HỒ ĐẮC PHƯƠNG

GIÁO TRÌNH
NGUYÊN LÝ HỆ ĐIỀU HÀNH

NHÀ XUẤT BẢN GIÁO DỤC VIỆT NAM

HỒ ĐẮC PHƯƠNG

Giáo trình

NGUYÊN LÝ HỆ ĐIỀU HÀNH

(Tái bản lần thứ nhất)

TRƯỜNG CAO ĐẲNG
CÔNG NGHỆ & KINH TẾ HÀ NỘI
THƯ VIỆN

cuu duong than cong . com

NHÀ XUẤT BẢN GIÁO DỤC VIỆT NAM

cuu duong than cong . com

cuu duong than cong . com

Công ty Cổ phần sách Đại học - Dạy nghề – Nhà xuất bản Giáo dục Việt Nam giữ quyền công bố tác phẩm.

19 – 2010/CXB/254 – 2244/GD Mã số : 7B751y0 – DAI

MỤC LỤC

Trang

Chương 1. GIỚI THIỆU CHUNG **7**

1.1. MÁY TÍNH VÀ PHẦN MỀM 7
1.2. LỊCH SỬ PHÁT TRIỂN CỦA HỆ ĐIỀU HÀNH 13
CÂU HỎI ÔN TẬP 13

Chương 2. SỬ DỤNG HỆ ĐIỀU HÀNH **14**

2.1. MÔ HÌNH TÍNH TOÁN TRỪU TƯỢNG 14
2.2. TÀI NGUYÊN 15
2.3. TIẾN TRÌNH 18
2.4. LUỒNG 25
2.5. ĐỐI TƯỢNG 27
2.6. NHẬN XÉT 28
CÂU HỎI ÔN TẬP 28

cuu duong than cong . com

Chương 3. CẤU TRÚC HỆ ĐIỀU HÀNH **29**

3.1. PHÂN TÍCH CÁC YẾU TỐ TÁC ĐỘNG ĐẾN HỆ ĐIỀU HÀNH 29
3.2. CÁC CHỨC NĂNG CƠ BẢN 33
3.3. CÁC PHƯƠNG THỨC CÀI ĐẶT HỆ ĐIỀU HÀMH 37
3.4. NHẬN XÉT 41
CÂU HỎI ÔN TẬP 41

Chương 4. TIẾN TRÌNH **42**

4.1. TIẾN TRÌNH VÀ TRẠNG THÁI TIẾN TRÌNH 42
4.2. THAO TÁC TRÊN TIẾN TRÌNH 45
4.3. MÔ TẢ TIẾN TRÌNH 48
4.4. LUỒNG 53
4.5. CÀI ĐẶT HỆ ĐIỀU HÀNH 57
4.6. NHẬN XÉT 59
CÂU HỎI ÔN TẬP 60

cuu duong than cong . com

Chương 5. ĐIỀU PHỐI TIẾN TRÌNH **61**

5.1. CƠ CHẾ ĐIỀU PHỐI 61
5.2. CÁC PHƯƠNG PHÁP ĐIỀU PHỐI 67
5.3. THUẬT TOÁN ĐỘC QUYỀN 71

5.4. THUẬT TOÁN KHÔNG ĐỘC QUYỀN 77
5.5. NHẬN XÉT 84
CÂU HỎI ÔN TẬP 84

Chương 6. TƯƠNG TRANH VÀ ĐỒNG BỘ 85

6.1. CÁC KHÁI NIỆM CƠ BẢN 85
6.2. ĐỘC QUYỀN TRUY XUẤT – GIẢI PHÁP PHẦN MỀM 88
6.3. ĐỒNG BỘ HÓA – GIẢI PHÁP PHẦN CỨNG 94
6.4. GIẢI PHÁP ĐỒNG BỘ CƠ BẢN 96
6.5. NHỮNG VẤN ĐỀ ĐỒNG BỘ KINH ĐIỂN 101
6.6. CÁC GIẢI PHÁP ĐỒNG BỘ CAO CẤP 104
6.7. CƠ CHẾ IPC 109
6.8. NHẬN XÉT 113
CÂU HỎI ÔN TẬP 114

Chương 7. BẾ TẮC 115

7.1. MÔ HÌNH HỆ THỐNG 115
7.2. ĐẶC ĐIỂM CỦA BẾ TẮC 116
7.3. NGĂN CHẶN BẾ TẮC 119
7.4. TRÁNH BẾ TẮC 121
7.5. PHÁT HIỆN BẾ TẮC 126
7.6. KHẮC PHỤC BẾ TẮC 129
7.7. NHẬN XÉT 130
CÂU HỎI ÔN TẬP 131

Chương 8. QUẢN LÝ THIẾT BỊ 132

8.1. NGUYÊN LÝ HOẠT ĐỘNG 132
8.2. CHIẾN LƯỢC QUẢN LÝ THIẾT BỊ 134
8.3. TRÌNH ĐIỀU KHIỂN THIẾT BỊ 150
CÂU HỎI ÔN TẬP 152

Chương 9. QUẢN LÝ BỘ NHỚ 153

9.1. CÁC LOẠI ĐỊA CHỈ 153
9.2. KHÔNG GIAN ĐỊA CHỈ 157
9.3. HOÁN CHUYỂN 158
9.4. CẤP PHÁT LIÊN TỤC 159
9.5. PHÂN TRANG 163
9.6. PHÂN ĐOẠN 173
9.7. KẾT HỢP PHÂN ĐOẠN VỚI PHÂN TRANG 178
9.8. NHẬN XÉT 180
CÂU HỎI ÔN TẬP 181

cuu duong than cong . com

Chương 10. BỘ NHỚ ẢO 182

10.1. ĐẶT VẤN ĐỀ 182
10.2. PHÂN TRANG THEO YÊU CẦU 183
10.3. HIỆU SUẤT PHÂN TRANG THEO YÊU CẦU 186
10.4. THAY THẾ TRANG 188
10.5. THUẬT TOÁN THAY THẾ TRANG 190
10.6. CẤP PHÁT FRAME 197
10.7. PHÂN ĐOẠN THEO YÊU CẦU 200
10.8. NHẬN XÉT 201
CÂU HỎI ÔN TẬP 201

Chương 11. HỆ THỐNG FILE 202

11.1. FILE 202
11.2. CÀI ĐẶT FILE Ở MỨC THẤP 209
11.3. HỆ THỐNG THƯ MỤC 219
11.4. BẢO VỆ FILE CHIA SẺ 226
11.5. TÍNH THỐNG NHẤT CỦA NGỮ NGHĨA 229
11.6. PHỤC HỒI SAU LỖI 231
11.7. NHẬN XÉT 232
CÂU HỎI ÔN TẬP 233

Chương 12. BẢO VỆ VÀ AN NINH 234

12.1. CÁC VẤN ĐỀ CƠ BẢN 234
12.2. XÁC THỰC 238
12.3. KIỂM CHỨNG 241
12.4. CÀI ĐẶT MA TRẬN QUYỀN TRUY CẬP 249
12.5. HẬU QUẢ TỪ CHƯƠNG TRÌNH 255
12.6. GIÁM SÁT NGUY CƠ 262
12.7. MẬT MÃ VÀ ỨNG DỤNG 264
12.8. NHẬN XÉT 268
CÂU HỎI ÔN TẬP 269

TÀI LIỆU THAM KHẢO 270

cuu duong than cong . com

Chương 1
GIỚI THIỆU CHUNG

Chương này khái quát về Hệ điều hành (HĐH) và lịch sử phát triển HĐH. Trước tiên, giới thiệu tổng quan khái niệm phần mềm và vị trí cụ thể của HĐH trong hệ thống phần mềm. Tiếp đến, trình bày hai động lực phát triển của HĐH hiện đại là: ***Trừu tượng hóa*** *và* ***Chia sẻ tài nguyên phần cứng****.*

1.1. MÁY TÍNH VÀ PHẦN MỀM

Con người sử dụng máy tính thông qua phần mềm. Phần mềm được phân loại theo mục đích sử dụng: ***Phần mềm ứng dụng*** giải quyết vấn đề cụ thể (MS Word, MS Excel). ***Phần mềm hệ thống*** thực thi những nhiệm vụ liên quan tới quá trình thực thi các chương trình ứng dụng. Bên cạnh đó, phần mềm hệ thống cung cấp những chức năng mà phần cứng không thể cung cấp, giúp lập trình viên phát triển ứng dụng,... HĐH là phần mềm hệ thống quan trọng nhất.

Mục tiêu quan trọng của HĐH là cho phép nhiều phần mềm ứng dụng cùng nhau sử dụng phần cứng máy tính một cách có trật tự. Chia sẻ làm tăng hiệu suất sử dụng hệ thống, vì các chương trình khác nhau đồng thời sử dụng những bộ phận phần cứng khác nhau. Do đó, giảm thời gian cần thiết để thực hiện một nhóm chương trình. Để chia sẻ an toàn và có hiệu quả, HĐH phải nằm sát phần cứng. Phần mềm hệ thống và tất cả các phần mềm ứng dụng gián tiếp sử dụng phần cứng thông qua HĐH.

1.1.1. Phần mềm hệ thống

Với người lập trình ứng dụng, phần mềm hệ thống đơn giản hóa môi trường lập trình và cho phép sử dụng hiệu quả phần cứng. Phần mềm hệ thống có chức năng làm môi trường thực thi cho ngôn ngữ lập trình. Trong

UNIX, chức năng này được cài đặt bằng ngôn ngữ C (bằng cách sử dụng các file.h), ví dụ:

- Thư viện vào/ra chuẩn (I/O) thực hiện các thao tác vào/ra thông qua bộ đệm trên dòng dữ liệu.
- Thư viện toán học để tính toán các hàm toán học.
- Thư viện đồ họa cung cấp hàm hiển thị hình ảnh trên màn hình đồ họa.

Một chức năng khác của phần mềm hệ thống là cung cấp hệ thống giao diện cửa sổ. WINDOWS là phần mềm hệ thống cung cấp các cửa sổ (một thiết bị đầu cuối ảo) cho chương trình ứng dụng. Lập trình viên phát triển phần mềm ứng dụng sử dụng những hàm để đọc và ghi lên cửa sổ như thể cửa sổ là một thiết bị đầu cuối, thậm chí cửa sổ này không gắn với bất kỳ thiết bị vật lý nào. Phần mềm hệ thống chịu trách nhiệm ánh xạ thiết bị đầu cuối ảo lên một vùng cụ thể trên màn hình. Một thiết bị đầu cuối vật lý có thể hỗ trợ nhiều thiết bị đầu cuối ảo.

HĐH cung cấp giao diện (là các hàm) để phần mềm hệ thống và phần mềm ứng dụng sử dụng khi muốn dùng tài nguyên hệ thống. HĐH là phần mềm độc lập, hỗ trợ nhiều ứng dụng trong các lĩnh vực khác nhau. Phần mềm ứng dụng sử dụng sự trừu tượng hóa tài nguyên do HĐH cung cấp khi làm việc với phần cứng. HĐH cho phép các ứng dụng khác nhau chia sẻ tài nguyên phần cứng thông qua chính sách quản lý tài nguyên. Trừu tượng hóa tài nguyên và chia sẻ là hai khía cạnh cơ bản của HĐH.

1.1.2. Trừu tượng hóa tài nguyên phần cứng

Bằng cách che dấu chi tiết hoạt động phần cứng thông qua mô hình trừu tượng hoạt động của phần cứng, phần mềm hệ thống giúp lập trình viên sử dụng phần cứng dễ dàng hơn. Mặc dù giúp đơn giản hóa cách thức điều khiển phần cứng, mô hình trừu tượng cũng giới hạn khả năng lập trình viên thao tác trực tiếp trên phần cứng vì có những thao tác phần cứng không thể trừu tượng hóa được. Có thể coi máy rút tiền tự động ATM là sự trừu tượng hóa việc rút tiền ở ngân hàng. ATM có thể cung cấp một thao tác trừu tượng cho phép người dùng rút 1 triệu hay 2 triệu đồng từ tài khoản chỉ thông qua ấn một nút duy nhất. Tuy nhiên, nếu người dùng muốn rút đúng 1,3 triệu

đồng thì phải nhấn một số nút khác nhau: đầu tiên ấn nút rút tiền, sau đó rút từ tài khoản với lượng tiền sẽ rút là 1,3 triệu đồng.

Phần cứng có thể coi là Tài nguyên hệ thống và bất kỳ tài nguyên cụ thể nào đều có giao diện riêng định nghĩa các thao tác mà lập trình viên có thể thực hiện trên tài nguyên. Tuy nhiên, phần mềm hệ thống vẫn có thể tiếp tục trừu tượng hơn nữa để đơn giản hóa giao diện tài nguyên cụ thể. Để sử dụng tài nguyên, người lập trình không nhất thiết phải biết giao diện cụ thể của tài nguyên, mà chỉ cần biết giao diện trừu tượng (bỏ qua hoạt động chi tiết của thiết bị). Do đó, lập trình viên có thể tập trung vào các vấn đề ở mức cao hơn. Các tài nguyên giống nhau có thể được trừu tượng thành một giao diện thống nhất. Ví dụ, phần mềm hệ thống có thể trừu tượng hoạt động ổ đĩa mềm và ổ đĩa cứng thành giao diện ổ đĩa trừu tượng. Người lập trình chỉ cần có kiến thức chung nhất về hoạt động của ổ đĩa trừu tượng, mà không cần biết chi tiết các thao tác vào/ra trên ổ đĩa cứng hay ổ đĩa mềm.

Giả sử phải phát triển ứng dụng phân tích xu thế đầu tư của thị trường chứng khoán. Việc thiết kế và chỉnh sửa đoạn mã thực hiện việc đọc/ghi thông tin trên ổ đĩa chiếm một phần không nhỏ trong toàn bộ đoạn mã chương trình. Kỹ năng cần thiết để viết phần mềm điều khiển ổ đĩa khác kỹ năng phân tích thị trường chứng khoán. Nếu có kiến thức chung về hoạt động của ổ đĩa, lập trình viên ứng dụng không cần quan tâm thao tác vào/ra của ổ đĩa cứng. Trừu tượng tài nguyên là cách tiếp cận tối ưu, vì người lập trình ứng dụng sử dụng mô hình trừu tượng để thực hiện việc đọc/ghi ổ đĩa. Phần mềm điều khiển ổ đĩa là ví dụ về phần mềm hệ thống. Lập trình viên có thể tập trung vào các vấn đề của ứng dụng, chứ không cần quan tâm đến những thứ không liên quan. Nói cách khác, phần mềm hệ thống "trong suốt" với người sử dụng, nhưng rất quan trọng với lập trình viên.

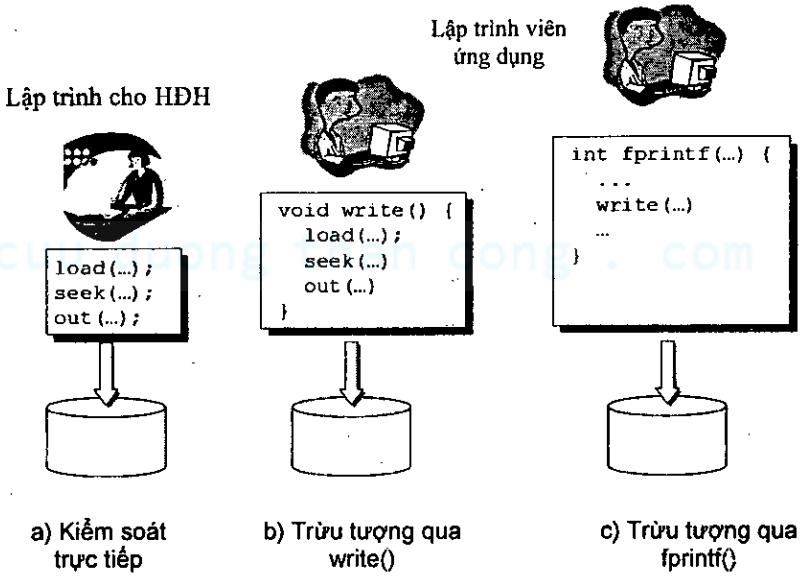
1.1.3. Ví dụ về trừu tượng hóa thiết bị ổ đĩa

Ý tưởng trừu tượng hóa tài nguyên có thể được hiểu rõ thông qua ví dụ hoạt động thiết bị ổ đĩa (Hình 1.1). Phần mềm điều khiển thiết bị sao chép một khối thông tin từ bộ nhớ chính tới bộ nhớ đệm của thiết bị bằng chỉ thị **Load(block, length, device)**. Để chuyển đầu đọc/ghi tới một vị trí cụ thể trên bề mặt đĩa sử dụng chỉ thị **Seek(device, track)**. Thao tác ghi một khối dữ liệu từ vùng đệm vào thiết bị là **out(device, sector)**.

Do đó, cần một nhóm lệnh liên tiếp để ghi khối thông tin từ bộ nhớ chính ra ổ đĩa như Hình 1.1a. Để đơn giản công việc của người phát triển ứng dụng, hệ thống đóng gói những câu lệnh trong Hình 1.1a vào một thủ tục. Hàm **write** (Hình 1.1b), tạo thành một mức trừu tượng hóa cao hơn.

Bước trừu tượng cao hơn ở phần mềm hệ thống cho phép xem ổ đĩa là nơi lưu trữ file. Phần mềm hệ thống sử dụng định danh file (fileID) như một mức trừu tượng ổ đĩa. Khi đó, thư viện (*stdio* của C) có thể cung cấp hàm để viết biến nguyên *datum* vào thiết bị tại một vị trí nào đó trong file bằng hàm **fprintf(fileID, "%d", datum)**.

Mức trừu tượng này có thể được sử dụng cho các thao tác đọc/ghi băng từ nếu phần mềm hệ thống triển khai sự trừu tượng đó cho thiết bị băng từ.



Hình 1.1. Trừu tượng hóa ổ đĩa cứng

Quá trình trừu tượng hóa có thể diễn ra ở nhiều cấp độ. Phần cứng được điều khiển thông qua một giao diện, phần mềm hệ thống ở mức cao có thể tiếp tục trừu tượng hóa tài nguyên này thông qua một mức giao diện cao hơn. Ví dụ trình bày ở trên minh họa rất rõ điều này.

1.1.4. Chia sẻ tài nguyên phần cứng

Tài nguyên trừu tượng và tài nguyên vật lý có thể được nhiều tiến trình đang thực thi đồng thời dùng chung (khái niệm tiến trình – sự thực thi của một chương trình được đề cập trong Chương 2). Có hai kiểu chia sẻ là theo

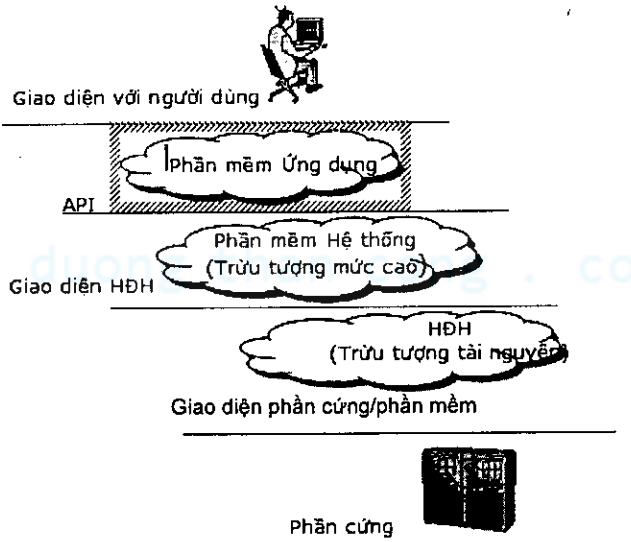
không gian và theo thời gian. Trong chia sẻ theo không gian, tài nguyên được chia ra thành nhiều đơn vị riêng biệt. Các tiến trình đồng thời được cấp phát các đơn vị tài nguyên khác nhau. Đối với các tiến trình, bộ nhớ hay ổ đĩa là tài nguyên phân chia theo không gian.

Trong phương thức chia sẻ theo thời gian, tài nguyên không bị chia nhỏ mà được cấp phát trọn vẹn cho một tiến trình trong một khoảng thời gian. Sau đó, tài nguyên bị thu hồi và cấp phát cho tiến trình khác. Ví dụ, tài nguyên theo kiểu này là bộ vi xử lý. Bộ vi xử lý được điều phối giữa nhiều tiến trình đang nắm giữ các tài nguyên khác nhau. Máy tính chỉ có duy nhất một bộ vi xử lý, nhưng người dùng có cảm giác nhiều chương trình khác nhau cùng được thực hiện.

Có hai khía cạnh quan trọng khi chia sẻ tài nguyên. *Thứ nhất,* hệ thống phải có khả năng kiểm soát việc truy cập tài nguyên qua chính sách cấp phát. *Thứ hai,* hệ thống phải có khả năng cô lập các tài nguyên chia sẻ khi cần thiết. Để ngăn cản các truy cập không hợp lệ, HĐH phải kiểm chứng tiến trình muốn sử dụng tài nguyên có hợp lệ hay không. Cô lập tài nguyên là khả năng HĐH ngăn cản tiến trình truy cập trái phép đến tài nguyên đã được cấp phát cho tiến trình khác. Cơ chế cô lập bộ nhớ cho phép hai chương trình được tải đồng thời vào những phần khác nhau của bộ nhớ. Cơ chế cô lập bộ vi xử lý buộc các tiến trình chia sẻ bộ vi xử lý của hệ thống một cách tuần tự. Không tiến trình nào có thể thay đổi hay tham chiếu đến nội dung bộ nhớ được cấp phát cho tiến trình khác.

Tuy nhiên, phần mềm hệ thống phải cho phép các chương trình đang thực thi có thể chia sẻ quyền truy cập tài nguyên khi cần thiết. Giải quyết vấn đề cô lập tài nguyên phát sinh vấn đề mới. Giả sử người lập trình có ý định cho phép hai chương trình đang chạy chia sẻ tài nguyên (bên cạnh bộ xử lý). HĐH phải đảm bảo cơ chế cô lập, nhưng không được ngăn cản việc truy cập các tài nguyên chia sẻ. Nếu phần mềm hệ thống không cài đặt chính xác cơ chế cô lập tài nguyên, thì không thể đảm bảo cô lập được tài nguyên. Đến lượt mình, phần mềm hệ thống phải có bộ phận đáng tin cậy triển khai việc cô lập tài nguyên, sao cho chương trình ứng dụng không thể vi phạm cơ chế. Bộ phận đáng tin cậy này được cài đặt trong HĐH. Thậm chí phần mềm HĐH phải phụ thuộc vào phần cứng để thực hiện các phần quan trọng nhất của cơ chế cô lập tài nguyên. Các cơ chế chia sẻ tài nguyên trừu tượng thường được cài đặt tại phần mềm hệ thống ở tầng cao, nhưng các cơ chế này phụ thuộc vào thao tác đã được kiểm chứng là đáng tin cậy của HĐH.

Hình 1.2 minh họa sự khác biệt giữa phần mềm hệ thống và HĐH. *Thứ nhất,* phần mềm hệ thống cài đặt mô hình trừu tượng các tài nguyên mà lập trình viên có thể sử dụng, HĐH trực tiếp thực hiện trừu tượng hóa các tài nguyên vật lý. *Thứ hai,* HĐH cung cấp các thao tác cơ sở hoàn toàn đáng tin cậy để quản lý việc chia sẻ tài nguyên. Hình 1.2 minh họa một số giao diện cơ bản giữa những thành phần khác nhau trong hệ thống. Phần mềm ứng dụng sử dụng giao diện lập trình ứng dụng với phần mềm hệ thống, phần mềm hệ thống sử dụng giao diện với HĐH và HĐH sử dụng giao diện phần mềm/phần cứng để tương tác với phần cứng (hệ thống phân cấp trong hình vẽ chỉ mang tính tương đối, ví dụ chương trình ứng dụng hoàn toàn có thể thi hành trực tiếp một số chỉ thị phần cứng).



Hình 1.2. Phần mềm và HĐH

1.1.5. Các máy tính không có phần mềm hệ thống

Với các máy tính cá nhân thời kỳ đầu tiên, có thể viết phần mềm ứng dụng mà không cần trừu tượng hóa hoặc chia sẻ tài nguyên, nên cũng không cần đến phần mềm hệ thống. Các thiết bị đơn giản đến mức không cần trừu tượng hóa tài nguyên và không có nhu cầu hỗ trợ chạy đồng thời nhiều chương trình. Chương trình ứng dụng chịu trách nhiệm thực hiện tất cả các thao tác vào/ra (bàn phím, màn hình, ổ đĩa). Theo thời gian, các thiết bị vào/ra ngày càng đa dạng và phức tạp; phần mềm điều khiển những thiết bị như vậy dần trở nên quá phức tạp đối với đa phần người lập trình ứng dụng.

Các hãng chế tạo máy tính cá nhân bắt đầu đưa cơ chế trừu tượng tài nguyên vào ROM (ví dụ các thủ tục cơ sở BIOS trong máy tính cá nhân IBM).

Sức mạnh ngày càng tăng của máy tính cá nhân dẫn đến nhu cầu thực thi đồng thời nhiều tiến trình, điều này đòi hỏi cơ chế chia sẻ CPU. Kết quả là phần mềm hệ thống của máy tính cá nhân hiện nay cũng triển khai cơ chế chia sẻ tài nguyên. Với các máy tính tương thích IBM, bước đi này dẫn đến sự ra đời của nhiều HĐH như Microsoft Windows thay thế cho MS-DOS.

1.2. LỊCH SỬ PHÁT TRIỂN CỦA HỆ ĐIỀU HÀNH

Theo thống kê của Từ điển Wikipedia, hiện nay có khoảng 80 HĐH viết cho máy tính, chưa kể đến các thiết bị khác. Như đã nói, HĐH là chương trình quản lý tài nguyên phần cứng, nhằm tạo ra môi trường cho phép người sử dụng thực thi các chương trình ứng dụng. Ngày nay, một chiếc điện thoại di động cũng cần HĐH (như Symbian OS, Windows CE,...), thậm chí người ta còn viết ra những phiên bản Linux cho các thiết bị giải trí số như Xbox, Play Station,... Tuy nhiên, cách đây khoảng nửa thế kỷ, khái niệm HĐH còn chưa ra đời. Một chiếc máy tính khổng lồ của một viện nghiên cứu cùng với rất nhiều các chương trình tính toán phức tạp (của thời đó) được điều phối không phải bởi HĐH, mà bởi "người điều hành" (operator)! Chính sự phát triển của tốc độ xử lý, dung lượng bộ nhớ cùng yêu cầu thực thi chương trình ngày càng phức tạp, đã đặt ra nhu cầu cần có một chương trình tự động điều phối các tài nguyên máy tính phần cứng cũng như phần mềm – từ đó mà HĐH ra đời.

Chặng đường phát triển của HĐH gắn liền với sự tiến hóa của phần cứng máy tính, vì HĐH là phần mềm mức thấp, phụ thuộc rất nhiều vào kiến trúc máy tính. Có thể thấy mỗi bước tiến của công nghệ chế tạo máy tính lại tạo ra một bước đột phá mới cho HĐH.

cuu duong than cong . com

CÂU HỎI ÔN TẬP

1. Trình bày sự khác biệt giữa phần mềm hệ thống và phần mềm ứng dụng.
2. Khái niệm trừu tượng có ưu điểm gì?
3. Tại sao phải có HĐH nằm giữa hệ thống phần mềm và phần cứng?

Chương 2
SỬ DỤNG HỆ ĐIỀU HÀNH

Chương này trình bày các thành phần cơ bản trong môi trường lập trình do HĐH cung cấp theo quan điểm của lập trình viên và nhà thiết kế hệ thống. Trước tiên, tìm hiểu mô hình máy tính được sử dụng trong HĐH hiện đại thông qua việc mô tả các tài nguyên nói chung và tài nguyên file nói riêng. Tiếp theo, tìm hiểu về quá trình hình thành tiến trình, bao gồm các ví dụ trong HĐH UNIX. Phần cuối chương giới thiệu hai mô hình luồng (thread) và đối tượng (object).

cuu duong than cong . com

2.1. MÔ HÌNH TÍNH TOÁN TRỪU TƯỢNG

Người lập trình ứng dụng quan niệm máy tính là thiết bị có thể truy cập, biến đổi và lưu trữ thông tin. HĐH cung cấp môi trường giúp người lập trình định nghĩa các thao tác xử lý thông tin cơ bản nhất thông qua những khái niệm về đơn vị thực thi chương trình cũng như các thành phần cần thiết trong quá trình tính toán. Trong HĐH hiện đại, đơn vị tính toán nhỏ nhất là ***Tiến trình*** (*process*) và đơn vị lưu trữ thông tin nhỏ nhất là ***Tập tin*** (*file*). Các thành phần hệ thống khác có thể là tài nguyên được sử dụng trong quá trình tính toán. Chương trình định rõ hành vi của một hay nhiều tiến trình bằng cách định nghĩa tường minh cách đọc thông tin từ file, phương pháp biến đổi thông tin bằng cách sử dụng tài nguyên hệ thống, sau đó lưu trữ thông tin ra file. Thiết bị lưu trữ thường là nơi lưu trữ file, vì thế chương trình tương tác với thông tin thông qua giao diện file. Các tài nguyên khác có giao diện riêng, được nhà thiết kế HĐH đưa ra để định nghĩa mức độ trừu tượng tài nguyên. Ví dụ, thiết bị hiển thị ảnh nhị phân có giao diện được xây dựng xung quanh khối bộ nhớ hiển thị. Chương trình ứng dụng sẽ sao chép thông tin lên các khối bộ nhớ đặc biệt và hình ảnh tương ứng sẽ được hiển

thị trên màn hình. Phần mềm hệ thống có thể cung cấp thêm nhiều mức độ trừu tượng hóa thông tin như file chỉ số, cơ sở dữ liệu. HĐH cung cấp cửa sổ làm việc cho các thiết bị hiển thị đầu/cuối cũng như tiến trình là đơn vị tính toán cơ sở. Tuy nhiên, với phần lớn HĐH, tiến trình và file là hai giao diện cơ bản và quan trọng nhất.

2.2. TÀI NGUYÊN

Thực thể được coi là tài nguyên nếu thỏa mãn cả hai yêu cầu sau:

- Tiến trình phải yêu cầu thực thể từ HĐH.
- Tiến trình tạm thời ngừng hoạt động đến khi thực thể yêu cầu được cấp phát.

2.2.1. File

File có thể được xem là luồng byte nằm trên thiết bị lưu trữ ngoài và được xác định qua tên gọi. Thông tin được lưu trữ bằng cách mở file (tạo ra một bản mô tả file với tên xác định), sau đó ghi từng khối byte lên. Tương tự, có thể truy cập thông tin lưu trong file bằng cách mở file rồi đọc từng khối byte. HĐH chịu trách nhiệm cài đặt hệ thống file trên thiết bị lưu trữ cố định như ổ đĩa, bằng cách ánh xạ luồng byte vào các khối lưu trữ trên thiết bị. File được phân biệt với các tài nguyên khác vì:

1. File là hình thức lưu trữ thông tin phổ biến nhất trong máy tính.
2. HĐH thường lấy file làm cơ sở để mô hình hóa các tài nguyên khác.

File có bản mô tả cho phép HĐH ghi lại hiện trạng sử dụng file, tình trạng từng file, các thông tin về việc ánh xạ luồng byte trong file tới vị trí cụ thể trên thiết bị lưu trữ.

Hệ thống file theo chuẩn POSIX

File POSIX là tập hợp tuần tự các byte có định danh. BSD (Berkeley Software Distribution) UNIX có giao diện file khác. Giao diện hệ thống file POSIX chỉ cung cấp một vài thao tác file cơ bản (Bảng 2.1).

Chương trình hoàn chỉnh sau minh họa cách sử dụng file qua giao diện POSIX. Chương trình này sao chép từng ký tự từ file có tên **in_test** tới file **out_test** bằng cách mở file **in_test** để đọc từng byte và sau đó ghi vào file **out_test**.

```
#include<stdio.h>
#include<fcntl.h>
 int main()
{
      int  inFile outFile;
      char  *inFileName = "in_test";
      char  *outFileName = "out_test";
      int  len;
      char  c;
      inFile = open(inFileName, O_RDONLY);
      outFile = open(outFieName, O_WRONLY);
      while (len = read((inFile, &c, 1) > 0)) // Vòng đọc đọc toàn bộ file
                                                        write(outFile, &c, 1);
      close(inFile); // Đóng file
      close(outFile);
}
```

Bảng 2.1

Lời gọi hệ thống	Miêu tả
open	Lời gọi open có tham số là tên file (kể cả đường dẫn) mà chúng ta chuẩn bị đọc hoặc ghi thông tin. Lời gọi này có thể có thêm tham số xác định chế độ truy cập file (tức là file mở theo chế độ chỉ đọc hay đọc/ghi). Khi file được mở, con trỏ file trỏ vào byte đầu tiên trong luồng byte (nếu file rỗng, vị trí này là vị trí để ghi byte đầu tiên). Nếu thành công, lời gọi trả về một giá trị nguyên không âm, được gọi là thẻ file. Sau đó người sử dụng sẽ sử dụng thẻ file khi tham chiếu tới file.
close	Lời gọi close là đóng file, sau đó giải phóng các tài nguyên hệ thống sử dụng để mô tả trạng thái file.
read	Lời gọi read có các tham số là thẻ file (giá trị được open trả về), địa chỉ và kích thước bộ đệm. Thông thường, lời gọi này khiến tiến trình gọi bị phong tỏa cho tới khi hoàn thành quá trình đọc. Tuy nhiên, ngữ nghĩa này có thể thay đổi với một hàm thích hợp trong hàm fcntl.
write	Lời gọi write tương tự như read nhưng write được sử dụng để ghi thông tin lên file.
lseek	Lời gọi lseek di chuyển con trỏ đọc/ghi trong luồng byte tới vị trí xác định. Sự di chuyển này ảnh hưởng đến các lệnh đọc và ghi tiếp sau.
fcntl	Lời gọi fcntl (viết tắt của file control) cho phép gửi một yêu cầu điều khiển bất kỳ tới HĐH. Ví dụ, thao tác đọc file thông thường phong tỏa tiến trình đang được gọi nếu nó thực hiện đọc một file trống; khi sử dụng fcntl, thao tác đọc file có thể trả lại người gọi nếu hành động cố gắng đọc file phong tỏa tiến trình đang gọi.

2.2.2. Các tài nguyên khác

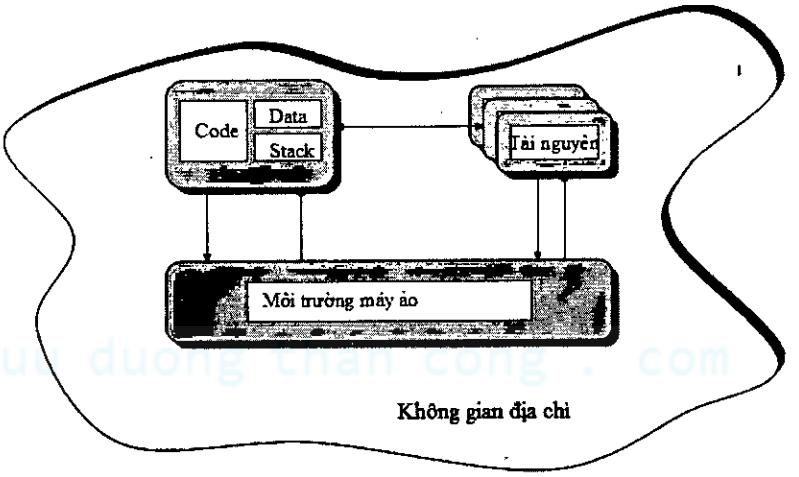
Tài nguyên phần cứng là thành phần trừu tượng bất kỳ mà chương trình cần có trước khi thực thi. Nếu yêu cầu tài nguyên mà chưa được đáp ứng thì tiến trình không thể tiếp tục thực thi mà sẽ bị phong tỏa cho đến khi có đủ tài nguyên cần thiết. CPU là tài nguyên mà bất kỳ tiến trình nào cũng cần phải có nếu muốn thực thi. Ít khi tiến trình yêu cầu cấp phát tài nguyên CPU một cách tường minh, nhưng để thực thi, nhất thiết phải có CPU. Cấp phát tài nguyên CPU cho các tiến trình là chức năng quan trọng của HĐH. Bộ nhớ cũng được xem là tài nguyên. Chương trình có thể yêu cầu tham chiếu đến khu vực bộ nhớ cần thiết trong khi thực thi hoặc đưa ra các yêu cầu cấp phát bộ nhớ động. Ổ đĩa cứng cũng là tài nguyên, vì lập trình viên muốn đọc thông tin từ ổ đĩa thì trước khi đọc dữ liệu, tiến trình phải được cấp phát quyền sử dụng ổ đĩa. HĐH có giao diện hỗ trợ chương trình truy cập một số kiểu tài nguyên khác, chẳng hạn CPU, bộ nhớ, bàn phím và màn hình hiển thị. Trong trường hợp giao diện với tất cả kiểu tài nguyên giống nhau, lập trình viên sử dụng tài nguyên dễ dàng hơn nhiều so với trường hợp các kiểu tài nguyên có giao diện khác nhau. UNIX áp dụng phương pháp này. Nói chung người thiết kế HĐH cố gắng cài đặt giao diện tài nguyên tương tự giao diện file, mặc dù điều này chưa chắc thực hiện được trong một số trường hợp.

Bộ phận quản lý bộ nhớ của HĐH UNIX cấp phát bộ nhớ cho tiến trình căn cứ trên nhu cầu bộ nhớ của chương trình. Có nhiều phương pháp quản lý bộ nhớ. Một vài phiên bản HĐH UNIX còn có cơ chế hoán chuyển: khi xuất hiện nhiều yêu cầu cấp phát CPU hoặc cấp phát bộ nhớ (do nhiều tiến trình khác nhau yêu cầu) thì HĐH sẽ thu hồi vùng nhớ của một vài tiến trình (trạng thái của tiến trình cũng như hình ảnh tiến trình trong bộ nhớ sẽ được chuyển ra lưu tạm trên thiết bị lưu trữ ngoài). Các thao tác này "trong suốt" với tiến trình, nhưng thỉnh thoảng người dùng vẫn có thể thấy khi tốc độ hệ thống suy giảm. Trong HĐH UNIX, cơ chế trừu tượng hóa cũng được áp dụng cho thiết bị. Giao diện với thiết bị cũng có các lời gọi open, close, read, write, lseek và ioctl giống như giao diện file. Các thao tác read/write thao tác trên luồng byte, vì thế thao tác đọc từ thiết bị cũng giống thao tác đọc file.

2.3. TIẾN TRÌNH

Tiến trình là chương trình đang được thi hành tuần tự. Tiến trình (minh họa trên Hình 2.1) bao gồm các thành phần sau:

- Đoạn mã cần thực thi.
- Dữ liệu để chương trình thực hiện các phép biến đổi.
- Tài nguyên cần thiết để thực thi chương trình.
- Trạng thái thực thi của tiến trình.



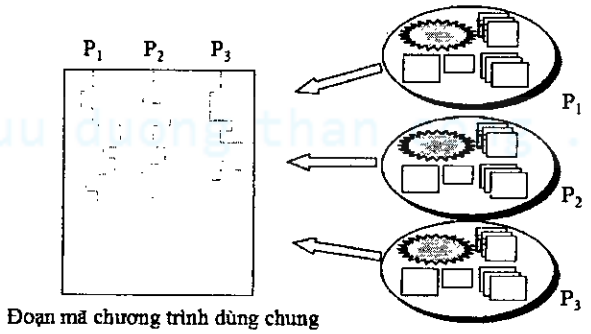
Hình 2.1. Môi trường của tiến trình

Để tiến trình thực thi cần có môi trường máy trừu tượng quản lý việc chia sẻ, cô lập tài nguyên giữa nhiều tiến trình. Trạng thái tiến trình được dùng để ánh xạ trạng thái của môi trường vào trạng thái vật lý của máy tính. Ví dụ về trạng thái tiến trình là chỉ thị nào trong đoạn mã chương trình hiện thời đang được thực hiện. Hình 2.1 minh họa môi trường máy trừu tượng quản lý các tiến trình và tài nguyên, cấp phát tài nguyên cho tiến trình khi có yêu cầu.

Cần phân biệt khái niệm *Chương trình* và *Tiến trình*. Chương trình là đối tượng tĩnh, cấu thành bởi các dòng lệnh, xác định hành vi của tiến trình khi xử lý trên một tập hợp dữ liệu nào đó. Tiến trình mang tính động, là chương trình đang được thực thi trên tập hợp dữ liệu cụ thể và sử dụng tài nguyên do hệ thống cung cấp. Nhiều tiến trình có thể cùng thực hiện một chương trình, nhưng mỗi tiến trình có dữ liệu và tài nguyên riêng như minh

họa trên Hình 2.2. Cụ thể hơn, mỗi tiến trình có bản ghi trạng thái lưu trữ các thông tin như chỉ thị nào đang được thực hiện, hay những tài nguyên được HĐH cấp phát. Tiến trình chỉ có thể được thực hiện khi đã có đủ tài nguyên cần thiết. Tiến trình có thể được mô tả qua mô hình thực thi. Cơ sở trình bày về tiến trình ở đây chỉ mang tính miêu tả và được giới hạn là chương trình chạy trên máy tính truyền thống.

HĐH có cấu trúc dữ liệu riêng để mô tả tiến trình. Khi tạo mới tiến trình, HĐH tạo ra bản ghi mô tả tiến trình tương ứng (bản ghi này được gọi là Khối điều khiển tiến trình). Khi tiến trình kết thúc, cấu trúc dữ liệu này cũng sẽ bị xóa bỏ. Bản mô tả tiến trình ở các HĐH có thể khác nhau, tuy nhiên, thường có trường định danh tiến trình; định danh các tài nguyên đã cấp phát cho tiến trình; giá trị các thanh ghi của CPU. Tiến trình được tham chiếu thông qua bộ mô tả.



Hình 2.2. Chia sẻ đoạn mã giữa nhiều tiến trình

Tiến trình là đơn vị tính toán nhỏ nhất trong máy tính, đơn vị cơ sở này được chương trình đang thực thi sử dụng để chia sẻ CPU. Mặc dù đơn vị tính toán là tiến trình, nhưng một số HĐH hiện đại có thể cài đặt hai đơn vị tính toán cơ sở khác là thread và object. Giữa thread và object không có mối quan hệ tường minh. Một số HĐH sử dụng thread để cài đặt object.

2.3.1. Tạo mới tiến trình

Khi khởi động, máy tính phải bắt đầu thực hiện các chỉ thị nằm trong bộ nhớ. Tiến trình đầu tiên này sẽ thực hiện nhiệm vụ tải bộ nạp (mục 4.2). Bộ nạp tiếp tục tải HĐH vào bộ nhớ trong. Sau đó, máy tính bắt đầu thực thi chương trình của HĐH. Vậy, các tiến trình tiếp theo được khởi tạo như thế

nào? Cách thông thường để tạo mới tiến trình là thực hiện lời gọi hệ thống ***spawn***. Một trường hợp riêng của *spawn* là *fork* – tạo ra một tiến trình mới để thực hiện chương trình giống như tiến trình đang thực thi. Năm 1963, Conway đưa ra ba hàm FORK, JOIN và QUIT. Những hàm cơ bản này được sử dụng để tạo mới và thực hiện một lớp tiến trình. Không giống các HĐH hiện đại, các tiến trình được tạo ra nhờ lời gọi FORK truyền thống thực hiện trong cùng không gian địa chỉ, có nghĩa là, chúng cùng chia sẻ bản sao của một chương trình và tất cả các thông tin. Khi thực hiện, tiến trình được tạo mới tham chiếu tới cùng một biến chứ không phải các biến riêng của mình. Hành vi của các câu lệnh được xác định như sau:

- **FORK(lable)**: Tạo ra một tiến trình mới (xác định bởi chính chương trình đang thực thi). Tiến trình này nằm trong cùng không gian địa chỉ với tiến trình gốc và bắt đầu thực thi từ chỉ thị có nhãn label. Tiến trình vừa thực hiện lệnh FORK, vừa thực hiện câu lệnh tiếp theo. Ngay khi tiến trình mới được tạo ra, hai tiến trình cha và con cùng tồn tại và thực thi.
- **QUIT()**: Kết thúc tiến trình. Tiến trình bị loại bỏ và HĐH xóa bộ mô tả tiến trình.
- **JOIN(count)**: Kết hợp hai hoặc nhiều tiến trình thành một. Thực thi lệnh này tương đương với thực thi đoạn mã sau:

```
count = count – 1; // Giảm biến count đi 1
if (count!=0) QUIT(); // Kết thúc nếu đây là tiến trình cuối cùng
```

  trên biến dùng chung count. Tại thời điểm bất kỳ chỉ duy nhất một tiến trình được phép thực hiện lệnh JOIN. Ngay khi tiến trình bắt đầu thực hiện lời gọi hệ thống JOIN, không tiến trình nào được quyền sử dụng CPU cho đến khi tiến trình này thực hiện xong lời gọi hệ thống. Đoạn mã này thực hiện phương pháp độc quyền truy xuất (sẽ được trình bày chi tiết trong Chương 6) do việc thực hiện lệnh JOIN không bị ngắt giữa chừng.

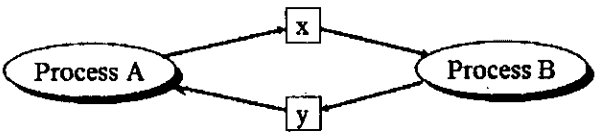
FORK, JOIN và QUIT được sử dụng để mô tả các công việc tính toán cấu thành các tiến trình tuần tự, phối hợp với nhau và thực hiện đồng thời trong cùng một không gian địa chỉ duy nhất. Chú ý là các tiến trình chia sẻ cả dữ liệu lẫn chương trình.

Tuy nhiên, các lời gọi tạo mới tiến trình trong HĐH hiện đại (gọi là fork, CreatProcess,...) tạo ra tiến trình con có không gian địa chỉ riêng. Phương pháp cũ tuy cho phép chia sẻ mã chương trình và dữ liệu giữa tiến trình con với tiến trình cha cũng như giữa các tiến trình anh em, nhưng chương trình quản lý bộ nhớ không thể cô lập bộ nhớ của các tiến trình. Tiến trình con có không gian bộ nhớ riêng, cho phép tách biệt hoàn toàn các tiến trình khác nhau. Quan trọng hơn, tiến trình con phải có khả năng thực hiện chương trình khác với chương trình của tiến trình cha. Nếu không thì tất cả tiến trình con sẽ giống một tiến trình đầu tiên (vì các tiến trình khác đều được khởi tạo gián tiếp hoặc trực tiếp từ tiến trình ban đầu). Do vậy, trong HĐH hiện đại phải có kỹ thuật để tiến trình con thực thi một chương trình cụ thể.

2.3.2. Cách sử dụng FOLK, JOIN, QUIT

Quan sát đoạn chương trình trong Hình 2.3. Tiến trình A (procA) tính toán một vài giá trị (trong <compute section A1>), sau đó cập nhật biến dùng chung x. Trong khi đó, tiến trình B bắt đầu thực hiện procB, mặc dù nhẽ ra B chưa được phép thực hiện câu lệnh retrieve(x) cho tới khi tiến trình A cập nhật xong x. Tương tự, A không được thực hiện retrieve(y) trên biến dùng chung y cho đến khi B hoàn thành update(y). Đoạn mã này phức tạp, bởi vì hai tiến trình thực hiện vòng lặp và tiến trình này có thể thực hiện vòng lặp nhanh hơn tiến trình kia. Khi đó giá trị trước của x hoặc y có thể bị mất, vì tiến trình có tốc độ thực thi nhanh có thể thay đổi giá trị trước khi giá trị ấy được tiến trình kia đọc.

```
procA() {                          procB() {
  while(TRUE) {                      while(TRUE) {
    <compute section A1>;              retrieve(x);
    update(x);                         <compute section B1>;
    <compute section A2>;              update(y);
    retrieve(y);                       <compute section B2>;
  }                                  }
}                                  }
```



Hình 2.3. Phối hợp giữa hai tiến trình

Bằng các hàm tạo mới nguyên thủy, hai tiến trình khác nhau A và B có thể thực hiện đồng thời và phối hợp với nhau để ngăn chặn việc thay đổi giá trị biến dùng chung trước khi đọc. Hình 2.4 minh họa việc thiết lập trình tự thực hiện của các đoạn mã lệnh. Việc gộp A và B vào một chương trình là kết quả việc sử dụng chung không gian địa chỉ của lệnh JOIN. Vì lời gọi hệ thống FORK sử dụng nhãn để xác định vị trí khởi đầu của tiến trình mới, nên chúng ta cũng phải sử dụng nhãn trong ngôn ngữ bậc cao.

Hành vi của tiến trình trong UNIX được xác định bởi đoạn mã chương trình, đoạn dữ liệu và đoạn ngăn xếp. Đoạn mã chương trình chứa những chỉ thị đã được biên dịch ra ngôn ngữ máy, đoạn dữ liệu chứa các biến tĩnh, đoạn ngăn xếp chứa ngăn xếp trong quá trình thực thi (được sử dụng để lưu trữ biến tạm thời). Nhiều file nguồn được dịch, biên dịch và liên kết thành một file khả thi với tên mặc định a.out (tất nhiên người lập trình có thể đặt bất kỳ tên nào cho file khả thi). File khả thi xác định ba đoạn của chương trình (Hình 2.5). Trong đoạn mã chương trình, địa chỉ lệnh rẽ nhánh và địa chỉ các thủ tục nằm trong địa chỉ đoạn mã chương trình. Dữ liệu tĩnh (biến khai báo trong chương trình C) nằm trong đoạn dữ liệu và cũng được định nghĩa trong file khả thi. Hệ thống tạo ra đoạn dữ liệu, khởi tạo giá trị cho các biến cũng như không gian lưu trữ cho các biến trong quá trình tiến trình thực thi. Đoạn ngăn xếp dùng để cấp phát bộ nhớ cho các biến động của chương trình.

```
L0:  count = 2;                L0:  count = 2;
     <compute section A1>;          <compute section A1>;
     update(x);                     update(x);
     FORK(L2);                      FORK(L2);
     <compute section A2>;          retrieve(y);
L1:  JOIN(count);                   <compute section B1>
     retrieve(y);                   update(y>;
     goto L0;                       FORK(L3)
L2:  retrieve(x);              L1:  JOIN(count);
     <compute section B1>;          retrieve(y);
     update(y);                     goto L0;
     FORK(L3);                 L2:  <compute section A2>;
     goto L1;                       goto L1;
L3:  <compute section B2>      L3:  <compute section B2>
     QUIT();                        QUIT();
```

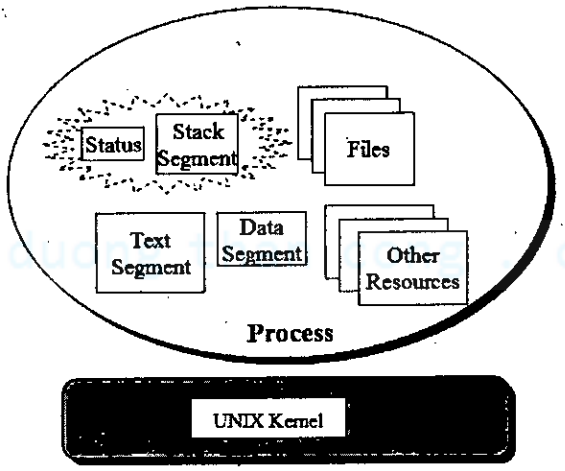
Hình 2.4. Sử dụng các hàm nguyên thủy

File khả thi được tạo ra bởi trình biên dịch, trình liên kết và bộ tải. File khả thi không định nghĩa tiến trình, mà định nghĩa đoạn mã chương trình và

khuôn dạng các thành phần dữ liệu mà tiến trình sẽ sử dụng trong quá trình thực hiện chương trình. Khi đoạn mã của tiến trình được tải vào bộ nhớ trong, hệ thống mới tạo ra đoạn dữ liệu và đoạn ngăn xếp tương ứng.

Tiến trình có duy nhất một định danh PID (process identifier), về bản chất là một con trỏ, một biến có giá trị nguyên – trỏ vào hàng nào đó trong bảng mô tả tiến trình nằm trong nhân UNIX. Mỗi hàng trong bảng này ứng với bản mô tả của tiến trình nào đó. Tiến trình sử dụng định danh tiến trình mình muốn tham chiếu tới làm tham số cho lời gọi tham chiếu. Lệnh ps của UNIX liệt kê tất cả các tiến trình trong hệ thống cùng với định danh tiến trình và định danh người sử dụng tạo ra tiến trình.

Trong UNIX, lời gọi *fork* tạo mới một tiến trình. Tiến trình cha sử dụng *fork* tạo ra tiến trình con có định danh xác định. Đoạn chương trình, đoạn dữ liệu và đoạn ngăn xếp của tiến trình con là bản sao của tiến trình cha. Tiến trình con có thể truy cập tới tất cả các file mà tiến trình cha mở. Tiến trình cha và tiến trình con thực hiện trong không gian địa chỉ riêng. Điều này có nghĩa là, dù chúng có cùng truy cập đến một biến, thì tiến trình con và tiến trình cha tham chiếu đến bản sao thông tin riêng của mình. Các tiến trình không chia sẻ không gian địa chỉ bộ nhớ, do đó tiến trình cha và tiến trình con không thể giao tiếp thông qua biến dùng chung. Trong UNIX, hai tiến trình chỉ có thể truyền thông với nhau thông qua file dữ liệu. Tạo mới một tiến trình UNIX được trình bày kỹ trong phần sau.



Hình 2.5. Tiến trình trong UNIX

Ngoài ra hệ thống UNIX còn cung cấp một số lời gọi hệ thống, chẳng hạn **execve** cho phép tiến trình có thể tải một chương trình khác vào không gian địa chỉ của mình.

Execve (char *path, char * argv[], char *envp[]);

Lời gọi hệ thống này cho phép chương trình xác định qua đường dẫn **path** thay thế chương trình hiện tại được tiến trình thực hiện. Sau khi **execve** thực hiện xong, chương trình gọi nó không còn được tải nữa. Không có khái niệm trở về sau **execve**, vì chương trình gọi không còn nằm trong bộ nhớ. Chương trình mới sử dụng danh sách tham số **argv** và các biến môi trường **envp**.

```
#include <sys/wait.h>
#define NULL 0
int main (void) {
        if (fork() == 0){ /* Đây là chương trình con */
                execve ("child", NULL, NULL);
                exit (0); /* Kết thúc*/
        }
        /* Đoạn mã của tiến trình con*/
        printf (" Process [%d]: Parent in execution...\n ", getpid ());
        sleep (2);
        if (wait (NULL) > 0) /* Tiến trình con kết thúc */
                printf (" Process [%d]: Parent detects terminating child \n",
getpid());
        printf (" Process [%d]: Parent terminating...\n",.getpid())
}
```

(a) Tiến trình cha

```
int main (void) {
        /* Tiến trình con thi hành chương trình mới (thay thế chương trình của
        tiến trình cha) */
                printf (" Process [%d]: child in execution ... \n", getpid ());
                sleep (1);
                printf (" Process [%d]: child terminating ... \n", getpid ());
}
```

(b) Tiến trình con

Hình 2.6. Tiến trình cha và con

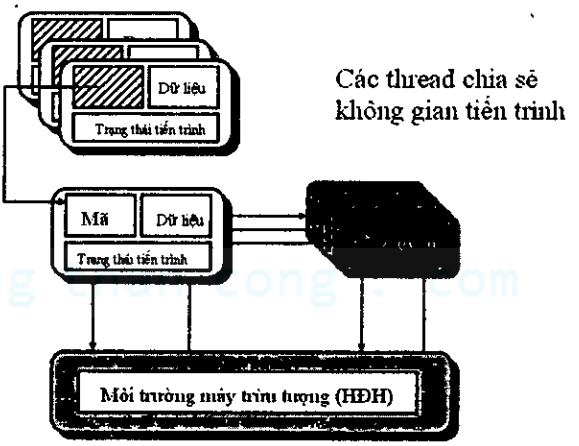
Lời gọi hệ thống **wait** (**waitpid** được sử dụng rộng rãi hơn) cho phép tiến trình cha xác định tiến trình con của mình đã hoàn thành chưa. Chi tiết trạng thái kết thúc của tiến trình con được trả về tiến trình cha thông qua giá trị tham chiếu của **wait**. Tuy nhiên, tiến trình cha có thể bỏ qua, không quan tâm đến tiến trình con. Lời gọi **wait** không phân biệt giữa các tiến trình con, trong khi các biến thể của lời gọi **waitpid** cho phép tiến trình cha có thể đợi một tiến trình con cụ thể (dựa trên PID) hoàn thành. Khi kết thúc, tài nguyên

của tiến trình bao gồm bản mô tả tiến trình trong nhân HĐH được giải phóng. HĐH báo hiệu cho tiến trình cha khi tiến trình con kết thúc, nhưng HĐH chưa xóa bản mô tả tiến trình con cho tới khi tiến trình cha nhận được tín hiệu. Tiến trình cha thực hiện lời gọi **wait** để ghi nhận mình đã biết tiến trình con hoàn thành.

Chương trình cha trong Hình 2.6a minh họa cách sử dụng **execve** và **wait**. Trong ví dụ này, tiến trình cha tạo ra tiến trình con bằng lời gọi **fork**, sau đó thực hiện lời gọi **printf** và **sleep**. Sau khi gọi **fork**, tiến trình con thực hiện **execve** để thay thế chương trình trong Hình 2.6a bằng chương trình trong Hình 2.6b. Sau khi tiến trình con thực hiện xong **execve** trong Hình 2.6a thì câu lệnh được thực hiện tiếp theo là câu lệnh đầu tiên trong Hình 2.6b.

2.4. LUỒNG

Mô hình luồng cho phép nhiều thực thể cùng thực hiện một chương trình, đồng thời sử dụng chung các file và thiết bị. Bên cạnh tiến trình, luồng là kiểu đơn vị tính toán cơ sở mà HĐH có thể điều phối. Trong mô hình này, tiến trình là thực thể mang tính trừu tượng, được cấp phát các kiểu tài nguyên khác nhau, tuy nhiên không có thành phần thực hiện chương trình. Thread (đôi khi được gọi là tiểu trình) là thực thể thực hiện chương trình và sử dụng các tài nguyên của tiến trình tương ứng. Tiến trình có thể có nhiều luồng. Các thread anh em là các thread nằm trong cùng một tiến trình, chia sẻ chương trình và tài nguyên của tiến trình. Trong mô hình thread, tiến trình theo kiểu truyền thống là tiến trình có duy nhất một thread thực hiện chương trình.



Hình 2.7. Các thread trong một process

Việc điều phối thread tương tự điều phối tiến trình trong mô hình truyền thống, thực hiện chuyển đổi quyền điều khiển CPU giữa các thread cạnh tranh. Trong một vài hệ thống, bộ điều phối thread là chương trình người dùng, tuy nhiên trong nhiều trường hợp, bộ điều phối được tích hợp bên trong HĐH. Số lượng thông tin trạng thái của thread không nhiều, do đó việc điều phối thread đơn giản hơn điều phối tiến trình. Động lực quan trọng nhất của việc sử dụng thread là giảm thiểu thời gian chuyển ngữ cảnh giữa các thread. Điều này cho phép chuyển quyền sử dụng CPU giữa các đơn vị tính toán cơ sở (thread) với chi phí quản lý phụ trội thấp nhất. Tương tự, bộ quản lý thread có bản mô tả thread ghi lại giá trị các thanh ghi và nội dung ngăn xếp cho từng thread. Vì tất cả các thread của tiến trình chia sẻ một chương trình dùng chung, chương trình này sẽ định nghĩa hành vi của tiến trình. Lập trình viên dễ dàng xây dựng chương trình có nhiều thread tương tác với nhau để quản lý tài nguyên dùng chung đã được cấp phát cho tiến trình cha.

Các hệ thống giao diện đồ họa dưới dạng cửa sổ thường sử dụng mô hình thread để quản lý nhiều cửa sổ trên cùng một màn hình. Giả sử hệ thống cửa sổ được xây dựng thông qua một tiến trình quản lý màn hình vật lý và có nhiều thread (ứng với cửa sổ). Tất cả các thread chạy cùng một đoạn mã, cùng chia sẻ màn hình vật lý, nhưng mỗi thread quản lý một cửa sổ riêng. Trong kiểu hệ thống này, thời gian đáp ứng rất quan trọng, do đó phải giảm thiểu thời gian chuyển ngữ cảnh giữa các cửa sổ.

Ngày nay, thread là cơ chế quan trọng khi lập trình song song. Tuy thực hiện trong không gian địa chỉ của tiến trình, nhưng thread lại là đơn vị tính toán độc lập và bộ điều phối có thể kiểm soát mức độ tiến triển của từng thread. Các thread cùng nhau chia sẻ tài nguyên được cấp phát cho tiến trình cha. Thread là một cách đơn giản để lọc và chia công việc trong tiến trình.

Thread trong C:

Thread có thể được hỗ trợ ở mức nhân của HĐH hoặc ở mức thư viện. Ở mức thư viện, package thread trong thư viện POSIX được sử dụng rộng rãi vì là một phần của chuẩn POSIX. Các package này có giao diện chương trình tương tự package process của UNIX, mặc dù kernel UNIX chưa chắc đã hỗ trợ thread (Sun Solaris và Linux hỗ trợ chế độ thread qua giao diện POSIX).

Đoạn mã sau minh họa cách thức chương trình tạo ra thread bằng cách sử dụng thư viện, thread con được tạo ra qua hàm **Chread_fork**. Sau khi sinh ra, thread con dùng chung các biến tĩnh với thread cha và với các thread anh em. Các thread con có thể trì hoãn thread cha, hoặc ngược lại thông qua lời gọi **Ctheard_yield**.

```
#include <cthreads.h>
...
int main (int argc, char *argv[])
{
    t_handle = cthread_fork (tChild, I);
    /* A child thread is now executing the tChild function */
    Cthread_yield (); /* Yield to another thread */
}

void tChild (int me)
{
    /* This function is executed by the child */
    ...
    Cthread_yield (); /* Yield to another thread */
    ...
}
```

cuu duong than cong . com

2.5. ĐỐI TƯỢNG

Ý tưởng đối tượng xuất phát từ ngôn ngữ lập trình mô phỏng. Đối tượng là thực thể có tính tự trị nằm trong hệ thống. Chương trình mô phỏng có thể coi như chương trình quản lý một số lượng lớn các đơn vị tính toán riêng biệt, mỗi đơn vị tính toán thực hiện một nhiệm vụ nào đó tại một thời điểm cụ thể và các đơn vị tính toán có thể có quan hệ với nhau. Ngôn ngữ lập trình mô phỏng Simula 67 đưa ra ý tưởng về lớp (class), dùng để định nghĩa hành vi của đơn vị tính toán mô phỏng, giống như chương trình định nghĩa hành vi tiến trình. Định nghĩa lớp bao gồm các phương tiện cho phép đối tượng khai báo dữ liệu riêng của mình. Như vậy, lớp giống một kiểu dữ liệu trừu tượng có trạng thái riêng (tập hợp giá trị của các biến riêng) và được thực hiện như một đơn vị tính toán tự trị. Quá trình mô phỏng được định nghĩa bằng cách xác định tập thể hiện của các lớp, là các đối tượng và giao tiếp giữa các đối tượng thông qua việc gửi thông điệp.

Đối tượng chỉ phản ứng lại với các thông điệp. Sau khi tạo ra, đối tượng có thể nhận thông điệp từ các đối tượng khác. Đối tượng phản ứng lại bằng cách thực hiện tính toán trên dữ liệu của mình và sau đó gửi thông điệp trả lời. Do hành vi của đối tượng được xác định thông qua định nghĩa lớp, lập trình viên định nghĩa các lớp và cách thức khởi tạo đối tượng từ định nghĩa lớp. Ngày nay, hướng đối tượng được sử dụng trong tất cả các ứng dụng. Ngôn ngữ lập trình hướng đối tượng đã đưa ra mô hình lập trình ứng dụng mới. Bởi tính phổ biến của mô hình này, một số HĐH hiện nay được cài đặt bằng cách sử dụng đối tượng (HĐH Spring). Hướng đối tượng ngày càng có vai trò quan trọng trong các HĐH hiện đại.

2.6. NHẬN XÉT

Đối với lập trình viên ứng dụng, HĐH là môi trường tính toán cấu thành bởi tiến trình, file và các tài nguyên khác. Tiến trình là đơn vị tính toán nhỏ nhất mà hệ thống có thể điều phối được, nó biểu diễn sự thực thi của chương trình. File được sử dụng để lưu trữ thông tin giữa các phiên làm việc. Tất cả HĐH đều cho phép tổ chức file dưới dạng một luồng byte tuần tự, tuy nhiên, một số HĐH còn hỗ trợ nhiều loại file có cấu trúc phức tạp. Các tài nguyên khác bao gồm CPU, bộ nhớ, thiết bị và bất kỳ thành phần nào mà tiến trình phải yêu cầu HĐH cung cấp. Các tài nguyên, đặc biệt là file sẽ được hệ thống kiểm soát và tiến trình phải yêu cầu cấp phát trước khi thực thi. Tiến trình có chương trình xác định hành vi của mình, tài nguyên cần thiết để hoạt động và dữ liệu sẽ thao tác trên đó.

CÂU HỎI ÔN TẬP

1. Phân biệt tiến trình với chương trình.
2. Định nghĩa tài nguyên của hệ thống.
3. Trình bày ưu điểm của luồng so với tiến trình.

Chương 3
CẤU TRÚC HỆ ĐIỀU HÀNH

Chương 2 trình bày quan điểm của người lập trình ứng dụng về HĐH và nghiên cứu HĐH dưới góc độ người lập trình hệ thống. Chương 3 trình bày các đặc điểm chung của HĐH. Các chương sau đề cập đến từng vấn đề cụ thể của HĐH. Trước tiên sẽ phân tích những nhân tố chính ảnh hưởng đến quá trình thiết kế HĐH. Sau đó sẽ trình bày các chức năng cơ bản của HĐH như quản lý thiết bị, quản lý tài nguyên, quản lý bộ nhớ, quản lý file. Phần cuối trình bày những công nghệ được HĐH sử dụng để thực hiện những chức năng trên.

3.1. PHÂN TÍCH CÁC YẾU TỐ TÁC ĐỘNG ĐẾN HỆ ĐIỀU HÀNH

HĐH trừu tượng hóa các tài nguyên máy tính, giúp lập trình viên phát triển chương trình ứng dụng, ví dụ tiến trình là sự trừu tượng hóa hoạt động của bộ vi xử lý; file là sự trừu tượng hóa của thiết bị lưu trữ. Trong quá trình tính toán, trạng thái tiến trình thay đổi liên tục khi sử dụng các tài nguyên. HĐH cung cấp các hàm để tiến trình tạo mới hoặc kết thúc tiến trình khác; yêu cầu, sử dụng hoặc giải phóng tài nguyên; phối hợp hành động với những tiến trình có liên quan. Ngoài ra, HĐH cần có khả năng quản lý và cấp phát tài nguyên theo yêu cầu; hỗ trợ chia sẻ và khi chia sẻ phải có phương thức kiểm soát sao cho không gây nên bất kỳ sự cố nào. Bên cạnh những yêu cầu quản lý cơ bản, các yếu tố sau đây cũng tác động đến quá trình thiết kế và phát triển của HĐH:

- Hiệu suất.
- Bảo vệ và an ninh.

- Tính chính xác.
- Khả năng bảo trì.
- Thương mại.
- Chuẩn và Hệ thống mở.

3.1.1. Hiệu suất

Ở mức thấp nhất, HĐH cung cấp giao diện lập trình và cơ chế quản lý việc chia sẻ tài nguyên. Hai chức năng trên đóng vai trò quản lý vì không trực tiếp giải quyết vấn đề của người sử dụng, mà chỉ tạo ra môi trường giúp chương trình ứng dụng thực thi. Chức năng quản lý tài nguyên giúp sử dụng hệ thống dễ dàng hơn, nhưng lại đòi hỏi một chi phí phụ trội nào đó. Ví dụ, giao diện trừu tượng giúp cho việc viết chương trình dễ dàng hơn, nhưng lại làm chậm tốc độ thực thi của chương trình. Ví dụ, tốc độ thực hiện thao tác lên file chậm hơn nhiều so với thao tác trực tiếp lên ổ đĩa cứng. Khi đưa chức năng mới vào HĐH, phải đánh giá xem chức năng thêm mới vào có ảnh hưởng tới hiệu suất tổng thể của hệ thống hay không. Nhiều ràng buộc về hiệu suất đã ngăn cản việc tích hợp thêm tính năng mới vào HĐH. Do phần cứng ngày càng mạnh, các nhà thiết kế đã tích hợp thêm nhiều chức năng vào HĐH và bỏ qua vấn đề suy giảm hiệu suất. Tích hợp các ngôn ngữ lập trình bậc cao, đối tượng, chức năng bộ nhớ ảo, đồ họa và kết nối mạng vào HĐH minh chứng cho xu thế này. Không có tiêu chí rõ ràng để xác định rằng, liệu một chức năng với chi phí cài đặt cao có nên đưa vào HĐH không. Vấn đề này sẽ được giải quyết dựa trên từng tình huống cụ thể và phải được phân tích chi tiết, liệu tính ưu việt của chức năng mới có đáng với hiệu suất bị suy giảm hay không.

3.1.2. Bảo vệ và an ninh

HĐH đa chương trình cho phép nhiều tiến trình thực thi tại cùng một thời điểm. Như vậy, HĐH phải có cơ chế ngăn cản không cho tiến trình tác động lên hoạt động của tiến trình khác. Tiến trình cũng không được phép sử dụng tài nguyên không hợp lệ. Do đó, HĐH phải có khả năng cấp phát riêng tài nguyên cho tiến trình, hoặc cấp phát tài nguyên cho nhiều tiến trình dùng chung. Ngoài ra, HĐH phải có cơ chế kiểm soát bằng phần mềm để vừa bảo đảm cơ chế cô lập nhưng vẫn có khả năng chia sẻ tài nguyên. Cơ chế bảo vệ

là công cụ để HĐH triển khai biện pháp an ninh được người quản trị hệ thống thiết lập. Biện pháp an ninh định nghĩa phương pháp quản lý việc truy cập tới tài nguyên. Chẳng hạn, tại một thời điểm chỉ cho phép duy nhất một tiến trình được quyền mở file để ghi, nhưng cho phép nhiều tiến trình có thể mở file để đọc. Cơ chế bảo vệ file có thể cài đặt biện pháp này thông qua cơ chế khóa đọc và khóa ghi file. Cơ chế bảo vệ thường được cài đặt trong HĐH. Tuy nhiên, có vấn đề nảy sinh trong quá trình thiết kế: Nếu sau khi HĐH thiết lập một biện pháp thì làm thế nào để ngăn cản phần mềm ứng dụng thay đổi nó? Đây là một vấn đề quan trọng trong các HĐH hiện đại. Bảo vệ tài nguyên là một lĩnh vực cụ thể trong nghiên cứu về HĐH (Chương 12). Tuy nhiên, cũng như vấn đề hiệu suất, tính năng này quan trọng đến mức mọi HĐH đều phải xét đến yếu tố an ninh khi đưa bất kỳ tính năng mới nào vào HĐH. Như sẽ trình bày trong mục 3.3, trên thực tế vấn đề này được giải quyết là phần cứng phải có khả năng phân biệt được giữa phần mềm HĐH và phần mềm ứng dụng.

3.1.3. Tính chính xác

cuu duong than cong . com

Một số phần mềm có thể được coi là "tin cậy", nhưng một số phần mềm bị coi là "không tin cậy". Phương pháp bảo vệ của hệ thống phụ thuộc nhiều vào các thao tác chính xác của phần mềm HĐH "tin cậy". Mỗi chức năng phải có những yêu cầu cụ thể. Điều này cho phép nhà thiết kế có thể nói chức năng X, dưới điều kiện Y có hoạt động chính xác không. Ví dụ, không thể xác định được bộ điều phối có hoạt động chính xác không, nếu không biết bộ điều phối cần phải thực hiện công việc gì. Nói chung, rất khó đưa ra yêu cầu cụ thể cho phần mềm HĐH. Có một nhánh nghiên cứu HĐH thực hiện đánh giá thiết kế và triển khai thiết kế có đáp ứng được yêu cầu hay không. Những nhà thiết kế HĐH khác chỉ sử dụng những công cụ hình thức để chứng minh phần mềm hệ thống đáng tin cậy. Tính chính xác là yếu tố hết sức cơ bản phải được cân nhắc khi muốn tích hợp thêm chức năng vào HĐH.

3.1.4. Khả năng bảo trì

Vào những năm 1960, HĐH đã phức tạp đến mức không ai có thể hiểu mọi dòng chương trình trong mã nguồn của nó. Bên cạnh mối quan tâm về tính chính xác, một vấn đề mới nảy sinh là: Làm thế nào để thay đổi phần mềm HĐH nhưng vẫn đảm bảo độ tin cậy, độ chính xác của kết quả và

không phát sinh lỗi mới? Một nhóm những nhà thiết kế có xu hướng cho phép sản phẩm của mình dễ bảo trì, mà không chú ý nhiều đến yếu tố phổ dụng và hiệu suất hệ thống.

3.1.5. Thương mại

Phần lớn các HĐH thương mại hiện đại được phát triển trên nền tảng HĐH đa chương trình chia sẻ thời gian có kết hợp khả năng kết nối mạng. UNIX là HĐH chia sẻ thời gian và những phiên bản đầu tiên của BSD UNIX hay AT&T vẫn tiếp tục là các hệ thống chia sẻ thời gian. UNIX chiếm phần lớn thị phần trong môi trường đa chương trình (máy tính cá nhân và máy trạm). Mặt khác, môi trường máy tính cá nhân bị ràng buộc bởi các sản phẩm của Microsoft (trước kia là DOS và bây giờ là Microsoft Windows), có thể là do quan hệ hợp tác chặt chẽ giữa Microsoft với hãng IBM sản xuất phần cứng. Ngày nay, công nghệ phần cứng máy tính cá nhân hội tụ dần với công nghệ máy trạm và khi đó phải hỗ trợ đa chương trình trên máy tính cá nhân. Vấn đề đặt ra ở đây là, trong khi UNIX hỗ trợ đa chương trình, thì HĐH Windows lại được sử dụng rộng rãi hơn. Điều này khiến người lập trình và người sử dụng phải lựa chọn hoặc HĐH đa nhiệm tương thích với DOS, chẳng hạn Windows XP hay Windows NT, hoặc là HĐH thương mại UNIX nào đó. Có thể sau này, các HĐH thương mại sẽ hội tụ vào một giải pháp duy nhất hoặc thị trường vẫn tiếp tục hỗ trợ nhiều HĐH. Trong cả hai trường hợp, thị trường và các yếu tố thương mại chứ không phải các yếu tố công nghệ sẽ là nhân tố ảnh hưởng chính.

Sự thành công của HĐH UNIX và Microsoft Windows ảnh hưởng lớn đến quá trình phát triển của HĐH nói chung. Để được thị trường chấp nhận, HĐH mới phải có ngôn ngữ lập trình (chương trình dịch, trình kết nối và bộ tải), trình soạn thảo văn bản và thư viện runtime. Những môi trường tính toán hiện đại có nhiều công cụ và ứng dụng, tất cả đều viết trên nền các HĐH thông dụng. Do đó, bên cạnh những cải tiến, HĐH mới phải cung cấp môi trường cho phép thực thi các ứng dụng có sẵn.

3.1.6. Chuẩn và hệ thống mở

Những thay đổi về việc sử dụng máy tính trong tổ chức doanh nghiệp diễn ra vào cuối những năm 1980. Trước thời điểm này, các tổ chức thường

mua tất cả thiết bị máy tính từ cùng một nhà sản xuất. Tuy nhiên, theo quy luật kinh tế, người dùng cuối sẽ lợi hơn rất nhiều nếu có thể mua được thiết bị trong một thị trường mở và cạnh tranh. Nhu cầu sử dụng thiết bị từ nhiều nhà sản xuất khác nhau chính là động lực phát triển công nghệ Hệ thống mở. Điều này cho phép các doanh nghiệp có thể sử dụng máy tính, HĐH và ứng dụng từ nhiều nhà sản xuất khác nhau. Hệ thống mở tác động mạnh mẽ đến sự thành công của các doanh nghiệp sản xuất thiết bị công nghệ thông tin (CNTT). Mục tiêu của kiến trúc hệ thống mở là cho phép người dùng sau làm việc trên một mạng máy tính với nhiều chủng loại thiết bị khác nhau.

Cần ít nhất ba chiến lược đối với hệ thống mở:

- **Tích hợp ứng dụng:** Giao diện người dùng của tất cả các chương trình ứng dụng nên giống nhau. Các tiện ích quản lý thông tin và thiết bị cần được chuẩn hóa sao cho tạo nên giao diện nhất quán với người sử dụng.
- **Khả năng tương thích:** Các chương trình ứng dụng phải có khả năng cài đặt trên nhiều nền tảng phần cứng khác nhau.
- **Khả năng liên tác:** Các tiện ích trong môi trường mạng được chuẩn hóa sao cho đơn giản hóa việc truy cập tới các máy tính khác.

cuu duong than cong . com

Mục tiêu của chuẩn POSIX là giải quyết phần lớn những khía cạnh của hệ thống mở. Cụ thể, POSIX.1 chuẩn hóa giao diện của chương trình ứng dụng với HĐH chứ không phải cách thức cài đặt UNIX. Chuẩn này khuyến khích các hãng sản xuất khác nhau sử dụng cùng một giao diện POSIX, khi đó các ứng dụng viết ra có khả năng chạy trên nhiều HĐH UNIX khác nhau. Phần lớn HĐH UNIX tuân theo chuẩn này.

3.2. CÁC CHỨC NĂNG CƠ BẢN

cuu duong than cong . com

Bên cạnh nhiệm vụ trừu tượng hóa quá trình tính toán và quản lý tài nguyên hệ thống, HĐH cần quan tâm đến nhiều khía cạnh thực tế. Chẳng hạn hiệu suất, an ninh, tính chính xác, tính dễ bảo trì của hệ thống. Nói chung, không thống nhất được HĐH cần có những chức năng gì, do đó trong giáo trình này chỉ trình bày những chức năng cơ bản đã được thừa nhận rộng rãi, đó là: quản lý thiết bị; quản lý tiến trình và tài nguyên; quản lý bộ nhớ và quản lý file.

3.2.1. Quản lý thiết bị

Ngoại trừ CPU và bộ nhớ trong, phần lớn HĐH coi tất cả các thiết bị khác là giống nhau. Chương trình quản lý thiết bị quy định cách thức sử dụng một chủng loại thiết bị. Nói chung, nhiệm vụ của HĐH là cấp phát, cô lập và chia sẻ thiết bị theo chính sách định trước. Thậm chí HĐH không hỗ trợ chế độ đa chương trình cũng phải có trình quản lý thiết bị. Trước kia, mã nguồn của HĐH được cung cấp cùng phần cứng. Nếu muốn kết nối thêm thiết bị vào máy tính, người sử dụng phải cài thêm driver của thiết bị vào HĐH. Nếu không có mã nguồn HĐH thì không thể biên dịch lại HĐH để gắn thêm driver mới. Hạn chế này thúc đẩy sự phát triển khả năng cấu hình lại driver trong HĐH hiện đại. Driver của thiết bị có thể được biên dịch và cài đặt thêm vào HĐH mà không cần dịch lại HĐH. Tuy quản lý thiết bị là một phần quan trọng, nhưng lại tương đối đơn giản trong thiết kế HĐH. Nội dung về quản lý thiết bị được trình bày trong Chương 8.

3.2.2. Quản lý tiến trình và tài nguyên

Tiến trình là đơn vị tính toán cơ sở, được người lập trình định nghĩa, còn tài nguyên là các thành phần trong môi trường tính toán mà tiến trình cần có để thực thi. Quản lý tiến trình và quản lý tài nguyên có thể nằm tách biệt, nhưng đa số HĐH kết hợp lại trong một module. Trong Chương 2 đã lấy mô hình tiến trình của HĐH UNIX minh họa cách thức định nghĩa một môi trường tính toán. HĐH UNIX cho phép tạo mới, hủy, phong tỏa và thực thi một tiến trình. Tương tự, HĐH có hỗ trợ luồng, hay hướng đối tượng (Window NT) cung cấp môi trường cho phép quản lý những đơn vị tính toán cơ sở tương ứng. Thành phần quản lý tài nguyên có trách nhiệm cấp phát tài nguyên (nếu có) cho các tiến trình có nhu cầu.

Bộ phận này cho phép nhiều người dùng (hoặc nhiều tiến trình) chia sẻ máy tính, bằng cách cấp phát CPU luân phiên giữa các tiến trình để mỗi tiến trình có thể sử dụng CPU trong khoảng thời gian phù hợp. Vấn đề chính của việc quản lý tiến trình và tài nguyên là làm thế nào để cô lập việc truy cập tài nguyên của các tiến trình (theo chính sách định trước) và làm thế nào để các tiến trình vượt qua cơ chế cô lập khi có chính sách chia sẻ tài nguyên giữa nhiều tiến trình. Cơ chế cấp phát tài nguyên phải kết hợp chặt chẽ với tiện ích quản lý tiến trình và tài nguyên. Cơ chế này bao gồm việc biểu diễn

tài nguyên; thực hiện cấp phát và sử dụng tài nguyên theo chính sách định trước. Quản lý tiến trình và tài nguyên được trình bày trong Chương 8 và 9.

3.2.3. Quản lý bộ nhớ

Chương trình quản lý bộ nhớ chịu trách nhiệm quản lý và cấp phát tài nguyên bộ nhớ chính. Tiến trình yêu cầu và sử dụng bộ nhớ theo định nghĩa của chương trình tương ứng. Bộ phận quản lý bộ nhớ cấp phát theo chính sách định trước. Chia sẻ khiến vấn đề thiết kế phức tạp hơn, vì chương trình quản lý bộ nhớ phải tích hợp cả cơ chế cô lập (để tiến trình không được truy cập vào không gian bộ nhớ của tiến trình khác) lẫn cơ chế cho phép các tiến trình có thể chia sẻ vùng nhớ chung.

HĐH hiện đại còn có công nghệ bộ nhớ ảo (mở rộng bộ nhớ chính lớn hơn giới hạn kích thước vật lý bằng cách sử dụng thêm thiết bị lưu trữ ngoài), cho phép tiến trình tham chiếu đến phần bộ nhớ lưu trên ổ đĩa cứng như thể đó là bộ nhớ trong. Quản lý bộ nhớ ảo phức tạp hơn nhiều so với quản lý bộ nhớ truyền thống, vì phải kết hợp chính sách quản lý bộ nhớ trong và chính sách quản lý ổ đĩa cứng. Các chương trình quản lý bộ nhớ trên HĐH hiện đại thậm chí còn cho phép tiến trình có thể truy cập và chia sẻ bộ nhớ vật lý của một máy tính khác. Xây dựng nên bộ nhớ ảo dùng chung phân tán bằng cách cho phép các tiến trình trao đổi thông điệp trên đường truyền kết nối các máy tính. Khi đó, chương trình quản lý bộ nhớ kết hợp các chức năng nguyên thủy của mình với chức năng kết nối mạng.

3.2.4. Quản lý file

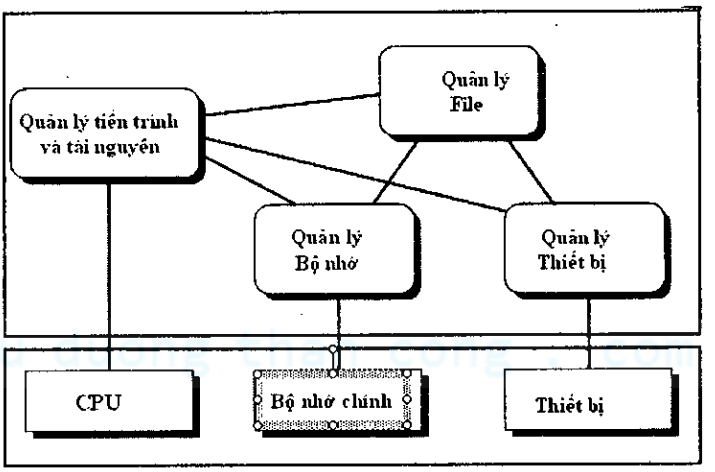
File là sự trừu tượng hóa của thiết bị lưu trữ. Thông tin nằm trên bộ nhớ chính sẽ bị ghi đè nếu khu vực bộ nhớ đó được cấp phát cho tiến trình khác. Nếu muốn lưu trữ lâu dài, dữ liệu cần được ghi ra thiết bị lưu trữ ngoài (chẳng hạn ổ đĩa). Như đã trình bày trong Chương 2, chính nhu cầu trừu tượng hóa các thao tác vào/ra trên thiết bị lưu trữ ngoài là một trong những động lực phát triển của HĐH. Chính vì vậy, file được xem là giao diện trừu tượng quan trọng nhất của HĐH.

Các bộ phận quản lý file khác nhau có các giao diện trừu tượng khác nhau, từ dạng đơn giản (như mô hình thiết bị lưu trữ dưới dạng một luồng byte) tới hết sức phức tạp (chẳng hạn mô hình cơ sở dữ liệu quan hệ). Trong

HĐH hiện đại, hệ thống file được cài đặt phân tán, cho phép tiến trình không chỉ truy cập dữ liệu trên máy tính của mình mà còn có thể truy cập tới dữ liệu trên các máy tính khác thông qua mạng máy tính. Chương 11 sẽ trình bày cách thức sử dụng tiến trình tương tác với hệ thống file thông qua giao diện lập trình ứng dụng.

3.2.5. Kết hợp các chức năng

Phần này trình bày cách thức kết hợp các chức năng cơ bản trong HĐH. Hình 3.1 minh họa quan hệ giữa các module cơ bản (kết nối giữa các module chỉ quan hệ tương tác giữa chúng). Bộ phận quản lý tiến trình và tài nguyên tạo nên tiến trình và môi trường thực thi trên nền CPU. Bộ phận này sử dụng các giao diện trừu tượng do nhiều thành phần quản lý tài nguyên khác cung cấp. Khác với các bộ phận quản lý tài nguyên khác, quản lý bộ nhớ được xếp riêng như một bộ phận độc lập của HĐH. Khi công nghệ bộ nhớ ảo thông dụng, trách nhiệm của bộ phận quản lý bộ nhớ cũng tăng lên. Quản lý file thực hiện việc trừu tượng các thao tác xuất/nhập trên thiết bị thành các thao tác đơn giản, dễ sử dụng. Bộ phận quản lý thiết bị điều khiển thao tác đọc/ghi trên thiết bị lưu trữ thứ cấp và được cài đặt dưới dạng trình điều khiển thiết bị.



Hình 3.1. Quan hệ giữa các chức năng trong tổ chức HĐH

Cấp phát tài nguyên là trách nhiệm của bộ phận quản lý tiến trình và tài nguyên. Bộ phận quản lý file sử dụng tiện ích đọc/ghi thiết bị do bộ phận quản lý thiết bị cung cấp. Các chương trình quản lý thiết bị có thể đọc/ghi

trực tiếp vào bộ nhớ chính, nên bộ phận quản lý file có quan hệ chặt chẽ với bộ phận quản lý bộ nhớ, đặc biệt trong hệ thống hỗ trợ bộ nhớ ảo. Vì thế, hệ thống tách rời bốn module để cô lập các chức năng, nhưng các module này vẫn gắn kết chặt chẽ với nhau.

3.3. CÁC PHƯƠNG THỨC CÀI ĐẶT HỆ ĐIỀU HÀMH

Các HĐH hiện đại cài đặt theo một trong ba cơ chế cơ bản sau:

- **Chế độ vi xử lý (Processor mode)**: Sử dụng bit chế độ để phân biệt giữa tiến trình HĐH hay tiến trình người dùng.
- **Nhân HĐH (Kernel)**: Tất cả các bộ phận chủ yếu của HĐH được đặt trong nhân. Kernel là module phần mềm cực kỳ đáng tin cậy, hỗ trợ tất cả các phần mềm khác hoạt động.
- **Phương thức yêu cầu dịch vụ hệ thống**: Vấn đề này liên quan tới cách thức tiến trình người sử dụng yêu cầu dịch vụ của HĐH, bằng cách gọi hàm hệ thống hay gửi thông điệp tới tiến trình hệ thống.

cuu duong than cong . com

3.3.1. Chế độ của bộ vi xử lý

CPU hiện đại thường có bit chế độ để xác định khả năng thực hiện của tiến trình trên CPU. Bit này có thể được thiết lập ở chế độ supervisor (giám sát toàn bộ hệ thống) hay ở chế độ người dùng. Trong chế độ giám sát, CPU có thể thực hiện bất kỳ chỉ thị nào. Trong chế độ người dùng, CPU chỉ thực hiện được một số chỉ thị nhất định. Những chỉ thị chỉ có thể thực thi ở chế độ giám sát, được gọi là chỉ thị giám sát hay chỉ thị đặc quyền để phân biệt với chỉ thị thường.

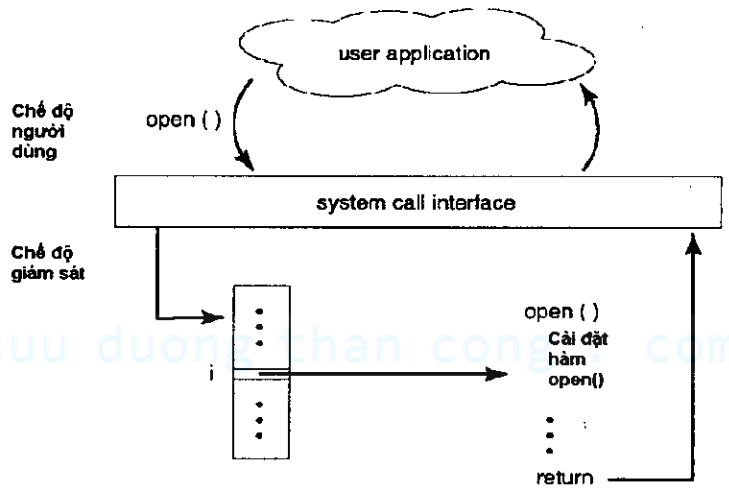
Chỉ thị vào/ra (I/O) là chỉ thị đặc quyền, vì vậy chương trình ứng dụng nếu thực thi trong chế độ người dùng không thể thực hiện thao tác vào/ra. Chương trình ứng dụng phải yêu cầu HĐH thay mặt mình thực hiện vào/ra. Khi đó, một chỉ thị phần cứng đặc biệt sẽ chuyển CPU sang chế độ giám sát và gọi driver thiết bị (driver sẽ thực hiện chỉ thị vào/ra). Bit chế độ cũng được sử dụng để giải quyết các tình huống liên quan đến an ninh hệ thống đã đề cập trong mục 3.1. Các chỉ thị có thể thay đổi trạng thái bảo vệ hiện thời của hệ thống, được gọi là chỉ thị đặc quyền. Ví dụ, HĐH sử dụng một số thanh ghi trong CPU hay những vùng nhớ đặc biệt lưu trữ trạng thái tiến

trình, con trỏ đến các nguồn tài nguyên. Khi muốn thay đổi nội dung vùng nhớ hay thanh ghi này, phải sử dụng những chỉ thị load và store đặc quyền.

Các dòng máy tính cũ như CPU Intel 8088/8086 không có bit chế độ. Do vậy, chúng không phân biệt được chỉ thị đặc quyền và chỉ thị người dùng. Kết quả là không thể cung cấp cơ chế cô lập bộ nhớ trên những máy tính này, bất kỳ chương trình người dùng nào cũng có thể tải một giá trị bất kỳ vào thanh ghi cơ sở đoạn (base segment register). Tiến trình có thể truy xuất tới bất kỳ đoạn bộ nhớ nào.

Những CPU trong họ Intel sau này đều có bit chế độ, vì vậy, chỉ có thể thay đổi giá trị thanh ghi cơ sở bằng các chỉ thị đặc quyền. Các CPU Intel đời mới tương thích "ngược" với CPU dòng 8088/8086 để phần mềm viết trên dòng CPU cũ có thể thực thi được trên hệ thống mới.

Hệ thống có thể mở rộng bit chế độ để xác định những vùng nhớ nào được sử dụng khi CPU trong chế độ giám sát và khi trong chế độ người dùng (Hình 3.2). Nếu bit chế độ được chuyển sang chế độ giám sát thì tiến trình đang chiếm dụng CPU có thể truy cập tới bất kỳ ô nhớ nào. Nếu ở trong chế độ người dùng, tiến trình chỉ có thể truy xuất tới vùng nhớ người dùng. Như vậy, vùng nhớ cũng có hai không gian là không gian người dùng và không gian bảo vệ.



Hình 3.2. Chế độ người dùng và chế độ giám sát

Nhìn chung, bit chế độ tăng cường khả năng bảo vệ của HĐH. Bit chế độ được thiết lập (đặt giá trị 1) bằng chỉ thị trap ở chế độ người dùng

(trap còn được gọi là chỉ thị yêu cầu chuyển sang chế độ giám sát). Chỉ thị thiết lập giá trị 1 cho bit chế độ và rẽ nhánh tới một vị trí xác định trong vùng nhớ hệ thống. Điều này tương tự như ngắt phần cứng. Những thủ tục của HĐH được tải vào trong vùng nhớ hệ thống và được bảo vệ vì không thể nạp mã của chương trình người dùng vào vùng nhớ đó. Vì mã hệ thống nằm trong vùng nhớ hệ thống, nên chỉ có mã chương trình hệ thống mới được gọi qua trap. Sau khi hoàn thành xong lời gọi ở chế độ giám sát, HĐH thiết lập lại bit chế độ để quay trở về chế độ người dùng.

3.3.2. Nhân hệ điều hành (Kernel)

Có thể coi bộ phận trong phần mềm hệ thống thực thi trong chế độ giám sát gọi là kernel hay nhân của HĐH. Kernel là phần mềm đáng tin cậy, có nghĩa là khi thiết kế và cài đặt, kernel sẽ triển khai cơ chế bảo vệ mà phần mềm bị coi là không đáng tin cậy (thực thi trong vùng nhớ người dùng), không thay đổi được. Tính chính xác của hệ thống không thể dựa trên những phần mở rộng của HĐH thực hiện trong chế độ người dùng. Vì vậy, khi thiết kế chức năng nào đó, vấn đề quan trọng đặt ra là liệu có tích hợp chức năng đó trong kernel hay không. Nếu đặt trong kernel, nó sẽ thực hiện trong vùng nhớ bảo vệ và có thể truy cập tới toàn bộ kernel. Nó cũng được những bộ phận khác trong kernel coi là đáng tin cậy. Nếu thực hiện trong chế độ người dùng, chức năng không truy cập được các cấu trúc dữ liệu của kernel. Chú ý rằng, có thể dễ dàng đặt một thủ tục trong kernel, nhưng chi phí để thực hiện cơ chế trap và kiểm chứng khi gọi thủ tục là cao.

