của cấp phát liên tục. Với cấp phát liên kết, mỗi tập tin là một danh sách các khối đĩa được liên kết; các khối đĩa có thể được phân tán khắp nơi trên đĩa. Thư mục chứa một con trỏ chỉ tới khối đầu tiên và các khối cuối cùng của tập tin. Thí dụ, một tập tin có 5 khối có thể bắt đầu tại khối số 9, tiếp tục là khối 16, sau đó khối 1, khối 10 và cuối cùng khối 25 (như hình 4.15). Mỗi khối chứa một con trỏ chỉ tới khối kế tiếp. Các con trỏ này không được làm sẵn dùng cho người dùng. Do đó, nếu mỗi khối là 512 bytes, và địa chỉ đĩa (con trỏ) yêu cầu 4 bytes thì phần chứa dữ liệu của khối là 508 bytes.



Hình 4.15 Không gian đĩa được cấp phát liên kết.

Để tạo một tập tin mới, chúng ta đơn giản tạo một mục từ mới trong thư mục. Với cấp phát liên kết, mỗi mục từ thư mục có một con trỏ chỉ tới khối đĩa đầu tiên của tập tin. Con trỏ này được khởi tạo tới *nil* (giá trị con trỏ cuối danh sách) để chỉ một tập tin rỗng. Trường kích thước cũng được đặt tới 0. Một thao tác viết tới tập tin làm một khối trống được tìm thấy bằng hệ thống quản lý không gian trống, sau đó khối mới này được viết tới và được liên kết tới cuối tập tin. Để đọc một tập tin, chúng ta đơn giản đọc các khối bằng cách lần theo các con trỏ từ khối này tới khối khác.

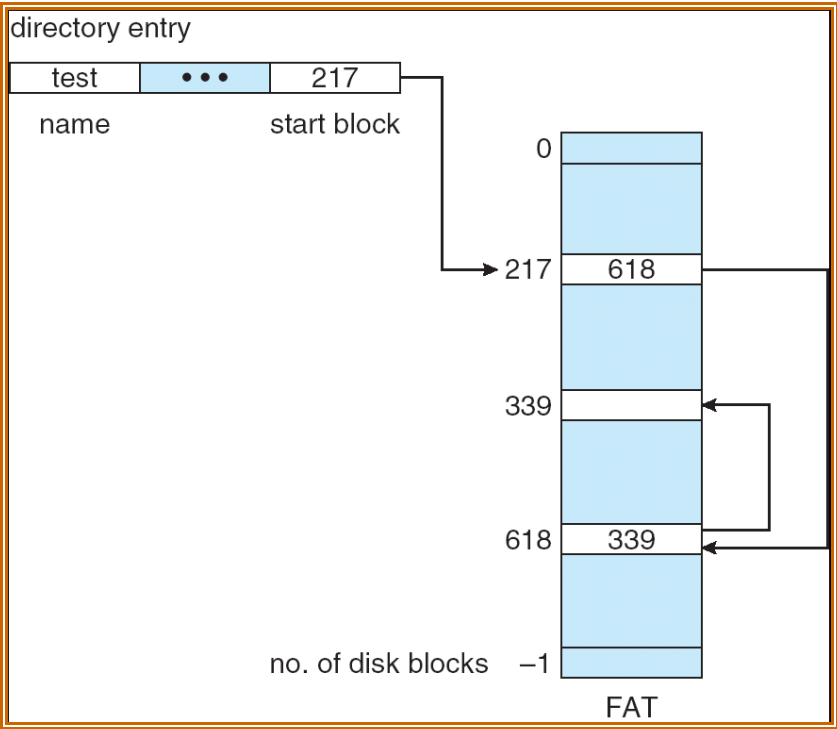
Không có sự phân mảnh ngoài với cấp phát liên kết, và bất cứ khối trống trên danh sách không gian trống có thể được dùng để thỏa mãn yêu cầu. Kích thước của một tập tin không cần được khai báo khi tập tin đó được tạo. Một tập tin có thể tiếp tục lớn lên với điều kiện là các khối trống sẵn có. Do đó, nó không bao giờ cần thiết để hợp nhất không gian trống.

Tuy nhiên, cấp phát liên kết có một vài nhược điểm. Vấn đề chủ yếu là nó có thể được dùng hiệu quả chỉ cho các tập tin truy xuất tuần tự. Để tìm khối thứ *i* của tập tin, chúng ta phải bắt đầu tại điểm bắt đầu của tập tin đó, và lần theo con trỏ cho đến khi chúng ta nhận được khối thứ *i*. Mỗi truy xuất tới con trỏ yêu cầu một thao tác đọc đĩa, và đôi khi là một tìm kiếm đĩa. Do đó, nó không đủ hỗ trợ một khả năng truy xuất trực tiếp cho các tập tin cấp phát liên kết.

Một nhược điểm khác của cấp phát liên kết là không gian được yêu cầu cho các con trỏ. Nếu một con trỏ yêu cầu 4 bytes của khối 512 bytes thì 0.77% của đĩa được dùng cho các con trỏ thay vì là thông tin.

Một giải pháp thông thường để giải quyết vấn đề này là tập hợp các khối vào các **nhóm** (clusters) và cấp phát các nhóm hơn là các khối. Ví dụ, hệ thống tập tin có thể định nghĩa nhóm gồm 4 khối và thao tác trên đĩa chỉ trong đơn vị nhóm thì các con trỏ dùng % nhỏ hơn của không gian của tập tin. Phương pháp này cho phép ánh xạ khối luận lý tới vật lý vẫn còn đơn giản, nhưng cải tiến thông lượng đĩa và giảm không gian được yêu cầu cho cấp phát khối và quản lý danh sách trống. Chi phí của tiếp cận này là tăng phân mảnh trong vì nhiều không gian hơn bị lãng phí nếu một nhóm chỉ đầy một phần hơn là một khối đầy một phần. Các nhóm có thể được dùng để cải tiến thời gian truy xuất đĩa cho nhiều giải thuật khác nhau vì thế chúng được dùng trong hầu hết các hệ điều hành.

Một vấn đề khác của cấp phát liên kết là khả năng tin cậy. Vì các tập tin được liên kết với nhau bởi các con trỏ được phân tán khắp đĩa, xem xét điều gì xảy ra nếu một con trỏ bị mất hay bị phá hỏng. Một con bọ (bug) trong phần mềm hệ điều hành hay lỗi phần cứng đĩa có thể dẫn tới việc chọn con trỏ sai. Lỗi này có thể dẫn tới việc liên kết vào danh sách không gian trống hay vào một tập tin khác. Các giải pháp một phần là dùng các danh sách liên kết đôi hay lưu tên tập tin và số khối tương đối trong mỗi khối; tuy nhiên, các cơ chế này yêu cầu nhiều chi phí hơn cho mỗi tập tin.



Hình 4.16 Bảng định vị file.

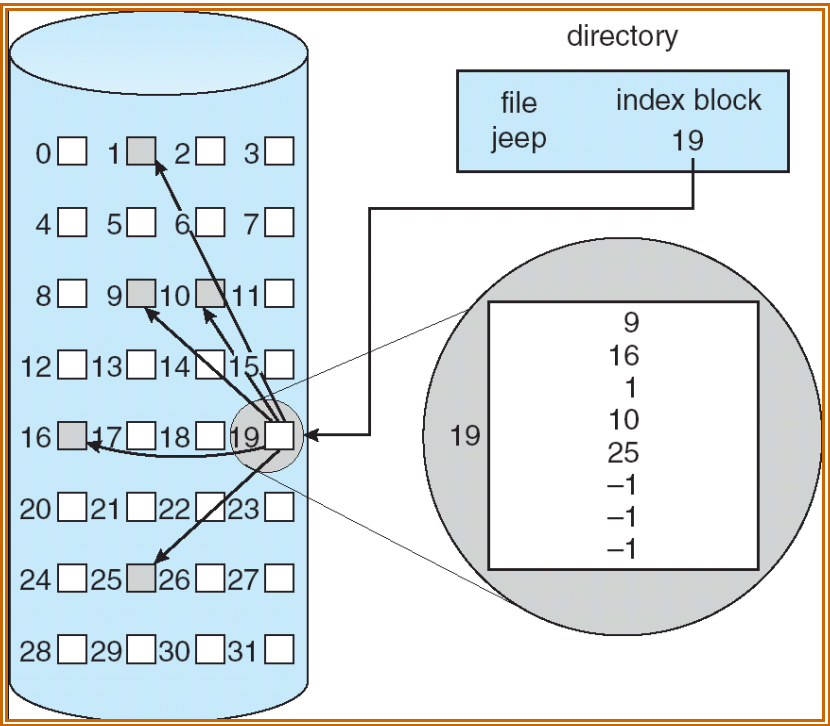
Một thay đổi quan trọng trên phương pháp cấp phát liên kết là dùng bảng định vị tập tin (file allocation table-FAT). Điều này đơn giản nhưng là phương pháp cấp phát không gian đĩa hiệu quả được dùng bởi hệ điều hành MS-DOS và OS/2. Một phần đĩa tại phần bắt đầu của mỗi phân khu được thiết lập để chứa bảng này. Bảng này có một mục từ cho mỗi khối đĩa và được lập chỉ mục bởi khối đĩa. FAT được dùng nhiều như là một danh sách liên kết. Mục từ thư mục chứa số khối của khối

