

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 có 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 trực 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 (đèn tròn) 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 ảnh xạ. Tuy nhiên, mô hình thiết bị hướng khồi 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.

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.

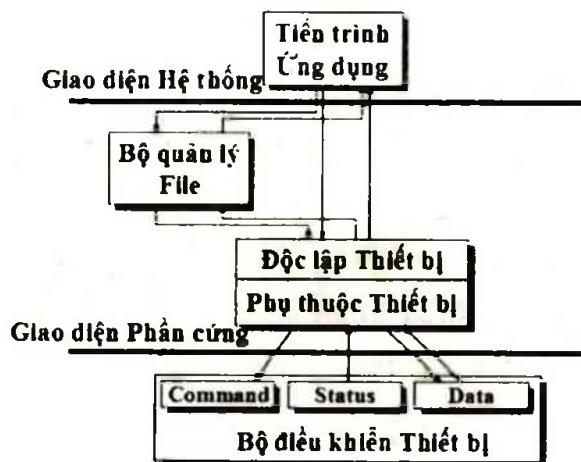
Giao diện giữa bộ điều khiển và thiết bị là giao diện ở mức thấp. Ở mức cao có thể coi ổ đĩa được chia thành nhiều track, track có 256 sector và sector chứa 512 byte. Tuy nhiên, ở mức vật lý thấp nhất (bên trong ổ đĩa) là dòng các bit, bắt đầu với một tiêu đề preamble (chuỗi bit đánh dấu sự bắt đầu), 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 Correcting Code – ECC). Phần tiêu đề preamble được ghi trong quá trình định dạng đĩa, chứa



Hình 8.1. Bộ điều khiển SCSI

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 l่าน 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 đọc 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



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 truy xuất thiết bị thấp nhất. Có nhiều phương pháp truy xuất 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 ngắn, 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 HDH 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 ngắn (cơ chế vào/ra có ngắn), hoặc hoàn toàn bằng trình điều khiển thiết bị nếu không sử dụng ngắn (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:

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.

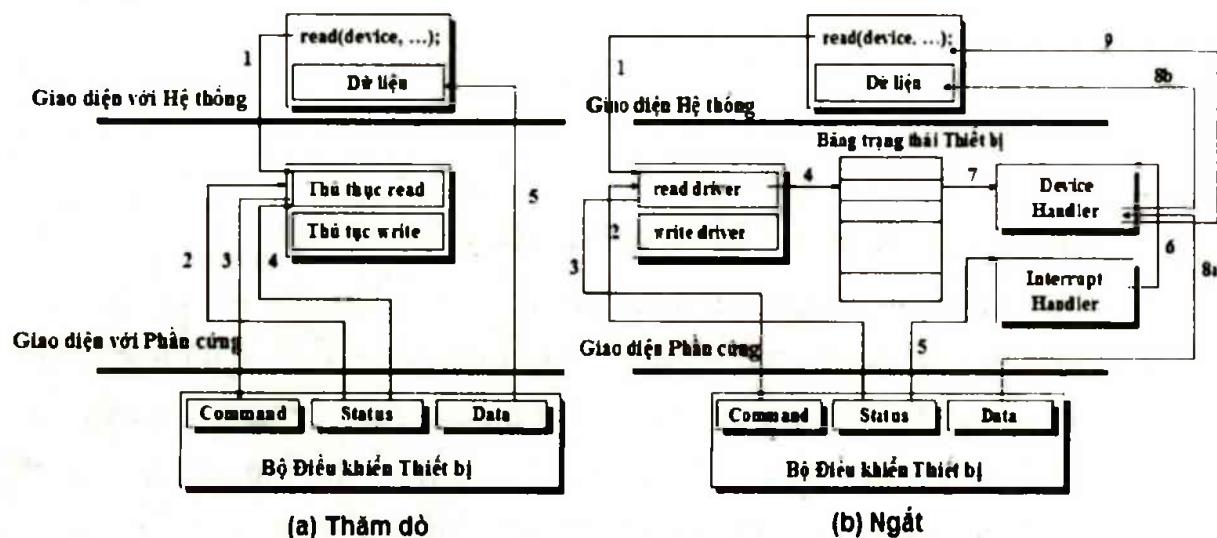
8.2.2. Vào/ra trực tiếp qua thăm dò

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 ngắn 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.

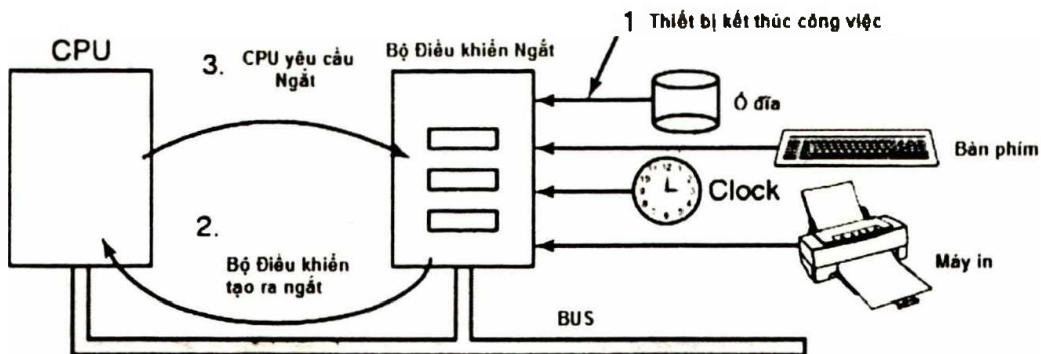
Thao tác ghi dữ liệu được thực hiện tương tự. Trong cơ chế vào/ra trực tiếp qua thăm dò, yêu cầu vào/ra được thực hiện thông qua tương tác với trình điều khiển, điều này "che dấu" toàn bộ hoạt động của phần cứng của bộ điều khiển. Nhược điểm của cách tiếp cận này là hiệu suất sử dụng CPU thấp do CPU 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.



Hình 8.3. Cài đặt bộ điều khiển thiết bị

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 trả 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 HDH 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 chi 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 vẫn 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 trả tới chỉ thị 9, 10 hoặc 11.

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

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

Chú ý rằng, các chỉ thị đăng sau chỉ thị do PC trả 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 ngắt (imprecise interrupt). 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:

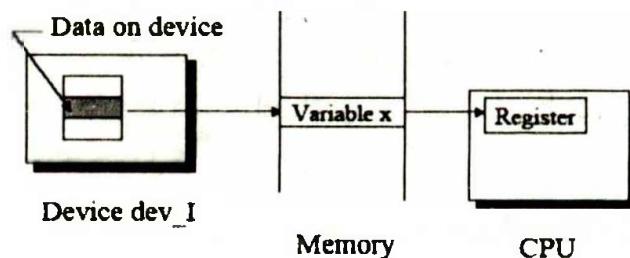
1. Tiến trình ứng dụng yêu cầu đọc dữ liệu.
2. Trình điều khiển kiểm tra thanh ghi trạng thái xem thiết bị có rỗi hay không. Nếu thiết bị bận, trình điều khiển đợi.

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.

```

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

```



Hình 8.5. Xử lý qua ngắt

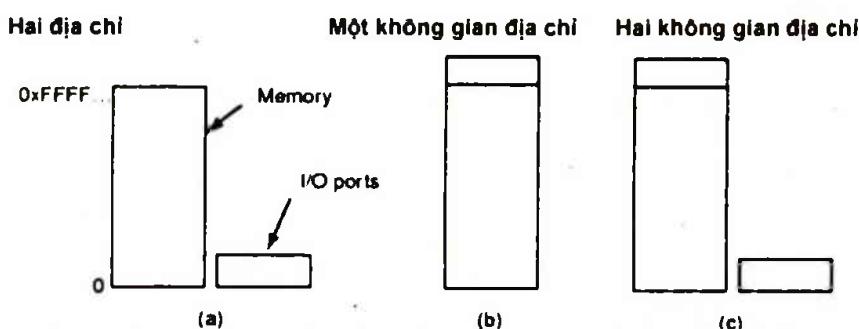
Thao tác ghi dữ liệu diễn ra tương tự. Với tiến trình ứng dụng, thao tác này tương tự lời gọi hệ thống. Tuy nhiên, thời gian ghi có thể lớn hơn rất nhiều, đặc biệt trong hệ thống thăm dò. Thời gian này phụ thuộc vào thời gian tính toán, tốc độ thực hiện vào/ra và khoảng thời gian thăm dò thiết bị. Đối 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

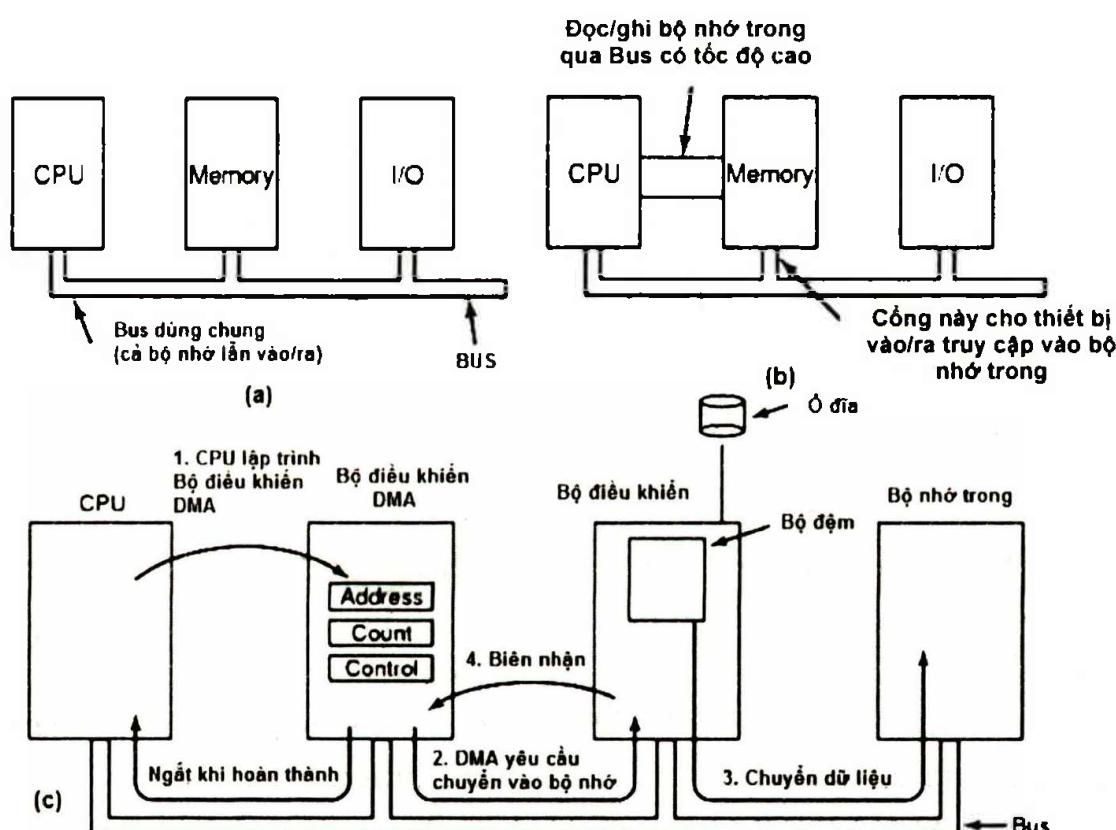
không chỉ làm giảm kích thước kernel mà còn tránh xung đột giữa các bộ điều khiển. Thứ ba, với vào/ra qua ánh xạ bộ nhớ, mọi chỉ thị nếu tham chiếu được tới bộ nhớ cũng sẽ tham chiếu được thanh ghi điều khiển. Ví dụ, nếu chỉ thị **TEST** kiểm tra từ nhớ có bằng 0 hay không, thì cũng có thể kiểm tra được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

LOOP:

TEST	POR_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 chữ 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 **POR_4** sẽ khiến nội dung **POR_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   ioop            // 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,