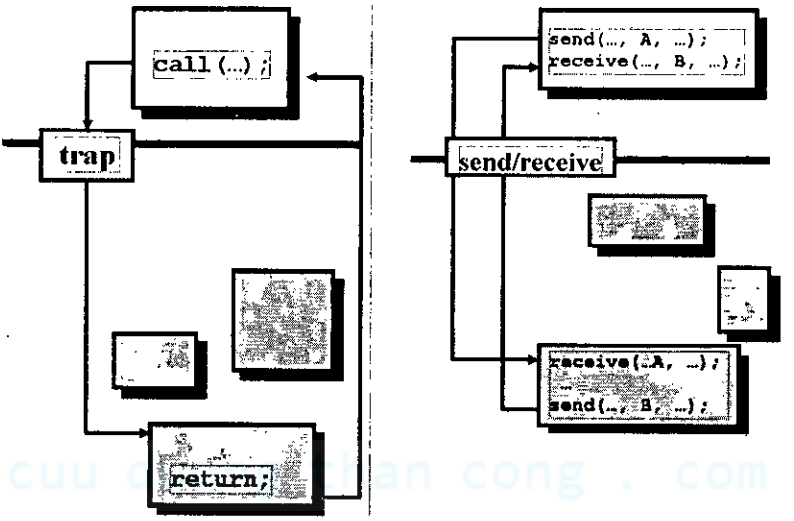
3.3.3. Yêu cầu dịch vụ từ hệ điều hành

Chương trình thực thi trong chế độ người dùng có thể yêu cầu dịch vụ của *kernel* bằng một trong hai kỹ thuật sau:

- Lời gọi hệ thống.
- Chuyển thông điệp.

Hình 3.3 minh họa sự khác biệt giữa hai kỹ thuật này. Đầu tiên, giả sử tiến trình người dùng muốn gọi một hàm hệ thống cụ thể nào đấy (biểu diễn dưới dạng hình chữ nhật tô bóng). Với kỹ thuật lời gọi hệ thống, tiến trình người dùng sử dụng chỉ thị trap (xem mục 3.3.1). Ý tưởng này là lời gọi hệ thống giống lời gọi thủ tục bình thường trong chương trình ứng dụng; HĐH

cung cấp thư viện mà mỗi hàm trong thư viện ứng với một lời gọi hệ thống cụ thể. Khi chương trình ứng dụng gọi, chỉ thị trap được thi hành để chuyển CPU sang chế độ giám sát, sau đó phân nhánh tới hàm chức năng tương ứng. Khi hàm chức năng thực thi xong, hệ thống chuyển CPU sang chế độ người dùng và trả lại quyền điều khiển cho tiến trình người dùng (tương tự việc quay trở lại sau khi kết thúc một thủ tục bình thường).



Hình 3.3. Hai kỹ thuật trong việc phát triển HĐH

Trong giải pháp chuyển thông điệp, tiến trình người dùng tạo ra thông điệp A miêu tả dịch vụ mình cần. Sau đó gửi thông điệp này cho tiến trình HĐH bằng thủ tục send. Giống trap, thủ tục send kiểm tra thông điệp, chuyển CPU sang chế độ giám sát và sau đó gửi thông điệp đến tiến trình cài đặt chức năng được yêu cầu. Trong khi đó, tiến trình người dùng chờ kết quả thực hiện dịch vụ bằng hàm đợi thông điệp receive. Khi hoàn tất dịch vụ, tiến trình HĐH gửi thông điệp trả lời B cho tiến trình người dùng.

Hiệu suất của HĐH sử dụng giao diện lời gọi hệ thống cao hơn HĐH sử dụng trao đổi thông điệp, thậm chí ngay khi lời gọi hệ thống được triển khai qua chỉ thị trap. Kỹ thuật lời gọi hệ thống có một đặc tính rất thú vị là không cần bất kỳ tiến trình HĐH nào. Thay vào đó, tiến trình hoạt động trong chế độ người dùng chuyển sang chế độ giám sát khi thực hiện đoạn mã trong kernel và khi trở về (sau lời gọi HĐH) thì quay trở lại chế độ người dùng. Nếu được thiết kế dưới dạng các tiến trình tách biệt, HĐH dễ dàng chiếm lấy quyền kiểm soát hệ thống trong những trường hợp đặc biệt hơn là trong

thiết kế mà kernel là tập hợp các hàm chức năng được tiến trình người dùng thực hiện trong chế độ giám sát.

3.4. NHẬN XÉT

HĐH tạo ra môi trường cho ứng dụng thực thi. Môi trường này quản lý việc tạo lập và thực thi tiến trình, quản lý việc sử dụng tài nguyên của tiến trình. Bên cạnh những yêu cầu cơ bản, HĐH có thể có một vài yêu cầu đặc thù khác. Mọi tính năng được cài đặt bằng phần mềm đều có một chi phí quản lý làm giảm hiệu suất của các thao tác cơ bản, do đó phải cân nhắc giữa giá trị của chức năng mới và sự suy giảm hiệu suất tổng thể. Thiết kế và cài đặt chức năng phải được tính toán kỹ lưỡng, nhằm đảm bảo hiệu suất hệ thống. HĐH phải tạo ra môi trường chia sẻ an toàn, sao cho các tiến trình không cản trở lẫn nhau. Phần mềm HĐH thường lớn và phức tạp, nhưng phải hoạt động chính xác và có thể bảo trì được. Các yếu tố thương mại, khả năng tương thích, các chuẩn mở cũng tác động mạnh mẽ lên công nghệ HĐH. HĐH hiện đại tích hợp bộ phận quản lý tiến trình và tài nguyên, ngoài ra còn có bộ phận quản lý những tài nguyên khác như bộ nhớ, file và driver. Các bộ phận này có quan hệ chặt chẽ với nhau. Các kỹ thuật triển khai HĐH hiện đại thường dựa trên một vài công nghệ cơ bản. CPU có bit chế độ xác định chế độ hoạt động là chế độ giám sát và chế độ người dùng. Nếu CPU đang ở trong chế độ người dùng và muốn thiết lập bit chế độ để chuyển sang chế độ giám sát, CPU phải thực thi chỉ thị trap để thiết lập bit chế độ và sau đó rẽ nhánh tới mã HĐH. Nếu đang ở trong chế độ giám sát, CPU có thể chuyển sang chế độ người dùng mà không cần hành động đặc biệt nào. Bộ phận HĐH thực thi trong chế độ giám sát được gọi là kernel. Một số HĐH hiện đại sử dụng giao diện lời gọi hệ thống để tiến trình người dùng có thể thực thi đoạn mã HĐH trong chế độ giám sát. Một số HĐH khác thiết kế kernel như những tiến trình hoạt động độc lập tương tác với tiến trình ứng dụng bằng cách trao đổi thông điệp.

CÂU HỎI ÔN TẬP

1. Trình bày các yếu tố tác động lên sự phát triển của HĐH.
2. Trình bày các chức năng chính của một HĐH.
3. Trình bày các phương thức cài đặt HĐH chính.

Chương 4
TIẾN TRÌNH

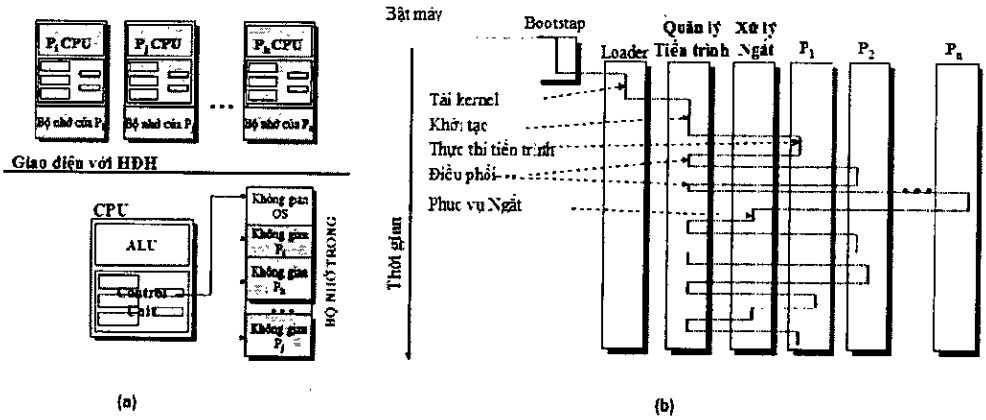
Trong những máy tính thế hệ đầu tiên, tại một thời điểm cụ thể chỉ có duy nhất một chương trình được phép chạy và sử dụng toàn bộ tài nguyên hệ thống. Các hệ thống máy tính hiện đại ngày nay, ngay cả đối với máy tính cá nhân chỉ một người sử dụng, đều cho phép tải nhiều chương trình vào bộ nhớ trong để thực thi đồng thời. Như vậy, cần có cơ chế đan xen hoạt động của các chương trình khác nhau. Nhu cầu này dẫn đến sự xuất hiện khái niệm tiến trình - là chương trình đang trong quá trình thực thi. Tiến trình là đơn vị thực thi cơ sở trong hệ thống chia sẻ thời gian thực. Ngoài ra, HĐH phải có cơ chế cấp phát tài nguyên cho tiến trình theo cơ chế định trước và cơ chế cho phép tiến trình trao đổi thông tin với nhau. Như vậy, dù công việc chính vẫn là thực thi chương trình người dùng, nhưng HĐH phải kiểm soát nhiều công việc khác của hệ thống. Nội dung của chương này trình bày về tiến trình, trạng thái của tiến trình và cách thức HĐH quản lý tiến trình.

4.1. TIẾN TRÌNH VÀ TRẠNG THÁI TIẾN TRÌNH

Thế nào là hoạt động của CPU? Hệ thống phân chia theo lô (batch system) gọi là công việc (job), trong khi hệ thống chia sẻ theo thời gian thực (time-shared system) gọi là chương trình người dùng (user programs) hay tác vụ (task). Ngay trong hệ thống đơn nhiệm như MS-DOS và Macintosh, một người dùng cũng có thể chạy đồng thời vài chương trình. Thậm chí khi ứng dụng nào đó đang chạy, thì HĐH vẫn phải thực thi chương trình bên trong của mình. Ở khía cạnh nào đó, tất cả các hoạt động trên khá giống nhau và chúng ta gọi là tiến trình.

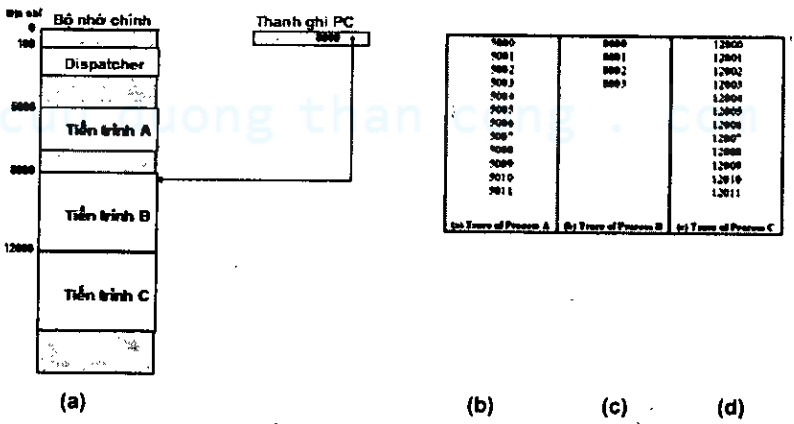
4.1.1. Tiến trình

Thông thường, tiến trình được xem là chương trình đang hoạt động. Quá trình thực thi tiến trình diễn ra tuần tự (tại một thời điểm chỉ có duy nhất một chỉ thị trong tiến trình được thực hiện).



Hình 4.1. Tiến trình

Chương trình khác với tiến trình: Chương trình là thực thể tĩnh (nội dung file exe, com ghi trên ổ đĩa cứng), trong khi tiến trình là thực thể động, với con trỏ chương trình mô tả chỉ thị sẽ được thực hiện kế tiếp và các tài nguyên được cấp phát trong quá trình thực thi. Trong Hình 4.1, chúng ta thấy có ba tiến trình là P_i, P_j và P_k. Mỗi tiến trình nằm ở vị trí khác nhau trong bộ nhớ trong và trạng thái tiến trình biến đổi liên tục theo thời gian. Mặc dù hai tiến trình có thể cùng là của một chương trình (chung khối mã), nhưng chúng được thực thi độc lập với nhau. Bên cạnh đó, tiến trình có thể sinh ra nhiều tiến trình con (mục 4.2).



Hình 4.2. Ba tiến trình A, B và C trong Bộ nhớ

4.1.2. Trạng thái tiến trình

Trong Hình 4.2a, ba tiến trình A, B và C lần lượt nằm ở vùng nhớ có địa chỉ 5000, 8000 và 12000. Như vậy, luồng thực thi của tiến trình A sẽ là 5000, 5001, 5002,... (Hình 4.2b). Tương tự với B và C, nhưng khi xen kẽ hoạt động, các tiến trình A, B, C và tiến trình của HĐH sẽ diễn ra như trong Hình 4.3. Để có thể thiết kế HĐH, cần mô hình hóa hành vi tiến trình.

Thời gian	Chỉ thị	
1	5000	
2	5001	
3	5002	
4	5003	
5	5004	
6	5005	
----------		Hết giờ
7	100	
8	101	
9	102	
10	103	
11	104	
12	105	
13	8000	
14	8001	
15	8002	
16	8003	Yêu cầu vào/ra

17	100	
18	101	
19	102	
20	103	
21	104	
22	105	
23	12000	
24	12001	
25	12002	
26	12003	

Thời gian	Chỉ thị	
27	12004	
28	12005	
----------		Hết giờ
29	100	
30	101	
31	102	
32	103	
33	104	
34	105	
35	5006	
36	5007	
37	5008	
38	5009	
39	5010	
40	5011	
----------		Hết giờ
41	100	
42	101	
43	102	
44	103	
45	104	
46	105	
47	12006	
48	12007	
49	12008	
50	12009	
51	12010	
52	12011	
----------		Hết giờ

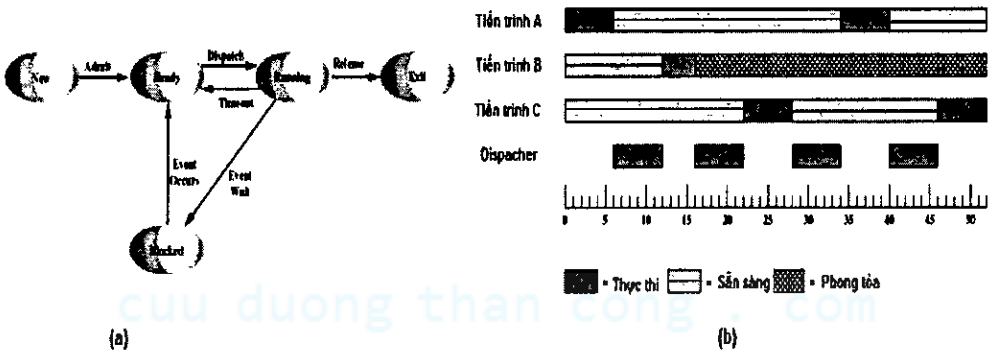
Hình 4.3. Các tiến trình hoạt động xen kẽ

Tiến trình liên tục thay đổi trạng thái trong quá trình thực thi. Tiến trình có thể ở một trong các trạng thái sau:

- **Tạo mới (New):** Tiến trình vừa được tạo ra.
- **Đang thực thi (Running):** Các chỉ thị của tiến trình lần lượt được thực thi.
- **Phong tỏa (Blocked):** Tiến trình chờ đợi sự kiện nào đó để chạy tiếp (ví dụ như hoàn tất thao tác vào/ra hoặc đợi tín hiệu).
- **Sẵn sàng (Ready):** Chờ để được sử dụng CPU. Đã có đủ tất cả các tài nguyên cần thiết.

- **Kết thúc (Exit):** Hoàn tất việc thực thi.

Chú ý, tên trạng thái trên các HĐH có thể khác nhau. Tuy nhiên, 5 trạng thái nêu trên xuất hiện trên mọi hệ thống. Tất nhiên, trên HĐH thực thì số lượng các trạng thái lớn hơn nhiều. Chú ý rằng, trên hệ thống có một CPU thì tại một thời điểm, chỉ có duy nhất một tiến trình sử dụng CPU (đang thực thi), trong khi các tiến trình còn lại ở trạng thái sẵn sàng hay chờ. Lược đồ trạng thái được minh họa trong Hình 4.4. Với các tiến trình minh họa trong Hình 4.3, quá trình biến đổi trạng thái của A, B và C được minh họa trong Hình 4.4b.



Hình 4.4. Mô hình trạng thái của tiến trình

4.2. THAO TÁC TRÊN TIẾN TRÌNH

HĐH cần có cơ chế tạo mới và chấm dứt hoạt động của tiến trình.

4.2.1. Tạo mới tiến trình

Khi tiến trình mới được đưa vào hệ thống, HĐH tạo ra cấu trúc dữ liệu (được trình bày ở phần sau) để quản lý tiến trình. Đồng thời, HĐH cấp phát một loạt tài nguyên cần thiết cho tiến trình hoạt động (chẳng hạn bộ nhớ). Có bốn nguyên nhân tạo mới tiến trình:

- Trong hệ thống lô, khi có công việc được đưa vào, hệ thống tạo một tiến trình để thực hiện công việc.
- Trong hệ thống tương tác, khi người dùng đăng nhập hệ thống.
- HĐH tạo mới tiến trình để thực hiện công việc nào đó. Ví dụ, khi người dùng in văn bản, HĐH tạo tiến trình thực hiện việc in. Điều

này cho phép tiến trình người dùng có thể ngay lập tức tiếp tục thực hiện công việc mà không cần quan tâm khi nào tiến trình in thực hiện xong.

- Tiến trình tạo ra tiến trình con.

Việc tạo mới trong 3 phương pháp đầu "trong suốt" với người dùng. Phương pháp thứ 4 cho phép người dùng tạo ra các tiến trình mới một cách linh hoạt. Ví dụ, tiến trình phục vụ máy in có thể tạo ra các tiến trình con phục vụ mỗi yêu cầu in.

Thông thường, tiến trình cần lượng tài nguyên nhất định (thời gian sử dụng CPU, bộ nhớ, file, các thiết bị vào/ra) để thực hiện công việc. Khi mới được tạo ra, tiến trình con có thể lấy tài nguyên trực tiếp từ HĐH, hoặc chỉ có thể lấy tài nguyên từ tiến trình cha. Tiến trình cha có thể phân chia tài nguyên của mình cho tiến trình con, hoặc chia sẻ các tài nguyên (như file, bộ nhớ) với các tiến trình con. Hạn chế tiến trình con chỉ được sử dụng một phần tài nguyên của tiến trình cha, ngăn cản việc một tiến trình cố tình sinh ra quá nhiều tiến trình con khiến hệ thống quá tải.

Bên cạnh việc cấp phát tài nguyên, tiến trình cha có thể truyền các tham số khởi tạo cho tiến trình con.

Khi tiến trình sinh ra tiến trình con thì:

- Tiến trình cha tiếp tục chạy cùng với tiến trình con.
- Tiến trình cha chờ cho đến khi một vài hoặc tất cả các tiến trình con kết thúc.

Xét về khía cạnh không gian địa chỉ cũng có hai trường hợp sau:

- Tiến trình con là bản sao của tiến trình cha.
- Tiến trình con tải một chương trình khác để thực hiện.

Để mô tả sự khác biệt, xét HĐH UNIX. Trong UNIX, mỗi tiến trình được xác định qua định danh tiến trình. Tiến trình mới được tạo ra qua lời gọi hệ thống **fork()**. Không gian tiến trình con là bản sao của không gian tiến trình cha. Cơ chế này cho phép tiến trình cha có thể trao đổi dễ dàng với tiến trình con. Cả hai tiến trình cha và con tiếp tục thực thi chỉ thị đứng ngay sau **fork()**. Tuy nhiên, với tiến trình con, giá trị trả về của **fork()** là 0, còn với tiến trình cha, giá trị trả về là định danh tiến trình con. Lời gọi hệ thống **execve()**

thường được gọi ngay sau **fork()** để thay thế không gian bộ nhớ bằng một chương trình mới, tức là tải một file nhị phân mới vào bộ nhớ (xóa bỏ nội dung bộ nhớ của chương trình gọi **execve()** và sau đó thực thi chương trình mới được tải). Tiến trình cha có thể tạo nhiều tiến trình con khác, hoặc không làm gì khi các tiến trình con đang thực thi, hoặc có thể sử dụng lời gọi **wait()** để tự loại khỏi hàng đợi sẵn sàng cho đến khi tiến trình con kết thúc.

4.2.2. Kết thúc tiến trình

Có nhiều nguyên nhân kết thúc tiến trình:

1. Tiến trình kết thúc khi thực hiện xong chỉ thị cuối cùng và yêu cầu HĐH loại bỏ qua lời gọi hệ thống **exit()**. Lúc này tiến trình con có thể trả dữ liệu cho tiến trình cha. Tất cả các tài nguyên của tiến trình, bao gồm tài nguyên vật lý và bộ nhớ ảo, các file đang mở, các bộ đệm vào/ra sẽ được HĐH thu hồi.

2. Tiến trình có thể bị một tiến trình khác chấm dứt qua lời gọi hệ thống phù hợp (chẳng hạn **abort**). Thường chỉ tiến trình cha mới được quyền thực hiện lời gọi này để chấm dứt hoạt động của tiến trình con (nếu không thì người dùng có thể kết thúc công việc của người khác). Chú ý, tiến trình cha cần phải biết được định danh của các tiến trình con. Chính vì thế, khi tạo ra một tiến trình mới, hệ thống trả định danh tiến trình con cho tiến trình cha.

Tiến trình cha có thể kết thúc một tiến trình con vì các lý do sau đây:

- Tiến trình con chiếm dụng nhiều tài nguyên, vượt quá số lượng được cấp phát.
- Nhiệm vụ của tiến trình con không còn cần thiết nữa.
- Tiến trình cha kết thúc và HĐH không cho phép tiến trình con của nó chạy tiếp.

Trong trường hợp đầu, tiến trình cha cần có cơ chế xác định trạng thái của tiến trình con. Nhiều hệ thống, ví dụ như HĐH VMS không cho phép tiến trình con tồn tại nếu tiến trình cha đã kết thúc. Trong những hệ thống này, nếu tiến trình kết thúc (một cách bình thường hay bất thường), thì tất cả các tiến trình con phải được kết thúc. Hiện tượng này được gọi là kết thúc dạng tháp và được HĐH tự động thực hiện.

3. Tiến trình đòi hỏi lượng bộ nhớ quá lớn mà hệ thống không đáp ứng được.

4. Tiến trình truy cập tới các vùng bộ nhớ bị cấm.

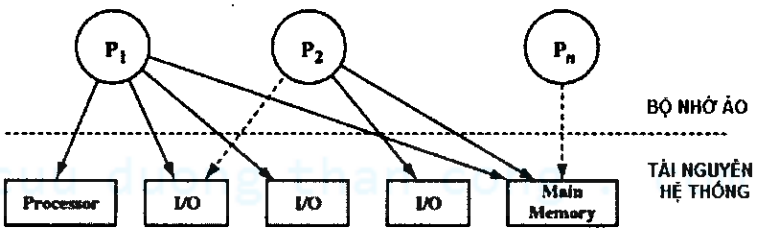
5. Tiến trình sử dụng tài nguyên không hợp lệ.

6. Tiến trình bị lỗi vào/ra, chẳng hạn đọc một đĩa mềm hỏng.

Để hình dung việc bắt đầu và kết thúc của tiến trình, xét HĐH UNIX. Trong UNIX, tiến trình có thể kết thúc bằng cách gọi **exit** và tiến trình cha có thể chờ sự kiện này bằng lời gọi hệ thống **wait**. **wait()** trả về cho tiến trình cha định danh của tiến trình con đã kết thúc. Nếu tiến trình cha kết thúc thì tất cả tiến trình con cũng bị HĐH chấm dứt. Nếu không có tiến trình cha, UNIX không biết phải báo trạng thái hoạt động của tiến trình con cho tiến trình nào.

4.3. MÔ TẢ TIẾN TRÌNH

HĐH quản lý tài nguyên hệ thống (bộ nhớ, CPU, thiết bị vào/ra,...) và cấp phát khi tiến trình yêu cầu. Ví dụ, hệ thống đa chương trình trên Hình 4.5, P_1 đang thực thi và nắm quyền kiểm soát hai thiết bị vào/ra; P_2 nằm trong bộ nhớ nhưng ở chế độ phong tỏa vì đợi một thiết bị vào/ra đã được cấp phát cho P_1; P_n đang ở trạng thái treo vì bị chuyển ra ổ đĩa cứng. Vấn đề đặt ra ở đây là làm thế nào để HĐH quản lý được các tiến trình cũng như tài nguyên đã được cấp phát?

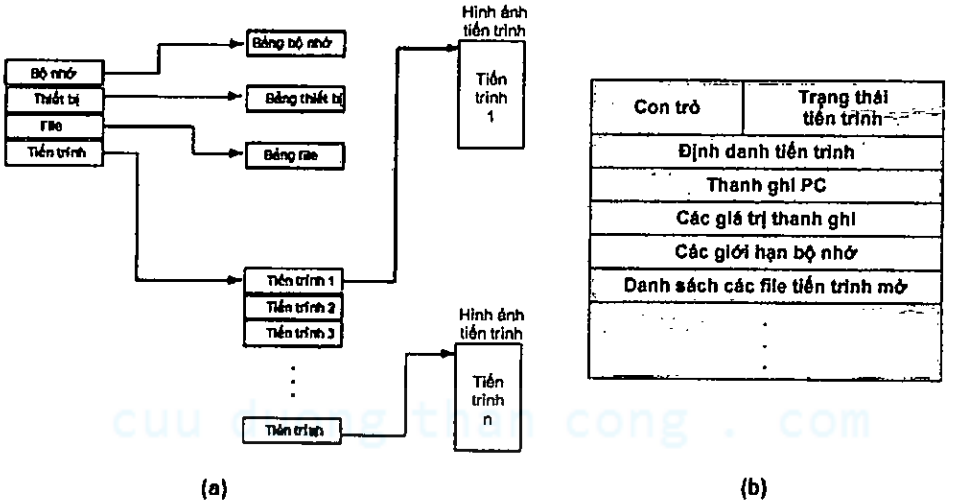


Hình 4.5. Tiến trình và tài nguyên

4.3.1. Khối điều khiển tiến trình (PCB)

Để nắm được thông tin về đối tượng mình quản lý, HĐH thường tạo ra các bảng ghi trạng thái hiện thời về từng đối tượng. Ý tưởng này được minh họa trong Hình 4.6. HĐH thường có 4 bảng để quản lý bộ nhớ, thiết bị, file

và tiến trình. Bảng quản lý bộ nhớ ghi thông tin về bộ nhớ cấp phát cho tiến trình, các thuộc tính bảo vệ của vùng bộ nhớ (trình bày trong Chương 8 và 9). Bảng vào/ra chứa thông tin về thiết bị. Chẳng hạn, thiết bị còn rỗi hay đã cấp phát cho tiến trình nào. Nếu đang thực hiện một thao tác vào/ra thì HĐH cần xác định trạng thái của thao tác. Bảng file cung cấp các thông tin về vị trí của file trên ổ đĩa cứng. Cuối cùng là bảng theo dõi tất cả các tiến trình. Bốn bảng này có quan hệ chặt chẽ với nhau.



Hình 4.6. Các bảng quản lý trong HĐH

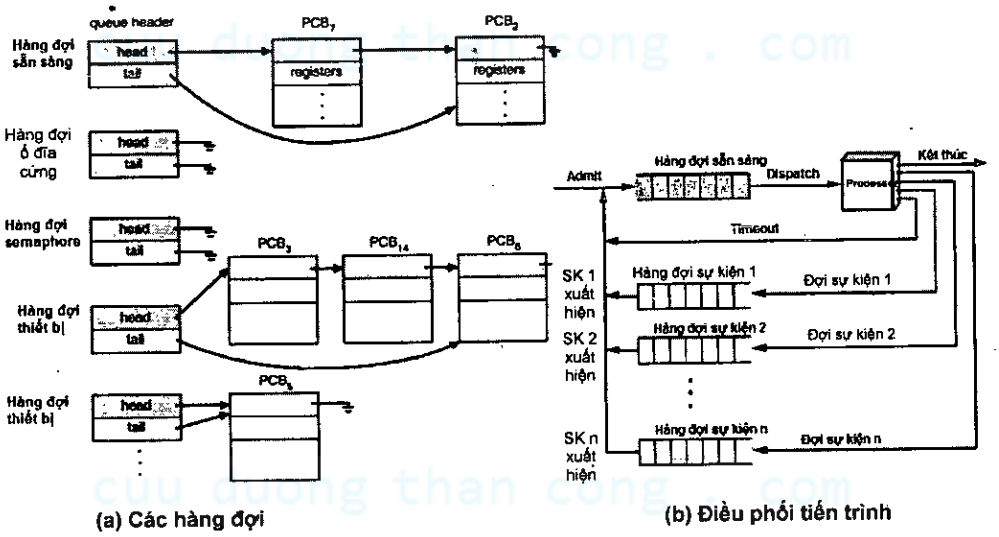
Tiến trình được HĐH kiểm soát thông qua khối điều khiển tiến trình (Process Control Block – PCB). Khối này chứa các thông tin liên quan đến tiến trình sau đây:

- **Trạng thái tiến trình (Process state):** Có thể là tạo mới, sẵn sàng, đang chạy, phong tỏa hay kết thúc.
- **Con trỏ chương trình (program couter):** Chứa địa chỉ của chỉ thị sẽ được thực thi kế tiếp.
- **Các thanh ghi CPU (CPU register):** Số lượng các thanh ghi phụ thuộc vào kiến trúc máy tính. Các thanh ghi này có thể cho phép người dùng truy cập (thanh ghi tích lũy, thanh ghi chỉ mục, thanh ghi ngăn xếp, các thanh ghi đa chức năng, thanh ghi trạng thái hay còn gọi là thanh ghi cờ) cùng các thanh ghi đặc biệt dùng cho quá trình kiểm soát của HĐH.

- **Thông tin về điều phối CPU:** Quyền ưu tiên của tiến trình, con trỏ đến hàng đợi điều phối và các tham số điều phối khác.
- **Thông tin quản lý bộ nhớ:** Giá trị thanh ghi cơ sở và thanh ghi giới hạn; bảng trang, bảng đoạn. Thông tin này phụ thuộc vào hệ thống quản lý bộ nhớ (xem Chương 9).
- **Trạng thái vào/ra:** Danh sách các thiết bị vào/ra tiến trình đang sử dụng, danh sách các file đang mở.

4.3.2. Điều phối tiến trình

Mục tiêu đa nhiệm là tận dụng tối đa CPU bằng cách cho phép nhiều tiến trình chạy đồng thời. Trên hệ thống có một CPU, hai tiến trình không thể thực thi cùng lúc. Nếu có nhiều tiến trình thì các tiến trình phải đợi đến khi CPU rỗi. Như vậy, có hai vấn đề được đặt ra: Khi nào chuyển quyền sử dụng CPU của tiến trình (gọi tắt là chuyển quyền) và HĐH cần thực hiện công việc gì?



Hình 4.7. Các hàng đợi và việc điều phối

Ngay khi tạo ra, tiến trình được đặt vào hàng đợi công việc (job queue). Hàng đợi này chứa tất cả các tiến trình trong hệ thống. Những tiến trình đã nằm trong bộ nhớ và sẵn sàng sử dụng CPU được đặt trong hàng đợi sẵn sàng. Hàng đợi này thường được cài đặt dưới dạng danh sách liên kết. Header của hàng đợi chứa hai con trỏ, trỏ đến phần tử PCB đầu và PCB cuối

danh sách. PCB có trường con trỏ, trỏ đến tiến trình kế tiếp trong hàng đợi. Trong hệ thống có thể có nhiều hàng đợi. Khi chiếm dụng CPU, tiến trình thực thi trong một khoảng thời gian nào đó trước khi kết thúc hoặc tạm dừng (do rất nhiều nguyên nhân như đợi một sự kiện cụ thể, đợi yêu cầu vào/ra, cố tình truy cập vào vùng nhớ không được phép,...). Trường hợp yêu cầu vào/ra, tiến trình có thể phải đợi thao tác trên thiết bị hoàn thành. Khi hệ thống có nhiều tiến trình, thiết bị có thể trong trạng thái bận do phải phục vụ nhiều yêu cầu. Do đó, tiến trình buộc phải chờ cho đến khi thiết bị rỗi. Danh sách các tiến trình chờ đợi thiết bị vào/ra gọi là hàng đợi thiết bị. Mỗi thiết bị có hàng đợi riêng (Hình 4.7a).

Trong điều phối tiến trình, sơ đồ hàng đợi là kỹ thuật biểu diễn thông dụng (Hình 4.7b). Ở đây ta thấy có hai loại hàng đợi là hàng đợi sẵn sàng và các hàng đợi thiết bị.

Tiến trình mới sẽ được đặt vào hàng đợi sẵn sàng. Tiến trình sẽ chờ đợi cho đến khi được quyền sử dụng CPU. Khi đang thực thi, nếu phải đợi một sự kiện nào đó, tiến trình chuyển sang hàng đợi ứng với sự kiện tương ứng. Các sự kiện có thể xảy ra là:

- Tiến trình yêu cầu thực hiện vào/ra.
- Tiến trình có thể tạo ra các tiến trình con và chờ cho đến khi tiến trình con kết thúc.
- Tiến trình có thể bị chiếm đoạt quyền sử dụng CPU (chẳng hạn do ngắt). Sau đó tiến trình được đưa trở lại hàng đợi sẵn sàng.

Trong hai trường hợp đầu, tiến trình chuyển từ trạng thái đang thực thi sang trạng thái phong tỏa, sau đó lại quay trở lại trạng thái sẵn sàng. Tiến trình tiếp tục chuyển trạng thái một cách tuần hoàn như vậy cho đến khi kết thúc. Lúc này tiến trình sẽ bị xóa khỏi tất cả các hàng đợi và PCB của tiến trình cũng như mọi tài nguyên được cấp phát bị HĐH thu hồi.

4.3.3. Bộ điều phối

Tiến trình dịch chuyển giữa các hàng đợi trong suốt thời gian sử dụng của mình. HĐH có nhiệm vụ di chuyển tiến trình giữa các hàng đợi. Công việc này được thực hiện bởi bộ điều phối. Trong hệ thống lô, số lượng các tiến trình từ bên ngoài đưa vào hệ thống thường nhiều hơn số lượng các tiến trình có thể thực thi ngay lập tức. Các tiến trình này được lưu ở đâu đó

(thường là ổ đĩa) để thực thi sau. Bộ điều phối dài hạn lựa chọn các tiến trình từ nhóm này để tải vào bộ nhớ. Bộ điều phối ngắn hạn cấp phát CPU cho một tiến trình nào đó sẵn sàng thực thi đã nằm trong bộ nhớ.

Điểm khác biệt cơ bản giữa hai bộ điều phối trên là tần suất hoạt động. Bộ điều phối ngắn hạn chọn một tiến trình nắm quyền điều khiển CPU. Một tiến trình có thể chạy trong vài mili giây (ms) trước khi tạm dừng đợi yêu cầu vào/ra hoàn thành. Thường thì trong 100ms, bộ điều phối ngắn hạn chạy một lần. Vì chạy giữa hai lần thực thi, nên bộ điều phối ngắn hạn cần phải chạy cực nhanh. Nếu mỗi lần mất 10ms để quyết định xem tiến trình nào được chạy tiếp trong khoảng 100ms tiếp theo thì khi đó $10/(100 + 10) \approx 9\%$ CPU bị lãng phí cho công việc điều phối tiến trình. Tần suất chạy của bộ điều phối dài hạn thấp hơn nhiều, có thể khoảng thời gian giữa hai lần tạo mới hai tiến trình kế tiếp là một vài phút. Bộ điều phối dài hạn phải kiểm soát số lượng các tiến trình nằm trong bộ nhớ. Nếu số lượng này ổn định thì trung bình số lượng tiến trình được tạo mới xấp xỉ số lượng tiến trình rời khỏi hệ thống. Như vậy, bộ điều phối dài hạn được gọi chỉ khi có tiến trình kết thúc. Vì khoảng thời gian giữa hai lần chạy là khá dài nên bộ điều phối dài hạn có nhiều thời gian lựa chọn tiến trình nào được tải vào bộ nhớ.

Bộ điều phối dài hạn có vai trò quan trọng. Thường tiến trình có đặc điểm hướng vào/ra hoặc hướng tính toán. Tiến trình hướng vào/ra thường tốn thời gian thực hiện vào/ra hơn thời gian tính toán trên CPU. Ngược lại, tiến trình hướng tính toán có ít yêu cầu vào/ra mà phần lớn thời gian sử dụng CPU. Một điều quan trọng là bộ điều phối dài hạn phải chọn một nhóm hỗn hợp các tiến trình hướng vào/ra và hướng tính toán. Hệ thống chỉ đạt được hiệu suất cao nhất khi có sự kết hợp giữa các tiến trình hướng vào/ra và các tiến trình hướng tính toán.

Một vài hệ thống có thể không có bộ điều phối dài hạn, ví dụ, hệ thống chia sẻ thời gian thực. Sự ổn định của những hệ thống này phụ thuộc vào khả năng của phần cứng hoặc sự tự điều chỉnh của người dùng. Nếu hiệu suất hệ thống giảm đến mức quá thấp, cách giải quyết đơn giản là một vài người dùng thoát khỏi hệ thống.

Một số HĐH chia sẻ thời gian thực có thể có thêm một bộ điều phối trung gian, gọi là bộ điều phối trung hạn, có nhiệm vụ chuyển bớt các tiến trình trong bộ nhớ ra ổ đĩa cứng để giảm mức đa chương trình. Sau đó, tiến

trình có thể được đưa trở lại bộ nhớ để khôi phục việc thực thi. Phương pháp này gọi là sự hoán chuyển và sẽ được trình bày trong Chương 11.

4.3.4. Chuyển ngữ cảnh

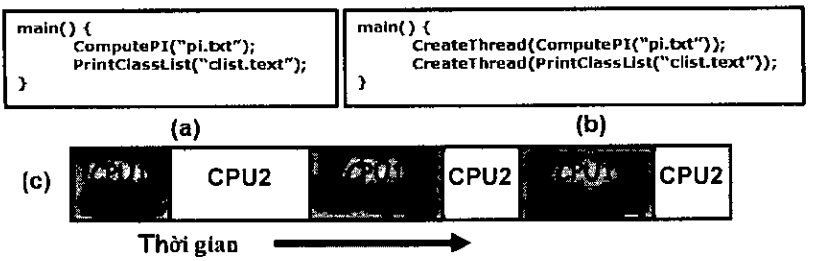
Muốn chuyển đổi quyền sử dụng CPU giữa các tiến trình thì phải lưu lại trạng thái tiến trình cũ và nạp lại trạng thái tiến trình mới. Quá trình này được gọi là chuyển ngữ cảnh (context switch). Thời gian thực hiện chuyển ngữ cảnh hoàn toàn lãng phí, vì hệ thống không làm công việc hữu ích. Tốc độ chuyển ngữ cảnh phụ thuộc vào thông số của từng máy tính như: tốc độ bộ nhớ, số lượng các thanh ghi phải lưu giữ và liệu máy tính có các chỉ thị đặc biệt hỗ trợ chuyển ngữ cảnh (chẳng hạn, có chỉ thị để nạp và lưu tất cả các thanh ghi cùng một lúc) hay không. Thường tốc độ này nằm trong khoảng từ 1 đến 1000ms.

Thời gian chuyển ngữ cảnh phụ thuộc vào sự hỗ trợ từ phần cứng. Ví dụ, một số bộ vi xử lý như DECSYSTEM-20 có nhiều tập hợp các thanh ghi. Chuyển ngữ cảnh đơn giản chỉ là chuyển con trỏ đến tập hợp các thanh ghi hiện thời. Tất nhiên, nếu số lượng các tiến trình lớn hơn số tập các thanh ghi thì hệ thống cũng phải lưu trữ giá trị của thanh ghi lên bộ nhớ giống như trên. HĐH càng phức tạp thì công việc chuyển ngữ cảnh càng nhiều. Trong Chương 9 (Quản lý bộ nhớ) ta thấy có thể phải di chuyển khá nhiều dữ liệu trong mỗi lần chuyển ngữ cảnh. Ví dụ, không gian bộ nhớ của tiến trình hiện tại cần phải được lưu giữ cũng như phải chuẩn bị không gian bộ nhớ cho tiến trình tiếp theo. Cách thức lưu giữ không gian này phụ thuộc vào phương thức quản lý bộ nhớ của HĐH. Nói chung, chuyển ngữ cảnh là "nút cổ chai", làm giảm đáng kể hiệu suất hệ thống, nên người ta hay sử dụng các luồng.

4.4. LUỒNG

cuu duong than cong . com

Trong các HĐH trước đây, tiến trình được cấp phát lượng tài nguyên và bộ nhớ riêng. Tuy nhiên, trong nhiều trường hợp, việc sử dụng chung tài nguyên giữa các tiến trình hoạt động đồng thời lại có nhiều ưu điểm hơn, ví dụ, chương trình Web server có thể phục vụ đồng thời nhiều yêu cầu khác nhau. Điều này cũng tương tự lời gọi hệ thống **fork()** tạo ra một luồng chương trình mới, tức là có con trỏ chương trình mới, nhưng thực thi công việc trên cùng một không gian địa chỉ.



Hình 4.8. Ưu điểm của luồng

Xét chương trình được viết bằng ngôn ngữ C trong Hình 4.8a, chương trình có hai hàm, ComputePI sẽ tính giá trị số π và viết vào file pi.txt, PrintClassList in danh sách lớp từ file clist.txt. Vì π là giá trị vô hạn, nên ComputePI chạy vô tận, và như thế PrintClassList sẽ không bao giờ được dùng. Tuy nhiên, nếu cải tiến chương trình như trong Hình 4.8b, chương trình sẽ có hai thread chạy đồng thời, thread thứ nhất chạy ComputePI và thread thứ hai chạy PrintClassList. Hai hàm này luân phiên nhau sử dụng CPU như trong Hình 4.8c, do đó hàm thứ 2 không bị trì hoãn vĩnh viễn. Qua ví dụ trên, chúng ta thấy khái niệm luồng có rất nhiều ưu điểm, nên phần lớn các HĐH hiện đại đều cài đặt công nghệ luồng.

Cấu trúc luồng:

Luồng đôi khi được gọi là tiểu trình (Light Weight Process), là đơn vị cơ sở trong việc sử dụng CPU, bao gồm con trỏ chương trình, tập các thanh ghi và ngăn xếp. Luồng cùng chia sẻ đoạn mã chương trình, dữ liệu và một số tài nguyên. Các luồng hợp tác với nhau để thực hiện một tác vụ. Trong mô hình này, tiến trình theo định nghĩa trước tương đương với tác vụ được thực thi bởi một luồng duy nhất. Tác vụ không có luồng sẽ không thể hoạt động và luồng phải nằm trong duy nhất một tác vụ nào đó. Các luồng luân phiên nhau sử dụng CPU và chi phí việc tạo luồng mới thấp hơn rất nhiều so với việc tạo mới tiến trình. Mặc dù chuyển ngữ cảnh giữa các luồng vẫn yêu cầu thay đổi tập các thanh ghi, nhưng không cần gọi chức năng quản lý bộ nhớ. Một số hệ thống triển khai công nghệ luồng thông qua thư viện mức người dùng thay vì sử dụng lời gọi hệ thống, bởi vậy việc chuyển ngữ cảnh luồng không cần gọi đến HĐH hay làm gián đoạn kernel. Chuyển ngữ cảnh giữa luồng ở mức người dùng diễn ra rất nhanh và không cần sự can thiệp của HĐH, điều này giúp rút ngắn tốc độ thực hiện công việc. Do vậy, đây là giải pháp hợp lý khi phải xây dựng chương trình cần phục vụ nhiều yêu cầu

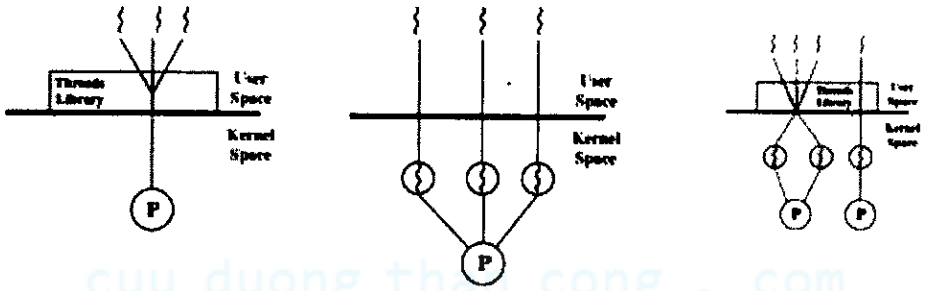
cùng lúc. Tuy nhiên, luồng ở mức người dùng có một số mặt hạn chế. Trong trường hợp phần nhân (kernel) là một luồng duy nhất, thì khi bất kỳ luồng ở mức người dùng thực hiện lời gọi hệ thống sẽ khiến toàn bộ tác vụ phải đợi đến khi lời gọi hệ thống trả lại kết quả.

Chúng ta so sánh mô hình luồng với mô hình tiến trình. Đối với đa tiến trình thì các tiến trình hoạt động độc lập với nhau, mỗi tiến trình có con trỏ chương trình, hệ thống ngăn xếp và không gian địa chỉ riêng. Kiểu tổ chức này có lợi khi công việc tiến trình thực hiện độc lập với nhau. Ví dụ, đối với hệ thống có một CPU, tiến trình file server sẽ bị dừng hoạt động (trạng thái phong tỏa) khi đang đợi đọc ổ đĩa. Hiệu suất hệ thống sẽ được cải thiện nếu các tiến trình file server khác vẫn có thể làm việc trong khi tiến trình kia bị phong tỏa vì các tiến trình này có cùng chương trình.

Xét trên nhiều mặt, hoạt động của luồng giống tiến trình. Luồng có thể ở một trong các trạng thái như sẵn sàng, phong tỏa, thực thi hoặc kết thúc. Giống tiến trình, các luồng cùng nhau chia sẻ CPU và tại một thời điểm chỉ một luồng duy nhất được quyền thực thi. Luồng trong một tiến trình được thực thi tuần tự và mỗi luồng có ngăn xếp, con trỏ đếm chương trình riêng. Luồng cũng có thể tạo ra luồng con, có thể bị phong tỏa và tạm dừng do đợi thực hiện thao tác vào/ra; trong khi một luồng đang ở trạng thái phong tỏa thì các luồng khác được phép thực thi. Tuy nhiên, điểm khác biệt ở đây là các luồng có quan hệ với nhau. Vì luồng có thể truy cập đến bất kỳ địa chỉ nào trong không gian địa chỉ tiến trình, nên có thể đọc hay viết vào ngăn xếp của luồng khác. Do đó, không có cơ chế bảo vệ ngăn cách giữa các luồng và thực sự cơ chế này cũng không cần thiết.

Bây giờ quay lại ví dụ tiến trình file server bị phong tỏa trong mô hình tiến trình. Trong trường hợp này, không tiến trình phục vụ nào khác được thực thi cho đến khi tiến trình trên được giải tỏa. Ngược lại, nếu tác vụ có nhiều luồng khác nhau, thì khi một luồng phục vụ bị phong tỏa thì ngay lập tức luồng thứ hai thực hiện cùng một nhiệm vụ có thể được khởi động để thực thi. Trong ứng dụng này, việc kết hợp nhiều luồng thực hiện cùng một công việc làm tăng thông lượng cũng như hiệu suất hệ thống. Các ứng dụng khác, chẳng hạn bài toán sản xuất – tiêu thụ cài đặt bằng cách chia sẻ bộ đệm cũng có thể tận dụng ưu điểm luồng. Thành phần sản xuất hay tiêu thụ có thể là các luồng của cùng một tiến trình. Khi đó chi phí phụ trội cho việc chuyển ngữ cảnh được giảm thiểu và hiệu suất hệ thống được nâng cao.

Để minh họa ưu điểm sự thực thi đồng thời trong mô hình luồng, giả sử cần phải viết file server trong hệ thống không hỗ trợ luồng, khi đó server phải hoàn thành xong một yêu cầu nào đó rồi mới có thể phục vụ yêu cầu kế tiếp. Nếu như yêu cầu thứ nhất phải đợi để đọc ổ đĩa cứng thì CPU hoàn toàn lãng phí trong khoảng thời gian này. Vì vậy, số yêu cầu thực hiện trung bình tính trong 1 giây sẽ ít hơn rất nhiều so với trường hợp đa luồng xử lý song song. Nếu không có công nghệ luồng, người thiết kế hệ thống phải tìm cách giảm thiểu việc suy giảm hiệu suất bằng cách bắt chước cấu trúc hoạt động song song của hệ thống luồng thông qua việc sử dụng các tiến trình truyền thống. Khi đó cấu trúc của chương trình tuần tự trở nên phức tạp.



(a) Luồng ở mức người dùng (b) Luồng ở mức nhân (c) Giải pháp kết hợp

Hình 4.9. Luồng ở các mức khác nhau

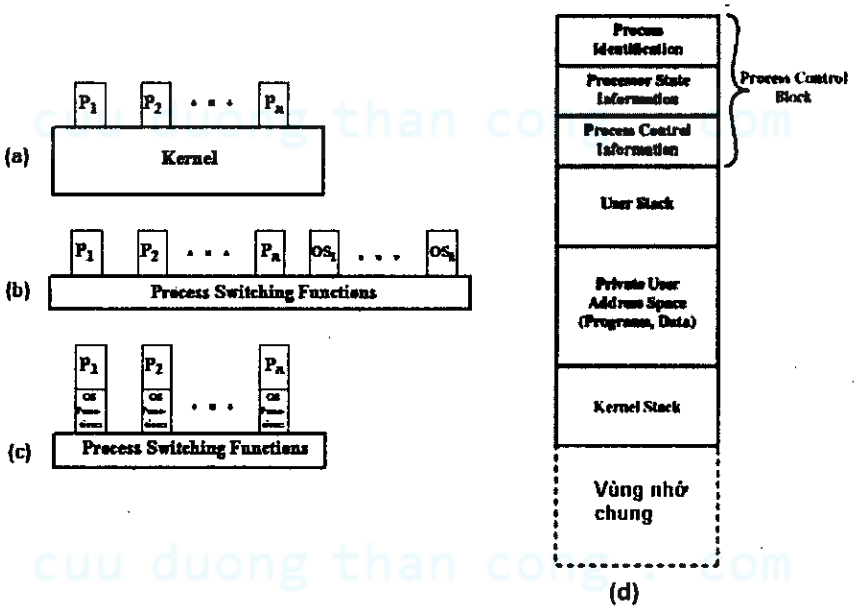
Có nhiều kiểu luồng khác nhau như luồng có thể được kernel hỗ trợ (HĐH Mach và OS/2). HĐH cung cấp các lời gọi hệ thống cho luồng tương tự lời gọi hệ thống thao tác trên tiến trình (Hình 4.9b). Một phương pháp khác là luồng hỗ trợ ở mức người dùng, thông qua lời gọi các hàm thư viện (Project Andrew của CMU) (Hình 4.9a). Luồng ở mức người dùng không liên quan đến kernel, do đó chuyển ngữ cảnh sẽ nhanh hơn nhiều so với hệ thống luồng do kernel hỗ trợ. Tuy nhiên, bất kỳ lời gọi nào tới HĐH sẽ khiến toàn bộ các luồng của cùng tiến trình phải đợi vì kernel chỉ điều phối tiến trình (kernel không biết khái niệm luồng) và tiến trình trong trạng thái đợi không được cấp phát CPU. Điều phối có thể không công bằng. Xét hai tiến trình a và b: a có 1 và b có 100 luồng. Vì mỗi tiến trình cùng nhận được một lượng tử thời gian như nhau, nên luồng của tiến trình a sẽ chạy nhanh hơn luồng của tiến trình b 100 lần. Đối với hệ thống mà kernel hỗ trợ luồng, việc chuyển ngữ cảnh giữa các luồng tốn thời gian hơn vì chính kernel phải thực hiện việc chuyển ngữ cảnh (thông qua ngắt). Tuy nhiên, các luồng

được thực hiện một cách độc lập, nên thời gian tiến trình b nhận được bằng 100 lần thời gian mà tiến trình a nhận được. Không những thế, tiến trình b có thể đồng thời gọi 100 lời gọi hệ thống, nhiều hơn rất nhiều so với trường hợp chính tiến trình này chạy trên hệ thống hỗ trợ luồng ở mức người dùng.

Trong một số hệ thống (HĐH Solaris 2), người ta cài đặt cả hai cơ chế hỗ trợ luồng để nâng cao hiệu suất (Hình 4.9c).

4.5. CÀI ĐẶT HỆ ĐIỀU HÀNH

Hoạt động của HĐH giống hoạt động của bất kỳ phần mềm nào khác, tức là một chương trình nào đó chạy trên CPU, vậy HĐH cũng phải chia sẻ quyền sử dụng CPU với các tiến trình khác. Tuy nhiên, HĐH là một tập hợp các tiến trình hay là một tiến trình? Hình 4.10 minh họa các phương pháp cài đặt HĐH.



Hình 4.10. Các phương pháp cài đặt HĐH

☞ ***Kernel không là tiến trình nào cả***

Kernel HĐH nằm ngoài tất cả tiến trình, có vùng nhớ và ngăn xếp riêng (Hình 4.10a). Khi tiến trình đang hoạt động bị ngắt hoặc gọi lời gọi hệ thống, kernel lưu lại trạng thái tiến trình và nắm lại quyền sử dụng CPU.

HĐH sẽ thực hiện các chức năng cần thiết, rồi sau đó khôi phục hoạt động của tiến trình bị ngắt. Khái niệm tiến trình chỉ áp dụng với chương trình người dùng. Mã HĐH được thực thi riêng biệt khi CPU trong chế độ giám sát.

☞ ***HĐH thực thi trong tiến trình người dùng***

Một phương pháp được áp dụng phổ biến trong các HĐH trên dòng máy tính nhỏ hoặc siêu nhỏ là thực thi tất cả phần mềm của HĐH trong ngữ cảnh tiến trình người dùng. HĐH bao gồm các thường trình, được người sử dụng gọi nhằm thực hiện công việc nào đó. Thường trình này vẫn chạy trong không gian bộ nhớ của tiến trình người dùng như minh họa trong Hình 4.10b.

Hình 4.10d minh họa cấu trúc của tiến trình trong phương pháp này. Bên cạnh vùng mã và dữ liệu thông thường, tiến trình có vùng nhớ chung là nơi đặt đoạn mã thuộc HĐH. Các đoạn mã này được sử dụng chung giữa nhiều tiến trình trong hệ thống.

Khi xuất hiện ngắt hay tiến trình gọi lời gọi hệ thống, CPU chuyển sang chế độ giám sát và quyền điều khiển được chuyển cho đoạn mã của HĐH. Khi đó, ngữ cảnh của tiến trình được lưu lại và chuyển sang ngữ cảnh của chương trình HĐH. Tuy nhiên, việc thực thi vẫn xảy ra trong tiến trình người sử dụng. Tức là có chuyển ngữ cảnh, nhưng không chuyển chương trình. Nếu sau khi hoàn thành công việc, HĐH thấy cần khôi phục tiến trình vừa bị phong tỏa thì HĐH chỉ cần chuyển ngữ cảnh sang tiến trình đó. Sau khi tiến trình bị tạm thời phong tỏa để đợi một dịch vụ của HĐH, tiến trình có thể nhanh chóng được khôi phục mà không cần 2 lần chuyển ngữ cảnh. Nếu HĐH muốn cho tiến trình khác chạy, thì HĐH chuyển quyền điều khiển cho bộ phận điều phối để lựa chọn cấp phát CPU cho một tiến trình khác. Hệ thống hoạt động ổn định và chính xác vì đoạn mã thực hiện những việc trên không phải đoạn mã người dùng mà là đoạn mã HĐH. Vì có bit chế độ, nên chương trình người dùng không thể can thiệp vào chương trình của HĐH ngay cả khi chương trình HĐH thực thi trong không gian bộ nhớ người sử dụng. Điểm chú ý ở đây là phân biệt tiến trình và chương trình. Tiến trình sẽ chạy hai loại chương trình, chương trình người dùng và

chương trình HĐH. Chương trình HĐH thực thi trong tất cả các tiến trình đều giống nhau.

☞ ***HĐH dựa trên tiến trình***

Phương pháp thứ 3 được minh họa trên Hình 4.10c. HĐH được triển khai thông qua một nhóm các tiến trình hệ thống, các tiến trình hệ thống này cài đặt các chức năng HĐH cơ sở và thực thi trong chế độ giám sát. Ưu điểm của phương pháp này chính là tính module, khiến các thành phần của HĐH có giao diện tường minh với nhau.

☞ ***Vi nhân (Microkernel)***

Phương pháp này được áp dụng trong Windows NT. Microkernel là một hạt nhân (rất nhỏ) của HĐH, chỉ cung cấp các chức năng cơ sở nhất (lời gọi hệ thống). Các dịch vụ khác (do kernel cung cấp) được những chương trình (gọi là server) thực thi trong không gian người sử dụng. Ví dụ, Microkernel cung cấp dịch vụ quản lý không gian địa chỉ, quản lý luồng, truyền thông liên tiến trình, nhưng không cung cấp các dịch vụ như kết nối mạng hay hiển thị thông tin trên màn hình. Ưu điểm của giải pháp này là khi thêm một dịch vụ mới không cần phải thay đổi kernel. Tính an ninh của hệ thống được tăng cường vì phần lớn các thao tác thực thi ở chế độ người dùng chứ không phải trong chế độ giám sát.

4.6. NHẬN XÉT

Tiến trình là chương trình trong giai đoạn thực thi. Trong quá trình thực thi, tiến trình trải qua nhiều trạng thái. Tại mỗi thời điểm, trạng thái là hoạt động cụ thể của tiến trình như tạo mới, sẵn sàng, thực thi, phong tỏa, kết thúc. HĐH quản lý tiến trình qua khối điều khiển tiến trình (PCB). Tiến trình khi không thực thi được đặt trong hàng đợi nào đấy. Có hai loại hàng đợi chính trong HĐH là hàng đợi vào/ra và hàng đợi sẵn sàng. Hàng đợi sẵn sàng chứa những tiến trình sẵn sàng thực thi và đang chờ đến lượt sử dụng CPU. Bộ điều phối dài hạn lựa chọn nhóm tiến trình được quyền thực thi. Bộ điều phối ngắn hạn chọn một tiến trình nằm trong hàng đợi sẵn sàng để thực thi. Các tiến trình trong hệ thống có thể được thực hiện một cách đồng thời và do đó có thể chia sẻ thông tin, tăng tốc độ tính toán, cô lập các thành phần. Để thực hiện được đồng thời, hệ thống cần có cơ chế tạo mới và xóa

bỏ tiến trình. Luồng – đơn vị cơ bản được quyền sử dụng CPU dùng chung phần mã, phần dữ liệu và một số tài nguyên hệ thống khác với những luồng ngang hàng, trong cùng một tiến trình (hay tác vụ).

CÂU HỎI ÔN TẬP

1. Định nghĩa tiến trình và trạng thái tiến trình.
2. Xác định các thành phần chính của khối điều khiển tiến trình.
3. Trình bày nhiệm vụ của bộ điều phối tiến trình.

cuu duong than cong . com

cuu duong than cong . com

Chương 5
ĐIỀU PHỐI TIẾN TRÌNH

*Điều phối tiến trình là lập kế hoạch cấp phát quyền sử dụng CPU. Mục đích sử dụng máy tính quyết định chính sách điều phối. Chú ý phân biệt **cơ chế** quyết định cách thức cấp phát CPU với **chính sách** xác định thứ tự "được sử dụng CPU" của tiến trình. Chương này trình bày cơ chế điều phối và hai lớp chính sách: độc quyền sử dụng (nonpreemptive) và không độc quyền sử dụng (preemptive). Thuật toán độc quyền cho phép tiến trình chiếm dụng CPU cho tới khi hoàn thành công việc. Trong thuật toán không độc quyền, tiến trình chỉ được sử dụng CPU trong một khoảng thời gian và khi hết thời gian, bộ điều phối chuyển quyền sử dụng CPU sang cho tiến trình khác cho dù tiến trình đang chạy chưa hoàn thành công việc. Chúng ta sẽ trình bày một số cơ chế được sử dụng trong các HĐH hiện đại cũng như phương thức cài đặt chúng.*

5.1. CƠ CHẾ ĐIỀU PHỐI

Trong HĐH đa chương trình, nhiều tiến trình được tải vào bộ nhớ trong và cùng nhau chia sẻ CPU. Do vậy, cần phương thức để HĐH cùng các tiến trình ứng dụng chia sẻ CPU. Ngoài ra, thời gian thực hiện vào/ra lớn hơn nhiều thời gian tiến trình sử dụng CPU, do đó trong hệ thống đa chương trình, tiến trình thực hiện vào/ra nên nhường quyền sử dụng CPU cho tiến trình khác. Cơ chế điều phối là bộ phận của bộ quản lý tiến trình, chịu trách nhiệm loại bỏ tiến trình đang chiếm giữ CPU và sau đó lựa chọn theo chính sách định trước một tiến trình nào đó trong trạng thái sẵn sàng để thực thi.

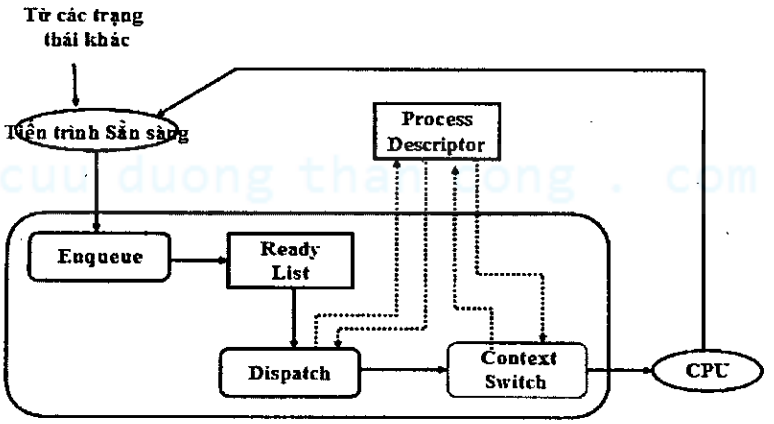
5.1.1. Bộ điều phối tiến trình

Trách nhiệm chính của bộ điều phối là luân phiên cho phép các tiến trình thực thi (tức là sử dụng CPU). Khi tiến trình đang thực thi mất quyền

sử dụng CPU (chuyển sang trạng thái sẵn sàng hoặc phong tỏa) thì một tiến trình khác ở trạng thái sẵn sàng sẽ được cấp phát CPU (tiến trình này chuyển sang trạng thái thực thi). Phương pháp điều phối xác định (1) thời điểm tiến trình không được tiếp tục sử dụng CPU và (2) tiến trình nào được cấp phát CPU. Cơ chế điều phối quyết định cách thức bộ quản lý tiến trình xác định thời điểm điều phối CPU và cách thức cấp phát/thu hồi CPU.

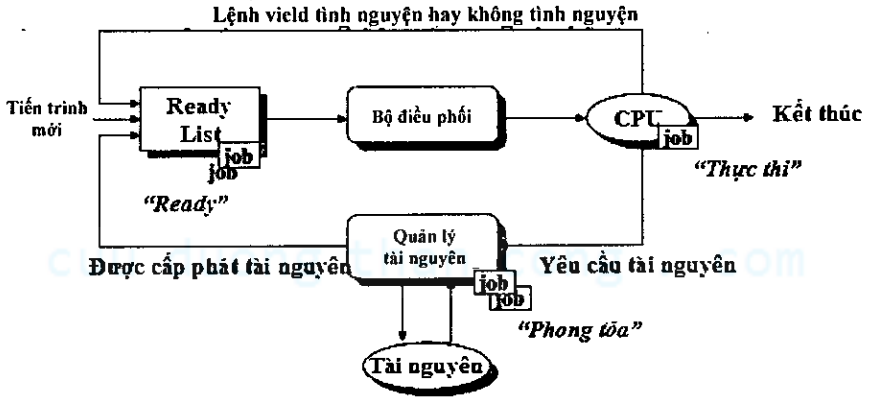
Hình 5.1 minh họa ba thành phần cơ bản của bộ điều phối là Enqueuer (bộ phận đưa tiến trình vào hàng đợi), Dispatcher (bộ điều vận) và Context Switcher (bộ chuyển ngữ cảnh). Khi tiến trình chuyển sang trạng thái sẵn sàng, khối PCB tương ứng được cập nhật đồng thời Enqueuer đặt con trỏ để khối PCB này nằm trong danh sách các tiến trình sẵn sàng sử dụng CPU. Danh sách này là cấu trúc dữ liệu của bộ quản lý tiến trình, thường được cài đặt dưới dạng hàng đợi các con trỏ, mỗi con trỏ "chỉ" tới PCB một tiến trình ở trạng thái sẵn sàng. Enqueuer tính chỉ số ưu tiên để cấp phát CPU cho tiến trình khi tiến trình được đưa vào trạng thái sẵn sàng, chỉ số ưu tiên cũng được xác định khi quyết định tiến trình nào sẽ bị loại khỏi danh sách sẵn sàng.

Khi chuyển quyền sử dụng CPU giữa hai tiến trình, bộ chuyển ngữ cảnh lưu lại giá trị của các thanh ghi trong CPU vào khối PCB của tiến trình cũ. Bộ phận nào gọi bộ phận chuyển ngữ cảnh sẽ xác định thời điểm chuyển quyền sử dụng CPU và ở đây có hai cơ chế là *tình nguyện* (tiến trình đang thực thi tự động giải phóng CPU bằng cách gọi bộ chuyển ngữ cảnh) và *không tình nguyện* (ngắt khiến chương trình đang thực thi phải từ bỏ quyền sử dụng CPU - trình xử lý ngắt thực hiện chuyển ngữ cảnh).



Hình 5.1. Cấu trúc bộ điều phối

Ngay sau đó, bộ điều vận được thực thi (ngữ cảnh của bộ điều vận được khôi phục) để lựa chọn và cấp phát CPU cho một tiến trình sẵn sàng nào đó (bằng cách chuyển ngữ cảnh sang tiến trình được lựa chọn). Hình 5.2 minh họa luồng đi của tiến trình trên hệ thống có một CPU. Tiến trình tự nguyện giải phóng CPU khi yêu cầu tài nguyên (chuyển quyền điều khiển cho bộ quản lý tài nguyên hoặc bộ điều phối). Tiến trình tự động giải phóng hoặc có thể bị cưỡng đoạt quyền sử dụng CPU. Tiến trình quay về trạng thái sẵn sàng khi tự nguyện (hoặc bị cưỡng ép) giải phóng CPU. Nếu như tiến trình giải phóng CPU do phải đợi tài nguyên, tiến trình sẽ không được quyền sử dụng CPU cho đến khi có đủ tài nguyên cần thiết. Sau đó, tiến trình mới chuyển sang trạng thái sẵn sàng và quay về hàng đợi sẵn sàng.



Hình 5.2. Luồng đi của các tiến trình trong hệ thống

5.1.2. Lưu lại ngữ cảnh của tiến trình

Rõ ràng mỗi lần chuyển quyền sử dụng CPU giữa hai tiến trình sẽ xuất hiện hai lần chuyển ngữ cảnh. Đầu tiên, ngữ cảnh của tiến trình đang chạy được lưu lại để khôi phục ngữ cảnh của bộ điều vận. Kế tiếp lưu lại ngữ cảnh bộ điều vận và tải ngữ cảnh của tiến trình mới. Công việc chuyển ngữ cảnh có thể ảnh hưởng lớn tới hiệu suất hệ thống, vì phải lưu lại giá trị nhiều thanh ghi bằng các chỉ thị LOAD và STORE. Do đó, nếu **n** là số lượng thanh ghi chung, **m** là số lượng thanh ghi trạng thái, **b** là số lượng chỉ thị cần thiết để lưu lại một thanh ghi và **K** là thời gian cần thiết để lưu lại một thanh ghi thì việc chuyển ngữ cảnh mất **(n + m) × b × K** đơn vị thời gian.

Giả sử thời gian lưu trữ một đơn vị thông tin vào bộ nhớ trong là 50ns (10^{-9}s). Giả sử bus kết nối giữa CPU và bộ nhớ có 16 đường và độ lớn

mỗi thanh ghi là 32 bit. Như vậy, để lưu giá trị một thanh ghi cần 2 × 50ns. Giả sử hệ thống có 32 thanh ghi đa mục đích và 8 thanh ghi trạng thái, tổng thời gian để lưu các thanh ghi là 40 × 2 × 50ns = 4ms. Cần thêm 4ms để tải các trạng thái của một tiến trình khác vào trong CPU (ở đây đã bỏ qua thời gian chạy của bộ điều vận). Như vậy, thời gian chuyển ngữ cảnh lớn hơn 8ms. Với bộ xử lý 200MHZ, một chỉ thị thực thi trong khoảng 10ns, thì trong 8ms, CPU có thể thực hiện 800 chỉ thị có ích.

Để giảm thời gian chuyển ngữ cảnh, một số kiến trúc máy tính có nhiều bộ thanh ghi, mỗi bộ thanh ghi ứng với một tiến trình. Thời gian chuyển ngữ cảnh giảm xuống vì chỉ cần lựa chọn bộ thanh ghi nào được đưa ra sử dụng (không cần lưu lại giá trị các thanh ghi).

5.1.3. Chia sẻ tự nguyện CPU

Khi nào bộ điều phối hoạt động? Cách đơn giản nhất là mỗi tiến trình định kỳ gọi bộ điều phối, tình nguyện từ bỏ quyền sử dụng CPU. Một số phần cứng có chỉ thị **yield** cho phép tiến trình làm việc này. Chỉ thị này tương tự các lời gọi thủ tục, vì phải lưu lại địa chỉ chỉ thị thực thi kế tiếp và sau đó rẽ nhánh tới một địa chỉ khác. Điểm khác biệt ở đây là địa chỉ của chỉ thị tiếp theo không được lưu trong ngăn xếp của tiến trình mà lưu trong vùng nhớ được thiết kế sẵn. Cũng có sự tương đồng giữa hàm **yield** và trình xử lý ngắt. Chỉ thị **yield** có thể được cài đặt như sau:

```
yield(r, s)
{
    memory[r] = PC:
    PC = memory[s];
}
```

Khi tiến trình p_1 thực hiện **yield(r,s)**, thì tham số **r** đóng vai trò định danh của tiến trình p_1 (là địa chỉ khối PCB ứng với p_1). Có thể xác định **r** bằng cách xem tiến trình nào đang chiếm dụng CPU. Thường trong CPU có thanh ghi trạng thái tiến trình. Khi chuẩn bị thực thi, nội dung một số trường trong PCB ứng với tiến trình được tải vào thanh ghi trạng thái tiến trình. Giá trị **r** được xác định bằng cách đọc nội dung thanh ghi trạng thái này. Để đơn giản, rút gọn **yield(r,s)** thành **yield(*, s)**, trong đó * xác định tiến trình đang thực thi (là tiến trình gọi **yield**). Tham số **s** xác định tiến trình được thực thi kế tiếp – thực thi ngay sau khi chỉ thị **yield()** của tiến trình thứ nhất thực thi xong. Trong lúc thực thi p_1, trước khi thực thi **yield**, **memory[r]** không xác

định. Sau khi **yield** thực thi, **memory[r]** chứa địa chỉ chỉ thị đứng ngay sau **yield** trong tiến trình p_1 (để sau này có thể khôi phục) và thanh ghi PC chứa địa chỉ của chỉ thị nào đó trong tiến trình p_2 sẽ được thực thi kế tiếp. Quyền sử dụng CPU được chuyển từ tiến trình gọi **yield** sang tiến trình mà địa chỉ của chỉ thị cuối cùng được thực thi lưu trong **memory[s]**. Giả sử **memory[s]** chứa địa chỉ của chỉ thị cuối cùng được thực thi trong p_2, tiến trình p_1 có thể thực thi **yield(*, s)** để chuyển quyền điều khiển cho p_2 và tương tự sau này p_2 có thể dùng **yield(*,r)** để khởi động lại p_1.

Nếu có nhiều tiến trình ở trạng thái sẵn sàng sử dụng CPU, p_2 có thể đóng vai trò bộ điều phối với định danh **scheduler**. Khi đó tiến trình có thể thực hiện **yield(*, scheduler)** khi không muốn sử dụng CPU. Sau đó bộ điều phối lựa chọn tiến trình s ở trạng thái sẵn sàng và gọi **yield(*, s)**. Dạng cộng tác đa chương trình này được phát triển trên dòng máy cá nhân của Xerox Alto. Nhiều nhân viên của Xerox đã chuyển việc sang Apple nên công nghệ cộng tác này đã được đưa vào các phiên bản đầu tiên của HĐH Macintosh.

Tuy nhiên, **yield** có nhược điểm là tiến trình có thể không tự nguyện "hợp tác" với các tiến trình khác. Nếu tiến trình không gọi **yield** thì không tiến trình nào sử dụng được CPU cho tới khi tiến trình đang sử dụng CPU thực hiện vào/ra. Điều này trở nên khó khăn nếu tiến trình đang thực thi lại nằm trong vòng lặp vô hạn và không thực hiện vào/ra. Khi đó tiến trình sẽ không bao giờ giải phóng CPU và kết quả các tiến trình khác phải đợi vô hạn. Do đó, hệ thống phải có khả năng định kỳ ngắt các tiến trình đang chạy, tức là một hình thức chia sẻ không tình nguyện.

5.1.4. Chia sẻ CPU không tình nguyện

Ngắt có thể định kỳ ngưng quá trình thực thi của tiến trình, nói cách khác là "ép" tiến trình thực thi chỉ thị **yield**. Điều này có thể được thực hiện nhờ bộ định thời bên trong hệ thống tạo ra ngắt theo chu kỳ. Thiết bị này được sử dụng như sau: người lập trình hệ thống thiết lập một khoảng chu kỳ thời gian cho thiết bị. Cứ mỗi lần hết một chu kỳ thời gian, thiết bị tạo ra một cảnh báo dưới dạng ngắt. Hình 5.3 minh họa cơ chế đơn giản của bộ định thời – thủ tục **IntervalTimer**. Mỗi lần đồng hồ thời gian thực (đồng hồ thạch anh trong máy tính) thực hiện xong T lần dao động của chu kỳ tinh thể thạch anh (T xung), thiết bị phần cứng định thời kết thúc một thao tác. Biến yêu cầu ngắt **InterruptRequest** nhận giá trị TRUE tương ứng với phần cứng

thiết lập cờ yêu cầu ngắt. Kết quả là cứ hết **K*T** xung (đơn vị thời gian dao động của tinh thể thạch anh), cờ yêu cầu ngắt được thiết lập. Khoảng thời gian **K** (số xung) có thể được thiết lập nhờ hàm **SetInterval** minh họa trong Hình 5.3. Bộ định thời theo kiểu này được gọi là bộ định thời có thể lập trình được. Ngắt sinh ra sau mỗi **K** xung khiến bộ điều khiển đồng hồ gọi bộ điều khiển ngắt. Về mặt chức năng logic, chỉ thị này tương đương **yeild**. Trình điều khiển thiết bị ngắt thời gian gọi bộ điều phối. Tiến trình đang thực thi không gọi **yeild**, nhưng hệ thống vẫn bảo đảm bộ điều phối chạy sau mỗi **K** xung. Bộ điều phối sử dụng cơ chế chia sẻ CPU không tình nguyện còn được gọi là bộ điều phối không độc quyền (preemptive).

```
IntervalTimer() {
    InterruptCount--;
    if(InterruptCount <= 0) {
        InterruptRequest = TRUE;
        InterruptCount = K;
    }
}

SetInterval(programmableValue) {
    K = programmableValue:
    InterruptCount = K;
}
```

cuu duong than cong . com

Hình 5.3. IntervalTimer

5.1.5. Hiệu suất

Bộ điều phối có ảnh hưởng lớn tới hiệu suất máy tính. Nếu bộ điều phối cấp phát CPU ngay cho tiến trình sẵn sàng thực thi, tiến trình sẽ tốn ít thời gian nằm chờ ở hàng đợi và được sử dụng CPU ngay khi cần thiết. Khi đó, hiệu suất chỉ phụ thuộc vào tốc độ phần cứng máy tính. Ngược lại, nếu tiến trình bị "bỏ quên" trong hàng đợi, khoảng thời gian tiến trình nằm tại hàng đợi lớn hơn rất nhiều so với khoảng thời gian sử dụng CPU. Hiệu suất cũng bị tác động bởi thời gian thực hiện chuyển ngữ cảnh. Chi phí phụ trội này lại bị ảnh hưởng bởi phần cứng. Một nhân tố khác ảnh hưởng đến hiệu suất là chính sách điều phối - xác định thời gian tiến trình đợi để được sử dụng CPU khi tiến trình đã ở trạng thái sẵn sàng. So với yếu tố thời gian chuyển ngữ cảnh, yếu tố về chính sách ảnh hưởng tới hiệu suất nhiều hơn. Ở đây chúng ta xét ảnh hưởng của điều phối tới hiệu suất tổng thể hệ thống. Thông thường, tại một thời điểm có một vài tiến trình sẵn sàng thực thi. Khi nào bộ

điều vận chọn tiến trình được thực thi kế tiếp? Tiêu chí nào được sử dụng trong quá trình lựa chọn? Tiến trình mãi mãi bị "bỏ quên" trong hàng đợi sẽ không bao giờ sử dụng được CPU để hoàn thành công việc của mình, hiện tượng này gọi là "chết đói". Nếu tiến trình được lựa chọn ngay khi sẵn sàng, khi đó thời gian thực thi thực sự của tiến trình sẽ tiệm cận với tốc độ phần cứng. Phương pháp và chính sách điều phối sẽ định ra tiêu chí lựa chọn tiến trình thực thi kế tiếp. Các cơ chế trình bày ở đây được sử dụng để cài đặt một phương pháp đã được người quản trị hoặc người thiết kế HĐH lựa chọn. Có những phương pháp tập trung vào hiệu suất hệ thống tổng thể, có phương pháp mong muốn chia sẻ công bằng CPU giữa các tiến trình, thậm chí có phương pháp cố gắng tối ưu hiệu suất của một lớp tiến trình cụ thể. Nói chung, hiệu suất sẽ quyết định lựa chọn phương pháp điều phối phù hợp.

5.2. CÁC PHƯƠNG PHÁP ĐIỀU PHỐI

Làm thế nào để bộ điều phối cấp phát CPU cho các tiến trình theo mục tiêu cụ thể? Có nên cấp phát dựa theo độ ưu tiên không? Có nên cấp phát công bằng không? Có nên gán độ ưu tiên cao cho các tiến trình có thời gian thực thi ngắn (hoặc dài) không? Những phương pháp cơ bản như vậy đã được nghiên cứu trong nhiều năm và tương tự bài toán điều phối trong ngành vận trù, chẳng hạn cách thức phục vụ khách hàng trong ngân hàng. Ví dụ trong hệ thống thời gian thực, các tiến trình phải được cấp phát CPU sớm để có thể kết thúc công việc trước mốc thời gian cụ thể. Trong hệ thống chia sẻ thời gian, mục tiêu điều phối là phân chia công bằng thời lượng sử dụng CPU giữa các tiến trình hay giữa các người dùng, hoặc giảm thiểu thời gian phản hồi cho người dùng. Phương pháp thích hợp phụ thuộc vào mục tiêu của HĐH.

cuu duong than cong . com

Trong HĐH hiện đại, mức độ ưu tiên trong (hay đơn giản là mức độ ưu tiên) quyết định thứ tự sử dụng CPU của tiến trình. Qua đó có thể cài đặt bất cứ phương pháp điều phối nào. Ví dụ, với điều phối theo độ ưu tiên ngoài, người dùng được gán một mức ưu tiên. Độ ưu tiên (trong) của bất kỳ tiến trình nào do người dùng tạo ra bằng độ ưu tiên của người dùng. Độ ưu tiên có thể được gán động. Giả sử mục đích chia sẻ cộng bằng CPU (nếu n tiến trình hoạt động trong K đơn vị thời gian thì mỗi tiến trình được phép sử

dụng CPU trong K/n đơn vị thời gian). Có thể thực hiện điều này bằng cách tăng độ ưu tiên của tiến trình đang nằm trong hàng đợi sẵn sàng, nhưng giảm độ ưu tiên của tiến trình đang thực thi. Các phương thức điều chỉnh độ ưu tiên ứng với các phương pháp điều phối khác nhau.

Ngắt đơn giản hóa việc cài đặt phương pháp chia sẻ CPU không tình nguyện. Nếu bộ quản lý tiến trình sử dụng bộ định thời (timer) điều khiển việc chuyển quyền sử dụng CPU, thì hệ thống xác định lượng tử thời gian là khoảng thời gian giữa hai ngắt liên tiếp. Trong trường hợp tiến trình kết thúc trước khi sử dụng hết lượng tử thời gian, tiến trình giải phóng CPU và bộ điều phối cấp phát CPU cho tiến trình khác. Dĩ nhiên, khi đó tiến trình mới phải được cấp phát nguyên một lượng tử thời gian, vì vậy bộ điều phối phải có khả năng thiết lập lại giá trị cho bộ định thời.

Với tập hợp cụ thể các tiến trình nằm trong hàng đợi sẵn sàng và thời lượng sử dụng CPU của mỗi tiến trình xác định, thì bộ điều phối không độc quyền hoàn toàn có thể xác định kế hoạch tối ưu cho một mục đích cụ thể (với điều kiện không đưa thêm tiến trình mới vào hàng đợi trong quá trình phục vụ các tiến trình khác). Thuật toán tối ưu xác định số lượng lượng tử thời gian sử dụng CPU của mỗi tiến trình, sau đó liệt kê tất cả các kế hoạch sử dụng CPU theo thứ tự nào đó. Phương pháp điều phối tối ưu căn cứ theo tiêu chí nào đó sẽ rà soát toàn bộ các kế hoạch nhằm chọn ra cái tốt nhất. Tuy nhiên, có một vài vấn đề trong phương pháp này:

– Khi có tiến trình mới đến hàng đợi trong khi hệ thống đang phục vụ các tiến trình hiện tại thì phải lập kế hoạch lại.

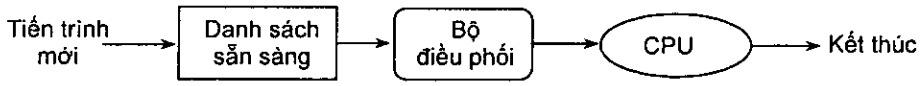
– Phải xác định được thời lượng sử dụng CPU thực tế của mỗi tiến trình trước khi tiến trình thực thi. Tuy nhiên, khó thực hiện được điều này.

– Các thuật toán điều phối tối ưu cho n tiến trình có độ phức tạp $O(n^2)$, nghĩa là lượng thời gian bộ điều phối sử dụng để lập kế hoạch tối ưu lớn hơn lượng thời gian thực sự dành cho việc phục vụ các tiến trình.

Giả sử: $P = \{p_i \mid 0 \le i < n\}$ là tập các tiến trình. $S(p_i)$ là trạng thái của tiến trình p_i, trong đó $S(p_i) \in$ {đang thực thi, sẵn sàng, phong tỏa}. Thời gian phục vụ $\tau(p_i)$ là tổng lượng thời gian sử dụng CPU cần thiết để tiến trình p_i hoàn thành nhiệm vụ. Thời gian chờ $W(p_i)$ là thời gian chờ của tiến trình, được tính từ khi tiến trình vào hàng đợi cho đến thời điểm đầu tiên

tiến trình sử dụng CPU. Thời gian lưu lại trong hệ thống $T_{TRnd}(p_i)$ là lượng thời gian từ khi tiến trình vào hàng đợi cho đến khi tiến trình hoàn tất công việc.

Mô hình tiến trình và các đại lượng thời gian trên được sử dụng khi so sánh hiệu suất các thuật toán điều phối. Trong hệ thống lô, thời gian lưu lại hệ thống là yếu tố quan trọng nhất, vì nó phản ánh lượng thời gian người dùng phải đợi để nhận được kết quả từ máy tính. Ở đây, thời gian lưu lại trong hệ thống trung bình là thời gian trung bình để hoàn thành một tiến trình (hay công việc) và nghịch đảo của đại lượng này là thông lượng hệ thống - số lượng công việc hoàn thành trong một đơn vị thời gian. Trong hệ thống lô, về mặt kỹ thuật, thời gian lưu lại hệ thống của công việc khác với thời gian lưu lại trong hệ thống của tiến trình do phải tính đến thời gian xếp hàng, cấp phát bộ nhớ và điều phối. Vì hệ thống lô quan tâm đến công việc nhiều hơn tiến trình, nên thời gian lưu lại hệ thống của công việc đóng vai trò quan trọng. Hệ thống chia sẻ thời gian thường tập trung vào thời gian thực thi riêng lẻ của từng tiến trình, chẳng hạn thời gian thi hành lệnh của người dùng. Thời gian này lại được chia thành hai giai đoạn là thời gian chờ đợi (do phải cạnh tranh quyền sử dụng CPU) và thời gian phục vụ. Với người dùng đang tương tác trực tiếp với máy tính, thì thời gian cần thiết để máy tính gửi lại một vài thông tin phản hồi nào đó (tức là thời gian chờ) có yếu tố rất quan trọng. Do vậy, trong hệ thống chia sẻ thời gian, đại lượng thời gian chờ được xem là tiêu chí đánh giá hiệu suất quan trọng nhất.



Hình 5.4. Mô hình điều phối đơn giản

Chú ý, mô hình tiến trình trên không tính đến hành vi của bộ quản lý tài nguyên (ngoại trừ khi xét đến trạng thái phong tỏa). Tiến trình có thể yêu cầu thời gian sử dụng CPU là khoảng liên tục, hoặc chia thành các khoảng thời gian sử dụng tại các thời điểm khác nhau, xen kẽ với các yêu cầu tài nguyên. Mô hình điều phối tiến trình được giản thể để bỏ qua tác động của việc tranh chấp tài nguyên (ngoại trừ tài nguyên CPU). Hình 5.4 minh họa điều này, tiến trình chỉ có thể ở một trong hai trạng thái là đang thực thi hoặc sẵn sàng. Các đại lượng đo hiệu suất trình bày ở trên, ngoại trừ thời

gian lưu lại trong hệ thống bỏ qua thời gian tiến trình bị phong tỏa do phải đợi tài nguyên. Mô hình này hoạt động như sau: Tiến trình lần đầu tiên bước vào trạng thái sẵn sàng sẽ yêu cầu thời gian sử dụng CPU xác định. Tiến trình sau khi được đáp ứng, có nghĩa là tổng thời gian thực thi trên CPU bằng thời gian phục vụ yêu cầu – sẽ kết thúc. Trên thực tế, có thể xem tiến trình được mô hình theo kiểu này khi tạo ra đã có đủ tài nguyên cần thiết (do đó sẵn sàng thực thi) và tiến trình sẽ kết thúc khi yêu cầu thêm tài nguyên mới. Các tiến trình trình bày trong các chương trước có thể xem như một chuỗi "nhỏ" các tiến trình không bị gián đoạn. Các thuật toán điều phối được chia làm hai lớp là lớp độc quyền (nonpreemptive) và lớp không độc quyền (preemptive). Trong thuật toán độc quyền, khi đã thực thi, tiến trình sẽ không giải phóng CPU cho đến khi hoàn thành. Các thuật toán không độc quyền thường dựa trên độ ưu tiên sau: tại bất kỳ thời điểm nào tiến trình đang sử dụng CPU luôn có độ ưu tiên cao nhất. Nếu tiến trình mới xuất hiện có độ ưu tiên cao hơn tiến trình hiện tại đang thực thi, thì tiến trình đang sử dụng CPU phải nhường quyền sử dụng CPU cho tiến trình mới.

5.2.1. Phân tiến trình thành nhiều tiến trình nhỏ

Trong quá trình hoạt động, các thao tác tính toán và vào/ra của tiến trình xen kẽ nhau. Giả sử tiến trình có k lần tính toán trên CPU và k lần thực hiện vào/ra. Như vậy, tổng thời gian phục vụ có thể là: $\tau(p_i) = \tau_1 + \tau_2 + \ldots + \tau_k$. Giả sử tiến trình p_i được chia thành k tiến trình nhỏ $p_{i1}, p_{i2}, \ldots, p_{ik}$, trong đó τ_{ij} là thời gian sử dụng CPU của p_{ij}. Mỗi p_{ij} sẽ được thi hành như một tiến trình độc lập và không gián đoạn. Bộ điều phối không độc quyền có thể chia τ_{ij} ra nhiều lượng tử thời gian trong quá trình điều phối cho p_{ij}. Nếu mỗi tiến trình yêu cầu k thao tác vào/ra khác nhau thì τ_i xen kẽ với các thời gian $d_1, d_2, \ldots$, trong đó d_k là thời gian thực hiện thao tác vào/ra thứ k. Vì thế, tổng thời gian tiến trình sử dụng CPU và thời gian thực hiện vào/ra là: $\tau_1 + d_1 + \tau_2 + d_2 + \ldots + \tau_k + d_k$. Với tiến trình hướng tính toán, τ_i tương đối lớn so với d_j. Với tiến trình hướng vào/ra, τ_i tương đối nhỏ so với d_j.

5.2.2. Đánh giá xấp xỉ tải hệ thống

Phụ thuộc vào các mục đích sử dụng hệ thống, các thuật toán điều phối sử dụng các tiêu chí khác nhau để lựa chọn tiến trình từ hàng đợi. Người ta có hai lựa chọn:

(1) Phân tích một thuật toán cụ thể trên một tải giả thiết và sau đó dự đoán hiệu suất của từng thuật toán.

(2) Sử dụng một tải thực và đánh giá hiệu suất từng thuật toán trên tải thực. Mục đích trong giáo trình này là đánh giá các phương pháp điều phối khác nhau chứ không tập trung nghiên cứu sâu vào việc dự đoán hiệu suất. Tuy vậy, chúng ta vẫn xét hiệu suất khi so sánh các thuật toán khác nhau. Mục đích việc nghiên cứu ở đây là giúp bạn đọc hình dung được các thuật toán thi hành như thế nào trên các tải khác nhau. Người ta dự đoán hiệu suất của từng thuật toán trên một tải ngẫu nhiên bất kỳ bằng cách tính trung bình cộng các đại lượng đo trên từng tiến trình riêng lẻ (vì xác suất tiến trình đi đến hàng đợi phân bố ngẫu nhiên).

Tải vào hệ thống có thể được mô tả bởi tốc độ tiến trình đến hàng đợi và thời gian phục vụ $\tau(p_i)$. Giả sử λ là số lượng trung bình các tiến trình đến hàng đợi (được tính là số lượng các tiến trình đến hàng đợi trong một đơn vị thời gian), $1/\lambda$ sẽ là thời gian trung bình giữa hai lần đến liên tiếp. Giả sử μ biểu diễn tốc độ phục vụ trung bình ($1/\mu$ sẽ là thời gian phục vụ trung bình của tiến trình). Nếu bỏ qua thời gian chuyển ngữ cảnh và giả thiết CPU có đủ khả năng đáp ứng toàn bộ tải, thì khoảng thời gian CPU bận là

$$\rho = \lambda \times \frac{1}{\mu} = \frac{\lambda}{\mu}.$$

Nếu $\rho > 1$, CPU sẽ bị "quá tải" (lượng công việc đến lớn hơn khả năng phục vụ) và điều này không phụ thuộc vào việc sử dụng bất kỳ thuật toán điều phối nào. Đồng thời, nếu kích thước hàng đợi hữu hạn thì xuất hiện tình trạng tràn bộ đệm (overflow), vì tốc độ đến của các tiến trình lớn hơn tốc độ hệ thống có thể phục vụ. Hệ thống chỉ có thể tiến tới trạng thái ổn định khi $\lambda < \mu$ ($\rho < 1$).

Ví dụ, trong 1 phút có 10 tiến trình đến hệ thống (nghĩa là $\lambda = 10$ tiến trình/phút) và thời gian phục vụ trung bình cho mỗi tiến trình là 3 giây (nghĩa là: $1/\mu = 3$ giây = 1/20 phút hoặc $\mu = 20$ tiến trình/phút). Tải hệ thống sẽ là: $\rho = \lambda/\mu$ = (10 tiến trình/phút) / (20 tiến trình/phút) = 0,5 = 50%.

5.3. THUẬT TOÁN ĐỘC QUYỀN

Thuật toán độc quyền cho phép tiến trình chiếm dụng CPU thực thi cho tới khi hoàn tất công việc. Trong hệ thống sử dụng thuật toán độc quyền,

không tiến trình nào chuyển từ trạng thái thực thi sang trạng thái sẵn sàng. Chỉ khi hoàn thành nhiệm vụ, tiến trình mới nhường quyền sử dụng CPU cho bộ điều phối. Thuật toán độc quyền "vay mượn" khá nhiều thuật toán kinh điển trong vận trù học (Nghiên cứu các phương pháp điều phối liên quan đến con người, chẳng hạn cách thức điều phối việc phục vụ khách hàng trong ngân hàng, sân bay hay siêu thị. Khi bắt đầu được phục vụ, khách hàng sẽ nhận được toàn bộ sự phục vụ cho đến khi xong việc mà không bị người khác chen ngang). Phương pháp này phù hợp với hệ thống không sử dụng ngắt thời gian để gọi bộ điều phối. Tại mỗi thời điểm có duy nhất một tiến trình sử dụng CPU và sau khi thực hiện xong sẽ nhường CPU cho tiến trình.

