CHƯƠNG 1: TỔNG QUAN VỀ HỆ ĐIỀU HÀNH

1.1 Khái niệm hệ điều hành

Phần mềm máy tính có thể chia thành nhiều loại: chương trình hệ thống quản lý sự hoạt động của chính máy tính. Chương trình ứng dụng, giải quyết các vấn đề liên quan đến việc sử dụng và khai thác máy tính của người sử dụng. Hệ điều hành thuộc nhóm các chương trình hệ thống.

Hệ điều hành là một *hệ thống các chương trình* hoạt động giữa người sử dụng (user) và phần cứng của máy tính. Mục tiêu của hệ điều hành là cung cấp một môi trường thuận lợi để người sử dụng có thể thi hành các chương trình. Hệ điều hành thực hiện quản lý các tài nguyên máy tính. Hệ điều hành làm cho máy tính dễ sử dụng hơn, thuận lợi hơn và hiệu quả hơn.

Để đạt được mục tiêu trên hệ điều hành phải thực hiện 2 chức năng chính sau đây:

- Giả lặp một máy tính mở rộng

Hệ điều hành che đậy các chi tiết phần cứng của máy tính bởi một máy tính mở rộng, máy tính mở rộng này có đầy đủ các chức năng của máy tính thực nhưng đơn giản và dễ sử dụng hơn. Theo đó khi cần tác động vào máy tính thực người sử dụng chỉ cần tác động vào máy tính mở rộng, mọi sự chuyển đổi thông tin điều khiển từ máy tính mở rộng sang máy tính thực hoặc ngược lại đều do hệ điều hành thực hiện.

Mục đích của chức năng này là giúp người sử dụng khai thác các chức năng của phần cứng máy tính dễ dàng và hiệu quả hơn.

- Quản lý tài nguyên của hệ thống:

  + Tài nguyên phần cứng: CPU, RAM, I/O device…

Nhu cầu tài nguyên nhiều, do vậy cần quản lý, điều phối tài nguyên một cách có hiệu quả. Hệ điều hành còn phải tổ chức bảo vệ các không gian nhớ đã cấp cho các chương trình, tiến trình để tránh sự truy cập bất hợp lệ và sự tranh chấp bộ nhớ giữa các chương trình, tiến trình, đặc biệt là các tiến trình đồng thời hoạt động trên hệ thống.

  + Tài nguyên phần mềm (data)

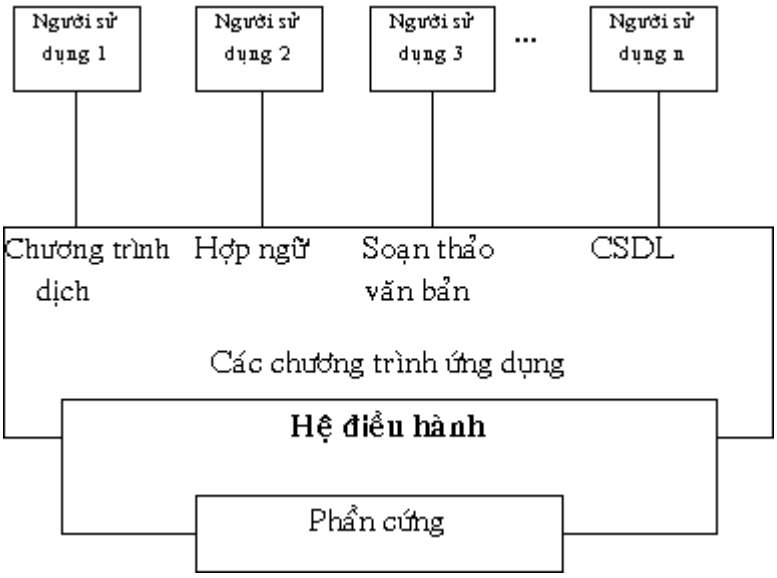
Trong trường hợp nhiều tiến trình đồng thời sử dụng một không gian nhớ hay một tập tin nào đó. Trong trường hợp này hệ điều hành phải tổ chức việc chia sẻ và giám sát việc truy xuất đồng thời trên các tài nguyên nói trên sao cho việc sử dụng tài nguyên có hiệu quả nhưng tránh được sự mất mát dữ liệu và làm hỏng các tập tin.

Hệ điều hành là một phần quan trọng của hầu hết các hệ thống máy tính. Một hệ thống máy tính thường được chia làm bốn phần chính : phần cứng, hệ điều hành, các chương trình ứng dụng và người sử dụng.

Phần cứng bao gồm CPU, bộ nhớ, các thiết bị nhập xuất, đây là những tài nguyên của máy tính.

Chương trình ứng dụng như các chương trình dịch, hệ thống cơ sở dữ liệu, các trò chơi, và các chương trình thương mại. Các chương trình này sử dụng tài nguyên của máy tính để giải quyết các yêu cầu của người sử dụng.

Hệ điều hành điều khiển và phối hợp việc sử dụng phần cứng cho những ứng dụng khác nhau của nhiều người sử dụng khác nhau. Hệ điều hành cung cấp một môi trường mà các chương trình có thể làm việc hữu hiệu trên đó.



Hình 1.1 Mô hình trừu tượng của hệ thống máy tính

1.2 Lịch sử phát triển của hệ điều hành

Thế hệ 1 (1945 – 1955)

Vào khoảng giữa thập niên 1940, Howard Aiken ở Havard và John von Neumann ở Princeton, đã thành công trong việc xây dựng máy tính dùng ống chân

không. Những máy này rất lớn với hơn 10000 ống chân không nhưng chậm hơn nhiều so với máy rẻ nhất ngày nay.

Mỗi máy được một nhóm thực hiện tất cả từ thiết kế, xây dựng lập trình, thao tác đến quản lý. Lập trình bằng ngôn ngữ máy tuyệt đối, thường là bằng cách dùng bảng điều khiển để thực hiện các chức năng cơ bản. Ngôn ngữ lập trình chưa được biết đến và hệ điều hành cũng chưa nghe đến.

Vào đầu thập niên 1950, phiếu đục lổ ra đời và có thể viết chương trình trên phiếu thay cho dùng bảng điều khiển.

Thế hệ 2 (1955 – 1965)

Sự ra đời của thiết bị bán dẫn vào giữa thập niên 1950 làm thay đổi bức tranh tổng thể. Máy tính trở nên đủ tin cậy hơn. Nó được sản xuất và cung cấp cho các khách hàng. Lần đầu tiên có sự phân chia rõ ràng giữa người thiết kế, người xây dựng, người vận hành, người lập trình, và người bảo trì.

Để thực hiện một công việc (một chương trình hay một tập hợp các chương trình), lập trình viên trước hết viết chương trình trên giấy (bằng hợp ngữ hay FORTRAN) sau đó đục lỗ trên phiếu và cuối cùng đưa phiếu vào máy. Sau khi thực hiện xong nó sẽ xuất kết quả ra máy in.

Hệ thống xử lý theo lô ra đời, nó lưu các yêu cầu cần thực hiện lên băng từ, và hệ thống sẽ đọc và thi hành lần lượt. Sau đó, nó sẽ ghi kết quả lên băng từ xuất và cuối cùng người sử dụng sẽ đem băng từ xuất đi in.

Hệ thống xử lý theo lô hoạt động dưới sự điều khiển của một chương trình đặc biệt là tiền thân của hệ điều hành sau này. Ngôn ngữ lập trình sử dụng trong giai đoạn này chủ yếu là FORTRAN và hợp ngữ.

Thế hệ 3 (1965 – 1980)

Trong giai đoạn này, máy tính được sử dụng rộng rãi trong khoa học cũng như trong thương mại. Máy IBM 360 là máy tính đầu tiên sử dụng mạch tích hợp (IC). Từ đó kích thước và giá cả của các hệ thống máy giảm đáng kể và máy tính càng phỗ biến hơn. Các thiết bị ngoại vi dành cho máy xuất hiện ngày càng nhiều và thao tác điều khiển bắt đầu phức tạp.

Hệ điều hành ra đời nhằm điều phối, kiểm soát hoạt động và giải quyết các yêu cầu tranh chấp thiết bị. Chương trình hệ điều hành dài cả triệu dòng hợp ngữ và do hàng ngàn lập trình viên thực hiện.

Sau đó, hệ điều hành ra đời khái niệm ***đa chương***. CPU không phải chờ thực hiện các thao tác nhập xuất. Bộ nhớ được chia làm nhiều phần, mỗi phần có một công việc (job) khác nhau, khi một công việc chờ thực hiện nhập xuất CPU sẽ xử lý các công việc còn lại. Tuy nhiên khi có nhiều công việc cùng xuất hiện trong bộ nhớ, vấn đề là phải có một cơ chế bảo vệ tránh các công việc ảnh hưởng đến nhau. Hệ điều hành cũng cài đặt thuộc tính spool.

Giai đoạn này cũng đánh dấu sự ra đời của ***hệ điều hành chia xẻ thời gian*** như CTSS của MIT. Đồng thời các hệ điều hành lớn ra đời như MULTICS, UNIX và hệ thống các máy mini cũng xuất hiện như DEC PDP-1.

Thế hệ 4 (1980 - nay)

Máy tính dùng mạch tích hợp cỡ lớn.

Giai đoạn này đánh dấu sự ra đời của máy tính cá nhân, đặc biệt là hệ thống IBM PC với hệ điều hành MS-DOS và Windows sau này. Bên cạnh đó là sự phát triển mạnh của các hệ điều hành tựa Unix trên nhiều hệ máy khác nhau như Linux. Ngoài ra, từ đầu thập niên 90 cũng đánh dấu sự phát triển mạnh mẽ của ***hệ điều hành mạng*** và ***hệ điều hành phân tán***.

Các hệ điều hành ra đời ở giai đoạn này như: Ubuntu, Windows 7, Windows 8,…

1.3. Phân loại hệ điều hành

1.3.1 Hệ điều hành xử lý theo lô đơn giản

Hệ điều hành thực hiện các công việc lần lượt theo những chỉ thị đã được xác định trước. Khi một công việc chấm dứt thì hệ thống sẽ tự động thực hiện công việc tiếp theo mà không cần sự can thiệp từ bên ngoài, do đó hệ thống đạt tốc độ thực hiện cao. Để thực hiện được điều này hệ điều hành phải có bộ giám sát thường trực để giám sát việc thực hiện của các công việc trong hệ thống, bộ phận này thường trú trong bộ nhớ chính. Trong hệ điều hành này khi hệ thống cần thực hiện một công việc thì nó phải lưu chương trình và dữ liệu của công việc vào hàng đợi các công việc, sau đó sẽ

thực hiện lần lượt từng bộ chương trình và dữ liệu của công việc tương ứng trong hàng đợi và cho ra lần lượt các kết quả.

Với cách tổ chức công việc, thì hệ thống không thể thay đổi chương trình và dữ liệu của các công việc ngay cả khi chúng còn nằm trong hàng đợi, đây là một hạn chế. Mặt khác trong quá trình thực hiện công việc, nếu công việc chuyển sang truy xuất trên thiết bị vào/ra thì processor rơi vào trạng thái chờ, điều này gây lãng phí thời gian xử lý processor.

1.3.2 Hệ điều hành xử lý theo lô đa chương

Hệ điều hành có khả năng thực hiện nhiều công việc, nhiều chương trình đồng thời. Khi cần thực hiện nhu công việc đồng thời hệ điều hành sẽ nạp một phần code và data của các công việc vào bộ nhớ (các phần còn lại sẽ được nạp sau tại thời điểm thích hợp) và tất cả đều ở trạng thái sẵn sàng thực hiện, sau đó hệ điều hành bắt đầu thực hiện một công việc nào đó, nhưng khi công việc đang thực hiện cần truy xuất thiết bị vào/ra thì processor sẽ được chuyển sang thực hiện các công việc khác, và cứ như thế hệ điều hành chuyển hướng processor để thực hiện hết các phần công việc trong bộ nhớ cũng như các công việc mà hệ thống yêu cầu.

Hệ điều hành này có 2 ưu điểm đó là tiết kiệm được bộ nhớ, vì không nạp hết code và data của các công việc vào bộ nhớ, và hạn chế thời gian rỗi của processor. Tuy nhiên nó phải chịu chi phí cao cho việc lập lịch processor. Ngoài ra hệ điều hành còn phải giải quyês việc chia sẻ bộ nhớ chính cho các công việc khác nhau. Ví dụ hệ điều hành MS_DOS là hệ điều hành đơn nhiệm, đa chương.

1.3.3 Hệ điều hành chia xẻ thời gian

Khái niệm chia sẻ thời gian ra đời đã đánh dấu một bước phát triển mới của hệ điều hành trong việc điều khiển các hệ thống đa người dùng. Chia sẻ thời gian ở đây chính là chia sẻ thời gian xử lý của processor cho các công việc, các tiến trình đang ở trong trạng thái sẵn sàng thực hiện.

Nguyên tắc của hệ điều hành chia sẻ thời gian tương tự như trong hệ điều hành xử lý theo lô đa chương nhưng việc chuyển processor từ công việc, tiến trình này sang công việc, tiến trình khác không phụ thuộc vào việc công việc, tiến trình hiện tại có truy xuất đến thiết bị vào/ra hay không mà chỉ phụ thuộc vào sự điều phối processor của hệ điều hành. Công việc điều phối processor của hệ điều hành rất phức tạp phụ thuộc vào nhiều yếu tố khác nhau.

Trong hệ điều hành này thời gian chuyển đổi processor giữa các công việc là rất nhỏ nên ta có cảm giác các công việc thực hiện song song với nhau. Với hệ điều hành này người sử dụng có thể yêu cầu hệ điều hành thực hiện nhiều chương trình, công việc đồng thời với nhau.

Hệ điều hành chia sẻ thời gian là mở rộng logic của hệ điều hành đa chương và nó thường được gọi là hệ điều hành đa nhiệm (Multitasking). Hệ điều hành Windows 9x/NT là các hệ điều hành đa nhiệm.

1.3.4 Hệ điều hành đa vi xử lý.

Là các hệ điều hành dùng điều khiển sự hoạt động của các hệ thống máy tính có nhiều bộ vi xử lý. Các hệ điều hành đa vi xử lý (multiprocessor) gồm có 2 loại:

- Đa xử lý đối xứng (SMP: symmetric): Trong hệ thống này vi xử lý nào cũng có thể chạy một loại tiểu trình bất kỳ, các vi xử lý giao tiếp với nhau thông qua một bộ nhớ dùng chung. Hệ SMP cung cấp một cơ chế chịu lỗi và khả năng cân bằng tải tối ưu hơn, vì các tiểu trình của hệ điều hành có thể chạy trên bất kỳ vi xử lý nào nên nguy cơ xảy ra tình trạng tắc nghẽn ở CPU giảm đi đáng kể. Vấn đề đồng bộ giữa các vi xử lý được đặt lên hàng đầu khi thiết kế hệ điều hành cho hệ thống cho hệ thống SMP. Hệ điều hành Windows NT, Windows 2000 là các hệ điều hành đa xử lý đối xứng.

- Đa xử lý bất đối xứng (ASMP: asymmetric): Hệ điều hành dành ra một hoặc hai vi xử lý để sử dụng riêng, các vi xử lý còn lại dùng để điều khiển các chương trình của người sử dụng. Hệ AMSP đơn giản hơn nhiều so với hệ SMP, nhưng trong hệ này nếu có một vi xử lý trong các vi xử lý dành riêng cho hệ điều hành bị hỏng thì hệ thống có thể ngừng hoạt động.

1.3.5 Hệ điều hành mạng

Là các hệ điều hành dùng để điều khiển sự hoạt động của mạng máy tính. Ngoài các chức năng cơ bản của một hệ điều hành, các hệ điều hành mạng còn phải thực hiện việc chia sẻ và bảo vệ tài nguyên của mạng. Hệ điều hành Windows9x/NT, Windows 2000, Linux.

1.3.6 Hệ điều hành xử lý thời gian thực

Hệ điều hành này có khả năng cho kết quả tức thời, chính xác sau mỗi công việc.

Một hệ điều hành xử lý thời gian thực phải được định nghĩa tốt, thời gian xử lý nhanh. Hệ thống phải cho kết quả chính xác trong khoảng thời gian bị thúc ép nhanh nhất. Có hai hệ thống xử lý thời gian thực là hệ thống thời gian thực cứng và hệ thống thời gian thực mềm.

Hệ thống thời gian thực cứng là công việc được hoàn tất đúng lúc đó dữ liệu thường được lưu trong bộ nhớ ngắn hạn hay trong ROM. Việc xử lý theo thời gian thực sẽ xung đột với tất cả hệ thống liệt kê ở trên.

Dạng thứ hai là hệ thống thời gian thực mềm, mỗi công việc có một độ ưu tiên riêng và sẽ được thi hành theo độ ưu tiên đó. Có một số lĩnh vực áp dụng hữu hiệu phương pháp này là multimedia hay thực tại ảo.

1.4 Các thành phần của hệ điều hành

a)Thành phần quản lý tiến trình

Một ***tiến trình*** là một chương trình đang được thi hành. Một tiến trình phải sử dụng tài nguyên như thời gian sử dụng CPU, bộ nhớ, tập tin, các thiết bị nhập xuất để hoàn tất công việc của nó. Các tài nguyên này được cung cấp khi tiến trình được tạo hay trong quá trình thi hành.

Một tiến trình là hoạt động (active) hoàn toàn-ngược lại với một tập tin trên đĩa là thụ động (passive)-với một bộ đếm chương trình cho biết lệnh kế tiếp được thi hành.Việc thi hành được thực hiện theo cơ chế tuần tự , CPU sẽ thi hành từ lệnh đầu đến lệnh cuối.

Một tiến trình được coi là một đơn vị làm việc của hệ thống. Một hệ thống có thể có nhiều tiến trình cùng lúc , trong đó một số tiến trình là của hệ điều hành, một số tiến trình là của người sử dụng. các tiến trình này có thể diễn ra đồng thời.

Vai trò của hệ điều hành trong việc quản lý tiến trình là :

- Tạo và hủy các tiến trình của người sử dụng và của hệ thống.

- Tạm dừng và thực hiện tiếp một tiến trình.

- Cung cấp các cơ chế đồng bộ tiến trình.

- Cấp phát tài nguyên cho tiến trình.

- Cung cấp các cơ chế giao tiếp giữa các tiến trình.

