CHƯƠNG 5. QUẢN LÝ VÀO RA

5.0. Mở đầu

Khả năng hiệu suất của một hệ thống máy tính không chỉ phụ thuộc vào kiểu bộ vi xử lý và bề rộng từ (16, 32 hay 64 Bit), đặt biệt còn phụ thuộc một cách thực chất vào tốc độ, mà với nó, các dữ liệu có thể được dịch chuyển giữa các thiết bị vào - ra (kiểu bộ nhớ quảng đại, kiểu kết nối mạng...) và hệ thống bộ nhớ chính - bộ vi xử lý. Ở các ứng dụng khoa học thuần tuý, khả năng tính toán có thể đạt tới hàng triệu phép tính dấu phẩy động. Trong sự khác biệt với điều đó, các yêu cầu của các thiết bị tính toán thông thường thì bao gồm một sự pha trộn muôn màu muôn vẻ các kiểu chương trình khác nhau: các thành phần tính toán, các ứng dụng về ngân hàng dữ liệu, các nhiệm vụ quản lý... Cho nên, tỷ suất các chương trình *(benchmark programms)* áp dụng những nhiệm vụ mà việc sửa chữa lỗi của chúng được xác định một cách mạnh mẽ bởi một hệ thống gồm bộ vi xử lý, bộ nhớ chính, bộ nhớ quảng đại và cấu trúc vận chuyển dữ liệu.

Theo đó, việc đưa vào-ra các dữ liệu đóng vai trò rất quan trọng. Từ đó, chúng ta coi trọng và quan tâm tới cấu trúc vào-ra của các hệ điều hành. Cụ thể, vấn đề này đã được nghiên cứu ở các chương trước; ở chương này, chúng ta sẽ tiếp tục khảo sát điều này một cách kỷ lưỡng hơn.

5.1. Phân loại nhiệm vụ

Trước đây, trong các hệ điều hành, những quan hệ qua lại giữa chương trình người sử dụng và các thiết bị xuất-nhập rất là khăng khít; mỗi người lập trình ứng dụng đã biên soạn cho mình một hệ thống tệp tin hiệu quả để gia tăng dòng dữ liệu giữa các ứng dụng của anh ta và các thiết bị ngoại vi. Tuy nhiên, phương pháp này không chỉ dẫn tới mỗi dòng dữ liệu có một chương trình hướng thiết bị riêng, mà còn dẫn tới các lỗi và các sự lẫn lộn khi nhiều chương trình muốn truy cập lên một thiết bị như nhau. Vì có điều đó xảy ra, nên hệ thống đa người sử dụng làm việc rất ì ạch; để giải quyết vấn đề này, khi đó, các thành phần chương trình kiểu thiết bị được tách chia ra hay được tích hợp lại thành các đơn thể riêng lẻ, gọi là các bộ kích tạo của hệ điều hành *(operating-system-driver)*. Điều này không chỉ yêu cầu di chuyển một chương trình ra khỏi các cấu trúc máy tính khác nhau và trợ giúp để phòng tránh các lỗi, mà đặt biệt còn tiết kiệm sức lực cho người lập trình ứng dụng.

Nhiệm vụ cơ bán của một bộ kích tạo *(driver)* là ở chỗ phải bao quát tất cả các bước khởi xướng của thiết bị và các cơ chế chuyển đổi dữ liệu trước chương trình ứng dụng (sau một giao diện của hệ điều hành). Với các quan niệm đã được đề cập ở chương 1, bộ kích tạo chính là một máy ảo; nó được dùng làm cầu nối trung gian giữa hệ điều hành máy vật lý,

Các nhiệm vụ của bộ kích tạo thì bị giới hạn bởi việc khởi xướng các cấu trúc dữ liệu và các thiết bị như việc viết hay đọc các dữ liệu. Thêm vào đó, còn có những nhiêmh vụ mà chúng chỉ có thể thực hiện cùng với hệ điều hành, đó là:

- Việc chuyển đổi mô hình lập trình logic tới các yêu cầu thiết bị chuyên dụng;
- Việc chỉnh lý các tiến trình đọc hay viết đối với thiết bị;
- Việc phối hợp các thiết bị khác nhau thành các kiểu giống nhau;
- Việc ghi chéo các dữ liệu dự trữ vào buffer...

Người ta có thể tổng hợp các nhiệm vụ bổ sung vào trong một lớp phần mềm, do đó, nói chung, nhiều lớp của các máy ảo hay bộ kích tạo nằm giữa tiến trình người sử dụng và máy vật lý. Hình 5.1 minh hoạ điều này.

Trạng thái NSD	**Tiến trình NSD**
Trạng thái nhân	**Bộ phân bổ nhân**
	Quản lý dãy tuần tự
	Ghi vào bộ đệm
	Bộ kích tạo
	Bộ điều khiển
	Thiết bị

Hình 5.1. Các lớp cơ bản của quản lý thiết bị

Việc giới thiệu các phân lớp cho phép dẫn vào những nhiệm vụ bổ sung cho việc sử lý dữ liệu ở dạng các lớp đặc biệt thành dãy trình tự xử lý. Thí dụ, bộ kích tạo ổ đĩa coi ổ đĩa như là một thiết bị nhớ mà địa chỉ nhớ của nó được xác định nhờ nhiều thông số khác nhau như ổ đĩa, số sector, số đĩa từ... Nó chuyển đổi các yêu cầu đọc /viết thành các địa chỉ logic của bộ nhớ ổ đĩa, các yêu cầu này xuất phát từ một kiểu tuyến tính đơn giản của [0..N] địa chỉ bộ nhớ. Bấy giờ, người ta đặt thêm một bộ kích tạo cho việc chuyển đổi một địa chỉ logic thành một địa chỉ vậ lý. Tức là khi đó, một bộ kích tạo tệp tin sẽ thực hiện việc chuyển đổi một địa chỉ logic tương đối ở trong một tệp tin thành địa chỉ logic tuyệt đối của thiết bị bộ nhớ, mà trên đó tệp tin tồn tại.

5.1.1. Các lớp xử lý I/O ở Unix.

Nguyên tắc phân lớp của nhã hệ điều hành Unix được trình bày như trên hình 1.6 ở trong chương đầu. Ơ đây, với các lớp bổ sungm hệ thống lưu thông của hệ điều hành Unix ấn bản System V có điều kiện để kết hợp các bước xử lý khác nhau thành quá trình xử lý; chức năng này của hệ điều hành gọi là bộ kích tạo *(driver)* của hệ điều hành. Hình 5.2 chỉ ra hệ thống lưu thông tín hiệu ở trong hệ điều hành Unix.

Hình 5.2 trang 184.

Ở hình 5.2(a) cho thấy, khi thiết bị là hướng ký tự , thì bộ kích tạo nhận dạng ký tự đặc biệt bị đẩy vào lộ trình xử lý *(processing route)*; lộ trình này nhận biết các chữ cái đặc biệt; các chữ cái này dịch vụ với tư cách là các lệnh (thí dụ: FEL để xoá chữ cái cuối cùng, Control-C để bẻ gãy tiến trình đang xảy ra...); ngoài ra, lộ trình này còn nhận biết các ký hiệu đặc biệt (thí dụ: 6ký tự trống cho một TAB ký tự...) và tạo ra những hoạt động thích hợp.

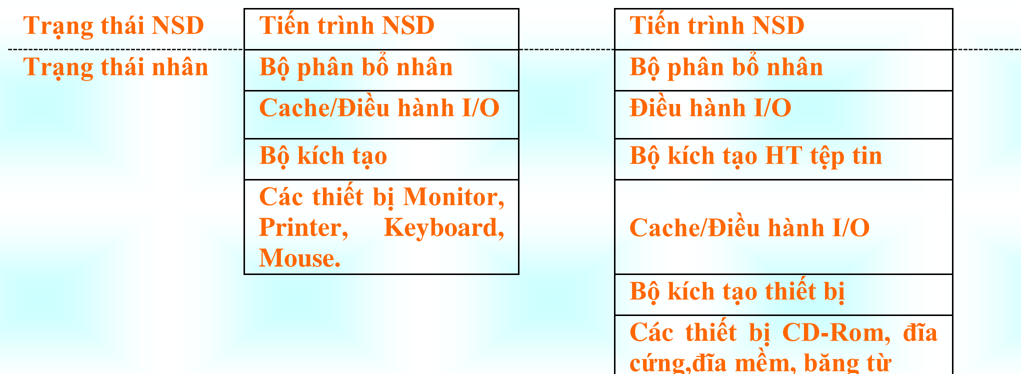
Một giao diện nguyên sơ *(raw interface)* cho phép gởi đi hay đón nhận các ký tự một cách trực tiếp mà không cần xử lý cao hơn dòng ký tự (tức là không cần tới các lệnh ECHO hay Control-C). Lộ trình này còn đặc biệt hữu ích đối với việc kết nối các dữ liệu giữa máy tính với những mục đíchtrao đổi dữ liệu, vì ở đây, tất cả các ký tự được bao hàm như các dữ liệu, mà không phải là các chữ cái hay tín hiệu điều khiển.

5.1.2. Các lớp xử lý I/O của Windows NT.

Việc phân lớp ở trong Windows NT thì năng động và phụ thuộc vào việc dịch vụ hệ thống. Hình 5.3 chỉ ra: bên trái là sự phân lớp đơn giản cho các thiết bị nối tiếp, còn bên phải là đa lớp cho bộ nhớ quảng đại.

Bộ kích tạo hệ thống tệp tin *(file-system-driver):*

Tất cả các gọi hệ thống đối với việc xuất-nhập được bó gói và được chuyển đi tới quản lý I/O để chuyển đổi thành dạng một gói yêu cầu I/O (I/O *reagest package:*IRP). Mỗi một trong các gói chuyển đổi này chứa đựng các địa chỉ người nhận này chứa đựng các địa chỉ người nhận cũng như không gian dữ liệu *(data-space);*cho nên, thông tin trong một lớp dịch chuyển từ trên xuống dưới tới tiến trình người sử dụng và thông báo kết quả trở lại. Vì thế, nhiều trạng thái và nhiều lớp sẽ đi qua. Ngoài ra, sự tồn tại của bộ kích tạo thật là năng động: Trong sự khác biệt cới hệ điều hành Unix, ở đây, trong khi điều hành, người ta có thể đưa vào hay lấy đi một bộ kích tạo theo ý muốn để điền đầy những nhiêmh vụ xác định.



Hình 5.3. Sự phân lớp đơn giản và đa cấp ở hệ điều hành Windows NT

Bộ kích tạo với ảnh xạ các cluster hư hỏng.

Vì việc quản lý tệp tin vẫn do tự bộ kích tạo điều khiển, do đó, trong hệ thống Windows NT nhiều hệ thống tệp tin có thể tồn tại song song với nhiều bộ kích tạo. Thật vật, người ta thấy rằng, đối với mỗi hệ thống các tệp tin cho các hệ điều hành khác nhau (Windows NT, OS/2 hay MS-DOS) đều có một bộ kích tạo riêng. Tất cả các tác vụ truy cập đối với các hệ thống tệp tin thì được thực thi như những phương pháp của một bộ kích tạo hệ thống tệp tin. Chúng phân biệt không chỉ ở các cấu trúc dữ liệu và hiệu suất dịch vụ (nghĩa là chúng tạo ra những cái đó với sự trợ giúp của bộ kích tạo thiết bị trên bộ nhớ quảng đại), mà còn ở sự phản ứng trước các lỗi xuất hiện. nếu thí dụ ở trong bảng FAT của MS-DOS hay trong hệ thống tệp tin của OS/2 xuất hiện các lỗi không thể sửa được vì các đơn vị bộ nhớ sai hỏng (tức là các *block* hay các *cluster* bị sai hỏng), thì do đó, khi có lỗi, Bit hư hỏng *(corrupted-bit)* được thiết lập. Muốn sử dụng các tệp tin trên clucter hư hỏng này, người ta phải tiến hành kiểm tra sửa chữa bằng một chương trình trợ giúp *(check disk)* được gọi với lệnh chkdsk: với chương trình này, các tệp tin nói ở trên được điền vào hệ thống tệp tin bộ nhứo quảng đại, cụ thể các đơn vị bộ nhớ phù hợp điền vào bảng *bad clucter mapping*, và do đó, *cluster* hư hỏng bị lloại ra khỏi các ứng dụng tiếp theo.