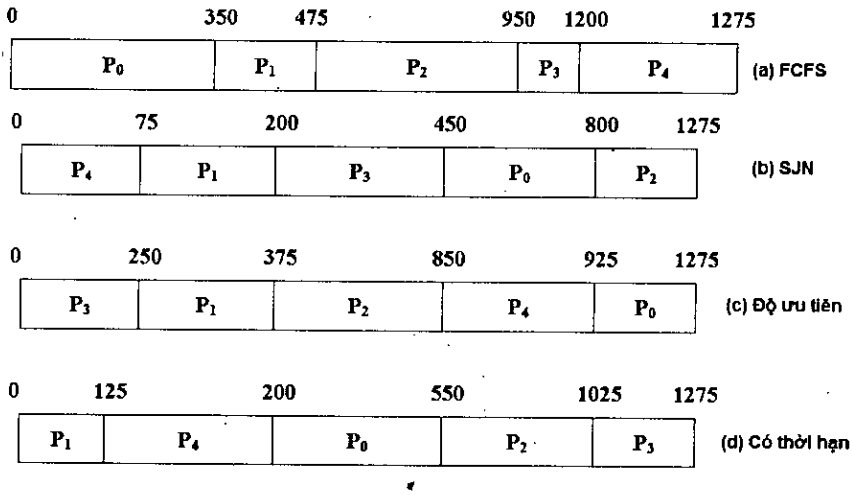
5.3.1. Đến trước phục vụ trước (First-Come-First-Served – FCFS)

Trong thuật toán FCFS, độ ưu tiên của tiến trình phụ thuộc vào thời điểm tiến trình được đưa vào hàng đợi. Bộ phận Enqueuer gắn cho tiến trình nhãn thời gian - là thời điểm tiến trình vào hàng đợi. Sau đó bộ điều vận lựa chọn tiến trình nào có nhãn thời gian bé nhất để thực thi. Hàng đợi sẵn sàng có thể được cài đặt bằng cấu trúc dữ liệu FIFO (First-In-First-Out) đơn giản (trong đó mỗi thành phần trong cấu trúc dữ liệu trỏ tới PCB của một tiến trình). Enqueuer đưa tiến trình mới vào đuôi hàng đợi, còn bộ điều vận chọn tiến trình ở đầu hàng đợi. Mặc dù dễ cài đặt, nhưng FCFS không quan tâm đến thời gian phục vụ và cũng bỏ qua các tiêu chí đánh giá chất lượng khác như thời gian đợi hay thời gian lưu lại hệ thống trung bình. Nói chung FCFS ít được áp dụng trên các hệ thống thực.

i	$\tau(p_i)$	Độ ưu tiên	Thời hạn chót
0	350	5	575
1	125	2	550
2	475	3	1050
3	250	1	Không xác định
4	75	4	200

Hình 5.5. Một tải giả thiết

Như minh họa trong Hình 5.5, giả sử có 5 tiến trình trong hàng đợi và thứ tự đến hàng đợi của chúng lần lượt là p_0, p_1, p_2, p_3, p_4. Thuật toán FCFS điều phối các tiến trình như trong Hình 5.6a.



Hình 5.6. Các kế hoạch điều phối theo thuật toán khác nhau

Qua Hình 5.6a, chúng ta xác định thời gian lưu lại hệ thống của mỗi tiến trình trong thuật toán điều phối FCFS:

$$T_{TRnd}(p_0) = \tau(p_0) = 350$$

$$T_{TRnd}(p_1) = \tau(p_0) + T_{TRnd}(p_0) = 125 + 350 = 475$$

$$T_{TRnd}(p_2) = \tau(p_2) + T_{TRnd}(p_1) = 475 + 475 = 950$$

$$T_{TRnd}(p_3) = \tau(p_3) + T_{TRnd}(p_2) = 250 + 950 = 1200$$

$$T_{TRnd}(p_4) = \tau(p_4) + T_{TRnd}(p_3) = 75 + 1200 = 1275$$

Thời gian phản hồi trung bình là:

$$\overline{T}_{TRnd} = \frac{350 + 475 + 950 + 1200 + 1275}{5} = 850$$

Thời gian đợi của các tiến trình lần lượt là:

$$W(p_0) = 0$$

$$W(p_1) = T_{TRnd}(p_1) = 375$$

$$W(p_2) = T_{TRnd}(p_2) = 475$$

$$W(p_3) = T_{TRnd}(p_3) = 950$$

$$W(p_4) = T_{TRnd}(p_4) = 1200$$

Thời gian đợi trung bình là:

$$\overline{W} = \frac{0 + 350 + 475 + 950 + 1200}{5} = 595$$

5.3.2. Công việc ngắn nhất phục vụ trước (Shortest Job Next – SJN)

Thuật toán điều phối SJN lựa chọn tiến trình có thời gian phục vụ ngắn nhất. Thời gian lưu lại hệ thống của tiến trình p_i là tổng thời gian phục vụ của tất cả các tiến trình nằm trong hàng đợi có thời gian phục vụ ít hơn p_i. SJN giảm thiểu thời gian đợi trung bình vì những tiến trình có thời gian phục vụ ngắn sẽ được thực thi trước. Tuy nhiên, trong trường hợp có rất nhiều tiến trình cần phục vụ, cơ chế điều phối này có thể ngăn cản những tiến trình có thời gian phục vụ lớn được quyền thực thi. Hiện tượng tiến trình lớn có thể không được phục vụ (chết đói) là khiếm khuyết lớn trong SJN. Chúng ta vẫn giả sử danh sách hàng đợi chứa các tiến trình minh họa trong Hình 5.5. Ở đây thứ tự đến không quan trọng mà chỉ cần tất cả tiến trình nằm trong hàng đợi tại thời điểm điều phối và trong quá trình điều phối không xuất hiện thêm tiến trình mới. Thuật toán SJN tạo ra bảng kế hoạch minh họa trên Hình 5.6b.

Từ Hình 5.6b, chúng ta xác định:

$T_{TRnd}(p_0) = \tau(p_0) + \tau(p_3) + \tau(p_1) + \tau(p_4) = 350 + 250 + 125 + 75 = 800$

$T_{TRnd}(p_0) = \tau(p_1) + \tau(p_4) = 125 + 75 = 200$

$T_{TRnd}(p_0) = \tau(p_2) + \tau(p_0) + \tau(p_3) + \tau(p_1) + \tau(p_4)$

$= 450 + 350 + 250 + 125 + 75 = 1275$

$T_{TRnd}(p_0) = \tau(p_3) + \tau(p_1) + \tau(p_4) = 250 + 125 + 75 = 450$

$T_{TRnd}(p_0) = \tau(p_4) = 75$

Thời gian lưu lại hệ thống trung bình là:

$$\overline{T}_{TRnd} = \frac{800 + 200 + 1275 + 450 + 75}{5} = 560$$

Thời gian đợi của các tiến trình như sau:

$W(p_0) = 450$

$W(p_1) = 75$

$W(p_2) = 800$

$W(p_3) = 200$

$W(p_4) = 0$

Thời gian đợi trung bình là:

$$\overline{W} = \frac{450 + 75 + 800 + 200 + 0}{5} = 350$$

SJF được áp dụng trên hệ thống theo lô để giảm thiểu thời gian lưu lại hệ thống trung bình và người ta cũng mong muốn đạt được tính chất này trên hệ thống tương tác. Trong hệ thống tương tác, người dùng đánh lệnh, hệ thống thực hiện lệnh, gửi kết quả; người dùng đánh tiếp lệnh, hệ thống tiếp tục xử lý,... Nếu coi lệnh là "công việc", chúng ta có thể giảm thiểu thời gian phản hồi trung bình bằng cách lựa chọn tiến trình cần ít CPU nhất để thực thi. Tuy vậy, khó xác định chính xác tiến trình nào có tính chất này mà chỉ có thể ước lược dựa trên các hành vi trong quá khứ. Giả sử thời gian thực thi của một lệnh trong lần đầu là T_0 và thời gian thực thi kế tiếp của lệnh này là T_1. Chúng ta có thể ước lược thời gian thực thi của lệnh bằng cách sử dụng thêm trọng số α: $\alpha T_0 + (1 - \alpha)T_1$. Giá trị trọng số α quyết định giá trị ước lược mới có phụ thuộc nhiều vào giá trị cũ hay không. Với $\alpha = 1/2$, ta lần lượt có giá trị các ước lược: T_0, $T_0/2 + T_1/2$, $T_0/4 + T_1/4 + T_2/2$, $T_0/8 + T_1/8 + T_2/4 + T_3/2$,... Sau ba lần thực thi, hệ số của T_0 trong giá trị ước lược chỉ còn 1/8.

5.3.3. Điều phối theo độ ưu tiên

Trong điều phối theo độ ưu tiên, tiến trình được cấp phát CPU căn cứ theo độ ưu tiên được gán từ bên ngoài (ở đây chúng ta coi số bé có độ ưu tiên cao). Độ ưu tiên trong được xác định căn cứ vào thao tác của tiến trình trong môi trường tính toán, chẳng hạn như độ ưu tiên được xác định theo thời gian phục vụ áp dụng trong SJN. Độ ưu tiên ngoài phản ánh tầm quan trọng của công việc, thường được người sử dụng xác định từ bên ngoài. Độ ưu tiên ngoài của tiến trình có thể bị người sử dụng thay đổi ("người quan trọng có quyền ưu tiên cao hơn"), bản chất công việc ("tiến trình phải tắt lò phản ứng hạt nhân khi nhiệt độ vượt ngưỡng nào đó"), hay bất kỳ một tiêu chuẩn ưu tiên nào khác.

Yếu tố quan trọng nhất đối với hiệu suất trong thuật toán điều phối theo độ ưu tiên là việc gán độ ưu tiên cho tiến trình. Trong cơ chế điều phối này, tiến trình có độ ưu tiên thấp vẫn có thể bị "chết đói". Tuy nhiên, có thể giải quyết vấn đề bằng cách tính độ ưu tiên từ bên trong. Hệ thống có thể dùng

khoảng thời gian đợi của tiến trình làm tham số để xác định độ ưu tiên. Thời gian đợi càng dài thì độ ưu tiên càng cao. Khi đó hiện tượng "chết đói" sẽ không xuất hiện. Chúng ta vẫn sử dụng ví dụ trong Hình 5.5. Cơ chế điều phối theo độ ưu tiên tạo ra bảng kế hoạch như Hình 5.6c.

Chúng ta tính:

$$T_{TRnd}(p_0) = \tau(p_0) + \tau(p_4) + \tau(p_2) + \tau(p_1) + \tau(p_3)$$
$$= 350 + 75 + 475 + 125 + 250 = 1275$$

$$T_{TRnd}(p_1) = \tau(p_1) + \tau(p_3) = 125 + 250 = 375$$

$$T_{TRnd}(p_2) = \tau(p_2) + \tau(p_1) + \tau(p_3) = 475 + 12 + 250 = 850$$

$$T_{TRnd}(p_3) = \tau(p_3) = 250$$

$$T_{TRnd}(p_4) = \tau(p_4) + \tau(p_2) + \tau(p_1) + \tau(p_3) = 75 + 475 + 125 + 250 = 925$$

Thời gian lưu lại hệ thống trung bình là:

$$\overline{T}_{TRnd} = \frac{1275 + 375 + 850 + 250 + 925}{5} = 735$$

Thời gian đợi của các tiến trình là:

$$W(p_0) = 925$$
$$W(p_1) = 250$$
$$W(p_2) = 375$$
$$W(p_3) = 0$$
$$W(p_4) = 850$$

Thời gian đợi trung bình là:

$$\overline{W} = \frac{925 + 25 + 375 + 0 + 850}{5} = 480$$

5.3.4. Điều phối có thời hạn (Deadline Scheduling)

Trong hệ thống thời gian thực, có những tiến trình phải được kết thúc trước một thời hạn về thời gian nhất định. Tiêu chí quan trọng để đánh giá hiệu suất là xem hệ thống có thể thực hiện xong các tiến trình trước thời hạn cuối cùng hay không. Các tiêu chí về thời gian đợi và thời gian lưu lại hệ thống không còn quan trọng. Bộ điều phối phải xác định được thời gian thực thi của tất cả các tiến trình. Tiến trình chỉ được chấp nhận vào hàng đợi nếu

bộ điều phối đảm bảo tiến trình có thể kết thúc trước thời hạn chót. Trong hệ thống truyền thông đa phương tiện, thời hạn thực hiện có thể được sử dụng để ngăn ngừa hiện tượng jitter (tốc độ đến đích của các gói tin không đều). Cột thứ 4 trong Hình 5.5 minh họa ví dụ về thời hạn cuối cho mỗi tiến trình. Hình 5.6d minh họa một kế hoạch đáp ứng được tiêu chí thời hạn cho toàn bộ tiến trình.

5.4. THUẬT TOÁN KHÔNG ĐỘC QUYỀN

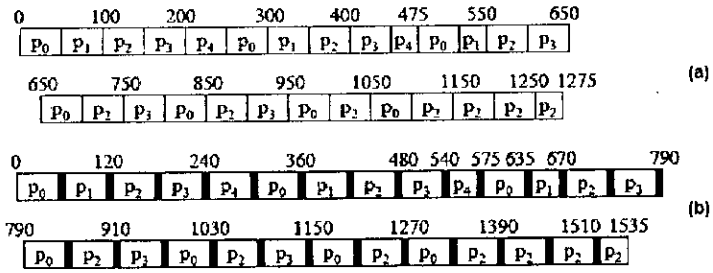
Trong thuật toán không độc quyền, tiến trình có độ ưu tiên cao nhất luôn được cấp phát CPU. Tiến trình có độ ưu tiên thấp phải nhường CPU cho tiến trình có độ ưu tiên cao. Bộ điều phối được gọi mỗi khi có tiến trình mới sẵn sàng thực thi hoặc sau khi hết mỗi lượng tử thời gian (nếu hệ thống sử dụng bộ định thời). Các thuật toán điều phối không độc quyền có thể đáp ứng nhanh chóng tiến trình có độ ưu tiên cao và bảo đảm các tiến trình chia sẻ công bằng CPU. Có thể sử dụng thuật toán SJN và thuật toán Độ ưu tiên theo thuật toán không độc quyền.

Với SJN, tiến trình có thời gian phục vụ nhỏ nhất được cấp phát CPU trước. Giả sử tiến trình p_j đến khi p_i đang thực thi, hệ thống so sánh $\tau(p_i)$ và $\tau(p_j)$. Trước khi p_j đến, p_i là tiến trình có thời gian phục vụ nhỏ nhất. Xét hệ thống có tải minh họa trong Hình 5.5 và sử dụng cơ chế điều phối SJN không độc quyền. Khi p_1 đang thực thi (sau khi p_4 thực thi xong), nếu xuất hiện tiến trình có thời gian phục vụ 35 thì hệ thống kiểm tra xem p_1 có cần nhiều hơn 35 đơn vị thời gian để thực hiện không. Nếu có, p_1 phải nhường CPU cho tiến trình mới. Tương tự, giả sử hệ thống không độc quyền sử dụng thuật toán điều phối theo độ ưu tiên với tải hệ thống minh họa trên Hình 5.5. Giả sử p_2 đang thực thi (sau p_3 và p_1) và xuất hiện tiến trình có độ ưu tiên 2 thì p_2 phải quay lại hàng đợi để nhường CPU cho tiến trình mới. Khi trình bày các thuật toán độc quyền, chúng ta bỏ qua chi phí chuyển ngữ cảnh giữa các tiến trình, vì giả định tiến trình hoàn thành công việc mà không bị gián đoạn. Trong thuật toán không độc quyền thuần túy, mỗi khi xuất hiện ngắt, tiến trình đang thực thi có thể phải nhường CPU cho tiến trình mới có độ ưu tiên cao hơn. Do đó, trong các hệ thống điều phối không độc quyền, chi phí chuyển ngữ cảnh là một yếu tố quan trọng.

5.4.1. Điều phối xoay vòng (Round Robin – RR)

Xoay vòng (RR) là thuật toán điều phối được sử dụng rộng rãi nhất, vì có thể phân chia công bằng thời gian sử dụng CPU giữa các tiến trình. Thuật toán này rất phù hợp trong hệ thống đa chương trình tương tác: Nếu trong hệ thống có n tiến trình thì mỗi tiến trình nhận được xấp xỉ 1/n đơn vị xử lý thời gian thực (xấp xỉ vì phải tính đến chi phí điều phối và chuyển ngữ cảnh). Nếu tiến trình kết thúc trước khi hết lượng tử thời gian, hệ thống phải khởi động bộ điều phối để cấp phát mới nguyên một lượng tử thời gian sử dụng CPU cho tiến trình khác. Tiến trình mới sẽ được đặt trong hàng đợi. Tuy nhiên, vị trí nào trong hàng đợi phụ thuộc vào phương pháp cài đặt. Nếu hàng đợi là danh sách liên kết vòng, tiến trình mới được đặt ngay sau tiến trình vừa được thực thi, như thế n – 1 tiến trình kia sẽ được phục vụ trước tiến trình mới. Nếu hàng đợi được cài đặt dưới dạng danh sách và bộ điều vận lựa chọn tiến trình theo thứ tự thì tiến trình mới được đặt ở cuối hàng đợi và điều này không phụ thuộc vào tiến trình nào đang thực thi khi tiến trình mới xuất hiện. Trung bình tiến trình mới phải đợi n/2 lượng tử thời gian trước khi được cấp phát CPU.

Xét ảnh hưởng của thời gian chuyển cảnh trong điều phối xoay vòng: Giả sử C là thời gian chuyển ngữ cảnh giữa các tiến trình người dùng (đôi khi coi C đủ nhỏ để có thể bỏ qua). Trong mỗi n(q + C) đơn vị thời gian thực, mỗi tiến trình trong n tiến trình nhận được q đơn vị thời gian sử dụng CPU. Có thể dễ dàng cài đặt cơ chế điều phối xoay vòng trên hệ thống sử dụng ngắt thời gian, vì có thể thiết lập để cứ sau một lượng tử thời gian, bộ định thời sẽ tạo ra ngắt. Sau đó, trình xử lý ngắt thời gian sẽ gọi bộ điều phối. Tiến trình đang ở trạng thái phong tỏa có thể chuyển sang trạng thái sẵn sàng khi có đủ tài nguyên cần thiết (trừ tài nguyên CPU). Ngắt do bộ định thời tạo ra giúp hệ thống xác định khi nào tiến trình đang thực thi sử dụng hết lượng tử thời gian. Khi đó bộ điều phối sẽ thực hiện các việc sau: điều chỉnh hàng đợi, thiết lập lại bộ định thời, cho phép tiến trình ở đầu hàng đợi thực thi. Giả sử danh sách các tiến trình ở trạng thái sẵn sàng như trong Hình 5.5; lượng tử thời gian cho mỗi lần cấp phát là 50 và bỏ qua thời gian chuyển ngữ cảnh. Kế hoạch được minh họa trên Hình 5.7a.



Hình 5.7. Kế hoạch điều phối theo thuật toán xoay vòng

Từ biểu đồ này, chúng ta xác định được thời gian lưu lại hệ thống của tiến trình:

$T_{TRnd}(p_0) = 1100$

$T_{TRnd}(p_1) = 550$

$T_{TRnd}(p_2) = 1275$

$T_{TRnd}(p_3) = 950$

$T_{TRnd}(p_4) = 475$

Thời gian lưu lại hệ thống trung bình là:

$$\overline{T}_{TRnd} = \frac{1100 + 550 + 1275 + 950 + 475}{5} = \frac{4350}{5} = 870.$$

Thời gian đợi của các tiến trình:

$W(p_0) = 0$

$W(p_1) = 50$

$W(p_2) = 100$

$W(p_3) = 150$

$W(p_4) = 200$

Thời gian đợi trung bình là:

$$\overline{W} = \frac{0 + 50 + 100 + 150 + 200}{5} = \frac{500}{5} = 100.$$

Thời gian đợi chứng tỏ ưu điểm của điều phối xoay vòng (cũng như các thuật toán dựa trên lượng tử thời gian) khi xét theo tiêu chí sau bao lâu tiến trình bắt đầu được phục vụ. Tuy nhiên, thời gian lưu lại hệ thống trung bình không khác mấy so với các thuật toán độc quyền.

Xét ví dụ trên nhưng tính thêm thời gian chuyển ngữ cảnh là 10 (Hình 5.7b). Thời gian lưu lại hệ thống của các tiến trình là:

$T_{TRnd}(p_0) = 1320$

$T_{TRnd}(p_1) = 660$

$T_{TRnd}(p_2) = 1535$

$T_{TRnd}(p_3) = 1140$

$T_{TRnd}(p_4) = 565$

Thời gian lưu lại hệ thống trung bình là:

$$\overline{T}_{TRnd} = \frac{1320 + 660 + 1535 + 1140 + 565}{5} = \frac{5220}{5} = 1044.$$

Thời gian đợi của từng tiến trình là:

$W(p_0) = 0$

$W(p_1) = 60$

$W(p_2) = 120$

$W(p_3) = 180$

$W(p_4) = 240$

cuu duong than cong . com

Thời gian đợi trung bình là:

$$\overline{W} = \frac{0 + 60 + 120 + 180 + 240}{5} = \frac{600}{5} = 120$$

5.4.2. Hàng đợi nhiều mức

Hàng đợi nhiều mức là dạng tổng quát của thuật toán điều phối theo độ ưu tiên: tất cả các tiến trình có cùng độ ưu tiên nằm trên cùng một hàng đợi. Có hai cấp độ điều phối CPU là giữa các hàng đợi (có độ ưu tiên khác nhau) và giữa các tiến trình nằm trong cùng hàng đợi (có cùng độ ưu tiên). Nếu sử dụng điều phối hàng đợi theo thuật toán không độc quyền thì phải phục vụ toàn bộ các tiến trình trong hàng đợi 1, rồi mới đến các tiến trình nào trong hàng đợi 2,... Trong mỗi hàng đợi k, CPU có thể được cấp phát theo bất kỳ thuật toán điều phối nào.

cuu duong than cong . com

Có thể cải tiến thuật toán này để phân bổ quyền sử dụng CPU giữa các hàng đợi chứ không phải luôn luôn ưu tiên hàng đợi có độ ưu tiên cao. Ví dụ, hàng đợi j có thể nhận được $1/2^j$ thời gian. Trong 100s thời gian thực

(bỏ qua thời gian chuyển ngữ cảnh) các tiến trình trong hàng đợi 1 nhận được 50s; hàng đợi 2 nhận được 25s; hàng đợi 3 nhận được 12,5s... Thuật toán điều phối càng phức tạp thì thời gian chuyển ngữ cảnh càng lớn, vì thế hầu hết các thuật toán điều phối trong hệ thống chia sẻ thời gian thường là hàng đợi nhiều mức đơn giản và trong mỗi hàng đợi áp dụng thuật toán xoay vòng.

☞ ***Tiến trình tiền cảnh và tiến trình hậu cảnh***

Các hệ thống chia sẻ thời gian thường hỗ trợ các tiến trình tiền cảnh (foreground) và tiến trình hậu cảnh (background). Các tiến trình tiền cảnh thực hiện tương tác với người dùng có độ ưu tiên cao, trong khi tiến trình hậu cảnh có độ ưu tiên thấp chỉ được thực thi khi không thực thi tiến trình tiền cảnh nào. Tiến trình tiền cảnh luôn luôn có độ ưu tiên cao hơn tiến trình hậu cảnh.

Có nhiều thuật toán đáp ứng kiểu phân chia tiến trình tiền cảnh/hậu cảnh. Chẳng hạn, tiến trình xử lý ngắt có thể chạy với mức ưu tiên 1, tiến trình điều khiển thiết bị ở mức 2, tiến trình tương tác với người dùng ở mức 3, trình soạn thảo ở mức 4, các công việc lô thông thường ở mức 5, còn các công việc lô cần nhiều thời gian thực thi ở mức 6. Dĩ nhiên, lựa chọn như trên chỉ mang tính tương đối, vì độ ưu tiên của tiến trình có thể thay đổi trong quá trình thực thi, phụ thuộc vào đang tính toán trong giai đoạn nào. Ví dụ, nếu tiến trình soạn thảo văn bản có tương tác với người dùng sử dụng quá nhiều CPU thì độ ưu tiên của nó có thể bị giảm (do sử dụng CPU vượt mức cho phép). Ngoài ra, có thể cho phép tiến trình tăng độ ưu tiên trong giai đoạn cần thực hiện nhiều tính toán với lý do người dùng cần chiếm dụng CPU trong thời gian tương đối lâu mới có một dịch vụ với chất lượng chấp nhận được. Hệ thống cho phép các tiến trình thay đổi độ ưu tiên trong các hàng đợi con, được gọi là một hàng đợi nhiều mức có phản hồi.

☞ ***Điều phối trong BSD UNIX***

BSD UNIX sử dụng cơ chế điều phối hàng đợi 32 mức: tiến trình hệ thống nằm trong hàng đợi 0 đến 7, tiến trình thực thi trong không gian người dùng nằm trong hàng đợi 8 đến 31. Bộ điều vận lựa chọn tiến trình từ hàng đợi có độ ưu tiên cao trước. Trong một hàng đợi, BSD UNIX sử dụng cơ chế điều phối xoay vòng. Lượng tử thời gian phụ thuộc vào phiên bản cài đặt, nhưng nói chung bé hơn 100μs.

Ngoài ra, tiến trình còn có độ ưu tiên ngoài, gọi là *nice* – được sử dụng để xác định (nhưng còn phụ thuộc vào các yếu tố khác) tiến trình sẽ nằm trong hàng đợi nào khi ở trạng thái sẵn sàng. Giá trị *nice* biến thiên từ –20 đến 20, trong đó –20 là mức ưu tiên cao nhất. Cứ sau mỗi một lượng tử thời gian, bộ điều phối tính lại độ ưu tiên của mỗi tiến trình, giá trị này là hàm số của *nice* và lượng thời gian tiến trình yêu cầu sử dụng CPU (yêu cầu càng nhiều thì độ ưu tiên càng giảm).

Hàm **sleep** tương tự **yield** (với việc lưu lại ngữ cảnh), khi tiến trình gọi **sleep**, bộ điều phối được gọi.

☞ ***Điều phối luồng trong Window NT***

Bộ điều phối luồng trong Window NT có nhiều mức có phản hồi, với mục tiêu phục vụ thật nhanh các luồng cần phải đáp ứng ngay lập tức. Bộ điều phối hỗ trợ 32 cấp độ điều phối khác nhau: 16 hàng đợi có mức ưu tiên cao nhất gọi là hàng đợi mức thời gian thực (real-time queue), tiếp theo là 15 hàng đợi, các hàng đợi có mức ưu tiên thay đổi được (variable level queues) và hàng đợi có mức ưu tiên thấp nhất là hàng đợi mức hệ thống (system level). Bộ điều phối cố gắng giới hạn số lượng các luồng được đưa vào các hàng đợi thời gian thực, nhằm làm giảm thiểu sự tranh chấp trong các luồng có mức độ ưu tiên cao. Tuy nhiên, Window NT không phải hệ thống thời gian thực, nên không đảm bảo luồng ở mức ưu tiên cao có thể nhận được quyền sử dụng CPU trước một thời hạn chót nào đó. Hàng đợi mức hệ thống là một "luồng rỗng" ứng với hệ thống rơi vào trạng thái nghỉ (idle system). Nghĩa là, khi trong hệ thống không có luồng nào khả thi, hệ thống sẽ thực thi luồng rỗng (là luồng không có trang trong bộ nhớ) cho tới khi xảy ra ngắt và xuất hiện luồng khả thi. Bộ điều phối hoàn toàn không độc quyền, có nghĩa là khi chuyển sang trạng thái sẵn sàng, luồng được đặt trong trong một hàng đợi nào đấy phụ thuộc vào độ ưu tiên.

5.4.3. Điều phối có đảm bảo (Guaranteed Scheduling)

Hệ thống có thể "hứa" đảm bảo một chất lượng nào đó cho người sử dụng và sau đó cố gắng thực hiện lời hứa. Một đảm bảo dễ "hứa" và dễ thực hiện là "Nếu có n người sử dụng hệ thống (hay tiến trình) thì mỗi người (hay tiến trình) sẽ nhận được 1/n năng lực xử lý CPU". Để thực hiện lời hứa, hệ thống phải kiểm soát được thời gian sử dụng CPU của tiến trình. Kế tiếp xác

định lượng thời gian mà hệ thống dành cho tiến trình (là khoảng thời gian tính từ khi tiến trình được tạo ra) chia cho n. Tỷ lệ hai đại lượng này là tỷ lệ thời gian tiến trình thực sự sử dụng CPU. Ví dụ, tỷ lệ 0,5 nghĩa là tiến trình mới sử dụng 1/2 thời gian được cấp phát của mình. Tiến trình có tỷ lệ thấp nhất sẽ được lựa chọn trước để chạy.

5.4.4. Điều phối quay xổ số (Lottery)

Hứa với người sử dụng và cố gắng thực hiện lời hứa là một ý tưởng hay, nhưng lại khó cài đặt. Thuật toán xổ số cũng đem lại kết quả tương tự nhưng dễ cài đặt hơn. Mỗi tiến trình được phát "vé xổ số" cho tài nguyên cần thiết (thời gian sử dụng CPU cũng là tài nguyên). Khi nào cần đưa ra quyết định cấp phát, bộ điều phối "quay xổ số". Tiến trình nào có "vé số" trúng thưởng được quyền sử dụng tài nguyên. Trong trường hợp điều phối CPU, hệ thống quay số 50 lần trong 1s và tiến trình "trúng số" sử dụng CPU trong 20ms. Hệ thống có thể cho những tiến trình quan trọng nhiều vé số (làm tăng cơ hội "trúng số"). Nếu hệ thống có 100 "vé số" và cấp phát cho một tiến trình 20 "vé số", tiến trình này được sử dụng 20% CPU (tính trong thời gian dài). Các tiến trình khi hợp tác có thể trao đổi "vé số" với nhau. Ví dụ, khi gửi thông điệp yêu cầu tới tiến trình Server, tiến trình Client gửi kèm toàn bộ "vé số" của mình và sau đó tự phong tỏa. Với lượng "vé" từ Client, cơ hội thực thi của Server tăng lên. Khi thực thi xong, Server gửi trả lại toàn bộ "vé số" trong thông điệp trả lời để Client nhanh chóng được thực thi. Điều phối theo kiểu quay xổ số được sử dụng trong nhiều hệ thống không thích hợp với các kiểu điều phối khác. Chẳng hạn, trong hệ thống cài đặt video server, một vài tiến trình gửi luồng âm thanh hình ảnh tới các khách hàng, nhưng theo các tỷ lệ nén khác nhau (10, 20, 25 frame/s). Bằng cách phân phối cho mỗi tiến trình trên lần lượt 10, 20, 25 "vé số", hệ thống có vẻ phân phối thời gian sử dụng CPU theo tỷ lệ 10 : 20 : 25.

cuu duong than cong . com

5.4.5. Điều phối công bằng

Từ trước đến giờ, chúng ta trình bày điều phối tiến trình nhưng không quan tâm đến người sử dụng (NSD) nào tạo ra tiến trình. Do đó, nếu NSD 1 tạo ra 9 tiến trình, NSD 2 tạo ra 1 tiến trình, bộ điều phối sử dụng cơ chế xoay vòng và các tiến trình có độ ưu tiên bằng nhau thì tiến trình 1 sẽ chiếm 90% CPU, trong khi NSD 2 chỉ được 10%. Để ngăn ngừa tình huống này, nhiều hệ thống đã xét tới NSD tạo ra tiến trình trước khi thực hiện điều phối.

Trong cơ chế này, mỗi NSD được cấp phát một thời lượng sử dụng CPU và quá trình điều phối phải bảo đảm điều này. Ví dụ, nếu hệ thống có 2 NSD, mỗi NSD được quyền chiếm 50% thời lượng CPU. NSD 1 có 4 tiến trình A, B, C và D; NSD 2 có 1 tiến trình E. Nếu điều phối theo kiểu xoay vòng, kết quả điều phối có thể là: **A, E, B, E, C, E, D, A, E, B, E, C, E, D, A, E, B, E, C, E, D, A, E, B, E, C, E, D**. Nếu NSD 2 được cấp phát CPU gấp đôi NSD 1 thì kết quả điều phối có thể là: **A, B, E, C, D, A, B, E, C, D, A, B, E, C, D, A, B, E, C, D, A, B, E, C, D, A, B, E, C, D.**

5.5. NHẬN XÉT

Bộ điều phối chịu trách nhiệm cấp phát CPU cho nhiều tiến trình sẵn sàng thực thi. Bộ điều phối được bộ định thời gọi định kỳ hoặc bất cứ khi tiến trình đang thực thi tự động giải phóng CPU qua chỉ thị **yield** hoặc yêu cầu tài nguyên. Từ danh sách các tiến trình ở trạng thái sẵn sàng, bộ điều phối chọn ra một tiến trình để cấp phát CPU. Thuật toán điều phối có thể được chia thành hai nhóm: độc quyền hoặc không độc quyền. Thuật toán độc quyền cho phép tiến trình chạy tới khi hoàn thành công việc, trong khi thuật toán không độc quyền sử dụng đồng hồ định khoảng thời gian và bộ điều phối sẽ định kỳ cấp phát CPU cho các tiến trình. Thuật toán FCFS và SJN có độ ưu tiên và thời hạn thuộc lớp thuật toán độc quyền, trong khi các thuật toán xoay vòng và hàng đợi nhiều mức thuộc nhóm không độc quyền. Thuật toán điều phối có thể được cài đặt theo nhiều cách khác nhau. Các bộ điều phối phức tạp trong BSD UNIX và Windows NT sử dụng cơ chế hàng đợi nhiều mức, nhưng có cải tiến. Điều phối được coi là "trái tim" của bộ quản lý tài nguyên CPU với trách nhiệm điều khiển việc chia sẻ CPU giữa nhiều tiến trình. Một khi môi trường máy tính cho phép nhiều tiến trình thực hiện đồng thời thông qua điều phối, thì bộ quản lý tiến trình phải có cơ chế cho phép các tiến trình phối hợp hoạt động với nhau. Điều này sẽ được trình bày trong chương kế tiếp.

CÂU HỎI ÔN TẬP

1. Trình bày nhiệm vụ chính của bộ điều phối tiến trình.
2. Tại sao phải lưu lại ngữ cảnh của tiến trình khi chuyển đổi?
3. Trình bày các thuật toán điều phối chính.

Chương 6
TƯƠNG TRANH VÀ ĐỒNG BỘ

*Việc quản lý tiến trình và luồng trong HĐH hiện đại được chia thành ba loại: **đa chương trình** (multiprogramming) – nhiều tiến trình chạy đồng thời trên một CPU; **đa xử lý** (multiprocessing) – nhiều tiến trình hoạt động trong hệ thống có nhiều CPU và **xử lý phân tán** (distributed system) – nhiều tiến trình hoạt động trong môi trường phân tán. Trong cả 3 kiểu trên, vấn đề thiết kế cơ bản vẫn là Tương tranh – các tiến trình hoạt động đồng thời. Mặc dù lợi ích rất lớn, nhưng tương tranh cũng làm nảy sinh nhiều vấn đề, chẳng hạn như cạnh tranh tài nguyên, bế tắc,... Truy cập đồng thời tới dữ liệu chia sẻ có thể gây ra sự không nhất quán trong dữ liệu. Chương này sẽ trình bày các kỹ thuật khác nhau nhằm đảm bảo trong cùng không gian địa chỉ logic, khi phối hợp với nhau, các tiến trình hoạt động theo một trật tự nào đó để duy trì tính thống quán của dữ liệu.*

6.1. CÁC KHÁI NIỆM CƠ BẢN

Có hai cơ chế phối hợp giữa các tiến trình là chia sẻ trực tiếp với nhau qua không gian địa chỉ logic hoặc chia sẻ dữ liệu thông qua file (tiến trình viết dữ liệu vào file và tiến trình khác đọc file). Tiến trình được gọi là cộng tác (cooperating), nếu có thể ảnh hưởng tới hoặc bị ảnh hưởng bởi tiến trình khác. Tiến trình dùng chung dữ liệu với tiến trình khác được xem là tiến trình cộng tác.

6.1.1. Hợp tác qua chia sẻ

Có nhiều lý do để tạo ra môi trường cộng tác, đó là:

- **Chia sẻ thông tin:** Khi nhiều người dùng cùng muốn sử dụng một tài nguyên thông tin nào đó (ví dụ file), thì môi trường hệ thống phải cho phép nhiều tiến trình đồng thời truy cập đến tài nguyên.

- **Tăng tốc độ tính toán:** Nếu muốn một nhiệm vụ hoàn thành trong thời gian ngắn nhất có thể, chúng ta có thể chia nhiệm vụ thành các nhiệm vụ nhỏ và các nhiệm vụ nhỏ có thể được thực thi song song. Chú ý, việc tăng tốc này chỉ có thể thực hiện khi hệ thống có nhiều đơn vị tính toán độc lập (có nhiều CPU hay các kênh vào/ra).
- **Tính module (hay tính tách biệt):** Chúng ta mong muốn chia các chức năng hệ thống thành các tiến trình riêng biệt như đã phân tích trong Chương 3.
- **Tính thuận tiện:** Người dùng có thể thực hiện song song nhiều việc như soạn thảo, in và biên dịch chương trình.

Để đảm bảo các tiến trình có thể thực thi đồng thời và cộng tác với nhau, hệ thống phải có cơ chế cho phép các tiến trình trao đổi dữ liệu cũng như đồng bộ hóa hoạt động. Bài toán sản xuất – tiêu thụ là ví dụ điển hình của vấn đề hợp tác. Tiến trình sản xuất tạo ra, còn tiến trình tiêu thụ sử dụng thông tin. Ví dụ, chương trình in tạo ra các ký tự cho driver máy in sử dụng. Để tiến trình sản xuất và tiêu thụ thực thi đồng thời, hệ thống cần bộ đệm để tiến trình sản xuất đưa thông tin vào và tiến trình tiêu thụ lấy thông tin ra. Như vậy, tiến trình sản xuất và tiến trình tiêu thụ phải được đồng bộ hóa để tiến trình tiêu thụ không được sử dụng thông tin chưa tạo ra. Khi đó, tiến trình tiêu thụ phải chờ nếu bộ đệm rỗng.

```
Tiến trình SẢN XUẤT
Repeat
        ...
        Tạo một item, đặt vào nextp
        ...
        while counter=n do {};
        buffer[in] := nextp;
        in := in + 1 mod n;
        counter := counter + 1;
until false;
```

```
Tiến trình TIÊU THỤ
Repeat
        while counter=0 do ();
        Nextc := buffer[out];
        Out := out + 1 mod n;
        Counter := counter - 1;
        ...
        Tiêu thụ item trong nextc
        ...
until false;
```

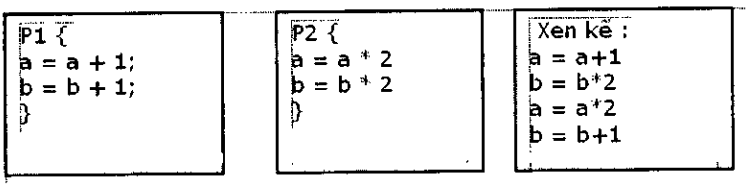
Hình 6.1. Hai tiến trình có quan hệ với nhau

Nếu độ lớn bộ đệm vô hạn, bộ phận sản xuất có thể đưa thông tin vào bất cứ lúc nào. Nhưng nếu kích thước bộ đệm hữu hạn, thì khi bộ đệm đầy thì phía sản xuất phải chờ.

Sau đây là ví dụ giải quyết bài toán bộ đệm hữu hạn bằng cách sử dụng bộ nhớ chung. Các tiến trình của cả hai phía sản xuất và tiêu thụ sử dụng chung các biến: **n, in, out, counter**; mảng **buffer** chứa các phần tử thuộc kiểu **item**. Giá trị khởi tạo của **in, out** là 0. Bộ đệm dùng chung được cài đặt thông qua mảng tuần hoàn với hai con trỏ: **in** và **out**. **in** trỏ vào vị trí trống tiếp theo trong bộ đệm, còn **out** trỏ vào vị trí đầu tiên trong bộ đệm có chứa thông tin. Bộ đệm rỗng khi **in = out**; và đầy khi **(in + 1) mod n = out**. Hình 6.1 minh họa cách cài đặt tiến trình sản xuất và tiêu thụ. Khối **While (điều kiện) do {}** chỉ làm nhiệm vụ kiểm tra điều kiện lặp cho đến khi điều kiện này nhận giá trị sai. Tiến trình sản xuất sử dụng biến cục bộ **nextp** để lưu thông tin mới được sinh ra và tiến trình tiêu thụ sử dụng biến cục bộ **nextc** chứa thông tin lấy ra.

Chúng ta xét tiếp ví dụ thứ 2, trong hệ thống có 2 biến **a** và **b** luôn được cập nhật thường xuyên, nhưng cần đảm bảo **a = b**. Giả sử tiến trình P_1 và P_2 thực hiện công việc trong Hình 6.2. Giả sử lúc đầu **a = b**, nhưng sau đó P_1 và P_2 chạy đồng thời. P_1 chạy chỉ thị thứ nhất, sau đó bị phong tỏa, P_2 được cấp phát CPU, P_2 chạy xong thì P_1 được khôi phục. Việc xen kẽ câu lệnh được minh họa trong Hình 6.2(c). Sau đó **a ≠ b** - hệ thống đã rơi vào trạng thái không nhất quán. Do đó, cần có cơ chế ngăn cản P_1 và P_2 chạy đồng thời.

cuu duong than cong . com



(a) (b) (c)

Hình 6.2. Các tiến trình phối hợp

6.1.2. Hợp tác qua truyền thông

cuu duong than cong . com

Trong hai ví dụ trên, mỗi tiến trình có ngữ cảnh hoạt động riêng, tương tác giữa chúng là gián tiếp. Hai tiến trình tuy chia sẻ biến dùng chung, nhưng không xác định nhau một cách tường minh (mặc dù vẫn phải đảm bảo tính toàn vẹn của dữ liệu). Khi hợp tác qua truyền thông, các tiến trình tham gia thực thi hướng tới một mục tiêu chung. Truyền thông giúp các tiến trình đồng bộ, kết hợp với nhau. Truyền thông nói chung được thực hiện qua

việc trao đổi thông điệp. HĐH hoặc thư viện phải cung cấp các hàm cơ sở để gửi và nhận thông điệp

6.2. ĐỘC QUYỀN TRUY XUẤT – GIẢI PHÁP PHẦN MỀM

6.2.1. Nhu cầu độc quyền truy xuất

Xét ví dụ bài toán sản xuất – tiêu thụ trong mục 6.1.1, **counter** sẽ tăng mỗi khi thêm **item** mới vào bộ đệm và giảm khi xóa một **item** ra khỏi bộ đệm. Khi thực thi riêng rẽ, hai thủ tục này chạy đúng nhưng vấn đề nảy sinh khi thực thi đồng thời. Giả sử hiện tại **counter** = 5, **procuder** và **consumer** thực thi đồng thời "**counter := counter + 1**" và "**counter := counter - 1**". Khi những câu lệnh này thực thi đồng thời, **counter** có thể nhận giá trị 4, 5 hoặc 7, mặc dù về mặt logic, kết quả chính xác phải là 5. Tại sao lại có điều này? Khi thi hành trên một dòng kiến trúc máy tính cụ thể, "**counter := counter + 1**" có thể chuyển sang ngôn ngữ máy như sau:

$register_1$:= counter;

$register_1$:= $register_1$ + 1;

counter := $register_1$;

cuu duong than cong . com

với **$register_1$** là thanh ghi nằm trong CPU. Tương tự "**counter := counter - 1**" được chuyển thành:

$register_2$:= counter;

$register_2$:= $register_2$ - 1;

counter := $register_2$;

$register_2$ cũng là thanh ghi nằm trong CPU. Thậm chí **$register_1$** và **$register_2$** có thể là cùng một thanh ghi vật lý. Sự thực thi đồng thời "**counter := counter + 1**" và "**counter := counter - 1**" ở mức cao tương đương sự thực thi chuỗi các chỉ thị máy ở mức thấp. Giả sử các chỉ thị này xen kẽ theo thứ tự sau:

cuu duong than cong . com

T_0:	producer	thực thi	$register_1$:= counter	{$register_1$ = 5}
T_1:	producer	thực thi	$register_1$:= $register_1$ + 1	{$register_1$ = 6}
T_2:	consumer	thực thi	$register_2$:= counter	{$register_2$ = 5}
T_3:	cosumer	thực thi	$register_2$:= $register_2$ - 1	{$register_2$ = 4}
T_4:	producer	thực thi	counter := $register_1$	{counter = 6}
T_5:	consumer	thực thi	counter := $register_2$	{counter = 4}

Kết quả **counter = 4**. Nếu đảo ngược thứ tự T_4 và T_5 thì **counter = 6**. Nguyên nhân là cả hai tiến trình đồng thời thao tác trên biến **counter**. Các tiến trình ở trạng thái *tranh đoạt điều khiển* (race condition) khi nhiều tiến trình cùng cập nhật vào biến dùng chung và kết quả việc thực thi phụ thuộc vào thứ tự thực hiện cụ thể của các tiến trình. Do vậy, phải đảm bảo tại thời điểm cụ thể chỉ có duy nhất một tiến trình được thay đổi biến dùng chung (các tiến trình phải đồng bộ với nhau).

6.2.2. Miền găng (Critical - Section)

Xét hệ thống gồm n tiến trình $\{P_0, P_1, \ldots, P_{n-1}\}$. Mỗi tiến trình có đoạn mã gọi là **miền găng** chứa các lệnh có thể thay đổi các biến dùng chung. Hệ thống phải đảm bảo tại bất kỳ thời điểm nào tối đa chỉ có một tiến trình được thi hành đoạn mã trong miền găng (gọi là bước vào miền găng). Khi đó biến dùng chung chỉ bị tác động bởi tối đa một tiến trình. Khi đó các tiến trình thay phiên nhau bước vào miền găng và hệ thống vẫn đảm bảo độc quyền truy xuất tài nguyên dùng chung. Vấn đề miền găng là thiết kế giao thức đồng bộ hóa các tiến trình. Nói chung, mỗi tiến trình phải xin phép bước vào miền găng, thực hiện cập nhật dữ liệu dùng chung rồi thông báo thoát khỏi miền găng.

Giải pháp cho miền găng phải thỏa mãn cả 3 yêu cầu sau:

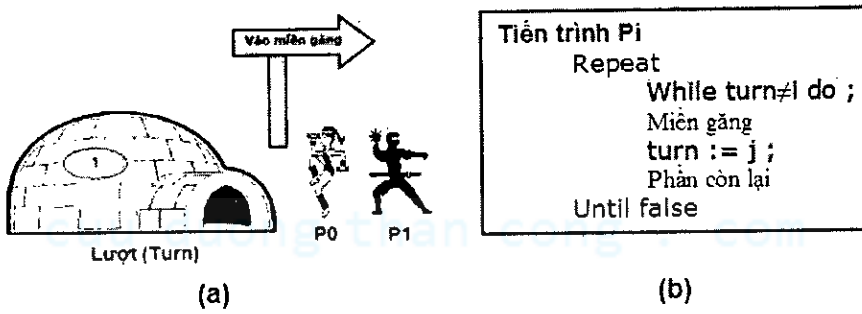
1. **Độc quyền truy xuất (Mutual Exclusion)**: Nếu tiến trình P_i đang trong miền găng thì không tiến trình nào được bước vào miền găng.
2. **Tiến triển (Progress)**: Nếu không có tiến trình nào ở trong miền găng và có một số tiến trình muốn vào miền găng thì một tiến trình nào đó phải được vào miền găng.
3. **Giới hạn đợi (bounded waiting)**: Thời gian từ khi tiến trình yêu cầu cho đến khi thực sự bước vào miền găng phải bị chặn bởi giới hạn nào đó.

Ở đây chúng ta giả định mỗi tiến trình đều hoạt động, tức là có tốc độ thực thi. Tuy nhiên, không so sánh tốc độ giữa các tiến trình. Các giải pháp đưa ra ở đây không phụ thuộc vào kiến trúc phần cứng máy tính hay số lượng CPU trong hệ thống. Tuy nhiên, giả sử các chỉ thị cơ bản trong ngôn ngữ máy (chẳng hạn **load**, **store** và **test**) được thực thi đơn nhất (không thể chia nhỏ việc thực thi hơn nữa). Nếu hai chỉ thị như vậy được thực thi đồng

thời, thì kết quả tương đương với việc thực thi tuần tự của chúng theo trật tự nào đấy. Khi trình bày, chúng ta chỉ định nghĩa các biến phục vụ mục đích đồng bộ, còn cấu trúc chung của tiến trình P_i bất kỳ là có dạng:

```
repeat
    entry section
    Vào miền găng
    exit section
    Phần còn lại
until false;
```

Phần **entry** và **exit** được bôi đen để nhấn mạnh tầm quan trọng của chúng.



Hình 6.3. Giải pháp thứ nhất

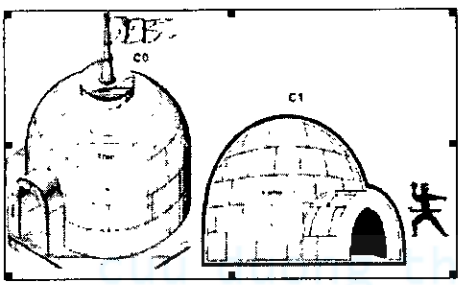
6.2.3. Giải pháp thứ nhất cho hai tiến trình

Chú ý rằng, bất kỳ giải pháp độc quyền truy xuất nào cũng đều dựa trên cơ chế độc quyền nào đó ở phần cứng. Ví dụ, tại một thời điểm, chỉ có thể xảy ra một truy cập bộ nhớ. Người ta sử dụng quy tắc "lều Eskimo" mang tính minh họa. Lều và cửa lều Eskimo rất bé, chỉ có thể cho phép đúng một người đi qua (Hình 6.3a). Bảng đen trong lều chỉ ghi được một số (0 hoặc 1). Giả sử hai người P_0 và P_1 muốn phối hợp với nhau vào miền găng. Người muốn vào miền găng đầu tiên phải vào lều để kiểm tra bảng đen. Nếu trên bảng đen ghi tên ứng với người đó (0 với P_0, 1 với P_1) thì được bước vào miền găng. Ngược lại, người đó phải đi ra khỏi lều và tiếp tục đợi. Khi rời miền găng, người vừa bước vào miền găng phải quay lại lều để ghi tên người kia trên bảng đen. Hình 6.3b là mã cho hai tiến trình P_0 và P_1. Để

thuận tiện, khi biểu diễn P_i ta sử dụng P_j để chỉ tiến trình còn lại, tức là **j = 1 - i**. Hai tiến trình cùng chia sẻ biến **turn** (khởi tạo bằng 0 hoặc 1). Biến **turn** giống bảng đen. Nếu **turn = i** thì P_i được phép bước vào miền găng. Giải pháp này bảo đảm tại một thời điểm chỉ có duy nhất một tiến trình ở trong miền găng. Tuy nhiên, yêu cầu tiến triển không được đảm bảo vì thuật toán này luôn đòi hỏi các tiến trình phải luân phiên bước vào miền găng. Chẳng hạn, khi **turn = 0**, P_0 biến mất thì mãi mãi P_1 không thể vào được miền găng.

6.2.4. Giải pháp thứ hai cho hai tiến trình

Vấn đề của giải pháp 1 là chỉ ghi thông tin về người muốn vào miền găng trên bảng đen, trong khi cần biết trạng thái của cả hai tiến trình. Để giải quyết, mỗi người phải có dấu hiệu xác định họ có vào miền găng hay không, khi đó dù người này biến mất thì người kia vẫn có thể vào miền găng mà không bị ảnh hưởng gì. Quy tắc được sửa đổi như sau: Mỗi người có lều riêng (C_0 và C_1) và có thể xem (nhưng không được quyền sửa) bảng đen trong lều của người kia. Người muốn vào miền găng sẽ định kỳ kiểm tra bảng đen trong lều người kia cho đến khi nhìn thấy có chữ **false** (dấu hiệu người kia không muốn vào). Lúc này người đó quay về lều của mình, viết **true** lên bảng đen (thông báo mình vào miền găng) và bước vào miền găng. Sau khi rời miền găng, người đó viết **false** lên bảng đen của mình.



(a)

```
Tiến trình Pi
        var flag: array[0..1] of boolean;
        Repeat
                flag[i] := true  ;
                while flag[j] do;
                Miền găng
                flag[i] := false;
                Phần còn lại
        Until false;
```

(b)

Hình 6.4. Thuật toán thứ hai

Thay thế **turn** bằng mảng **flag: array[0..1] of boolean** đóng vai trò lều riêng của mỗi người (C_0 và C_1). Các phần tử của mảng được khởi tạo là **false**. Giá trị **flag[i] = true** có ý nghĩa P_i sẵn sàng bước vào miền găng. Cấu trúc P_i minh họa trong Hình 6.4b. Đầu tiên P_i đặt **flag[i] = true** để thông báo

mình sẵn sàng bước vào miền găng. Sau đó P_i kiểm tra xem P_j đã bước vào miền găng chưa. Nếu P_j đã (hoặc chuẩn bị) vào, P_i sẽ đợi cho tới khi P_j bước ra khỏi miền găng (tức là **flag[j] = false**). Khi đó P_i mới được vào miền găng. Khi rời miền găng, P_i đặt lại **flag[i] = false** để cho phép P_j (nếu có nhu cầu) bước vào miền găng.

Thuật toán này không đảm bảo điều kiện tiến triển. Xét chuỗi thực thi dưới đây khi P_0 và P_1 đồng thời muốn vào miền găng:

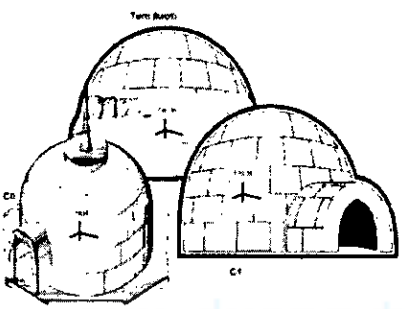
T_0: P_0 đặt flag[0] = true

T_1: P_1 đặt flag[1] = true

Bây giờ P_0 và P_1 sẽ lặp mãi trong vòng lặp **while**. Thuật toán phụ thuộc về thời gian thực hiện của hai tiến trình. Trình tự thực hiện trên có thể xuất hiện do trong hệ thống có nhiều CPU hoặc ngắt (chẳng hạn, do bộ định thời gây ra) xuất hiện ngay sau T_0 và quyền điều khiển CPU được chuyển sang tiến trình khác. Thay đổi thứ tự câu lệnh, đặt **flag[i] = true** và câu lệnh kiểm tra giá trị của **flag[j]** cũng không giải quyết được vấn đề này, mà còn có thể cho phép cả hai tiến trình cùng bước vào miền găng, vi phạm yêu cầu độc quyền truy xuất.

cuu duong than cong . com

6.2.5. Giải pháp thứ ba cho hai tiến trình



```
Tiến trình Pi
    var flag: array[0..1] of boolean;
    turn: 0..1;
    Repeat
        flag[i] := true;
        turn := j;
        while(flag[j] and turn=j) do;
        Miền găng
        flag[i] := false;
        Phần còn lại
    Until false;
```

(a) (b)

cuu duong than cong . com

Hình 6.5. Thuật toán 3

Kết hợp hai thuật toán trên, chúng ta đưa ra giải pháp hoàn chỉnh cho miền găng đáp ứng cả 3 yêu cầu. Bên cạnh lều riêng của mỗi người, còn có lều đóng vai trò "trọng tài", trong đó có bảng đen ghi người nào được quyền vào miền găng (Hình 6.5a). Các tiến trình chia sẻ mảng **flag** và biến **turn**.

Ban đầu, **flag[0] = flag[1] = false**, giá trị khởi tạo của **turn** (0 hay 1) không quan trọng. Cấu trúc P_i được minh họa trong Hình 6.5b. Để vào miền găng, P_i đặt **flag[i] = true** và sau đó kiểm tra xem P_j đã vào miền găng chưa. Nếu cả hai tiến trình cùng muốn vào miền găng, **turn** sẽ được gán giá trị **i** và **j** gần như đồng thời. Lệnh gán thực hiện sau sẽ ghi đè lên kết quả của lệnh gán trước. Giá trị sau cùng của **turn** quyết định tiến trình nào được phép vào miền găng.

Bây giờ chứng minh thuật toán này thỏa mãn cả ba yêu cầu đã nêu trong 6.2.2. Để chứng minh (1), ta thấy P_i chỉ vào miền găng khi **flag[j] = false** hoặc **turn = i**. Giả sử khi cả hai tiến trình cùng muốn vào miền găng một lúc thì **flag[0] = flag[1] = true**. Tuy nhiên, hai tiến trình này không thể bước vào miền găng cùng một lúc vì **turn** chỉ có thể nhận giá trị hoặc 0 hoặc 1. Như vậy, điều kiện độc quyền truy xuất được bảo đảm. Để chứng minh điều kiện (2) và (3), chú ý tiến trình P_i bị ngăn cản vào miền găng khi và chỉ khi nó bị tắc trong vòng lặp **while do** điều kiện **flag[j] = true** và **turn = j**; vòng lặp này chỉ được thực hiện đúng một lần. Nếu P_j không sẵn sàng vào miền găng thì **flag[j] = false** và P_i có thể vào miền găng. Nếu P_j đặt **flag[j] = true** và đang thực thi trong vòng lặp **while** thì **turn = i** hoặc bằng **j**. Nếu **turn = i**, P_i sẽ được vào miền găng. Nếu **turn = j**, P_j sẽ vào miền găng. Tuy nhiên, khi ra khỏi miền găng, P_j sẽ đặt lại **flag[j] = false** để cho phép P_i vào miền găng. Nếu đặt **flag[j] = true** thì P_j cũng sẽ đặt **turn = i**. Vì P_i không thay đổi giá trị biến **turn** trong vòng lặp **while**, P_i sẽ vào miền găng (thỏa mãn yêu cầu tiến triển) sau khi P_j vào nhiều nhất một lần (thỏa mãn yêu cầu giới hạn đợi).

6.2.6. Giải pháp đa tiến trình

Thuật toán 3 giải quyết vấn đề miền găng cho hai tiến trình. Bây giờ sẽ trình bày thuật toán *Hiệu bánh mỳ* cho n tiến trình. Thuật toán này có tên gọi như vậy, vì điều phối việc phục vụ tại những nơi đòi hỏi phải duy trì trật tự trong tình trạng lộn xộn (các cửa hàng).

Khi vào cửa hàng, khách hàng nhận được một số thứ tự (STT). Cửa hàng sẽ phục vụ khách hàng có STT nhỏ nhất. Tuy nhiên, thuật toán không đảm bảo hai tiến trình (hai khách hàng) có STT khác nhau. Khi đó tiến trình với tên xếp trước (theo thứ tự nào đó) được phục vụ trước. Vì vậy, nếu P_i và P_j có cùng STT và **i < j** thì P_i có độ ưu tiên cao hơn.

Cấu trúc dữ liệu chung là hai mảng **choosing: array[0 .. n - 1] of boolean** và **number: array[0 .. n - 1] of integer**. Ban đầu cấu trúc dữ liệu này được khởi tạo lần lượt là **false** và **0**. Để thuận tiện, chúng ta định nghĩa **(a, b) < (c, d)** nếu **a < c** hoặc nếu **a = c** và **b < d**. Cấu trúc P_i được minh họa trong Hình 6.6. Để chứng minh tính đúng đắn của thuật toán Hiệu bánh mỳ, đầu tiên ta cần chứng minh nếu P_i trong miền găng và P_k **(k ≠ i)** có STT **number[k] ≠ 0** thì **(number[i], i < number[k], k)**. Với kết quả này dễ dàng chứng minh được điều kiện độc quyền truy xuất. Thực vậy, coi P_i trong miền găng và P_k đang cố gắng vào miền găng. Khi thực thi câu lệnh **while** thứ hai trong trạng thái **j = i**, P_k thấy **number[i] ≠ 0** và **(number[i], i) < (number[k], k)**. Vòng lặp tiếp tục cho tới khi P_i ra khỏi miền găng. Để chứng minh yêu cầu tiến triển và giới hạn đợi, hãy để ý các tiến trình vào miền găng theo thứ tự đến trước phục vụ trước.

```
Repeat
      choosing[i]:=true;
      number[i] := max(number[0], number[1],...,number[n-1])  + 1;
      choosing[i]:= false;
      for j := 0 to n-1 do
            begin
                  while choosing[j] do;
                  while number[j] ≠ 0 and (number[j],j)< (number[i],i) do;
            end;
      Vào miền găng
      number[i]:= 0;
      Phần còn lại
Until false
```

Hình 6.6. Thuật toán vào miền găng cho nhiều tiến trình

6.3. ĐỒNG BỘ HÓA – GIẢI PHÁP PHẦN CỨNG

Phần cứng có thể giúp việc lập trình đơn giản hơn và nâng cao hiệu suất hệ thống. Phần này giới thiệu một vài giải pháp bằng phần cứng có khả năng giải quyết hiệu quả vấn đề miền găng.

6.3.1. Che ngắt

Trong hệ thống có một bộ vi xử lý, có thể giải quyết vấn đề miền găng bằng cách che ngắt (không cho ngắt xảy ra), trong khi chỉnh sửa biến dùng chung. Như vậy, hệ thống đảm bảo chuỗi chỉ thị thao tác trên biến dùng

chung được thực hiện trọn vẹn mà không bị gián đoạn bởi tiến trình khác. Do các tiến trình khác không thể xen vào, nên biến dùng chung không bị thay đổi một cách bất thường. Thuật toán thực hiện như minh họa trên Hình 6.7. Tuy nhiên, che ngắt trên hệ thống nhiều CPU tốn thời gian hơn vì phải gửi thông điệp đến tất cả các CPU. Tiến trình không thể vào miền găng ngay, hiệu suất hệ thống bị suy giảm.

```
Tiến trình
    Repeat
        Che ngắt (không cho ngắt xảy ra)
        Miền găng
        Cho phép ngắt xảy ra
        Phần còn lại
    Until false;
```

Hình 6.7. Miền găng bằng cách che ngắt

6.3.2. Các chỉ thị đặc biệt

Nhiều kiến trúc máy tính có chỉ thị phần cứng đặc biệt cho phép kiểm tra và chỉnh sửa nội dung một từ hoặc tráo đổi nội dung hai từ trong bộ nhớ một cách đơn nhất. Chúng có thể sử dụng giải quyết vấn đề miền găng.

```
function Test-and_Set(var target:boolean): boolean;
    begin
        Test-and-Set:= target;
        targer:= true;
    end;
```

Chỉ thị Test-and-Set

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

Chỉ thị Swap

Hình 6.8. Các chỉ thị đặc biệt

Đặc điểm quan trọng của chỉ thị kiểu này là sự thực thi một cách nguyên vẹn và đơn nhất, nghĩa là quá trình thực hiện chỉ thị không bị gián đoạn. Vì vậy, nếu hai chỉ thị **Test-and-Set** được thực thi đồng thời (trên các CPU khác nhau), chúng sẽ thực thi tuần tự theo thứ tự nào đó.

Với chỉ thị **Test-and-Set**, có thể thi hành độc quyền truy xuất bằng việc khai báo biến **boolean lock** khởi tạo giá trị **false**. Cấu trúc P_i được minh họa trong Hình 6.8. Chỉ thị **swap** tráo đổi nội dung 2 từ được định nghĩa trong Hình 6.8. Giống **Test-and-Set**, **swap** được thực thi một cách đơn nhất. Hình 6.9 minh họa cách thực hiện độc quyền truy xuất. Ưu điểm ở đây là khả

năng áp dụng đơn giản trên hệ thống có một hoặc có nhiều CPU. Tuy nhiên, hiện tượng busy-waiting, khả năng chết đói và bế tắc vẫn còn có thể xảy ra.

```
Repeat
        while Test-and-Set(lock) do;
        Miền găng
        Lock := false;
        Phần còn lại
Until false;
```

(a) Độc quyền truy xuất bằng Test-and-Set

```
Repeat
        key:= true;
        repeat
                Swap(lock, key);
        until key=false;
        Miền găng
        Lock := false;
        Phần còn lại
Until false;
```

(b) Độc quyền truy xuất bằng Swap

Hình 6.9. Độc quyền truy xuất bằng các chỉ thị đặc biệt

6.4. GIẢI PHÁP ĐỒNG BỘ CƠ BẢN

6.4.1. Khóa (Lock)

Với ví dụ về "lều Eskimo", chúng ta giả sử ở cửa lều có một khóa. Người muốn vào miền găng sẽ đến trước cửa lều. Nếu lều bị khóa, người đó phải đợi. Nếu lều chưa khóa, người đó khóa lều lại, đi vào miền găng (cầm theo chìa khóa). Sau khi thoát khỏi miền găng, người đó quay trở về mở khóa lều cho những người muốn vào miền găng. Hình 6.10a minh họa cách thức sử dụng khóa khi muốn vào miền găng. Tuy nhiên, cần giải quyết vấn đề hai người đồng thời muốn dùng khóa. Người ta có thể sử dụng giải pháp che ngắt khi muốn đóng và mở khóa (minh họa trên Hình 6.10b và c).

```
Tiến trình Pi
        var lock boolean;
        Repeat
                while (lock) ;
                lock = true;
                Miền găng
                lock = false;
                Phần còn lại
        Until false;
```

(a) Cách sử dụng khóa

```
enter (lock) {
        disableInterrupts();
        while (lock) {
        enableInterrupts();
        disableInterrupts();
        }
        lock = true;
        enableInterrupts();
}
```

(b) Đóng khóa

```
exit (lock) {
        disableInterrupts();
        lock = true;
        enableInterrupts();
}
```

(c) Mở khóa

Hình 6.10. Khóa và cách cài đặt khóa

Nhược điểm chính của các giải pháp trên là tình trạng chờ bận (*Busy waiting*). Khi có tiến trình ở trong miền găng, bất kỳ tiến trình khác muốn vào miền găng sẽ thực hiện vòng lặp (trong đoạn mã xin vào miền găng) để

kiểm tra xem đã đến lượt mình vào chưa. Tình trạng *Busy waititng* lãng phí các chu kỳ CPU mà lẽ ra các tiến trình khác có thể sử dụng để thực hiện công việc hữu ích. Tình trạng này còn được gọi là khóa xoay (*Spinlock*) (vì các tiến trình quay quanh vòng lặp trong khi chờ khóa). Ưu điểm của *spinlock* trong hệ thống đa bộ xử lý là không cần phải chuyển ngữ cảnh khi một tiến trình đợi khóa (vì chuyển ngữ cảnh có thể chiếm một lượng thời gian đáng kể).

6.4.2. Semaphore

Semaphore S là biến nguyên mà sau khi khởi tạo chỉ được truy cập qua hai thao tác đơn nhất là **wait** và **signal**. Hình 6.11a là định nghĩa của *wait* và *signal*. Toán tử **wait** và **signal** thay đổi giá trị semaphore được thực hiện một cách đơn nhất. Tức là khi một tiến trình đang thay đổi giá trị semaphore thì không tiến trình nào được quyền thay đổi giá trị semaphore. Ngoài ra, trong trường hợp **wait(S)**, việc kiểm tra giá trị **S (S ≤ 0)** và việc thay đổi **S := S - 1** (nếu có) phải được thực thi liên tục không gián đoạn. Cách cài đặt các toán tử này được trình bày trong mục 6.4.3. Nhưng trước tiên ta trình bày cách sử dụng semaphore.

```
wait(S):
        while S ≤ 0 do;
        S := S-1;
signal(S) :
        S := S+1;
```

(a) Thao tác Wait và Signal

```
Repeat
    Wait(mutex);
    Miền găng
    Signal (mutex);
    Phần còn lại
Until false;
```

(b) Độc quyền truy xuất bằng Semaphore

Hình 6.11. Định nghĩa và cách sử dụng Semaphore

Có thể sử dụng semaphore để giải quyết vấn đề miền găng cho n tiến trình. n tiến trình này chia sẻ semaphore **mutex** (**mut**ual **ex**clusion) khởi tạo giá trị 1. Tiến trình P_i minh họa trong Hình 6.11b. "Lều Eskimo" trong ví dụ trước được cải tiến như sau: Bảng đen sử dụng để ghi số lượng người trong lều (lều có thể bố trí nhiều ghế ngồi). Người muốn vào miền găng (giả sử A) bước vào lều, tăng giá trị ghi trong bảng đen lên 1. Nếu giá trị này là 1, A có quyền bước vào miền găng. Nếu giá trị này khác 1, A phải ngồi đợi trong lều. Sau khi thoát khỏi miền găng, A phải quay trở về lều, giảm giá trị ghi trên bảng đen đi 1 và nếu giá trị này khác 0, thì người khác đang đợi trong lều được vào miền găng.

Ngoài ra, cũng có thể sử dụng semaphore để giải quyết nhiều vấn đề đồng bộ hóa khác. Giả sử hai tiến trình chạy đồng thời: P_1 với lệnh S_1 và P_2 với lệnh S_2. Giả thiết rằng S_2 chỉ được thực thi sau khi S_1 đã thực thi xong. Có thể giải quyết vấn đề này bằng cách cho P_1 và P_2 dùng chung **semaphore synch** khởi tạo bằng 0 và chèn vào câu lệnh: **{S1;signal(synch);}** cho P_1 và câu lệnh **{wait(synch);S2;}** cho P_2. Vì **synch** khởi tạo bằng 0, P_2 sẽ thực thi S_2 chỉ sau khi P_1 thực hiện **signal(synch)** sau S_1.

6.4.3. Cài đặt semaphore

Với định nghĩa trên, semaphore vẫn bị tình trạng *busy waiting*. Để khắc phục cần định nghĩa lại toán tử **wait** và **signal** của semaphore. Tiến trình gọi **wait** và thấy giá trị semaphore không dương sẽ phải chờ. Tuy nhiên, thay vì thực hiện vòng lặp, tiến trình phong tỏa chính nó. Toán tử **block** đặt tiến trình vào hàng đợi (hàng đợi này gắn với semaphore) và tiến trình chuyển sang trạng thái đợi (waiting). Sau đó điều khiển được chuyển cho bộ điều phối CPU để lựa chọn tiến trình khác thực thi. Tiến trình bị phong tỏa chờ semaphore S sẽ được khởi động lại (bằng toán tử **wakeup**) khi tiến trình nào đấy thi hành toán tử **signal**. Khi đó trạng thái tiến trình này chuyển từ đợi (waiting) sang sẵn sàng (ready).

```
type semaphore = record
        value:integer;
        L: Danh sách tiến trình;
end;                                    (a)
```

```
signal(S):
        S.value:= S.value + 1;
        if S.value ≤ 0 then begin
                Lấy một tiến trình P từ S.L;
                wakeup(P);
        end;
                                        (b)
```

```
wait(S):
        S.value := S.value - 1;
        if S.value < 0
        then begin
                Đặt tiến trình này S.L;
                Phong tỏa tiến trình;
        end;
                                        (c)
```

Hình 6.12. Cài đặt semaphore

Semaphore có thể cài đặt dưới dạng bản ghi (Hình 6.12a), với hai thành phần: một giá trị nguyên và một danh sách các tiến trình đợi. Toán tử **signal** lấy một tiến trình trong danh sách đợi để kích hoạt. Hình 6.12b và c minh họa toán tử **signal** và **wait**. Khi gọi **block**, tiến trình sẽ tự treo (chuyển sang trạng thái chờ) và chuyển sang nằm trong hàng đợi. Toán tử **wakeup(P)** khôi

phục tiến trình P bị phong tỏa. Hai toán tử này được HĐH cài đặt dưới dạng các lời gọi hệ thống cơ bản.

Theo định nghĩa cũ, semaphore thuộc kiểu *busy waiting* và giá trị semaphore không âm. Semaphore theo kiểu mới có thể nhận giá trị âm, khi đó độ âm xác định số lượng các tiến trình đợi trên semaphore. Thực tế này là do trong toán tử **wait**, chúng ta đảo lệnh trừ lên trước lệnh kiểm tra. Dễ dàng cài đặt danh sách các tiến trình đợi thông qua trường liên kết trong khối điều khiển tiến trình (PCB). Mỗi semaphore chứa một giá trị nguyên và một con trỏ trỏ tới danh sách PCB. Để thêm và xóa các tiến trình khỏi danh sách và để bảo đảm điều kiện giới hạn đợi, hệ thống có thể sử dụng hàng đợi FIFO (first-in, first-out). Khi đó, semaphore trỏ tới cả hai đầu của hàng đợi. Chú ý là cách sử dụng semaphore chính xác không phụ thuộc vào cách cài đặt danh sách đợi trong semaphore.

Toán tử semaphore phải được thực hiện một cách toàn vẹn và đơn nhất, nghĩa là hai tiến trình không thể cùng lúc lại có thể thực thi toán tử **wait** và **signal** trên cùng một semaphore. Đây là vấn đề miền găng và có thể giải quyết theo hai cách sau: Trong môi trường có một CPU, có thể che ngắt khi thực thi **wait** hoặc **signal**. Khi đó chỉ thị của các tiến trình khác không thể xen kẽ vào chỉ thị của toán tử. Nhưng trong môi trường nhiều CPU, khó thực hiện việc chặn ngắt. Chỉ thị của các tiến trình khác nhau (chạy trên các CPU khác nhau) có thể xen kẽ với nhau tùy ý. Nếu phần cứng không có chỉ thị đặc biệt thì có thể sử dụng giải pháp phần mềm cho vấn đề miền găng, khi đó tình trạng busy waiting vẫn xuất hiện với thủ tục **wait** và **signal**. Chúng ta chỉ loại bỏ tình trạng busy waiting khi chương trình ứng dụng bước vào miền găng. Tình trạng busy waiting bị giới hạn trong miền găng của toán tử **wait** và **signal**. Do độ lớn của miền này khá nhỏ (nếu được tối ưu sẽ không vượt quá 10 chỉ thị), nên nếu tình trạng busy waiting có xuất hiện thì cũng chỉ trong thời gian rất ngắn.

6.4.4. Bế tắc và Chết đói

Cài đặt semaphore như trên có thể dẫn đến trường hợp nhiều tiến trình đợi vô vọng một sự kiện do một tiến trình khác cũng đang trong trạng thái đợi gây ra. Sự kiện đợi ứng với việc thực thi toán tử **wait**. Khi hệ thống rơi vào tình huống này, các tiến trình liên quan sẽ ở trong tình trạng bế tắc. Ví

dụ, hệ thống có hai tiến trình P_0 và P_1, mỗi tiến trình truy cập vào hai semaphore S và Q để đặt giá trị 1:

P_0	P_1
wait(S);	**wait(Q);**
wait(Q);	**wait(S);**
...	...
signal(S);	**signal(Q);**
signal(Q);	**signal(S);**

Giả sử P_0 thực thi **wait(S)**, sau đó P_1 thực thi **wait(Q)**. Khi thực thi **wait(Q)**, P_0 phải đợi cho đến khi P_1 thực thi xong **signal(Q)**. Tương tự, khi thực thi **wait(S)**, P_1 phải đợi cho đến khi P_0 thực thi xong **signal(S)**. Rõ ràng hai toán tử **signal** này không thể được thực thi, P_0 và P_1 rơi vào bế tắc. Ta nói một tập hợp các tiến trình trong tình trạng bế tắc nếu mỗi tiến trình trong tập đợi một sự kiện do một tiến trình khác trong tập gây ra. Phần lớn các sự kiện trình bày ở đây liên quan đến cấp phát tài nguyên. Tuy nhiên, có nhiều loại sự kiện khác cũng có thể gây ra bế tắc và sẽ được trình bày trong chương sau. Vấn đề khác liên quan đến tình trạng bế tắc là trạng thái bị phong tỏa vĩnh viễn hay chết đói (starvation), là hiện tượng khi các tiến trình chờ vô định trong semaphore. Phong tỏa vĩnh viễn có thể xảy ra nếu ta thêm và xóa các tiến trình khỏi danh sách gắn với semaphore theo thứ tự vào sau ra trước (LIFO).

6.4.5. Semaphore nhị phân

Cấu trúc semaphore nói trên gọi là *counting semaphore*, vì giá trị nguyên có thể nhận giá trị tùy ý. Semaphore nhị phân là semaphore mà giá trị nguyên chỉ là 0 hoặc 1. Cài đặt semaphore nhị phân đơn giản hơn counting semaphore do tận dụng được kiến trúc phần cứng. Bây giờ sẽ trình bày việc cài đặt counting semaphore dựa trên semaphore nhị phân. Để cài đặt counting semaphore S bằng semaphore nhị phân, ta sử dụng các cấu trúc dữ liệu sau: **S1: binary-semaphore**; **S2: binary-semaphore**; **C: integer**. Khởi đầu **S1 = 1**; **S2 = 0** và **C** nhận giá trị ban đầu của counting semaphore S. Toán tử **wait** và **signal** trong counting semaphore được cài đặt trong Hình 6.13.

```
signal():
        wait(S1);
        C := C + 1;
                if (C <= 0) then signal (S2);
        else signal(S1);
```

```
wait():
        wait(s1);
        C := C - 1;
        if C < 0 then begin
                signal(S1);
                wait(S2);
        end
        signal(S1);
```

Hình 6.13. Cài đặt counting semaphore bằng semaphore nhị phân

6.5. NHỮNG VẤN ĐỀ ĐỒNG BỘ KINH ĐIỂN

Trong phần này, trình bày một số vấn đề đồng bộ liên quan đến điều khiển song song. Những vấn đề này thường được sử dụng như bài toán mẫu để kiểm tra các phương pháp đồng bộ hóa mới.

6.5.1. Vấn đề bộ đệm giới hạn (The Bounded-Buffer Problem)

cuu duong than cong . com

Ở đây trình bày giải pháp tổng quát cho vấn đề Bộ đệm giới hạn (xem mục 6.1). Giả sử vùng đệm dùng chung có khả năng lưu giữ n **item**. Semaphore **Mutex** đảm bảo độc quyền truy xuất với vùng đệm và được khởi tạo giá trị 1. Semaphore **empty** và **full** đếm số bộ đệm trong vùng chưa sử dụng và số bộ đệm đã được sử dụng. Semaphore **empty** được khởi tạo giá trị **n**, semaphore **full** được khởi tạo giá trị **0**. Mã cho tiến trình producer và consumer được minh họa trong Hình 6.14.

6.5.2. Vấn đề đọc – ghi (The Readers and Writers)

cuu duong than cong . com

Nhiều tiến trình hoạt động đồng thời có thể dùng chung đối tượng dữ liệu (file hay bản ghi). Đối tượng có thể truy xuất theo chế độ chỉ đọc (reader) hoặc chế độ cập nhật (writer) vào đối tượng dùng chung. Hiển nhiên, sẽ không có chuyện gì xảy ra nếu hai tiến trình reader cùng đọc. Tuy nhiên, cần phải đồng bộ khi một tiến trình writer và một số tiến trình khác (writer hoặc reader) cùng truy cập đồng thời đến đối tượng dùng chung. Giải pháp ở đây là tiến trình writer phải được độc quyền truy cập tới đối tượng dùng chung. Đây là vấn đề đồng bộ đọc – ghi và thường được sử dụng để

kiểm tra các công cụ đồng bộ mới. Vấn đề đọc – ghi có một số biến thể liên quan đến quyền ưu tiên. Dị bản đơn giản nhất gọi là vấn đề đọc – ghi đầu tiên, yêu cầu không tiến trình reader nào phải đợi trừ khi có tiến trình writer đã được phép sử dụng đối tượng dùng chung. Nói cách khác, không tiến trình reader nào phải đợi tiến trình reader khác hoàn thành chỉ vì có một tiến trình writer đang đợi.

```
Tiến trình Sản xuất
    Repeat
        ...
        Tạo ra item và đặt vào nextp
        ...
        wait (empty);
        wait (mutex);
        ...
        Đưa nextp vào bộ đệm
        ...
        signal (mutex);
        signal (full);
    until false;
```

```
Tiến trình Tiêu thụ
    Repeat
        wait (full);
        wait (mutex);
        ...
        Chuyển một item vào nextc
        ...
        signal (mutex);
        signal (empty);
        ...
        Tiêu thụ item trong nextc
        ...
    until false;
```

Hình 6.14. Bài toán sản xuất – tiêu thụ

```
Tiến trình GHI
    Repeat
        Wait (wrt);
        ...
        Thực hiện việc Ghi
        ...
        signal (wrt );
    until false;
```

```
Tiến trình ĐỌC
    Repeat
        Wait(mutex);
        readcount:= readcount + 1;
        if readcount=1 then wait(wrt);
        signal(mutex);
        ...
        Thực hiện việc Đọc
        ...
        wait(mutex);
        readcount := readcount -1 ;
        if readcount=0 then signal(wrt);
        signal(mutex);
    until false;
```

Hình 6.15. Reader và Writer

cuu duong than cong . com

Trong dị bản thứ hai, tiến trình writer sẵn sàng sẽ được ghi ngay khi có thể. Tức là nếu tiến trình writer đang đợi để truy cập vào đối tượng chia sẻ, không tiến trình reader mới nào được đọc. Chú ý rằng, hai phiên bản trên có thể dẫn đến tình trạng "chết đói": trường hợp đầu tiên là writer, trường hợp thứ hai là reader. Vì lý do này, nhiều biến thể khác của vấn đề đã được đưa ra. Trong phần này, chúng ta trình bày giải pháp cho vấn đề đọc – ghi thứ nhất. Sử dụng cấu trúc dữ liệu dùng chung sau đây: hai semaphore **mutex**,

wrt và số nguyên **readcount**. Giá trị khởi tạo của semaphore **mutex** và **wrt** là 1, của **readcount** là 0. Semaphore **wrt** được dùng chung giữa hai tiến trình writer và reader, semaphore **mutex** được dùng để đảm bảo độc quyền truy xuất với **readcount. Readcount** đếm số tiến trình đang đọc đối tượng. Semaphore **wrt** thực hiện độc quyền truy xuất giữa các writer. Ngoài ra **wrt** còn được dùng bởi tiến trình reader đầu tiên hoặc cuối cùng khi bước vào hay bước ra miền găng. Hai tiến trình này được minh họa trên Hình 6.15. Chú ý, khi một tiến trình writer đang ở trong miền găng thì với **n** tiến trình reader đợi, có một tiến trình reader xếp hàng trong **wrt** và **n – 1** tiến trình còn lại xếp hàng trong **mutex**. Khi tiến trình writer thi hành **signal(wrt)**, hệ thống có thể khôi phục lại hoặc một tiến trình reader, hoặc một tiến trình writer đang chờ. Lựa chọn như thế nào phụ thuộc vào bộ điều phối.

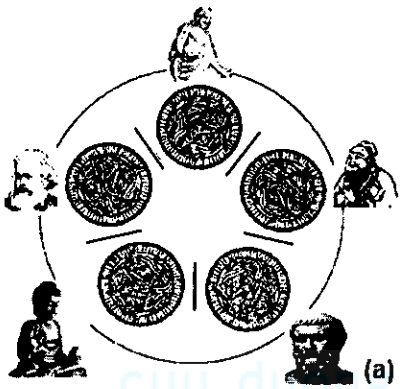
6.5.3. "Bữa ăn tối của các triết gia"

Giả sử có 5 triết gia chỉ suy nghĩ và ăn. Các triết gia ngồi quanh bàn tròn, trên bàn có một nồi cơm và 5 chiếc đũa (Hình 6.16a). Các triết gia không trao đổi với nhau. Khi đói triết gia cố gắng nhặt hai chiếc đũa gần mình nhất. Trong một lần lấy, triết gia chỉ có thể nhặt lên được một chiếc đũa. Triết gia không thể "cướp" đũa ở trong tay người khác. Khi có cả đôi đũa, triết gia sẽ ăn và buông đôi đũa của mình sau khi ăn xong. Sau đó triết gia lại tiếp tục suy nghĩ. Bữa ăn của các triết gia được xem là vấn đề đồng bộ hóa kinh điển, không phải chỉ vì sự quan trọng trong thực tế mà còn là ví dụ của một lớp lớn các vấn đề kiểm soát đồng bộ. Chẳng hạn, việc cấp phát tài nguyên cho các tiến trình sao cho hệ thống không rơi vào trạng thái bế tắc hay chết đói.

Một giải pháp đơn giản là xem chiếc đũa là semaphore. Hành động Nhặt đũa thực hiện qua toán tử **wait** và hành động Buông đũa thực hiện qua toán tử **signal**. Như vậy, dữ liệu được chia sẻ là: **chopstick: array [0..4] of semaphore**; ở đây tất cả các phần tử trong mảng **chopstick** được khởi tạo là 1. Cấu trúc triết gia thứ i được minh họa trên Hình 6.16b. Mặc dù đảm bảo hai triết gia ngồi cạnh nhau không thể ăn cùng lúc, giải pháp này không được sử dụng vì có thể gây bế tắc. Giả sử 5 triết gia đói cùng lúc, mỗi người lấy chiếc đũa bên trái. Khi đó tất cả phần tử của mảng **chopstick** nhận giá trị 0. Khi cố lấy chiếc đũa bên phải, tất cả các triết gia sẽ bị đợi mãi mãi. Chúng ta có thể cải tiến để giảm bớt tình trạng bế tắc, chẳng hạn:

- Chỉ cho phép triết gia nhặt lên một chiếc đũa khi đôi đũa chưa được sử dụng.
- Sử dụng giải pháp phi đối xứng. Triết gia mang số lẻ nhặt chiếc đũa trái trước chiếc đũa phải, còn triết gia mang số chẵn nhặt chiếc đũa phải trước chiếc đũa trái.

Bất cứ giải pháp nào cho vấn đề Bữa ăn của các triết gia phải ngăn ngừa trường hợp một triết gia nào đó bị chết đói. Giải quyết bế tắc chưa chắc loại trừ được tình trạng chết đói.




```
Thuật toán của các Triết gia:
    Repeat
        wait (chopstick[i] );
        wait (chopsick[i+1 mod 5]);
        ...
        Ăn
        ...
        signal (chopstick[i]);
        signal (chopstick[i+1 mod 5]);
        ...
        Suy nghĩ
        ...
    Until false;                    (b)
```

Hình 6.16. Bữa ăn tối của triết gia

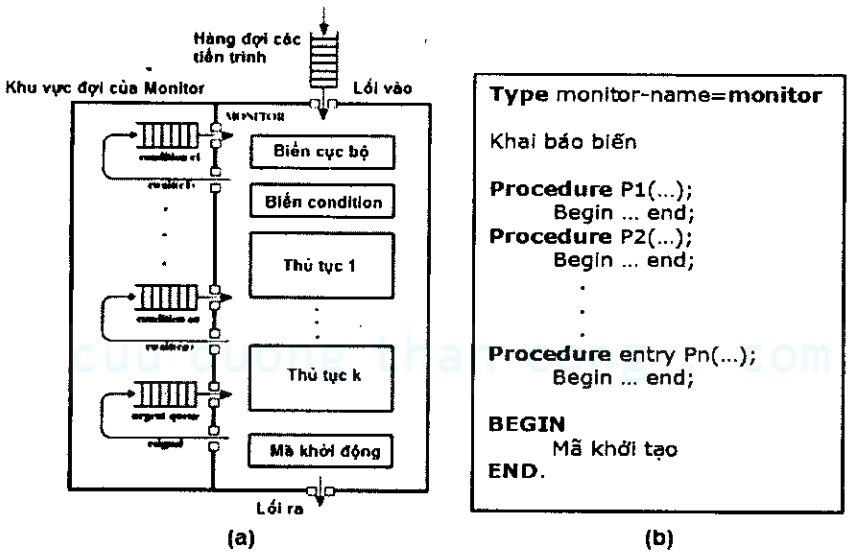
6.6. CÁC GIẢI PHÁP ĐỒNG BỘ CAO CẤP

6.6.1. Monitor và biến điều kiện Condition

Monitor - một cấu trúc đồng bộ mức cao là tập hợp các toán tử được người lập trình định nghĩa trước. Khai báo kiểu monitor gồm hai phần: phần khai báo biến cục bộ và phần thân là các thủ tục hay hàm thao tác trên kiểu đó. Cú pháp của Monitor minh họa trên Hình 6.17b. Chú ý rằng, các thủ tục/hàm định nghĩa bên trong monitor chỉ được truy cập tới những biến được khai báo cục bộ trong monitor và các tham số hình thức. Tương tự biến cục bộ trong monitor chỉ được truy cập qua những thủ tục cục bộ.

Cấu trúc monitor đảm bảo tại một thời điểm chỉ có đúng một tiến trình được thực hiện bên trong monitor. Do vậy, người lập trình không cần viết mã cho sự đồng bộ này một cách tường minh. Tuy nhiên, cấu trúc monitor như đã được định nghĩa chưa đủ mạnh để mô hình hóa một số lược đồ đồng

bộ. Vì vậy, bổ sung thêm vào monitor một cơ chế đồng bộ khác là cấu trúc **condition**. Lập trình viên cần tự viết lược đồ đồng bộ riêng của mình để có thể định nghĩa một hoặc nhiều biến thuộc kiểu **condition**. Chỉ toán tử **wait** và **signal** mới có thể được gọi trên biến kiểu **condition**. Tiến trình gọi **wait** sẽ bị treo cho đến khi tiến trình khác gọi **signal** trên biến điều kiện tương ứng. Toán tử **x.signal** khôi phục đúng một tiến trình bị treo. Nếu không có tiến trình bị treo, toán tử **signal** không có tác dụng, nghĩa là trạng thái của **x** sẽ giữ nguyên như thể **signal** chưa thi hành (Hình 6.17a). Điều này khác với toán tử **signal** của semaphore luôn luôn biến đổi trạng thái của semaphore.



Hình 6.17. Monitor

Giả sử tiến trình P gọi toán tử **x.signal()** khi tiến trình Q treo trong **x** (kiểu condition). Rõ ràng nếu Q được khôi phục lại để tiếp tục thi hành, thì sau **x.signal()**, P phải đợi vì nếu không cả P và Q sẽ cùng hoạt động đồng thời trong monitor. Chú ý cả hai tiến trình về mặt khái niệm có thể tiếp tục thực thi công việc bình thường của mình. Có thể có hai khả năng sau:

(1) P đợi đến khi Q rời khỏi monitor, hoặc đợi điều kiện khác.

(2) Q đợi đến khi P rời khỏi monitor, hoặc đợi điều kiện khác.

Có những lý lẽ hợp lý trong việc lựa chọn (1) hoặc (2). Khi P đang thi hành trong monitor, lựa chọn (2) hợp lý hơn. Tuy nhiên, nếu chúng ta cho phép P tiếp tục, đến thời điểm Q được khôi phục thì điều kiện logic mà Q đang đợi có thể không còn đúng nữa.

6.6.2. Bài toán "Bữa ăn tối của các triết gia"

Chúng ta minh họa những khái niệm này bằng cách trình bày giải pháp không có bế tắc cho bài toán Bữa ăn tối của các triết gia. Chú ý rằng, triết gia chỉ được phép nhặt đũa lên nếu chúng chưa được nhặt. Cần phân biệt sự khác nhau trong ba trạng thái của triết gia, ta có cấu trúc dữ liệu **state:array[0...4] of (thinking, hungry, eating)**. Triết gia thứ i có thể đặt giá trị **state[i] = eating** chỉ khi hai người ngồi cạnh không ăn **(state[i + 4 mod 5] # eating and state[i + 1 mod 5] # eating)**, ta khai báo **self: array[0...4] of condition**. Triết gia thứ i có thể phải chờ ngay cả khi đói vì chưa có đủ đũa. Việc cấp phát đũa được kiểm soát bởi monitor **dp** – là một thể hiện của kiểu monitor **dining-philosopher** được minh họa trong Hình 6.18. Triết gia trước khi ăn phải gọi **pickup**. Khi đó, tiến trình "triết gia" này bị treo tạm thời. Sau khi ăn xong, triết gia gọi toán tử **putdown** và tiếp tục suy nghĩ. Triết gia thứ i sử dụng các toán tử **putdown** và **pickup** theo đúng trình tự: **dp.pickup(i)** ⇒ **Ăn** ⇒ **dp.putdown(i)**. Dễ dàng chứng minh giải pháp này đảm bảo không có hai người ngồi cạnh nhau đồng thời ăn và không xảy ra tình trạng bế tắc. Tuy nhiên, chú ý rằng một triết gia có thể bị chết đói.

cuu duong than cong . com

```
Type dining-philosoper = monitor
      var state   : array[0...4] of (thinking, hungry, eating );
      var self    : array[0...4] of condition;
.....................................................................
      procedure entry pickup (i:0... 4) {
            state[i]:= hungry;
            test(i);
            if state[i] # eating then self[i].wait;
      }
      procedure entry putdown (i:0... 4) {
            state[i]:= thinking;
            test (i+4 mod 5);                  Các hàm cục bộ
            test (i+1 mod 5);
      }
      procedure test (k:0...4) {
            if (state[k+4 mod 5] # eating)      and
            (state[k]=hungry)                   and
            (state[k+1 mod 5]# eating) {
                  state[k]:= eating;
                  self[k].signal;
            };
      }
.....................................................................
BEGIN
      for i:= 0 to 4 state[i]:= thinking;       Mã khởi tạo
END
```

cuu duong than cong . com

Hình 6.18. Monitor giải quyết bài toán Bữa ăn tối của các triết gia

6.6.3. Cài đặt monitor bằng semaphore

```
Hàm F
  wait(mutex);
  Thân hàm F
  if next-count > 0
      signal (next)
  else signal (mutex);
```

a)

```
x.wait() {
  x-count:= x-count + 1;
  if next-count > 0
      signal (next)
  else
      signal (mutex);
  wait (x-sem);
  x-count:= x-count - 1;
}
```

b)

```
x.signal() {
    if x-count > 0 {
    next-count := next-
    count + 1;
    signal (x-sem);
    wait (next );
    next-count := next-
    count -1;
    }
}
```

c)

Hình 6.19. Cài đặt monitor và condition

Mỗi monitor gắn với một semaphore **mutex** (được khởi tạo giá trị 1). Tiến trình phải thực thi **wait(mutex)** trước khi bước vào monitor và thực thi **signal(mutex)** sau khi rời khỏi monitor. Tiến trình gọi **wait** phải đợi đến khi tiến trình đang chiếm giữ semaphore hoặc rời khỏi semaphore, hoặc bị dừng lại tạm thời, nên hệ thống cài đặt thêm semaphore **next** được khởi tạo là 0. Tiến trình đang ở trong semaphore **mutex** có thể bị phong tỏa tại **next**. Biến nguyên **next-count** đếm số tiến trình bị phong tỏa tại **next**. Vì vậy, thủ tục F cải tiến được minh họa trong Hình 6.19a. Như vậy, đảm bảo được độc quyền truy xuất trong monitor.

Với **x** là kiểu condition, ta có semaphore **x-sem** và biến nguyên **x-count**, cả hai được khởi tạo là 0. Toán tử **x.wait** và **x.signal** được minh họa trên Hình 6.19b và c.