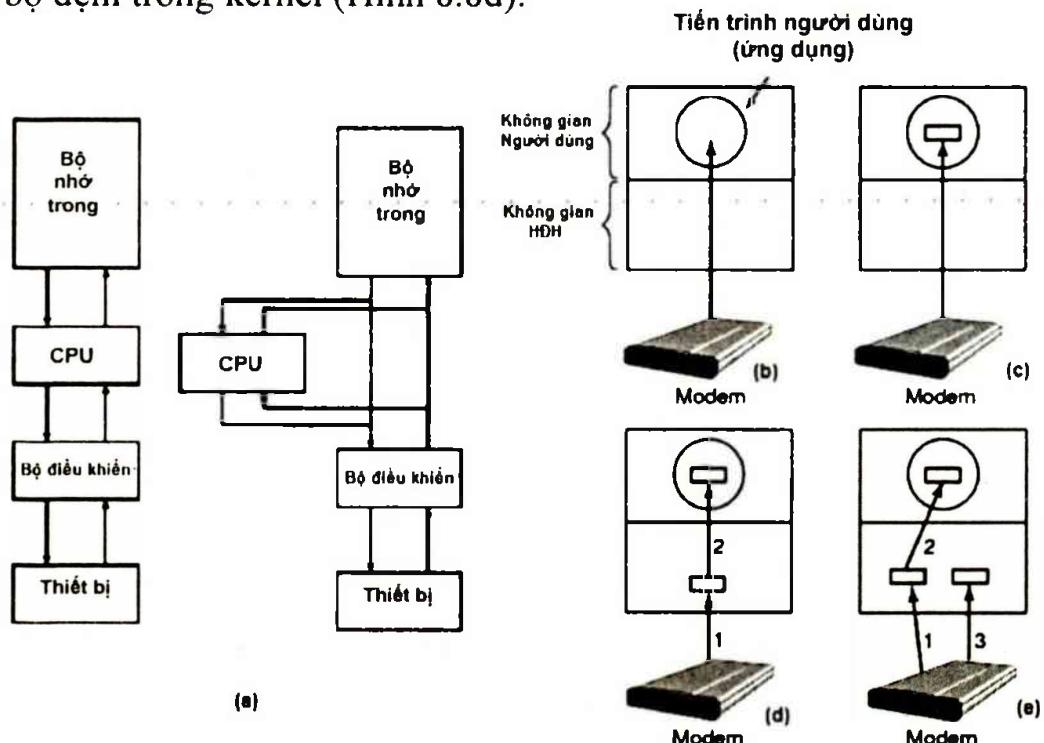
bộ điều khiển DMA khởi tạo việc chuyển dữ liệu bằng cách phát ra yêu cầu đọc tới bộ điều khiển ổ đĩa. Bộ điều khiển ổ đĩa không xác định được yêu cầu này là từ CPU hay từ bộ điều khiển DMA. Do địa chỉ bộ nhớ được ghi trên đường bus địa chỉ, nên bộ điều khiển ổ đĩa lấy byte kế tiếp trong bộ nhớ của mình 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 tồn 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).

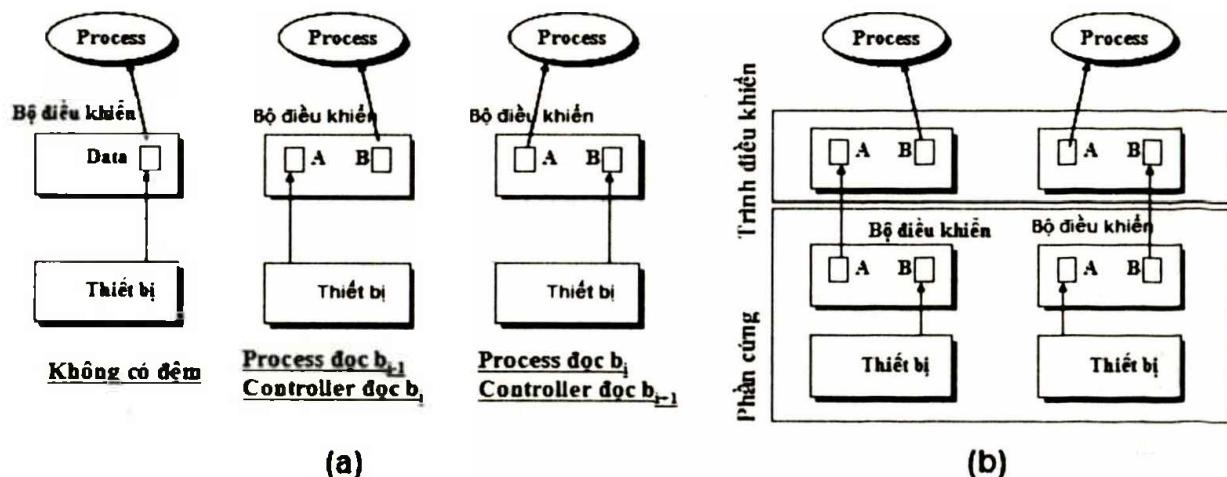


Hình 8.8. Bộ đệm dữ liệu

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.

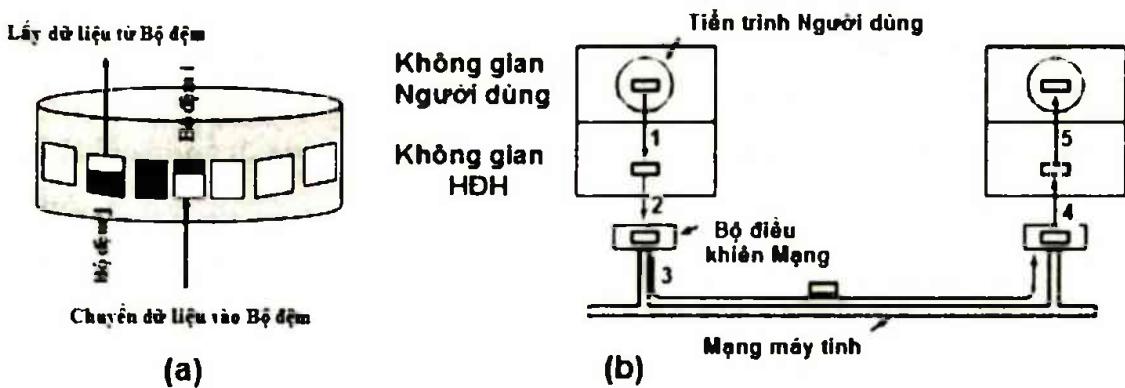


Hình 8.9. Kỹ thuật bộ đệm dữ liệu

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 đồ 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 HDH 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 HDH.

☞ API – Giao diện lập trình ứng dụng

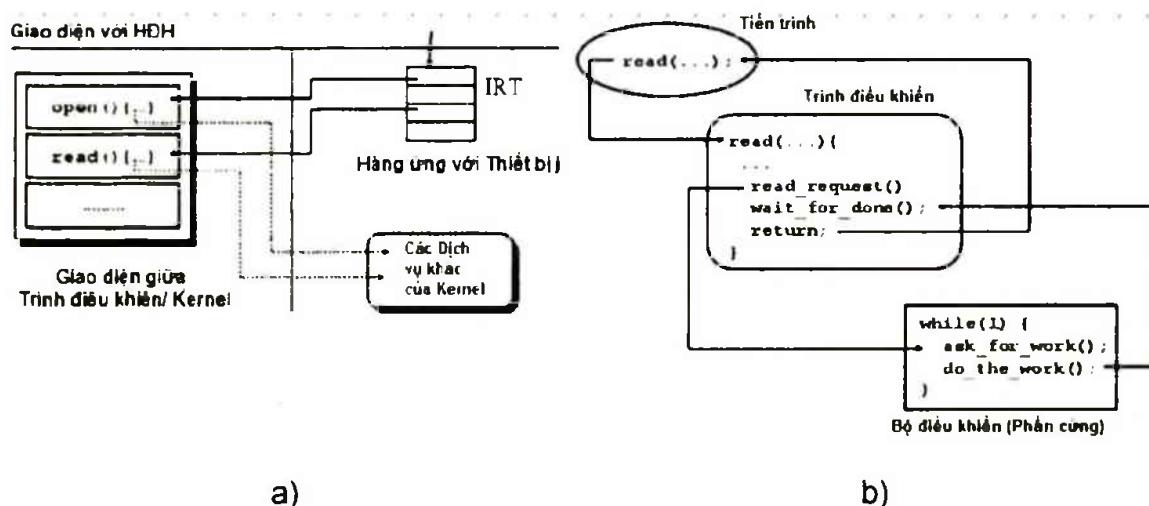
API là giao diện người lập trình sử dụng để thực hiện vào/ra trên thiết bị. Nhiệm vụ chính của thiết bị là truyền thông hoặc lưu trữ. Các thiết bị "vào" thường "tạo" ra thông tin đưa vào hệ thống. Thông tin có thể không có khuôn dạng (đến từ mạng máy tính), hoặc có khuôn dạng xác định (đến từ ổ đĩa). 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.



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 Hình 8.11a, bảng gián tiếp (IRT) của thiết bị j "trò" 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 (Indirect 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 bằng 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 ngắn, trình xử lý ngắn 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.

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

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.

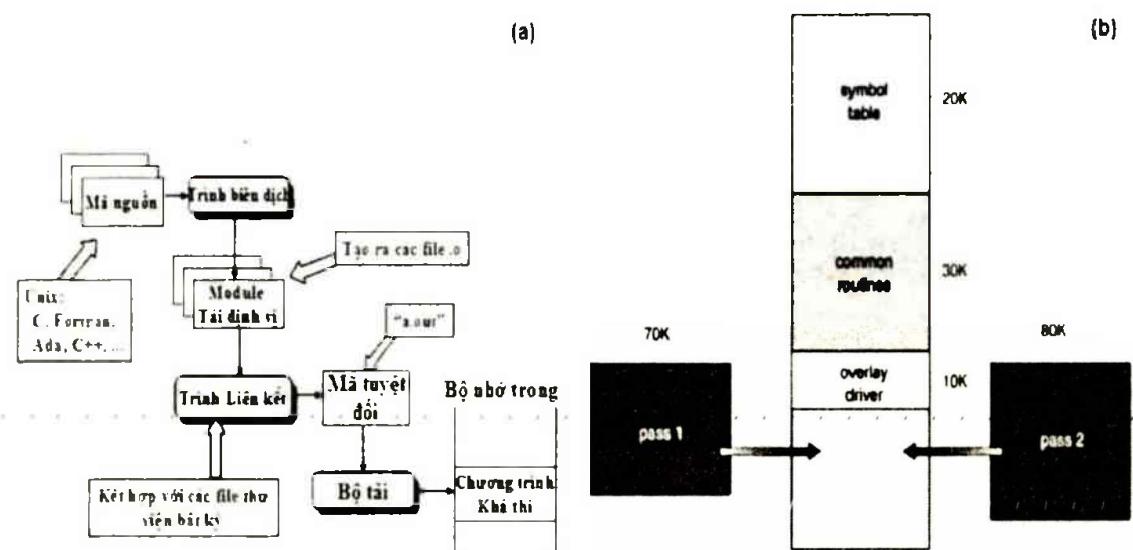
Trình liên kết chỉnh lại các địa chỉ. Đoạn mã trong Hình 9.1b được chuyển sang Hình 9.1c ở vị trí 1008. Như vậy, địa chỉ tương đối 0 trong Hình 9.1b chuyển thành địa chỉ 1008 trong Hình 9.1c. Chú ý, địa chỉ trong module tuyệt đối vẫn bắt đầu từ 0. Tại thời điểm tải, module tuyệt đối được tải vào bộ nhớ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

<pre> *** static int gVar; (a) *** int proc_a(int arg){ *** ... *** gVar = 7; *** put_record(gVar); *** } </pre>	<pre> Code Segment Relative Address Generated Code 0000 (Other modules) ... 1008 entry proc_a ... 1220 load =7, R1 1224 store R1, 0136 1228 push 1036 1232 call 2334 ... 1399 (End of proc_a) ... 2334 entry put_record ... 2670 (optional symbol table) ... 2999 (last location in the code segment) </pre>	<pre> Data Segment Relative Address Generated variable space ... 0136 [Space for gVar variable] ... 1000 (last location in the data segment) </pre>
<pre> Relative Address Generated Code 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) ... 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) </pre>	<pre> Code Segment Relative Address Generated Code 0000 0008 entry proc_a ... 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) </pre>	<pre> Data Segment Relative Address Generated variable space ... 0036 [Space for gVar variable] ... 0049 (last location in the data segment) </pre>
(d)	(c)	(b)

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 tài 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 thẻ 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. Phù (overlay)