Ngược lại, ở hệ thống tệp tin Windows, điều này được thực hiện một cách năng động. Thật vậy, việc sắp xếp các cluster ảo thành các cluster logic được điều chỉnh thích hợp; các loại cluter này cũng đã được nói tới trong mục 4.5 ở trên. Bây giờ, chúng ta khảo sát một thí dụ ở hình 5.4, ở đây một cluter đã bị khuyết tật,nó không có thể được bộ kích tạo thiết bị kiến lập thành những đơn vị lưu trữ *(reserved-unit)*.

hình 5.4 trang 186

Cuối cùng, nếu không còn phần dư thừa của bộ nhớ quảng đại và cũng không thể có bản sao bảo vệ tồn tại, khi đó, một thông báo lỗi "Read/Write Error" được đưa ra trên màn hình.

Đối với hệ thống tệp tin Windows NT, nếu không có cluster còn trống có thể sử dụng được, do đó, một nguyên tắc, Bit bị hỏng của đĩa từ sẽ được kiến lập; và khi thực hiện, một chương trình kiểm tra sửa chữa (chkdsk) được tiến hành một cách tự động.

Nếu các dữ liệu của cluster hỏng không thể phục hồi được và cũng không còn cluster thay thế, do đó, ngoài ra, tại mỗi lần vào-ra trên cluster này, một thông báo lỗi "Warning Eorror" được đưa tới người điều hành hay người sử dụng, bấy giờ, vì không còn dữ liệu khả dụng nên độ lệch lỗi không tồn tại và nội dung của cluster hư hỏng phải được khai khẩn khi đọc từ bản sao.

5.2. Các mô hình thiết bị

Sự phát triển của bộ kích tạo *(driver)* đối với một hệ điều hành được quyết định bởi tính chất của các thiết bị vật lý, tức là chúng phải tạo lập nên các tính chất mà hệ điều hành mong đợi. Điều đó rất cần thiết, nó làm cho người lập trình hiểu nhiều hơn về thiết bị. Vì rất nhiều thiết bị vó đặc điểm giống nhau, do đó, nó có lợi cho việc nghiên cứu các kiểu dáng để hiểu sâu sắc các thông số đặc trưng.

Thật vậy, chúng ta có thể phân biệt một cách thô thiển giữa hai loại thiết bị: các thiết bị sử dụng các thông tin định vị với việc truy cập tuỳ chọn và thiết bị với sự chuyển đổi dữ liệu nối tiếp mà không cần thông tin địa chỉ.

Bây giờ, chúng ta nghiên cứu bộ nhớ ổ đĩa từ là đại diện cho nhóm thiết bị tuỳ chọn.

5.2.1. Bộ nhớ ổ đĩa từ

Bộ nhớ đĩa từ bao gồm một đĩa bằng nhôm được phủ một lớp mỏng bột ôxyt sắt. Trên một thanh kim loại mảnh có gắn một đầu từ để đọc/viết, và đĩa được quay tròn quanh một trục. Đầu từ có thể làm nhiễm bột từ trên các vòng hình xuyến *(track)* của đĩa, khi đó gọi là quá trình viết. Ngược lại, khi trượt qua đầu tự một cách khác nhau, lớp bột từ làm kích thích cảm ứng ở trong cuộn dây đầu từ, khi đó gọi là quá trình đọc. Khi đọc, các thông tin được định hình bằng từ tính, mỗi sự thay đổi cảm biến được thông dịch thành các Bit, do đó, trên mỗi vòng xuyến của đĩa từ, một dãy tuần tự các Bit được lưu trữ. Vì một vòng xuyến có thể lưu trữ rất nhiều thông tin, vì vật, người ta chia nhỏ vòng xuyến thành nhiều phần có độ lớn khác nhau, mỗi phần gọi là một sector.

Bây giờ, người ta còn điều khiển thanh kim loại có gắn đầu từ di chuyển trên đĩa từ (đĩa cứng) từ tâm ra mép biên hay ngược lại: để đọc và viết được trên nhiều vòng xuyến khác nhau.

Mô hình cơ bản của một bộ nhớ đĩa từ thì bao giừo cũng giống nhau. Ngoài việc phủ trên bề mặt đĩa một lớp bột ôxyt sắt (đĩa cứng), người ta còn phủ một lớp bột ôxyt crôm (đĩa CD-ROM) hay tráng một lớp môi trường quang hoạt tính (đĩa ĐV), và người ta còn chế tạo đĩa từ bằng chất dẻo có phủ bột từ (đĩa mềm).

Tất nhiên, người ta có thể sắp xếp nhiều đĩa từ với nhau từ trên xuống dưới với mỗi mặt đĩa có bố trí một đầu từ đọc/viết *(read/write head).*

hình 5.6 trang187

Nếu nhiều đĩa từ được lắp chồng lên nhau cùng theo một trục quay, do đó, bằng kết cấu cơ khí, các đầu từ cũng được chuyển động theo mong muốn. Ở trên các đĩa từ khác nhau trong hệ , tất cả các đầu từ luôn luôn cùng ở trên một vòng xuyến có số nhóm giống nhau thì ở trên nhau và tạo thành một vỏ hình trụ ảnh ảo (xem hình 5.6). Các vòng xuyến có số điều khiển giống nhau được biểu thị một nhóm hình trụ *(cylinder group).*

Ở phần cấu tạo, một đầu từ phải được dịch chuyển một cách cơ học để đọc hết trọn một vòng xuyến. Nếu người ta muốn đặt chính xác ở giữa hình xuyến để nhận biết điểm bắt đầu của vòng xuyến và các khoảng chia của vòng xuyến, do đó, ngoài ra còn có những thông tin trợ giúp như các kích thích đều đặn hay các thông tin hiệu chuẩn đặt biệt thì cũng rất cần thiết. Đối với người lập trình, các thông số sau đây của mô hình cấu tạo đĩa từ là rất quan trọng:

✧ Thời gian truy cập trung bình t_s còn gọi là thời gian tòm kiếm trung bình *(averrage seek time)* cho một vòng xuyến: khoảng 8 |10ms cho ổ đĩa cứng, khoảng 100ms cho ổ đĩa CD-ROM, khoảng 250ms cho ổ đĩa mềm...

✧ Khi tốc độ quay là cố định, người ta phải chờ đợi trên vòng xuyến một khoảng thời gian xác định t_D, gọi là thời gian quay trễ *(rotational delay time)*, cho tới khi đạt được một sector mong muốn xuất hiện dưới đầu từ. Trong trường hợp xấu nhất, đó là thời gian trễ cho một vòng quay t_R.

✧ Tốc độ vận chuyển lớn nhất còn gọi là tỷ phần vận chuyển dữ liệu H, nó phụ thuộc vào mật độ từ tính của đĩa từ trên một đơn vị chiều dài của vòng xuyến (thứ nguyên của mật độ từ tính: Bit/mm).

Do vậy, các bộ nhớ đĩa từ gia tăng dung lượng lưu trữ một cách tuyến tính khi thời gian truy cập như nhau. Ngược lại, khi các vòng xuyến gia tăng dung lượng, người ta nhận thấy: để truy cập một dung lượng như nhau, với vòng xuyến có bán kính nhỏ thì đầu từ truy cập một thời gian ít hơn khi truy cập trên vòng xuyến có bán kính lớn hơn. Trường hợp đặc biệt, khi trên mỗi vòng xuyến có một đầu từ, khi đó không có chuyển động đầu từ xảy ra nữa *(fixed head disk).*

Mô hình nói trên của việc truy cập đĩa cứng đòi hỏi phải đáp ứng những nhu cầu thiết yếu sau đây: nhu cầu về thời gian truy cập trung bình t_s là thời gian để đầu từ chuyển động tới đúng vòng xuyến *(track)* cần tìm; nhu cầu về thời gian quay trễ t_D là thời gian để đầu từ tìm thấy đúng sector cần tìm; và nhu cầu về thời gian t_T để vận chuyển dữ liệu *(data tranfer time)*. Việc làm trễ quá trình truy cập trên đĩa từ có 3 nguyên nhân chủ yếu:

- Do chất lượng cơ cấu cơ khí của đầu từ;
- Do tốc độ quay của đĩa từ và chiều dài (đường kính) khác nhau của track;
- Do dung lượng mật độ thông tin trên một track (thí dụ mỗi track cùng được nạp một dung lượng m Byte).

Nếu gọi k là dung lượng của dữ liệu cần truy cập và giả sử thời gian quay trễ để tìn đúng sector t_s bằng một nửa thời gian quay trọn một vòng xuyến t_R:

$$T_D = t_R/2 \qquad (5.1)$$

Khi đó thời gian truy cập tổng cộng T được thiết lập bằng biểu thức:

$$T = t_S + t_R/2 + (k/m).t_R \qquad (5.2)$$

Từ biểu thức (5.2), ta nhận thấy T biến thiên tuyến tính theo k. Khi đó tỷ phần vận chuyển dữ liệu H được xác định:

$$H = k/t_T \qquad (5.3)$$

Với biểu thức (5.3), ta có nhận xét: H và t_T đều là những đại lượng phụ thuộc vào k; tuỳ theo cấu hình của hệ thống mà t_T và k sẽ có một quan hệ nào đó; do đó, H là một hàm phi tuyến, nó không biến thiên tỷ lệ thuận với k. Để giải quyết vấn đề này của hệ điều hành, một đơn vị bộ nhớ (có thể có độ lớn 1 *sector,* 1 *block* hay 1 *page*) được trợ giúp để chuyển dịch tới bước hoạt động nhân tử, mà điều đó được yêu cầu do sự tối ưu hoá khả năng sử dụng bộ nhớ chính qua các bảng trang (xem mục 3.3.6). Đáng tiếc, độ lớn tệp tin trung bình nằm ở giá trị 1kByte. Do đó việc sử dụng dung lượng bộ nhớ bị giảm xuống một cách mạnh mẽ, khi nếu độ lớn trang thay đổi quá độ lớn 1kByte của tệp tin; vì thế, người ta phải từ bỏ việc gia tăng tốc độ chuyển vận dữ liệu qua việc gia tăng độ lớn trang.

Ngoài ra còn tồn tại một vấn đề mữa về mô hình đĩa từ, đó là: tất cả các track (vòng xuyến) của đĩa từ có chiều dài khác nhau. Nếu bây giờ, mật độ từ tính và số Bit trên một đơn vị mm^2 của bề mặt đĩa từ là như nhau khắp nơi, do đó, mỗi track có một số lượng Bit khác nhau: số Bit của track có bán kính nhỏ thì ít hơn số Bit của track có bán kính lớn. Nhưng vì thời gian quay một vòng của các track có bán kính khác nhau là như nhau. Cho nên, tại mỗi track, chúng ta có một tỷ phần dữ liệu khác nhau *(Bist/sec)*. Đối với nghành công nghệ thông tin, điều này được mô phỏng thêm một vấn đề: vì nhịp Bit đối với mỗi track cũng như đối với phần mềm của bộ kích tạo *(driver)* phải được định nghĩa một cách khác thường, bởi lẽ, số lượng các đơn vị bộ nhớ *(sector)* trên mỗi track là khác nhau.

Đề phòng tránh các tiêu chí phần mềm và phần cứng, tất cả các track của đĩa từ được phân chia thành số lượng các sector như nhau. Do đó, các track ở gần trung tâm quay nhất có thể dược mô tả chỉ tới một giưoi hạn dung lượng của chúng. Sự không đều đặn của cách phân lớp này tồn tại sẽ dẫn đến lỗi. Do đó các track ở trong cùng (gần tâm quay) thường ít được sử dụng (thí dụ trên thực tế đối với mỗi đĩa mềm chỉ sử dụng 80 track, đáng lẽ 82); thêm vào đó, đa số các track có một sự bảo vệ biến động cao.