Chuyển sang vấn đề thứ tự phục hồi các tiến trình bên trong monitor, nếu có một số tiến trình bị treo trong biến **x** kiểu condition và một tiến trình nào đó thi hành toán tử **x.signal()**, khi đó phải xác định tiến trình bị treo nào sẽ được khôi phục? Một cách giải quyết đơn giản là thuật toán FCFS: tiến trình đợi lâu nhất sẽ được khôi phục trước. Tuy nhiên, thuật toán đơn giản này không phù hợp trong nhiều tình huống. Vì vậy, có thể người ta sử dụng cấu trúc conditional-wait có dạng: **x.wait(c);** trong đó **c** là một biểu thức kiểu nguyên được tính khi thi hành toán tử **wait**. Giá trị của **c** là giá trị ưu tiên được lưu giữ cùng với tên của tiến trình bị treo. Khi toán tử **x.signal()** thi hành, tiến trình nào có giá trị ưu tiên nhỏ nhất sẽ được khôi phục.

```
R.acquire(t);

        Truy cập và Sử
        dụng Tài nguyên;
        trong t đơn vị
        thời gian

R.release;
```

a)

```
Type resource-allocation = monitor

  var      busy  : boolean;
           x     : condition;

  procedure acquire (time: integer)
  {
      if busy then x.wait(time );
      busy:= true;
  }
  procedure release() {
      busy:= false;
      x.signal;
  }

  BEGIN
      busy:= false;
  END.
```

b)

Hình 6.20. Ví dụ cấp phát tài nguyên

Để minh họa cho kỹ thuật này, xét monitor điều khiển việc cấp phát tài nguyên cho các tiến trình cạnh tranh được minh họa trong Hình 6.20b. Khi yêu cầu tài nguyên, tiến trình cần chỉ rõ thời gian dự kiến sử dụng tài nguyên tối đa. Monitor sẽ cấp tài nguyên cho tiến trình có thời gian yêu cầu ngắn nhất. Tiến trình muốn truy cập tới tài nguyên phải thực hiện như trong Hình 6.20a. Ở đây, R thuộc kiểu resource-allocation. Tuy nhiên, monitor không đảm bảo trình tự truy cập trước đó sẽ được bảo tồn. Cụ thể:

- Tiến trình có thể truy cập trái phép tới tài nguyên.
- Tiến trình có thể không bao giờ giải phóng tài nguyên đã được cấp phát.
- Tiến trình có thể thử giải phóng tài nguyên mà mình chưa yêu cầu.
- Tiến trình có thể yêu cầu cùng một tài nguyên liên tiếp hai lần (mà chưa giải phóng tài nguyên trước).

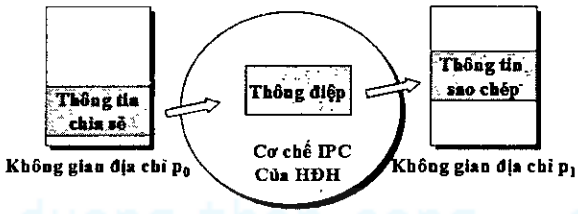
Chú ý, cấu trúc miền găng cũng có những khó khăn tương tự, và về mặt bản chất, những khó khăn này là động lực để phát triển cấu trúc vùng găng và monitor. Trước đây chúng ta phải lo lắng về cách sử dụng semaphore chính xác. Bây giờ chúng ta lo lắng về việc sử dụng chính xác những toán tử ở mức cao (mà trình biên dịch không thể giúp đỡ lập trình viên được nữa).

Một giải pháp khả dĩ cho những vấn đề trên là đặt các toán tử truy cập tài nguyên bên trong monitor *resource-allocation*. Thế nhưng, điều này dẫn đến việc điều phối sẽ tuân theo thuật toán điều phối của monitor chứ không

phải thuật toán mà chúng ta mong muốn. Để đảm bảo tất cả tiến trình thực hiện theo đúng trình tự chính xác, phải giám sát tất cả các chương trình có sử dụng monitor *resource-allocation* và các tài nguyên của nó. Có hai điều phải kiểm tra để đảm bảo tính đúng đắn của hệ thống. Thứ nhất, tiến trình người dùng phải sử dụng các toán tử của monitor theo đúng thứ tự. Thứ hai, phải chắc chắn một tiến trình "bất hợp tác" không thể vi phạm độc quyền truy xuất của monitor, tức là có thể truy cập trực tiếp tới tài nguyên dùng chung mà không sử dụng quy tắc truy cập.

6.7. CƠ CHẾ IPC

Những cơ chế đồng bộ trên đòi hỏi các tiến trình có biến dùng chung. Trong phần này, trình bày cơ chế truyền thông liên tiến trình cho phép các tiến trình trao đổi dữ liệu và đồng bộ hoạt động thông qua việc chuyển thông điệp. Thông điệp (message) là một khối thông tin theo khuôn dạng nào đó (đã được thống nhất trước) được trao đổi giữa hai tiến trình. Cơ chế bảo vệ bộ nhớ trong các HĐH hiện đại không cho các tiến trình xâm nhập dù vô tình hay cố ý vào không gian nhớ của tiến trình khác. Để thực hiện trao đổi thông tin, HĐH đóng vai trò trung chuyển thông điệp. IPC là cơ chế cho phép tiến trình sao chép nội dung thông tin muốn gửi vào thông điệp và sau đó HĐH sao chép nội dung thông điệp vào không gian địa chỉ tiến trình nhận (Hình 6.21).

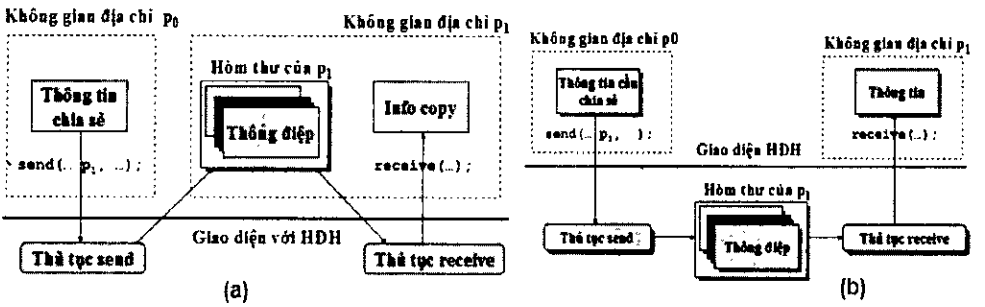


Hình 6.21. Cơ chế truyền thông liên tiến trình

6.7.1. Hộp thư

Hình 6.22 minh họa việc thông tin được sao chép trực tiếp sang không gian địa chỉ của tiến trình nhận, mà phía nhận có thể không biết sự kiện này xảy ra. Có thể tránh điều này bằng cách không chuyển thông tin trừ khi phía nhận đưa ra yêu cầu nhận tường minh. HĐH có thể giữ tạm thông điệp ở hộp thư trước khi sao chép thông tin sang không gian địa chỉ của tiến trình

nhận. Hộp thư có thể để ở không gian HĐH (Hình 6.22b) hoặc không gian địa chỉ tiến trình nhận (Hình 6.22a). Nếu đặt ở không gian người nhận, thủ tục nhận có thể cài đặt bằng thư viện người dùng và trình biên dịch và bộ tải phải cấp phát bộ nhớ làm hộp thư. Tiến trình người dùng khi chạy có thể vô tình phá hỏng hộp thư. Nếu hộp thư đặt trong không gian HĐH, phía nhận sẽ chỉ nhận được thông điệp khi tường minh gọi thủ tục nhận. Hộp thư sẽ không bị tiến trình người dùng thay đổi (do đặt trong không gian HĐH), nhưng điều này lại khiến HĐH phải tốn bộ nhớ làm hộp thư.



Hình 6.22. Hộp thư

cuu duong than cong . com

6.7.2. Chuẩn thông điệp

Để hai bên có thể hiểu nhau, thông điệp phải tuân theo một quy tắc nào đấy. Thông thường, thông điệp có tiêu đề chứa các thông tin liên quan đến thông điệp (chẳng hạn định danh tiến trình gửi, nhận, kích thước thông điệp,...).

6.7.3. Thao tác gửi và nhận

Dung lượng của hộp thư là số lượng thông điệp tối đa hộp thư có thể chứa. Thuộc tính này có thể nhận 3 giá trị:

cuu duong than cong . com

- **Dung lượng 0**: Hộp thư không thể có thông điệp đợi nhận. Tiến trình gửi phải đợi đến khi tiến trình phía bên kia nhận được thông điệp. Hai tiến trình phải đồng bộ hóa khi truyền thông điệp.
- **Dung lượng hữu hạn**: Hộp thư kích thước n có thể chứa tối đa n thông điệp. Nếu hộp thư chưa đầy, thông điệp mới gửi được đặt trong hộp thư (có thể chính thông điệp hoặc địa chỉ thông điệp được đặt vào hộp thư). Sau đó tiến trình gửi có thể tiếp tục thực hiện công

việc của mình. Khi hộp thư đầy, tiến trình gửi phải đợi cho đến khi hộp thư có chỗ trống.

- **Dung lượng vô hạn**: Mọi thông điệp có thể được đặt trong hộp thư. Tiến trình gửi không bao giờ phải ngừng.

Trường hợp dung lượng 0 được gọi là chuyển thông điệp không có bộ đệm; còn hai trường hợp kia là bộ đệm tự động. Trong trường hợp dung lượng khác 0, sau khi gửi thành công thông điệp, tiến trình không xác định được khi nào thông điệp đến đích. Nếu thông tin này quan trọng trong quá trình tính toán, tiến trình gửi phải hỏi tiến trình nhận một cách tường minh để biết khi nào thông điệp đến đích. Ví dụ, tiến trình P gửi thông điệp đến Q và có thể tiếp tục thực thi chỉ khi thông điệp đến đích. Tiến trình P thực hiện hai lệnh sau: **Send(Q, mess)** và **Receive(Q, mess)**, còn tiến trình Q thực thi hai lệnh: **Receive(P, mess)** và **Send(P, ack)**. Kiểu giao tiếp này được gọi là giao tiếp đồng bộ.

Có một trường hợp đặc biệt không nằm hoàn toàn trong ba giải pháp nêu trên:

cuu duong than cong . com

- Sau khi gửi thông điệp, tiến trình vẫn tiếp tục thực thi. Tuy nhiên, tiến trình nhận chưa nhận thông điệp mà tiến trình gửi lại tiếp tục gửi tiếp thông điệp khác, thì thông điệp đầu tiên bị mất. Ưu điểm của phương pháp này là thông điệp lớn không bị lặp nhiều lần. Nhược điểm là làm phức tạp công việc lập trình. Các tiến trình cần đồng bộ hóa một cách tường minh để đảm bảo thông điệp đến đích và cả bên gửi lẫn bên nhận không đồng thời thao tác trên bộ đệm.
- Tiến trình sau khi gửi thông điệp sẽ đợi cho đến khi phía bên kia trả lời. Phương pháp này được sử dụng trong HĐH Thoth. Thông điệp có kích thước cố định (8 từ). Tiến trình P sau khi gửi thông điệp sẽ bị phong tỏa cho đến khi phía nhận nhận được thông điệp và gửi thông điệp trả lời **reply(P, mess)**. Thông điệp trả lời sẽ nằm đè lên vùng đệm của thông điệp gốc. Điểm khác biệt duy nhất giữa thông điệp gửi và trả lời là thông điệp gửi làm tiến trình gửi bị phong tỏa, còn thông điệp trả lời làm cả hai tiến trình tiếp tục hoạt động.

Phương thức giao tiếp đồng bộ này có thể dễ dàng mở rộng thành hệ thống gọi thủ tục từ xa (Remote Procedure Call - RPC) với đầy đủ chức

năng. Mọi hệ thống RPC đều dựa trên một ý tưởng là, trong một hệ thống có một CPU, lời gọi thủ tục hay chương trình con giống như việc chuyển thông điệp, tiến trình gửi sẽ bị treo cho đến khi tiến trình nhận gửi thông điệp phản hồi biên nhận. Sau này có thể thấy RPC có thể mở rộng cho phép các tiến trình trên các máy tính khác nhau có thể phối hợp làm việc với nhau.

6.7.4. Xử lý biệt lệ (Exception Conditions)

Ưu điểm của hệ thống chuyển thông điệp nổi bật trong môi trường phân tán vì tiến trình có thể nằm trên các máy tính khác nhau. Tuy nhiên, khả năng xuất hiện lỗi trong môi trường này rất lớn, do đó cần một cơ chế khắc phục (xử lý biệt lệ).

☞ ***Tiến trình bị kết thúc***

Tiến trình gửi hoặc nhận có thể bị phong tỏa trước khi xử lý xong thông điệp. Tình huống này có thể khiến thông điệp không bao giờ nhận được hoặc tiến trình phải đợi một thông điệp sẽ không bao giờ đến. Xét 2 trường hợp sau:

1. Tiến trình P đợi thông điệp từ tiến trình Q đã kết thúc. Nếu không có cơ chế giải quyết, P sẽ bị phong tỏa vĩnh viễn. Trong trường hợp này, hệ thống có thể hoặc cho P kết thúc hoặc báo cho P biết Q đã kết thúc.
2. Tiến trình P có thể gửi một thông điệp cho tiến trình Q đã bị ngắt. Không có vấn đề gì xảy ra trong phương pháp bộ đệm tự động, P tiếp tục hoạt động. Nếu P cần xác định thông điệp của mình đã được Q xử lý chưa, P phải lập trình tường minh để đợi xác nhận từ Q. Trong trường hợp không có bộ đệm, P sẽ bị phong tỏa hoàn toàn. Giống như trong trường hợp 1, hệ thống có thể hoặc cho P kết thúc, hoặc báo cho P biết Q đã kết thúc.

☞ ***Mất thông điệp***

Thông điệp P gửi cho Q có thể bị mất (do lỗi phần cứng hay lỗi đường truyền). Có ba phương thức cơ bản khắc phục điều này:

1. HĐH chịu trách nhiệm phát hiện và sau đó gửi lại thông điệp bị mất.
2. Tiến trình gửi có trách nhiệm phát hiện và truyền lại thông điệp nếu thấy cần thiết.

3. HĐH chịu trách nhiệm phát hiện mất thông điệp. Sau đó HĐH báo cho tiến trình gửi thông điệp bị mất. Tiến trình gửi tự động đưa ra phương thức giải quyết.

Không phải việc phát hiện lỗi lúc nào cũng quan trọng. Trên thực tế, bên cạnh các giao thức mạng không đảm bảo chuyển thông điệp tin cậy thì vẫn có giao thức đảm bảo tính tin cậy khi chuyển (TCP). Chính người sử dụng phải chỉ rõ (nghĩa là hoặc báo với hệ thống, hoặc tự lập trình) xem có cần đảm bảo tin cậy trong quá trình truyền không.

Làm thế nào để phát hiện mất thông điệp? Phương pháp phổ biến nhất là sử dụng cơ chế timeout (hết thời gian). Sau khi gửi thông điệp đi, tiến trình gửi sẽ đợi thông điệp biên nhận từ phía bên kia trong một khoảng thời gian định trước. Nếu hết khoảng thời gian này mà chưa nhận được phản hồi, HĐH (hoặc tiến trình) xem như thông điệp đã mất và gửi lại thông điệp. Có thể thông điệp không bị mất, mà chỉ đến muộn. Khi đó, sẽ có hai bản sao của cùng một thông điệp truyền qua mạng. Có thể sử dụng kỹ thuật đánh số thứ tự để phân biệt các thông điệp.

cuu duong than cong . com

☞ ***Thông điệp bị lỗi***

Thông điệp có thể đến đích, nhưng bị lỗi trên đường truyền (ví dụ nguyên nhân do nhiễu xuất hiện trên kênh truyền). Trường hợp này cũng giống như trường hợp mất thông điệp. Thường HĐH phải gửi lại thông điệp gốc. Các kỹ thuật như tổng kiểm tra (checksum) hay bit chẵn lẻ (parity bit) được sử dụng để phát hiện lỗi trong thông điệp.

6.8. NHẬN XÉT

Khi nhiều tiến trình tuần tự kết hợp với nhau để chia sẻ dữ liệu, HĐH phải đảm bảo độc quyền truy xuất. Một cách giải quyết là đảm bảo tại một thời điểm chỉ có đúng một tiến trình (hay thread) thực thi đoạn mã trong miền găng. Nhiều thuật toán được đưa ra để giải quyết vấn đề miền găng. Nhược điểm chính trong việc sử dụng đoạn mã ở mức người dùng là tình trạng busy waiting. Semaphore khắc phục được nhược điểm này. Semaphore có thể được sử dụng để giải quyết rất nhiều vấn đề đồng bộ hóa và có thể được cài đặt hiệu quả nếu phần cứng hỗ trợ các thao tác đơn nhất. Vùng găng có thể được sử dụng để giải quyết vấn đề độc quyền truy xuất

cũng như các vấn đề đồng bộ một cách an toàn và hiệu quả. Monitor cung cấp kỹ thuật đồng bộ để chia sẻ những loại dữ liệu trừu tượng. Biến kiểu condition cung cấp phương pháp cho phép thủ tục tự phong tỏa chính nó cho đến khi nó được phép khôi phục.

CÂU HỎI ÔN TẬP

1. Tại sao phải chia sẻ thông tin và những vấn đề gặp phải khi chia sẻ thông tin?
2. Trình bày các giải pháp độc quyền truy xuất.
3. Trình bày các giải pháp đồng bộ cơ bản.

cuu duong than cong . com

cuu duong than cong . com

Chương 7
BẾ TẮC

*Trong hệ thống, nhiều tiến trình cạnh tranh nhau quyền sử dụng lượng hữu hạn tài nguyên. Khi yêu cầu tài nguyên mà chưa được đáp ứng, tiến trình bị phong tỏa. Có thể tiến trình sẽ ở mãi trong trạng thái phong tỏa nếu tài nguyên yêu cầu bị các tiến trình khác cũng ở trạng thái phong tỏa chiếm giữ. Tình huống này được gọi là **bế tắc** và đã được nói qua ở phần semaphore trong Chương 6. Chương này trình bày các phương thức giải quyết bế tắc của HĐH.*

cuu duong than cong . com

7.1. MÔ HÌNH HỆ THỐNG

Trong quá trình hoạt động, các tiến trình cạnh tranh nhau quyền sử dụng tài nguyên hệ thống. Có nhiều kiểu tài nguyên, trong mỗi kiểu có thể có nhiều đối tượng giống nhau. Ví dụ, không gian bộ nhớ, CPU, file và thiết bị vào/ra (như máy in và đĩa từ) là các kiểu tài nguyên. Nếu hệ thống có hai CPU, thì kiểu tài nguyên CPU sẽ có hai đối tượng. Khi tiến trình yêu cầu một kiểu tài nguyên, hệ thống có thể cấp phát bất kỳ đối tượng nào của kiểu tài nguyên cho tiến trình. Tiến trình có thể sử dụng nhiều tài nguyên để hoàn thành công việc của mình, tuy nhiên, lượng tài nguyên tiến trình yêu cầu không thể vượt quá tổng lượng tài nguyên của hệ thống. Tiến trình sử dụng tài nguyên theo trình tự:

cuu duong than cong . com

1. **Yêu cầu (Request)**: Nếu không được hệ thống đáp ứng ngay thì tiến trình phải đợi cho đến có.
2. **Sử dụng (Use)**: Tiến trình sử dụng tài nguyên.
3. **Giải phóng (Release)**: Tiến trình trả tài nguyên cho hệ thống.

HĐH thường cung cấp cho chương trình người dùng các lời gọi hệ thống thực hiện yêu cầu và giải phóng thiết bị; yêu cầu mở và đóng file; yêu cầu cấp phát và giải phóng bộ nhớ. Yêu cầu và giải phóng các tài nguyên khác cũng có thể được thực hiện thông qua các thao tác **wait** và **signal** trên semaphore. HĐH có bảng tài nguyên hệ thống, qua đó xác định được trạng thái cấp phát tài nguyên (đã cấp phát hay chưa cấp phát, cấp phát cho ai,...). Nếu yêu cầu tài nguyên mà không được đáp ứng, tiến trình bị phong tỏa tại hàng đợi tài nguyên.

Tập hợp tiến trình ở trong trạng thái bế tắc nếu mỗi tiến trình đợi một sự kiện gây ra bởi tiến trình khác nằm trong cùng tập hợp. Ở đây quan tâm chủ yếu đến sự kiện chiếm giữ và giải phóng tài nguyên.

7.2. ĐẶC ĐIỂM CỦA BẾ TẮC

Bế tắc là tình trạng không mong muốn, vì khi xảy ra không tiến trình nào có thể kết thúc, tài nguyên hệ thống bị chiếm giữ. Trước khi trình bày các giải pháp xử lý, cần tìm hiểu một số đặc điểm của hiện tượng này.

cuu duong than cong . com

7.2.1. Điều kiện cần

Bế tắc sẽ xảy ra nếu bốn điều kiện sau đồng thời xuất hiện:

1. **Độc quyền truy xuất (Mutual Exclusion)**: Ít nhất một tài nguyên bị nắm giữ thuộc kiểu không thể dùng chung. Nghĩa là, tại một thời điểm chỉ có tối đa một tiến trình được quyền sử dụng tài nguyên.
2. **Giữ và chờ (Hold and wait)**: Phải có tiến trình đang nắm giữ tài nguyên đồng thời lại chờ tài nguyên bị tiến trình khác chiếm giữ.
3. **Không chiếm đoạt (No preemption)**: Hệ thống không thể tước đoạt tài nguyên của tiến trình, trừ khi tiến trình chủ động giải phóng tài nguyên khi không cần thiết.
4. **Vòng đợi (Circular wait)**: Tồn tại tập hợp các tiến trình $\{P_0, P_1, ..., P_n\}$ đang trong trạng thái chờ tài nguyên. Trong đó P_0 chờ tài nguyên bị P_1 nắm giữ, P_1 chờ tài nguyên bị P_2 nắm giữ,..., P_{n-1} chờ tài nguyên bị P_n nắm giữ và P_n chờ tài nguyên bị P_0 nắm giữ.

cuu duong than cong . com

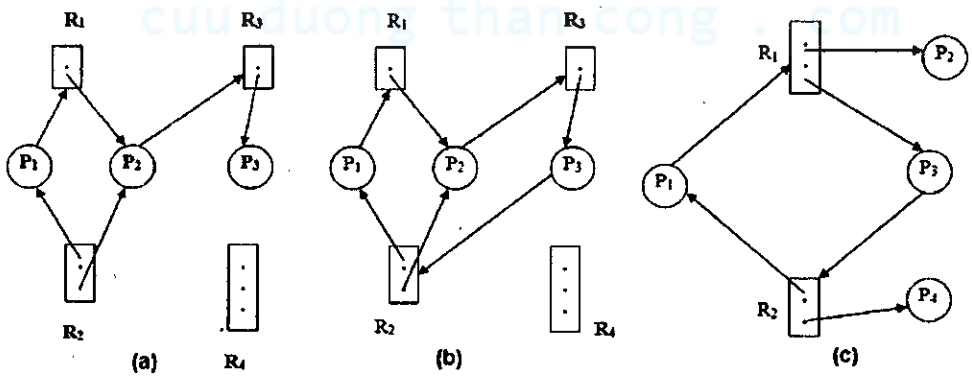
Cả bốn điều kiện trên phải đồng thời xuất hiện thì hiện tượng bế tắc mới xảy ra. Điều kiện Vòng đợi kéo theo điều kiện Giữ và Chờ, do đó bốn điều

kiện này không hẳn hoàn toàn độc lập với nhau. Tuy nhiên, sẽ thuận tiện hơn nếu xét tách biệt các điều kiện này.

7.2.2. Đồ thị phân phối tài nguyên

Đồ thị có tập đỉnh V và tập cung E. Tập đỉnh V được phân thành hai loại: $P = \{P_1, P_2,..., P_n\}$ ứng với các tiến trình và $R = \{R_1, R_2,..., R_m\}$ ứng với kiểu tài nguyên của hệ thống. *Cung yêu cầu* $P_i \rightarrow R_j$ hướng từ tiến trình P_i tới tài nguyên R_j, có ý nghĩa P_i đã yêu cầu một đối tượng của kiểu tài nguyên R_j và hiện thời P_i đang trong trạng thái đợi. *Cung phân phối* $R_j \rightarrow P_i$ hướng từ tài nguyên R_j tới tiến trình P_i, có nghĩa là một đối tượng của kiểu tài nguyên R_j đã được cấp phát cho tiến trình P_i. Trên Hình 7.1a, tiến trình P_i được biểu diễn là cung tròn, tài nguyên R_j là hình vuông. Do kiểu tài nguyên R_j có thể có nhiều đối tượng, chúng ta biểu diễn mỗi đối tượng là một chấm nhỏ bên trong hình vuông. Lưu ý, cung yêu cầu chỉ nối tới hình vuông R_j, trong khi đó cung phân phối phải được chỉ rõ xuất phát từ chấm nào bên trong hình vuông.

cuu duong than cong . com



Hình 7.1. Đồ thị phân phối tài nguyên

cuu duong than cong . com

Khi tiến trình P_i yêu cầu đối tượng tài nguyên R_j, hệ thống bổ sung một cung yêu cầu vào đồ thị phân phối tài nguyên. Khi có thể đáp ứng yêu cầu, hệ thống ngay lập tức chuyển cung yêu cầu thành cung phân phối. Cung phân phối bị xóa bỏ khi tiến trình giải phóng tài nguyên.

Đồ thị phân phối tài nguyên trên Hình 7.1a mô tả tình huống sau:

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

$E = \{P_1 \rightarrow R_1, P_2 \rightarrow R_3, R_1 \rightarrow P_2, R_2 \rightarrow P_2, R_2 \rightarrow R_1, R_3 \rightarrow P_3\}$.

Số đối tượng của mỗi tài nguyên: R_1 có 1 đối tượng, R_2 có 2 đối tượng, R_3 có 1 đối tượng, R_4 có 3 đối tượng.

Trạng thái các tiến trình: P_1 đang giữ một đối tượng tài nguyên R_2 và đang đợi một đối tượng R_1. P_2 đang giữ một đối tượng R_1 và đang đợi thêm một đối tượng R_3. P_3 đang giữ một đối tượng R_3.

Với đồ thị phân phối tài nguyên, có thể thấy, nếu không có chu trình trên đồ thị thì không tiến trình nào bị bế tắc. Tuy nhiên, nếu đồ thị có chứa chu trình, bế tắc chưa chắc, mà chỉ có khả năng xuất hiện. Nếu mỗi kiểu tài nguyên có đúng một đối tượng, thì chu trình dẫn đến bế tắc và mỗi tiến trình trong chu trình đều rơi vào trạng thái bế tắc. Khi đó, chu trình trong đồ thị là điều kiện cần và đủ cho trạng thái bế tắc. Nhưng, nếu kiểu tài nguyên có nhiều đối tượng, chu trình không nhất thiết kéo theo bế tắc (chu trình là điều kiện cần nhưng chưa đủ).

Xét đồ thị phân phối tài nguyên trên Hình 7.1b, giả sử P_3 yêu cầu một đối tượng R_2. Do hệ thống không còn đối tượng tài nguyên R_2 rỗi nên cung yêu cầu $P_3 \rightarrow R_2$ được thêm vào đồ thị. Lúc này, xuất hiện hai chu trình: ($P_1 \rightarrow R_1 \rightarrow P_2 \rightarrow R_3 \rightarrow P_3 \rightarrow R_2 \rightarrow P_1$) và ($P_2 \rightarrow R_3 \rightarrow P_3 \rightarrow R_2 \rightarrow P_2$). Ba tiến trình P_1, P_2 và P_3 bế tắc: P_2 đợi R_3 – bị chiếm dụng bởi P_3. P_3 đợi P_1 và P_2 giải phóng R_2, còn P_1 đợi P_2 giải phóng R_1. Đồ thị phân phối tài nguyên trên Hình 7.1c có chu trình ($P_1 \rightarrow R_1 \rightarrow P_3 \rightarrow R_2 \rightarrow P_1$). Tuy nhiên, không có bế tắc vì P_4 có thể giải phóng một đối tượng R_2. Tài nguyên này sau đó có thể được cấp phát cho P_3 (chu trình bị phá vỡ). Tóm lại, nếu đồ thị phân phối tài nguyên không có chu trình, thì hệ thống chắc chắn không ở trong trạng thái bế tắc. Ngược lại, nếu có chu trình, hệ thống có thể rơi vào trạng thái bế tắc. Chúng ta cần chú ý tới điểm này khi giải quyết hiện tượng bế tắc.

7.2.3. Giải quyết bế tắc

Có ba giải pháp cho hiện tượng bế tắc như sau:

1. Đảm bảo hệ thống không rơi vào trạng thái bế tắc.
2. Cho phép hệ thống rơi vào trạng thái bế tắc rồi sau đó khắc phục.
3. Bỏ qua tất cả các vấn đề và giả định bế tắc không bao giờ xuất hiện trong hệ thống. Phương pháp này được sử dụng trong nhiều HĐH, kể cả UNIX.

Phần này chỉ mô tả ngắn gọn các giải pháp, chi tiết được trình bày trong phần sau. Để đảm bảo không xảy ra bế tắc, hệ thống có thể sử dụng phương pháp Ngăn chặn hoặc phương pháp Tránh. Ngăn chặn là phương pháp ngăn không cho ít nhất một trong bốn điều kiện cần của bế tắc xuất hiện. Ngược lại, phương pháp tránh đòi hỏi tiến trình phải báo trước cho HĐH các thông tin liên quan đến nhu cầu tài nguyên của mình. Với những thông tin biết trước này, khi tiến trình xin cấp phát tài nguyên, hệ thống có thể quyết định liệu có nên cấp phát cho tiến trình không. Với mỗi yêu cầu, hệ thống xét lượng tài nguyên chưa cấp phát, lượng tài nguyên đã cấp phát, các yêu cầu cấp phát và giải phóng tài nguyên sắp xuất hiện để quyết định xem có thỏa mãn yêu cầu hay không.

Nếu hệ thống không sử dụng phương pháp nào trong hai phương pháp trên, trạng thái bế tắc có khả năng xuất hiện. Khi đó hệ thống cần xác định bế tắc xuất hiện chưa và sau đó thì khắc phục như thế nào.

Nếu hệ thống không đảm bảo bế tắc không xuất hiện và cũng không có cơ chế kiểm tra và khắc phục bế tắc, hệ thống có thể rơi vào trạng thái bế tắc mà không nhận ra. Lúc đó, hiệu suất tổng thể hệ thống bị suy giảm vì nhiều tài nguyên bị các tiến trình trong trạng thái phong tỏa chiếm giữ, ngày càng nhiều tiến trình rơi vào bế tắc khi yêu cầu thêm tài nguyên. Cuối cùng, hệ thống ngừng hoạt động và phải khởi động lại một cách thủ công. Mặc dù giải pháp này có vẻ không ổn, nhưng trên thực tế lại được sử dụng rộng rãi nhất. Nếu bế tắc xuất hiện không thường xuyên (một lần/năm) việc cài đặt các giải pháp Ngăn chặn, Tránh hay Phát hiện và Khắc phục bế tắc quá tốn kém.

7.3. NGĂN CHẶN BẾ TẮC

Bế tắc có bốn đặc điểm được xem là điều kiện cần. Như vậy, chỉ cần ngăn không cho một trong bốn điều kiện này xuất hiện là có thể ngăn chặn bế tắc.

7.3.1. Độc quyền truy xuất (Mutual Exclusion)

Điều kiện độc quyền truy xuất luôn đúng với những tài nguyên không thể chia sẻ (ví dụ máy in). Ngược lại, kiểu tài nguyên có thể chia sẻ (ví dụ file với thuộc tính chỉ đọc) không đòi hỏi độc quyền truy xuất, do đó không

liên quan tới hiện tượng bế tắc. Do bản chất tài nguyên, nên không thể ngăn chặn bế tắc bằng cách loại bỏ điều kiện độc quyền truy xuất.

7.3.2. Giữ và chờ

Để ngăn chặn điều kiện Giữ và Chờ, phải đảm bảo chỉ khi không nắm giữ tài nguyên thì tiến trình mới được yêu cầu tài nguyên. Giải pháp thứ nhất là trước khi thực thi, tiến trình yêu cầu tất cả các tài nguyên cần thiết. Giải pháp thứ hai là, trước khi yêu cầu thêm tài nguyên, tiến trình phải giải phóng tất cả các tài nguyên mình đang chiếm giữ. Để minh họa sự khác biệt giữa hai giải pháp này, xét tiến trình sao lưu dữ liệu từ băng từ vào ổ cứng, sắp xếp các file, sau đó in kết quả ra máy in. Nếu tiến trình yêu cầu tất cả tài nguyên ngay từ đầu thì phải yêu cầu cả ba loại tài nguyên (băng từ, ổ cứng và máy in). Mặc dù chỉ sử dụng vào giai đoạn cuối, nhưng tiến trình sẽ giữ máy in trong suốt quá trình thực thi. Phương pháp thứ hai cho phép tiến trình chỉ yêu cầu băng từ và ổ đĩa lúc đầu. Tiến trình chuyển dữ liệu từ băng từ vào đĩa, rồi giải phóng cả hai tài nguyên này. Tiến trình tiếp tục yêu cầu ổ đĩa và máy in. Sau khi chuyển dữ liệu từ ổ đĩa ra máy in, tiến trình giải phóng tất cả các tài nguyên rồi kết thúc. Các giải pháp này có hai nhược điểm chính. Thứ nhất, hiệu quả sử dụng tài nguyên thấp vì nhiều loại tài nguyên bị chiếm giữ ngay từ đầu nhưng chưa chắc được sử dụng ngay. Thứ hai, có thể xảy ra hiện tượng chết đói. Tiến trình có thể phải đợi vô hạn vì có thể các tài nguyên cần thiết đã bị cấp phát cho tiến trình khác.

7.3.3. Không chiếm đoạt

Chúng ta có thể ngăn chặn bằng cách: Khi tiến trình đang giữ một số tài nguyên và yêu cầu thêm tài nguyên đã bị chiếm giữ, thì trước khi tiến trình bị phong tỏa, hệ thống sẽ thu hồi toàn bộ tài nguyên đã cấp phát cho tiến trình. Tài nguyên sau khi thu hồi được đưa vào danh sách tài nguyên chưa cấp phát. Tiến trình sẽ chỉ khởi động lại nếu được cấp phát lại tài nguyên cũ và tài nguyên mới yêu cầu. Một giải pháp linh hoạt hơn là, khi tiến trình yêu cầu tài nguyên, hệ thống kiểm tra xem có đáp ứng được không. Nếu có, hệ thống cấp phát cho tiến trình. Nếu không, hệ thống kiểm tra xem những tài nguyên này có bị chiếm giữ bởi các tiến trình đang trong trạng thái phong tỏa hay không. Nếu có, hệ thống thu hồi tài nguyên của tiến trình đang bị phong tỏa để cấp phát cho tiến trình yêu cầu. Ngược lại, tiến trình yêu cầu

sẽ phải đợi. Trong khi bị phong tỏa, một số tài nguyên của tiến trình có thể bị thu hồi cho các tiến trình khác. Tiến trình chỉ có thể khởi động lại khi được cấp phát thêm tài nguyên mới và lấy lại các tài nguyên đã bị thu hồi khi còn ở trạng thái phong tỏa. Phương pháp này thường áp dụng cho những tài nguyên mà trạng thái có thể lưu lại và khởi tạo dễ dàng (thanh ghi hay bộ nhớ trong), nhưng không thể áp dụng cho các loại tài nguyên như máy in hay băng từ.

7.3.4. Vòng đợi

Có thể ngăn chặn điều kiện "vòng đợi" bằng cách đánh số thứ tự cho tất cả các tài nguyên, và tiến trình phải yêu cầu tài nguyên theo thứ tự tăng dần. Giả sử $R = \{R_1, R_2,... R_m\}$ là tập tài nguyên, ta gán cho mỗi tài nguyên một số thứ tự (STT) nguyên duy nhất. Cách đánh STT phải đảm bảo có thể so sánh được STT của hai tài nguyên, nhằm xác định tài nguyên nào đứng trước, tài nguyên nào đứng sau. Nghĩa là về mặt hình thức, ta định nghĩa hàm $F: R \rightarrow N$, trong đó N là tập các số nguyên. Ví dụ, nếu R bao gồm băng từ, ổ đĩa và máy in thì hàm F có thể định nghĩa như sau: F(băng từ) =1, F(ổ đĩa) = 5, F(máy in) = 12. Xét phương pháp ngăn chặn "bế tắc" sau đây: Tiến trình phải yêu cầu tài nguyên theo STT tăng dần. Nghĩa là, lúc đầu tiến trình có thể yêu cầu bất cứ tài nguyên R_i nào. Sau đó, nếu yêu cầu thêm tài nguyên R_j thì $F(R_j) > F(R_i)$. Nếu cần nhiều đối tượng trong cùng một kiểu tài nguyên, tiến trình phải yêu cầu trong một yêu cầu duy nhất. Ví dụ, với hàm F trên, nếu muốn sử dụng băng từ và máy in tại cùng thời điểm, thì tiến trình phải yêu cầu băng từ trước khi yêu cầu máy in. Giải pháp tương đương là khi yêu cầu R_j, tiến trình phải giải phóng tất cả các tài nguyên R_i mà $F(R_i) > F(R_j)$. Nếu sử dụng hai quy tắc này, thì vòng đợi không bao giờ xảy ra (có thể chứng minh bằng phản chứng).

cuu duong than cong . com

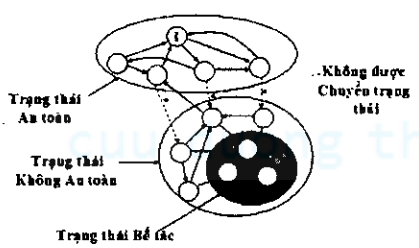
7.4. TRÁNH BẾ TẮC

Các thuật toán ngăn chặn bế tắc đặt ra quy tắc mà tiến trình phải tuân theo khi yêu cầu tài nguyên nhằm ngăn cản một trong bốn điều kiện bế tắc xuất hiện. Nhược điểm của giải pháp này là giảm hiệu suất sử dụng tài nguyên cũng như hiệu suất tổng thể hệ thống. Một thuật toán tránh bế tắc khác là, yêu cầu tiến trình cung cấp thêm thông tin về việc sử dụng tài

nguyên. Ví dụ, trong hệ thống có một băng từ và một máy in, hệ thống có thể biết tiến trình P yêu cầu băng từ trước yêu cầu máy in. Ngược lại, tiến trình Q yêu cầu máy in trước băng từ. Với những hiểu biết khi nào tiến trình sẽ yêu cầu và giải phóng tài nguyên, hệ thống có thể quyết định việc cấp phát tài nguyên. Khi có yêu cầu, hệ thống xét trạng thái cấp phát tài nguyên (các tài nguyên hiện có; các tài nguyên đã cấp phát; các yêu cầu xin và giải phóng tài nguyên trong tương lai) để quyết định xem có cấp phát tài nguyên cho tiến trình hay không. Các thuật toán khác nhau có thể đòi hỏi biết trước các kiểu thông tin khác nhau. Thông thường, tiến trình khai báo số lượng cực đại các kiểu tài nguyên mà nó cần.

7.4.1. Trạng thái an toàn

Hệ thống ở trong trạng thái an toàn nếu có thể cấp phát đầy đủ tài nguyên cho mỗi tiến trình mà không bị "bế tắc". Định nghĩa một cách hình thức, hệ thống ở trong trạng thái an toàn chỉ khi tồn tại một dãy an toàn. Dãy tiến trình $<P_1, P_2,... P_n>$ được gọi là an toàn trong trạng thái phân phối tài nguyên hiện thời, nếu yêu cầu tài nguyên của tiến trình P_i có thể được đáp ứng bằng các tài nguyên chưa cấp phát cùng các tài nguyên đang bị giữ bởi tất cả các P_j, với $j < i$. Khi đó, nếu tài nguyên tiến trình P_i đang cần không được đáp ứng ngay, P_i có thể đợi tới khi tất cả các P_j kết thúc. Sau đó, P_i có thể được cấp phát đủ tài nguyên cần thiết để thực hiện nhiệm vụ của mình. Khi P_i kết thúc, P_{i+1} có thể lấy được các tài nguyên cần thiết,... Nếu không tồn tại dãy như vậy, hệ thống được xem là không an toàn.



(a)

Tiến trình	Lượng cần tối đa	Lượng cần hiện thời
P_0	10	5
P_1	4	2
P_2	9	2

(b)

Hình 7.2. Minh họa trạng thái bế tắc

Trạng thái an toàn không phải là trạng thái bế tắc. Ngược lại, trạng thái bế tắc là một trạng thái không an toàn. Tuy nhiên, không phải trạng thái

không an toàn nào cũng là trạng thái bế tắc (Hình 7.2a). Trạng thái không an toàn có thể dẫn đến bế tắc. Chừng nào hệ thống còn ở trạng thái an toàn, HĐH có thể ngăn chặn bế tắc. Trong trạng thái không an toàn, HĐH không thể ngăn chặn các tiến trình yêu cầu tài nguyên và có thể dẫn đến bế tắc.

Để minh hoạ, xét hệ thống có 12 băng từ với 3 tiến trình: P_0, P_1, P_2, trong đó P_0 cần tối đa 10 băng từ, P_1 cần 4 băng từ và P_2 cần 9 băng từ. Giả sử tại thời điểm t_0, P_0 đang giữ 5, P_1 giữ 2 và P_2 giữ 2 băng từ (3 băng từ chưa cấp phát). Ví dụ này minh họa trên Hình 7.2b. Tại thời điểm t_0, hệ thống trong trạng thái an toàn vì dãy $<P_1, P_0, P_2>$ thỏa mãn điều kiện an toàn. P_1 có thể lấy 2 băng từ, thực thi, rồi giải phóng (hệ thống có 5 băng từ rỗi); kế đó P_0 lấy thêm 5 băng từ, thực thi và giải phóng (lúc này hệ thống có 10 băng từ rỗi). Cuối cùng P_2 thực thi và giải phóng các tài nguyên (cuối cùng hệ thống có 12 băng từ rỗi). Lưu ý, từ trạng thái an toàn có thể dẫn đến trạng thái không an toàn. Giả sử tại thời điểm t_1, P_2 yêu cầu và được cấp phát thêm 1 băng từ. Hệ thống sẽ không ở trạng thái an toàn nữa. Lúc này, chỉ tiến trình P_1 có thể lấy đủ băng từ theo nhu cầu của mình và khi P_1 trả lại, hệ thống có 4 băng từ. Trong khi đó, P_0 có thể yêu cầu thêm 5 hoặc P_2 yêu cầu thêm 6 băng từ, do hệ thống không thể đáp ứng được nên P_0 và P_2 rơi vào trạng thái bế tắc. Sai lầm ở đây là cấp phát thêm một băng từ cho P_2. Nếu bắt P_2 đợi tới khi các tiến trình khác kết thúc và giải phóng tài nguyên, hệ thống có thể tránh được trạng thái bế tắc.

Có thể đảm bảo bế tắc không xuất hiện bằng cách bắt hệ thống luôn ở trạng thái an toàn. Ban đầu, hệ thống ở trạng thái an toàn. Khi tiến trình yêu cầu tài nguyên, hệ thống phải quyết định xem liệu có nên cấp phát không. Yêu cầu chỉ được đáp ứng nếu sau khi cấp phát, hệ thống vẫn ở trạng thái an toàn. Có thể khi yêu cầu và hệ thống có đủ tài nguyên, nhưng tiến trình vẫn phải đợi. Do đó hiệu suất sử dụng tài nguyên cũng không cao.

7.4.2. Thuật toán Đồ thị cấp phát tài nguyên

Nếu mỗi kiểu tài nguyên chỉ có đúng một đối tượng, thì có thể áp dụng đồ thị cấp phát tài nguyên trong mục 7.2.2 để tránh bế tắc. Ngoài cung Yêu cầu và Phân phối có thêm cung Nhu cầu. Cung nhu cầu $P_i \rightarrow R_j$ chỉ ra tiến trình P_i có thể yêu cầu tài nguyên R_j tại thời điểm nào đó trong tương lai. Cung này có hướng giống cung yêu cầu, nhưng được biểu diễn bởi một

đường đứt nét. Khi tiến trình P_i yêu cầu tài nguyên R_j, cung nhu cầu $P_i \rightarrow R_j$ biến thành cung yêu cầu. Tương tự, khi P_i giải phóng R_j, cung phân phối $R_j \rightarrow P_i$ được đổi thành cung nhu cầu $P_i \rightarrow R_j$. Giả sử tiến trình P_i yêu cầu tài nguyên R_j. Yêu cầu chỉ có thể được thỏa mãn khi việc chuyển cung yêu cầu $P_i \rightarrow R_j$ thành cung phân phối không tạo nên chu trình trong đồ thị cấp phát tài nguyên. Ở đây thuật toán xác định chu trình trong đồ thị có độ phức tạp $O(n^2)$, với n là số tiến trình trong hệ thống. Nếu không có chu trình, thì sau khi cấp phát tài nguyên, hệ thống vẫn ở trong trạng thái an toàn. Ngược lại, nếu có chu trình, việc cấp phát sẽ đặt hệ thống trong trạng thái không an toàn. Do vậy, tiến trình P_i sẽ phải đợi.

7.4.3. Thuật toán Ngân hàng

Thuật toán đồ thị cấp phát tài nguyên không áp dụng được nếu kiểu tài nguyên có nhiều đối tượng. Khi đó, phải sử dụng thuật toán Ngân hàng (được áp dụng ở bộ phận cho vay của ngân hàng). Người vay (tiến trình) sẽ được ngân hàng (bộ phận quản lý tài nguyên) cấp một hạn mức tín dụng (số tiền cực đại khách hàng được vay). Khách hàng có thể vay dưới hạn mức tín dụng của mình. Khi vay thêm, khách hàng chưa phải trả ngay lập tức khoản vay trước. Việc cho vay của ngân hàng được xác định dựa trên trạng thái của ngân hàng, tức là lượng tiền mặt có sẵn, hạn mức tín dụng và số tiền vay của từng khách hàng. Nếu ngân hàng có thể thỏa mãn ít nhất một khách hàng nào đó (kể cả khi người này vay tối đa bằng đúng hạn mức tín dụng) thì ngân hàng có thể đợi đến thời hạn thanh toán, rồi thu toàn bộ số tiền này về, sau đó tiếp tục phục vụ khách hàng khác. Nếu tất cả khách hàng đều có thể vay được theo nhu cầu của mình (tất nhiên không vượt quá hạn mức tín dụng) và sau đó trả lại tiền cho ngân hàng thì ngân hàng ở trong trạng thái an toàn. Trong thuật toán Ngân hàng, tiến trình thông báo số lượng cực đại các đối tượng của mỗi kiểu tài nguyên cần thiết. Khi tiến trình yêu cầu tài nguyên, hệ thống phải xác định liệu sau khi cấp phát thì hệ thống có còn trong trạng thái an toàn không? Nếu có, tài nguyên được cấp phát; ngược lại, tiến trình phải đợi. Sau đây là một số cấu trúc dữ liệu biểu diễn trạng thái hệ thống với n là số các tiến trình của hệ thống và m là số các kiểu tài nguyên:

Available[1..m]: Số lượng các tài nguyên trong mỗi kiểu tài nguyên. **Available[j] = k**, nghĩa là kiểu tài nguyên R_j có k đối tượng chưa cấp phát.

Max[1..n, 1..m]: Ma trận xác định yêu cầu lớn nhất của mỗi tiến trình. **Max[i, j] = k**, nghĩa là tiến trình P_i có thể yêu cầu nhiều nhất k đối tượng tài nguyên R_j.

Allocation[1..n, 1..m]: Ma trận xác định số lượng đối tượng của mỗi kiểu tài nguyên hiện thời được cấp phát cho mỗi tiến trình. **Allocation[i, j] = k**, nghĩa là P_i hiện thời được cấp phát k đối tượng R_j.

Need[1..n, 1..m]: Ma trận xác định lượng tài nguyên cần thiết của mỗi tiến trình. **Need[i, j] = k**, nghĩa là P_i có thể cần thêm k đối tượng tài nguyên R_j để hoàn thành nhiệm vụ.

Như vậy, **Need[i, j] = Max[i, j] – Allocation[i, j]**.

Giả sử X và Y là vector có độ dài n. Ta định nghĩa $X \leq Y$ khi và chỉ khi $X[i] \leq Y[i]$ với mọi i = 1, 2,..., n. Ví dụ, nếu X = {1, 7, 3, 2} và Y = {0, 3, 2, 1}, thì Y < X. Định nghĩa Y < X nếu $Y \leq X$ và $Y \neq X$. Có thể xem mỗi hàng trên ma trận Allocation và Need như một vector, ký hiệu là $Allocation_i$ và $Need_i$. $Allocation_i$ xác định những tài nguyên hiện thời P_i đang nắm giữ, $Need_i$ xác định những tài nguyên mà P_j có thể yêu cầu để hoàn thành nhiệm vụ.

☞ ***Thuật toán xác định trạng thái an toàn***

Thuật toán sau xác định trạng thái của hệ thống có an toàn hay không:

1. **Giả sử Work và Finish là vector có độ dài m và n. Khởi tạo Work = Availble và Finish[i] = false với i = 1, 2,..., n.**
2. **Tìm i thỏa mãn cả hai điều kiện: Finish[i] = false và $Need_i \leq$ Work. Nếu không có i nào như thế, chuyển qua bước 4.**
3. **Work = Work + $Allocation_i$; Finish[i] = true. Quay lại bước 2.**
4. **Nếu Finish[i] = true với tất cả i thì hệ thống ở trong trạng thái an toàn.**

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

☞ ***Thuật toán Yêu cầu Tài nguyên***

Giả sử $Request_i$ là vector yêu cầu của tiến trình P_j. **Requesti[j] = k**, nghĩa là tiến trình P_j muốn k đối tượng của kiểu tài nguyên R_j. Khi tiến trình P_j yêu cầu tài nguyên, các bước sau đây được thực hiện lần lượt:

1. **Nếu $Request_i \le Need_i$, chuyển sang bước 2. Ngược lại, đưa ra một điều kiện lỗi do tiến trình vượt quá yêu cầu tối đa của nó.**
2. **Nếu $Request_i \le Available$, chuyển sang bước 3. Ngược lại, P_j phải đợi vì chưa đủ tài nguyên.**
3. **Buộc hệ thống làm như thể đã cấp phát các tài nguyên P_j yêu cầu bằng cách sửa các trạng thái như sau:**
   a. **Available = Available – Request**
   b. **Allocation = $Allocation_i$ + $Request_j$**
   c. **Needi = $Need_i$ – $Request_i$**
4. **Nếu hệ thống vẫn ở trong trạng thái an toàn sau khi cấp phát tài nguyên, hệ thống sẽ thực sự cấp phát tài nguyên cho P_j. Tuy nhiên, nếu trạng thái mới không an toàn, P_j phải đợi $Request_i$ và hệ thống khôi phục lại trạng thái cấp phát tài nguyên cũ.**

Ví dụ minh họa:

	Allocation	Max	Available
	$R_0R_1R_2R_3$	$R_0R_1R_2R_3$	$R_0R_1R_2R_3$
P_0	2 0 1 1	3 2 1 4	1 2 2 2
P_1	0 1 2 1	0 2 5 2	
P_2	4 0 0 3	5 1 0 5	
P_3	0 2 1 4	1 5 3 0	
P_4	1 0 3 0	3 0 3 3	

(a)

Trạng thái an toàn	Trạng thái không an toàn
$R_0R_1R_2R_3$	$R_0R_1R_2R_3$
2 0 1 1	2 0 1 1
0 1 2 1	0 1 2 1
4 0 0 3	4 0 0 3
0 2 1 4	1 2 1 0
1 0 3 0	1 0 3 0

(b)

Hình 7.3. Ví dụ về trạng thái hệ thống

Giả sử hệ thống có 4 kiểu tài nguyên R_0, R_1, R_2, R_3 với số lượng tương ứng là 8, 5, 9, 7. Yêu cầu cực đại tài nguyên cũng như lượng tài nguyên hiện thời cấp phát cho các tiến trình được minh họa trong Hình 7.3a. Lượng tài nguyên còn lại của hệ thống là (1, 2, 2, 2). Ta thấy dãy (P_2, P_4, P_1, P_3, P_0) an toàn, nên hệ thống đang trong trạng thái an toàn. Hình 7.3b minh họa 2 ví dụ về hệ thống trong trạng thái an toàn và không an toàn.

7.5. PHÁT HIỆN BẾ TẮC

Nếu không ngăn chặn hoặc tránh, hệ thống có thể rơi vào trạng thái bế tắc. Khi đó, hệ thống cần xác định bế tắc xuất hiện chưa và sau đó khắc phục như thế nào? Cơ chế phát hiện và khắc phục đòi hỏi những chi phí phụ

trội không chỉ trong việc ghi nhớ những thông tin cần thiết cho thuật toán phát hiện, mà còn ở những mất mát chắc chắn sẽ có trong giai đoạn khắc phục.

7.5.1. Kiểu tài nguyên chỉ có một đối tượng

Trường hợp này có thể áp dụng thuật toán Đồ thị đợi - chờ được xây dựng từ đồ thị cấp phát tài nguyên, bằng cách xóa các đỉnh kiểu tài nguyên và các cung liên quan. Chính xác hơn, cung P_i tới P_j trong một đồ thị đợi - chờ có ý nghĩa là tiến trình P_j đang nắm giữ tài nguyên mà P_i cần. Tồn tại cung $P_i \rightarrow P_j$ trong đồ thị đợi - chờ khi và chỉ khi đồ thị cấp phát tài nguyên tương ứng chứa hai cung $P_i \rightarrow R_q$ và $R_q \rightarrow P_j$ với tài nguyên R_q nào đó. Giống như trước kia, bế tắc xuất hiện trong hệ thống khi và chỉ khi có chu trình trong đồ thị đợi - chờ. Để xác định bế tắc, hệ thống cần xây dựng đồ thị đợi - chờ và định kỳ thực thi chương trình tìm kiếm chu trình trên đồ thị này. Độ phức tạp của thuật toán là $O(n^2)$, trong đó n là số đỉnh của đồ thị.

7.5.2. Kiểu tài nguyên có nhiều đối tượng

Nếu kiểu tài nguyên có nhiều đối tượng thì không áp dụng được kỹ thuật Đồ thị đợi - chờ, mà phải sử dụng thuật toán xác định bế tắc bằng cách kiểm tra mọi dãy cấp phát có thể. Thuật toán có các cấu trúc dữ liệu **Available[1..m], Allocation[1..n, 1..m]** và **Request[1..n, 1..m]**.

1. **Giả sử Work và Finish là vector tương ứng có độ dài m và n. Khởi tạo Work = Available. Với i = 1, 2,..., n. Nếu $Allocation_i \neq 0$ thì Finish[i] = false; ngược lại, Finish[i] = true.**
2. **Tìm chỉ số i thỏa mãn cả hai điều kiện sau: (Finish[i] == false) và ($Request_i \leq Work$). Nếu không tìm được i nào như vậy, chuyển sang bước 4.**
3. **Work = Work + $Allocation_i$.Finish[i] = true, chuyển sang bước 2.**
4. **Nếu với $1 \leq i \leq n$ mà Finish[i] == false, thì hệ thống ở trong trạng thái bế tắc. Hơn nữa, tiến trình P_i bị bế tắc.**

Độ phức tạp của thuật toán là $O(m \times n^2)$. Hệ thống thu hồi các tài nguyên đã cấp phát cho P_i (trong bước 3) ngay khi xác định được $Request_i \leq Work$ vì P_i hiện thời không liên quan đến bế tắc ($Request_i \leq Work$). Do đó có thể lạc quan, giả sử P_i không yêu cầu thêm tài nguyên và sẽ sớm

trả lại hệ thống tất cả các tài nguyên đang nắm giữ. Nếu giả thiết này sai, bế tắc có thể xảy ra và sẽ được xác định trong lần thực thi thuật toán kế tiếp.

Để minh họa thuật toán, xét hệ thống có năm tiến trình từ P_0 tới P_4, ba kiểu tài nguyên R_0, R_1, R_2 có số đối tượng tương ứng là 7, 2 và 6. Giả sử tại thời điểm T_0, trạng thái cấp phát tài nguyên được minh họa trên Hình 7.4a. Hệ thống không ở trong tình trạng bế tắc, vì khi thuật toán thực thi, dãy $<P_0, P_2, P_3, P_1, P_4>$ dẫn đến Finish[i] = true với mọi i. Tuy nhiên, nếu P_2 yêu cầu thêm một đối tượng kiểu tài nguyên R_3. Ma trận Request thay đổi như trong Hình 7.4b. Hệ thống bây giờ rơi vào trạng thái bế tắc. Mặc dù có thể thu hồi tài nguyên đang cấp phát cho P_0, số lượng các tài nguyên sẵn có không đủ đáp ứng các yêu cầu của các tiến trình khác. Do đó, bế tắc xảy ra và liên quan đến các tiến trình P_1, P_2, P_3 và P_4.

	Allocation	Request	Available
	$R_0R_1R_2$	$R_0R_1R_2$	$R_0R_1R_2$
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	

(a)

	Request
	$R_0R_1R_2$
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

(b)

Hình 7.4. Ví dụ về thuật toán Tránh bế tắc

7.5.3. Sử dụng thuật toán Phát hiện bế tắc

Thời điểm thực thi thuật toán phát hiện phụ thuộc vào hai yếu tố là mức độ xuất hiện và ảnh hưởng của bế tắc. Nếu bế tắc xảy ra thường xuyên, hệ thống phải liên tục thực thi thuật toán phát hiện. Khi bế tắc, tài nguyên cấp phát cho những tiến trình bế tắc bị lãng phí và theo thời gian, số lượng các tiến trình bế tắc gia tăng. Trong trường hợp cực đoan, hệ thống có thể kiểm tra bế tắc xuất hiện chưa mỗi khi có yêu cầu không được đáp ứng. Trong trường hợp này, có thể xác định không chỉ các tiến trình liên quan tới bế tắc, mà cả tiến trình gây ra bế tắc. Tất nhiên, thực hiện thường xuyên thuật toán phát hiện bế tắc gây ra chi phí phụ trội lớn. Giải pháp ít tốn kém hơn là thực hiện ở tần suất thấp. Ví dụ, một lần một giờ, hoặc khi hiệu suất sử dụng CPU giảm xuống dưới một ngưỡng nào đó (bế tắc làm giảm thông lượng hệ

thống cũng như hiệu suất sử dụng CPU). Nếu thực thi thuật toán phát hiện tại các thời điểm ngẫu nhiên, sẽ có thể có nhiều chu trình bế tắc trong đồ thị tài nguyên. Khi đó, không xác định được tiến trình nào là nguyên nhân gây ra bế tắc.

7.6. KHẮC PHỤC BẾ TẮC

Làm thế nào để khắc phục bế tắc? Giải pháp đầu tiên là thông báo cho người quản trị và người quản trị có trách nhiệm xử lý. Giải pháp thứ hai là để cho hệ thống tự động phục hồi. Có hai lựa chọn để phá vỡ bế tắc:

– Thứ nhất, loại bỏ một số tiến trình để phá vỡ chu trình.

– Thứ hai, thu hồi tài nguyên của các tiến trình bế tắc.

7.6.1. Chấm dứt tiến trình

Hệ thống thu hồi tài nguyên của các tiến trình bị chấm dứt. Có hai giải pháp chấm dứt:

1. Chấm dứt tất cả các tiến trình bế tắc: Phá vỡ toàn bộ chu trình bế tắc, nhưng phí tổn khá lớn vì nhiều tiến trình bế tắc có thể đã thực thi trong thời gian dài, và một phần kết quả thực thi bị loại bỏ sau đó có thể phải thực hiện lại.
2. Lần lượt chấm dứt từng tiến trình cho tới khi phá vỡ chu trình bế tắc. Cứ sau khi kết thúc một tiến trình, hệ thống kiểm tra xem bế tắc còn hay không.

Việc chấm dứt tiến trình có thể không đơn giản. Nếu chấm dứt một tiến trình đang trong quá trình cập nhật file có thể gây lỗi cho file. Nếu sử dụng phương pháp (2), hệ thống phải xác định thứ tự chấm dứt tiến trình (hoặc nhóm tiến trình) để phá vỡ bế tắc. Quyết định này phụ thuộc vào chính sách của hệ thống, tương tự như vấn đề điều phối CPU. Nói chung, đây là vấn đề kinh tế, nên chấm dứt tiến trình với chi phí thấp nhất, do đó phải tính đến:

1. Độ ưu tiên của tiến trình.
2. Tiến trình thực thi được bao lâu, cần thêm bao lâu để kết thúc?
3. Tiến trình sử dụng bao nhiêu tài nguyên và là những kiểu tài nguyên gì (liệu có dễ dàng chiếm đoạt không)?

4. Tiến trình cần thêm bao nhiêu tài nguyên để hoàn thành?
5. Có bao nhiêu tiến trình cần bị chấm dứt?
6. Liệu tiến trình có đòi hỏi tương tác với người dùng hay là ở dạng lô?

7.6.2. Chiếm đoạt tài nguyên

Có thể chấm dứt bế tắc bằng cách lần lượt thu hồi tài nguyên của một số tiến trình và chuyển chúng cho những tiến trình khác cho tới khi chu trình bế tắc bị phá vỡ. Nếu sử dụng cách này, có 3 vấn đề cần xét:

1. **Lựa chọn nạn nhân:** Tiến trình bị thu hồi tài nguyên được gọi là nạn nhân. Trong quá trình thực hiện, hệ thống quyết định thứ tự thu hồi để giảm thiểu chi phí phụ trội đến mức thấp nhất. Các yếu tố phải tính đến như số lượng tài nguyên mà tiến trình đang nắm giữ, thời gian thực thi của tiến trình.
2. **Quay lui:** Rõ ràng tiến trình "nạn nhân" không thể tiếp tục chạy bình thường do thiếu tài nguyên. Hệ thống phải đưa tiến trình này quay lại một trạng thái an toàn, để sau này có thể khôi phục lại. Nhưng khó xác định thế nào là trạng thái an toàn? Do vậy, cách giải quyết đơn giản nhất là quay lui toàn bộ (chấm dứt tiến trình và sau đấy khởi động lại). Dĩ nhiên, nếu chỉ quay lại trạng thái vừa đủ để chấm dứt bế tắc sẽ hiệu quả hơn nhiều. Nhưng khi đó hệ thống phải duy trì nhiều thông tin trạng thái của các tiến trình đang chạy.
3. **Chết đói:** Hệ thống phải đảm bảo không xảy ra hiện tượng chết đói (có tiến trình luôn luôn bị chọn làm "nạn nhân"). Trong hệ thống đặt yếu tố kinh tế lên hàng đầu, có thể có những tiến trình luôn bị chọn làm nạn nhân và không thể thực thi. Hệ thống có thể giải quyết bằng cách giới hạn số lần bị chọn làm "nạn nhân" của một tiến trình.

7.7. NHẬN XÉT

Trạng thái bế tắc xuất hiện khi nhiều tiến trình đang chờ đợi vô định, một sự kiện được một tiến trình khác cũng đang trong trạng thái đợi gây ra. Về mặt nguyên tắc, có ba phương pháp xử lý bế tắc:

1. Buộc hệ thống không bao giờ rơi vào trạng thái bế tắc.
2. Cho phép hệ thống rơi vào trạng thái bế tắc, sau đó khắc phục.
3. Bỏ qua mọi vấn đề, giả định bế tắc không bao giờ xuất hiện.

Điều kiện cần để bế tắc xuất hiện là: Độc quyền truy xuất; giữ và chờ; không chiếm đoạt và vòng đợi. Để ngăn chặn bế tắc, hệ thống chỉ cần ngăn cản sự xuất hiện của ít nhất một trong bốn điều kiện trên. Giải pháp tránh bế tắc khác là, tiến trình báo trước về cách sử dụng tài nguyên của mình. Thuật toán Ngân hàng cần phải biết số lượng lớn nhất mỗi lớp tài nguyên tiến trình yêu cầu. Nếu hệ thống không có phương pháp nào để bảo đảm bế tắc không xuất hiện, thì cần phải phát hiện và khắc phục bế tắc. Thuật toán phát hiện được sử dụng để xác định bế tắc xuất hiện chưa, nếu có, hệ thống phải khắc phục bằng cách loại bỏ hoặc thu hồi tài nguyên của tiến trình bế tắc. Trong hệ thống chọn lựa "nạn nhân" dựa trên các yếu tố chi phí, cần chú ý đến hiện tượng "chết đói".

CÂU HỎI ÔN TẬP

1. Trình bày mô hình bế tắc.
2. Trình bày các điều kiện xảy ra bế tắc và các giải pháp tương ứng.
3. Trình bày các giải pháp ngăn chặn bế tắc.
4. Trình bày các phương pháp tránh và phát hiện bế tắc.

cuu duong than cong . com

Chương 8
QUẢN LÝ THIẾT BỊ

HĐH kiểm soát hoạt động các thiết bị vào/ra gắn với máy tính bằng cách phát lệnh điều khiển thiết bị; phát hiện, xử lý ngắt và quản lý lỗi. HĐH cung cấp giao diện đơn giản giúp người sử dụng và lập trình viên sử dụng thiết bị dễ dàng. Để dễ ghép nối các thiết bị mới, giao diện giữa hệ thống với các thiết bị nên giống nhau. Tốc độ vào/ra nhỏ hơn tốc độ tính toán (CPU) hàng triệu lần và tỷ lệ này không có xu hướng giảm. Do vậy, các hệ thống máy tính hiện đại cố gắng thực hiện xen kẽ thao tác vào/ra với thao tác tính toán để tăng hiệu suất tổng thể của hệ thống. Chương này trình bày cách thức HĐH quản lý thiết bị vào/ra; cách thức HĐH quản lý thiết bị, các phương pháp quản lý cơ bản, cách sử dụng bộ đệm để tăng hiệu suất vào/ra và cấu trúc tổng quát của trình điều khiển thiết bị.

8.1. NGUYÊN LÝ HOẠT ĐỘNG

Kỹ sư điện tử coi thiết bị vào/ra là tổ hợp vi mạch, dây dẫn, nguồn điện, motor. Lập trình viên "giao tiếp" với thiết bị thông qua giao diện phần mềm và đây là khía cạnh mà chúng ta quan tâm.

8.1.1. Phân loại thiết bị vào/ra

Thiết bị vào/ra có thể được chia thành hai loại là thiết bị hướng khối và thiết bị hướng ký tự. Thiết bị hướng khối lưu trữ thông tin trong các khối có kích thước cố định (ví dụ ổ đĩa từ), thường nằm trong khoảng từ 512 tới 32768 byte. Tính chất của thiết bị hướng khối là có thể đọc hoặc ghi các khối độc lập với nhau. Nếu tiếp tục phân loại "mịn" hơn, có thể phân thiết bị hướng khối ra hai kiểu là kiểu có thể đánh địa chỉ và kiểu không thể đánh địa chỉ. Ổ đĩa là thiết bị có thể đánh địa chỉ cho từng khối, vì trục quay có

khả năng dịch tới bất kỳ vị trí nào. Một kiểu thiết bị vào/ra khác là hướng ký tự (thiết bị tạo ra hoặc nhận dãy từng ký tự). Không thể đánh địa chỉ và không có quá trình dịch chuyển trong kiểu thiết bị này. Máy in, card mạng, con chuột (để trỏ) và hầu hết các thiết bị không hoạt động giống ổ đĩa đều có thể xem như thiết bị hướng ký tự. Phân loại này chưa thực sự đầy đủ, vì một số thiết bị không nằm trong loại nào. Chẳng hạn, đồng hồ hay màn hình với bộ nhớ ánh xạ. Tuy nhiên, mô hình thiết bị hướng khối và hướng ký tự khá tổng quát để có thể làm nền tảng giúp HĐH tạo ra cơ chế quản lý hoạt động của thiết bị. Sự chênh lệch rất lớn giữa tốc độ các thiết bị khiến việc thiết kế phần mềm khó khăn hơn rất nhiều. Mặc dù tốc độ hoạt động của thiết bị được cải thiện, nhưng tốc độ này không tăng nhanh bằng tốc độ CPU.

8.1.2. Bộ điều khiển thiết bị

Thiết bị vào/ra gồm hai phần cơ bản là phần cơ khí và phần điện tử, được thiết kế độc lập với nhau. Phần điện tử được gọi là device controller (bộ điều khiển) hoặc adapter (bộ điều hợp). Trên máy tính cá nhân (PC), bộ điều khiển thường là bản mạch in có thể cắm vào khe cắm mở rộng. Phần cơ khí là thiết bị.

cuu duong than cong . com

Bảng mạch của bộ điều khiển thường có phần kết nối với thiết bị. Nhiều bộ điều khiển có thể quản lý 2, 4 hoặc 8 thiết bị giống hệt nhau. Thường giao diện giữa bộ điều khiển và thiết bị được chuẩn hóa, hoặc theo chuẩn chính thức (IEEE, ANSI, ISO), hoặc chuẩn không chính thức nhưng được thừa nhận rộng rãi. Ví dụ, phần lớn các sản phẩm ổ đĩa đều tuân theo chuẩn IDE hay SCSI.

Hình 8.1. Bộ điều khiển SICI

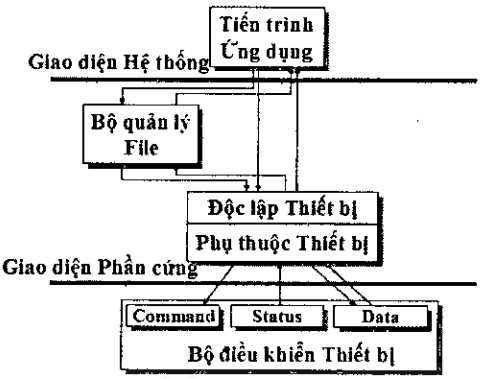
cuu duong than cong . com

Giao diện giữa bộ điều khiển và thiết bị là giao diện ở mức thấp. Ở mức cao có thể coi ổ đĩa được chia thành nhiều track, track có 256 sector và sector chứa 512 byte. Tuy nhiên, ở mức vật lý thấp nhất (bên trong ổ đĩa) là dòng các bit, bắt đầu với một tiêu đề preamble (chuỗi bit đánh dấu sự bắt đầu), sau đó là 4096 bit ứng với một sector và cuối cùng là các bit kiểm tra (checksum), các bit này còn được gọi là mã sửa lỗi (Error Corecting Code – ECC). Phần tiêu đề preamble được ghi trong quá trình định dạng đĩa, chứa

số hiệu sector và cylinder, kích thước sector và các thông tin phục vụ cho mục đích đồng bộ. Công việc của bộ điều khiển ổ đĩa là chuyển dòng bit sang khối byte và sửa những lỗi có thể. Đầu tiên các bit lần lượt được ghép lại thành một khối byte và lưu tạm trong bộ nhớ đệm của bộ điều khiển. Sau đó, bộ điều khiển thực hiện kiểm tra giá trị các bit sửa lỗi và nếu xác định không có lỗi thì khối byte được chuyển vào bộ nhớ trong. Bộ điều khiển màn hình cũng là một dạng thiết bị hoạt động theo chuỗi bit liên tiếp. Bộ điều khiển đọc byte (là mã ký tự cần hiển thị) từ bộ nhớ và sau đó tạo ra tín hiệu điều chỉnh dòng CRT in ra màn hình. Bộ điều khiển cũng tạo ra tín hiệu để yêu cầu dòng CRT quét lại theo chiều dọc sau khi đã quét hết một dòng, và quét lại điểm đầu tiên của màn hình sau khi quét xong toàn bộ màn hình. Nếu không có bộ điều khiển, HĐH phải tự thực hiện việc quét màn hình. Nhưng bây giờ, HĐH chỉ cần khởi tạo bộ điều khiển với một vài tham số (số lượng ký tự hay số điểm ảnh trên một dòng và số lượng dòng trên màn hình). Sau khi khởi động, bộ điều khiển trực tiếp kiểm soát thiết bị.

8.2. CHIẾN LƯỢC QUẢN LÝ THIẾT BỊ

cuu duong than cong . com



cuu duong than cong . com

Hình 8.2. Bộ phận quản lý thiết bị của HĐH

Trong HĐH, bộ phận quản lý thiết bị điều khiển thiết bị và cung cấp cho tiến trình sử dụng mức trừu tượng thiết bị thấp nhất. Có nhiều phương pháp trừu tượng hóa khác nhau. Phương pháp đơn giản nhất là vào/ra trực tiếp (tiến trình trực tiếp chuyển dữ liệu tới các thanh ghi trong bộ điều khiển thiết bị). Nếu có thêm cơ chế thăm dò (polling), phần mềm điều khiển thiết bị sẽ kiểm tra thanh ghi trạng thái trong bộ điều khiển để xác định yêu cầu

nào đó đã hoàn thành chưa. Tuy nhiên, nếu hệ thống sử dụng ngắt, tiến trình ứng dụng không phải tự mình kiểm tra xem thiết bị đã thực hiện xong công việc chưa. Phương pháp thứ hai là vào/ra qua ánh xạ bộ nhớ. Bộ nhớ của thiết bị được ánh xạ vào một phần không gian bộ nhớ hệ thống. Cơ chế cuối cùng là sử dụng cơ chế truy cập bộ nhớ trực tiếp (DMA – Direct Memory Access), tức là có thiết bị chịu trách nhiệm sao chép dữ liệu từ bộ nhớ đến thanh ghi trong bộ điều khiển thiết bị mà không cần CPU điều khiển.

8.2.1. Tổ chức hệ thống vào/ra

Trong HĐH hiện đại, có hai phương pháp cài đặt bộ phận quản lý thiết bị: hoặc thông qua sự tương tác giữa trình điều khiển thiết bị (device driver) với bộ quản lý ngắt (cơ chế vào/ra có ngắt), hoặc hoàn toàn bằng trình điều khiển thiết bị nếu không sử dụng ngắt (cơ chế vào/ra thăm dò). Hình 8.3 minh họa các thành phần trong cả hai cơ chế này. Nếu muốn sử dụng thiết bị, tiến trình ứng dụng gửi lệnh cần thực hiện (cùng với dữ liệu nếu có) tới trình điều khiển thiết bị.

Trình điều khiển thực hiện hai việc sau:

cuu duong than cong . com

1. Cài đặt API – giao diện để ứng dụng sử dụng khi muốn tương tác với thiết bị.
2. Dịch những lệnh mức cao (từ ứng dụng gửi xuống) thành các lệnh mức thấp (phụ thuộc vào từng thiết bị cụ thể).

Để thuận tiện, giao diện giữa các trình điều khiển nên giống nhau, vì khi đó, người lập trình nếu biết sử dụng một thiết bị sẽ dễ dàng làm việc với các thiết bị khác. Tuy nhiên, do chịu trách nhiệm quản lý từng thiết bị đặc thù, nên trình điều khiển phải có khả năng đưa ra lệnh cho từng thiết bị và rõ ràng các lệnh này phụ thuộc vào bộ điều khiển thiết bị.

cuu duong than cong . com

8.2.2. Vào/ra trực tiếp qua thăm dò

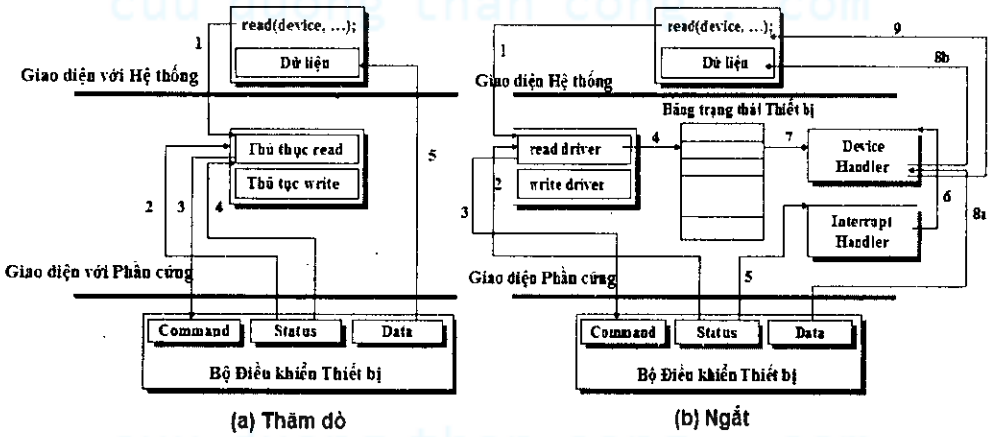
Trong cơ chế vào/ra trực tiếp, CPU chịu trách nhiệm chuyển dữ liệu giữa bộ nhớ chính và thanh ghi trong bộ điều khiển thiết bị. Trình điều khiển đọc cờ busy-done trên thiết bị, hoặc sử dụng ngắt để "thăm dò" thiết bị hoàn thành nhiệm vụ chưa. Ta trình bày hai thao tác cơ bản nhất của CPU là khởi động thiết bị và thăm dò thanh ghi trạng thái.

Hình 8.3a minh họa các bước chuyển dữ liệu từ thiết bị vào bộ nhớ trong:

1. Tiến trình ứng dụng yêu cầu đọc dữ liệu.
2. Trình điều khiển kiểm tra thanh ghi trạng thái xem thiết bị có bận không. Nếu bận, trình điều khiển phải chờ.
3. Trình điều khiển khởi tạo thiết bị bằng cách ghi lệnh cần thực hiện (read) vào thanh ghi lệnh của bộ điều khiển.
4. Trình điều khiển kiểm tra thiết bị đã thực hiện xong lệnh đọc hay chưa bằng cách định kỳ đọc thanh ghi trạng thái.
5. Trình điều khiển sao chép giá trị (các) thanh ghi dữ liệu trong bộ điều khiển vào không gian nhớ của tiến trình.

Thao tác ghi dữ liệu được thực hiện tương tự. Trong cơ chế vào/ra trực tiếp qua thăm dò, yêu cầu vào/ra được thực hiện thông qua tương tác với trình điều khiển, điều này "che dấu" toàn bộ hoạt động của phần cứng của bộ điều khiển. Nhược điểm của cách tiếp cận này là hiệu suất sử dụng CPU thấp do CPU phải thường xuyên kiểm tra tất cả các thiết bị xem đã hoàn thành thao tác vào/ra chưa. Có thể khắc phục điều này bằng cách sử dụng ngắt.

cuu duong than cong . com

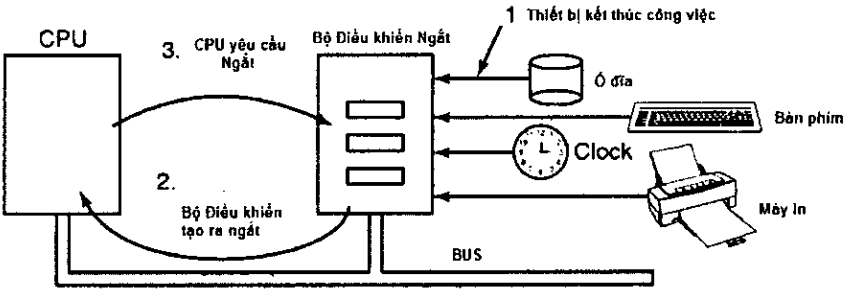


Hình 8.3. Cài đặt bộ điều khiển thiết bị

cuu duong than cong . com

8.2.3. Vào/ra sử dụng ngắt

Đối với hệ thống máy tính cá nhân, cấu trúc ngắt được thể hiện trong Hình 8.4. Ngắt hoạt động như sau: Khi thực hiện xong công việc được giao, thiết bị gửi tín hiệu tới chip điều khiển ngắt gắn trên bo mạch chủ. Phụ thuộc vào tín hiệu đến từ dây dẫn nào, bộ điều khiển ngắt xác định được thiết bị gây ra ngắt.



Hình 8.4. Cơ chế hoạt động của ngắt

Nếu tại thời điểm nào đó có đúng một ngắt xuất hiện, bộ điều khiển sẽ xử lý ngắt ngay lập tức. Nhưng khi xuất hiện nhiều ngắt, thì hệ thống ưu tiên xử lý ngắt có độ ưu tiên cao trước. Bộ điều khiển ngắt gán số thứ tự cho mỗi đường địa chỉ xác định thiết bị và tạo ra tín hiệu ngắt tương ứng gửi tới CPU. Tín hiệu ngắt khiến CPU tạm dừng công việc đang thực hiện để chuyển sang xử lý ngắt. Số hiệu đường địa chỉ được sử dụng làm chỉ số tra cứu trong bảng vector ngắt để xác định địa chỉ đầu tiên của đoạn chương trình xử lý ngắt. Bảng vector ngắt có thể được ghi cứng trong máy, hoặc có thể nằm trong bộ nhớ, và hệ thống sử dụng thanh ghi đặc biệt trong CPU trỏ tới địa chỉ bảng vector ngắt. Ngay sau khi bắt đầu chạy, thủ tục xử lý ngắt sẽ ghi giá trị xác định nào đó trên một cổng của bộ điều khiển ngắt vào/ra để xác nhận ngắt đã được xử lý, khi đó bộ điều khiển có thể tạo ra ngắt mới. CPU chỉ gửi tín hiệu xác nhận cho tới khi có thể xử lý được ngắt tiếp theo, và do đó tránh được xung đột khi nhiều ngắt xảy ra đồng thời.

Hệ thống phải ghi lại một số thông tin trước khi khởi động thủ tục xử lý ngắt. Những thông tin nào được ghi và ghi vào đâu phụ thuộc vào kiến trúc CPU. Ít nhất phải ghi lại thanh ghi PC để sau khi xử lý xong ngắt còn quay lại chương trình cũ. Nhưng cũng có một số kiến trúc phức tạp yêu cầu phải ghi lại tất cả các thanh ghi.

Nếu lưu vào thanh ghi, thì các ngắt liên tiếp nhau có thể ghi đè lên thanh ghi và do đó dữ liệu có thể bị mất. Do vậy, phần lớn CPU ghi thông tin vào ngăn xếp. Ở đây có thể lưu vào ngăn xếp của HĐH hoặc ngăn xếp của tiến trình người dùng. Một vấn đề khác là, CPU hiện đại đều theo cấu trúc đường ống (pipeline), hoặc cấu trúc siêu vô hướng (superscalar). Trong các hệ thống máy tính trước kia, sau khi kết thúc một chỉ thị, chương trình

hoặc phần cứng kiểm tra xem có xuất hiện ngắt hay không. Nếu có, giá trị PC được ghi vào ngăn xếp và chương trình xử lý ngắt bắt đầu thực thi. Sau khi xử lý xong ngắt, PC được khôi phục từ ngăn xếp để khôi phục tiến trình bị gián đoạn trước kia. Chế độ kiểu này ngầm định rằng, nếu ngắt xuất hiện ngay sau chỉ thị A, tất cả chỉ thị trước A đã hoàn thành và không chỉ thị nào sau A được thực thi. Trong kiến trúc đường ống, CPU có thể thực thi cùng lúc nhiều chỉ thị. Giả sử ngắt xuất hiện trong khi đường ống đang đầy, các chỉ thị đang ở trong các trạng thái thực hiện khác nhau. Khi ngắt xuất hiện, giá trị PC không phản ánh chính xác những chỉ thị nào đã bắt đầu thi hành và chỉ thị nào chưa thi hành. PC chỉ là địa chỉ của chỉ thị kế tiếp sẽ được lấy và đưa vào đường ống, chứ không phải địa chỉ của chỉ thị được xử lý trong CPU. Do vậy, phải định nghĩa rõ ràng ranh giới giữa chỉ thị đã và chỉ thị chưa thực thi. Tuy nhiên, phần cứng không thể xác định được ranh giới này. Do đó, khi thực hiện xong ngắt chỉ với địa chỉ trong thanh ghi PC, HĐH không thể làm đầy lại đường ống như cũ. Cần xác định chỉ thị cuối cùng được thực thi, tuy nhiên chức năng này thường rất phức tạp, vì cần phân tích trạng thái máy tính. Trên kiến trúc siêu vô hướng vấn đề còn phức tạp hơn, vì các chỉ thị có thể được thực thi không theo thứ tự, nên không thể định nghĩa rõ ràng ranh giới giữa chỉ thị đã thực hiện và chưa thực hiện. Chẳng hạn, có thể chỉ thị 1, 2, 3, 5 và 8 đã thực hiện và chỉ thị 4, 6, 7, 9 chưa thực hiện. Bên cạnh đó, thanh ghi PC có thể trỏ tới chỉ thị 9, 10 hoặc 11.

Ngắt nghiêm ngặt (precise interrupt) đặt máy tính trong một trạng thái rõ ràng và có 4 thuộc tính:

1. Thanh ghi PC trỏ tới một chỉ thị xác định.
2. Tất cả các chỉ thị trước chỉ thị do PC trỏ tới đã hoàn toàn thực thi xong.
3. Mọi chỉ thị sau chỉ thị được PC trỏ tới đều chưa thực thi.
4. Trạng thái thực thi của chỉ thị được PC trỏ tới hoàn toàn xác định.

Chú ý rằng, các chỉ thị đằng sau chỉ thị do PC trỏ tới không bị cấm thực thi. Tuy nhiên, nếu chỉ thị này gây ra thay đổi trên thanh ghi hoặc bộ nhớ, thì các thay đổi này cần phải được khôi phục lại trạng thái ban đầu trước khi ngắt xảy ra.

Ngắt không đảm bảo các yêu cầu trên được gọi là ngắt không nghiêm ngặt (imprecise interupt). Kiểu ngắt này làm người viết HĐH gặp rất nhiều

khó khăn, do phải xác định được chỉ thị nào đã và chỉ thị nào đang thực thi. Máy tính với ngắt không nghiêm ngặt thường phải ghi lại rất nhiều thông tin trạng thái vào ngăn xếp, để HĐH có thể xác định được chỉ thị nào đang thực thi trong giai đoạn nào. Phải ghi lượng lớn thông tin vào bộ nhớ mỗi khi gặp ngắt làm tăng thời gian xử lý ngắt. Điều này dẫn tới nghịch lý là, các CPU siêu vô hướng tốc độ cực cao đôi khi không thích hợp với các công việc thời gian thực do xử lý ngắt quá chậm.

Một số máy tính được thiết kế để vừa có ngắt nghiêm ngặt, vừa có ngắt không nghiêm ngặt. Một số kiến trúc siêu vô hướng, như Pentium Pro và các sản phẩm thuộc dòng này sử dụng ngắt nghiêm ngặt, cho phép các chương trình viết trên nền 386, 486 và Pentium I hoạt động chính xác (superscalar chỉ bắt đầu xuất hiện trong Pentium Pro; Pentium I chỉ có 2 đường ống). Khi đó, mạch logic của bộ điều khiển ngắt bên trong CPU cực kỳ phức tạp. Bộ điều khiển ngắt phải đảm bảo khi xuất hiện ngắt, thì mọi chỉ thị ở trước một mốc nào đó phải được thực thi xong và tất cả chỉ thị phía sau mốc không gây ảnh hưởng gì tới trạng thái máy. Ở đây, giá phải trả không phải là thời gian mà là diện tích chip và sự phức tạp khi thiết kế. Nếu không cần ngắt nghiêm ngặt (để tương thích ngược với các ứng dụng cũ), thì diện tích này trên bo mạch có thể được sử dụng cài đặt thêm cache. Ngược lại, ngắt không nghiêm ngặt làm HĐH phức tạp và chậm hơn. Do đó, thật sự khó xác định được phương pháp nào ưu việt hơn.

Với ngắt, trình điều khiển không phải thăm dò xem thiết bị đã hoàn thành nhiệm vụ hay chưa. Khi kết thúc công việc, bộ điều khiển báo lại cho trình điều khiển thiết bị. Bộ quản lý thiết bị trong cơ chế vào/ra sử dụng ngắt có 3 thành phần:

- Bảng trạng thái thiết bị.
- Trình xử lý ngắt.
- Bộ phận thực hiện của thiết bị.

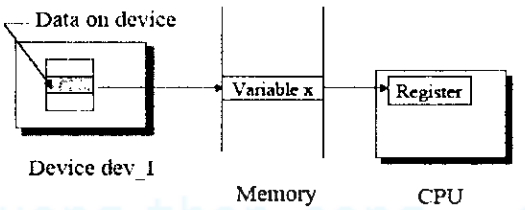
Hình 8.3b minh họa các bước đọc dữ liệu trên hệ thống sử dụng ngắt:

1. Tiến trình ứng dụng yêu cầu đọc dữ liệu.
2. Trình điều khiển kiểm tra thanh ghi trạng thái xem thiết bị có rỗi hay không. Nếu thiết bị bận, trình điều khiển đợi.

3. Trình điều khiển ghi lệnh khởi tạo vào thanh ghi lệnh của bộ điều khiển.
4. Khi hoàn thành công việc, trình điều khiển ghi vào bảng trạng thái thiết bị các thông tin liên quan đến thao tác. Bảng này chứa thông tin về tất cả các thiết bị trong hệ thống, mỗi hàng ứng với một thiết bị cụ thể. Trình điều khiển ghi lại một số thông tin về thiết bị mình quản lý. Các thông tin này có thể là địa chỉ trả về sau khi thao tác thực hiện xong và có thể là một số tham số đặc biệt khác. Kế tiếp, trình điều khiển kết thúc và chuyển quyền điều khiển cho bộ điều phối.
5. Khi kết thúc công việc được giao, thiết bị tạo ra ngắt gửi đến CPU và trình xử lý ngắt tương ứng được thực thi.
6. Trình xử lý ngắt xác định thiết bị nào tạo ngắt. Sau đó sẽ rẽ nhánh đến trình xử lý ngắt của thiết bị.
7. Trình xử lý ngắt đọc bảng trạng thái thiết bị để lấy thông tin về thao tác vừa thực thi.
8. Trình xử lý ngắt chuyển dữ liệu từ (các) thanh ghi dữ liệu trong bộ điều khiển vào không gian nhớ của người sử dụng.
9. Trình xử lý ngắt chuyển quyền điều khiển cho tiến trình ứng dụng.

cuu duong than cong . com

```
. . .                          . . .
read(dev_I, "%d", x);          startRead(dev_I, "%d", x);
y = f(x)                       . . .
. . .                          While(stillReading()) ;
                               y = f(x)
                               . . .
```



cuu duong than cong . com

Hình 8.5. Xử lý qua ngắt

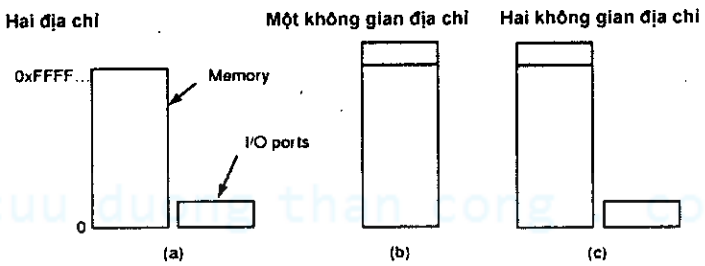
Thao tác ghi dữ liệu diễn ra tương tự. Với tiến trình ứng dụng, thao tác này tương tự lời gọi hệ thống. Tuy nhiên, thời gian ghi có thể lớn hơn rất nhiều, đặc biệt trong hệ thống thăm dò. Thời gian này phụ thuộc vào thời gian tính toán, tốc độ thực hiện vào/ra và khoảng thời gian thăm dò thiết bị. Đối với hệ thống thăm dò, cần tính thêm khoảng thời gian từ khi thiết bị thực sự hoàn thành xong công việc cho tới khi tiến trình ứng dụng phát hiện

ra sự kiện này để tiếp tục thực thi công việc của mình. Do đó, bằng cách tương tác với thiết bị thông qua giao diện phần mềm, HĐH cho phép tiến trình thực hiện tính toán trên CPU trong khi tiến trình khác sử dụng thiết bị. Mỗi tiến trình vẫn được đảm bảo thực thi một cách tuần tự, cho dù có gián đoạn. Các thao tác vào/ra có tính chất tuần tự, nghĩa là khi lập trình viên sử dụng lệnh **read** trong chương trình thì lệnh **read** phải thực hiện xong trước khi thực thi các lệnh sau. Ví dụ, khi tiến trình thực hiện đoạn chương trình minh họa trên Hình 8.5, t_0 là thời điểm sau khi tiến trình sử dụng lời gọi hệ thống để đọc từ thiết bị, nhưng lời gọi này chưa thực hiện xong. Nếu hàm **f** với tham số **x** được tính trước khi thao tác đọc thực hiện, thì tiến trình sẽ sử dụng giá trị **x** cũ. Để tránh tình trạng này, sau khi gọi **read**, tiến trình phải bị phong tỏa cho đến khi **read** hoàn thành.

Từng tiến trình riêng lẻ không nhận được ích lợi của việc xen kẽ thao tác tính toán với thao tác vào/ra, nhưng hiệu suất hệ thống tổng thể tăng đáng kể, vì CPU được tận dụng tối đa (khi tiến trình đang thực thi phải đợi vào/ra và bị phong tỏa, tiến trình khác sử dụng CPU).

8.2.4. Vào/ra qua ánh xạ bộ nhớ

Để tương tác với thiết bị phải có chỉ thị thao tác trên thanh ghi của bộ điều khiển trong tập chỉ thị ngôn ngữ máy, để HĐH có thể ra lệnh cho thiết bị (ghi lệnh tương ứng lên thanh ghi lệnh) và xác định trạng thái thiết bị (đọc thanh ghi trạng thái).



Hình 8.6. Giao tiếp giữa CPU và bộ điều khiển

Có hai cách để CPU trao đổi dữ liệu với bộ điều khiển thiết bị. Thứ nhất, mỗi thanh ghi của bộ điều khiển được gán định danh là số nguyên, gọi là số hiệu cổng vào/ra (I/O port number). CPU sử dụng chỉ thị **IN REG, PORT** để đọc thanh ghi có định danh **PORT** vào thanh ghi **REG** trong CPU.

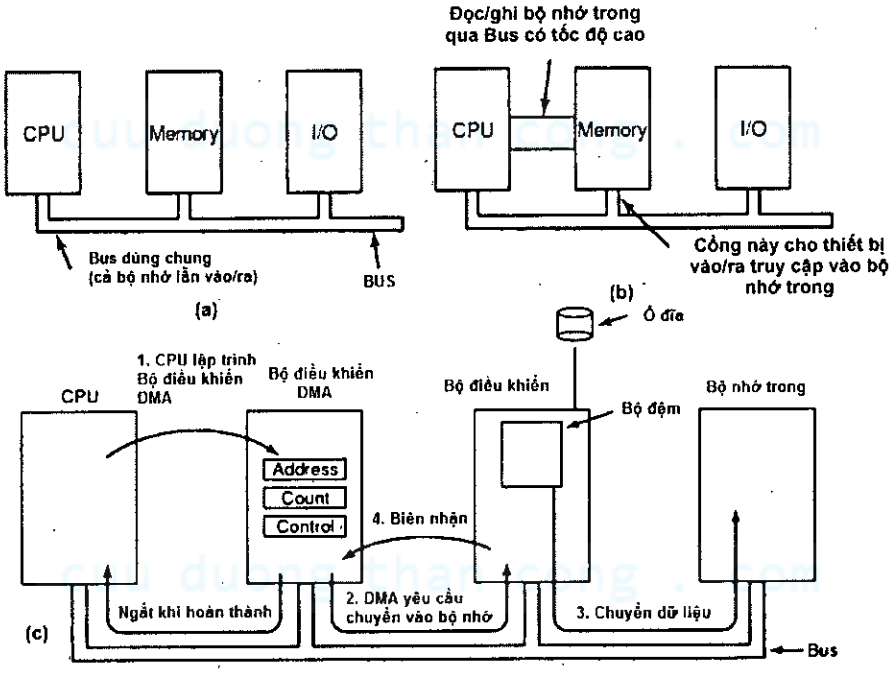
Các máy tính thời kỳ đầu đều làm việc theo cách này. Trong Hình 8.6a, không gian địa chỉ bộ nhớ độc lập với không gian vào/ra của bộ điều khiển. Chỉ thị **IN R0, 4** (chuyển nội dung của cổng I/O có số hiệu 4 vào thanh ghi **R0**) khác với chỉ thị **MOV R0, 4** (đưa nội dung ô nhớ thứ 4 trong bộ nhớ chính vào **R0**). Giá trị 4 trong hai chỉ thị trên ứng với hai không gian địa chỉ khác nhau. Giải pháp thứ hai, ánh xạ tất cả các thanh ghi vào không gian bộ nhớ chính (Hình 8.6b). Mỗi thanh ghi được gán một địa chỉ duy nhất và không trùng với bất kỳ địa chỉ ô nhớ nào trong bộ nhớ chính. Hệ thống như vậy được gọi là vào/ra qua ánh xạ bộ nhớ. Thông thường, địa chỉ các thanh ghi nằm ở đầu không gian địa chỉ. Giải pháp thứ ba lai giữa hai giải pháp trên (Hình 8.6c), thực hiện ánh xạ bộ nhớ vào/ra (nằm trên bộ điều khiển) vào không gian bộ nhớ trong, nhưng lại gán định danh cổng vào/ra cho các thanh ghi điều khiển. Pentium sử dụng kiến trúc này với vùng địa chỉ 604KB tới 1MB dùng cho bộ nhớ thiết bị và số hiệu cổng từ 0 đến 64KB. Hệ thống này hoạt động như sau: CPU đặt địa chỉ cần đọc (có thể nằm ở bộ nhớ trong hay bộ nhớ vào/ra) lên bus địa chỉ và phát lệnh **READ** trên bus điều khiển. Có thể sử dụng thêm một tín hiệu khác để xác định từ cần đọc nằm trong bộ nhớ vào/ra hay bộ nhớ trong. Nếu sử dụng chung không gian bộ nhớ (như Hình 8.6b), cả bộ nhớ trong lẫn tất cả thiết bị vào/ra đều phải so sánh địa chỉ trên bus địa chỉ với khoảng địa chỉ của mình. Nếu địa chỉ yêu cầu nằm trong phạm vi quản lý, bộ phận tương ứng sẽ trả lời yêu cầu.