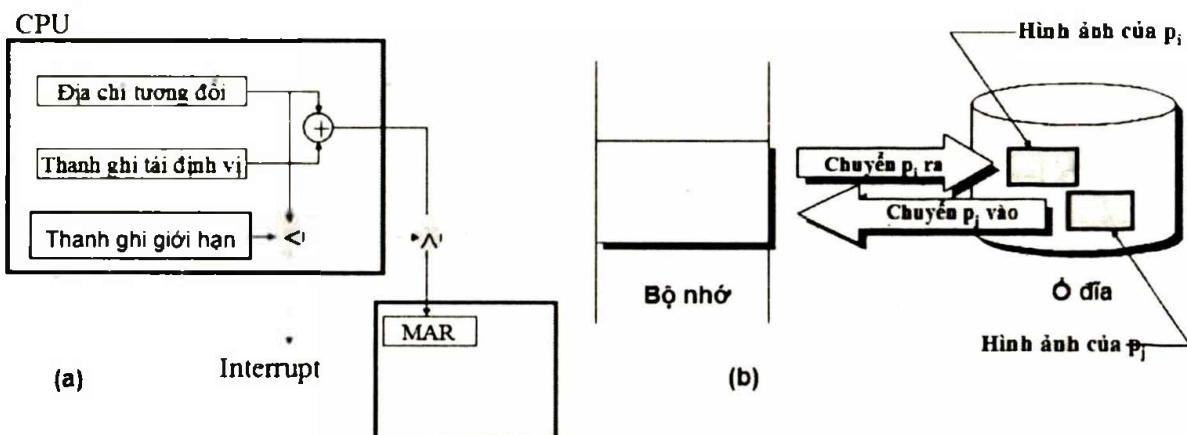
Tài phù là công nghệ cho phép tiến trình lớn hơn lượng bộ nhớ cấp phát. Ý tưởng của kỹ thuật này là tại một thời điểm chỉ lưu trong bộ nhớ những chỉ thị và dữ liệu thực sự cần thiết. Khi không cần thiết, chỉ thị và dữ liệu sẽ được đưa ra ngoài để lấy chỗ cho chỉ thị và dữ liệu cần thiết. Ví dụ, một trình 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 phù và lần lượt được đưa vào bộ nhớ. Giống như tài động, phù không đòi hỏi HĐH hỗ trợ. Người lập trình thiết kế và lập trình cho cấu trúc vùng phù 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 phù) 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 $R + \text{max}$ 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 HDH được đặt trong vùng nhớ thấp.

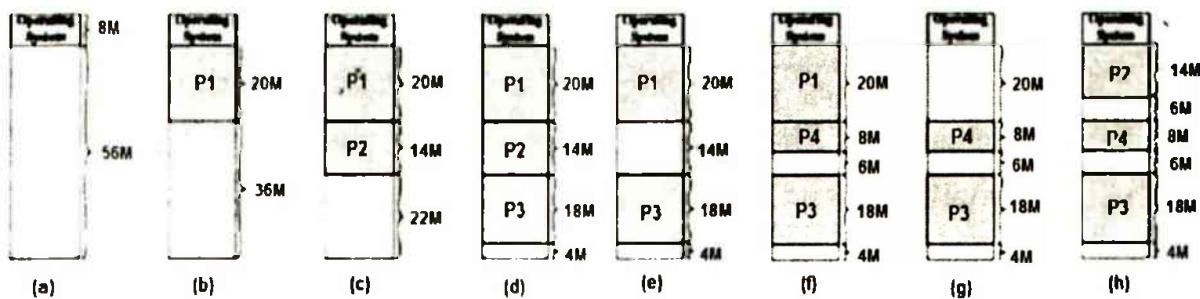
9.4.1. Cấp phát một vùng nhớ liên tục

Không chỉ bảo vệ đoạn mã và dữ liệu của HDH (không cho tiến trình người dùng thay đổi) mà HDH 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 HDH lẫn chương trình và dữ liệu người dùng đều được bảo vệ.

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 rõ, 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 HDH 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).

HDH 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, HDH tìm một khối trống đủ lớn cho tiến trình. Nếu tìm thấy, HDH 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 điêu phổi. 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 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 khói nhớ đúng theo yêu cầu thì sẽ thửa 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 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 này 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.

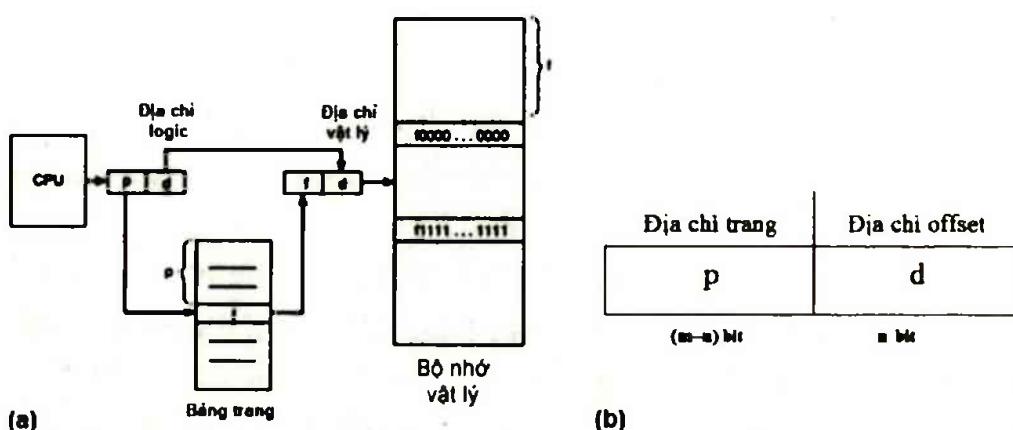
9.5. PHÂN TRANG

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.

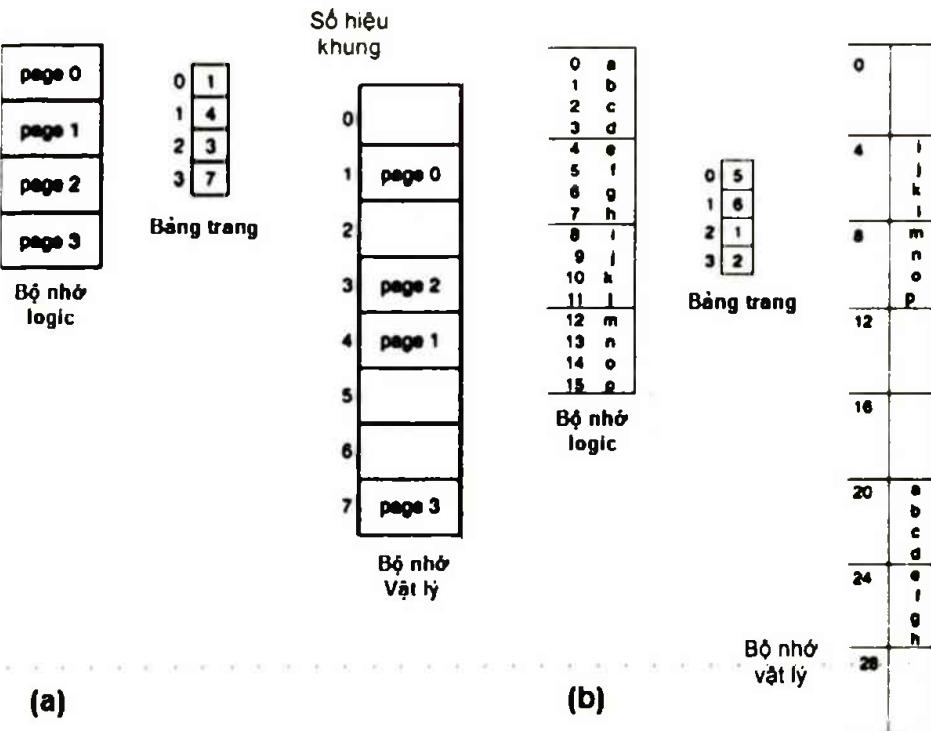
9.5.1. Phương pháp cơ bản

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

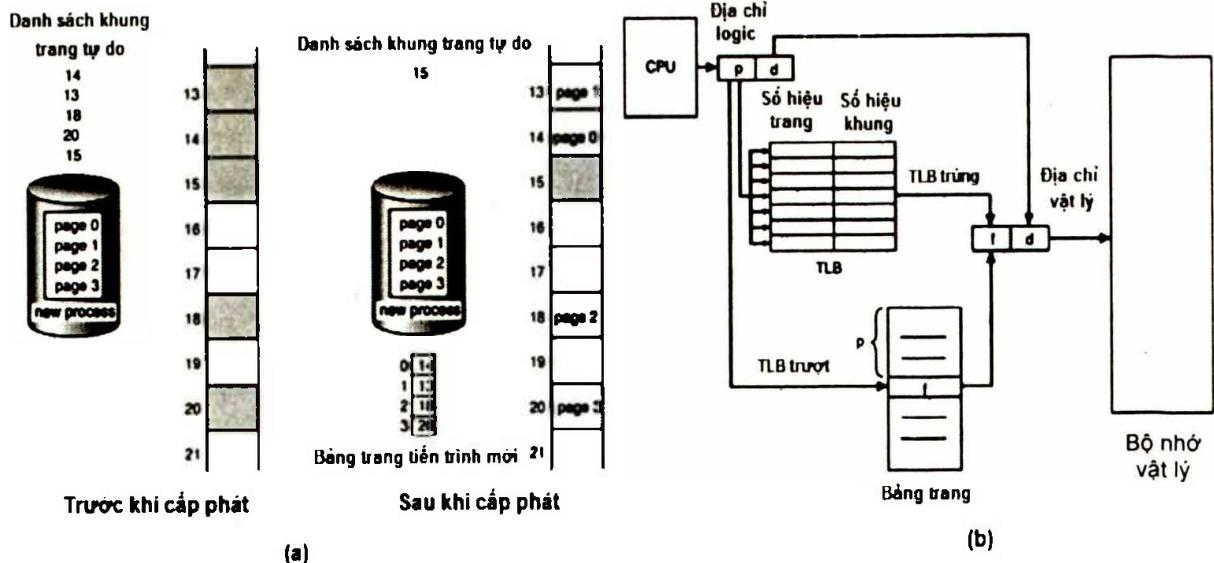
Kích thước trang được kiến trúc phần cứng quy định, thường là lũy thừa của 2, biến thiên từ 512 byte đến 16MB. Nếu kích thước không gian địa chỉ logic là 2^m , kích thước trang là 2^n đơn vị (byte hoặc word), thì $(m - n)$ bit cao của địa chỉ logic xác định số hiệu trang và n bit thấp xác định địa chỉ 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 ($5 \times 4 + 0$). Địa chỉ logic 3 (trang 0, địa chỉ tương đối 3) ứng với địa chỉ vật lý 23 ($5 \times 4 + 3$).



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 phôi 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 trình: 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 khôi điều khiển tiến trình. Khi chọn tiến trình thực thi, bộ điều phôi phải khôi phục lại ngữ cảnh của tiến trình.

Phản ứng

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) trả 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, TLB 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.