- Cung cấp cơ chế kiểm soát deadlock

b)Thành phần quản lý bộ nhớ chính :

Bộ nhớ là thiết bị lưu trữ duy nhất mà CPU có thể truy xuất trực tiếp. Bộ nhớ chính có thể xem như một mảng kiểu byte hay kiểu word. Mỗi phần tử đều có địa chỉ. Đó là nơi lưu dữ liệu được CPU truy xuất một cách nhanh chóng so với các thiết bị nhập/xuất. CPU đọc những chỉ thị từ bộ nhớ chính. Các thiết bị nhập/xuất cài đặt cơ chế DMA cũng đọc và ghi dữ liệu trong bộ nhớ chính. Thông thường bộ nhớ chính chứa các thiết bị mà CPU có thể định vị trực tiếp. Ví dụ CPU truy xuất dữ liệu từ đĩa, những dữ liệu này được chuyển vào bộ nhớ qua lời gọi hệ thống nhập/xuất.

Một chương trình muốn thi hành trước hết phải được ánh xạ thành địa chỉ tuyệt đối và nạp vào bộ nhớ chính. Khi chương trình thi hành, hệ thống truy xuất các chỉ thị và dữ liệu của chương trình trong bộ nhớ chính. Ngay cả khi tiến trình kết thúc, dữ liệu vẫn còn trong bộ nhớ cho đến khi một tiến trình khác được ghi chồng lên.

Hệ điều hành có những vai trò như sau trong việc quản lý bộ nhớ chính :

- Lưu giữ thông tin về các vị trí trong bộ nhớ đã được sử dụng và tiến trình nào đang sử dụng.

- Quyết định tiến trình nào được nạp vào bộ nhớ chính, khi bộ nhớ đã có thể dùng được.

- Cấp phát và thu hồi bộ nhớ khi cần thiết.

- Bảo vệ bộ nhớ

c) Thành phần quản lý bộ nhớ phụ

Bộ nhớ chính quá nhỏ để có thể lưu giữ mọi dữ liệu và chương trình, ngoài ra dữ liệu sẽ mất khi không còn được cung cấp năng lượng. Hệ thống máy tính ngày nay cung cấp ***hệ thống lưu trữ phụ***. Đa số các máy tính đều dùng đĩa để lưu trữ cả chương trình và dữ liệu. Hầu như tất cả chương trình : chương trình dịch, hợp ngữ, thủ tục, trình soạn thảo, định dạng... đều được lưu trữ trên đĩa cho tới khi nó được thực hiện, nạp vào trong bộ nhớ chính và cũng sử dụng đĩa để chứa dữ liệu và kết quả xử lý. Vai trò của hệ điều hành trong việc quản lý đĩa :

- Quản lý vùng trống trên đĩa.

- Định vị lưu trữ thông tin trên đĩa.

- Lập lịch cho vấn đề ghi/đọc thông tin trên đĩa của đầu từ.

d) Quản lý hệ thống vào/ ra :

Một trong những mục tiêu của hệ điều hành là *che dấu* những đặc thù của các thiết bị phần cứng đối với người sử dụng thay vào đó là một lớp thân thiện hơn, người sử dụng dể thao tác hơn. Một hệ thống vào/ra bao gồm :

- Thành phần quản lý bộ nhớ chứa vùng đệm (buffering), lưu trữ (caching) và spooling (vùng chứa).

- Giao tiếp điều khiển thiết bị (device drivers) tổng quát.

- Bộ điều khiển cho các thiết bị xác định.

Chỉ có bộ điều khiển cho các thiết bị xác định mới hiểu đến cấu trúc đặc thù của thiết bị mà nó mô tả.

e) Thành phần quản lý tập tin :

Máy tính có thể lưu trữ thông tin trong nhiều dạng thiết bị vật lý khác nhau : băng từ, đĩa từ, đĩa quang, ... Mỗi dạng có những đặc thù riêng về mặt tổ chức vật lý. Mỗi thiết bị có một bộ kiểm soát như bộ điều khiển đĩa (disk driver) và có những tính chất riêng. Những tính chất này là tốc độ, khả năng lưu trữ, tốc độ truyền dữ liệu và cách truy xuất.

Để cho việc sử dụng hệ thống máy tính thuận tiện, hệ điều hành cung cấp một cái nhìn logic đồng nhất về hệ thống lưu trữ thông tin. Hệ điều hành định nghĩa một đơn vị lưu trữ logic là tập tin. Hệ điều hành tạo một ánh xạ từ tập tin đến vùng thông tin trên đĩa và truy xuất những tập tin này thông qua thiết bị lưu trữ.

Một tập tin là một tập hợp những thông tin do người tạo ra nó xác định. Thông thường một tập tin đại diện cho một chương trình và dữ liệu. Dữ liệu của tập tin có thể là số, là ký tự, hay ký số.

Vai trò của hệ điều hành trong việc quản lý tập tin :

- Tạo và xoá một tập tin.

- Tạo và xoá một thư mục.

- Hỗ trợ các thao tác trên tập tin và thư mục.

- Tạo mối quan hệ giữa tập tin và bộ nhớ phụ chứa tập tin.

- Sao lưu dự phòng các tập tin trên các thiết bị lưu trữ.

- Bảo vệ tập tin khi có hiện tượng truy xuất đồng thời

-Tạo cơ chế truy xuất tập tin thông qua tên tập tin

f) Hệ thống bảo vệ :

Trong một hệ thống nhiều người sử dụng và cho phép nhiều tiến trình diễn ra đồng thời, các tiến trình phải được bảo vệ đối với những hoạt động khác.Do đó, hệ thống cung cấp cơ chế để đảm bảo rằng tập tin, bộ nhớ, CPU, và những tài nguyên khác chỉ được truy xuất bởi những tiến trình có quyền. Ví dụ, bộ nhớ đảm bảo rằng tiến trình chỉ được thi hành trong phạm vi địa chỉ của nó. Bộ thời gian đảm bảo rằng không có tiến trình nào độc chiếm CPU. Cuối cùng các thiết bị ngoại vi cũng được bảo vệ.

Hệ thống bảo vệ là một cơ chế kiểm soát quá trình truy xuất của chương trình, tiến trình, hoặc người sử dụng với tài nguyên của hệ thống. Cơ chế này cũng cung cấp cách thức để mô tả lại mức độ kiểm soát.

Hệ thống bảo vệ cũng làm tăng độ an toàn khi kiểm tra lỗi trong giao tiếp giữa những hệ thống nhỏ bên trong.

g) Thành phần thông dịch lệnh :

Một trong những phần quan trọng của hệ điều hành là hệ thống thông dịch lệnh, đó là giao tiếp giữa người sử dụng và hệ điều hành. Thành phần này chính là Shell.

Một số hệ điều hành chứa Shell trong nhân của nó, một số hệ điều hành khác thì Shell được thiết kế dưới dạng một chương trình đặc biệt.

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

Shell nhận lệnh thông qua cơ chế dòng lệnh, đó chính là nơi giao tiếp giữa người sử dụng và hệ điều hành, mỗi hệ điều hành khác nhau có cơ chế dòng lệnh khác nhau, với MS_DOS đó là con trỏ lệnh và dấu nhắc của hệ điều hành (C:\>), với Windows 9x đó là nút Start/Run. Tập tin Command.com chính là Shell của MS_DOS.

Trong môi trường đơn nhiệm, ví dụ như MS_DOS, khi tiến trình đáp ứng yêu cầu hoạt động thì Shell sẽ chuyển sang trạng thái chờ, để chờ cho đến khi tiến trình đáp ứng yêu cầu kết thúc thì Shell trở lại trạng thái sẵn sang nhận lệnh mới.

Trong môi trường hệ điều hành đa nhiệm, ví dụ như Windows 9x sau khi phát sinh tiến trình đáp ứng yêu cầu và đưa nó vào trạng thái hoạt động thì Shell sẽ chuyển sang trạng thái sẵn sang nhận lệnh mới, nhờ vậy Shell có khả năng khởi tạo nhiều tiến trình đáp ứng yêu cầu để nó hoạt động song song với nhau, hay chính xác hơn trong môi trường hệ điều hành đa nhiệm, người sử dụng có thể khởi tạo nhiều chương trình để nó hoạt động đồng thời với nhau.

Chú ý: hầu hết các ngôn ngữ lập trình đều hỗ trợ các công cụ để người sử dụng hay ngươi lập trình có thể gọi Shell ngay trong các ứng dụng của họ. Khi một ứng dụng cần gọi thực hiện một chương trình nào đó thì:

Trong Assembly, các ứng dụng gọi hàm 4Bh/21h của MS_DOS.

Trong Pascal, các ứng dụng gọi thủ tục Exec

Trong VisualBasic, các ứng dụng gọi hàm/ thủ tục shell. Ví dụ dòng lệnh sau: Shell”C:\Windows\Notepad.exe” có thể gọi thực hiện chương trình Notepad của Windows.

Trong Windows 9x/WindowsNT, các ứng dụng gọi hàm ShellExecute.

Chú ý: Cần phải phân biệt sự khác nhau giữa Shell và System Call. Shell tạo môi trường giao tiếp giữa người sử dụng và hệ điều hành, System call tạo môi trường giao tiếp giữa chương trình người sử dụng và hệ điều hành.

h) Thành phần quản lý mạng

Xem xét đến các vấn đề liên lạc giữa các tiến trình, chia sẻ tài nguyên chung, vấn đề bảo mật trên các tiến trình trong các hệ thống khác nhau.

1.5 Các cấu trúc của hệ thống

a) Hệ thống đơn khối (monolithic systems)

Trong hệ thống này hệ điều hành là một tập hợp các thủ tục, mỗi thủ tục có thể gọi thực hiện một thủ tục khác bất kỳ lúc nào khi cần thiết.

Thông thường hệ điều hành bắt đầu là một hệ thống nhỏ, đơn giản và có giới hạn.

Cấu trúc tối thiểu phân chia các thủ tục trong hệ thống thành 3 cấp độ:

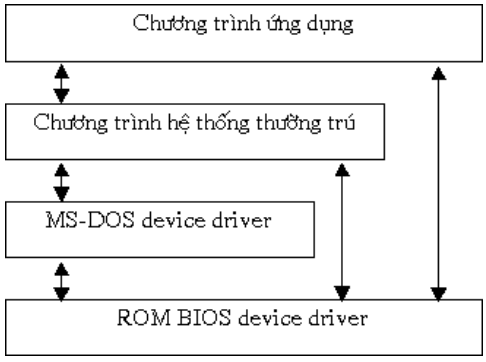
- Một chương trình chính(chương trình của người sử dụng) gọi đến một thủ tục dịch vụ của HĐH. Lời gọi này được gọi là lời gọi hệ thống (system call)
- Một tập các thủ tục dịch vụ để đáp ứng những lời gọi hệ thống từ các chương trình của người sử dụng
- Một tập các thủ tục tiện ích hỗ trợ các thủ tục dịch vụ trong việc thực hiện các lời gọi hệ thống

Nhược điểm:

- Không có sự che dấu dữ liệu, mỗi thủ tục có thể gọi đến tất cả các thủ tục khác. Chương trình ứng dụng có thể truy xuất các thủ tục cấp thấp tác động đến cả phần cứng do vậy HĐH khó kiểm soát và bảo vệ hệ thống.
- Các thủ tục dịch vụ mang tính chất tĩnh, nó chỉ hoạt động khi được gọi bởi chương trình của người sử dụng, điều này làm cho hệ điều hành thiếu chủ động trong việc quản lý môi trường

MS-DOS là một hệ điều hành có cấu trúc đơn giản, nó cung cấp những chức năng cần thiết nhất trong một không gian nhỏ nhất do sự giới hạn của phần cứng mà nó chạy trên đó và không chia thành những đơn thể rõ rệt.

Hình 1.2 Cấu trúc của MS-DOS



Mặc dù MS-DOS có cấu trúc nhưng giữa giao diện và chức năng không có sự phân chia rõ rệt. Các chương trình ứng dụng có thể truy xuất trực tiếp các thủ tục nhập xuất cơ bản và ghi trực tiếp lên màn hình hay bộ điều khiển đĩa.

b) Các hệ thống phân lớp

Hệ thống được chia thành một số lớp, mỗi lớp dược xây dựng dựa vào các lớp bên trong. Lớp trong cùng thường là phần cứng, lớp ngoài cùng là giao tiếp với người sử dụng.

Mỗi lớp là một đối tượng trừu tượng chứa đựng bên trong nó các dữ liệu và các thao tác xử lý dữ liệu đó. Lớp n chứa đựng một cấu trúc dữ liệu và các thủ tục có thể được gọi bởi lớp n+1 hoặc ngược lại có thể gọi các thủ tục ở lớp n-1.

Nhận xét:

-Hệ thống này mang tính đơn thể, nên dễ cài đặt, tìm lỗi và kiểm chứng hệ thống.

Ưu điểm là tính module. Các lớp được chọn dựa trên cơ sở lớp trên sử dụng chức năng và các dịch vụ chỉ của lớp dưới nó. Tiếp cận này đơn giản hóa việc gỡ rối và kiểm tra hệ thống. Lớp đầu tiên có thể được gỡ rối mà không có bất cứ sự quan tâm nào cho lớp còn lại của hệ thống. Bởi vì theo định nghĩa, nó chỉ sử dụng phần cứng cơ bản để cài đặt các chức năng của nó. Một khi lớp đầu tiên được gỡ rối, chức năng sửa lỗi của nó có thể được đảm đương trong khi lớp thứ 2 được gỡ rối, ...Nếu một lỗi được tìm thấy trong khi gỡ rối cho một lớp xác định, lỗi phải được nằm trên lớp đó vì các lớp bên dưới đã được gỡ rối rồi. Do đó, thiết kế và cài đặt hệ thống được đơn giản hóa khi hệ thống được phân chia thành nhiều lớp.

Mỗi lớp được cài đặt chỉ với các thao tác được cung cấp bởi các lớp bên dưới. Một lớp không cần biết các thao tác được cài đặt như thế nào; nó chỉ cần biết các thao tác đó làm gì. Do đó, mỗi lớp che giấu sự tồn tại của cấu trúc dữ liệu, thao tác và phần cứng từ các lớp cấp cao hơn.

-Các nhà thiết kế gặp khó khăn trong việc xác định số lượng lớp, thứ tự và chức năng của mỗi lớp.

Khó khăn chính của tiếp cận phân lớp liên quan tới việc định nghĩa cẩn thận các lớp vì một lớp chỉ có thể sử dụng các lớp bên dưới nó. Thí dụ, trình điều khiển thiết bị cho không gian đĩa được dùng bởi các giải thuật bộ nhớ ảo phải nằm ở tại cấp thấp hơn trình điều khiển thiết bị của các thủ tục quản lý bộ nhớ vì quản lý bộ nhớ yêu cầu khả năng sử dụng không gian đĩa.

Các yêu cầu có thể không thật sự rõ ràng. Thường thì các trình điều khiển lưu trữ dự phòng nằm trên bộ định thời CPU vì trình điều khiển cần phải chờ nhập/xuất và CPU có thể được định thời lại trong thời gian này. Tuy nhiên, trên hệ thống lớn, bộ định thời có thể có nhiều thông tin hơn về tất cả quá trình đang hoạt động hơn là có thể đặt vừa trong bộ nhớ. Do đó, thông tin này có thể cần được hoán vị vào và ra bộ nhớ, yêu cầu thủ tục trình điều khiển lưu trữ dự phòng nằm bên dưới bộ định thời CPU.

-Trong một số trường hợp lời gọi thủ tục có thể lan truyền đến các thủ tục khác ở các lớp bên trên nên chi phí cho vấn đề truyền tham số và chuyển đổi ngữ cảnh tăng lên, dẫn đến lời gọi hệ thống trong cấu trúc này thực hiện chậm hơn so với các cấu trúc khác

Vấn đề cuối cùng với các cài đặt phân lớp là chúng có khuynh hướng ít hiệu quả hơn các loại khác. Thí dụ, khi chương trình người dùng thực thi thao tác nhập/xuất, nó thực thi một lời gọi hệ thống. Lời gọi hệ thống này được bẫy (trapped) tới lớp nhập/xuất, nó yêu cầu tầng quản lý bộ nhớ, sau đó gọi tầng định thời CPU, sau đó được truyền tới phần cứng. Tại mỗi lớp, các tham số có thể được hiệu chỉnh, dữ liệu có thể được truyền,…Mỗi tầng thêm chi phí cho lời gọi hệ thống; kết quả thực sự là lời gọi hệ thống mất thời gian lâu hơn khi chúng thực hiện trên hệ thống không phân tầng.

Cấu trúc lớp này lần đầu tiên được thiết kế và áp dụng cho hệ điều hành THE (Technische Hogeschool Eindhoven). Hệ thống này được chia thành sáu lớp như hình sau:

Lớp 5	Chương trình của người sử dụng
Lớp 4	Tạo buffer cho thiết bị nhập xuất
Lớp 3	Device driver thao tác màn hình
Lớp 2	Quản lý bộ nhớ
Lớp 1	Lập lịch CPU
Lớp 0	Phần cứng

Hình 1.3 Cấu trúc của hệ điều hành THE

Các ví dụ khác như cấu trúc lớp của hệ điều hành VENUS và OS/2

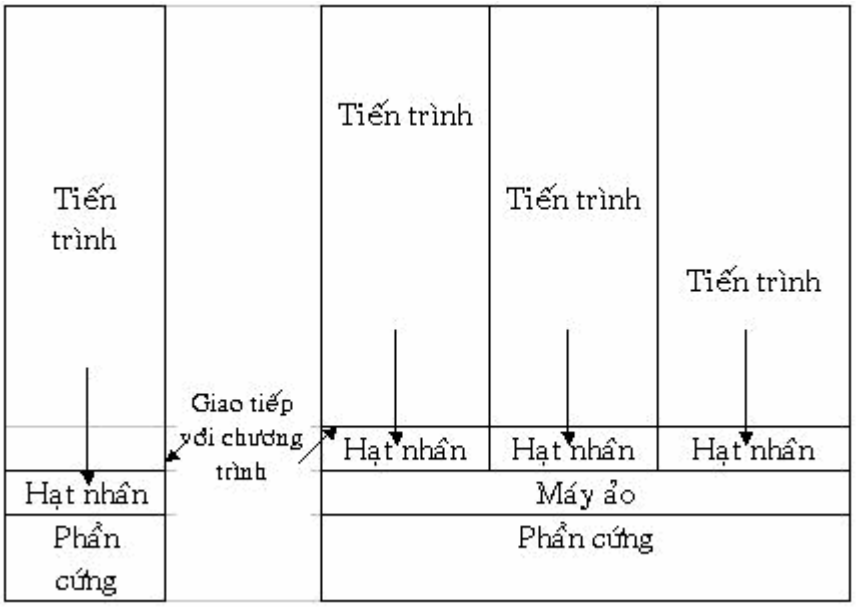
c) Máy ảo (Virtual Machine)

Các máy ảo là những bản sao ảo chính xác các đặc tính phần cứng của máy tính thực sự và cho phép hệ điều hành hoạt động trên đó như trên phần cứng thực sự. Phần nhân hệ thống thực hiện giám sát máy ảo chịu trách nhiệm giao tiếp với phần cứng, chia sẻ tài nguyên hệ thống để tạo ra nhiều máy ảo, hoạt động độc lập với nhau, để cung cấp cho lớp bên trên.