Tối ưu việc truy cập ổ đĩa cứng ở các Unix

Để tối ưu khả năng của hệ thống tệp tin, người ta có thể thực hiện việc sử dụng các biện pháp: tại mọt cơ cấu định vị kết hợp, tất cả các đầu từ sẽ truy cập nhanh trên track hay trên các đơn vị nhỏ block của các track của một trụ ảo *(virtual cylinder)*, nếu các đầu từ đồng thời ở trên cylinder này. Ở hệ điều hành Unix, mỗi nhóm track (các vòng xuyến cùng bán kính) của một cylinder chứa đựng một nút đặt biệt (của cây B) và một sự quản lý đặt biệt. Cho nên, nếu nút chỉ số mới *(new index node)* và các block cấp phát tệp tin, do đó, điều này được xảy ra một cách tự động trên từng nhóm cylinder. Qua đó, việc đọc/viết các tệp tin được tăng tốc rất nhanh, vì sau khi truy cập trên một nuýt chỉ số *(i-node)*, tất cả các tác vụ tiếp theo sẽ không cần dùng thời gian nữa để định vị đầu từ đọc/viết.

Giao diện thiết bị (device interface):

Khác với trước đây, ở cấu trúc máy tính định hướng bus hệ thống thì không có các kênh phần cứng đặc biệt và do đó, không có các lênh được dự kiến trước của bộ vi xử lý. Đặc biệt, tất cả các thiết bị có thể được làm đáp ứng một cách hệ thống nhất như bộ nhớ dưới mộtđịa chỉ ở trong không gian của bộ nhớ chính *(memory mapped* I/O*)*.

Hình 5.7 trang 190

Ở trong máy tính, các khoảng nhớ bên trong (của thanh ghi, bộ đệm...) được một bộ điều khiển thiết bị tạo lập để điều khiển thiết bị trên khoảng địa chỉ của bộ nhớ chính. Sau đây, những cái đó được lần lượt giới thiệu.

- *Thanh ghi điều khiển (control-register):*

Nội dung lưu trữ của một tệp tin (*data-word)* được bộ điều khiển đọc và thực hiện. nó phục vụ như một thanh ghi trạng thái; mỗi Bit có một ý nghĩa đặc biệt. Thí dụ: Bit 4=1→ kết thúc việc đọc; Bit 8=1→ xuất hiện lỗi đọc

- Thanh ghi lệnh *(command-register*):

Ở loại thanh ghi này, mã được viết cho một lệnh (đọc/viết; tạo dạng/định vị...), thì lệnh này cần thiết phải được thực hiện. việc viết vào một mã lệnh ở trong thanh ghi này được thông dịch như là việc gọi thủ tục; khi đó, các thông số trên thanh này cũng bao gồm cả nội dung của các thanh ghi khác.

- Thanh ghi địa chỉ *(address-register*):

Thanh ghi địa chỉ chứa đựng địa chỉ trên thiết bị (bao hàm: *device, track, track-group, ector, disk, read/write-head)* và số dung lượng vận chuyển (Byte).

- Bộ đệm dữ liệu (*data-buffer)*:

Hầu hết các bộ đệm xuất-nhập được tách chia để sao cho mỗi một bộ đệm chỉ được đọc và một bộ đệm khác chỉ được viết.

- Hệ thống ngắt *(interupt-system)*: Sau khi thực hiện một lệnh, ngắt hệ thống có thể phát động một ngắt khác...

Nếu thiết bị chiếm rất nhiều thông số, mà người ta có thể điều chỉnh chúng, do đó, người ta dùng nhiều thanh ghi, và khi đó sẽ xảy ra trường hợp nhiều không gian địa chỉ bị loại bỏ. Ở các không gian địa chỉ nhỏ, thông thường chỉ có một địa chỉ tồn tại, khi đó, người ta phải viết địa chỉ tới gói tin nhiệm vụ hay các gói tin dữ liệu tuần tự với kích cỡ đã được khẳng định chính xác.

Một cơ chế quan trọng đối với việc trao đổi thông tin với thiết bị là việc di dịch toàn bộ khối dữ liệu giữa bộ đệm thiết bị của bộ điều khiển với bộ nhớ chính. Điều này được thực hiện một cách phụ thuộc vào bộ vi xử lý (với các cíp vĩ mạch

chuyên dụng); khi đó, người ta gọi trường hợp này là truy cập trực tiếp *(direct-memory access:* DMA). Các nhịp DMA tồn tại không chỉ trong hệ thống bộ nhớ, đặc biệt còn tồn tại trên card điều khiển *(Control-card).* Vì vậy, chúng có thể làm việc một cách độc lập với nhau nhờ có các kênh DMA.

5.2.2. Bộ nhớ đa ổ đĩa từ

Một phương pháp được mở rộng để nâng cao dung lượng đĩa từ là ở chỗ quản lý kết hợp nhiều đĩa từ dung lượng nhỏ thành một ổ đĩa ảo có dung lượng lớn. Thật vậy, đối với một đĩa từ, chúng ta có một ổ đĩacngs chỉ với một dung lượng hạn hẹp, do đó, nhiệm vụ của chúng là biến ổ đĩa có dung lượng nhỏ này thành ổ đĩa có giá trị. Một hệ thống để giải quyết vấn đề này được biểu thị là RAID *(redundal array of inexpensive disk).* Hệ thống này được ưu chuộng một cách tuyệt đối với các ngân hàng dữ liệu lớn và với nhiều mục đích khác.

Đối với hệ điều hành, sự biến thể phần cứng có ý nghĩa: Bộ kích tạo của hệ điều hành chỉ cho chúng ta thấy, người ta có thể tạo nên một ổ đĩa ảo năng động duy nhất và đem lại một dung lượng lớn hơn. Khi ố. Đối với một giao diện để dẫn tới các lớp cao hơn của hệ điều hành, thì việc thực hiện tập hợp các đĩa từ là chẳng có vấn đề gì.

Một lợi thế tiếp theo của hệ thống đa bộ nhớ đĩa cứng có thể được sử dụng mạnh mẽ hơn, nếu ở trong hệ điều hành tồn tại nhiều tiến trình độc lập hay các tiến trình threads (xem chương 2). Sự độc lập của chúng, và với cái đó, sự loại bỏ lẫn nhau nhanh chóng được phòng tránh, nếu các dữ liệu độc lập nhau được thu gom trên một bộ nhớ kết hợp. Nếu bây giờ chúng ta thay thế nhiều ổ đĩa cứng bởi một ổ đĩa duy nhất với các đầu từ được nối cứng (xem hình 5.6), do đó, các dữ liệu của các tiến trình độc lập có thể được đặt trên các bộ nhớ đĩa từ độc lập, và như vậy, việc truy cập dữ liệu xảy ra nhanh hơn đáng kể. Ở hình 5.8, một sự phân chia các dữ liệu như vậy thì được nhìn thấy trên hệ thống đa ổ đĩa từ. Do đó, không gian tổng của bộ nhớ logic được phân chia thành các khoảng; một cách tương ứng, các khoảng lớn bằng nhau của ổ đĩa từ khác nhau được bộ kích tạo coi là khoảng bộ nhớ liên kết và lập thành một khoảng bộ nhớ thống nhất cho một nhóm tiến trình. Bộ kích tạo của hệ điều hành có nhiệm vụ: phải quyết định việc phân bổ và kết hợp các dữ liệu và phải thực hiện việc tạo lập các đơn vị bộ nhớ ảo trên các đơn vị bộ nhớ logic của các ổ đĩa từ khác nhau.

Tổ chức ổ đĩa từ ở trong Windows NT:

Ở hệ điều hành Windows NT, khi tạo khuôn dạng lần đầu cho các khoảng bộ nhớ logic, một sự lựa chọn được cho trước để tổ chức thành các dải băng.

Một sự lựa chọn tương tự hay khác nhau là ở chỗ, phải liên kết nhiều đoạn có độ lớn bất kỳ của các đĩa từ khác nhau tới một dung lượng của ổ đĩa logic. Không gian địa chỉ của ổ đĩa ảo có thể được phân bổ không đều đặn trên các đĩa từ khác nhau.

hình 5.8 trang 192

Nếu chúng ta làm cho những dải băng rất mỏng, thí dụ khoảng 4 Byte hay tạo thành một klock lớn hơn, do đó, bộ nhớ đệm dữ liệu này được phân chia thành những khoảng tuần tự có chiều dài 4 Byte và cho nên các dữ liệu được đọc và viết rất nhanh trên ổ đĩa từ. Một hệ thống có cấu hình truy cập nhanh nhưng không có tổn thất như vừa nêu được biểu thị là hệ thống RAID-0.

Bên cạnh khả năng truy cập nhanh rất cao và dung lượng bộ nhớ lớn, các hệ thống của đa ổ đĩa từ còn có một ưu điểm quan trọng nữa là: với hệ thống này, người ta có thể loại bỏ các tổn hao của đĩa từ. Đối với kiểu sai số lỗi hay sai số tổn hao, người ta nhận thấy với các dữ liệu của một ổ đĩa từ có một bản sao *(copy)* chính xác trên một đĩa từ của một ổ đĩa khác; còn đĩa từ khác thì được tạo thành ảnh xạ của một trong các đĩa từ trong quan hệ với các dữ liệu của chúng. Hình 5.9 chỉ ra một sơ đồ của một cấu hình ảnh xạ đĩa từ.

Tuy nhiên, bản copy này phải luôn thường xuyên thực hiện. bộ kích tạo có thể dịch vụ theo một cấu hình như vậy; tại mỗi quá trình viết lên đĩa Disk1, nó phải tác dụng bản copy trên đĩa Disk2; tức là, có một cái gì đó đã tạo ra khả năng chịu tải của hệ điều hành và của phần cứng I/O. Trong trường hợp hệ thống đa ổ đĩa từ có tổn thất thì cấu hình của ảnh xạ ổ đĩa từ được gọi là DAID-1. Nếu chúng ta thực hiện ý tưởng của ảnh xạ ổ đĩa từ ở trong cấu hình dãi băng *(stripes)* như ở hình 5.8, do đó, chúng ta sẽ nhận được một hệ thống nhanh tuy có tổn hao, gọi là RAID-0/1. Tóm lại, người ta phải tiến hành thực nghiệm để điều hoà sự tổn hao thời gian tại quá trình đọc của ảnh xạ đĩa từ.

hình 5.9 trang 193

Bộ kích tạo sử dụng trạng thái tồn tại chốc lát của các đĩa từ để nó có thể lựa chọn việc đọc lần lượt 2 đĩa từ, mà các đầu đọc của các track mong muốn ở cạnh đó; còn đầu từ kia có thể sẵn sàng được định vị cho lần đọc tiếp theo hay có thể đọc/viết ở các cung đoạn khác chưa được ảnh xạ.

Nhờ sự khéo trí này, người ta đã gia tăng một cách đặc sắc nhu cầu không gian nhớ lớn đối với hệ thống RAID-0/1. Thật vậy, mỗi Bit của một từ dữ liệu được viết lên một đĩa tà; đối với một từ 32 Bit, người ta cần dùng tới 32 ổ đĩa từ. Sau đó, mỗi từ được dự kiến với một ký hiệu chuyên dụng, thí dụ: một phần bổ sung 6 Bit; do đó, ở hiệu suất cuối cùng của một từ dữ liệu được bộ điều khiển phân bổ trên 38 ổ đĩa từ. Điều đó không chỉ có ý nghĩa đối với dòng dữ liệu đồ sộ, mà còn tạo khả năng có một độ chính xác kết quả cao. Vì mỗi một liên hiêpk Bit của một từ dữ liệu 38 Bit là không thể có khả năng; cho nê, khi các đĩa từ có hư hỏng hay khiếm khuyết, người ta chỉ chốt lại mã hợp lý 32 Bit.