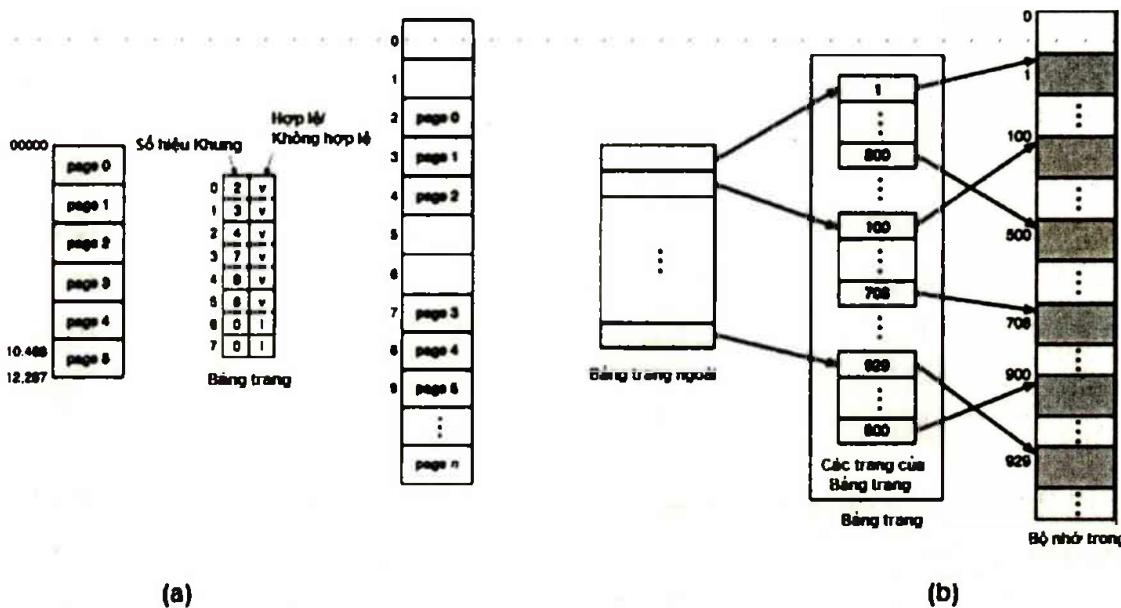
Tỷ lệ trúng (hit ratio) là tỷ lệ tìm thấy số hiệu trang trong TLB, 80% nghĩa là tìm thấy số hiệu trang mong muốn trong 80% số lần tìm kiếm. Nếu tìm kiếm trên TLB mất 20ns và mất 100ns truy xuất bộ nhớ, thì truy xuất bộ nhớ khi có TLB mất 120ns trong trường hợp tìm thấy số hiệu trang trên TLB. Nếu không 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 $0,8 \times 120 + 0,20 \times 220 = 140$ ns. Như vậy, thời gian truy xuất bộ nhớ chậm nhất 40ns (từ 100ns đến 140ns). Với tỷ lệ trúng là 98%, thì thời gian truy cập hiệu dụng là $0,98 \times 120 + 0,02 \times 220 = 122$ ns. Tỷ lệ trúng khá cao này cũng chỉ giúp giảm tốc độ truy xuất bộ nhớ 22%. Hiện 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 ($0 \rightarrow 16383$), có thể có một chương trình chỉ sử dụng các địa chỉ từ $0 \rightarrow 10468$. 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 $10469 \rightarrow 12287$ 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.



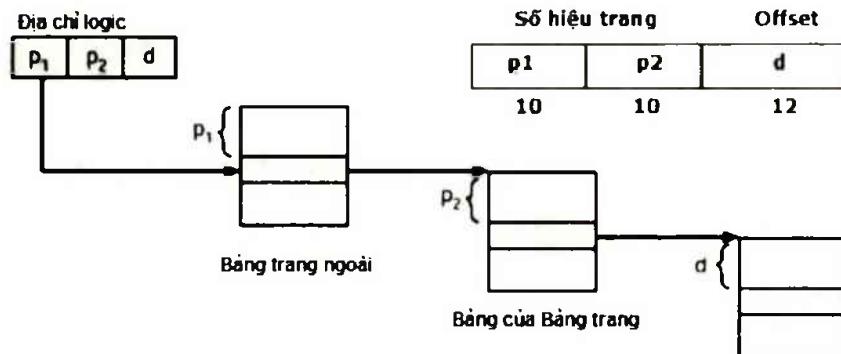
Hình 9.8. Ví dụ bảng trang

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

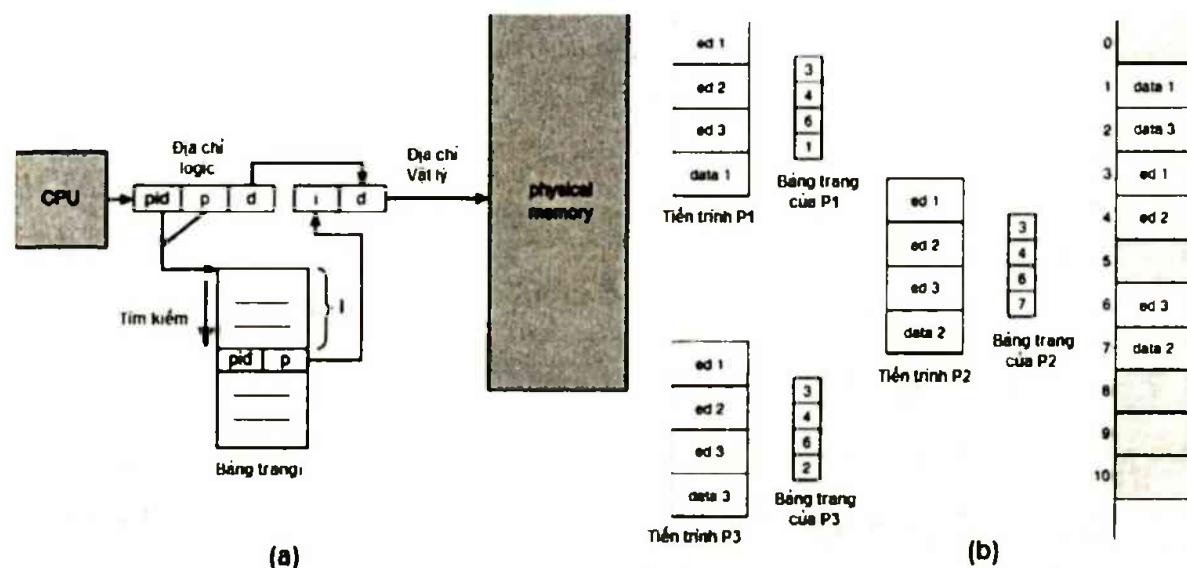
pháp này đôi khi được áp dụng trên một vài bộ vi xử lý 32 bit để tăng tính linh hoạt và hiệu quả. Có một số hệ thống áp dụng nhiều mức, chẳng hạn kiến trúc SPARC (với địa chỉ 32 bit) hỗ trợ phương pháp phân trang 3 mức, kiến trúc Motorola 68030 32 bit hỗ trợ phương pháp phân trang 4 mức.

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

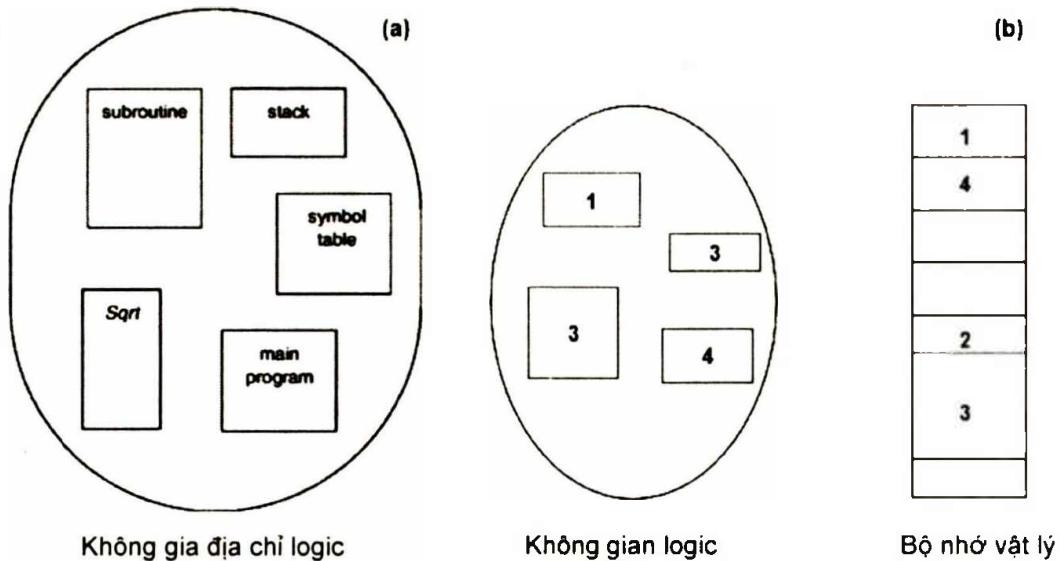
Có thể dùng chung các chương trình được sử dụng thường xuyên như trình biên dịch, hệ thống cửa sổ, hệ quản trị cơ sở dữ liệu,... Để có thể chia sẻ, mã chương trình phải là mã thuần túy. Việc chia sẻ bộ nhớ giữa các tiến trình trong một hệ thống cũng giống cách các thread chia sẻ không gian địa chỉ của 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,...).



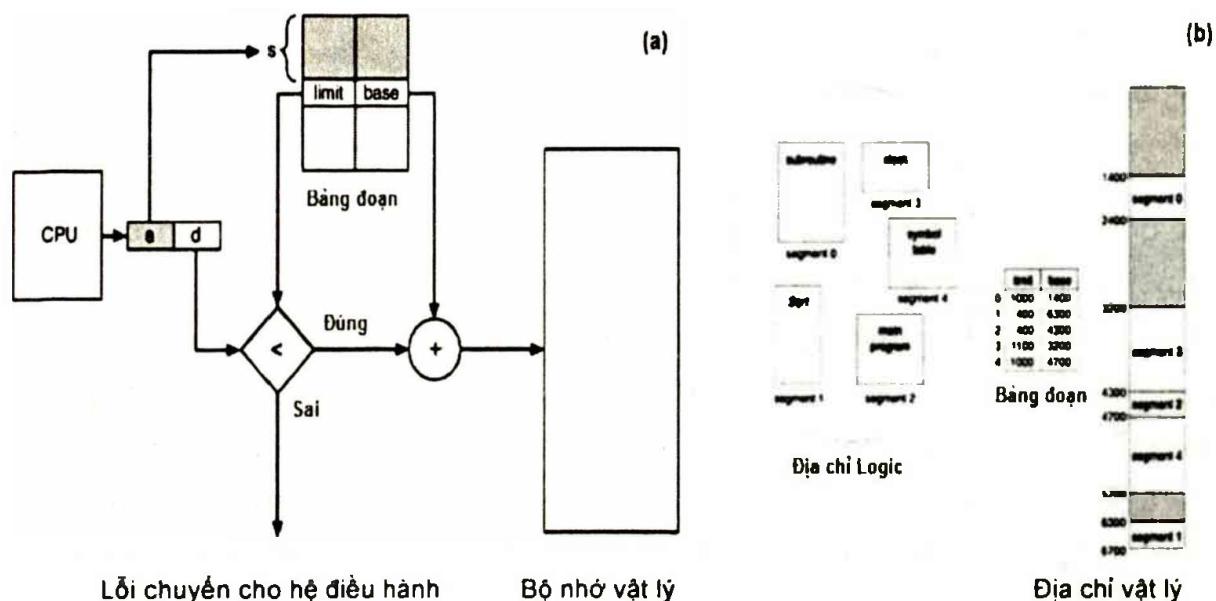
Hình 9.12. Minh họa phân đoạn

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

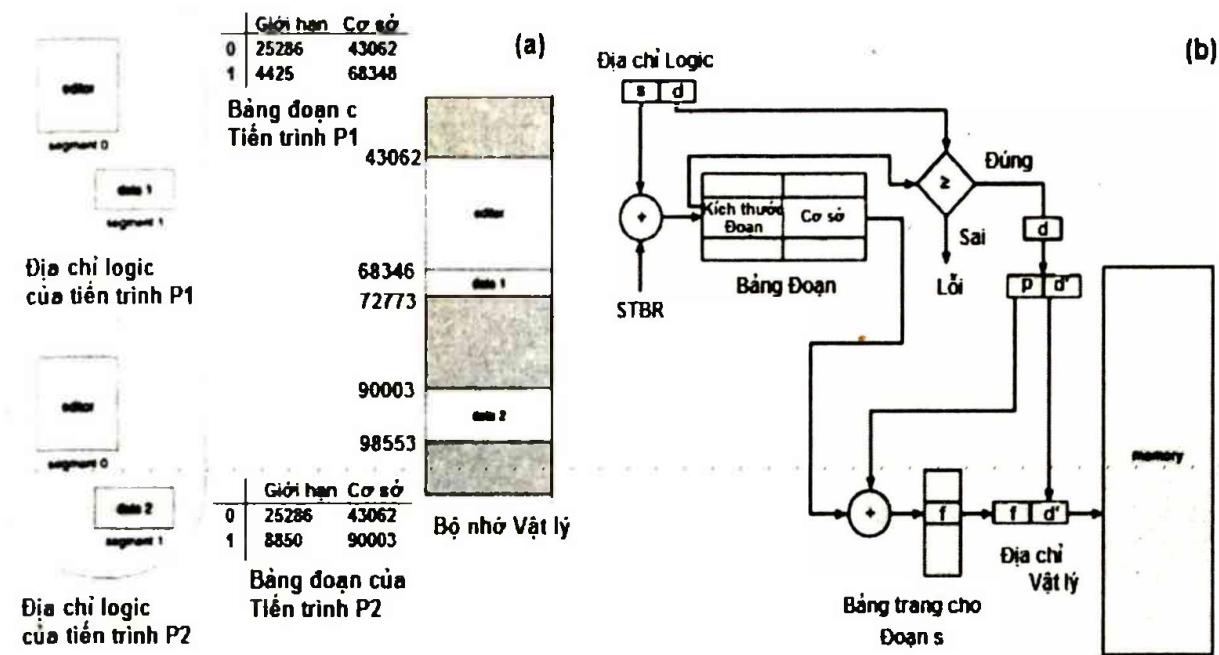
chương trình có nhiều đoạn thì bảng phân đoạn phải được đặt trong bộ nhớ. Thanh ghi cơ sở bảng phân đoạn (Segment-table base register - STBR) trỏ đến bảng đoạn. Vì số đoạn trong mỗi chương trình có thể khác nhau, nên cần thêm thanh ghi kích thước bảng phân đoạn STLR (Segment-table length register). Với 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 phân 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 phân đoạn nằm trong khối điều khiển tiến trình. Bảng phân đoạn được bộ điều phối sử dụng để khởi tạo phần cứng thực hiện ánh xạ mỗi khi tiến trình thực thi. Chia sẻ phân đoạn được thực hiện bằng cách cho các hàng trong bảng phân 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 phân đoạn. Những phân 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 chia sẻ 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 phân 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 phân 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.