Với cấu trúc này mỗi tiến trình hoạt động trên một máy ảo độc lập và nó có cảm giác như đang sở hữu một máy tính thực sự.

Mục đích của việc sử dụng máy ảo là xây dựng các hệ thống đa chương với nhiều tiến trình thực hiện đồng thời, mỗi tiến trình được cung cấp một máy ảo với đầy đủ tài nguyên, tất nhiên là tài nguyên ảo, để nó thực hiện được.

Vấn đề phức tạp nhất của máy ảo là hệ thống đĩa. Giả sử hệ thống chỉ có ba bộ điều khiển đĩa nhưng có tới bảy máy ảo. Như vậy không thể gán cho mỗi máy ảo một bộ điều khiển đĩa và giải pháp là xây dựng hệ thống đĩa ảo.



Hình 1.4 So sánh giữa máy thực và máy ảo

Nhận xét:

- Việc cài đặt các phần mềm giả lặp phần cứng để tạo ra máy ảo thường rất khó khăn và phức tạp.

- Trong hệ thống này vấn đề bảo vệ tài nguyên hệ thống và tài nguyên đã cấp phát cho tiến trình, sẽ trở nên đơn giản hơn vì mỗi tiến trình thực hiện trên một máy tính (ảo) độc lập với nhau nên việc tranh chấp tài nguyên là không thể xảy ra.

- Nhờ hệ thống máy ảo mà một ứng dụng được xây dựng trên hệ điều hành có thể hoạt động được trên hệ điều hành khác.

Trong môi trường hệ điều hành Windows 9x người sử dụng có thể thực hiện được các ứng dụng được thiết kế để thực hiện trên môi trường MS_DOS, sở dĩ như vậy vì Windows đã cung cấp cho các ứng dụng này một máy ảo DOS (VMD: Virtual Machine DOS) để nó hoạt động như đang hoạt động trong hệ điều hành DOS. Tương tự như trong môi trường hệ điều hành Windows NT người sử dụng có thể thực hiện được các ứng dụng được thiết kế trên một số hệ điều hành khác, có được điều này là nhờ cấu trúc của Windows NT có chứa các hệ thống con (subsystem) môi trường tương thích với các môi trường hệ điều hành khác như : Win32, OS/2,..các ứng dụng khi cần thiết thực hiện trên Windows NT sẽ thực hiện trong các hệ thống con môi trường tương ứng, đúng với môi trường mà ứng dụng đó được tạo ra.

d) Mô hình Client/Server

Các hệ điều hành hiện đại thường chuyển dần các nhiệm vụ của hệ điều hành ra các lớp bên ngoài nhằm thu nhỏ phần cốt lõi của hệ điều hành thành hạt nhân cực tiểu (kernel) sao cho chỉ phần hạt nhân này chỉ phụ thuộc vào phần cứng. Để thực hiện được điều này hệ điều hành xây dựng theo mô hình Client/Server, theo mô hình này hệ điều hành bao gồm nhiều tiến trình đóng vai trò server có các chức năng chuyên biệt như quản lý tiến trình, quản lý bộ nhớ,…phần hạt nhân của hệ điều hành chỉ thực hiện nhiệm vụ tạo cơ chế thông tin liên lạc giữa các tiến trình client và các tiến trình server.

Như vậy các tiến trình trong hệ thống được chia thành 2 loại:

- Tiến trình bên ngoài hay tiến trình của chương trình người sử dụng được gọi là các tiến trình client.

- Tiến trình của hệ điều hành được gọi là các tiến trình server.

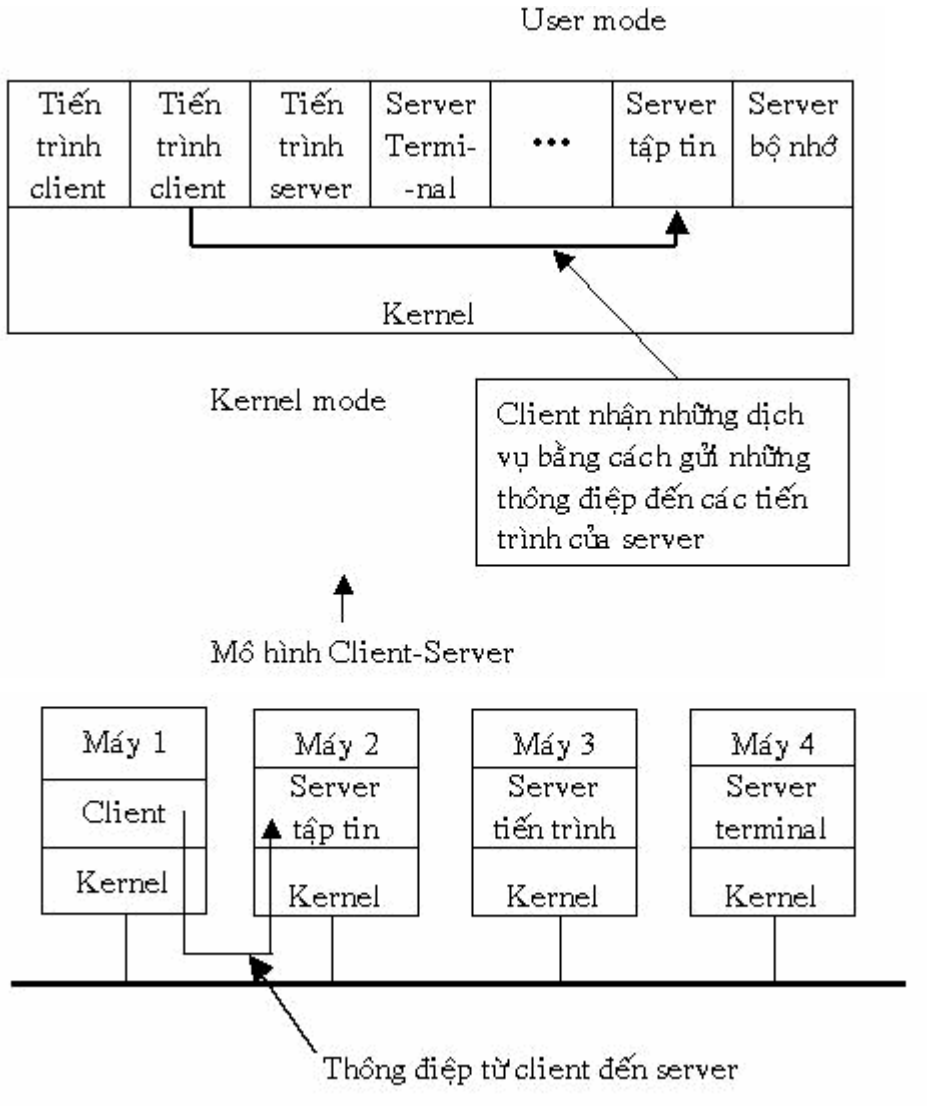
Khi cần thực hiện một chức năng hệ thống các tiến trình Client sẽ gởi yêu cầu tới tiến trình server tương ứng, tiến tình server sẽ xử lý và trả lời kết quả cho tiến trình client.

Nhận xét:

-Hệ thống này dễ thay đổi và dễ mở rộng hệ điều hành. Để thay đổi các chức năng của hệ điều hành chỉ cần thay đổi server tương ứng, để mở rộng hệ điều hành chỉ cần thêm các server mới vào hệ thống.

-Các tiến trình server của hệ điều hành hoạt động trong chế độ không đặc quyền nên không thể truy cập trực tiếp tới phần cứng, điều này giúp cho hệ thống được bảo vệ tốt hơn.

Ví dụ: WindowsNT



Hình 1.5 Mô hình Client-Server trong hệ thống phân tán

1.6 Các tính chất cơ bản của hệ điều hành

a) Tin cậy

Mọi hoạt động, mọi thông báo của HĐH đều phải chuẩn xác, tuyệt đối. chỉ khi nào biết chắc chắn là đúng thì HĐH mới cung cấp thông tin cho người sử dụng. Để đảm bảo được yêu cầu này, phần thiết bị kỹ thuật phải có những phương tiện hỗ trợ kiểm tra tính đúng đắn của dữ liệu trong các phép lưu trữ và xử lý. Trong các trường hợp còn lại HĐH thông báo lỗi và ngừng xử lý trao quyền quyết định cho người vận hành hoặc người sử dụng.

b) An toàn

Hệ thống phải tổ chức sao cho chương trình và dữ liệu không bị xoá hoặc bị thay đổi ngoài ý muốn trong mọi trường hợp và mọi chế độ hoạt động. Điều này đặc biệt quan trọng khi hệ thống là đa nhiệm. Các tài nguyên khác nhau đòi hỏi những yêu cầu khác nhau trong việc đảm bảo an toàn.

c) Hiệu quả

Các tài nguyên của hệ thống phải được khai thác triệt để sao cho ngay cả điều kiện tài nguyên hạn chế vẫn có thể giải quyết những yêu cầu phức tạp. Một khía cạnh quan trọng của đảm bảo hiệu quả là duy trì đồng bộ trong toàn bộ hệ thống, không để các thiết bị tốc độ chậm trì hoãn hoạt động của toàn bộ hệ thống.

d) Tổng quát theo thời gian

HĐH phải có tính kế thừa, đồng thời có khả năng thích nghi với những thay đổi có thể có trong tương lai. Tính thừa kế là rất quan trọng ngay cả với các hệ điều hành thế hệ mới. Đối với việc nâng cấp, tính kế thừa là bắt buộc. Các thao tác, thông báo là không được thay đổi, hoặc nếu có thì không đáng kể và phải được hướng dẫn cụ thể khi chuyển từ phiên bản này sang phiên bản khác, bằng các phương tiện nhận biết của hệ thống. Đảm bảo tính kế thừa sẽ duy trì và phát triển đội ngũ người sử dụng-một nhân tố quan trọng để HĐH có thể tồn tại. Ngoài ra người sử dụng cũng rất quan tâm, liệu những kinh nghiệm và kiến thức của mình về HĐH hiện tại còn được sử dụng bao lâu nữa. Khả năng thích nghi với những thay đổi đòi hỏi HĐH phải được thiết kế theo một số nguyên tắc nhất định.

e) Thuận tiện

Hệ thống phải dễ dàng sử dụng, có nhiều mức hiệu quả khác nhau tuỳ theo kiến thức và kinh nghiệm người dùng. Hệ thống trợ giúp phong phú để người sử dụng có thể tự đào tạo ngay trong quá trình khai thác.

Trong một chừng mực nào đó, các tính chất trên mâu thuẫn lẫn nhau. Mỗi HĐH có một giải pháp trung hoà, ưu tiên hợp lý ở tính chất này hay tính chất khác.

1.7 Nguyên lý xây dựng chương trình hệ điều hành

a) Module

- HĐH phải được xây dựng từ các module độc lập nhưng có khả năng liên kết thành một hệ thống có thể thu gọn hoặc mở rộng tuỳ ý.

- Các module đồng cấp quan hệ với nhau thông qua dữ liệu vào và ra.

- Tồn tại quan hệ phân cấp khi các liên kết các module tạo thành những module có khả năng giải quết những vấn đề phức tạp hơn.

b) Nguyên tắc tương đối trong định vị

Các modul chương trình được viết theo địa chỉ tương đối kể từ đầu bộ nhớ. Khi thực hiện chúng mới được định vị tại vùng bộ nhớ cụ thể. Nguyên tắc này cho phép hệ thống sử dụng bộ nhớ một cách linh hoạt và hệ điều hành không bị phụ thuộc vào cấu hình bộ nhớ cụ thể.

c) Nguyên tắc Macroproccessor

Theo nguyên tắc này khi có nhiệm vụ cụ thể hệ thống sẽ xây dựng các phiếu yêu cầu, liệt kê các bước phải thực hiện và trên cơ sở đó xây dựng chương trình tương ứng, sau đó thực hiện chương trình nói trên. Mọi hệ điều hành đều phải xây dựng nguyên lý này trong đối thoại giữa người và máy trên ngôn ngữ vận hành. Dĩ nhiên độ sâu trong việc phân tích và xây dựng chương trình là khác nhau ở những hệ thống khác nhau. Chính nguyên tắc này đã làm cho quá trình đối thoại được linh hoạt mà không cần tới một chương trình dịch phức tạp.

d) Nguyên tắc khởi tạo trong cài đặt

Nguyên tắc Macroproccessor có thể áp dụng không những với từng nhiệm vụ mà còn với toàn bộ HĐH hoặc các thành phần của nó. Người sử dụng được cung cấp các bộ chương trình cài đặt. Chương trình cài đặt sẽ tạo phiên bản làm việc thích hợp với các tham số kỹ thuật hiện có, loại bỏ những modul không cần thiết để có một phiên bản tối ưu cả về cấu trúc lẫn phương thức hoạt động

e) Nguyên tắc lập chức năng

Mỗi công việc bao giờ cũng có nhiều cách thực hiện khác nhau với những tổ hợp modul khác nhau. Nguyên tắc này trước hết đảm bảo độ an toàn của hệ thống cao: vẫn có thể khai thác hệ thống bình thường ngay cả khi thiếu hoặc hỏng nhiều thành phần hệ thống. Ngoài ra, với nguyên tắc này người sử dụng sẽ thoải mái hơn khi giao tiếp với hệ thống: với một công việc, ai nhớ hoặc thích phương tiện nào thì sử dụng phương tiện đó. Như vậy người sử dụng khai thác được cả những hiệu ứng phụ của các modul chương trình. Đôi khi trong hệ thống tồn tại nhiều modul khác nhau cùng giải quyết một vấn đề, chẳng hạn có nhiều chương tình dịch cho một ngôn ngữ thuật toán

nào đó. Sự đa dạng đó cho phép người sử dụng chọn giải thuật tối ưu đối với bài toán của mình.

f) Nguyên tắc giá trị chuẩn

Một modun, câu lệnh…có thể có nhiều tham số. Việc nhớ hết các tham số: số lượng, ý nghĩa, quy cách…là vô cùng phức tạp và câu lệnh hoặc chương trình trở nên cồng cách một cách không cần thiết. Lối thoát ra khỏi tình trạng đó là chuẩn bị sẵn bộ giá trị các tham số ứng với trường hợp thường gặp nhất. Nếu trong câu lệnh hay lời gọi modul thiếu tham số nào thì hệ thống sẽ bổ sung bằng các giá trị quy ước trước. Nguyên tắc này thể hiện rất rõ trong các hệ thống cài đặt.

g) Nguyên tắc bảo vệ nhiều mức

Để đảm bảo an toàn hệ thống và an toàn dữ liệu, chương trình và dữ liệu phải được bảo vệ bằng nhiều khóa ở nhiều mức. Ví dụ đối với file, có thể bảo vệ ở mức cả đãi từ hoặc từng thư mục hay từng file riêng biệt, bảo vệ thường xuyên hay từng chế độ mở file…Việc bảo vệ nhiều mức đã làm giảm đáng kể các lỗi không cố ý. Nguyên tắc này được nghiên cứu áp dụng rất hiệu quả với thông tin ghi trong RAM.

1.8 Các hình thái giao tiếp

a) Hình thái dòng lệnh

Người sử dụng giao tiếp với hệ điều hành qua các dòng lệnh, mỗi lệnh có các tham số tương ứng

-Ưu điểm:

Dễ xây dựng và giảm công sức cho người xây dựng hệ thống.

Người sử dụng có thể đưa tham số của lệnh một cách chính xác theo mong muốn.

- Nhược điểm:

Tốc độ đưa lệnhvào chậm, người sử dụng phải nhớ các tham số.

Đối với các thao tác viên không có kinh nghiệm, thì hình thái này gây cản trở đến hiệu quả làm việc.

Hình thái giao tiếp này bị cản trở bởi hàng rào ngôn ngữ.

b) Hình thái thực đơn

Người sử dụng giao tiếp với hệ điều hành thông qua các thực đơn, các thực đơn thường có dạng trải xuống(popup). Mỗi thực đơn con tương ứng với một chức năng. Các tham số có thể được đưa vào thông qua giao tiếp với người sử dụng.

-Ưu điểm:

Hình thái này không yêu cầu nhớ lệnh

Người sử dụng có thể truy nhập vào thực đơn qua bàn phím hoặc qua chuột

- Nhược điểm:

Hình thái giao tiếp này bị cản trở bởi hàng rào ngôn ngữ.

Đôi khi các từ trên thực đơn không nêu bật được chức năng của nó.

c) Hình thái cửa sổ-biểu tượng

Người sử dụng giao tiếp với hệ điều hành thông qua các thanh công cụ và các biểu tượng. Mỗi biểu tượng tương ứng với một chức năng. Các tham số có thể được đưa vào thông qua giao tiếp với người sử dụmg.

-Ưu điểm:

Hình thái này không yêu cầu nhớ lệnh

Người sử dụng không bị hàng rào ngôn ngữ gây cản trở.

- Nhược điểm:

Có thể có rất nhiều biểu tượng do đó gây sự nhập nhằng về chức năng.

Không thuận lợi khi thao tác bằng bàn phím.

d) Hình thái kết hợp

HĐH thường kết hợp nhiều hình thái giao tiếp để tạo ra tính thân thiện với người sử dụng. Ví dụ: việc kết hợp thực đơn với các biểu tượng, hoặc kết hợp giữa các biểu tượng với các từ gợi ý.

Hình thái giao tiếp kết hợp này khắc phục được các nhược điểm của các hình thái giao tiếp đơn lẻ.

