CHƯƠNG 5: QUẢN LÝ VÀO/RA

Một trong những chức năng chính của hệ điều hành là điều khiển tất cả các thiết bị vào/ra (Input/Output – I/O) của máy tính. Nó phải tạo ra các lệnh gửi tới thiết bị, giải quyết các ngắt, và xử lý lỗi. Nó cũng cung cấp một giao diện giữa các thiết bị và phần còn lại của hệ thống giúp cho việc sử dụng trở nên đơn giản và dễ dàng. Ở một mức độ nhất định, giao diện này sẽ phù hợp với mọi thiết bị (sự độc lập với thiết bị - device independence). Mã lệnh vào/ra là một phần quan trọng của hệ điều hành. Cách quản lý vào/ra của hệ điều hành chính là chủ đề của chương này.

Chương này được tổ chức như sau: Trước hết ta sẽ tìm hiểu một số nguyên lý của phần cứng vào/ra, và sau đó sẽ tìm hiểu về phần mềm vào/ra. Phần mềm vào/ra có thể có cấu trúc nhiều lớp, mỗi lớp có một nhiệm vụ riêng. Ta sẽ tìm hiểu về nhiệm vụ của từng lớp, cũng như cách thức mà chúng gắn kết với nhau.

Tiếp theo phần giới thiệu, ta sẽ xem xét một số thiết bị vào/ra cụ thể như: các ổ đĩa, đồng hồ, bàn phím, và màn hình. Với mỗi thiết bị ta sẽ tìm hiểu cả phần cứng và phần mềm. Cuối cùng sẽ là vấn đề quản lý điện năng.

5.1 CÁC NGUYÊN LÝ CỦA PHẦN CỨNG VÀO/RA

Các đối tượng khác nhau sẽ tìm hiểu phần cứng vào/ra theo những cách khác nhau. Các kỹ sư điện thì quan tâm tới các chip, dây dẫn, nguồn cấp, motor, và các thành phần vật lý khác của phần cứng. Các lập trình viên thì quan tâm tới giao diện với phần mềm – các lệnh mà phần cứng chấp nhận, các chức năng mà nó thực hiện, và các lỗi có thể xảy ra. Trong cuốn sách này ta quan tâm tới vấn đề lập trình cho các thiết bị vào/ra, chứ không phải là vấn đề thiết kế, chế tạo, hay bảo dưỡng chúng. Phạm vi nghiên cứu của chúng ta là cách thức lập trình cho phần cứng, chứ không phải các hoạt động diễn ra bên trong chúng. Tuy nhiên, việc lập trình cho các thiết bị vào/ra thường liên quan mật thiết tới các hoạt động bên trong. Ở ba mục tiếp theo ta sẽ cung cấp một số nền tảng chung về phần cứng vào/ra liên quan tới sự lập trình. Nó có thể liên quan tới một số vấn đề đã giới thiệu trong mục 1.4.

5.1.1 Các thiết bị vào/ra

Các thiết bị vào/ra có thể được chia thành hai loại như sau: **các thiết bị khối (block devices)** và **các thiết bị kí tự (character devices)**. Một thiết bị khối sẽ lưu trữ thông tin theo từng khối có kích thước cố định, mỗi khối có một địa chỉ riêng. Kích thước của một khối thường nằm trong phạm vi từ 512 byte tới 32768 bytes. Đặc điểm quan trọng của một thiết bị khối là có thể đọc hoặc ghi từng khối một cách độc lập với các khối còn lại. Đĩa là một thiết bị khối phổ biến nhất.

Nếu bạn xem xét kỹ hơn thì sẽ nhận thấy rằng ranh giới giữa các thiết bị có thể địa chỉ hoá theo khối và các thiết bị khác không được rõ ràng lắm. Mọi người đều đồng ý rằng đĩa là một thiết bị có thể địa chỉ hoá theo khối, vì dù cánh tay đĩa đang ở bất cứ vị trí nào, nó cũng có thể tìm đến một cylinder khác, và chờ ở đó cho tới khi khối dữ liệu cần thiết quay tới bên dưới đầu đọc của nó. Bây giờ ta sẽ xem xét một ổ băng từ hay dùng để sao lưu. Băng từ chứa các khối dữ liệu tuần tự. Nếu ổ băng từ nhận được một lệnh yêu cầu đọc khối thứ N, nó luôn có thể quay băng tới vị trí của khối N. Hoạt động này diễn ra tương tự như việc tìm kiếm dữ liệu trên đĩa, nhưng sẽ tốn thời gian hơn. Ngoài ra, tuỳ từng thời điểm có thể ghi hoặc không thể ghi một khối dữ liệu vào giữa băng. Thậm chí nếu có thể sử dụng băng từ như một thiết bị khối truy nhập ngẫu nhiên, thì cũng không có nghĩa nó là một thiết bị chuyên dùng theo cách đó.

Loại thiết bị vào/ra khác là thiết bị kí tự. Một thiết bị kí tự có thể gửi hoặc nhận một chuỗi các ký tự, mà không cần quan tâm tới bất cứ cấu trúc khối nào. Nó không cần địa chỉ hoá và không cần thực hiện các hoạt động tìm kiếm. Máy in, card mạng, chuột, và phần lớn các thiết bị khác không giống đĩa đều có thể coi là các thiết bị kí tự.

Cách phân loại này có vẻ chưa hoàn toàn hợp lý. Một số thiết bị không biết xếp vào loại nào. Ví dụ như đồng hồ, nó không được địa chỉ hoá theo khối, cũng không truyền hay nhận các chuỗi kí tự. Công việc của nó chỉ là tạo ra các ngắt theo những khoảng thời gian định trước. Cũng không biết xếp bộ nhớ hiển thị vào loại nào. Mặc dù vậy, cách phân loại thiết bị như trên vẫn

được dùng làm cơ sở cho một số phần mềm hệ điều hành xử lý việc vào/ra độc lập với thiết bị. Ví dụ như hệ thống file, nó chỉ xử lý các thiết bị khối trừu tượng, và để các vấn đề phụ thuộc thiết bị cho phần mềm cấp thấp hơn.

Các thiết bị vào/ra có rất nhiều tốc độ khác nhau, điều này gây khó khăn cho phần mềm khi phải làm việc với các tốc độ dữ liệu khác nhau đó. Hình 5-1 liệt kê tốc độ dữ liệu của một số thiết bị phổ biến. Hầu hết các thiết bị này đều có tốc độ được cải thiện dần theo thời gian.

Thiết bị	Tốc độ dữ liệu
Keyboard (bàn phím)	10 bytes/sec
Mouse (chuột)	100 bytes/sec
56K modem	7 KB/sec
Telephone channel (kênh thoại)	8 KB/sec
Dual ISDN lines (đường truyền ISDN kép)	16 KB/sec
Laser printer (máy in laser)	100 KB/sec
Scanner (máy quét)	400 KB/sec
Classic Ethernet (mạng Ethernet cũ)	1.25 MB/sec
USB (Universal Serial Bus)	1.5 MB/sec
Digital camcorder (máy quay kỹ thuật số)	4 MB/sec
IDE disk (đĩa cứng IDE)	5 MB/sec
40x CD-ROM	6 MB/sec
Fast Ethernet (mạng Ethernet tốc độ cao)	12.5 MB/sec
ISA bus	16.7 MB/sec
EIDE (ATA-2) disk (đĩa cứng EIDE)	16.7 MB/sec
FireWire (IEEE 1394) (công nghệ FireWire)	50 MB/sec
XGA Monitor (màn hình XGA)	60 MB/sec
SONET OC-12 network (mạng SONET OC-12)	78 MB/sec
SCSI Ultra 2 disk (đĩa cứng SCSI Ultra 2)	80 MB/sec
Gigabit Ethernet	125 MB/sec
Ultrium tape (băng từ Ultrium)	320 MB/sec
PCI bus	528 MB/sec
Sun Gigaplane XB backplane (bảng mạch nối)	20 GB/sec

Hình 5-1 Tốc độ dữ liệu của một số thiết bị điển hình.

5.1.2 Bộ điều khiển thiết bị

Các đơn vị vào/ra điển hình thường bao gồm thành phần cơ khí và thành phần điện tử. Nó được chia thành hai phần riêng nhằm tạo ra một kiến trúc tổng quát và được modul hoá. Thành phần điện tử được gọi là **bộ điều khiển thiết bị (device controller)** hoặc **bộ điều hợp (adapter)**. Trên các máy tính cá nhân, nó thường có dạng một vỉ mạch (card) có thể cắm trên các khe mở rộng. Thành phần cơ khí chính là bản thân thiết bị. (xem lại hình 1-5).

Các vỉ mạch điều khiển thường có một cổng kết nối, từ đó cáp sẽ được gắn để nối tới thiết bị. Nhiều bộ điều khiển có thể xử lý hai, bốn, hoặc thậm chí tám thiết bị cùng loại. Nếu giao diện giữa bộ điều khiển và thiết bị là một giao diện chuẩn (dựa trên tiêu chuẩn của các tổ chức ANSI, IEEE, hoặc ISO), thì các công ty khác nhau có thể chế tạo ra các bộ điều khiển và các thiết bị phù hợp với nhau. Ví dụ, nhiều công ty khác nhau có thể sản xuất ra các ổ đĩa có cùng giao diện tiêu chuẩn IDE hoặc SCSI.

Giao diện giữa bộ điều khiển và thiết bị thường là một giao diện ở cấp rất thấp. Ví dụ, một đĩa từ có thể được định dạng với 256 sector/ 1 track, mỗi sector có 512 byte. Tuy nhiên, cái thực sự được lưu trữ trên đĩa là một chuỗi các bít nối tiếp nhau, bắt đầu bằng **phần tiêu đề (preamble),** tiếp theo là 4096 bít của một sector, cuối cùng là phần mã kiểm tra (checksum), còn được gọi là

mã sửa lỗi (Error-Correcting Code - ECC). Phần tiêu đề được ghi vào khi định dạng đĩa, nó chứa số hiệu cylinder và sector, kích thước sector, và các thông tin tương tự, cũng như các thông tin để đồng bộ hoá.

Nhiệm vụ của bộ điều khiển thiết bị là chuyển đổi dãy bít nối tiếp thành một khối nhiều byte, và thực hiện việc sửa lỗi nếu cần. Đầu tiên, khối các byte này sẽ được lắp ghép lại từ nhiều bít, tại một bộ đệm bên trong bộ điều khiển. Sau khi phần checksum đã được kiểm tra, và không thấy có lỗi, nó có thể được sao chép vào bộ nhớ chính.

Bộ điều khiển của màn hình cũng hoạt động giống như một thiết bị nối tiếp cấp thấp. Nó đọc các byte chứa kí tự cần hiển thị từ bộ nhớ, và gửi các tín hiệu điều khiển tới bộ phát tia CRT để hiện chúng lên màn hình. Bộ điều khiển cũng phát ra các tín hiệu lái tia, để điều khiển các tia điện tử quét ngang và quét dọc trên màn hình. Nếu không có bộ điều khiển CRT, người lập trình hệ điều hành sẽ phải tự viết các chương trình điều khiển quét. Với bộ điều khiển sẵn có, hệ điều hành chỉ việc khởi tạo cho nó một vài tham số, như số lượng kí tự hoặc điểm ảnh trên một dòng, số lượng dòng trên màn hình, và để cho bộ điều khiển làm công việc điều khiển quét tia.

5.1.3 Ánh xạ không gian vào/ra tới bộ nhớ

Mỗi bộ điều khiển thiết bị (controller) có một vài thanh ghi dùng để liên lạc với CPU. Bằng cách ghi dữ liệu vào các thanh ghi này, hệ điều hành có thể ra lệnh cho thiết bị gửi dữ liệu, nhận dữ liệu, bật hoặc tắt thiết bị, hay các hành động khác. Nhờ đọc dữ liệu từ các thanh ghi này, hệ điều hành có thể biết được trạng thái của thiết bị, xem nó có sẵn sàng để nhận lệnh mới hay không…

Ngoài các thanh ghi điều khiển, nhiều thiết bị còn có một bộ đệm dữ liệu mà hệ điều hành có thể đọc hoặc ghi. Ví dụ, một giải pháp chung cho các máy tính để hiển thị điểm ảnh lên màn hình là sử dụng video RAM (bộ nhớ hiển thị), đây chính là một bộ đệm dữ liệu, mà các chương trình và hệ điều hành có thể ghi vào.

Một vấn đề cần quan tâm là làm thế nào mà CPU có thể liên lạc với các thanh ghi điều khiển và bộ đệm dữ liệu của thiết bị. Có hai cách sau đây. Cách thứ nhất là mỗi thanh ghi điều khiển sẽ được cấp một **địa chỉ cổng** (**I/O port number**), đó là một số nguyên 8 hoặc 16 bít. Bằng cách dùng một lệnh đặc biệt như sau:

```
IN REG,PORT
```

CPU có thể đọc dữ liệu từ thanh ghi điều khiển có địa chỉ cổng là PORT, và chứa kết quả vào thanh ghi REG bên trong CPU. Tương tự như vậy, sử dụng lệnh:

```
OUT PORT,REG
```

CPU có thể ghi dữ liệu từ thanh ghi REG của nó vào một thanh ghi điều khiển. Hầu hết các máy tính trước đây, kể cả các máy tính lớn, như IBM 360 và các thế hệ kế tiếp của nó, đều dùng cách này.

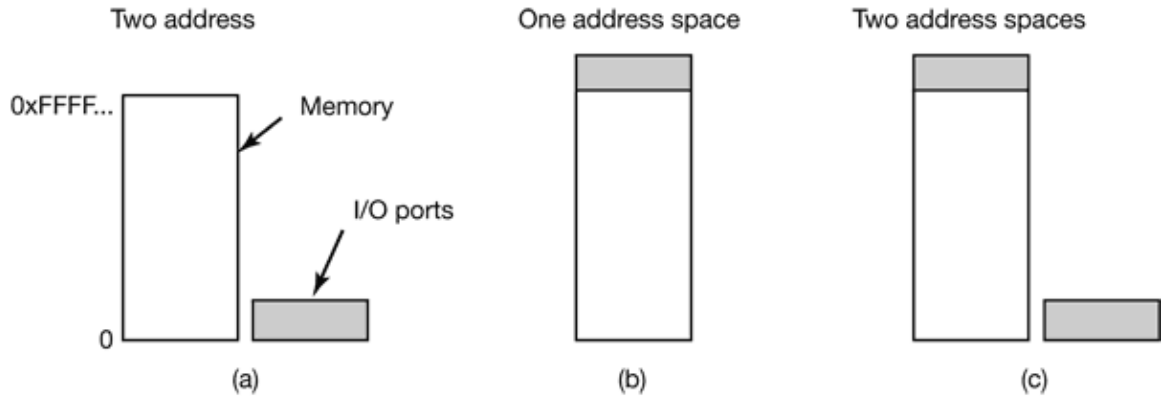
Theo cách này, không gian địa chỉ bộ nhớ và không gian địa chỉ vào/ra là hoàn toàn khác biệt, như minh hoạ ở hình 5-2(a). Các lệnh:

```
IN R0,4
```

và

```
MOV R0,4
```

là hoàn toàn khác nhau trong thiết kế này. Lệnh thứ nhất sẽ đọc nội dung của cổng vào/ra có số hiệu là 4, rồi đặt dữ liệu đọc được vào R0, trong khi lệnh thứ hai sẽ đọc nội dung của ô nhớ số 4 và đặt vào R0. Giá trị 4 trong các ví dụ này sẽ tham chiếu tới hai không gian địa chỉ không liên quan đến nhau.



Hình 5-2 (a) Tách riêng không gian địa chỉ bộ nhớ và không gian địa chỉ vào/ra. (b) Ánh xạ không gian vào/ra tới bộ nhớ. (c) Mô hình hỗn hợp.

Cách thứ hai, đã được giới thiệu trên hệ thống PDP-11, là ánh xạ tất cả các thanh ghi điều khiển vào không gian bộ nhớ, như minh hoạ ở hình 5-2(b). Mỗi thanh ghi điều khiển được cấp một địa chỉ bộ nhớ duy nhất (địa chỉ này sẽ không dùng để cấp cho bộ nhớ). Hệ thống này được gọi là **Ánh xạ cổng vào/ra tới bộ nhớ (memory-mapped I/O).** Thường thì các địa chỉ dùng để ánh xạ nằm ở phần trên của không gian địa chỉ. Một giải pháp hỗn hợp, sử dụng cả ánh xạ cổng (hoặc bộ đệm) tới bộ nhớ và tách riêng không gian cổng – bộ nhớ được minh hoạ trên hình 5-2(c). Hệ thống Pentium sử dụng kiến trúc này, với các địa chỉ từ 640 K đến 1M dành riêng cho các bộ đệm dữ liệu của thiết bị, nhằm tương thích với IBM PC, thêm vào đó nó còn có các cổng được đánh số từ 0 tới 64K.

Các hệ thống này làm việc như thế nào? Trong tất cả các trường hợp, khi CPU muốn đọc một word từ bộ nhớ hoặc từ cổng vào/ra, nó sẽ đặt địa chỉ cần truy nhập lên các đường dây địa chỉ của bus, rồi gửi tín hiệu READ (đọc) tới đường dây điều khiển của bus. Sẽ cần dùng tới một đường tín hiệu khác để xác định xem đối tượng truy nhập là không gian vào/ra hay không gian bộ nhớ. Nếu là không gian bộ nhớ, bộ nhớ sẽ gửi đáp ứng trở lại. Còn nếu là không gian vào/ra, thiết bị vào/ra sẽ gửi tín hiệu phản hồi. Nếu chỉ có không gian bộ nhớ [mô hình trên hình 5-2(b)], thì tất cả các modul bộ nhớ và các thiết bị vào/ra sẽ phải so sánh giá trị trên các đường dây địa chỉ với danh sách địa chỉ mà nó phục vụ. Nếu địa chỉ cần truy nhập nằm trong danh sách của nó, nó sẽ gửi một tín hiệu phản hồi. Do không có địa chỉ nào được cấp phát chung cho cả bộ nhớ và thiết bị vào/ra, nên sẽ không xảy ra xung đột.

Cả hai cách địa chỉ hoá cho controller đều có những mặt mạnh và mặt yếu riêng. Bây giờ ta sẽ tìm hiểu các ưu điểm của phương pháp ánh xạ cổng vào/ra tới bộ nhớ. Thứ nhất, nếu cần tới các lệnh vào/ra đặc biệt để đọc hay ghi vào các thanh ghi điều khiển thiết bị, thì việc truy nhập tới chúng sẽ phải dùng đến các lệnh assembly, do không có cách nào để thi hành các lệnh IN và OUT trong C hay C++. Gọi một thủ tục để thực hiện điều đó sẽ làm tăng chi phí cho việc điều khiển vào/ra. Ngược lại, nếu áp dụng phương pháp ánh xạ cổng vào/ra tới bộ nhớ, các thanh ghi điều khiển thiết bị cũng sẽ giống như các biến trong bộ nhớ, và có thể truy nhập từ chương trình C giống như các biến khác. Nhờ việc ánh xạ vào bộ nhớ, các chương trình điều khiển thiết bị vào/ra có thể được viết hoàn toàn bằng C. Còn nếu không thì sẽ phải sử dụng thêm các mã lệnh assembly.

Thứ hai, khi ánh xạ cổng vào/ra tới bộ nhớ, sẽ không cần áp dụng kỹ thuật bảo vệ đặc biệt nào để tách các tiến trình của người dùng ra khỏi công việc thực hiện vào/ra. Tất cả các hệ điều hành đều phải làm điều đó, nhằm hạn chế việc đặt phần không gian địa chỉ của các thanh ghi điều khiển vào không gian địa chỉ ảo của người dùng. Tốt hơn, nếu mỗi thiết bị có các thanh ghi điều khiển nằm trên một trang nào đó trong không gian địa chỉ, thì hệ điều hành có thể cấp cho một tiến trình của người dùng quyền điều khiển các thiết bị cụ thể, mà những người khác không có, bằng cách đơn giản là đặt các trang nhớ cần thiết vào bảng trang của nó. Giải pháp này có thể cho phép các chương trình điều khiển thiết bị khác nhau được đặt vào các không gian địa chỉ khác

nhau, điều này không những giúp giảm kích thước của kernel, mà còn giúp tránh xung đột giữa các chương trình điều khiển thiết bị.

Thứ ba, với phương pháp ánh xạ cổng vào/ra tới bộ nhớ, mọi lệnh có thể tham chiếu tới bộ nhớ cũng đều có thể tham chiếu tới các thanh ghi. Ví dụ, nếu có một lệnh TEST, để kiểm tra một word của bộ nhớ có bằng 0 hay không, có thể dùng nó để kiểm tra xem một thanh ghi điều khiển có bằng 0 hay không (điều này có thể báo hiệu một thiết bị đang rỗi, và có thể gửi một lệnh mới tới đó). Mã lệnh assembly minh hoạ cho trường hợp này như sau:

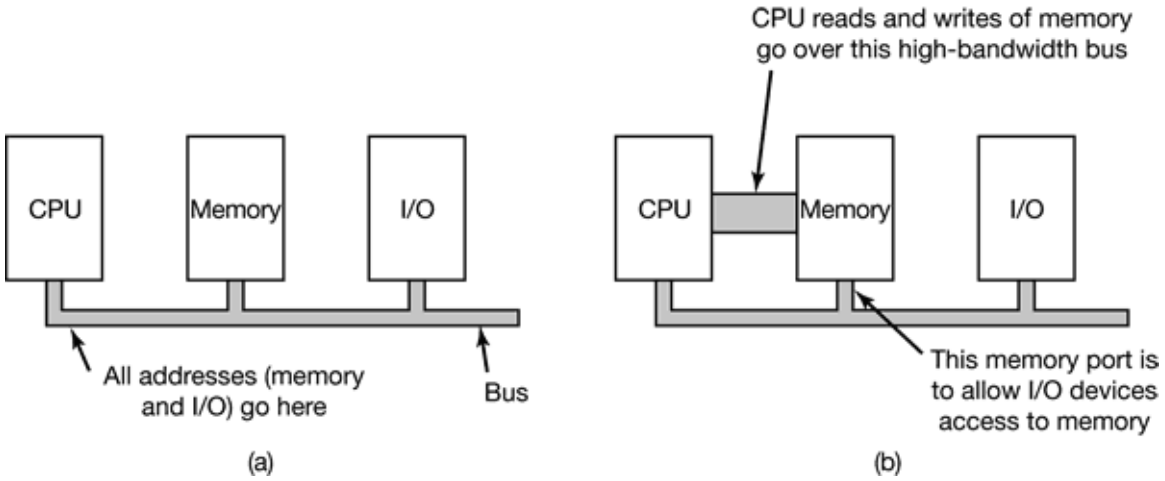
```
LOOP:   TEST PORT 4        // Kiểm tra cổng 4 có bằng 0 hay không
        BEQ READY          // Nếu bằng 0 thì sẵn sàng
        BRANCH LOOP        // Nếu khác 0 thì tiếp tục kiểm tra
READY:
```

Nếu không sử dụng ánh xạ cổng vào/ra tới bộ nhớ, trước tiên thanh ghi điều khiển phải được đọc vào trong CPU, rồi tiến hành kiểm tra, sẽ cần tới hai lệnh chứ không phải một. Trong trường hợp của vòng lặp trên, sẽ phải dùng thêm một lệnh thứ tư, điều đó sẽ làm chậm tốc độ của hệ thống khi muốn tìm một thiết bị còn rỗi.

Khi thiết kế máy tính, tất cả các yếu tố đều cần được cân bằng, và trong trường hợp này cũng vậy. Sự ánh xạ không gian vào/ra tới bộ nhớ cũng có những nhược điểm. Thứ nhất, hầu hết các máy tính ngày nay đều có các bộ nhớ cache. Việc đọc một thanh ghi điều khiển thiết bị vào cache sẽ rất tai hại. Xét vòng lặp bằng mã assembly nói trên trong trường hợp có mặt cache. Tham chiếu đầu tiên tới cổng 4 sẽ lưu nội dung của nó vào cache. Các tham chiếu tiếp theo sẽ chỉ lấy giá trị từ cache, và thậm chí không hỏi gì tới thiết bị. Sau đó, khi thiết bị đã sẵn sàng, phần mềm cũng không có cách nào phát hiện ra điều đó. Và như vậy vòng lặp sẽ không bao giờ ngừng.

Để tránh tình huống nói trên, phần cứng sẽ phải được trang bị thêm tính năng để có thể lựa chọn và tắt các bộ nhớ cache. Tính năng này sẽ làm cho cả phần cứng và hệ điều hành trở nên phức tạp hơn.

Thứ hai, nếu chỉ có một không gian địa chỉ, thì tất cả các modul bộ nhớ và các thiết bị vào/ra đều phải kiểm tra tất cả các tham chiếu tới bộ nhớ, để xác định đối tượng cần đáp ứng. Nếu máy tính chỉ có một bus, như trên hình 5-3(a), thì việc để tất cả các bộ phận tiến hành kiểm tra tới tất cả các địa chỉ không phải là công việc khó khăn.



Hình 5-3 (a) Kiến trúc đơn bus. (b) Kiên trúc bộ nhớ dùng bus kép.

Tuy nhiên, khuynh hướng của các máy tính cá nhân hiện nay là sử dụng một bus riêng có tốc độ cao để giao tiếp với bộ nhớ, như minh hoạ ở hình 5-3(b), điều này cũng có thể tìm thấy trên các máy tính lớn. Bus này đã giúp tăng cường hiệu suất của bộ nhớ, và không gây hại gì tới các thiết bị vào/ra tốc độ chậm. Các hệ thống Pentium thậm chí có tới ba loại bus ngoài (gồm bus dùng cho bộ nhớ, PCI, và ISA), như minh hoạ trên hình 1-11.

Vấn đề rắc rối khi sử dụng một bus dành riêng cho bộ nhớ trên các máy tính dùng phương pháp ánh xạ nói trên là: các thiết bị vào/ra không có cách nào để nhìn thấy được địa chỉ của bộ nhớ khi chúng được truyền trên bus riêng, nên chúng không thể gửi phản hồi lại được. Thêm nữa, sẽ phải cần tới các tiêu chuẩn đặc biệt mới có thể thực hiện được sự ánh xạ này trong hệ thống có nhiều bus. Một giải pháp là tiến hành gửi tất cả các yêu cầu tham chiếu tới bộ nhớ. Nếu bộ nhớ không phản hồi được, thì CPU sẽ thử trên các bus khác. Thiết kế này có thể hoạt động được, nhưng sẽ đòi hỏi độ phức tạp cao của phần cứng.

Giải pháp thứ hai là đặt một thiết bị chờ trên bus bộ nhớ để kiểm tra tất cả các địa chỉ xuất hiện tại đó, xem chúng có liên quan tới thiết bị vào/ra không. Vấn đề ở đây là các thiết bị vào/ra có thể sẽ không xử lý được các yêu cầu với tốc độ cao như của bộ nhớ.

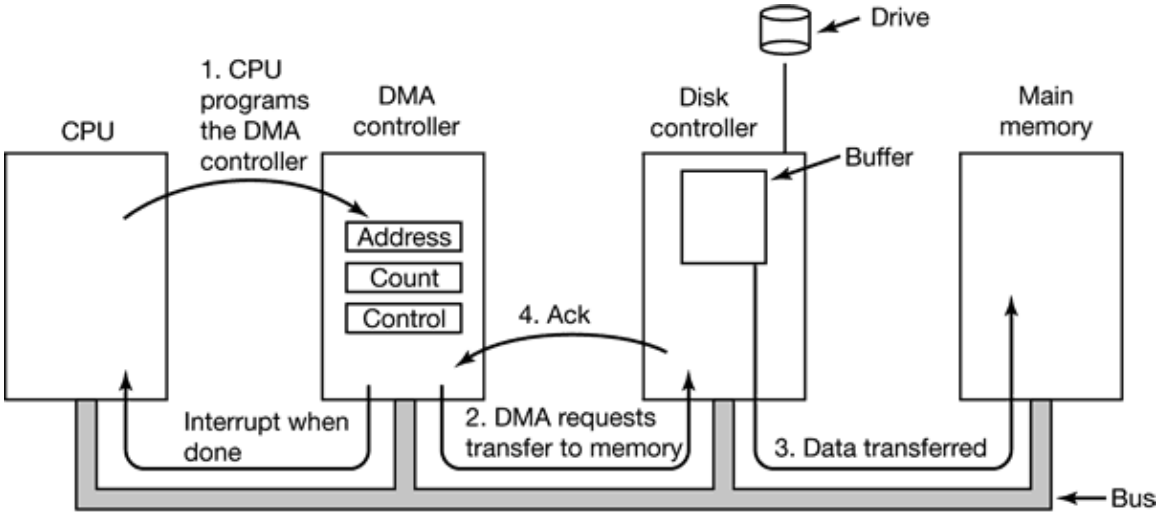
Giải pháp thứ ba, đã được sử dụng trên hệ thống Pentium ở hình 1-11, là tiến hành lọc các địa chỉ bằng một chip cầu nối tới PCI. Chip này chứa một số thanh ghi được nạp khi khởi động. Ví dụ, dải nhớ từ 640K tới 1M có thể được đánh dấu là không dùng cho bộ nhớ. Các địa chỉ trong phạm vi được đánh dấu sẽ được gửi tới PCI bus chứ không gửi tới bộ nhớ. Nhược điểm của giải pháp này là tại thời điểm khởi động sẽ phải xác định được địa chỉ nào không phải là địa chỉ bộ nhớ. Do mỗi giải pháp đều có những ưu và nhược điểm, nên việc cân nhắc, lựa chọn để cân bằng các yếu tố là điều không thể tránh được.

5.1.4 Truy nhập trực tiếp bộ nhớ (Direct Memory Access - DMA)

Dù CPU có hỗ trợ hay không hỗ trợ khả năng ánh xạ không gian vào/ra tới bộ nhớ, thì nó vẫn cần địa chỉ hoá các controller để có thể trao đổi dữ liệu với chúng. CPU có thể yêu cầu dữ liệu từ các controller từng byte một, nhưng điều đó sẽ làm lãng phí thời gian CPU, nên người ta thường áp dụng một giải pháp, gọi là **Truy nhập trực tiếp bộ nhớ - DMA (Direct Memory Access).** Hệ điều hành chỉ có thể sử dụng DMA nếu phần cứng được trang bị bộ điều khiển DMA (DMA controller), mà hầu hết các hệ thống đều có. Đôi khi controller này được tích hợp luôn vào bộ điều khiển đĩa hoặc các controller khác, nhưng một thiết kế như vậy sẽ đòi hỏi phải có các bộ điều khiển DMA riêng cho từng thiết bị. Phổ biến hơn, người ta sử dụng một bộ điều khiển DMA (ví dụ, nằm trên bảng mạch chính) để dùng chung cho nhiều thiết bị.

Mỗi khi được định vị, bộ điều khiển DMA sẽ truy nhập tới một bus hệ thống độc lập với CPU, như minh hoạ ở hình 5-4. Nó có chứa một vài thanh ghi mà CPU có thể đọc hoặc ghi, bao gồm thanh ghi địa chỉ bộ nhớ, thanh ghi đếm số byte, cùng một hoặc nhiều thanh ghi điều khiển. Các thanh khi điều khiển sẽ xác định cổng vào/ra cần sử dụng, hướng truyền dữ liệu (đọc từ thiết bị hay ghi vào thiết bị), đơn vị truyền (truyền từng byte hay từng word), và số lượng byte cần truyền trong một lần.

Để giải thích cách thức làm việc của DMA, trước hết ta sẽ tìm hiểu cách đọc đĩa khi không có DMA. Đầu tiên, bộ điều khiển đĩa sẽ đọc một khối dữ liệu (một hoặc nhiều sector) từ đĩa, tuần tự từng bít một, cho tới khi toàn bộ khối được đọc vào bộ đệm bên trong bộ điều khiển. Tiếp theo, nó sẽ tính toán checksum để kiểm tra xem có lỗi truyền dữ liệu không. Sau đó bộ điều khiển sẽ phát sinh một ngắt. Khi hệ điều hành hoạt động, nó có thể đọc khối dữ liệu từ bộ đệm trên bộ điều khiển đĩa (từng byte hoặc từng word) bằng một vòng lặp, mỗi vòng sẽ đọc một byte hoặc một word từ thanh ghi thiết bị trên bộ điều khiển, và cất nó vào bộ nhớ chính.



Hình 5-4 Hoạt động của DMA.

Khi sử dụng DMA, quá trình sẽ không giống như vậy. Đầu tiên CPU sẽ lập trình cho DMA controller bằng cách đặt thông tin vào các thanh ghi trong controller, để nó biết được đối tượng sẽ trao đổi dữ liệu (bước 1 trên hình 5-4). Nó cũng gửi một lệnh tới bộ điều khiển đĩa, yêu cầu đọc dữ liệu từ đĩa và cất vào bộ đệm trong bộ điều khiển, rồi kiểm tra checksum. Khi đã có dữ liệu hợp lệ trong bộ đệm, DMA sẽ bắt đầu.

DMA controller khởi đầu quá trình truyền dữ liệu bằng cách gửi một yêu cầu đọc dữ liệu qua bus tới bộ điều khiển đĩa (bước 2). Yêu cầu này cũng giống như các yêu cầu đọc khác, và bộ điều khiển đĩa sẽ không biết (và cũng không quan tâm) là nó đến từ CPU hay DMA controller. Thường thì địa chỉ bộ nhớ nhận dữ liệu sẽ được đặt vào các đường dây địa chỉ trên bus, nên khi bộ điều khiển đĩa lấy được word tiếp theo từ bộ đệm, nó cũng sẽ biết được nơi cần ghi dữ liệu đến. Việc ghi dữ liệu vào bộ nhớ diễn ra trên một chu kỳ sử dụng bus chuẩn khác (bước 3). Khi quá trình ghi hoàn thành, bộ điều khiển đĩa sẽ gửi một tín hiệu Ack (báo hiệu thành công) qua bus tới DMA controller (bước 4). DMA controller sẽ tăng địa chỉ của ô nhớ và giảm bộ đếm số byte. Nếu bộ đếm byte vẫn lớn hơn 0, thì các bước từ 2 tới 4 sẽ được lặp lại cho tới khi bộ đếm bằng 0. Lúc đó, DMA controller sẽ gửi tín hiệu ngắt tới CPU để thông báo quá trình truyền dữ liệu đã hoàn tất. Khi hệ điều hành hoạt động, nó sẽ không phải sao chép dữ liệu từ bộ đệm vào bộ nhớ nữa, vì dữ liệu đã ở đó.

Các DMA controller có rất nhiều khác biệt trong cấu trúc của nó. DMA controller đơn giản nhất có thể xử lý được một lần truyền tại một thời điểm, như vừa mô tả ở trên. Các controller phức tạp hơn có thể được lập trình để xử lý nhiều lần truyền đồng thời. Trong các controller đó sẽ có nhiều nhóm thanh ghi, mỗi nhóm dùng cho một kênh truyền. CPU sẽ khởi đầu bằng cách nạp vào mỗi nhóm thanh ghi một bộ tham số liên quan tới quá trình truyền của nó. Mỗi kênh truyền phải sử dụng một bộ điều khiển thiết bị riêng. Cứ sau mỗi word được truyền (từ bước 2 tới bước 4 trên hình 5-4), DMA controller sẽ quyết định thiết bị nào sẽ được phục vụ tiếp theo. Nó có thể sử dụng giải thuật round-robin, hoặc cấp các mức ưu tiên khác nhau cho các thiết bị. Nhiều yêu cầu tới các bộ điều khiển thiết bị khác nhau có thể bị treo, do sợ gây ra sự nhầm lẫn giữa các tín hiệu Ack. Thường thì mỗi kênh DMA sẽ sử dụng một đường tín hiệu Ack riêng trên bus vì lý do này.

Nhiều bus có thể hoạt động ở hai chế độ: chế độ truyền từng word và chế độ truyền khối. Một số DMA controller cũng có thể hoạt động ở hai chế độ đó. Với chế độ thứ nhất, quá trình hoạt động diễn ra như đã mô tả ở trên: DMA controller yêu cầu truyền một word và thực hiện điều đó. Nếu CPU cũng muốn sử dụng bus, nó sẽ phải chờ. Kỹ thuật này được gọi là **đánh cắp chu kỳ (cycle stealing)** vì bộ điều khiển thiết bị sẽ thỉnh thoảng "lấy trộm" một vài chu kỳ sử dụng bus của CPU, làm nó chậm đi một chút. Còn ở chế độ truyền khối, DMA controller sẽ thông báo với thiết bị để chiếm lĩnh bus, rồi tiến hành truyền toàn bộ khối dữ liệu, sau đó giải phóng bus. Cách này được gọi là **chế độ truyền loạt (burst mode)**. Nó có hiệu quả cao hơn kỹ thuật đánh cắp chu

kỳ, vì nó chiếm dụng toàn bộ bus trong suốt thời gian truyền, và có thể truyền được nhiều word. Nhược điểm của nó là có thể sẽ bắt CPU và các thiết bị khác phải chờ lâu, nếu như khối dữ liệu có kích thước lớn.