Sữa chữa lỗi nhờ hình thành tính chẵn lẻ:

Ở đây yêu cầu, chúng ta muốn lưu trữ n Bit một cách độc lập với nhau. Cách tiến hành như sau: Khi có thông tin điều khiển, chúng ta tạo lập được Bit thứ n+1, Bit này có tính chẵn lẻ p. Vì thế, người ta gọi p là hàm khác dấu (hàm XOR hay hàm logic cộng modul 2) của hai biến số a và b. Hàm này được biểu diễn bởi biểuthcs:

$$p = a \oplus b \quad (5.4)$$

Theo đại số Boole, hàm này có bảng chân lý:

a	b	p=a⊕ b
0	0	0
0	1	1
1	0	1
1	1	0

Đồng thời hàm XOR có các tính chất:

$$a \oplus 0 = a$$
$$a \oplus a = 0 \quad (5.5)$$

Từ bảng chân lý và các tính chất (5.5), ta suy ra:

$$p \oplus b = a \oplus b \oplus b = a \oplus 0$$

Hay $p \oplus b = a$ (5.6)

Với biểu thức (5.6), bây giờ ta hiểu rằng, khi tính chẵn lẻ p đã biêt, thì kết quả của một Bit a có thể được cho qua, vì chúng ta có thể phục hồi a trực tiếp từ p và b. Tất nhiên, điều đó sẽ dẫn tới, nếu biến b tự tồn tại trở lại từ nhiều số hạng, chẳng hạn:

$$b = b_1 \oplus b_2 \oplus ... \oplus b_n \quad (5.7)$$

Từ biểu thức (5.7) này chúng ta có nhận xét: tại n biến nhị phân, mà từ đó chúng ta thiết lập tính chẵn lẻ của hệ thống, thì ta thấy một trong các biến với sự trợ giúp của tính chẵn lẻ giữa các biến được khai khẩn và phục hồi trở lại. Nếu bây giờ, chúng ta chỉ bình luận 8 biến nhị phân (chúng được kết hợp với nhau trong một Byte) song song nhau, do đó, điều sẽ dẫn đến đối với mỗi Bit ở Byte này hay đối với các Byte khác: Nếu chúng ta nạp mỗi Bit (hay vả dãy Bit) trên một bộ nhớ khác, do đó, khi bộ nhớ hư hỏng, chúng ta có thể phục hồi trực tiếp Bit lỗi từ n Bit chưa thay đổi. Khi đó, nếu chúng ta tạo ra được nhiều Bit sữa chữa lỗi, do đó, chúng ta sẽ khắc phục được khiếm khuyết của nhiều ổ đĩa từ.

Hệ thống sữa chữa lỗi RAID-2 được giới thiệu trong hình 5.10 ở dưới. Các từ dữ liệu (data Word) được biểu thị thành các cột; mỗi cột lại được chia ra các Bit. O trong bộ điều khiển, dòng Bit của mội đĩa từ lại được chia nhỏ thành từng Byte, do vậy, các đĩa từ thương mại thông thường có thể trở thành các hệ thống bộ nhớ.

hình 5.10 trang 194

Nếu chúng ta liên kết các Bit đã sữa chữa lỗi thành một khoảng ngắn và nạp chúng vào một đĩa từ, khi đó, chúng ta nhận được một hệ thống sữa chữa lỗi đĩa từ RAID-3. Vì ở đây, thông tin sai số lỗi đãi tà của tất cả các block thì được nạp trên một ổ đĩa từ duy nhất, do đó, ở đây chỉ có một đĩa từ duy nhất này không tính tới. Bất giờ nếu chúng ta gộp các Bit dữ liệu thành các khoảng riêng lẻ (thí dụ thành các *block*), và như ở hệ thống RAID-0, chúng ta nạp mỗi khoảng này lên một đĩa từ riêng biệt; khi đó, hệ thống sữa chữa lỗi này được biểu thị là RAID-4.

Sự quyết định cho cái đó là ở chỗ, phải bảo đảm các thông tin sai số lỗi (*fault information)* thường xuyên được đọc đối với mỗi block bộ nhớ ở trên một ổ đĩa từ khác, và do đó, phải phân bổ một cách cân xứng sức chịu tải I/O của hệ thống ở tất cả các đĩa từ. Khi đó, người ta nhận được hệ thống sữa lỗi RAID-5. Hình 5.11 chỉ ra sơ đồ của hệ thống này.

hình 5.11 trang 195

Đối với kiến trúc hệ điều hành, nhiệm vụ chủ yếu là, phải vận chuyển toàn bộ kể cả các diễn biến lỗi một cách điều hoà vào trong các hệ thống con RAID, và phải làm cho hệ điều hành độc lập với hệ thống bộ nhớ. Đối với hệ điều hành, chỉ một hệ thống bộ nhớ ảo và đơn giản mới là một hệ thống an toàn và tiện lợi.

Hạn chế lỡi nhờ ở đĩa ánh xạ và hệ thống RAID ở trong Windows NT

Ơ trong hệ điều hành Windows NT, đối với việc quản lý một tệp tin có sai số lỗi sẽ có một bộ kích tạo đặc biệt *(special driver)*, gọi là bộ kích tạo FtDisk. Bộ này được chuyển đỏi giữa hai bộ: bộ kích tạo hệ thống NT và bộ kích tạo thiết bih; bộ kích tạo FtDisk tạo ra một máy ảo như là một lớp trung gian *(middle class)*; máy ảo vừa tạo cần phải tự mình đón bắt tất cả ác lỗi xuất hiện, mà các lớp cao hơn không thể nhìn thấy và chưa nhận biết được. Nếu bây giờ có một lỗi xuất hiện (chẳng hạn *bad sector*), do đó, các dữ liệu block chứa đựng lỗi sẽ được ổ đĩa ánh xạ đọc hay được phục vụ với sự trợ giúp của thông tin sữa lỗi (dải chẵ lẻ). Sau đó, một sector mới được yêu cầu cho sector chứa đựng lỗi, gọi là ảnh xạ sector lỗi*(bad sector mapping)* và các dữ liệu được viết lên sector này. Với cái đó, trong hệ thống lại tồn tại hiện tượng dư thừa dữ liệu. Nếu không còn các sector trống *(free space sectors*) hay nếu thiết bị không có sector ảnh xạ, do đó, block (có các dữ liệu được phục hồi) được đưa trở lại bộ kích tạo hệ thống tệp tin.

Nếu việc tổ chức ổ đĩa sai số lỗi không được tạo lập, do đó, bộ kích tạo FtDisk thông báo lỗi, chẳng hạn lỗi đọc/viết nhằm chỉ cho hệ thống tệp tin hay người sử dụng có phản ứng lại điều đó.

5.2.3. Tạo các ổ đĩa RAM *(RAM-Disk)*

Ở trong hệ điều hành, đối với các tính chất của bộ đĩa cứng, chỉ có bộ kích tạo mới có đầy đủ chức trách để tạo ra một máy ảo. Cho nên, người ta cũng có thể sử

dụng một khoảng bộ nhớ chính RAM như là một ổ đĩa ảo ở vị trí của một bộ nhớ thực đĩa cứng, mà không cần lưu ý điều đó ở trong hệ điều hành. Cách làm này được gọi là phương pháp tạo ổ đĩa hay phương pháp tạo ổ đĩa RAM.

Tất nhiên, việc truy cập trên ổ đĩa RAM như vậy rất nhanh vì đối với các tệp tin trên ổ đĩa này thì tkhông có thời gian trễ cơ học khi đọc/viết. Trước khi tắt máy tính, người ta cần phải làm sáng tỏ một phần rất nhỏ của bộ nhớ chính như là một thiết bị bộ nhớ đĩa cứng(?). Đó là vì: Các chiến lược hiệu nghiệm đối với bộ nhớ Cache và việc phân trang mà trước đây đã được minh hoạ bằng nhiều hình vẽ về các khoảng tệp tin trên không gian địa chỉ ảo, đã làm giảm đi đáng kể thời gian truy cập và làm cho việc quản lý bộ nhớ chính năng động hơn. Vì thế, câu trả lời cho câu hỏi trên phụ thuộc mạnh mẽ vào hệ thống được sử dụng.

✧ Việc tạo một ổ đĩa RAM với mục đích: nếu người ta phải làm việc với một phần mềm cho trước xác định, thì phì mềm này tạo ra những tệp tin trên ổ đĩa này, mà chúng cũng được sử dụng bình thường như trên các ổ đĩa khác. Một ví dụ về điều đó là trình biên dịch *(compiler):* Nếu các tệp tin trung gian của quá trình biên dịch nằm trên ổ đĩa RAM, do đó, nâng cao một cách đáng kể tốc độ biên dịch.

✧ Một tình huống khác cũng thuộc các hệ thống này, khi hệ điều hành không thể phù hợp các khoảng địa chỉ lớn, qua bộ nhớ chính, thì nó phải có dung lượng m lớn hơn giới hạn bắt buộc 640kByte, do đó, người ta có thể sử dụng trình biên dịch qua một ổ đĩa RAM. Ở bộ kích tạo ổ đĩa RAM, người ta có thể vượt qua rào cản nhờ thủ thuật định vị tuần tự thích hợp.

5.2.4. Các thiết bị nối tiếp.

Chúng ta nhớ lại trong chương 4, ở hầu hết các thiêt bị, dữ liệu có thể truy cập một cách tuần tự hay tuỳ chọn; nhưng tới khi ở trên thiết bị (như bàn phím,máy in...) thì điều đó không còn có thể bình thường được nữa. Mối liên hệ giữa *các thiết bị nối tiếp* được biểu thị là một phương pháp để chỉ các dữ liệu phải và sẽ được đưa chuyển như thế nào(?): Đó là phương pháp vận chuyển lần lượt từng ký tự mà không cần thông tin địa chỉ của thiết bị.

Tất nhiên, phương pháp này xảy ra với tốc độ hạn chế (khoảng vài *kByte/sec*). Đó là do các điều kiện biên kinh điển: do thiết bị đầu cuối, do máy in chạy chậm, do tốc độ gỏ bàn phím... Tuy nhiên, các quan hệ về mối *liên kết nối tiếp* vẫn mang lại kết quả rất nhanh; các kiểu này không cần dịch vụ của bộ kích tạo; ở đây, các khối dữ liệu được chuyển vận nhờ việc truy cập trực tiếp bộ nhớ. Việc điều khiển mối liên hệ này được thực hiện với sự trợk giúp của các kỹ thuật viên bằng cách pha trộn hợp lý đối với bộ kích tạo các thiết bị định hưởng khối và định hướng ký tự.

Giao diện thiết bị:

Giao diện để dẫn tới bộ điều khiển các thiết bị nối tiếp (cũng giống như các thiết bị tuỳ chọn) ở trong khoảng các địa chỉ bộ nhớ chỉ chính sách thể được thích hợp qua những điểm sau đây:

✧ Thanh ghi điều khiển quan tâm tới các thông báo trạng thái, tốc độ vận chuyển (thí dụ 1200,2400,4800,9600 Baud= 1 Bit/sec của vận chuyển dữ liệu và điều khiển) và cả phương pháp vận chuyển đồng bộ hay không đồng bộ. Khi vận chuyển không đồng bộ, các dữ liệu truyền đạt trong các khoảng cách khôngđều đặn; khi vận chuyển đồng bộ, tất cả các ký tự đều xảy ra trong khoảng thời gian cố định.

✧ Một thanh ghi nhập (dữ liệu) chứa đựng ký tự nhận được đầu tiên, rồi nó được bộ kích tạo đọc tại đó.

✧ Ở thanh ghi xuất (dữ liệu), ký tự gởi đi được bộ kích tạo viết; rồi nó được bộ điều khiển trực tiếp gới đi.

✧ Hệ thống ngắt sẽ thoát ra một ngắt sau mỗi ký tự được nhận hay được gởi đi.