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 đổi chiều 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 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 HDH 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.

Chương 10

BỘ NHỚ ẢO

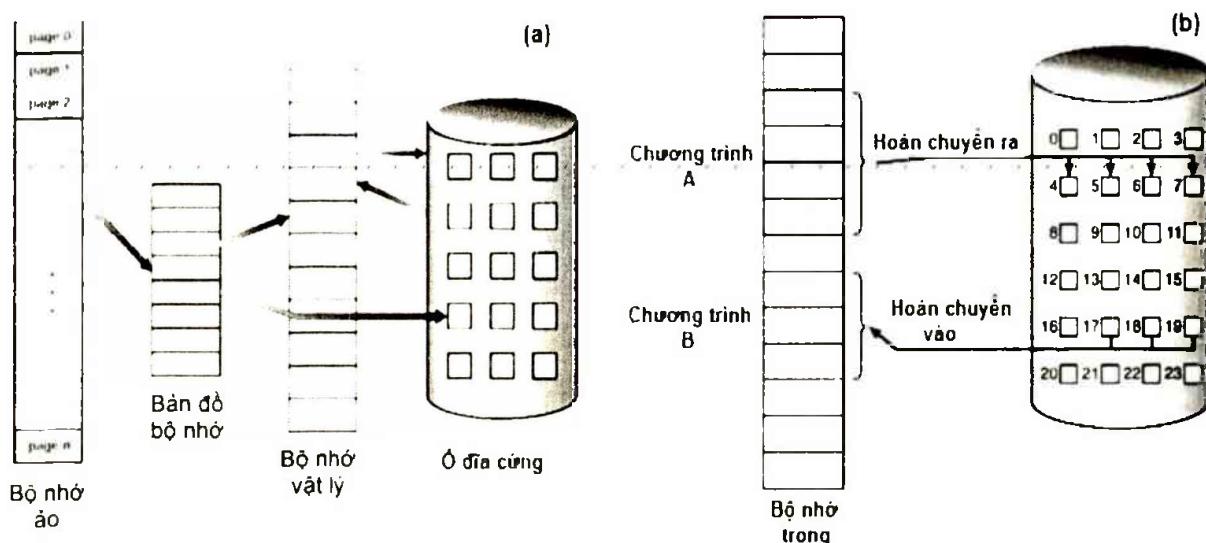
Mục tiêu của các phương pháp quản lý bộ nhớ là tải đồng thời nhiều tiến trình vào bộ nhớ. Chương 9 đã trình bày cách tải toàn bộ tiến trình vào trong bộ nhớ trước khi thực thi. Công nghệ bộ nhớ ảo được giới thiệu trong chương này lại cho phép thực thi các tiến trình không nằm trọn vẹn trong bộ nhớ. Khi 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.

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 HDH 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.



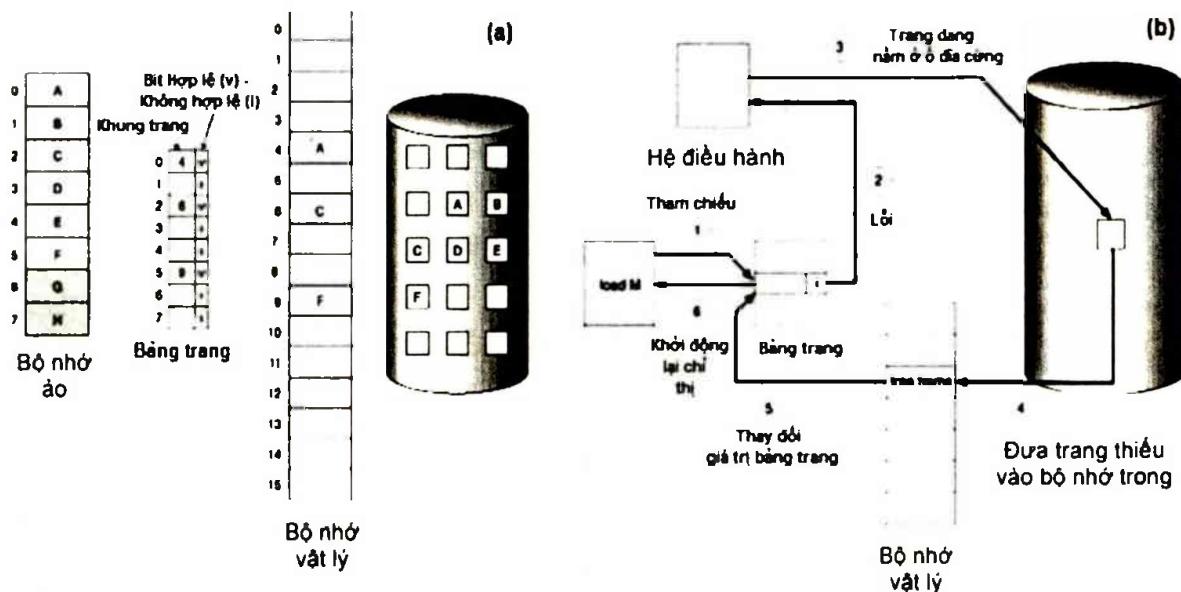
Hình 10.1. Bộ nhớ ảo và bộ nhớ vật lý

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, HDH 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. Vẫn 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 ngũ 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 thuận 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 trái 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.

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 ($0 \leq p \leq 1$). Thời gian truy cập có ích được tính như sau:

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

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 chuyên 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 trữ 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 chi tiết 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:

$$\begin{aligned}
 \text{Thời gian truy cập có ích} &= (1 - p) \times 100 + p \times (25ms) \\
 &= (1 - p) \times 100 + p \times 25.000.000 = 100 + 24.9910.990 \times p
 \end{aligned}$$

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:

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

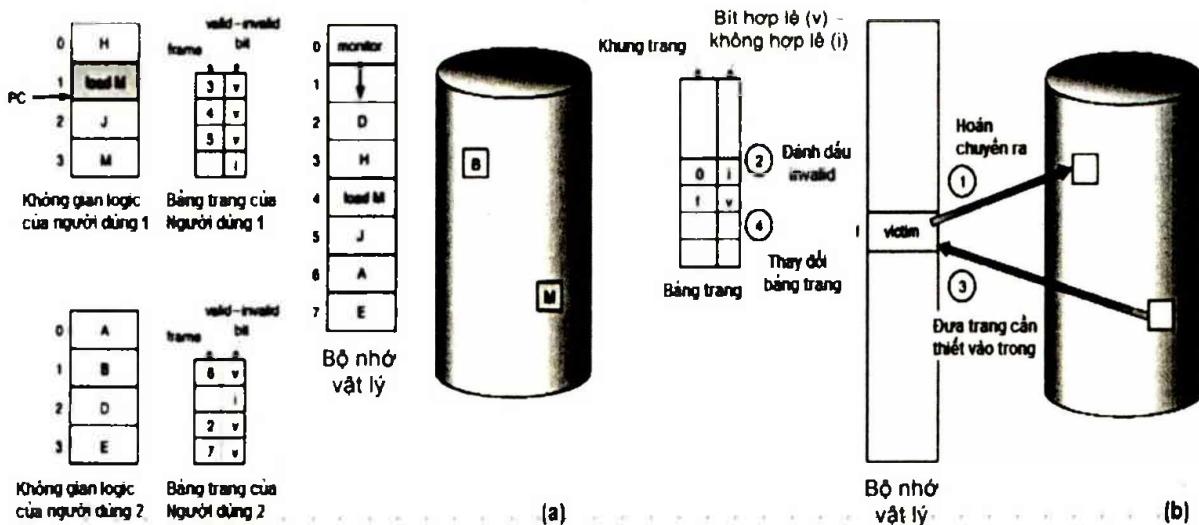
Tức là, chỉ được xuất hiện một lỗi trang trong 2500000 lần truy cập bộ nhớ. Giữ tỷ lệ lỗi trang ở mức thấp đóng vai trò cực kỳ quan trọng trong hệ thống phân trang theo yêu cầu. Nếu thời gian truy cập có ích tăng, thì tốc độ thực thi tiến trình sẽ giảm đáng kể. HĐH dành một vùng trong ổ đĩa để lưu giữ 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 chưa 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 ô 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 ô 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 ô 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 ô 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.

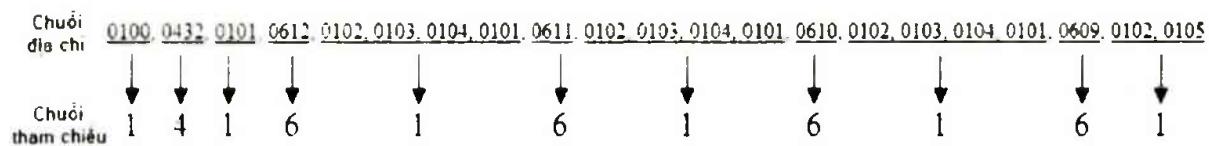
Phần sau trình bày hai vấn đề quan trọng khi triển khai phương pháp phân trang theo yêu cầu là thuật toán Cấp phát frame và thuật toán Thay thế trang. Nếu có nhiều tiến trình nằm trong bộ nhớ, HĐH phải quyết định cấp phát cho mỗi tiến trình bao nhiêu frame. Hơn nữa, khi có yêu cầu thay thế trang, HĐH 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 (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	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	0	0	0	0	0	0	4	4	4	4	3	3	
1		1	1	1	0	0	0	0	0	2	2	2	1	1	1	1	1	1	0	0	0	0	0	4	
2		2	2	2	1	1	1	1	1	3	3	3	2	2	2	2	2	2	1	1	1	1	1	1	
													3	3	3	3	3	3	2	2	2	2	2	2	

(a) Có ba Frame

(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 HDH 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	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	0	0	0	0	4	4	4	4	0	2	2	2	1	1	1	3	3	3	0	0	0	6	6	6	7
1	0	0	0	0	0	3	3	3	3	3	3	3	6	6	6	7	1	0	0	0	2	2	2	1	1	1	3	3	3	4	4	4	4
2	3	1	1	1	1	1	1	1	1	1	1	1	5	5	5	2	3	3	3	0	0	0	2	2	2	1	1	1	5	5	5	5	

(a) Thuật toán tối ưu

(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. Dỉnh 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 xắp 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 *xáp xi* thuật toán thay thế LRU.