Mô hình ta vừa thảo luận đôi khi được gọi là **chế độ fly-by**, DMA controller sẽ thông báo với các bộ điều khiển thiết bị để truyền dữ liệu trực tiếp tới bộ nhớ chính. Có một chế độ khác mà một số DMA controller sử dụng, trong đó bộ điều khiển thiết bị sẽ gửi word dữ liệu tới DMA controller, DMA controller sẽ tạo ra một yêu cầu bus thứ hai để ghi word dữ liệu vào nơi cần đến. Giải pháp này đòi hỏi có thêm chu kỳ bus cho mỗi word truyền đi, nhưng lại mềm dẻo hơn vì nó cũng có thể thực hiện việc sao chép trực tiếp dữ liệu từ thiết bị này tới thiết bị khác, và thậm chí từ vùng nhớ này tới vùng nhớ khác (bằng cách gửi một yêu cầu đọc tới bộ nhớ, rồi gửi một yêu cầu ghi tới bộ nhớ tại một địa chỉ khác).

Hầu hết các DMA controller sử dụng các địa chỉ bộ nhớ vật lý cho việc truyền dữ liệu. Sử dụng địa chỉ vật lý đòi hỏi hệ điều hành phải chuyển đổi địa chỉ ảo của bộ đệm dự kiến sang địa chỉ vật lý, và ghi địa chỉ vật lý này vào thanh ghi địa chỉ của DMA controller. Có một giải pháp khác mà một vài DMA controller sử dụng là ghi địa chỉ ảo vào DMA controller. Sau đó DMA controller phải dùng MMU để chuyển địa chỉ ảo này thành địa chỉ vật lý. Chỉ khi nào MMU là một phần của bộ nhớ (có thể, nhưng hiếm gặp) chứ không phải là một bộ phận của CPU, thì mới có thể đặt trực tiếp các địa chỉ ảo lên bus.

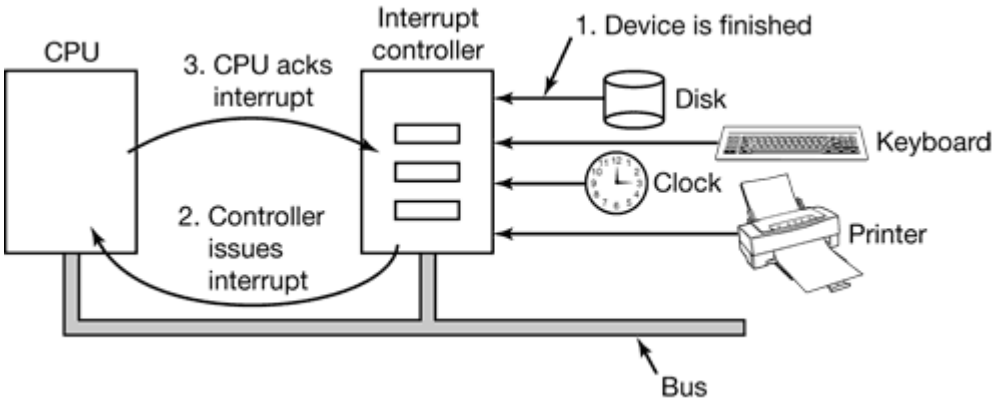
Ở trên ta đã đã đề cập tới việc đọc dữ liệu từ đĩa và cất vào bộ đệm trên bộ điều khiển đĩa trước khi DMA bắt đầu. Bạn có thể thắc mắc tại sao bộ điều khiển không chứa dữ liệu vào luôn bộ nhớ ngay sau khi đọc nó từ đĩa. Nói cách khác, tại sao lại cần tới một bộ đệm trong bộ điều khiển? Có hai lý do: Thứ nhất, khi cất dữ liệu vào bộ đệm, bộ điều khiển đĩa có thể kiểm tra checksum trước khi truyền. Nếu checksum bị sai, một tín hiệu lỗi sẽ phát sinh, và quá trình truyền dữ liệu sẽ không diễn ra.

Lý do thứ hai là khi bắt đầu truyền dữ liệu từ đĩa, các bít được truyền đi với một tốc độ không đổi, không cần biết bộ điều khiển đĩa có sẵn sàng nhận hay không. Nếu bộ điều khiển cứ cố ghi dữ liệu trực tiếp vào bộ nhớ, nó sẽ phải chiếm dụng bus hệ thống cho mỗi word được truyền. Nếu bus đang bận vì có một thiết bị khác đang sử dụng (ví dụ, bus đang trong chế độ truyền loạt), thì bộ điều khiển đĩa sẽ phải chờ. Nếu word dữ liệu tiếp theo được truyền tới bộ điều khiển khi word trước đó chưa kịp ghi vào bộ nhớ, bộ điều khiển sẽ phải lưu nó vào đâu đó. Nếu bus liên tục bận, bộ điều khiển đĩa có thể sẽ phải kết thúc việc ghi dữ liệu, và thực hiện các công việc quản lý khác. Còn khi sử dụng bộ đệm, sẽ không cần tới bus cho tới khi DMA bắt đầu, nên thiết kế của bộ điều khiển sẽ đơn giản hơn nhiều, vì việc truyền dữ liệu trực tiếp tới bộ nhớ sẽ không bị giới hạn về mặt thời gian. (Thực ra một số bộ điều khiển cũ đã làm như vậy vì chúng chỉ chuyển tới bộ nhớ một lượng dữ liệu rất nhỏ, nhưng khi bus bận liên tục thì quá trình truyền phải bị dừng, và gây ra lỗi.)

Không phải tất cả các máy tính đều sử dụng DMA. Lý do là vì CPU chính thường có tốc độ lớn hơn nhiều so với DMA controller, và có thể thực hiện công việc nhanh hơn (trong trường hợp sự hạn chế tốc độ không phải do thiết bị vào/ra). Nếu không có công việc gì để làm, thì việc bắt CPU (có tốc độ cao) phải ngồi chờ DMA controller (có tốc độ thấp) thực hiện vào/ra quả là điều ngớ ngẩn. Ngoài ra, việc loại bỏ DMA và bắt CPU làm tất cả mọi việc cũng sẽ giúp tiết kiệm được chi phí, điều này rất quan trọng đối với các dòng máy tính giá thấp.

5.1.5 Nhắc lại về các ngắt

Ta đã giới thiệu sơ qua về các ngắt trong mục 1.4.3, nhưng vẫn có điều cần phải nói thêm. Trong một hệ thống máy tính cá nhân điển hình, cấu trúc ngắt có thể được mô tả như trên hình 5-5. Ở cấp độ phần cứng, các ngắt làm việc như sau: Khi một thiết bị vào/ra kết thúc công việc, nó sẽ tạo ra một ngắt (giả sử các ngắt đã được bật bởi hệ điều hành). Thiết bị làm điều đó bằng cách gửi một tín hiệu vào bus mà nó được cấp. Tín hiệu này sẽ được tiếp nhận bởi chip điều khiển ngắt trên bảng mạch chính, chip này sẽ quyết định điều phải làm.



Hình 5-5 Quá trình xảy ra ngắt. Các thiết bị và bộ điều khiển ngắt (interrupt controller) kết nối với nhau thông qua các đường tín hiệu ngắt trên bus chứ không phải là đường dây riêng.

Bộ điều khiển ngắt sẽ xử lý ngắt ngay khi thiết bị gửi ngắt tới (nếu nó không bận xử lý các ngắt khác, và không có ngắt nào đang phải chờ). Còn nếu nó đang xử lý một ngắt khác, hoặc một thiết bị khác cũng đồng thời gửi tín hiệu ngắt đến, trên một đường truyền ngắt có mức ưu tiên cao hơn, thì thiết bị sẽ phải chờ một chút. Trong trường hợp này, thiết bị sẽ tiếp tục gửi tín hiệu ngắt vào bus cho tới khi được CPU phục vụ.

Để xử lý ngắt, bộ điều khiển ngắt sẽ đặt một số hiệu ngắt lên các đường dây địa chỉ, nhằm xác định thiết bị muốn được phục vụ, và gửi một tín hiệu để ngắt hoạt động của CPU.

Tín hiệu ngắt khiến CPU phải dừng công việc đang làm để chuyển sang một công việc khác. Số hiệu ngắt trên các đường dây địa chỉ được dùng làm chỉ số để truy nhập vào bảng vector ngắt, nhằm lấy một bộ đếm chương trình mới. Bộ đếm này trỏ tới vị trí bắt đầu của thủ tục xử lý ngắt tương ứng. Các trap và các ngắt cùng sử dụng chung kỹ thuật này, và thường dùng chung bảng vector ngắt. Bảng vector ngắt này có thể nằm tại một vị trí cố định trên máy, hoặc cũng có thể nằm ở bất cứ đâu trong bộ nhớ, và do một thanh ghi của CPU (được nạp bởi hệ điều hành) trỏ tới.

Ngay sau khi bắt đầu chạy, thủ tục xử lý ngắt sẽ gửi một thông báo tới bộ điều khiển ngắt bằng cách ghi một giá trị nào đó lên một trong các cổng vào/ra của bộ điều khiển. Điều đó nghĩa là bộ điều khiển đã được rảnh rỗi, và có thể tiếp tục xử lý ngắt khác. CPU có thể trì hoãn việc gửi thông báo này (cho tới khi nó sẵn sàng xử lý ngắt tiếp theo) nhằm tránh hiện tượng đua tranh khi đồng thời xuất hiện nhiều ngắt. Một số máy tính cũ trước đây không có chíp điều khiển ngắt tập trung, nên mỗi bộ điều khiển thiết bị sẽ chiếm dụng những ngắt riêng.

Phần cứng luôn lưu lại một số thông tin nào đó trước khi khởi động một thủ tục xử lý ngắt. Loại thông tin và vị trí cất thông tin đối với các CPU khác nhau cũng khác nhau. Tối thiểu thì cũng phải lưu lại bộ đếm chương trình, để có thể khôi phục lại tiến trình bị ngắt. Hơn nữa thì có thể cất tất cả các thanh ghi thường dùng, và cả một số lượng lớn các thanh ghi nội bộ nữa.

Một vấn đề cần quan tâm là vị trí cất các thông tin này. Giải pháp thứ nhất là cất chúng vào các thanh ghi nội bộ mà hệ điều hành có thể đọc được. Tuy nhiên, nếu áp dụng giải pháp này thì không được phép giải phóng bộ điều khiển ngắt cho tới khi tất cả các thông tin liên quan đã được lấy ra khỏi các thanh ghi, nhằm tránh nguy cơ một ngắt khác sẽ ghi đè lên các thanh ghi nội bộ đó. Điều này sẽ làm cho khoảng thời gian chết (khoảng thời gian các ngắt bị chặn) kéo dài ra, và có thể dẫn tới việc mất mất các tín hiệu ngắt hay dữ liệu.

Do đó, hầu hết các CPU cất các thông tin nói trên vào ngăn xếp. Tuy nhiên, giải pháp này cũng có nhược điểm. Trước hết, sẽ phải cất thông tin vào ngăn xếp nào? Nếu sử dụng ngăn xếp hiện hành, có thể đó sẽ là một ngăn xếp của tiến trình người dùng. Con trỏ ngăn xếp có thể sẽ không hợp lệ, và gây ra một lỗi nghiêm trọng khi phần cứng cố ghi một vài word vào đó. Ngoài ra, nó có thể đang trỏ tới cuối một trang nhớ. Sau vài lần ghi vào bộ nhớ, trang nhớ sẽ đầy, và có thể xảy ra lỗi trang. Việc xảy ra lỗi trang trong khi đang xử lý ngắt phần cứng sẽ gây ra một vấn đề lớn: không biết sẽ phải lưu trạng thái hiện hành vào đâu để xử lý lỗi trang?

Còn nếu sử dụng ngăn xếp của kernel, con trỏ ngăn xếp sẽ có cơ hội lớn hơn nhiều để đạt được sự hợp lệ cũng như khả năng trỏ một tới trang nhớ phù hợp. Tuy nhiên, việc chuyển đổi

sang chế độ kernel sẽ đòi hỏi phải thay đổi ngữ cảnh MMU, và có thể sẽ làm mất hiệu lực của phần lớn hoặc toàn bộ cache và TLB. Việc nạp lại tất cả những thông tin này, dù tĩnh hay động thì cũng sẽ làm tăng thời gian xử lý ngắt, và do đó làm lãng phí thời gian CPU.

Một vấn đề khác là hầu hết các CPU hiện đại đều áp dụng mô hình kênh dẫn (pipeline) hoặc siêu hướng (superscalar - xử lý song song) để thực hiện lệnh. Trong các hệ thống cũ, mỗi khi một lệnh thi hành xong, vi chương trình hoặc phần cứng sẽ kiểm tra xem có ngắt nào đang chờ được xử lý không. Nếu có thì bộ đếm chương trình và PSW sẽ được cất vào ngăn xếp, rồi tiến hành xử lý ngắt đó. Sau khi trình xử lý ngắt đi vào hoạt động, tiến trình sẽ được thi hành trở lại, với các giá trị của bộ đếm chương trình và PSW lấy từ ngăn xếp.

Mô hình này đã ngầm giả thiết rằng nếu một ngắt xảy ra ngay sau khi thực hiện lệnh nào đó, thì tất cả các lệnh trước đó và cả lệnh này đã được thi hành xong, và chưa có lệnh nào sau đó được thi hành. Trên các hệ thống cũ thì giả thiết này luôn đúng, còn trên các hệ thống hiện đại thì chưa chắc.

Trước hết, ta sẽ xem lại mô hình kênh dẫn trên hình 1-6(a). Điều gì sẽ xảy ra nếu xuất hiện ngắt trong khi kênh dẫn đang đầy (trường hợp phổ biến)? Có nhiều lệnh đang ở trong các trạng thái thi hành khác nhau. Khi ngắt xảy ra, giá trị của bộ đếm chương trình có thể không phản ánh đúng ranh giới giữa các lệnh đã được thi hành và các lệnh chưa được thi hành. Có khả năng nó sẽ phản ánh địa chỉ của lệnh tiếp theo sẽ được lấy về và đặt vào kênh dẫn, chứ không phải là địa chỉ của lệnh vừa được xử lý bởi đơn vị thi hành lệnh.

Vì vậy, có thể tồn tại một gianh giới giữa các lệnh thực sự được thi hành và chưa được thi hành, nhưng phần cứng không nhận biết được nó. Do đó khi hệ điều hành phải trở về từ ngắt, nó không thể lấy địa chỉ đang chứa trong bộ đếm chương trình để nạp vào kênh dẫn. Nó phải tìm ra vị trí của lệnh được thi hành cuối cùng, thường là một công việc phức tạp và đòi hỏi phải có sự phân tích trạng thái hệ thống.

Mặc dù tình huống này đã tệ, các ngắt trên một hệ thống siêu hướng như trên hình 1-6(b) có thể còn tệ hơn. Vì các lệnh có thể thi hành không tuân theo trật tự, nên sẽ không có ranh giới rõ ràng giữa các lệnh đã thi hành và các lệnh chưa thi hành. Có thể là các lệnh 1, 2, 3, 5, và 8 đã được thi hành, còn các lệnh 4, 6, 7, 9, 10… thì chưa. Hơn nữa, bộ đếm chương trình có thể đang trỏ đến lệnh 9, 10, hoặc 11.

Một ngắt đặt hệ thống vào một trạng thái hoàn toàn xác định được gọi là **ngắt xác định - precise interrupt** (Walker và Cragon, 1995). Các ngắt như vậy có bốn thuộc tính sau:

1. Bộ đếm chương trình (PC - Program Counter) được cất vào một vị trí đã biết.
2. Tất cả các lệnh nằm trước vị trí mà PC trỏ tới đều đã được thi hành đầy đủ.
3. Không có lệnh nào nằm ở sau vị trí PC trỏ tới đã được thi hành.
4. Trạng thái thi hành của lệnh mà PC đang trỏ tới đã được xác định.

Chú ý rằng không có sự ngăn cấm nào đối với các lệnh nằm sau vị trí mà PC trỏ tới. Chỉ có điều mọi thay đổi mà chúng thực hiện đối với các thanh ghi hay bộ nhớ phải được huỷ bỏ trước khi ngắt xảy ra. Điều đó sẽ chứng tỏ rằng lệnh tại vị trí con trỏ vừa được thi hành, và và các lệnh nằm sau con trỏ chưa được thi hành. Tuy nhiên, điều này phải được làm rõ trong từng trường hợp cụ thể. Thông thường, nếu ngắt xuất hiện do thực hiện vào/ra, lệnh sẽ chưa được khởi động lại ngay. Còn nếu ngắt do một trap hay lỗi trang, thì PC sẽ trỏ tới lệnh gây ra lỗi, nên nó có thể khởi động lại sau đó.

Một ngắt không thoả mãn những điều kiện trên được gọi là **ngắt không xác định - imprecise interrupt**, và tạo ra những tình huống vô cùng khó xử đối với những người thiết kế hệ điều hành, những người hiện vẫn đang phải tìm hiểu về cái đã xảy ra và cái sẽ xảy ra. Các hệ thống có ngắt không xác định thường phải lưu rất nhiều trạng thái nội bộ lên ngăn xếp, để giúp hệ điều hành xác định được cái đã xảy ra. Việc lưu một lượng lớn thông tin trong bộ nhớ mỗi khi có ngắt sẽ làm chậm tốc độ xử lý ngắt và thậm chí khó khôi phục lại được. Điều này dẫn tới tình huống mỉa mai là các CPU siêu hướng tốc độ cao đôi khi lại không thể xử lý được công việc theo thời gian thực, vì các ngắt thực hiện quá chậm.

Một số máy tính được thiết kế để sử dụng cả các ngắt (và trap) xác định lẫn không xác định. Ví dụ, việc sử dụng các ngắt vào/ra xác định, và các trap (xảy ra do các lỗi chương trình nghiêm trọng) không xác định cũng là một ý tưởng không tồi, vì không cần phải cố gắng để khởi động lại tiến trình đang chạy. Một số máy sử dụng một bít để thiết lập cho tất cả các ngắt trở thành xác định. Việc thiết lập bít này khiến CPU phải cẩn thận ghi lại mọi thứ mà nó đang thực hiện và duy trì một bản sao của các thanh ghi, nhờ vậy nó có thể phát sinh một ngắt xác định tại bất kỳ thời điểm nào. Chi phí cho công việc này ảnh hưởng rất lớn tới hiệu suất hệ thống.

Một số hệ thống siêu hướng, như Pentium Pro và các hệ thống tương thích với nó, có các ngắt xác định để cho phép các chương trình cũ viết cho 386, 486, và Pentium I có thể chạy được (kiến trúc siêu hướng trên Pentium Pro được mô tả như một bộ vi xử lý Pentium I có hai kênh dẫn). Cái giá phải trả cho việc sử dụng các ngắt xác định là phải tạo ra một cấu trúc logic vô cùng phức tạp bên trong CPU, để đảm bảo rằng khi bộ điều khiển ngắt gửi tín hiệu gây ngắt, tất cả các lệnh nằm trước con trỏ đều được phép kết thúc, và không lệnh nào nằm sau vị trí con trỏ được phép tạo ra những hiệu ứng ảnh hưởng tới trạng thái của hệ thống. Đó là chi phí phải trả, không phải bằng thời gian, mà là sự phức tạp bên trong thiết kế chip. Nếu không cần đến các ngắt xác định dùng cho mục đích tương thích với các dòng máy trước, chip này sẽ có thêm chỗ dành cho việc tích hợp thêm cache, và giúp CPU chạy nhanh hơn. Mặt khác, các ngắt không xác định lại làm cho hệ điều hành trở nên phức tạp hơn và chạy chậm hơn, nên khó có thể nói giải pháp nào tốt hơn.

5.2 CÁC NGUYÊN LÝ CỦA PHẦN MỀM VÀO/RA

Bây giờ ta sẽ tìm hiểu về phần mềm vào/ra. Đầu tiên ta sẽ tìm hiểu mục đích của phần mềm vào/ra, và sau đó là các giải pháp thực hiện vào/ra khác nhau dưới góc độ hệ điều hành.

5.2.1 Mục đích của phần mềm vào/ra

Một khái niệm quan trọng trong thiết kế phần mềm vào/ra là **sự độc lập thiết bị (device independence)**. Điều đó có nghĩa là có thể viết chương trình để truy nhập vào bất cứ thiết bị vào/ra nào, mà không phải biết trước thiết bị. Ví dụ, một chương trình muốn đọc dữ liệu từ một file có thể đọc trên đĩa mềm, đĩa cứng, hay CD-ROM, mà không cần phải thay đổi chương trình cho mỗi thiết bị khác nhau đó. Tương tự như vậy, có thể gõ một lệnh như sau:

```
sort <input >output
```

và để nó làm việc với dữ liệu đến từ đĩa mềm, đĩa cứng IDE, đĩa cứng SCSI, hay bàn phím, và kết xuất dữ liệu ra bất kỳ loại đĩa hay màn hình nào. Để làm được như vậy, hệ điều hành sẽ phải quan tâm tới các vấn đề gây ra bởi các thiết bị khác nhau, và đòi hỏi các lệnh rất khác nhau để đọc hay ghi.

Một vấn đề liên quan chặt chẽ với độc lập thiết bị là sử dụng **tên đồng nhất (uniform naming)**. Tên của một file hay thiết bị sẽ là một chuỗi kí tự hay một số nguyên, và không phụ thuộc vào loại thiết bị. Trên UNIX, tất cả các đĩa có thể được tích hợp vào hệ thống file theo một cách bất kỳ, nên người dùng sẽ không cần phải quan tâm tới tên nào ứng với thiết bị nào. Ví dụ, một đĩa mềm có thể được **lắp ghép (mounted)** vào đỉnh của thư mục */usr/ast/backup*, nên việc sao chép một file vào thư mục */usr/ast/backup/monday* sẽ tương đương với việc sao chép file vào đĩa mềm. Tương tự như vậy, tất cả các file và thiết bị có thể được địa chỉ hoá theo cùng một cách: sử dụng đường dẫn.

Một vấn đề quan trọng khác đối với phần mềm vào/ra là **xử lý lỗi (error handling).** Nói chung, việc xử lý lỗi thường theo khuynh hướng càng gần với phần cứng thì càng tốt. Nếu controller phát hiện ra một lỗi đọc dữ liệu, nó sẽ cố gắng sửa lỗi (nếu có thể). Còn nếu không thể thì chương trình điều khiển thiết bị sẽ phải xử lý nó, có lẽ sẽ bằng cách đọc lại khối dữ liệu. Nhiều lỗi chỉ thoáng qua rất nhanh, ví dụ như các lỗi đọc dữ liệu gây ra bởi hạt bụi bám trên đầu đọc, và nó sẽ hết khi tiến hành đọc lại. Nếu các lớp cấp thấp không thể xử lý được vấn đề thì các lớp cấp cao hơn sẽ được thông báo về vấn đề đó. Trong nhiều trường hợp, việc khắc phục lỗi có thể được thực hiện một cách "trong suốt" tại các lớp cấp thấp, và các lớp cấp cao thậm chí không hề biết là lỗi đã xảy ra.

Một vấn đề nữa là sự truyền **đồng bộ (synchronous)** hay **không đồng bộ (asynchronous)**. Hầu hết các quá trình vào/ra vật lý là không đồng bộ - CPU khởi động quá trình truyền rồi chuyển sang làm công việc khác, cho tới khi xuất hiện ngắt. Các chương trình của người dùng sẽ dễ dàng hơn nhiều trong việc ghi dữ liệu nếu các hoạt động vào/ra được đồng bộ - sau một hàm hệ thống để đọc dữ liệu, chương trình sẽ tự động treo cho tới khi dữ liệu nằm sẵn sàng trong bộ đệm. Để làm được điều đó thì hệ điều hành sẽ phải điều khiển các ngắt giống như việc dừng các chương trình của người dùng.

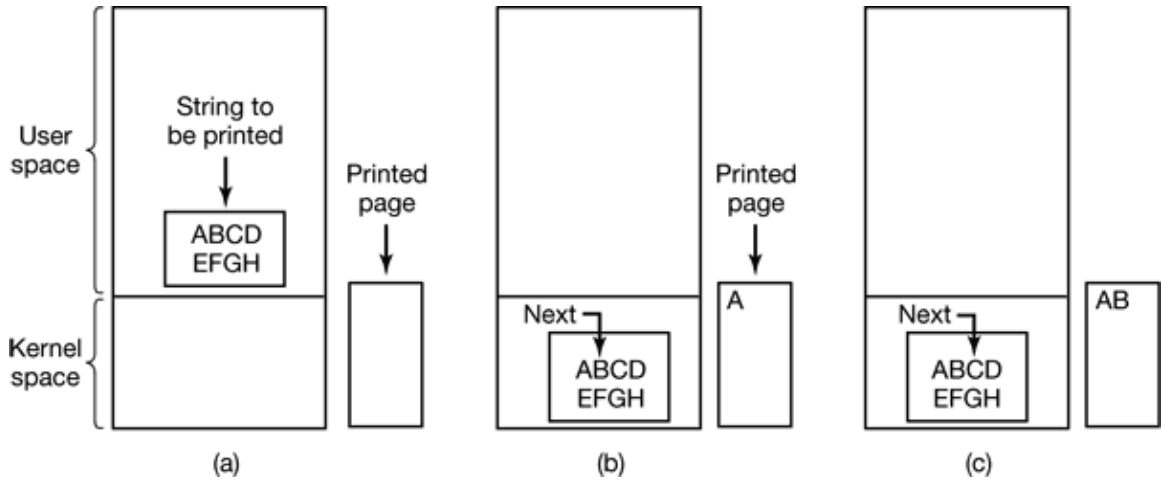
Một vấn đề khác của phần mềm vào/ra là bộ đệm. Thường thì dữ liệu sau khi ra khỏi thiết bị sẽ không được cất trực tiếp vào đích cuối cùng. Ví dụ, khi nhận được một gói tin đến từ mạng, hệ điều hành sẽ không biết phải cất nó vào đâu, cho tới khi tạm cất gói tin vào một vị trí nào đó rồi kiểm tra nó. Cũng như vậy, đối với một số thiết bị phục vụ theo thời gian thực (ví dụ như các thiết bị âm thanh số), dữ liệu phải được đặt vào và lấy ra khỏi bộ đệm trước, để có thể xử lý tốc độ khi bộ đệm đầy và tốc độ khi bộ đệm rỗng một cách riêng biệt, nhằm tránh sự tràn bộ đệm. Sử dụng bộ đệm sẽ phải thực hiện một khối lượng sao chép lớn, và ảnh hưởng tới hiệu suất vào/ra.

Cuối cùng ta sẽ đề cập tới các khái niệm về thiết bị dùng chung được và thiết bị không dùng chung được. Một số thiết bị vào/ra, ví dụ như ổ đĩa, có thể cho phép nhiều người dùng sử dụng đồng thời. Sẽ không có rắc rối gì khi nhiều người dùng mở các file trên cùng một ổ đĩa tại một thời điểm. Các thiết bị khác, như ổ băng từ chẳng hạn, sẽ chỉ phục vụ riêng cho một người dùng cho tới khi người đó kết thúc công việc. Lúc đó người dùng khác mới có thể sử dụng băng từ. Việc hai người dùng hoặc nhiều hơn cùng ghi các khối dữ liệu một cách ngẫu nhiên vào băng từ là không thể thực hiện được. Việc sử dụng các thiết bị không dùng chung được cũng tạo ra nhiều vấn đề khác nhau, như sự bế tắc chẳng hạn. Như vậy, hệ điều hành sẽ phải xử lý cả hai loại thiết bị - dùng chung được và không dùng chung được - theo một cách nào đó nhằm tránh xảy ra rắc rối.

5.2.2 Vào/ra theo chương trình (Programmed I/O)

Về cơ bản, có ba cách khác nhau để thực hiện vào/ra. Trong mục này ta sẽ xem xét cách thứ nhất – Vào/ra theo chương trình. Trong hai mục tiếp theo ta sẽ nghiên cứu các cách thức còn lại (Vào/ra điều khiển ngắt và vào/ra bằng DMA). Cách thực hiện vào/ra đơn giản nhất là để CPU làm tất cả mọi việc, cách này gọi là **Vào/ra theo chương trình**.

Cách minh hoạ đơn giản nhất cho vào/ra theo chương trình là sử dụng một ví dụ. Xét một tiến trình của người dùng muốn gửi một chuỗi gồm 8 kí tự "ABCDEFGH" ra máy in. Trước hết nó phải đặt chuỗi vào bộ đệm trong không gian người dùng, như trên hình vẽ 5-6(a).



Hình 5-6 Các bước in một chuỗi kí tự.

Sau đó tiến trình người dùng sẽ tạo một lời gọi hệ thống để giành quyền sử dụng máy in. Nếu máy in đang được sử dụng bởi một tiến trình khác, lời gọi này sẽ bị lỗi, nó sẽ trả về một mã lỗi

hoặc bị dừng cho tới khi máy in rảnh rỗi, tuỳ vào cách xử lý của hệ điều hành và các tham số của lời gọi. Khi đã có được máy in, tiến trình của người dùng sẽ tạo một lời gọi hệ thống để thông báo với hệ điều hành việc gửi chuỗi kí tự ra máy in.

Tiếp theo, hệ điều hành sẽ sao chép bộ đệm đang chứa chuỗi kí tự (tạm gọi là *p*) vào không gian kernel, nơi nó có thể truy nhập dễ dàng hơn (vì kernel có thể sẽ phải thay đổi bản đồ bộ nhớ để lấy dữ liệu từ không gian người dùng). Sau đó nó sẽ kiểm tra xem máy in đã sẵn sàng chưa. Nếu chưa thì nó sẽ chờ. Ngay khi máy in sẵn sàng, hệ điều hành sẽ sao chép kí tự đầu tiên vào thanh ghi dữ liệu của máy in, trong ví dụ này sử dụng phương pháp ánh xạ cổng vào/ra tới bộ nhớ. Hành động này sẽ kích hoạt máy in. Kí tự này có thể sẽ chưa hiện ra ngay, vì một số máy in sẽ gửi một dòng hoặc một trang dữ liệu vào bộ đệm trước khi in đồng loạt tất cả. Tuy nhiên, trên hình 5-6(b) ta thấy kí tự đầu tiên đã được in, và hệ thống đã đánh dấu kí tự "B" là kí tự sẽ được in tiếp theo.

Ngay khi kí tự đầu tiên được sao chép vào máy in, hệ điều hành sẽ kiểm tra xem máy in có sẵn sàng để chấp nhận một lệnh khác không. Nói chung, máy in thường có một thanh ghi nữa dùng để chứa trạng thái. Hành động ghi vào thanh ghi dữ liệu sẽ làm cho trạng thái máy in trở thành không sẵn sàng. Khi bộ điều khiển máy in đã xử lý xong kí tự hiện hành, nó sẽ thông báo trạng thái sẵn sàng của máy in bằng cách thiết lập một số bít trong thanh ghi trạng thái, hoặc đặt một giá trị nào đó vào thanh ghi này.

Hệ điều hành sẽ chờ cho tới khi máy in sẵn sàng trở lại. Khi đó kí tự tiếp theo sẽ được in, như minh hoạ ở hình 5-6(c). Vòng lặp tiếp tục cho tới khi toàn bộ chuỗi được in hết. Sau đó quyền điều khiển được trả về cho tiến trình của người dùng.

Các hành động nói trên của hệ điều hành được tóm tắt lại trong hình 5-7. Đầu tiên dữ liệu được sao chép vào kernel. Sau đó hệ điều hành khởi tạo một vòng lặp để kết xuất từng kí tự. Có thể nhìn thấy diện mạo cơ bản của phương pháp Vào/ra theo chương trình trên hình minh hoạ này, cứ mỗi khi kết xuất được một kí tự thì CPU lại tiếp tục thăm dò thiết bị xem nó có sẵn sàng nhận kí tự tiếp theo không. Hành động này được gọi là **polling (thăm dò)** hay **busy waiting (chờ bận).**

```
copy_from_user(buffer, p, count);          /* p là bộ đệm của kernel */
for (i = 0; i < count; i++) {              /* Lặp với tất cả các kí tự */
    while (*printer_status_reg != READY) ; /* Lặp cho tới khi sẵn sàng */
    *printer_data_register = p[i];         /* Kết xuất một kí tự */
}
return_to_user();
```

Hình 5-7 Ghi một chuỗi kí tự ra máy in bằng phương pháp Vào/ra theo chương trình.

Vào/ra theo chương trình là một phương pháp đơn giản, nhưng có nhược điểm là sẽ chiếm dụng hoàn toàn CPU trong thời gian thực hiện vào/ra. Nếu thời gian in một kí tự rất ngắn (vì tất cả các máy in đều sao chép kí tự vào bộ đệm của nó), thì thời gian chờ có thể chấp nhận được. Ngoài ra, trong các hệ thống nhúng, nơi CPU không có việc gì khác để làm, thì việc dành thời gian để chờ bận cũng là điều chấp nhận được. Tuy nhiên, trên các hệ thống phức tạp hơn, nơi mà CPU phải làm rất nhiều công việc khác nhau, thì việc chờ bận sẽ ảnh hưởng lớn tới hiệu quả hệ thống. Khi đó sẽ phải cần tới một phương pháp tốt hơn.

5.2.3 Vào/ra điều khiển ngắt (Interrupt-Driven I/O)

Bây giờ ta sẽ xem xét trường hợp máy in không đưa kí tự vào bộ đệm, mà in trực tiếp mỗi khi có kí tự đến. Nếu máy in có thể in được 100 kí tự/giây, thì mỗi kí tự sẽ tốn mất 10 ms. Điều này nghĩa là sau khi kí tự được đặt vào thanh ghi dữ liệu của máy in, CPU sẽ phải ở trong một vòng lặp chờ bận 10 ms, cho tới khi được phép kết xuất kí tự tiếp theo. Chỉ có 10 ms thì sẽ không đủ thời gian để chuyển đổi ngữ cảnh và chạy một tiến trình khác, tức là thời gian đó của CPU bị lãng phí.

Một giải pháp để CPU có thể làm một công việc gì đó (trong lúc chờ máy in đạt trạng thái sẵn sàng) là sử dụng các ngắt. Khi có hàm hệ thống yêu cầu in chuỗi, bộ đệm được sao chép vào không gian kernel (như ta đã nói ở trên), và kí tự đầu tiên được sao chép tới máy in ngay khi máy in chấp nhận. Lúc này CPU sẽ gọi bộ phận điều độ, và một tiến trình nào đó sẽ được chạy. Tiến trình in chuỗi kí tự sẽ bị dừng cho tới khi toàn bộ chuỗi được in xong. Các công việc cần thực hiện khi có hàm hệ thống được trình bày ở hình 5-8(a).

```
copy_from_user(buffer, p, count);
enable_interrupts();
while(*printer_status_reg != READY) ;
*printer_data_register = p[0];
scheduler();
```

(a)

```
if(count == 0) {
    unblock_user();
} else {
    *printer_data_register = p[i];
    count = count - 1;
    i = i + 1;
}
acknowledge_interrupt();
return_from_interrupt();
```

(b)

Hình 5-8 Ghi một chuỗi kí tự ra máy in bằng phương pháp Vào/ra điều khiển ngắt. (a) Mã thi hành khi có hàm hệ thống. (b) Thủ tục xử lý ngắt.

Khi máy in in xong một kí tự và chuẩn bị nhận kí tự tiếp theo, nó sẽ phát ra một tín hiệu ngắt. Ngắt này sẽ dừng tiến trình hiện hành và lưu lại trạng thái của tiến trình đó. Sau đó thủ tục xử lý ngắt máy in được chạy. Một phiên bản đơn giản của thủ tục này được trình bày ở hình 5-8(b). Nếu không có kí tự nào cần in nữa, thủ tục xử lý ngắt sẽ đánh thức tiến trình người dùng đang bị dừng. Còn nếu vẫn có kí tự cần in, nó sẽ gửi kí tự tiếp theo ra máy in, thông báo hoàn thành ngắt, và trở về tiến trình hiện hành (là tiến trình đang chạy khi xảy ra ngắt), tiến trình này lại tiếp tục thực hiện công việc của nó.