Ngoài ra, hầu hết các thiết bị nối tiếp làm việc khi có dòng điều khiển được đặt trên thanh ghi điều khiển. Thông thường, việc điều khiển phần cứng được thực hiện qua các dây mảnh kim loại (chuẩn RS 232), do đó, các ký tự được (XON,XOFF...) gởi đi để điều khiển. Nếu khi nhận, bộ đệm nhận bị đe doạ đầy tràn, do đó, nó sẽ gởi đi dòng ký tự XOFF, và tạo thời cơ cho người gởi dừng lại việc gởi. Nếy bộ đêmh còn trống, thì người nhận sẽ gới đi dòng ký tự XON và báo cho người gởi tiếp tục gởi đi. Tuy nhiên, cơ chế này chỉ hoạt động, nếu người nhận gởi tín hiệu của anh ta kịp thời, do đó, người gởi có đủ thời gian để tiếp nhận và thông dịch ký tự điều khiển, và chuyển vào bộ kích tạo trước khi bộ đệm bên bị tràn.

5.3. Mô hình hoá và việc thực thi của bộ kích tạo

Một cách truyền thống, muốn viết một bộ kích tạo thiết bị đòi hỏi phải có một sự hiểu biết cơ bản cao hơn của người lập trình về thiết bị và bộ điều khiển cũng như về các chi tiết đặc sắc của sự sống lao động của hệ điều hành. Do đó, hầu hết các lỗi ở trong các hệ điều hành được đúc kết lại thành các bộ kích tạo mới; những bộ kích tạo này đã được dàn xếp với các thiết bị ngoại vi mới ở trong hệ điều hành và có thể hạn chế được các lỗi không tránh được ở trạng thái nhân.

Để giải quyết vấn đề này, điều quan trọng là các nhà sản xuất hệ điều hành đã đưa ra một giao diện hiển thị rõ ràng đối với bộ kích tạo thiết bị. Ngay cả sư tồn tại của các bộ kích tạo cao hơn đã trợ giúp rất nhiều khi tạo lập các bộ kích tạo mới khác.

Các bộ kích tạo này làm thoả mãn tất cả các nhiêmh vụ quản lý các thiết bị độc lập (như dẫn các danh sách gói tin, quản lý bộ đệm Cache...). Một lợ thế nữa là:

nếu người ta treo một kích tạo mới trực tiếp vào hệ điều hành, thì không cần phải gắn vào một nhân hệ điều hành mới.

Vấn đề tiếp theo là đàm luận về các lỗi. Sau đây, chúng ta sẽ nói tới một vài giao diện công tác cùng các chức năng của chúng.

5.3.1. Giao diện của bộ kích tạo ở Unix

Một cách quen thuộc, các bộ kích tạo Unix không thể chất tải thêm, mà chúng có thể được chuyển đổi hay hoặc có thể được kết nối với nhân một cách tính tại. Mỗi bộ kích tạo thực thi một dãy của nhiều hay ít các thủ tục đã được khẳng định; các thủ tục này được liên kết thành một bảng. Vì các thủ tục này được viết bằng ngôn ngữ C, cho nên, các bộ kích tạo sẽ được tiến trình người sử dụng gọi bởi các lệnh như open(), close(), read(), và write(). Với các thủ tục này, các việc điền vào để xác định của cấu trúc người sử dụng (u.u_offset, u.u_count, u.u_error...) được sử dụng như những bằng chứng để thông báo lỗi trở lại.

Các thủ tục của bộ kích tạo đối với một thiết bị có dạng là XX, thí dụ: XX=mt cho thiết bị băng từ; XX=rk cho ổ đĩa cứng. Sau đây sẽ giới thiệu một vài thủ tục thông dụng:

✧ XX_init là một thủ tục để thực thi bộ kích tạo và thực thi thiết bị. Điều đó được thực hiện khi khởi động hệ thống.

✧ XX_read, XX write là thủ tục để đọc/viết trực tiếp các dữ liệu (thí dụ các block bộ nhớ) theo kiểu nguyên sơ (raw modus)

✧ XX_open, XX_close là các thủ tục, mà nó được gọi bởi cá hệ thống open() và close(). Những thủ tục này dịch vụ chủ yếu để khởi động thiết bị mỗi khi sử dụng, và có ý nghĩa đặc biệt đối với thiết bị. Thí dụ, với thủ tục mt_close, một băng từ tính *(magnet band)* được bắt đầu quấn lại, với cái đó, người ta có thể lấy băng ra.

✧ XX_ioctl là một gọi hệ thống, nó nhận được các thông số, mà người ta đã đưa lại khi gọi hệ thống ioctl(), nó cũng được dùng để điều chỉnh thiết bị, điều chỉnh tốc độ vận chuyển và điều chỉnh phương pháp khi thiết bị liên kết nối tiếp.

✧ XX_strategy là một thủ tục dùng để đọc/viết các block riêng lẻ *(block device)*, và ở bên cạnh việc chuyển đổi số logic các block tới địa chỉ bộ nhớ vật lý, nó còn quan tâm thêm chiến lược đọc/viết để cải thiện cú pháp (xem mục 5.4 dưới)

Các thông tin được ghi nhớ ở đầu các block bộ nhớ. Vì thủ tục strategy được gọi một cách không đồng bộ từ bộ điều khiển ngắt để chuyển bllock kế tiếp cho thiết bị, khi tất cả các thông báo lỗi được chứa đựng ở đầu cấu trúc block, vì bộ kích tạo có thể truy cập trên tiến trình người sử dụng để nhận thấy các tham số.

✧ XX_intr là một lập thức dịch vụ ngắt *(interrup service routine)*. Ở Unix, nó thuộc bộ kích tạo và được viết bằng ngôn ngữ C. Lập thức này được dùng để sẵn sàng cứu thoát các thanh ghik khi đó nó được gọi là các lập thức ngôn ngữ máy (assemblerroutine), được dùng để vận chuyển dữ liệu cũng như để thiết đặt bộ

chuyển đổi truy cập trực tiếp DMA với sự trợ giúp của danh sách chuỗi các nhiệm vụ.

Ngoài ra, còn có các thủ tục khác, mà chúng thì phụ thuộc vào các ấn bản của Unix và được làm đầy bởi các chức năng đặc biệt.

Tất cả các thue tục của các bộ kích tạo được định nghĩa như là những địa chỉ ở trong cấu trúc bản ghi (với tệp tin conf.h) trên một thiết bị và được sắp xếp thành một bảng trung tâm (với tệp tin conf.c); bảng này có thể được tạo ra bởi một chương trình config. Mỗi một thiết bị hay mỗi một bộ điều khiển được thu xếp một số rõ ràng, gọi là số thiết bị chuyên dụng (major device number). Nếu một trong các thủ tục được gọi,do đó, điều đó chỉ xảy ra trên các bảng trung tâm. Mõi lần điền vào bảng thì bao gồm mộtcấu trúc bản ghi, mà trong đó, các địa chỉ của các thủ tục của bộ kích tạo thiết bị được dẫn tới. Chỉ số của mỗi lần điền vào thì được kết nối với một gọi hệ thống duy nhất tới các bộ kích tạo; chúng được sắp xếp thành 2 giai đoạn: theo thiết bị hướng khối*(block device switch:bdevsw)* và theo thiết bị hướng ký tự *(character device switch: cdevsw).*

Trích dẫn từ tệp tin conf.c của hệ điều hành OS/2:

```
struck bdevsw bdevsw[ ] = {
        { tmopen, tmclose, tmstrategy, tmdump,          /*0*/
                0, B_TAPE },
        { nodev, nodev, nodev, nodev,                   /*1*/
                0,B_TAPE },
        {xyopen, nulldev, xystrategy, xydump,           /*2*/
                xysize, 0}      ...};
struct cdevsw cdevsw[ ] = {
        {cnopen, cnclose, cnread, cnwrite,                      /*0*/
        cnioctl, nullev, cnselect, 0,
                0,      0,      },
        { nodev, nodev, nodev, nodev,                   /*1*/
          nodev, nodev, nodev,  0,
        swcinfo,        0,      },
        { syopen, nulldev, suread, sywrite,             /*2*/
        syioctl, nulldev, syselect, 0,
                0,      0,      },      ...};
```

Ở đây, các thiết bị hướng block có các gọi hệ thống đồng bộ: open, close, strategy, dump, psize, và flage; các thiết bị hướng ký tự có các gọi hệ thống: open, close, read, write, ioctl, reset, select, mmap, stream, segmap. Các lập thức dịch vụ ngắt không đồng bộ thì không điền vào ở đây, mà điền vào trong một đơn thể assembler bị tách biệt.

Mỗi thiết bị logic không có một sự thực thi các thủ tục của bộ kích tạo: Nếu không tồn tại việc thực thi, do đó, điều này dẫn ra với gọi hệ thống nodev và nó dẫn tới thông báo lỗi khi gọi hệ thống; nếu đơn giản điều đó không xảy ra, do đó, nó sẽ bắt đầu với gọi với gọi hệ thống nulldev. Nếu một thiết bị được thích hợp

hướng ký tự cũng như hướng block, do đó, các thủ tục của các bộ kích tạo thich hợp của thiết bị được nhận biết ở trong cả hai bảng.

Khi xuất hiện các lỗi (máy không mở, track bị hỏng, có các lỗi khi đọc...) thì chỉ số lỗi thích hợp sẽ được viết vào cấu trúc người sử dụng của tiến trình người sử dụng. Nếu bộ kích tạo nhận biết một trạng thái, mà bộ kích tạo không còn thống trị trạng thái này nữa(địa chỉ bộ đệm lỗi) và vì thế trạng thái này không thể đếm xỉa tới, do đó, còn lại khả năng cuối cùng để gọi thủ tục panic; thủ tục này in một text lỗi và dừng toàn bộ hệ điều hành.

5.3.2. Giao diện của bộ kích tạo ở trong Windows NT.

Các bộ kích tạo ở trong Windows NT phải chứa đựng những công dụng xác định. Theo R.Nigar (1997), thuộc điều đó có những thủ tục sau đây:

✧ *Thủ tục khởi động (load driver)*

Thủ tục này được thực hiện nhờ bộ điềuhành I/O, nếu bộ kích tạo hệ thống được nạp, khi đó nó sẽ tạo lập một đối tượng, mà với đối tượng này, trình điều hành I/O nhận biết thiết bị và trên thiết bị này, những thủ tục kế tiếp nó dưới đây được tham chiếu

✧ *Thủ tục để khởi động bộ vận chuyển dữ liệu(start I/O)*

Như vậy, phải có tối thiểu một thủ tục để thoát bộ vận chuyển dữ liệu (*cancel I/O*). Việc thực hiện và việc lựa chọn thủ tục này thì phụ thuộc chắc chắn vào kích cỡ dịch vụ của bộ vận chuyển dữ liệu.

✧ *Lập thức dịch vụ ngắt (interrupt service rountine: ISR):*

Hệ thống ngắt ở trong Windows NT giúp việc điều khiển hệ thống khi có ngắt thiết bị yêu cầu tới thủ tục này. Vì ngắt này xảy ra với ưu tiên rất cao, nó được lập thức ISR chờ đợi, do đó, khi có gọi ISR xuất hiện, lập thức này sẽ biến đổi thành gọi thủ tục DPC (*deferred procedure call:* DPC), và gọi này giống như là một gói tin công tác (*job package)* treo vào hàng đợi của các DPC. Nếu các DPC tiến hành với ưu tiên thấp thì một ngắt khác sẽ bẻ gãy chúng.

✧ *Thủ tục DPC bao gồm lập thức ISR:*

Các thủ tục DPC sẽ được gọi, nếu ưu tiên của các tiến trình đang diễn rơi vào ưu tiên được định nghĩa của DPC. Thủ tục DPC hoàn thiện công việc chính của lập thức ISR, đặc biệt làm hoàn thiện bộ chuyển vận dữ liệu và phân bổ nhiệm vụ kế tiếp từ hàng đợi.