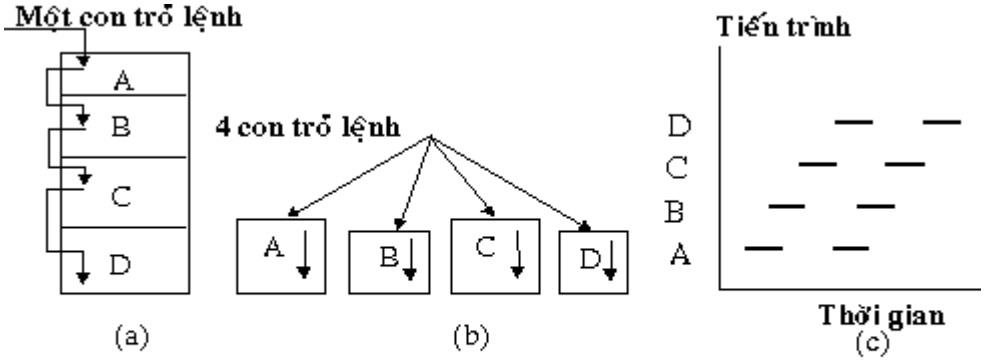
CHƯƠNG 2 QUẢN LÝ TIẾN TRÌNH

2.1 Tổng quan về tiến trình

2.1.1 Tiến trình (Process) và mô hình đa tiến trình (Multiprocess)

Tiến trình là một chương trình đang xử lý, sỡ hữu một con trỏ lệnh, tập các thanh ghi và các biến. Để hoàn thành công việc của mình, một tiến trình có thể cần đến một số tài nguyên – như CPU, bộ nhớ chính, các tập tin và thiết bị nhập/xuất.

Cần phân biệt hai khái niệm *chương trình* và *tiến trình*. Một chương trình là một thực thể thụ động, chứa đựng các chỉ thị điều khiển máy tính để tiến hành một tác vụ nào đó ; khi cho thực hiện các chỉ thị này, chương trình chuyển thành tiến trình, là một thực thể hoạt động, với con trỏ lệnh xác định chỉ thị kế tiếp sẽ thi hành, kèm theo tập các tài nguyên phục vụ cho hoạt động của tiến trình.



Hình 2.1 (a) Đa chương với 4 chương trình
(b) Mô hình khái niệm với 4 chương trình độc lập
(c) Tại một thời điểm chỉ có một chương trình hoạt động

Mô hình đa tiến trình

Mô hình đa tiến trình là mô hình mà trong hệ thống có nhiều tiến trình hoạt động đồng thời với nhau. Về nguyên tắc để thực hiện điều này thì hệ thống phải có nhiều processor, mỗi processor thực hiện một tiến trình, nhưng mong muốn của hệ điều hành cũng nhơ của người sử dụng là thực hiện sự đa chương trên hệ thống chỉ có một processor. Để thực hiện điều này hệ điều hành đã sử dụng mô hình đa tiến trình để tạo ra sự song song giả hay tạo ra các processor logic từ các processor vật lý. Các processor có thể hoạt động song song với nhau, mỗi processor logic chịu trách nhiệm thực hiện một tiến trình.

Hệ điều hành cấp processor cho một tiến trình trong số các tiến trình ở trạng thái sẵn sàng để tiến trình này hoạt động, sau một khoảng thời gian nào đó, hệ điều hành thu hồi processor của tiến trình này để cấp cho một tiến trình sẵn sàng khác, sau

đó hệ điều hành lại thu hồi processor từ tiến trình mà nó vừa cấp để cấp cho tiến trình khác, có thể là tiến trình trước đây bị hệ điều hành thu hồi processor khi nó chưa kết thúc, và cứ như thế cho đến khi tất cả các tiến trình mà hệ điều hành khởi tạo đều hoạt động và kết thúc được. Điều đáng chú ý trong mô hình đa tiến trình này là khoảng thời gian chuyển processor từ tiến trình này sang tiến trình khác hay khoảng thời gian giữa hai lần được cấp processor của một tiến trình là rất nhỏ nên các tiến trình có cảm giác luôn sở hữu processor (logic) hay hệ thống có cảm giác các tiến trình hoạt động song song nhau. Hiện tượng này được gọi là hiện tượng song song giả.

Với mô hình đa tiến trình hệ thống có được 2 điều lợi:

- Tiết kiệm được bộ nhớ: vì không phải nạp tất cả chương trình vào bộ nhớ mà chỉ nạp các tiến trình cần thiết nhất, sau đó tùy theo yêu cầu mà có thể nạp tiếp các tiến trình khác.

- Cho phép các chương trình hoạt động song song nên tốc độ xử lý của toàn hệ thống tăng lên và khai thác tối đa thời gian xử lý của processor.

Nhu cầu xử lý đồng hành

Một tiến trình khi thi hành trải qua nhiều chu kỳ xử lý (sử dụng CPU) và chu kỳ nhập xuất (sử dụng các thiết bị nhập xuất) đan xen nhau

+ Tăng hiệu suất sử dụng CPU và các tài nguyên khác.

Một tiến trình khi thi hành, thực hiện thao tác nhập xuất thì CPU được sử dụng cho tiến trình khác.

+ Tăng tốc xử lý

Với các bài toán có bản chất xử lý song song thì chia ra thành nhiều phần có khả năng hoạt động độc lập.

2.1.2 Tiểu trình (Thread) và mô hình đa tiểu trình (Multithread)

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

Ví dụ : Một server quản lý tập tin thỉnh thoảng phải tự khóa để chờ các thao tác truy xuất đĩa hoàn tất.Nếu server có nhiều dòng xử lý, hệ thống có thể xử lý các yêu cầu mới trong khi một dòng xử lý bị khoá. Như vậy việc thực hiện chương trình sẽ có hiệu quả hơn. Điều này không thể đạt được bằng cách tạo hai tiến trình server riêng biệt vì cần phải chia sẻ cùng một vùng đệm, do vậy bắt buộc phải chia sẻ không gian địa chỉ.

Tiểu trình là một đơn vị xử lý cơ bản trong hệ thống, nó hoàn toàn tương tự như tiến trình. Mỗi tiểu trình xử lý tuần tự các chỉ thị máy của nó, sỡ hữu một con trỏ lệnh, tập các thanh ghi và một không gian stack riêng.

Các tiểu trình chia sẻ thời gian xử lý của processor giống như cách của tiến trình, nhờ đó mà các tiểu trình có thể hoạt động song song (giả) với nhau.

Các tiến trình tạo thành những thực thể độc lập. Mỗi tiến trình có một tập tài nguyên và một môi trường riêng (một con trỏ lệnh, một Stack , các thanh ghi và không gian địa chỉ). Các tiến trình hoàn toàn độc lập với nhau, chỉ có thể liên lạc thông qua các cơ chế thông tin giữa các tiến trình mà hệ điều hành cung cấp.

Ngược lại, một tiến trình đơn có thể bao gồm nhiều tiểu trình, các tiểu trình trong cùng một tiến trình lại chia sẻ một không gian địa chỉ chung , nhờ đó các tiểu trình có thể chia sẻ các biến toàn cục của tiến trình và có thể truy xuất lên các vùng nhớ stack của nhau. Cấu trúc này không đề nghị một cơ chế bảo vệ nào, và điều này cũng không thật cần thiết vì các tiểu trình trong cùng một tiến trình thuộc về cùng một sỡ hữu chủ đã tạo ra chúng trong ý định cho phép chúng hợp tác với nhau.

2.1.3 Phân loại tiến trình

Các tiến trình trong hệ thống có thể chia thành 2 loại: tiến trình tuần tự và tiến trình song song.

- Tiến trình tuần tự:

Hai hay nhiều tiến trình gọi là tuần tự khi điểm kết thúc của tiến trình này là sự bắt đầu của tiến trình khác.

- Tiến trình song song

Điểm bắt đầu của tiến trình này nằm giữa điểm bắt đầu và kết thúc của tiến trình khác. Tiến trình song song được chia thành nhiều loại:

+ Tiến trình song song có quan hệ thông tin: trong quá trình hoạt động các tiến trình thường trao đổi thông tin với nhau, trong một số tường hợp tiến trình gởi thông báo cần phải nhận được tín hiệu từ tiến trình nhận để tiếp tục, điều này dễ dẫn đến bế tắc khi tiến trình nhận tín hiệu không ở trong trạng thái nhận hay tiến trình gửi không ở trong trạng thái nhận thông báo trả lời.

+ Tiến trình song song độc lập: là các tiến trình hoạt động song song nhưng không có quan hệ thông tin với nhau, trong trường hợp này hệ điều hành phải thiết lập cơ chế bảo vệ dữ liệu của các tiến trình, và cấp phát tài nguyên cho các tiến trình một cách hợp lý.

-Tiến trình song song phân cấp: Trong quá trình hoạt động một tiến trình có thể khởi tạo các tiến trình khác hoạt động song song với nó, tiến trình khởi tạo được gọi là tiến trình cha, tiến trình được tạo gọi là tiến trình con. Trong mô hình này hệ điều hành phải giải quyết vấn đề cấp phát tài nguyên cho các tiến trình con. Hệ điều hành đưa ra hai mô hình quản lý tài nguyên: Thứ nhất, mô hình tập trung, hệ điều hành chịu trách nhiệm phân phối tài nguyên cho tất cả các tiến trình trong hệ thống. Thứ hai, mô hình phân tán hệ điều hành cho phép tiến trình con nhận tài nguyên từ tiến trình cha, tức là tiến trình khởi tạo có nhiệm vụ nhận tài nguyên từ hệ điều hành để cấp phát cho các tiến trình mà nó tạo ra, và nó có nhiệm vụ thu hồi lại tài nguyên đã cấp phát trả về cho hệ điều hành trước khi kết thúc.

-Tiến trình song song đồng mức: là các tiến trình hoạt động song song sử dụng chung tài nguyên theo nguyên tắc lần lượt, mỗi tiến trình sau một khoảng thời gian chiếm giữ tài nguyên phải tự động trả lại tài nguyên cho tiến trình kia.

Trong tài liệu này chúng ta chỉ khảo sát sự hoạt động của các tiến trình song song trên các hệ thống đơn bộ xử lý.

Đối với người sử dụng thì trong hệ thống chỉ có 2 nhóm tiến trình. Thứ nhất là các tiến trình của hệ điều hành. Thứ hai, là các tiến trình của chương trình người sử dụng. Các tiến trình của hệ điều hành hoạt động trong chế độ đặc quyền, nhờ đó mà nó có thể truy xuất vào vào các vùng dữ liệu được bảo vệ của hệ thống. Trong khi đó các tiến trình của chương trình người sử dụng hoạt động trong chế độ không đặc quyền, nên nó không thể truy xuất vào hệ thống thông qua các tiến trình của hệ điều hành bằng cách thực hiện một lời gọi hệ thống.

2.1.4. Các trạng thái của tiến trình

Trạng thái của tiến trình tại một thời điểm được xác định bởi hoạt động hiện thời của tiến trình tại thời điểm đó.

Tiến trình hai trạng thái: Not Running và Running (Một số ít hệ điều hành cho phép)

Tiến trình ba trạng thái: Ready, Running, Blocked

Tại một thời điểm, một tiến trình có thể nhận một trong các trạng thái sau đây :

- New (Mới tạo) : tiến trình đang được tạo lập.

- Ready (Sẵn sàng) : tiến trình chờ được cấp phát CPU để xử lý.

Ngaysau khi khởi tạo tiến trình, đưa tiến trình vào hệ thống và cấp phát đầy đủ tài nguyên (trừ processor) cho tiến trình, hệ điều hành đưa tiến trình vào trạng thái ready

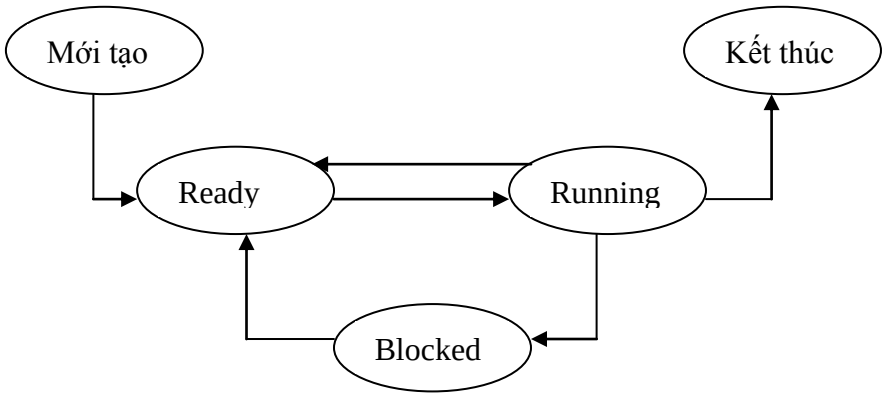
- Running (thực hiện) : các chỉ thị của tiến trình đang được xử lý bởi processor.

- Blocked (khóa) : tiến trình đang chờ được cấp phát thêm tài nguyên, hay chờ một sự kiện nào đó xảy ra hay một quá trình vào ra kết thúc.

- Kết thúc : tiến trình hoàn tất xử lý.

Tại một thời điểm, chỉ có một tiến trình có thể nhận trạng thái *running* trên một bộ xử lý bất kỳ. Trong khi đó, nhiều tiến trình có thể ở trạng thái *blocked* hay *ready*.

Sơ đồ cung chuyển đổi trạng thái:



Hình 2.3

1) Tiến trình được khởi tạo, được đưa vào hệ thống, được cấp phát đầy đủ tài nguyên chỉ thiếu processor

2) Tiến trình được cấp processor để bắt đầu xử lý

3) Tiến trình hoàn thành xử lý và kết thúc

4) Tiến trình bị bộ điều phối tiến trình thu hồi processor để cấp phát cho tiến trình khác(hết thời gian được quyền sử dụng processor mà bộ điều phối dành cho tiến trình, có tiến trình mới có độ ưu tiên cao hơn)

5) Tiến trình đang chờ một sự kiện nào đó xảy ra hay đang chờ một thao tác vào/ra kết thúc, hay tài nguyên mà tiến trình yêu cầu chưa được hệ điều hành đáp ứng.

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

Tiến trình 4 trạng thái: Ready, Running, Blocked, Suspend(trạng thái của một tiến trình khi nó đang được lưu trữ trên bộ nhớ phụ)

Tiến trình 5 trạng thái: Ready, Running, Blocked, Blocked-Suspend (chuyển từ trạng thái blocked sang bộ nhớ phụ), Ready-Suspend (chuyển từ trạng thái blocked sang bộ nhớ phụ)

2.1.5. Cấu trúc dữ liệu của khối quản lý tiến trình

Hệ điều hành quản lý các tiến trình trong hệ thống thông qua khối quản lý tiến trình (process control block -PCB). PCB là một vùng nhớ lưu trữ các thông tin mô tả cho tiến trình, với các thành phần chủ yếu bao gồm :

- Định danh của tiến trình (1) : giúp phân biệt các tiến trình khác trong hệ thống

- Trạng thái tiến trình (2): xác định hoạt động hiện hành của tiến trình.

- Ngữ cảnh của tiến trình (3): mô tả các tài nguyên tiến trình đang trong quá trình, hoặc để phục vụ cho hoạt động hiện tại, hoặc để làm cơ sở phục hồi hoạt động cho tiến trình, bao gồm các thông tin về:

Trạng thái CPU: bao gồm nội dung các thanh ghi, quan trọng nhất là con trỏ lệnh IP lưu trữ địa chỉ câu lệnh kế tiếp tiến trình sẽ xử lý. Các thông tin này cần được lưu trữ khi xảy ra một ngắt, nhằm có thể cho phép phục hồi hoạt động của tiến trình đúng như trước khi bị ngắt.

Bộ xử lý: dùng cho máy có cấu hình nhiều CPU, xác định số hiệu CPU mà tiến trình đang sử dụng.

Bộ nhớ chính: danh sách các khối nhớ được cấp cho tiến trình.

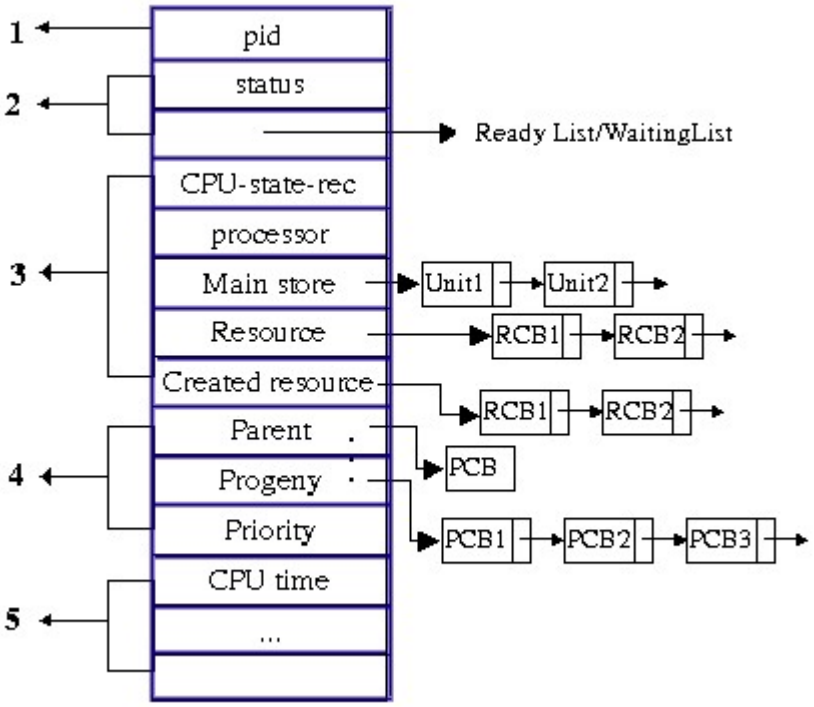
Tài nguyên sử dụng: danh sách các tài mguyên hệ thống mà tiến trình đang sử dụng.

Tài nguyên tạo lập: danh sách các tài nguyên được tiến trình tạo lập.

-Thông tin giao tiếp (4): phản ánh các thông tin về quan hệ của tiến trình với các tiến trình khác trong hệ thống :

Tiến trình cha: tiến trình tạo lập tiến trình này .

Tiến trình con: các tiến trình do tiến trình này tạo lập .



Hình 2.4 Khối mô tả tiến trình

Độ ưu tiên : giúp bộ điều phối có thông tin để lựa chọn tiến trình được cấp CPU.

-Thông tin thống kê (5): đây là những thông tin thống kê về hoạt động của tiến trình, như thời gian đã sử dụng CPU,thời gian chờ. Các thông tin này có thể có ích cho công việc đánh giá tình hình hệ thống và dự đoán các tình huống tương lai.