5.2.4 Vào/ra bằng DMA (I/O using DMA)

Một nhược điểm dễ thấy của phương pháp vào/ra điều khiển ngắt là phải phát sinh ngắt mỗi khi in xong một kí tự. Các ngắt sẽ chiếm dụng thời gian, nên điều này sẽ gây lãng phí thời gian CPU. Có một giải pháp khác là sử dụng DMA. Ý tưởng này sẽ để DMA controller thực hiện việc gửi kí tự ra máy in, chứ không dùng đến CPU. Về bản chất, DMA cũng là phương pháp vào/ra theo chương trình, chỉ khác ở chỗ DMA controller sẽ làm mọi việc, chứ không phải là CPU chính. Các mã lệnh minh hoạ được trình bày ở hình 5-9.

```
copy_from_user(buffer, p, count);
set_up_DMA_controller();
scheduler();
```

(a)

```
acknowledge_interrupt();
unblock_user();
return_from_interrupt();
```

(b)

Hình 5-9 In một chuỗi kí tự bằng DMA. (a) Mã thi hành khi có hàm hệ thống in. (b) Thủ tục xử lý ngắt.

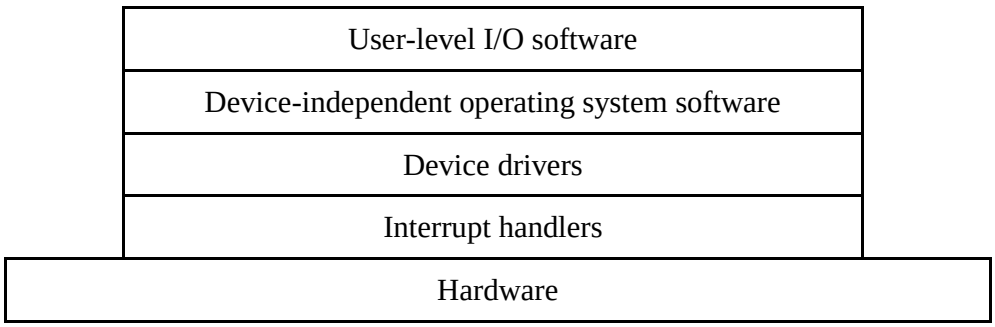
Ưu điểm lớn của DMA là giảm được số lượng ngắt. Trước đây mỗi khi in một kí tự phải phát sinh một ngắt, còn bây giờ thì in hết cả bộ đệm mới cần tới một ngắt. Điều này rất có ích khi cần in nhiều kí tự và khi các ngắt quá chậm. Tuy nhiên, DMA controller thường có tốc độ chậm hơn nhiều so với CPU. Nếu DMA controller không có khả năng điều khiển các thiết bị tốc độ cao, hay CPU lại thường không có việc gì để làm khi chờ ngắt DMA, thì viêc áp dụng vào/ra điều khiển ngắt hoặc vào/ra theo chương trình có thể sẽ tốt hơn.

5.3 CÁC LỚP PHẦN MỀM VÀO/RA

Phần mềm vào/ra thường được tổ chức với bốn lớp, như minh hoạ ở hình 5-10. Mỗi lớp có một chức năng riêng và một giao diện riêng đối với lớp liền kề. Trên các hệ thống khác nhau thì chức năng và giao diện của các lớp cũng khác nhau, nên những thảo luận dưới đây (trình bày lần lượt từ lớp dưới lên lớp trên) không nói về một hệ thống cụ thể nào.

5.3.1 Các trình xử lý ngắt (Interrupt handlers)

Đối với hầu hết các thao tác vào/ra, việc gặp phải các ngắt là điều không tránh khỏi, cho dù nó có khó chịu chăng nữa. Điều khó chịu này có thể được ẩn đi, được che dấu bên dưới lớp vỏ của hệ điều hành, nên chỉ có một bộ phận nhỏ của hệ điều hành biết về chúng. Cách tốt nhất để dấu chúng là phải dừng chương trình điều khiển thiết bị (driver) đã khởi tạo quá trình vào/ra đó, cho tới khi quá trình vào/ra hoàn thành và xuất hiện ngắt. Chương trình điều khiển này có thể dừng chính nó, ví dụ bằng cách thực hiện phép toán `down` trên một semaphore, bằng phép toán `wait` trên một biến điều kiện, bằng cách `receive` trên một thông báo, hoặc những cách khác tương tự.



Hình 5-10 Các lớp của hệ thống phần mềm vào/ra.

Khi xảy ra ngắt, thủ tục ngắt sẽ làm bất cứ điều gì để xử lý ngắt. Sau đó nó có thể đánh thức driver đã khởi động nó. Trong một số trường hợp nó sẽ hoàn thành phép toán `up` trên một semaphore. Trong các trường hợp khác nó sẽ thực hiện phép toán `signal` đối với biến điều kiện trên một monitor. Cũng có thể nó sẽ gửi một thông báo để đánh thức driver. Trong tất cả các trường hợp, hiệu quả thực của ngắt sẽ là sự đánh thức một driver bị dừng trước đó, để nó có thể chạy. Mô hình này sẽ hoạt động tốt nhất khi các driver được cấu trúc giống như các tiến trình của kernel, với các trạng thái riêng, ngăn xếp riêng, và bộ đếm chương trình riêng.

Tất nhiên là thực tế không đơn giản như vậy. Việc xử lý một ngắt không phải chỉ là vấn đề lấy ngắt, thực hiện phép toán `up` trên semaphore nào đó, và thi hành lệnh IRET để trở về tiến trình trước đó. Sẽ có rất nhiều việc mà hệ điều hành phải làm. Bây giờ ta sẽ liệt kê một số công việc dưới dạng các bước phải thực hiện bằng phần mềm khi ngắt cứng xảy ra. Cần chú ý là các chi tiết cụ thể sẽ phụ thuộc vào từng hệ thống, nên có thể một số bước ở đây sẽ không cần thiết đối với một hệ thống cụ thể nào đó, và có thể sẽ phải cần đến những bước khác nữa. Ngoài ra, các bước cũng có thể diễn ra theo những trật tự khác.

1. Cất các thanh ghi (kể cả PSW) nếu nó chưa được cất bởi ngắt cứng.
2. Thiết lập ngữ cảnh cho thủ tục xử lý ngắt. Công việc này bao gồm cả việc cài đặt TLB, MMU, và bảng trang.
3. Thiết lập ngăn xếp cho thủ tục xử lý ngắt.
4. Thông báo với bộ điều khiển ngắt. Nếu không có bộ điều khiển ngắt tập trung thì cần bật lại các ngắt.
5. Sao chép các thanh ghi từ nơi chúng được cất (có thể là ngăn xếp nào đó) tới bảng tiến trình.
6. Chạy thủ tục xử lý ngắt. Nó sẽ lấy thông tin từ các thanh ghi của bộ điều khiển thiết bị gây ra ngắt.

7. Chọn tiến trình để chạy tiếp theo. Nếu ngắt làm cho một tiến trình có mức ưu tiên cao (đang bị dừng) trở thành sẵn sàng thì có thể tiến trình này sẽ được chạy.
8. Thiết lập ngữ cảnh MMU cho tiến trình sẽ được chạy tiếp theo. Cũng có thể cần cài đặt cả TLB.
9. Nạp các thanh ghi của tiến trình mới, bao gồm cả PSW.
10. Bắt đầu chạy tiến trình mới.

Như đã trình bày, quá trình xử lý ngắt không hề đơn giản. Có thể sẽ phải dùng đến rất nhiều lệnh của CPU, đặc biệt là trên các máy sử dụng bộ nhớ ảo và có các bảng trang phải được cài đặt, hoặc phải lưu trạng thái của MMU (ví dụ phải lưu các bít *R* và *M*). Trên một số máy có thể phải quản lý cả TLB và cache CPU khi chuyển đổi giữa chế độ người dùng và kernel, điều này sẽ làm tăng thêm chi phí hệ thống.

5.3.2 Các chương trình điều khiển thiết bị (Device Drivers)

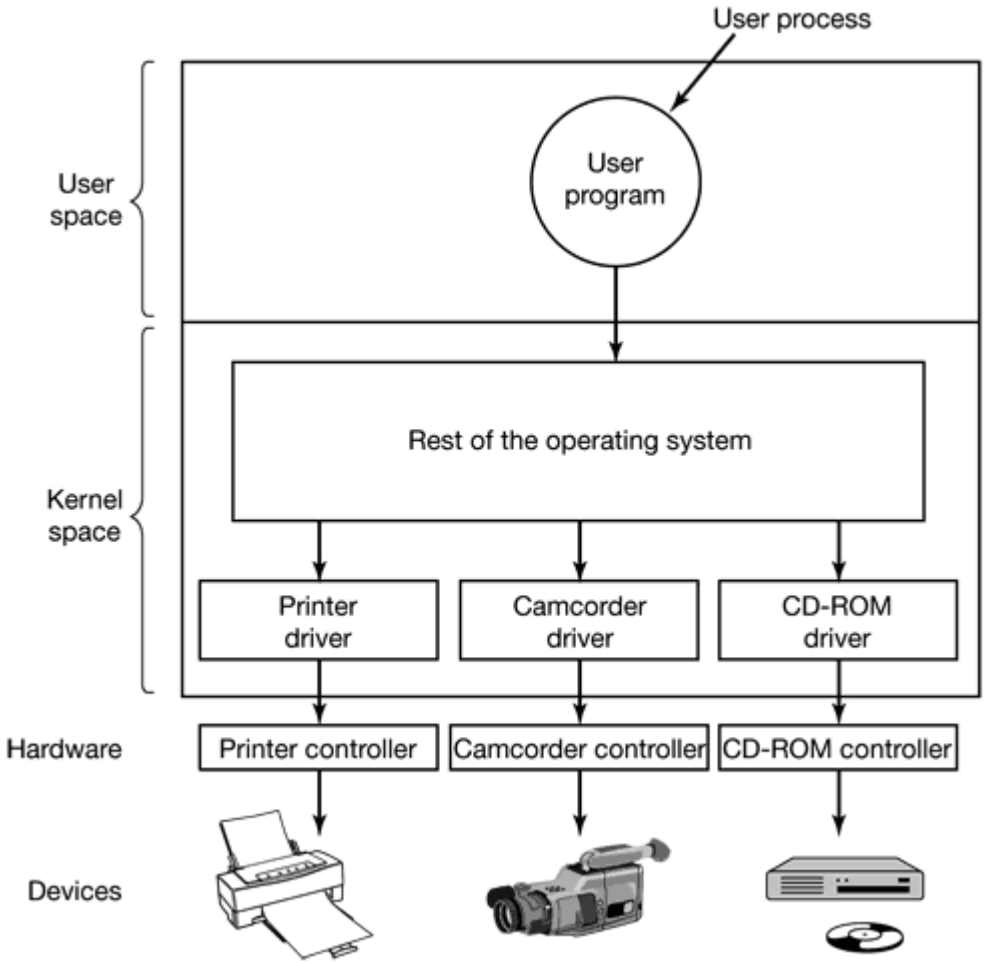
Ở đầu chương này ta đã tìm hiểu về nhiệm vụ của các bộ điều khiển thiết bị. Ta thấy rằng mỗi controller lại có một vài thanh ghi thiết bị dùng để nhận lệnh, hoặc một vài thanh ghi thiết bị dùng để đọc trạng thái, hoặc cả hai loại. Số lượng thanh ghi thiết bị và dạng lệnh đối với các thiết bị khác nhau cũng khác nhau. Ví dụ, chương trình điều khiển chuột phải nhận thông tin từ chuột, bao gồm thông số về khoảng cách di chuyển và loại nút đã được bấm. Trái lại, chương trình điều khiển đĩa lại cần biết về các sector (cung từ), track (rãnh từ), cylinder (trụ), head (đầu đọc), arm motion (chuyển động của cánh tay đĩa), motor đĩa, thời gian di chuyển đầu đọc, và tất cả các thông tin khác để có thể làm việc hiệu quả. Hiển nhiên là các thông tin này sẽ rất khác nhau.

Vì thế, mỗi thiết bị vào/ra kết nối với máy tính đều cần tới các mã lệnh đặc trưng cho thiết bị để điều khiển nó. Các mã này được gọi là **trình điều khiển thiết bị (device driver)**, nó thường được viết bởi nhà sản xuất và được phân phối kèm theo thiết bị. Do mỗi hệ điều hành lại cần có các trình điều khiển thiết bị riêng của nó, nên các nhà sản xuất thiết bị thường cung cấp trình điều khiển cho một số hệ điều hành phổ biến.

Mỗi trình điều khiển thiết bị thường chỉ dùng để điều khiển cho một loại thiết bị, hoặc một lớp các thiết bị có quan hệ gần gũi với nhau. Ví dụ, một trình điều khiển đĩa SCSI thường điều khiển được nhiều loại đĩa SCSI, với các dung lượng và tốc độ khác nhau, và có lẽ cũng điều khiển được cả ổ CD-ROM SCSI. Nhưng con chuột và cần điều khiển (joystick) là hai thiết bị khác nhau, nên sẽ cần tới các driver khác nhau. Tuy nhiên, không có quy định nào cấm một driver điều khiển cho nhiều thiết bị không liên quan đến nhau. Đó chỉ không phải là một ý kiến hay thôi.

Để có thể truy nhập vào phần cứng thiết bị, tức là các thanh ghi của controller, trình điều khiển thiết bị phải là một bộ phận của kernel, ít nhất là với các kiến trúc hiện nay. Thực ra, có thể xây dựng các driver chạy trong không gian người dùng, và dùng các lời gọi hệ thống để thực hiện đọc hay ghi vào các thanh ghi thiết bị. Thiết kế này sẽ là một ý tưởng hay, do nó tách phần kernel khỏi các driver, và tách các driver ra khỏi nhau. Làm như vậy sẽ giúp loại trừ một nguyên nhân chính gây đổ vỡ hệ thống - các lỗi của driver, chúng thường ảnh hưởng xấu tới kernel theo cách này hay cách khác. Tuy nhiên, do các hệ điều hành hiện nay vẫn muốn các driver chạy trên kernel, nên ta sẽ xem xét về mô hình này.

Do các nhà thiết kế hệ điều hành đều biết rằng các thành phần của driver (được cung cấp bởi nhà sản xuất thiết bị) sẽ được cài đặt vào hệ điều hành, nên hệ điều hành cần phải có một kiến trúc cho phép thực hiện việc cài đặt đó. Nghĩa là cần phải có một mô hình cụ thể cho driver, và cách thức mà nó tương tác với các phần còn lại của hệ điều hành. Các driver thường nằm bên dưới các thành phần khác của hệ điều hành, như minh hoạ ở hình 5-11.



Hình 5-11.

Các hệ điều hành thường phân loại các driver theo loại của thiết bị. Có hai loại thiết bị phổ biến là các thiết bị khối (ví dụ như các ổ đĩa), chúng chứa nhiều khối dữ liệu có thể địa chỉ hoá một cách độc lập, và các thiết bị kí tự (như bàn phím và máy in), chúng thường gửi hoặc nhận một dòng kí tự.

Hầu hết các hệ điều hành đều định nghĩa một giao diện chuẩn mà tất cả các driver khối phải hỗ trợ, và một giao diện chuẩn thứ hai dành cho tất cả các driver kí tự. Các giao diện này bao gồm nhiều thủ tục mà phần còn lại của hệ điều hành có thể gọi và yêu cầu driver thực hiện. Các thủ tục điển hình là đọc một khối dữ liệu (đối với thiết bị khối), hay ghi một chuỗi kí tự (đối với thiết bị kí tự).

Trên một số hệ thống, hệ điều hành là một chương trình nhị phân chứa tất cả các driver đã được biên dịch. Giải pháp này từng được áp dụng trên các hệ thống UNIX, vì chúng được chạy tại các trung tâm máy tính, và các thiết bị vào/ra hồi đó cũng hiếm khi thay đổi. Nếu có thêm thiết bị mới, người quản trị hệ thống chỉ cần biên dịch lại kernel với driver mới, để tạo ra một chương trình nhị phân mới.

Cùng với sự phát triển của máy tính cá nhân, với vô số các thiết bị vào/ra của chúng, mô hình nói trên không áp dụng được. Rất ít người dùng có khả năng biên dịch lại hay liên kết lại kernel, thậm chí ngay cả khi họ có mã nguồn hoặc các mô đun đối tượng (object) thì cũng hiếm khi làm được như vậy. Thay vào đó, các hệ điều hành (mà khởi đầu là MS-DOS) đi theo một mô hình khác, các driver sẽ được nạp vào hệ thống trong quá trình thi hành. Các hệ thống khác nhau sẽ thực hiện nạp driver theo những cách khác nhau.

Một driver có thể có nhiều chức năng. Chức năng cơ bản nhất là nhận các yêu cầu đọc và ghi trừu tượng từ các phần mềm độc lập với thiết bị (nằm ở bên trên nó), và kiểm tra xem có thực

hiện được không. Chúng cũng phải thực hiện một vài chức năng khác nữa. Ví dụ, driver sẽ phải khởi động thiết bị (nếu cần). Nó cũng có thể phải quản lý các nhu cầu về điện năng, và ghi lại các biến cố.

Nhiều driver thiết bị có cấu trúc tương tự nhau. Một driver điển hình sẽ bắt đầu bằng việc kiểm tra các tham số đầu vào, để xem chúng có hợp lệ không. Nếu không thì nó sẽ trả về một mã lỗi. Còn nếu hợp lệ thì nó sẽ chuyển các tham số trừu tượng về dạng cụ thể. Đối với trình điều khiển đĩa, nó sẽ chuyển đổi số hiệu khối dữ liệu tuyến tính thành các số hiệu cụ thể của head, track, sector, và cylinder.

Tiếp theo, driver sẽ kiểm tra xem liệu thiết bị có ở trong trạng thái bận không. Nếu có thì yêu cầu sẽ được xếp vào hàng đợi để chờ được xử lý. Còn nếu thiết bị đang rỗi thì sẽ phải kiểm tra trạng thái phần cứng của nó, xem nó có thể đáp ứng ngay được yêu cầu không. Có thể sẽ phải bật thiết bị lên, hoặc khởi động motor trước khi bắt đầu truyền dữ liệu. Khi thiết bị được bật và sẵn sàng để chạy, quá trình điều khiển thực sự có thể bắt đầu.

Điều khiển một thiết bị nghĩa là gửi một chuỗi các lệnh tới thiết bị đó. Driver là nơi mà chuỗi lệnh sẽ được xác định, tuỳ thuộc vào cái mà nó phải làm. Sau khi driver biết được lệnh nào sẽ được gửi đi, nó bắt đầu ghi chúng vào các thanh ghi thiết bị của controller. Mỗi khi ghi một lệnh tới controller, nó có thể sẽ kiểm tra xem controller có chấp nhận lệnh đó không, và có sẵn sàng để nhận lệnh tiếp theo không. Quá trình đó tiếp tục diễn ra cho tới khi tất cả các lệnh đã được gửi xong. Một số controller có thể được cung cấp danh sách liên kết của các lệnh (nằm trong bộ nhớ), và được yêu cầu đọc và xử lý chúng, mà không cần tới sự trợ giúp của hệ điều hành.

Sau khi các lệnh đã được gửi, có thể xảy ra một trong hai tình huống: Trong nhiều trường hợp, driver phải chờ cho tới khi controller làm điều gì đó cho nó, nên nó sẽ tự dừng cho tới khi xuất hiện ngắt để đánh thức nó. Tuy nhiên, trong các trường hợp khác, nó sẽ kết thúc mà không cần dừng lại. Ví dụ cho tình huống thứ hai, việc cuộn màn hình trong chế độ hiển thị kí tự sẽ đòi hỏi phải ghi một số byte vào các thanh ghi của controller. Sẽ không cần tới sự vận động cơ khí nào, nên toàn bộ hoạt động có thể thực hiện xong trong vài nano giây.

Trong tình huống thứ nhất, driver bị dừng sẽ được đánh thức bởi một ngắt. Còn trong tình huống thứ hai, nó sẽ không bao giờ ngủ. Trong cả hai trường hợp, sau khi hoàn thành công việc, driver sẽ phải kiểm tra lỗi. Nếu mọi cái đều diễn ra tốt đẹp, driver có thể chuyển dữ liệu cho phần mềm độc lập thiết bị (ví dụ chuyển một khối dữ liệu vừa đọc được). Cuối cùng, nó trả các thông tin về trạng thái lỗi cho đối tượng gọi nó. Nếu có các yêu cầu khác trong hàng đợi, một trong số chúng sẽ được lựa chọn để thực hiện. Còn nếu hàng đợi trống rỗng, driver sẽ dừng để chờ yêu cầu tiếp theo.

Mô hình nói trên đã được đơn giản hoá. Trên thực tế, các vấn đề sẽ phức tạp hơn nhiều. Ví dụ, một thiết bị vào/ra có thể hoàn thành công việc trong khi driver đang chạy, nó sẽ gửi ngắt tới driver. Ngắt này sẽ ảnh hưởng tới hoạt động của driver. Trên thực tế, nó có thể làm cho driver hiện hành bị chạy lại từ đầu. Ví dụ, trong khi driver mạng đang xử lý một gói tin mà nó nhận được, rất có thể sẽ xuất hiện một gói tin khác. Nó sẽ phải chạy lại, nghĩa là driver bị gọi lần thứ hai trong khi lần gọi thứ nhất chưa kết thúc.

Trên một hệ thống có thể cài đặt nóng, các thiết bị có thể được lắp vào hoặc tháo ra trong khi máy tính đang chạy. Kết quả là trong khi driver đang bận đọc từ một thiết bị nào đó, hệ thống có thể sẽ thông báo với nó là người dùng vừa đột ngột tháo thiết bị khỏi hệ thống. Không những phải dừng quá trình vào/ra hiện hành một cách an toàn (không được làm hỏng các cấu trúc dữ liệu của kernel), mà còn phải khéo léo loại bỏ các yêu cầu truy nhập chưa được thực hiện, kể cả các đối tượng đưa ra các yêu cầu đó. Hơn nữa, việc lắp thêm các thiết bị mới có thể khiến kernel phải sắp xếp lại tài nguyên (ví dụ, sắp xếp lại các đường yêu cầu ngắt), tách các thiết bị cũ khỏi driver, và cấp nó cho thiết bị mới.

Các driver không được phép tạo ra lời gọi hệ thống, nhưng chúng thường phải tương tác với phần còn lại của kernel. Thường thì việc gọi tới các thủ tục của kernel có thể được chấp nhận. Ví dụ như các lời gọi để cấp phát hay thu hồi các trang bộ nhớ dùng làm bộ đệm. Cũng cần tới các lời gọi khác để quản lý MMU, các bộ định thời, DMA controller, bộ điều khiển ngắt…

5.3.3 Phần mềm vào/ra độc lập thiết bị (Device-Independent I/O Software)

Mặc dù có một số bộ phận của phần mềm vào/ra phụ thuộc vào thiết bị, nhưng các bộ phận khác lại độc lập với thiết bị. Sự khác biệt chính giữa lớp phần mềm độc lập thiết bị với lớp driver nằm ở sự phụ thuộc vào hệ thống (và thiết bị), chứ không phải ở vị trí các lớp, vì một số chức năng của phần mềm độc lập thiết bị cũng có thể được thực hiện ở lớp driver để tăng hiệu suất. Hình 5-12 liệt kê các chức năng điển hình của phần mềm độc lập thiết bị.

Đồng nhất giao diện cho các driver
Làm bộ đệm
Thông báo lỗi
Phân phối và giải phóng các thiết bị chuyên biệt
Cung cấp kích thước khối dữ liệu độc lập với thiết bị

Hình 5-12 Các chức năng của phần mềm vào/ra độc lập thiết bị.

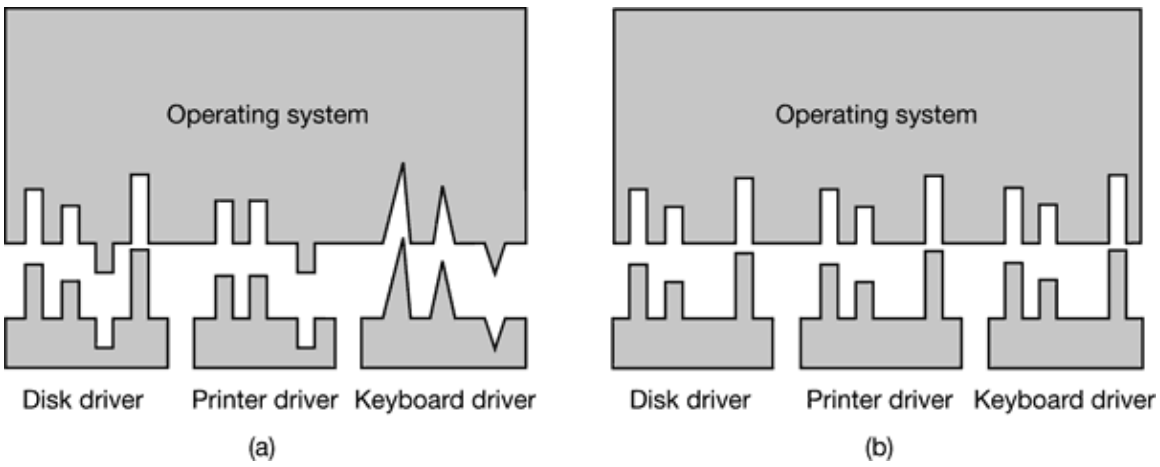
Chức năng cơ bản của phần mềm độc lập thiết bị là thực hiện các chức năng vào/ra chung cho mọi thiết bị, và cung cấp một giao diện đồng nhất cho các phần mềm cấp người dùng. Phần tiếp theo ta sẽ nghiên cứu chi tiết hơn về các vấn đề này.

Đồng nhất giao diện cho các driver

Một vấn đề quan trọng của hệ điều hành là phải làm cho tất cả các thiết bị vào/ra, cũng như các driver trông có vẻ tương tự nhau. Nếu ổ đĩa, máy in, bàn phím… đều sử dụng các giao diện riêng, thì mỗi khi lắp thêm thiết bị mới, hệ điều hành sẽ phải được thay đổi cho phù hợp với thiết bị. Việc sửa đổi hệ điều hành chỉ vì có thêm thiết bị mới không phải là một ý kiến hay.

Một khía cạnh của vấn đề này là giao diện giữa các driver và phần còn lại của hệ điều hành. Trên hình 5-13(a) ta thấy một tình huống mà mỗi driver lại có một giao diện riêng đối với hệ điều hành. Điều đó nghĩa là các driver khác nhau sẽ cung cấp các chức năng khác nhau cho hệ thống. Và điều đó cũng có nghĩa là các chức năng mà kernel cung cấp cho các driver khác nhau cũng khác nhau. Như vậy, giao diện với mỗi thiết bị mới sẽ đòi hỏi rất nhiều công sức lập trình.

Trái lại, trên hình 5-13(b) ta thấy một thiết kế khác, trong đó tất cả các driver đều có giao diện giống nhau. Như vậy sẽ dễ dàng hơn cho việc lắp thêm các thiết bị mới, cung cấp cho nó một giao diện driver phù hợp. Điều đó nghĩa là những người viết driver sẽ phải biết rõ việc cần làm (ví dụ, họ sẽ phải cung cấp các chức năng nào, và họ có thể gọi các chức năng nào của kernel). Trên thực tế, không phải tất cả các thiết bị đều hoàn toàn giống nhau, nhưng số loại thiết bị cũng không phải là nhiều, và thậm chí chúng cũng gần giống nhau. Ví dụ, các thiết bị khối và kí tự tuy khác nhau, nhưng vẫn có nhiều chức năng chung.



Hình 5-13 (a) Không có giao diện driver chuẩn. (b) Sử dụng giao diện driver chuẩn.

Một khía cạnh khác của việc đồng nhất giao diện là cách đặt tên cho các thiết bị vào/ra. Phần mềm độc lập thiết bị sẽ ánh xạ các tên thiết bị tới driver phù hợp. Ví dụ, trong UNIX, một tên thiết bị như */dev/disk0* sẽ xác định một nút i duy nhất cho một file đặc biệt, và nút i này sẽ chứa số hiệu thiết bị chính, số hiệu này dùng để định vị driver phù hợp. Nút i cũng chứa số hiệu thiết bị phụ, là một tham số dùng để gửi cho driver, nhằm xác định đơn vị đọc (hay ghi) dữ liệu. Tất cả các thiết bị đều có số hiệu chính và số hiệu phụ, còn tất cả các driver đều được truy nhập bằng cách sử dụng số hiệu chính để lựa chọn driver.

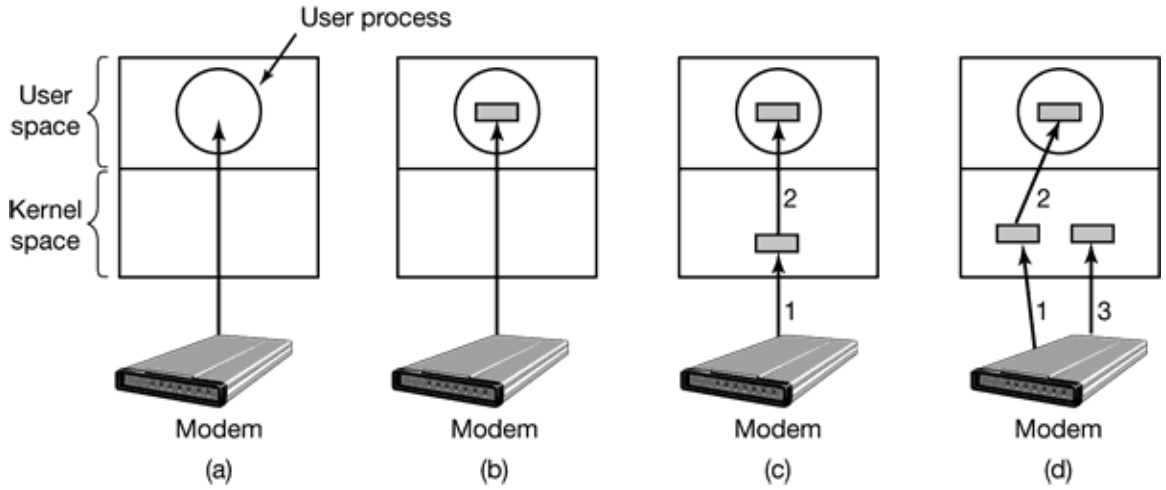
Liên quan tới việc đặt tên là vấn đề bảo vệ. Làm cách nào để ngăn không cho người dùng truy nhập vào các thiết bị mà họ không có quyền sử dụng? Trên UNIX và Windows 2000, các thiết bị xuất hiện trong hệ thống file dưới dạng các đối tượng được đặt tên, nghĩa là các nguyên tắc bảo vệ file cũng có thể áp dụng cho thiết bị vào/ra. Quản trị hệ thống có thể thiết lập quyền hạn truy nhập cho từng thiết bị.

Bộ đệm

Bộ đệm cũng là một vấn đề đối với cả thiết bị khối và thiết bị kí tự, với các lý do khác nhau. Xét một tiến trình muốn đọc dữ liệu từ modem. Một giải pháp để xử lý các kí tự nhận được là tiến trình người dùng sẽ gọi hàm hệ thống `read`, và dừng lại để chờ kí tự. Mỗi kí tự đến sẽ gây ra một ngắt. Thủ tục xử lý ngắt sẽ chuyển kí tự đó cho tiến trình người dùng, và đánh thức nó. Sau khi đặt kí tự vào đâu đó, tiến trình sẽ gọi hàm để đọc một kí tự khác, rồi lại dừng. Mô hình này được minh hoạ ở hình 5-14(a).

Vấn đề là tiến trình người dùng sẽ phải thức dậy mỗi khi có kí tự đến. Việc cho phép một tiến trình chạy nhiều lần, mỗi lần trong một khoảng thời gian ngắn như vậy sẽ không đem lại hiệu quả, nên đây không phải là một thiết kế tốt.

Một bản cải tiến được trình bày ở hình 5-14(b). Trong đó tiến trình người dùng cung cấp một bộ đệm có *n* kí tự nằm trong không gian người dùng, và thực hiện đọc *n* kí tự. Thủ tục xử lý ngắt sẽ đặt kí tự nhận được vào bộ đệm này cho tới khi nó đầy. Sau đó nó sẽ đánh thức tiến trình người dùng. Giải pháp này có hiệu quả cao hơn nhiều so với giải pháp ở trên, nhưng vẫn còn mặt hạn chế: Điều gì sẽ xảy ra nếu trang nhớ chứa bộ đệm bị đưa ra ngoài, trong khi một kí tự lại được đưa đến? Có thể khoá bộ đệm trong bộ nhớ, nhưng nếu nhiều tiến trình cùng khoá trang nhớ thì số lượng trang nhớ sẵn sàng trong bộ nhớ sẽ giảm xuống, và làm giảm hiệu suất chung.



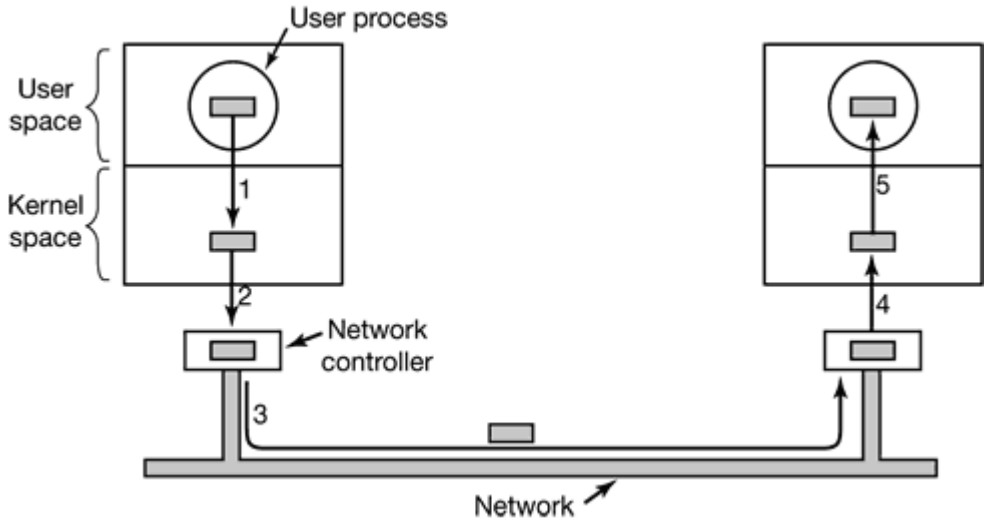
Hình 5-14 (a) Đầu vào không bộ đệm. (b) Bộ đệm trong không gian người dùng. (c) Sử dụng bộ đệm trong kernel, rồi sao chép sang không gian người dùng. (d) Bộ đệm kép trong kernel.

Một giải pháp khác là tạo một bộ đệm trong kernel, và trình xử lý ngắt sẽ đặt các kí tự vào đó, như trình bày ở hình 5-14(c). Khi bộ đệm đầy thì trang chứa bộ đệm của người dùng sẽ được nạp vào bộ nhớ (nếu cần), bộ đệm trong kernel sẽ được sao chép sang bộ đệm của người dùng (chỉ bằng một thao tác sao chép đơn giản). Giải pháp này đem lại hiệu quả cao hơn giải pháp trước.

Tuy nhiên, giải pháp này vẫn có vấn đề: Điều gì sẽ xảy ra nếu các kí tự được đưa đến khi trang nhớ chứa bộ đệm người dùng đang được đưa vào từ đĩa? Do bộ đệm kernel đã đầy nên sẽ không có chỗ để chứa chúng. Một giải pháp khác là sử dụng thêm một bộ đệm nữa trong kernel. Sau khi bộ đệm thứ nhất đầy (và trước khi nó được làm rỗng) thì sẽ cần tới bộ đệm thứ hai, như minh hoạ trên hình 5-14(d). Khi bộ đệm thứ hai đầy, có thể sao chép nó sang bộ đệm người dùng (giả sử người dùng yêu cầu nhận được nó). Trong khi bộ đệm thứ hai sao chép dữ liệu sang không gian người dùng, thì có thể dùng bộ đệm thứ nhất để nhận các kí tự. Theo cách này, cả hai bộ đệm sẽ thay nhau hoạt động: khi bộ đệm này đang được sao chép tới không gian người dùng, thì bộ đệm kia sẽ nhận dữ liệu mới. Giải pháp này được gọi là **bộ đệm kép**.