✧ *Thủ tục để thoát bộ vận chuyển dữ liệu (completion rountine)*

Thủ tục này có thể dẫn tới nhờ một bộ kích tạo ở trong các gói yêu cầu I/O, do đó, nó sẽ được gọi sau khi bộ kích tạo sâu hơn kết thúc hoạt động của nó. Thủ tục này tiếp tục dẫn tới những thông tin về kết quả, các lỗi xuất hiện hay sự bẻ gãy của các bộ kích tạo cao hơn (thí dụ bộ kích tạo hệ thống tệp tin); đồng thời, nó tạo khả năng cho bộ kích tạo này kết thúc nhiệm vụ một cách thích ứng.

✧ *Thủ tục ghi chép lỗi (error logging):*

Thủ tục này dẫn tiếp các thông tin về bộ điều hành I/O, mà nó đã viết các lỗi này thành một tệp tin các lỗi.

Khi tải bộ kích tạo thì không tạo ra một đối tượng kích tạo *(driver object)* cho việc truy cập trình điều hành I/O, mà nó tạo ra một đối tượng thiết bị (device object) cho mỗi thiết bị cũng như cho mỗi chức năng thiết bị. Các đối tượn thiết bị cho thấy điểm quan hệ để sử dụng bộ kích tạo. Thí dụ khi mở tệp tin \Device\Floppy0\Text\bs_file.doc (xem hình 4.7), tên đường dẫn \Device\Floppy0\ bị tách chia: tên này chính là tên của đối tượng thiết bị mà nó có quan hệ với trình điều hành I/O. Trình điều hành I/O được hệ thống tệp tin trao nhiệm vụ nhằm đạt được các gói tin yêu cầu I/O qua đối tượng thiết bị ở bộ kích tạo có thẩm quyền về cái đó. Cho nên, tất cả các đối tượng thiêts bị của thiết bị vật lý, thiết bị logic, và thiết bị ảo đều được kết nối với bộ kích tạo thiết bị nhờ một bộ chỉ thị ở trên đối tượng kích tạo thiết bị của chúng; ngược lại, chúng là thành phần của một danh sách đối tượng của bộ kúch tạo, mà với vái đó, trình điều hành I/O có thể kiểm tra lại khi thoát khỏi bộ kích tạo *(unload driver)*, mà một trong các đối tượng thiết bị nào đó được gặp lại. Hình 5.12 minh hoạ một kiểu kết nối đó.

hình 5.12 trang 202

Ngoài các chức năng của bộ kích tạo thiết bị, người ta còn lưu ý tới những điều kiện phụ khác nhau của chúng. Một trong các nhân tố quan trọng là việc mà hệ thống có thể được thực hiện như thế nào trên các vi xử lý khác nhau của hệ thống đa vi xử lý.

Khi truy cập trên các dữ liệu, điều này cưỡng bức một sự sắp xếp để có thể sử dụng được các dữ liệu toàn cục hay các dữ liệu chia sẻ khác nhau. Điều đó có ý nghĩa đối với một bộ kích tạo khi mã của nó được làm việc đồng thời trên nhiều bộ vi xử lý, cho nên phải lưu ý việc truy cập của nó trên các thanh ghi của một thiết bị . tất nhiên, cả việc truy cập trên cấu trúc dữ liệu của bộ kích tạo (với danh sách các nhiệm vụ) cũng phải được chú ý trước.

Khác với các hệ thống đíun vi xử lý, ở hệ thống ngắt, cần phải tôn trọng việc truy cập trên cấu trúc dự liệu của bộ kích tạo. Lập thức dịch vụ ngát làm việc khi có ưutiên cao nhất, nó có thể không xuất phát từ đó để có một vài lối dẫn vào thanh ghi thiết bị, danh sách nhiệm vụ... Một bản coppy của bộ kích tạo trên bộ vi xử lý khác có ưu tiên thấp có thể đồng thời truy cập trên các dữ liệu/ cho nên, ở trong nhân hệ điều hành có các thủ tục đồng bộ nhân đặc biệt để dẫn tới các thủ tục như: các thủ tục bận chờ *(busy wait)*, thủ tục khoá vòng *(spin lock)*...

Một nhiệm vụ tiếp theo của bộ kích tạo là việc phản ứng lại sự sụt áp nguồn điện *(power failure)*. Ở đây, nó được lưới điện dự kiến, nó có thể nhận được sự cung cấp ngay lập tức sai khi tín hiệu mất nguồn điện khoảng vài mili giây. Khi đó, các dữ liệu quan trọng được bảo vệ. Một bộ kích tạo để phân biệt được các phần nhân tử trong hoạt động của nó; các phần này khôngđược phép phá hỏng dữ liệu để nhằm đảm bảo việc tích hợp các dữ liệu. Ngoài ra, trong giai đoạn này phải đảm bảo không để xảy ra tình trạng mất nguồn điện lâu hơn; sau khi ngăn cản việc

ngắt mất nguồn điện, các dữ liệu riêng lẻ được chép lại và tiếp đến ngắt được trả tự do. Ngược lại, khi mất nguồn điện, thiết bị phải được đặt vào trnạg thái được điều khiển quen thuộc trước khi nguồn điện được đóng lại.

Nếu hệ thống cung cấp một khoảng RAM có nuôi pin, thì khi đó, có thể tất cả các dữ liệu tiến trình quan trọng (ngữ cảnh tiến trình) được cứu, do đó, nó có thể phải tiếp tục thực hiện chương trình sau khi tái phục hồi điện áp nguồn.

5.4. Các chiến lước tối ưu các thiết bị nhờ bộ kích tạo.

Nếu chúng ta viết một bộ kích tạo cho một thiết bị, thì trong đó, chúng ta phải làm thoả mãn các Bit điều khiển thiết bị. Với các tài liệu ít ỏi của nhà sản xuất, các bộ kích tạo này phải đựoc thông dịch một cách sáng tạo.bằng các chiến lược sẽ được nghiên cứu dưới đây, chúng ta có điều kiện để thực thi các bộ kích tạo và để tối ưu việc truy cập đối với một thiết bị. Cụ thể, một khả năng duy nhất đối với chúng ta, đó là việc thiết kế một chiến lược định thời cho việc truy cập ổ đĩa.

5.4.1. Các chiến lược định thời cho truy cập ổ đĩa

Bộ kích tạo của một thiết bị có địa chỉ xác định thì nó gần như thực hiện liên tục một dãy các nhiệm vụ. Đối với một thiết bị, chẳng hạn một ổ đĩa từ tính, bao giờ cũng có thơhì gian trễ cơ học, do đó, mỗi nhiệm vụ được kết nối theo một thời gian chờ. Thời gian chờ này thì phụ thuộc vào trạng thái của thiết bị khi phân bổ nhiệm vụ. Khi truy cập ổ đĩa thì đó là trạng thái đọc/viết của đầu từ, tức là khi đó một track (có một chỉ số nào đó) mà trên đó chứa đựng các block (gồm nhiều sector) tồn tại kế cạnh nhau cần đọc và viết. Nếu bộ kích tạo của thiết bị có mặt trong các hàng chờ đợi nhiệm vụ, dó đó, nó có thể thử tìm kiếm để tăng tốc độ công việc nhờ sắp xếp theo lớp các nhiệm vụ ở trong danh sách. Điều này thì thích hợp với chiến lược định thời giiống như chiến lược phân bổ các tiến trình tại một bộ vi xử lý đã nghiên cứu ở chương 2.

Đối với chiến lược định thời, đến nay có rất nhiều phương pháp khác nhau. Hình 5.13 giới thiệu phương pháp nổi tiếng nhất đang thịnh hành trên thế giới. Phương pháp này quy tụ các chiến lược sau:

hình 5.13 trang 204

Chiến lược *đến trước dịch vụ trước (first come first serve:* FCFS):

Chiến lược này là đơn giản nhất nhưng mỹ mãn nhất; vì nó thực hiện theo nguyên tắc *công bằng xã hội*: tất cả mọi nhiệm vụ được thực hiện theo một dãy tuần tự mà trong đó chúng xuất hiện.

Thí dụ: Giả sử một đĩa cứng có 20 track và chứa đựng một danh sách các nhiémh vụ. Đầu từ đang ở vị trí track số 6, dãy nhiệm vụ của nó được xác định: nó phải chuyển vị đến làm việc tại các track theo thứ tự 8, 19, 3, 14, 2, 15, 7. Nếu lấu

bề rộng track làm đơn vị tình thì đoạn đường mà đầu từ phải dịch chuyển cơ học là:

2+11+16+11+12+13+8 = 73 track

✧ Chiến lược *thời gian tìm kiếm ngắn nhất trước nhất (shortest seek time first:SSTF):*

Nội dung của chiến lược này là, đầu từ có thể chuyển đổi từ một track này tới một track kia mà không cần phải lưu ý khoảng cách giữa chúng. Điều đó có ý nghĩa, đầu tiên, người ta có thể hoàn thành các nhiệm vụ có phạm vi gần nhau nhằm tránh được đầu từ phải di chuyển nhiều.

Để lần lượt lựa chọn nhiệm vụ kế cạnh, mà địa chỉ track của nó khoảng cách nhỏ nhất tới trạng thái địa chỉ hiện hành, thì chiến lược này thích hợp với trường hợp chuyển động tìm kiếm track của đầu từ là ngắn nhất.

Thí dụ: Danh sách nhiệm vụ của đầu từ có dãy tuần tự với các số của track: 7, 8, 3, 2, 14, 15, 19. Theo chiến lược này, quỹ đạo của đầu từ được chỉ trong hình 5.13 và đoạn đường của nó được xác định:

1+1+5+12+1+14 = 25 track.

Người ta có thể so sánh chiến lược này với chiến lược *Job ngắn nhất trước nhất* của phương pháp định thời đã nói ở mục 2.2 ở trước. Còn vấn đề cần nói: Khi dãy tuần tự các nhiệm vụ không thuận tiện, tức là một nhiệm vụ nằm ở cuối đĩa từ (hay cuối khoảng đĩa chỉ) có thể bị thiêtj thòi và dẫn đến *trạng thái chết đói (hurry-status).*

✧ Chiến lược *quét ổ đĩa (Scan, C-Scan)*

Ý tưởng có bản của chiến lược này là ở chỗ, các track được rà (SCAN) lần lượt một cách hệ thống và trên con đường này tất cả các nhiệm vụ được thực hiện.

Thí dụ: Việc xử lý công việc SCAN có một danh sách các nhiệm vụ được thiết lập; chẳng hạn đầu từ đang ở vị trí track số 6, được khởi hành theo hướng các track có chỉ số lớn dần, với dãy tuần tự:7, 8, 14, 15, 19, 3, 2. Do vậy, tổng quảng đường mà đầu từ phải đi qua là;

1+1+6+1+4+16+1 = 30 track.

Thông thường kết thúc chu trình làm việc, đầu từ sẽ ở vị trí có địa chỉ của track thấp nhất hay cao nhất; do đó, hướng tìm kiếm và hướng làm việc ít bị thay đổi, làm giảm thiếu đáng kể thời gian cơ học. Chúng ta nhận thấy rằng, nếu xác suất nhiệm vụ qua các track là như nhau; do đó, khi thay đổi hướng chuyển động của đầu từ ở cuối hành trình (thí dụ tại track có chỉ số lớn nhất), thì độ tập trung cao nhất của các nhiệm vụ là ở đầu hành trình (tại các track có chỉ số nhỏ nhất). Điều đó thì cũng giống như trường hợp quét lá đa đình làng: Khi người ta vừa quét xong đường chổi cuối cùng ở cuối sân đình, thì phí đầu sânn đình là đa đã rụng đầy(!).

Cho nên điều có lợi là: dãy tuần tự công việc mới được bắt đầu từ điểm dừng của dãy tuần tự công việc lần trước, chứ không cần phải quay lại hướng làm việc như lúc đầu. Trong hệ điều hành, trạng thái này quen thuộc và nó được gọi là quét tuần hoàn (cirular scan: C-SCAN). Lấy ví dụ cho trường hợp với dãy các nhiệm vụ: 7, 8, 14, 15, 19, 2, 3. Hành trình làm việc của đầu từ cũng xuất hiện từ track số 6 và được chỉ ở trên hình 5.13 ở trên.