☞ **Thuật toán Nhiều bit tham chiếu**

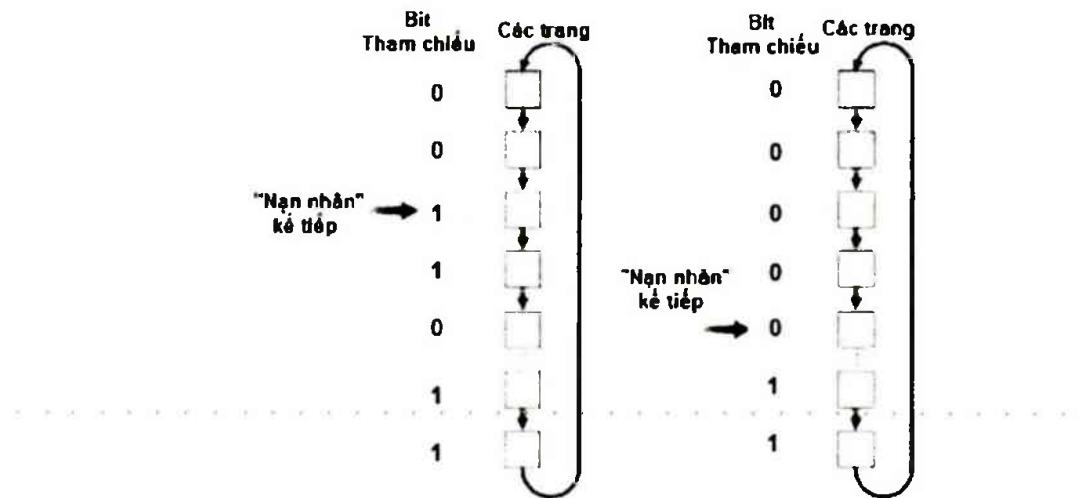
Sử dụng càng nhiều bit tham chiếu, hệ thống càng biết rõ tình trạng sử dụng của trang. Giả sử bên cạnh trường Bit tham chiếu, mỗi trang trong bộ nhớ có thêm trường Lịch sử tham chiếu 8 bit. Sau khoảng thời gian định kỳ (chẳng hạn 100ms), bộ định thời tạo ngắt để chuyển quyền điều khiển cho HDH. HDH dịch 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, HDH 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 (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, HDH 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.

☞ 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 hỗ 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 thuận 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 này 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.

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 bắt 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 đinh. 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 thuận tiện 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 độ đa chương (cho phép nhiều tiến trình chạy cùng một lúc).

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.

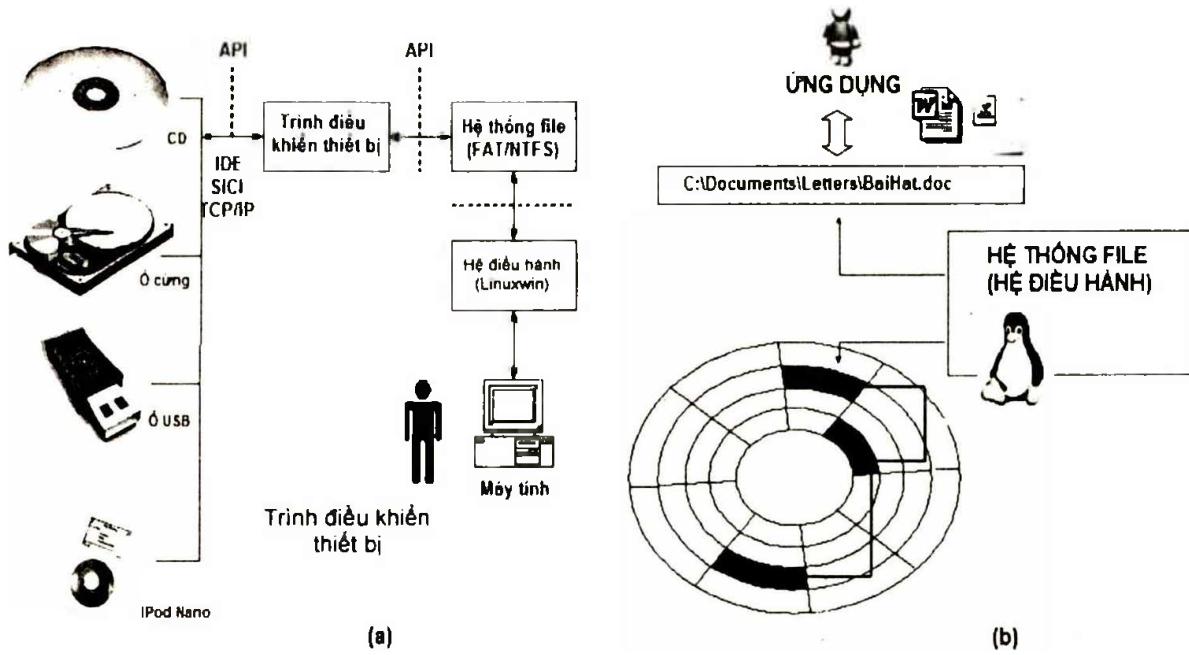
Chương 11

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.

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 thông 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 bàn 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

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 cùng 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 trả về 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) trả về 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 cùng 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

thẻ, 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

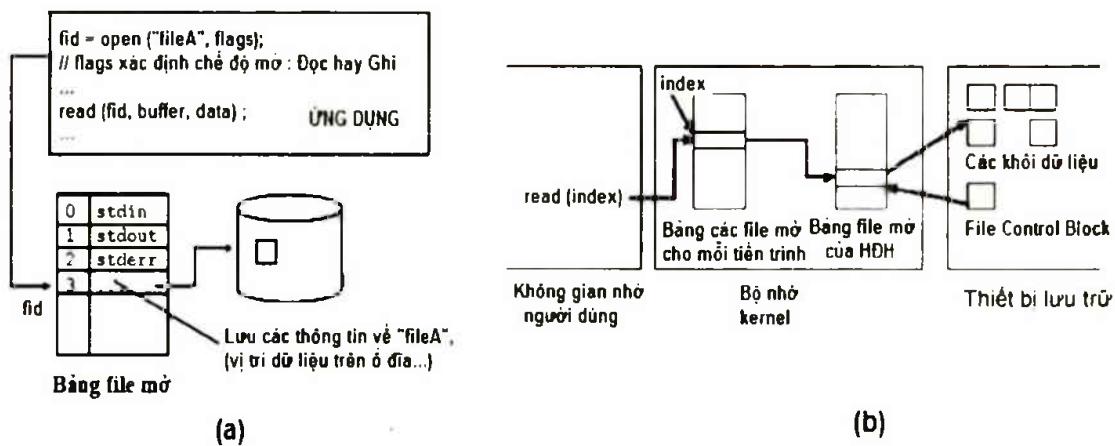
Có hai vấn đề cần quan tâm khi thiết kế hệ thống file. Đầu tiên là cách định nghĩa hệ thống file theo quan điểm người dùng (xác định thế nào là file, các thuộc tính cũng như những thao tác được phép thực hiện trên file, cấu trúc thư mục để tổ chức file). Vấn đề thứ hai là thuật toán và cấu trúc dữ liệu 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 khôi logic với khôi vật lý chứa dữ liệu tương ứng. Biết về cách thức cấp phát cũng như vị trí của khôi dữ liệu trong file, hệ thống file ánh xạ địa chỉ logic của khôi sang địa chỉ vật lý. Bên cạnh việc quản lý các khôi 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 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.

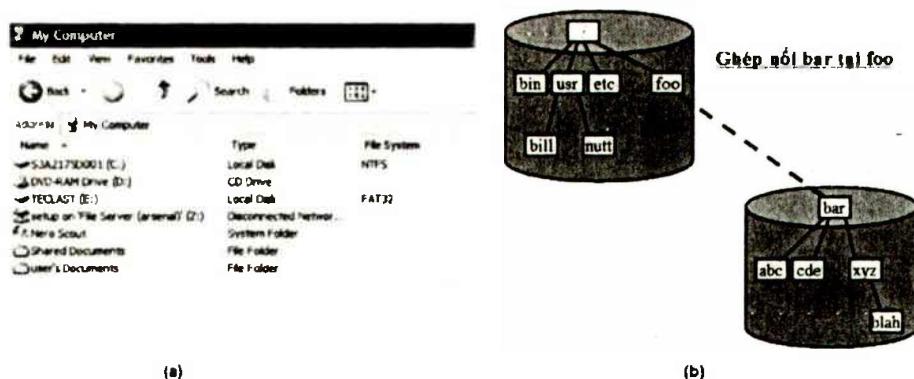
Khi tiến trình yêu cầu đọc hay ghi file, HĐH tìm kiếm trên cấu trúc thư mục để kiểm tra các tham số, định vị các khôi dữ liệu trên thiết bị, thực hiện thao tác trên từng khôi dữ liệu. Mỗi thao tác yêu cầu hệ thống tiến hành khá nhiều công việc phụ trợ. Do vậy, trước khi sử dụng, cần phải "mở" file. Khi 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 khôi 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.

HDH 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.

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, ô đĩa cứng có thể được gắn vào điểm (ô 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.



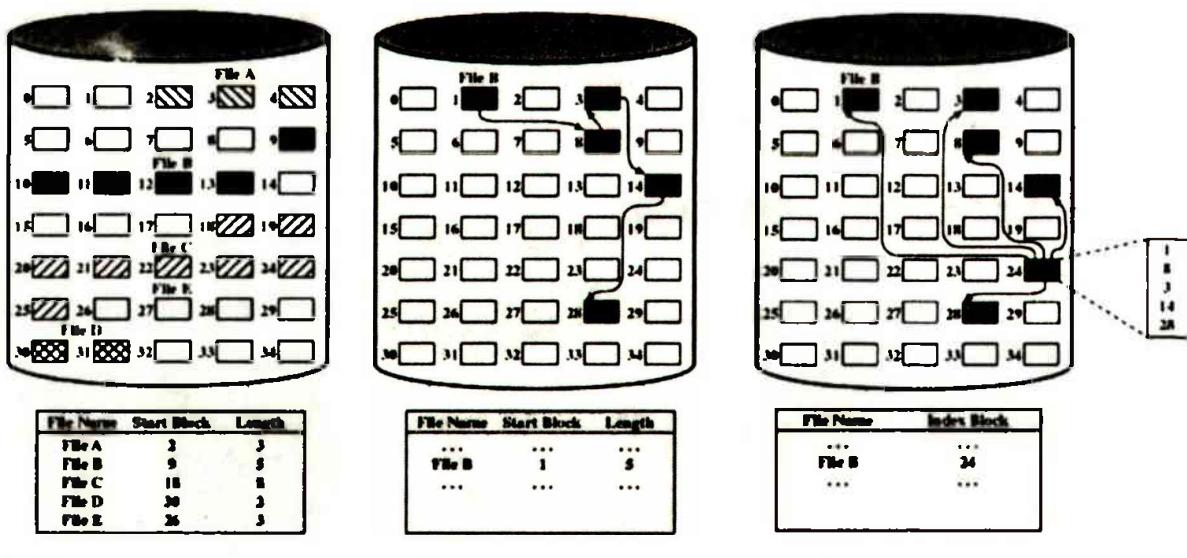
Hình 11.3. Ghép nối hệ thống file

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* (contiguous); *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 (Contiguous)

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: $b, b + 1, \dots, b + n - 1$. Trong thư mục chứa file sẽ có mục ghi lại địa chỉ khối bắt đầu và kích thước file.



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 khồi có kích thước n từ danh sách các khồi 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 khồi 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 khồi 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 mọc nối (Linked Allocation)

Trong phương pháp này, file là các khối (có thể nằm rải rác) trên ổ đĩa được mọc 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 (hình 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 "mọc" 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 mọc 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 khôi trống trong hệ thống. Tuy nhiên, cấp phát mọc 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.

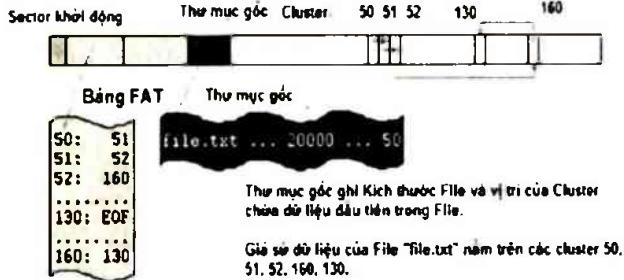
Nếu con trỏ và khối có kích thước lần lượt là 4 và 512 byte, hệ thống mất 0,78% dung lượng ổ đĩa lưu trữ con trỏ. Giải pháp khắc phục vấn đề này là nhóm các khối thành cụm và cấp phát theo đơn vị cụm. Khi đó, không gian lưu giữ con trỏ sẽ giảm đi. Phương pháp này cho phép giữ nguyên ánh xạ các khối logic 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 mốc nối, vì khi các khối trong file được mốc 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 HDH hoặc lỗi phần cứng có thể "phá hủy" con trỏ nào đó. Có thể con trỏ trả về không gian trống hay vào một file khác. Giải pháp cho vấn đề này là dùng danh sách con trỏ mốc 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 mốc 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.