2.1.6. Các thao tác điều khiển tiến trình

Hệ điều hành cung cấp các thao tác chủ yếu sau đây trên một tiến trình :

-Tạo lập tiến trình (create)

-Kết thúc tiến trình (destroy)

-Tạm dừng tiến trình (suspend)

-Tái kích hoạt tiến trình (resume)

-Thay đổi độ ưu tiên tiến trình

a) Tạo lập tiến trình

Một tiến trình được tạo lập khi:

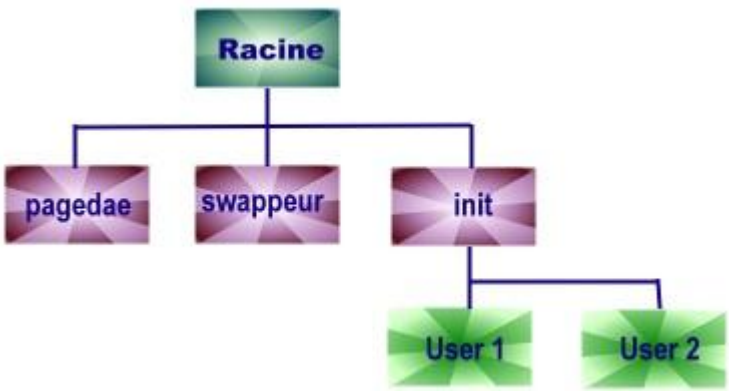
-Người sử dụng chạy một chương trình.

-Hệ điều hành thực hiện một số dịch vụ.

-Tiến trình cha sinh tiến trình con

-Một người truy nhập hệ thống.

Trong quá trình xử lý, một tiến trình có thể tạo lập nhiều tiến trình mới bằng cách sử dụng một lời gọi hệ thống tương ứng. Tiến trình gọi lời gọi hệ thống để tạo tiến trình mới sẽ được gọi là tiến trình *cha*, tiến trình được tạo gọi là tiến trình *con*. Mỗi tiến trình con đến lượt nó lại có thể tạo các tiến trình mới…quá trình này tiếp tục



sẽ tạo ra một *cây tiến trình*. Ví dụ trong UNIX: lời gọi hệ thống là fork

Hình 2.5 Một cây tiến trình trong hệ thống UNIX

Các công việc hệ điều hành cần thực hiện khi tạo lập tiến trình bao gồm :

- Định danh cho tiến trình mới phát sinh

-Đưa tiến trình vào danh sách quản lý của hệ thống

- Xác định độ ưu tiên cho tiến trình

- Tạo PCB cho tiến trình

-Cấp phát các tài nguyên ban đầu cho tiến trình

Khi một tiến trình tạo lập một tiến trình con, tiến trình con có thể sẽ được hệ điều hành trực tiếp cấp phát tài nguyên hoặc được tiến trình cha cho thừa hưởng một số tài nguyên ban đầu.

Khi một tiến trình tạo tiến trình mới, tiến trình ban đầu có thể xử lý theo một trong hai khả năng sau :

Tiến trình cha tiếp tục xử lý đồng hành với tiến trình con. Ví dụ UNIX

Tiến trình cha chờ đến khi một tiến trình con nào đó, hoặc tất cả các tiến trình con kết thúc xử lý. Ví dụ MSDOS

Các hệ điều hành khác nhau có thể chọn lựa các cài đặt khác nhau để thực hiện thao tác tạo lập một tiến trình.

b). Kết thúc tiến trình

Một tiến trình kết thúc khi:

- Tiến trình hoàn tất công việc.

- Tiến trình kết thúc khi vượt quá thời hạn

- Tiến trình kết thúc khi sử dụng quá tài nguyên quy định

- Tiến trình kết thúc khi bộ nhớ không đủ.

- Tiến trình vi phạm một số quy định

- Tiến trình mắc một số lỗi về phép toán

- Thiết bị ngoại vi bị lỗi

-Khi các lệnh bị sai

- Khi có quyền ưu tiên

- Dữ liệu sai

- HĐH dừng một số tiến trình

Một tiến trình kết thúc xử lý khi nó hoàn tất chỉ thị cuối cùng và sử dụng một lời gọi hệ thống để yêu cầu hệ điều hành hủy bỏ nó. Đôi khi một tiến trình có thể kết thúc xử lý của một tiến trình khác bằng một lời gọi hệ thống tương ứng. Khi một tiến trình kết thúc, hệ điều hành thực hiện các công việc :

-Thu hồi các tài nguyên hệ thống đã cấp phát cho tiến trình

-Hủy tiến trình khỏi tất cả các danh sách quản lý của hệ thống

- Hủy bỏ PCB của tiến trình

Hầu hết các hệ điều hành không cho phép các tiến trình con tiếp tục tồn tại nếu tiến trình cha đã kết thúc. Trong những hệ thống như thế, hệ điều hành sẽ tự động phát sinh một loạt các thao tác kết thúc tiến trình con.

2.1.7 Cấp phát tài nguyên cho tiến trình

Khi có nhiều người sử dụng đồng thời làm việc trong hệ thống, hệ điều hành cần phải cấp phát các tài nguyên theo yêu cầu cho mỗi người sử dụng. Do tài nguyên hệ thống thường rất giới hạn và có khi không thể chia sẻ, nên hiếm khi tất cả các yêu cầu tài nguyên đồng thời đều được thỏa mãn. Vì thế cần phải nghiên cứu một phương pháp để chia sẻ một số tài nguyên hữu hạn giữa nhiều tiến trình người dùng đồng thời. Hệ điều hành quản lý nhiều loại tài nguyên khác nhau (CPU, bộ nhớ chính, các thiết bị ngoại vi …), với mỗi loại cần có một cơ chế cấp phát và các chiến lược cấp phát hiệu qủa. Mỗi tài nguyên được biểu diễn thông qua một cấu trúc dữ liệu, khác nhau về chi tiết cho từng loại tài nguyên, nhưng cơ bản chứa đựng các thông tin sau :

- Định danh tài nguyên

- Trạng thái tài nguyên : đây là các thông tin mô tả chi tiết trạng thái tài nguyên : phần nào của tài nguyên đã cấp phát cho tiến trình, phần nào còn có thể sử dụng ?

- Hàng đợi trên một tài nguyên : danh sách các tiến trình đang chờ được cấp phát tài nguyên tương ứng.

- Bộ cấp phát : là đoạn code đảm nhiệm việc cấp phát một tài nguyên đặc thù. Một số tài nguyên đòi hỏi các giải thuật đặc biệt (như CPU, bộ nhớ chính, hệ thống tập tin), trong khi những tài nguyên khác (như các thiết bị nhập/xuất) có thể cần các giải thuật cấp phát và giải phóng tổng quát hơn.

Hình 2.6 Khối quản lý tài nguyên

Các mục tiêu của kỹ thuật cấp phát :

Bảo đảm một số lượng hợp lệ các tiến trình truy xuất đồng thời đến các tài nguyên không chia sẻ được.

Cấp phát tài nguyên cho tiến trình có yêu cầu trong một khoảng thời gian trì hoãn có thể chấp nhận được.

Tối ưu hóa sự sử dụng tài nguyên.

Để có thể thõa mãn các mục tiêu kể trên, cần phải giải quyết các vấn đề nảy sinh khi có nhiều tiến trình đồng thời yêu cầu một tài nguyên không thể chia sẻ.

2.2. Điều phối tiến trình

Trong môi trường đa chương, có thể xảy ra tình huống nhiều tiến trình đồng thời sẵn sàng để xử lý. Mục tiêu của các hệ phân chia thời gian (time-sharing) là chuyển đổi CPU qua lại giữa các tiến trình một cách thường xuyên để nhiều người sử dụng có thể tương tác cùng lúc với từng chương trình trong quá trình xử lý.

Bộ điều phối tiến trình có nhiệm vụ xem xét và quyết định khi nào thì dừng tiến trình hiện tại để thu hồi processor và chuyển processor cho tiến trình khác, và khi có được processor thì chọn tiến trình nào trong số các tiến trình ở trạng thái ready để cấp processor cho nó.

2.2.1. Mục tiêu điều phối

Bộ điều phối không cung cấp cơ chế, mà đưa ra các quyết định. Các hệ điều hành xây dựng nhiều chiến lược khác nhau để thực hiện việc điều phối, nhưng tựu chung cần đạt được các mục tiêu sau :

a) Sự công bằng (Fairness) :

Các tiến trình chia sẻ thời gian xử lý của CPU một cách công bằng, không có tiến trình nào phải chờ đợi vô hạn để được cấp phát CPU

b) Tính hiệu qủa (Efficiency) **:**Hệ thống phải tận dụng được CPU nhiều nhất có thể.Trong hệ thống thực, nó nên nằm trong khoảng từ 40% (cho hệ thống được nạp tải nhẹ) tới 90% (cho hệ thống được nạp tải nặng).

c) Thời gian đáp ứng hợp lý (Response time) :

Cực tiểu hoá thời gian hồi đáp cho các tương tác của người sử dụng

d) Thời gian lưu lại trong hệ thống hợp lý (Turnaround Time) :

Cực tiểu hóa thời gian hoàn tất các tác vụ xử lý theo lô.

e) Thông lượng tối đa (Throughput) :

Cực đại hóa số công việc được xử lý trong một khoảng thời gian.

Tuy nhiên thường không thể thỏa mãn tất cả các mục tiêu kể trên vì bản thân chúng có sự mâu thuẫn với nhau mà chỉ có thể dung hòa chúng ở mức độ nào đó.

2.2.2 Điều phối độc quyền và điều phối không độc quyền (preemptive/nopreemptive)

Thuật toán điều phối cần xem xét và quyết định thời điểm chuyển đổi CPU giữa các tiến trình. Hệ điều hành có thể thực hiện cơ chế điều phối theo nguyên lý *độc quyền* hoặc *không độc quyền*.

Điều phối độc quyền : Nguyên lý điều phối *độc quyền* cho phép một tiến trình khi nhận được CPU sẽ có quyền độc chiếm CPU đến khi hoàn tất xử lý hoặc tự nguyện giải phóng CPU. Khi đó quyết định điều phối CPU sẽ xảy ra trong các tình huống sau:

- Khi tiến trình chuyển từ trạng thái đang xử lý(running) sang trạng thái bị khóa blocked (ví dụ chờ một thao tác nhập xuất hay chờ một tiến trình con kết thúc…).

- Khi tiến trình kết thúc.

Các giải thuật độc quyền thường đơn giản và dễ cài đặt. Tuy nhiên chúng thường không thích hợp với các hệ thống tổng quát nhiều người dùng, vì nếu cho phép một tiến trình có quyền xử lý bao lâu tùy ý, có nghĩa là tiến trình này có thể giữ CPU một thời gian không xác định, có thể ngăn cản những tiến trình còn lại trong hệ thống có một cơ hội để xử lý.

Điều phối không độc quyền :

Bộ phận điều phối tiến trình có thể tạm dừng tiến trình đang xử lý để thu hồi processor của nó, để cấp cho tiến trình khác, sao cho phù hợp với công tác điều phối hiện tại

Các quyết định điều phối xảy ra khi :

- Khi tiến trình chuyển từ trạng thái đang xử lý (running) sang trạng thái bị khóa blocked (ví dụ chờ một thao tác nhập xuất hay chờ một tiến trình con kết thúc…).

- Khi tiến trình chuyển từ trạng thái đang xử lý (running) sang trạng thái ready (ví dụ xảy ra một ngắt).

- Khi tiến trình chuyển từ trạng thái chờ (blocked) sang trạng thái ready (ví dụ một thao tác nhập/xuất hoàn tất).

- Khi tiến trình kết thúc.

Các thuật toán điều phối theo nguyên tắc không độc quyền ngăn cản được tình trạng một tiến trình độc chiếm CPU, nhưng việc tạm dừng một tiến trình có thể dẫn đến các mâu thuẫn trong truy xuất, đòi hỏi phải sử dụng một phương pháp đồng bộ hóa thích hợp để giải quyết.

Trong các hệ thống sử dụng nguyên lý điều phối độc quyền có thể xảy ra tình trạng các tác vụ cần thời gian xử lý ngắn phải chờ tác vụ xử lý với thời gian rất dài hoàn tất! Nguyên lý điều phối độc quyền thường chỉ thích hợp với các hệ xử lý theo lô.

Đối với các hệ thống tương tác(time sharing), các hệ thời gian thực (real time),cần phải sử dụng nguyên lý điều phối không độc quyền để các tiến trình quan trọng có cơ hội hồi đáp kịp thời. Tuy nhiên thực hiện điều phối theo nguyên lý không độc quyền đòi hỏi những cơ chế phức tạp trong việc phân định độ ưu tiên, và phát sinh thêm chi phí khi chuyển đổi CPU qua lại giữa các tiến trình.

2.2.3. Các danh sách sử dụng trong quá trình điều phối.

Hệ điều hành sử dụng hai loại danh sách để thực hiện điều phối các tiến trình là *danh sách sẵn sàng (ready list)* và *danh sách chờ đợi(waiting list)*.

Danh sách sẵn sàng (Reay List): dùng để chứa các tiến trình ở trạng thái sẵn sàng.

Danh sách đợi (Waiting List): dùng để chứa các tiến trình đang đợi để được bổ sung vào danh sách sẵn sàng.

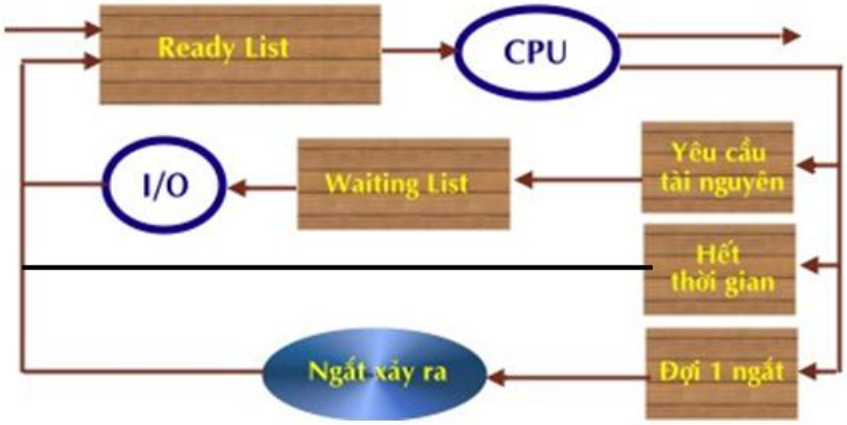
Hệ điều hành chỉ sử dụng một danh sách sẵn sàng cho toàn hệ thống, nhưng mỗi một tài nguyên (thiết bị ngoại vi) có một danh sách chờ đợi riêng bao gồm các tiến trình đang chờ được cấp phát tài nguyên đó.

Quá trình xử lý của một tiến trình trải qua những chu kỳ chuyển đổi qua lại giữa danh sách sẵn sàng và danh sách chờ đợi. Sơ đồ dưới đây mô tả sự điều phối các tiến trình dựa trên các danh sách của hệ thống.

Thoạt đầu tiến trình mới được đặt trong danh sách các tiến trình sẵn sàng (ready list), nó sẽ đợi trong danh sách này cho đến khi được chọn để cấp phát CPU và bắt đầu xử lý. Sau đó có thể xảy ra một trong các tình huống sau :

Tiến trình phát sinh một yêu cầu một tài nguyên mà hệ thống chưa thể đáp ứng, khi đó tiến trình sẽ được chuyển sang danh sách các tiến trình đang chờ tài nguyên tương ứng.

Tiến trình có thể bị bắt buộc tạm dừng xử lý do một ngắt xảy ra, khi đó tiến trình được đưa trở lại vào danh sách sẵn sàng để chờ được cấp CPU cho lượt tiếp theo.



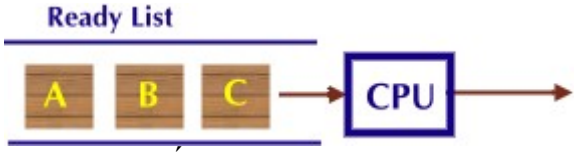
Hình 2.8 Sơ đồ chuyển đổi giữa các danh sách điều phối

Trong trường hợp đầu tiên, tiến trình cuối cùng sẽ chuyển từ trạng thái blocked sang trạng thái ready và lại được đưa trở vào danh sách sẵn sàng. Tiến trình lặp lại chu kỳ này cho đến khi hoàn tất tác vụ thì được hệ thống hủy bỏ khỏi mọi danh sách điều phối.

2.2.4. Các chiến lược điều phối

a). Chiến lược FIFO(First In First Out)

Trong chiến lược này, khi processor rỗi thì hệ điều hành sẽ cấp processor cho tiến trình đầu tiên trong readylist, đây là tiến trình được chuyển vào readylist sớm nhất, có thể là tiến trình được đưa vào hệ thống sớm nhất. FiFo được sử dụng trong điều phối độc quyền nên khi tiến trình được cấp processor nó sẽ sở hữu processor cho đến khi kết thúc xử lý hay phải đợi một thao tác vào/ra hoàn thành, khi đó tiến trình



chủ động trả lại processor cho hệ thống.

Ví dụ :

Tiến trình	Thời điểm vào RL	Thời gian xử lý
P1	0	24
P2	1	3
P3	2	3

Thứ tự cấp phát CPU cho các tiến trình là :

P1	P2	P3
0	‘24	27 30

thời gian chờ đợi được xử lý là 0 đối với P1, (24 -1) với P2 và (24+3-2) với P3. Thời gian chờ trung bình là (0+23+25)/3 = 16 milisecondes.

Thời gian lưu:

P1: 24, P2: 26, P3: 28

Thời gian lưu trung bình là: 26.

Ưu điểm: đơn giản, dễ cài đặt.