đầu tiên trong tập tin. Mục từ bảng được lập chỉ mục bởi số khối đó sau đó chứa số khối của khối tiếp theo trong tập tin. Chuỗi này tiếp tục cho đến khi khối cuối cùng, có giá trị cuối tập tin đặc biệt như mục từ bảng. Các khối không được dùng được hiển thị bởi giá trị bằng 0. Cấp phát một khối mới tới một tập tin là một vấn đề đơn giản cho việc tìm mục từ bảng có giá trị 0 đầu tiên và thay thế giá trị kết thúc tập tin trước đó với địa chỉ của khối mới. Sau đó, số 0 được thay thế với giá trị kết thúc tập tin. Một thí dụ minh họa là cấu trúc FAT của hình 4.16 cho một tập tin chứa các khối đĩa 217, 618 và 339.

Cơ chế cấp phát FAT có thể dẫn tới số lượng lớn tìm kiếm đầu đọc đĩa nếu FAT không được lưu trữ trong cache. Đầu đọc đĩa phải di chuyển tới điểm bắt đầu của phân khu để đọc FAT và tìm vị trí khối sau đó di chuyển tới vị trí của chính khối đĩa đó. Trong trường hợp xấu nhất, cả hai di chuyển xảy ra cho mỗi khối đĩa. Lợi điểm là thời gian truy xuất ngẫu nhiên được cải tiến vì đầu đọc đĩa có thể tìm vị trí của bất cứ khối nào bằng cách đọc thông tin trong FAT.

4.2.5.3. Cấp phát chỉ mục

Cấp phát liên kết giải quyết việc phân mảnh ngoài và vấn đề khai báo kích thước của cấp phát liên tục. Tuy nhiên, cấp phát liên kết không hỗ trợ truy xuất trực tiếp hiệu quả vì các con trỏ chỉ tới các khối được phân tán với chính các khối đó qua đĩa và cần được lấy lại trong thứ tự. Cấp phát được lập chỉ mục giải quyết vấn đề này bằng cách mang tất cả con trỏ vào một vị trí: khối chỉ mục (index block).



Hình 4.17 Cấp phát không gian đĩa lập chỉ mục

Mỗi tập tin có khối chỉ mục của chính nó, khối này là một mảng các địa chỉ khối đĩa. Mục từ thứ i trong khối chỉ mục chỉ tới khối i của tập tin. Thư mục chứa địa chỉ của khối chỉ mục (hình 4.17). Để đọc khối i , chúng ta dùng con trỏ trong mục từ

khối chỉ mục để tìm và đọc khối mong muốn. Cơ chế này tương tự như cơ chế phân trang.

Khi một tập tin được tạo, tất cả con trỏ trong khối chỉ mục được đặt tới *nil*. Khi khối thứ *i* được viết đầu tiên, khối được chứa từ bộ quản lý không gian trống và địa chỉ của nó được đặt trong mục từ khối chỉ mục.

Cấp phát được lập chỉ mục hỗ trợ truy xuất trực tiếp, không gặp phải sự phân mảnh ngoài vì bất cứ khối trống trên đĩa có thể đáp ứng yêu cầu thêm không gian.

Cấp phát được lập chỉ mục gặp phải sự lãng phí không gian. Chi phí con trỏ của khối chỉ mục thường lớn hơn chi phí con trỏ của cấp phát liên kết. Xét trường hợp thông thường trong đó chúng ta có một tập tin với chỉ một hoặc hai khối. Với cấp phát liên kết, chúng ta mất không gian của chỉ một con trỏ trên khối (một hay hai con trỏ). Với cấp phát được lập chỉ mục, toàn bộ khối chỉ mục phải được cấp phát thậm chí nếu một hay hai con trỏ là khác *nil*.

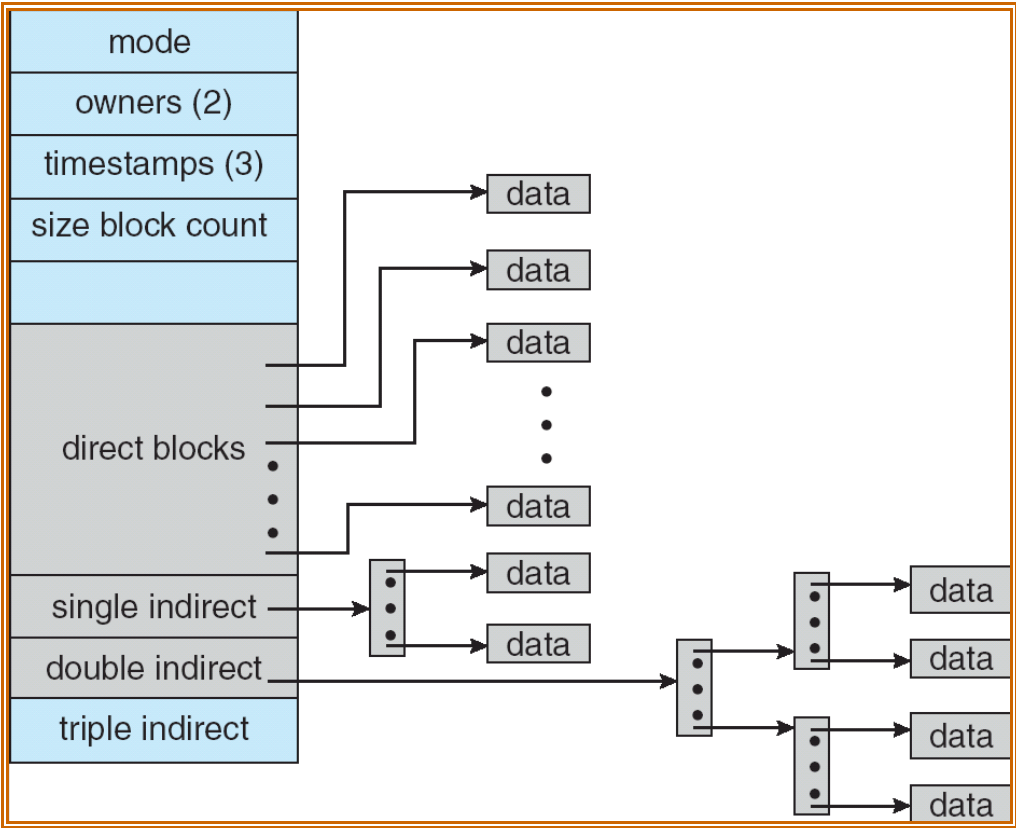
Điểm này sinh ra câu hỏi khối chỉ mục nên lớn bao nhiêu? Mỗi tập tin phải có một khối chỉ mục để mà chúng ta muốn khối chỉ mục nhỏ nhất có thể. Tuy nhiên, nếu khối chỉ mục quá nhỏ nó không thể quản lý đủ các con trỏ cho một tập tin lớn và một cơ chế sẽ phải sẵn có để giải quyết vấn đề này:

- **Cơ chế liên kết** (linked scheme): một khối chỉ mục thường là một khối đĩa. Do đó, nó có thể được đọc và viết trực tiếp bởi chính nó. Để cho phép đối với các tập tin lớn, chúng ta có thể liên kết nhiều khối chỉ mục với nhau. Thí dụ, một khối chỉ mục có thể chứa một header nhỏ cho tên tập tin và một tập hợp của các địa chỉ 100 khối đĩa đầu tiên. Địa chỉ tiếp theo (từ cuối cùng trong khối chỉ mục) là *nil* (đối với một tập tin nhỏ) hay một con trỏ tới khối chỉ mục khác (cho một tập tin lớn)

- **Chỉ mục nhiều cấp** (multilevel index): một biến dạng của biểu diễn liên kết là dùng khối chỉ mục cấp 1 để chỉ tới khối chỉ mục cấp 2. Khối cấp 2 chỉ tới các khối tập tin. Để truy xuất một khối, hệ điều hành dùng chỉ mục cấp 1 để tìm một khối chỉ mục cấp 2 và khối đó tìm khối dữ liệu mong muốn. Tiếp cận này có thể được tiếp tục tới cấp 3 hay cấp 4, tùy thuộc vào kích thước tập tin lớn nhất được mong muốn. Với khối có kích thước 4,096 bytes, chúng ta có thể lưu 1,024 con trỏ 4 bytes trong một khối chỉ mục. Chỉ mục hai cấp cho phép 1,048,576 khối dữ liệu, cho phép tập tin có kích thước tới 4GB.

- **Cơ chế kết hợp** (combined scheme): một biến dạng khác được dùng trong UFS là giữ 15 con trỏ đầu tiên của khối chỉ mục trong inode của tập tin. 12 con trỏ đầu tiên của 15 con trỏ này chỉ tới khối trực tiếp (direct blocks); nghĩa là chúng chứa các địa chỉ của khối mà chứa dữ liệu của tập tin. Do đó, dữ liệu đối với các tập tin nhỏ (không lớn hơn 12 khối) không cần một khối chỉ mục riêng. Nếu kích thước khối là 4 KB, thì tới 48 KB dữ liệu có thể truy xuất trực tiếp. 3 con trỏ tiếp theo chỉ tới các khối gián tiếp (indirect blocks). Con trỏ khối gián tiếp thứ nhất là địa chỉ của khối gián tiếp đơn (single indirect blocks). Khối gián tiếp đơn là một khối chỉ mục không chứa dữ liệu nhưng chứa địa chỉ của các khối chứa dữ liệu. Sau đó, có con trỏ khối gián tiếp đôi (double indirect block) chứa địa chỉ của một khối mà khối này chứa địa chỉ của các khối chứa con trỏ chỉ tới khối dữ liệu thật sự. Con trỏ cuối cùng chứa địa chỉ của khối gián tiếp ba (triple indirect block). Với

phương pháp này, số khối có thể được cấp phát tới một tập tin vượt quá lượng không gian có thể đánh địa chỉ bởi các con trỏ tập tin 4 bytes hay 4 GB. Nhiều cài đặt UNIX gồm Solaris và AIX của IBM hỗ trợ tới 64 bit con trỏ tập tin. Các con trỏ có kích thước này cho phép các tập tin và hệ thống tập tin có kích thước tới terabytes. Một inode được hiển thị trong hình 4.18.



Hình 4.18 Lược đồ kết hợp - UNIX (4K bytes per block)

4.2.6. Quản lý không gian trống

Vì không gian trống là giới hạn nên chúng ta cần dùng lại không gian từ các tập tin bị xoá cho các tập tin mới nếu có thể. Để giữ vết của không gian đĩa trống, hệ thống duy trì một danh sách không gian trống. Danh sách không gian trống ghi lại tất cả khối đĩa trống. Để tạo tập tin, chúng ta tìm trong danh sách không gian trống lượng không gian được yêu cầu và cấp phát không gian đó tới tập tin mới. Sau đó, không gian này được xoá từ danh sách không gian trống. Khi một tập tin bị xoá, không gian đĩa của nó được thêm vào danh sách không gian trống. Mặc dù tên của nó là danh sách nhưng danh sách không gian trống có thể không được cài như một danh sách.

4.2.6.1. Bit vector

Thường thì danh sách không gian trống được cài đặt như một bản đồ bit (bit map) hay một vector bit (bit vector). Mỗi khối được biểu diễn bởi 1 bit. Nếu khối là trống, bit của nó được đặt là 1, nếu khối được cấp phát bit của nó được đặt là 0.

Thí dụ, xét một đĩa khi các khối 2, 3, 4, 5, 8, 9, 10, 11, 12, 13, 17, 18, 25, 26, và 27 là trống và các khối còn lại được cấp phát. Bản đồ bit không gian trống sẽ là:

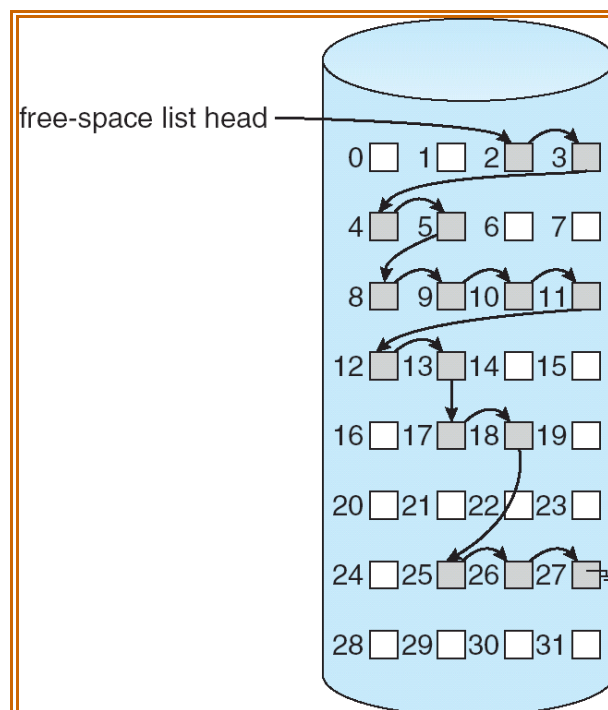
001111001111110001100000011100000...

Lợi điểm chính của tiếp cận này là tính tương đối đơn giản và hiệu quả của nó trong việc tìm khối trống đầu tiên, hay n khối trống tiếp theo trên đĩa.

Một lần nữa, chúng ta thấy các đặc điểm phân cứng định hướng chức năng phần mềm. Tuy nhiên, các vector bit là không đủ trừ khi toàn bộ vector được giữ trong bộ nhớ chính. Giữ nó trong bộ nhớ chính là có thể cho các đĩa nhỏ hơn, như trên các máy vi tính nhưng không thể cho các máy lớn hơn. Một đĩa 1.3 GB với khối 512 bytes sẽ cần một bản đồ bit 332 KB để ghi lại các khối trống. Gộp bốn khối vào một nhóm có thể giảm số này xuống còn 83 KB trên đĩa.

4.2.6.2. Danh sách liên kết

Một tiếp cận khác để quản lý bộ nhớ trống là liên kết tất cả khối trống, giữ một con trỏ tới khối trống đầu tiên trong một vị trí đặc biệt trên đĩa và lưu nó trong bộ nhớ. Khối đầu tiên này chứa con trỏ chỉ tới khối đĩa trống tiếp theo,...Trong thí dụ trên, chúng ta có thể giữ một con trỏ chỉ tới khối 2 như là khối trống đầu tiên. Khối 2 sẽ chứa một con trỏ chỉ tới khối 3, khối này sẽ chỉ tới khối 4,...(hình 4.19).



Hình 4.19 Danh sách không gian trống được liên kết trên đĩa

Tuy nhiên, cơ chế này không hiệu quả do để duyệt danh sách, chúng ta phải đọc mỗi khối, yêu cầu thời gian nhập/xuất đáng kể. Việc duyệt danh sách trống không là hoạt động thường xuyên. Thường thì, hệ điều hành cần một khối trống để mà nó có thể cấp phát khối đó tới một tập tin, vì thế khối đầu tiên trong danh sách trống

được dùng. Phương pháp FAT kết hợp với đếm khối trống thành cấu trúc dữ liệu cấp phát.