Hai phương pháp trên có ưu, nhược điểm riêng. Ưu điểm vào/ra qua ánh xạ bộ nhớ là đơn giản công việc của người lập trình. Thứ nhất, trong hệ thống phải sử dụng chỉ thị vào/ra đặc biệt để đọc/ghi thanh ghi trên bộ điều khiển thiết bị, lập trình viên phải sử dụng mã Assembly vì trong ngôn ngữ C/C++ không có lệnh ứng với chỉ thị **IN/OUT**. Ngược lại, với vào/ra qua ánh xạ bộ nhớ, thanh ghi trên bộ điều khiển giống bất kỳ ô nhớ nào trong bộ nhớ và có thể đánh địa chỉ như mọi biến khác. Do đó, trong hệ thống vào/ra qua ánh xạ bộ nhớ, có thể viết trình điều khiển thiết bị bằng ngôn ngữ C. Thứ hai, với vào/ra qua ánh xạ bộ nhớ, không cần sử dụng cơ chế bảo vệ đặc biệt nào để hạn chế tiến trình người dùng thực hiện vào/ra. HĐH chỉ cần ngăn không đặt không gian địa chỉ chứa thanh ghi của bộ điều khiển vào không gian bộ nhớ tiến trình người dùng. Nếu thanh ghi điều khiển của các thiết bị khác nhau nằm trên các trang nhớ khác nhau, thì HĐH có thể hạn chế tiến trình người dùng chỉ được sử dụng một số thiết bị nhất định. Trình điều khiển các thiết bị nằm trong các không gian địa chỉ khác nhau, điều này

không chỉ làm giảm kích thước kernel mà còn tránh xung đột giữa các bộ điều khiển. Thứ ba, với vào/ra qua ánh xạ bộ nhớ, mọi chỉ thị nếu tham chiếu được tới bộ nhớ cũng sẽ tham chiếu được thanh ghi điều khiển. Ví dụ, nếu chỉ thị **TEST** kiểm tra từ nhớ có bằng 0 hay không, thì cũng có thể kiểm tra được thanh ghi điều khiển có nhận giá trị 0 hay không. Đoạn mã trong ngôn ngữ Assembly kiểu như sau:

```
LOOP:
        TEST      PORT_4      //kiểm tra nếu port_4 bằng 0
        BEQ       READY       //nếu bằng 0, nhảy tới READY
        BRANCH    LOOP        //nếu không, tiếp tục kiểm tra.
READY:
```

Nếu không dùng cơ chế ánh xạ bộ nhớ, muốn thực hiện điều này thì đầu tiên phải sao chép giá trị thanh ghi điều khiển vào thanh ghi nào đó trong CPU, rồi sau đó mới kiểm tra. Quá trình này cần 2 chứ không phải 1 chỉ thị như cơ chế ánh xạ bộ nhớ.



Hình 8.7. Kiến trúc bộ nhớ và DMA

Ánh xạ bộ nhớ cũng có nhược điểm. Phần lớn máy tính hiện đại có cache bộ nhớ. Xét đoạn mã Assembly ở trên, lần tham chiếu đầu tiên tới **PORT_4** sẽ khiến nội dung **PORT_4** được lưu tạm trên cache. Trong các lần

tham chiếu kế tiếp, giá trị trong cache sẽ được lấy ra so sánh, chứ không phải giá trị thực trong **PORT_4**. Khi thiết bị sẵn sàng, phần mềm không có cách nào để phát hiện, và vòng lặp **LOOP** sẽ lặp vô hạn. Để tránh tình trạng này, trước tiên phần cứng phải có khả năng không cho cache một trang nhớ cụ thể. Chức năng này làm kiến trúc phần cứng lẫn HĐH trở nên phức tạp. Thứ hai, nếu chỉ có một không gian địa chỉ, module bộ nhớ và thiết bị vào/ra cần kiểm tra xem tham chiếu bộ nhớ có truy xuất vào khu vực nhớ của mình hay không. Điều này dễ dàng thực hiện nếu máy tính chỉ có một bus như Hình 8.7a. Tuy nhiên, các máy tính hiện đại có bus chuyên dụng tốc độ cao nối giữa CPU và bộ nhớ (Hình 8.7b) để tăng hiệu suất sử dụng bộ nhớ. Khi đó, trong hệ thống ánh xạ bộ nhớ có bus bộ nhớ chuyên dụng, thiết bị vào/ra có thể "không thấy" địa chỉ bộ nhớ khi địa chỉ này được đặt trên bus chuyên dụng. Một giải pháp là, đầu tiên CPU gửi địa chỉ tham chiếu tới bộ nhớ trong trước. Nếu không có, CPU mới gửi ra các bus khác. Phần cứng trong kiểu thiết kế này hết sức phức tạp. Giải pháp thứ hai là đặt thiết bị snoop trên bus bộ nhớ để thiết bị này chuyển tất cả các địa chỉ trong không gian vào/ra tới thiết bị tương ứng. Nhược điểm của phương pháp này là thiết bị vào/ra có thể không xử lý kịp tốc độ của bộ nhớ. Giải pháp thứ ba được sử dụng trên Pentium là sử dụng chip cầu PCI để lọc địa chỉ. Chip này chứa khoảng các thanh ghi được nạp tại thời điểm hệ thống khởi động. Ví dụ, địa chỉ từ 640KB đến 1MB được đánh dấu không nằm trong bộ nhớ chính. Yêu cầu tới khoảng địa chỉ này sẽ được chuyển tới bus PCI thay vì tới bộ nhớ.

8.2.5. Truy cập trực tiếp bộ nhớ (DMA)

Trong cơ chế vào/ra trực tiếp qua thăm dò, trình điều khiển chuyển dữ liệu từ vùng nhớ tiến trình người dùng tới thanh ghi trên bộ điều khiển thiết bị khi ghi dữ liệu và thực hiện điều ngược lại khi đọc dữ liệu. Trong cơ chế vào/ra có sử dụng ngắt, trình xử lý ngắt ứng với thiết bị chịu trách nhiệm sao chép dữ liệu. Để chuyển một khối dữ liệu từ bộ nhớ chính đến thanh ghi của bộ điều khiển, hệ thống sử dụng đoạn mã tương tự sau:

```
LOAD    R2, = LENGTH_OF_BLOCK    // R2 là thanh ghi chỉ số
Loop:   LOAD  R1, [data_area, R2]  // Tải block[i]
        STORE R1, 0xFFFF0124       // Đặt dữ liệu vào thanh
                                      ghi dữ liệu của ctrl
        INCR  R2                   // Tăng chỉ số
        BGE   loop                 // Quay lại vòng lặp
```

Trong cả hai trường hợp, CPU đều phải tham gia vận chuyển dữ liệu. Tuy nhiên, với cơ chế truy cập trực tiếp bộ nhớ, bộ điều khiển DMA có khả năng đọc/ghi thông tin trực tiếp vào bộ nhớ mà không cần sự can thiệp của CPU. DMA có thể được sử dụng trong cả cơ chế phần mềm thăm dò lẫn cơ chế ngắt. Bộ điều khiển DMA có phần logic cho phép chuyển dữ liệu giữa bộ nhớ chính và bộ điều khiển thiết bị. Điều này hoàn toàn được thực hiện bằng phần cứng của bộ điều khiển DMA chứ không cần sự can thiệp từ CPU (Tuy nhiên, CPU và DMA có thể tranh chấp quyền sử dụng bus dữ liệu). DMA làm tăng đáng kể hiệu suất vào/ra. Với DMA, bộ điều khiển thiết bị có thể không cần thanh ghi dữ liệu, vì bộ điều khiển DMA có thể chuyển dữ liệu trực tiếp giữa thiết bị và bộ nhớ chính. Tuy nhiên, bộ điều khiển DMA phải chứa thanh ghi địa chỉ để trình điều khiển thiết bị có thể xác lập địa chỉ bộ nhớ trong nơi nhận dữ liệu.

Bộ điều khiển DMA có thể được tích hợp với bộ điều khiển ổ đĩa và các bộ điều khiển khác. Phần lớn hệ thống máy tính hiện đại sử dụng bộ điều khiển DMA nằm trên bo mạch chủ điều hợp việc trao đổi dữ liệu cho nhiều thiết bị (có thể thực hiện đồng thời). Bộ điều khiển DMA có thể truy cập tới bus hệ thống độc lập với CPU như minh họa trên Hình 8.7c. Bộ điều khiển DMA có các thanh ghi cho phép CPU truy cập tới thanh ghi địa chỉ bộ nhớ, thanh ghi đếm và thanh ghi điều khiển. Thanh ghi điều khiển xác định sử dụng cổng vào/ra nào, hướng truyền (đọc hay ghi), đơn vị truyền (byte hay word) và khối lượng truyền trong một lần.

Xét thao tác đọc ổ đĩa khi không sử dụng DMA, đầu tiên bộ điều khiển đọc một khối (một hoặc nhiều sector) từ ổ đĩa cứng vào bộ nhớ của bộ điều khiển. Tiếp theo, bộ điều khiển tính checksum để xác định khối vừa đọc có lỗi hay không. Sau đó, bộ điều khiển tạo ra một ngắt. Cuối cùng, HĐH có thể đọc khối dữ liệu từ bộ đệm trong bộ điều khiển bằng cách sử dụng một vòng lặp lần lượt chuyển từng byte trong thanh ghi dữ liệu của bộ điều khiển vào bộ nhớ trong.

Nếu sử dụng DMA như minh họa trên Hình 8.7c, thì đầu tiên CPU thiết lập các thanh ghi để bộ điều khiển DMA biết chuyển dữ liệu gì và đến đâu (bước 1). Đồng thời, CPU cũng phát lệnh cho bộ điều khiển đĩa đọc dữ liệu từ ổ đĩa vào bộ nhớ nằm bên trong bộ điều khiển ổ đĩa. Khi dữ liệu nằm trong bộ nhớ của bộ điều khiển đĩa, DMA có thể được khởi động. Bước 2,

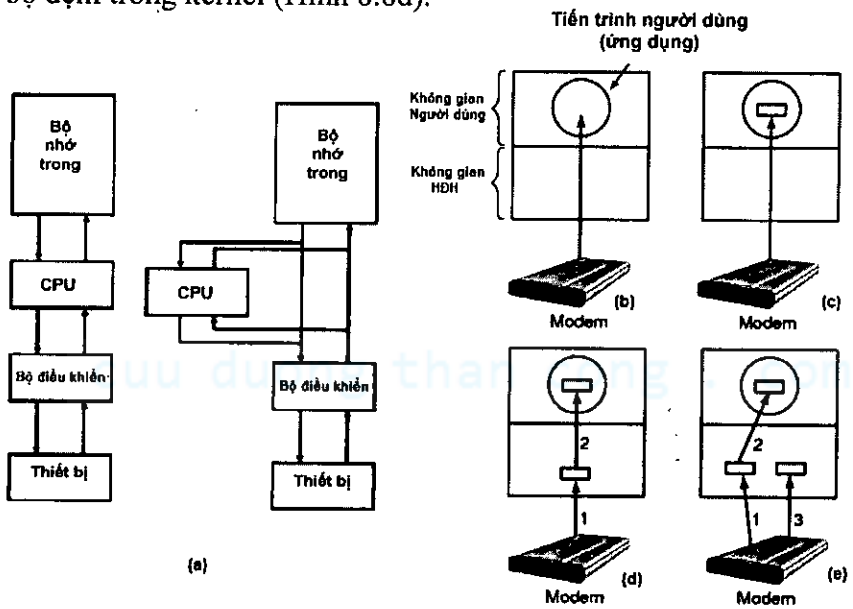
bộ điều khiển DMA khởi tạo việc chuyển dữ liệu bằng cách phát ra yêu cầu đọc tới bộ điều khiển ổ đĩa. Bộ điều khiển ổ đĩa không xác định được yêu cầu này là từ CPU hay từ bộ điều khiển DMA. Do địa chỉ bộ nhớ được ghi trên đường bus địa chỉ, nên bộ điều khiển ổ đĩa lấy byte kế tiếp trong bộ nhớ của mình ghi vào ô nhớ thích hợp. Quá trình ghi bộ nhớ trong diễn ra trong chu kỳ bus kế tiếp (bước 3). Khi ghi xong, bộ điều khiển ổ đĩa gửi tín hiệu biên nhận tới bộ điều khiển DMA (bước 4). Bộ điều khiển DMA tăng địa chỉ bộ nhớ và giảm thanh ghi đếm byte. Nếu thanh ghi đếm vẫn lớn hơn 0, DMA thực hiện bước 2 tới 4 cho tới khi thanh ghi đếm bằng 0. Lúc này, bộ điều khiển DMA sẽ tạo ra ngắt báo cho CPU biết đã hoàn thành quá trình truyền dữ liệu. Có nhiều kiểu bộ điều khiển DMA với độ phức tạp khác nhau. Đơn giản nhất là kiểu chỉ truyền một byte tại một thời điểm như mô tả trên. Phức tạp hơn là kiểu cho phép truyền nhiều luồng dữ liệu một lúc. Như vậy, bộ điều khiển có nhiều nhóm thanh ghi, mỗi nhóm ứng với một kênh truyền. CPU sẽ nạp các tham số thích hợp vào một nhóm thanh ghi. Mỗi lần truyền thực hiện trên một bộ điều khiển thiết bị riêng.

8.2.6. Bộ đệm (Buffer)

Trong kỹ thuật bộ đệm dữ liệu, khi thiết bị ở trạng thái rỗi, bộ quản lý thiết bị thực hiện đọc/ghi dữ liệu trước khi tiến trình yêu cầu thực hiện. Đệm vào là việc đọc dữ liệu vào bộ nhớ trong, đệm ra là việc ghi dữ liệu ra thiết bị, cả hai thao tác này có thể thực hiện song song với tiến trình và trước khi tiến trình đưa ra yêu cầu thực sự. Bộ đệm cho phép tiến trình sử dụng xen kẽ CPU và thiết bị một cách tường minh (minh họa trên Hình 8.8a).

Chú ý đến đặc điểm tiến trình: Tiến trình hướng vào/ra có nhiều thao tác vào/ra (ví dụ, tiến trình sao chép dữ liệu giữa hai thiết bị) và tiến trình hướng tính toán có nhiều thao tác tính toán trên CPU (tiến trình tìm số nguyên tố). Tiến trình bình thường có thể chia thành nhiều pha, các pha hướng tính toán và hướng vào/ra xen kẽ nhau. Bộ phận quản lý bộ nhớ sẽ sử dụng các kỹ thuật quản lý bộ nhớ để làm giảm thời gian thực hiện vào/ra trung bình. Xét việc chuyển dữ liệu ra modem không sử dụng bộ đệm như minh họa trên Hình 8.8b. Tiến trình người dùng sử dụng lời gọi hệ thống **write** để chuyển đi **n** ký tự. Hệ thống có hai lựa chọn: Một là phong tỏa tiến trình người dùng cho tới khi tất cả ký tự được ghi thành công, nhưng cách

này tốn nhiều thời gian, đặc biệt là trên những đường truyền có tốc độ thấp (chẳng hạn đường điện thoại). Cách thứ hai là cho phép tiến trình người dùng thực thi ngay lập tức. Trình điều khiển thực hiện việc xuất trong khi tiến trình người dùng tiếp tục thực hiện công việc tính toán của mình. Vấn đề ở đây là: Làm thế nào để tiến trình người dùng xác định được việc xuất dữ liệu đã hoàn thành và do vậy có thể sử dụng tiếp bộ đệm? Hệ thống có thể tạo ra tín hiệu hoặc ngắt phần mềm, nhưng kiểu lập trình này rất khó và đặc biệt dễ gây xung đột. Một giải pháp tốt hơn là kernel sao chép dữ liệu tới bộ đệm trong kernel (Hình 8.8d).

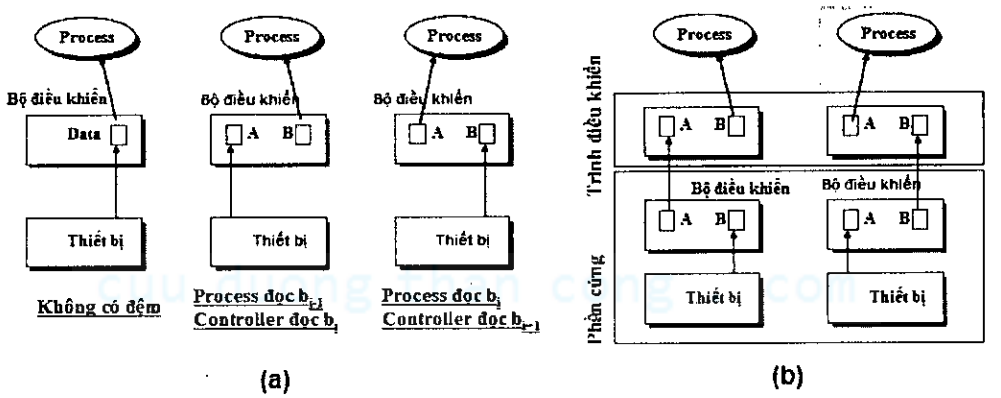


Hình 8.8. Bộ đệm dữ liệu

Modem là thiết bị kiểu ký tự đọc/ghi từng byte dữ liệu. Bộ điều khiển modem có thanh ghi dữ liệu với kích thước 1 byte chứa ký tự cuối cùng nhận được sau lệnh **read**. Khi có lời gọi **read** kế tiếp, trình điều khiển chuyển lệnh tương ứng cho bộ điều khiển, và đến lượt mình bộ điều khiển ra lệnh thiết bị đặt byte kế tiếp vào thanh ghi dữ liệu của bộ điều khiển (Hình 8.9a). Bộ đệm nằm bên trong bộ điều khiển giúp tiến trình giảm đáng kể thời gian đợi ký tự, vì bộ điều khiển đã lưu tạm dữ liệu trước. Trong Hình 8.9b, ký tự kế tiếp được đặt vào thanh ghi dữ liệu B của bộ điều khiển trước khi tiến trình yêu cầu đọc. Kế tiếp, thiết bị đọc dữ liệu và chuyển byte tiếp theo vào thanh ghi A (kể cả khi tiến trình chưa có yêu cầu đọc). Sau đó,

tiến trình yêu cầu ký tự đã được đặt từ trước vào thanh ghi B. Lúc này, thiết bị sẽ đọc ký tự kế tiếp vào thanh ghi A. Thiết bị sẽ đọc ký tự (i + 1) song song với việc tiến trình sử dụng ký tự i. Sự phối hợp này trở nên "hoàn hảo" nếu tốc độ đọc của bộ điều khiển bằng tốc độ tiến trình "tiêu thụ" ký tự.

Có thể đặt thêm bộ đệm giữa bộ điều khiển và trình điều khiển thiết bị (Hình 8.9b), khi đó có hai bộ đệm trong hệ thống. Bộ đệm thứ nhất lưu tạm dữ liệu cho tầng cao hơn lấy, trong khi bộ đệm kia lưu dữ liệu lấy từ tầng bên dưới. Chú ý, kỹ thuật này hoàn toàn áp dụng được cho thiết bị hướng khối, chẳng hạn ổ đĩa. Khi đó, kích thước bộ đệm phải đủ lớn để chứa được một khối dữ liệu.

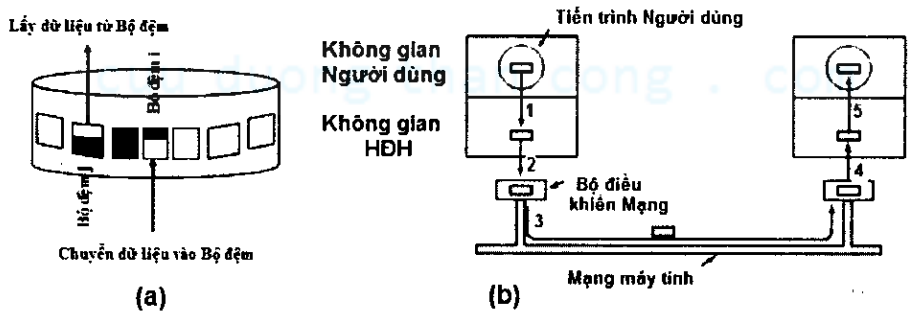


Hình 8.9. Kỹ thuật bộ đệm dữ liệu

Hệ thống có thể có nhiều bộ đệm như trong Hình 8.10a. Bộ phận tạo ra dữ liệu (với thao tác **read** là bộ điều khiển, với thao tác **write** là CPU) ghi dữ liệu vào bộ đệm i, trong khi bộ phận tiêu thụ dữ liệu (bộ điều khiển trong thao tác **write** và CPU trong thao tác **read**) đọc dữ liệu từ bộ đệm j. Khi đó, các bộ đệm từ j đến (n - 1) và từ 0 đến (i - 1) đầy. Bộ phận tiêu thụ có thể đọc dữ liệu trong bộ đệm j đến (n - 1) và từ 0 đến (i - 1). Bộ phận sản xuất có thể ghi vào các bộ đệm i đến (j - 1), trong khi bộ phận tiêu thụ đọc từ bộ đệm j. Trong kỹ thuật đệm vòng (circular buffer) như vậy, bộ phận sản xuất không thể ghi vượt qua bộ phận tiêu thụ, vì nếu không sẽ làm mất dữ liệu chưa tiêu thụ. Bộ phận sản xuất chỉ được ghi tới bộ đệm j - 1 nếu bộ đệm j đang được đợi tiêu thụ.

Ảnh hưởng của bộ đệm tới hiệu suất phụ thuộc nhiều vào đặc điểm tiến trình. Tiến trình hướng vào/ra thường đọc bộ đệm ngay khi bộ điều khiển

chuyển dữ liệu vào, hoặc ghi ngay dữ liệu khi bộ điều khiển bắt đầu ghi dữ liệu ra thiết bị. Tiến trình hướng tính toán có xu hướng ngược lại: bộ đệm vào thường đầy trong khi bộ đệm ra thường rỗng. Tiến trình đơn giản thường là hướng vào/ra, còn tiến trình phức tạp thường có nhiều pha, có pha hướng vào/ra và có pha hướng tính toán. Các pha xen kẽ nhau tận dụng tối đa ưu điểm của bộ đệm, vì trong thời gian tiến trình thực hiện tính toán, bộ điều khiển nhanh chóng đổ dữ liệu vào (hoặc ra) bộ đệm. Khi bộ đệm đầy (hoặc trống), tiến trình chuyển sang pha hướng tính toán. Tuy nhiên, bộ đệm cũng làm suy giảm hiệu suất khi dữ liệu bị trung chuyển qua nhiều bộ đệm. Xét ví dụ mạng máy tính hoạt động trên Hình 8.10b. Đầu tiên, tiến trình người dùng thực hiện lời gọi hệ thống để gửi thông điệp qua mạng. HĐH sao chép gói dữ liệu tới bộ đệm trong HĐH để tiến trình người dùng tiếp tục thực thi (bước 1). Khi thực thi, trình điều khiển chuyển gói dữ liệu tới bộ điều khiển để xuất (bước 2). Trình điều khiển không gửi trực tiếp dữ liệu trên đường truyền, vì phải đảm bảo tốc độ gửi cố định. Trình điều khiển không thể đảm bảo có thể lấy dữ liệu từ bộ nhớ trong với tốc độ nhất định do kênh DMA và các thiết bị vào/ra khác có thể tranh chấp bus dùng chung. Chỉ cần gửi lỗi một byte, có thể làm hỏng toàn bộ gói dữ liệu. Đặt gói dữ liệu trong bộ đệm của bộ điều khiển sẽ tránh được vấn đề này. Sau khi được chuyển vào bộ đệm bên trong bộ điều khiển, gói dữ liệu sẽ được truyền ra mạng (bước 3). Sau khi bit cuối cùng đến đích, toàn bộ gói dữ liệu được đặt vào bộ đệm của bộ điều khiển nơi nhận. Tiếp đến, gói dữ liệu được chuyển tiếp vào bộ đệm trong kernel (bước 4). Cuối cùng, gói dữ liệu được sao chép tới bộ đệm tiến trình người dùng. Thông thường, tiến trình nhận sẽ gửi lại một biên nhận. Chỉ khi nhận được biên nhận, phía gửi mới được gửi tiếp gói dữ liệu mới. Rõ ràng, càng nhiều bộ đệm, tốc độ truyền dữ liệu càng chậm.



Hình 8.10. Bộ đệm xoay vòng và ảnh hưởng tới hiệu suất

8.3. TRÌNH ĐIỀU KHIỂN THIẾT BỊ

Tiến trình ứng dụng gọi trình điều khiển thiết bị khi muốn thực hiện vào/ra. Trình điều khiển "dịch" yêu cầu này thành lệnh mà bộ điều khiển có thể "hiểu" được. Sau đó, hoặc trình điều khiển sẽ thăm dò xem bộ điều khiển đã hoàn thành công việc hay chưa, hoặc sẽ ghi lại các thông tin vào bảng thiết bị trong trường hợp sử dụng ngắt. Ngoài việc phải đưa ra các lệnh đặc thù cho từng bộ điều khiển thiết bị, trình điều khiển thiết bị còn phải:

1. Cung cấp API để ứng dụng thực hiện vào/ra trên thiết bị. Giao diện API giữa các trình điều khiển nên giống nhau.
2. Đảm bảo phối hợp hoạt động của tiến trình ứng dụng và bộ điều khiển thiết bị.
3. Tối ưu hiệu suất tổng thể của hệ thống với phương pháp điều khiển thích hợp.

8.3.1. Giao diện của trình điều khiển

Mỗi HĐH xây dựng riêng cho mình một kiến trúc hệ thống vào/ra với hai giao diện cơ bản là API (giao diện cho người lập trình) và giao diện với nhân HĐH.

☞ ***API – Giao diện lập trình ứng dụng***

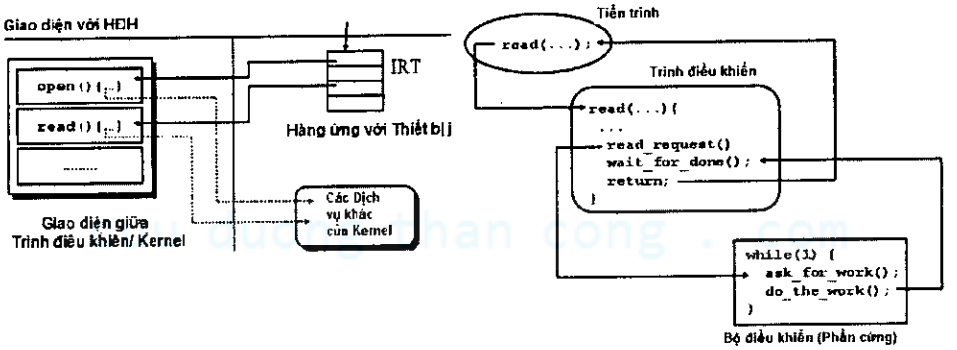
API là giao diện người lập trình sử dụng để thực hiện vào/ra trên thiết bị. Nhiệm vụ chính của thiết bị là truyền thông hoặc lưu trữ. Các thiết bị "vào" thường "tạo" ra thông tin đưa vào hệ thống. Thông tin có thể không có khuôn dạng (đến từ mạng máy tính), hoặc có khuôn dạng xác định (đến từ ổ đĩa). Tiến trình sẽ "ghi" thông tin lên thiết bị "ra", có thể là thiết bị truyền thông, hoặc thiết bị lưu trữ để sau này sử dụng lại. Trình điều khiển chịu trách nhiệm kiểm soát trạng thái thiết bị (bận hay rỗi) hay tiến trình nào đang sử dụng thiết bị. Bên cạnh thông tin lưu trong bảng trạng thái thiết bị, phải lưu trữ các thông tin về đặc tính của từng loại thiết bị. Phần lớn giao diện của trình điều khiển có lời gọi **open** và **close** với mục đích để khởi tạo (xin cấp phát thiết bị, tạo các bảng tương ứng) và kết thúc quá trình sử dụng thiết bị (đánh dấu thiết bị ở trạng thái rỗi).

Trình điều khiển cung cấp các hàm để ứng dụng sử dụng khi muốn trao đổi dữ liệu với thiết bị. Mặc dù cố gắng để tạo ra một giao diện (là hệ thống

các hàm) thống nhất (về cách gọi, truyền tham số) cho tất cả các loại thiết bị, nhưng mục tiêu này khó đạt được vì các kiểu thiết bị có bản chất khác nhau, nên cách sử dụng cũng khác nhau.

☞ *Giao diện với nhân HĐH*

Trình điều khiển thực thi các chỉ thị đặc quyền khi ra lệnh cho bộ điều khiển thiết bị, nghĩa là trình điều khiển được thực thi như một bộ phận của HĐH, chứ không phải bộ phận của chương trình ứng dụng. Trình điều khiển còn có khả năng đọc/ghi thông tin trên không gian địa chỉ của nhiều tiến trình, vì thiết bị có thể được dùng chung. Trong các HĐH cũ, trình điều khiển thường được tích hợp vào HĐH bằng cách thay đổi mã nguồn HĐH và sau đó biên dịch lại.



a) b)

Hình 8.11. Giao diện giữa HĐH và trình điều khiển thiết bị

Các HĐH hiện đại đơn giản hóa việc cài đặt thiết bị bằng cách cho phép cài đặt trình điều khiển mà không cần biên dịch lại HĐH (hệ thống sẽ được cấu hình lại thông qua những chỉ thị đặc biệt). Khi thiết kế, mã HĐH kết buộc động với các chức năng của trình điều khiển. Ví dụ trong hHình 8.11a, bảng gián tiếp (ITR) của thiết bị j "trỏ" tới các module cài đặt các hàm **open, read**... của trình điều khiển. Trình điều khiển được chuẩn hóa để có giao diện API giống nhau. Vì trình điều khiển thiết bị được tích hợp vào nhân sau khi nhân đã được biên dịch, do đó nhân phải cung cấp giao diện cho trình điều khiển thiết bị để xin cấp phát vùng nhớ làm bộ đệm để cập nhật các bảng trong nhân. HĐH sử dụng bảng tham chiếu gián tiếp (Inderict Reference Table – IRT) để truy cập tới các phần khác nhau trong trình điều khiển dựa trên định danh thiết bị và tên hàm cần sử dụng. Nhân cung cấp

giao diện API – là một phần giao diện của HĐH. Khi tiến trình gọi lời gọi hệ thống, nhân sẽ chuyển lời gọi này tới trình điều khiển nhờ bảng gián tiếp. Khi cài đặt trình điều khiển, thông tin về IRT được đưa cho HĐH sử dụng trong quá trình thực thi.

8.3.2. Tương tác giữa CPU và thiết bị

CPU và thiết bị là các thực thể tách rời, có khả năng hoạt động độc lập với nhau. Bộ phận quản lý thiết bị phải cung cấp cách thức để tiến trình thực thi trên CPU có thể đồng bộ hoạt động với thiết bị mà tiến trình sử dụng. Hình 8.11b minh họa ba bộ phận có liên quan trong thao tác vào/ra và cách thức phối hợp hoạt động giữa chúng. Trong hình vẽ, thao tác của phần cứng bộ điều khiển, trình điều khiển thiết bị và phần mềm ứng dụng được minh họa bằng ngôn ngữ C. Vì bộ điều khiển là phần cứng luôn hoạt động nên được đặt trong vòng lặp vô hạn **for**. Đầu tiên, thiết bị "đợi việc" – tức là đợi lệnh được đặt vào thanh ghi lệnh của bộ điều khiển (đợi bộ điều khiển thiết lập cờ **busy**). Sau khi nhận lệnh, bộ điều khiển thực hiện lệnh và sẽ báo hiệu khi thực hiện xong lệnh (bằng cách thiết lập cờ **done** trong thanh ghi trạng thái). Trình điều khiển thiết bị (một phần của HĐH) được người dùng gọi khi cần thiết. Về mặt khái niệm, có thể xem mỗi chức năng của trình điều khiển như một thủ tục, thủ tục này được tiến trình ứng dụng gọi. Khi chức năng nào đó được gọi, trình điều khiển thực hiện một "lời gọi phần cứng" tới bộ điều khiển bằng cách thiết lập giá trị các thanh ghi trong bộ điều khiển. Trong hệ thống thăm dò, trình điều khiển sẽ thăm dò xem thiết bị đã thực hiện xong công việc hay chưa. Trong trường hợp sử dụng ngắt, trình xử lý ngắt cho thiết bị được gọi. Trong cả hai trường hợp, tiến trình ứng dụng yêu cầu vào/ra sẽ được trả lại quyền sử dụng CPU giống như trả về sau khi gọi thủ tục trong môi trường lập trình truyền thống.

cuu duong than cong . com

CÂU HỎI ÔN TẬP

1. Trình bày các phương pháp quản lý thiết bị.
2. Trình bày ưu điểm của vào/ra qua ánh xạ bộ nhớ.
3. Trình bày ưu điểm của DMA.

Chương 9
QUẢN LÝ BỘ NHỚ

Để nâng cao hiệu suất bằng cách chia sẻ CPU giữa nhiều tiến trình, hệ thống phải tải nhiều tiến trình vào trong bộ nhớ, nghĩa là bộ nhớ được sử dụng trong chế độ chia sẻ. Chương này trình bày, so sánh các phương pháp quản lý bộ nhớ khác nhau, từ những thuật toán quản lý bộ nhớ cực kỳ đơn giản trong những thế hệ máy tính đầu tiên đến các thuật toán phân trang kết hợp phân đoạn phức tạp.

9.1. CÁC LOẠI ĐỊA CHỈ

cuu duong than cong . com

9.1.1. Kết buộc địa chỉ

Đối với người lập trình, chương trình viết ra có khuôn dạng tương tự Hình 9.1a. Trình biên dịch cấp phát không gian cho biến **gVar** trong module đối tượng – là module có khả năng tái định vị. Tuy nhiên, rất có thể hàm **put_record** lại nằm trong module đối tượng khác, do đó trình biên dịch chưa kết buộc hàm này với địa chỉ cụ thể nào cả. Trình biên dịch tạo ra đoạn mã tương tự như Hình 9.1b. **gVal** nằm ở ô 0036 và trình biên dịch sẽ ghi lại ánh xạ này vào bảng Biểu tượng (bảng này nằm ở vị trí 0600 trong module). Chỉ thị ở địa chỉ 0220 chuyển giá trị 7 vào thanh ghi R1, chỉ thị kế tiếp ở địa chỉ 0224 chuyển giá trị R1 vào ô 0036, kết hợp lại là lệnh gán **gVar = 7**. Trình biên dịch thực hiện lời gọi bằng cách đặt giá trị **gVar** vào ngăn xếp (chỉ thị có địa chỉ 0228), sau đó gọi **put_record** (chỉ thị ở địa chỉ 0232). Chú ý, **put_record** nằm ở module đối tượng khác. Vì chưa có đủ thông tin nên trình biên dịch không thể xác định được vị trí **put_record** (công việc này sẽ để lại cho trình liên kết). Trình biên dịch ghi thông tin về tham chiếu này tại bảng tham chiếu ngoài (bảng nằm ở địa chỉ 0400). Trình liên kết kết hợp các module tái định vị để sinh mã tuyệt đối như minh họa trong Hình 9.1c.

Trình liên kết chỉnh lại các địa chỉ. Đoạn mã trong Hình 9.1b được chuyển sang Hình 9.1c ở vị trí 1008. Như vậy, địa chỉ tương đối 0 trong Hình 9.1b chuyển thành địa chỉ 1008 trong Hình 9.1c. Chú ý, địa chỉ trong module tuyệt đối vẫn bắt đầu từ 0. Tại thời điểm tải, module tuyệt đối được tải vào bộ nhớ để tạo nên hình ảnh tiến trình trong bộ nhớ (Hình 9.1d).

```
...
static int gVar;        (a)
...
int proc_a(int arg){
    ...
    gVar = 7;
    put_record(gVar);
    ...
}
```

```
Code Segment                          Data Segment                    (c)
Relative  Generated                   Relative  Generated
Address   Code                        Address   variable space
0000      (Other modules)             ...
...                                   0136      [Space for gVar variable]
1008      entry      proc_a           ...
...                                   1000      (last location in the data segment)
1220      load       =7, R1
1224      store      R1, 0136
1228      push       1036
1232      call       2334
...
1399      (End of proc_a)
...       (Other modules)
2334      entry      put_record
...
2670      (optional symbol table)
...
2999      (last location in the code segment)
```

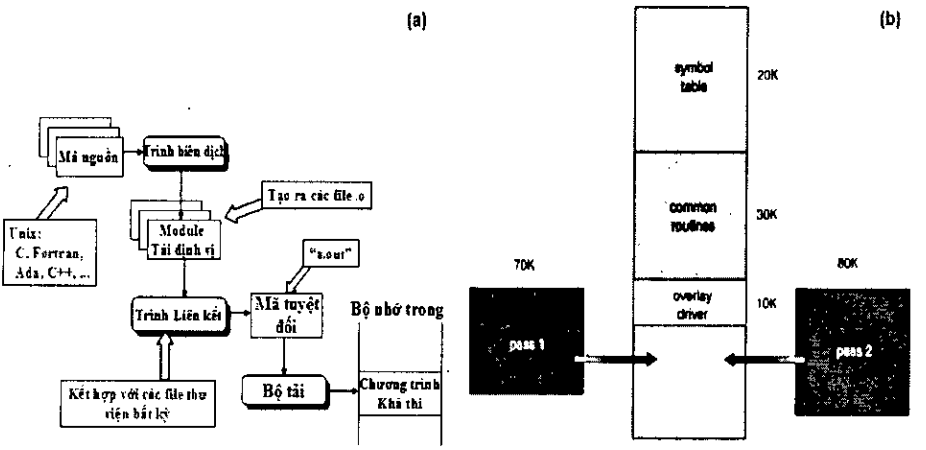
```
Relative  Generated
Address   Code                        (d)
0000      (Other process's programs)
4000      (Other modules)
...
5008      entry      proc_a
...
5036      [Space for gVar variable]
...
5220      load       =7, R1
5224      store      R1, 7136
5228      push       5036
5232      call       6334
...
5399      (End of proc_a)
...       (Other modules)
6334      entry      put_record
...
6670      (optional symbol table)
...
6999      (last location in the code segment)
7000      (first location in the data segment)
...
7136      [Space for gVar variable]
...
8000      (Other process's programs)
```

```
Code Segment                          Data Segment                    (b)
Relative  Generated                   Relative  Generated
Address   Code                        Address   variable space
0000      ...                         ...
...                                   0036      [Space for gVar variable]
0008      entry      proc_a           ...
...                                   0049      (last location in the data segment)
0220      load       =7, R1
0224      store      R1, 0036
0228      push       0036
0232      call       'put_record'
...
0400      External reference table
...
0404      'put_record'      0232
...
0500      External definition table
...
0540      'proc_a'   0008
...
0600      (symbol table)
...
0799      (last location in the code segment)
```

Hình 9.1. Quá trình biên dịch

Thường chương trình khả thi (kết quả của trình liên kết) được lưu trữ dưới dạng file nhị phân, chẳng hạn các file .COM hay .EXE trong HĐH MS-DOS. Chương trình sẽ được tải vào bộ nhớ và hệ thống tạo lập tiến trình tương ứng để thực thi. Phụ thuộc vào phương pháp quản lý bộ nhớ, tiến trình có thể hoán chuyển giữa ổ đĩa và bộ nhớ trong suốt quá trình thực thi. Tiến trình người dùng có thể được tải vào vị trí bất kỳ trong bộ nhớ. Vì thế, mặc dù không gian địa chỉ bắt đầu từ 0, nhưng địa chỉ của chỉ thị đầu tiên trong tiến trình không nhất thiết phải là 0. Thường chương trình nguồn phải qua nhiều bước trước khi thực thi (Hình 9.2a). Địa chỉ tại mỗi bước được biểu diễn theo nhiều cách khác nhau. Địa chỉ trong chương trình nguồn mang tính biểu tượng (**gVar**). Chương trình dịch sẽ "kết buộc" địa chỉ

biểu tượng này với địa chỉ khả định vị (chẳng hạn, cách vị trí bắt đầu của module 0036 byte). Đến lượt trình liên kết hay bộ tải sẽ kết buộc địa chỉ khả định vị với địa chỉ tuyệt đối (chẳng hạn 5036). Mỗi quá trình kết buộc là thực hiện ánh xạ từ không gian địa chỉ này sang không gian địa chỉ khác.



Hình 9.2. Kết buộc địa chỉ trong quá trình ánh xạ bộ nhớ

Quá trình kết buộc chỉ thị và dữ liệu với địa chỉ cụ thể trong bộ nhớ có thể thực hiện tại bất kỳ thời điểm nào sau đây:

- **Thời điểm biên dịch:** Nếu tại thời điểm biên dịch biết được tiến trình sẽ nằm ở đâu trong bộ nhớ, trình biên dịch có thể sinh mã với địa chỉ tuyệt đối. Ví dụ, nếu biết trước vị trí bắt đầu của tiến trình người dùng là R, thì mã do trình biên dịch sinh ra bắt đầu từ vị trí R. Nếu sau đó, vị trí khởi đầu thay đổi thì phải biên dịch lại chương trình. Chương trình dạng .COM của MS-DOS có mã là địa chỉ tuyệt đối được sinh tại thời điểm biên dịch.
- **Thời điểm tải**: Nếu tại thời điểm biên dịch không biết trước vị trí trong bộ nhớ của tiến trình, thì trình biên dịch sinh mã khả định vị. Trong trường hợp này, sự kết buộc địa chỉ bị trì hoãn đến thời điểm tải. Nếu địa chỉ khởi đầu thay đổi, thì chỉ cần nạp lại mã khả định vị để sinh ra địa chỉ tuyệt đối mà không cần biên dịch lại chương trình.
- **Thời điểm thực thi**: Nếu trong quá trình thực thi, tiến trình có thể di chuyển từ vùng nhớ này sang vùng nhớ khác, thì sự kết buộc địa chỉ phải trì hoãn đến thời điểm thực thi. Để thực hiện điều này, cần có những phần cứng đặc biệt được trình bày trong mục 9.2.

9.1.2. Tải động (Dynamic loading)

Thủ tục (lưu trên ổ đĩa dưới định dạng có thể liên kết và tải) chỉ được tải vào bộ nhớ khi cần thiết. Khi được gọi, hệ thống kiểm tra xem thủ tục đã nằm trong bộ nhớ hay chưa. Nếu chưa, bộ tải đưa thủ tục vào bộ nhớ và cập nhật lại bảng địa chỉ của chương trình. Sau đó, HĐH thực thi thủ tục vừa tải vào. Trong chương trình có nhiều đoạn mã xử lý biệt lệ (ví dụ thủ tục xử lý lỗi) với đặc điểm ít được sử dụng, do đó bộ nhớ được tiết kiệm do không phải tải toàn bộ chương trình vào. Tải động không đòi hỏi HĐH hỗ trợ. Người lập trình thiết kế chương trình để tận dụng ưu điểm của chiến lược này.

9.1.3. Liên kết động

Phần lớn HĐH hỗ trợ liên kết tĩnh (thư viện ngôn ngữ lập trình giống module đối tượng và được bộ tải kết hợp thành hình ảnh nhị phân của chương trình). Khái niệm liên kết động tương tự khái niệm tải động, trong đó quá trình liên kết bị trì hoãn lại. Đặc tính này được sử dụng với thư viện hệ thống, chẳng hạn thư viện các thủ tục con của ngôn ngữ lập trình. Nếu không có đặc tính này, hình ảnh nhị phân của tất cả các chương trình trong hệ thống phải chứa bản sao thư viện ngôn ngữ. Điều này lãng phí không gian ổ đĩa và bộ nhớ chính. Với liên kết động, hình ảnh nhị phân của chương trình không chứa thủ tục của thư viện hệ thống mà chứa "đại diện" của nó. "Đại diện" là đoạn mã nhỏ dùng để xác định vị trí của thủ tục tương ứng trong bộ nhớ, hoặc làm thế nào để tải vào bộ nhớ nếu thủ tục chưa nằm trong bộ nhớ. Khi thực thi, "đại diện" sẽ kiểm tra liệu thủ tục mà mình đại diện đã nằm trong bộ nhớ chưa, nếu chưa thì tải thủ tục vào. Sau đó, thủ tục thật sẽ thay thế "đại diện". Trong những lần thực thi kế đó, thủ tục được thực thi ngay. Ngoài ra, chỉ cần một bản sao của thủ tục cho tất cả các tiến trình dùng chung.

Khi có nhiều phiên bản, chương trình lựa chọn sử dụng phiên bản thư viện mới nhất. Nếu không có cơ chế liên kết động, thì phải liên kết lại chương trình khi có thêm thư viện mới. Để tránh trường hợp chương trình ngẫu nhiên sử dụng phiên bản thư viện mới không tương thích với phiên bản cũ, hệ thống sẽ tích hợp thông tin về phiên bản trong cả chương trình lẫn thư viện. Có thể có nhiều phiên bản của cùng một thư viện được tải vào bộ nhớ

và chương trình sẽ quyết định sử dụng phiên bản thư viện nào. Do đó, chỉ những chương trình được biên dịch với phiên bản thư viện mới bị ảnh hưởng trong những hệ thống chưa có phiên bản mới. Những chương trình được liên kết trước khi có thư viện mới sẽ tiếp tục sử dụng thư viện cũ. Liên kết động đòi hỏi HĐH hỗ trợ, vì nếu các tiến trình không được phép truy xuất vào vùng nhớ của nhau thì chỉ HĐH mới có thể cho phép nhiều tiến trình truy xuất đến cùng địa chỉ nhớ.

9.1.4. Phủ (overlay)

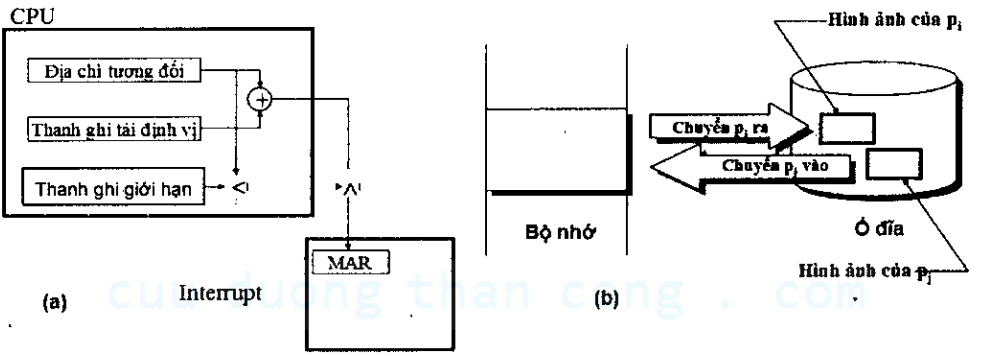
Tải phủ là công nghệ cho phép tiến trình lớn hơn lượng bộ nhớ cấp phát. Ý tưởng của kỹ thuật này là tại một thời điểm chỉ lưu trong bộ nhớ những chỉ thị và dữ liệu thực sự cần thiết. Khi không cần thiết, chỉ thị và dữ liệu sẽ được đưa ra ngoài để lấy chỗ cho chỉ thị và dữ liệu cần thiết. Ví dụ, một trình biên dịch gồm nhiều pha khác nhau như: tiền xử lý, xây dựng cây cú pháp, biên dịch sơ bộ, tối ưu hóa, tạo mã máy. Các pha diễn ra tuần tự và độc lập với nhau. Đoạn mã của mỗi pha đặt trong một phủ và lần lượt được đưa vào bộ nhớ. Giống như tải động, phủ không đòi hỏi HĐH hỗ trợ. Người lập trình thiết kế và lập trình cho cấu trúc vùng phủ chính xác. Điều này đòi hỏi người lập trình phải biết đầy đủ về cấu trúc, mã lệnh, cũng như cấu trúc dữ liệu của chương trình. Nếu kích thước chương trình rất lớn (chương trình nhỏ không cần kỹ thuật phủ) thì điều này trở nên đặc biệt khó khăn.

9.2. KHÔNG GIAN ĐỊA CHỈ

Địa chỉ CPU tạo ra là địa chỉ logic, địa chỉ CPU chuyển cho bộ phận quản lý bộ nhớ (địa chỉ sẽ được tải vào thanh ghi địa chỉ bộ nhớ MAR) là địa chỉ vật lý. Nếu kết buộc tại thời điểm tải và thời điểm dịch, địa chỉ vật lý và địa chỉ logic là một. Còn kết buộc tại thời điểm thực thi thì hai địa chỉ này khác nhau và địa chỉ logic còn được gọi là địa chỉ ảo. Toàn bộ địa chỉ logic do chương trình sinh ra tạo thành không gian địa chỉ logic, còn tập tất cả các địa chỉ vật lý ứng với những địa chỉ logic này tạo nên không gian địa chỉ vật lý.

Đơn vị quản lý bộ nhớ (MMU) là thiết bị phần cứng thực hiện ánh xạ địa chỉ ảo sang địa chỉ vật lý. Hình 9.3a minh họa quá trình ánh xạ thông qua thanh ghi tái định vị. Địa chỉ vật lý là tổng địa chỉ tương đối do tiến

trình sinh ra với giá trị lưu trong thanh ghi tái định vị. Ví dụ, nếu thanh ghi tái định vị có giá trị 4000, địa chỉ người dùng muốn là 25 thì MMU sinh ra địa chỉ 4025. MS-DOS chạy trên dòng CPU Intel 80X86 sử dụng bốn thanh ghi tái định vị khi tải và thực thi tiến trình. Chương trình người dùng không "nhìn" thấy địa chỉ vật lý mà chỉ sinh ra địa chỉ logic. Phần cứng chuyển địa chỉ logic sang địa chỉ vật lý và chỉ xác định địa chỉ ô nhớ được tham chiếu cho đến khi tham chiếu được tạo ra. Lúc này có hai loại địa chỉ là địa chỉ logic (từ 0 đến max) và địa chỉ vật lý (từ R + 0 đến R + max với R là giá trị thanh ghi tái định vị). Quá trình ánh xạ này là trọng tâm việc quản lý bộ nhớ.



Hình 9.3. Quản lý bộ nhớ

9.3. HOÁN CHUYỂN

Tiến trình có thể tạm thời bị hoán chuyển (swap) từ bộ nhớ trong ra ổ đĩa, sau đó được đưa trở lại bộ nhớ để tiếp tục thực thi. Nếu sử dụng thuật toán điều phối CPU xoay vòng, khi hết lượng tử thời gian cấp phát, trình quản lý bộ nhớ sẽ chuyển tiến trình đã kết thúc ra ngoài rồi đưa tiến trình khác vào phần bộ nhớ vừa được giải phóng (Hình 9.3b). Trong khi đó, bộ điều phối CPU sẽ cấp phát CPU cho tiến trình khác đã nằm trong bộ nhớ. Tốc độ hoán chuyển tiến trình phải đủ nhanh để luôn có tiến trình khả thi nằm trong bộ nhớ. Lượng tử thời gian phải đủ để tiến trình thực hiện được lượng công việc có ích giữa hai lần hoán chuyển. Một số HĐH sử dụng độ ưu tiên trong việc hoán chuyển: tiến trình có mức ưu tiên cao có thể chiếm quyền sử dụng CPU của tiến trình có độ ưu tiên thấp, bộ phận hoán chuyển đưa tiến trình có độ ưu tiên thấp ra ngoài. Khi tiến trình có mức ưu tiên cao

kết thúc, tiến trình mức ưu tiên thấp được đưa trở lại bộ nhớ để tiếp tục thực hiện. Nếu kết buộc tại thời điểm biên dịch hoặc thời điểm tải, tiến trình phải quay về đúng vùng nhớ cũ. Nếu kết buộc bị trì hoãn đến thời điểm thực thi, HĐH có thể hoán chuyển tiến trình tới vị trí khác. Bộ điều phối kiểm tra tiến trình được chọn thực thi nằm trong bộ nhớ hay chưa. Nếu chưa và không còn vùng nhớ trống thì bộ điều phối sẽ hoán chuyển tiến trình hiện đang nằm trong bộ nhớ ra ổ đĩa để lấy chỗ cho tiến trình mới. Khi đó, thời gian chuyển ngữ cảnh trong hệ thống khá lớn. Giả sử tiến trình người dùng có kích thước 100KB và tốc độ truyền dữ liệu của ổ đĩa cứng là 1MGB/s. Thời gian hoán chuyển tiến trình giữa bộ nhớ và ổ đĩa là 100/1000 = 1/10s = 100ms. Giả sử thời gian trễ là 8ms và bỏ qua thời gian dịch chuyển đầu đọc của ổ đĩa cứng, thì thời gian hoán chuyển là 108ms. Tổng thời gian hoán chuyển ra và hoán chuyển vào là 216ms. Thường không gian hoán chuyển là một vùng riêng biệt trên ổ đĩa, độc lập với hệ thống file, nên có thể bỏ qua thời gian dịch chuyển đầu đọc/ghi. Để tận dụng CPU hiệu quả, thời gian thực thi của tiến trình phải lớn hơn thời gian hoán chuyển. Vì thế, trong thuật toán điều phối CPU theo kiểu xoay vòng, lượng tử thời gian phải lớn hơn 0,216s.

Thời gian truyền (chiếm phần lớn thời gian hoán chuyển) tỷ lệ với khối lượng hoán chuyển. Do đó, nếu xác định được chính xác khối lượng bộ nhớ tiến trình người dùng sẽ sử dụng (chứ không phải toàn bộ kích thước tiến trình), thì chỉ cần hoán chuyển phần bộ nhớ thực sự cần thiết và do đó giảm thời gian hoán chuyển. Vì thế, tiến trình với yêu cầu bộ nhớ động sử dụng các lời gọi hệ thống khi yêu cầu hay giải phóng bộ nhớ. Nếu tiến trình P_1 đợi vào/ra và bộ đệm vào/ra nằm trong bộ nhớ người dùng, thì P_1 không được phép hoán chuyển. Giả sử thao tác vào/ra phải đợi vì thiết bị bận. Khi đó, nếu hoán chuyển P_1 ra ngoài và đưa tiến trình P_2 vào thế chỗ, thao tác vào/ra có thể ghi vào vùng bộ nhớ mà bây giờ đã cấp cho P_2. Có thể khắc phục vấn đề này bằng cách, đặt bộ đệm vào/ra trong khu vực nhớ của HĐH. Khi đó, sự trao đổi giữa HĐH và bộ nhớ tiến trình chỉ xảy ra khi tiến trình được chuyển vào trong.

9.4. CẤP PHÁT LIÊN TỤC

Bộ nhớ thường được chia thành hai vùng là vùng dành cho HĐH và vùng của tiến trình người dùng. Vị trí đặt HĐH phụ thuộc vào vị trí bảng

vector ngắt. Không mất tính tổng quát, giả sử HĐH được đặt trong vùng nhớ thấp.

9.4.1. Cấp phát một vùng nhớ liên tục

Không chỉ bảo vệ đoạn mã và dữ liệu của HĐH (không cho tiến trình người dùng thay đổi) mà HĐH phải đảm bảo tiến trình người dùng không thể xâm phạm vào vùng nhớ của nhau. Có thể thực hiện điều này bằng cách sử dụng thanh ghi tái định vị và thanh ghi giới hạn (limit register) như minh họa trong Hình 9.3a. Thanh ghi tái định vị chứa giá trị địa chỉ vật lý nhỏ nhất, còn thanh ghi giới hạn chứa địa chỉ logic lớn nhất. Công việc ánh xạ đã được trình bày trong mục 9.2. Việc bảo vệ được thực hiện bằng cách kiểm tra xem địa chỉ logic có bé hơn thanh ghi giới hạn không. Khi chọn tiến trình A thực thi, trong quá trình chuyển ngữ cảnh, bộ điều phối thiết lập thanh ghi tái định vị và thanh ghi giới hạn tương ứng với tiến trình A. Vì mọi địa chỉ do CPU tạo ra sẽ được kiểm tra, nên cả bộ nhớ HĐH lẫn chương trình và dữ liệu người dùng đều được bảo vệ.

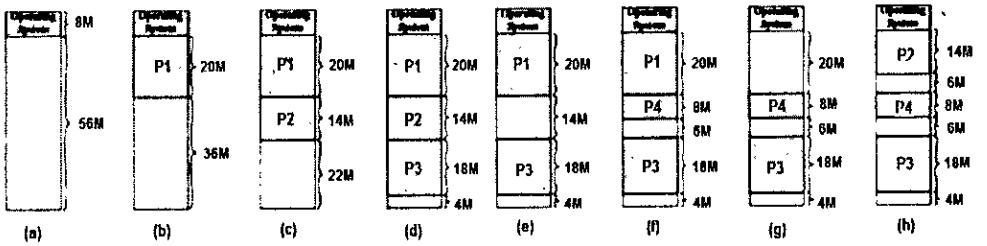
cuu duong than cong . com

9.4.2. Cấp phát nhiều vùng nhớ liên tục

Để chứa được nhiều tiến trình, hệ thống có thể chia bộ nhớ ra nhiều phân vùng có kích thước cố định, mỗi phân vùng chứa duy nhất một tiến trình. Khi phân vùng rỗi, tiến trình nào đó trong hàng đợi nhập sẽ được tải vào. Nếu tiến trình kết thúc, phân vùng tương ứng sẽ được giải phóng để cấp phát cho tiến trình khác. Phương pháp còn được gọi là MFT này lần đầu tiên được cài đặt trên HĐH IBM OS/360, sử dụng chủ yếu trên môi trường xử lý theo lô. Tuy nhiên, nhiều ý tưởng giới thiệu ở đây cũng có thể áp dụng trong môi trường chia sẻ thời gian - môi trường áp dụng phương pháp quản lý bộ nhớ phân đoạn (mục 9.6).

cuu duong than cong . com

HĐH ghi lại thông tin xác định tình trạng các vùng nhớ đã cấp phát hay chưa. Tại thời điểm ban đầu, toàn bộ bộ nhớ trong trạng thái chưa cấp phát, và có thể xem như một khối nhớ khả dụng lớn. Khi cấp phát, HĐH tìm một khối trống đủ lớn cho tiến trình. Nếu tìm thấy, HĐH cấp cho tiến trình một lượng nhớ vừa đủ theo yêu cầu, đồng thời lưu giữ phần bộ nhớ khả dụng còn lại để đáp ứng những yêu cầu bộ nhớ trong tương lai.



Hình 9.4. Ví dụ cấp phát bộ nhớ

Giả sử tại thời điểm (a) hệ thống có 64MB bộ nhớ, HĐH chiếm 8MB đầu tiên, 56MB dành cho tiến trình người dùng như minh họa trong Hình 9.4. Tại thời điểm (b) xuất hiện tiến trình P1 có kích thước 20MB, P1 được đưa vào bộ nhớ, bộ nhớ trống còn 36MB. Tại thời điểm (c) tiến trình P2 có kích thước 14MB xuất hiện và được cấp phát bộ nhớ. Thời điểm (d) tiến trình P3 đến và được cấp phát bộ nhớ, bộ nhớ còn trống 4MB. Nếu tiến trình P4 với kích thước 8MB đến thì bộ nhớ trống không đủ chỗ cho P4. Tuy nhiên, tại thời điểm (e) tiến trình P2 kết thúc và bộ nhớ đủ chỗ chứa P4, do đó tại thời điểm (f), P4 được đưa vào bộ nhớ.

Tiến trình lúc đầu nằm trong hàng đợi nhập. Dựa trên nhu cầu sử dụng bộ nhớ của từng tiến trình và tổng lượng bộ nhớ khả dụng, HĐH xác định tiến trình nào được cấp phát bộ nhớ. Kế tiếp, tiến trình được tải vào bộ nhớ và cạnh tranh quyền sử dụng CPU. Khi kết thúc, tiến trình giải phóng bộ nhớ, phần không gian nhớ này lại được HĐH cấp phát cho tiến trình khác. Tại bất kỳ thời điểm nào, HĐH có danh sách các khối nhớ khả dụng và hàng đợi nhập. HĐH có thể sắp xếp hàng đợi nhập theo thuật toán điều phối. Bộ nhớ lần lượt được cấp phát cho các tiến trình cho đến khi không thể đáp ứng vì không có khối nhớ khả dụng nào đủ lớn. HĐH có thể đợi cho đến khi có khối nhớ khả dụng đủ lớn, hoặc tìm tiếp trong hàng đợi nhập để lựa chọn tiến trình có nhu cầu bộ nhớ ít hơn.

Có thể có nhiều khoảng trống với kích cỡ khác nhau nằm rải rác trong bộ nhớ. Khi cần cấp phát, hệ thống tìm kiếm một khoảng trống đủ lớn để cấp. Khoảng trống này bị tách ra, một phần cấp cho tiến trình; phần còn lại trở thành khoảng trống mới. Tiến trình khi kết thúc sẽ giải phóng vùng nhớ được cấp phát và vùng nhớ này lại được đánh dấu là khoảng trống. Có thể hợp nhất hai khoảng trống nằm kề nhau thành khoảng trống lớn hơn.

Thủ tục này là ví dụ điển hình của vấn đề cấp phát tài nguyên: Làm thế nào để đáp ứng yêu cầu bộ nhớ có kích thước N từ danh sách khoảng trống? Tùy theo tiêu chí nào đó, hệ thống tìm kiếm trên tập hợp các khoảng trống để lựa chọn khoảng trống tối ưu.

- **First-fit**: Cấp phát khoảng trống có kích thước đủ lớn đầu tiên. Việc tìm kiếm bắt đầu từ khoảng trống đầu tiên trong danh sách, hoặc ngay sau khoảng trống vừa được chọn trước đó. Quá trình tìm kiếm kết thúc ngay sau khi tìm thấy.
- **Best-fit**: Cấp phát khoảng trống đủ lớn nhỏ nhất. Phương pháp này tạo ra khoảng trống còn lại nhỏ nhất.
- **Worst-fit**: Cấp phát khoảng trống lớn nhất. Phương pháp này tạo ra khoảng trống còn lại lớn nhất.

Có thể cải tiến hiệu suất tìm kiếm của Best-fit và Worst-fit bằng cách sắp xếp khoảng trống theo kích thước. Các kết quả mô phỏng cho thấy thuật toán First-fit và Best-fit chạy nhanh và tận dụng bộ nhớ tốt hơn so với Worst-fit. Không thuật toán nào trong hai thuật toán First-fit và Best-fit thực sự vượt trội hơn về khả năng tận dụng bộ nhớ, tuy nhiên, thuật toán First-fit nhìn chung chạy nhanh hơn Best-fit.

9.4.3. Phân mảnh ngoài và phân mảnh trong

Khi nhiều tiến trình được tải vào, rồi sau đó giải phóng bộ nhớ, không gian bộ nhớ trống bị phân thành nhiều mảnh nhỏ. *Phân mảnh ngoài* là hiện tượng khi tổng lượng bộ nhớ trống đủ lớn để đáp ứng một yêu cầu nào đó, nhưng các khoảng trống không liên tục mà rải rác trên toàn bộ nhớ. Xét minh họa trên Hình 9.4, giả sử sau thời điểm e, có tiến trình kích thước 16MB đến, hệ thống có 2 khoảng trống 14MB và 4MB với tổng dung lượng 18MB, nhưng lại không thể cấp phát cho tiến trình 16MB. Trong trường hợp xấu nhất, giữa mọi cặp tiến trình luôn xuất hiện một khối trống có kích thước quá bé. Nếu hợp nhất được tất cả các khoảng trống này thành một khối nhớ trống lớn, hệ thống có thể cấp phát cho nhiều tiến trình. Mức độ phân mảnh ngoài còn phụ thuộc vào tổng dung lượng bộ nhớ và kích thước trung bình của các tiến trình. Ví dụ, từ những phân tích thống kê về thuật toán first-fit cho thấy, cứ trong N khối nhớ được cấp phát thì N/2 khối nhớ

sẽ không sử dụng được do hiện tượng phân mảnh. Vậy, có tới 1/3 không gian bộ nhớ bị lãng phí. Đặc điểm này được gọi là luật 50%.

Xét một khoảng trống 8464 byte. Giả sử tiến trình kế tiếp yêu cầu 8462 byte bộ nhớ. Nếu cấp phát khối nhớ đúng theo yêu cầu thì sẽ thừa ra một khoảng trống 2 byte. Chi phí quản lý khoảng trống 2 byte này lớn hơn rất nhiều so với chính giá trị khoảng trống 2 byte đem lại. Đây chính là hiện tượng *phân mảnh trong.*

Giải pháp khắc phục hiện tượng phân mảnh ngoài là thu gọn (compaction): dồn các khoảng trống rải rác thành một khoảng lớn. Kỹ thuật này chỉ áp dụng được nếu quá trình tái định vị động được thực hiện ở thời điểm thực thi (vì chỉ cần thay đổi giá trị thanh ghi tái định vị). Trong trường hợp đơn giản, hệ thống dồn tất cả tiến trình về một đầu và dồn tất cả các khoảng trống về đầu kia bộ nhớ để tạo nên một khoảng trống khả dụng lớn. Để giảm chi phí, có thể chỉ thu gọn một phần tạo ra khoảng trống vừa đủ đáp ứng ngay lập tức yêu cầu của tiến trình nào đó.

Có thể sử dụng hoán chuyển cùng với thu gọn. Tiến trình có thể bị chuyển từ bộ nhớ chính ra ổ cứng và sau đó chuyển về bộ nhớ chính. Khi chuyển ra, bộ nhớ do tiến trình chiếm giữ được hệ thống cấp phát cho tiến trình khác. Tuy nhiên, khi đưa tiến trình trở lại bộ nhớ chính, có một vấn đề nảy sinh. Nếu sử dụng kỹ thuật tái định vị tĩnh, tiến trình phải quay lại đúng vùng bộ nhớ mà trước đây nó sử dụng. Như vậy, có thể một vài tiến trình phải được đưa ra để tạo khoảng trống theo yêu cầu. Nếu sử dụng kỹ thuật tái định vị động, tiến trình có thể được đưa vào bất kỳ vùng nhớ nào. Trong trường hợp này HĐH tìm một khoảng trống, dùng kỹ thuật thu gọn nếu thấy cần thiết rồi tải tiến trình vào. Một giải pháp đối với kỹ thuật thu gọn là đưa những tiến trình cần di chuyển ra ngoài, sau đó tải lại vào vị trí khác trong bộ nhớ. Nếu kỹ thuật hoán chuyển đã được tích hợp vào hệ thống thì có thể dễ dàng cài đặt thêm kỹ thuật thu gọn.

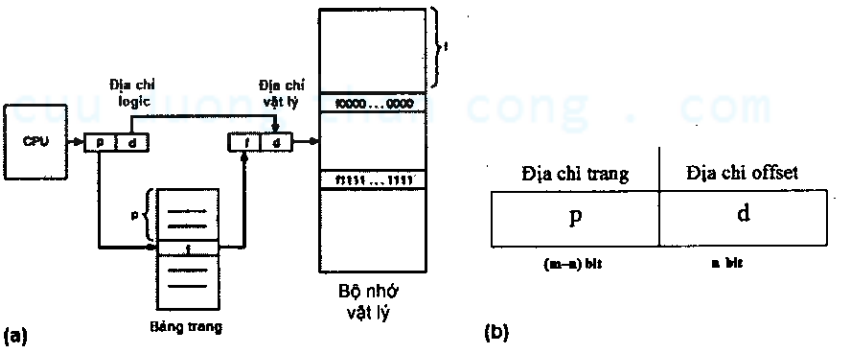
9.5. PHÂN TRANG

Hệ thống theo phương pháp phân trang cho phép không gian địa chỉ logic của tiến trình không nằm liên tục trong bộ nhớ vật lý, như thế có thể tải tiến trình vào bộ nhớ nếu bộ nhớ còn đủ chỗ. Phân trang khắc phục được nhiều vấn đề của các phương pháp quản lý bộ nhớ trước đây, chẳng hạn

phân mảnh ngoài trên cả bộ nhớ lẫn ổ cứng ngoài. Hiện tượng phân mảnh trên ổ đĩa còn nghiêm trọng hơn vì khó áp dụng kỹ thuật thu gọn.

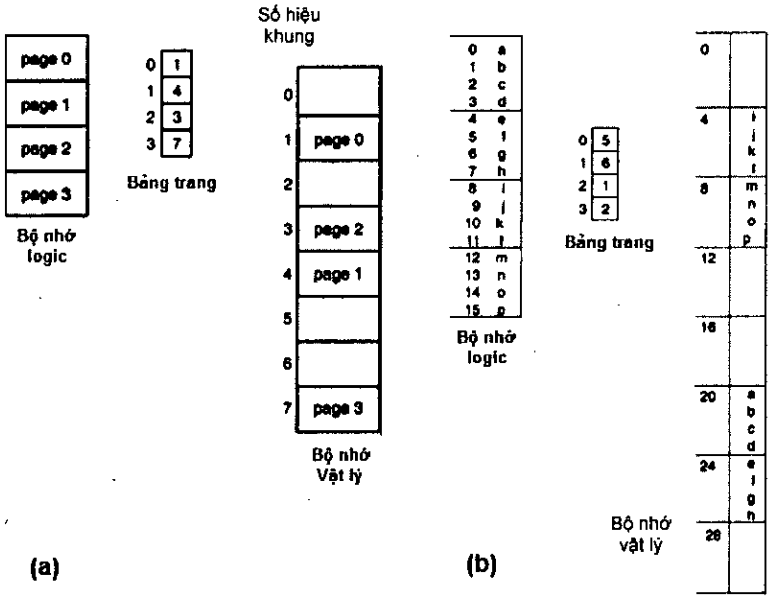
9.5.1. Phương pháp cơ bản

Bộ nhớ vật lý được chia thành các khung trang có kích thước cố định. Bộ nhớ logic cũng được chia thành các trang (page). Kích thước trang và khung trang bằng nhau. Trước khi thực thi, các trang của tiến trình nằm trên ổ đĩa sẽ được tải vào bất kỳ khung trang chưa sử dụng nào của bộ nhớ. Ổ cứng cũng được chia thành các khối có kích thước bằng kích thước khung trang. Hình 9.5 minh họa phần cứng hỗ trợ phân trang. Địa chỉ CPU tạo ra được chia thành hai phần là địa chỉ trang (p) và địa chỉ tương đối trong trang (d). Địa chỉ trang được sử dụng làm chỉ mục đến bảng trang (page table). Bảng trang lưu trữ địa chỉ cơ sở của mỗi trang trong bộ nhớ vật lý. Địa chỉ cơ sở cộng với địa chỉ tương đối trong trang tạo ra địa chỉ vật lý (địa chỉ tuyệt đối).



Hình 9.5. Phần cứng hỗ trợ phân trang

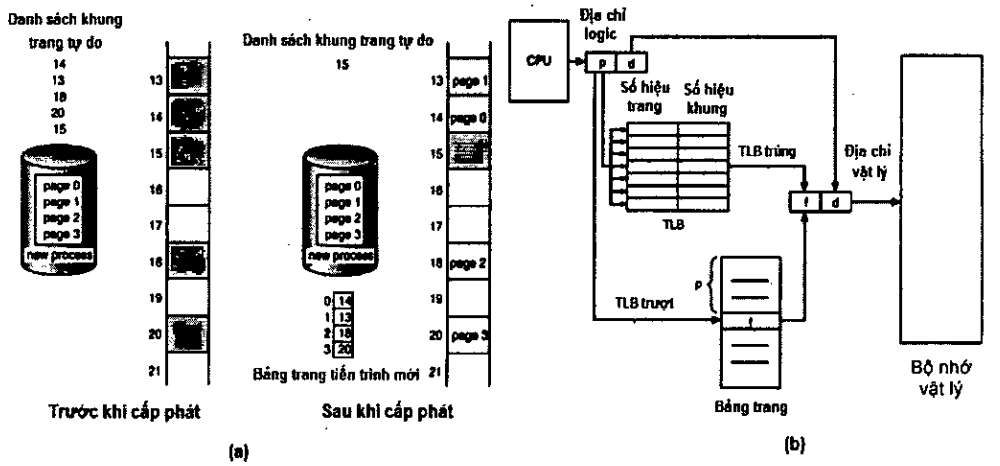
Kích thước trang được kiến trúc phần cứng quy định, thường là lũy thừa của 2, biến thiên từ 512 byte đến 16MB. Nếu kích thước không gian địa chỉ logic là 2^m, kích thước trang là 2^n đơn vị (byte hoặc word), thì (m – n) bit cao của địa chỉ logic xác định số hiệu trang và n bit thấp xác định địa chỉ tương đối trong trang (Hình 9.5b). Ví dụ, xét bộ nhớ ở Hình 9.6b. Giả sử trang có kích thước 4 byte và không gian bộ nhớ vật lý 32 byte (8 trang). Địa chỉ logic 0 ứng với trang 0, địa chỉ tương đối 0. Trong bảng trang, trang 0 ở khung 5. Như vậy, địa chỉ logic 0 ứng với địa chỉ vật lý 20 (5 × 4 + 0). Địa chỉ logic 3 (trang 0, địa chỉ tương đối 3) ứng với địa chỉ vật lý 23 (5 × 4 + 3).



Hình 9.6

Phân trang là trường hợp tái định vị động (phần cứng biến đổi (hay ánh xạ) địa chỉ logic với địa chỉ vật lý). Phân trang tổng quát hóa kỹ thuật trình bày trong mục 9.4 bằng cách sử dụng bảng chứa nhiều thanh ghi tái định vị, mỗi thanh ghi ứng với một khung trang. Với phân trang, hiện tượng phân mảnh ngoài không xuất hiện (bất cứ khung trang tự do nào cũng có thể được cấp phát cho tiến trình). Tuy nhiên, vẫn có hiện tượng phân mảnh trong. Vì khung trang là đơn vị cấp phát cơ sở, nên nếu nhu cầu bộ nhớ của một tiến trình không là bội số của kích thước trang, khung trang cuối cùng có thể không được sử dụng hết. Ví dụ, nếu kích thước trang là 1024 byte thì tiến trình 2049 byte sẽ cần 2 trang cộng với 1 byte. Hệ thống vẫn phải cấp phát 3 khung trang cho tiến trình, dẫn tới sự phân mảnh trong trên khung trang cuối cùng. Nếu kích thước tiến trình độc lập với kích thước trang, mức độ phân mảnh trong trung bình của một tiến trình là một nửa trang. Điều này dẫn đến đề xuất giảm kích thước trang. Tuy nhiên, có thể giảm chi phí phụ trội để quản lý bảng trang bằng cách tăng kích thước trang. Cũng như vậy, các thao tác vào/ra trên ổ đĩa sẽ hiệu quả hơn nếu khối lượng dữ liệu trao đổi lớn.

Khi chuẩn bị thực thi, HĐH kiểm tra kích thước tiến trình (số lượng trang nhớ). Mỗi trang nhớ cần một khung trang nên tiến trình n trang cần n khung trang. Lần lượt các trang của tiến trình được nạp vào khung nào đó trong số các khung được cấp phát, và số hiệu khung được đặt vào hàng tương ứng trong bảng trang của tiến trình (Hình 9.7a).



Hình 9.7. Bảng trang

Phân trang tách bạch quan niệm về bộ nhớ của người dùng với vị trí chương trình nằm trong bộ nhớ vật lý. Lập trình viên xem chương trình của mình là một không gian liên tục và bộ nhớ chứa duy nhất chương trình của mình. Trên thực tế, chương trình nằm rải rác trong bộ nhớ vật lý, và trong bộ nhớ vật lý cũng chứa nhiều chương trình khác. Quá trình chuyển đổi địa chỉ che dấu sự khác biệt này và hoàn toàn "trong suốt" với người dùng. Với trách nhiệm quản lý bộ nhớ vật lý, HĐH phải xác định khung trang nào được cấp phát, khung trang nào còn tự do (chưa cấp phát),... Các thông tin này được lưu giữ trong cấu trúc dữ liệu gọi là bảng khung. Trong bảng khung, mỗi hàng ứng với một khung trang vật lý và xác định tình trạng đã cấp phát hay tự do của khung trang và trong trường hợp đã cấp phát thì khung được cấp phát cho trang nào của tiến trình nào. HĐH phải ghi nhớ bảng trang của tiến trình (giống như ghi nhớ nội dung các thanh ghi cơ sở). Bộ điều phối sử dụng bảng trang để khởi tạo lại phần cứng thực hiện ánh xạ ngay trước khi tiến trình bắt đầu thực thi. Chính vì thế, phân trang làm tăng thời gian chuyển ngữ cảnh giữa các tiến trình.

9.5.2. Cấu trúc bảng trang

Ngữ cảnh hoạt động của tiến tình: Con trỏ đến bảng trang riêng của tiến trình cũng như giá trị các thanh ghi trong quá trình thực thi được HĐH lưu trong khối điều khiển tiến trình. Khi chọn tiến trình thực thi, bộ điều phối phải khôi phục lại ngữ cảnh của tiến trình.

☞ ***Phần cứng***

Có nhiều cách cài đặt, nhưng đơn giản nhất là đặt bảng trang trong các thanh ghi chuyên dụng có tốc độ đọc/ghi cao. Do đó, có thể nhanh chóng xác định trang nằm trong khung trang nào. Mỗi truy xuất đến bộ nhớ đều phải thông qua bảng trang, do đó tốc độ là yếu tố quan trọng nhất. Khi chuyển ngữ cảnh, bộ điều phối CPU nạp lại các thanh ghi này (giống như nạp lại các thanh ghi khác của CPU). Các chỉ thị nạp hoặc thay đổi nội dung thanh ghi chuyên dụng là chỉ thị đặc quyền, chỉ HĐH mới được phép thay đổi ánh xạ bộ nhớ. DEC PDP-11 áp dụng công nghệ này (địa chỉ 16 bit và kích thước trang 8KB). Như vậy, bảng trang có 8 dòng được lưu giữ trong các thanh ghi chuyên dụng. Có thể sử dụng các thanh ghi làm bảng trang nếu bảng trang nhỏ vừa phải (256 hàng). Nếu bảng trang rất lớn thì không thể sử dụng thanh ghi chuyên dụng nữa, mà phải đặt bảng trang trong bộ nhớ chính, và thanh ghi bảng trang cơ sở (PRBR) trỏ đến bảng trang của tiến trình. Thay đổi bảng trang chỉ cần thay đổi một thanh ghi, về căn bản giảm được thời gian chuyển ngữ cảnh.

Hiệu suất giải pháp này không cao, vì mỗi lần đọc/ghi phải truy xuất bộ nhớ hai lần. Bước thứ nhất, phải truy cập tới bảng trang để xác định số hiệu khung trang (bảng trang được xác định qua thanh ghi PTBR). Bước thứ hai, số hiệu khung được cộng với địa chỉ tương đối để tạo ra địa chỉ vật lý thực sự. Do đó, tốc độ truy xuất bộ nhớ bị giảm đi hai lần.

Có thể khắc phục bằng cách đặt một phần bảng trang trên bộ nhớ kết hợp có tốc độ rất cao (Translation Look-aside Buffer - TLB). Bộ nhớ này gồm nhiều thanh ghi, mỗi thanh ghi gồm hai phần là khóa và giá trị. Khóa ứng với số hiệu trang trong khi giá trị ứng với số hiệu khung trang. Khi hệ thống đưa ra một khóa, khóa này đồng thời được so sánh với khóa của tất cả các thanh ghi. Nếu trùng với một khóa nào đó, trường giá trị tương ứng là kết quả cần tìm. Tốc độ tìm kiếm trên TLB rất nhanh, tuy nhiên chi phí cài đặt phần cứng khá đắt. Thông thường, TBL có từ 8 đến 20410 thanh ghi. TLB chỉ chứa một số hàng trong bảng trang. Với một địa chỉ logic, HĐH sử dụng số hiệu trang trong địa chỉ làm khóa tìm kiếm trên TLB. Nếu tìm thấy, HĐH xác định ngay được số hiệu khung trang tương ứng. Nếu không tìm thấy thì phải tiếp tục tìm kiếm trên bảng trang nằm trong bộ nhớ. Sau khi xác định được số hiệu khung thì có thể truy xuất tới bộ nhớ. Có thể bổ sung

số hiệu trang và số hiệu khung vào TLB để các tham chiếu sau được xác định nhanh hơn (Hình 9.7b). Nếu TBL đầy, HĐH phải lựa chọn một số hàng để thay thế. Khi chuyển ngữ cảnh phải xóa toàn bộ TBL để đảm bảo tiến trình được thực thi kế tiếp không sử dụng thông tin của tiến trình cũ.

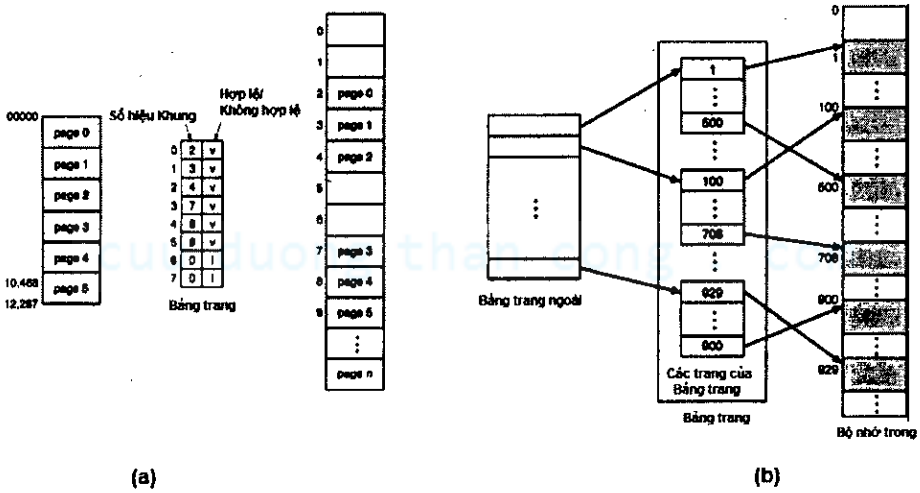
Tỷ lệ trúng (hit ratio) là tỷ lệ tìm thấy số hiệu trang trong TLB, 80% nghĩa là tìm thấy số hiệu trang mong muốn trong 80% số lần tìm kiếm. Nếu tìm kiếm trên TLB mất 20ns và mất 100ns truy xuất bộ nhớ, thì truy xuất bộ nhớ khi có TLB mất 120ns trong trường hợp tìm thấy số hiệu trang trên TLB. Nếu không tìm thấy số hiệu trang (vẫn mất 20ns tìm kiếm), thì trước hết phải truy xuất tới bảng trang trong bộ nhớ để tìm số hiệu khung (100ns), sau đó mới truy xuất byte mong muốn ở bộ nhớ (100ns), tổng cộng mất 220ns. Để xác định thời gian truy xuất bộ nhớ hiệu dụng (effective memory-access time), phải xét đến trọng số xác suất xuất hiện của từng trường hợp. Thời gian truy xuất hiệu dụng là 0,8 × 120 + 0,20 × 220 = 140ns. Như vậy, thời gian truy xuất bộ nhớ chậm mất 40ns (từ 100ns đến 140ns). Với tỷ lệ trúng là 98%, thì thời gian truy cập hiệu dụng là 0,98 × 120 + 0,02 × 220 = 122ns. Tỷ lệ trúng khá cao này cũng chỉ giúp giảm tốc độ truy xuất bộ nhớ 22%. Hiển nhiên, tỷ lệ trúng có liên quan đến số lượng các thanh ghi liên kết. Khi số lượng này từ 16 đến 512, tỷ lệ trúng có thể từ là 80 đến 98%. CPU Motorola 68030 (dùng trong hệ thống Apple Macintosh) có 22 thanh ghi TBL, CPU Intel 80486 có 32 thanh ghi và tỷ lệ trúng đạt tới 98%.

☞ ***Bảo vệ***

Mỗi khung trang có 1 bit bảo vệ, thường được lưu trong bảng trang. Ngoài ra, có thể sử dụng thêm một bit để xác định trang có thuộc tính đọc/ghi hay chỉ đọc. Mỗi tham chiếu đến bộ nhớ phải đi qua bảng trang để tìm số hiệu khung trang. Song song với việc xác định số hiệu khung, hệ thống kiểm tra bit bảo vệ để ngăn cản thao tác ghi trên trang có thuộc tính chỉ đọc. Thao tác ghi như vậy gây ra lỗi phần cứng và quyền điều khiển được chuyển cho HĐH.

Hệ thống có thể tạo ra phần cứng cung cấp chế độ bảo vệ chỉ đọc, đọc-ghi, hoặc chỉ thực thi. Những thao tác không hợp lệ sẽ gây ra lỗi chuyển quyền điều khiển cho HĐH. Thông thường, hệ thống gắn thêm một bit, gọi là bit hợp lệ /không hợp lệ cho mỗi hàng của bảng trang. Bit này có giá trị "hợp lệ" (v) khi trang tương ứng nằm trong vùng địa chỉ logic của tiến trình;

có giá trị "không hợp lệ" (i) trong trường hợp ngược lại. Ví dụ, trong hệ thống với không gian địa chỉ 14 bit (0 → 16383), có thể có một chương trình chỉ sử dụng các địa chỉ từ 0 → 10468. Với trang kích thước 2KB, ta có tình huống minh họa trong Hình 9.8a. Các địa chỉ ở các trang 0, 1, 2, 3, 4 và 5 được ánh xạ bình thường qua bảng trang. Tuy nhiên, truy cập tới địa chỉ ở trang 6 hoặc 7 sẽ bị lỗi do tham chiếu trang không hợp lệ. Do kích thước chương trình là 10468, tham chiếu vượt quá giá trị này về mặt logic là không hợp lệ. Tuy nhiên, tham chiếu đến trang 5 vẫn được xem là hợp lệ, do đó truy xuất đến địa chỉ 10469 → 12287 là hợp lệ (chỉ địa chỉ từ 12288 đến 16383 mới không hợp lệ). Đây chính là hiện tượng phân mảnh trong của việc phân trang.



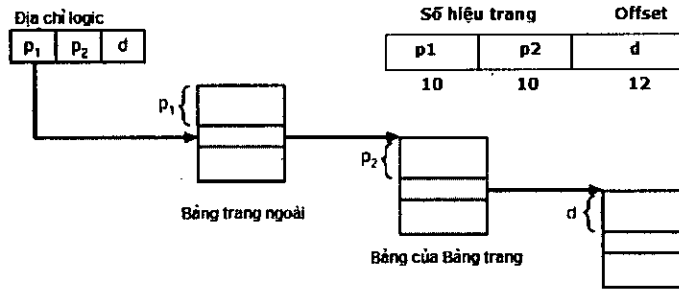
Hình 9.8. Ví dụ bảng trang

Hiếm khi tiến trình sử dụng toàn bộ mà chỉ sử dụng một phần của không gian địa chỉ. Khi đó, nếu mỗi trang trong dải địa chỉ chiếm một dòng tương ứng trong bảng trang sẽ gây nên tình trạng lãng phí. Một số hệ thống có thêm thanh ghi độ dài bảng trang (PTLR) để chỉ kích thước bảng trang. Giá trị này được kiểm tra trong mỗi lần truy xuất bộ nhớ để đảm bảo truy xuất nằm trong dải địa chỉ hợp lệ.

9.5.3. Phân trang đa mức

Phần lớn hệ thống máy tính hiện đại hỗ trợ không gian địa chỉ lớn (2^{32} đến 2^{64}), khi đó kích thước bảng trang trở nên rất lớn. Chẳng hạn, nếu không gian địa chỉ logic 32 bit, kích thước trang là 4KB (2^{12} byte) thì bảng

trang có thể có tới một triệu hàng ($2^{32}/2^{12}$). Mỗi hàng gồm 4 byte, nên mỗi tiến trình có thể cần tới 4MB bộ nhớ làm bảng trang. Tất nhiên, không thể cấp phát bảng trang trong một khu vực bộ nhớ liên tục mà nên chia bảng trang thành nhiều bảng nhỏ.



Hình 9.9. Phân trang đa mức

Thông thường, hệ thống sử dụng phương pháp phân trang hai mức: phân trang cho chính bảng trang. Xét hệ thống 32 bit địa chỉ ở trên và trang có kích thước 4KB. Địa chỉ logic được chia thành hai phần là số hiệu trang 20 bit và địa chỉ tương đối trong trang 12 bit. Do phân trang bảng trang nên số hiệu trang cũng được chia thành hai phần là P1 (10 bit) là chỉ mục đến bảng trang ngoài và P2 (10 bit) là độ dịch chuyển trong trang của bảng trang ngoài.

Phương pháp chuyển đổi địa chỉ cho kiến trúc này được minh họa trên Hình 9.9. Kiến trúc VAX hỗ trợ phân trang 2 mức. VAX có bus địa chỉ 32 bit và kích thước trang 512 byte. Không gian địa chỉ logic của tiến trình được chia thành bốn đoạn bằng nhau, mỗi đoạn gồm 2^{30} byte. Mỗi đoạn ứng với một phần không gian địa chỉ logic khác nhau của tiến trình (2 bit cao đầu tiên của địa chỉ logic xác định đoạn, 21 bit tiếp theo là số hiệu trang logic nằm trong đoạn, 9 bit cuối cùng là địa chỉ tương đối trong trang mong muốn). Bằng cách phân chia bảng trang như vậy, HĐH có thể bỏ những đoạn không sử dụng cho đến khi tiến trình cần đến chúng. Giả sử hệ thống với không gian địa chỉ 64 bit, kích thước trang 4KB, như vậy bảng trang sẽ có 2^{52} dòng. Nếu sử dụng phương pháp phân trang hai mức, thì mỗi bảng trang nên nằm trong đúng một trang, tức là một trang sẽ có 2^{10} hàng, mỗi hàng 4 byte. Bảng trang ngoài sẽ gồm 2^{42} hàng, tức là 2^{44} byte = 16GB. Rõ ràng, không thể cấp phát bảng trang ngoài trên một khu vực nhớ liên tục lớn như vậy. Như vậy, phải chia bảng trang ngoài ra thành nhiều bảng nhỏ. Giải

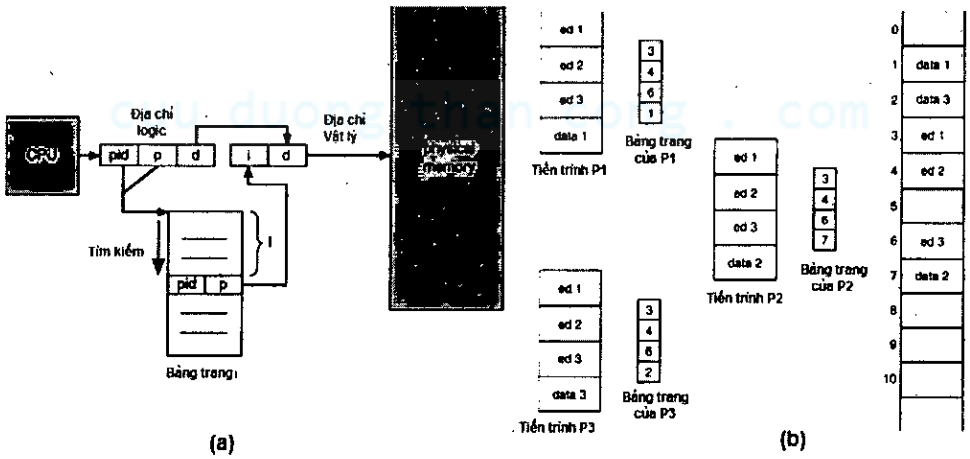
pháp này đôi khi được áp dụng trên một vài bộ vi xử lý 32 bit để tăng tính linh hoạt và hiệu quả. Có một số hệ thống áp dụng nhiều mức, chẳng hạn kiến trúc SPARC (với địa chỉ 32 bit) hỗ trợ phương pháp phân trang 3 mức, kiến trúc Motorola 68030 32 bit hỗ trợ phương pháp phân trang 4 mức.

Phân trang đa mức làm suy giảm hiệu suất hệ thống. Giả sử bảng trang ở mỗi mức đều nằm trong bộ nhớ, chuyển địa chỉ logic sang địa chỉ vật lý có thể cần bốn lần truy xuất bộ nhớ. Thời gian truy xuất bộ nhớ tăng gấp năm lần. Tuy nhiên, nếu có cache hỗ trợ thì hiệu suất có thể được đảm bảo. Giả sử tỷ lệ trúng là 98% thì thời gian truy cập hiệu dụng là

$$0{,}98 \times 120 + 0{,}02 \times 520 = 128\text{ns}.$$

Do đó, thậm chí với nhiều mức tra cứu bảng trang, thời gian truy xuất bộ nhớ cũng chỉ giảm 28%.

9.5.4. Bảng trang nghịch đảo



Hình 9.10. Bảng trang nghịch đảo và chia sẻ bảng trang

HĐH sử dụng bảng trang để ánh xạ địa chỉ logic sang địa chỉ vật lý cho mỗi tham chiếu bộ nhớ. Bảng trang được sắp xếp theo địa chỉ ảo, HĐH có khả năng xác định ngay lập tức bất kỳ hàng nào. Tuy nhiên, kích thước bảng trang có thể rất lớn, chiếm nhiều bộ nhớ vật lý. Bảng trang nghịch đảo có thể khắc phục phần nào vấn đề này. Mỗi hàng trên bảng trang nghịch đảo ứng với một khung trang của bộ nhớ vật lý và chứa địa chỉ ảo của trang nhớ nằm trong khung trang đó cùng với thông tin về tiến trình sở hữu trang. Hệ thống chỉ có duy nhất một bảng trang, và mỗi khung trang trong bộ nhớ vật

lý ứng với một hàng duy nhất. Hình 9.10a minh họa bảng trang nghịch đảo. Một số hệ thống sử dụng bảng trang nghịch đảo là IBM System/38, IBM RISC System 6000, IBM RT.