Bộ đệm cũng rất quan trọng khi cần gửi dữ liệu đi. Ví dụ, làm cách nào để gửi dữ liệu ra modem mà không cần dùng tới bộ đệm? [Sử dụng mô hình trên hình vẽ 5-14(b)]. Tiến trình người dùng sẽ thi hành hàm hệ thống `write` để gửi *n* kí tự. Hệ thống có hai sự lựa chọn. Nó có thể dừng tiến trình người dùng lại cho tới khi tất cả các kí tự đã được ghi ra, nhưng làm vậy sẽ tốn nhiều thời gian chiếm dụng đường truyền điện thoại. Nó cũng có thể giải phóng tiến trình người dùng ngay lập tức và thực hiện vào/ra, trong khi tiến trình người dùng sẽ tính toán cái gì đó, nhưng điều này dẫn tới một điều còn tệ hại hơn: làm cách nào tiến trình người dùng có thể biết được quá trình gửi dữ liệu đã hoàn thành, và nó có thể sử dụng lại bộ đệm? Hệ thống có thể phát ra một tín hiệu hoặc ngắt mềm, nhưng làm vậy sẽ gây khó khăn cho lập trình, và có thể gây ra hiện tượng đua tranh. Một giải pháp tốt hơn là để kernel sao chép dữ liệu sang bộ đệm của nó, tương tự như trên hình 5-14(c) (nhưng theo một cách khác), và đánh thức tiến trình ngay lập tức. Bây giờ sẽ không còn vấn đề gì xảy ra khi quá trình vào/ra hoàn thành. Tiến trình người dùng có thể sử dụng lại bộ đệm ngay khi nó thức dậy.

Bộ đệm là một kỹ thuật được sử dụng rộng rãi, nhưng nó cũng có những hạn chế. Nếu phải đọc bộ đệm quá nhiều lần thì hiệu suất chung sẽ giảm xuống. Ví dụ, xét một mạng như trên hình 5-15. Ở đây người dùng thực hiện một hàm hệ thống để ghi dữ liệu lên mạng. Kernel sẽ sao chép gói tin vào bộ đệm của nó, để tiến trình người dùng có thể tiếp tục ngay lập tức (bước 1).



Hình 5-15 Mạng có thể cần tới nhiều lần sao chép gói tin.

Khi driver được gọi, nó sẽ sao chép gói tin sang controller để gửi đi (bước 2). Lý do nó không gửi trực tiếp dữ liệu ra đường dây từ bộ nhớ kernel là vì mỗi khi truyền một gói tin, nó phải thực hiện với một tốc độ không đổi. Driver không thể đảm bảo được việc truyền từ bộ nhớ với tốc độ cố định, vì các kênh DMA và các thiết bị vào/ra khác có thể "ăn cắp" mất nhiều chu kỳ. Chỉ cần

một word đến chậm là sẽ hỏng cả gói tin. Bằng cách sử dụng bộ đệm cho các gói tin bên trong controller, ta có thể tránh được vấn đề này.

Sau khi gói tin đã được sao chép vào bộ đệm bên trong controller, nó sẽ được gửi ra mạng (bước 3). Các bít sẽ đến được nơi nhận sau khi được gửi đi, nên ngay khi bít cuối cùng được gửi xong, nó sẽ đến được bên nhận, ở đó gói tin cũng được đưa vào bộ đệm trong controller. Tiếp theo, gói tin sẽ được sao chép vào bộ đệm kernel của bên nhận (bước 4). Cuối cùng, nó được sao chép vào bộ đệm của tiến trình nhận (bước 5). Thông thường, bên nhận sẽ gửi tín hiệu Ack trở lại (để báo nhận thành công). Khi bên gửi nhận được Ack, nó có thể tiếp tục gửi gói tin khác. Tuy nhiên, cần nói rõ là tất cả các sao chép này sẽ làm chậm tốc độ truyền khá nhiều, vì tất cả các bước phải được thực hiện tuần tự.

Thông báo lỗi

Lỗi là một vấn đề khá phổ biến trong hoạt động vào/ra (các hoạt động khác thường ít gặp lỗi hơn). Khi chúng xảy ra, hệ điều hành phải xử lý chúng theo cách tốt nhất có thể. Có nhiều lỗi mang tính đặc thù của thiết bị, và phải được xử lý bởi driver thích hợp. Nhưng nhìn chung thì việc xử lý lỗi là độc lập với thiết bị.

Một lớp các lỗi vào/ra thuộc về lỗi lập trình. Chúng xảy ra khi tiến trình yêu cầu một điều gì đó không thể thực hiện, ví dụ như ghi dữ liệu vào một thiết bị nhập (bàn phím, chuột, máy quét…), hay đọc dữ liệu từ một thiết bị xuất (như máy in, máy vẽ…). Các lỗi khác như cung cấp địa chỉ bộ đệm không hợp lệ hoặc khác tham số, hay thiết bị không hợp lệ (ví dụ yêu cầu truy nhập ổ đĩa 3, trong khi hệ thống chỉ có hai ổ đĩa). Việc giải quyết các lỗi trên thường theo khuynh hướng: gửi thông báo lỗi tới đối tượng đã yêu cầu.

Một lớp lỗi khác bao gồm các lỗi vào/ra thực sự, ví dụ như cố ghi dữ liệu vào vùng đĩa đã bị hỏng, hay cố đọc dữ liệu từ máy quay kỹ thuật số đã tắt. Trong các trường hợp đó, driver sẽ xác định điều cần phải làm. Nếu driver không biết phải làm gì thì có thể sẽ phải chuyển vấn đề đó cho phần mềm độc lập thiết bị.

Cái mà phần mềm sẽ làm còn tuỳ thuộc vào môi trường và bản chất của lỗi. Nếu là một lỗi đọc dữ liệu đơn giản, và có thể tương tác với người dùng, thì nó sẽ cho hiện một hộp thoại yêu cầu người dùng lựa chọn. Các tuỳ chọn thường là cố lặp lại thao tác một số lần nào đó, bỏ qua lỗi, hay kết thúc tiến trình gọi. Nếu không tương tác được với người dùng, có lẽ lựa chọn thực tế duy nhất là kết thúc hàm hệ thống với một mã lỗi.

Tuy nhiên, một số lỗi có thể không xử lý được theo cách đó. Ví dụ khi một cấu trúc dữ liệu quan trọng (như thư mục gốc hay danh sách vùng đĩa trống) bị phá huỷ. Trong trường hợp này, hệ thống sẽ phải hiển thị thông báo rồi kết thúc.

Phân phối và giải phóng các thiết bị chuyên biệt

Một số thiết bị, như ổ ghi CD, chỉ cho phép một tiến trình sử dụng nó tại một thời điểm. Điều đó khiến hệ điều hành phải kiểm tra các yêu cầu sử dụng thiết bị, và sẽ chấp nhận hoặc từ chối chúng, tuỳ thuộc vào tình trạng thiết bị được yêu cầu có sẵn sàng hay không. Một giải pháp đơn giản để xử lý các yêu cầu này là tiến trình gọi sẽ tiến hành mở file đặc biệt ứng với thiết bị. Nếu thiết bị chưa sẵn sàng, hành động mở sẽ thất bại. Việc đóng một thiết bị chuyên biệt như vậy sẽ giải phóng cho nó.

Một giải pháp khác là áp dụng các kỹ thuật đặc biệt để quản lý thiết bị chuyên biệt. Việc cố giành được một thiết bị không sẵn sàng sẽ khiến tiến trình gọi bị dừng (chứ không bị kết thúc). Tiến trình bị dừng sẽ được đưa vào hàng đợi. Sau đó, dù lâu hay chóng, khi thiết bị được yêu cầu đã sẵn sàng, tiến trình đầu tiên trong hàng đợi sẽ được phép sử dụng thiết bị và tiếp tục thi hành.

Kích thước khối dữ liệu độc lập với thiết bị

Các ổ đĩa khác nhau có thể có kích thước sector khác nhau. Phần mềm độc lập thiết bị sẽ phải che dấu được điều này, và cung cấp các khối dữ liệu có kích thước đồng nhất cho các lớp cao hơn. Ví dụ như có thể nối vài sector thành một khối dữ liệu logic. Theo cách này, các lớp cao hơn chỉ phải xử lý các thiết bị trừu tượng có cùng kích thước khối dữ liệu logic, hoàn toàn độc lập với

kích thước của các sector vật lý. Tương tự như vậy, một số thiết bị kí tự gửi dữ liệu lần lượt từng byte một (ví dụ như modem), trong khi các thiết bị khác gửi dữ liệu với đơn vị lớn hơn (ví dụ như card giao diện mạng). Những khác biệt như vậy có thể được ẩn đi.

5.3.4 Phần mềm vào/ra trong không gian người dùng (User-Space I/O Software)

Mặc dù phần lớn các phần mềm vào/ra nằm trong hệ điều hành, vẫn có một bộ phận nhỏ của chúng (bao gồm các thư viện) được liên kết với các chương trình của người dùng, hay thậm chí toàn bộ chương trình được chạy bên ngoài kernel. Các lời gọi hệ thống, kể cả các lời gọi tới hàm hệ thống vào/ra, thường được tạo ra bởi các thủ tục trong thư viện. Khi một chương trình C sử dụng lệnh:

```
count = write(fd, buffer, nbytes);
```

thì thủ tục thư viện *write* sẽ được liên kết với chương trình, và được chứa trong chương trình nhị phân nằm trong bộ nhớ khi chạy. Tập hợp tất cả các thủ tục thư viện này là một phần của hệ thống vào/ra.

Trong khi các thủ tục này không phải làm gì hơn ngoài việc đặt các tham số của chúng vào một vị trí thích hợp cho lời gọi hệ thống, thì vẫn có các thủ tục khác phải làm việc thực sự. Định dạng cụ thể của đầu vào và đầu ra được thực hiện bởi các thủ tục thư viện. Một ví dụ từ C là thủ tục *printf*, đầu vào của nó có định dạng chuỗi và một vài biến khác, tạo thành một chuỗi ASCII, rồi gọi thủ tục `write` để hiện chuỗi ra ngoài. Dưới đây là một ví dụ:

```
printf("The square of %3d is %6d\n", i, i*i);
```

Thủ tục này định dạng một chuỗi gồm 14 kí tự "The square of ", sau đó là giá trị của *i* dài 3 kí tự, rồi đến chuỗi "is" dài 4 kí tự, tiếp theo là i^2 dài sáu kí tự, cuối cùng là kí tự xuống dòng.

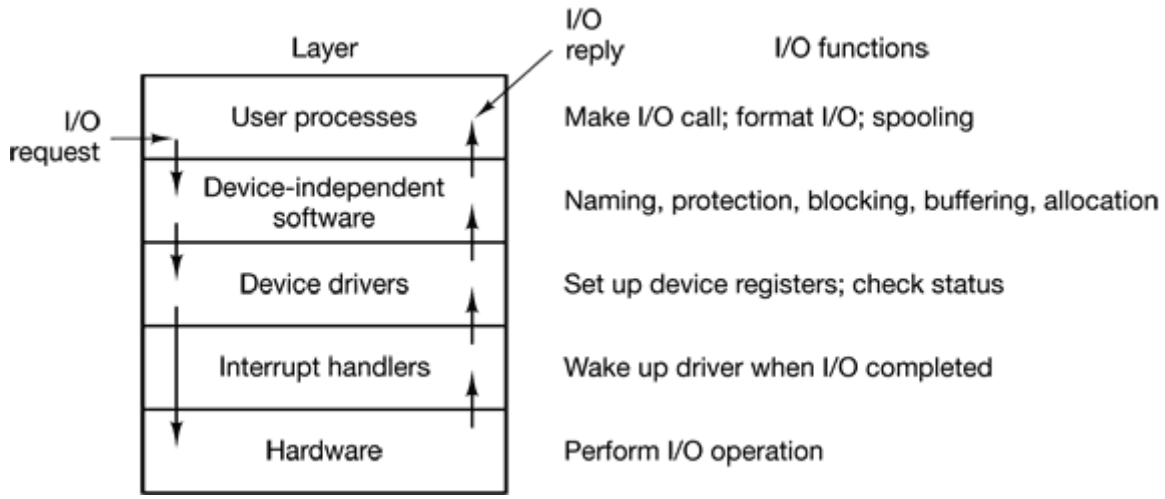
Một ví dụ tương tự là thủ tục nhập dữ liệu *scanf*, nó đọc dữ liệu đầu vào rồi cất dữ liệu vào các biến theo một định dạng chuỗi giống như ở thủ tục *printf*. Thư viện vào/ra chuẩn có chứa nhiều thủ tục vào/ra, và tất cả đều chạy như một bộ phận của chương trình người dùng.

Không phải tất cả các phần mềm vào/ra cấp người dùng đều có chứa các thủ tục của thư viện. Có một hệ thống quan trọng khác gọi là **spooling**. Spooling là một cách thức để xử lý các thiết bị vào/ra trong hệ thống đa chương trình. Xét một thiết bị spooling điển hình: Máy in. Mặc dù về mặt kỹ thuật người dùng có thể gán một file đặc biệt cho máy in, giả sử một tiến trình mở nó rồi không làm gì trong vài giờ sau đó. Sẽ không tiến trình nào khác có thể in ấn được.

Thay vì làm như vậy, người ta sẽ sử dụng một tiến trình đặc biêt, gọi là **daemon**, và một thư mục đặc biệt, gọi là **thư mục spooling (spooling directory)**. Để in một file, trước hết tiến trình sẽ đặt toàn bộ file cần in vào thư mục spooling. Sau đó thì daemon (tiến trình duy nhất có quyền sử dụng file đặc biệt của máy in) sẽ in các file trong thư mục này. Việc bảo vệ file đặc biệt (tránh sự truy nhập trực tiếp của người dùng) sẽ giúp loại trừ sự lãng phí thiết bị do bị một tiến trình chiếm dụng quá lâu.

Spooling không chỉ được áp dụng cho máy in. Nó có thể được dùng cho nhiều tình huống khác. Ví dụ, quá trình truyền file qua mạng thường sử dụng daemon mạng. Để gửi một file tới đâu đó, người dùng sẽ đặt file vào trong thư mục spooling mạng. Sau đó, daemon mạng sẽ lấy nó ra để gửi đi. Một ứng dụng cụ thể của truyền file spooling là hệ thống USENET News. Mạng này có chứa hàng triệu máy trên khắp thế giới, kết nối với nhau qua Internet. Có hàng ngàn nhóm tin tức với nhiều chủ đề khác nhau. Để gửi một bản tin mới, người dùng sẽ gọi một chương trình gửi tin, nó sẽ nhận bản tin và gửi bản tin này vào thư mục spooling, từ đó bản tin được gửi tới các máy khác. Toàn bộ hệ thống tin tức này chạy bên ngoài hệ điều hành.

Hình 5-16 minh hoạ một hệ thống vào/ra, với tất cả các lớp và chức năng chính của từng lớp. Lớp dưới cùng là phần cứng, tiếp theo là lớp các trình xử lý ngắt, driver thiết bị, phần mềm độc lập thiết bị, và cuối cùng là các tiến trình của người dùng.



Hình 5-16 Các lớp của hệ thống vào/ra và chức năng chính của mỗi lớp.

Các mũi tên trên hình 5-16 xác định hướng điều khiển. Ví dụ, khi một chương trình người dùng muốn đọc một khối dữ liệu từ file, hệ điều hành sẽ được gọi để thực hiện yêu cầu này. Phần mềm độc lập thiết bị sẽ tìm khối dữ liệu trong bộ đệm. Nếu không tìm thấy, nó sẽ gọi chương trình điều khiển thiết bị (driver) để gửi yêu cầu tới phần cứng, nhằm đọc dữ liệu từ đĩa. Sau đó tiến trình sẽ bị dừng cho tới khi quá trình đọc đĩa hoàn tất.

Khi thực hiện đọc đĩa xong, phần cứng sẽ phát ra một tín hiệu ngắt. Trình xử lý ngắt sẽ được chạy để xác định điều vừa xảy ra (tức là xác định ngắt phát ra từ thiết bị nào). Sau đó nó sẽ đọc trạng thái của thiết bị, rồi đánh thức tiến trình đang ngủ để kết thúc quá trình vào/ra, tiến trình người dùng sẽ tiếp tục.

5.4 CÁC Ổ ĐĨA

Bây giờ ta sẽ bắt đầu nghiên cứu về một số thiết bị vào/ra thực tế. Trước hết ta sẽ tìm hiểu các ổ đĩa. Sau đó ta sẽ nghiên cứu về đồng hồ, bàn phím, và màn hình.

5.4.1 Phần cứng đĩa

Có rất nhiều loại đĩa. Phổ biến nhất là đĩa từ (đĩa cứng và đĩa mềm). Chúng có tốc độ đọc ghi nhanh, và là loại bộ nhớ thứ cấp lý tưởng (dùng trong phân trang, chứa hệ thống file…). Đôi khi người ta sử dụng tập hợp các đĩa này để tạo thành thiết bị lưu trữ có độ tin cậy cao. Để phân phối chương trình, dữ liệu, và phim ảnh, người ta còn sử dụng các loại đĩa quang (CD-ROM, CD-Recordable và DVD). Trong các mục tiếp theo ta sẽ mô tả về phần cứng, rồi tới phần mềm dùng cho các thiết bị này.

Đĩa từ

Đĩa từ được tổ chức thành các cylinder (trụ), mỗi cylinder lại chứa nhiều track (rãnh) ứng với các đầu đọc khác nhau. Các track được chia thành nhiều sector (cung từ), số lượng sector trên mỗi track thường từ 8 tới 32 (đối với đĩa mềm), và khoảng vài trăm (đối với đĩa cứng). Số lượng đầu đọc (head) có thể từ 1 tới 16.

Một số đĩa từ có rất ít thành phần điện tử, và chỉ có thể tạo ra các dòng bít nối tiếp đơn giản. Đối với các đĩa này thì bộ điều khiển đĩa (controller) phải làm hầu hết mọi việc. Trên các đĩa khác, cụ thể như các đĩa **IDE (Integrated Drive Electronics)**, ổ đĩa đã có sẵn một mạch vi điều khiển (microcontroller) đảm nhiệm một phần công việc, điều này sẽ cho phép bộ điều khiển đĩa làm việc với tập lệnh ở cấp cao hơn.

Một tính năng quan trọng của bộ điều khiển đĩa là khả năng thực hiện tìm kiếm dữ liệu trên hai hoặc nhiều ổ đĩa cùng một lúc. Điều này được gọi là **tìm kiếm xếp chồng (overlapped seeks)**. Trong khi bộ điều khiển và phần mềm đang chờ quá trình tìm kiếm trên ổ đĩa thứ nhất

hoàn tất, bộ điều khiển có thể khởi tạo quá trình tìm kiếm trên một ổ đĩa khác. Nhiều bộ điều khiển đĩa cũng có thể đọc hoặc ghi trên một ổ đĩa trong khi vẫn đang tìm kiếm dữ liệu trên một hoặc nhiều ổ khác, nhưng bộ điều khiển đĩa mềm không thể đọc hay ghi vào hai ổ đĩa tại cùng một thời điểm. (Đọc hoặc ghi đòi hỏi bộ điều khiển đĩa mềm phải di chuyển các bít theo đơn vị thời gian là micro giây, nên mỗi lần truyền dữ liệu sẽ tiêu tốn hầu hết công suất tính toán của nó). Trên đĩa cứng thì khác, nó có các bộ điều khiển tích hợp, và trong một hệ thống có nhiều đĩa cứng thì chúng vẫn có thể hoạt động đồng thời, ít nhất là trong quá trình truyền dữ liệu giữa đĩa và bộ đệm của bộ điều khiển đĩa. Tuy nhiên, chỉ có thể thực hiện một quá trình truyền dữ liệu giữa bộ điều khiển và bộ nhớ chính tại một thời điểm. Khả năng thực hiện hai hoặc nhiều hoạt động đồng thời giúp làm giảm thời gian truy nhập trung bình khá nhiều.

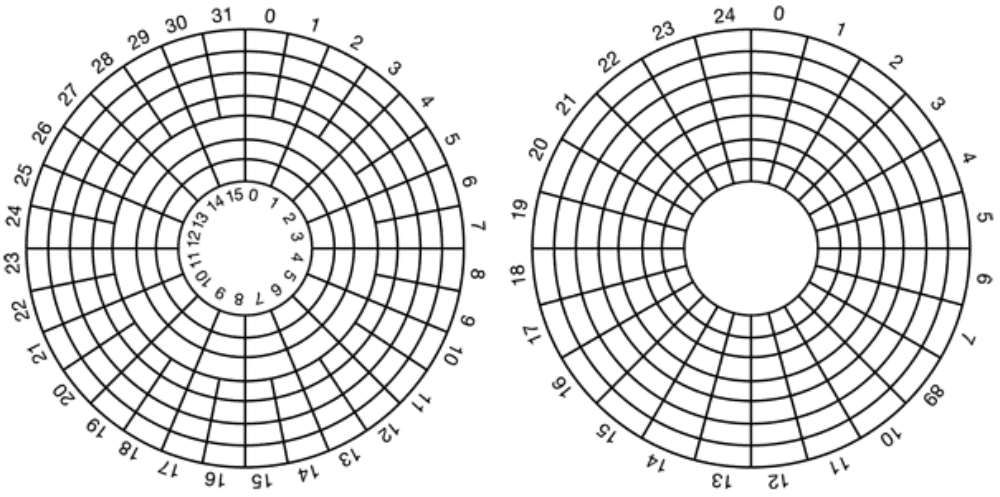
Hình 5-17 trình bày một bảng so sánh các tham số của ổ đĩa trên chiếc IBM PC đầu tiên với một ổ đĩa cứng hiện đại, để thấy được sự thay đổi của các ổ đĩa sau hai thập kỷ. Điều thú vị là không phải tất cả các tham số đều được thay đổi, có tham số chỉ thay đổi một chút. Thời gian tìm kiếm trung bình được giảm đi bảy lần, tốc độ truyền nhanh hơn 1300 lần, trong khi dung lượng đĩa tăng gấp 50 000 lần. Có những cải tiến phải được thực hiện dần từng bước, còn mật độ bít trên bề mặt đĩa lại tăng nhanh hơn rất nhiều.

Các tham số	Đĩa mềm IBM 360 -KB	Đĩa cứng WD 18300
Số lượng cylinder	40	10601
Số Track trên 1 cylinder	2	12
Số Sector trên 1 track	9	281 (avg)
Tổng số Sector trên đĩa	720	35742000
Số Byte trên 1 sector	512	512
Dung lượng đĩa	360 KB	18.3 GB
Thời gian tìm kiếm (các cylinder liền kề)	6 msec	0.8 msec
Thời gian tìm kiếm trung bình	77 msec	6.9 msec
Thời gian quay (Rotation time)	200 msec	8.33 msec
Thời gian Motor dừng (hay khởi động)	250 msec	20 sec
Thời gian truyền 1 sector	22 msec	17 µsec

Hình 5-17 Các tham số của đĩa mềm IBM PC 360 KB và đĩa cứng Western Digital WD 18300.

Một điều cần lưu ý khi xem xét các tham số của đĩa cứng hiện đại là: thông số hình học của nó (dùng trong phần mềm điều khiển đĩa) có thể khác so với định dạng vật lý. Ở các đĩa cũ hơn, số lượng sector trên một track là giống nhau đối với tất cả các cylinder. Các đĩa hiện đại được chia thành nhiều vùng, các vùng bên ngoài sẽ có nhiều sector hơn các vùng bên trong. Hình 5-18(a) minh hoạ một đĩa nhỏ có hai vùng. Vùng bên ngoài có 32 sector trên một track, vùng bên trong có 16 sector trên 1 track. Một đĩa thực tế, như WD 18300 chẳng hạn, thường có 16 vùng, với số lượng sector tăng khoảng 4% khi đi từ vùng bên trong ra vùng bên ngoài liền kề.

Để che dấu sự phức tạp này, hầu hết các đĩa hiện đại đều sử dụng một kiến trúc hình học ảo trong giao tiếp với hệ điều hành. Một phần mềm sẽ chuyển đổi các thông tin phức tạp đó về dạng đơn giản như *x* cylinder, *y* head, và *z* sector trên 1 track. Bộ điều khiển đĩa sẽ ánh xạ lại các yêu cầu truy nhập dạng (*x*, *y*, *z*) về các thông số cylinder, head, và sector thực tế. Kiến trúc hình học ảo của ổ đĩa vật lý ở hình 5-18(a) được trình bày ở hình 5-18(b). Trong cả hai trường hợp, đĩa đều có 192 sector, chỉ có trật tự sắp xếp của chúng là khác nhau.



Hình 5-18 (a) Kiến trúc hình học thực của một đĩa có hai vùng.
(b) Kiến trúc hình học ảo của đĩa này.

Đối với các máy tính Pentium, giá trị cực đại của ba tham số trên thường là (65535, 16, và 63), do phải giữ sự tương thích với các máy IBM PC cũ. Các máy này sử dụng các trường có kích thước là 16, 4, và 6 bít để chứa ba tham số trên, trong đó các cylinder và head được đánh số bắt đầu từ 0, còn sector được đánh số từ 1. Với các tham số này, và kích thước mỗi sector là 512 byte, thì dung lượng tối đa của đĩa sẽ là 31.5 GB. Để vượt qua giới hạn đó, nhiều ổ đĩa hiện nay đã cung cấp một hệ thống gọi là **địa chỉ hoá khối logic (logical block addressing)**, trong đó các sector trên đĩa được đánh số liên tục từ 0, và không cần quan tâm tới kiến trúc hình học của đĩa.

RAID

Hiệu suất CPU đã tăng theo hàm mũ trong suốt thập kỷ qua, cứ 18 tháng nó lại tăng gần như gấp đôi. Nhưng hiệu suất đĩa lại không được như vậy. Vào những năm 1970, thời gian tìm kiếm trung bình trên các ổ đĩa của máy tính mini là từ 50 đến 100 ms. Còn bây giờ thời gian tìm kiếm nhỏ hơn 10 ms. Trong phần lớn các ngành công nghiệp (ví dụ như ô tô hay máy bay), việc tăng hiệu suất từ 5 tới 10 lần trong hai thập kỷ sẽ là một sự kiện trọng đại, nhưng trong ngành công nghiệp máy tính thì đó là một điều đáng xấu hổ. Do đó, khoảng cách giữa hiệu suất CPU và hiệu suất đĩa đang ngày càng gia tăng.

Như ta đã thấy, xử lý song song ngày càng được sử dụng nhiều để tăng hiệu suất CPU. Và việc thực hiện vào/ra song song cũng có thể là một ý tưởng tốt. Trong bài viết năm 1988, Patterson và cộng sự đã đề nghị sáu phương pháp để tăng hiệu suất đĩa, độ tin cậy, hoặc cả hai (Patterson và cộng sự, 1988). Các ý kiến này đã nhanh chóng được áp dụng trong công nghiệp, và tạo ra một lớp các thiết bị vào/ra mới gọi là **RAID**. Patterson và cộng sự định nghĩa RAID là **Redundant Array of Inexpensive Disks**, nhưng các hãng sản xuất đã định nghĩa lại chữ I thành "Independent" thay cho "Inexpensive" (có lẽ vì họ muốn dùng các đĩa đắt tiền?). Do một kẻ tội phạm cũng có lúc sẽ được dùng đến (giống như RISC đối với CISC – theo Patterson), nên kẻ đối lập ở đây chính là **SLED (Single Large Expensive Disk)**!

Ý tưởng cơ bản của RAID là lắp đặt một hộp chứa nhiều đĩa vào máy tính, chẳng hạn như một server lớn, thay thế bộ điều khiển đĩa bằng bộ điều khiển RAID, sao chép dữ liệu thông qua RAID, mọi hoạt động vẫn diễn ra như bình thường. Nói cách khác, đối với hệ điều hành thì một hệ thống RAID trông cũng giống như SLED, nhưng có hiệu suất cao hơn và có độ tin cậy cao hơn. Do các đĩa SCSI có hiệu suất cao, giá thành thấp, và có khả năng lắp tới 7 ổ đĩa trên một bộ điều khiển (hoặc lên tới 15 ổ đối với SCSI mở rộng), nên hầu hết các RAID đều bao gồm một bộ

điều khiển RAID SCSI và một hộp các đĩa SCSI, mà dưới con mắt của hệ điều hành thì chúng giống như một đĩa đơn lớn. Theo cách này, các phần mềm cũng không phải thay đổi gì khi sử dụng RAID, đó là một điều rất quan trọng đối với các nhà quản trị hệ thống.

Thêm vào đó, để có thể hoạt động giống một đĩa đơn, các hệ thống RAID đều có đặc tính phân phối dữ liệu giữa các ổ đĩa, để cho phép hoạt động song song. Các giải pháp khác nhau về vấn đề này cũng do Patterson và cộng sự đưa ra, và chúng được biết với các cái tên từ RAID mức 0 tới RAID mức 5. Ngoài ra còn có các mức khác nữa mà ta không thảo luận ở đây. Thuật ngữ "mức" có lẽ cũng chưa chính xác lắm, do không có sự phân cấp nào cả; chỉ đơn giản là sáu hình thức tổ chức khác nhau.

RAID mức 0 được minh hoạ ở hình 5-19(a). Một đĩa đơn ảo sẽ được tạo ra nhờ RAID, nó được chia thành nhiều phần (strip), mỗi phần có k sector. Các sector từ 0 đến $k - 1$ ứng với strip 0, sector từ k đến $2k - 1$ ứng với strip 1,… Với $k = 1$ thì mỗi strip là một sector; với $k = 2$ thì mỗi strip có hai sector… RAID mức 0 sắp xếp các strip liên tiếp nhau trên các đĩa khác nhau theo kiểu round-robin, như mô tả ở hình 5-19(a) với bốn ổ đĩa. Sự phân phối dữ liệu trên nhiều ổ khác nhau như vậy được gọi là **striping**. Ví dụ, nếu phần mềm phát ra một lệnh để đọc một khối dữ liệu gồm bốn strip liên tiếp, bộ điều khiển RAID sẽ chia nhỏ lệnh này thành bốn lệnh riêng rẽ, mỗi lệnh sẽ đọc dữ liệu trên một đĩa, và được thực hiện song song. Phần mềm sẽ không biết gì về quá trình vào/ra song song này cả.

RAID mức 0 hoạt động tốt nhất khi có các yêu cầu lớn về dữ liệu, càng lớn càng tốt. Nếu dữ liệu yêu cầu có kích thước gấp nhiều lần kích thước của strip, một số ổ đĩa sẽ nhận được nhiều yêu cầu hơn các ổ khác, nên khi chúng hoàn thành yêu cầu thứ nhất thì sẽ phải thực hiện yêu cầu thứ hai. Để làm được điều đó thì bộ điều khiển sẽ chia nhỏ yêu cầu, và gửi các lệnh thích hợp tới các ổ đĩa thích hợp theo đúng trình tự, và sau đó tập hợp lại kết quả một cách chính xác. Hiệu suất đạt được là tuyệt vời, và sự thực hiện cũng không có gì khó khăn cả.

RAID mức 0 hoạt động tệ nhất với các hệ điều hành chỉ đọc ghi dữ liệu theo từng sector. Kết quả vẫn chính xác, nhưng không có sự thực hiện song song, và do đó không cải thiện được hiệu suất. Một nhược điểm nữa là độ tin cậy của nó kém hơn SLED. Nếu một hệ thống RAID gồm bốn ổ đĩa, mỗi ổ đĩa có thời gian hoạt động ổn định trung bình là 20000 giờ, thì xác suất gặp lỗi sẽ tăng lên bốn lần. Như vậy cứ khoảng 5000 giờ thì một ổ đĩa có thể bị lỗi và toàn bộ dữ liệu có thể bị mất. Một hệ thống SLED có tuổi thọ trung bình 20000 giờ sẽ đáng tin hơn gấp bốn lần. Vì không có thông tin dự phòng (redundancy) nào trong thiết kế này, nên nó chưa phải là RAID thực sự.

Giải pháp tiếp theo, RAID mức 1, được trình bày ở hình 5-19(b), nó thực sự là RAID. Số đĩa được tăng gấp đôi, như vậy sẽ có bốn đĩa chính thức và bốn đĩa dùng để dự phòng. Khi ghi dữ liệu, tất cả các strip được ghi làm hai bản. Còn khi đọc thì sử dụng bản nào cũng được, cũng có thể đọc song song trên cả hai bản. Do đó hiệu suất ghi dữ liệu sẽ không cao hơn so với đĩa đơn, nhưng hiệu suất đọc có thể tăng gấp đôi. Khả năng chống lỗi thì tuyệt vời: nếu có một ổ đĩa bị hỏng thì chỉ việc sử dụng bản sao còn lại để thay thế. Việc khắc phục hệ thống cũng rất đơn giản, chỉ việc thay một ổ đĩa mới, rồi sao chép toàn bộ dữ liệu từ ổ dự phòng vào đó.

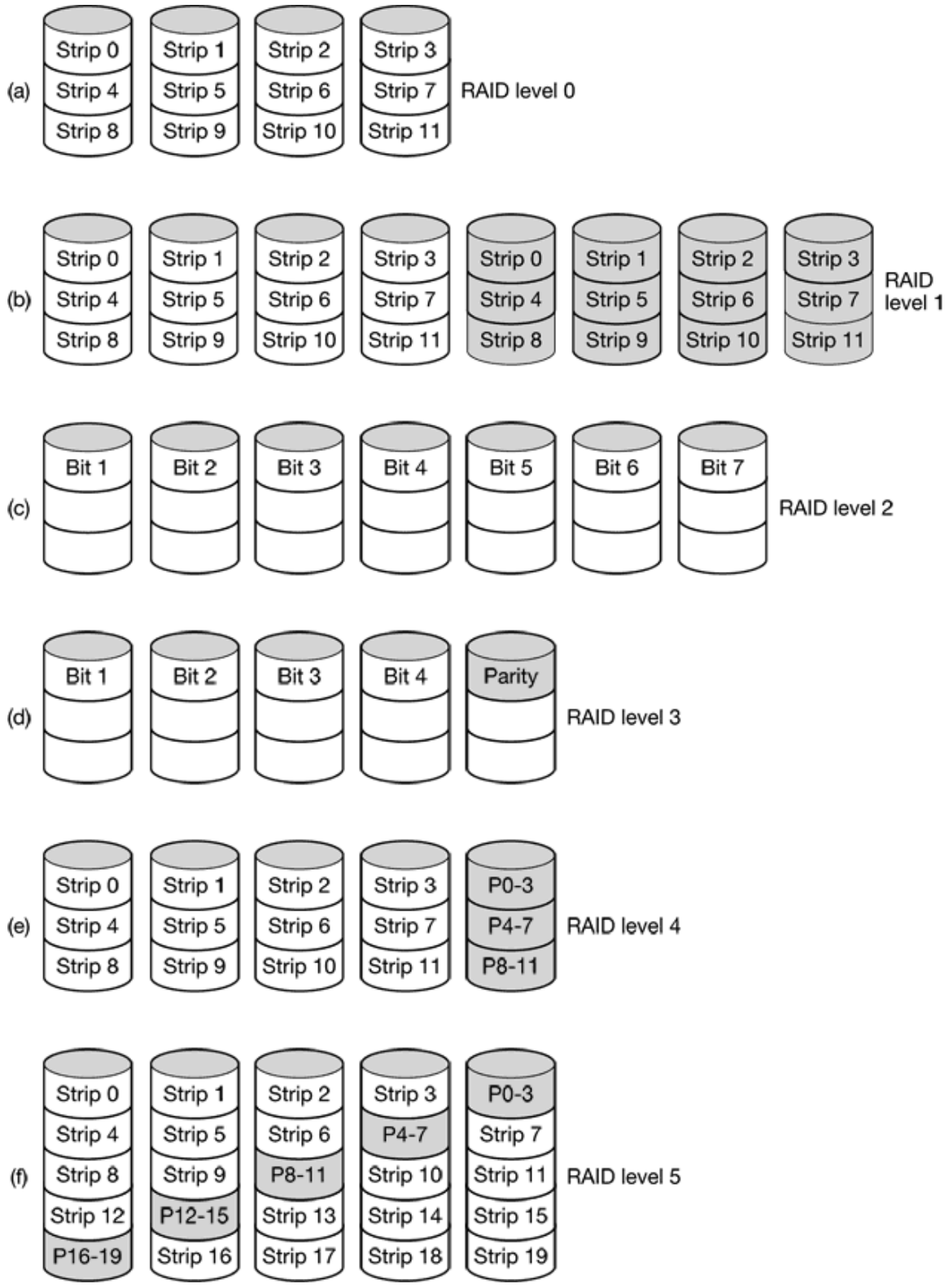
Không giống như RAID mức 0 và mức 1 (sử dụng các strip gồm nhiều sector), RAID mức 2 hoạt động dựa trên các word dữ liệu, thậm chí có thể là từng byte. Hãy tưởng tượng việc chia mỗi byte của đĩa đơn ảo thành từng cặp 4 bít, rồi lắp thêm mã Hamming vào để tạo thành word 7 bít, trong đó các bít 1, 2, và 4 là các bít chẵn lẻ. Xem hình 5-19(c), bảy ổ đĩa được đồng bộ về vị trí của cánh tay đĩa và chiều quay. Word dữ liệu 7 bít, gồm cả mã Hamming, sẽ được ghi vào bảy đĩa đó, mỗi bít trên một đĩa.