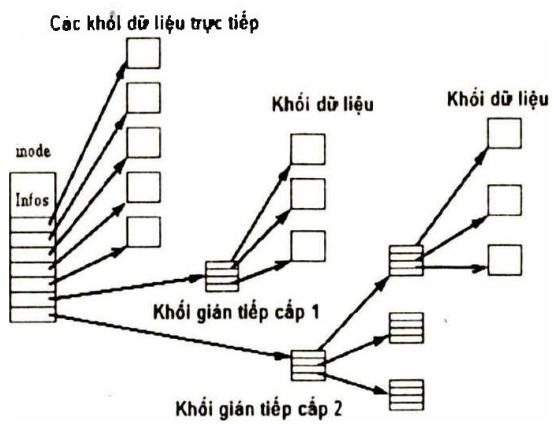
Cấp phát theo chỉ mục

Phương pháp cấp phát mốc 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.



(a) Bảng FAT trong MS-DOS



(b) inode trong UNIX

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 mọc nồi, hệ thống chỉ tồn 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 khôi 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 khôi dữ liệu trên ổ đĩa. Vì hệ thống dễ dàng lưu giữ địa chỉ khôi đầu tiên của file trong bộ nhớ, nên có thể xác định ngay lập tức vị trí của khôi thứ i (hoặc khôi tiếp theo) trên ổ đĩa để đọc trực tiếp. Với phương thức cấp phát móc nối, hệ thống cũng lưu địa chỉ của khôi kế tiếp trong bộ nhớ và có thể đọc khôi này trực tiếp. Khi đó, truy cập tuần tự không gặp vấn đề, nhưng truy cập trực tiếp đến khôi 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 mốc 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 mốc 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 khôi chỉ mục đã nằm trong bộ nhớ, hệ thống có thể truy cập trực tiếp tới các khôi dữ liệu. Tuy nhiên, điều này lại gây lãng phí bộ nhớ. Nếu thiếu không gian nhớ, đầu tiên HĐH phải đọc khôi chỉ mục, kế đó mới đến khôi dữ liệu mong muốn. Đối với chỉ mục hai mức, có thể cần tới hai lần đọc khôi chỉ mục. Nếu kích thước file quá lớn, để truy cập khôi nằm ở cuối file, có thể phải đọc tất cả các khôi 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 khôi muôn đọc.

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 khôi) 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 khôi đĩa "tự do", là các khôi chưa cấp phát cho bất kỳ file hay thư mục nào. Khi tạo mới file, HĐH lấy từ danh sách đủ số khôi trống cần thiết để cấp phát cho file. Các khôi trống sau đó bị loại bỏ khỏi danh sách không gian trống. Khi xóa file, các khôi chứa dữ liệu của file sẽ được đưa vào danh sách không gian trống.

- **Vector bit:** Trong kỹ thuật bản đồ (hay vector) bit, mỗi khôi (sector trên ổ đĩa) ứng với một bit. Bit nhận giá trị 1 nếu khôi tự do và nhận giá trị 0 nếu ngược lại. Ví dụ, nếu các khôi 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: 00111100111110001100000011100000. Phương pháp này đơn giản và HDH dễ dàng tìm được n khồi trống cần thiết trên ổ đĩa. HDH Macintosh của Apple sử dụng phương pháp vector bit để quản lý không gian trống. Để tìm khồi "tự do" đầu tiên, HDH tuần tự kiểm tra các bit trong bàn đờ. Hiệu suất hệ thống không cao trừ khi toàn bộ bàn đờ 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 khồi 512 byte sẽ cần tới một bàn đờ bit kích thước 310KB.

- **Danh sách móc nối:** HDH có thể móc nối tất cả các khồi đĩa trống với nhau, và lưu giữ con trỏ đến khồi 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. Khồi trống đầu tiên chứa con trỏ đến khồi 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 khồi, và điều này mất nhiều thao tác đọc/ghi. Tuy nhiên, hiếm khi phải duyệt qua toàn bộ danh sách các khồi. Thông thường, HDH chỉ cần một khồi trống để cấp phát cho file nào đó, vì thế khồi đầu tiên trong danh sách sẽ được sử dụng.
- **Nhóm các khồi trống:** Có thể cài tiến phương pháp trên bằng cách lưu địa chỉ n khồi tự do trong khồi tự do đầu tiên. n – 1 khồi đầu tiên sẽ thực sự là các khồi tự do. Khồi cuối cùng chứa địa chỉ n khồi tự do tiếp theo, và cứ tiếp tục như vậy. Khi đó, có thể nhanh chóng cấp phát một số lượng lớn các khồi tự do.
- **Đếm số khồi trống:** Thường HDH cấp phát hoặc thu hồi một vài khồi liền kề cùng lúc. HDH có thể sử dụng đặc điểm này bằng cách không lưu giữ danh sách toàn bộ địa chỉ n khồi tự do, mà lưu giữ địa chỉ khồi tự do đầu tiên và số lượng n khồi tự do ngay sau khồi đầu tiên. Mục trong danh sách không gian trống có hai thành phần là địa chỉ một khồi tự do trên ổ đĩa và số lượng các khồi 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. Ở 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.

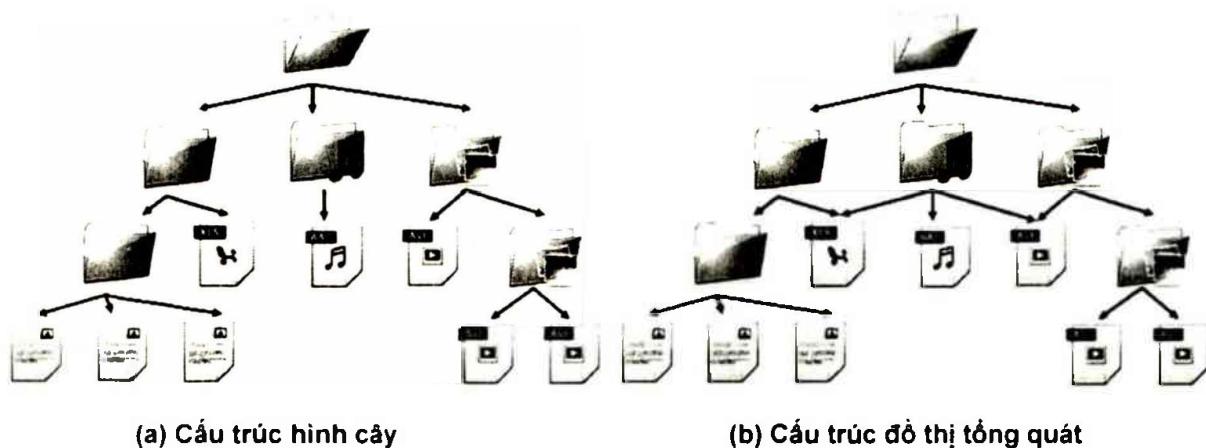
☞ *Thư mục có cấu trúc kiểu cây*

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 tùy 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 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 HDH cấu hình cố định từ trước.



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 HDH (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, HDH 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 HDH biết. Khi đưa thêm vào hệ thống một đĩa cứng, đĩa mềm, hay đĩa CD, HDH 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 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, HDH thực hiện tìm kiếm trên thư mục. Nếu file thuộc kiểu liên kết, thì HDH 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 HDH 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 HDH UNIX, các file kiểu liên kết vẫn được để lại khi xóa file gốc, và người dùng phải tự xác định đây 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 này 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 chương 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 chương trình trong đồ thị. Do đó, vấn đề phức tạp trong cấu trúc đồ thị có chương là tránh sự xuất hiện của chương khi bổ sung thêm file kiểu liên kết vào hệ thống. Làm thế nào để biết được liên kết mới có tạo nên chương hay không? Có một vài thuật toán để phát hiện chương 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 chương.

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 mốc nối để làm giảm thời gian xóa file. Tuy nhiên, tốc độ tìm kiếm file trên danh sách tuyển 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 tuyển 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 tuyển 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 mốc 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 mốc 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.

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.

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.

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 khôi ở ổ đĩa.

Bộ phận kiểm tra tính thống quán tiến hành so sánh dữ liệu trong cấu trúc thư mục với các khôi 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 điện 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 trùu 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.

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 để cấp 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

thông 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 HDH nào, hệ thống vẫn gặp nguy hại. Tuy nhiên, trong khuôn khổ quyền sách này, chúng ta chỉ xét mức độ an ninh của HDH.

An ninh trong HDH phụ thuộc rất nhiều vào nền tảng phần cứng bên dưới. Ví dụ, an ninh trong HDH MS-DOS rất kém, vì nền tảng phần cứng cài đặt HDH 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 HDH 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 HDH 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.

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 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 lỗi hỏng 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 đây là nguy cơ an ninh tiềm tàng. Người phát triển phần mềm cần cân nhắc xem chi phí khắc phục lỗi hỏng 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.

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 hàng 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 lỗi hỏng 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 mờ, 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 "**Bác Hồ Sống Mãi Trong Sự Nghiệp Của Chúng Ta**" 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.

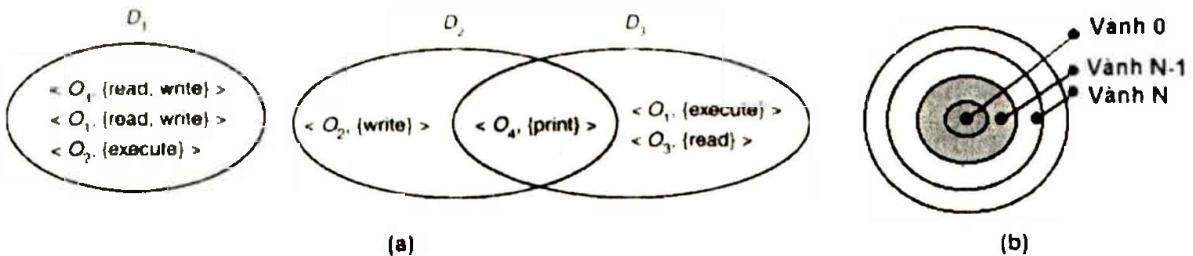
Một phương pháp khác là hệ thống chia sẻ với người dùng một bí mật (**secret**). Bí mật này không bao giờ được truyền qua môi trường không an toàn để có thể bị lộ. Hệ thống sẽ gửi cho người dùng "**Seed**" – là một số hay một dãy ký tự ngẫu nhiên. **Secret** và **Seed** được sử dụng để tính giá trị $f(\text{secret}, \text{seed})$. 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

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 $\langle O_4, \{\text{print}\} \rangle$ 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 xác 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 xác 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 HDH (thực thi trong chế độ giám sát) trước tiến trình người dùng (thực thi trong chế độ người dùng). Trong HDH 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 HDH UNIX và MULTICS.

☞ UNIX

Trong HDH 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.

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 – Đù. 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 – Đù).

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 .

object domain	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)
(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.

		<i>object</i>	F_1	F_2	F_3
		<i>domain</i>			
D_1		execute		write*	
D_2		execute	read*	execute	
D_3		execute			

(a)
(b)

		<i>object</i>	F_1	F_2	F_3
		<i>domain</i>			
D_1		execute			write*
D_2		execute	read*	execute	
D_3		execute	read		

Hình 12.3. Chuyển quyền

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) .
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) chưa 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 chưa 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 chưa 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) chưa 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 .

object domain	F_1	F_2	F_3	object domain	F_1	F_2	F_3
D_1	owner execute		write	D_1	owner execute		
D_2		read* owner	read* owner write*	D_2		owner read* write*	read* owner write*
D_3	execute			D_3		write	write

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.

object domain	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 \langle Miền bảo vệ, Đối tượng, Tập các quyền \rangle . Khi tiến trình thực thi thao tác M trên đối tượng O_j trong miền bảo vệ D_i , HĐH sẽ xác định $\langle D_i, O_j, R_k \rangle$ 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

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 \langle Miền bảo vệ, Tập các quyền \rangle . 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 $\langle D_i, R_k, O_j \rangle$, 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.