Địa chỉ ảo trong hệ thống gồm 3 phần: <ID-tiến trình, số hiệu trang, địa chỉ tương đối>. Mỗi dòng trên bảng trang nghịch đảo là cặp <ID-tiến trình, số hiệu trang>. Khi truy xuất thì <ID-tiến trình, số hiệu trang> được sử dụng làm khóa tìm kiếm trên bảng trang nghịch đảo. Nếu tìm thấy ở dòng i, địa chỉ vật lý được xác định từ <i, địa chỉ tương đối>. Nếu không tìm thấy, thì đó là địa chỉ không hợp lệ. Bảng trang nghịch đảo làm giảm lượng bộ nhớ vật lý cần thiết để lưu giữ các bảng trang. Tuy nhiên, bảng trang nghịch đảo không chứa đầy đủ thông tin về không gian địa chỉ logic của tiến trình. Do đó, tiến trình vẫn cần bảng trang riêng, nhưng bảng trang riêng không cần đặt trong bộ nhớ. Giải pháp này có thể gây ra nhiều lỗi trang. Mặc dù tiết kiệm được bộ nhớ, nhưng thời gian cần thiết để tìm kiếm bảng trang tăng. Bảng trang nghịch đảo được sắp xếp theo địa chỉ vật lý, nhưng tra cứu lại dựa trên địa chỉ ảo, do đó có thể cần phải tìm kiếm rất lâu trên toàn bộ bảng. Để khắc phục, có thể sử dụng bảng băm để hạn chế việc tìm kiếm trên một hoặc một số nhỏ các hàng ở bảng trang. Nhưng truy xuất đến bảng băm là thêm một lần tham chiếu bộ nhớ, vì thế mỗi lần đọc/ghi cần ít nhất hai lần đọc bộ nhớ vật lý (một trên bảng băm và một trên bảng trang). Để cải thiện hiệu suất, có thể sử dụng TLB để lưu các dòng đã được định vị trước.

9.5.5. Chia sẻ trang

Mã có thuộc tính thuần túy (reentrant) là đoạn mã không thay đổi trong suốt quá trình thực thi, do đó có thể được chia sẻ như minh họa trên Hình 9.10b. Ưu điểm khác của phân trang là khả năng chia sẻ đoạn mã chương trình. Xét hệ thống hỗ trợ 40 người dùng và người nào cũng sử dụng chương trình soạn thảo văn bản. Nếu trình soạn thảo gồm 150KB mã chương trình và 50KB dữ liệu thì ta sẽ cần 8000KB cho 40 người dùng. Mỗi tiến trình có bản sao riêng giá trị của các thanh ghi và vùng nhớ riêng lưu trữ dữ liệu (vì dữ liệu của 2 tiến trình sẽ khác nhau). Trong Hình 9.10b, ba tiến trình dùng chung ba trang chứa mã chương trình soạn thảo (kích thước mỗi trang là 50KB), nhưng mỗi tiến trình có trang dữ liệu riêng. Chỉ có duy nhất một bản mã chương trình soạn thảo trong bộ nhớ. Bảng trang của ba tiến trình chứa ánh xạ vào ba khung trang vật lý chứa mã chương trình soạn thảo, nhưng

trang dữ liệu được ánh xạ vào các khung trang khác nhau. Mặc dù có 40 người dùng, hệ thống chỉ cần duy nhất một bản mã chương trình soạn thảo (150KB), cộng thêm 40 trang dữ liệu cho mỗi người dùng. Lượng bộ nhớ cần thiết chỉ còn 2150KB.

```
int tmp:
int f(int i)
{
  tmp = tmp * 2:
  return tmp:
}
```

(a) Mã không thuần túy

```
int f(int i)
{
  int tmp:
  tmp = tmp * 2:
  return tmp:
}
```

(b) Mã thuần túy

Hình 9.11. Mã thuần túy và mã không thuần túy

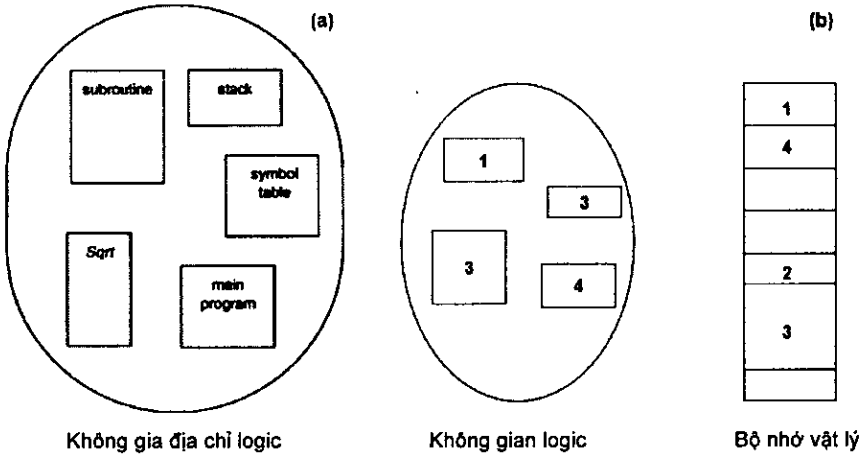
Có thể dùng chung các chương trình được sử dụng thường xuyên như trình biên dịch, hệ thống cửa sổ, hệ quản trị cơ sở dữ liệu,... Để có thể chia sẻ, mã chương trình phải là mã thuần túy. Việc chia sẻ bộ nhớ giữa các tiến trình trong một hệ thống cũng giống cách các thread chia sẻ không gian địa chỉ của tác vụ. Thực hiện chia sẻ bộ nhớ trên những hệ thống sử dụng bảng trang nghịch đảo không dễ, vì việc chia sẻ bộ nhớ thường được thực hiện bằng cách cho phép hai hay nhiều địa chỉ logic ánh xạ đến cùng một địa chỉ vật lý. Nhưng trong phương pháp này, một khung trang vật lý không thể chứa nhiều địa chỉ ảo được.

9.6. PHÂN ĐOẠN

Lập trình viên thường không coi bộ nhớ là dãy tuyến tính các byte mà là tập hợp các đoạn có kích thước khác nhau, thực hiện các chức năng khác nhau, giữa các đoạn không có quan hệ thứ tự xác định (Hình 9.12a).

9.6.1. Phương thức cơ bản

Chương trình gồm chương trình chính với nhiều chương trình con, thủ tục, hàm, module, cùng với các cấu trúc dữ liệu khác. Mỗi thành phần module hoặc dữ liệu được xác định qua tên gọi. Người lập trình nói về "bảng ký hiệu", "hàm sqrt", "biến a" mà không quan tâm tới chúng nằm ở đâu trong bộ nhớ. Kích thước mỗi đoạn mã là khác nhau. Các thành phần trong mỗi đoạn mã được xác định bởi khoảng cách tương đối của chúng với điểm đầu tiên của đoạn (câu lệnh đầu tiên của chương trình, ký hiệu thứ 17 trong bảng ký hiệu, câu lệnh thứ 5 trong hàm sqrt,...).



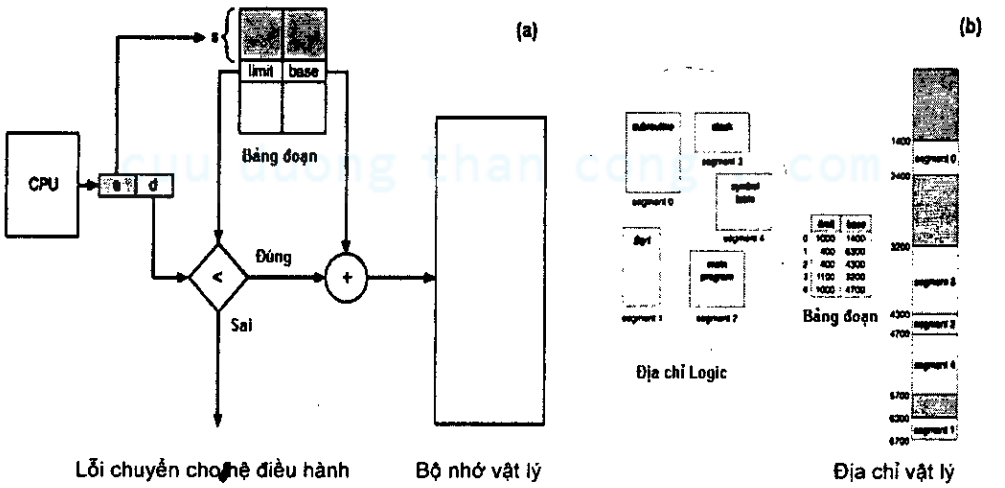
Hình 9.12. Minh họa phân đoạn

Trong phương pháp phân đoạn, không gian địa chỉ logic là tập hợp các đoạn có tên và kích thước xác định. Địa chỉ tuyệt đối xác định qua tên đoạn và khoảng cách tương đối trong đoạn (trong phân trang, người dùng đưa ra một địa chỉ duy nhất, và phần cứng sẽ chia địa chỉ này thành số hiệu trang và khoảng cách tương đối trong trang). Để đơn giản, các đoạn được đánh số và dùng số hiệu đoạn thay cho tên gọi. Do đó, địa chỉ logic gồm hai thành phần là <số hiệu đoạn, địa chỉ tương đối trong đoạn>. Khi biên dịch, trình biên dịch sẽ tự động tạo ra các phân đoạn tương ứng với chương trình nguồn. Trình biên dịch Pascal có thể tạo các đoạn khác nhau cho: (1) các biến toàn cục; (2) ngăn xếp để thủ tục sử dụng lưu giữ các tham số và địa chỉ trả về; (3) đoạn mã của mỗi thủ tục hoặc hàm; (4) các biến cục bộ của mỗi thủ tục và hàm. Trình biên dịch Fortran có thể tạo đoạn riêng cho khối mã được dùng nhiều lần. Mảng cũng có thể được đưa vào các đoạn riêng rẽ. Bộ tải sẽ tải tất cả những đoạn này và gán số hiệu đoạn cho chúng.

9.6.2. Phần cứng

Mặc dù người dùng có thể xác định một đối tượng trong chương trình qua địa chỉ đoạn và địa chỉ tương đối trong đoạn, nhưng bộ nhớ vật lý vẫn chỉ là chuỗi byte. Do đó, cần có phương thức chuyển địa chỉ hai thành phần thành địa chỉ vật lý. Việc ánh xạ được thực hiện bằng bảng phân đoạn. Mỗi hàng của bảng gồm Địa chỉ cơ sở đoạn (base) xác định địa chỉ vật lý của byte đầu tiên trong đoạn và Giới hạn đoạn (limit) xác định kích thước đoạn như minh họa trong Hình 9.13a.

Hình 9.13a minh họa cách sử dụng bảng phân đoạn. Địa chỉ logic gồm số hiệu đoạn (s) và địa chỉ tương đối trong đoạn (d); s được sử dụng làm chỉ mục đến bảng phân đoạn, d phải nằm trong khoảng từ 0 đến giới hạn đoạn. Nếu không thỏa mãn thì đây là lỗi địa chỉ logic vượt ra khỏi giới hạn đoạn và HĐH sẽ chiếm lấy quyền sử dụng CPU. Nếu thỏa mãn, d được cộng với địa chỉ cơ sở đoạn để tạo ra địa chỉ vật lý thực sự. Có thể coi bảng phân đoạn là mảng các cặp thanh ghi cơ sở - thanh ghi giới hạn. Ví dụ, xét trường hợp trên Hình 9.13b. Ta có 5 phân đoạn nằm trong bộ nhớ vật lý và được đánh số từ 0 tới 4. Mỗi hàng trong bảng phân đoạn ứng với một phân đoạn, cung cấp địa chỉ bắt đầu (địa chỉ cơ sở) của đoạn trong bộ nhớ vật lý, kích thước đoạn (giới hạn đoạn). Ví dụ, kích thước đoạn 2 là 400 byte, địa chỉ bắt đầu là 4300. Do đó, tham chiếu tới byte 53 của phân đoạn 2 được ánh xạ tới vị trí 4300 + 53 = 4353. Tham chiếu tới byte 1222 của phân đoạn 0 gây ra lỗi vì kích thước của phân đoạn chỉ có 1000 byte.



Hình 9.13. Phần cứng phân đoạn và ví dụ

9.6.3. Cài đặt bảng phân đoạn

Kỹ thuật phân đoạn giống mô hình quản lý bộ nhớ phân vùng, điểm khác biệt là chương trình có thể có nhiều đoạn. Giống bảng phân trang, bảng phân đoạn có thể nằm trên các thanh ghi tốc độ truy xuất cao hoặc trong bộ nhớ chính. Nếu sử dụng thanh ghi, tốc độ truy xuất tới bảng phân đoạn giảm. Hơn thế nữa, khả năng thực hiện đồng thời thao tác cộng địa chỉ tương đối với địa chỉ cơ sở để tạo ra địa chỉ vật lý và thao tác so sánh với kích thước giới hạn đoạn cũng làm tăng tốc độ truy xuất. Tuy nhiên, nếu

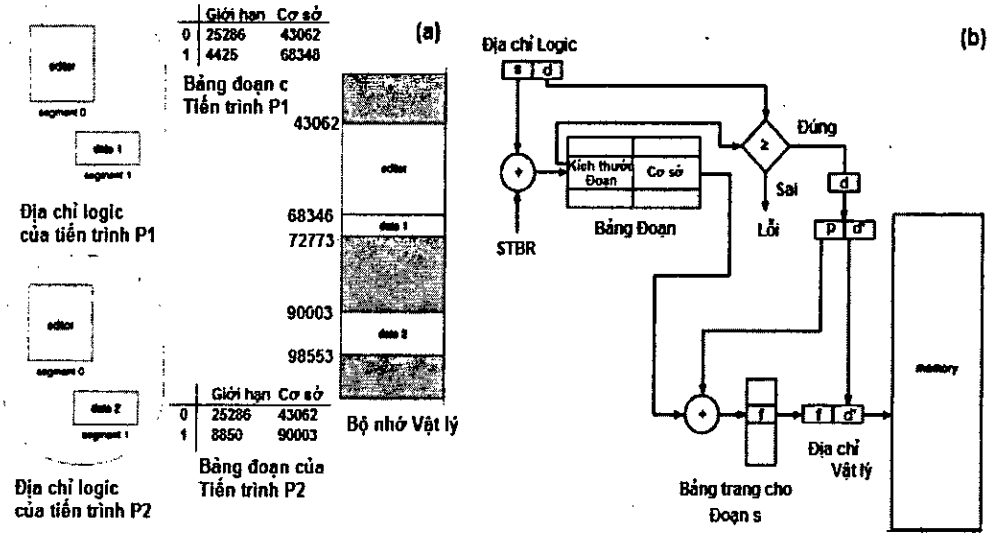
chương trình có nhiều đoạn thì bảng phân đoạn phải được đặt trong bộ nhớ. Thanh ghi cơ sở bảng phân đoạn (Segment-table base register - STBR) trỏ đến bảng đoạn. Vì số đoạn trong mỗi chương trình có thể khác nhau, nên cần thêm thanh ghi kích thước bảng phân đoạn STLR (Segment-table length register). Với địa chỉ logic (s, d), HĐH kiểm tra xem số hiệu đoạn s có hợp lệ không (tức là s < STLR). Sau đó cộng s với STBR, được kết quả (STBR + s) là chỉ mục của đoạn trong bảng phân đoạn. Hàng ứng với chỉ mục này được đọc và xử lý như sau: Dựa trên kích thước đoạn để kiểm tra xem địa chỉ tương đối trong đoạn có hợp lệ không (d < kích thước đoạn). Nếu hợp lệ thì địa chỉ vật lý được xác định bằng cách cộng địa chỉ cơ sở của đoạn với địa chỉ tương đối trong đoạn. Giống phân trang, ánh xạ này cũng cần tới hai tham chiếu bộ nhớ cho mỗi địa chỉ logic, sẽ làm hệ thống máy tính chậm đi hai lần. Có thể sử dụng TLB để lưu lại các hàng được sử dụng thường xuyên nhất.

9.6.4. Bảo vệ và chia sẻ

Ưu điểm của kỹ thuật phân đoạn là cơ chế bảo vệ. Mặc dù việc sử dụng các đoạn tương tự nhau, tuy nhiên có đoạn chứa chỉ thị, có đoạn chứa dữ liệu. Trong kiến trúc hiện đại, các chỉ thị mang tính chất không tự sửa đổi, nên đoạn chỉ thị có thể được định nghĩa là chỉ đọc hoặc chỉ thực thi. Phần cứng thực hiện ánh xạ bộ nhớ sẽ kiểm tra bit bảo vệ trong mỗi hàng của bảng đoạn để ngăn chặn truy xuất bất hợp lệ vào bộ nhớ (như cố tình ghi lên đoạn chỉ đọc, hoặc sử dụng đoạn có thuộc tính chỉ thực thi làm dữ liệu). Bằng cách đặt mảng trong một đoạn, phần cứng quản lý bộ nhớ sẽ tự động kiểm tra chỉ số mảng có hợp lệ không và ngăn cản truy xuất tới các phần tử nằm bên ngoài mảng.

Kỹ thuật phân đoạn có khả năng chia sẻ mã chương trình và dữ liệu. Bảng đoạn nằm trong khối điều khiển tiến trình. Bảng đoạn được bộ điều phối sử dụng để khởi tạo phần cứng thực hiện ánh xạ mỗi khi tiến trình thực thi. Chia sẻ đoạn được thực hiện bằng cách cho các hàng trong bảng đoạn của hai tiến trình khác nhau trỏ đến cùng một vùng địa chỉ vật lý. Xét ví dụ việc sử dụng trình soạn thảo văn bản trong hệ thống chia sẻ. Một trình soạn thảo hoàn chỉnh có thể gồm nhiều đoạn. Những đoạn này có thể được chia sẻ giữa nhiều người dùng, tiết kiệm đáng kể bộ nhớ vật lý. Thay vì đặt n bản

sao của trình soạn thảo, bây giờ chỉ cần một bản duy nhất trong bộ nhớ. Mỗi người dùng vẫn có những đoạn riêng không chia sẻ để lưu giữ các biến cục bộ.



Hình 9.14. Ví dụ bảng đoạn

Cũng có thể chỉ chia sẻ từng phần chương trình. Ví dụ, các gói chương trình con chung có thể dùng chung giữa nhiều người dùng nếu chúng nằm trong những phân đoạn chia sẻ và có tính chất chỉ đọc. Ví dụ, hai chương trình FORTRAN có thể sử dụng cùng một chương trình con Sqrt, thì chỉ cần một bản sao chương trình con Sqrt nằm trong bộ nhớ. Mặc dù kỹ thuật chia sẻ này có vẻ khá đơn giản, nhưng có những yếu tố rất tinh tế cần tính đến. Những đoạn mã chương trình thường chứa tham chiếu tới chính nó. Ví dụ, tham số của một lệnh nhảy có điều kiện thường là một địa chỉ đích nào đó. Địa chỉ này gồm hai phần: số hiệu đoạn và địa chỉ tương đối trong đoạn. Số hiệu đoạn của địa chỉ đích sẽ là số hiệu của đoạn trên. Như vậy, số hiệu của đoạn mã dùng chung này trong tất cả các tiến trình liên quan phải giống nhau. Ví dụ, chúng ta muốn chia sẻ chương trình con Sqrt, số hiệu của đoạn này trong một tiến trình là 4, trong tiến trình khác lại là 17. Vậy làm thế nào để chương trình con Sqrt có thể tham chiếu đến chính nó? Bởi vì chỉ có duy nhất một bản sao hàm Sqrt trong bộ nhớ vật lý, có thể hàm này tham chiếu tới chính nó bằng cùng một cách thức giống như cả hai tiến trình, nghĩa là phải có một số hiệu đoạn duy nhất, dùng chung cho cả hai tiến trình. Khi số

lượng người dùng tăng, thì khó tìm được một số hiệu phân đoạn chung cho tất cả các tiến trình.

Những đoạn dữ liệu có thuộc tính chỉ đọc và không chứa con trỏ cũng như các đoạn mã chương trình tham chiếu đến chính nó một cách gián tiếp có thể được chia sẻ giữa các tiến trình và trong các tiến trình có thể có số hiệu đoạn khác nhau. Ví dụ, địa chỉ rẽ nhánh trong lệnh nhảy có điều kiện là khoảng cách dịch chuyển so với con đếm chương trình hiện thời. Điều này tránh cho chương trình phải trực tiếp sử dụng số hiệu đoạn khi tham chiếu.

9.6.5. Phân mảnh

Bộ điều phối dài hạn thực hiện cấp phát bộ nhớ cho tất cả các đoạn của tiến trình người dùng. Việc này tương tự như phân trang, ngoại trừ đoạn có kích thước khác nhau, trong khi kích thước trang cố định. Do vậy, phương pháp phân đoạn bộ nhớ với kích thước thay đổi là vấn đề cấp phát động và thường được giải quyết bằng thuật toán best-fit hoặc first-fit. Phân đoạn có thể gây hiện tượng phân mảnh ngoài khi tất cả các khối tự do quá nhỏ để chứa được một phân đoạn nào đó. Khi đó, tiến trình bị phong tỏa cho đến khi hệ thống có thể đáp ứng được hoặc hệ thống phải thực hiện thu gọn bộ nhớ để tạo ra khoảng trống đủ lớn. Hiện tượng phân mảnh ngoài trong phương pháp phân đoạn phụ thuộc chủ yếu vào kích thước đoạn trung bình. Ở một thái cực, có thể đặt mỗi tiến trình trong một phân đoạn. Đây chính là giải pháp phân vùng với độ lớn của vùng không cố định. Ở thái cực khác, mỗi byte có thể được đặt trong một đoạn và có khả năng tái định vị độc lập. Khi đó triệt tiêu được phân mảnh ngoài; tuy nhiên, mỗi byte lại cần tới một thanh ghi cơ sở để tái định vị, phải sử dụng gấp đôi bộ nhớ. Nếu kích thước trung bình của đoạn nhỏ thì phân mảnh ngoài cũng nhỏ.

cuu duong than cong . com

9.7. KẾT HỢP PHÂN ĐOẠN VỚI PHÂN TRANG

Phân đoạn và phân trang đều có ưu, nhược điểm riêng. Trên thực tế, trong cả hai dòng vi xử lý phổ biến nhất hiện nay, dòng Motorola 68000 thiết kế dựa trên không gian địa chỉ phẳng, trong khi dòng Intel 80x86 lại dựa trên phân đoạn. Cả hai đều kết hợp các mô hình bộ nhớ để hướng tới sự hội tụ giữa phân đoạn và phân trang. Có thể nhìn thấy sự kết hợp này trên HĐH MULTICS và dòng Intel 380.

☞ *MULTICS*

Trong MULTICS, địa chỉ logic được tách thành số hiệu đoạn 18 bit và địa chỉ tương đối trong đoạn (offset) 16 bit. Mặc dù phương pháp này tạo ra không gian địa chỉ 34 bit, chi phí quản lý (tính theo đơn vị bộ nhớ) của bảng đoạn là chấp nhận được (tiến trình có bao nhiêu đoạn thì bảng đoạn có bấy nhiêu hàng).

Tuy nhiên, đoạn kích thước 64KB, mỗi từ 36 bit nên kích thước trung bình của đoạn tương đối lớn và gây ra hiện tượng phân mảnh ngoài. Bên cạnh đó, thời gian tìm kiếm để cấp phát đoạn khá lớn. Để khắc phục hai nhược điểm này, MULTICS thực hiện phân trang cho đoạn. Phân trang loại trừ hiện tượng phân mảnh ngoài vì trang nhớ có thể nằm trong bất cứ khung trống nào. Mỗi trang trong MULTICS có kích thước 1 Kword. Do đó, địa chỉ tương đối 16 bit trong đoạn được chia thành 6 bit số hiệu trang và 10 bit cho địa chỉ tương đối trong trang. Số hiệu trang được đối chiếu với bảng trang để xác định số hiệu khung tương ứng. Cuối cùng, số hiệu khung được kết hợp với địa chỉ tương đối trong trang để tạo ra địa chỉ vật lý. Phương pháp biến đổi địa chỉ được minh họa trên Hình 9.14b. Chú ý, sự khác biệt giữa giải pháp này và phân đoạn nguyên thủy là hàng trong bảng đoạn không chứa địa chỉ cơ sở của đoạn, mà là địa chỉ cơ sở của bảng trang cho đoạn đó.

Mỗi đoạn có bảng trang riêng. Tuy nhiên, vì kích thước đoạn bị giới hạn nên bảng trang không cần phải có kích thước đầy đủ. Số lượng dòng trong bảng trang là số lượng trang thực sự cần thiết. Giống như phân trang, trang cuối cùng của mỗi phân đoạn thông thường sẽ không được sử dụng hết. Do đó, trung bình phân mảnh trong trong mỗi đoạn là nửa trang.

Phương pháp phân trang cho đoạn của MULTIC cũng hết sức đơn giản. Vì số hiệu đoạn là 18 bit, ta có thể có tới 262144 đoạn, đòi hỏi bảng đoạn vô cùng lớn. Để khắc phục vấn đề này, MULTIC phân trang cho bảng phân đoạn. Số hiệu trang (18 bit) bị chia ra thành 8 bit số hiệu trang và 10 bit địa chỉ tương đối trong trang. Vì thế, bảng đoạn được mô tả bởi bảng trang chứa được lên tới 2^8 dòng. Do đó, địa chỉ logic của MULTICS có khuôn dạng (s1, s2, d1, d2). Trong đó s1 là một chỉ mục trong bảng trang của bảng phân đoạn và s2 là độ dịch chuyển trong bảng đoạn. Với s1 và s2 ta đã có trang chứa bảng đoạn cần xác định. Khi đó, d1 là khoảng cách trong bảng trang

của đoạn; cuối cùng, d2 là là địa chỉ tương đối trong trang ứng với từ cần truy xuất. Để đảm bảo đạt được hiệu suất chấp nhận được, hệ thống có 16 thanh ghi TLB để lưu địa chỉ của 16 trang vừa được truy xuất gần đây nhất. Mỗi thanh ghi bao gồm 2 phần là khóa và giá trị. Trường khóa 24 bit chứa cả số hiệu đoạn lẫn số hiệu trang. Trường giá trị là số hiệu khung tương ứng.

9.8. NHẬN XÉT

Cơ chế quản lý bộ nhớ của HĐH đa chương trình tiến hóa từ đơn giản (hệ thống một người dùng) đến phức tạp (hệ thống kết hợp cả phân trang và phân đoạn). Yếu tố quyết định sẽ sử dụng phương pháp quản lý bộ nhớ nào là sự hỗ trợ từ phần cứng. Mọi địa chỉ bộ nhớ do CPU tạo ra phải được kiểm tra tính hợp lệ trước khi ánh xạ đến địa chỉ vật lý. Không thể kiểm tra hiệu quả bằng phần mềm. Các thuật toán quản lý bộ nhớ được thảo luận ở đây (cấp phát liên tục, phân trang, phân đoạn, kết hợp cả phân trang lẫn phân đoạn) khác nhau ở nhiều khía cạnh.

- **Hỗ trợ của phần cứng:** Phương pháp đơn và đa phân vùng chỉ cần thanh ghi cơ sở và thanh ghi giới hạn, trong khi phân trang và phân đoạn cần có bảng xác định ánh xạ địa chỉ.
- **Hiệu suất:** Độ phức tạp của thuật toán tỷ lệ với thời gian cần thiết biến đổi địa chỉ logic sang địa chỉ vật lý. Với hệ thống đơn giản, chỉ cần so sánh hoặc cộng, các phép toán này được thực hiện rất nhanh. Phân trang và phân đoạn cũng có thể nhanh như thế nếu bảng trang/đoạn nằm trong các thanh ghi tốc độ cao. Tuy nhiên, nếu bảng nằm trong bộ nhớ, tốc độ truy xuất bộ nhớ của người dùng có thể bị suy giảm đáng kể. TLB có thể được sử dụng để khắc phục một phần sự suy giảm hiệu suất.
- **Hiện tượng phân mảnh:** Thông thường hệ thống đa chương trình sẽ hoạt động hiệu quả hơn với mức độ đa chương trình cao. Với một tập các tiến trình xác định, hệ thống có thể tăng mức độ đa chương trình bằng cách đưa nhiều tiến trình trong bộ nhớ. Để thực hiện điều này, phải giảm sự lãng phí bộ nhớ hoặc hiện tượng phân mảnh. Hệ thống với đơn vị cấp phát có kích thước cố định, như phương pháp đơn phân vùng và phân trang xuất hiện hiện tượng phân mảnh trong.

Hệ thống mà đơn vị cấp phát có kích thước thay đổi, như phương pháp đa phân vùng và phân đoạn lại xuất hiện hiện tượng phân mảnh ngoài.

- **Tái định vị:** Giải pháp cho vấn đề phân mảnh ngoài là thu gọn bộ nhớ (dịch chuyển các chương trình trong bộ nhớ mà không làm thay đổi nội dung chương trình). Việc này đòi hỏi địa chỉ logic phải được tái định vị tại thời điểm thực thi.
- **Hoán chuyển:** Bất cứ giải pháp nào cũng cần thêm khả năng hoán chuyển. Sau các khoảng thời gian định kỳ do HĐH xác định (thường do chính sách điều phối CPU quy định), các tiến trình được chuyển từ bộ nhớ chính ra ổ đĩa cứng và sau đó được chuyển trở lại bộ nhớ chính. Phương pháp này cho phép tại cùng thời điểm có nhiều tiến trình thực thi đồng thời.
- **Chia sẻ:** Một cách để tăng mức độ đa chương trình là chia sẻ mã và dữ liệu giữa các người dùng khác nhau. Thông thường chia sẻ chỉ thực hiện được với phương pháp phân trang hoặc phân đoạn do có thể dùng chung các đơn vị thông tin cơ sở (trang hoặc đoạn). Chia sẻ là biện pháp để chạy nhiều tiến trình với lượng bộ nhớ giới hạn, nhưng chương trình và dữ liệu được chia sẻ phải được thiết kế hết sức cẩn thận.
- **Bảo vệ:** Nếu cung cấp phân trang và phân đoạn, các vùng khác nhau của chương trình người dùng có thể được khai báo các thuộc tính chỉ thực thi, chỉ đọc, hoặc đọc – ghi.

CÂU HỎI ÔN TẬP

1. Phân biệt các loại địa chỉ bộ nhớ.
2. Trình bày ưu điểm của tải động và liên kết động.
3. Trình bày các phương pháp quản lý bộ nhớ động.

cuu duong than cong . com

Chương 10
BỘ NHỚ ẢO

Mục tiêu của các phương pháp quản lý bộ nhớ là tải đồng thời nhiều tiến trình vào bộ nhớ. Chương 9 đã trình bày cách tải toàn bộ tiến trình vào trong bộ nhớ trước khi thực thi. Công nghệ bộ nhớ ảo được giới thiệu trong chương này lại cho phép thực thi các tiến trình không nằm trọn vẹn trong bộ nhớ. Khi đó, kích thước chương trình có thể lớn hơn kích thước bộ nhớ vật lý và công nghệ này tách bạch hình ảnh bộ nhớ vật lý với hình ảnh bộ nhớ logic dưới góc độ người dùng. Tuy nhiên, cài đặt bộ nhớ ảo không dễ và có thể làm giảm hiệu suất hệ thống.

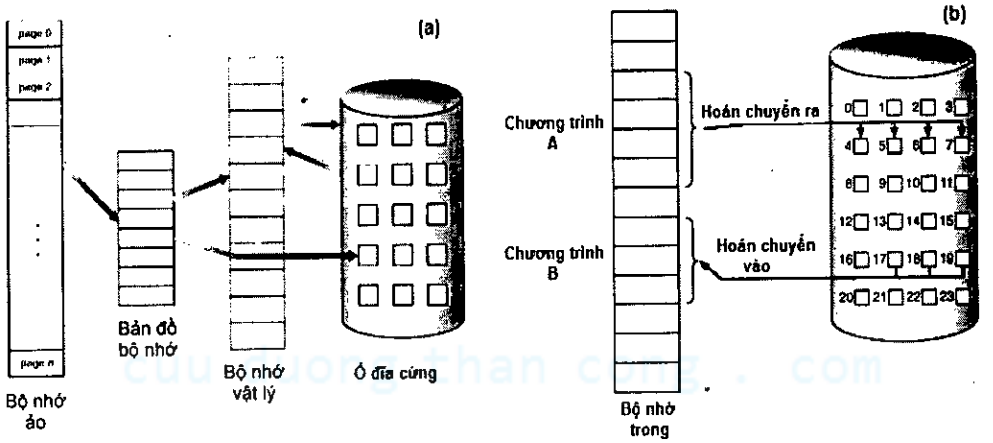
cuu duong than cong . com

10.1. ĐẶT VẤN ĐỀ

Kỹ thuật phủ hay nạp động (Chương 9) cho phép tạo ra chương trình có kích thước lớn hơn bộ nhớ vật lý được cấp phát. Trên thực tế, nhiều chương trình máy tính có khả năng thực thi kể cả khi chưa nằm trọn vẹn trong bộ nhớ (có thể không thực thi những đoạn mã xử lý lỗi nếu lỗi không xuất hiện). Thậm chí, ngay cả khi cần sử dụng toàn bộ chương trình thì cũng không nhất thiết phải sử dụng toàn bộ tại một thời điểm.

Khả năng thực thi chương trình mà chỉ cần tải một phần chương trình vào bộ nhớ đem lại nhiều ưu điểm cho cả người sử dụng lẫn hệ thống. Thứ nhất, chương trình không bị khống chế bởi kích thước bộ nhớ vật lý, vì bộ nhớ ảo tách hẳn bộ nhớ logic khỏi bộ nhớ vật lý. Lập trình viên có thể viết những chương trình trong không gian bộ nhớ ảo có kích thước cực lớn, việc lập trình trở nên đơn giản hơn (không phải tự mình thực hiện tải ra/tải vào như trong kỹ thuật phủ). Thứ hai, chương trình chiếm ít bộ nhớ vật lý hơn, nên bộ nhớ có thể chứa nhiều chương trình, do đó tăng hiệu suất sử dụng

CPU. Cuối cùng, thời gian tải hay hoán chuyển chương trình người dùng vào bộ nhớ giảm, nên chương trình có thể chạy nhanh hơn. Mặc dù phương pháp phân trang theo yêu cầu phổ biến hơn, tuy nhiên bộ nhớ ảo cũng có thể cài đặt trong hệ thống phân đoạn. Trong nhiều hệ thống hỗ trợ phương pháp phân đoạn kết hợp phân trang (đoạn được chia thành nhiều trang), thì dưới góc độ người dùng, bộ nhớ vẫn được chia thành nhiều đoạn, nhưng HĐH lại cài đặt kỹ thuật phân trang theo yêu cầu. Phân đoạn theo yêu cầu cũng có thể cài đặt bộ nhớ ảo, tuy nhiên, thay thế đoạn phức tạp hơn thay thế trang vì kích thước đoạn không cố định.



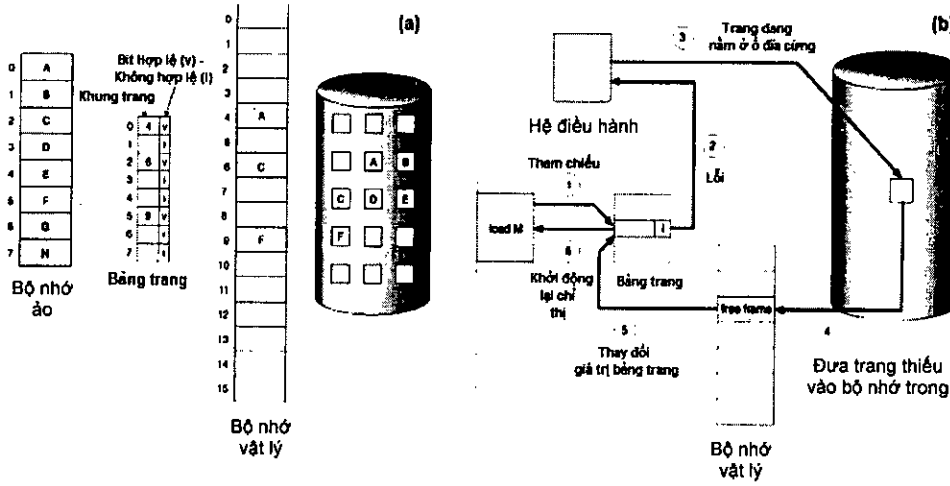
Hình 10.1. Bộ nhớ ảo và bộ nhớ vật lý

10.2. PHÂN TRANG THEO YÊU CẦU

Phân trang theo yêu cầu tương tự phân trang hoán chuyển. Các tiến trình nằm trên ổ đĩa được bộ phân trang tải vào bộ nhớ khi thực thi. Chú ý, thay vì đưa toàn bộ tiến trình vào, bộ phân trang chỉ đưa vào bộ nhớ những trang mà tiến trình thực sự cần đến. Ở đây có thể xem tiến trình là tập hợp các trang, chứ không phải một không gian địa chỉ liên tục. Bộ hoán chuyển tác động lên toàn bộ tiến trình, trong khi đối tượng của bộ phân trang là từng trang riêng lẻ. Trước khi tải tiến trình vào bộ nhớ, bộ phân trang dự đoán những trang sẽ được sử dụng và chỉ đưa những trang này vào. Do không phải tải các trang không dùng, nên giảm thời gian chuyển đổi và tiết kiệm không gian bộ nhớ vật lý.

Để xác định trang nằm trong bộ nhớ hay nằm trên ổ đĩa, HĐH sử dụng bit hợp lệ - không hợp lệ (valid-invalid) như trong mục 9.5.2. Khi bit có giá

trị *valid*, trang tương ứng hợp lệ và nằm trong bộ nhớ. Trong trường hợp ngược lại, trang tương ứng hoặc không hợp lệ (không nằm trong không gian địa chỉ của tiến trình), hoặc hợp lệ nhưng hiện nằm trên ổ đĩa. Tình huống này được mô tả trong Hình 10.2a.



Hình 10.2. Xử lý lỗi trang

Trang chưa đưa vào bộ nhớ không gây ảnh hưởng gì nếu không được sử dụng. Do đó, nếu dự đoán đúng và đưa vào bộ nhớ tất cả các trang cần thiết thì tiến trình vẫn chạy như thể toàn bộ tiến trình đã nằm trong bộ nhớ. Vấn đề chỉ phát sinh khi tiến trình truy cập đến trang chưa tải vào bộ nhớ. Tình huống này gây ra lỗi trang. Trong quá trình giải mã địa chỉ, phần cứng phân trang nhận ra trang không hợp lệ, nên yêu cầu HĐH xử lý. Lỗi này do HĐH chưa tải trang cần thiết vào bộ nhớ, chứ không phải do truy cập tới vùng địa chỉ không hợp lệ. Các bước xử lý lỗi được minh họa trên Hình 10.2b như sau:

- Kiểm tra địa chỉ truy nhập có nằm trong vùng địa chỉ hợp lệ không.
- Nếu yêu cầu truy nhập không hợp lệ, kết thúc công việc. Nếu yêu cầu hợp lệ nhưng trang yêu cầu chưa nằm trong bộ nhớ, thì HĐH tải trang vào bộ nhớ.
- Tìm một frame trống (lấy từ danh sách frame trống).
- Yêu cầu đọc trang mong muốn từ ổ đĩa vào frame mới tìm thấy.
- Khi hoàn tất việc đọc ổ đĩa, cập nhật bảng trạng thái của tiến trình và bảng phân trang để chỉ ra trang yêu cầu đã được đưa vào bộ nhớ.

- Khởi động lại chỉ thị bị ngắt do lỗi trang. Lúc này tiến trình truy cập vào trang nhớ bình thường như thể trang đó luôn nằm trong bộ nhớ.

Khi bị lỗi trang, HĐH lưu lại ngữ cảnh tiến trình, nên tiến trình có thể khởi động lại tại đúng vị trí bị gián đoạn, ngoại trừ rằng, trang thiếu này đã có trong bộ nhớ.

Tiến trình có thể bắt đầu thực thi khi chưa có trang nào trong bộ nhớ. Khi thực thi chỉ thị đầu tiên, ngay lập tức tiến trình bị phong tỏa do lỗi trang. Sau khi tải trang cần thiết vào bộ nhớ, tiến trình tiếp tục chạy, lỗi trang lại xuất hiện cho tới khi nào tất cả các trang cần thiết được đưa vào bộ nhớ. Phương pháp này có thể gọi là *phân trang theo yêu cầu thuần túy* (không tải trang không cần thiết vào bộ nhớ).

Về mặt lý thuyết, hoàn toàn có khả năng chỉ thị của chương trình tham chiếu tới nhiều trang nhớ (một trang cho chính chỉ thị và nhiều trang cho dữ liệu). Do vậy, một chỉ thị có thể gây ra nhiều lỗi trang. Điều này làm giảm đáng kể hiệu suất máy tính. Thống kê khi phân tích các tiến trình đang chạy cho thấy khả năng trên rất hiếm, vì các chương trình có xu hướng sử dụng *một miền tham chiếu cục bộ* (locality of reference). Do vậy, phân trang theo yêu cầu làm tăng hiệu suất hệ thống. Phần cứng hỗ trợ phân trang theo yêu cầu giống phần cứng hỗ trợ phân trang hay hoán chuyển:

- **Bảng phân trang**: Có thể đánh dấu trang là không hợp lệ thông qua bit valid-invalid, hoặc giá trị đặc biệt nào đó của bit bảo vệ.
- **Bộ nhớ thứ cấp**: Dùng để lưu trữ tất cả trang chưa nạp vào bộ nhớ chính. Bộ nhớ thứ cấp thường là ổ đĩa cứng. Vùng trên ổ đĩa dùng cho việc hoán chuyển được gọi là vùng hoán chuyển.

Tiến trình phải được khởi động ngay sau khi lỗi trang. Lỗi trang có thể xảy ra với bất kỳ yêu cầu truy cập bộ nhớ nào. Nếu xảy ra trong giai đoạn lấy chỉ thị về hệ thống, có thể khởi động lại bằng cách lấy lại chỉ thị. Nếu lỗi xảy ra trong giai đoạn đọc toán hạng, hệ thống phải lấy lại chỉ thị, giải mã và đọc lại toán hạng. Trường hợp phức tạp hơn, xét chỉ thị ADD A, B, C có 3 toán hạng thực hiện cộng giá trị A và B rồi đặt kết quả vào C. Các bước tiến hành sẽ là:

1. **Lấy chỉ thị từ bộ nhớ về và giải mã chỉ thị (ADD).**
2. **Đọc A.**

3. **Đọc B.**
4. **Cộng A với B.**
5. **Lưu kết quả vào C.**

Nếu lỗi trang xảy ra khi lưu kết quả vào C (vì lúc đó C nằm ở trang chưa được tải vào bộ nhớ), hệ thống phải tải trang chứa C vào bộ nhớ, sửa lại bảng phân trang, khởi động lại chỉ thị. Sau đó, chỉ thị lại được đưa vào CPU, giải mã, đọc hai toán hạng A và B, thực hiện lại phép cộng.

Vấn đề trở nên phức tạp khi chỉ thị tham chiếu đến nhiều địa chỉ khác nhau. Ví dụ, chỉ thị MVC của hệ thống IBM 360/370 (move character) di chuyển một khối byte giữa hai vị trí (có thể xen phủ lẫn nhau). Nếu cả khối nguồn và khối đích trải ra khỏi biên của trang, lỗi trang có thể xuất hiện ngay khi mới di chuyển được một phần dữ liệu. Có thể xử lý theo hai hướng. Hướng thứ nhất, vi mã (microcode) tính toán và truy cập trước tới điểm mút của cả hai khối. Lỗi trang nếu xảy ra sẽ bị phát hiện và xử lý sớm. Giải pháp thứ hai là, dùng thanh ghi tạm lưu trữ giá trị của miền bị ghi đè. Nếu xảy ra lỗi trang, tất cả giá trị cũ sẽ được ghi trở lại vị trí bộ nhớ lúc trước. Thao tác này khôi phục trạng thái bộ nhớ trước khi thi hành chỉ thị.

cuu duong than cong . com

10.3. HIỆU SUẤT PHÂN TRANG THEO YÊU CẦU

Ta sẽ tính thời gian truy cập bộ nhớ có ích để đánh giá ảnh hưởng của phân trang theo yêu cầu tới hiệu suất hệ thống. Giả sử thời gian truy cập bộ nhớ (ký hiệu *ma*) nằm trong khoảng 10 đến 200ns. Nếu không có lỗi trang, thời gian truy cập có ích bằng thời gian truy cập bộ nhớ trong. Nếu xảy ra lỗi trang, hệ thống phải tải trang từ ổ đĩa cứng, sau đó mới truy nhập đến trang nhớ mong muốn.

Giả sử p là xác suất lỗi trang ($0 \le p \le 1$). Thời gian truy cập có ích được tính như sau:

cuu duong than cong . com

$$\textit{Thời gian}_{\textit{truy cập có ích}} = (1 - p) \times ma + p \times \textit{thời gian xử lý lỗi trang}$$

Hệ thống phải thực hiện các hành động xử lý lỗi trang sau:

1. Chuyển quyền điều khiển cho HĐH.
2. Lưu lại ngữ cảnh của tiến trình.
3. Xác nhận ngắt hiện tại do lỗi trang gây ra.

4. Kiểm tra việc tham chiếu đến trang là hợp lệ và xác định vị trí của trang trên ổ đĩa cứng.
5. Ra lệnh tải trang từ ổ đĩa cứng vào frame trống:
   a. Đợi trong hàng đợi của thiết bị.
   b. Chờ đầu đọc dịch chuyển trên ổ đĩa.
   c. Bắt đầu chuyển trang vào frame trống.
6. Trong lúc chờ đợi, chuyển CPU cho tiến trình khác (không bắt buộc).
7. Xuất hiện ngắt từ ổ đĩa cứng (hoàn tất thao tác đọc).
8. Lưu ngữ cảnh của tiến trình đang chiếm dụng CPU (nếu có bước 6).
9. Xác nhận ngắt là ngắt từ ổ đĩa cứng.
10. Cập nhật lại bảng phân trang để chỉ ra trang cần truy cập hiện đã nằm trong bộ nhớ.
11. Chờ CPU được cấp phát lại cho tiến trình này.
12. Khôi phục lại ngữ cảnh tiến trình bị phong tỏa ở bước 1, sau đó tiếp tục thi hành chỉ thị đã bị ngắt.

Không phải xuất hiện tất cả 12 bước này. Ví dụ, bước 6 và bước 8 có thể không xảy ra. Quá trình xử lý lỗi trang có 3 giai đoạn chính:

1. Xử lý ngắt gây lỗi trang.
2. Đọc trang bộ nhớ yêu cầu.
3. Khởi động lại tiến trình.

Có thể rút ngắn thời gian thứ nhất và thứ ba vài trăm chỉ thị nếu viết mã tốt. Thời gian thực hiện những công việc trên từ 1 đến 100ms. Mặt khác, thời gian hoán chuyển trang xấp xỉ 24ms (ổ đĩa cứng thông thường có độ trễ trung bình 8ms, thời gian đầu đọc dịch chuyển 15ms và thời gian truyền dữ liệu 1ms). Như vậy, tổng thời gian thực hiện phân trang xấp xỉ 25ms (tính cả thời gian dành cho phần cứng lẫn phần mềm). Lưu ý rằng, ta mới chỉ xét thời gian phục vụ của thiết bị. Nếu nhiều tiến trình đang chờ thiết bị, thì phải cộng thêm cả thời gian đợi, và như vậy, thời gian hoán chuyển còn lớn hơn. Nếu lấy thời gian xử lý lỗi trang trung bình là 25ms và thời gian truy cập bộ nhớ là 100ns, thì thời gian truy cập có ích tính theo ns sẽ là:

$$\text{Thời gian}_{\text{truy cập có ích}} = (1 - p) \times 100 + p \times (25ms)$$

$$= (1 - p) \times 100 + p \times 25.000.000 = 100 + 24.9910.990 \times p$$

Thời gian truy cập có ích phụ thuộc vào tỷ lệ lỗi trang. Nếu 1000 lần truy cập xuất hiện 1 lỗi trang, thời gian truy cập có ích là 25ms. Tốc độ máy tính giảm đi 250 lần. Nếu muốn hiệu suất chỉ giảm còn 10% thì:

$$110 > 100 + 25.000.000 \times p \Leftrightarrow 10 > 25.000.000 \times p \Leftrightarrow p < 0,0000004.$$

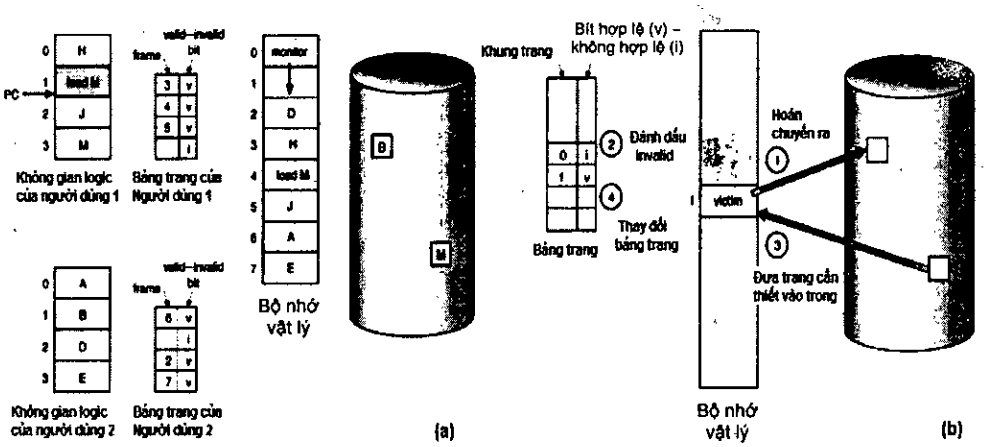
Tức là, chỉ được xuất hiện một lỗi trang trong 2500000 lần truy cập bộ nhớ. Giữ tỷ lệ lỗi trang ở mức thấp đóng vai trò cực kỳ quan trọng trong hệ thống phân trang theo yêu cầu. Nếu thời gian truy cập có ích tăng, thì tốc độ thực thi tiến trình sẽ giảm đáng kể. HĐH dành một vùng trong ổ đĩa để lưu giữ các trang nhớ, vùng này được gọi là vùng hoán chuyển. Cách quản lý và sử dụng vùng hoán chuyển ảnh hưởng đến hiệu suất. Vùng hoán chuyển được chia thành các sector có kích thước lớn và không phải tìm kiếm theo tên file, nên tốc độ đọc/ghi trên vùng hoán chuyển nhanh hơn trên hệ thống file. Có thể cải thiện hiệu suất hệ thống phân trang bằng cách sao chép toàn bộ file vào vùng hoán chuyển ngay khi khởi động tiến trình, sau đó thực hiện phân trang theo yêu cầu từ vùng hoán chuyển. Nếu kích thước vùng hoán chuyển bị giới hạn, thì có thể sử dụng phương pháp khác trong trường hợp sử dụng file nhị phân. Các trang cần thiết được tải trực tiếp từ hệ thống file. Tuy nhiên, khi sử dụng thủ tục thay thế trang, các khung trang trong bộ nhớ có thể bị ghi đè (vì các trang này có thuộc tính chỉ đọc) và sẽ được tải lại vào bộ nhớ từ hệ thống file nếu cần. Một lựa chọn khác là, đầu tiên sẽ lấy trang trực tiếp từ hệ thống file. Khi hoán chuyển ra ổ đĩa, trang được lưu vào vùng hoán chuyển. Giải pháp này đảm bảo chỉ lấy trang từ hệ thống file một lần duy nhất, tất cả những lần sau được lấy từ vùng hoán chuyển.

10.4. THAY THẾ TRANG

Phân trang theo yêu cầu chỉ tải những trang cần thiết vào bộ nhớ. Do đó, tăng mức độ đa nhiệm vì bộ nhớ chứa được nhiều tiến trình. Tuy nhiên, tăng mức độ đa nhiệm lại dẫn đến tình trạng cấp phát bộ nhớ quá khả năng, tức là khi tổng lượng bộ nhớ các tiến trình yêu cầu vượt quá khả năng hệ thống có thể đáp ứng.

Khi xảy ra lỗi trang, phần cứng chuyển quyền điều khiển cho HĐH. HĐH kiểm tra để xác nhận đây là sự kiện lỗi trang chứ không phải là lỗi truy cập bộ nhớ trái phép. HĐH xác định vị trí trang trên ổ đĩa cứng, nhưng

sau đó nhận ra không còn frame nào trống, vì toàn bộ bộ nhớ lúc này đã được sử dụng (Hình 10.3a).



Hình 10.3. Thay thế trang

HĐH có nhiều cách giải quyết, chẳng hạn chấm dứt tiến trình hoặc hoán chuyển tiến trình khác ra ngoài. Giải pháp trình bày ở đây là *thay thế trang.* Nếu không có frame trống, HĐH thu hồi một frame bằng cách chuyển trang nằm trong frame ra ngoài (cập nhật lại bảng phân trang để chỉ ra trang vừa chuyển hiện không nằm trong bộ nhớ). Sau đó, HĐH có thể tải trang mà tiến trình cần vào frame trống. Thủ tục xử lý lỗi trang được bổ sung thêm việc thay thế trang:

1. Tìm trang cần tải trên ổ đĩa cứng.
2. Tìm một frame trống; nếu thấy frame trống thì sử dụng; nếu không thấy, dùng thuật toán thay trang để tìm trang làm "nạn nhân" chuyển ra ngoài. Ghi trang "nạn nhân" vào ổ cứng, cập nhật bảng phân trang và bảng frame.
3. Tải trang cần thiết vào frame vừa được giải phóng, cập nhật lại bảng phân trang và bảng frame.
4. Khởi động lại tiến trình.

Nếu không có frame trống, hệ thống phải hoán chuyển hai trang (một trang ra và một trang vào). Thời gian xử lý lỗi trang tăng gấp đôi, nên thời gian truy nhập có ích cũng tăng theo.

Kỹ thuật *bit thay đổi* (modify bit) có thể giảm thời gian xử lý lỗi trang. Bit này được gắn với mỗi frame, xác định trang nằm trong frame có khác

trang "gốc" nằm trên ổ đĩa hay không. Khi trang được ghi, phần cứng thiết lập giá trị 1 cho bit này để chỉ ra trang đã bị thay đổi (so với trang gốc). Khi chọn trang thay thế, hệ thống kiểm tra bit thay đổi tương ứng. Bit có giá trị 1, nghĩa là trang đã bị thay đổi sau khi chuyển vào bộ nhớ. Nếu chuyển trang này ra thì cần ghi lại vào ổ cứng. Nếu bit có giá trị 0, thì trang chưa bị thay đổi từ khi đưa vào bộ nhớ, vì thế hệ thống không phải ghi lại trang (vì trang trong bộ nhớ giống trang trong ổ đĩa). Công nghệ này cũng áp dụng được cho trang có thuộc tính chỉ đọc (ví dụ như, những trang chứa mã nhị phân). Thời gian xử lý lỗi trang giảm đáng kể vì thời gian dành cho thao tác vào/ra giảm xuống còn một nửa khi chọn trang chưa bị thay đổi.

Thông qua cơ chế thay thế trang, lập trình viên có thể thao tác trên bộ nhớ ảo lớn hơn bộ nhớ vật lý. Bộ nhớ logic tồn tại độc lập và không bị ràng buộc bởi kích thước bộ nhớ vật lý. Tiến trình người dùng có kích thước 30 trang vẫn có thể thực thi trên 10 frame, và HĐH sử dụng thuật toán thay thế trang để tìm frame trống khi cần thiết. Nội dung trang bị thay thế được ghi ra ổ cứng. Những tham chiếu sau này đến trang bị chuyển ra sẽ gây ra tình trạng lỗi trang. Khi đó, trang nói trên lại được đưa trở lại bộ nhớ và rất có thể lại thế chỗ cho một trang khác.

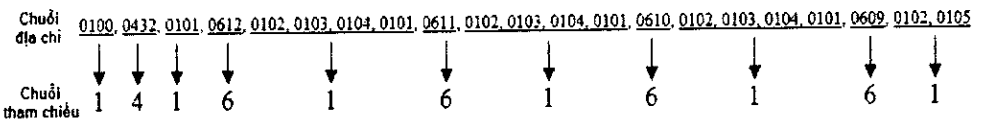
Phần sau trình bày hai vấn đề quan trọng khi triển khai phương pháp phân trang theo yêu cầu là thuật toán Cấp phát frame và thuật toán Thay thế trang. Nếu có nhiều tiến trình nằm trong bộ nhớ, HĐH phải quyết định cấp phát cho mỗi tiến trình bao nhiêu frame. Hơn nữa, khi có yêu cầu thay thế trang, HĐH phải chọn frame nào để thay thế.

10.5. THUẬT TOÁN THAY THẾ TRANG

Có nhiều thuật toán thay thế trang khác nhau và mỗi HĐH lựa chọn một phương pháp thay thế cụ thể với tỷ lệ lỗi trang ít nhất. Để đánh giá hiệu suất, cần thực hiện thuật toán trên một dãy tham chiếu bộ nhớ cụ thể và tính toán số lỗi trang. Có thể sinh ngẫu nhiên dãy tham chiếu giả (thông qua một bộ tạo số ngẫu nhiên), hoặc theo dõi một hệ thống đang vận hành và ghi lại địa chỉ mỗi tham chiếu bộ nhớ. Cách sau tạo ra nhiều dữ liệu (khoảng 1 triệu địa chỉ/s). Để giảm lượng dữ liệu nhận được, cần chú ý hai điểm:

1. Với kích thước trang xác định (trang thường có kích thước cố định được quy định bởi phần cứng hoặc hệ thống), chỉ quan tâm đến địa chỉ trang chứ không quan tâm đến địa chỉ ô nhớ.
2. Nếu tồn tại một tham chiếu tới trang p, thì các tham chiếu liền kề tới trang p không gây lỗi trang, vì p đã nằm trong bộ nhớ sau tham chiếu đầu tiên.

Giả sử kích thước trang là 100 byte, khi theo dõi tiến trình cụ thể, chuỗi địa chỉ và chuỗi tham chiếu tương ứng được minh họa trên Hình 10.4.



Hình 10.4. Chuỗi tham chiếu bộ nhớ

Để xác định số lỗi trang của thuật toán thay thế trang trên chuỗi tham chiếu trên, phải biết số lượng frame trống. Rõ ràng, nhiều frame trống thì số lỗi trang giảm. Ví dụ, với chuỗi tham chiếu trên, nếu số lượng frame trống không nhỏ hơn 3, thì tối đa chỉ có 3 lỗi trang, mỗi lỗi trang xuất hiện trong lần tham chiếu đầu tiên. Mặt khác, nếu chỉ có một frame trống, thì HĐH phải thay thế trang cho tất cả các tham chiếu, kết quả có 11 lỗi trang.

10.5.1. Thuật toán FIFO

FIFO là thuật toán thay thế trang đơn giản nhất. Thuật toán ghi lại thời điểm trang nhớ được đưa vào bộ nhớ và trang "cũ nhất" sẽ bị thay thế. Hình 10.5a minh họa thuật toán FIFO với chuỗi tham chiếu R = 012301401234. Ta thấy xuất hiện 9 lỗi trang (các trang có gạch dưới).

Frame	0	1	2	3	0	1	4	0	1	2	3	4
0	0	0	0	3	3	3	4	4	4	4	4	4
1		1	1	1	0	0	0	0	0	2	2	2
2			2	2	2	1	1	1	1	1	3	3

(a) Có ba Frame

Frame	0	1	2	3	0	1	4	0	1	2	3	4
0	0	0	0	0	0	0	4	4	4	4	3	3
1		1	1	1	1	1	1	0	0	0	0	4
2			2	2	2	2	2	2	1	1	1	1
3				3	3	3	3	3	3	2	2	2

(b) Có bốn Frame

Hình 10.5. Ví dụ về FIFO

Mặc dù dễ hiểu, dễ lập trình, nhưng hiệu suất FIFO không cao. Ví dụ, nếu trang bị thay thế là những trang chứa mã khởi tạo chương trình được sử dụng từ rất lâu và hiện nay không còn cần nữa. Mặt khác, những trang này

có thể chứa những biến được sử dụng nhiều lần và khởi tạo ngay từ đầu chương trình. Dù trang đang được sử dụng bị thay thế, thì hệ thống vẫn hoạt động bình thường. Sau khi bị đưa ra, lỗi trang sẽ xuất hiện gần như tức thời và HĐH phải tải lại trang đó vào bộ nhớ. Một trang nào đó lại bị đưa ra để đưa trang kia trở lại bộ nhớ. Thay thế sai làm tăng tỷ lệ lỗi trang và làm chậm tiến trình.

Tuy nhiên, nếu hệ thống có 4 frame thì ta thấy số lỗi là 10 (lớn hơn số lỗi trong trường hợp có 3 frame là 9). Kết quả không mong muốn này được gọi là dị thường Belady. Dị thường này cho thấy thực tế là, đối với một số thuật toán thay thế trang, tỷ lệ lỗi trang có thể tăng khi tăng số frame được cấp phát. Thường chúng ta cho rằng, cung cấp thêm bộ nhớ cho một tiến trình sẽ giúp cải thiện hiệu suất của nó. Tuy nhiên, trong những nghiên cứu gần đây cho thấy giả định trên không phải luôn đúng.

10.5.2. Thuật toán tối ưu (Optimal Algorithm)

Thay thế trang tối ưu là thuật toán có tỷ lệ lỗi trang thấp nhất. Thuật toán này được gọi là OPT hay MIN, và không có dị thường Belady. Thuật toán tối ưu thay thế trang không được sử dụng trong thời gian dài nhất. Ví dụ, trong chuỗi tham chiếu mẫu R = 2031203120316457, thuật toán thay thế trang tối ưu sinh ra 10 lỗi trang (Hình 10.6). Thuật toán OPT khó cài đặt, vì cần phải biết trước về chuỗi tham chiếu, do đó thuật toán này sử dụng chủ yếu trong nghiên cứu so sánh.

Frame	2	0	3	1	2	0	3	1	2	0	3	1	6	4	5	7
0	2	2	2	2	2	2	2	2	2	0	0	0	0	4	4	4
1		0	0	0	0	0	3	3	3	3	3	3	6	6	6	7
2			3	1	1	1	1	1	1	1	1	1	1	1	5	5

(a) Thuật toán tối ưu

Frame	2	0	3	1	2	0	3	1	2	0	3	1	6	4	5	7
0	2	2	2	1	1	1	3	3	3	0	0	0	6	6	6	7
1		0	0	0	2	2	2	1	1	1	3	3	3	4	4	4
2			3	3	3	0	0	0	2	2	2	1	1	1	5	5

(b) Thuật toán LRU

Hình 10.6. Thuật toán tối ưu và LRU

10.5.3. Thuật toán ít được sử dụng gần đây nhất (LRU)

Cài đặt thuật toán tối ưu không khả thi, tuy nhiên vẫn có thể cài đặt thuật toán "tựa" thuật toán tối ưu. Khác với FIFO sử dụng thời điểm trang được đưa vào bộ nhớ, LRU (Least Recently Used) sử dụng thời điểm trang được sử dụng và thay thế trang không được sử dụng trong khoảng thời gian dài nhất (Hình 10.6b). Thuật toán LRU ghi lại thời điểm cuối cùng trang được sử dụng. Khi thay thế, LRU chọn trang không được sử dụng trong

khoảng thời gian dài nhất. Phương pháp này chính là thuật toán tối ưu, nhưng nhìn vào quá khứ, chứ không phải nhìn vào tương lai (Nếu S^R là nghịch đảo của chuỗi tham chiếu S, thì tỷ lệ lỗi trang đối với thuật toán OPT áp dụng trên S là tỷ lệ lỗi trang của thuật toán LRU áp dụng trên S^R). Thuật toán LRU cần nhiều sự trợ giúp từ phần cứng. Có hai cách để sắp xếp thứ tự các frame theo thời điểm truy cập cuối cùng:

1. **Bộ đếm (Counter)**: Bảng phân trang có trường thời gian sử dụng ghi lại thời điểm cuối cùng trang được tham chiếu. Hệ thống thay thế trang có giá trị thời gian nhỏ nhất. Phương pháp này đòi hỏi tìm kiếm trên bảng phân trang khi thay thế và cập nhật trường thời gian sử dụng trong mỗi lần truy nhập bộ nhớ.
2. **Ngăn xếp (Stack)**: Có thể sử dụng ngăn xếp lưu trữ địa chỉ trang. Trang bị tham chiếu được chuyển lên đỉnh ngăn xếp. Đỉnh ngăn xếp luôn là trang được sử dụng gần đây nhất và đáy ngăn xếp là trang ít được sử dụng nhất. Vì các trang có thể được lấy ra từ giữa ngăn xếp, nên cách cài đặt đơn giản nhất là sử dụng danh sách liên kết kép (với một con trỏ đầu và một con trỏ đuôi). Việc loại bỏ một trang và đưa lên đỉnh ngăn xếp cần thay đổi ít nhất 6 con trỏ. Tất nhiên, mỗi lần cập nhật làm tăng chi phí, nhưng bù lại không cần phải tìm kiếm khi thay thế, trang nằm ở đáy ngăn xếp là trang LRU.

Chú ý, không thể cài đặt LRU nếu không có phần cứng hỗ trợ. Phải thực hiện cập nhật đồng hồ hay ngăn xếp với mọi tham chiếu bộ nhớ. Nếu sử dụng ngắt để phần mềm thực hiện cập nhật trên những cấu trúc dữ liệu như vậy khi có tham chiếu xảy ra, thì tốc độ truy xuất bộ nhớ giảm và kéo theo hiệu suất hệ thống giảm.

10.5.4. Các thuật toán xấp xỉ LRU

Nếu không có phần cứng hỗ trợ, hệ thống phải cài đặt thuật toán thay thế trang khác. Chẳng hạn, sử dụng thêm trường Bit tham chiếu trong mỗi hàng của bảng phân trang. Trường Bit tham chiếu được phần cứng thiết lập (đặt giá trị 1) mỗi khi trang tương ứng bị tham chiếu (đọc hay ghi). Ban đầu HĐH sẽ đặt tất cả các bit này giá trị 0. Khi tiến trình thực thi, phần cứng sẽ đặt bit ứng với các trang bị tham chiếu giá trị 1. Dù không biết được trình tự sử dụng, nhưng trong khoảng thời gian cụ thể, có thể biết được trang nào

được sử dụng và trang nào chưa sử dụng. Thông tin này dẫn đến một số thuật toán *xấp xỉ* thuật toán thay thế LRU.

☞ ***Thuật toán Nhiều bit tham chiếu***

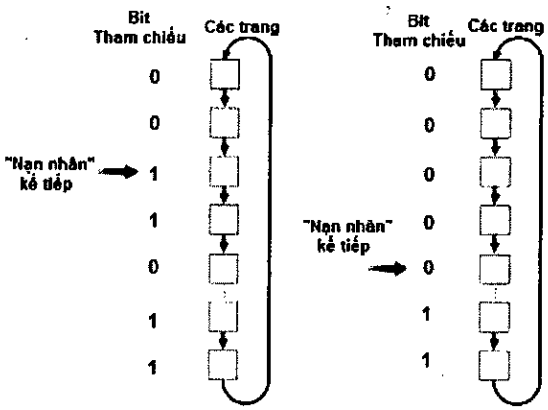
Sử dụng càng nhiều bit tham chiếu, hệ thống càng biết rõ tình trạng sử dụng của trang. Giả sử bên cạnh trường Bit tham chiếu, mỗi trang trong bộ nhớ có thêm trường Lịch sử tham chiếu 8 bit. Sau khoảng thời gian định kỳ (chẳng hạn 100ms), bộ định thời tạo ngắt để chuyển quyền điều khiển cho HĐH. HĐH dịch Bit tham chiếu trong mỗi trang vào bit cao nhất trong trường Lịch sử tham chiếu 8 bit, dịch các bit khác trong trường này sang phải 1 bit, loại bỏ bit thấp nhất. Trường 8 bit này ghi lại lịch sử quá trình sử dụng trang trong 8 khoảng thời gian (800ms) trước đó. Nếu trường có giá trị 00000000 thì trang không được sử dụng lần nào trong 8 khoảng thời gian trước. Trang mà trường Lịch sử tham chiếu có giá trị 11000100 được sử dụng gần đây hơn so với trang có giá trị 01110110. Trang nào có trường Lịch sử tham chiếu nhỏ nhất là trang LRU và có thể bị thay thế. Số lượng bit ghi quá trình lịch sử phụ thuộc vào khả năng phần cứng. Trong trường hợp số bit này bằng 0, chỉ còn duy nhất trường Bit tham chiếu. Thuật toán này gọi là thuật toán thay thế trang "cơ hội thứ hai".

☞ ***Thuật toán "cơ hội thứ hai"***

"Cơ hội thứ hai" chính là thuật toán FIFO. Khi trang được chọn làm "nạn nhân" thay thế, HĐH kiểm tra trường Bit tham chiếu tương ứng. Nếu có giá trị 0, trang sẽ bị thay thế. Nếu ngược lại, hệ thống cho trang này cơ hội thứ hai ở lại trong bộ nhớ và tiếp tục lựa chọn "nạn nhân" kế tiếp bằng thuật toán FIFO. Khi trang có cơ hội thứ hai, bit tham chiếu tương ứng bị xóa và thời gian được thiết lập là thời gian hiện tại. Do vậy, khi có "cơ hội thứ hai", trang sẽ không bị thay thế cho đến khi tất cả những trang khác bị thay thế (hoặc cũng đều có "cơ hội thứ hai"). Hơn nữa, trang được sử dụng thường xuyên (đủ để duy trì giá trị 1 cho bit tham chiếu tương ứng) sẽ không bị thay thế.

Một cách cài đặt thuật toán này là hàng đợi vòng. Hệ thống sử dụng con trỏ trỏ tới trang sẽ được thay thế. Khi cần frame, con trỏ dịch chuyển tới phía trước cho đến khi tìm thấy một trang với bit tham chiếu 0. Trong quá trình dịch chuyển, HĐH xóa đi các bit tham chiếu (như Hình 10.7). Khi tìm

được, trang "nạn nhân" sẽ bị thay thế và trang mới được chèn vào hàng đợi vòng tại vị trí đó. Trong trường hợp xấu nhất, khi tất cả các bit tham chiếu có giá trị 1, con trỏ duyệt qua toàn bộ hàng đợi, cho mỗi trang một "cơ hội thứ hai" – đây là thuật toán FIFO.



Hình 10.7. Thuật toán "Cơ hội thứ hai"

☞ ***Thuật toán "Cơ hội thứ hai" cải tiến***

Có thể cải tiến thuật toán "cơ hội thứ hai" bằng cách xét cặp (Bit tham chiếu, Bit thay đổi). Với 2 bit này có 4 khả năng sau:

(0, 0): Trang gần đây chưa được sử dụng và sửa đổi là trang tốt nhất để thay thế.

(0, 1): Trang gần đây không được sử dụng nhưng đã bị sửa đổi, lựa chọn này không tốt lắm vì cần ghi trang ra ổ đĩa cứng trước khi thay thế.

(1, 0): Trang được sử dụng gần đây nhưng chưa bị sửa đổi, có thể sớm được dùng lại.

(1, 1): Trang gần đây được sử dụng và đã bị sửa đổi, có thể được sử dụng lại và nếu thay thế thì phải ghi lại ra ổ đĩa cứng.

Khi gọi thuật toán thay thế, trang sẽ rơi vào 1 trong 4 khả năng trên. Hệ thống áp dụng thuật toán quay vòng, nhưng không xác định trang "nạn nhân" có Bit tham chiếu là 1 hay không, mà xác định trang rơi vào lớp khả năng nào. Trang đầu tiên thuộc lớp có độ ưu tiên thấp nhất sẽ được thay thế. Lưu ý, có thể duyệt qua hàng đợi vòng nhiều lần trước khi tìm thấy trang thay thế. Thuật toán này được cài đặt trong HĐH của máy Macintosh.

Ở đây, hệ thống xác định độ ưu tiên của những trang đã bị sửa đổi để giảm thời gian đọc/ghi trên ổ đĩa cứng.

10.5.5. Các thuật toán đếm (Counting Algorithm)

Một vài thuật toán thay thế trang sử dụng bộ đếm – đếm số lần trang được tham chiếu.

☞ ***Thuật toán LFU***

Thuật toán thay thế trang LFU (The Least Frequently Used – ít sử dụng thường xuyên nhất) sử dụng trang có giá trị bộ đếm nhỏ nhất để thay thế. Lý do là trang được sử dụng gần đây sẽ có bộ đếm lớn. Thuật toán này có thể rơi vào tình huống xấu, khi trang được sử dụng nhiều lần trong suốt pha khởi tạo của tiến trình, nhưng sau đó không bao giờ được sử dụng nữa. Vì đã từng được sử dụng nhiều, nên giá trị biến đếm của trang này lớn, nên trang vẫn sẽ nằm trong bộ nhớ ngay cả khi không cần thiết. Có thể khắc phục vấn đề này bằng cách, cứ sau một khoảng thời gian định kỳ dịch bộ đếm sang phải 1 bit, khiến giá trị bộ đếm bị chia đôi.

cuu duong than cong . com

☞ ***Thuật toán MFU***

Thuật toán thay thế trang MFU (The Most Frequently Used – sử dụng nhiều nhất) dựa trên lý luận rằng, trang có giá trị bộ đếm nhỏ nhất có thể mới được đưa vào, nhưng chưa được sử dụng.

Thuật toán LFU và thuật toán MFU rất ít phổ biến vì cài đặt phức tạp.

10.5.6. Các thuật toán bổ trợ

Bên cạnh thuật toán thay thế trang, hệ thống có thể sử dụng thêm nhiều kỹ thuật khác. Chẳng hạn, duy trì danh sách các frame chưa sử dụng (trống). Khi lỗi trang xuất hiện, frame được lựa chọn như trước đây. Tuy nhiên, trang cần thiết được tải vào frame trống (lấy trong danh sách). Tiến trình chạy nhanh hơn vì có thể khởi động lại ngay sau khi bị lỗi trang mà không cần đợi trang trong frame được ghi ra ổ đĩa cứng. Trong quá trình giải phóng frame, sau khi nội dung trang trong frame được ghi ra ổ đĩa cứng, frame cũng được đưa vào danh sách frame trống. Mở rộng ý tưởng này, hệ thống có thể ghi nhớ danh sách các trang đã bị sửa đổi. Khi ổ cứng rỗi, trang bị thay đổi sẽ được ghi vào ổ đĩa. Sau đó Bit thay đổi của trang được đặt giá

trị 0. Khi đó, nếu trang này bị chọn làm "nạn nhân" thay thế, thì không cần phải ghi lại ra ổ đĩa nữa.

Một kỹ thuật khác là hệ thống ghi lại trang cuối cùng nằm trong các frame trống. Vì nội dung trong frame chưa bị sửa đổi sau khi trang trong frame ghi ra ổ đĩa, nên trang cũ có thể được sử dụng ngay lập tức từ tập các frame trống nếu frame đó chưa bị cấp phát. Khi lỗi trang xuất hiện, đầu tiên hệ thống kiểm tra xem trang cần thiết có còn nằm trong danh sách frame trống không. Nếu có, hệ thống sử dụng lại. Ngược lại, hệ thống mới phải tải trang vào một frame trống. Kỹ thuật này được cài đặt trên hệ thống VAX/VMS với thuật toán thay thế trang FIFO. Khi thay thế nhầm một trang vẫn còn được sử dụng, trang này nhanh chóng được lấy lại từ danh sách frame trống mà không phải tải vào từ ổ cứng.

10.6. CẤP PHÁT FRAME

Cần cấp phát bao nhiêu bộ nhớ cho các tiến trình khác nhau? Trường hợp đơn giản nhất là hệ thống chỉ có một tiến trình. Xét hệ thống như vậy với kích thước bộ nhớ vật lý là 64 frame. HĐH chiếm 10 frame, còn lại 54 frame cho tiến trình ứng dụng. Với phương pháp phân trang theo yêu cầu thuần tuý, 54 frame ban đầu nằm trong danh sách frame trống. Khi bắt đầu thi hành, tiến trình ứng dụng có thể tạo ra một chuỗi lỗi trang, 54 lỗi trang đầu tiên lấy dần các frame trong danh sách frame trống. Khi không còn frame trống, hệ thống sử dụng thuật toán thay thế trang, chọn 1 trong 54 trang nhớ làm "nạn nhân" chuyển ra ngoài. Quá trình này cứ thế tiếp tục. Khi tiến trình kết thúc, các frame được đưa trở lại danh sách trống. Vấn đề nảy sinh khi phân trang theo yêu cầu kết hợp với đa chương trình. Đa chương trình cho phép tải nhiều tiến trình vào bộ nhớ cùng một lúc.

10.6.1. Số frame tối thiểu

Không thể cấp phát nhiều hơn tổng số frame của hệ thống (trừ khi có chia sẻ trang). Ngoài ra, tiến trình muốn thực thi cần có một lượng tối thiểu frame (tùy thuộc vào kiến trúc máy tính). Hiển nhiên, khi số frame được cấp phát cho mỗi tiến trình giảm, thì tỷ lệ lỗi trang tăng lên và làm tiến trình chạy chậm đi.

Nếu lỗi trang xảy ra trước khi chỉ thị hoàn thành, chỉ thị phải khởi động lại từ trạng thái trước khi bắt đầu thực hiện. Vì vậy, hệ thống cần có đủ frame để lưu giữ tất cả các trang nhớ khác nhau mà chỉ thị tham chiếu tới. Ví dụ, xét dòng kiến trúc máy tính nào đó mà chỉ thị tham chiếu bộ nhớ chỉ tham chiếu tới một địa chỉ duy nhất. Vì thế, cần ít nhất một frame cho chỉ thị và một frame cho địa chỉ được tham chiếu. Nếu cho phép đánh địa chỉ gián tiếp cấp 1 (ví dụ: chỉ thị LOAD AX, [17689] ở trang 16 sẽ lấy nội dung ô nhớ 17689 (giả sử là X) làm địa chỉ, tải nội dung ô nhớ có địa chỉ X vào thanh ghi AX), thì tối đa CPU có thể phải truy xuất tới 3 frame khi thực thi chỉ thị này.

Tình huống xấu nhất là trong kiến trúc cho phép địa chỉ có nhiều cấp độ gián tiếp. Về mặt lý thuyết, lệnh LOAD thông thường có thể tham chiếu đến một địa chỉ gián tiếp, địa chỉ này tham chiếu đến một địa chỉ gián tiếp khác (ở trang khác), và cứ như vậy, cho tới khi mọi trang trong bộ nhớ ảo được tham chiếu hết. Vì thế, trong trường hợp xấu nhất, toàn bộ bộ nhớ ảo phải nằm trong bộ nhớ vật lý. Để khắc phục điều này, phải đặt một giới hạn cấp độ gián tiếp (ví dụ: giới hạn cấp độ gián tiếp tối đa cho một chỉ thị là 16). Khi tham chiếu gián tiếp đầu tiên xuất hiện, con đếm được đặt giá trị 16 và giá trị con đếm này tự động giảm đi 1 sau mỗi lần tham chiếu gián tiếp. Nếu con đếm giảm đến 0, lỗi quá tải tham chiếu gián tiếp xuất hiện để trả quyền điều khiển cho HĐH. Giới hạn này làm giảm số lượng tham chiếu bộ nhớ trên mỗi chỉ thị xuống còn 17, và do đó mỗi tiến trình cần tối thiểu 17 frame.

Số frame tối thiểu cho mỗi tiến trình được quy định bởi kiến trúc máy tính, số frame tối đa bị giới hạn bởi bộ nhớ vật lý. Trong khoảng cận trên và cận dưới này, hệ thống phải đưa ra phương pháp cấp phát phù hợp.

10.6.2. Thuật toán cấp phát

Để chia m frame cho n tiến trình, đơn giản nhất là phương pháp cấp phát đều, mỗi tiến trình được m/n frame.

Tuy nhiên trên thực tế, các tiến trình có nhu cầu bộ nhớ khác nhau. Nếu hệ thống có 60 frame và chỉ có hai tiến trình A (có độ lớn 10KB) và B (140KB), thì không thể cấp phát cho mỗi tiến trình đúng 30 frame. Tiến trình A chỉ cần đúng 10 frame, vì thế lãng phí 21 frame. Một cách giải quyết là sử dụng phương pháp cấp phát tỷ lệ. Lượng bộ nhớ cấp phát cho mỗi tiến

trình tỷ lệ với kích thước tiến trình. Giả sử kích thước bộ nhớ ảo của tiến trình p_i là s_i, đặt $S = \sum s_i$. Khi đó, nếu tổng số frame hiện có là m, thì tiến trình p_i nhận được $a_i = \frac{s_i}{S} \times m$ frame. Tất nhiên, a_i là phần nguyên của tỷ lệ trên và phải lớn hơn số frame tối thiểu của tiến trình. Áp dụng cấp phát tỷ lệ, có thể chia 60 frame cho hai tiến trình là tiến trình 10 trang nhớ nhận được 4 frame và tiến trình 140 trang nhận được 56 frame.

Với cấp phát đều hay cấp phát tỷ lệ, số lượng frame cấp phát cho mỗi tiến trình phụ thuộc vào mức độ đa chương trình. Nếu mức đa chương tăng, HĐH thu bớt mỗi tiến trình một số frame để cấp phát cho tiến trình mới. Ngược lại, nếu mức độ đa chương giảm thì mỗi tiến trình nhận được nhiều frame hơn.

Chú ý, trong 2 phương pháp cấp phát trên, tiến trình có độ ưu tiên cao cũng bị đối xử giống tiến trình có độ ưu tiên thấp. Mặc dù tiến trình có độ ưu tiên cao nên được ưu tiên cấp phát bộ nhớ hơn để đảm bảo tốc độ thi hành. Có thể vẫn sử dụng phương pháp cấp phát tỷ lệ, nhưng tỷ lệ frame được cấp không chỉ phụ thuộc vào kích thước tiến trình mà còn vào độ ưu tiên của tiến trình.

cuu duong than cong . com

10.6.3. Cấp phát toàn cục và cục bộ

Với nhiều tiến trình cạnh tranh bộ nhớ, có thể phân thuật toán thay thế trang vào hai nhóm là thay thế toàn cục và thay thế cục bộ. Thay thế toàn cục cho phép tiến trình lựa chọn frame để thay thế trong toàn bộ danh sách frame, cho dù frame đó đã cấp phát cho tiến trình khác. Như vậy, tiến trình có thể lấy frame từ tiến trình khác. Thay thế cục bộ yêu cầu tiến trình chỉ chọn thay thế trong những frame đã được cấp phát cho mình.

Xét phương pháp cho phép tiến trình có độ ưu tiên cao có thể lấy frame từ tiến trình có độ ưu tiên thấp. Tiến trình có độ ưu tiên cao có thể tăng số frame được cấp phát của mình bằng cách lấy frame của các tiến trình có độ ưu tiên thấp. Với thay thế cục bộ, tổng số frame cấp phát cho mỗi tiến trình không đổi. Với thay thế toàn cục, tiến trình có thể lấy frame từ tiến trình khác để làm tăng số frame được cấp phát của mình.

Trong thuật toán thay thế toàn cục, tiến trình không thể quản lý được tỷ lệ lỗi trang. Tập hợp trang trong bộ nhớ của tiến trình phụ thuộc không chỉ

vào hành vi phân trang của tiến trình, mà còn phụ thuộc vào hành vi phân trang của tiến trình khác. Bởi vậy, các tiến trình giống nhau có thể có thời gian thực hiện tương đối khác nhau, và phụ thuộc vào những yếu tố ngoài. Trong khi đó, với phương pháp thay thế cục bộ, tập hợp trang trong bộ nhớ của tiến trình chỉ bị ảnh hưởng bởi hành vi phân trang của tiến trình đó.

10.7. PHÂN ĐOẠN THEO YÊU CẦU

Mặc dù phân trang theo yêu cầu được đánh giá là hệ thống bộ nhớ ảo hiệu quả nhất, nhưng cần nhiều phần cứng hỗ trợ. Đó chính là nguyên nhân ra đời phân đoạn theo yêu cầu. Kiến trúc Intel 80286 không phân trang, nhưng phân đoạn bộ nhớ. HĐH OS/2 khi chạy trên kiến trúc này sẽ dùng phần cứng phân đoạn để cài đặt phân đoạn theo yêu cầu – một cách thức gần giống phân trang theo yêu cầu. OS/2 cấp phát bộ nhớ theo đoạn và kiểm soát các đoạn thông qua bộ mô tả đoạn (segment descriptors) – chứa thông tin về kích thước, mức bảo vệ, vị trí của đoạn. Không nhất thiết phải tải tất cả các đoạn của tiến trình vào trong bộ nhớ. Trong bộ mô tả đoạn, mỗi đoạn sẽ có trường bit hợp lệ, nhằm xác định đoạn hiện có nằm trong bộ nhớ hay không. Khi tiến trình truy cập đến đoạn nào đấy, phần cứng sẽ kiểm tra mã hợp lệ. Nếu đoạn đã nằm trong bộ nhớ, tiến trình tiến hành truy cập bình thường. Nếu đoạn chưa nằm trong bộ nhớ, phần cứng tạo ra ngắt (lỗi đoạn) để HĐH nắm lấy quyền sử dụng CPU giống như phân trang theo yêu cầu. Sau đó, OS/2 hoán chuyển một đoạn sang ổ cứng và tải đoạn được yêu cầu vào bộ nhớ chính. Chỉ thị bị tạm dừng trước khi phát sinh lỗi đoạn được khôi phục để tiếp tục thực hiện. Để xác định đoạn nào bị thay thế trong trường hợp lỗi đoạn, OS/2 sử dụng bit truy cập (accessed bit) trong bộ mô tả đoạn. Bit truy cập giống bit tham chiếu trong phân trang theo yêu cầu, được thiết lập (nhận giá trị 1) khi bất cứ byte nào trong đoạn được đọc hoặc ghi. Sử dụng một hàng đợi duy trì chỉ mục cho mỗi đoạn trong bộ nhớ. Sau mỗi khoảng thời gian xác định, OS/2 đưa danh sách các đoạn vừa được truy cập vào đầu hàng đợi. Sau đó sẽ xóa bit truy cập của các đoạn. Hàng đợi này sắp xếp theo thứ tự thời gian và đoạn được truy cập gần nhất nằm tại đỉnh. Bên cạnh đó, OS/2 có lời gọi hệ thống cho các tiến trình thông báo cho HĐH các đoạn không được loại bỏ, hoặc phải luôn nằm trong bộ nhớ. Thông tin này giúp cho HĐH sắp xếp thứ tự các đoạn trong hàng đợi. Khi lỗi đoạn xuất

hiện, trước tiên bộ phận quản lý bộ nhớ phải xác định xem có đủ bộ nhớ trống để tải đoạn vào không. Thu gọn bộ nhớ có thể được thực hiện để làm giảm phân mảnh ngoài. Nếu sau khi thu gọn mà vẫn không đủ bộ nhớ trống, phải thực hiện thay thế đoạn. Đoạn nằm cuối hàng đợi sẽ được thay thế bằng cách hoán chuyển ra ổ đĩa cứng. Nếu vùng trống trong bộ nhớ vừa được giải phóng đủ lớn, đoạn được yêu cầu sẽ được tải vào bộ nhớ, bộ mô tả đoạn được cập nhật. Nếu vẫn thiếu bộ nhớ, hệ thống lại phải thay thế thêm một đoạn khác và quá trình này lại tiếp tục.

10.8. NHẬN XÉT

Người sử dụng mong muốn thực hiện tiến trình có không gian địa chỉ logic lớn hơn lượng bộ nhớ được cấp phát. Lập trình viên có thể thực hiện điều này bằng cách tổ chức chương trình theo kỹ thuật phủ, nhưng công việc này rất phức tạp. Kỹ thuật bộ nhớ ảo cho phép ánh xạ không gian địa chỉ logic lên một vùng nhớ vật lý có kích thước nhỏ hơn. Bộ nhớ ảo làm tăng mức độ đa chương trình, nâng cao hiệu suất sử dụng CPU. Hơn nữa, người lập trình không còn bận tâm về vấn đề bộ nhớ thực. Phương pháp phân trang thuần túy chỉ tải trang nhớ thực sự cần đến vào. Tham chiếu đầu tiên gây ra lỗi trang để chuyển quyền điều khiển cho HĐH. HĐH tra cứu bảng phân trang xác định trang nằm ở đâu trên ổ cứng, tìm một frame trống để tải trang từ ổ đĩa cứng vào. Sau đó HĐH cập nhật bảng phân trang và khởi động lại chỉ thị bị dừng do lỗi trang. Giải pháp này cho phép tiến trình chạy ngay cả khi vùng nhớ cần thiết chưa nằm trong bộ nhớ chính. Nếu giữ được tỷ lệ lỗi trang tương đối thấp, hiệu suất của hệ thống hoàn toàn chấp nhận được. Phân trang theo yêu cầu cho phép giảm số lượng frame cấp phát cho tiến trình, điều này làm tăng mức đa chương (cho phép nhiều tiến trình chạy cùng một lúc).

cuu duong than cong . com

CÂU HỎI ÔN TẬP

1. Phân biệt bộ nhớ ảo và bộ nhớ vật lý.
2. Trình bày phương pháp phân trang, phân đoạn.
3. Trình bày các vấn đề liên quan đến hiệu suất với kỹ thuật phân trang.
4. Trình bày các vấn đề khi phải thay thế trang.

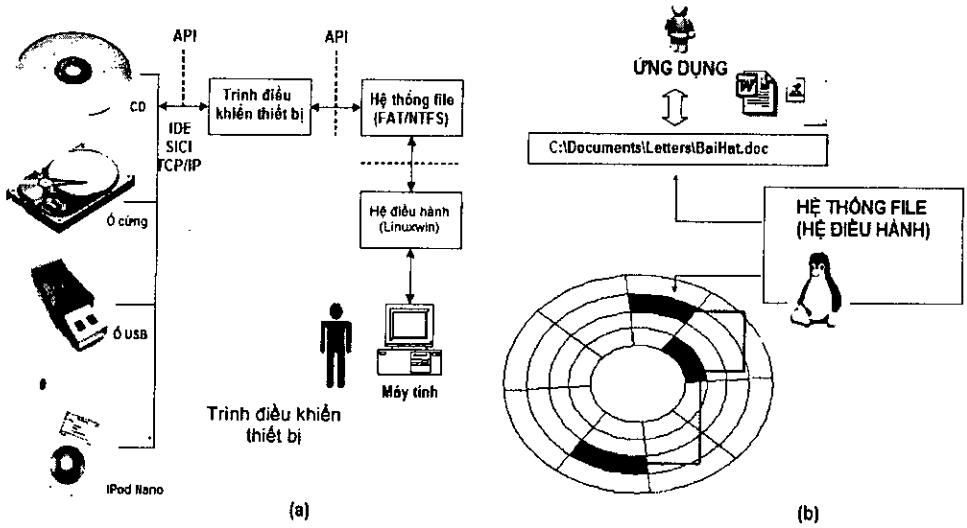
Chương 11
HỆ THỐNG FILE

File là cơ chế lưu trữ thông tin lâu dài. Người sử dụng, lập trình viên và chương trình lưu thông tin trên thiết bị lưu trữ bằng cách ghi lên file. Các chương trình khả thi cũng được ghi thành file lưu trên thiết bị. Do đó, quản lý hệ thống file là nhiệm vụ quan trọng của HĐH. Hệ thống file bao gồm hai thành phần độc lập là tập hợp các file và cấu trúc thư mục – có nhiệm vụ tổ chức và cung cấp thông tin về tất cả các file trong hệ thống. Chương này trình bày các khía cạnh khác nhau của file và cấu trúc thư mục; các phương pháp bảo vệ và chia sẻ file.

cuu duong than cong . com

11.1. FILE

File là tập hợp thông tin được lưu cùng nhau trên thiết bị và được xác định qua tên gọi. Đa phần người dùng đọc/ghi thiết bị thông qua việc đọc/ghi file. Do đó, file là mức trừu tượng thiết bị lưu trữ cao nhất và quan trọng nhất. Ví dụ như trong Hình 11.1b, người dùng và chương trình soạn thảo văn bản có thể ghi và sử dụng file BaiHat.doc mà không cần biết file này ghi trên các sector như thế nào. Qua hệ thống file, HĐH thống nhất (về mặt logic) tất cả các môi trường lưu trữ vật lý khác nhau. Trong Hình 11.1a, đối với người sử dụng, file trên ổ CD hay trên ổ đĩa cứng, ổ USB là hoàn toàn tương tự. Thông thường, file chứa chương trình (dạng mã nguồn hay mã đối tượng) và dữ liệu. File có thể có cấu trúc xác định, nhưng cũng có thể không theo định dạng nào cả. File là chuỗi các byte, dòng văn bản, hay bản ghi mà nội dung được người tạo và người sử dụng thống nhất định nghĩa.



Hình 11.1. Vị trí, vai trò của hệ thống file

11.1.1. Thuộc tính file

Để thuận tiện, tên file thường là chuỗi ký tự, ví dụ "BaiHat.doc". Có thể phân biệt (như UNIX) hoặc không phân biệt (như Windows) chữ hoa/chữ thường trong tên file. Sau khi tạo ra, file độc lập với tiến trình, người dùng và ngay cả hệ thống tạo ra file. Ví dụ, người B có thể chỉnh sửa file "BaiHat.doc" do người A tạo ra. Trong thư mục chứa file, sẽ có mục ứng với file, mục này có thể coi là một bản ghi với các trường là thuộc tính của file. Thư mục được trình bày ở phần sau, ở đây chỉ cần xem thư mục là một mảng các bản ghi, mỗi bản ghi (hay mục) ứng với một file. Trong hệ thống chứa nhiều file, kích thước thư mục có thể lên tới nhiều MB. Giống file, thư mục cũng được lưu trữ trên thiết bị và tải vào bộ nhớ từng phần khi cần thiết. Số lượng thuộc tính của file phụ thuộc vào HĐH, nhưng thông dụng nhất là các thuộc tính sau:

- **Tên (Name)**: Tên biểu tượng của file được người dùng sử dụng khi muốn truy xuất tới file.
- **Kiểu (Type):** Thông tin này cần thiết cho các hệ thống hỗ trợ nhiều kiểu file khác nhau.
- **Vị trí (Location):** Địa chỉ các khối dữ liệu của file trên thiết bị.
- **Kích thước file (Size):** Kích thước file hiện tại (tính theo byte, word hay block), có thể thông tin này cũng chứa thêm tham số xác định kích thước file tối đa.