Máy CM-2 của The Thinking Machines sử dụng giải pháp này, với các word dữ liệu dài 32 bít và 6 bít chẵn lẻ tạo thành word Hamming dài 38 bít, cộng thêm một bít chẵn lẻ nữa cho cả word, 39 bít đó sẽ được ghi vào 39 ổ đĩa. Tổng thông lượng là rất lớn, vì trong khoảng thời gian để ghi 1 sector nó có thể ghi được tới 32 sector dữ liệu. Hơn nữa, nếu có một ổ đĩa bị hỏng thì cũng không gây ra điều gì nghiêm trọng, vì cũng chỉ là mất một bít trong tổng số 39 bít, đôi khi có thể sử dụng các mã Hamming để khắc phục lỗi này ngay lập tức.

Tuy nhiên, giải pháp này đòi hỏi phải đồng bộ được sự quay của tất cả các ổ đĩa, và nó chỉ có ý nghĩa khi sử dụng một số lượng lớn các đĩa (thậm chí với 32 đĩa chứa dữ liệu và 6 đĩa chứa bít chẵn lẻ, chi phí lên tới 19%). Nó cũng đòi hỏi phải có nhiều bộ điều khiển, do nó phải thực hiện tính toán các mã Hamming với từng bít.

RAID mức 3 là một phiên bản đơn giản của RAID mức 2. Nó được minh hoạ trên hình 5-19(d). Chỉ có một bít chẵn lẻ được tính toán cho mỗi word dữ liệu, rồi ghi vào ổ đĩa chẵn lẻ. Giống như RAID mức 2, các ổ đĩa phải được đồng bộ chính xác, do mỗi word dữ liệu được cất rải rác trên nhiều ổ đĩa.

Thoạt nhìn ta có thể cho rằng việc chỉ sử dụng một bít chẵn lẻ có thể giúp phát hiện lỗi, chứ không giúp sửa lỗi được. Trong trường hợp các lỗi xuất hiện ngẫu nhiên thì điều này là đúng. Tuy nhiên, trong trường hợp có một ổ đĩa bị hỏng, thì 1 bít chẵn lẻ này có thể giúp khắc phục lỗi, do ta đã biết vị trí của bít lỗi. Khi một ổ đĩa hỏng, bộ điều khiển chỉ việc giả thiết tất cả các bít của nó bằng 0. Nếu word thu được có bít chẵn lẻ bị sai, thì chứng tỏ bít trên ổ đĩa hỏng có giá trị bằng 1. Mặc dù cả RAID mức 2 và mức 3 đều có tốc độ dữ liệu rất cao, nhưng số lượng các yêu cầu vào/ra mà nó có thể xử lý trong mỗi giây cũng chẳng cao hơn so với một ổ đĩa đơn.



Hình 5-19 RAID từ mức 0 tới mức 5. Các ổ đĩa dự phòng và chẵn lẻ được tô đậm.

RAID mức 4 và 5 lại làm việc với các strip, chứ không làm việc với các word riêng lẻ, và không đòi hỏi phải đồng bộ các ổ đĩa. RAID mức 4 [xem hình 5-19(e)] cũng giống như RAID mức 0, nhưng có thêm một ổ đĩa để chứa strip chẵn lẻ của các strip dữ liệu. Ví dụ, nếu mỗi strip có *k* byte, thì tất cả các strip dữ liệu liên quan sẽ được XOR với nhau, kết quả thu được là một

strip chẵn lẻ dài *k* byte. Nếu một ổ đĩa bị hỏng, byte bị mất có thể được khôi phục lại nhờ thông tin trên ổ đĩa chẵn lẻ.

Thiết kế này giúp tăng cường sự an toàn nếu có ổ đĩa bị hỏng, nhưng lại rất phiền phức khi cần thực hiện những thay đổi nhỏ. Nếu muốn thay đổi nội dung một sector, sẽ phải đọc tất cả các ổ đĩa để tính toán lại các bít chẵn lẻ, và sau đó phải ghi lại chúng. Cũng có thể đọc các dữ liệu cũ và các bít chẵn lẻ cũ, rồi tính toán các bít chẵn lẻ mới từ chúng. Thậm chí trong trường hợp tối ưu nhất, một cập nhật nhỏ cũng đòi hỏi hai lần đọc và hai lần ghi.

Kết quả là ổ đĩa chẵn lẻ có thể bị quá tải, và có thể dẫn tới tắc nghẽn. Sự tắc nghẽn này có thể được giải quyết nhờ RAID mức 5, bằng cách phân phối các bít chẵn lẻ ra các ổ đĩa khác nhau, theo kiểu round-robin, như minh hoạ ở hình 5-19(f). Tuy nhiên, nếu xảy ra biến cố hỏng đĩa, thì việc khôi phục lại dữ liệu liệu trên ổ đĩa hỏng sẽ phức tạp hơn.

CD-ROM

Vài năm trở lại đây, các ổ đĩa quang ngày càng trở nên phổ biến. Chúng có mật độ ghi cao hơn các ổ đĩa từ truyền thống. Các đĩa quang đầu tiên được phát triển để ghi các chương trình truyền hình, nhưng chúng có thể được sử dụng như một thiết bị lưu trữ của máy tính. Do có khả năng lưu trữ rất lớn, đĩa quang đã được tập trung nghiên cứu, và chúng đã có những bước phát triển đáng kinh ngạc.

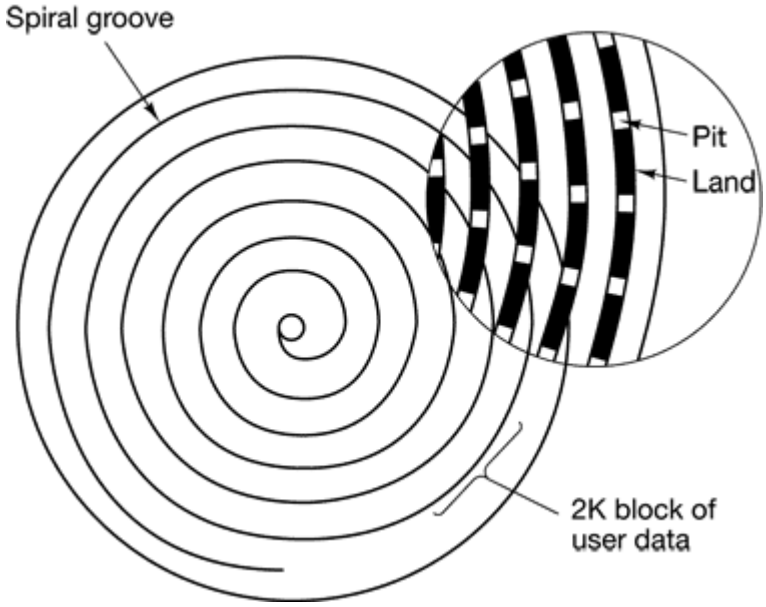
Thế hệ đĩa quang đầu tiên do hãng điện tử Philips của Hà Lan phát minh ra, dùng để chứa các bộ phim. Chúng rộng 30 cm và được gọi là LaserVision, nhưng chúng đã không được đón nhận, ngoại trừ ở Nhật Bản.

Năm 1980, Hãng Philips kết hợp với Sony phát minh ra đĩa CD (Compact Disc), nó đã nhanh chóng thay thế đĩa nhựa 33 1/3 vòng/phút trong ngành âm nhạc (trừ một số người sành sỏi vẫn thích đĩa nhựa hơn). Các chi tiết kỹ thuật về đĩa CD được tuân theo một tiêu chuẩn quốc tế (IS 10149), thường được gọi là **Red Book**, do vỏ của nó có màu đỏ. (Các tiêu chuẩn quốc tế - International Standards được đưa ra bởi tổ chức International Organization for Standardization, nó bao gồm một nhóm các tiêu chuẩn tương tự như ANSI, DIN… Mỗi tiêu chuẩn có một số hiệu IS). Các tiêu chuẩn này giúp cho các đĩa CD của các nhà sản xuất âm nhạc và các máy nghe đĩa của các hãng điện tử khác nhau có thể hoạt động cùng nhau. Tất cả các đĩa CD có đường kính 120 mm và dày 1.2 mm, với một lỗ thủng ở giữa rộng 15 mm. Đầu tiên, các đĩa audio CD đạt được thành công tại thị trường lưu trữ số tầm trung. Chúng được coi là có thể bền tới 100 năm. Hãy đợi đến năm 2080 để kiểm tra lại thông tin này.

Đĩa CD được sản xuất bằng cách dùng tia laser công suất lớn tạo các hố có đường kính 0.8 micrô mét trên một đĩa chủ bọc thuỷ tinh. Từ đĩa chủ này người ta sẽ tạo ra một khuôn đúc, nó có các điểm gồ lên tại vị trí của các hố do tia laser tạo ra. Bên trong khuôn người ta đổ đầy nhựa polycarbonate lỏng để tạo thành đĩa CD, đĩa này sẽ có các hố lõm giống hệt như trên đĩa chủ. Sau đó một lớp nhôm rất mỏng sẽ được phủ lên trên bề mặt polycarbonate, rồi quét lên đó một lớp sơn bảo vệ, trên cùng người ta sẽ dán một cái nhãn. Các hố lõm trên lớp polycarbonate được gọi là các **pit** (lõm), còn các vùng không bị khoét lõm được gọi là **land** (phẳng).

Khi đọc dữ liệu, một điốt laser sẽ chiếu tia sáng có bước sóng 0.78 micrô mét lên bề mặt đĩa. Tia laser được chiếu vào bề mặt polycarbonate, nên các pit sẽ hướng về phía tia laser chiếu tới. Vì các pit có độ cao bằng 1/4 bước sóng ánh sáng laser, nên ánh sáng phản xạ của một pit sẽ lệch pha một nửa bước sóng so với ánh sáng phản xạ từ bề mặt. Kết quả là hai thành phần này sẽ gây nhiễu lẫn nhau, và tạo ra ánh sáng yếu hơn so với ánh sáng phản xạ từ một land. Nhờ vậy máy đọc đĩa có thể phân biệt được các pit và land. Mặc dù có vẻ sẽ đơn giản hơn khi sử dụng một pit để chứa giá trị 0, và một land để chứa giá trị một, nhưng người ta thường lấy sự chuyển đổi thạng thái từ pit sang land (hoặc từ land sang pit) để biểu thị giá trị 1, các trạng thái còn lại sẽ ứng với giá trị 0. Giải pháp này có độ tin cậy cao hơn.

Các pit và land được ghi theo một đường xoắn ốc, bắt đầu từ vị trí gần với lỗ thủng ở giữa đĩa, rồi mở rộng ra phía rìa đĩa. Đường xoắn ốc cuốn quanh đĩa khoảng 22188 vòng (khoảng 600 vòng trên 1 mm). Nếu trải ra nó có thể dài tới 5.6 km. Đường xoắn này được minh hoạ trên hình 5-20.



Hình 5.20 Cấu trúc ghi dữ liệu trên CD-ROM.

Để có thể chơi nhạc với một tốc độ ổn định thì dòng các pit và land phải có một tốc độ không đổi. Do đó tốc độ quay của đĩa CD phải giảm xuống khi đầu đọc di chuyển từ bên trong ra bên ngoài đĩa. Ở bên trong đĩa, tốc độ quay là 530 vòng/phút để đạt được vận tốc dài mong muốn là 120 cm/s; còn ở bên ngoài đĩa tốc độ quay phải là 200 vòng/phút để đạt được vận tốc dài như trên. Vận tốc dài không đổi của đĩa CD cũng khác so với đĩa từ. Đĩa từ có vận tốc góc không đổi, độc lập với vị trí của đầu đọc. Ngoài ra, tốc độ 530 vòng/phút còn kém xa so với tốc độ của đĩa từ (từ 3600 tới 7200 vòng/phút).

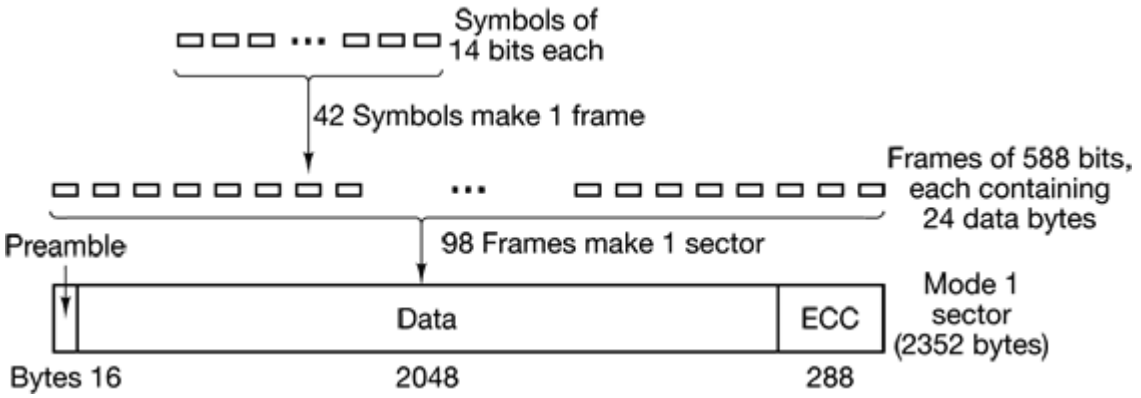
Năm 1984, hãng Philips và Sony nhận ra khả năng sử dụng đĩa CD trong việc lưu trữ dữ liệu máy tính, nên họ đã xuất bản cuốn **Sách Vàng (Yellow Book)** để định nghĩa một tiêu chuẩn cụ thể cho đĩa **CD-ROM** (**Compact Disc - Read Only Memory**). Để vẫn có thể chiếm lĩnh thị trường quan trọng là audio CD, các đĩa CD-ROM vẫn có cùng kích thước vật lý với các đĩa audio CD, cũng tương thích và về mặt quang học và cơ khí, chúng cũng được sản xuất bằng cách đúc khuôn polycarbonate. Kết quả là nó không chỉ làm giảm sự biến thiên của tốc độ motor, mà còn giảm giá thành sản xuất một đĩa CD-ROM xuống dưới 1 đô la.

Cuốn Yellow Book đã định nghĩa định dạng chung cho dữ liệu máy tính. Nó cũng cải thiện khả năng sửa lỗi của hệ thống. Định dạng cơ bản của CD-ROM bao gồm cả việc mã hoá các byte thành các kí tự (symbol) 14 bít. Như ta đã thấy ở trên, 14 bít là đủ để mã hoá Hamming cho dữ liệu 8 bít, với 2 bít để dành. Trên thực tế, người ta sử dụng một hệ thống mã hoá mạnh hơn. Việc ánh xạ từ 14 về 8 bít khi đọc dữ liệu được thực hiện bởi phần cứng, thông qua bảng tra cứu.

Ở cấp độ tiếp theo một nhóm 42 kí tự liên tiếp nhau được ghép thành một khung (frame) 588 bít. Mỗi khung có 192 bít dữ liệu (24 byte). 396 bít còn lại được dùng cho việc sửa lỗi và điều khiển. Giải pháp này được sử dụng cho các đĩa audio CD và CD-ROM.

Cuốn Yellow Book cũng ghép 98 khung thành một sector của CD-ROM, như minh hoạ trên hình 5-21. Mỗi sector này bắt đầu bằng 16 byte tiêu đề, trong đó 12 byte đầu tiên có giá trị là 00FFFFFFFFFFFFFFFFFFFF00 (hexadecimal), nó giúp máy đọc đĩa nhận ra vị trí bắt đầu của sector. Ba byte tiếp theo chứa số hiệu sector, nó rất quan trọng vì việc tìm kiếm dữ liệu trên đĩa CD-ROM (theo đường xoắn ốc) phức tạp hơn nhiều so với trên đĩa từ (có các vòng trong đồng tâm). Để tìm kiếm dữ liệu, phần mềm trên ở đĩa phải tính toán (gần đúng) vị trí cần đọc dữ liệu, chuyển đầu đọc tới đó, rồi bắt đầu tìm kiếm phần tiêu đề. Byte cuối cùng của tiêu đề để xác định mô hình hoạt động.

Cuốn Yellow Book định nghĩa hai mô hình hoạt động. Mô hình 1 được trình bày trên hình 5-21, với 16 byte tiêu đề, 2048 byte dữ liệu, và 288 byte mã sửa lỗi (mã xen chéo Reed-Solomon). Mô hình 2 ghép trường dữ liệu và trường ECC (error-correcting code) thành một trường dữ liệu dài 2336 byte dùng cho các ứng dụng không cần (hoặc không thể áp dụng) mã sửa lỗi, như trong các ứng dụng audio và video. Chú ý rằng để đạt được độ tin cậy hoàn hảo, người ta dùng tới ba cấp sửa lỗi khác nhau: sửa lỗi kí tự, sửa lỗi khung, và sửa lỗi sector. Các lỗi bít đơn có thể được sửa ở cấp thấp nhất, các lỗi loạt ngắn có thể được sửa ở cấp khung, và các lỗi còn lại có thể được xử lý ở cấp sector. Chi phí để đạt được độ tin cậy này tốn tới 98 khung 588 bít (7203 byte) cho 2048 byte dữ liệu, tức là hiệu suất chỉ có 28%.



Hình 5-21 Mô hình dữ liệu logic trên CD-ROM.

Các ổ CD-ROM tốc độ đơn hoạt động ở tốc độ 75 sector/s, tức là tốc độ dữ liệu sẽ bằng 153600 byte/s trong mô hình 1 và bằng 175200 byte/s trong mô hình 2. Các ổ đĩa tốc độ kép có tốc độ nhanh gấp đôi… Cứ như vậy, một ổ 40x có thể đạt được tốc độ 40 x 153600 byte/s (giả sử giao diện ổ, bus, và hệ điều hành có thể xử lý dữ liệu với tốc độ này). Một đĩa audio CD chuẩn có thể chứa 74 phút âm nhạc nếu sử dụng mô hình 1, với dung lượng bằng 681 984 000 bytes. Con số này thường được ghi là 650 MB vì 1 MB bằng 2^{20} byte (1 048 576 bytes), chứ không phải là 1 000 000 byte.

Chú ý rằng thậm chí ổ CD-ROM 32x (4 915 200 byte/s) cũng không thể nhanh bằng một ổ đĩa từ SCSI-2 có tốc độ 10 MB/sec, mặc dù nhiều ổ CD-ROM sử dụng giao diện SCSI (cũng có các ổ CD-ROM sử dụng IDE). Thời gian tìm kiếm dữ liệu trên CD-ROM thường mất tới vài trăm mili giây, rõ ràng ổ CD-ROM không thể có hiệu suất cao bằng ổ đĩa từ, dù nó có dung lượng lớn.

Năm 1986, hãng Philips cho ra đời cuốn **Sách Xanh (Green Book)**, đưa thêm các khả năng chèn âm thanh, phim ảnh, và dữ liệu vào cùng một sector, đó là các tính năng cần thiết cho một đĩa CD-ROM đa phương tiện.

Một vấn đề nan giải trên CD-ROM là hệ thống file. Để có thể sử dụng đĩa CD-ROM trên các máy tính khác nhau, cần phải có sự thống nhất về hệ thống file trên CD-ROM. Các đại diện của nhiều công ty máy tính đã họp tại Lake Tahoe, High Sierras, ở vùng giáp gianh giữa California và Nevada, họ đã đưa ra một hệ thống file gọi là **High Sierra**. Sau đó nó đã được nâng cấp thành một tiêu chuẩn quốc tế (IS 9660). Tiêu chuẩn này có ba cấp. Cấp 1 sử dụng tên file có tối đa 8 kí tự và phần mở rộng có tối đa 3 kí tự (giống như quy ước tên file của MS-DOS). Tên file chỉ có thể chứa các chữ cái in hoa, chữ số, và dấu gạch dưới. Các thư mục có thể lồng vào nhau tối đa tám tầng, tên thư mục không có phần mở rộng. Cấp 1 đòi hỏi tất cả các file phải nằm kề nhau, điều này không phải là vấn đề khó khăn đối với các thiết bị chỉ ghi một lần. Bất cứ đĩa CD-ROM nào đạt chuẩn IS 9660 Cấp 1 đều có thể đọc bằng MS-DOS, máy tính Apple, máy tính dùng UNIX, hoặc bất cứ máy tính nào khác. Các nhà sản xuất đĩa CD-ROM đánh giá tính năng này là một trong những thay đổi quan trọng.

IS 9660 Cấp 2 cho phép sử dụng tên dài 32 kí tự, và Cấp 3 cho phép các file không nhất thiết phải nằm kề nhau. Phần mở rộng Rock Ridge (cách gọi theo tên một thị trấn trong bộ phim của

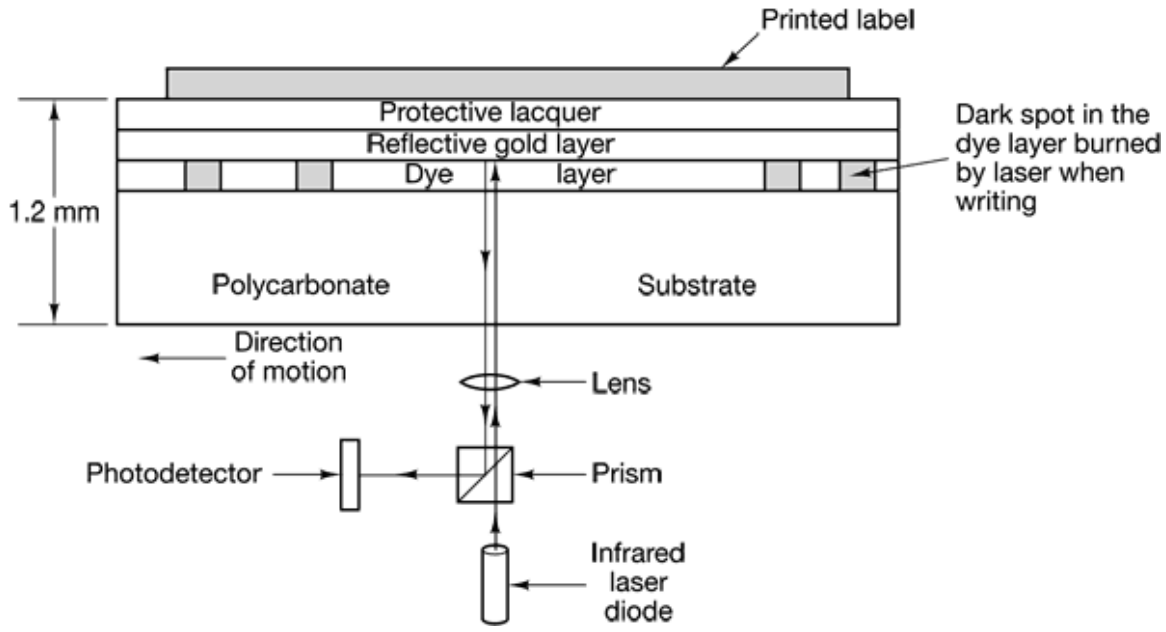
Gene Wilder - *Blazing Saddles*) cho phép sử dụng các tên rất dài (trên UNIX), UIDs, GIDs, và các liên kết biểu tượng, nhưng các CD-ROM không tuân theo chuẩn cấp 1 sẽ không thể đọc được trên tất cả các máy tính.

Các đĩa CD-ROM được sử dụng rất rộng rãi để chứa games, phim ảnh, sách giáo khoa, bản đồ, và các công việc liên quan khác ở tất cả các thể loại. Hầu hết các phần mềm thương mại được cung cấp bằng CD-ROM. Nhờ có dung lượng lớn và giá thành thấp, nó ngày càng trở nên phù hợp với rất nhiều ứng dụng.

CD-Recordable

Ban đầu, các thiết bị dùng để sản xuất ra đĩa CD-ROM chủ rất đắt. Nhưng trong ngành công nghiệp máy tính thì chẳng cái gì có thể giữ giá quá lâu. Vào giữa những năm 1990, các máy ghi CD với kích thước không lớn hơn các ổ đọc CD đã trở nên phổ biến trong hầu hết các cửa hàng máy tính. Thiết bị này vẫn khác xa so với các ổ đĩa từ vì chỉ có thể ghi một lần, CD-ROM không thể xoá được. Tuy nhiên, chúng cũng đã nhanh chóng trở thành một giải pháp lưu trữ cho các ổ đĩa cứng lớn, và cũng cho phép các cá nhân hay các công ty tự sản xuất các đĩa CD-ROM của riêng họ, hoặc tạo ra các đĩa chủ để sản xuất đĩa CD thương mại. Các ổ đĩa này được gọi là **CD-R** (**CD-Recordable**).

Về mặt vật lý, các đĩa CD-R cũng có đường kính 120 mm và bằng polycarbonate giống như CD-ROM, chỉ khác ở chỗ chúng có một đường rãnh rộng 0.6 mm để định hướng tia laser khi ghi. Rãnh này có một độ lệch hình sin khoảng 0.3 mm tại tần số 22.05 kHz để tạo ra một hồi tiếp liên tục, nhờ vậy có thể kiểm tra một cách chính xác tốc độ quay và điều chỉnh lại nếu cần. Các đĩa CD-R trông cũng giống như đĩa CD-ROM, chỉ khác là chúng có màu vàng chứ không phải màu bạc. Lớp phản xạ của chúng được làm bằng vàng thực sự, chứ không phải nhôm. Không giống như các đĩa CD bạc (có các hố lõm vật lý), sự khác biệt giữa pit và land trên các đĩa CD-R được làm giả bằng màu sắc. Có một lớp thuốc nhuộm nằm giữa lớp polycarbonate và lớp phản xạ bằng vàng, như trên hình 5-22. Có hai loại thuốc nhuộm: cyanine có màu xanh lá cây và pthalocyanine có màu vàng cam. Các nhà hoá học có thể sẽ tranh luận không ngừng về việc màu nào sẽ tốt hơn. Các thuốc nhuộm này cũng tương tự như loại được sử dụng trong nhiếp ảnh. Điều này giải thích tại sao Eastman Kodak và Fuji lại là những nhà sản xuất chủ yếu của đĩa CD-R trắng.



Hình 5-22 Mặt cắt ngang của đĩa CD-R và tia laser. Đĩa CD-ROM bạc cũng có cấu trúc tương tự, chỉ khác là không có lớp thuốc nhuộm và được phủ bằng nhôm thay vì bằng vàng.

Ở trạng thái ban đầu, lớp thuốc nhuộm là trong suốt và để cho tia laser đi qua tới lớp phản xạ bằng vàng. Khi ghi, tia laser được chuyển sang công suất cao (8 - 16 mW). Tia này đập vào một điểm trên lớp thuốc nhuộm, làm nóng nó, phá vỡ một liên kết hoá học. Điều này làm thay đổi cấu trúc phân tử và tạo ra một điểm tối. Khi đọc (với công suất 0.5 mW), bộ tách sóng quang sẽ phân biệt được sự khác nhau giữa điểm tối (nơi thuốc nhuộm bị biến đổi) và và các vùng trong suốt (không bị biến đổi). Sự khác biệt này được dùng để phân biệt giữa các pit và các land, ngay cả khi nó được đọc trên các ổ CD-ROM truyền thống, hoặc thậm chí trên các máy nghe audio CD.

Không có loại đĩa CD nào có thể tồn tại được nếu không tuân theo các tiêu chuẩn nhất định, do đó đĩa CD-R cũng có một cuốn sách, gọi là **Orange Book**, được xuất bản năm 1989. Cuốn sách này định nghĩa về CD-R, và cũng đưa ra một định dạng mới, **CD-ROM XA**, nó cho phép các đĩa CD-R có thể được ghi thêm từng phần, hôm nay ghi vài sector, ngày mai ghi vài sector, tháng sau vẫn có thể ghi tiếp. Một nhóm các sector liên tiếp nhau trong một lần ghi được gọi là một **CD-ROM track.**

Một trong những công dụng đầu tiên của CD-R là dùng cho máy PhotoCD của hãng Kodak. Trong hệ thống này, khách hàng có thể mang đến một cuộn phim và máy PhotoCD cũ của họ tới bộ phận xử lý ảnh, nhờ nạp thêm các ảnh mới vào máy PhotoCD cũ đó. Các ảnh mới được scan từ phim âm bản, rồi ghi vào một track độc lập trên máy PhotoCD. Việc ghi thêm này là cần thiết vì tại thời điểm đó, một đĩa CD-R trắng có giá rất đắt.

Tuy nhiên, việc ghi thêm này tạo ra một vấn đề mới. Trước khi có cuốn Orange Book, tất cả các đĩa CD-ROM đều có một bảng danh mục **VTOC (Volume Table of Contents)** ở đầu đĩa. Giải pháp này không áp dụng được khi ghi thêm. Cuốn Orange Book đưa ra giải pháp cung cấp cho mỗi track một bảng VTOC của riêng nó. Danh sách các file trong VTOC có thể chứa một vài hoặc tất cả các file nằm trong track trước đó. Khi đưa đĩa CD-R vào ổ, hệ điều hành sẽ tìm kiếm trên tất cả các track để xác định VTOC mới nhất, nó sẽ cung cấp trạng thái hiện hành của đĩa. Bằng cách chỉ lưu giữ thông tin về một số file (chứ không phải tất cả các file) của track trước đó trong VTOC hiện hành, giải pháp này đã tạo ra cảm giác rằng các file đó đã bị xoá. Các track cũng có thể được nhóm lại thành các **session**, tạo ra các đĩa CD-ROM **multisession.** Các máy nghe audio CD tiêu chuẩn không thể xử lý các đĩa CD multisession do chúng chỉ sử dụng một bảng VTOC ở đầu đĩa.

Mỗi track phải được ghi một cách liên tục, không được ngắt giữa chừng. Do đó, dữ liệu đến từ đĩa cứng phải có tốc độ đủ lớn để cung cấp cho hoạt động ghi. Nếu file dữ liệu cần ghi nằm rải rác trên đĩa cứng, thời gian tìm kiếm nó có thể làm cho dòng dữ liệu gửi đến CD-R bị chậm và gây ra lỗi (buffer underrun). Lỗi này giúp bạn có được một chiếc khay đẹp và sáng bóng (nhưng khá đắt) dùng để đựng đồ uống, hoặc một miếng đồ chơi màu vàng rộng 120 mm! Phần mềm CD-R thường đề nghị tập hợp các file dữ liệu đầu vào thành một image (bộ đệm) nằm liên tiếp nhau, có kích thước 650-MB. Nhưng quá trình này thường làm tăng gấp đôi thời gian ghi, đòi hỏi phải có 650 MB đĩa trống, và vẫn không bảo vệ được các ổ đĩa cứng khỏi sự quá tải, và phải hiệu chỉnh lại nhiệt độ khi chúng trở nên quá nóng.

Đĩa CD-R giúp các cá nhân và công ty có thể dễ dàng sao chép các đĩa CD-ROM (và audio CD), điều này dẫn tới sự vi phạm về bản quyền. Nhiều giải pháp được đưa ra để hạn chế điều đó và làm cho việc đọc các đĩa CD-ROM vi phạm trở nên khó khăn hơn so với các phần mềm có bản quyền. Một trong số các giải pháp là ghi tất cả độ dài của file trên CD-ROM bằng multigigabyte, điều này sẽ ngăn cản sự sao chép các file trên đĩa CD-ROM sang đĩa cứng bằng các phần mềm sao chép tiêu chuẩn. Độ dài thực của file được ghi vào phần mềm của nhà cung cấp hoặc được dấu đi (có thể sử dụng giải pháp mã hoá) trên các vùng không dùng đến của CD-ROM. Một giải pháp khác là cố ý sử dụng các ECC bị sai trên một số sector nhất định, để các phần mềm sao chép CD phải "sửa" các lỗi này. Phần mềm ứng dụng sẽ kiểm tra các ECC, và từ chối làm việc khi thấy chúng không bị sai nữa. Sử dụng các khoảng trống không hợp chuẩn giữa các track và các lỗi vật lý khác cũng là những giải pháp có thể áp dụng.

CD-Rewritable

Mặc dù người ta thường sử dụng các thiết bị ghi một lần như giấy và phim ảnh, vẫn có nhu cầu về đĩa CD có thể ghi lại nhiều lần. Có một công nghệ hiện nay gọi là **CD-RW (CD-Re Writable)**, nó sử dụng thiết bị có cùng kích thước với CD-R. Tuy nhiên, thay vì sử dụng các thuốc nhuộm như cyanine hay pthalocyanine, CD-RW sử dụng một hỗn hợp gồm bạc, indium, antimony, và tellurium để tạo thành lớp ghi dữ liệu. Hỗn hợp này có hai trạng thái: kết tinh và vô định hình, chúng có hệ số phản xạ khác nhau.

Các ổ CD-RW sử dụng tia laser với ba mức công suất khác nhau. Tại mức công suất cao nhất, tia laser sẽ làm chảy hỗn hợp, chuyển nó từ trạng thái kết tinh có hệ số phản xạ cao sang trạng thái vô định hình có hệ số phản xạ thấp tương ứng với một pit. Ở mức công suất trung, hỗn hợp chảy ra và trở về trạng thái kết tinh tự nhiên của nó, tương ứng với land. Mức công suất thấp dùng để đọc dữ liệu, không có sự chuyển đổi trạng thái.

Lý do CD-RW không thay thế được CD-R là vì đĩa CD-RW trắng đắt hơn nhiều so với đĩa CD-R trắng. Ngoài ra, đối với các ứng dụng sao lưu đĩa cứng, thì đĩa CD-R sẽ an toàn hơn do không thể xoá được.

DVD

Định dạng cơ bản của CD/CD-ROM có từ năm 1980. Từ đó tới nay công nghệ đã thay đổi rất nhiều, các đĩa quang có dung lượng cao hơn đã trở nên khả thi về mặt kinh tế, và nhu cầu về chúng cũng rất lớn. Hollywood mong muốn thay thế các băng video analog bằng các đĩa kỹ thuật số, do các đĩa có chất lượng cao hơn, rẻ hơn, bền hơn, thể tích nhỏ hơn nên tốn ít chỗ chứa, và không phải tua lại. Các công ty điện tử dân dụng cũng đang muốn tìm kiếm một sản phẩm mới đặc biệt, và nhiều công ty máy tính muốn tích hợp các tính năng đa phương tiện vào phần mềm của họ.

Sự gặp gỡ giữa công nghệ và nhu cầu của ba ngành công nghiệp vô cùng hùng mạnh và giàu có đã cho ra đời đĩa **DVD**, viết tắt của **Digital Video Disk** (Đĩa hình kỹ thuật số), nhưng ngày nay thường được hiểu là **Digital Versatile Disk** (Đĩa đa năng kỹ thuật số). Các đĩa DVD sử dụng cùng một thiết kế với đĩa CD, chúng có đường kính 120 mm và được đúc bằng polycarbonate. Các pit và land được chiếu bởi một đi ốt laser và tín hiệu được đọc bởi bộ tách sóng quang. Dưới đây là một số đặc điểm mới:

1. Các pit có kích thước nhỏ hơn (0.4 micro mét so với 0.8 micro mét của đĩa CD).
2. Đường xoắn ốc chặt hơn (khoảng cách giữa các track là 0.74 micro mét so với 1.6 micro mét của đĩa CD).
3. Sử dụng laser đỏ mảnh hơn (0.65 micro mét so với 0.78 micro mét của đĩa CD).