4.3. Hệ thống nhập/xuất

Vai trò của hệ điều hành trong nhập/xuất máy tính là quản lý và điều khiển các thao tác nhập/xuất và các thiết bị nhập/xuất. Trong phần này chúng ta sẽ mô tả các khái niệm cơ bản của phần cứng nhập/xuất. Kế đến chúng ta sẽ thảo luận các dịch vụ nhập/xuất được cung cấp bởi hệ điều hành và hiện thân của các dịch vụ này trong giao diện ứng dụng nhập/xuất. Sau đó, chúng ta giải thích hệ điều hành làm cầu nối giữa giao diện phần cứng và giao diện ứng dụng như thế nào. Cuối cùng, chúng ta thảo luận các khía cạnh năng lực của nhập/xuất và các nguyên lý thiết kế hệ điều hành để cải tiến năng lực nhập/xuất.

4.3.1. Các khái niệm cơ bản

Điều khiển các thiết bị được nối kết tới máy tính là mối quan tâm chủ yếu của người thiết kế hệ điều hành. Vì các thiết bị nhập/xuất rất khác nhau về chức năng và tốc độ (xem xét chuột, đĩa cứng, và CD-ROM) nên sự đa dạng về phương pháp là cần thiết để điều khiển chúng. Các phương pháp này hình thành một hệ thống nhập/xuất con (I/O subsystem) của nhân, tách rời phần còn lại của nhân từ sự phức tạp của việc quản lý các thiết bị nhập/xuất.

Công nghệ thiết bị nhập/xuất thể hiện hai xu hướng trái ngược nhau. Xu hướng thứ nhất, chúng ta tăng sự chuẩn hoá phần mềm và giao diện phần cứng. Xu hướng này giúp chúng ta hợp tác những thế hệ thiết bị được cải tiến vào các máy tính và hệ điều hành đã có. Xu hướng thứ hai, chúng ta tăng sự đa dạng của các thiết bị nhập/xuất. Thiết bị mới là rất khác với các thiết bị trước đó đã tạo ra một trở ngại để hợp nhất chúng vào máy tính và hệ điều hành của chúng ta. Trở ngại này được giải quyết bởi sự kết hợp kỹ thuật phần cứng và phần mềm. Các thành phần phần cứng nhập/xuất cơ bản như cổng, bus và bộ điều khiển thiết bị chứa trong một dãy rộng các thiết bị nhập/xuất. Để đóng gói các chi tiết và sự khác biệt của các thiết bị khác nhau, nhân của hệ điều hành được chỉ dẫn để dùng các modules trình điều khiển thiết bị. Các trình điều khiển thiết bị (device driver) hiện diện một giao diện truy xuất thiết bị đồng nhất tới hệ thống con nhập/xuất, như các lời gọi hệ thống cung cấp một giao diện chuẩn giữa ứng dụng và hệ điều hành.

4.3.2. Phần cứng nhập/xuất

Các máy tính điều hành nhiều loại thiết bị. Hầu hết chúng thuộc các chủng loại phổ biến như thiết bị lưu trữ (đĩa, băng từ), thiết bị truyền (card mạng, modem) và thiết bị giao diện người dùng (màn hình, bàn phím, chuột),.... Mặc dù có sự đa dạng về các thiết bị nhập/xuất, nhưng chúng ta chỉ cần hiểu một vài khái niệm như các thiết bị được gán như thế nào và phần mềm có thể điều khiển phần cứng như thế nào.

Một thiết bị giao tiếp với một hệ thống máy tính bằng cách gởi các tín hiệu qua dây cáp hay thậm chí qua không khí. Các thiết bị giao tiếp với máy bằng một điểm nối kết (cổng-port) như cổng tuần tự. Nếu một hay nhiều thiết bị dùng một tập hợp dây dẫn, nối kết được gọi là bus. Một bus là một tập hợp dây dẫn và giao thức được

định nghĩa chặt chẽ để xác định tập hợp thông điệp có thể được gửi qua dây. Trong thuật ngữ điện tử, các thông điệp được truyền bởi các mẫu điện thế điện tử được áp dụng tới các dây dẫn với thời gian được xác định. Khi thiết bị A có một cáp gắn vào thiết bị B, thiết bị B có một cáp gắn vào thiết bị C và thiết bị C gắn vào một cổng máy tính, sự sắp xếp này được gọi là chuỗi nối tiếp. Một chuỗi nối tiếp thường điều hành như một bus.

4.3.2.1. Thăm dò

Giao thức hoàn chỉnh cho việc giao tiếp giữa máy tính và bộ điều khiển rất phức tạp nhưng ký hiệu bắt tay (handshaking) là đơn giản. Chúng ta giải thích bắt tay bằng thí dụ sau. Chúng ta giả sử rằng 2 bits được dùng để hợp tác trong mỗi quan hệ người sản xuất-người tiêu thụ giữa bộ điều khiển và máy chủ. Bộ điều khiển hiển thị trạng thái của nó thông qua bit bận (busy bit) trong thanh ghi trạng thái. Bộ điều khiển đặt bit bận khi nó đang làm việc và xoá bit bận khi nó sẵn sàng nhận lệnh tiếp theo. Máy tính ra tín hiệu mong muốn bằng bit sẵn sàng nhận lệnh (command-ready bit) trong thanh ghi lệnh. Máy tính thiết lập bit sẵn sàng nhận lệnh khi một lệnh sẵn dùng cho bộ điều khiển thực thi. Thí dụ, máy tính viết dữ liệu xuất thông qua một cổng, hợp tác với bộ điều khiển bằng cách bắt tay như sau:

1. Máy tính lặp lại việc đọc bit bận cho tới khi bit này bị xoá
2. Máy tính thiết lập bit viết trong thanh ghi lệnh và viết một byte vào thanh ghi dữ liệu xuất
3. Máy tính đặt bit sẵn sàng nhận lệnh
4. Khi bộ điều khiển nhận thấy rằng bit sẵn sàng nhận lệnh được đặt, nó đặt bit bận
5. Bộ điều khiển đọc thanh ghi lệnh và thấy lệnh viết. Nó đọc thanh ghi xuất dữ liệu để lấy một byte và thực hiện nhập/xuất tới thiết bị.
6. Bộ điều khiển xoá bit sẵn sàng nhận lệnh, xoá bit lỗi trong thanh ghi trạng thái để hiển thị rằng thiết bị nhập/xuất thành công, và xoá bit bận để hiển thị rằng nó được kết thúc.