- **Bảo vệ (Protection):** Thông tin về quyền truy cập, xác định người dùng nào có quyền đọc/ghi (hoặc thực thi) file.
- **Thời gian, định danh người dùng:** Các thời điểm sau có thể được ghi lại: (1) Thời điểm tạo file; (2) Thời điểm sửa chữa cuối cùng; (3) Thời điểm sử dụng cuối cùng.

Những thông tin này giúp cho việc bảo vệ, bảo mật hay kiểm tra quá trình sử dụng file.

11.1.2. Kiểu file (File type)

Thao tác của HĐH trên những kiểu file được hỗ trợ sẽ hợp lý hơn. Ví dụ, nếu xác định được file là mã chương trình nhị phân, HĐH sẽ không cho phép người dùng in nội dung file để tạo ra các trang in vô nghĩa. Trong HĐH MS-DOS, tên file được tách thành hai phần, ngăn cách bởi dấu chấm: phần tên có tối đa tám ký tự và phần mở rộng có tối đa ba ký tự. Phần mở rộng xác định kiểu file và các thao tác có thể thực hiện trên file. Chỉ file với phần mở rộng là ".com", ".exe" (kiểu file chứa mã nhị phân khả thi), hay ".bat" (file chứa lệnh) mới có thể thực thi. Mặc dù HĐH chỉ hỗ trợ một vài phần mở rộng, nhưng chương trình ứng dụng sẽ chỉ rõ kiểu file chúng có thể xử lý. Chẳng hạn, chương trình bảng tính Excel tạo ra file bảng tính có đuôi ".xls". Phần mở rộng như vậy không được HĐH quy định và hỗ trợ, mà chỉ được xem như "gợi ý" xác định ứng dụng nào có thể xử lý file.

Trong HĐH Apple Macintosh, file có kiểu, chẳng hạn "text" là kiểu văn bản, "pict" là kiểu hình ảnh. Thuộc tính tạo lập file là tên chương trình tạo ra file và được HĐH thiết lập ngay khi chương trình ứng dụng tạo ra file. Ví dụ, thuộc tính của file do trình xử lý văn bản tạo ra là tên chương trình xử lý văn bản. Khi file được mở (người dùng nhấn đúp chuột vào biểu tượng file), HĐH tự động gọi trình xử lý văn bản mở file.

11.1.3. Cấu trúc file

Dữ liệu mà ứng dụng xử lý thường được tổ chức dưới dạng *bản ghi logic* với nhiều trường. Ví dụ, Công ty điện thoại có thể lưu hóa đơn điện thoại của khách hàng dưới dạng file, mỗi file có trường Tên khách hàng, Số tiền phải trả,... Các hệ thống ổ đĩa cứng thường chia thành các khối có kích thước xác định, phụ thuộc vào kích thước sector vật lý. Thao tác vào/ra trên

ổ đĩa được thực hiện theo đơn vị khối (Bản ghi vật lý). Kích thước bản ghi vật lý và bản ghi logic tiến trình ứng dụng cần đọc khác nhau, do đó phải "gói" một số bản ghi logic vào trong khối vật lý. Như vậy, hệ thống file phải chịu trách nhiệm ánh xạ từ cấu trúc file sang các khối byte trên thiết bị lưu trữ. Cấu trúc dữ liệu cần ghi lên ổ đĩa đầu tiên sẽ được "làm phẳng" thành chuỗi byte, rồi chuỗi byte này được ghi thành từng khối trên thiết bị lưu trữ. Quá trình đọc ngược lại cũng diễn ra tương tự. Vấn đề ở đây là, đặt chức năng chuyển đổi bản ghi thành luồng byte này ở đâu? Ở trình ứng dụng hay tại hệ thống file của HĐH? Tức là, HĐH có hỗ trợ cấu trúc file hay không?

UNIX coi file là luồng byte. Địa chỉ byte là khoảng cách tới byte đầu tiên của file. Khi đó, bản ghi logic có kích thước một byte. Hệ thống file tự động "đóng gói" và "mở gói" các byte vào trong khối đĩa cứng vật lý (thường là 512 byte một khối). Việc "đóng gói" có thể được trình ứng dụng của người dùng hay chính HĐH thực hiện. Trong cả hai trường hợp, file được xem như chuỗi các khối. Tất cả các hàm vào/ra cơ bản thao tác trên khối. Ổ đĩa được cấp phát theo đơn vị khối, vì thế khối cuối cùng của file chưa chắc được sử dụng hết, điều này gây nên lãng phí. Nếu kích thước khối là 512 byte, file kích thước 2000 byte sẽ được cấp phát bốn khối (2048 byte), 48 byte cuối cùng bị lãng phí. Đây là hiện tượng phân mảnh trong, nói chung xuất hiện trên tất cả các hệ thống file. Kích thước khối vật lý càng lớn, hiện tượng phân mảnh trong càng trầm trọng.

Kiểu file có thể xác định cấu trúc file. Một số HĐH có thể hỗ trợ nhiều cấu trúc file khác nhau. HĐH hỗ trợ nhiều cấu trúc file trở nên "cồng kềnh" vì cần nhiều module cho các cấu trúc riêng biệt. Một số HĐH (UNIX hay MS-DOS) chỉ hỗ trợ lượng tối thiểu các cấu trúc file. UNIX coi file là chuỗi byte 8 bit và không thực hiện phiên dịch luồng byte. Phương pháp này tuy linh hoạt, nhưng không hỗ trợ ứng dụng. Chính ứng dụng phải thực hiện ánh xạ luồng byte vào cấu trúc phù hợp. Trong HĐH Macintosh, file bao gồm phần tài nguyên và phần dữ liệu. Phần tài nguyên chứa thông tin người dùng có thể nhìn thấy, chẳng hạn, nhãn của các nút bấm mà chương trình hiển thị. Người dùng ở nước khác có thể thay đổi lại nhãn những nút bấm này theo ngôn ngữ của mình bằng các công cụ do HĐH Macintosh cung cấp. Phần dữ liệu chứa mã chương trình và dữ liệu mà người dùng không sửa đổi được. Để thực hiện công việc tương tự trên UNIX hay MS-DOS, lập trình viên

phải thay đổi mã nguồn, sau đó biên dịch lại mã nguồn, hoặc lập trình viên tạo ra file dữ liệu mà người dùng có thể thay đổi được. Như vậy, lập trình viên sẽ thuận tiện hơn nếu HĐH hỗ trợ các cấu trúc file sử dụng thường xuyên, nhưng nếu hỗ trợ nhiều cấu trúc thì HĐH cồng kềnh. Tuy nhiên, tất cả HĐH phải hỗ trợ cấu trúc file khả thi để hệ thống có thể tải và chạy chương trình.

11.1.4. File mức thấp

HĐH cung cấp các lời gọi hệ thống để tạo, ghi, đọc, dịch chuyển vị trí con trỏ, xóa, ghép nối file. Các thao tác phức tạp hơn là tổ hợp của các thao tác cơ sở.

- **Tạo mới file**: Đầu tiên HĐH tìm các vị trí trống trong thiết bị lưu trữ. Tiếp theo, một mục ứng với file mới được chèn vào hệ thống thư mục. Các thành phần trong mục ghi tên file, vị trí dữ liệu của file trên thiết bị cũng như các thông tin khác.
- **Ghi file**: Ứng dụng sử dụng lời gọi hệ thống với tham số xác định file được ghi và thông tin sẽ ghi. Sau đó HĐH tìm kiếm trong cấu trúc thư mục để xác định vị trí file trong cấu trúc thư mục. Hệ thống sử dụng con trỏ ghi trỏ tới vị trí sẽ được ghi trong lần ghi kế tiếp. Con trỏ ghi tự động cập nhật sau mỗi thao tác ghi.
- **Đọc file**: Ứng dụng sử dụng lời gọi hệ thống với tham số xác định file cần đọc và vị trí (trong bộ nhớ) chứa dữ liệu đọc ra từ file. HĐH tìm kiếm vị trí file trong hệ thống thư mục, sử dụng con trỏ đọc (read pointer) trỏ tới vị trí sẽ được đọc trong lần đọc kế tiếp. Giá trị con trỏ đọc sẽ tự động cập nhật sau mỗi lần đọc. Tuy nhiên, phần lớn file mở ở chế độ vừa đọc, vừa ghi, nên hệ thống chỉ sử dụng một con trỏ là con trỏ vị trí hiện thời trong file. Cả hai lời gọi đọc và ghi đều cùng sử dụng con trỏ này để tiết kiệm không gian nhớ cũng như giảm độ phức tạp hệ thống.
- **Thay đổi vị trí con trỏ file**: Con trỏ vị trí file hiện thời sẽ được thiết lập giá trị mới. Thao tác này không đòi hỏi đọc dữ liệu từ ổ đĩa cứng.
- **Xóa file**: HĐH tìm kiếm file cần xóa trong cấu trúc thư mục. Nếu tìm thấy, HĐH thu hồi toàn bộ không gian dữ liệu của file, xóa mục ứng với file trong hệ thống thư mục.

Để tránh việc duyệt hệ thống thư mục để tìm file trong mỗi lần thao tác, nhiều HĐH "mở" file trong lần đầu tiên sử dụng. HĐH có bảng chứa thông tin về tất cả các file đang mở. Khi thực hiện thao tác, HĐH sử dụng chỉ mục trong bảng mà không cần duyệt hệ thống thư mục. Khi không còn được sử dụng, tiến trình đóng file và HĐH xóa hàng ứng với file trong bảng file mở.

Một số HĐH tự động thực hiện mở file trong lần tham chiếu đầu tiên tới file. File cũng được đóng lại tự động khi tiến trình mở file kết thúc. Tuy nhiên, phần lớn các HĐH yêu cầu lập trình viên thực hiện tường minh lời gọi hệ thống mở file (**open**) trước khi sử dụng. Thao tác này có tham số là tên file và HĐH sẽ tìm kiếm trên cấu trúc thư mục để sao chép bản ghi ứng với file trong cấu trúc thư mục vào bảng file mở (ở đây giả thiết người sử dụng có quyền truy cập tới file). Lời gọi **open** trả cho tiến trình mở file con trỏ trỏ tới hàng nào đó trong bảng file mở. Con trỏ này chứ không phải tên file được sử dụng trong tất cả thao tác kế tiếp.

Việc cài đặt thao tác mở (**open**) và đóng (**close**) trong môi trường đa người dùng rất phức tạp, vì cùng lúc có thể nhiều người mở cùng một file. Do đó, HĐH có hai mức bảng khác nhau. *Bảng cấp tiến trình* chứa thông tin về tất cả các file mà tiến trình mở. Bảng này ghi lại thông tin liên quan đến cách sử dụng file của tiến trình, ví dụ con trỏ file hiện thời ứng với mỗi file được mở. Mỗi hàng trong bảng cấp tiến trình trỏ tới hàng nào đó trong *Bảng file mở hệ thống*. Một vài thông tin sau HĐH cần xác định khi mở file:

- **Con trỏ file (File pointer)**: Nếu lời gọi read/write không có tham số xác định vị trí cần đọc/ghi trong file (thường là khoảng cách tới vị trí đầu tiên của file), thì HĐH phải kiểm soát vị trí đọc/ghi cuối cùng, còn gọi là con trỏ vị trí file hiện thời. Mỗi tiến trình thao tác trên file có một con trỏ duy nhất, do đó cần được lưu giữ độc lập với các thuộc tính cố định của file.
- **Bộ đếm file mở**: Khi đóng file, HĐH giải phóng các vùng ứng với file trên bảng file mở hệ thống (để tiết kiệm không gian lưu trữ). Nếu có nhiều tiến trình cùng mở một file, thì hệ thống phải đợi đến khi không còn tiến trình nào sử dụng file nữa mới được xóa hàng tương ứng trong bảng file mở hệ thống. Bộ đếm file ghi lại số lần mở và đóng file, và nhận giá trị 0 trong lần đóng cuối cùng. Hệ thống sau đó có thể giải phóng các tài nguyên cần thiết.

- **Vị trí file trên ổ đĩa:** Nhiều thao tác yêu cầu hệ thống sửa đổi dữ liệu trong file. Do đó, cần lưu thông tin về vị trí file để tránh việc đọc lại từ ổ đĩa cứng vị trí dữ liệu trong mỗi thao tác. Một số HĐH có cơ chế ánh xạ một phần file vào bộ nhớ trong để cho phép một phần không gian địa chỉ ảo có liên kết logic tới một phần nào đó trong file. Đọc/ghi vùng bộ nhớ này tương đương đọc/ghi file. Thao tác đóng file dẫn đến việc dữ liệu trong vùng nhớ ánh xạ được ghi lại vào ổ đĩa cứng. Nhiều tiến trình có thể ánh xạ cùng một file lên bộ nhớ ảo của mình - một hình thức chia sẻ dữ liệu. Bất kỳ tiến trình nào sửa đổi dữ liệu trong bộ nhớ ảo thì các tiến trình khác đều "nhìn" thấy.

11.1.5. File có cấu trúc

Ứng dụng coi file là tập hợp các bản ghi phải biến luồng byte từ thiết bị thành các bản ghi. Cấu trúc bản ghi do ứng dụng quy định. Có nhiều phương pháp để hệ thống phân biệt các bản ghi khác nhau trong file.

– Kích thước bản ghi cố định.

– Bản ghi có tiêu đề, trong tiêu đề có trường độ dài bản ghi.

– Cuối bản ghi có cụm ký tự đặc biệt đánh dấu điểm kết thúc.

Khi đó, bên cạnh các hàm thao tác file cơ sở, HĐH phải có thêm những hàm làm việc với bản ghi (bổ sung, xóa bản ghi, thay đổi nội dung bản ghi,...).

11.1.6. Các phương pháp truy cập file

Các HĐH thường hỗ trợ nhiều kiểu truy cập file khác nhau.

- **Truy cập tuần tự:** Thông tin trong file được đọc tuần tự, hết bản ghi này sang bản ghi khác. Thao tác **read** (**write**) đọc (ghi) phần kế tiếp và tự động dịch chuyển con trỏ đọc. HĐH có thể tiến hành dịch chuyển con trỏ lên phía trước hoặc phía sau n bản ghi.
- **Truy cập trực tiếp:** File được xem là dãy các bản ghi hoặc khối có đánh số thứ tự và tiến trình có thể đọc/ghi tại bất kỳ bản ghi (hay khối) nào. File truy cập trực tiếp có ưu điểm lớn khi cần truy cập ngay lập tức một khối lượng thông tin lớn. Các hệ cơ sở dữ liệu (CSDL) thường hoạt động theo kiểu này. Khi truy vấn đối tượng cụ

thể, HĐH tính toán để xác định địa chỉ của khối chứa dữ liệu và sau đó đọc trực tiếp khối để lấy ra thông tin cần thiết.

- **Các phương thức truy cập khác:** Từ phương thức truy cập trực tiếp, có thể cài đặt phương thức truy cập thông qua bảng chỉ số file. Bảng chỉ số giống như mục lục ở cuối sách, nó chứa con trỏ đến các khối dữ liệu khác nhau. Để truy xuất tới bản ghi trong file, đầu tiên HĐH tìm kiếm trên bảng chỉ số, sau đó sử dụng con trỏ để truy cập trực tiếp tới bản ghi nằm trong file. Bảng chỉ số của file có kích thước rất lớn, có thể quá lớn để có thể lưu trong bộ nhớ. Giải pháp là tạo bảng chỉ số cho chính bảng chỉ số file, khi đó bảng chỉ số file cấp 1 sẽ chứa con trỏ đến bảng chỉ số file cấp 2, và chỉ số file cấp 2 mới trỏ đến đối tượng dữ liệu thực sự.

11.2. CÀI ĐẶT FILE Ở MỨC THẤP

11.2.1. Tổ chức hệ thống file

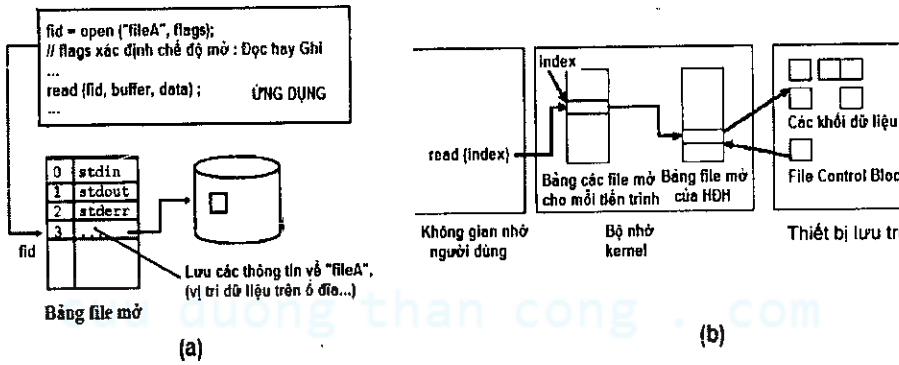
Có hai vấn đề cần quan tâm khi thiết kế hệ thống file. Đầu tiên là cách định nghĩa hệ thống file theo quan điểm người dùng (xác định thế nào là file, các thuộc tính cũng như những thao tác được phép thực hiện trên file, cấu trúc thư mục để tổ chức file). Vấn đề thứ hai là thuật toán và cấu trúc dữ liệu ánh xạ hệ thống file logic sang thiết bị lưu trữ vật lý thứ cấp.

Chính bản thân hệ thống file cũng được cài đặt trong nhiều tầng khác nhau. Cấu trúc trong Hình 11.1a minh họa quan điểm thiết kế phân tầng. Mỗi tầng sử dụng dịch vụ của tầng bên dưới để cung cấp dịch vụ mới cho tầng bên trên.

Tầng thấp nhất là trình điều khiển thiết bị, thực hiện trao đổi dữ liệu giữa bộ nhớ và thiết bị. Có thể xem trình điều khiển thiết bị như một trình thông dịch: nhận lệnh từ tầng cao chuyển xuống (chẳng hạn "retrieve block 12345") và ra lệnh cho tầng thấp hơn bằng cách ghi chỉ thị vào vị trí cụ thể trong bộ nhớ của bộ điều khiển vào/ra (thanh ghi lệnh) để yêu cầu bộ điều khiển thực hiện thao tác nào đó trên thiết bị.

Hệ thống file đưa ra lệnh tổng quát cho trình điều khiển thiết bị phù hợp. Mỗi khối vật lý có một địa chỉ cụ thể nào đó trên thiết bị (drive 1, cylinder 73,...). Nếu hệ thống có nhiều loại thiết bị lưu trữ khác nhau, thì

cũng cần nhiều trình điều khiển khác nhau. Nhưng hệ thống file che dấu tất cả những trình điều khiển này với các tầng bên trên. Hệ thống file chịu trách nhiệm xác định quan hệ giữa khối logic với khối vật lý chứa dữ liệu tương ứng. Biết về cách thức cấp phát cũng như vị trí của khối dữ liệu trong file, hệ thống file ánh xạ địa chỉ logic của khối sang địa chỉ vật lý. Bên cạnh việc quản lý các khối chưa được sử dụng, hệ thống file phải cài đặt cấu trúc thư mục, cài đặt cơ chế bảo vệ và an ninh. Một số hệ thống có thể phân hệ thống file thành hai tầng: tầng cơ sở cài đặt một hệ thống file cụ thể nào đó (FAT, NTFS,...) và tầng logic che dấu tất cả các đặc tính của các tầng cơ sở khác nhau.



Hình 11.2. Bảng file mở

Để tạo file mới, trình ứng dụng gọi hệ thống file thông qua HĐH. Sau đó, hệ thống file đọc thư mục phù hợp vào trong bộ nhớ, tạo ra một mục mới trong thư mục, sau đó ghi lại vào ổ đĩa. Một số HĐH (như UNIX) coi thư mục là file đặc biệt, có trường xác định là file kiểu thư mục. Một số HĐH như Windows NT lại xem thư mục là kiểu thực thể riêng, độc lập với file.

Khi tiến trình yêu cầu đọc hay ghi file, HĐH tìm kiếm trên cấu trúc thư mục để kiểm tra các tham số, định vị các khối dữ liệu trên thiết bị, thực hiện thao tác trên từng khối dữ liệu. Mỗi thao tác yêu cầu hệ thống tiến hành khá nhiều công việc phụ trội. Do vậy, trước khi sử dụng, cần phải "mở" file. Khi mở file, HĐH tìm trong cấu trúc thư mục để xác định mục ứng với file. Thường thì, một phần cấu trúc thư mục được lưu trữ tạm trong bộ nhớ để tăng tốc độ tìm kiếm. Nếu tìm thấy file, tất cả thông tin liên quan đến file (kích thước, quyền sở hữu, quyền truy cập, vị trí các khối dữ liệu) được sao chép vào một hàng nào đó trong bảng file mở trong bộ nhớ chính. Sau khi gọi open, HĐH trả lại cho chương trình người dùng chỉ mục trong bảng ứng với file (chỉ mục này là số thứ tự của hàng chứa thông tin về file trong bảng

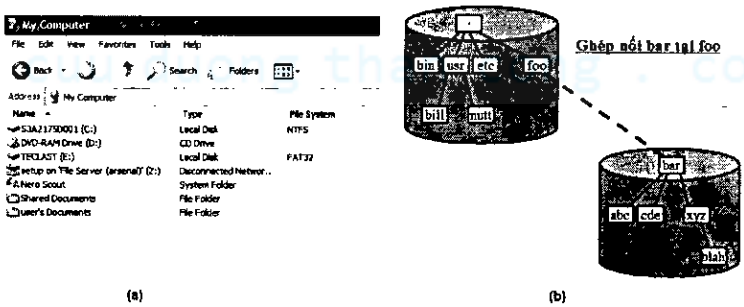
file mở). Tất cả các tham chiếu sau này tới file sử dụng chỉ mục làm tham số chứ không sử dụng tên file nữa. Chú ý, chỉ mục này có nhiều tên gọi khác nhau, HĐH UNIX gọi là bản mô tả file (file descriptor), HĐH Windows NT gọi là thẻ file (file handle). Sau khi file không còn được sử dụng, toàn bộ thông tin cập nhật về file sẽ được sao chép lại vào trong cấu trúc thư mục gốc nằm trên ổ đĩa.

HĐH UNIX sử dụng hai mức bảng. Mỗi tiến trình có bảng file mở riêng, là các con trỏ trỏ tới bảng file mở hệ thống. Bảng này nằm trong bộ nhớ chính và chứa thông tin cũng như vị trí các khối dữ liệu của file đang được sử dụng. Khi mở file, tất cả các thông tin về file (ngoại trừ các khối dữ liệu thực sự) sẽ nằm trong bộ nhớ. Như vậy, bất kỳ tiến trình nào có nhu cầu có thể nhanh chóng truy cập tới file. Trên thực tế, khi thực hiện lời gọi **open**, hệ thống kiểm tra bảng file mở để xem file đã được tiến trình nào đó mở hay chưa. Nếu có, một mục mới được chèn vào bảng file mở, cấp tiến trình và trỏ tới mục ứng với file trong bảng cấp hệ thống. Nếu chưa có, hệ thống sao chép các thông tin cần thiết và chèn thêm một mục mới vào bảng file mở cấp hệ thống và cấp tiến trình.

cuu duong than cong . com

11.2.2. Ghép nối hệ thống file (Mount)

Phải ghép nối hệ thống file vào HĐH trước khi sử dụng. Đầu tiên HĐH sẽ kiểm tra tính hợp lệ của hệ thống file trên thiết bị, sau đó HĐH "ghi lại" hệ thống file vào cấu trúc thư mục của HĐH tại một điểm "ghép nối" xác định, tức là HĐH gắn kết hệ thống file vào HĐH. Trong Hình 11.3a, ổ đĩa cứng có thể được gắn vào điểm (ổ logic) C trong My Computer. Phương pháp này cho phép HĐH duyệt qua cấu trúc thư mục và di chuyển giữa các hệ thống file khi cần thiết.



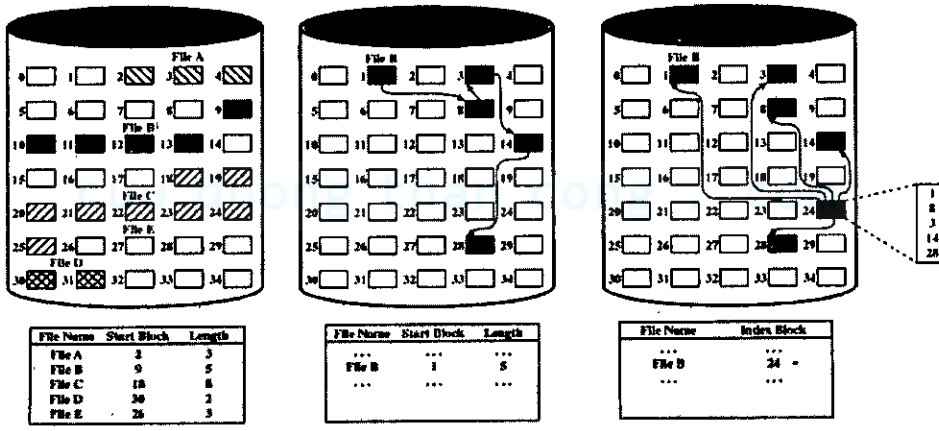
Hình 11.3. Ghép nối hệ thống file

11.2.3. Quản lý khối trong ổ đĩa cứng

Khả năng truy cập trực tiếp của ổ đĩa cứng cho phép cài đặt hệ thống file một cách linh hoạt. Khi có nhiều file ghi trên ổ đĩa, thì vấn đề đặt ra là cấp phát không gian cho những file này như thế nào để sử dụng không gian ổ đĩa tối ưu và có thể truy cập nhanh chóng tới file. Có ba phương thức cấp phát không gian ổ đĩa được sử dụng rộng rãi là *liên tục* (contigous); *móc nối* (linked) và *chỉ mục* (indexed). Mỗi phương pháp có ưu, nhược điểm riêng. Do vậy, một vài hệ thống (chẳng hạn, Data General's RDOS cho dòng máy tính NOVA) hỗ trợ cả ba phương pháp. Song phần lớn các hệ thống chỉ sử dụng một phương thức xác định.

☞ ***Cấp phát liên tục (Contigous)***

Dữ liệu của file nằm trên các khối vật lý liền kề nhau như minh họa trong Hình 1.14a. Khi đó, truy cập khối (b + 1) sau khi truy cập khối b thường không phải dịch chuyển đầu đọc. Khi cần phải dịch chuyển đầu đọc (từ sector cuối của cylinder tới sector đầu tiên của cylinder kế tiếp), bước dịch chỉ là một track. Do đó, tổng số các lần dịch chuyển đầu đọc/ghi trên ổ đĩa khi truy cập tới file được cấp phát liên tục có giá trị nhỏ nhất. Các khối dữ liệu của file được xác định bởi địa chỉ của khối đầu tiên và kích thước file (tính theo đơn vị khối). Nếu có kích thước n khối và bắt đầu từ vị trí b thì file nằm trong các khối: b, b + 1, ..., b + n – 1. Trong thư mục chứa file sẽ có mục ghi lại địa chỉ khối bắt đầu và kích thước file.



(a) Cấp phát liên tục (b) Cấp phát móc nối (c) Cấp phát theo chỉ mục

Hình 11.4. Các phương pháp cấp phát

Vấn đề trong cấp phát liên tục là tìm kiếm không gian trống dành cho file mới, vấn đề tương tự bài toán cấp phát bộ nhớ trong mục 8.4 là: Làm cách nào để cấp phát khối có kích thước n từ danh sách các khối tự do? First-fit hay Best-fit là hai phương pháp phổ biến nhất. Bằng mô phỏng, người ta đã chứng minh hai phương pháp trên hiệu quả hơn Worst-fit xét trên tiêu chí thời gian và tận dụng môi trường lưu trữ. Phương pháp cấp phát liên tục cũng xảy ra hiện tượng phân mảnh ngoài (ổ đĩa bị phân thành nhiều "mảnh" trống, không "mảnh" nào đủ lớn để lưu trữ dữ liệu). Mức độ ảnh hưởng của hiện tượng phân mảnh ngoài phụ thuộc vào tổng dung lượng ổ đĩa và kích thước file trung bình. Để ngăn ngừa hiện tượng phân mảnh, có thể sử dụng chương trình thu gọn, dồn tất cả không gian trống của ổ đĩa vào một không gian liên tục. Cái giá phải trả cho việc dồn ổ đĩa chính là thời gian.

Một vấn đề quan trọng khác là cấp phát bao nhiêu khối trống cho file? Kích thước file phải được xác định ngay khi cấp phát. Trong một số trường hợp, việc xác định đơn giản (ví dụ khi thực hiện sao chép file có sẵn), tuy nhiên tại thời điểm tạo mới, rất khó ước lượng kích thước file. Nếu cấp phát cho file khối trống có kích thước nhỏ, sẽ không thể mở rộng file. Nếu lại sử dụng phương pháp cấp phát Best-fit, không gian lân cận ở cả hai phía của file có thể đã được cấp phát. Giải pháp thứ nhất là, chấm dứt chương trình người dùng với thông báo lỗi thích hợp. Người dùng sau đó phải được cấp phát nhiều không gian hơn và chạy lại chương trình. Việc chạy lại như thế rất tốn kém. Để ngăn ngừa, người dùng ước lượng kích thước file khá lớn, nhưng điều này lại dẫn đến lãng phí ổ đĩa. Giải pháp thứ hai là, hệ thống tự động tìm một khoảng trống lớn hơn, sau đó sao chép nội dung của file sang vị trí mới và giải phóng không gian cũ.

Kể cả khi kích thước file đã được xác định trước, thì việc cấp phát trước có thể không hiệu quả. Giả sử kích thước file tăng dần dần trong một thời gian dài (hàng tháng hoặc hàng năm), thì file vẫn phải được cấp phát với kích thước tối đa, thậm chí kể cả khi phần lớn không gian này có thể không được sử dụng trong một thời gian dài. Vì thế, hiện tượng phân mảnh trong xuất hiện.

Có thể khắc phục các vấn đề trên bằng cách cải tiến thuật toán cấp phát liên tục. Mỗi lần cấp phát là cấp phát một khu vực không gian liên tục của ổ

đĩa. Sau đó, nếu lượng cấp phát chưa đủ, hệ thống sẽ cấp phát thêm cho file một khu vực liên tục nữa, gọi là khu vực mở rộng (extend). Vị trí các khối dữ liệu của file được xác định thông qua vị trí khối đầu tiên và số lượng khối, cùng với con trỏ tới khối đầu tiên trong khu vực mở rộng. Trên một vài hệ thống, người dùng làm chủ file có thể thiết lập kích thước mở rộng, nhưng việc thiết lập này không hiệu quả nếu thiết đặt sai. Phân mảnh trong vẫn có thể xuất hiện nếu kích thước mở rộng quá lớn và phân mảnh ngoài sẽ xuất hiện khi các khu vực mở rộng có kích thước khác nhau được cấp phát và sau đó giải phóng.

☞ ***Cấp phát móc nối (Linked Allocation)***

Trong phương pháp này, file là các khối (có thể nằm rải rác) trên ổ đĩa được móc nối lại với nhau. Thư mục ghi lại con trỏ tới khối đầu tiên và con trỏ tới khối cuối cùng của file. Ví dụ, "file B" với năm khối có thể bắt đầu từ khối thứ 1, tiếp đến khối thứ 8, sau đó là khối 3, khối 14 và kết thúc ở khối 28 (hinhg 11.4b). Mỗi khối chứa con trỏ đến khối tiếp theo. Dĩ nhiên, người sử dụng không "nhìn thấy" những con trỏ này. Nếu kích thước khối là 512 byte và địa chỉ trên ổ đĩa chiếm 4 byte, thì mỗi khối chỉ chứa 508 byte dữ liệu thực.

Để tạo file mới, HĐH tạo mục mới trong cấu trúc thư mục. Mục chứa con trỏ tới khối dữ liệu đầu tiên của file trên ổ đĩa. Giá trị khởi tạo của con trỏ này là NIL để xác định đây là file rỗng (chưa có nội dung gì). Trường kích thước file cũng được thiết lập giá trị 0. Lời gọi ghi file (write) yêu cầu bộ phận quản lý không gian trống tìm một khối tự do để ghi dữ liệu vào, sau đó khối mới được "móc" vào cuối file. Để đọc file, hệ thống "lần" theo con trỏ từ khối này sang khối khác. Hiện tượng phân mảnh ngoài không xuất hiện trong cấp phát móc nối, vì bất kỳ khối trống nào đều có thể được sử dụng. Hệ thống cũng không cần xác định trước kích thước file. File có thể được mở rộng chừng nào còn khối trống trong hệ thống. Tuy nhiên, cấp phát móc nối cũng có nhược điểm. Vấn đề lớn nhất là phương pháp cấp phát này chỉ có thể được sử dụng hiệu quả cho các file truy cập tuần tự. Để tìm khối thứ i của file, HĐH bắt đầu từ khối đầu tiên, sau đó lần theo con trỏ cho tới khi tìm được khối i. Để xác định giá trị con trỏ, hệ thống cần một lần đọc đĩa, và đôi khi là một lần dịch chuyển đầu đọc ghi.

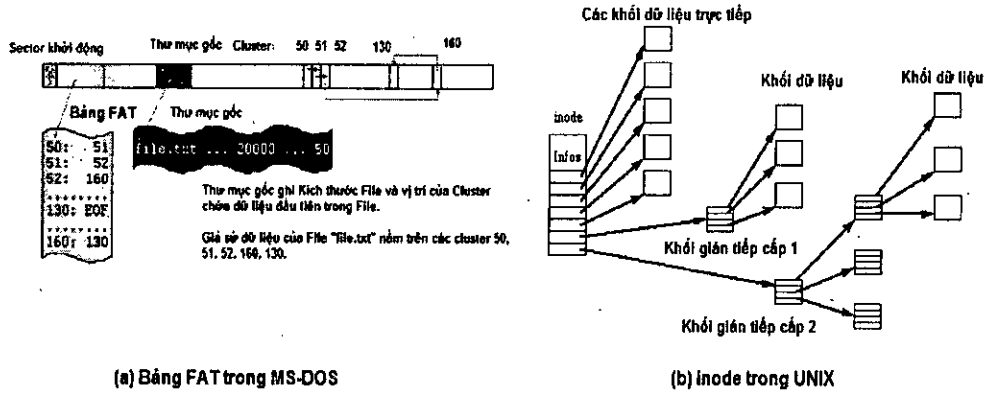
Nếu con trỏ và khối có kích thước lần lượt là 4 và 512 byte, hệ thống mất 0,78% dung lượng ổ đĩa lưu trữ con trỏ. Giải pháp khắc phục vấn đề này là nhóm các khối thành cụm và cấp phát theo đơn vị cụm. Khi đó, không gian lưu giữ con trỏ sẽ giảm đi. Phương pháp này cho phép giữ nguyên ánh xạ các khối logic tới các khối vật lý, nhưng cải thiện đáng kể tốc độ trao đổi dữ liệu với ổ đĩa (do số lần dịch chuyển đầu đọc ít hơn); giảm không gian cần thiết cho bộ quản lý cấp phát khối tự do và quản lý danh sách trống. Tuy nhiên, nếu kích thước cụm lớn thì hiện tượng phân mảnh trong có thể xuất hiện khi cụm cuối cùng của file không được sử dụng hết. Chú ý đến độ tin cậy khi cài đặt giải pháp móc nối, vì khi các khối trong file được móc nối với nhau bằng các con trỏ nằm rải rác trên toàn bộ ổ đĩa, hệ thống sẽ gặp vấn đề nếu mất một con trỏ. Lỗi trong phần mềm HĐH hoặc lỗi phần cứng có thể "phá hủy" con trỏ nào đó. Có thể con trỏ trỏ tới không gian trống hay vào một file khác. Giải pháp cho vấn đề này là dùng danh sách con trỏ móc nối kép, hoặc lưu giữ tên file và số thứ tự khối tương ứng trong mỗi khối. Nhược điểm của giải pháp này là chi phí quản lý lại tăng lên.

Bảng cấp phát file (FAT – File Allocation Table) trong MS-DOS hay OS-2 là biến thể của phương thức cấp phát móc nối. Bảng FAT nằm tại vị trí đầu tiên trong mỗi phân vùng. Mỗi ô trong bảng ứng với một khối trên ổ đĩa, các ô được sắp xếp tuyến tính. Bảng FAT đóng vai trò một danh sách liên kết. Thư mục chứa file ghi lại số thứ tự của khối dữ liệu đầu tiên của file. Trong bảng FAT, giá trị của ô ứng với số thứ tự này là địa chỉ của ô chứa khối dữ liệu tiếp theo của file. Chuỗi dây chuyền này cứ tiếp tục cho đến khối cuối cùng. Giá trị của ô ứng với khối cuối cùng nhận giá trị kết thúc file (EOF). Giá trị của ô ứng với các khối tự do chưa sử dụng là 0. Ví dụ, Hình 11.5a minh họa cấu trúc bảng FAT cho một file bao gồm các khối thứ 50, 51, 52, 130 và 160. Để cải thiện hiệu suất, bảng FAT được lưu trong bộ nhớ trong.

☞ ***Cấp phát theo chỉ mục***

Phương pháp cấp phát móc nối xóa bỏ hiện tượng phân mảnh ngoài và file không phải khai báo trước kích thước. Nhưng, nếu không có bảng FAT, các con trỏ nằm rải rác trên ổ đĩa; để truy cập đến khối nào đó trong file,

HĐH phải dò từ con trỏ đầu tiên. Cấp phát theo chỉ mục khắc phục vấn đề này bằng cách dồn toàn bộ con trỏ file vào cùng một vị trí: Khối chỉ mục.



Hình 11.5. Bảng FAT và inode

Khối chỉ mục của file là mảng chứa địa chỉ các khối dữ liệu nằm trên ổ đĩa (minh họa trên Hình 11.5b). Thành phần thứ i trong mảng là địa chỉ khối thứ i của file (giống như con trỏ). Thư mục chứa file ghi lại địa chỉ khối chỉ mục. Khi file mới được tạo ra, tất cả các con trỏ trong khối chỉ mục được thiết lập giá trị NIL (null). Trong lần đầu tiên ghi khối thứ i, bộ quản lý không gian trống tìm một khối tự do, và địa chỉ khối này được đưa vào thành phần thứ i trong khối chỉ mục. Cấp phát theo chỉ mục hỗ trợ truy cập trực tiếp và không có hiện tượng phân mảnh trong, vì hệ thống có thể cấp phát bất kỳ khối trống nào trên ổ đĩa.

Tuy nhiên, cấp phát theo chỉ mục có thể lãng phí không gian ổ đĩa, vì chi phí phụ trội cho khối chỉ mục lớn. Giả sử kích thước file chỉ là một hoặc hai khối. Với cấp phát móc nối, hệ thống chỉ tốn không gian cho một con trỏ duy nhất trên mỗi khối (do vậy, tổng cộng chỉ cần lưu trữ một đến hai con trỏ). Với cấp phát chỉ mục, file vẫn phải được cấp phát một khối trọn vẹn để làm khối chỉ mục, và trong khối chỉ mục chỉ có một hoặc hai con trỏ trỏ đến dữ liệu thực sự, các con trỏ còn lại đều có giá trị NIL. Mỗi file phải có riêng một khối chỉ mục, do vậy kích thước khối chỉ mục càng nhỏ, hệ thống càng tiết kiệm dung lượng ổ đĩa. Tuy nhiên, nếu kích thước quá nhỏ, khối chỉ mục có thể không lưu giữ hết được các con trỏ cần thiết nếu file kích thước lớn. Khi đó, có thể sử dụng một vài cách bổ trợ sau:

- **Phương pháp liên kết**: File không chỉ có một, mà có nhiều khối chỉ mục liên kết với nhau. Phần tử cuối cùng trong mỗi khối chỉ mục có

thể được sử dụng làm địa chỉ cho khối chỉ mục kế tiếp (nếu không có, trường này nhận giá trị NIL).

- **Chỉ mục nhiều tầng:** Có thể sử dụng khối chỉ mục mức 1 trỏ tới các khối chỉ mục mức 2, và khối chỉ mục mức 2 chứa địa chỉ các khối dữ liệu của file. Để truy cập tới một khối cụ thể, HĐH sử dụng chỉ mục ở mức 1 để tìm kiếm khối chỉ mục mức 2 chứa địa chỉ khối dữ liệu mong muốn. Phương pháp này có thể được tiếp tục mở rộng thành 3 hay 4 mức, phụ thuộc vào kích thước file lớn nhất mong muốn.
- **Sơ đồ kết hợp:** BSD UNIX lưu giữ 15 con trỏ trong mục ứng với file (Hình 11.5b). 12 con trỏ đầu tiên trỏ trực tiếp tới các khối dữ liệu. Như vậy, các file nhỏ (ít hơn 12 khối dữ liệu) không cần sử dụng khối chỉ mục. Nếu kích thước khối là 4K, thì 48KB dữ liệu có thể được truy cập trực tiếp. Ba con trỏ tiếp theo trỏ đến các khối gián tiếp. Con trỏ đầu tiên chứa địa chỉ của khối gián tiếp cấp một. Khối này không chứa dữ liệu mà chứa địa chỉ của các khối chứa dữ liệu. Con trỏ tiếp theo trỏ tới khối gián tiếp cấp hai chứa địa chỉ của các khối gián tiếp cấp một. Con trỏ cuối cùng trỏ tới khối gián tiếp cấp ba. Cấp phát theo chỉ mục cũng gặp các vấn đề về hiệu suất giống như cấp phát móc nối. Mặc dù các khối chỉ mục có thể được lưu tạm trong bộ nhớ, nhưng các khối dữ liệu có thể nằm rải rác trên ổ đĩa cứng.

Hiệu suất các phương pháp:

Chúng ta so sánh các phương thức trình bày trên theo hai tiêu chí là hiệu quả lưu trữ và thời gian truy cập khối dữ liệu. Việc đầu tiên là xác định mục tiêu sử dụng của hệ thống. Hệ thống mà đa phần các truy cập tuần tự sẽ sử dụng phương thức cài đặt khác với hệ thống thường xuyên phải phục vụ các truy cập ngẫu nhiên. Với bất kỳ kiểu truy cập nào, trong phương pháp cấp phát liên tục chỉ cần duy nhất một lần truy cập để đọc một khối dữ liệu trên ổ đĩa. Vì hệ thống dễ dàng lưu giữ địa chỉ khối đầu tiên của file trong bộ nhớ, nên có thể xác định ngay lập tức vị trí của khối thứ i (hoặc khối tiếp theo) trên ổ đĩa để đọc trực tiếp. Với phương thức cấp phát móc nối, hệ thống cũng lưu địa chỉ của khối kế tiếp trong bộ nhớ và có thể đọc khối này trực tiếp. Khi đó, truy cập tuần tự không gặp vấn đề, nhưng truy cập trực tiếp đến khối thứ i có thể cần tới i lần đọc đĩa. Vì thế, cấp phát móc nối không nên sử dụng cho hệ thống cần phục vụ nhiều yêu cầu truy cập trực tiếp.

Một số hệ thống hỗ trợ truy cập file trực tiếp bằng cách sử dụng phương thức cấp phát liên tục và hỗ trợ truy cập file tuần tự bằng phương thức cấp phát móc nối. Khi đó, phải khai báo kiểu truy cập ngay khi tạo file. Nếu chế độ truy cập tuần tự, các khối dữ liệu của file được móc nối với nhau và điều này không tối ưu khi truy cập trực tiếp. Các khối dữ liệu của file hỗ trợ truy cập trực tiếp sẽ nằm liền kề nhau, có thể phục vụ cả truy cập tuần tự lẫn truy cập trực tiếp, nhưng kích thước file phải được khai báo khi tạo file. Lúc này, HĐH phải có cấu trúc dữ liệu và thuật toán phù hợp để hỗ trợ cả hai phương thức cấp phát.

Phương pháp cấp phát chỉ mục phức tạp hơn. Nếu khối chỉ mục đã nằm trong bộ nhớ, hệ thống có thể truy cập trực tiếp tới các khối dữ liệu. Tuy nhiên, điều này lại gây lãng phí bộ nhớ. Nếu thiếu không gian nhớ, đầu tiên HĐH phải đọc khối chỉ mục, kế đó mới đến khối dữ liệu mong muốn. Đối với chỉ mục hai mức, có thể cần tới hai lần đọc khối chỉ mục. Nếu kích thước file quá lớn, để truy cập khối nằm ở cuối file, có thể phải đọc tất cả các khối chỉ mục ở nhiều mức khác nhau. Như thế, hiệu suất của cấp phát theo chỉ mục phụ thuộc vào cấu trúc chỉ mục, kích thước file và vị trí khối muốn đọc.

cuu duong than cong . com

Một vài hệ thống hỗ trợ cả phương thức cấp phát liên tục lẫn cấp phát chỉ mục, bằng cách sử dụng cấp phát liên tục cho file nhỏ (kích thước file từ 3 đến 4 khối) và tự động chuyển đổi sang cấp phát chỉ mục nếu kích thước file lớn. Nhiều phương pháp cải tiến khác có thể đã và đang được áp dụng. Do chênh lệch cực lớn giữa tốc độ CPU và tốc độ ổ đĩa, nên có thể đưa vào hệ thống hàng nghìn chỉ thị để tiết kiệm một vài bước dịch chuyển đầu đọc/ghi.

11.2.4. Quản lý không gian trống

Phần này trình bày phương thức cài đặt danh sách các khối đĩa "tự do", là các khối chưa cấp phát cho bất kỳ file hay thư mục nào. Khi tạo mới file, HĐH lấy từ danh sách đủ số khối trống cần thiết để cấp phát cho file. Các khối trống sau đó bị loại bỏ khỏi danh sách không gian trống. Khi xóa file, các khối chứa dữ liệu của file sẽ được đưa vào danh sách không gian trống.

cuu duong than cong . com

- **Vector bit:** Trong kỹ thuật bản đồ (hay vector) bit, mỗi khối (sector trên ổ đĩa) ứng với một bit. Bit nhận giá trị 1 nếu khối tự do và nhận giá trị 0 nếu ngược lại. Ví dụ, nếu các khối 2, 3, 4, 5, 8, 9, 10, 11, 12, 13, 17, 18, 25, 26, 27 trên ổ đĩa chưa được cấp phát thì bản đồ bit sẽ

là: 00111100111111000110000001110000. Phương pháp này đơn giản và HĐH dễ dàng tìm được n khối trống cần thiết trên ổ đĩa. HĐH Macintosh của Apple sử dụng phương pháp vector bit để quản lý không gian trống. Để tìm khối "tự do" đầu tiên, HĐH tuần tự kiểm tra các bit trong bản đồ. Hiệu suất hệ thống không cao trừ khi toàn bộ bản đồ bit được lưu trong bộ nhớ chính (và thường xuyên được ghi vào ổ đĩa). Điều này có thể áp dụng cho ổ đĩa dung lượng bé, nhưng không thích hợp với các ổ đĩa có dung lượng lớn. Ổ đĩa 1,3GB và khối 512 byte sẽ cần tới một bản đồ bit kích thước 310KB.

- **Danh sách móc nối:** HĐH có thể móc nối tất cả các khối đĩa trống với nhau, và lưu giữ con trỏ đến khối trống đầu tiên ở một vị trí đặc biệt trên ổ đĩa. Để tăng hiệu suất, con trỏ này cũng được lưu trong bộ nhớ. Khối trống đầu tiên chứa con trỏ đến khối trống thứ 2, và cứ tiếp tục như vậy. Phương pháp này không hiệu quả, vì nếu muốn duyệt danh sách, hệ thống phải đọc từng khối, và điều này mất nhiều thao tác đọc/ghi. Tuy nhiên, hiếm khi phải duyệt qua toàn bộ danh sách các khối. Thông thường, HĐH chỉ cần một khối trống để cấp phát cho file nào đó, vì thế khối đầu tiên trong danh sách sẽ được sử dụng.
- **Nhóm các khối trống:** Có thể cải tiến phương pháp trên bằng cách lưu địa chỉ n khối tự do trong khối tự do đầu tiên. n – 1 khối đầu tiên sẽ thực sự là các khối tự do. Khối cuối cùng chứa địa chỉ n khối tự do tiếp theo, và cứ tiếp tục như vậy. Khi đó, có thể nhanh chóng cấp phát một số lượng lớn các khối tự do.
- **Đếm số khối trống:** Thường HĐH cấp phát hoặc thu hồi một vài khối liền kề cùng lúc. HĐH có thể sử dụng đặc điểm này bằng cách không lưu giữ danh sách toàn bộ địa chỉ n khối tự do, mà lưu giữ địa chỉ khối tự do đầu tiên và số lượng n khối tự do ngay sau khối đầu tiên. Mục trong danh sách không gian trống có hai thành phần là địa chỉ một khối tự do trên ổ đĩa và số lượng các khối tự do liền kề.

11.3. HỆ THỐNG THƯ MỤC

11.3.1. Thư mục

Hệ thống phải quản lý toàn bộ các file trên thiết bị lưu trữ. Đầu tiên, hệ thống file được chia thành các phân vùng (partition) là cấu trúc bậc thấp để

lưu trữ file và thư mục. Ổ đĩa cũng có thể được chia ra thành nhiều phân vùng, tạo nên các khu vực tách biệt, mỗi phân vùng có thể xem như một thiết bị lưu trữ riêng biệt. Một số hệ thống lại cho phép nhóm nhiều ổ đĩa thực vào một phân vùng duy nhất. Người sử dụng chỉ quan tâm đến cấu trúc logic của thư mục và file, mà không cần quan tâm đến vấn đề cấp phát không gian vật lý cho file. Trong cả hai trường hợp, phân vùng đều có thể xem như một ổ đĩa ảo.

Bước thứ hai, phải ghi lại thông tin về các file (như tên, địa chỉ, kích thước và kiểu) vào trong một cấu trúc dữ liệu đặc biệt gọi là Thư mục. Thư mục có thể coi là danh sách các bản ghi (hay các mục), mỗi mục ứng với một file cụ thể. Có thể cài đặt thư mục theo nhiều cách khác nhau. Hệ thống cần có khả năng tạo mới, xóa, tìm kiếm mục dựa trên thuộc tính file, liệt kê danh sách tất cả các mục trong thư mục, duyệt trên toàn bộ hệ thống thư mục.

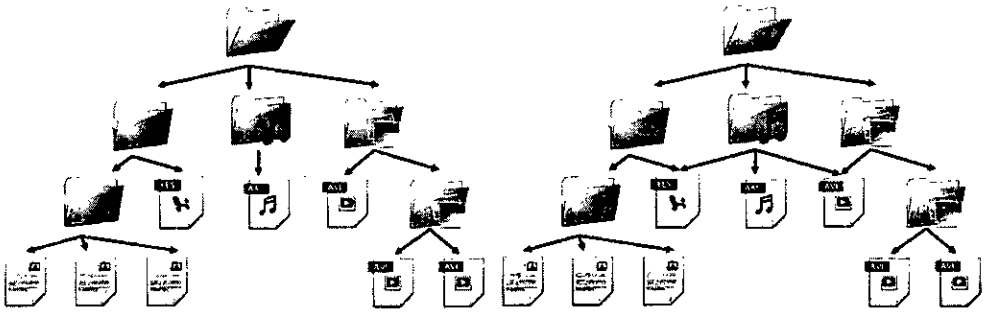
☞ ***Thư mục có cấu trúc kiểu cây***

Các hệ thống máy tính đầu tiên chưa có khái niệm thư mục, hoặc mỗi người dùng có một thư mục riêng lưu tất cả các file của mình. Các hệ thống sau này mới có thư mục cấu trúc dạng cây có độ cao tuỳ ý (Hình 11.6a). Điều này cho phép người sử dụng tạo ra các thư mục con của riêng mình và tổ chức, sắp xếp file một cách hợp lý. Trong HĐH MS-DOS, hệ thống thư mục có cấu trúc dạng cây - là dạng cấu trúc được phổ biến nhất. Cây có một thư mục gốc. Mỗi file có một đường dẫn duy nhất đến, đường dẫn này bắt đầu từ thư mục gốc, đi qua tất cả các thư mục con rồi mới đến file.

Thư mục chứa file và thư mục con. Nhiều HĐH coi thư mục cũng chỉ là một kiểu file đặc biệt, được xử lý theo cách đặc biệt. Cấu trúc nội tại của thư mục nói chung tương tự nhau. Hệ thống phải có các lời gọi để tạo mới hay xóa thư mục.

Tại thời điểm bất kỳ, người sử dụng có thư mục hiện thời. Khi tham chiếu tới file nào đó, hệ thống trước tiên sẽ tìm kiếm trên thư mục hiện thời. Nếu không tìm thấy thì người sử dụng phải chỉ ra đường dẫn đầy đủ hoặc chuyển thư mục hiện thời sang thư mục chứa file cần quan tâm. Để giúp người dùng thay đổi thư mục hiện thời, hệ thống có lời gọi hệ thống với tham số là tên thư mục sẽ là thư mục hiện thời mới. Thư mục hiện thời khi

người dùng lần đầu đăng nhập vào hệ thống được HĐH cấu hình cố định từ trước.



(a) Cấu trúc hình cây (b) Cấu trúc đồ thị tổng quát

Hình 11.6. Cấu trúc thư mục

Có hai loại đường dẫn là đường dẫn tuyệt đối và đường dẫn tương đối. Đường dẫn tuyệt đối bắt đầu từ thư mục gốc và sau đó là tuyến đường qua các thư mục trung gian dẫn đến file. Đường dẫn tương đối xác định một tuyến đường bắt đầu từ thư mục hiện thời.

Khả năng tạo ra thư mục con cho phép người sử dụng tổ chức hệ thống file của mình một cách linh hoạt. Có thể đặt các file có liên quan vào cùng một thư mục. Một vấn đề ở đây là khi xóa thư mục: Nếu thư mục rỗng, có thể dễ dàng xóa mục tương ứng trong cấu trúc thư mục mẹ. Tuy nhiên, có thể thư mục bị xóa không rỗng (thậm chí có thể chứa cả thư mục con). Một số HĐH (MS-DOS) chỉ cho phép xóa thư mục rỗng. Người sử dụng phải xóa tất cả các file trong thư mục định xóa. Phương pháp này có thể khiến người sử dụng tốn nhiều công sức. Phương pháp thứ hai, ví dụ lệnh **rm** trong UNIX có tùy chọn cho phép người sử dụng có thể xóa tất cả các file và thư mục con bên trong thư mục bị xóa. Chú ý rằng, có thể dễ dàng cài đặt cả hai phương pháp trên. Phương pháp sau tuy tiện lợi hơn, nhưng nhược điểm là có thể xóa toàn bộ cấu trúc thư mục chỉ với một lệnh. Nếu lệnh đó được đưa ra do nhầm lẫn, một khối lượng lớn file và thư mục con có thể bị mất.

Với hệ thống thư mục có cấu trúc cây, người sử dụng không những có thể truy cập đến file của mình mà còn có thể truy cập tới file của người khác. Ví dụ, người sử dụng B có thể truy cập các file của người sử dụng A thông qua đường dẫn tuyệt đối đến file của A. Chú ý, đường dẫn trong thư mục cấu trúc cây có thể dài. Để cho phép người dùng sử dụng các chương

trình mà không phải nhớ đường dẫn dài, HĐH Macintosh tiến hành một cách tự động việc tìm các chương trình khả thi, bằng cách sử dụng một file đặc biệt gọi là "Desktop file", chứa tên và vị trí tất cả các chương trình khả thi mà HĐH biết. Khi đưa thêm vào hệ thống một đĩa cứng, đĩa mềm, hay đĩa CD, HĐH xem xét toàn bộ cấu trúc thư mục, tìm kiếm các chương trình khả thi trên thiết bị và ghi lại các thông tin thích hợp. Cơ chế này hỗ trợ khả năng thực thi bằng cách kích đúp hai lần chuột được trình bày ở trên. Thao tác kích đúp trên file khiến hệ thống đọc thuộc tính tạo file (thuộc tính creator) và hệ thống tìm kiếm trên "Desktop file" chương trình tương ứng. Nếu tìm thấy, chương trình khả thi phù hợp được tải vào bộ nhớ để thực thi với file được kích đúp làm tham số vào.

☞ ***Thư mục đồ thị phi chu trình***

Giả sử A và B cùng làm chung trong một dự án, các file liên quan trong dự án được lưu trong thư mục project. Cả hai lập trình viên đều muốn thư mục project nằm trong thư mục riêng của mình (là a và b tương ứng với người A và B). Project phải là thư mục chia sẻ. Mặc dù thư mục hay file chia sẻ sẽ tồn tại tại nhiều vị trí khác nhau trong hệ thống thư mục, nhưng nó lại không phải là hai bản sao khác nhau. Nếu mỗi người có bản sao riêng của file, thì khi người này thay đổi file, người kia sẽ không nhìn thấy sự thay đổi. Với file chia sẻ, trên thực tế chỉ có duy nhất một file, do đó mọi thay đổi đều được tất cả mọi người nhìn thấy ngay lập tức. Cấu trúc cây ngăn cấm việc chia sẻ file hay thư mục con. Cây thư mục theo đồ thị có chu trình cho phép chia sẻ thư mục và file (Hình 11.6b). Cùng một file hay thư mục con có thể nằm trong hai thư mục khác nhau. Như vậy, đồ thị có chu trình là sự tổng quát hóa thư mục cấu trúc cây hết sức tự nhiên.

Trong trường hợp có nhiều người làm việc theo nhóm, tất cả các file được chia sẻ sẽ được đặt chung trong một thư mục. Thư mục của mỗi người dùng đều chứa thư mục chia sẻ này như một thư mục con. Thậm chí khi chỉ có một người sử dụng, người đó cũng có nhu cầu đặt một số file nào đó trong nhiều thư mục con khác nhau. Ví dụ, chương trình mã nguồn được viết cho dự án cụ thể sẽ nằm trong thư mục chứa các chương trình mã nguồn và cũng nằm trong thư mục của dự án đó.

Có nhiều phương thức cài đặt việc chia sẻ. Cách thông dụng nhất (được UNIX áp dụng) là sử dụng file có kiểu liên kết. Thuộc tính của kiểu liên kết

là đường dẫn đầy đủ, hoặc đường dẫn tương đối tới đối tượng khác. Khi tham chiếu tới file nào đó, HĐH thực hiện tìm kiếm trên thư mục. Nếu file thuộc kiểu liên kết, thì HĐH tiếp tục xác định tên (đường dẫn) tới file (hay thư mục) thực.

Mặc dù linh hoạt hơn cấu trúc cây, nhưng trên thực tế, cấu trúc thư mục đồ thị có chu trình phức tạp hơn nhiều. Thứ nhất, file có thể có nhiều đường dẫn tuyệt đối. Do đó, các tên file khác nhau có thể cùng trỏ đến một file. Trường hợp này cũng tương tự vấn đề bí danh trong ngôn ngữ lập trình. Nếu chúng ta thực hiện việc duyệt toàn bộ hệ thống file (để tìm một file, để xác định thông tin về file, hay sao chép tất cả file vào thiết bị lưu trữ dự phòng), vấn đề này trở nên quan trọng vì hệ thống không muốn duyệt nhiều lần qua cùng một đối tượng chia sẻ.

Xóa file cũng gặp vấn đề khi HĐH thu hồi không gian ổ đĩa đã cấp phát cho file (để có thể tái sử dụng không gian này). Nếu thu hồi ngay khi xóa file, thì các con trỏ trỏ đến file này trở nên vô nghĩa. Thậm chí, trong trường hợp các file có kiểu "liên kết" trỏ tới địa chỉ thực sự trên ổ đĩa cứng, mà không gian ổ đĩa này đã được cấp phát cho file khác, thì file liên kết có thể trỏ vào file mà người sử dụng không mong muốn.

Xử lý tình huống này trong hệ thống sử dụng kiểu file liên kết không khó, vì xóa một liên kết không ảnh hưởng gì đến file gốc, chỉ duy nhất liên kết bị loại bỏ. Tuy nhiên, khi mục ứng với file trong thư mục bị xóa, không gian ổ đĩa của file bị thu hồi, khi đó có thể để lại các "con trỏ" không trỏ tới file nào cả. Hệ thống có thể tìm kiếm các liên kết này để loại bỏ, nhưng trừ phi mỗi file lưu lại danh sách các liên kết trỏ tới, nếu không công việc tìm kiếm này rất tốn kém. Một cách khác là, cứ để các file kiểu liên kết này cho đến khi ai đó định truy xuất tới. Đến khi đó hệ thống mới có thể xác định rằng, file mà liên kết "trỏ tới" không còn tồn tại. Lỗi gây ra tương tự như việc truy cập vào file không có (Trong trường hợp này, người thiết kế hệ thống phải cân nhắc cần làm gì khi một file bị xóa và sau đó tạo mới file có tên giống hệt file vừa bị xóa, thì các liên kết đến file bị xóa vẫn được coi là hợp lệ và trỏ tới file mới). Trong HĐH UNIX, các file kiểu liên kết vẫn được để lại khi xóa file gốc, và người dùng phải tự xác định đấy có phải là file mình muốn dùng không.

Một giải pháp xóa file khác là vẫn giữ file cho đến khi tất cả các tham chiếu bị xóa. Để thực hiện điều này, hệ thống cần có kỹ thuật xác định khi nào tham chiếu cuối cùng đến file bị xóa. Hệ thống có thể thiết lập cho mỗi file một danh sách các tham chiếu. Khi có thêm một liên kết tới file này, HĐH chèn thêm mục mới vào danh sách tham chiếu của file. Khi xóa đi một liên kết, HĐH xóa đi mục tương ứng trong danh sách. File chỉ bị xóa khi danh sách tham chiếu rỗng. Nhược điểm của giải pháp này là danh sách tham chiếu đến file cụ thể nào đó có thể rất lớn. Tuy nhiên, hệ thống không cần lưu giữ toàn bộ danh sách này, mà hệ thống chỉ cần ghi nhớ số lượng các tham chiếu thông qua một bộ đếm. Khi có thêm một liên kết, bộ đếm tăng lên một và khi một liên kết bị xóa, giá trị bộ đếm giảm đi một. Khi bộ đếm bằng 0, file có thể được xóa hẳn, vì không còn tham chiếu nào "trỏ" đến file. HĐH UNIX sử dụng phương pháp này (mỗi file có thuộc tính xác định số lượng các "con trỏ" đến file). Tuy nhiên, một vấn đề nảy sinh là có thể tạo ra một chu trình trong hệ thống thư mục.

☞ ***Thư mục đồ thị tổng quát***

Vấn đề phát sinh trong cấu trúc đồ thị có chu trình là khả năng xuất hiện chu trình. Việc chèn thêm file hay thư mục con vào bên trong thư mục cấu trúc cây vẫn duy trì hình trạng cây của cấu trúc thư mục. Tuy nhiên, bổ sung thêm file hay thư mục kiểu "liên kết" trỏ tới một thư mục đã có có thể phá vỡ cấu trúc cây, hình thành cấu trúc đồ thị tổng quát.

Lợi ích chính của đồ thị phi chu trình là thuật toán duyệt đồ thị để xác định có bao nhiêu tham chiếu tới file tương đối đơn giản. Để hiệu suất không suy giảm, cần tránh việc duyệt nhiều lần qua cùng một đối tượng dùng chung. Nếu đã tìm kiếm trên một thư mục dùng chung, mà không thấy file mong muốn, hệ thống không quay lại thư mục này.

Nếu cho phép tồn tại chu trình, vì tính chính xác cũng như hiệu suất, hệ thống cần tránh việc tìm kiếm nhiều lần trên một thư mục vì có thể duyệt trong những vòng lặp vô hạn. Một giải pháp cho vấn đề này là hạn chế số lượng các thư mục được truy xuất trong một lần tìm kiếm.

Một vấn đề tương tự là khi nào xác định xóa file. Trong cấu trúc thư mục có chu trình, nếu bộ đếm nhận giá trị 0, nghĩa là không có tham chiếu nào đến file hay thư mục, thì file có thể được xóa. Tuy nhiên, hoàn toàn có

khả năng khi chu trình xuất hiện thì bộ đếm có thể không bao giờ nhận giá trị 0, kể cả khi file không còn bị tham chiếu nữa. Điều dị thường này xảy ra do file có thể tự tham chiếu trong cấu trúc thư mục. Trong trường hợp này, hệ thống phải sử dụng bộ thu gom tài nguyên để xác định khi nào tham chiếu cuối cùng bị xóa để thu hồi không gian ổ đĩa. Bộ thu gom này duyệt toàn bộ hệ thống file, đánh dấu tất cả những mục có thể truy cập được. Vòng duyệt kế tiếp chuyển tất cả các mục không được đánh dấu vào danh sách không gian trống (một thủ tục đánh dấu tương tự có thể được áp dụng để bảo đảm rằng, quá trình duyệt hay tìm kiếm sẽ chỉ duyệt qua mọi mục trong hệ thống file đúng một lần). Bộ thu gom tài nguyên cực kỳ hữu ích nếu hệ thống file nằm trên ổ đĩa cứng. Tuy nhiên, do chiếm nhiều thời gian thực hiện nên hiếm khi được sử dụng. Bộ thu gom tài nguyên chỉ cần thiết trong trường hợp có khả năng xuất hiện chu trình trong đồ thị. Do đó, vấn đề phức tạp trong cấu trúc đồ thị có chu trình là tránh sự xuất hiện của chu trình khi bổ sung thêm file kiểu liên kết vào hệ thống. Làm thế nào để biết được liên kết mới có tạo nên chu trình hay không? Có một vài thuật toán để phát hiện chu trình trong đồ thị, tuy khối lượng tính toán khá lớn do phải đọc nhiều thông tin trên ổ đĩa cứng. Nói chung, cấu trúc thư mục cây thông dụng hơn cấu trúc đồ thị có chu trình.

11.3.2. Cài đặt thư mục

Thuật toán cấp phát và quản lý thư mục ảnh hưởng lớn đến hiệu suất và độ tin cậy của hệ thống file. Phần này sẽ trình bày và đánh giá một vài giải pháp.

☞ ***Danh sách tuyến tính***

Danh sách tuyến tính là phương thức cài đặt thư mục đơn giản nhất, mỗi mục trong danh sách chứa tên, thuộc tính và con trỏ tới khối dữ liệu của file. Hệ thống có thể là toàn bộ danh sách các mục trong thư mục. Phương thức này cài đặt đơn giản, nhưng hiệu suất sử dụng không cao. Để tạo file mới, đầu tiên HĐH phải tìm kiếm trên thư mục để đảm bảo không có file nào trùng tên với file định tạo mới. Sau đó, HĐH chèn thêm mục mới vào cuối thư mục. Để xóa file, HĐH tìm kiếm rồi xóa mục ứng với file, sau đó thu hồi không gian ổ đĩa chứa dữ liệu của file. Để sử dụng lại các mục trong danh sách, HĐH thực hiện các bước sau: Đầu tiên, đánh dấu mục đó chưa sử

dụng (bằng cách gán cho mục đó một tên đặc biệt, chẳng hạn tên trống, hoặc sử dụng một trường đặc biệt xác định mục đã được sử dụng hay chưa), hoặc HĐH gắn mục này vào danh sách các mục chưa sử dụng của thư mục. Một phương pháp khác là sao chép mục cuối cùng được sử dụng trong danh sách các mục vào vị trí mục vừa bị xóa, điều này làm giảm kích thước của thư mục. Hệ thống có thể cài đặt bằng danh sách móc nối để làm giảm thời gian xóa file. Tuy nhiên, tốc độ tìm kiếm file trên danh sách tuyến tính không cao. Tốc độ truy cập thông tin trong thư mục ảnh hưởng lớn đến người sử dụng. Trên thực tế, nhiều HĐH triển khai phần mềm lưu giữ tạm thời các thông tin về những thư mục được sử dụng thường xuyên nhất. Truy cập nhanh vào bộ nhớ cache tránh được việc đọc thông tin từ ổ đĩa. Một danh sách đã được sắp xếp cho phép tìm kiếm nhị phân và làm giảm thời gian tìm kiếm trung bình. Tuy nhiên, sắp xếp sẽ tăng thời gian tạo mới và xóa file, vì hệ thống có thể phải đọc/ghi cấu trúc thư mục nhiều lần để sắp xếp các mục trong danh sách. Có thể sử dụng cấu trúc cây nhị phân để khắc phục nhược điểm này.

☞ ***Bảng băm***

HĐH có thể sử dụng cấu trúc bảng băm để lưu giữ thông tin trong thư mục. Các mục vẫn nằm trong danh sách tuyến tính, nhưng thư mục có thêm bảng băm. Mỗi hàng trong bảng băm gồm hai trường là giá trị được tính toán từ tên file và con trỏ đến file tương ứng trong danh sách tuyến tính. Bảng băm giúp giảm đáng kể thời gian tìm kiếm trong thư mục. Việc chèn và xóa file tương đối đơn giản. HĐH phải ngăn ngừa tình trạng xung đột, là tình huống khi hai tên file có giá trị băm giống nhau. Khó khăn chính với bảng băm là kích cỡ cố định của bảng băm và sự phụ thuộc vào hàm băm trên kích thước bảng băm. Một phương pháp khác là để mỗi hàng trong bảng băm là danh sách móc nối chứ không phải một giá trị riêng lẻ. Vấn đề xung đột có thể giải quyết bằng cách chèn thêm mục mới trong danh sách móc nối. Tìm kiếm sẽ bị chậm đi do có thể phải tìm kiếm trên danh sách các tên file có trùng giá trị băm. Tuy nhiên, điều này vẫn nhanh hơn tìm kiếm trên toàn bộ thư mục.

11.4. BẢO VỆ FILE CHIA SẺ

Phải đảm bảo thông tin lưu trong hệ thống không bị mất do lỗi phần cứng (tính tin cậy) và không bị truy nhập bất hợp pháp (vấn đề bảo vệ). Hệ

thống đảm bảo tính tin cậy bằng cách định kỳ thực hiện sao lưu dự phòng (sau một khoảng thời gian, người quản trị hệ thống hoặc chương trình tự động sao lưu lại các file trên ổ đĩa vào băng từ). Băng từ chứa một bản sao của hệ thống file và sẽ được sử dụng nếu hệ thống file trên ổ cứng bị phá hủy. Bảo vệ có thể được thực hiện bằng nhiều cách khác nhau, phụ thuộc vào mức độ phức tạp của hệ thống (đơn hay đa người dùng).

11.4.1. Các kiểu truy cập

Cơ chế bảo vệ có thể thực hiện bằng cách không cho phép truy cập file của người khác. Thái cực đối nghịch là cho phép tự do truy xuất mà không cần cơ chế bảo vệ. Cả hai cách tiếp cận trên đều quá cực đoan để áp dụng. Vấn đề ở đây là truy cập có kiểm soát, tức là đặt ra các giới hạn cho từng kiểu truy cập file. Truy cập được phép thực hiện hay bị ngăn cản không cho thực hiện phụ thuộc vào nhiều yếu tố, một trong số đó là kiểu truy cập. Các kiểu truy cập trên file sau đây có thể kiểm soát được:

- **Đọc/ghi** (read/write): Đọc/ghi file.
- **Thi hành** (execute): Tải file vào bộ nhớ để thi hành.
- **Thêm** (append): Viết thông tin mới vào cuối file.
- **Xóa** (delete): Xóa file và giải phóng không gian trên ổ đĩa cứng của file.
- **Liệt kê** (list): Liệt kê tên và các thuộc tính của file.

Những thao tác phức tạp hơn (đổi tên, sao chép, soạn thảo file) có thể được cài đặt bằng một chương trình hệ thống sử dụng những lời gọi hệ thống ở mức thấp, và cơ chế bảo vệ sẽ được cài đặt ở mức thấp. Mỗi cơ chế bảo vệ có ưu, nhược điểm riêng và chỉ phù hợp trong từng hoàn cảnh cụ thể. Một hệ thống máy tính nhỏ có một người sử dụng, có thể không cần cơ chế bảo vệ phức tạp. Vấn đề bảo vệ được trình bày chi tiết trong Chương 12.

11.4.2. Danh sách và nhóm truy cập

Đơn giản nhất là cho phép truy nhập căn cứ theo định danh người dùng. Người dùng khác nhau có thể có những kiểu truy cập tới file hay thư mục khác nhau. Có thể cài đặt bằng cách gắn mỗi file hay thư mục một danh sách truy cập, xác định tên và kiểu truy cập được phép của mỗi người dùng.

Khi người dùng yêu cầu truy cập tới file cụ thể, HĐH kiểm tra danh sách truy cập gắn với file. Nếu người dùng được phép thì hệ thống cho phép truy cập. Ngược lại, người dùng bị từ chối. Nhược điểm của phương pháp này là danh sách truy cập có thể rất lớn. Giả sử, nếu cho phép tất cả mọi người có quyền đọc một file, hệ thống phải ghi định danh toàn bộ người dùng với quyền đọc vào danh sách truy cập của file. Phương pháp này dẫn đến hai kết quả không mong muốn là:

- Việc xây dựng danh sách như vậy rất tốn kém, đặc biệt khi hệ thống không biết trước danh sách người dùng.
- Mục ứng với file trong thư mục không còn có kích thước cố định như trước kia, mà kích thước bây giờ có thể thay đổi được. Việc quản lý không gian ổ đĩa trở nên phức tạp hơn rất nhiều.

Có thể cải tiến bằng cách thu gọn danh sách truy nhập bằng cách phân lớp người dùng sử dụng file thành ba nhóm:

- **Chủ sở hữu** (Owner): Người dùng tạo ra file chính là chủ sở hữu của file.
- **Nhóm** (Group): Tập hợp các người dùng chia sẻ file theo cùng một kiểu truy nhập tương tự nhau, hợp thành một nhóm, hay nhóm làm việc.
- **Những người còn lại** (Universe): Là tất cả các người dùng khác trong hệ thống.

Để đảm bảo tính tin cậy, hệ thống cần kiểm soát chặt chẽ tư cách thành viên trong mỗi nhóm. Trong UNIX, chỉ người quản trị hệ thống mới được quyền tạo nhóm và bổ sung thành viên vào nhóm. Trong VMS, mỗi file có một danh sách truy cập (còn gọi là danh sách kiểm soát truy nhập), liệt kê những người dùng nào có thể truy nhập file. Chủ nhân của file có thể đọc và điều chỉnh danh sách này.

Cơ chế bảo vệ của UNIX có ba trường để thiết lập mức độ bảo vệ cho 3 nhóm. Mỗi trường là 3 bit rwx (r kiểm soát quyền đọc, w - quyền ghi, và x - quyền thực thi), mỗi bit xác định việc cho phép hay cấm truy cập tới file theo kiểu truy cập tương ứng. Trường thứ nhất dành cho chủ nhân của file, trường thứ hai cho nhóm và trường thứ ba cho tất cả người dùng còn lại. Mỗi file cần 9 bit lưu trữ thông tin bảo vệ. Phương pháp này không linh hoạt bằng danh sách truy cập.

11.4.3. Các phương pháp bảo vệ khác

Có thể đặt mật khẩu cho từng file, truy xuất file cũng phải xuất trình mật khẩu giống như khi đăng nhập vào máy tính phải có mật khẩu. Hạn chế của kỹ thuật này là, nếu mỗi file có một mật khẩu thì người sử dụng phải nhớ rất nhiều mật khẩu. Nếu chỉ dùng một mật khẩu chung cho tất cả các file thì khi lộ mật khẩu, tất cả các file đều có thể bị truy xuất. Để giải quyết vấn đề này, một vài HĐH (như TOPS-20) quản lý người dùng bằng cách đặt mật khẩu cho mỗi thư mục con, chứ không phải cho từng file riêng biệt. HĐH VM/CMS cho phép dùng ba mật khẩu ứng với quyền đọc, quyền ghi và quyền truy cập. Nếu một file ứng với một mật khẩu, người sử dụng hoặc có quyền truy xuất toàn bộ file hoặc không có quyền gì cả. Để kiểm soát truy xuất ở mức độ chi tiết hơn, hệ thống cần sử dụng nhiều mật khẩu. Thậm chí các HĐH một người sử dụng như HĐH MS-DOS và Macintosh cũng có cơ chế bảo vệ file đơn giản. Ban đầu thiết kế của các HĐH này không tính đến vấn đề bảo vệ. Tuy nhiên, do mạng máy tính trở nên phổ biến và việc chia sẻ file là tất yếu, nên cơ chế bảo vệ phải được tích hợp vào HĐH. Chú ý, thiết kế chức năng cho một HĐH mới dễ dàng hơn rất nhiều so với việc bổ sung tính năng mới vào một HĐH cũ.

Trong thư mục có nhiều cấp, HĐH không chỉ bảo vệ file mà phải bảo vệ tập hợp file nằm trong thư mục. Do đó, thư mục cũng cần được bảo vệ. Kiểm soát các thao tác truy xuất trên thư mục khác với kiểm soát file. Hệ thống phải kiểm soát việc tạo mới và xóa file trong thư mục, liệt kê nội dung của thư mục. Nếu đường dẫn tới file nằm trong thư mục nào đó thì người dùng phải có quyền truy xuất cả thư mục lẫn file. Trong hệ thống theo cấu trúc đồ thị có chu trình, thì file có thể có nhiều đường dẫn và người dùng có thể có các quyền truy xuất khác nhau tới file phụ thuộc vào sử dụng đường dẫn nào.

cuu duong than cong . com

11.5. TÍNH THỐNG NHẤT CỦA NGỮ NGHĨA

Tính thống nhất về mặt ngữ nghĩa (Consistency Semantic) xác định ngữ nghĩa khi đồng thời nhiều người dùng truy xuất vào file dùng chung. Khi một người dùng thay đổi dữ liệu dùng chung, những thay đổi này ảnh hưởng tới những người dùng khác như thế nào? Đây là tiêu chuẩn để đánh giá hệ

thống file chia sẻ. Ở đây, giả định tiến trình phải mở (và đóng) file trước (và sau khi) sử dụng. Một chuỗi các thao tác đọc/ghi file nằm giữa cặp thao tác đóng/mở file được gọi là một phiên làm việc. Dưới đây trình bày một vài ví dụ về nhất quán ngữ nghĩa.

11.5.1. Ngữ nghĩa UNIX

Tính thống nhất về mặt ngữ nghĩa trong UNIX được định nghĩa như sau: Nếu người dùng viết vào một file đang mở thì tất cả những người khác cũng đang mở file này sẽ "nhìn" thấy ngay lập tức. Nếu trong chế độ dùng chung mà tất cả người dùng có chung vị trí con trỏ hiện thời trong file, thì khi một người dùng dịch chuyển con trỏ sẽ ảnh hưởng tới tất cả những người dùng khác. Ở đây, file có duy nhất một hình ảnh trong hệ thống và được nhiều người sử dụng xen kẽ. Những ngữ nghĩa này thường liên quan đến quá trình cài đặt, trong đó file gắn với một vị trí vật lý duy nhất được độc quyền truy xuất. Tranh chấp sử dụng tài nguyên khiến tiến trình ứng dụng bị phong tỏa.

cuu duong than cong . com

11.5.2. Ngữ nghĩa theo phiên

Hệ thống file Andrew định nghĩa nhất quán ngữ nghĩa như sau: Việc người dùng viết vào một file đang mở không được những người dùng khác đang mở file này thấy ngay lập tức. Khi đóng file, chỉ nhìn thấy được những thay đổi trong file trong những phiên làm việc sau. Theo ngữ nghĩa này, tại một thời điểm, file có thể gắn với một vài hình ảnh (có thể khác nhau) trong bộ nhớ. Do đó, nhiều người dùng được phép thực hiện cả quyền truy xuất đọc và ghi đồng thời lên các hình ảnh của cùng một file mà không bị phong tỏa (do phải đợi). Hầu như không có ràng buộc nào với việc điều phối truy xuất.

cuu duong than cong . com

11.5.3. Ngữ nghĩa chia sẻ file chỉ đọc

Một phương pháp khác là các file chia sẻ không thay đổi được. Khi file được khai báo là chia sẻ thì file không thể bị chỉnh sửa. Một file như vậy có hai đặc tính quan trọng: Tên file không được sử dụng lại và nội dung file không thể sửa đổi. Nội dung file cố định chứ không chứa các thông tin có thể thay đổi được.

11.6. PHỤC HỒI SAU LỖI

Do file và thư mục được lưu trong cả bộ nhớ chính lẫn trên ổ đĩa, HĐH phải đảm bảo để khi có sự cố, dữ liệu không bị mất hoặc thiếu nhất quán.

11.6.1. Kiểm tra tính thống nhất

Một phần thông tin thư mục được lưu trong bộ nhớ chính để tăng tốc độ truy cập. Thông tin thư mục lưu trong bộ nhớ thường "mới hơn" thông tin nằm trên ổ đĩa, vì cập nhật được ghi vào bộ nhớ trước khi ghi ra ổ đĩa.

Khi máy tính bị ngắt nguồn điện đột ngột, bảng file mở (trong bộ nhớ) bị mất và bất kỳ thay đổi nào cũng bị mất theo. Khi đó, hệ thống file trên ổ đĩa có thể rơi vào trạng thái không nhất quán (trạng thái thực sự của một số file không giống như mô tả trong cấu trúc thư mục nằm trên ổ đĩa). Thông thường, hệ thống chạy chương trình đặc biệt trong quá trình khởi động máy tính để kiểm tra và xử lý những chỗ không nhất quán trong các khối ở ổ đĩa.

Bộ phận kiểm tra tính thống quán tiến hành so sánh dữ liệu trong cấu trúc thư mục với các khối dữ liệu trên ổ đĩa và cố gắng chỉnh sửa những điểm không nhất quán gặp phải. Loại vấn đề nào bộ phận kiểm tra có thể tìm thấy và mức độ thành công trong việc khắc phục những vấn đề đó phụ thuộc vào các thuật toán quản lý và cấp phát không gian trống.

11.6.2. Sao lưu và khôi phục

Xác suất hỏng của ổ đĩa khá cao, do vậy cần kỹ thuật đảm bảo dữ liệu không bị mất. Có một số phần mềm hệ thống thực hiện sao lưu dữ liệu từ ổ đĩa sang thiết bị lưu trữ thứ cấp khác (băng từ, đĩa quang). Để khôi phục, chỉ cần sao chép lại dữ liệu từ thiết bị sao lưu. Để giảm thiểu khối lượng sao chép, hệ thống có thể sử dụng thông tin về file ghi trong cấu trúc thư mục. Chẳng hạn, nếu chương trình sao lưu biết được từ sau lần sao lưu cuối cùng, file chưa bị thay đổi, thì file không cần phải sao lưu. Một kế hoạch sao lưu điển hình có thể được thực hiện như sau:

- Ngày thứ 1: Sao lưu tất cả các file nằm trong ổ đĩa (sao lưu toàn bộ) lên thiết bị lưu trữ thứ nhất.

- Ngày thứ 2: Sao lưu tất cả các file đã thay đổi từ ngày thứ 1 (sao lưu phụ trợ) lên thiết bị lưu trữ thứ 2.
- Ngày thứ 3: Sao lưu tất cả các file đã thay đổi từ ngày thứ 2 lên thiết bị lưu trữ thứ 3.
- ...
- Ngày thứ N: Sao lưu tất cả các file đã thay đổi từ ngày thứ N – 1 lên thiết bị lưu trữ thứ N và sau đó quay trở lại ngày thứ 1.

Người ta có thể khôi phục toàn bộ ổ đĩa bằng cách khôi phục từ bản sao lưu đầy đủ của ngày thứ 1 và tiếp tục với các bản sao lưu phụ trợ từ ngày thứ 2 đến ngày thứ N. Ưu điểm của sao lưu theo chu kỳ là có thể khôi phục bất kỳ file nào đã bị xóa một cách tình cờ trong chu kỳ sao lưu, bằng cách khôi phục lại file bị xóa từ bản sao của ngày trước đó.

11.7. NHẬN XÉT

File là kiểu dữ liệu trừu tượng được HĐH định nghĩa và cài đặt. File là dãy các bản ghi logic, mỗi bản ghi logic có thể là một byte, một dòng (độ dài cố định hoặc biến đổi được), hoặc một đơn vị dữ liệu phức tạp hơn. HĐH có thể hỗ trợ một số loại bản ghi đặc biệt và để chương trình ứng dụng hỗ trợ các loại khác. Nhiệm vụ chủ yếu của HĐH là ánh xạ file logic lên các thiết bị lưu trữ vật lý như đĩa từ hoặc băng từ. Do kích thước bản ghi vật lý của thiết bị có thể khác kích thước bản ghi logic, nên có thể phải ghép các bản ghi logic vào các bản ghi vật lý. Nhiệm vụ này có thể được HĐH hoặc chương trình ứng dụng thực hiện.

Mỗi thiết bị lưu trữ đều có bảng danh mục thiết bị, hoặc thư mục gốc liệt kê vị trí các file trên thiết bị. Thư mục liệt kê file cùng các thuộc tính khác như vị trí file trên thiết bị, kích thước file, kiểu file, người sở hữu file, thời gian tạo file, thời gian truy cập cuối cùng. Thư mục có cấu trúc cây cho phép người dùng tạo ra các thư mục con để tổ chức quản lý file. Một hệ thống thư mục có cấu trúc đồ thị, có chu trình cho phép dùng chung file và thư mục con, nhưng phức tạp trong việc tìm kiếm và xóa. Cấu trúc đồ thị tổng quát cho phép chia sẻ file và thư mục một cách mềm dẻo, nhưng lại cần bộ thu gom tài nguyên để khôi phục lại các khoảng đĩa trống không dùng đến. Do file là cơ chế chủ đạo để lưu trữ thông tin trong hệ thống máy tính,

nên bảo vệ file là điều cực quan trọng. Truy xuất đến file có thể được kiểm soát với từng kiểu: đọc/ghi, thực thi, bổ sung, liệt kê thư mục,... Bảo vệ file có thể thực hiện bằng mật khẩu, bằng danh sách truy xuất, hoặc bởi các kỹ thuật phi cấu trúc. Hệ thống file thường được cài đặt theo cơ chế phân tầng. Tầng thấp xử lý các đặc tính vật lý của thiết bị lưu trữ. Tầng cao xử lý theo tên file và các đặc tính logic của file. Tầng giữa thực hiện ánh xạ file logic vào môi trường lưu trữ vật lý.

CÂU HỎI ÔN TẬP

1. Xác định các thuộc tính chính của file.
2. Trình bày ưu điểm của trừu tượng file.
3. Trình bày cách thức cài đặt file ở mức thấp.
4. Trình bày các vấn đề gặp phải khi cài đặt hệ thống thư mục.

cuu duong than cong . com

cuu duong than cong . com

Chương 12
BẢO VỆ VÀ AN NINH

HĐH phải bảo vệ các đối tượng trong hệ thống (file, bộ nhớ, CPU,...), chỉ cho những tiến trình có quyền sử dụng hợp pháp được quyền sử dụng tài nguyên. Bảo vệ đề cập đến cơ chế hệ thống kiểm soát quyền truy nhập của chương trình, tiến trình, hoặc người dùng tới tài nguyên cụ thể. Cơ chế bao gồm các phương tiện để đặc tả các quyền kiểm soát cũng như cách thức HĐH cài đặt. Cần phân biệt khái niệm Bảo vệ (protection) và An ninh (security). An ninh là thước đo sự tin cậy, tính toàn vẹn của hệ thống và dữ liệu. An ninh bao trùm lên bảo vệ. Kỹ thuật bảo vệ chỉ vận hành chính xác nếu người sử dụng hệ thống không phá vỡ quy tắc để truy cập trái phép tài nguyên. Đáng tiếc là điều này hiếm khi đạt được trong thực tế. Khi đó, an ninh hệ thống trở nên hữu ích. Hệ thống được xem là an toàn (có an ninh) nếu mọi tài nguyên được sử dụng và truy cập như mong đợi trong mọi tình huống. Rõ ràng không thể nào đạt được mức an ninh tuyệt đối. Tuy nhiên, cơ chế an ninh phải làm giảm thiểu các lỗ hổng an ninh. Chương này trình bày chi tiết vấn đề bảo vệ và phát triển mô hình bảo vệ trong quá trình cài đặt. Thông tin (bao gồm dữ liệu và chương trình) cũng như tài nguyên vật lý của hệ thống cần được bảo vệ để chống lại những truy cập trái phép, những thay đổi do phá hoại ác ý hay những chỗ không nhất quán do nhiều nguyên nhân vô tình khác.

12.1. CÁC VẤN ĐỀ CƠ BẢN

Khi hệ thống máy tính trở nên phức tạp với nhiều người dùng, nhu cầu bảo vệ tính toàn vẹn của hệ thống xuất hiện. Hệ thống phải sử dụng nhiều kỹ thuật bảo vệ hiện đại để tăng sự tin cậy, ngăn ngừa tác hại do phá hoại. Bảo vệ là vấn đề nội bộ: trong quá trình hoạt động, chương trình phải tuân thủ chính sách sử dụng tài nguyên. Cơ chế bảo vệ làm tăng tính tin cậy bằng cách phát hiện lỗi tiềm ẩn tại giao diện giữa các hệ thống con. Phát hiện sớm có thể tránh được trường hợp lỗi tại hệ thống con này ảnh hưởng xấu đến hệ

thống con khác. Tài nguyên được bảo vệ không bị lạm dụng bởi người dùng chưa kiểm chứng, hoặc không có quyền truy cập. Hệ thống bảo vệ cung cấp phương thức để phân biệt giữa truy cập được phép và truy cập bị cấm.

An ninh không chỉ đòi hỏi phải có hệ thống bảo vệ phù hợp, mà còn phải giám sát tác động từ môi trường bên ngoài. Bảo vệ sẽ trở nên vô ích nếu kẻ tấn công có thể điều khiển hệ thống hoặc dễ dàng xóa bỏ file (được lưu trữ trên ổ đĩa, băng từ,...). An ninh còn thuộc về chính sách quản lý, chứ không chỉ là vấn đề của riêng HĐH. Có thể chia hành động phá hoại an ninh ra làm hai loại là cố ý và vô tình. Ngăn ngừa các phá hoại cố ý dễ hơn so với các phá hoại vô tình. Khái niệm an ninh được nhắc đến trong nhiều tài liệu là sự kết hợp của bộ ba C-I-A: Confidentiality (tính Bí mật), Integrity (tính Toàn vẹn) và Availability (tính Sẵn sàng truy cập). Chúng ta sẽ lấy an ninh trong việc gửi thư để làm rõ hơn về các khái niệm này. Giả sử Alice muốn gửi thư cho Bob và Trudy muốn đọc trộm bức thư. Tính bí mật phát biểu rằng, Trudy không được phép đọc bức thư của Alice. Tính toàn vẹn đảm bảo bức thư mà Bob nhận được không bị Trudy sửa đổi trên đường truyền. Cuối cùng, tính sẵn sàng truy cập nói lên bức thư không bị mất mà sẽ đến tay Bob (Trudy không phá hủy được bức thư).

12.1.1. Chính sách và cơ chế

Hệ thống cần có cơ chế (*mechanism*) thực hiện chính sách (*policy*) sử dụng tài nguyên. Chính sách có thể được thiết lập bằng nhiều cách như thiết kế cố định trong giai đoạn thiết kế hệ thống, người quản trị thiết lập thông qua bộ phận quản lý hệ thống. Ngoài ra, chính bản thân người sử dụng cũng có thể thiết lập chính sách bảo vệ file và chương trình của riêng mình. Cơ chế bảo vệ cần linh hoạt để triển khai được nhiều kiểu chính sách khác nhau. Chính sách sử dụng tài nguyên phụ thuộc vào từng ứng dụng và có thể thay đổi theo thời gian. Chính vì thế, chúng ta không chỉ quan tâm tới cơ chế bảo vệ trong giai đoạn thiết kế HĐH. Người lập trình ứng dụng có thể coi cơ chế bảo vệ là công cụ để đảm bảo tài nguyên và chương trình do mình tạo ra được sử dụng một cách có kiểm soát. Ở đây, tập trung vào cơ chế bảo vệ do HĐH cung cấp, để người thiết kế ở mức ứng dụng có thể sử dụng trong quá trình thiết kế phần mềm, nhằm đảm bảo khả năng bảo vệ riêng cho mình. Cần phân biệt hai khái niệm chính sách và cơ chế. Cơ chế

xác định cách thức triển khai một điều gì đó, còn chính sách quyết định cái gì cần phải được thực hiện. Điểm khác biệt này cực kỳ quan trọng để đảm bảo tính linh hoạt trong hệ thống bảo vệ. Chính sách có thể thay đổi theo hệ thống, hoặc theo thời gian. Thiết kế hệ thống không hợp lý có thể khiến khi thay đổi chính sách phải thay đổi cơ chế thực hiện bên dưới. Chúng ta mong muốn xây dựng một cơ chế tổng quát, để khi thay đổi chính sách chỉ cần chỉnh sửa một vài tham số hệ thống có liên quan.

12.1.2. Cơ chế xác thực

Vấn đề an ninh hàng đầu trong HĐH là *Xác thực người dùng*. Hoạt động của hệ thống bảo vệ phụ thuộc vào khả năng định danh các tiến trình đang chạy. Khả năng này lại phụ thuộc vào việc có xác định được người dùng nào tạo ra tiến trình.

12.1.3. Cơ chế kiểm chứng

Tài nguyên trong hệ thống có thể là phần cứng (bộ vi xử lý, phân đoạn bộ nhớ, máy in, ổ đĩa cứng, băng từ) hay phần mềm (file, chương trình, semaphore) và được HĐH gán cho một định danh duy nhất. Chỉ có thể sử dụng tài nguyên thông qua các thao tác đã được định nghĩa tường minh. Tiến trình chỉ được phép truy cập đến những tài nguyên được phép. Hơn thế nữa, để hạn chế tác hại mà một tiến trình bị lỗi có thể gây ra trong hệ thống, cần áp dụng nguyên lý Biết - Vừa - Đủ (Need - To - Know): tại bất cứ thời điểm nào, tiến trình chỉ được truy cập tới những tài nguyên *thực sự cần thiết* để hoàn thành nhiệm vụ của mình. Ví dụ, khi tiến trình P gọi thủ tục A, thủ tục A chỉ nên được phép truy cập đến các biến cục bộ của A và các tham số hình thức P truyền cho A; A không được phép truy cập tới tất cả các biến của P.

12.1.4. Mật mã

Khi mạng máy tính trở nên phổ biến, nhiều thông tin bí mật được truyền qua những môi trường không tin cậy (thông tin có thể bị nghe trộm hoặc thay đổi). Để giữ bí mật thông tin, có thể sử dụng phương pháp mã hóa như sau:

1. Thông tin (văn bản gốc) được mã hóa, tức là từ dạng ban đầu (bản rõ) chuyển sang dạng mã hóa (bản mã). Ở dạng này, dù bị đọc trộm cũng không có ý nghĩa gì.