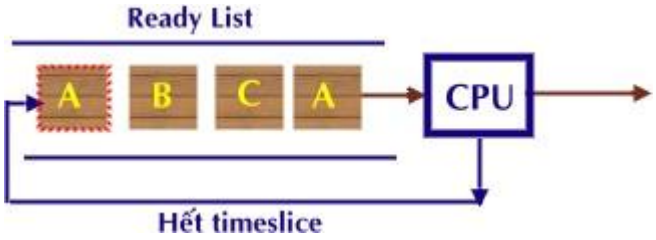
Một số hạn chế:

- Thời gian chờ đợi trung bình lớn nên không phù hợp với các hệ thống chia sẻ thời gian.
- Khả năng tương tác kém khi nó được áp dụng trên các hệ thống uniprocessor
- Nếu các tiến trình ở đầu Readylist cần nhiều thời gian của processor thì các tiến trình ở cuối readylist sẽ phải chờ đợi lâu mới được cấp processor

b). Chiến lược phân phối xoay vòng (RR: Round Robin)

Nguyên tắc : Danh sách sẵn sàng được xử lý như một danh sách vòng, bộ điều phối lần lượt cấp phát cho từng tiến trình trong danh sách một khoảng thời gian tối đa sử dụng CPU cho trước gọi là *quantum*. Tiến trình ở đầu readylist thì được cấp phát CPU trước. Đây là một giải thuật điều phối không độc quyền : khi một tiến trình sử dụng CPU đến hết thời gian quantum dành cho nó, hệ điều hành thu hồi CPU và cấp cho tiến trình kế tiếp trong danh sách. Nếu tiến trình bị khóa hay kết thúc trước khi sử dụng hết thời gian quantum, hệ điều hành cũng lập tức cấp phát CPU cho tiến trình khác. Khi tiến trình tiêu thụ hết thời gian CPU dành cho nó mà chưa hoàn tất, tiến trình được đưa trở lại vào cuối danh sách sẵn sàng để đợi được cấp CPU trong lượt kế tiếp.

Ví dụ :



Hình 2.10 Điều phối Round Robin

Tiến trình	Thời điểm vào RL	Thời gian xử lý
P1	0	24
P2	1	3
P3	2	3

Nếu sử dụng quantum là 4 milisecondes, thứ tự cấp phát CPU sẽ là

P1	P2	P3	P1	P1	P1	P1	P1
0	‘4	7	10	14	18	22	26 30

Thời gian chờ:

P1: 6, P2:3, P3:5

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

Đảm bảo tính công bằng.

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

Hiệu quả điều phối phụ thuộc vào độ dài quantum

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

c) Điều phối với độ ưu tiên

Mỗi tiến trình được gán cho một độ ưu tiên nhất định

Độ ưu tiên của tiến trình có thể được phát sinh tự động bởi hệ thống hoặc được gán tường minh trong chương trình của người sử dụng. Độ ưu tiên của tiến trình có 2 loại: thứ nhất là độ ưu tiên tĩnh là độ ưu tiên gán trước cho tiến trình và không thay đổi trong suốt thời gian sống của tiến trình. Thứ hai độ ưu tiện động là độ ưu tiên được gán cho tiến trình trong quá tình hoạt động của nó, hệ điều hành sẽ gán lại độ ưu tiên cho tiến trình khi môi trường xử lý tiến trình thay đổi.

Tại thời điểm điều phối, tiến trình có độ ưu tiên cao nhất sẽ được chọn để cấp phát CPU. Các tiến trình có độ ưu tiên cao nhất bằng nhau thì tiến trình nào đến trước thì sẽ được cấp trước. Giải thuật điều phối với độ ưu tiên có thể theo nguyên tắc độc quyền hay không độc quyền. Khi một tiến trình được đưa vào danh sách các tiến trình sẵn sàng, độ ưu tiên của nó được so sánh với độ ưu tiên của tiến trình hiện hành đang xử lý. Giải thuật điều phối với độ ưu tiên và không độc quyền sẽ thu hồi CPU từ tiến trình hiện hành để cấp phát cho tiến trình mới nếu độ ưu tiên của tiến trình này cao hơn tiến trình hiện hành. Một giải thuật độc quyền sẽ chỉ đơn giản chèn tiến trình mới vào danh sách sẵn sàng tại vị trí thích hợp, và tiến trình hiện hành vẫn tiếp tục xử lý hết thời gian dành cho nó.

Ready list luôn được xếp theo thứ tự giảm dần của độ ưu tiên kể từ đầu danh sách. Điều này có nghĩa là tiến trình được chọn để cấp processor là tiến trình ở đầu readylist.

Ví dụ : (độ ưu tiên 1 > độ ưu tiên 2> độ ưu tiên 3)

Tiến trình	Thời điểm vào RL	Độ ưu tiên	Thời gian xử lý
P1	0	3	24
P2	1	1	3
P3	2	2	3

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

P1	P2	P3
0	‘24	27 30

Thời gian chờ trung bình là 16

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

P1	P2	P3	P1
0	‘1	4	7 30

Thời gian chờ:

P1: 6, P2:0, P3:2

Thời gian chờ trung bình là 8/3

Thảo luận : Tình trạng 'đói CPU' (starvation) là một vấn đề chính yếu của các giải thuật sử dụng độ ưu tiên. Các giải thuật này có thể để các tiến trình có độ ưu tiên thấp chờ đợi CPU vô hạn ! Để ngăn cản các tiến trình có độ ưu tiên cao chiếm dụng CPU vô thời hạn, bộ điều phối sẽ giảm dần độ ưu tiên của các tiến trình này sau mỗi ngắt đồng hồ. Nếu độ ưu tiên của tiến trình này giảm xuống thấp hơn tiến trình có độ ưu tiên cao thứ nhì, sẽ xảy ra sự chuyển đổi quyền sử dụng CPU.Quá trình này gọi là sự 'lão hóa' *(aging)* tiến trình.

d). Chiến lược công việc ngắn nhất (Shortest-job-first SJF)

Nguyên tắc : Đây là một trường hợp đặc biệt của giải thuật điều phối với độ ưu tiên. Trong giải thuật này, độ ưu tiên *p* được gán cho mỗi tiến trình là nghịch đảo của thời gian xử lý *t* mà tiến trình yêu cầu : $p = 1/t$. Khi CPU được tự do, nó sẽ được cấp phát cho tiến trình yêu cầu ít thời gian nhất để kết thúc- tiến trình ngắn nhất. Giải thuật này cũng có thể độc quyền hay không độc quyền. Sự chọn lựa xảy ra khi có một tiến trình mới được đưa vào danh sách sẵn sàng trong khi một tiến trình khác đang xử lý. Tiến trình mới có thể sỡ hữu một yêu cầu thời gian sử dụng CPU cho lần tiếp theo (CPU-burst) ngắn hơn thời gian còn lại mà tiến trình hiện hành cần xử lý. Giải thuật SJF không độc quyền sẽ dừng hoạt động của tiến trình hiện hành, trong khi giải thuật độc quyền sẽ cho phép tiến trình hiện hành tiếp tục xử lý. Nếu hai tiến trình có cùng thời gian sử dụng CPU, tiến trình đến trước sẽ đươc yêu cầu CPU trước.

Ví dụ :

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

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

P1	P4	P3	P2
0	6	8	12 20

Thời gian chờ:

P1:0, P2: 11, P3: 6, P4:3

Thời gian chờ trung bình là 5.

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

P1	P4	P1	P3	P2
0	3	5	8	12 20

Thời gian chờ:

P1:2, P2:11, P3: 6, P4:0

Thời gian chờ trung bình là 19/4

Thảo luận : Giải thuật này cho phép đạt được thời gian chờ trung bình cực tiểu. Khó khăn thực sự của giải thuật SJF là không thể biết được thời gian yêu cầu chu kỳ CPU tiếp theo? Chỉ có thể dự đoán giá trị này theo cách tiếp cận sau : gọi *tn* là độ dài của thời gian xử lý lần thứ *n*, t_{n+1} là giá trị dự đoán cho lần xử lý tiếp theo. Với hy vọng giá trị dự đoán sẽ gần giống với các giá trị trước đó, có thể sử dụng công thức:

$$t_{n+1} = a\,tn + (1-a)t_n$$

Trong công thức này,tn chứa đựng thông tin gần nhất ; t_n chứa đựng các thông tin quá khứ được tích lũy. Tham số a (0 <= a <= 1) kiểm soát trọng số của hiện tại gần hay quá khứ ảnh hưởng đến công thức dự đoán.

e) Chiến lược điều phối với nhiều mức độ ưu tiên

Nguyên tắc : Ý tưởng chính của giải thuật là phân lớp các tiến trình tùy theo độ ưu tiên của chúng để có cách thức điều phối thích hợp cho từng nhóm. Danh sách sẵn sàng được phân tách thành các danh sách riêng biệt theo cấp độ ưu tiên, mỗi danh sách bao gồm các tiến trình có cùng độ ưu tiên và được áp dụng một giải thuật điều phối thích hợp để điều phối. Ngoài ra, còn có một giải thuật điều phối giữa các nhóm, thường giải thuật này là giải thuật không độc quyền và sử dụng độ ưu tiên cố định.Một tiến trình thuộc về danh sách ở cấp ưu tiên *i* sẽ chỉ được cấp phát CPU khi các danh sách ở cấp ưu tiên lớn hơn *i* đã trống.

Hình 2.11 Điều phối nhiều cấp ưu tiên

Thông thường, một tiến trình sẽ được gán vĩnh viễn với một danh sách ở cấp ưu tiên *i* khi nó được đưa vào hệ thống. Các tiến trình không di chuyển giữa các danh sách. Cách tổ chức này sẽ làm giảm chi phí điều phối, nhưng lại thiếu linh động và có thể dẫn đến tình trạng 'đói CPU' cho các tiến trình thuộc về những danh sách có độ ưu tiên thấp. Do vậy có thể xây dựng giải thuật điều phối nhiều cấp ưu tiên và xoay vòng. Giải thuật này sẽ chuyển dần một tiến trình từ danh sách có độ ưu tiên cao xuống danh sách có độ ưu tiên thấp hơn sau mỗi lần sử dụng CPU. Cũng vậy, một tiến trình chờ quá lâu trong các danh sách có độ ưu tiên thấp cũng có thể được chuyển dần lên các danh sách có độ ưu tiên cao hơn. Khi xây dựng một giải thuật điều phối nhiều cấp ưu tiên và xoay vòng cần quyếtđịnh các tham số :

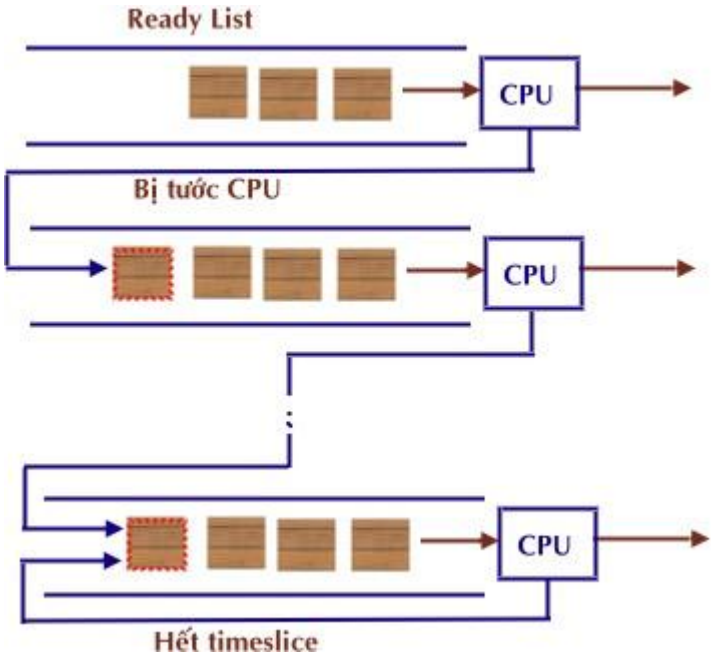
Số lượng các cấp ưu tiên

Giải thuật điều phối cho từng danh sách ứng với một cấp ưu tiên.

Phương pháp xác định thời điểm di chuyển một tiến trình lên danh sách có độ ưu tiên cao hơn.

Phương pháp xác định thời điểm di chuyển một tiến trình lên danh sách có độ ưu tiên thấp hơn.

Phương pháp sử dụng để xác định một tiến trình mới được đưa vào hệ thống sẽ thuộc danh sách ứng với độ tiên nào.



Hình 2.12 Điều phối Multilevel Feedback

2.3. Thông tin liên lạc giữa các tiến trình

2.3.1. Nhu cầu liên lạc giữa các tiến trình

Trong môi trường đa chương, một tiến trình không đơn độc trong hệ thống , mà có thể ảnh hưởng đến các tiến trình khác , hoặc bị các tiến trình khác tác động. Nói cách khác, các tiến trình là những thực thể độc lập , nhưng chúng vẫn có nhu cầu liên lạc với nhau để:

Chia sẻ thông tin: nhiều tiến trình có thể cùng quan tâm đến những dữ liệu nào đó, do vậy hệ điều hành cần cung cấp một môi trường cho phép sự truy cập đồng thời đến các dữ liệu chung.

Hợp tác hoàn thành tác vụ: đôi khi để đạt được một sự xử lý nhanh chóng, người ta phân chia một tác vụ thành các công việc nhỏ có thể tiến hành song song. Thường thì các công việc nhỏ này cần hợp tác với nhau để cùng hoàn thành tác vụ ban đầu, ví dụ dữ liệu kết xuất của tiến trình này lại là dữ liệu nhập cho tiến trình khác …Trong các trường hợp đó, hệ điều hành cần cung cấp cơ chế để các tiến trình có thể trao đổi thông tin với nhau.

2.3.2. Các Cơ Chế Thông Tin Liên lạc

a) Tín hiệu (Signal)

Tín hiệu là một cơ chế phần mềm tương tự như các ngắt cứng tác động đến các tiến trình. Một tín hiệu được sử dụng để thông báo cho tiến trình về một sự kiện nào đó xảy ra. Có nhiều tín hiệu được định nghĩa, mỗi một tín hiệu có một ý nghĩa tương ứng với một sự kiện đặc trưng.

Ví dụ : Một số tín hiệu của UNIX

Tín hiệu	Mô tả
SIGINT	Người dùng nhấn phím DEL để ngắt xử lý tiến trình
SIGQUIT	Yêu cầu thoát xử lý
SIGILL	Tiến trình xử lý một chỉ thị bất hợp lệ
SIGKILL	Yêu cầu kết thúc một tiến trình
SIGFPT	Lỗi floating – point xảy ra (chia cho 0)
SIGPIPE	Tiến trình ghi dữ liệu vào pipe mà không có reader
SIGSEGV	Tiến trình truy xuất đến một địa chỉ bất hợp lệ
SIGCLD	Tiến trình con kết thúc
SIGUSR1	Tín hiệu 1 do người dùng định nghĩa
SIGUSR2	Tín hiệu 2 do người dùng định nghĩa

Mỗi tiến trình sỡ hữu một bảng biểu diễn các tín hiệu khác nhau. Với mỗi tín hiệu sẽ có tương ứng một trình xử lý tín hiệu *(signal handler)* qui định các xử lý của tiến trình khi nhận được tín hiệu tương ứng.

Các tín hiệu được gởi đi bởi :

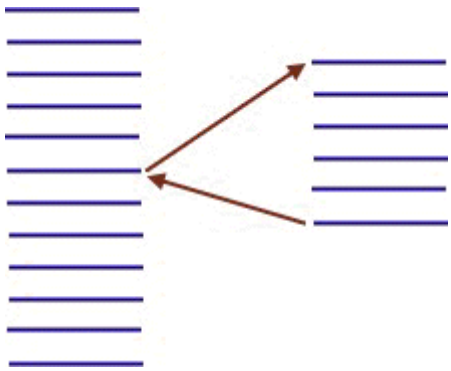
Phần cứng (ví dụ lỗi do các phép tính số học)

Hạt nhân hệ điều hành gởi đến một tiến trình (ví dụ lưu ý tiến trình khi có một thiết bị nhập/xuất tự do).

Một tiến trình gởi đến một tiến trình khác (ví dụ tiến trình cha yêu cầu một tiến trình con kết thúc)

Người dùng (ví dụ nhấn phím Ctl-C để ngắt xử lý của tiến trình)

Khi một tiến trình nhận một tín hiệu, nó có thể xử sự theo một trong các cách sau :



Hình2.13 Liên lạc bằng tín hiệu

Bỏ qua tín hiệu

Xử lý tín hiệu theo kiểu mặc định

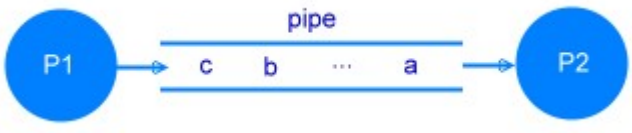
Tiếp nhận tín hiệu và xử lý theo cách đặc biệt của tiến trình.

Liên lạc bằng tín hiệu mang tính chất *không đồng bộ*, nghĩa là một tiến trình nhận tín hiệu không thể xác định trước thời điểm nhận tính hiệu. Hơn nữa các tiến trình không thể kiểm tra được sự kiện tương ứng với tín hiệu có thật sự xảy ra ? Cuối cùng, các tiến trình chỉ có thể thông báo cho nhau về một biến cố nào đó, mà không trao đổi dữ liệu theo cơ chế này được.

b) Pipe

Một pipe là một kênh liên lạc trực tiếp giữa hai tiến trình : dữ liệu xuất của tiến trình này được chuyển đến làm dữ liệu nhập cho tiến trình kia dưới dạng một dòng các byte.

Khi một pipe được thiết lập giữa hai tiến trình, một trong chúng sẽ ghi dữ liệu vào pipe và tiến trình kia sẽ đọc dữ liệu từ pipe. Thứ tự dữ liệu truyền qua pipe được bảo toàn theo nguyên tắc FIFO. Một pipe có kích thước giới hạn (thường là 4096 ký



tự)

Hình 2.14 Liên lạc qua pipe

Một tiến trình chỉ có thể sử dụng một pipe do nó tạo ra hay kế thừa từ tiến trình cha. Hệ điều hành cung cấp các lời gọi hệ thống read/write cho các tiến trình thực hiện thao tác đọc/ghi dữ liệu trong pipe. Hệ điều hành cũng chịu trách nhiệm đồng bộ hóa việc truy xuất pipe trong các tình huống:

Tiến trình đọc pipe sẽ bị khóa nếu pipe trống, nó sẽ phải đợi đến khi pipe có dữ liệu để truy xuất.

Tiến trình ghi pipe sẽ bị khóa nếu pipe đầy, nó sẽ phải đợi đến khi pipe có chỗ trống để chứa dữ liệu.