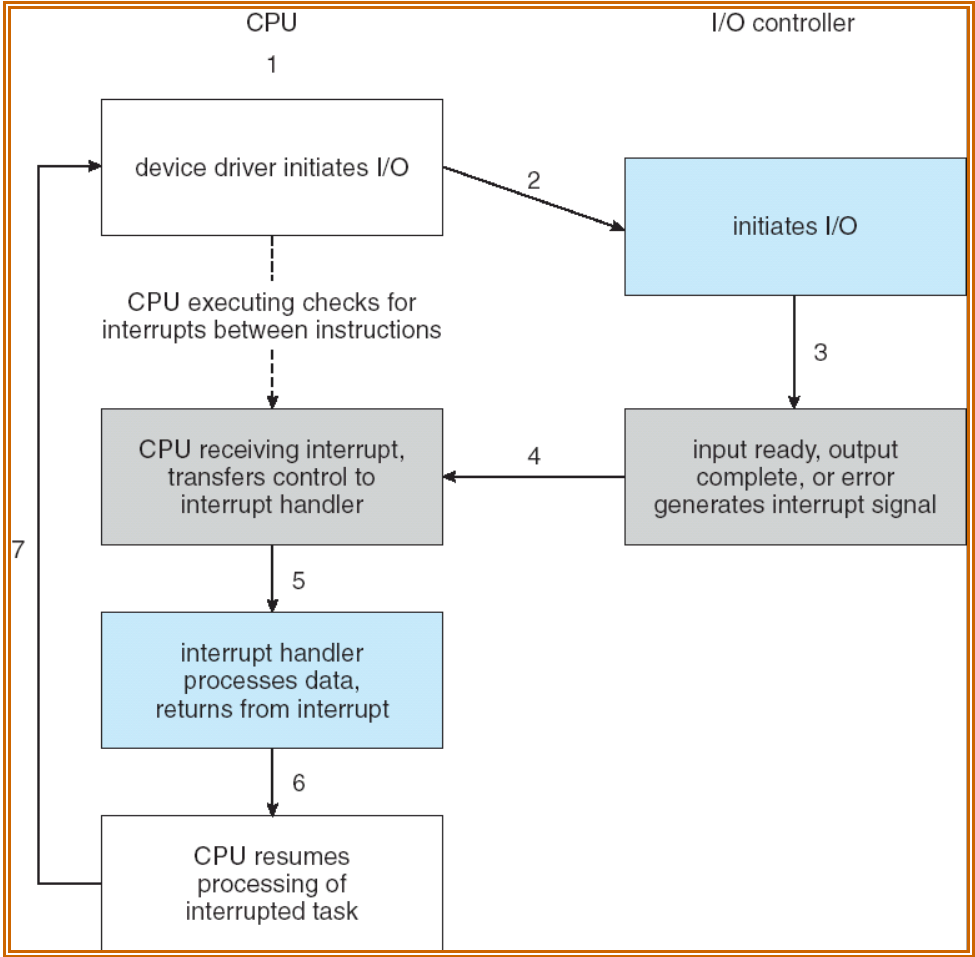
Vòng lặp này được lặp cho mỗi byte.

Trong bước 1, máy tính là chờ đợi bận hay thăm dò. Nó ở trong một vòng lặp, đọc thanh ghi trạng thái cho đến khi bit bận được xoá. Nếu bộ điều khiển và thiết bị nhanh thì phương pháp này là một phương pháp phù hợp. Nhưng nếu chờ lâu máy chủ chuyển sang một tác vụ khác. Sau đó, máy tính làm thế nào để biết khi nào bộ điều khiển rảnh? Đối với một số thiết bị, máy tính phải phục vụ thiết bị nhanh chóng hoặc dữ liệu sẽ bị mất. Thí dụ, khi dữ liệu đang truyền vào cổng tuần tự từ bàn phím, một vùng đệm nhỏ trên bộ điều khiển sẽ tràn và dữ liệu sẽ bị mất nếu máy tính chờ quá lâu trước khi trả về các bytes được đọc.

Trong nhiều kiến trúc máy tính, 3 chu kỳ lệnh CPU đủ để thăm dò một thiết bị: read một thanh ghi thiết bị, thực hiện phép tính luận lý and để lấy bit trạng thái và tách ra (branch) nếu khác 0. Rõ ràng, thao tác thăm dò cơ bản là đủ. Nhưng thăm dò trở nên không đủ khi được lặp lại nhiều lần, hiếm khi tìm một thiết bị sẵn sàng

phục vụ trong lần thăm dò đầu tiên, trong khi cần dùng CPU để xử lý cho các công việc khác. Trong trường hợp như thế, sẽ hiệu quả hơn để sắp xếp bộ điều khiển phần cứng thông báo cho CPU khi nào thiết bị sẵn sàng phục vụ hơn là yêu cầu CPU lặp lại việc thăm dò cho việc hoàn thành nhập/xuất. Cơ chế phần cứng cho phép một thiết bị thông báo tới CPU được gọi là ngắt (interrupt).

4.3.2.2. Ngắt

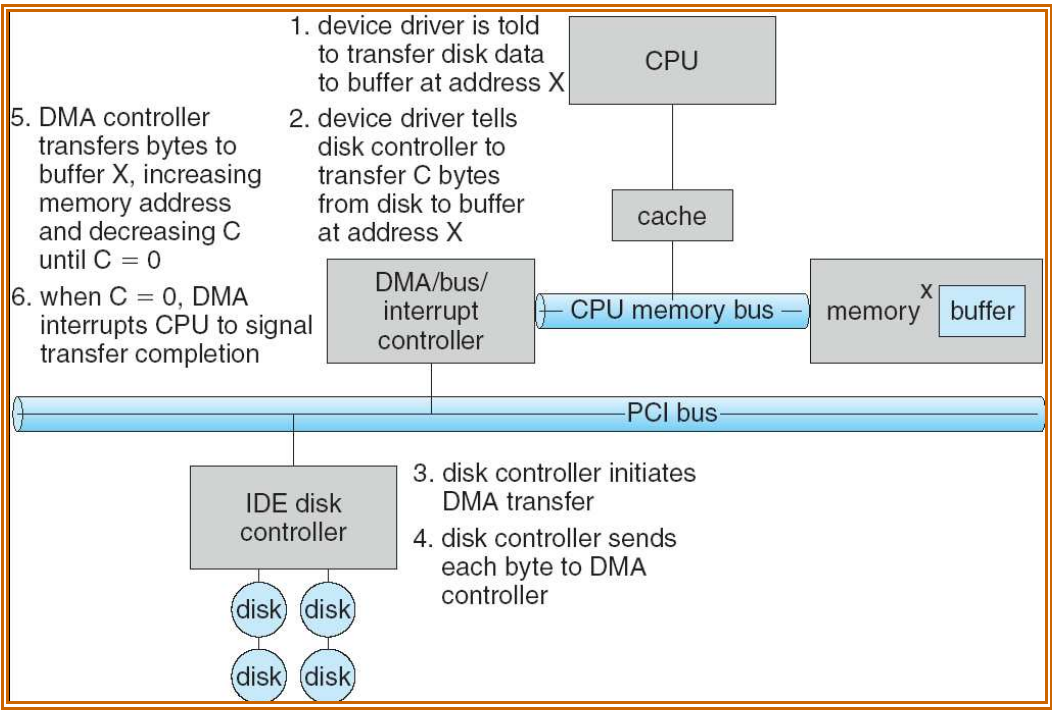


Hình 4.20 Chu trình nhập/xuất của ngắt - ổ đĩa

Cơ chế ngắt cơ bản làm việc như sau: phần cứng CPU có một dây dẫn được gọi là dòng yêu cầu ngắt (interrupt-request line) mà CPU cảm ứng sau khi thực thi mỗi chỉ thị. Khi một CPU phát hiện một bộ điều khiển xác nhận một tín hiệu trên dòng yêu cầu ngắt thì CPU lưu một lượng nhỏ trạng thái như giá trị hiện hành của con trỏ lệnh, và nhảy tới thủ tục của bộ quản lý ngắt (interrupt-handler) tại địa chỉ cố định trong bộ nhớ. Bộ quản lý ngắt xác định nguyên nhân gây ra ngắt, thực hiện xử lý cần thiết, thực thi chỉ thị return from interrupt để trả về CPU trạng thái thực thi trước khi ngắt. Chúng ta nói rằng bộ điều khiển thiết bị sinh ra một ngắt bằng cách xác định tín hiệu trên dòng yêu cầu ngắt và bộ quản lý xoá ngắt bằng cách phục vụ thiết bị. Hình 4.20 tóm tắt chu kỳ nhập/xuất hướng ngắt (interrupt-driven I/O cycle).

4.3.2.3. Truy xuất bộ nhớ trực tiếp

Đối với một thiết bị thực hiện việc truyền lớn như ổ đĩa, nó sẽ lãng phí khi dùng bộ vi xử lý để theo dõi các bit trạng thái và đẩy dữ liệu vào thanh ghi điều khiển từng byte một. Nhiều máy tính muốn giảm đi gánh nặng cho CPU bằng cách chuyển một số công việc này tới một bộ điều khiển có mục đích đặc biệt được gọi là bộ điều khiển truy xuất bộ nhớ trực tiếp (direct memory-access-DMA).