Bây giờ, chúng ta không đi theo con đường vừa nêu nữa, mà chuyển hướng làm việc khi tới nhiệm vụ cuối cùng. Phương pháp này được gọi là giải thuật nhìn theo*(look algorithmus:* LOOK) hay còn gọi giải thuật nhìn quay vòng (*circular look algorithmus:*C-LOOK).

Các chiến lược này đã được trình bày cho tới nay rất thích hợp, bởi vì thời gian chuyển động của đầu từ (*seek time)* để tìm kiếm một track thì rõ ràng lớn hơn thời gian trễ trên một track khi tiềm kiếm một sector. Nếu điều đó xảy ra, tức là tại đĩa cứng trên mỗi track có một đầu từ được lắp cứng *(fix head disk)*; do đó, danh sách các nhiệm vụ được sắp xếp theo các tiêu chuẩn khác. Khi đó các chiến lược định thời phải được nhận thức như sau:

- Chiến lược FCFS chỉ được dùng đối với các công việc đơn giản theo danh sách dãy tuần tự các nhiệm vụ xuất hiện
- Chiến lược xếp hàng các sector: Các nhiệm vụ được sắp xếp theo chỉ số sector tăng dần, để nhằm đạt được mộtquá trình xử lý liên tục. Vì điều này xảy ra đồng thời với tất cả các track, do đó, đối với mỗi chỉ số sector phải có một hàng đợi xác định.

Điều chúng ta cần đặt câu hỏi là: Các chiến lược được dẫn ra cho tới nay được đánh giá như thế nào? Cụ thể là: các chiến lược SSTF và SCAN làm cho các tiến trình đọc/viết các block riêng lẻ có thể bị *chết đói một cách lý thuyết*. Tuy nhiên, khi phân bổ nhiệm vụ thì điều đó không phải ngẫu nhiên, mà nó có thể tránh được nhờ một số thay đổi nhỏ các giải thuật thuần thiết.

Nhưng vẫn còn một câu hỏi khác: Việc sử dụng các giải thuật định thời được phức tạp hoá dẫn tới ích lợi gì? Việc mô hình hoá đã chỉ ra rằng: khi tất cả các nhiệm vụnặng nề đối với các giải thuật SCAN và C-SCAN, thì kết quả cho thấy tốt nhất. Tuy nhiên, điều đó đòi hỏi những công việc quản lý bổ sung, mà nó có lợi khi tải các nhiệm vụ nặng nề. Bấy giờ, nhiều công trình nghiên cứu chỉ ra rằng, công việc sẽ diễn biến bình thường khi danh sách chỉ chứa đựng một nhiệm vụ duy nhất. Trong trường hợp này, tất cả các chiến lược đều tốt, nhất là chiến lược FCFS vừa đơn giản vừa tốt nhất.

Các chiến lược để xử lý danh sách nhiệm vụ của block thì không phải là khả năng duy nhất để tối ưu thời gian truy cập. Điều quyết định để có thể thực hiện việc truy cập nhanh hơn là: Nếu chúng ta mô phỏng block logic với chỉ số trung tâm *(central index)* không phải trên sector 0 hay track 0 ở cuối đĩa từ; do đó, hành trình trung bình của đầu từ khi truy cập tới tất cả các track thì nhỏ hơn ở cuối và việc truy cập rõ ràng nhanh hơn. Ngược lại, nếu chỉ số trung tâm được giữ lại như một bản photocopy ở trong bộ nhở chính, do đó, chỉ có dãy các lệnh định vị trí là quyết định chứ không phải vị trí của chỉ số trung tâm.

5.4.2. Kểu bố trí đĩa từ xen kẽ *(interleaving)*

Một khả năng tối ưu tiếp theo được dẫn ra cho trường hợp: Khi các blocks được ổ đĩa chuyển cho bộ điều khiển nhanh hơn khi chúng có thể được chuyển vận tiếp theo nhờ đầu từ. Trong trường hợp này, block số 4 không được đọc ngay sau block số 3; vì đĩa từ đã quay tiếp trong khi vận chuyển block số3; và bấy giờ nó đọc block số 5 chưa không đọc block số 4. khi đó, bộ điều khiển chờ đợi vòng quay kế tiếp của đĩa từ để cuối cùng có thể đọc được block số 4. Bây giờ, nếu chúng ta đổi việc đánh số các block, chẳng hạn block số 5 nhận số 4, nhờ vậy, chúng ta có thể tiếp tục đọc mà không cần thời gian chờ đợi. Trong trường hợp này, nếu chỉ mỗi block vật lý thứ 2 được chọn để đánh số logic, do đó, cần dành nhiều thời gian cho việc xác định các block và gia tăng lưu lượng dữ liệu. Hình 5.14 (a) cho thấy việc đánh số vật lý được ghi chép ở ngời ổ đĩa. Trên các cung đĩa từ *(semgent)* ở phần bên trong, việc đánh số logic thích hợp được ghi nhận.

Kiểu dông nghệ này được gọi là phép bố trí xen kẻ (*interleaving)* đĩa từ; số lượng các block (bị bỏ sót lần đầu khi đánh số) là yếu tố xen kẽ. Hình 5.14 (b) là cách đánh số đối với yếu tố xen kẻ 1 và 2 được đặt đối diện nhau. Yếu tố này phải được bộ phận kích tạo thông báo đối với việc tạo kiểu dáng cho bộ điều khiển và nó phụ thuộc vào tốc độ vận chuyển của hệ thông I/O của hệ điều hành.

hình 5.14 trang 207

Những cơ chế được đàm luận cho đến nay để gia tăng hiệu suất bộ kích tạo cần phải được xem xét một cách toàn diện trong ngữ cảnh của bộ điều khiển. Một cách nguyên tắc, trường hợp có thể xuất hiện một cách dễ dàng: tất cả các biện pháp tối ưu đối với các bộ kích tạo đều được trang bị với những bộ vi xử lý độc đáo. Thí dụ, các bộ điều khiển như vậy có khả năng thay thế các block bị hư hỏng của đĩa từ nhờ các block dự trữ ở trên các track (chuyên dụng bình thường nhưng không dễ xen vào). Hình thái đặc biệt luôn luôn có hiệu lực, nếu có một block hư hỏng được nhận biết và được thuyên chuyển đo khỏi, tuy có thể không được bộ kích tạo nhận thấy. Với cái đó, tất cả sự nỗ lực của bộ kích tạo cho thấy, cần phải hạn chế chuyển động của đầu từ; cụ thể là: các chuyển động trung gian tới các track thay thế cho phép không cần tối ưu từ bên ngoài nữa. Do đó, mục đích của chúng ta là: Giao diện giữa bộ kích tạo và bộ điều khiển phải được nhà sản xuất hệ thống lo toan và có kế hoạch trước. Thí dụ, người ta có thể dẫn tới sự tối ưu các sự phụ thuộc của thiết bị như chuyển động của đầu từ và tìm kiếm sector cho bộ điều khiển. Như vậy, với việc lưu trữ các block bằng các địa chỉ logic đã làm cho hiệu suất nói chung gia tăng rõ ràng.

5.4.3. Kiểu đệm thêm *(bufferring)*

Người ta có thể đạt được một sự tối ưu hiệu suất quan trọng cho các bộ nhớ đĩa từ nhờ bộ đệm, còn gọi là bộ đệm dữ liệu *(data-cache)*, nó được tạo nên trên những bình diện khác nhau.

Trên bình diện của bộ kích tạo hệ thống tệp tin, điều tiện lợi là: nó được dùng để đệm thêm ở hầu hết các block được sử dụng cho một tệp tin; ở đó, người ta có thể phân nó thành hai loại khác nhau, đó là buffer đọc và buffer viết với một sự quản lý chặt chẽ.

Trên bình diện của bộ kích tạo thấp nhất, người ta có thể đệm thêm những đơn vị đọc/viết, thí dụ đệm thêm toàn bộ một track. Hầu hết các tác vụ đọc/viết theo có quan hệ với nhau trên các số sector tương ứng và có thể được thực hiện bởi track được đệm thêm một cách nhanh hơn thực thụ.

Tuy nhiên, ở đây, việc đệm thêm cũng có những vấn đề tương tự như đã được mô tả ở trong mục 3.5 chương trước. Khib quản lý, cần phải đảm bảo: những block và sector được mô tả sẽ được áp dụng cho việc thăm dò các tiến trình khác nhau để đảm bảo độ bền vững của dữ liệu.

Điều quan trọng là việc làm đồng bộ nội dung của buffer với bộ nhớ đĩa từ trước khi hệ thống máy tính được ngắt khỏi nguồn điện lưới. Khi mất nguồn điện lưới cung cấp. Điều đó sẽ được dẫn vào ngay lập tức nhờ các bộ kích tạo được đệm thêm.

Ơ đây, người ta phait lưu ý một cách trù tính giao diện với bộ điều khiển. Những bộ điều khiển như vậy thường được tích hợp bên trong một bộ Cache; ở đây, chẳng có mục đích gì hơn ngoài việc thiết đặt một bộ Cache trên bình dìnn bộ kích tạo.

Thí dụ kiểu đệm thêm ở Unix:

Việc đệm thêm dược thực hiện cho các tập tin đặc trưng nối tiếp. Từ lý do này, một dãy ký tự RETURN được sử dụng để đọc vào một bài text từ bàn phím. Đặc biệt, các Bit trạng thái của tệp tin đặc biệt này cho phép đọc ngay mỗi ký tự đối với một lần nhập ký tự thuần khiết mà không cần đệm hay để nén lại âm hưởng cục bộ của một ký tự trên màn hình.

Hệ thống bộ đệm đối với các thiết bị block sử dụng các block như những đơn vị bộ nhớ. Đối với mỗi bộ kích tạo có một danh sách các nhiệm vụ cho các block, mà các block này cần đọc/viết. Danh sách các block trống bị làm trễ gấp đôi và được liên kết lại thành một vùng nhớ tập trung. Ngoài ra, có 2 con đường dẫn tới các thiết bị block: con đường thứ 1 qua tên một tệp tin và do đó, qua hệ thống tệp tin nguyên sơ, tức là qua thủ tục XX_stragety và con đường thư 2 qua tệp tin đặc biệt như là một thiết bị nguyên sơ, tức là qua thủ tục XX_read/XX_write.

Vì nhờ việc đệm thêm của nút chỉ số *(index node)*, mà khi có sự tổn thất mạng, hệ thống tệp tin hay bị làm tổn hại; do đó, ở nhịp độ đều đặn (khoảng 30 giây) của thủ tục sync(), tất cả các bộ đệm được viết lên đĩa từ và qua đó, chúng định dạng các dữ liệu. Điều đó sẽ được thực hiện khi thoát khỏi hệ thống (*shut down).*

Thí dụ đệm thêm ở windows NT:

Để quản lý bộ đệm Cache cho việc xuất-nhập có một trình điều hành Cache đặc biệt. Trình điều hành này cấp phát các trang một cách năng động ở trong bộ nhớ chính và tạo ra một sự ảnh xạ bộ nhớ giữa các trang bộ nhớ chính và một tệp tin. Số lượng các đối tượng đoạn *(section object)* là năng động: nó phụ thuộc vào bộ nhớ chính được sử dụng cũng phụ thuộc vào sự thường xuyên truy cập trên các phần tệp tin. Điều đó sẽ xảy ra: các trang của trình điều hành bộ đệm Cache giống như các trang của một tiến trình xẽ được trình điều hành bộ nhớ ảo quản lý. Số lượng các trang tồn tại trong bộ nhớ chính sẽ được điều chỉnh qua cơ chế tập công tác *(working set)* hay cơ chế ảnh xạ *(mapping mechanismus),* xem mục 3.3 ở trên.