Đồng thời, các cải tiến này cũng làm tăng dung lượng đĩa lên gấp bảy lần, bằng 4.7 GB. Ổ DVD 1x có tốc độ là 1.4 MB/s (so với 150 KB/s của ổ CD). Không may, việc chuyển sang dùng laser đỏ cũng có nghĩa là các đầu DVD sẽ phải sử dụng một đầu đọc laser thứ hai hoặc một bộ chuyển đổi tín hiệu để đọc được các đĩa CD và CD-ROM hiện có. Ngoài ra, việc đọc các đĩa CD-R và CD-RW trên ổ DVD có thể không thực hiện được.

4.7 GB có đủ không? Có lẽ là đủ. Sử dụng chuẩn nén MPEG-2 (IS 13346), một đĩa DVD 4.7 GB có thể chứa được 133 phút phim với độ phân giải cao (720x480), các rãnh âm thanh có thể hỗ trợ tám ngôn ngữ, và có thể hỗ trợ tới 32 phụ đề. Khoảng 92% các bộ phim của Hollywood có độ dài dưới 133 phút. Tuy nhiên, một số ứng dụng như game đa phương tiện hay các tài liệu tra cứu cần tới dung lượng lớn hơn, và Hollywood cũng sẽ thích đặt nhiều phim trên cùng một đĩa, nên đã ra đời bốn định dạng sau:

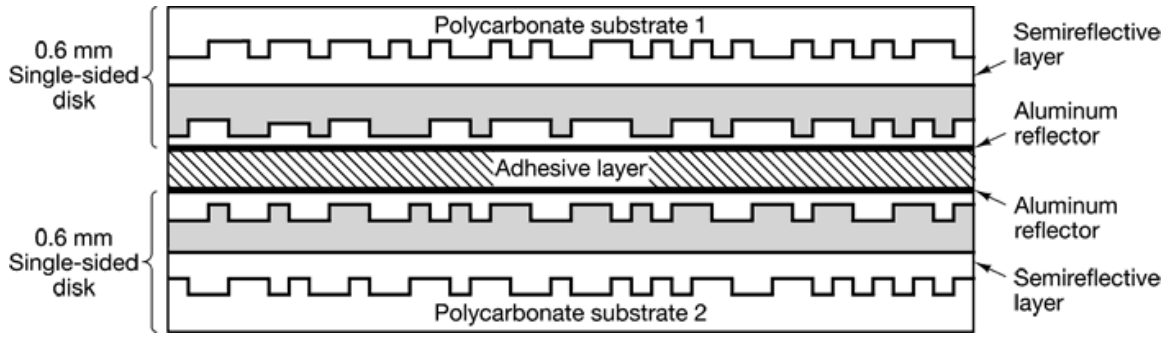
1. Đĩa một mặt - một lớp (4.7 GB).
2. Đĩa một mặt - hai lớp (8.5 GB).
3. Đĩa hai mặt - một lớp (9.4 GB).
4. Đĩa hai mặt - hai lớp (17 GB).

Tại sao lại có nhiều định dạng như vậy? Đó là một sự thận trọng. Các hãng Philips và Sony muốn dùng các đĩa một mặt - hai lớp cho các phiên bản dung lượng cao, nhưng Toshiba và Time Warner lại muốn dùng đĩa hai mặt - một lớp. Philips và Sony cho rằng người dùng không thích phải tự mình lật mặt đĩa, còn Time Warner không tin tưởng vào việc đặt hai lớp dữ liệu trên cùng

một mặt. Kết quả là sự thoả hiệp, sử dụng phối hợp tất cả các định dạng, và thị trường sẽ tự nó lựa chọn.

Công nghệ dual layering tạo ra hai lớp phản xạ, một lớp phản xạ hoàn toàn ở phía dưới và một lớp bán phản xạ ở bên trên. Tuỳ vào loại tia laser, nó sẽ phản xạ ở lớp này hay lớp khác. Lớp bên dưới sẽ cần có các pit và land lớn hơn một chút để có thể đọc được chính xác, nên dung lượng của lớp này nhỏ hơn lớp ở bên trên.

Các đĩa hai mặt được sản suất bằng cách lấy hai đĩa một mặt dày 0.6 mm và ghép phần lưng của chúng lại với nhau. Để bề dày của tất cả các đĩa giống nhau, mỗi đĩa một mặt sẽ bao gồm một đĩa dày 0.6 mm gắn trên một lớp nền. Cấu trúc của đĩa hai mặt - hai lớp được minh hoạ trên hình 5-23.



Hình 5-23 Đĩa DVD hai mặt - hai lớp.

Đĩa DVD được phát minh bởi một liên minh gồm 10 công ty điện tử dân dụng, trong đó có bảy công ty của Nhật Bản, cùng với sự hỗ trợ của các hãng phim lớn của Hollywood (trong số đó cũng có nhiều hãng thuộc sở hữu của các công ty điện tử Nhật bản nói trên). Các công ty máy tính và viễn thông không được mời đi picnic, và kết quả là các đĩa DVD được dùng để chứa phim cho thuê hoặc để bán. Các tính năng chuẩn của nó bao gồm khả năng nhảy qua các cảnh phim bẩn (cho phép các bậc phụ huynh tua qua để bảo vệ bọn trẻ), sáu kênh âm thanh, và hỗ trợ Pan-and-Scan. Tính năng sau cùng cho phép các đầu DVD có thể thay đổi lề phải và lề trái của hình ảnh phim (tỷ lệ thông thường của chiều rộng và chiều cao là 3:2), để có thể điều chỉnh cho phù hợp với tỷ lệ hiện hành trên truyền hình là 4:3.

Một vấn đề khác mà có lẽ ngành công nghiệp máy tính không thể nghĩ ra, đó là việc cố tình tạo ra sự không tương thích giữa các đĩa dùng ở Mỹ và các đĩa dùng ở Châu Âu, cũng như với các tiêu chuẩn dùng ở các lục địa khác. Hollywood đòi hỏi tính năng này vì những phim mới luôn được giới thiệu trước tiên ở Mỹ, rồi mới tới Châu Âu, nơi xuất khẩu các đĩa video vào Mỹ. Ý tưởng nói trên là để người Châu Âu không thể mua các phim của Mỹ quá sớm, nhờ vậy có thể giảm được sự kinh doanh phim ảnh của các rạp Châu Âu. Nếu Hollywood mà quản lý cả ngành công nghiệp máy tính nữa thì ta sẽ có các đĩa mềm 3.5 inch dùng cho nước Mỹ và đĩa mềm 9 cm dùng cho Châu Âu!

5.4.2 Định dạng đĩa

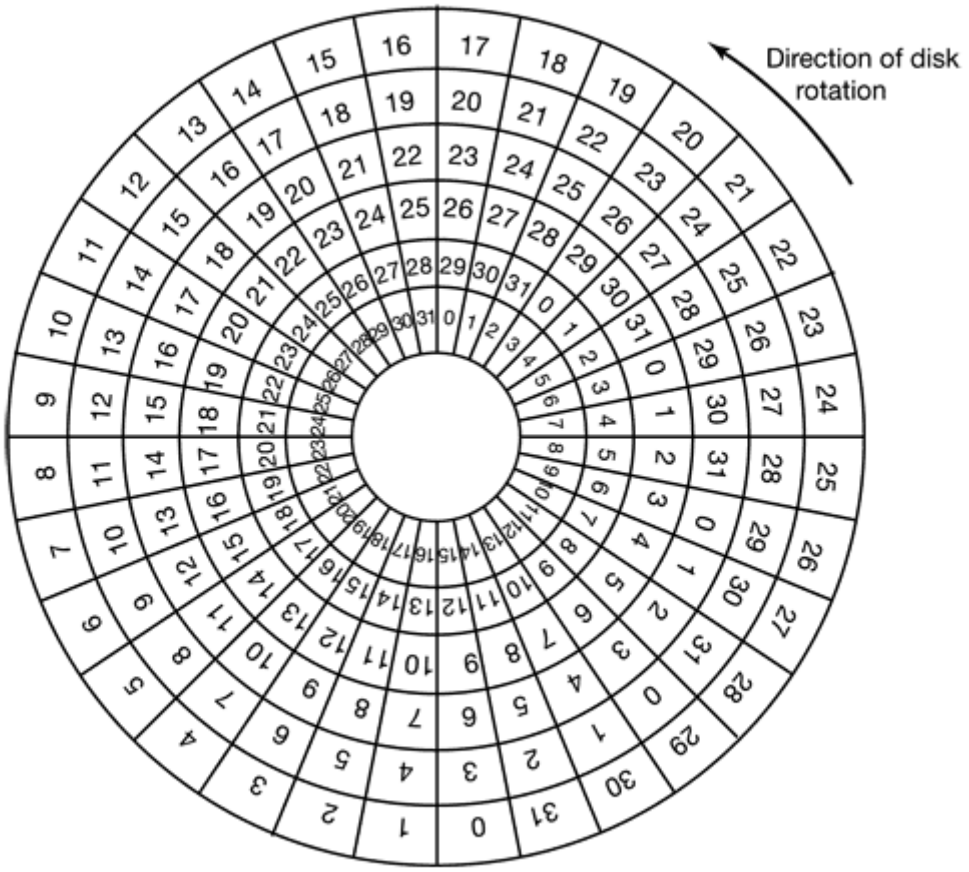
Đĩa cứng thường gồm một chồng đĩa được làm bằng hợp kim nhôm hoặc thuỷ tinh, có đường kính 5.25 inch hoặc 3.5 inch (hoặc thậm chí nhỏ hơn khi dùng cho máy tính xách tay). Bề mặt đĩa được phủ một lớp mỏng ôxít kim loại có từ tính. Khi mới được sản xuất xong thì trên đĩa chưa có bất cứ thông tin gì.

Trước khi có thể sử dụng đĩa, chúng phải được **định dạng cấp thấp (low-level format)**, điều này được thực hiện bằng phần mềm. Định dạng đĩa gồm nhiều rãnh tròn đồng tâm (track), mỗi rãnh chứa nhiều sector, giữa các sector có một khoảng trống nhỏ. Định dạng của một sector được trình bày ở hình 5-24.

Hình 5-24 Một sector đĩa.

Phần tiêu đề của sector (preamble) chứa các bít dùng để xác định vị trí bắt đầu của sector. Nó cũng chứa số hiệu sector, số hiệu cylinder, và một số thông tin khác. Kích thước vùng dữ liệu được xác định bởi chương trình định dạng cấp thấp. Hầu hết các đĩa sử dụng sector có vùng dữ liệu dài 512 byte. Trường ECC chứa thông tin dùng để sửa các lỗi gặp phải trong quá trình đọc dữ liệu. Kích thước và nội dung của trường này còn tuỳ thuộc vào từng nhà sản xuất, tuỳ thuộc vào không gian đĩa mà người thiết kế muốn sử dụng để nâng cao độ tin cậy, và độ phức tạp của mã ECC mà controller có thể xử lý. Trường ECC dài 16 byte cũng hay được sử dụng. Ngoài ra, tất cả các đĩa cứng đều có các sector dự phòng để thay thế cho các sector bị lỗi.

Vị trí của sector 0 trên hai track liền kề không thẳng hàng với nhau mà có một độ lệch. Độ lệch này được gọi là **cylinder skew (độ lệch cylinder)**, nó giúp cải thiện hiệu suất đĩa. Nhờ vậy có thể đọc nhiều track trong một lần đọc đĩa mà không bị mất dữ liệu. Để hiểu rõ điều này, ta sẽ xem xét vấn đề xảy ra trên cấu trúc đĩa ở hình 5-18(a). Giả sử có một yêu cầu đọc 18 sector trên track trong cùng của đĩa, bắt đầu từ sector 0. Việc đọc 16 sector đầu tiên sẽ cần một vòng quay của đĩa, và cần phải chuyển dịch ra bên ngoài để tìm kiếm sector thứ 17. Trong lúc đầu đọc dịch ra được một track thì sector 0 đã quay được một đoạn nên sẽ cần phải quay cả vòng đĩa để tìm lại sector này. Vấn đề này được giải quyết nhờ việc xếp các sector lệch nhau như trên hình 5-25.



Hình 5-25 Minh hoạ độ lệch cylinder.

Độ lệch cylinder còn phụ thuộc vào kiến trúc hình học của ổ đĩa. Ví dụ, ổ đĩa 10000 vòng/phút quay một vòng hết 6 ms. Nếu một track có 300 sector thì thời gian quay trên một sector là 20 μs. Nếu thời gian chuyển đổi giữa hai track là 800 μs thì trong thời gian đó đầu đọc có thể lướt qua được 40 sector, nên độ lệch cylinder bằng 40 sector, chứ phải phải chỉ là 3 sector như trên hình 5-25. Cần nói thêm rằng việc chuyển đổi giữa các đầu đọc cũng tốn một lượng thời gian nhất định, tức là có một **độ lệch đầu đọc (head skew)**, nhưng độ lệch này không lớn.

Sau khi định dạng cấp thấp, dung lượng đĩa sẽ giảm xuống tuỳ thuộc vào kích thước của phần tiêu đề sector, khoảng cách giữa các sector, và ECC, cũng như số lượng các sector dự phòng. Thường thì dung lượng đĩa sau khi định dạng sẽ giảm đi 20% so với trước khi định dạng. Các sector dự phòng không được tính vào dung lượng đĩa sau khi định dạng, nên tất cả các ổ đĩa cùng loại sẽ có dùng dung lượng khi xuất xưởng, không phụ thuộc vào số lượng sector bị lỗi (nếu số lượng sector bị lỗi lớn hơn số lượng sector dự phòng thì ổ đĩa sẽ bị loại bỏ).

Có một sự lộn xộn rất lớn về vấn đề dung lượng đĩa vì một số nhà sản xuất cố tình công bố dung lượng đĩa khi chưa định dạng, để nó trông có vẻ lớn hơn thực tế. Ví dụ , xét một đĩa chưa định dạng có dung lượng là 20×10^9 byte. Nó được bán với danh nghĩa là đĩa 20 GB. Tuy nhiên, sau khi định dạng, có lẽ dung lượng chứa dữ liệu của nó chỉ còn là $2^{34} = 17.2 \times 10^9$ byte. Và hệ điều hành sẽ chỉ công bỗ dung lượng đĩa là 16.0 GB chứ không phải là 17.2 GB vì phần mềm coi 1 GB bằng 2^{30} (1 073 741 824) byte, không phải là 10^9 (1 000 000 000) byte.

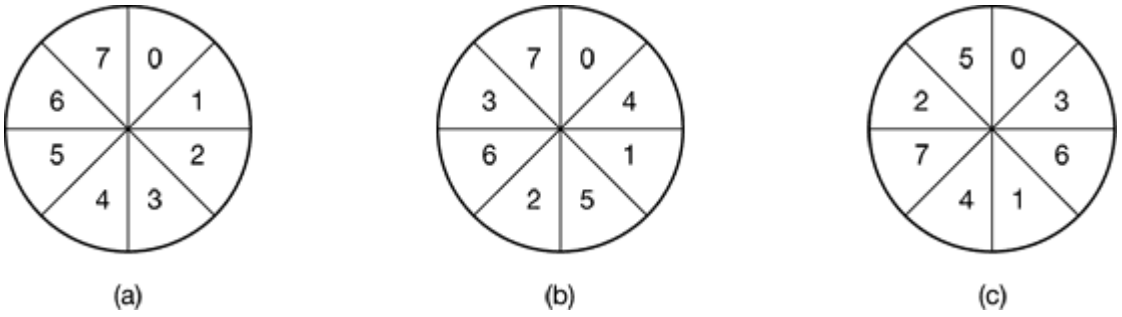
Tệ hơn nữa, trong lĩnh vực truyền số liệu, 1 Gbps bằng 1000 000 000 bit/s vì tiền tố *giga* có nghĩa là 10^9 (một kilomet bằng 1000 mét, chứ không phải là 1024 mét). Chỉ có trong kích thước của bộ nhớ và đĩa thì kilo, mega, giga, và tera mới tương ứng với 2^{10}, 2^{20}, 2^{30}, và 2^{40}.

Việc định dạng cũng ảnh hưởng tới hiệu suất đĩa. Nếu một đĩa 10000 vòng/phút có 300 sector trên một track, mỗi sector có 512 byte, thì sẽ mất 6 ms để đọc 153600 byte của một track với tốc độ dữ liệu là 25 600 000 byte/s hay 24.4 MB/s. Không thể đọc nhanh hơn nữa, dù sử dụng bất cứ giao diện đĩa nào, thậm chí cả khi sử dụng giao diện SCSI với tốc độ 80 MB/s hay 160 MB/s.

Thực ra việc đọc liên tục tại tốc độ này đòi hỏi phải có một bộ đệm lớn trong controller. Ví dụ, một controller có bộ đệm chỉ chứa được một sector nhưng lại nhận được lệnh phải đọc hai sector liên tiếp nhau. Sau khi đọc sector thứ nhất từ đĩa và thực hiện tính toán ECC, dữ liệu phải được chuyển vào bộ nhớ chính. Trong quá trình chuyển dữ liệu, sector tiếp theo đã quay qua đầu đọc. Khi quá trình sao chép dữ liệu vào bộ nhớ hoàn thành, controller sẽ phải chờ thêm khoảng gần một vòng quay của đĩa để đọc được sector thứ hai.

Vấn đề này có thể được giải quyết nhờ mô hình sắp xếp xen kẽ các sector khi định dạng đĩa. Trên hình 5-26(a) ta thấy cách thức sắp xếp thông thường (bỏ qua độ lệch cylinder). Trên hình 5-26(b) áp dụng mô hình **xen kẽ đơn (single interleaving)**, các sector có số hiệu liên tiếp sẽ không nằm cạnh nhau mà cách nhau một khoảng trống, để controller có thời gian sao chép dữ liệu vào bộ nhớ chính.

Nếu quá trình sao chép diễn ra rất chậm, có thể sẽ cần tới mô hình **xen kẽ kép (double interleaving)** như trên hình 5-26(c). Nếu controller có bộ đệm chỉ chứa được một sector, sẽ không có vấn đề gì xảy ra dù quá trình sao chép dữ liệu từ bộ đệm vào bộ nhớ được thực hiện bởi controller, CPU chính, hay chip DMA; nó vẫn cần tốn thời gian. Để tránh phải sử dụng mô hình xen kẽ, controller nên có bộ đệm chứa được cả một track. Nhiều đĩa cứng hiện đại được thiết kế như vậy.



Hình 5-26 (a) Không xen kẽ. (b) Xen kẽ đơn. (c) Xen kẽ kép.

Sau khi định dạng cấp thấp xong, cần phải tiến hành phân khu đĩa. Về mặt logic, mỗi phân khu giống như một đĩa riêng rẽ. Trên máy Pentium và phần lớn các máy tính khác, sector 0 chứa **master boot record (bản ghi khởi động chủ)**, nó chứa các mã lệnh dùng trong quá trình khởi động và bảng phân khu đĩa (partition table). Bảng phân khu chứa thông tin về sector bắt đầu và kích thước của từng phân khu. Trên máy Pentium, bảng phân khu có thể chứa thông tin về bốn phân khu. Nếu tất cả các phân khu đều sử dụng hệ điều hành Windows, chúng sẽ có tên là C:, D:, E:, và F:, và được đối xử như các ổ đĩa riêng rẽ. Ổ đĩa CD-ROM thứ nhất sẽ được gọi là F:. Để có thể khởi động từ đĩa cứng thì phải có một phân khu được đánh dấu là phân khu chủ động (active), thông tin này được lưu trong bảng phân khu.

Bước cuối cùng trước khi sử dụng đĩa là phải **định dạng cấp cao (high-level format)** cho từng phân khu. Quá trình này sẽ tạo ra một khối thông tin khởi động, thông tin để quản lý vùng đĩa trống (dưới dạng danh sách hoặc mảng), thư mục gốc, và một hệ thống file trống. Nó cũng đặt một thông tin vào bảng phân khu để thông báo về loại hệ thống file mà phân khu này sử dụng, vì nhiều hệ điều hành có thể hỗ trợ các hệ thống file không tương thích khác nhau (do lịch sử để lại). Lúc này hệ thống đã có thể khởi động được.

Khi bật nguồn điện, trước tiên BIOS sẽ được chạy và đọc bản ghi khởi động chủ vào bộ nhớ, rồi nhảy tới đoạn mã của bản ghi này. Đoạn mã khởi động này sẽ kiểm tra xem phân khu nào là chủ động, rồi đọc sector khởi động của phân khu đó (boot sector) vào bộ nhớ. Đoạn mã trên sector khởi động được thi hành để tìm kiếm thư mục gốc cho một chương trình nào đó (thường là hệ điều hành hoặc một chương trình mồi). Chương trình đó sẽ được nạp vào bộ nhớ và thi hành.

5.4.3 Các giải thuật điều khiển cánh tay đĩa

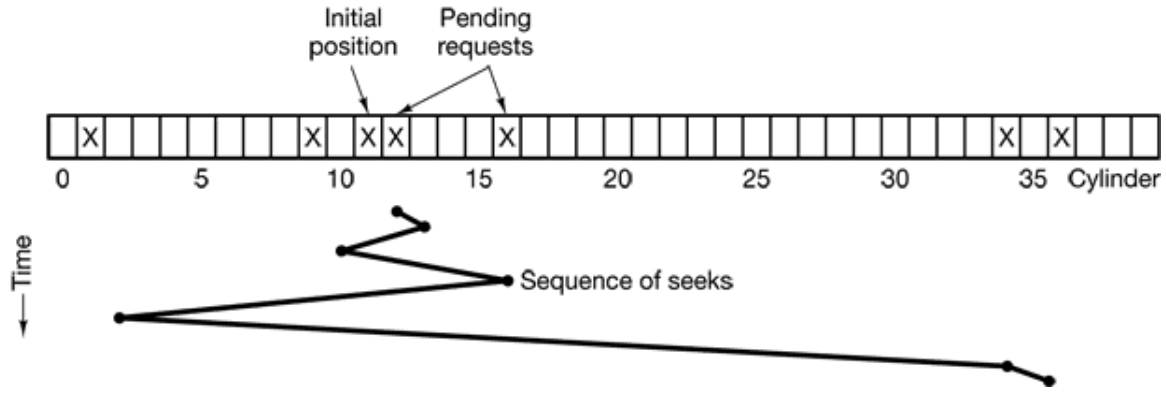
Trong mục này ta sẽ tìm hiểu một số giải pháp liên quan tới chương trình điều khiển đĩa nói chung. Trước hết, ta sẽ xem xét thời gian cần thiết để đọc hay ghi một khối dữ liệu của đĩa. Thời gian đó được xác định bởi ba yếu tố:

1. Thời gian tìm kiếm (thời gian dịch chuyển cánh tay đĩa tới cylinder phù hợp).
2. Thời gian chờ quay (thời gian chờ sector phù hợp quay tới dưới đầu đọc).
3. Thời gian truyền dữ liệu thực sự.

Với hầu hết các đĩa, thời gian tìm kiếm lớn hơn hẳn hai yếu tố còn lại, nên việc giảm thời gian tìm kiếm có thể cải thiện cơ bản hiệu suất của hệ thống.

Nếu chương trình điều khiển đĩa (driver) chấp nhận các yêu cầu gửi đến nó, và lần lượt thực hiện chúng theo trật tự xuất hiện, đó chính là giải thuật First-Come, First-Served (FCFS), đây cũng là một giải pháp. Tuy nhiên, có thể áp dụng một giải pháp khác khi đĩa bị quá tải. Trong khi cánh tay đĩa đang tìm kiếm vị trí thích hợp để đáp ứng một yêu cầu nào đó, những yêu cầu khác có thể xuất hiện từ các tiến trình khác. Nhiều chương trình điều khiển đĩa duy trì một bảng thông tin, sắp xếp theo số hiệu cylinder, nó chứa tất cả các yêu cầu chưa được giải quyết theo một danh sách liên kết bởi các entry trong bảng.

Với cấu trúc dữ liệu này, ta có thể cải tiến giải thuật first-come, first-served. Xét một đĩa có 40 cylinder. Giả sử xuất hiện yêu cầu muốn đọc một khối dữ liệu trên cylinder 11. Trong lúc tìm tới cylinder thứ 11, các yêu cầu mới xuất hiện và muốn đọc các cylinder lần lượt là 1, 36, 16, 34, 9, và 12. Chúng được đưa vào bảng quản lý các yêu cầu, theo một danh sách liên kết như minh hoạ trên hình 5-27.



Hình 5-27 Giải thuật Shortest Seek First (SSF).

Khi yêu cầu hiện hành (tìm cylinder 11) kết thúc, chương trình điều khiển đĩa sẽ lựa chọn một trong số các yêu cầu còn lại để thực hiện. Nếu sử dụng FCFS thì tiếp theo sẽ là cylinder 1, rồi tới 36… Giải thuật này sẽ khiến cánh tay đĩa phải chuyển động rất nhiều, lần lượt qua 10, 35, 20, 18, 25, và 3 cylinder, tổng cộng là chuyển động qua 111 cylinder.

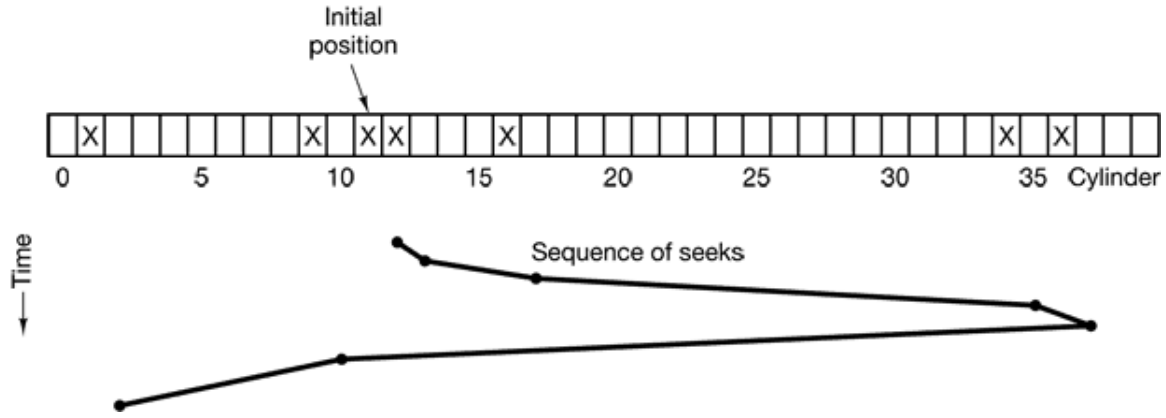
Thay vì như vậy, có thể lựa chọn các yêu cầu ở vị trí gần hơn để thực hiện tiếp theo, nhằm giảm thiểu thời gian tìm kiếm. Với các yêu cầu như trên hình 5-27, thứ tự thực hiện có thể lần lượt là 12, 9, 16, 1, 34, và 36 (xem hình vẽ). Như vậy, cánh tay đĩa sẽ chuyển động qua 1, 3, 7, 15, 33, và 2 cylinder, tổng cộng là 61 cylinder. Giải thuật này được gọi là **Shortest Seek First** (SSF), nó giảm được chuyển động của cánh tay đĩa xuống còn một nửa so với FCFS.

Không may, giải thuật SSF vẫn có một vấn đề. Giả sử tiếp tục có yêu cầu mới được giửi đến trong lúc các yêu cầu trong bảng 5-27 đang được xử lý. Ví dụ, sau khi dịch chuyển tới cylinder 16, một yêu cầu mới về cylinder 8 xuất hiện, nó sẽ được ưu tiên trước cylinder 1. Nếu sau đó tiếp tục xuất hiện yêu cầu đối với cylinder 13, cánh tay đĩa sẽ chuyển tới cylinder 13 chứ không phải cylinder 1. Với một đĩa bị quá tải, cánh tay đĩa có khuynh hướng nằm ở vùng trung tâm của đĩa trong hầu hết thời gian, nên các yêu cầu truy nhập cylinder ở xa sẽ phải chờ rất lâu cho tới khi không còn các yêu cầu ở vùng trung tâm nữa. Các yêu cầu cách xa vùng trung tâm sẽ không được phục vụ tốt. Ở đây có sự mâu thuẫn giữa mục tiêu giảm thiểu thời gian đáp ứng và sự công bằng trong phục vụ.

Các toà nhà cao tầng cũng gặp phải vấn đề này. Việc điều khiển cầu thang máy lên xuống cũng tương tự như việc điều khiển cánh tay của ổ đĩa. Các yêu cầu sử dụng thang máy (và yêu cầu truy nhập cylinder) xuất hiện một cách ngẫu nhiên. Máy tính điều khiển thang máy có thể dễ dàng quản lý chuỗi các yêu cầu của khách hàng, và phục vụ họ theo giải thuật FCFS. Nó cũng có thể sử dụng SSF.

Tuy nhiên, hầu hết các thang máy sử dụng một giải thuật khác để giải quyết mâu thuẫn giữa hiệu quả và sự công bằng. Chúng luôn chuyển động theo cùng một hướng cho tới khi không còn yêu cầu nào theo hướng đó nữa, rồi mới đổi hướng. Giải thuật này được gọi là **elevator algorithm (giải thuật thang máy)**, nó đòi hỏi phần mềm phải duy trì một bít để xác định hướng (*UP* hoặc *DOWN*). Khi một yêu cầu được hoàn thành, bộ điều khiển đĩa (hay thang máy) sẽ kiểm tra bít này. Nếu nó là *UP* thì cánh tay đĩa (hay thang máy) sẽ chuyển động tới yêu cầu tiếp theo có vị trí cao hơn. Nếu không còn yêu cầu nào ở vị trí cao hơn, bít này sẽ được đảo lại. Khi bít có giá trị *DOWN*, chiều dịch chuyển sẽ hướng về phía yêu cầu ở vị trí thấp hơn (nếu có).

Hình 5-28 minh hoạ giải thuật thang máy với bảy yêu cầu giống như ở hình 5-27, giả sử bít định hướng ban đầu có giá trị *UP*. Trật tự truy nhập các cylinder là 12, 16, 34, 36, 9, và 1, Cánh tay đĩa phải dịch chuyển qua 1, 4, 18, 2, 27, và 8 cylinder, tổng cộng là 60 cylinder. Trong trường hợp này giải thuật thang máy có hiệu quả tốt hơn giải thuật SSF một chút, mặc dù nó thường kém hơn. Một đặc điểm thú vị của giải thuật thang máy là dù tập hợp các yêu cầu có thay đổi thế nào, thì giới hạn trên của số lần dịch chuyển vẫn không thay đổi: nó bằng hai lần số lượng cylinder.



Hình 5-28 Giải thuật thang máy dùng để điều khiển đĩa.

Có một thay đổi nhỏ trong giải thuật này đã tạo ra một chút khác biệt về thời gian đáp ứng (Teory, 1972), đó là luôn quét theo cùng một hướng. Khi cylinder có số hiệu cao nhất (trong danh sách yêu cầu) đã được phục vụ, cánh tay đĩa sẽ chuyển xuống cylinder có số hiệu thấp nhất trong danh sách, rồi tiếp tục chuyển động lên trên. Về mặt hiệu quả, nó đã đề cao các cylinder có số hiệu thấp hơn các cylinder số hiệu cao.

Một số bộ điều khiển đĩa cung cấp giải pháp cho phần mềm để kiểm tra số hiệu sector hiện hành (sector đang nằm dưới đầu đọc). Với bộ điều khiển như vậy, có thể áp dụng một giải pháp tối ưu khác. Nếu có hai hoặc nhiều yêu cầu đối với cùng một cylinder, thì chương trình điều khiển có thể phát lệnh đọc sector tiếp theo sẽ quay tới vị trí đầu đọc. Chú ý là khi có nhiều track

trên một cylinder thì các yêu cầu liên tiếp nhau có thể sẽ đọc các track khác nhau trên cylinder đó mà không gặp phải điều gì khó khăn. Controller có thể lựa chọn bất cứ đầu đọc nào ngay lập tức, vì sự lựa chọn đầu đọc không liên quan tới việc dịch chuyển cánh tay đĩa hay thời gian chờ quay.

Nếu đĩa có đặc tính là thời gian tìm kiếm nhanh hơn nhiều so với thời gian chờ quay, thì có thể áp dụng một giải pháp khác. Các yêu cầu chưa được xử lý sẽ được sắp xếp theo số hiệu sector, và ngay khi sector tiếp theo quay tới dưới đầu đọc, cánh tay đĩa sẽ được chuyển tới track phù hợp để đọc hoặc ghi.

Với các đĩa cứng hiện đại, thời gian tìm kiếm và thời gian chờ quay ảnh hưởng rất lớn tới hiệu suất, vì thế việc chỉ đọc một hoặc hai sector mỗi lần sẽ làm giảm hiệu quả rất nhiều. Do đó, một số bộ điều khiển đĩa luôn đọc và cất vào bộ đệm nhiều sector một lúc, thậm chí cả khi chỉ có yêu cầu đọc một sector. Thường thì bất cứ yêu cầu nào chỉ đọc một sector cũng sẽ khiến sector đó và cả các sector tiếp theo, thậm chí toàn bộ track hiện hành được đọc, tuỳ thuộc vào khả năng chứa của bộ đệm (cache) trong controller. Ví dụ, đĩa được mô tả trên hình 5-17 có 2 MB hoặc 4 MB cache. Việc sử dụng cache như thế nào là do controller quyết định. Theo cách đơn giản nhất, cache được chia làm hai phần, một phần dùng để đọc và một phần dùng để ghi. Nếu sector cần đọc tiếp theo đã có sẵn trong cache, yêu cầu đọc đó sẽ được đáp ứng ngay lập tức.

Cần chú ý là vùng nhớ cache trên bộ điều khiển đĩa hoàn toàn độc lập với vùng nhớ cache của hệ điều hành. Cache của controller thường chứa các khối dữ liệu không thực sự được yêu cầu, nhưng lại được đọc vì chúng nằm ngay bên dưới đầu đọc trong quá trình đọc các dữ liệu cần thiết. Ngược lại, các cache do hệ điều hành quản lý sẽ chứa các khối dữ liệu hay được sử dụng, và hệ điều hành nghĩ rằng chúng còn có thể sẽ được sử dụng tiếp trong tương lai gần (ví dụ như khối dữ liệu chứa một thư mục của đĩa).

Khi nhiều ổ đĩa cùng dùng chung một controller, nó sẽ phải duy trì một bảng yêu cầu riêng cho mỗi ổ đĩa. Bất cứ khi nào ổ đĩa rảnh rỗi, quá trình tìm kiếm sẽ được thực hiện để di chuyển cánh tay đĩa tới cylinder mà nó sẽ cần tới tiếp theo (giả sử controller cho phép tìm kiếm xếp chồng). Khi quá trình truyền dữ liệu hiện hành kết thúc, có thể tiến hành kiểm tra để xem ổ đĩa nào có đầu đọc đang ở vị trí của cylinder cần thiết. Nếu có một hoặc nhiều ổ đĩa như vậy, quá trình truyền dữ liệu tiếp theo sẽ được bắt đầu trên một trong các ổ đó. Còn nếu không thì chương trình điều khiển sẽ phát động việc tìm kiếm trên chính ổ đĩa vừa hoàn thành quá trình truyền, và chờ cho tới khi xuất hiện ngắt thông báo về ổ đĩa đầu tiên có đầu đọc nằm đúng vị trí.

Cần chú ý rằng tất cả các giải thuật điều khiển đĩa nói trên đều ngầm giả thiết kiến trúc hình học thực của đĩa giống với kiến trúc hình học ảo. Nếu không thì mọi cái sẽ là vô nghĩa vì hệ điều hành không thể nói được là cylinder 40 hay cylinder 200 nằm gần cylinder 39 hơn. Mặt khác, nếu bộ điều khiển đĩa có thể chấp nhận đa yêu cầu, thì nó cũng có thể sử dụng các giải thuật này ngay bên trong nó. Trong trường hợp đó, các giải thuật vẫn hợp lệ, nhưng hoạt động ở cấp thấp hơn, bên trong controller.