Hình 4.21 Các bước trong việc truyền dữ liệu của DMA

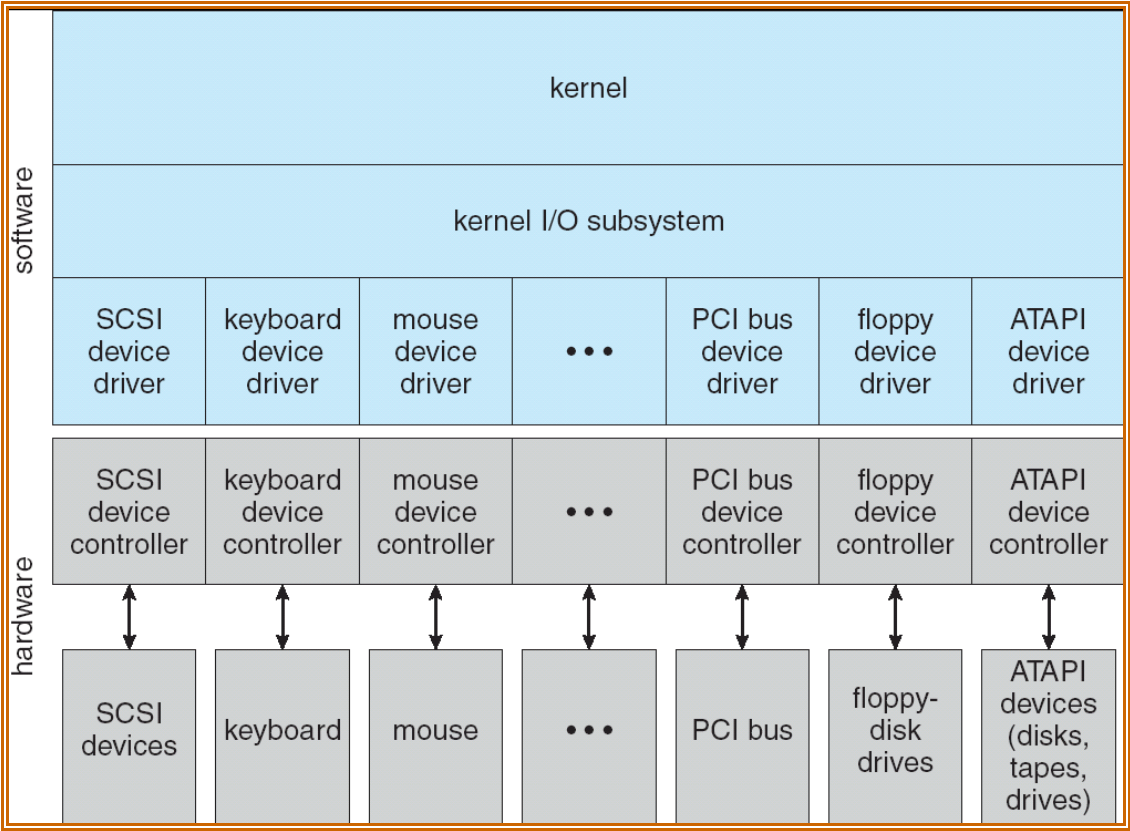
Để khởi tạo một thao tác chuyển DMA, máy tính viết một khối lệnh DMA vào bộ nhớ. Khối này chứa một con trỏ chỉ tới nguồn chuyển, một con trỏ chỉ tới đích chuyển và đếm số lượng byte được chuyển. CPU viết địa chỉ của khối lệnh này tới bộ điều khiển DMA, sau đó CPU tiếp tục làm công việc khác. Bộ điều khiển DMA xử lý để điều hành bus bộ nhớ trực tiếp, đặt các địa chỉ trên bus để thực hiện việc chuyển mà không có sự trợ giúp của CPU. Một bộ điều khiển DMA đơn giản là một thành phần chuẩn trong PCs, và bảng nhập/xuất bus chính (bus-mastering I/O boards) để PC thường chứa phần cứng DMA tốc độ cao. Quá trình này được mô tả trong hình 4.21.

4.3.3. Giao diện nhập/xuất ứng dụng

Trong phần này, chúng ta thảo luận các kỹ thuật cấu trúc và các giao diện cho hệ điều hành cho phép các thiết bị nhập/xuất được đối xử trong cách chuẩn, không đối. Thí dụ, chúng ta giải thích một ứng dụng có thể mở một tập tin trên đĩa mà không biết loại đĩa đó là gì và các đĩa mới và các thiết bị khác có thể được thêm tới máy tính như thế nào mà không làm hệ điều hành bị gián đoạn.

Như những vấn đề công nghệ phần mềm phức tạp khác, tiếp cận ở đây liên quan đến tính trừu tượng, bao gói và phân tầng phần mềm. Đặc biệt, chúng ta có thể trừu

tượng sự khác nhau chi tiết trong các thiết bị nhập/xuất bằng cách xác định một vài loại thông dụng. Mỗi loại thông dụng được truy xuất thông qua một tập hợp hàm chuẩn-một giao diện. Sự khác biệt này được bao gói trong module nhân được gọi là trình điều khiển thiết bị (device driver) mà qui định bên trong được áp đặt cho mỗi thiết bị, nhưng được nhập vào một trong những giao diện chuẩn. Hình 4.22 hiển thị cách các thành phần liên quan nhập/xuất của nhân được cấu trúc trong các tầng phần mềm.



Hình 4.22 Cấu trúc của nhân nhập/xuất

Mục đích của tầng chứa trình điều khiển thiết bị là che đậy sự khác biệt giữa các bộ điều khiển thiết bị từ hệ con nhập/xuất của nhân, nhiều lời gọi hệ thống nhập/xuất đóng gói các hành vi của thiết bị trong một vài lớp phát sinh để che đậy sự khác biệt từ các ứng dụng. Thực hiện hệ thống con nhập/xuất độc lập với phần cứng đơn giản hóa công việc của người phát triển hệ điều hành. Nó cũng đem lại sự thuận lợi cho các nhà sản xuất phần cứng. Họ thiết kế các thiết bị mới tương thích với giao diện bộ điều khiển chủ đã có (như SCSI-2) hay họ viết các trình điều khiển thiết bị để giao tiếp phần cứng mới đối với các hệ điều hành phổ biến. Do đó, các thiết bị ngoại vi mới có thể được gán tới một máy tính mà không phải chờ nhà cung cấp hệ điều hành phát triển thêm mã.

4.3.4. Hệ thống con nhập/xuất của nhân (kernel I/O subsystem)

Nhân cung cấp nhiều dịch vụ liên quan đến nhập/xuất. Một vài dịch vụ-định thời biểu, vùng đệm (buffering), vùng lưu trữ (cache), đặt trước thiết bị, quản lý lỗi-

được cung cấp bởi hệ thống con nhập/xuất của nhân và xây dựng trên phần cứng và cơ sở hạ tầng trình điều khiển thiết bị.

4.3.4.1. Định thời biểu nhập/xuất

Định thời biểu cho tập hợp các yêu cầu nhập xuất có nghĩa là xác định một thứ tự tốt để thực thi chúng. Thứ tự các ứng dụng phát ra lời gọi hệ thống rất hiếm khi là một chọn lựa tốt nhất. Định thời biểu có thể cải tiến năng toàn bộ lực hệ thống, có thể chia sẻ truy xuất thiết bị đồng đều giữa các quá trình và có thể giảm thời gian chờ đợi trung bình cho nhập/xuất hoàn thành. .

Người phát triển hệ điều hành cài đặt bộ định thời biểu bằng cách duy trì một hàng đợi cho mỗi thiết bị. Khi một ứng dụng phát ra một lời gọi hệ thống nhập/xuất nghẽn, yêu cầu được đặt vào hàng đợi cho thiết bị đó. Bộ định thời biểu nhập/xuất sắp xếp lại thứ tự của hàng đợi để cải tiến toàn bộ tính hiệu quả hệ thống và thời gian đáp ứng trung bình dựa theo kinh nghiệm bởi ứng dụng. Hệ điều hành cũng cố gắng giữ bình đẳng để mà không ứng dụng nào nhận được dịch vụ nghèo nàn hay nó cho dịch vụ ưu tiên đối với các yêu cầu trì hoãn nhạy cảm. Thí dụ, các yêu cầu từ hệ thống con bộ nhớ ảo có thể lấy độ ưu tiên qua các yêu cầu của ứng dụng.