2. Bản mã có thể được đặt trong file hoặc truyền qua kênh truyền không được bảo vệ.

3. Người nhận giải mã bản mã thành bản rõ (thông tin gốc ban đầu).

Thậm chí, kể cả khi thông tin mã hóa bị người dùng (hoặc chương trình) không được phép truy cập đọc, thì cũng không ảnh hưởng gì (trừ khi thông tin bị giải mã). Kỹ thuật mã hóa phải được thực hiện để không thể (hoặc cực khó) bẻ mã.

12.1.5. Các vấn đề an ninh

Bảo vệ tuyệt đối hệ thống trước mọi tấn công là điều không tưởng, nhưng có thể đưa ra các biện pháp để phía tấn công phải trả một giá rất lớn. Để đảm bảo an ninh hệ thống, trước tiên phải bảo vệ địa điểm đặt hệ thống máy tính trước các nguy cơ tấn công về mặt vật lý, hoặc kẻ xâm nhập bất hợp pháp. Kế đó phải giám sát nhân viên trong cơ quan, nhằm giảm thiểu nguy cơ tiềm ẩn. Chẳng hạn, nhân viên trao quyền truy cập cho tội phạm do nhận hối lộ. Nếu không thực hiện những điều này, thì dù sử dụng bất kỳ HĐH nào, hệ thống vẫn gặp nguy hại. Tuy nhiên, trong khuôn khổ quyển sách này, chúng ta chỉ xét mức độ an ninh của HĐH.

cuu duong than cong . com

An ninh trong HĐH phụ thuộc rất nhiều vào nền tảng phần cứng bên dưới. Ví dụ, an ninh trong HĐH MS-DOS rất kém, vì nền tảng phần cứng cài đặt HĐH này không có cơ chế bảo vệ bộ nhớ. Hiện nay, phần cứng hiện đại có thể cung cấp nhiều khả năng bảo vệ cho người thiết kế HĐH sử dụng. Bổ sung thêm tính năng khó khăn hơn so với việc thiết kế các tính năng ngay từ trước khi xây dựng hệ thống. Các HĐH hiện đại đều được thiết kế với tính năng an ninh ngay từ đầu và cài đặt ở nhiều cấp độ khác nhau, từ việc sử dụng mật khẩu để đăng nhập hệ thống cho tới việc cô lập các tiến trình đang thực thi đồng thời.

cuu duong than cong . com

Tấn công vào hệ thống có thể do cá nhân hay tổ chức thực hiện, thường là phát tán virus, trojan, worm, backdoor. Nguy hiểm và khó ngăn ngừa nhất là tấn công từ chối dịch vụ (DoS – Denial of Service), trong đó hàng nghìn máy tính cùng gửi gói tin đến máy chủ, khiến máy chủ không còn khả năng phục vụ và phải sụp đổ. Những lỗ hổng bảo mật nghiêm trọng nhất là lỗi tràn bộ đệm (buffer overflows), lỗi kiểm tra tham số đầu vào (incomplete

mediation) và lỗi giữa khoảng thời gian kiểm tra và thực thi (TOCTTOU – Time Of Check To Time Of Use).

Trong khi kẻ tấn công chỉ cần biết một điểm yếu của hệ thống và có nhiều thời gian để hành động, thì người bảo vệ phải khắc phục mọi lỗ hổng an ninh có thể và phải chịu áp lực rất lớn về thời gian và chi phí. Hơn thế nữa, những hệ thống an ninh vững chắc thường khó sử dụng. Bản thân người dùng cũng thích đặt mật khẩu đơn giản mà không nhận thức được đấy là nguy cơ an ninh tiềm tàng. Người phát triển phần mềm cần cân nhắc xem chi phí khắc phục lỗ hổng an ninh mà họ bỏ ra liệu có đem lại lợi nhuận sau này không, vì thông thường cài đặt các tính năng an ninh rất tốn kém. Thông thường, việc khắc phục lỗi được tiến hành các bước: kiểm tra hệ thống, xác định lỗi và sau đó là vá lỗi. Nhưng trở ngại của việc vá lỗi ở chỗ: Thứ nhất, không phải mọi lỗi đều được phát hiện. Thứ hai, áp lực về thời gian có thể khiến bản vá lỗi không được tốt. Thứ ba, bản vá lỗi phải đảm bảo không được ảnh hưởng đến các thành phần đang hoạt động tốt của hệ thống. Và cuối cùng, rất có thể chính bản vá lỗi lại tạo ra lỗi mới. Nói chung, mọi hệ thống đều có thể có lỗi, do sự phức tạp ngày càng tăng của hệ thống.

cuu duong than cong . com

12.2. XÁC THỰC

Cơ chế ***Xác thực người dùng*** dựa trên sự kết hợp của ba yếu tố sau: vật sở hữu của người dùng (khóa hoặc thẻ); kiến thức mà người dùng biết (định danh, mật khẩu) và đặc trưng người dùng (dấu vân tay, võng mạc hay chữ ký). Phần này chỉ trình bày cơ chế thông dụng nhất là mật khẩu.

12.2.1. Mật khẩu

Mật khẩu là cơ chế xác thực thông dụng nhất. Sau khi khai báo tên truy cập, người dùng xuất trình mật khẩu. Nếu mật khẩu đúng, người dùng được xác thực. Mật khẩu có thể bảo vệ hệ thống máy tính trong trường hợp không có cơ chế bảo vệ đầy đủ. Mật khẩu là một trường hợp đặc biệt của khóa hay khả năng (mục 12.4). Ví dụ, mỗi tài nguyên (chẳng hạn file) có một mật khẩu. Người dùng muốn sử dụng tài nguyên phải xuất trình mật khẩu. Nếu mật khẩu hợp lệ, truy cập được phép thực hiện. Các quyền truy cập khác nhau (quyền đọc, quyền thêm và quyền cập nhật file) có thể ứng với mật khẩu khác nhau.

12.2.2. Tính dễ lộ của mật khẩu

Mật khẩu được sử dụng rộng rãi vì tính đơn giản, dễ sử dụng. Tuy nhiên, mật khẩu có thể bị lộ do nhiều nguyên nhân: bị đoán, vô tình để lộ, hoặc trao đổi bất hợp pháp giữa người dùng hợp pháp và người dùng phi pháp.

Có hai cách thức đánh cắp mật khẩu phổ biến. Cách thứ nhất, kẻ xâm nhập (người hoặc chương trình) quen biết với người dùng hoặc nắm được thông tin cá nhân của người đó. Mọi người có xu hướng chọn mật khẩu là tên dễ nhớ (ngày sinh, tên người thân,...). Cách thứ hai, sử dụng phương pháp Dò tìm - vét cạn (brute-force): lần lượt thử mọi khả năng tổ hợp và hoán vị của các ký tự cho tới khi tìm ra mật khẩu hợp lệ. Mật khẩu ngắn sẽ dễ bị lộ bằng phương pháp này.

Mật khẩu có thể lộ do bị nhìn trộm, hoặc do bị "đọc lén" điện tử. Kẻ xâm nhập có thể đứng sau lưng người dùng đang đăng nhập và dễ dàng đoán được mật khẩu bằng cách đọc bàn phím. Người dùng có thể truy cập máy tính đã bị cài phần mềm theo dõi, kẻ xâm nhập có thể "nghe trộm" toàn bộ thông tin truyền qua mạng, bao gồm cả tên truy cập và mật khẩu. Mật khẩu bị lộ có thể do người dùng viết vào nơi nào đó dễ bị đọc trộm...

Mật khẩu có thể do hệ thống sinh ngẫu nhiên hoặc do người dùng lựa chọn. Mật khẩu do hệ thống sinh thường khó nhớ, người dùng có thể phải ghi lại vào đâu đó. Mật khẩu do người dùng lựa chọn thường dễ đoán. Ở một số hệ thống, định kỳ người quản trị kiểm tra mật khẩu, cảnh báo người dùng nếu mật khẩu quá ngắn hoặc dễ đoán. Một số hệ thống bắt buộc người dùng phải thay đổi mật khẩu đều đặn (ví dụ hằng tháng).

Mật khẩu có thể được thay đổi sau mỗi phiên làm việc. Mật khẩu mới được hệ thống hoặc người dùng lựa chọn tại thời điểm kết thúc phiên làm việc, và mật khẩu mới đó sẽ được sử dụng cho phiên làm việc tiếp theo. Nếu bị lộ, mật khẩu chỉ được sử dụng đúng một lần và điều này khiến người dùng hợp pháp không sử dụng được trong phiên làm việc kế tiếp. Do đó, người dùng hợp pháp có thể phát hiện ra lỗ hổng an ninh ngay tại phiên làm việc tiếp theo.

12.2.3. Mã hóa mật khẩu

Rất khó giữ bí mật mật khẩu (hoặc các phần của mật khẩu, hoặc thuật toán tạo mật khẩu). HĐH UNIX mã hóa danh sách mật khẩu để giữ bí mật,

mỗi người dùng có mật khẩu riêng. Hệ thống mã hóa mật khẩu bằng hàm một chiều. Hàm một chiều có tính chất tính hàm ngược rất khó (nghĩa là biết giá trị x dễ dàng tính được f(x), nhưng từ f(x) rất khó xác định được x). Hệ thống chỉ lưu lại mật khẩu đã mã hóa. Khi người dùng đăng nhập, mật khẩu đưa ra sẽ được mã hóa rồi so sánh với mật khẩu đã mã hóa lưu trên máy. Cho dù mật khẩu mã hóa bị lộ, cũng không thể xác định được mật khẩu thật. Do đó, việc giữ bí mật file mật khẩu không cần thiết.

Mặc dù đã mã hóa mật khẩu, nhưng bất kỳ ai có bản sao file chứa mật khẩu mã hóa cũng có thể dùng chương trình giải mã để dò tìm. Ví dụ, chương trình mã hóa các từ trong từ điển rồi so sánh với mật khẩu (đã mã hóa). Nếu người dùng chọn mật khẩu là từ trong từ điển, mật khẩu sẽ bị lộ. Với máy tính mạnh, quá trình so sánh như vậy không tốn thời gian. Vì thuật toán mã hóa của UNIX mở, nên hacker tạo ra danh sách mật khẩu (mật khẩu đã mã hóa để có thể nhanh chóng dò ra mật khẩu thật). Vì thế, các phiên bản mới của UNIX ẩn đi đường dẫn file mật khẩu. Người dùng nên sử dụng mật khẩu tổ hợp, ví dụ, hai từ trên từ điển cách nhau bằng dấu chấm câu. Mật khẩu như vậy khó bị lộ hơn, vì phải thử một lượng cực lớn tổ hợp các từ trong từ điển. Để ngăn ngừa việc dò mật khẩu bằng từ điển, một số hệ thống không cho phép dùng các từ có trong từ điển làm mật khẩu.

Một điểm yếu nữa của UNIX là nhiều hệ thống UNIX chỉ xét 8 ký tự đầu tiên, do vậy người sử dụng rất khó chọn mật khẩu dài. Bên cạnh đó, một số hệ thống không cho phép người dùng chọn các từ có trong từ điển làm mật khẩu. Lời khuyên ở đây là người sử dụng có thể chọn một câu nào đó dễ nhớ, chẳng hạn "**B**ác **H**ồ **S**ống **M**ãi **T**rong **S**ự **N**ghiệp **C**ủa **C**húng **T**a" và chọn mật khẩu là các chữ cái bắt đầu của mỗi từ, trong trường hợp này mật khẩu là BHSMTSNCCT.

12.2.4. Mật khẩu dùng một lần

Để tránh lộ mật khẩu do bị nhìn trộm, hay do bị bắt trên đường truyền qua mạng, hệ thống có thể sử dụng mật khẩu kép. Khi phiên làm việc mới bắt đầu, hệ thống lựa chọn và hiển thị ngẫu nhiên phần mật khẩu thứ nhất (có thể xem là câu hỏi); người dùng phải đưa ra mật khẩu thứ hai (câu trả lời). Như vậy, người dùng bị hệ thống "thách đố" và phải giải đáp được thách đố. Có thể tổng quát hóa phương pháp này bằng cách chọn thuật toán

làm mật khẩu. Ví dụ, thuật toán có thể là một hàm số nguyên f. Hệ thống đưa ra một số nguyên ngẫu nhiên x, người dùng tính f(x) và gửi trả kết quả cho hệ thống. Hệ thống cũng tính f(x), nếu hai kết quả ăn khớp với nhau, người dùng được chứng thực.

Một phương pháp khác là hệ thống chia sẻ với người dùng một bí mật (**secret**). Bí mật này không bao giờ được truyền qua môi trường không an toàn để có thể bị lộ. Hệ thống sẽ gửi cho người dùng "**Seed**" – là một số hay một dãy ký tự ngẫu nhiên. **Secret** và **Seed** được sử dụng để tính giá trị **f(secret, seed)**. Vì máy tính cũng có hàm **f**, nên cũng tính được giá trị hàm số này. Nếu hai giá trị giống nhau, người dùng đăng nhập thành công. Trong lần truy cập kế tiếp, người dùng sẽ có **seed** mới và các bước trên được lặp lại, tuy nhiên giá trị mật khẩu đã thay đổi.

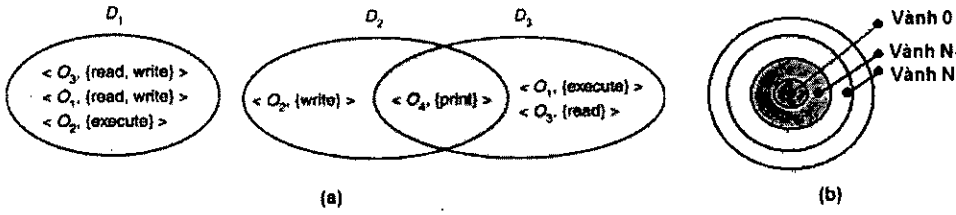
Trong hệ thống sử dụng mật khẩu dùng một lần, mật khẩu thay đổi theo mỗi phiên giao dịch, và cho dù có bắt được mật khẩu thì mật khẩu đó cũng không dùng lại được trong lần sau. Như vậy, mật khẩu dùng một lần là cách thức duy nhất để ngăn cản việc ăn trộm mật khẩu. Có nhiều hệ thống mật khẩu một lần. Các sản phẩm thương mại như SecureID sử dụng phần cứng cài đặt thuật toán mã hóa. Cách thức này cũng được áp dụng trong hệ thống thẻ tín dụng (credit card), hệ thống máy rút tiền tự động (ATM). Giá trị **seed** có thể phụ thuộc vào thời gian sinh ra **seed**. Người dùng sử dụng bàn phím để đánh vào giá trị **secret** (PIN - Personal Identification Number). Một biến thể của mật khẩu sử dụng một lần là sử dụng code book hay one–time pad – liệt kê danh sách các mật khẩu chỉ được sử dụng một lần. Khi đó, các mật khẩu nằm trong danh sách sẽ được lần lượt sử dụng và sau đó bị xóa bỏ ngay lập tức. Hệ thống S/Key áp dụng phương pháp này.

12.3. KIỂM CHỨNG

cuu duong than cong . com

12.3.1. Miền bảo vệ

Miền bảo vệ của tiến trình xác định toàn bộ các đối tượng và những thao tác mà tiến trình có thể thực hiện được trên đối tượng. Khả năng thực thi thao tác trên một đối tượng gọi là quyền truy cập. Có nhiều kiểu quyền truy cập khác nhau: quyền đọc, quyền ghi, quyền thực thi,... Miền bảo vệ là tập hợp các quyền truy cập, mỗi quyền là cặp <Tên đối tượng, Tập hợp các quyền>.



Hình 12.1. Ví dụ miền truy cập

Các miền không nhất thiết phải tách rời mà có thể chồng lên nhau. Trong Hình 12.1a ta thấy, quyền truy cập **<O_4, {print}>** nằm trong cả hai miền D_2, D_3 - nghĩa là tiến trình thực thi trong miền D_2 hoặc D_3 có thể in đối tượng O_4. Chú ý, tiến trình thực thi trong miền D_1 chỉ được đọc hay ghi trên đối tượng O_1. Mặt khác, chỉ tiến trình trong miền D_3 mới có quyền thực thi đối tượng O_1.

Sự liên kết giữa tiến trình và miền có thể mang tính chất tĩnh (nếu tập tài nguyên của tiến trình được ấn định trong suốt thời gian hoạt động), hoặc mang tính chất động. Thiết lập miền bảo vệ động phức tạp hơn miền bảo vệ tĩnh. Nếu sự liên kết được ấn định từ trước và nếu vẫn muốn giữ vững nguyên tắc Biết - Vừa - Đủ thì cần có cơ chế cho phép thay đổi nội dung miền. Xét tiến trình thực thi trong hai giai đoạn, chỉ đọc trong giai đoạn thứ nhất và chỉ ghi trong giai đoạn thứ hai. Nếu miền thiết lập cố định ngay từ đầu, hệ thống phải định nghĩa cho miền cả quyền đọc lẫn quyền ghi. Tuy nhiên, cách này khiến miền có nhiều quyền hơn cần thiết (vì giai đoạn một chỉ cần đọc, trong khi giai đoạn hai chỉ cần ghi). Điều này vi phạm nguyên tắc Biết – Vừa – Đủ. Như vậy, hệ thống phải có khả năng thay đổi nội dung miền, sao cho miền chỉ gồm những quyền truy cập cần thiết tối thiểu. Nếu liên kết giữa tiến trình và miền mang tính chất động, thì cần có cơ chế cho phép tiến trình chuyển từ miền này sang miền khác trong quá trình thực thi, hoặc hệ thống cho phép thay đổi nội dung miền. Nếu không cho thay đổi, hệ thống có thể tạo ra miền mới tương đương với miền cũ (nội dung đã được thay đổi) và chuyển tiến trình sang miền mới. Miền có thể được hiểu theo nhiều cách khác nhau:

- **Người dùng là một miền:** Tập tất cả các đối tượng tiến trình được quyền truy xuất phụ thuộc vào định danh người dùng tạo ra tiến trình. Sự chuyển đổi miền xuất hiện khi thay đổi người dùng, thường

là khi người dùng thoát khỏi hệ thống và người dùng khác đăng nhập hệ thống.

- **Tiến trình là một miền**: Tập tất cả các đối tượng được phép truy cập phụ thuộc vào định danh tiến trình. Chuyển miền diễn ra khi một tiến trình gửi thông điệp đến tiến trình khác và sau đó chờ phúc đáp.
- **Thủ tục là một miền**: Tập các đối tượng được phép truy xuất tương ứng với các biến cục bộ được khai báo bên trong thủ tục. Miền được chuyển khi thủ tục được gọi.

12.3.2. Ví dụ về miền

Thông thường, hệ thống có hai chế độ là giám sát và người dùng. Trong chế độ giám sát, tiến trình có thể thực thi các chỉ thị đặc quyền và nắm được toàn bộ quyền điều khiển máy tính. Trong chế độ người dùng, tiến trình chỉ có thể sử dụng các chỉ thị không đặc quyền. Như vậy, có thể bảo vệ HĐH (thực thi trong chế độ giám sát) trước tiến trình người dùng (thực thi trong chế độ người dùng). Trong HĐH đa chương trình, mô hình hai chế độ không đáp ứng được nhu cầu bảo vệ, vì thế cần có mô hình bảo vệ tốt hơn. Sau đây là mô hình trong HĐH UNIX và MULTICS.

☞ *UNIX*

Trong HĐH UNIX, miền bảo vệ gắn với người sử dụng. Sự chuyển đổi miền bảo vệ tương ứng với sự chuyển đổi tạm thời định danh người dùng. Quá trình chuyển đổi trên hệ thống file như sau: Mỗi file có Định danh sở hữu (*owner identification*) và bit Miền bảo vệ (còn được gọi là bit Định danh hiệu dụng – *setuid bit*). Khi người dùng (với id = A) bắt đầu thực thi file thuộc quyền sở hữu của B (id = B) mà bit Định danh hiệu dụng của file này có giá trị 0 (*off*) thì định danh người dùng *(user-id)* của tiến trình là A. Nếu bit Định danh hiệu dụng có giá trị 1 (*on*) thì định danh người dùng của tiến trình được thiết lập là B. Khi tiến trình kết thúc thì sự chuyển đổi định danh người dùng cũng kết thúc. Kỹ thuật thay đổi miền bảo vệ bằng cách tạm thời thay đổi định danh người dùng được sử dụng rộng rãi, đặc biệt khi cần cung cấp cho người sử dụng các chương trình tiện ích.

Những hệ thống có độ an toàn cao có thể không cho phép thay đổi định danh người dùng. Trong trường hợp này cần tới những kỹ thuật đặc biệt để

người dùng có thể sử dụng các tiện ích hệ thống. Ví dụ sử dụng tiến trình nền *(daemon process)* thực thi ngay trong thời gian khởi động hệ thống và được coi là của một người dùng đặc biệt. Người dùng bình thường chạy chương trình riêng của mình và gửi yêu cầu tới tiến trình nền khi cần sử dụng tiện ích hệ thống. HĐH TOP-20 áp dụng cách thức trên.

Cần chú ý khi viết chương trình đặc quyền. Bởi vì sự bất cẩn dù rất nhỏ cũng có thể làm giảm khả năng bảo vệ của toàn bộ hệ thống.

☞ ***MULTICS***

Trong MULTICS, miền bảo vệ được tổ chức phân cấp thông qua các vành lồng nhau. Mỗi vành ứng với duy nhất một miền bảo vệ (Hình 12.1b). Các vành được đánh số từ 0 đến 7. Giả sử hai vành D_i và D_j là hai miền bảo vệ. Nếu $j < i$, thì D_i là tập con của D_j, nghĩa là tiến trình chạy trong miền D_j có nhiều đặc quyền hơn tiến trình chạy trong miền D_i. Tiến trình chạy trong miền D_0 có nhiều đặc quyền nhất. Nếu chỉ có hai vành, mô hình này tương tự mô hình sử dụng bit chế độ để phân biệt chế độ người dùng (ứng với miền D_0) và chế độ giám sát (ứng với miền D_1).

Không gian địa chỉ trong MULTIC được chia thành các phân đoạn, mỗi phân đoạn gắn với một vành bảo vệ. Bảng mô tả phân đoạn có trường xác định đoạn thuộc vành nào. Bảng mô tả này cũng chứa ba bit truy cập ứng với ba quyền: đọc/ghi, và thực thi. Ở đây chúng ta không quan tâm đến chính sách quyết định quan hệ giữa phân đoạn và vành bảo vệ. Mỗi tiến trình được gắn một số hiệu vành hiện thời (current-ring-number) tương ứng với vành bảo vệ mà tiến trình đó đang thực thi. Khi đang thực thi trong vành i, tiến trình không thể truy cập tới phân đoạn của vành bảo vệ j nếu $j < i$. Tiến trình chỉ có thể truy cập tới phân đoạn của miền bảo vệ k mà $k \geq i$. Kiểu truy cập như vậy hiển nhiên sẽ bị ràng buộc vào bit truy cập của phân đoạn bộ nhớ.

cuu duong than cong . com

Sự chuyển đổi vành bảo vệ trong MULTICS xuất hiện khi tiến trình gọi thủ tục trong vành bảo vệ khác. Tiến trình thực thi trong vành 0 có toàn quyền quyết định với hệ thống. Để có thể kiểm soát được sự chuyển đổi miền bảo vệ, bảng mô tả phân đoạn được bổ sung thêm các trường sau:

- **Khoảng truy cập**: Một cặp số nguyên b_1, b_2 thỏa mãn $b_1 \leq b_2$.
- **Giới hạn**: Số nguyên b_3 sao cho $b_3 > b_2$.
- **Danh sách cổng**: Xác định các cổng mà phân đoạn có thể truy xuất.

Tiến trình thực thi f trong miền có số hiệu vành hiện thời i gọi thủ tục g trong miền nào đó có khoảng truy cập (b_1, b_2) thì lời gọi được phép thực hiện trong trường hợp $b_1 \leq i \leq b_2$. Các trường hợp còn lại sẽ được HĐH xử lý như sau:

- Nếu $i < b_1$, lời gọi được phép thực hiện vì tiến trình chuyển đến vành có ít quyền bảo vệ hơn. Tuy nhiên, nếu có việc truyền tham số từ f (miền bảo vệ cao) đến g (miền bảo vệ thấp) thì tham số phải được sao chép tới đoạn nhớ của miền bảo vệ thấp vì g không thể đọc được vùng nhớ của f.
- Nếu $i > b_2$, lời gọi hợp lệ khi $b_3 \leq i$, khi đó lời gọi sẽ được hướng qua các cổng cho phép (định nghĩa qua trường Danh sách cổng ở trên). Điều này cho phép các tiến trình (có ít đặc quyền) có thể gọi các thủ tục tại các miền bảo vệ cao (có nhiều đặc quyền hơn) nhưng theo cách thức hệ thống có thể kiểm soát được.

Nhược điểm của cấu trúc vành là không đảm bảo nguyên tắc Biết – Vừa – Đủ. Nếu chúng ta muốn một đối tượng chỉ được truy suất trong miền D_j và không được truy suất trong miền D_i thì $j < i$. Như vậy bất kỳ tiến trình nào ở miền j cũng có thể truy suất bất kỳ đối tượng nào trong miền i (vi phạm nguyên tắc Biết – Vừa – Đủ).

Hàng rào bảo vệ càng mạnh thì hệ thống càng khó sử dụng và hiệu suất hệ thống cũng bị suy giảm. Do đó phụ thuộc vào mục đích sử dụng hệ thống mà có thể cài đặt các tính năng bảo vệ khác nhau. Ví dụ có thể sử dụng một hệ thống bảo vệ phức tạp trên máy tính lưu trữ điểm số của sinh viên trong trường Đại học. Nhưng hệ thống bảo vệ phức tạp lại không phù hợp cho máy tính làm nhiệm vụ tính toán số liệu khoa học bởi vì lúc này tốc độ, hiệu suất hoạt động mới là yêu cầu quan trọng nhất. Đó chính là lý do phải tách cơ chế ra khỏi chính sách bảo vệ. Như vậy, tùy theo nhu cầu của mình, người sử dụng hệ thống có thể thiết lập cơ chế bảo vệ đơn giản hay phức tạp. Để tách cơ chế khỏi chính sách bảo vệ, cần có một mô hình bảo vệ tổng quát hơn.

12.3.3. Ma trận truy cập

Trong ma trận truy cập, hàng biểu diễn miền bảo vệ, cột biểu diễn đối tượng cần bảo vệ. Phần tử trong ma trận xác định các quyền truy cập của

miền trên đối tượng tương ứng. Do đối tượng được xác định qua cột, nên có thể bỏ qua tên đối tượng trong quyền truy cập, vì thế access(i, j) xác định tập hợp các thao tác mà tiến trình thực thi trong miền bảo vệ D_i có thể thực hiện trên đối tượng O_j.

domain \ object	F_1	F_2	F_3	printer
D_1	read		read	
D_2				print
D_3		read	execute	
D_4	read write		read write	

(a)

domain \ object	F_1	F_2	F_3	laser printer	D_1	D_2	D_3	D_4
D_1	read		read			switch		
D_2				print			switch	switch
D_3		read	execute					
D_4	read write		read write		switch			

(b)

Hình 12.2. Ma trận quyền truy cập

Xét ma trận truy cập trong Hình 12.2a. Ma trận gồm 4 miền và 4 đối tượng: ba file (F_1, F_2, F_3) và máy in laser. Khi thực thi trong miền D_1, tiến trình có thể đọc file F_1 và F_3. Tiến trình thực thi trong miền D_3 có quyền đọc file F_2, nhưng không có quyền đọc file F_1 hay F_3. Chú ý, chỉ tiến trình thực thi trong miền D_2 mới in được trên máy in laser.

Thông qua ma trận truy cập, hệ thống có thể cài đặt nhiều loại chính sách khác nhau. Cơ chế bảo vệ ở đây gồm hai phần: cài đặt ma trận và đảm bảo các nguyên tắc an ninh của hệ thống. Hệ thống phải đảm bảo tiến trình thực thi trong miền D_i chỉ có thể thực hiện được đúng các thao tác đã liệt kê tại các phần tử ở hàng i. Chính sách bảo vệ xác định nội dung các phần tử của ma trận quyền truy cập. Người dùng có thể quyết định nội dung các phần tử của ma trận truy cập. Khi người dùng tạo mới đối tượng O_j, cột O_j được thêm vào ma trận với các giá trị khởi tạo ban đầu do người dùng thiết lập. Khi cần thiết, người dùng có thể thêm quyền vào các ô ở cột j của ma trận.

Ma trận truy cập là cơ chế thích hợp để tạo lập và kiểm soát mối liên kết giữa tiến trình và miền bảo vệ. Tiến trình chuyển từ miền bảo vệ này sang miền bảo vệ khác thông qua thao tác chuyển (switch) (lúc này miền bảo vệ cũng được xem là đối tượng) như minh họa trên Hình 12.2b. Tương tự, thay đổi nội dung ma trận truy cập tương ứng với việc thực hiện thao tác nào đó trên ma trận (ma trận cũng được coi là đối tượng).

Bây giờ chúng ta xét những thao tác có thể thực hiện được trên các đối tượng đặc biệt (miền bảo vệ và ma trận truy cập) và làm thế nào để các tiến trình có thể thực thi được các thao tác đó. Tiến trình có thể chuyển từ miền bảo vệ D_i sang D_j khi và chỉ khi thao tác switch nằm trong phần tử (i, j) của ma trận quyền truy cập – nghĩa là *switch* $\in$ access*(i, j)*. Trong Hình 12.2b, tiến trình chạy trên miền D_2 có thể chuyển sang miền D_3 và D_4.

Chúng ta định nghĩa thêm các thao tác: Sao chép (**copy**), Sở hữu (**owner**) và Kiểm soát (**control**). Đây là các thao tác thay đổi giá trị các phần tử trong ma trận quyền truy cập - tức là thay đổi quyền cho miền bảo vệ trên từng đối tượng.

domain \ object	F_1	F_2	F_3
D_1	execute		write*
D_2	execute	read*	execute
D_3	execute		

(a)

domain \ object	F_1	F_2	F_3
D_1	execute		write*
D_2	execute	read*	execute
D_3	execute	read	

(b)

Hình 12.3. Chuyển quyền

cuu duong than cong . com

Thao tác được đánh dấu (*) nghĩa là miền bảo vệ tương ứng (hàng trong ma trận) có thể sao chép thao tác đó sang các miền (hàng) khác. Quyền sao chép (*copy)* cho phép sao chép quyền truy cập trong phạm vi một cột (nghĩa là cho cùng một đối tượng). Ví dụ trong Hình 12.3a, tiến trình chạy trong miền D_2 có thể sao chép quyền read gắn với F_2. Khi đó, ma trận Hình 12.3a biến thành ma trận Hình 12.3b sau khi áp dụng quyền sao chép giữa hai miền trên F_2.

Có hai biến thể của phương pháp này:

1. Thao tác chuyển quyền: Quyền được chuyển hẳn từ ô (i, j) sang ô (k, j).

cuu duong than cong . com

2. Việc sao chép quyền có thể bị giới hạn. Khi quyền R^* được sao chép từ ô (i, j) sang ô (k, j) thì miền bảo vệ D_k không thể sao chép quyền R (chú ý không phải R^*) sang miền bảo vệ khác.

Hệ thống có thể chọn chỉ cung cấp một quyền sao chép hoặc cung cấp cả ba quyền sao chép: Sao chép (copy), Chuyển (transfer), Sao chép có Hạn chế (limited copy). Quyền sao chép cho phép tiến trình sao chép một vài

quyền giữa các ô cùng cột. Quyền sở hữu (owner) kiểm soát các thao tác thêm, xóa quyền. Nếu ô (i, j) chứa quyền sở hữu thì tiến trình thực thi trong miền D_i có thể thêm, xóa bất kỳ quyền nào trong cột j. Ví dụ, Hình 12.4a, miền D_1 chứa quyền sở hữu F_1, vì vậy có thể thêm và xóa bất kỳ quyền nào trong cột F_1. Tương tự, miền D_2 chứa quyền sở hữu F_2 và F_3, do vậy có thể thêm hoặc bớt quyền trong các cột này. Ma trận quyền truy cập Hình 12.4a biến đổi thành ma trận quyền truy cập Hình 12.4b.

Quyền Sao chép và Sở hữu cho phép tiến trình thay đổi thông tin trong cột. Quyền Kiểm soát chỉ dành cho đối tượng là miền bảo vệ, được sử dụng để thay đổi dữ liệu trong hàng. Nếu ô (i, j) chứa quyền Kiểm soát thì tiến trình thực thi trong miền D_i có thể xóa bất kỳ quyền truy cập nào trong dòng j. Ví dụ, trong Hình 12.5 chúng ta bổ sung thêm quyền Kiểm soát vào ô truy cập (D_2, D_4) thì tiến trình thực thi trong miền D_2 có thể thay đổi miền D_4.

domain \ object	F_1	F_2	F_3
D_1	owner execute		write
D_2		read* owner	read* owner write*
D_3	execute		

(a)

domain \ object	F_1	F_2	F_3
D_1	owner execute		
D_2		owner read* write*	read* owner write*
D_3		write	write

(b)

Hình 12.4. Quyền sở hữu owner

Mặc dù quyền sao chép và quyền sở hữu có thể giới hạn sự lan truyền các quyền truy cập, nhưng lại không thể ngăn chặn thông tin bị rò rỉ. Không thể đảm bảo thông tin nằm trong đối tượng không thể bị chuyển ra khỏi môi trường thực thi.

domain \ object	F_1	F_2	F_3	laser printer	D_1	D_2	D_3	D_4
D_1	read		read			switch		*
D_2		*		print			switch	switch control
D_3		read	execute					
D_4	write		write		switch			

Hình 12.5. Quyền Kiểm soát - Control

Các thao tác trên đối tượng miền bảo vệ và ma trận quyền truy cập minh họa được khả năng cài đặt và kiểm soát ma trận truy cập đáp ứng yêu cầu bảo vệ động. Đối tượng và miền mới có thể được tạo mới trong mô hình ma trận quyền truy cập.

12.4. CÀI ĐẶT MA TRẬN QUYỀN TRUY CẬP

Phần này trình bày các phương pháp cài đặt ma trận quyền truy cập và so sánh ưu, nhược điểm của chúng.

12.4.1. Bảng toàn cục (Global Table)

Bảng toàn cục là phương thức cài đặt ma trận quyền truy cập đơn giản nhất. Bảng là tập các bộ ba <Miền bảo vệ, Đối tượng, Tập các quyền>. Khi tiến trình thực thi thao tác M trên đối tượng O_j trong miền bảo vệ D_i, HĐH sẽ xác định <D_i, O_j, R_k> và sau đó kiểm tra $M \in R_k$ không. Nếu có, M được phép thực hiện; ngược lại lỗi biệt lệ xuất hiện. Nhược điểm của phương pháp này là kích thước bảng toàn cục khá lớn nên không thể lưu trong bộ nhớ chính mà phải lưu trên ổ đĩa cứng. Một hạn chế khác là không thể ghép nhóm các đối tượng đặc biệt. Ví dụ, nếu tất cả người sử dụng đều có quyền đọc đối tượng A thì tất cả các phần tử của cột ứng với A trong ma trận phải chứa quyền đọc A.

12.4.2. Danh sách Quyền truy cập cho đối tượng

cuu duong than cong . com

Cột trong ma trận có thể coi như danh sách quyền truy cập tới đối tượng. Rõ ràng có thể bỏ qua các ô rỗng. Mỗi đối tượng có một danh sách các cặp <Miền bảo vệ, Tập các quyền>. Danh sách này xác định tất cả các miền và các quyền truy suất của miền đến đối tượng.

Có thể định nghĩa tập các quyền truy suất mặc định. Giả sử có bộ ba <D_i, R_k, O_j >, khi tiến trình trong miền D_i thực thi thao tác M trên đối tượng O_j, hệ thống sẽ kiểm tra xem $M \in R_k$ hay không. Nếu có, M được phép thực hiện; nếu không, hệ thống tiếp tục kiểm tra trong tập các quyền mặc định. Trường hợp tìm thấy, M được thực hiện, ngược lại M bị ngăn cản và xuất hiện lỗi biệt lệ. Để nâng cao hiệu suất, hệ thống có thể kiểm tra tập quyền mặc định trước khi tìm trong danh sách các quyền ứng với đối tượng.

cuu duong than cong . com

12.4.3. Danh sách Khả năng của miền

Hàng của ma trận ứng với các quyền trên một miền bảo vệ. Danh sách khả năng (capability list) của miền định nghĩa nhóm đối tượng cùng các thao tác được thực hiện trên đối tượng đó. Đối tượng được biểu diễn thông qua tên hay địa chỉ vật lý. Để thực hiện thao tác M trên đối tượng O_j, tiến

trình phải xuất trình khả năng (chứng tỏ quyền thực hiện M trên đối tượng O_j). Nếu tiến trình có khả năng, HĐH cho phép thực hiện thao tác M. Các tiến trình thực thi trong miền bảo vệ không được truy suất trực tiếp tới Danh sách khả năng của miền bảo vệ. Danh sách này cũng được HĐH coi là đối tượng cần bảo vệ.

12.4.4. Cơ chế khóa và chìa (Lock - Key)

Khóa và chìa là sự thỏa hiệp giữa hai cơ chế danh sách quyền truy cập và danh sách khả năng. Mỗi đối tượng có một danh sách các chuỗi bit (được xác định duy nhất) - gọi là các khóa (*lock)*. Mỗi miền bảo vệ có một danh sách các chuỗi bit (cũng xác định duy nhất) gọi là các chìa khóa - hay chìa (*key)*. Tiến trình đang thực thi trong miền M chỉ có thể truy cập tới đối tượng O khi và chỉ khi miền M có chìa có thể "mở" được một trong các khóa của đối tượng O. Cũng như danh sách khả năng, danh sách chìa khóa của miền cũng phải được HĐH quản lý. Người sử dụng không được quyền trực tiếp xem và thay đổi danh sách chìa và danh sách khóa.

12.4.5. So sánh các phương pháp

Trong hệ thống cài đặt ma trận bằng Danh sách quyền truy cập, khi tạo đối tượng mới, người sử dụng có thể thiết lập miền bảo vệ nào được quyền thực hiện những thao tác nào trên đối tượng vừa tạo. Tuy nhiên khó lưu trữ tập trung thông tin về quyền truy cập của tất cả các miền và nếu danh sách quyền truy cập lớn thì việc tìm kiếm quyền truy cập đến đối tượng cụ thể trong danh sách tốn nhiều thời gian.

Ưu điểm của Danh sách khả năng là lưu tập trung thông tin về các quyền của một tiến trình. Tiến trình muốn thao tác trên tài nguyên phải "xuất trình" khả năng thực hiện điều này. Nhiệm vụ của hệ thống bảo vệ là xác định xem khả năng có hợp lệ hay không. Nhược điểm là khó hủy bỏ khả năng.

Cơ chế khóa và chìa là sự thỏa hiệp của hai cơ chế trên. Ưu điểm của cơ chế này là tính hiệu quả, linh động – phụ thuộc vào kích thước khóa. Khóa có thể được tự do chuyển giữa các miền khác nhau. Bên cạnh đó các đặc quyền truy cập có thể dễ dàng thu hồi bằng kỹ thuật thay đổi khóa gắn với đối tượng.

Phần lớn hệ thống kết hợp sử dụng Danh sách quyền truy cập và Danh sách khả năng. Trong lần sử dụng đối tượng đầu tiên của tiến trình, HĐH kiểm tra danh sách các quyền. Nếu quyền bị từ chối, HĐH tạo ra lỗi biệt lệ. Ngược lại, một khả năng được tạo ra và "gắn" vào tiến trình, tiến trình có thể thao tác trên đối tượng. Trong lần sử dụng kế tiếp, tiến trình xuất trình khả năng chứng tỏ mình có quyền thao tác trên đối tượng. Khi thông báo không sử dụng đối tượng nữa, hệ thống xóa khả năng khỏi tiến trình. Xét ví dụ Hệ thống file. Mỗi file có danh sách các quyền truy cập. Khi tiến trình mở file, HĐH tìm kiếm file trong cấu trúc thư mục, sau đó xác định quyền truy cập có hợp lệ không, rồi cấp phát bộ đệm. Tất cả các thông tin này được ghi vào bảng file mở của tiến trình. Thao tác mở file trả lại thẻ file - là chỉ số trong bảng file mở. Tất cả các thao tác sau đó trên file được thực hiện thông qua thẻ file. Thông tin lưu trong bảng cho biết vị trí file và bộ đệm tương ứng. Khi đóng file, thông tin lưu trong bảng file sẽ bị xóa. Do HĐH quản lý, người dùng không thể thay đổi được nội dung bảng file mở. HĐH cũng sẽ kiểm tra quyền truy cập trong mỗi lần sử dụng file (chẳng hạn ghi vào file chỉ có quyền đọc).

cuu duong than cong . com

12.4.6. Hủy bỏ quyền truy cập

Trong hệ thống bảo vệ động, đôi khi phải hủy bỏ quyền truy cập trên đối tượng dùng chung giữa nhiều người. Khi đó phải xác định:

- **Hủy bỏ ngay tức thì không:** Việc hủy bỏ xảy ra ngay lập tức hay để trễ một thời gian? Nếu trễ, thời gian trễ là bao nhiêu?
- **Hủy bỏ một phần hay tất cả:** Liệu có cho phép chỉ thu hồi một phần hay phải thu hồi toàn bộ các quyền gắn với một đối tượng?
- **Hủy bỏ tạm thời hay lâu dài:** Liệu quyền truy cập bị hủy bỏ hoàn toàn hay chỉ tạm thời bị hủy bỏ?

cuu duong than cong . com

Với kỹ thuật Danh sách quyền truy cập, việc hủy bỏ khá đơn giản: xóa quyền truy cập cần hủy bỏ khỏi danh sách. Việc hủy bỏ diễn ra ngay lập tức, và có thể hủy bỏ một phần trong thời gian tạm thời. Trong kỹ thuật Danh sách khả năng, vấn đề hủy bỏ phức tạp hơn vì các khả năng phân tán trên toàn bộ hệ thống, HĐH phải tìm kiếm các khả năng trước khi xóa. Sau đây là một vài phương pháp có thể áp dụng để thu hồi khả năng:

- **Thu hồi định kỳ (Reacquisition):** Định kỳ, các khả năng bị xóa khỏi miền bảo vệ. Nếu tiến trình muốn sử dụng một khả năng, có thể khả năng đã bị xóa. Tiến trình phải xin cấp phát lại khả năng. Nếu quyền truy cập đã bị hủy bỏ, tiến trình không thể xin lại khả năng cần thiết.
- **Con trỏ ngược (Back-Pointer):** Đối tượng có danh sách con trỏ lưu trữ các khả năng trên đối tượng. Khi cần hủy bỏ, hệ thống có thể xóa khả năng trong danh sách con trỏ. Phương pháp này được MULTICS sử dụng. Nhược điểm là chi phí cài đặt khá cao.
- **Gián tiếp (Indirection):** Các khả năng không trực tiếp trỏ tới đối tượng. Mỗi khả năng trỏ tới một ô duy nhất trong bảng toàn cục, ô này trỏ đến một đối tượng cụ thể. Muốn hủy bỏ khả năng, chỉ cần tìm trên bảng toàn cục để xóa đi ô liên quan đến khả năng. Sau này khi tiến trình đưa ra thao tác truy xuất, khả năng không còn được tìm thấy trên bảng toàn cục, do vậy truy xuất trở nên không hợp lệ.
- **Khóa (Key):** Khóa là một chuỗi bit duy nhất gắn với mỗi khả năng. Khóa được tạo ra cùng với khả năng nhưng tiến trình nắm giữ khả năng không có quyền thay đổi khóa. Khóa chủ (master key) gắn với đối tượng, có thể được tạo hoặc thay thế bằng thủ tục set-key. Khi tạo ra khả năng, giá trị hiện thời của khóa chủ cũng được gắn với khóa của khả năng. Khi kiểm tra khả năng, khóa của khả năng sẽ được so sánh với khóa chủ. Nếu hai khóa phù hợp nhau, thao tác được phép thực hiện; ngược lại lỗi biệt lệ phát sinh. Việc hủy bỏ khả năng được thực hiện bằng cách thay thế khóa chủ (thông qua thủ tục set-key), khi đó sẽ vô hiệu hóa khả năng trước đó (sử dụng khóa chủ cũ) trên đối tượng.

Phương pháp này không cho phép hủy bỏ có lựa chọn vì mỗi đối tượng chỉ có một khóa chủ duy nhất. Nếu gắn danh sách các khóa cho một đối tượng thì có thể thực hiện việc hủy bỏ có lựa chọn. Cuối cùng chúng ta có thể nhóm tất cả các khóa trong bảng toàn cục các khóa. Một khả năng được coi là hợp lệ nếu khóa của nó phù hợp với một vài khóa trong bảng toàn cục. Có thể hủy bỏ khả năng bằng cách xóa các khóa tương ứng trong bảng khóa toàn cục. Với kỹ thuật này, khóa có thể ứng với nhiều đối tượng và một số

khóa có thể gắn cho một đối tượng, và như vậy việc sử dụng sẽ vô cùng linh hoạt.

Trong cơ chế dựa trên khóa, không nên cho phép tất cả người sử dụng thực hiện được thao tác: định nghĩa khóa, chèn khóa, loại bỏ khóa khỏi danh sách. Chỉ nên cho phép người sở hữu đối tượng mới có quyền tạo khóa cho đối tượng. Vấn đề này thuộc về chính sách hệ thống bảo vệ của HĐH, chứ không liên quan đến cơ chế cài đặt.

12.4.7. Hệ thống bảo vệ dựa trên ngôn ngữ lập trình

Hệ thống bảo vệ thường là bộ phận trong nhân của HĐH, đóng vai trò "nhân viên an ninh" – thực hiện kiểm tra và kiểm chứng các yêu cầu truy suất tài nguyên. Để giảm tổng chi phí phụ trội của quá trình kiểm chứng, hệ thống cần phần cứng hỗ trợ. Bên cạnh đó, người sử dụng phải tùy biến được cơ chế bảo vệ nhằm đáp ứng tốt nhất nhu cầu của mình. Khi HĐH trở nên phức tạp thì mục tiêu bảo vệ cũng thay đổi: phải mở rộng phương thức có sẵn của HĐH sang các phương thức do chính người dùng định nghĩa. Chính sách sử dụng tài nguyên có thể rất khác nhau, phụ thuộc vào từng ứng dụng cụ thể và có thể thay đổi. Do vậy, hệ thống bảo vệ phải là công cụ để người lập trình ứng dụng sử dụng.

Trong các ngôn ngữ lập trình bậc cao, xác định quyền truy cập trên tài nguyên dùng chung được thực hiện trong giai đoạn khai báo tài nguyên. Kiểu lệnh khai báo như vậy được tích hợp vào ngôn ngữ lập trình bằng cách bổ sung thêm cơ chế khai báo kiểu (trong ngôn ngữ C là kiểu prototype cho hàm hoặc dùng kiểu typedef cho kiểu dữ liệu mới). Khi khai báo cơ chế bảo vệ trên kiểu dữ liệu, người thiết kế ứng dụng có thể xác định các yêu cầu bảo vệ cần thiết. Ưu điểm của cách tiếp cận này là:

- Nhu cầu bảo vệ chỉ cần phải khai báo, chứ không phải lập trình tường minh bằng cách gọi các thủ tục trong nhân HĐH.
- Yêu cầu bảo vệ được khai báo độc lập với cơ chế thực hiện bảo vệ.
- Người thiết kế ứng dụng không cần xây dựng các tiện ích thực hiện chính sách bảo vệ của mình.

Để cung cấp tính năng bảo vệ, ngôn ngữ lập trình bậc cao cần sự hỗ trợ từ phần cứng và HĐH bên dưới. Ngôn ngữ có các thủ tục thực hiện các

chính sách bảo vệ. Điều này cho phép lập trình viên đặc tả chính sách bảo vệ của riêng mình mà không quan tâm đến HĐH cài đặt cơ chế bảo vệ như thế nào.

Ngay cả khi hệ thống không cung cấp bộ phận bảo vệ ở mức nhân, vẫn có thể cài đặt cơ chế bảo vệ ngay trong ngôn ngữ lập trình. Nhưng nếu không có sự hỗ trợ ở mức nhân, cơ chế an ninh không hoàn toàn chắc chắn. Chương trình dịch sẽ tách các tham chiếu không vi phạm quy chế bảo vệ và tham chiếu có thể vi phạm. Tính an toàn, an ninh trong trường hợp này dựa trên giả định rằng tất cả những đoạn mã khả thi do chương trình dịch sinh ra không bị sửa đổi trước và trong khi thực thi.

Chúng ta sẽ so sánh ưu, nhược điểm của hai phương pháp trên.

- **An ninh**: Mức độ an ninh do nhân HĐH cung cấp tốt hơn so với việc kiểm tra các đoạn mã trong quá trình dịch chương trình. Nếu hệ thống bảo vệ xây dựng bằng chương trình dịch, mức độ an ninh phụ thuộc vào tính đúng đắn của chương trình dịch; một số cơ chế bên dưới của HĐH (an ninh của hệ thống file chứa chương trình khả thi). Hệ thống áp dụng chế độ bảo vệ có hỗ trợ ở mức nhân cũng vẫn phải phụ thuộc vào điều này, nhưng mức độ lệ thuộc ít hơn nhiều vì nhân nằm trong vùng nhớ vật lý xác định và chỉ được nạp từ một file cố định. Tóm lại, phần cứng hỗ trợ giúp ngăn ngừa được nhiều sự vi phạm cơ chế bảo vệ.
- **Tính linh hoạt**: Mặc dù nhân HĐH có thể cung cấp hàm cơ sở để người sử dụng tùy biến chính sách bảo vệ, nhưng điều này không mềm dẻo. Sử dụng ngôn ngữ lập trình, chính sách bảo vệ có thể được khai báo và cài đặt khi cần thiết. Nếu chưa đủ linh hoạt để cài đặt cơ chế bảo vệ, ngôn ngữ có thể được mở rộng hoặc thay thế bằng ngôn ngữ khác, và việc thay thế ngôn ngữ lập trình bao giờ cũng đơn giản hơn việc sửa đổi nhân HĐH.
- **Tính hiệu quả**: Nếu tính năng bảo vệ được phần cứng hỗ trợ trực tiếp, hệ thống đạt hiệu suất tối ưu. Nếu sử dụng phần mềm hỗ trợ, chúng ta chỉ kiểm soát được các truy cập tĩnh trong thời gian dịch chương trình, nhưng lại có thể tránh được phải sử dụng nhiều lời gọi hệ thống.

Nếu ngôn ngữ lập trình có tính năng bảo vệ, người sử dụng có thể mô tả chính sách phân chia và sử dụng tài nguyên ở mức cao. Trình biên dịch có thể thực hiện bảo vệ kể cả khi không có phần cứng hỗ trợ. Chương trình dịch có thể dịch đặc tả bảo vệ ra các lời gọi trong hệ thống bảo vệ của HĐH.

Tích hợp khả năng bảo vệ vào ngôn ngữ lập trình vẫn còn đang trong thời kỳ phát triển. Có thể bài toán bảo vệ sẽ là vấn đề được quan tâm sâu sắc trong những hệ thống mới với kiến trúc phân tán có yêu cầu khắt khe về bảo vệ dữ liệu. Do đó ta thấy tầm quan trọng trong việc lựa chọn ngôn ngữ thích hợp biểu diễn được các yêu cầu bảo vệ.

12.5. HẬU QUẢ TỪ CHƯƠNG TRÌNH

Trong môi trường mà người này sử dụng chương trình do người khác viết, hoàn toàn có thể xảy ra tình trạng dùng sai mục đích và dẫn tới những hậu quả khó lường. Có hai trường hợp điển hình là Trojan Horse và Trapdoor.

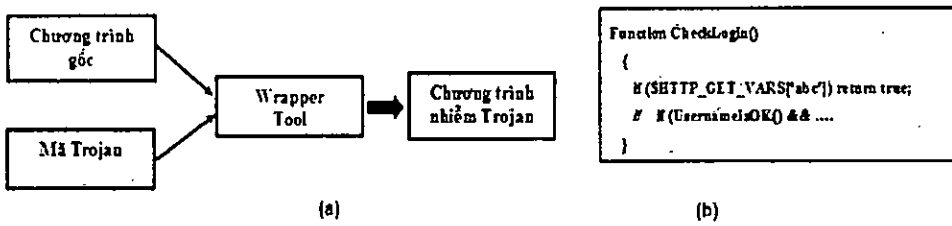
12.5.1. Trojan Horse (Ngựa thành Troy)

Rất nhiều hệ thống cho phép người dùng chạy chương trình do người khác viết. Nếu những chương trình này thực thi trong miền bảo vệ của người chạy chương trình với những quyền truy cập đặc biệt, chương trình có thể lạm dụng quyền. Trojan horse là một trong số những kiểu mã chương trình như vậy. Bên cạnh việc thực hiện các ứng dụng thông thường, chương trình này còn kín đáo thực hiện nhiều thao tác không được phép. Có hai loại Trojan:

- Toàn bộ mã đều là Trojan (loại này dễ phân tích và phát hiện).
- Mã Trojan được cài đặt thêm vào chương trình gốc, bổ sung thêm một vài tính năng (có thể là backdoor hoặc rootkit). Loại này chủ yếu xuất hiện trong hệ thống sử dụng mã nguồn mở vì có thể dễ dàng xâm nhập vào các mã nguồn có sẵn.

Ví dụ trong một login script, khi người dùng cung cấp user id, password để đăng nhập, ngoài việc chuyển những thông tin đó cho tiến trình thực hiện đăng nhập, Trojan lưu những thông tin đó lại để phục vụ những mục đích sau này. Người dùng thấy đăng nhập thành công nên không mấy nghi ngờ rằng mình đã bị mất thông tin cá nhân.

Có hai cách chính tạo ra Trojan. Cách thứ nhất là cài đặt Trojan vào mã nguồn của một chương trình. Cách này có điểm bất lợi là không phải lúc nào cũng có sẵn mã nguồn của một chương trình ưng ý nào đấy để thay đổi, cho nên cách này được sử dụng trên HĐH mã nguồn mở. Cách thứ hai là kẻ tấn công viết chương trình Trojan riêng bên cạnh chương trình gốc, rồi sau đó kết hợp hai chương trình này thành một chương trình duy nhất.



Hình 12.6. Trojan và Trapdoor

Trong Hình 12.6a, Wrapper Tool là công cụ để kết hợp hai chương trình. Một số Wrapper Tool cho phép đoạn mã độc hại được mã hóa trong chương trình kết quả, và chèn thêm thủ tục giải mã vào ngay trong chương trình đó (để tránh bị chương trình diệt virus phát hiện). Hầu hết Wrapper Tool đều hoạt động theo cơ chế: kết hợp hai (hay nhiều) chương trình thành một file trên ổ cứng, tuy nhiên khi chạy thì chúng lại vẫn là hai (hay nhiều) tiến trình thực thi độc lập.

12.5.2. Trapdoor

Trapdoor là "điểm vào" một module nhưng không được công bố. Trapdoor được người phát triển ứng dụng thêm vào trong quá trình phát triển phần mềm để thử nghiệm module phần mềm. Nguyên nhân là do các phần mềm ngày nay thường rất phức tạp, gồm nhiều module kết hợp với nhau. Mỗi module lại do một nhóm người phụ trách, nên trong quá trình phát triển, họ phải tự tạo ra điểm vào cho module của mình để kiểm nghiệm một số chức năng nào đó.

Ví dụ: Có hai module A và B. Module A cần băm một xâu thành xâu có độ dài là 32 bytes. Module B lấy giá trị vừa băm bởi Module A, tìm xem trong hệ thống cơ sở dữ liệu có giá trị đó hay chưa. Vậy khi Module A chưa hoàn thành, thì làm sao kiểm tra được Module B? Các nhà phát triển thêm

Trapdoor để giải quyết những vấn đề kiểu như vậy, tức là bằng cách đọc giá trị từ file nào đó thay vì lấy thông tin từ Module A.

Vì lý do an ninh, hầu hết Trapdoor sẽ bị xóa bỏ khi hoàn thành phần mềm. Có ba nguyên nhân chính dẫn tới việc còn tồn tại Trapdoor:

- Quên không xóa đi.
- Để lại với mục đích bảo trì.
- Cố tình để lại nhằm có những truy cập bất hợp pháp sau này.

Tuy nhiên, những bất cẩn như thế có thể gây ra hậu quả khôn lường. Xét ví dụ sau: File **index.php** trong thư mục **Admin** của website **http://abc.com**. File này cho phép thực hiện công việc quản trị trang web nếu đăng nhập thành công. Hàm **CheckLogin** được minh họa trên Hình 12.6b.

Đoạn **comment** là đoạn code để kiểm tra **username** và **password**. Trong đó có thể dùng một số thư viện mã hóa do module khác đảm nhiệm. Khi module đó chưa hoàn thành thì người viết hàm kiểm tra này sẽ tạm thời chưa sử dụng và tạo một "điểm vào" (dòng mã đầu tiên) nhằm kiểm tra khi đã đăng nhập thì có quản lý trang web được không. Giả sử "điểm vào" này không bị xóa khi sản phẩm chính thức được sử dụng, nếu người dùng vào link **http://abc.com/Admin/index.php** thì trang web sẽ yêu cầu login như bình thường. Nhưng nếu ai đó tình cờ gõ địa chỉ: **http://abc.com/Admin/index.php?abc=1** thì sẽ không phải login mà vẫn được phép quản lý trang web.

Trapdoor thông minh có thể được cài ngay bên trong chương trình biên dịch. Trình biên dịch có thể tạo ra mã đối tượng cùng với Trapdoor. Việc làm này đặc biệt nguy hiểm bởi vì sẽ không tìm ra vấn đề trong mã nguồn chương trình. Vấn đề nằm trong mã nguồn của chương trình dịch. Nói chung Trapdoor luôn là vấn đề phức tạp, bởi vì để tìm ra, phải phân tích mã nguồn của tất cả thành phần tạo nên hệ thống. Các hệ thống phần mềm có thể chứa hàng triệu dòng lệnh, và việc phân tích toàn bộ là điều bất khả thi.

Đa số HĐH cho phép tiến trình trong quá trình thực thi có thể sinh ra tiến trình con. Khi đó tài nguyên hệ thống có thể bị sử dụng sai mục đích. Có hai phương pháp phổ biến để phá hoại hệ thống theo phương thức này là Virus và Worm.

12.5.1. Virus

Virus là đoạn mã có khả năng tự nhân bản, gắn với một chương trình nào đó và thường cần tác động của con người để lây lan. Khả năng quan trọng của virus là tự nhân bản: tự động tạo ra bản sao của chính mình mà không cần con người can thiệp. Khả năng này giúp virus lây lan. Một khi được kích hoạt, virus có thể thực thi một số hoạt động gây tác hại to lớn như: xóa file, làm hỏng file, tìm các thông tin nhạy cảm hay cài backdoor. Có nhiều cơ chế lây lan của virus.

☞ ***Cơ chế bạn đồng hành (Companion Infection Techniques)***

Chương trình nhiễm virus có tên gần giống chương trình người dùng định chạy. Trong HĐH Windows, một cách để tạo bạn đồng hành với file EXE là tạo một file cùng tên nhưng phần mở rộng là .COM. Cơ chế này được phát hiện lần đầu ở virus có tên Globe vào năm 1992. Khi chạy file gốc có phần mở rộng .EXE, người dùng thường chỉ gõ tên file mà không gõ phần mở rộng. Windows sẽ ưu tiên chạy file .COM trước. File .COM thường ẩn đi để người dùng không nhìn thấy. Để "nạn nhân" không nghi ngờ, sau khi chạy file .COM (có virus) thì file .EXE vẫn được thi hành bình thường.

☞ ***Cơ chế viết đè (Overwriting Infection Technique)***

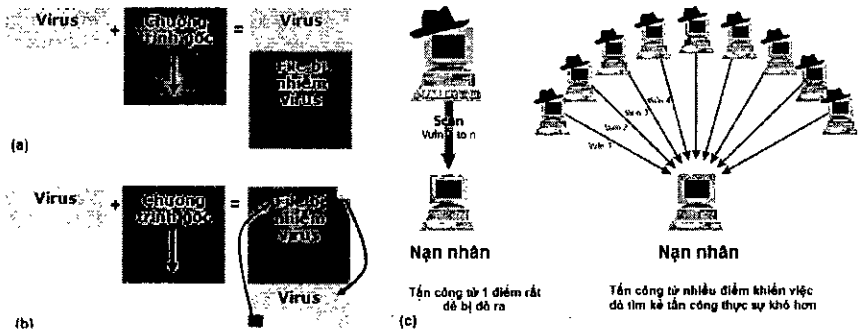
Virus mở file chương trình thực thi ra, thay đổi chương trình nguồn bằng đoạn mã virus rồi lưu lại. Khi thực thi chương trình, dù người dùng có thể nhận thấy mình chạy sai (vì mã đã bị thay đổi) thì cũng đã quá muộn.

☞ ***Cơ chế thêm vào trước (Prepending Infection Technique)***

Virus chèn đoạn mã của mình vào trước phần bắt đầu của chương trình bị nhiễm (Hình 12.7a). Cơ chế này phức tạp hơn cơ chế viết đè vì phải đảm bảo chương trình gốc sau khi bị thay đổi không bị hỏng. Khi chương trình bị nhiễm virus thực thi, HĐH chạy đoạn mã có virus trước vì đoạn mã này nằm đầu tiên.

☞ ***Cơ chế thêm vào sau (Appending Infection Technique)***

Cơ chế này tương tự như cơ chế trước, chỉ khác là mã virus sẽ được thêm vào sau cùng trong chương trình (Hình 12.7b).



Hình 12.7. Virus và Worm

Những cơ chế trên giúp virus có thể lây nhiễm vào những chương trình khả thi. Virus cũng có thể lây nhiễm vào một số file. Virus cũng có thể phối hợp các cơ chế này với nhau để đảm bảo sự tồn tại và phát triển của mình.

12.6.2. Sâu (Worm)

Khác virus, worm là loại chương trình có thể tự nhân bản thông qua môi trường mạng. Worm hoạt động như một chương trình độc lập trong khi virus phải được gắn thêm vào chương trình khác. Chúng ta cũng có thể coi worm là một dạng của virus. Trong thực tế, hai thuật ngữ này luôn có sự nhầm lẫn, nhưng thường thì mã độc hại nào cũng có thể gọi là virus. Ta có thể ví dụ trường hợp của W32/Nimda.A@mm, McAfee gọi là virus còn Symatec gọi là worm. Worm mang lại cho kẻ tấn công những lợi điểm to lớn sau:

☞ ***Có khả năng vượt qua một số lượng lớn hệ thống***

Giả sử kẻ tấn công muốn tấn công 10000 hệ thống trên khắp thế giới. Cứ cho là trong một giờ làm chủ được một hệ thống (là thời gian để vượt qua hệ thống an ninh, cài backdoor, xóa log và hàng loạt các hành động khác) thì nếu tuần tự tấn công, sẽ mất hơn một năm. Tuy nhiên, nếu dùng worm thì chỉ mất vài giờ (hoặc ít hơn) để làm chủ hoàn toàn. Bởi thế, worm làm tăng khả năng xâm nhập hệ thống.

☞ ***Khó lộ dấu vết***

Giả sử kẻ tấn công tìm kiếm lỗ hổng cho việc xâm nhập bằng cách chạy chương trình thực hiện gửi các gói tin tới máy tính "nạn nhân". Nếu thực hiện việc này tại một máy, kẻ tấn công phải gửi rất nhiều gói tin tới máy nạn

nhân. Do đó, máy nạn nhân có thể dễ dàng nhận thấy tất cả các gói tin thăm dò đến từ máy tính kẻ tấn công. Tuy nhiên, với 10000 máy tính đã nhiễm worm, kẻ tấn công có thể chia đều việc gửi gói tin thăm dò cho 10000 máy. Như vậy máy nạn nhân sẽ thấy hàng loạt gói tin gửi từ nhiều máy tính đến. Kẻ xâm nhập do đó khó bị lộ dấu vết nếu đã phát tán được worm.

☞ ***Tăng khả năng phá hoại***

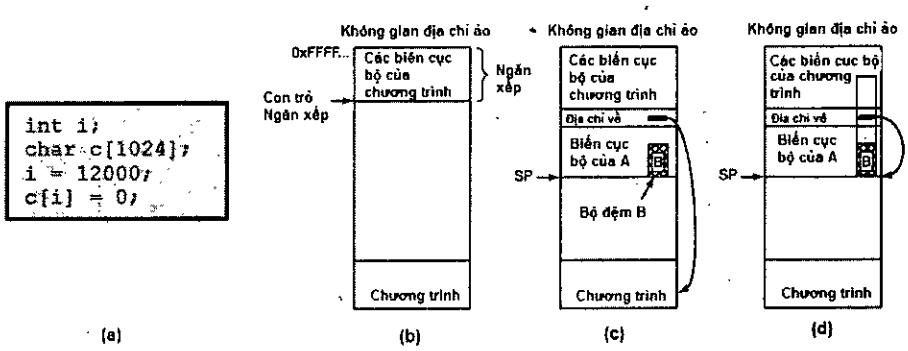
Nếu khi sử dụng một máy tính, kẻ tấn công đạt được một độ phá hoại là X thì có thể đạt được độ phá hoại n × X nếu tấn công từ n các máy nhiễm worm.

☞ ***Cấu tạo của worm***

Hình 12.9a minh họa các thành phần của worm. Các thành phần cấu tạo nên worm hoạt động tương tự như thành phần trong tên lửa: Warhead (đầu) dùng để xuyên thủng mục tiêu, bộ phận Propagation Engine để di chuyển tới mục tiêu, Target Selection Algorithm và Scanning Engine dùng để định vị mục tiêu và Payload chứa các vật liệu để phá hủy mục tiêu.

- **Warhead**. Để chế ngự hệ thống đích, trước hết worm cần dành được quyền truy cập vào máy tính nạn nhân. Nhiệm vụ của Warhead là mở đường xâm nhập mục tiêu (đầu nhọn cắm ngập vào đối tượng cần phá hủy). Có nhiều cách để làm việc này, chẳng hạn khai thác lỗi tràn bộ đệm. Trình biên dịch C không thực hiện kiểm tra truy cập có vượt quá giới hạn không, ví dụ đoạn mã minh họa trên Hình 12.8a – tuy hợp lệ về cú pháp nhưng lại không được kiểm tra. Do đó byte nào đó ở sau mảng c 10.976 byte sẽ bị viết đè giá trị 0 – điều này có thể gây tác hại lớn cho hệ thống. Trong Hình 12.8b, chúng ta thấy chương trình chính đang chạy có các biến cục bộ lưu trong ngăn xếp. Hình 12.8c minh họa khi chương trình chính gọi thủ tục A. Khi gọi thủ tục con, HĐH cất địa chỉ trả về vào ngăn xếp rồi chuyển quyền điều khiển cho A. A cũng dùng vùng ngăn xếp để lưu các biến cục bộ của mình. Giả sử A cần đường dẫn tuyệt đối của một file nào đó và đặt đường dẫn này vào một mảng cố định 250 byte (bộ đệm B). Khi người dùng chương trình gõ vào tên file với 2000 ký tự, thủ tục sẽ sao chép khối ký tự này vào bộ đệm. Khối ký tự sẽ làm tràn bộ đệm và ghi đè lên vùng bộ nhớ phía trên (Hình 12.8d). Khi đó có thể

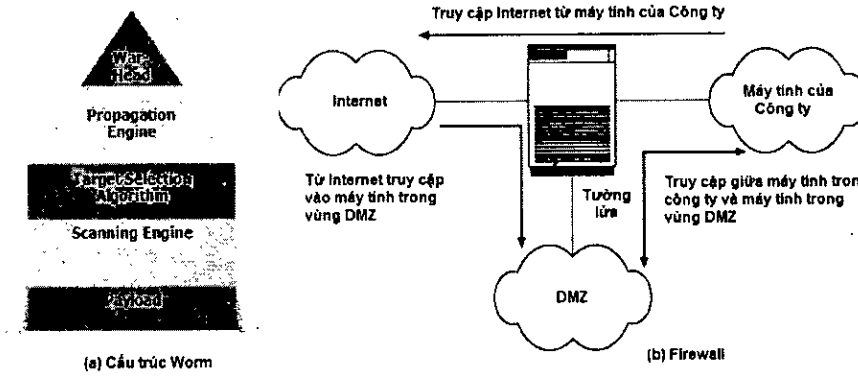
địa chỉ trả về chương trình chính cũng bị ghi đè, và kẻ tấn công có thể ghi địa chỉ chương trình của mình vào địa chỉ trả về. Sau khi thủ tục A chạy xong, thay vì quay lại chương trình chính, A chuyển quyền điều khiển cho chương trình của kẻ tấn công



Hình 12.8. Lỗi tràn bộ đệm

- **Propagation Engine:** Sau khi truy cập vào máy tính nạn nhân, Warhead lấy các chỉ thị từ Propagation Engine để tiếp tục hoạt động. Propagation Engine là kho chứa các chỉ thị để Warhead tải từng phần rồi thực thi. Có những khi worm được tải hoàn toàn vào trong hệ thống đích (như qua File Sharing và Email), lúc này Warhead và Propagation Engine là một. Một khi đã vào được bên trong máy tính nạn nhân, worm có thể dùng các cơ chế của virus để lây nhiễm qua file và ẩn mình kín đáo.
- **Target Selection Algorithm:** Một khi đã vào được bên trong hệ thống, thành phần này của worm bắt đầu hoạt động, tìm kiếm địa chỉ các nạn nhân kế tiếp thông qua địa chỉ Email, Host List, các máy trong cùng mạng.
- **Scanning Engine:** Sử dụng địa chỉ mà Target Selection Algorithm cung cấp, worm bắt đầu quét trên toàn mạng để tìm nạn nhân tiềm tàng. Với mỗi máy tính nạn nhân tìm thấy, worm gửi các packet thăm dò xem liệu Warhead có thể thâm nhập vào máy nạn nhân không, và nếu có thì lại đến công việc của Propagation Engine.
- **Payload:** Bộ phận này chứa các chỉ thị phá hoại máy tính đích. Cũng có một số loại worm không làm gì cụ thể ngoại trừ lấy máy tính nạn nhân làm trung gian cho việc tiếp tục lây lan, trong trường hợp đó,

Payload không chứa gì cả. Ngược lại, Payload thường gắn backdoor để toàn quyền kiểm soát máy nạn nhân từ xa.



Hình 12.9. Worm và Firewall

12.6. GIÁM SÁT NGUY CƠ

Có thể tăng cường an ninh hệ thống bằng hai kỹ thuật. Cách thứ nhất là giám sát nguy cơ. Hệ thống kiểm tra những chuỗi sự kiện có hành vi khả nghi nhằm phát hiện sớm hành động vi phạm quy tắc an ninh. Ví dụ điển hình của kỹ thuật này là đếm số lần nhập sai mật khẩu khi người dùng cố gắng đăng nhập. Nhiều lần đăng nhập sai có thể xem là dấu hiệu của việc dò mật khẩu. Hiện tại trên nhiều trang web, nếu bạn đăng nhập sai quá 5 lần, tài khoản có thể bị phong tỏa trong một khoảng thời gian nào đó. Việc này sẽ kéo dài thời gian dò tìm mật khẩu của kẻ xấu. Trong nhiều ngân hàng, nếu nhập sai số PIN trong nhiều lần liên tiếp, tài khoản ngân hàng sẽ bị phong tỏa cho đến khi chủ nhân thực sự của tài khoản đến làm việc trực tiếp với ngân hàng.

Một kỹ thuật khác là ghi nhật ký, ghi lại thời gian, người dùng và tất cả hình thức truy cập vào đối tượng. Sau khi hệ thống an ninh bị xâm phạm, người quản trị sử dụng nhật ký để tìm ra cách thức và thời gian phát sinh vấn đề. Những thông tin này cực kỳ hữu ích, kể cả khi cần khôi phục lại hệ thống sau khi bị phá hoại lẫn khi nâng cấp cải tiến cơ chế an ninh tốt hơn cho tương lai. Không may, nhật ký có thể rất lớn và người dùng không được quyền sử dụng tài nguyên dùng để ghi nhật ký (lãng phí tài nguyên).

Thay vì ghi nhật ký hoạt động có thể định kỳ quét hệ thống để phát hiện lỗ hổng an ninh. Có thể quét khi máy tính không được sử dụng, do đó gây ít ảnh hưởng tới người dùng hơn phương pháp ghi nhật ký. Một lần quét có thể kiểm tra những điểm yếu sau của hệ thống:

- Mật khẩu ngắn hoặc dễ đoán.
- Chương trình thiết lập bit định danh người dùng hiệu dụng không hợp lệ (nếu hệ thống hỗ trợ kỹ thuật này).
- Chương trình chưa kiểm chứng nằm trong thư mục hệ thống.
- Các tiến trình chạy lâu một cách đáng ngờ.
- Thiết lập các mức bảo vệ không đúng quy định trên các thư mục người dùng cũng như thư mục hệ thống,
- Thiết lập các mức bảo vệ không đúng quy định với file dữ liệu hệ thống, chẳng hạn file mật khẩu.
- Những thành phần nguy hiểm trong đường dẫn tìm kiếm file chương trình.
- Thay đổi chương trình hệ thống (có thể bị phát hiện do thay đổi checksum).

Bất cứ khi nào phát hiện ra lỗ hổng an ninh, hệ thống có thể tự động vá lại lỗ hổng hoặc thông báo cho người quản trị hệ thống. Các máy tính nối mạng dễ bị tấn công (về mặt an ninh) hơn các máy tính riêng lẻ. Thay vì tấn công từ các điểm truy cập cố định, chúng ta thường phải hứng chịu những tấn công từ những điểm chưa xác định hoặc từ nhiều máy tính khác - một vấn đề an ninh cực kỳ nghiêm trọng.

Chính phủ liên bang Mỹ chỉ coi hệ thống là an ninh nếu bên ngoài không thể kết nối tới. Ví dụ, một hệ thống ở mức bí mật nhất chỉ được truy cập trong tòa nhà cũng được xếp vào loại bí mật nhất. Mức độ an ninh của hệ thống giảm xuống nếu như cho phép hệ thống trao đổi dữ liệu với môi trường bên ngoài. Một số cơ quan chính phủ thực hiện những phòng ngừa bảo vệ rất nghiêm ngặt. Tất cả các thiết bị nối tới một trạm đầu cuối kết nối với máy tính được bảo vệ sẽ phải được cất an toàn trong văn phòng khi không sử dụng trạm đầu cuối. Một người muốn truy cập vào máy tính cần phải biết một tổ hợp khóa (là những thông tin phục vụ mục đích kiểm chứng) cũng như địa điểm đặt máy tính.

Điều không may cho các nhà quản trị hệ thống cũng như cho các chuyên gia an ninh máy tính là khó có thể khóa máy tính thường xuyên trong phòng và không cho ai sử dụng. Ví dụ mạng Internet hiện thời kết nối hàng triệu máy tính, và đây là một nguồn tài nguyên cực kỳ quý giá đối với nhiều cơ quan tổ chức cũng như cá nhân. Kẻ xấu sẽ sử dụng các công cụ có thể để truy cập vào các máy tính nối mạng. Vấn đề là làm thế nào để một máy tính an toàn có thể kết nối vào mạng không tin cậy? Giải pháp là sử dụng tường lửa (firewall) để tách biệt hệ thống tin cậy với môi trường không tin cậy. Tường lửa được cài đặt trên máy tính hoặc router ở giữa hệ thống tin cậy và môi trường bên ngoài. Nhiệm vụ của tường lửa là giới hạn các truy cập qua mạng giữa hai miền bảo vệ khác nhau. Ví dụ, Web server sẽ sử dụng giao thức http để trao đổi dữ liệu với Web client. Do đó tường lửa phải cho phép gói tin http đi qua.

Trên thực tế, tường lửa chia mạng máy tính thành nhiều miền bảo vệ. Chẳng hạn có thể coi mạng Internet là miền không tin cậy. Mạng bán tin cậy hay bán an ninh – còn được gọi là DMZ (demilitarized zone) là miền bảo vệ khác và mạng máy tính của công ty là miền bảo vệ thứ 3 (minh họa trên Hình 12.9b). Kết nối từ Internet tới máy tính trong miền DMZ cũng như từ máy tính công ty ra Internet được phép, nhưng kết nối theo chiều ngược lại (từ Internet hay máy tính trong miền DMZ đến máy tính của công ty) bị cấm. Đôi khi có thể cho phép một vài kết nối (có kiểm soát) từ máy tính của DMZ sang máy tính công ty. Ví dụ Web server trên DMZ có thể gửi truy vấn tới cơ sở dữ liệu bên trong công ty. Nhưng những truy vấn như thế chỉ có thể thực hiện trên một số máy tính cụ thể trong công ty chứ không phải tất cả các máy tính.

12.7. MẬT MÃ VÀ ỨNG DỤNG

cuu duong than cong . com

Mã hóa là áp dụng một quy tắc nào đó để biến dữ liệu (gọi là bản rõ) sang định dạng khác (bản mã). Ví dụ xâu "**Security**" có thể được mã hóa thành xâu "**Tfdvsjuz**" nếu quy tắc mã hóa là dịch chuyển mỗi ký tự trong xâu ban đầu một vị trí trong bảng chữ cái. Quá trình ngược lại được gọi là giải mã, từ bản mã chuyển thành bản rõ. Hai đối tượng trao đổi thông tin thỏa thuận và biết trước về quy tắc mã hóa/giải mã để những đối tượng khác không thể xem trộm được thông tin.

Mã hóa xuất phát từ nhu cầu che giấu thông tin không cho người khác phát hiện. Ví dụ trước khi sử dụng chương trình Yahoo Messenger, người dùng phải đăng nhập tên sử dụng và mật khẩu. Yahoo đã có cơ chế mã hóa mật khẩu trước khi gửi đi trên đường truyền Internet để cho dù hacker bắt được gói tin chứa mật khẩu thì mật khẩu cũng không bị lộ.

12.7.1. Khóa

Đa phần thuật toán mã hóa sử dụng Khóa. Tham số đầu vào của thuật toán ngoài xâu cần mã hóa còn thêm khóa là số nguyên hay số thực do người dùng chọn. Khóa làm tăng tính bảo mật của thuật toán. Xét ví dụ trên, nếu thay thế ký tự bằng ký tự ngay sau nó trong bảng chữ cái, xâu "**Security**" biến thành xâu "**Tfdvsjuz**". Trong trường hợp này có thể coi khóa bằng 1. Tương tự, nếu khóa bằng 5 hoặc bằng 10 thì xâu sẽ dịch chuyển 5 hay 10 vị trí. Như vậy, với cùng một thuật toán nhưng với các khóa khác nhau sẽ cho nhiều bản mã khác nhau. Những thuật toán mã hóa tốt còn có đặc điểm: chỉ cần một thay đổi nhỏ trong bản rõ hoặc khóa sẽ dẫn đến một sự thay đổi hoàn toàn ở bản mã. Có thể dễ dàng chỉ ra một số ưu điểm khi sử dụng khóa như sau:

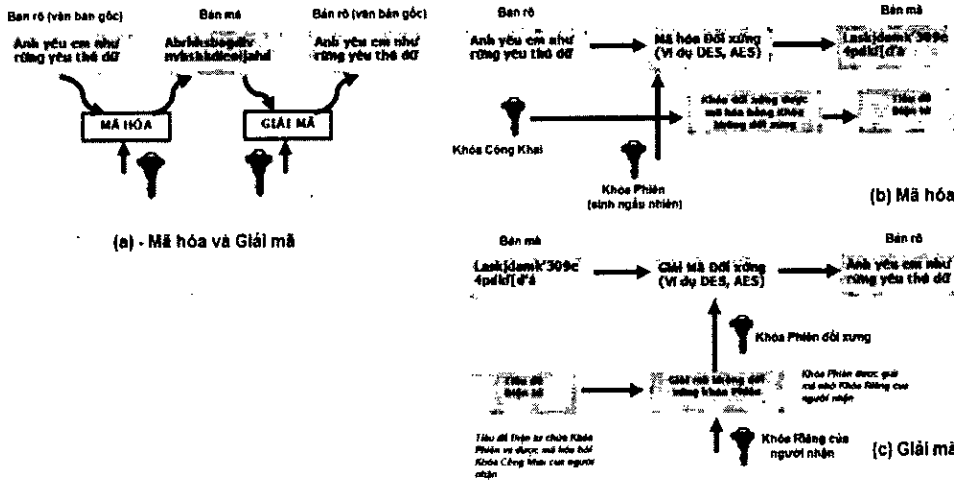
thuongmaitruongthancong . com

- Trong đa số các thuật toán mã hóa, một thay đổi nhỏ của khóa có thể dẫn đến thay đổi đáng kể, thậm chí bản mã hoàn toàn khác biệt.
- Nếu chỉ biết thuật toán mã hóa mà không biết khóa thì chưa khai thác được thông tin.
- Nghĩ ra thuật toán mới đòi hỏi nhiều thời gian và công sức, trong khi đó tạo khóa rất đơn giản. Nói cách khác, số thuật toán mã hóa hữu hạn còn số khóa dường như vô hạn (2^{128} khóa có độ dài 128 bit).

Nên công khai thuật toán mã hóa để cộng đồng người sử dụng thuật toán có thể nhìn thấy mã nguồn của thuật toán để tin tưởng sử dụng, hoặc giúp phát hiện lỗi và cải tiến thuật toán trở nên tốt hơn. Thay vì che dấu thuật toán, hệ thống chỉ dấu đi khóa, như vậy thậm chí không những giúp tính bảo mật cao hơn mà còn tăng sự linh hoạt lên đáng kể.

Khóa càng ngắn thì chương trình mã hóa/ giải mã chạy càng nhanh. 2^{128} là một con số vô cùng lớn, đảm bảo người bình thường không thể nào dò được khóa. Vấn đề đặt ra là nếu những máy tính mạnh hàng đầu thế giới vào cuộc với thuật toán dò tìm vét cạn thì khóa có độ dài 128 bit liệu có đủ an

toàn không? Nói chung, độ dài khóa được lựa chọn sao cho tốc độ phát triển của công nghệ trong tương lai gần cũng không đủ sức tìm được khóa.



Hình 12.10. Mã hóa và giải mã

Hiện nay có hai loại kỹ thuật mã hóa cơ bản là đối xứng và không đối xứng.

cuu duong than cong . com

- **Thuật toán đối xứng**: Sử dụng một khóa để mã hóa và giải mã. Ưu điểm là nhanh chóng và hiệu quả.
- **Thuật toán không đối xứng**: Sử dụng hai khóa: Khóa công khai để mã hóa và Khóa riêng để giải mã. Nhược điểm của thuật toán là chậm. Ưu điểm lớn nhất của thuật toán không đối xứng là cho phép 2 người không quen biết trên mạng có thể trao đổi dữ liệu với nhau.

Trên thực tế, người ta thường cài đặt mô hình lai của hai thuật toán. Về bản chất, dữ liệu được mã hóa và giải mã bằng thuật toán đối xứng cùng với khóa đối xứng, và sử dụng thuật toán mã hóa không đối xứng để mã hóa khóa đối xứng. Cụ thể quá trình mã hóa và giải mã được minh họa trong Hình 12.10.

cuu duong than cong . com

12.7.2. Ứng dụng mật mã trong an ninh

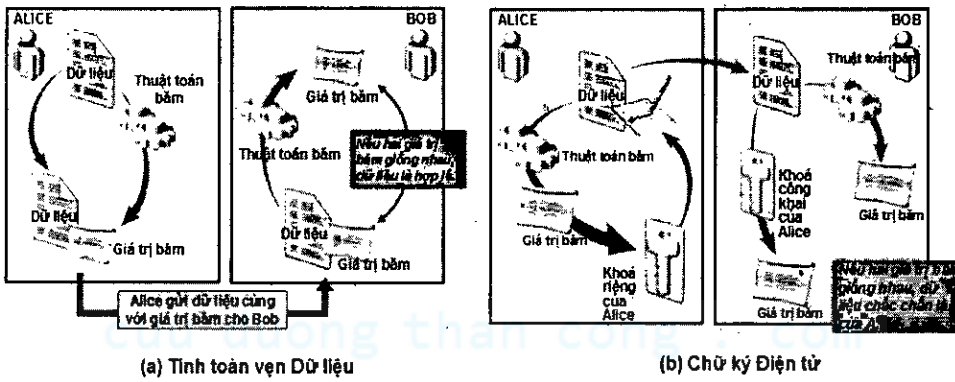
Có thể kết hợp hai phương pháp mã hóa đối xứng và không đối xứng để giải quyết một loạt các vấn đề an ninh mạng: Tính bí mật, Tính toàn vẹn dữ liệu, Chứng thực người dùng, Định danh thực thể, Tính chống từ chối.

☞ *Phương pháp băm*

Băm không phải là phương pháp mã hóa (do không có quá trình giải mã), nhưng lại đóng vai trò cực kỳ quan trọng. Một số hàm băm như MD5, SHA (Secure Hash Algorithm) đã và đang được ứng dụng rộng rãi trong thực tiễn. Bản chất của băm là hàm **f** biến một dữ liệu **x** bất kỳ thành một xâu có độ dài cố định **f(x)**. Một đặc tính quan trọng là khó tìm được hai thông điệp sau khi băm lại cho ra hai giá trị băm giống nhau. Một điểm nữa là cực kỳ khó xác định dữ liệu ban đầu từ giá trị băm cho trước. Đồng thời một thay đổi nhỏ trong dữ liệu ban đầu cũng khiến giá trị băm trở nên hoàn toàn khác biệt.

☞ *Tính toàn vẹn dữ liệu*

Giả sử người dùng Alice muốn gửi file dữ liệu đến Bob. Alice tính giá trị băm trên dữ liệu và gửi giá trị này cùng dữ liệu đến Bob. Bob nhận được dữ liệu sẽ dùng hàm băm để tạo ra giá trị băm mới. Nếu giá trị băm mới và giá trị băm Bob nhận được giống nhau, thì dữ liệu được coi là không bị thay đổi trên đường truyền (xác suất không phát hiện được sự thay đổi là rất nhỏ, có thể bỏ qua được, vì nếu dữ liệu bị thay đổi trên đường truyền thì giá trị băm mới sẽ khác giá trị băm ban đầu). Quá trình này được minh họa trên Hình 12.11a.



(a) Tính toàn vẹn Dữ liệu (b) Chữ ký Điện tử

Hình 12.11. Ứng dụng của mật mã

☞ *Chữ ký điện tử*

Có thể thông điệp Alice gửi Bob bị Trudy chặn lại và thay thế bằng thông điệp mới với giá trị băm mới. Bob kiểm tra thấy thông điệp (do Trudy gửi) hợp lệ và tưởng rằng đã nhận được thông điệp từ Alice. Như vậy, cần

thêm một cơ chế để khẳng định thông điệp gửi đến thực sự xuất phát từ Alice. Chữ ký điện tử hoạt động như sau: Sau khi thông điệp được băm, Alice mã hóa giá trị băm bằng khóa riêng của mình rồi gửi thông điệp cùng giá trị băm đã mã hóa cho Bob. Bob nhận được thông điệp sẽ giải mã bằng khóa công khai của Alice, đồng thời băm thông điệp của Alice ra một xâu. So sánh hai giá trị băm này, nếu giống nhau thì chắc chắn thông điệp đến từ Alice (vì chỉ Alice mới biết được khóa riêng của mình và không ai giả mạo được). Quá trình này được minh họa trên Hình 12.11b.

☞ ***Chứng nhận điện tử***

Chứng nhận điện tử giải quyết trường hợp Bob nhận được thông tin từ Alice nhưng muốn kiểm chứng xem tư cách của Alice. Chứng nhận điện tử giống như những chứng nhận trong cuộc sống hằng ngày: bằng lái xe, bằng tốt nghiệp, bằng khen... - là những bằng chứng khẳng định tư cách, khả năng của một người trong xã hội. Alice nếu muốn chứng minh cho mọi người biết mình là một công ty hay tổ chức nào thì phải xin cấp phát một chứng nhận bằng cách gửi khóa công khai của mình đến một cơ quan uy tín cấp phát chứng nhận. Cơ quan này xem xét và nếu thấy Alice có đủ tư cách sẽ ký bằng chữ ký điện tử của cơ quan cấp phát vào giấy chứng nhận gửi về cho Alice. Nếu có được giấy chứng nhận, mọi chủ thể trên mạng sẽ hoàn toàn tin tưởng vào Alice vì giấy chứng nhận đã có chữ ký của cơ quan cấp phát chứng nhận có uy tín.

12.8. NHẬN XÉT

Hệ thống máy tính bao gồm nhiều đối tượng. Các đối tượng cần được bảo vệ để không bị sử dụng sai mục đích. Đối tượng có thể là phần cứng (bộ nhớ, thời gian sử dụng CPU, thiết bị vào/ra) hoặc phần mềm (file, chương trình, dữ liệu trừu tượng). Quyền truy cập là sự được phép thực hiện thao tác nào đó trên đối tượng. Miền bảo vệ là tập hợp các quyền truy cập. Các tiến trình thực thi trong miền bảo vệ có thể sử dụng quyền truy cập thuộc miền này để thao tác trên đối tượng. Ma trận quyền truy cập là mô hình bảo vệ tổng quát. Ma trận cung cấp cơ chế bảo vệ mà không áp đặt bất kỳ chính sách bảo vệ cụ thể nào lên hệ thống hoặc người sử dụng. Sự tách biệt giữa hai khái niệm "chính sách" và "cơ chế" là yếu tố quan trọng khi thiết kế

HĐH. Ma trận quyền truy cập khá thưa và thường được cài đặt dưới dạng danh sách <miền, quyền bảo vệ> gắn với mỗi đối tượng, hoặc danh sách khả năng gắn với mỗi miền bảo vệ.

Chúng ta có thể thêm tính năng bảo vệ động trong ma trận quyền truy cập bằng cách coi miền bảo vệ và ma trận quyền truy cập cũng là đối tượng đặc biệt. Bảo vệ là vấn đề mang tính nội bộ. Vấn đề an ninh phải được thực hiện trên cả hai phương diện: hệ thống máy tính và môi trường cài đặt hệ thống (con người, nhà cửa, công việc...). Dữ liệu lưu trong hệ thống máy tính cần được bảo vệ khỏi những truy cập chưa kiểm chứng, các hành vi phá hoại cố ý hay vô tình, và hành vi khiến dữ liệu không nhất quán. Mật mã, hàm băm, chữ ký điện tử, chứng nhận điện tử là các thành phần không thể thiếu trong an ninh máy tính nói chung.

CÂU HỎI ÔN TẬP

1. Trình bày các vấn đề an ninh cơ bản.

2. Trình bày ưu điểm của trừu tượng file.

3. Trình bày khái niệm miền bảo vệ.

4. Trình bày các phương pháp cài đặt ma trận quyền truy cập.

cuu duong than cong . com

cuu duong than cong . com

TÀI LIỆU THAM KHẢO

1. Modern Operating Systems (3rd Edition) (GOAL Series) (Hardcover) by Andrew S. Tanenbaum (Author) Third Edition.
2. Computer Networking: A Top-Down Approach (4th Edition) (Hardcover) by James F. Kurose (Author), Keith W. Ross (Author).
3. Operating Systems: Internals and Design Principles (6th Edition) (GOAL Series) (Hardcover).
4. Operating System Concepts (Hardcover) eight edition by Abraham Silberschatz (Author), Peter Baer Galvin (Author), Greg Gagne (Author).
5. Operating System Concepts with Java (Hardcover) 3 edition by Abraham Silberschatz (Author), Peter Baer Galvin (Author), Greg Gagne (Author).

cuu duong than cong . com

Chịu trách nhiệm xuất bản :
Chủ tịch HĐQT kiêm Tổng Giám đốc NGÔ TRẦN ÁI
Phó Tổng Giám đốc kiêm Tổng biên tập NGUYỄN QUÝ THAO

Biên tập nội dung và sửa bản in :
ĐỖ HỮU PHÚ

Thiết kế mỹ thuật và trình bày bìa :
BÍCH LA

Thiết kế sách và chế bản :
ĐỖ PHÚ

cuu duong than cong . com

cuu duong than cong . com

cuu duong than cong . com

cuu duong than cong . com

GIÁO TRÌNH NGUYÊN LÝ HỆ ĐIỀU HÀNH

Mã số: 7B751y0 – DAI

In 1.000 bản (QĐ : 04), khổ 16 x 24 cm. In tại Công ty Cổ phần In Phúc Yên.

Địa chỉ : Đường Trần Phú, thị xã Phúc Yên, Vĩnh Phúc.

Số ĐKKH xuất bản : 19 – 2010/CXB/254 – 2244/GD.

In xong và nộp lưu chiểu tháng 2 năm 2010.

CÔNG TY CỔ PHẦN SÁCH ĐẠI HỌC - DẠY NGHỀ
HEVOBCO
25 HÀN THUYÊN - HÀ NỘI
Website : www.hevobco.com.vn

LONDON QUALITY CROWN QC 100
VƯƠNG MIỆN KIM CƯƠNG
CHẤT LƯỢNG QUỐC TẾ

TÌM ĐỌC
SÁCH THAM KHẢO VỀ TOÁN - TIN
CỦA NHÀ XUẤT BẢN GIÁO DỤC VIỆT NAM

1.	Toán học cao cấp (tập 1, 2, 3)	Nguyễn Đình Trí (Chủ biên)
2.	Bài tập Toán học cao cấp (tập 1, 2, 3)	Nguyễn Đình Trí (Chủ biên)
3.	Đồ thị và các thuật toán	Hoàng Chí Thành
4.	Giáo trình Nhập môn Hệ cơ sở dữ liệu	Nguyễn Tuệ
5.	Giáo trình Hệ điều hành Unix - Linux	Hà Quang Thụy (Chủ biên)
6.	Giáo trình Khai phá dữ liệu Web	Hà Quang Thụy (Chủ biên)
7.	Toán rời rạc ứng dụng trong tin học	Đỗ Đức Giáo
8.	Hướng dẫn giải Bài tập Toán rời rạc	Đỗ Đức Giáo
9.	Giáo trình Kỹ nghệ phần mềm	Nguyễn Văn Vỵ
10.	Giáo trình Nhập môn mạng máy tính	Hồ Đắc Phương

cuu duong than cong . com

Bạn đọc có thể mua sách tại các Công ty Sách – Thiết bị trường học ở các địa phương hoặc các Cửa hàng sách của Nhà xuất bản Giáo dục Việt Nam :

- Tại TP. Hà Nội : 25 Hàn Thuyên ; 187B Giảng Võ ; 232 Tây Sơn ; 23 Tràng Tiền ;
- Tại TP. Đà Nẵng : Số 15 Nguyễn Chí Thanh ; Số 62 Nguyễn Chí Thanh.
- Tại TP. Hồ Chí Minh : Cửa hàng 451B – 453 Hai Bà Trưng – Quận 3 ; 240 Trần Bình Trọng – Quận 5.
- Tại TP. Cần Thơ : Số 5/5, đường 30/4.

Website : www.hevobco.com.vn

8 934980 078686

Giá: 30.000đ