5.4.4 Xử lý lỗi

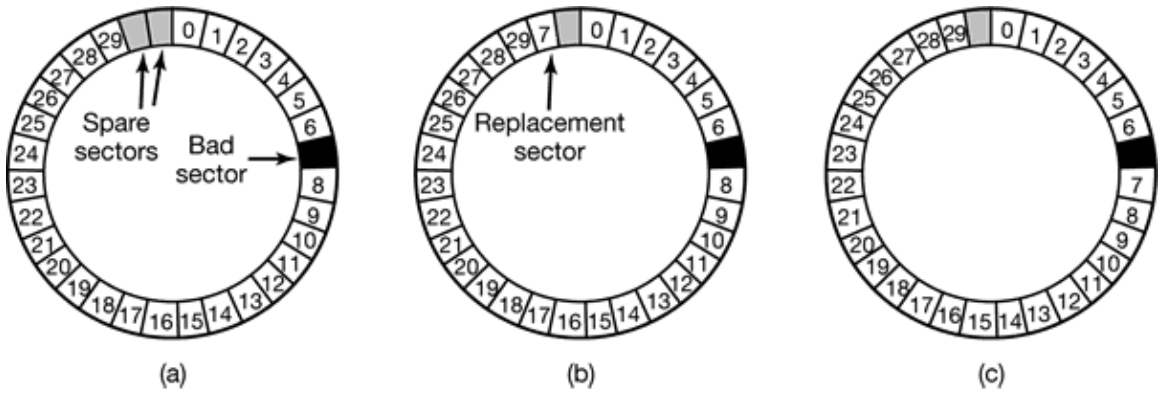
Các nhà sản xuất đĩa luôn mở rộng giới hạn công nghệ bằng cách tăng mật độ bít tuyến tính. Một track ở giữa đĩa 5.25 inch có chu vi bằng 300 mm. Nếu track này có 300 sector 512 byte thì mật độ ghi tuyến tính là khoảng 5000 bit/mm (vì còn phải dành chỗ cho phần tiêu đề, ECC, và khoảng trống giữa các sector). Việc ghi 5000 bit/mm đòi hỏi chất liệu nền phải thật đồng nhất và lớp ôxít bao phủ phải thật tốt. Không may, không thể sản xuất được một chiếc đĩa như vậy mà không có khuyết điểm. Ngay khi công nghệ sản xuất được cải tiến để đạt được mật độ như vậy, các nhà thiết kế đĩa sẽ lại muốn có mật độ cao hơn để tăng dung lượng. Điều đó có thể sẽ tiếp tục tạo ra các khiếm khuyết.

Quá trình sản xuất thường tạo ra các lỗi sector, nghĩa là sẽ không thể đọc chính xác dữ liệu đã ghi vào sector đó. Nếu lỗi này là nhỏ (chỉ ảnh hưởng tới một vài bít) thì vẫn có thể sử dụng sector này kết hợp với mã ECC để sửa lỗi. Nhưng nếu lỗi nghiêm trọng hơn, sẽ không thể giấu giếm được.

Có hai giải pháp đối với các khối đĩa bị lỗi: xử lý chúng ở cấp controller hoặc xử lý chúng ở cấp hệ điều hành. Đối với giải pháp thứ nhất, trước khi đĩa được xuất xưởng, chúng phải được

kiểm tra, và danh sách các sector hỏng được ghi lại trên đĩa. Mỗi sector hỏng sẽ được thay thế bằng một sector dự phòng.

Có hai cách để thực hiện việc thay thế sector. Trên hình 5-29(a) ta thấy một track có 30 sector dữ liệu và hai sector dự phòng. Sector 7 bị lỗi. Controller sẽ ánh xạ một sector dự phòng vào sector 7 như trên hình 5-29(b). Một cách khác là dịch tất cả các sector đi một vị trí, như trên hình 5-29(c). Trong cả hai trường hợp, controller phải nắm rõ thông tin về từng sector. Nó có thể quản lý các thông tin này trong một bảng nội tại (mỗi bảng ứng với một track), hoặc ghi lại phần tiêu đề để thay đổi số hiệu các sector. Nếu ghi lại phần tiêu đề thì giải pháp trên hình 5-29(c) sẽ tốn nhiều công sức hơn (vì phải ghi lại cả 23 tiêu đề), nhưng lại cho hiệu quả cao hơn do có thể đọc toàn bộ track trong một vòng quay.



Hình 5-29 (a) Track có một sector hỏng. (b) Thay thế sector hỏng bằng một sector dự phòng. (c) Dịch chuyển tất cả các sector đi một vị trí để bỏ qua sector hỏng.

Các lỗi cũng có thể phát sinh trong quá trình hoạt động, sau khi đĩa đã được cài đặt. Biện pháp đầu tiên khi phát hiện ra lỗi (và không xử lý được bằng ECC) là thực hiện đọc lại sector đó. Một số lỗi đọc chỉ là nhất thời, do chúng phát sinh bởi bụi bám trên đầu đọc, và sẽ biến mất trong lần đọc thứ hai. Nếu controller nhận thấy các lỗi xuất hiện lặp đi lặp lại trên một sector nào đó, nó có thể chuyển qua sector dự phòng trước khi sector đó bị hỏng hoàn toàn. Theo cách này, dữ liệu sẽ không bị mất, và thậm chí hệ điều hành lẫn người dùng đều không phải bận tâm về sự cố đó. Thường thì giải pháp trên hình 5-29(b) sẽ được sử dụng do các sector khác có thể đang chứa dữ liệu. Sử dụng giải pháp trên hình 5-29(c) sẽ không chỉ đòi hỏi phải ghi lại toàn bộ các tiêu đề, mà còn phải ghi lại toàn bộ dữ liệu nữa.

Ở phần trước ta đã nói tới hai giải pháp để xử lý lỗi: xử lý ở cấp controller hoặc xử lý ở cấp hệ điều hành. Nếu controller không có khả năng ánh xạ lại các sector như ta vừa thảo luận, thì hệ điều hành phải làm điều đó bằng phần mềm. Nghĩa là trước hết hệ điều hành phải có được danh sách các sector hỏng, bằng cách đọc chúng từ đĩa, hoặc đơn giản là kiểm tra toàn bộ đĩa. Một khi biết được các sector hỏng, nó có thể tạo ra bảng ánh xạ. Nếu hệ điều hành muốn sử dụng giải pháp như trên hình 5-29(c), nó sẽ phải dịch toàn bộ dữ liệu trong các sector (từ sector 7 tới sector 29) tiến lên một vị trí.

Nếu hệ điều hành thực hiện việc ánh xạ lại, nó phải đảm bảo rằng các sector hỏng không nằm trong bất cứ file dữ liệu nào, cũng như không nằm trong danh sách các sector trống. Một cách thức để thực hiện điều đó là tạo ra một file đặc biệt để chưa tất cả các sector hỏng. Nếu file này không có mặt trong hệ thống file, thì người dùng sẽ không đọc phải nó.

Tuy nhiên, vẫn còn một vấn đề tồn tại: đó là sự sao lưu. Nếu ổ đĩa được sao lưu bằng các file, điều quan trọng là chương trình sao lưu không được phép sao chép file chứa các khối đĩa lỗi. Để ngăn chặn điều này, hệ điều hành phải giấu file này đi, để chương trình sao lưu không thể tìm được nó. Nếu đĩa được sao lưu theo đơn vị sector chứ không phải bằng file, mọi cái sẽ khó khăn hơn (chứ không phải không làm được) khi phải ngăn chặn việc đọc các sector hỏng. Chỉ hy vọng

là chương trình sao lưu có đủ thông minh để bỏ cuộc sau 10 lần đọc sector thất bại, và chuyển sang làm việc với sector tiếp theo.

Các sector hỏng không phải là nguyên nhân duy nhất gây lỗi đĩa. Lỗi tìm kiếm có thể là do bộ phận cơ khí của cánh tay đĩa có vấn đề. Controller quản lý vị trí của cánh tay đĩa. Khi thực hiện tìm kiếm, nó phát ra một chuỗi xung tới motor điều khiển cánh tay (mỗi xung ứng với một cylinder), để dịch chuyển cánh tay tới cylinder mới. Khi cánh tay tới vị trí mới, controller sẽ đọc số hiệu cylinder từ phần tiêu đề của sector tiếp theo. Nếu số hiệu này không khớp với yêu cầu, một lỗi tìm kiếm đã xảy ra.

Hầu hết các bộ điều khiển đĩa cứng sẽ tự động sửa các lỗi tìm kiếm này, nhưng các bộ điều khiển đĩa mềm (kể cả trên máy Pentium) lại chỉ thiết lập một bít lỗi và chuyển phần còn lại cho chương trình điều khiển đĩa (driver). Driver sẽ xử lý lỗi bằng cách phát ra một lệnh `recalibrate` (lệnh hiệu chỉnh lại), để di chuyển cánh tay đĩa ra khỏi vị trí đó càng xa càng tốt, và reset lại giá trị cylinder hiện hành trong controller về 0. Thường thì điều này có thể giúp giải quyết vấn đề. Nếu vẫn chưa thành công thì phải sửa lại ổ đĩa.

Như ta đã thấy, controller thực sự là một máy tính chuyên dụng nhỏ, hoàn chỉnh, với phần mềm, các biến, bộ đệm, và đôi khi cũng có cả lỗi nữa. Thỉnh thoảng cũng có một chuỗi các biến cố hiếm như ngắt trên ổ đĩa xảy ra một cách đồng thời với lệnh recalibrate trên một ổ đĩa khác, gây ra một lỗi, và làm cho controller rơi vào một vòng lặp hoặc bị mất track dữ liệu mà nó đang làm việc. Các nhà thiết kế controller thường phải đề phòng những tình huống xấu nhất và cung cấp một chân tín hiệu dự phòng ngay trên chip. Khi xảy ra biến cố, chân tín hiệu này sẽ giúp controller quên đi mọi thứ mà nó đang thực hiện và reset lại. Nếu xảy ra lỗi nghiêm trọng, driver đĩa có thể thiết lập một bít để phát ra tín hiệu reset lại controller. Nếu điều này vẫn không giúp được gì thì driver chỉ còn cách là in ra thông báo lỗi rồi bỏ cuộc.

Việc hiệu chỉnh lại một đĩa có thể tạo ra những tiếng động vui tai, nhưng thường thì nó không gây ra tiếng động. Tuy nhiên, có một tình huống mà việc hiệu chỉnh sẽ gây ra một vấn đề nghiêm trọng: đó là trong các hệ thống thời gian thực. Khi đang xem một bộ phim từ đĩa cứng, hoặc đang ghi các file từ đĩa cứng sang CD-ROM, về cơ bản thì dòng bít dữ liệu đọc từ đĩa cứng phải có tốc độ đồng nhất. Trong các tình huống như vậy, việc hiệu chỉnh sẽ tạo ra các khoảng trống trong dòng bít và do đó không thể chấp nhận được. Các thiết bị đặc biệt - gọi là các đĩa **AV (Audio Visual disks)** - không bao giờ thực hiện hiệu chỉnh lại.

5.6 CÁC THIẾT BỊ CUỐI HƯỚNG KÍ TỰ

Các máy tính mục đích chung đều có ít nhất một bàn phím và một màn hình. Mặc dù bàn phím và màn hình trên các máy tính cá nhân là các thiết bị có công nghệ khác nhau, nhưng chúng có thể làm việc cùng nhau. Trên các máy tính lớn (mainframe), có rất nhiều người dùng kết nối từ xa, mỗi người lại có bàn phím và màn hình riêng. Các thiết bị này thường được gọi là **thiết bị cuối**. Ta cũng sẽ sử dụng thuật ngữ này, cả khi thảo luận về máy tính cá nhân (vì chưa tìm được thuật ngữ nào hay hơn).

Có rất nhiều loại thiết bị cuối. Dưới đây là ba loại phổ biến nhất:

1. Các thiết bị cuối độc lập, với giao diện nối tiếp RS-232, dùng cho máy tính lớn.
2. Các màn hình máy tính cá nhân, với giao diện đồ hoạ người dùng.
3. Các thiết bị cuối mạng.

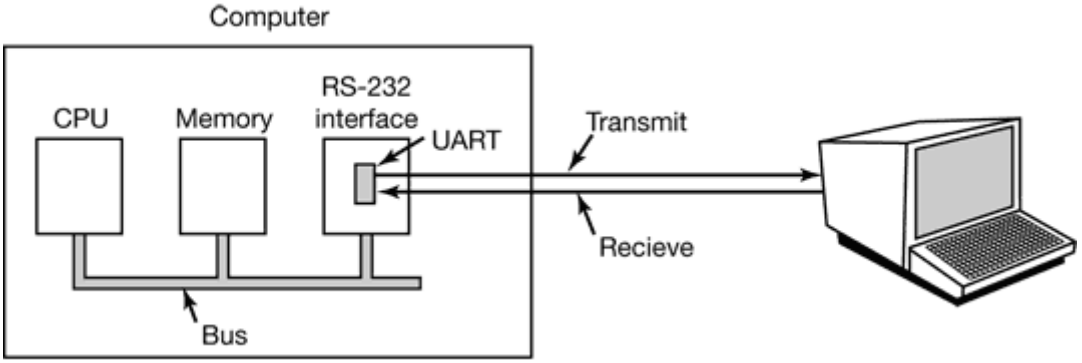
Mỗi loại thiết bị cuối lại có một vai trò riêng. Trong các mục tiếp theo ta sẽ lần lượt mô tả chúng.

5.6.1 Phần cứng thiết bị cuối RS-232

Thiết bị cuối RS-232 là những phần cứng gồm cả bàn phím và màn hình, và chúng sử dụng giao diện kết nối nối tiếp, truyền lần lượt từng bít (xem hình 5-34). Các thiết bị này sử dụng đầu nối 9 chân hoặc 25 chân, trong đó một chân dùng để truyền dữ liệu, một chân dùng để nhận dữ liệu, và một chân nối đất. Các chân khác dùng cho các chức năng điều khiển khác nhau, và phần lớn không được sử dụng. Các dây nối để truyền dữ liệu lần lượt từng bít (chứ không phải truyền

đồng thời 8 bít như trong giao diện song song của máy in) được gọi là các **đường truyền nối tiếp (serial line)**. Các modem cũng sử dụng giao diện này. Trên UNIX, các đường truyền nối tiếp có tên là */dev/tty1* và */dev/tty2*. Trên Windows chúng được gọi là *COM1* và *COM2*.

Để gửi một ký tự qua đường truyền nối tiếp tới thiết bị cuối RS-232 hoặc modem, máy tính phải truyền từng bít một, bắt đầu bằng bít tiêu đề (start bit), và kết thúc bằng 1 hoặc 2 bít dừng (stop bit) để báo hiệu kết thúc ký tự. Có thể chèn thêm một bít chẵn lẻ vào phía trước các bít dừng để cung cấp khả năng phát hiện lỗi, dù điều này thường chỉ áp dụng trong liên lạc với các hệ thống máy tính lớn.



Hình 5-34 Kết nối thiết bị cuối RS-232 với máy tính qua đường truyền nối tiếp.

Các thiết bị cuối RS-232 vẫn được sử dụng phổ biến trong hệ thống máy tính lớn để kết nối với người dùng từ xa, đôi khi có thể sử dụng modem và đường truyền điện thoại. Ta có thể thấy chúng được ứng dụng trong ngành hàng không, ngân hàng, và trong các lĩnh vực khác. Thậm chí khi ta thay thế chúng bằng các máy tính cá nhân, thì các máy tính cá nhân vẫn thường mô phỏng các thiết bị cuối RS-232 cũ để tránh phải thay đổi phần mềm máy tính lớn.

Các thiết bị cuối này cũng được sử dụng phổ biến trong các máy tính mini. Có một khối lượng phần mềm khổng lồ được xây dựng dựa trên các thiết bị cuối này. Ví dụ, tất cả các hệ thống UNIX đều hỗ trợ loại thiết bị này.

Hơn nữa, nhiều hệ thống UNIX hiện nay (và các hệ thống khác) còn cung cấp tính năng tuỳ chọn để tạo ra một cửa sổ chứa được các dòng văn bản. Nhiều lập trình viên hầu như chỉ làm việc trong chế độ văn bản trên các cửa sổ như vậy, thậm chí cả khi làm việc trên máy tính cá nhân hay các trạm làm việc cuối. Các cửa sổ này thường mô phỏng các thiết bị cuối RS-232 (hoặc các chuẩn ANSI dành cho thiết bị loại này), nên họ có thể chạy các phần mềm lớn có sẵn được viết cho các thiết bị như vậy. Trong nhiều năm, các phần mềm (như các bộ soạn thảo *vi* và *emacs*) đã được gỡ lỗi hoàn chỉnh và cực kỳ ổn định, những đặc tính mà các lập trình viên rất coi trọng.

Bàn phím và phần mềm dùng cho các cửa sổ mô phỏng thiết bị cuối này cũng giống như khi dùng cho các thiết bị cuối thực. Do các mô phỏng thiết bị cuối được sử dụng rất rộng rãi, phần mềm này vẫn rất quan trọng, nên ta sẽ mô tả nó trong hai mục tiếp theo.

Các thiết bị cuối RS-232 là các thiết bị hướng ký tự. Nghĩa là màn hình hay cửa sổ hiển thị bao gồm nhiều dòng văn bản, với một kích thước cực đại nào đó. Ví dụ như một màn hình thường có 25 dòng, mỗi dòng có 80 ký tự. Các thiết bị cuối này thường chỉ làm việc ở chế độ text, và thỉnh thoảng cũng hỗ trợ một số ký tự đặc biệt.

Do các máy tính và các thiết bị cuối làm việc với các ký tự, nhưng lại kết nối với với đường truyền nối tiếp truyền từng bít một, nên cần có các chip làm nhiệm vụ chuyển đổi từ dữ liệu ký tự sang các bít nối tiếp và ngược lại. Chúng được gọi là các chip UART (Universal Asynchronous Receiver Transmitters). Các UART được gắn vào máy tính bằng card giao diện RS-232 trên một bus như minh hoạ ở hình 5-34. Trên nhiều máy tính còn tích hợp sẵn một hoặc hai cổng nối tiếp vào bảng mạch chính.

Để hiển thị một ký tự, chương trình điều khiển thiết bị cuối sẽ ghi ký tự vào một bộ đệm trên card giao diện, rồi đẩy nó vào đường truyền nối tiếp nhờ bộ chuyển đổi UART. Ví dụ, với một

modem analog hoạt động ở tốc độ 56000 bps, sẽ chỉ cần khoảng 179 μs để gửi một ký tự. Với tốc độ truyền chậm như vậy, chương trình điều khiển sẽ gửi một ký tự ra card RS-232 và dừng lại, chờ cho tới khi xuất hiện ngắt gửi về từ card giao diện báo hiệu ký tự đã được truyền xong, và UART có thể nhận ký tự tiếp theo. UART có thể đồng thời gửi và nhận các ký tự. Một ngắt cũng có thể phát sinh khi nhận được một ký tự, và thường thì một số ít ký tự nhận được có thể được đưa vào bộ đệm. Chương trình điều khiển thiết bị cuối phải kiểm tra một thanh ghi khi nhận được tín hiệu ngắt để xác định nguyên nhân gây ra ngắt. Một số card giao diện còn chứa cả CPU và bộ nhớ, và có thể xử lý đa kênh, giúp giảm tải cho CPU chính trong các thao tác vào/ra.

Các thiết bị cuối RS-232 có thể được chia làm ba loại. Loại đơn giản nhất là các thiết bị in sao cứng (tương tự như máy in). Các ký tự được nhập từ bàn phím sẽ được truyền tới máy tính. Sau đó các ký tự gửi đi từ máy tính sẽ được in ra giấy. Các thiết bị cuối này đã lỗi thời và rất ít gặp.

Các thiết bị cuối CRT cũng làm việc theo cách như vậy, nhưng không phải là in ra giấy mà gửi ra màn hình. Chúng thường được gọi là máy đánh chữ bằng kính - glass tty, vì chức năng của chúng cũng tương tự như thiết bị in sao cứng. Từ "tty" là viết tắt của Teletype - máy điện báo đánh chữ, cũng là tên một công ty đi đầu trong kinh doanh thiết bị cuối máy tính. Các thiết bị này bây giờ cũng rất hiếm gặp.

Các thiết bị cuối CRT thông minh thực chất chính là những máy tính thu nhỏ. Chúng có CPU và bộ nhớ, và có thể chứa được phần mềm (thường nằm trong ROM). Nhìn từ góc độ hệ điều hành, khác biệt chính giữa "glass tty" và các thiết bị cuối thông minh là ở các mã thoát (escape). Ví dụ, bằng cách gửi ký tự ESC (có mã ASCII bằng 0x1B), theo sau là các ký tự khác, người ta có thể dịch chuyển con trỏ tới bất kỳ vị trí nào trên màn hình, chèn chuỗi ký tự vào giữa màn hình… Các thiết bị cuối thông minh hiện đang được sử dụng trong nhiều hệ thống máy tính lớn, và được mô phỏng bởi các hệ điều hành khác nhau. Tiếp theo ta sẽ thảo luận về phần mềm của chúng.

5.6.2 Phần mềm đầu vào

Bàn phím và màn hình là các thiết bị độc lập với nhau, nên ta sẽ nghiên cứu chúng một cách riêng rẽ. Nhưng chúng cũng có quan hệ với nhau, vì khi gõ các ký tự trên bàn phím thì chúng sẽ hiện ra ở màn hình.

Nhiệm vụ của chương trình điều khiển bàn phím (keyboard driver) là lấy dữ liệu đầu vào từ bàn phím và gửi nó cho các chương trình của người dùng, khi các chương trình này muốn đọc dữ liệu từ thiết bị cuối. Có hai cách để thực hiện điều đó. Ở cách thứ nhất, chương trình điều khiển chỉ việc nhận dữ liệu đầu vào và gửi nó đi mà không phải sửa đổi gì cả. Chương trình người dùng sẽ nhận được một chuỗi ký tự ASCII bao gồm toàn bộ các ký tự đã gõ. (Người ta sử dụng mã ASCII thay cho các mã phím vì các mã phím rất thô sơ và bị phụ thuộc vào từng dòng máy cụ thể).

Giải pháp này rất phù hợp với nhu cầu của các chương trình soạn thảo màn hình hiện đại như *emacs*, nó cho phép người dùng có thể gõ các ký tự một cách tuỳ ý. Tức là khi người dùng gõ *dste* chứ không phải là *date* và sau đó sửa lại bằng cách gõ ba phím xoá lùi rồi gõ *ate*, chương trình của người dùng sẽ nhận được một chuỗi gồm 11 mã ASCII như sau:

d s t e ← ← ← a t e CR

Không phải tất cả các chương trình đều muốn nhận được đầy đủ các ký tự như vậy. Thường thì chúng chỉ muốn nhận được dữ liệu đầu vào phù hợp, không cần quan tâm dữ liệu được gõ như thế nào. Điều này dẫn tới giải pháp thứ hai: Chương trình điều khiển sẽ xử lý các ký tự được nhập, và chỉ gửi các dòng dữ liệu phù hợp tới chương trình của người dùng. Giải pháp thứ nhất là hướng ký tự, còn giải pháp thứ hai là hướng dòng. Ban đầu chúng được gọi là chế độ **raw** (sống) và chế độ **cooked** (chín). Theo tiêu chuẩn POSIX thì chế độ hướng dòng được gọi là **canonical** (hợp chuẩn), còn chế độ raw được gọi là **Noncanonical** (không hợp chuẩn), mặc dù nhiều chi tiết của thiết bị cuối này có thể được thay đổi. Các hệ thống tương thích với POSIX cung cấp một số hàm thư viện để hỗ trợ việc lựa chọn chế độ hoạt động và có thể thay đổi các cấu hình thiết bị.

Nhiệm vụ trước tiên của chương trình điều khiển bàn phím là tiếp nhận các ký tự. Nếu mỗi lần ấn phím đều phát sinh ngắt, thì chương trình điều khiển có thể nhận được ký tự khi có ngắt. Nếu

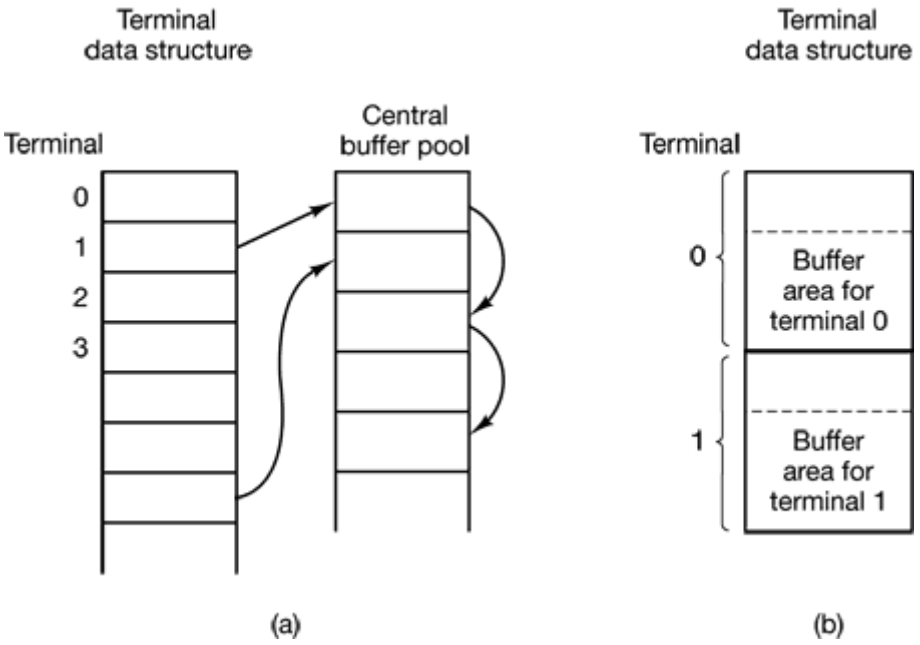
các ngắt được chuyển đổi thành các thông báo bởi một phần mềm cấp thấp, thì có thể đặt ký tự mới nhận được vào thông báo. Cũng có thể đặt ký tự đó vào một bộ đệm nhỏ trong bộ nhớ, và dùng thông báo để nói với chương trình điều khiển về ký tự vừa nhập. Cách tiếp cận thứ hai sẽ an toàn hơn nếu thông báo chỉ có thể gửi tới tiến trình đang chờ, và chương trình điều khiển bàn phím thì đang bận rộn với ký tự trước đó.

Nếu thiết bị cuối ở chế độ hợp chuẩn (cooked), các ký tự phải được lưu lại cho tới khi toàn bộ dòng dữ liệu được nhập xong, vì người dùng có thể muốn xoá một phần của nó sau đó. Thậm chí nếu thiết bị cuối ở chế độ raw, chương trình có thể vẫn chưa yêu cầu về dữ liệu đầu vào, nên các ký tự phải được đưa vào bộ đệm để có thể tiếp tục gõ phím. (Các nhà thiết kế hệ thống không cho phép người dùng được nhập trước dữ liệu có lẽ nên bị trừng phạt bằng cách bôi hắc ín rồi phủ lông chim lên, hoặc tệ hơn là bắt họ phải sử dụng chính hệ thống của họ!)

Có hai cách để cất ký tự vào bộ đệm. Trong cách thứ nhất, chương trình điều khiển sẽ sử dụng một chuỗi các bộ đệm chung, mỗi bộ đệm chứa được khoảng 10 ký tự. Kết hợp với mỗi thiết bị cuối là một cấu trúc dữ liệu, cấu trúc này có nhiều mục, trong đó có một con trỏ trỏ tới chuỗi các bộ đệm chứa dữ liệu nhập vào từ thiết bị cuối. Khi có nhiều ký tự hơn được nhập, sẽ cần tới nhiều bộ đệm hơn và chúng được móc nối vào chuỗi. Khi các ký tự được gửi tới chương trình của người dùng, các bộ đệm sẽ được giải phóng và xếp vào cuối chuỗi.

Cách tiếp cận thứ hai là đặt bộ đệm trực tiếp trong cấu trúc dữ liệu của thiết bị cuối, chứ không sử dụng chuỗi bộ đệm chung. Do mỗi lệnh người dùng gõ sẽ tiêu tốn một khoảng thời gian (ví dụ để biên dịch lại và liên kết một chương trình nhị phân lớn), rồi sau đó họ có thể sẽ gõ một vài dòng khác, nên để an toàn thì chương trình điều khiển sẽ cấp phát một bộ đệm khoảng 200 ký tự cho mỗi thiết bị cuối. Trong một hệ thống chia sẻ thời gian quy mô lớn với 100 thiết bị cuối, tổng kích thước các bộ đệm riêng lên tới 20K - rõ ràng là quá lãng phí, nên một bộ đệm chung với kích thước khoảng 5K có lẽ cũng đủ dùng. Tuy nhiên, một bộ đệm riêng cho mỗi thiết bị cuối sẽ giúp cho chương trình điều khiển trở nên đơn giản hơn (không cần phải quản lý danh sách liên kết nữa), và nó cũng được ưa chuộng hơn trên các máy tính cá nhân chỉ có một bàn phím. Hình 5-35 trình bày sự khác biệt giữa hai cách thức này.

Mặc dù bàn phím và màn hình là các thiết bị độc lập, nhiều người dùng vẫn có thói quen nhìn thấy ký tự họ gõ phải hiện ra trên màn hình. Một số thiết bị cũ hơn được trang bị phần cứng tự động hiển thị mỗi khi gõ phím, điều này không chỉ gây phiền toái khi cần nhập mật khẩu mà còn hạn chế tính mềm dẻo của các bộ soạn thảo hiện đại, cũng như các chương trình khác. May thay, đối với hầu hết các thiết bị cuối, sẽ không có gì được tự động hiện ra khi gõ phím. Tất cả sẽ được gửi về phần mềm trong máy tính để hiển thị ký tự (nếu muốn). Quá trình này được gọi là **echoing (gửi về)**.



Hình 5-35 (a) Sử dụng bộ đệm chung (Central buffer pool).
(b) Sử dụng bộ đệm riêng cho từng thiết bị cuối.

Việc gửi về là phức tạp vì một chương trình có thể ghi lên màn hình trong khi người dùng gõ bàn phím. Ít nhất thì chương trình điều khiển bàn phím cũng phải tính toán nơi đặt dữ liệu mới để tránh bị ghi đè lên bởi kết xuất của chương trình.

Việc gửi về cũng phức tạp khi có nhiều hơn 80 ký tự cần hiển thị trên một màn hình có độ dài một dòng tối đa là 80 ký tự. Tuỳ vào từng ứng dụng, việc ngắt các ký tự thừa xuống dòng tiếp theo có thể được thực hiện. Một số chương trình điều khiển sẽ cắt bỏ tất cả các ký tự vượt ra khỏi phạm vi của cột 80.

Một vấn đề khác là xử lý phím tab. Thường thì chương trình điều khiển sẽ tính toán vị trí hiện hành của con trỏ, chuyển nó cho cả đầu ra của chương trình và đầu ra của quá trình gửi về, rồi tính toán số lượng khoảng trống tương ứng để gửi về.

Bây giờ ta sẽ xem xét một vấn đề tương tự. Về mặt logic, ở cuối mỗi dòng văn bản cần có một ký tự CR (carriage return) để chuyển con trỏ về cột thứ nhất, và một ký tự LF (linefeed) để chuyển con trỏ xuống dòng tiếp theo. Việc đòi hỏi người dùng phải gõ cả hai ký tự này là không hợp lý (mặc dù một số thiết bị cuối có trang bị một phím ứng với cả hai ký tự này, với 50% cơ may là các phần mềm có thể chấp nhận chúng). Để làm được điều đó thì chương trình điều khiển phải chuyển đổi mọi thứ sang định dạng chuẩn mà hệ điều hành có thể sử dụng.

Nếu định dạng chuẩn chỉ chứa một ký tự LF (quy ước của UNIX), thì các ký tự CR sẽ được đổi thành LF. Nếu định dạng chuẩn chứa cả hai ký tự (quy ước của Windows), thì chương trình điều khiển sẽ tạo ra một ký tự LF khi nó nhận được ký tự CR, và tạo ra ký tự CR khi nhận được LF. Bất kể quy ước như thế nào, thiết bị cuối cũng có thể đòi hỏi cả LF và CR để gửi về nhằm cập nhật trạng thái màn hình. Do một máy tính lớn có thể có nhiều thiết bị cuối khác nhau kết nối với nó, chương trình điều khiển bàn phím sẽ phải chuyển đổi tất cả các tổ hợp phím CR/LF sang dạng chuẩn của hệ thống, và chuẩn bị cho quá trình gửi về diễn ra suôn sẻ. Khi hoạt động ở chế độ hợp chuẩn, một số ký tự đầu vào sẽ có những ý nghĩa đặc biệt. Hình 5-36 liệt kê các ký tự đặc biệt theo quy ước của POSIX. Ở đây mặc định tất cả các ký tự điều khiển sẽ không xung đột với các mã đầu vào hay các mã được sử dụng bởi chương trình, trừ hai ký tự cuối cùng ra thì tất cả các ký tự khác có thể được thay đổi bởi chương trình.

Ký tự	Tên trong POSIX	Giải thích

CTRL-H	ERASE	Xoá lùi một ký tự
CTRL-U	KILL	Xóa toàn bộ dòng hiện hành
CTRL-V	LNEXT	Chuyển ký tự tiếp theo về đúng dạng
CTRL-S	STOP	Dừng kết xuất
CTRL-Q	START	Bắt đầu kết xuất
DEL	INTR	Ngắt tiến trình (SIGINT)
CTRL-\	QUIT	Buộc kết thúc chương trình (SIGQUIT)
CTRL-D	EOF	Kết thúc file
CTRL-M	CR	Carriage return (không thay đổi được)
CTRL-J	NL	Linefeed (không thay đổi được)

Hình 5-36 Các ký tự được xử lý đặc biệt trong chế độ hợp chuẩn.

Ký tự *ERASE* cho phép người dùng xoá ký tự vừa gõ, nó thường là phím backspace (CTRL-H). Nó không đặt thêm ký tự vào hàng đợi mà xoá ký tự trước đó khỏi hàng. Nó sẽ gửi một chuỗi gồm ba ký tự: xoá lùi (backspace), khoảng trống (space), và xoá lùi để xoá ký tự trước đó khỏi màn hình. Nếu ký tự trước đó là một tab, việc xoá ký tự này còn phụ thuộc vào cách nó được xử lý khi gõ. Nếu nó gồm nhiều khoảng trống, sẽ cần có thêm thông tin để xác định số lượng khoảng trống. Nếu ký tự tab được lưu trong hàng đợi, có thể xoá nó và thực hiện kết xuất lại toàn bộ dòng. Trong hầu hết các hệ thống, việc xoá lùi chỉ xoá các ký tự trên dòng hiện hành. Nó sẽ không xoá ký tự CR và quay trở lại dòng trước đó.

Khi người dùng phát hiện ra một lỗi ở đầu dòng ký tự, họ thường xoá cả dòng đi và gõ lại từ đầu. Ký tự *KILL* sẽ giúp xoá toàn bộ dòng. Hầu hết các hệ thống sẽ xoá dòng ký tự trên màn hình, nhưng cũng có một vài hệ thống chỉ chèn thêm các ký tự CR và LF vì một số người dùng muốn xem lại dòng ký tự cũ. Do đó, cách thức hoạt động của *KILL* còn phụ thuộc vào sở thích. Giống như *ERASE*, nó cũng thường chỉ xoá ở dòng hiện tại chứ không xoá lên dòng trước đó. Khi một khối ký tự bị xoá, có thể chương trình điều khiển sẽ phải tốn một ít thời gian để giải phóng các bộ đệm dùng chung.

Đôi khi các ký tự *ERASE* hay *KILL* phải được coi là các dữ liệu thông thường. Ký tự *LNEXT* hoạt động giống như một **ký tự thoát (escape)**. Trên UNIX mặc định nó là CTRL-V. Ví dụ, các hệ thống UNIX cũ hơn thường dùng ký hiệu @ thay cho *KILL*, nhưng các hệ thống thư điện tử trên Internet lại sử dụng địa chỉ có dạng *linda@cs.washington.edu*. Ai đó thích dùng quy ước cũ có thể định nghĩa lại *KILL* là @, nhưng sau đó sẽ phải vất vả khi muốn gõ dấu @ trong địa chỉ email. Điều này có thể thực hiện được bằng cách gõ CTRL-V @. Ký tự CTRL-V có thể chuyển chính nó về đúng dạng bằng cách gõ CTRL-V CTRL-V. Khi nhìn thấy CTRL-V, chương trình điều khiển sẽ thiết lập một cờ để thông báo ký tự tiếp theo sẽ không bị coi là ký tự đặc biệt. Ký tự *LNEXT* sẽ không bị xếp vào hàng đợi.

Để người dùng có thể dừng màn hình khi đang cuộn, có thể sử dụng các mã điều khiển nhằm dừng màn hình và khởi động lại nó sau đó. Trên UNIX có các mã *STOP* (CTRL-S) và *START* (CTRL-Q). Chúng không được lưu lại, mà chỉ có tác dụng thiết lập hoặc xoá một cờ trong cấu trúc dữ liệu của thiết bị cuối. Mỗi khi thực hiện kết xuất, cờ này sẽ được kiểm tra. Nếu nó đã được thiết lập, sẽ không thực hiện kết xuất nữa. Thông thường, việc gửi về cũng sẽ bị chặn cùng với kết xuất chương trình.