Đối với việc đệm thêm khi xuất-nhập nối tiếp, các cơ chế đặc biệt phải được phát triển để nhằm nhận được các kết quả của các chương trình từ đa tác vụ không có chặn trước của Windows 3.1 (16 Bit) đến đa tác vụ có chặn trước của Windows NT. Đối với mỗi thiết bị, ở trong Windows 3.1, có một hàng đợi I/O duy nhất và đối với các thiết bị nối tiếp cũng vậy. Nếu trong cửa sổ khác nhau, các việc nhập cào (nhập ký tự, kích chuột...) được dẫn ra, do đó, chúng sẽ đặt một đơn vị nhập vào bộ đệm nhập. Một cách tiện lợi, một tiến trình đọc/viết đưới Windows 3.1 kép dài bất kỳ (không có chặn trước), cho đến khi nó hoàn thành công việc nhập vào và bị hãm lại khi đọc trên bộ đệm nhập, nếu công việc tiếp theo không còn nữa đối với chúng. Tiến trình thuộc cửa sổ này được hoạt động bởi bộ điều hành cửa sổ và đọc vào phần bộ đệm của nó.

Bây giờ, nếu chúng ta đi qua một giứoi hạn chặn trước, ở đó, một tiến trình có thể cùng được hoạt động ngay, nếu khoảng thời gian *(time slice)* của nó đã trôi qua, do đó, điều đó sẽ dẫn tới những vấn đề tại bộ nhớ đệm nhập: tiến trình (mới được tăng cường) này sẽ đọc các dữ liệu đã được xác định. Đó là trường hợp, khi tiến trình được dựng lại và được kết nối đầu cuối mà trước đó còn chứa đựng lỗi. Từ lý do này, ở Windows NT, mỗi tiến trình thread có một hàng đợi nhập riêng lẻ; việc nhập vào chưa được đọc này đối với một tiến trình thread được giữ lại tiến trình này và không được đọc thực thụ bới tiến trình thread kế cạnh; đó là hệ thống mạnh mẽ đối diện với các tiến trình chứa đựng lỗi.

Chúng ta có thể tích hợp 2 hệ thống khác nhau như Windows 3.1 và Windows NT như thế nào? Tính logic của các tiến trình không chặn trước ở Windows 3.1 cũng được dự đoán như tính logic của tiến trình trọng lượng nhẹ *(thread)*, Bây giừo, ý tưởng này chỉ ra, cần phải sử dụng cơ chế của tiến trình thread ở trong Windows NT. Do đó, những nhà thiết kế hệ điều hành Windó NT đã lý giải Windows 3.1 như một hệ thống con 16 Bit. Hệ thống này đã cho xuất phát các tác vụ 16 Bit như các tiến trình threads riêng lẻ. Tuy nhiên, một cách đúng mức, một bộ vi xử lý đã cứu thoát tiến trình này (cùng như tiến trình NT khác); khi trả lại bộ vi xử lý, tiến trình thread (xuất hiện sau cùng) nhận lại việc điều khiển một cách tự động; khi đó, coi như không xảy ra việc thay đổi tiến trình. Do đó, ở trong hệ thống Windows/DOS cũ, một số đặc điểm đạt được một cách tiện dụng.

Tuy nhiên, nó còn có thể hơn thế: ở trong không gian địa chỉ bị tách chía, các ứng dụng 16 Bit được diễn biến như những tiến trình độc đáo. Trong trường hợp

này chúng tỏ ra không còn ép buộc như trước đây dưới Windows 3.1; cho nên, điều này không còn có ý nghĩa ở tất cả các áp dụng cũ.

5.4.4. Đồng bộ và không đồng bộ việc vào-ra.

Việc vào-ra trong một chương trình thì cần thiết phải chờ đợi cho tới khi một gọi hệ thống được kết thúc một cách mỹ mãn. Khi đó, người ta gọii quá trình này là làm đồng bộ việc vào-ra *(I/O synchrone)*. Bấy giừơ, nó cần một khoảng thời gian cố định nào đó, cho tới khi việc vao-ra được thực hiện. thời gian này có thể được chương trình sử dụng một cách tốt hơn cho các công việc khác nhau. Ngay trong các chương trình đang tồn tại tiến trình thread, thì đều không thể nhận ra: tại sao tất cả các tiến trình thread chờ đợi việc vào-ra(!).

Một khả năng để loại bỏ việc hãm chặn vừa nói là việc thừa nhận các tiến trình threads như là những tiến trình trọng lượng nhẹ nhờ hệ điều hành Windows NT. Khi đó, một cách phổ biến, chỉ có một thread bị hãm, còn các threads khác của tiến trình thì không. Tuy nhiên, điều này không phải bao giờ cũng xảy ra trong hệ điều hành.

Một khả năng khác cho vấn đề này là vấn đề không đồng bộ tiến trình vào-ra (I/O asynchrone). Khi đó, gọi hệ thống sẽ dẫn tới chỉ một tác vụ xuất-nhập; kết quả sẽ được một tiến trình thread đón nhận một cách muộn hơn nhờ một lệnh đặc biệt. Kiểu gọi hệ thống này đặt ra cho hệ điều hành những yêu cầu đặc biệt; bởi vì ở đây, nhiệm vụ cũng như kết quả phải được quản lý và phải được lưu trữ trung gian thì phụ thuộc vào tiến trình uỷ nhiệm.

Không đồng bộ I/O ở Unix:

Ở các ấn bản thông thường của Unix, các gọi hệ thống cho phép chuyển đổi khi đọc/viết không bị ngăn hãm. Khi nhảy lui khỏi các thủ tục đọc/viết, số lượng các Bytes đọc/viết được hoàn trở lại. Nếu nó bằng 0 thì quá trình đọc/viết không xảy ra.

Việc dẫn tới một quá trình không đồng bộ I/O đối với một tiến trình thì bình thường là không thể được. Tuy nhiên với một sự khéo léo, chúng ta có thể đạt được điều này. Cho cái đó, chúng ta sử dụng cơ chế *fork()* theo mục 2.1.1 (với hình minh hoạ 2.4) để nhận được bản copy của tiến trình. Tại quá trình I/O, bản copy bị hãm lại một cách đại diện. Sau đó, nếu tại bản copy chúng ta dự định một cơ chế *exit()*, do đó tiến trình chính có thể tiếp tục làm việc, cho tới khi, với một cơ chế *wait()*, nó có thể đón nhận kết quả của tác vụ I/O.

Tuy nhiên, khả năng này là không thể thực tiến: để tạo lập được một tiến trình như vậy, phải tốn quá nhiều thời gian; vì rằng, kiểu vào-ra không đồng bộ này thì hầu như không mang lại lợi thế nào. Một quyết định cho cái đó là một sự kết nối nhanh chóng việc trao đổi thông tin với một tiến trình con (thí dụ qua bộ nhớ chia sẽ). Tiến trình con nay được tạo ra một lần và được sử dụng một bộ móc nối I/O

và đối với tiến trình con này được tạo ra một lần và được sử dụng như một bộ móc nối I/O và đối với tiến trình chính, nó chờ đợi một cách đại diện.

Cả hai cơ chế vừa nêu là một bộ phận thay thế không thể thiếu được với các khả năng không đồng bộ I/O ở trong Unix.

Không đồng bộ I/O ở Windows NT:

Trong hệ điều hành Windows NT, nếu việc vao-ra được thực hiện thì nó phụ thuộc vào một tham số, mà tham số này được dẫn ra bởi các cơ chế WriteFile(), ReadFile(), CreateFile()...

Trong trường hợp bình thường, gọi hệ thống được dẫn qua tất cả các lớp (dịch vụ hệ thống, trình điều hành I/O, bộ kích tạo thiết bị, ngắt chuyển đổi, nhảy lui) và dừng tiến trình tại một gọi hệ thống ReadFile(), cho tới khi các dữ liệu mong muốn trải ra. Ngược lại, nếu một thông số được chỉ ra overlapped (bị chồng lên nhau), do đó, dòng điều khiển sẽ đi tới việc hoãn lại và dẫn ra bộ chuyển đổi dữ liệu ngay lập tức trở lại gọi hệ thống. Tiến trình có thể tiếp tục làm việc, nhưng làm công việc khác. Để nhận được các số liệu mong muốn, một gọi hệ thống Wait (fileHandle) được thay thế, nó sẽ hãm chặn tiến trình thread, cho tới khi số liệu mong muốn được đưa ra. Đối tượng fileHandle được thuyên chuyển thành trạng thái *báo trước* và do đó, tiến trình được đánh thức. Bây giờ, nó có thể thực hiện gọi hệ thống ReadFile() và cuối cùng nó đọc các dữ liệu.

Tuy nhiên, người ta phải lưu ý, không sử dụng một tiến trình thread thứ hai (chẳng hạn fileHandle) để dẫn tới hay chờ đợi mộttiến trình I/O không đồng bộ; tức là, với đối tượng fileHandle, một tín hiệu đánh thức cả hai tiến trình threads, mà chỉ một trong hai của nó khi đón nhận sai, do đó, khi xuất nhập các dữ liệu đã cho thấy rằng tiến trình đã được chuyển đổi. Một lối thoát khỏi tình trạng này là sử dụng các đối tượng biến cố riêng lẻ hay sử dụng các gọi thủ tục không đồng bộ *(asynchronous procedůe calls:*APCs) cho mỗi tiến trình thread.

Một cách độc lập với cái đó, tiến trình thread (vừa gọi) lập tức nhận được sự điều khiển trở lại và có thể tiếp tục làm việc. Sau đó, tiến trình thread này phải tự dẫn vào trạng thái chờ đợi, thí dụ nhờ các gọi hệ thống SleepEx(), WaitForSingleObjectEx() hay WaitForMultipleObjectEx(). Nếu việc đọc/viết được kết thúc một cách không đồng bộ, do đó, tiến trình thread lại được đánh thức và cũng khi đó, thủ tục kết thúc được gọi, thủ tục này có thể tạo thời có cho bước tiếp theo.

5.5. Các bài tập của chương 5

Bài tập 5.1. Về định thời truy cập ổ đĩa

Trong chương này, bạn đã nghiên cứu các chiến lược định thời đối với việc truy cập ổ đĩa như FCFS, SSTF, LOOK, và SCAN.

a). Việc dò tìm khi truy cập ổ đĩa có thể bị làm đói như thế nào?

b). Các chiến lược riêng lẻ được biến hoá như thế nào, mà với điều này, khả năng làm đói không còn nữa?

Bài tập 5.2. Về phân bố đan chen nhau trên ổ đĩa từ

a). Sự phân bố đan chen trên ổ đĩa từ dùng để làm gì?

b). Một ổ đĩa từ có một hệ số đan chen nhau bằng 2. Ổ đĩa chứa trên mỗi vòng xuyến khoảng 80 sector, mỗi sector có 256 Byte và quay với tốc độ 3600 vòng/phút. Để đọc thông tin trên tất cả các sector của một vòng xuyến một cách tuần tự thì cần bao nhiêu thời gian? Giả thiết rằng, đầu từ đọc/viết luôn luôn định vị đúng và một nữa vòng quay được sử dụng cho tới kho sectors số 0 nằm dưới đầu từ.

c). bạn hãy nhắc lại vấn đề đối với một ổ đĩa tương tự, nhưng không có hệ số đan chen, và một ổ đĩa với hệ số đanchen bằng 3. Hỏi tỷ phần dữ liệu tăng gat giảm như thế nào

Bài tập 5.3 Lưu trữ trên bộ nhớ đệm

Bạn hãy quan sát mô hình các lớp sau đây:

Hệ thống tệp tin
Bộ kích tạo đao thiết bị
Bộ kích tạo đơn thiết bị
Bộ điều khiển
Thiết bị

Những ưu điểm và khuyết điểm nào đã được dẫn ra, khi lưu trữ đệm vào một trong các lớp được thực hiện? cho cái đó, bạn hãy phân biệt thành 4 trường hợhp. Bạn hãy lưu ý một cách đặc biệt khi dữ liệu linh động.