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 trữ 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).

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?

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 (qua thủ tục set-key), khi đó sẽ vô hiệu hóa khả năng trước đó (sử dụng khóa 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 trợ 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 HDH.

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 trình duyệt 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 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 ứng 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.
- Có 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)

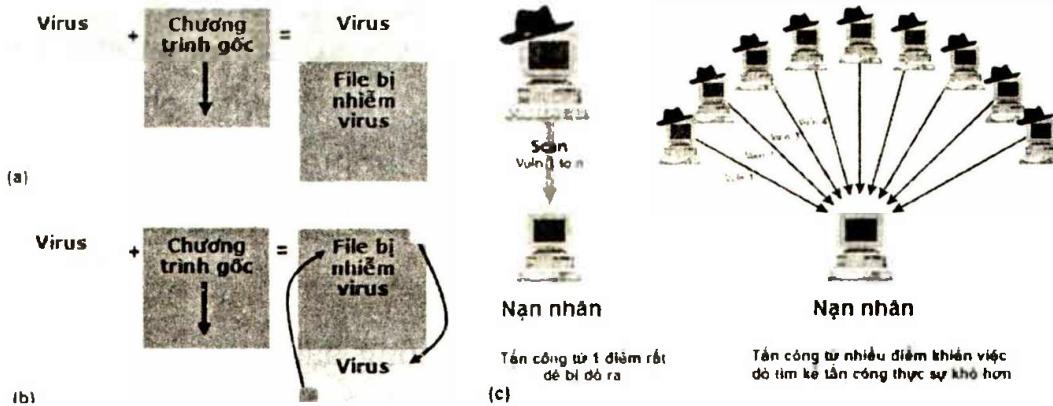
Virus mở file chương trình thực thi ra, thay đổi chương trình nguồn bằng đoạn mã virus rồi lưu lại. Khi thực thi chương trình, dù người dùng có thể nhận thấy mình chạy sai (vì mã đã bị thay đổi) thì cũng đã quá muộn.

☞ 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 khác 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 kiểm 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**

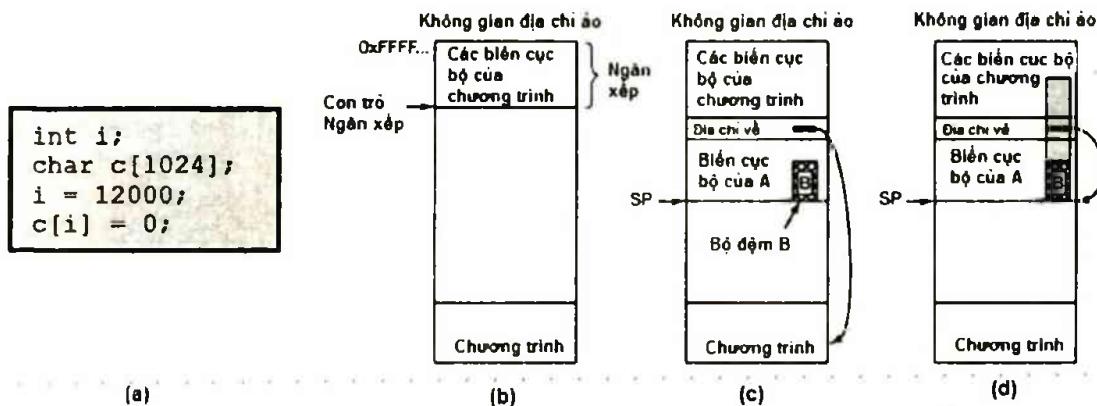
Nếu khi sử dụng một máy tính, kẻ tấn công đạt được một độ phá hoại là X thì có thể đạt được độ phá hoại $n \times X$ nếu tấn công từ n các máy nhiễm worm.

☞ **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 ché 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 lỗ 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 cắt 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 khôi ký tự này vào bộ đệm. Khôi ký tự sẽ làm tràn bộ đệm và ghi đè lên vùng bộ nhớ phía trên (Hình 12.8d). Khi đó có thể

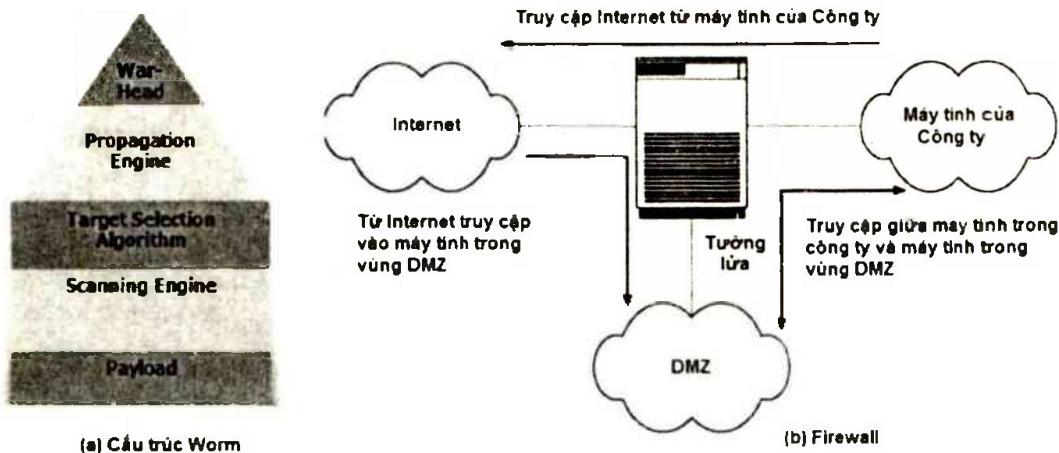
địa chỉ trả về chương trình chính cũng bị ghi đè, và kẻ tấn công có thể ghi địa chỉ chương trình của mình vào địa chỉ trả về. Sau khi thủ tục A chạy xong, thay vì quay lại chương trình chính, A chuyển quyền điều khiển cho chương trình của kẻ tấn công



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 xâm nhập. 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

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

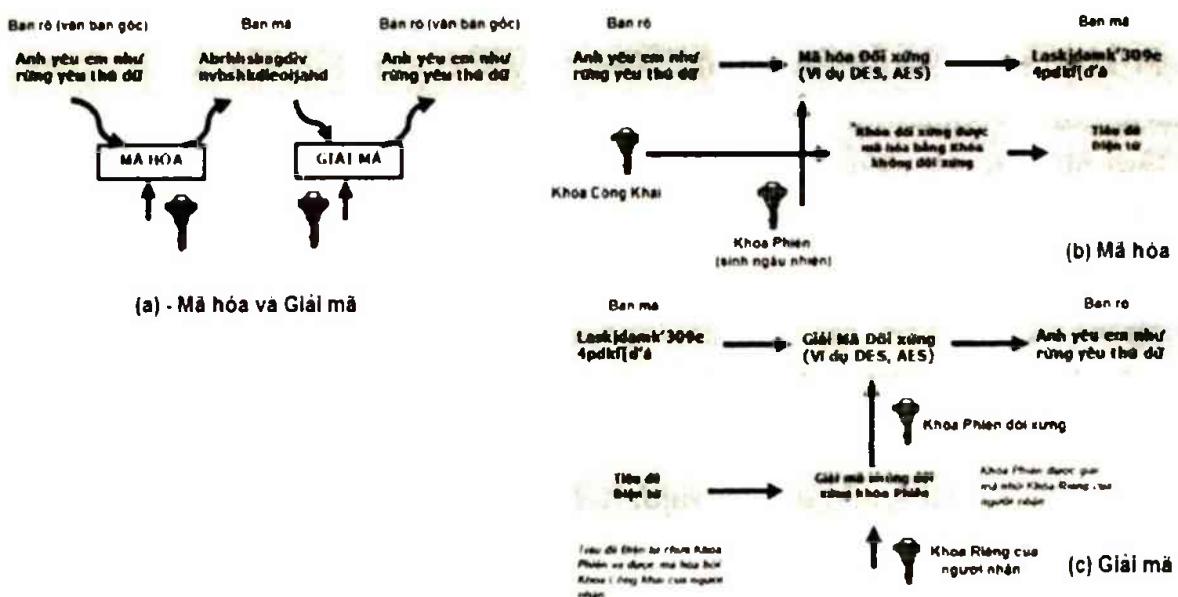
Đa phần thuật toán mã hóa sử dụng Khóa. Tham số đầu vào của thuật toán ngoài xâu cần mã hóa còn thêm khóa là số nguyên hay số thực do người dùng chọn. Khóa làm tăng tính bảo mật của thuật toán. Xét ví dụ trên, nếu thay thế ký tự bằng ký tự ngay sau nó trong bảng chữ cái, xâu "**Security**" biến thành 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:

- 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 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.

- **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.

12.7.2. Ứng dụng mật mã trong an ninh

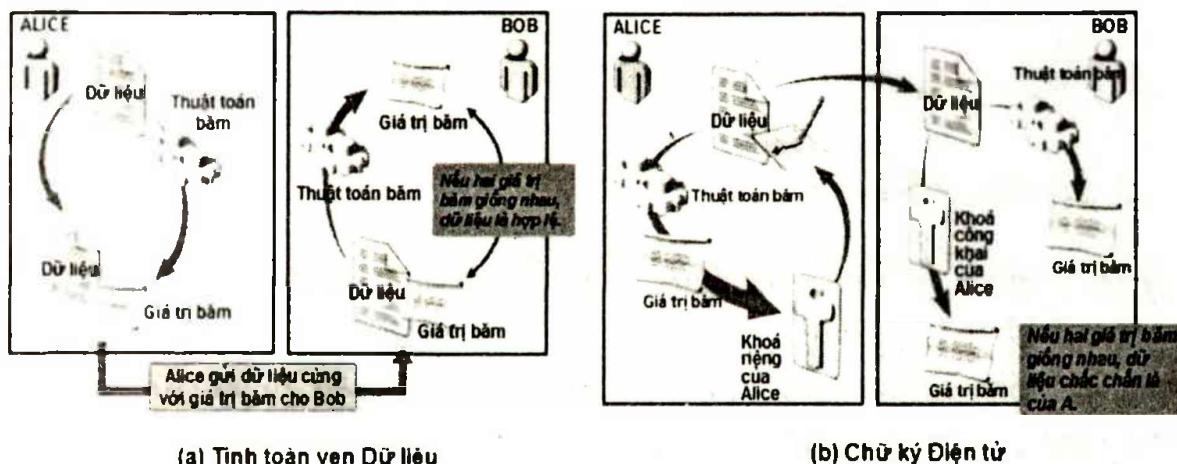
Có thể kết hợp hai phương pháp mã hóa đối xứng và không đối xứng để giải quyết một loạt các vấn đề an ninh mạng: Tính bí mật, Tính toàn vẹn dữ liệu, Chứng thực người dùng, Định danh thực thể, Tính chống tử chối.

☛ 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.



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 khăng 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 khăng 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 trùu 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 tông 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.

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).

Chịu trách nhiệm xuất bản:

Chủ tịch Hội đồng Thành viên kiêm Tổng Giám đốc NGÔ TRẦN ÁI
Tổng biên tập kiêm Phó Tổng Giám đốc NGUYỄN QUÝ THAO

Tổ chức bản thảo và chịu trách nhiệm nội dung:

Phó Tổng biên tập NGÔ ÁNH TUYẾT
Giám đốc Công ty CP Sách ĐH–DN NGÔ THỊ THANH BÌNH

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

Công ty CP Sách Đại học – Dạy nghề, Nhà xuất bản Giáo dục Việt Nam
giữ quyền công bố tác phẩm.

GIÁO TRÌNH NGUYÊN LÝ HỆ ĐIỀU HÀNH

Mã số: 7B751y2 – DAI

Số đăng kí KHXB: 16 - 2012/CXB/89 - 2050/GD.

In 700 cuốn (QĐ in số : 04), khổ 16 x 24 cm.

In tại Công ty Cổ phần in Phúc Yên.

In xong và nộp lưu chiểu tháng 2 năm 2012.



CÔNG TY CỔ PHẦN SÁCH ĐẠI HỌC - DẠY NGHỀ
HEVOBCO
25 HÀN THUYỀN - HÀ NỘI
Website : www.hevobco.com.vn



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 Vy |
| 10. Giáo trình Nhập môn mạng máy tính | Hồ Đắc Phương |

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 934994 119108



Giá: 49.000 đ