Liên lạc bằng pipe là một cơ chế liên lạc *một chiều (unidirectional)*, nghĩa là một tiến trình kết nối với một pipe chỉ có thể thực hiện một trong hai thao tác đọc hoặc ghi, nhưng không thể thực hiện cả hai. Một số hệ điều hành cho phép thiết lập hai pipe giữa một cặp tiến trình để tạo liên lạc hai chiều. Trong những hệ thống đó, có nguy cơ xảy ra tình trạng *tắc nghẽn* (deadlock) : một pipe bị giới hạn về kích thước, do vậy nếu cả hai pipe nối kết hai tiến trình đều đầy(hoặc đều trống) và cả hai tiến trình đều muốn ghi (hay đọc) dữ liệu vào pipe(mỗi tiến trình ghi dữ liệu vào một pipe), chúng sẽ cùng bị khóa và chờ lẫn nhau mãi mãi !

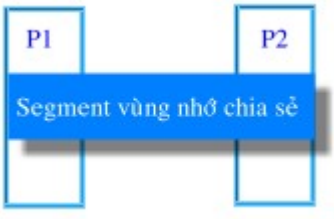
Cơ chế này cho phép truyền dữ liệu với cách thức không cấu trúc.

Ngoài ra, một giới hạn của hình thức liên lạc này là chỉ cho phép kết nối hai tiến trình có quan hệ cha-con, và trên cùng một máy tính.

c) Vùng nhớ chia sẻ

Cách tiếp cận của cơ chế này là cho nhiều tiến trình cùng truy xuất đến một vùng nhớ chung gọi là *vùng nhớ chia sẻ (shared memory)*. Không có bất kỳ hành vi truyền dữ liệu nào cần phải thực hiện ở đây, dữ liệu chỉ đơn giản được đặt vào một vùng nhớ mà nhiều tiến trình có thể cùng truy cập được.

Với phương thức này, các tiến trình chia sẻ một vùng nhớ vật lý thông qua trung gian không gian địa chỉ của chúng. Một vùng nhớ chia sẻ tồn tại độc lập với các tiến trình, và khi một tiến trình muốn truy xuất đến vùng nhớ này, tiến trình phải kết



gắn vùng nhớ chung đó vào không gian địa chỉ riêng của từng tiến trình, và thao tác trên đó như một vùng nhớ riêng của mình.

Hình 2.15 Liên lạc qua vùng nhớ chia sẻ

Đây là phương pháp nhanh nhất để trao đổi dữ liệu giữa các tiến trình. Nhưng phương thức này cũng làm phát sinh các khó khăn trong việc bảo đảm sự toàn vẹn dữ liệu (*coherence*) , ví dụ : làm sao biết được dữ liệu mà một tiến trình truy xuất là dữ liệu mới nhất mà tiến trình khác đã ghi ? Làm thế nào ngăn cản hai tiến trình cùng đồng thờighi dữ liệu vào vùng nhớ chung ?…Rõ ràng vùng nhớ chia sẻ cần được bảo vệ bằng những cơ chế đồng bộ hóa thích hợp..

Một khuyết điểm của phương pháp liên lạc này là không thể áp dụng hiệu quả trong các hệ phân tán , để trao đổi thông tin giữa các máy tính khác nhau.

d) Trao đổi thông điệp (Message)

Hệ điều hành còn cung cấp một cơ chế liên lạc giữa các tiến trình không thông qua việc chia sẻ một tài nguyên chung , mà thông qua việc gởi thông điệp. Để hỗ trợ cơ chế liên lạc bằng thông điệp, hệ điều hành cung cấp các hàm IPC chuẩn (Interprocess communication), cơ bản là hai hàm:

Send(message) : gởi một thông điệp

Receive(message) : nhận một thông điệp

Nếu hai tiến trình P và Q muốn liên lạc với nhau, cần phải thiết lập một mối liên kết giữa hai tiến trình, sau đó P, Q sử dụng các hàm IPC thích hợp để trao đổi thông điệp, cuối cùng khi sự liên lạc chấm dứt mối liên kết giữa hai tiến trình sẽ bị hủy. Có nhiều cách thức để thực hiện sự liên kết giữa hai tiến trình và cài đặt các theo tác send /receive tương ứng : liên lạc trực tiếp hay gián tiếp, liên lạc đồng bộ hoặc không đồng bộ , kích thước thông điệp là cố định hay không … Nếu các tiến trình liên lạc theo kiểu liên kết tường minh, các hàm Send và Receive sẽ được cài đặt với tham số :

Send(destination, message) : gởi một thông điệp đến địa chỉ đích

Receive(source,message) : nhận một thông điệp từ địa chỉ nguồn

Đơn vị truyền thông tin trong cơ chế trao đổi thông điệp là một thông điệp, do đó các tiến trình có thể trao đổi dữ liệu ở dạng có cấu trúc.

Định dạng thông điệp

Loại thông điệp
Địa chỉ đích
Địa chỉ nguồn
Độ dài thông điệp
Các thông tin điều kiển
Nội dung thông điệp

e) Sockets

Giới thiệu: Một socket là một thiết bị truyền thông hai chiều tương tự như tập tin, chúng ta có thể đọc hay ghi lên nó, tuy nhiên mỗi socket là một thành phần trong một mối nối nào đó giữa các máy trên mạng máy tính và các thao tác đọc/ghi chính là sự trao đổi dữ liệu giữa các ứng dụng trên nhiều máy khác nhau.

Sử dụng socket có thể mô phỏng hai phương thức liên lạc trong thực tế : liên lạc thư tín (socket đóng vai trò bưu cục) và liên lạc điện thoại (socket đóng vai trò tổng đài)

Các thuộc tính của socket:

Domaine: định nghĩa dạng thức địa chỉ và các nghi thức sử dụng. Có nhiều domaines, ví dụ UNIX, INTERNET, XEROX_NS, ...

Type: định nghĩa các đặc điểm liên lạc:

Sự tin cậy

Sự bảo toàn thứ tự dữ liệu

Lặp lại dữ liệu

Chế độ nối kết

Bảo toàn giới hạn thông điệp

Khả năng gởi thông điệp khẩn

Để thực hiện liên lạc bằng socket, cần tiến hành các thao tác :

Tạo lập hay mở một socket

Gắn kết một socket với một địa chỉ

Liên lạc : có hai kiểu liên lạc tùy thuộc vào chế độ nối kết:

Liên lạc trong chế độ không liên kết **:** liên lạc theo hình thức hộp thư:

hai tiến trình liên lạc với nhau không kết nối trực tiếp

mỗi thông điệp phải kèm theo địa chỉ người nhận.

Hình thức liên lạc này có đặc điểm được :

người gởi không chắc chắn thông điệp của học được gởi đến người nhận,

một thông điệp có thể được gởi nhiều lần,

hai thông điệp đượ gởi theo một thứ tự nào đó có thể đến tay người nhận theo một thứ tự khác.

Một tiến trình sau khi đã mở một socket có thể sử dụng nó để liên lạc với nhiều tiến trình khác nhau nhờ sử hai primitive *send* và *receive*.

Liên lạc trong chế độ nối kết**:**

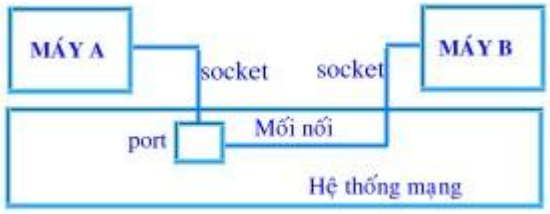
Một liên kết được thành lập giữa hai tiến trình. Trước khi mối liên kết này được thiết lập, một trong hai tiến trình phải đợi có một tiến trình khác yêu cầu kết nối.Có thể sử dụng socket để liên lạc theo mô hình client-server. Trong mô hình này, server sử

dụng lời gọi hệ thống listen và accept để nối kết với client, sau đó , client và server có thể trao đổi thông tin bằng cách sử dụng các primitive send và receive.

Hủy một socket

Ví dụ :

Trong nghi thức truyền thông TCP, mỗi mối nối giữa hai máy tính được xác định bởi một port, khái niệm port ở đây không phải là một cổng giao tiếp trên thiết bị vật lý mà chỉ là một khái niệm logic trong cách nhìn của người lập trình, mỗi port được tương ứng với một số nguyên dương.



Hình 2.16 Các socket và port trong mối nối TCP.

Hình 2.16 minh họa một cách giao tiếp giữa hai máy tính trong nghi thức truyền thông TCP. Máy A tạo ra một socket và kết buộc (bind) socket nầy với một port X (tức là một số nguyên dương có ý nghĩa cục bộ trong máy A), trong khi đó máy B tạo một socket khác và móc vào (connect) port X trong máy A.

Cơ chế socket có thể sử dụng để chuẩn hoá mối liên lạc giữa các tiến trình vốn không liên hệ với nhau, và có thể hoạt động trong những hệ thống khác nhau.

2.4 Đồng bộ hoá tiến trình

2.4.1 Nhu cầu đồng bộ hóa (synchronisation)

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

a) Yêu cầu độc quyền truy xuất (Mutual exclusion)

Các tài nguyên trong hệ thống được phân thành hai loại: tài nguyên có thể chia sẻ cho phép nhiều tiến trình đồng thời truy xuất, và tài nguyên không thể chia sẻ chỉ chấp nhận một (hay một số lượng hạn chế) tiến trình sử dụng tại một thời điểm. Tính không thể chia sẻ của tài nguyên thường có nguồn gốc từ một trong hai nguyên nhân sau đây:

Đặc tính cấu tạo phần cứng của tài nguyên không cho phép chia sẻ.

Nếu nhiều tiến trình sử dụng tài nguyên đồng thời, có nguy cơ xảy ra các kết quả không dự đoán được do hoạt động của các tiến trình trên tài nguyên ảnh hưởng lẫn nhau.

Để các tiến trình hoạt động đồng thời không cạnh tranh hay xung đột lẫn nhau khi sử dụng tài nguyên chung hệ điều hành phải tổ chức cho các tiến trình này độc quyền truy xuất trên các tài nguyên, nghĩa là hệ thống phải kiểm soát sao cho tại một thời điểm, chỉ có một tiến trình được quyền truy xuất một tài nguyên không thể chia sẻ.

b) Yêu cầu phối hợp (Synchronization)

Nhìn chung, mối tương quan về tốc độ thực hiện của hai tiến trình trong hệ thống là không thể biết trước, vì điều này phụ thuộc vào nhiều yếu tố động như tần suất xảy ra các ngắt của từng tiến trình, thời gian tiến trình được cấp phát bộ xử lý… Có thể nói rằng các tiến trình hoạt động không đồng bộ với nhau. Nhưng có những tình huống các tiến trình cần hợp tác trong việc hoàn thành tác vụ, khi đó cần phải đồng bộ hóa hoạt động của các tiến trình , ví dụ một tiến trình chỉ có thể xử lý nếu một tiến trình khác đã kết thúc một công việc nào đó …

2.4.2. Bài toán đồng bộ hoá

a)Vấn đề tranh đoạt điều khiển (race condition)

Các tiến trình hoạt động đồng thời thường cạnh tranh với nhau trong việc sử dụng tài nguyên chung. Hai tiến trình hoạt động đồng thời cùng ghi vào một không gian nhớ chung (một biến chung) trên bộ nhớ hay hai tiến trình đồng thời cùng ghi dữ liệu vào một file chia sẻ, đó là những biểu hiện của sự cạnh tranh về việc sử dụng tài nguyên dùng chung các tiến trình.

Tài nguyên găng

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

Ví dụ sau đây cho thấy hậu quả của việc sử dụng tài nguyên găng trong các chương trình có các tiến trình hoạt động đồng thời:

Giả sử có một ứng dụng giao dịch ngân hàng, hoạt động trong môi trường đa nhiệm, đa người dùng. Mỗi người sử dụng trong môi trường này khi cần thực hiện thao tác rút tiền từ trong tài khoản chung thì phải khởi tạo một tiến trình rút tiền, tiến trình rút tiền được chỉ thực hiện được khi số tiền rút nhỏ hơn số tiền còn lại trong tài khoản chung. Trong môi trường này có thể có nhiều người sử dụng đồng thời thực hiện thao tác rút tiền chung từ hệ thống.

Giả sử có hai tiến trình rút tiền P_1 và P_2, có thể hoạt động đồng thời và cùng chia sẻ một vùng nhớ chung lưu trữ biến *taikhoan* phản ánh thông tin về tài khoản. Mỗi tiến trình muốn rút một khoản tiền *tienrut* từ tài khoản:

```
if (taikhoan - tienrut >=0)

taikhoan = taikhoan - tienrut;

else

error(« khong the rut tien ! »);
```

Nếu tại một thời điểm nào đó:

Giả sử trong tài khoản hiện còn 800, P_1 muốn rút 500 và P_2 muốn rút 400.

Tiến trình P1 và P2 đồng thời rút tiền, thì theo nguyên tắc điều trên không xảy ra vì 500+400>800. Nhưng trong môi trường đa nhiệm, đa người sử dụng nếu hệ điều hành không giám sát tốt việc sử dụng tài nguyên dùng chung của các tiến trình hoạt động đồng thời thì điều trên vẫn có thể xảy ra. Tức là, cả 2 tiến trình P1 và P2 đều thành công trong thao tác rút tiền, mà ứng dụng cũng như hệ điều hành không hề phát hiện. Bởi vì, quá trình rút tiền của các tiến trình P1 và P2 có thể diễn ra như sau:

Sau khi đã kiểm tra điều kiện (taikhoan - tienrut >=0) và nhận kết quả là 300, P_1 hết thời gian xử lý mà hệ thống cho phép, hệ điều hành cấp phát CPU cho P_2.

P_2 kiểm tra cùng điều kiện trên, nhận được kết quả là 400 (do P_1 vẫn chưa rút tiền) và rút 400. Giá trị của *taikhoan* được cập nhật lại là 400.

Khi P_1 được tái kích hoạt và tiếp tục xử lý, nó sẽ không kiểm tra lại điều kiện (taikhoan - tienrut >=0)-vì đã kiểm tra trong lượt xử lý trước- mà thực hiện rút tiền. Giá trị của *taikhoan* sẽ lại được cập nhật thành -100. Tình huống lỗi xảy ra !

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

Trong trường hợp này tài nguyên găng chính là biến tài khoản

Các tình huống tương tự như thế - có thể xảy ra khi có nhiều hơn hai tiến trình đọc và ghi dữ liệu trên cùng một vùng nhớ chung, và kết quả phụ thuộc vào sự điều phối tiến trình của hệ thống- được gọi là các tình huống tranh đoạt điều khiển *(race condition)* .

Nguyên nhân tiềm ẩn của sự xung đột giữa các tiến trình hoạt động đồng thời khi sử dụng tài nguyên găng là: các tiến trình này hoạt động đồng thời với nhau một cách hoàn toàn độc lập và không trao đổi thông tin với nhau nhưng sự thực thi của các tiến trình này lại ảnh hưởng đến nhau.

b) Miền găng (critical section)

Để ngăn chặn các tình huống lỗi có thể nảy sinh khi các tiến trình truy xuất đồng thời một tài nguyên không thể chia sẻ, cần phải áp đặt một sự truy xuất độc quyền trên tài nguyên đó : khi một tiến trình đang sử dụng tài nguyên, thì những tiến trình khác không được truy xuất đến tài nguyên.

Đoạn chương trình trong đó có khả năng xảy ra các mâu thuẫn truy xuất trên tài nguyên chung được gọi là *miền găng (critical section)*. Trong ví dụ trên, đoạn mã :

```
if (taikhoan - tienrut >=0)

taikhoan = taikhoan - tienrut;
```

của mỗi tiến trình tạo thành một miền găng.

Có thể giải quyết vấn đề mâu thuẫn truy xuất nếu có thể bảo đảm tại một thời điểm chỉ có duy nhất một tiến trình được xử lý lệnh trong miền găng.

Một phương pháp giải quyết tốt bài toán miền găng cần thõa mãn 4 điều kiện sau:

-Không có hai tiến trình cùng ở trong miền găng cùng lúc.

-Không có giả thiết nào đặt ra cho sự liên hệ về tốc độ của các tiến trình, cũng như về số lượng bộ xử lý trong hệ thống.

-Một tiến trình tạm dừng bên ngoài miền găng không được ngăn cản các tiến trình khác vào miền găng.

-Không có tiến trình nào phải chờ vô hạn để được vào miền găng.

2.4.3 Các giải pháp đồng bộ hoá

2.4.3.1 Giải pháp « busy waiting »

2.4.3.1.1. Các giải pháp phần mềm

a) Sử dụng các biến cờ hiệu:

Tiếp cận : các tiến trình chia sẻ một biến chung đóng vai trò « chốt cửa » (lock) , biến này được khởi động là 0. Một tiến trình muốn vào miền găng trước tiên phải kiểm tra giá trị của biến lock. Nếu lock = 0, tiến trình đặt lại giá trị cho lock = 1 và đi vào miền găng. Nếu lock đang nhận giá trị 1, tiến trình phải chờ bên ngoài miền găng cho đến khi lock có giá trị 0. Như vậy giá trị 0 của lock mang ý nghĩa là không có tiến trình nào đang ở trong miền găng, và lock=1 khi có một tiến trình đang ở trong miền găng.

```
while (TRUE) {

while (lock==1);//wait
lock = 1;
critical-section ();
lock = 0;
Noncritical-section ();
```

}

Hình 3.5 Cấu trúc một chương trình sử dụng biến khóa để đồng bộ

Giải pháp đơn giản, dễ xây dựng, nhưng vẫn xuất hiện hiện tượng chờ đợi tích cực, gây lãng phí thời gian của processor.