Đôi khi cũng cần dừng đột ngột một chương trình đang chạy để gỡ lỗi. Các ký tự *INTR* (DEL) và *QUIT* (CTRL-\) có thể được sử dụng cho mục đích này. Trên UNIX, DEL sẽ gửi tín hiệu SIGINT tới tất cả các tiến trình được khởi động từ thiết bị cuối. Việc thực hiện DEL có thể phải khéo léo một chút. Vấn đề khó khăn là việc lấy thông tin từ chương trình điều khiển thiết bị để đưa tới bộ phận xử lý tín hiệu của hệ thống, rồi sau đó sẽ không cần tới thông tin này nữa. CTRL-

\ cũng tương tự như DEL, chỉ khác là nó gửi tín hiệu SIGQUIT để buộc kết thúc chương trình và kết xuất trạng thái ra file. Khi các phím này được ấn, chương trình điều khiển thiết bị sẽ thực hiện gửi về hai ký tự CR-LF và loại bỏ tất cả các dữ liệu đầu vào hiện có, để sẵn sàng cho một sự khởi đầu mới. Giá trị mặc định của *INTR* thường là CTRL-C chứ không phải DEL, do nhiều chương trình sử dụng DEL thay cho phím xoá lùi khi soạn thảo.

Một ký tự đặc biệt khác là *EOF* (CTRL-D), trong UNIX nó sẽ đáp ứng các yêu cầu đọc thiết bị cuối chưa được xử lý bằng các dữ liệu đang nằm trong bộ đệm, thậm chí cả khi bộ đệm còn trống. Việc gõ CTRL-D ở đầu dòng sẽ tương ứng với việc chương trình đọc 0 byte dữ liệu, theo quy ước nó sẽ được dịch là dấu hiệu kết thúc file, và các chương trình sẽ xử lý nó giống như khi đọc hết một file dữ liệu đầu vào.

Một số chương trình điều khiển thiết bị cuối còn cung cấp nhiều tính năng soạn thảo khác nữa. Chúng có các ký tự điều khiển đặc biệt để xoá một từ, dịch con trỏ về đầu từ hoặc cuối từ, di chuyển tới đầu dòng hoặc cuối dòng đang gõ, chèn văn bản vào giữa dòng… Việc thêm các tính năng này vào chương trình điều khiển thiết bị cuối sẽ khiến nó trở nên lớn hơn, và còn gây lãng phí nếu chương trình soạn thảo màn hình chỉ hoạt động trong chế độ raw.

5.6.3 Phần mềm đầu ra

Đầu ra thì đơn giản hơn đầu vào. Đối với hầu hết các trường hợp, máy tính gửi các ký tự ra thiết bị cuối và hiển thị chúng ở đó. Thông thường, một khối ký tự (ví dụ như một dòng) sẽ được gửi ra thiết bị cuối bằng một lời gọi hệ thống. Phương thức phổ biến trên các thiết bị RS-232 là sử dụng các bộ đệm đầu ra kết hợp với mỗi thiết bị cuối. Có thể sử dụng bộ đệm tập trung hoặc bộ đệm riêng giống như ở đầu vào. Khi chương trình ghi dữ liệu ra thiết bị cuối, trước hết nó sẽ được sao chép vào bộ đệm. Tương tự, các kết xuất của quá trình gửi về cũng sẽ được sao chép vào bộ đệm. Sau khi toàn bộ dữ liệu ra được sao chép vào bộ đệm, ký tự đầu tiên sẽ được kết xuất, và chương trình điều khiển sẽ "đi ngủ". Khi xuất hiện tín hiệu ngắt, ký tự tiếp theo sẽ được gửi ra…

Các chương trình soạn thảo màn hình và nhiều chương trình hiện đại khác có thể sẽ cần cập nhật màn hình theo những cách phức tạp, như thay thế một dòng nằm giữa màn hình. Để đáp ứng nhu cầu này, hầu hết các thiết bị cuối đều hỗ trợ nhiều lệnh dịch chuyển con trỏ, chèn hay xoá ký tự và dòng tại vị trí con trỏ… Các mã lệnh này thường được gọi là **chuỗi thoát (escape sequence)**. Ở thời kỳ cực thịnh của các thiết bị cuối RS-232, có tới hàng trăm loại thiết bị này, mỗi loại lại có chuỗi thoát riêng. Do đó, rất khó để viết phần mềm dùng được cho nhiều loại.

Có một giải pháp được giới thiệu trong Berkeley UNIX, nó sử dụng một cơ sở dữ liệu thiết bị cuối gọi là **termcap**. Gói phần mềm này định nghĩa nhiều hoạt động cơ bản, như di chuyển con trỏ màn hình theo hàng và cột (*row, column*). Để dịch chuyển con trỏ tới một vị trí cụ thể, phần mềm (hay chương trình soạn thảo) sử dụng một chuỗi thoát chung, sau đó nó được chuyển đổi thành chuỗi thoát cụ thể cho thiết bị cần ghi dữ liệu ra. Theo cách đó, chương trình soạn thảo có thể làm việc với bất kỳ thiết bị cuối nào được liệt kê trong cơ sở dữ liệu termcap.

Thậm chí trong công nghiệp người ta còn nhận thấy nhu cầu cần chuẩn hoá các chuỗi thoát, nên một tiêu chuẩn ANSI đã được phát triển. Một vài giá trị của nó được liệt kê ở hình 5-37.

Ta sẽ xem xét cách sử dụng các chuỗi thoát này trong các chương trình soạn thảo văn bản. Giả sử người dùng gõ một lệnh yêu cầu chương trình soạn thảo xoá toàn bộ dòng thứ 3, rồi xoá khoảng trống giữa dòng 2 và 4. Chương trình soạn thảo có thể gửi chuỗi thoát sau qua đường truyền nối tiếp tới thiết bị cuối:

```
ESC [ 3 ; 1 H ESC [ 0 K ESC [ 1 M
```

(các khoảng trống dùng trong chuỗi trên chỉ có tác dụng phân cách các ký hiệu, chũng sẽ không được truyền đi). Chuỗi này sẽ di chuyển con trỏ tới dòng thứ 3, xoá toàn bộ dòng, rồi sau đó xoá nốt dòng trống, làm cho tất cả các dòng tính từ dòng 3 được dịch lên một dòng. Dòng 4 trở thành dòng 3, dòng 5 trở thành dòng 4… Có thể sử dụng các chuỗi tương tự như vậy để chèn văn bản vào giữa màn hình, cũng có thể chèn hay xoá các từ theo cách tương tự.

Chuỗi thoát	Ý nghĩa

ESC [*n* A	Di chuyển lên *n* dòng
ESC [*n* B	Di chuyển xuống *n* dòng
SSC [*n* C	Dịch sang phải *n* khoảng trống
ESC [*n* D	Dịch sang trái *n* khoảng trống
ESC [*m* ; *n* H	Chuyển con trỏ tới toạ độ (*m*,*n*)
ESC [*s* J	Xoá màn hình từ vị trí con trỏ (0 tới cuối, 1 từ đầu, 2 tất cả)
ESC [*s* K	Xoá dòng từ vị trí con trỏ (0 tới cuối, 1 từ đầu, 2 tất cả)
ESC [*n* L	Chèn *n* dòng tại vị trí con transistorỏ
ESC [*n* M	Xoá *n* dòng tại vị trí con trỏ
ESC [*n* P	Xoá *n* ký tự tại vị trí con trỏ
ESC [*n* @	Chèn *n* ký tự tại vị trí con transistorỏ
ESC [*n* m	Đặt chế độ *n* (0 = bình thường, 4 = đậm, 5 = nhấp nháy, 7 = ngược)
ESC M	Cuộn màn hình ngược lên nếu con trỏ ở dòng trên cùng

Hình 5-37 Các chuỗi thoát theo chuẩn ANSI dùng cho các chương trình điều khiển thiết bị đầu ra. ESC là ký tự thoát ASCII (có mã bằng 0x1B), còn n, m, và s là các tham số tuỳ chọn.

5.10 NGHIÊN CỨU VỀ QUẢN LÝ VÀO/RA

Có khá nhiều nghiên cứu về quản lý vào/ra, nhưng hầu hết đều dựa trên các thiết bị cụ thể chứ không phải các thiết bị vào/ra tổng quát. Mục đích của các nghiên cứu thường là cải thiện hiệu suất theo cách này hay cách khác.

Hệ thống đĩa là một lĩnh vực được chú ý. Các giải thuật điều độ cánh tay đĩa cũ sử dụng một mô hình đĩa không còn áp dụng được nữa, nên Worthington và cộng sự (1994) đã tiến hành nghiên cứu về các mô hình dùng cho các đĩa hiện đại. RAID là một chủ đề nóng, được nhiều người nghiên cứu ở nhiều khía cạnh khác nhau. Alvarez và các cộng sự (1997) đã nghiên cứu về khả năng chịu đựng sự cố, giống như nhóm của Blaum đã làm (1994). Wilkes và các đồng nghiệp (1996) đã mô tả về một hệ thống RAID nâng cao mà họ xây dựng ở HP. Việc sử dụng nhiều thiết bị đòi hỏi khả năng xử lý song song phải tốt, nên vấn đề này cũng trở thành một chủ đề cần nghiên cứu (Chen và Towsley, 1996; Kallahalla và Varman, 1999). Nhóm của Lumb (2000) đã đưa ra ý tưởng về việc tận dụng thời gian rỗi sau khi tìm kiếm và trước khi sector cần tìm quay tới bên dưới đầu đọc để nạp trước dữ liệu. Thậm chí còn có thể loại trừ độ trễ do quay đĩa nhờ sử dụng các ổ đĩa thể rắn chế tạo bằng công nghệ vi điện tử (Griffin và cộng sự, 2000; Carley và cộng sự, 2000), hoặc sử dụng các ổ đĩa holographic (Orlov, 2000). Một công nghệ mới khác cũng được nghiên cứu là các ổ đĩa từ - quang (McDaniel, 2000).

Các thiết bị cuối SLIM tạo ra một phiên bản mới của hệ thống chia sẻ thời gian, trong đó tất cả các tính toán được thực hiện tập trung, và chỉ cung cấp cho người dùng các thiết bị như màn hình, chuột, phím, ngoài ra không còn gì khác (Schmidt và cộng sự, 1999). Sự khác biệt chính của nó với các hệ thống chia sẻ thời gian cũ là thay vì kết nối thiết bị cuối với máy tính qua modem 9600-bps, người ta sẽ sử dụng mạng Ethernet 10-Mbps, nó sẽ cung cấp đủ băng thông cho giao diện đồ hoạ đối với người dùng cuối.

GUI hiện được sử dụng khá phổ biến, và vẫn đang được tiếp tục nghiên cứu, ví dụ như các nghiên cứu về điều khiển bằng giọng nói (Malkewitz, 1998; Manaris và Harkreader, 1998; Slaughter và cộng sự, 1998; Van Buskirk và LaLomia, 1995). Cấu trúc nội tại của GUI cũng là một chủ đề được quan tâm (Taylor và cộng sự, 1995).

Máy tính xách tay ngày càng trở nên phổ biến, nên không có gì đáng ngạc nhiên khi ngày càng có nhiều người quan tâm nghiên cứu tới lĩnh vực tiết kiệm năng lượng pin bằng các giải pháp phần mềm (Ellis, 1999; Flinn và Satyanarayanan, 1999; Kravets và Krishnan, 1998; Nhóm của Lebeck, 2000; Larch và Smith, 1996; Nhóm của Lu, 1999).

5.11 TÓM TẮT

Vấn đề vào/ra thường ít được chú ý, nhưng đây là một chủ đề rất quan trọng. Một phần đáng kể của bất cứ hệ điều hành nào liên quan chặt chẽ tới vấn đề vào/ra. Có thể thực hiện vào/ra bằng một trong ba cách. Cách thứ nhất là sử dụng vào/ra theo chương trình, trong đó việc giao tiếp với CPU chính được thực hiện theo từng byte hay word, chúng được đặt vào một vòng lặp kín và chờ cho tới khi gửi hoặc nhận được dữ liệu tiếp theo. Cách thứ hai là sử dụng vào/ra điều khiển ngắt, trong đó CPU sẽ khởi tạo quá trình truyền một ký tự hay word, rồi chuyển sang làm công việc khác cho tới khi xuất hiện tín hiệu ngắt báo hiệu quá trình vào/ra hoàn thành. Cách thứ ba là sử dụng DMA, trong đó một chip riêng sẽ được dùng để quản lý quá trình truyền khối dữ liệu, và nó chỉ phát sinh ngắt khi toàn bộ khối dữ liệu đã được truyền xong.

Có bốn cấp độ vào/ra: các thủ tục xử lý ngắt, các chương trình điều khiển thiết bị, các phần mềm vào/ra độc lập thiết bị, và các thư viện vào/ra cùng các spooler chạy trong không gian người dùng. Các chương trình điều khiển thiết bị sẽ xử lý các chi tiết khi chạy thiết bị và cung cấp giao diện đồng nhất tới phần còn lại của hệ điều hành. Phần mềm vào/ra độc lập thiết bị hoạt động giống như một tầng đệm và đưa ra các thông báo lỗi.

Có rất nhiều loại đĩa khác nhau, bao gồm đĩa từ, các đĩa RAID, và nhiều loại đĩa quang khác nhau. Có thể sử dụng các giải thuật điều khiển cánh tay đĩa để cải thiện hiệu suất đĩa, nhưng sự xuất hiện của kiến trúc hình học ảo cũng làm vấn đề trở nên phức tạp hơn. Bằng cách ghép hai đĩa lại với nhau người ta có thể tạo ra một ổ đĩa mới với những đặc tính hữu ích nào đó.

Đồng hồ được sử dụng để quản lý thời gian thực, giới hạn thời gian chạy của tiến trình, xử lý các bộ định thời bảo vệ, và tính toán thời gian chiếm dụng CPU.

Các thiết bị hướng ký tự cũng có nhiều vấn đề cần quan tâm, như việc nhập các ký tự đặc biệt hay việc kết xuất các chuỗi thoát. Đầu vào có thể ở chế độ raw hoặc chế độ cooked, tuỳ thuộc vào mức độ mà chương trình muốn điều khiển đầu vào. Các chuỗi thoát ở đầu ra sẽ điều khiển sự di chuyển của con trỏ, và cho phép chèn hoặc xoá văn bản trên màn hình.

Nhiều máy tính cá nhân sử dụng giao diện đồ hoạ GUI ở đầu ra. Chúng dựa trên mô hình WIMP, bao gồm: các cửa sổ, biểu tượng, menu, và thiết bị con trỏ. Các chương trình trên GUI được điều khiển bằng các biến cố từ bàn phím, chuột, và các biến cố khác gửi tới chương trình để xử lý ngay khi chúng xảy ra.

Các thiết bị mạng cũng khá đa dạng. Một số thiết bị phổ biến nhất thường chạy phần mềm X, đây là một hệ thống tinh vi dùng để tạo ra nhiều loại GUI khác nhau. Một lựa chọn khác với X Windows là sử dụng giao diện cấp thấp, được thực hiện bằng cách truyền các điểm ảnh qua mạng. Các thử nghiệm với thiết bị SLIM cho thấy khả năng hoạt động đáng ngạc nhiên của kỹ thuật này.

Cuối cùng, việc quản lý điện năng là một vấn đề quan trọng trên các máy tính xách tay, vì thời gian hoạt động của pin là hữu hạn. Các kỹ thuật khác nhau có thể được hệ điều hành áp dụng để giảm điện năng tiêu thụ. Các chương trình cũng có thể giúp đỡ bằng cách giảm bớt chất lượng để kéo dài thời gian cho pin.

BÀI TẬP CHƯƠNG 5

1. Các công nghệ sản xuất chip hiện đại cho phép tích hợp toàn bộ controller, bao gồm tất cả các bus logic, lên cùng một chip giá rẻ. Ảnh hưởng của điều này tới mô hình ở hình 1-5 sẽ như thế nào?
2. Trong danh sách các tốc độ ở hình 5.1, liệu có thể quét một văn bản từ scanner rồi cất lên một đĩa EIDE được nối với bus ISA với tốc độ cao nhất? Hãy giải thích.

3. Hình 5-3(b) trình bày một giải pháp ánh xạ cổng vào/ra tới bộ nhớ ngay cả khi sử dụng các bus khác nhau cho bộ nhớ và thiết bị vào/ra. Cụ thể, nếu thất bại khi truy nhập bus bộ nhớ thì có thể thử truy nhập vào bus vào/ra. Một sinh viên ngành khoa học máy tính đã đưa ra ý tưởng cải tiến như sau: sử dụng cả hai bus song song để tăng tốc độ truy nhập vào thiết bị vào/ra. Bạn nghĩ gì về ý tưởng này?
4. Một bộ điều khiển DMA có bốn kênh. Bộ điều khiển có khả năng yêu cầu truyền một word 32 bít cứ sau 100 ns. Thời gian đáp ứng cũng bằng như vậy. Bus phải có tốc độ bao nhiêu để tránh được hiện tượng thắt nút cổ chai?
5. Giả sử một máy tính có thể đọc hoặc ghi một word nhớ trong 10 ns. Cũng giả sử rằng khi một ngắt xảy ra, tất cả các thanh ghi 32 bít của CPU, kể cả bộ đếm chương trình và PSW đều được cất vào ngăn xếp. Số lượng cực đại các ngắt mà máy tính này có thể xử lý trong một giây bằng bao nhiêu?
6. Trên hình 5-8(b), ngắt sẽ không được hoàn thành cho tới khi ký tự được gửi ra máy in. Liệu có thể đặt lệnh thông báo hoàn thành ngắt lên đầu của thủ tục xử lý ngắt được không? Nếu có thì hãy giải thích tại sao lại đặt nó ở cuối thủ tục như trong đoạn mã. Còn nếu không thì tại sao?
7. Một máy tính có kênh dẫn ba tầng như trên hình 1-6(a). Trong mỗi chu kỳ đồng hồ, một lệnh mới sẽ được lấy về từ bộ nhớ tại địa chỉ do bộ đếm chương trình trỏ tới, rồi đặt nó vào kênh dẫn, bộ đếm chương trình được tăng lên. Mỗi lệnh chiếm đúng một word nhớ. Các lệnh đã ở trong kênh dẫn được đẩy lên tầng tiếp theo. Khi có ngắt xảy ra, nội dung của bộ đếm chương trình hiện hành được cất vào ngăn xếp, và nó sẽ nhận giá trị mới là địa chỉ của chương trình xử lý ngắt. Sau đó kênh dẫn được dịch sang phải một tầng, và lệnh đầu tiên của chương trình xử lý ngắt được lấy về kênh dẫn. Liệu máy tính này có các ngắt xác định không?
8. Một trang văn bản in điển hình có 50 dòng, trên mỗi dòng có 80 ký tự. Hãy tưởng tượng một máy in có thể in được 6 trang/1 phút, và thời gian để ghi một ký tự vào thanh ghi đầu ra của máy in là rất nhỏ, có thể bỏ qua. Liệu có thể sử dụng phương pháp vào/ra điều khiển ngắt đối với máy in này, nếu mỗi ký tự cần in ứng với một ngắt và thời gian xử lý ngắt này là 50 μs.
9. Thế nào là "độc lập thiết bị"?
10. Trong bốn lớp phần mềm vào/ra, lớp nào sẽ thực hiện công việc nào dưới đây:
    (a) Tính toán số hiệu track, sector, và đầu đọc để đọc đĩa.
    (b) Ghi các lệnh vào các thanh ghi thiết bị.
    (c) Kiểm tra người dùng có quyền sử dụng thiết bị không.
    (d) Chuyển các số nguyên nhị phân sang mã ASCII để thực hiện in.
11. Dựa vào các số liệu ở hình 5-17, hãy cho biết tốc độ truyền giữa đĩa và controller bằng bao nhiêu (đối với đĩa mềm và đĩa cứng)? So sánh hai tốc độ đó với tốc độ của modem 56-Kbps và mạng Fast Ethernet 100-Mbps.
12. Xét một mạng cục bộ như sau: Người dùng tạo ra một lời gọi hệ thống để gửi các gói dữ liệu vào mạng. Sau đó hệ điều hành sẽ sao chép dữ liệu vào một bộ đệm của kernel. Rồi nó sao chép dữ liệu lên bảng mạch điều khiển mạng. Khi tất cả các byte đã nằm an toàn trong bộ điều khiển, chúng sẽ được gửi lên mạng với tốc độ 10 megabits/s. Bộ điều khiển mạng bên nhận sẽ nhận được từng bít sau mỗi micrô giây. Khi bít cuối cùng đến nơi, CPU đích sẽ nhận được tín hiệu ngắt, và kernel sẽ sao chép gói tin mới nhất vào một bộ đệm của kernel để kiểm tra. Sau khi xác định gói tin đó thuộc về người dùng nào, kernel sẽ sao chép nó vào không gian người dùng tương ứng. Nếu ta giả thiết mỗi ngắt và thời gian xử lý nó là 1 μs, mỗi gói có 1024 byte (bỏ qua phần tiêu đề), và thời gian sao chép 1 byte là 1 μs, thì tốc độ cực đại mà một tiến trình có thể truyền dữ liệu tới máy khác bằng bao nhiêu? Giả sử bên gửi sẽ bị dừng cho tới khi công việc kết thúc và thông báo hoàn thành được gửi trở lại. Để đơn giản, ta giả thiết thời gian gửi tín hiệu phản hồi là rất nhỏ và có thể bỏ qua.

13. Tại sao các file đầu ra của máy in thường được được đặt vào thư mục spooling trên đĩa trước khi in?
14. Độ lệch cylinder cần thiết trên đĩa 7200-vòng/phút bằng bao nhiêu, biết thời gian chuyển từ track này sang track khác là 1 ms? Đĩa có 200 sector/1 track, mỗi sector có 512 byte.
15. Hãy tính tốc độ dữ liệu cực đại (theo MB/s) của đĩa trong bài tập trên.
16. RAID mức 3 có thể sửa các lỗi bít đơn mà chỉ cần dùng tới một ổ đĩa chẵn lẻ. Đặc điểm của RAID mức 2 là gì? Nó cũng chỉ có thể sửa được một lỗi nhưng lại cần tới nhiều ổ đĩa hơn.
17. Một hệ thống RAID có thể bị sập nếu có hai hoặc nhiều ổ đĩa cùng bị hỏng trong một khoảng thời gian ngắn. Giả sử xác suất bị hỏng của một ổ đĩa trong một khoảng thời gian cho trước là *p*. Hãy tính xác suất xảy ra sập hệ thống RAID (có *k* ổ đĩa) trong khoảng thời gian đó.
18. Tại sao các ổ đĩa quang có mật độ dữ liệu cao hơn các ổ đĩa từ? Chú ý: Vấn đề này đòi hỏi một số kiến thức về vật lý phổ thông và cách tạo ra từ trường.
19. Nếu một bộ điều khiển đĩa ghi các byte mà nó nhận được từ đĩa vào bộ nhớ với tốc độ bằng với tốc độ nhận dữ liệu (không dùng bộ đệm trong bộ điều khiển), thì việc áp dụng xen kẽ có ích lợi gì không?
20. Một đĩa mềm sử dụng xen kẽ kép, như trên hình 5-26(c). Nó có 8 sector/1 track, mỗi sector có 512 byte, và quay với tốc độ 300 vòng/phút. Sẽ cần bao nhiêu thời gian để đọc tất cả các sector trên 1 track theo đúng thứ tự? Giả sử cánh tay đĩa đã được đặt vào đúng vị trí, và cần quay thêm 1/2 vòng để di chuyển đầu đọc tới sector 0. Tốc độ dữ liệu sẽ bằng bao nhiêu? Làm lại bài tập này với một đĩa không áp dụng xen kẽ (có cùng các thông số). Tốc độ dữ liệu bị giảm đi bao nhiêu khi áp dụng xen kẽ?
21. Nếu một đĩa sử dụng xen kẽ kép, liệu có cần tới độ lệch cylinder để tránh mất dữ liệu khi chuyển từ track này sang track khác không? Tại sao?
22. Có hai đĩa loại 5.25-inch, mỗi đĩa có 10000 cylinder. Chiếc đĩa mới hơn có mật độ ghi gấp đôi chiếc cũ. Như vậy chiếc đĩa mới sẽ có những đặc tính nào tốt hơn và đặc tính nào vẫn giống như cũ?
23. Một nhà sản xuất máy tính quyết định thiết kế lại bảng phân khu (partition table) cho đĩa cứng của máy Pentium, để có thể tạo ra nhiều hơn bốn phân khu. Sự thay đổi đó sẽ tạo ra những hệ quả gì?
24. Các yêu cầu truy nhập đĩa được gửi tới chương trình điều khiển đĩa theo trật tự các cylinder là 10, 22, 20, 2, 40, 6, và 38. Thời gian để dịch chuyển từ cylinder này sang cylinder khác là 6 ms. Như vậy thời gian cần thiết để tìm kiếm các cylinder trên bằng bao nhiêu khi áp dụng các giải thuật sau:
    (a) FCFS (First-Come, First-Served).
    (b) Chọn cylinder gần nhất.
    (c) Giải thuật thang máy (lúc đầu chuyển động từ dưới lên).

    Trong tất cả các trường hợp, cánh tay đĩa xuất phát từ cylinder 20.
25. Một nhà kinh doanh máy tính cá nhân tới thăm đại học South-West Amsterdam. ông ta kể rằng công ty của ông ta đã bỏ rất nhiều công sức để tăng tốc phiên bản UNIX của họ, và nó có thể chạy với tốc độ rất nhanh. Để lấy ví dụ, ông ta kể rằng phần mềm điều khiển đĩa của họ sử dụng giải thuật thang máy, và sắp xếp các yêu cầu về các sector trong cùng một cylinder vào hàng đợi theo trật tự các sector. Một sinh viên tên là Harry Hacker đã bị thuyết phục và mua một chiếc máy tính. Anh ta mang nó về nhà và viết một chương trình đọc ngẫu nhiên 10000 khối đĩa (block) từ đầu tới cuối đĩa. Nhưng thật ngạc nhiên, hiệu suất đọc mà anh ta đo được giống hệt với hiệu suất khi áp dụng giải thuật first-come, first-served. Có phải nhà kinh doanh đã nói dối không?
26. Trong phần thảo luận về thiết bị lưu trữ tin cậy sử dụng RAM ổn định, ta đã bỏ qua vấn đề sau đây: Điều gì sẽ xảy ra khi quá trình ghi tin cậy hoàn thành, nhưng hỏng hóc lại xảy ra trước khi hệ điều hành có thể ghi số hiệu khối dữ liệu hỏng vào RAM ổn định. Liệu

tình huống đua tranh này có làm phá sản ý tưởng về thiết bị lưu trữ tin cậy? Hãy giải thích.

27. Chương trình xử lý ngắt đồng hồ trên một máy tính dùng hết 2 ms (bao gồm cả chi phí chuyển đổi tiến trình) trong mỗi nhịp đồng hồ. Đồng hồ chạy ở tốc độ 60 Hz. Bao nhiêu phần trăm của CPU được dùng cho việc xử lý ngắt đồng hồ?
28. Nhiều phiên bản của UNIX sử dụng một số nguyên không dấu 32 bít để quản lý thời gian theo số lượng giây tính từ mốc thời gian chuẩn. Khi nào thì các hệ thống này sẽ bị tràn? Bạn có mong điều đó xảy ra không?
29. Một số máy tính cần tới một số lượng lớn các đường kết nối RS-232, ví dụ như máy chủ của các nhà cung cấp Internet. Vì vậy sẽ cần lắp thêm các card kết nối RS-232. Giả sử trên một card như vậy có chứa một bộ vi xử lý, nó phải lấy mẫu mỗi đường tín hiệu đến với tốc độ gấp 8 lần tốc độ baud để kiểm tra xem bít nhận được là 0 hay 1. Cũng giả sử rằng mỗi lần lấy mẫu như vậy mất 1 μs. Với đường truyền tốc độ 28800 bps hoạt động ở 3200 baud, thì bộ vi xử lý này có thể hỗ trợ được bao nhiêu đường như vậy? Chú ý rằng: Tốc độ baud của một đường truyền là số lượng tín hiệu thay đổi trong mỗi giây. Một đường truyền 3200 baud có thể cung cấp tốc độ bít bằng 28800 bps nếu mỗi tín hiệu được mã hoá 9 bít với các biên độ, tần số, và pha khác nhau. Các modem 56K không sử dụng RS-232 nên không được lấy làm ví dụ cho phần này.
30. Tại sao các thiết bị RS-232 được điều khiển bằng ngắt, còn các thiết bị ánh xạ tới bộ nhớ lại không được điều khiển bằng ngắt?
31. Xét hiệu suất của một modem 56-Kbps. Chương trình điều khiển gửi ra modem một ký tự rồi dừng lại. Khi ký tự được in, một ngắt sẽ phát sinh và một thông báo sẽ được gửi về chương trình điều khiển đang dừng, để nó tiếp tục gửi ký tự tiếp theo, rồi lại chuyển sang trạng thái dừng. Nếu thời gian để gửi thông báo, gửi ký tự, và thời gian dừng là 100 μs, thì bao nhiêu phần trăm thời gian của CPU bị tiêu tốn cho quá trình xử lý modem? Giả sử mỗi ký tự có một bít tiêu đề và một bít dừng, tất cả là 10 bít.
32. Một thiết bị hiển thị bitmap có độ phân giải là 1280 x 960 pixels. Để cuộn một cửa sổ, CPU (hoặc controller) phải dịch chuyển tất cả các dòng văn bản lên trên bằng cách sao chép các bít của chúng từ vùng này sang vùng khác trong Video RAM. Nếu một cửa sổ có 60 dòng, mỗi dòng có 80 ký tự (tổng cộng là 4800 ký tự), mỗi ký tự rộng 8 pixel và cao 16 pixel, thì sẽ mất bao nhiêu thời gian để cuộn toàn bộ cửa sổ, biết tốc độ sao chép là 50 ns/1 byte? Nếu tất cả các dòng đều có 80 ký tự thì tốc độ baud tương đương của thiết bị này bằng bao nhiêu? Nếu đặt một ký tự lên màn hình mất 5 μs thì trong một giây có thể hiển thị được bao nhiêu dòng?
33. Sau khi nhận được một ký tự DEL (SIGINT), chương trình điều khiển màn hình sẽ xoá bỏ tất cả các ký tự đang ở trong hàng đợi. Tại sao?
34. Người dùng sử dụng thiết bị RS-232 gửi một lệnh tới chương trình soạn thảo để xoá một từ trên dòng thứ 5, gồm các ký tự ở vị trí thứ 7 đến 12. Giả sử con trỏ không nằm ở dòng 5 khi lệnh này được phát ra, để xoá từ đó thì chương trình soạn thảo sẽ phải tạo ra chuỗi thoát ANSI như thế nào?
35. Nhiều thiết bị RS-232 sử dụng các chuỗi thoát để xoá dòng hiện hành và dịch chuyển tất cả các dòng văn bản lên trên một hàng. Bạn nghĩ thế nào nếu tính năng này được thực hiện bên trong thiết bị?
36. Trên các máy IBM PC hiển thị màu đầu tiên, việc ghi dữ liệu vào Video RAM khi đang đưa tia quét dọc CRT trở về đầu trang sẽ tạo ra các đốm sáng làm xấu màn hình. Một màn hình gồm 25 dòng, mỗi dòng có 80 ký tự, mỗi ký tự rộng 8 pixel và cao 8 pixel. Mỗi hàng có 640 pixel và được vẽ trong một lần quét ngang (mất 63.6 μs), kể cả thời gian quay về đầu dòng. Màn hình được vẽ lại 60 lần trong một giây, mỗi lần lại phải đưa tia quét dọc trở về đầu trang một lần. Thời gian mà Video RAM sẵn sàng để ghi dữ liệu vào sẽ chiếm tỷ lệ bao nhiêu?
37. Các nhà thiết kế hệ thống máy tính mong muốn rằng chuột có thể chuyển động với tốc độ tối đa bằng 20 cm/s. Nếu một đơn vị chuyển động (mickey) bằng 0.1 mm, và mỗi thông

báo của chuột dài 3 byte, thì tốc độ dữ liệu cực đại của chuột bằng bao nhiêu? (giả thiết thông báo được gửi sau mỗi mickey).

38. Các màu cơ bản gồm đỏ (red), xanh lá cây (green), và xanh da trời (blue), nghĩa là có thể tạo ra bất cứ màu nào bằng cách trộn lẫn ba màu đó. Liệu có bức ảnh nào không thể hiển thị được bằng 24-bit màu không?
39. Một cách để đặt ký tự lên màn hình bimap là sử dụng hàm bitblt để sao chép ký tự từ bảng font. Giả sử rằng kích thước các ký tự của font là 16 × 24 pixel trong chế độ màu true RGB color.
    (a) Mỗi ký tự sẽ chiếm bao nhiêu không gian trong bảng font?
    (b) Nếu sao chép một byte dữ liệu tốn mất 100 ns, kể cả các chi phí khác, thì tốc độ dữ liệu gửi tới màn hình sẽ bằng bao nhiêu ký tự/giây?
40. Giả sử cần 10 ns để sao chép một byte dữ liệu, sẽ cần bao nhiêu thời gian để hiển thị lại toàn bộ màn hình trong chế độ văn bản (có 25 dòng, mỗi dòng có 80 ký tự), sử dụng phương pháp ánh xạ màn hình tới bộ nhớ? Nếu là màn hình đồ hoạ 1024 × 768 pixel với 24-bit màu thì sẽ tốn bao nhiêu thời gian?
41. Trên hình 5-41 có sử dụng lớp dữ liệu *RegisterClass*. Trong mã X Window tương đương ở hình 5-46 không có lời gọi nào giống như vậy. Tại sao?
42. Ta đã có một ví dụ về cách vẽ hình chữ nhật trên màn hình bằng Windows GDI:
    ```
    Rectangle(hdc, xleft, ytop, xright, ybottom);
    ```
    Liệu có thực sự cần tới tham số đầu tiên (*hdc*), và nếu có thì nó là cái gì?
43. Một thiết bị SLIM được dùng để hiển thị trang Web có chứa phim hoạt hình với kích thước 400 pixel × 160 pixel, với tốc độ 10 khung hình/giây. Nếu sử dụng mạng Fast Ethernet 100-Mbps thì việc hiển thị phim hoạt hình đó sẽ tiêu tốn hết bao nhiêu phần trăm băng thông của mạng?
44. Nhận xét rằng hệ thống SLIM sẽ hoạt động tốt với mạng có tốc độ 1-Mbps (theo thử nghiệm). Liệu có xảy ra sự cố nếu cho nhiều người cùng sử dụng một lúc? Gợi ý: Xét trường hợp nhiều người dùng cùng xem chương trình truyền hình, và trường hợp họ cùng duyệt web.
45. Nếu điện áp CPU cực đại là *V*, được giảm xuống còn *V/n*, thì công suất của nó sẽ giảm xuống với hệ số $1/n^2$, và tốc độ đồng hồ của nó cũng giảm xuống với hệ số *1/n*. Giả sử người dùng gõ phím với tốc độ 1 ký tự/giây, nhưng thời gian CPU xử lý mỗi ký tự chỉ là 100 ms. Như vậy giá trị tối ưu của *n* bằng bao nhiêu, và sẽ tiết kiệm được bao nhiêu phần trăm năng lượng? Giả sử khi CPU rỗi thì năng lượng tiêu thụ bằng không.
46. Một máy tính xách tay được cài đặt tính năng tiết kiệm điện năng ở mức cao nhất, bao gồm cả tính năng tắt màn hình và đĩa cứng sau một khoảng thời gian không hoạt động. Một người dùng thỉnh thoảng chạy các chương trình UNIX trong chế độ văn bản, và thỉnh thoảng lại sử dụng hệ thống X Window. Cô ta rất ngạc nhiên khi thấy mức năng lượng tiêu thụ ít hơn nhiều so với khi chỉ sử dụng các chương trình ở chế độ văn bản. Tại sao?
47. Viết một chương trình mô phỏng thiết bị lưu trữ tin cậy. Sử dụng hai file lớn có kích thước cố định trên đĩa để mô phỏng cho hai ổ đĩa.