Một cách mà hệ thống con nhập/xuất cải tiến tính hiệu quả của máy tính là bằng cách định thời biểu các hoạt động nhập/xuất. Một cách khác là dùng không gian lưu trữ trong bộ nhớ chính hay trên đĩa, với các kỹ thuật được gọi là vùng đệm, vùng lưu trữ và vùng chứa.

4.3.4.2. Vùng đệm

Vùng đệm là một vùng bộ nhớ lưu trữ dữ liệu trong khi chúng được chuyển giữa hai thiết bị hay giữa thiết bị và ứng dụng. Vùng đệm được thực hiện với 3 lý do:

- Lý do thứ nhất là đối phó với việc không tương thích về tốc độ giữa người sản xuất và người tiêu dùng của dòng dữ liệu.
- Lý do thứ hai cho việc sử dụng vùng là làm thích ứng giữa các thiết bị có kích thước truyền dữ liệu khác nhau.
- Lý do thứ ba cho việc dùng vùng đệm là hỗ trợ ngữ nghĩa sao chép cho nhập/xuất ứng dụng.

4.3.4.3. Vùng lưu trữ

Vùng lưu trữ (cache) là một vùng bộ nhớ nhanh quản lý các bản sao dữ liệu. Truy xuất tới một bản sao được lưu trữ hiệu quả hơn truy xuất tới bản gốc. Thí dụ, các chỉ thị của quá trình hiện đang chạy được lưu trên đĩa, được lưu trữ trong bộ nhớ vật lý và được sao chép một lần nữa trong vùng lưu trữ phụ và chính. Sự khác nhau giữa vùng đệm là vùng lưu trữ là vùng đệm có thể giữ chỉ bản sao của thành phần dữ liệu đã có, ngược lại một vùng lưu trữ giữ vừa đủ một bản sao trên thiết bị lưu trữ nhanh hơn của một thành phần nằm ở một nơi nào khác. Vùng lưu trữ và vùng đệm có chức năng khác nhau nhưng đôi khi một vùng bộ nhớ có thể được dùng cho cả hai mục đích.

4.3.4.4. Vùng chứa và đặt trước thiết bị

Một vùng chứa (spool) là một vùng đệm giữ dữ liệu xuất cho một thiết bị như máy in mà không thể chấp nhận các dòng dữ liệu đan xen nhau. Mặc dù một máy in có thể phục vụ chỉ một công việc tại một thời điểm, nhưng nhiều ứng dụng muốn in đồng thời mà không có dữ liệu xuất của chúng đan xen với nhau. Hệ điều hành giải quyết vấn đề này bằng cách ngăn chặn tất cả dữ liệu xuất tới máy in. Dữ liệu xuất của mỗi ứng dụng được chứa trong một tập tin riêng. Khi một ứng dụng kết thúc việc in, hệ thống vùng chứa xếp tập tin chứa tương ứng cho dữ liệu xuất tới máy in. Hệ thống vùng chứa chép các tập tin được xếp hàng tới máy in một tập tin tại một thời điểm. Trong một hệ điều hành, vùng chứa được quản lý bởi một quá trình hệ thống chạy ở chế độ nền. Trong một số hệ điều hành khác, nó được quản lý bởi luồng nhân. Trong mỗi trường hợp, hệ điều hành cung cấp một giao diện điều khiển cho phép người dùng và người quản trị hệ thống hiển thị hàng đợi để xóa các công việc không mong muốn trước khi các công việc đó in, để tạm dừng việc in trong khi máy in được phục vụ...

4.3.4.5. Quản lý lỗi

Một hệ điều hành sử dụng bộ nhớ bảo vệ có thể chống lại nhiều lỗi phần cứng và ứng dụng vì thế một lỗi toàn hệ thống không là kết quả của mỗi sự trục trặc cơ học thứ yếu. Các thiết bị và truyền nhập/xuất có thể bị lỗi trong nhiều cách, có thể là các lý do tạm thời như mạng trở nên quá tải, hay các lý do thường xuyên như trình điều khiển đĩa bị lỗi. Các hệ điều hành thường trả giá cho tính hiệu quả vì các lỗi tạm thời. Ví dụ, lỗi đọc đĩa *read()* dẫn đến cố gắng làm lại *read()* và lỗi gọi dữ liệu trên mạng *send()* dẫn tới việc gọi lại *resend()* nếu giao thức được xác định rõ. Tuy nhiên, nếu một thành phần quan trọng bị lỗi thường xuyên thì hệ điều hành không nghĩ đến việc phục hồi.

Như một qui tắc thông thường, một lời gọi hệ thống nhập/xuất trả về 1 bit của thông tin về trạng thái của lời gọi, biểu thị thành công hay thất bại. Trong hệ điều hành UNIX, một biến nguyên có tên *errno* được dùng để trả về một mã lỗi- một trong 100 giá trị-hiển thị tính tự nhiên của lỗi (thí dụ: đối số vượt quá giới hạn, lỗi con trỏ, tập tin không thể mở,...). Ngược lại, một số phần cứng cung cấp thông tin lỗi được mô tả chi tiết mặc dù nhiều hệ điều hành hiện tại không được thiết kế để truyền đạt thông tin này tới ứng dụng.

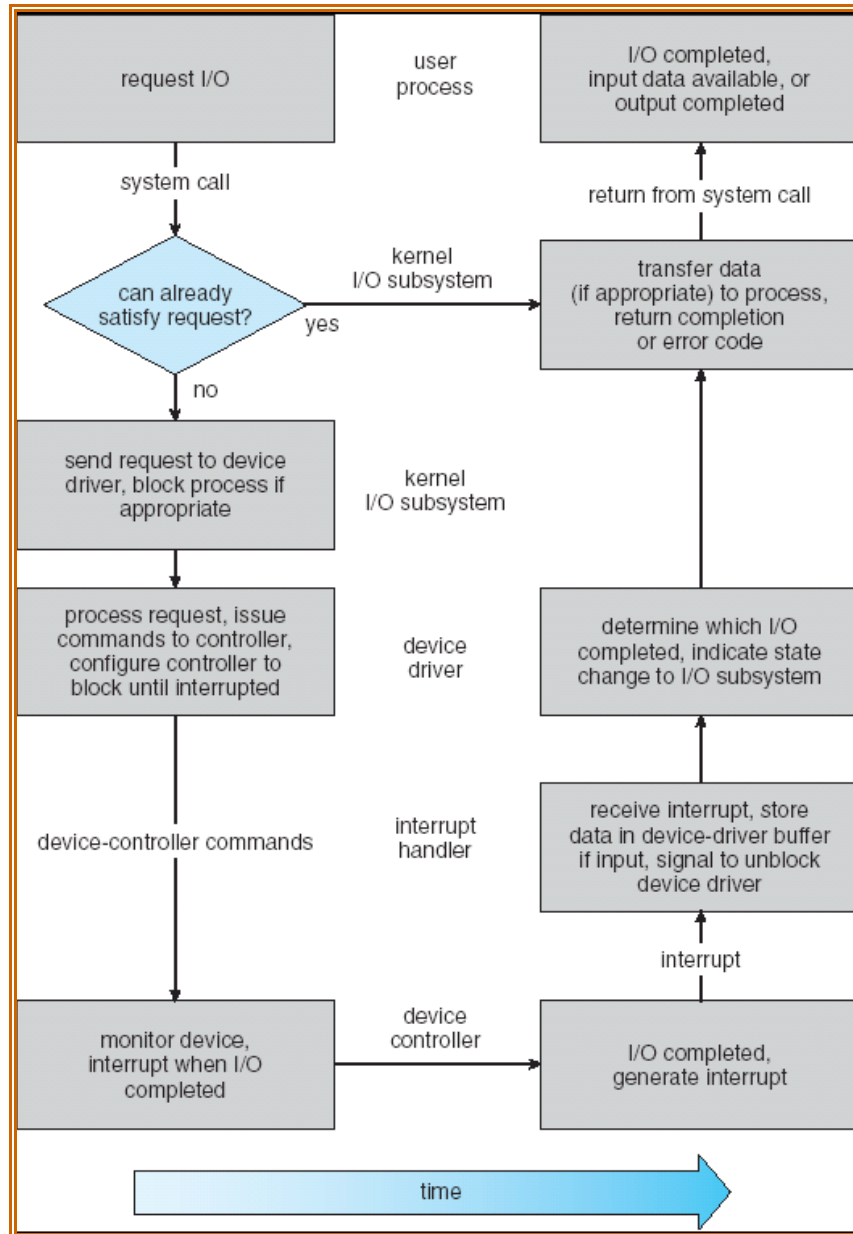
4.3.5. Chuyển đổi yêu cầu nhập/xuất từ thao tác phần cứng

Phần trước chúng ta mô tả việc bắt tay giữa một trình điều khiển thiết bị và bộ điều khiển thiết bị, nhưng chúng ta không giải thích cách hệ điều hành nối kết yêu cầu ứng dụng tới tập hợp dây mạng hay một sector đĩa xác định như thế nào. Chúng ta hãy xem xét một ví dụ đọc một tập tin từ đĩa. Ứng dụng tham chiếu tới dữ liệu bằng tên tập tin. Trong một đĩa, hệ thống tập tin ánh xạ từ tên tập tin thông qua các thư mục hệ thống tập tin để lấy không gian cấp phát của tập tin.

Các hệ điều hành hiện đại đạt được khả năng linh hoạt cao từ nhiều giai đoạn của bảng tra cứu trong đường dẫn giữa yêu cầu và bộ điều khiển thiết bị vật lý. Các cơ chế truyền yêu cầu giữa ứng dụng và trình điều khiển là phổ biến. Do đó, chúng ta có thể giới thiệu các thiết bị mới và trình điều khiển vào máy tính mà không biên

dịch lại nhân. Thật vậy, một số hệ điều hành có khả năng nạp trình điều khiển thiết bị theo yêu cầu. Tại thời điểm khởi động, hệ thống đầu tiên thăm dò các bus phần cứng để xác định thiết bị nào hiện diện và sau đó hệ thống nạp các trình điều khiển cần thiết ngay lập tức hay khi được yêu cầu bởi một yêu cầu nhập/xuất đầu tiên.

Bây giờ chúng ta mô tả chu trình sống điển hình của một yêu cầu đọc bị nghẽn, như trong hình 4.23. Hình này đề nghị rằng một thao tác nhập/xuất yêu cầu nhiều bước và tiêu tốn số lượng lớn chu kỳ CPU.



Hình 4.22 Cấu trúc của nhân nhập/xuất

(1) Một tiến trình phát ra một lời gọi hệ thống *read()* tới bộ mô tả tập tin đã được mở trước đó.

(2) Mã lời gọi hệ thống trong nhân kiểm tra tính đúng đắn của các tham số. Trong trường hợp nhập, nếu dữ liệu đã có sẵn trong vùng đệm thì dữ liệu được trả về tới quá trình và yêu cầu nhập/xuất được hoàn thành.

(3) Ngược lại, nhập/xuất vật lý cần được thực hiện để mã quá trình được xóa từ hàng đợi thực thi và được đặt vào hàng đợi chờ cho thiết bị, và yêu cầu nhập/xuất được lập thời biểu. Cuối cùng, hệ con nhập/xuất gửi yêu cầu tới trình điều khiển thiết bị. Phụ thuộc vào hệ điều hành, yêu cầu được gửi bằng lời gọi thủ tục con hay bằng thông điệp trong nhân.

(4) Trình điều khiển thiết bị cấp phát vùng đệm nhân để nhận dữ liệu và lập thời biểu nhập/xuất. Cuối cùng, trình điều khiển gửi lệnh tới bộ điều khiển thiết bị bằng cách viết vào thanh ghi điều khiển của thiết bị.

(5) Trình điều khiển thiết bị thao tác trên phần cứng thiết bị để thực hiện truyền dữ liệu.

(6) Trình điều khiển có thể thăm dò trạng thái và dữ liệu hay thiết lập truyền DMA vào bộ nhớ nhân. Chúng ta thừa nhận rằng truyền được quản lý bởi bộ điều khiển DMA sinh ra một ngắt khi việc truyền hoàn thành.

(7) Bộ quản lý ngắt tương ứng nhận ngắt bằng bảng vector ngắt, lưu bất cứ dữ liệu cần thiết, báo hiệu trình điều khiển thiết bị và trả về từ ngắt.

(8) Trình điều khiển thiết bị nhận tín hiệu, xác định yêu cầu nhập/xuất hoàn thành, xác định trạng thái yêu cầu và báo hiệu cho hệ con nhập/xuất nhân rằng yêu cầu đã hoàn thành.

(9) Nhân truyền dữ liệu hay trả về mã tới không gian địa chỉ của tiến trình được yêu cầu và di chuyển tiến trình từ hàng đợi chờ tới hàng đợi sẵn sàng.

(10) Di chuyển tiến trình tới hàng đợi sẵn sàng không làm nghẽn tiến trình. Khi bộ định thời biểu gán tiến trình tới CPU, tiến trình tiếp tục thực thi tại thời điểm hoàn thành của lời gọi hệ thống.

Một số từ khóa tiếng Anh

Từ khóa	Tên đầy đủ	Nghĩa tiếng Việt
CPU	Central Processing Unit	Đơn vị xử lý trung tâm
API	Application Programming Interface	Các hàm giao diện chương trình ứng dụng
PCB	Process Control Block	Khối điều khiển tiến trình
I/O	Input/Output	Vào/Ra
FCFS	First-come, First-served	Đến trước, phục vụ trước
RL	Ready List	Danh sách tiến trình sẵn sàng
RR	Round Robin	Phân phối xoay vòng
SJF	Shortest Job First	Công việc ngắn nhất
CR	Critical Resource	Tài nguyên căng
CS	Critical Section	Đoạn căng
	DeadLock	Tắc nghẽn
	Semaphore	Giải pháp đèn báo
	Logical address	Địa chỉ logic
	Logical Memory	Không gian nhớ logic
	Physical address	Địa chỉ vật lý
	Physical memory	Không gian nhớ vật lý
MMU	Memory Management Unit	Đơn vị quản lý bộ nhớ
	Segmentation	Phân đoạn bộ nhớ
	Paging	Phân trang bộ nhớ
	Page table	Bảng trang
VM	Virtual Memory	Bộ nhớ ảo
LRU	Least Recently Used	Chiến lược thay thay thế

		trang nạn nhân là trang lâu nhất chưa sử dụng đến trong quá khứ
FCB	File Control Block	Khởi điều khiển tập tin
FAT	File Allocation Table	Hệ thống quản lý tập tin
NTFS	New Technology File System	Hệ thống tập tin công nghệ mới
MFT	Master File Table	Bảng quản lý tập tin
	Interrupt	Ngắt
DMA	Direct Memory Access	Truy nhập bộ nhớ trực tiếp

Tài liệu tham khảo

[1]. Hà Quang Thụy, *Nguyên lý hệ điều hành*, NXB KHKT, 2002.

[2]. Nguyễn Gia Định, Nguyễn Kim Tuấn, *Nguyên lý Hệ điều hành*, NXB Khoa học và kỹ thuật, 2005

[3]. Nguyễn Phú Trường, *Giáo trình Hệ điều hành*, Đại học Cần Thơ, 2005.

[4]. Trần Hạnh Nhi, *Giáo trình Hệ điều hành nâng cao*, Đại học Khoa học tự nhiên, TP Hồ Chí Minh, 1998.

[5]. Andrew S. Tanenbaum, Albert S Woodhull, *Operating Systems: Design and Implementation*, 3rd edition, Prentice-Hall. 2006.

[6]. Andrew S. Tanenbaum, *Modern Operating Systems*, Prentice Hall, ISBN: 0136006639, 2007.

[7]. Silberschatz, Galvin, Gagne, *Operating System Concepts*, John Wiley & Sons, 2009

[8]. William Stallings, *Operating Systems: Internals and Design Principles*, 5th edition, Prentice-Hall, 2009.

[9]. <http://williamstallings.com/OperatingSystems/index.html>