Nếu một tiến trình trong đoạn găng không thể ra khỏi đoạn găng, thì các tiến trình chờ ngoài đoạn găng có thể chờ đợi vô hạn (vì lock không được đặt lại bằng 0)

Giải pháp này có thể vi phạm điều kiện thứ nhất: hai tiến trình có thể cùng ở trong miền găng tại một thời điểm. Giả sử một tiến trình nhận thấy lock = 0 và chuẩn bị vào miền găng, nhưng trước khi nó có thể đặt lại giá trị cho lock là 1, nó bị tạm dừng để một tiến trình khác hoạt động. Tiến trình thứ hai này thấy lock vẫn là 0 thì vào miền găng và đặt lại lock = 1. Sau đó tiến trình thứ nhất được tái kích hoạt, nó gán lock = 1 lần nữa rồi vào miền găng. Như vậy tại thời điểm đó cả hai tiến trình đều ở trong miền găng.

b) Sử dụng việc kiểm tra luân phiên :

Tiếp cận : Đây là một giải pháp đề nghị cho hai tiến trình. Hai tiến trình này sử dụng chung biến *turn* (phản ánh phiên tiến trình nào được vào miền găng), được khởi động với giá trị 0. Nếu *turn = 0*, tiến trình A được vào miền găng. Nếu *turn = 1*, tiến trình A đi vào một vòng lặp chờ đến khi *turn* nhận giá trị *0*. Khi tiến trình A rời khỏi miền găng, nó đặt giá trị *turn* về 1 để cho phép tiến trình B đi vào miền găng.

```
while (TRUE) {

while (turn != 0); // wait
critical-section ();
turn = 1;
Noncritical-section ();

                    }
```

(a) Cấu trúc tiến trình A

```
while (TRUE) {

while (turn != 1); // wait
critical-section ();
```

```
turn = 0;
Noncritical-section ();

                    }
```

(b) Cấu trúc tiến trình B

Hình 3.6 Cấu trúc các tiến trình trong giải pháp kiểm tra luân phiên

Giải pháp này dựa trên việc thực hiện sự kiểm tra nghiêm ngặt đến lượt tiến trình nào được vào miền găng. Do đó nó có thể ngăn chặn được tình trạng hai tiến trình cùng vào miền găng, nhưng lại có thể vi phạm điều kiện thứ ba: một tiến trình có thể bị ngăn chặn vào miền găng bởi một tiến trình khác không ở trong miền găng. Giả sử tiến trình B ra khỏi miền găng rất nhanh chóng. Cả hai tiến trình đều ở ngoài miền găng, và turn = 0. Tiến trình A vào miền găng và ra khỏi nhanh chóng, đặt lại giá trị của turn là1, rồi lại xử lý đoạn lệnh ngoài miền găng lần nữa. Sau đó, tiến trình A lại kết thúc nhanh chóng đoạn lệnh ngoài miền găng của nó và muốn vào miền găng một lần nữa. Tuy nhiên lúc này B vẫn còn mãi xử lý đoạn lệnh ngoài miền găng của mình, và turn lại mang giá trị 1 ! Như vậy, giải pháp này không có giá trị khi có sự khác biệt lớn về tốc độ thực hiện của hai tiến trình, nó vi phạm cả điều kiện thứ hai.

c) Giải pháp của Peterson

Tiếp cận : Petson đưa ra một giải pháp kết hợp ý tưởng của cả hai giải pháp kể trên. Các tiến trình chia sẻ hai biến chung :

```
int turn; // đến phiên ai

int interesse[2]; // khởi động là FALSE
```

Nếu *interesse[i]* = TRUE có nghĩa là tiến trình Pi muốn vào miền găng. Khởi đầu, *interesse[0]=interesse[1]=FALSE* và giá trị của est được khởi động là 0 hay 1. Để có thể vào được miền găng, trước tiên tiến trình Pi đặt giá trị *interesse[i]*=TRUE (xác định rằng tiến trình muốn vào miền găng), sau đó đặt turn=*j* (đề nghị thử tiến trình khác vào miền găng). Nếu tiến trình Pj không quan tâm đến việc vào miền găng (*interesse[j]=FALSE*), thì Pi có thể vào miền găng, nếu không, Pi phải chờ đến khi *interesse[j]=FALSE*. Khi tiến trình Pi rời khỏi miền găng, nó đặt lại giá trị cho *interesse[i]*= FALSE.

```
while (TRUE) {
```

```
        int j = 1-i; // j là tiến trình còn lại
interesse[i]= TRUE;
turn = j;
while (turn == j && interesse[j]==TRUE);
critical-section ();
interesse[i] = FALSE;
Noncritical-section ();

                    }
```

Hình 3.7 Cấu trúc tiến trình Pi trong giải pháp Peterson

giải pháp này ngăn chặn được tình trạng mâu thuẫn truy xuất : mỗi tiến trình Pi chỉ có thể vào miền găng khi *interesse[j]=FALSE* hoặc *turn = i*. Nếu cả hai tiến trình đều muốn vào miền găng thì *interesse[i] = interesse[j] =TRUE* nhưng giá trị của *turn* chỉ có thể hoặc là 0 hoặc là 1, do vậy chỉ có một tiến trình được vào miền găng.

2.4.3.1.2. Các giải pháp phần cứng

a) Cấm ngắt:

Một số vi xử lý cung cấp cấp chỉ thị CLI và STI để người lập trình thực hiện thao tác mở ngắt (STI: Setting Interrupt) và cấm ngắt (CLI: Clean Interrupt) của hệ thống trong lập trình. Người lập trình có thể dùng cặp chỉ thị này để đồng bộ hóa các tiến trình như sau: Trước khi vào miền găng tiến trình thực hiện chỉ thị CLI, để yêu cầu cấm các ngắt trong hệ thống, khi đó ngắt đồng hồ không thể phát sinh, nghĩa là không có một tiến trình nào khác có thể phát sinh, nhờ đó mà tiến trình trong đoạn găng toàn quyền sử dụng tài nguyên găng cho đến hết thời gian xử lý của nó. Khi kết thúc truy xuất tài nguyên găng, tiến trình ra khỏi miền găng, tiến trình thực hiện chỉ thị STI để cho phép ngắt trở lại. Khi đó các tiến trình khác có thể tiếp tục hoạt động và có thể vào miền găng.

Giải pháp đơn giản, dễ cài đặt. Tuy nhiên cần phải có sự hỗ trợ của vi xử lý và dễ gây ra hiện tượng treo toàn bộ hệ thống, khi tiến trình đang ở trong miền găng không có khả năng ra khỏi miền găng. Tiến trình không ra khỏi miền găng nên nó không thể thực hiện chỉ thị STI để mở ngắt cho hệ thống, nên hệ thống bị treo hoàn toàn.

Giải pháp này không thể sử dụng trên các hệ thống đa bộ xử lý, vì CLI chỉ cấm ngắt trên vi xử lý hiện tại chứ không thể cấm ngắt của các vi xử lý khác. Tức sau khi cấm ngắt, tiến trình trong miền găng vẫn có thể bị tranh chấp tài nguyên găng bởi các tiến trình trên các vi xử lý khác trong hệ thống.

b) Chỉ thị TSL (Test-and-Set):

Tiếp cận: đây là một giải pháp đòi hỏi sự trợ giúp của cơ chế phần cứng. Nhiều máy tính cung cấp một chỉ thị đặc biệt cho phép kiểm tra và cập nhật nội dung một vùng nhớ trong một thao tác không thể phân chia, gọi là chỉ thị *Test-and-Set Lock* (TSL) và được định nghĩa như sau:

```
Test-and-Setlock(boolean target)
{

Test-and-Setlock = target;
target = TRUE;

}
```

Nếu có hai chỉ thị TSL xử lý đồng thời (trên hai bộ xử lý khác nhau), chúng sẽ được xử lý tuần tự . Có thể cài đặt giải pháp truy xuất độc quyền với TSL bằng cách sử dụng thêm một biến lock, được khởi gán là FALSE. Tiến trình phải kiểm tra giá trị của biến lock trước khi vào miền găng, nếu lock = FALSE, tiến trình có thể vào miền găng.

```
while (TRUE) {

while(Test-and-Setlock(lock));
critical-section ();
lock = FALSE;
Noncritical-section ();

}
```

Hình 3.8 Cấu trúc một chương trình trong giải pháp TSL

Giải pháp đơn giản, dễ cài đặt nhưng cần phải có sự hỗ trợ của vi xử lý. Ngoài ra nó còn một hạn chế lớn là gây lãng phí thời gian xử lý của processor do tồn tại hiện tượng chờ tích cực trong vòng lặp. Hiện tượng chờ đợi tích cực là hiện tượng processor chỉ chờ một sự kiện nào đó xảy ra mà không làm gì cả.

Tất cả các giải pháp trên đây đều phải thực hiện một vòng lặp để kiểm tra liệu nó có được phép vào miền găng, nếu điều kiện chưa cho phép, tiến trình phải chờ tiếp tục trong vòng lặp kiểm tra này. Các giải pháp buộc tiến trình phải liên tục kiểm tra điều kiện để phát hiện thời điểm thích hợp được vào miền găng như thế được gọi các giải pháp « *busy waiting* ». Lưu ý rằng việc kiểm tra như thế tiêu thụ rất nhiều thời gian sử dụng CPU, do vậy tiến trình đang chờ vẫn chiếm dụng CPU. Xu hướng giải quyết vấn đề đồng bộ hoá là nên tránh các giải pháp « *busy waiting* ».

2.4.3.2. Các giải pháp « SLEEP and WAKEUP »

Để loại bỏ các bất tiện của giải pháp « busy waiting », chúng ta có thể tiếp cận theo hướng cho một tiến trình chưa đủ điều kiện vào miền găng chuyển sang trạng thái blocked, từ bỏ quyền sử dụng CPU. Để thực hiện điều này, cần phải sử dụng các thủ tục do hệ điều hành cung cấp để thay đổi trạng thái tiến trình. Hai thủ tục cơ bản *SLEEP* và *WAKEUP* thường được sử dụng để phục vụ mục đích này.

SLEEP là một lời gọi hệ thống có tác dụng tạm dừng hoạt động của tiến trình (blocked) gọi nó và chờ đến khi được một tiến trình khác « đánh thức ». Lời gọi hệ thống *WAKEUP* nhận một tham số duy nhất : tiến trình sẽ được tái kích hoạt (đặt về trạng thái ready).

Ý tưởng sử dụng SLEEP và WAKEUP như sau : khi một tiến trình chưa đủ điều kiện vào miền găng, nó gọi *SLEEP* để tự khóa đến khi có một tiến trình khác gọi *WAKEUP* để giải phóng cho nó. Một tiến trình gọi *WAKEUP* khi ra khỏi miền găng để đánh thức một tiến trình đang chờ, tạo cơ hội cho tiến trình này vào miền găng :

Cấu trúc chương trình trong giải pháp SLEEP and WAKEUP

```
int busy;   // 1 nếu miền găng đang bị chiếm, nếu không là 0
int blocked;   // đếm số lượng tiến trình đang bị khóa
while (TRUE) {
if (busy){
blocked = blocked + 1;
sleep();
}
else busy = 1;
```

critical-section ();

busy = 0**;**
if(blocked){
wakeup(process);
blocked = blocked - 1;
}

Noncritical-section ();

}

Khi sử dụng SLEEP và WAKEUP cần hết sức cẩn thận, nếu không muốn xảy ra tình trạng mâu thuẫn truy xuất trong một vài tình huống đặc biệt như sau : giả sử tiến trình A vào miền găng, và trước khi nó rời khỏi miền găng thì tiến trình B được kích hoạt. Tiến trình B thử vào miền găng nhưng nó nhận thấy A đang ở trong đó, do vậy B tăng giá trị biến *blocked* và chuẩn bị gọi *SLEEP* để tự khoá. Tuy nhiên trước khi B có thể thực hiện *SLEEP*, tiến trình A lại được tái kích hoạt và ra khỏi miền găng. Khi ra khỏi miền găng A nhận thấy có một tiến trình đang chờ (*blocked*=1) nên gọi *WAKEUP* và giảm giá trị của *blocked*. Khi đó tín hiệu *WAKEUP* sẽ lạc mất do tiến trình B chưa thật sự « ngủ » để nhận tín hiệu đánh thức !Khi tiến trình B được tiếp tục xử lý, nó mới goi *SLEEP* và tự khó vĩnh viễn !

Vấn đề ghi nhận được là tình trạng lỗi này xảy ra do việc kiểm tra tư cách vào miền găng và việc gọi SLEEP hay WAKEUP là những hành động tách biệ, có thể bị ngắt nửa chừng trong quá trình xử lý, do đó có khi tín hiệu WAKEUP gởi đến một tiến trình chưa bị khóa sẽ lạc mất.

Để tránh những tình huống tương tự, hệ điều hành cung cấp những cơ chế đồng bộ hóa dựa trên ý tưởng của chiến lược « SLEEP and WAKEUP » nhưng được xây dựng bao hàm cả phương tiện kiểm tra điều kiện vào miền găng giúp sử dụng an toàn.

a) Giải pháp dùng Semaphore (đèn báo)

Giải pháp này được **Dijkstra** đề xuất vào 1965. Semaphore được định nghĩa để sử dụng đồng bộ hóa như sau:

- Semaphore ***s*** là một *biến* nguyên khởi gán bằng một giá trị không âm e(s), đó là khả năng phục vụ của tài nguyên găng tương ứng.

- Ứng với s có một hàng đợi *f(s)*để lưu các tiến trình đang bị blocked (chờ) trên *s*

- Chỉ có hai thao tác được tác động đến semaphore s. Thao tác Down giảm s đi 1 đơn vị, thao tác Up tăng s lên 1 đơn vị

Mỗi tiến trình trước khi vào miền găng phải gọi Down để kiểm tra và xác lập quyền vào đoạn găng. Khi tiến trình gọi Down(s) thì hệ thống sẽ thực hiện như sau: e(s)=e(s)-1, nếu e(s)>=0 thì tiến trình tiếp tục xử lý và vào miền găng. Nếu e(s)<0 thì tiến trình phải vào hàng đợi để chờ cho đến khi e(s)>=0.

Cài đặt: Gọi *p* là tiến trình thực hiện thao tác

Down(s):

e(s) = e(s) - 1;

if e(s) < 0 {

status(P)= **blocked**;

enter(P,f(s));

}

Mỗi tiến trình ngay sau khi ra khỏi miền găng phải gọi Up để kiểm tra xem có tiến trình nào đang đợi trong hàng đợi hay không, nếu có thì đưa tiến trình trong hàng đợi vào miền găng. Khi tiến trình gọi Up thì hệ thống sẽ thực hiện như sau: e(S)=e(S)+1, nếu e(S)<=0 đưa một tiến trình trong hàng đợi vào miền găng

Up(s):

e(s) = e(s) + 1;

if e(s)<= 0 {

exit(Q,f(s)); //Q là tiến trình đang chờ trên s

status (Q) = **ready**;

enter(Q,ready-list);

}

Lưu ý cài đặt này có thể đưa đến một giá trị âm cho semaphore, khi đó trị tuyệt đối của semaphore cho biết số tiến trình đang chờ trên semaphore.

Điều quan trọng là các thao tác này cần thực hiện một cách không bị phân chia, không bị ngắt giữa chừng, có nghĩa là không một tiến trình nào được phép truy xuất đến semaphore nếu tiến trình đang thao tác trên semaphore này chưa kết thúc xử lý hay chuyển sang trạng thái blocked.

Sử dụng: có thể dùng semaphore để giải quyết vấn đề truy xuất độc quyền hay tổ chức phối hợp giữa các tiến trình.

Tổ chức truy xuất độc quyền với Semaphores: khái niệm *semaphore* cho phép bảo đảm nhiều tiến trình cùng truy xuất đến miền găng mà không có sự mâu thuẫn truy xuất. *n* tiến trình cùng sử dụng một semaphore s, e(s) được khởi gán là 1(tài nguyên găng này chỉ có thể đáp ứng cho một tiến trình nên e(s) được khởi gán bằng 1). Để thực hiện đồng bộ hóa, tất cả các tiến trình cần phải áp dụng cùng cấu trúc chương trình sau đây:

```
while (TRUE) {
Down(s)
critical-section ();
Up(s)
Noncritical-section ();
}
```

Ví dụ1:

Tiến trình thực hiện	Thao tác	E(s)	Trạng thái tiến trình
		1	
1. P1	Down(s)	0	P1 hoạt động
2. P2	Down(s)	-1	P2 chờ

3. P1	Up(s)	0	P2 hoạt động
4. P1	Down(s)	-1	P1 chờ
5. P2	Up(s)	0	P1 hoạt động

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

Có 6 tiến trình yêu cầu sử dụng tài nguyên găng tương ứng với s lần lượt là: A B C D E F

Độ ưu tiên của các tiến trình là(5 là độ ưu tiên cao nhất):

1 1 2 4 2 5

Thời gian các tiến trình cần sử dụng tài nguyên găng là:

4 2 2 2 1 1

T	Tiến trình	Thao tác	E(s)	CS	F(s)
0			1		
1	A	Down	0	A	
2	B	Down	-1	A	B
3	C	Down	-2	A	C,B
4	D	Down	-3	A	D,C,B
5	A	Up	-2	D	C,B
6	E	Down	-3	D	C,E,B
7	D	Up	-2	C	E,B
8	F	Down	-3	C	F,E,B
9	C	Up	-2	F	E,B
10	F	Up	-1	E	B
11	E	Up	0	B	
12	B	Up	1		

Tiến trình ra khỏi đoạn găng sẽ đánh thức tiến trình có độ ưu tiên cao nhất trong hàng đợi để đưa nó vào miền găng. Tiến trình được đưa vào hàng đợi sau nhưng có độ ưu tiên cao hơn sẽ được đưa vào miền găng trước các tiến trình được đưa vào hàng đợi trước nó.

Sử dụng semaphore để đồng bộ hóa tiến trình mang lại những thuận lợi sau:

- Mỗi tiến trình chỉ kiểm tra quyền vào miền găng một lần, khi chờ nó không làm gì cả, tiến trình ra khỏi đoạn găng phải đánh thức tiến trình đang ngủ.
- Không xuất hiện tượng chờ đợi tích cực, nên khai thác tối đa thời gian xử lý của process .
- Nhờ cơ chế hàng đợi mà hệ điều hành có thể thực hiện gán độ ưu tiên cho các tiến trình khi chúng ở trong hàng đợi.

b) Giải pháp dùng Monitors

Giải pháp này được Hoar đề xuất năm 1974 sau đó vào năm 1975 được Brinch và Hansnen đề xuất lại. Monitor là cấu trúc phần mềm đặc biệt được cung cấp bởi ngôn ngữ lập trình, nó bao gồm các thủ tục, các biến và các cấu trúc dữ liệu được định nghĩa bởi Monitor. Monitor được định nghĩa trong các ngôn ngữ lập trình như pascal+, Modula-2, Modula-3. Monitor có các tính chất sau:

1- Các biến và cấu trúc dữ liệu bên trong monitor chỉ có thể được thao tác bởi các thủ tục định nghĩa bên trong monitor đó. (*encapsulation*).

2-Một tiến trình muốn vào monitor phải gọi một thủ tục của monitor đó.

3-Tại một thời điểm, chỉ có một tiến trình duy nhất được hoạt động bên trong một monitor (*mutual exclusive*). Các tiến trình khác đã gọi monitor phải hoãn lại để chờ monitor rỗi.

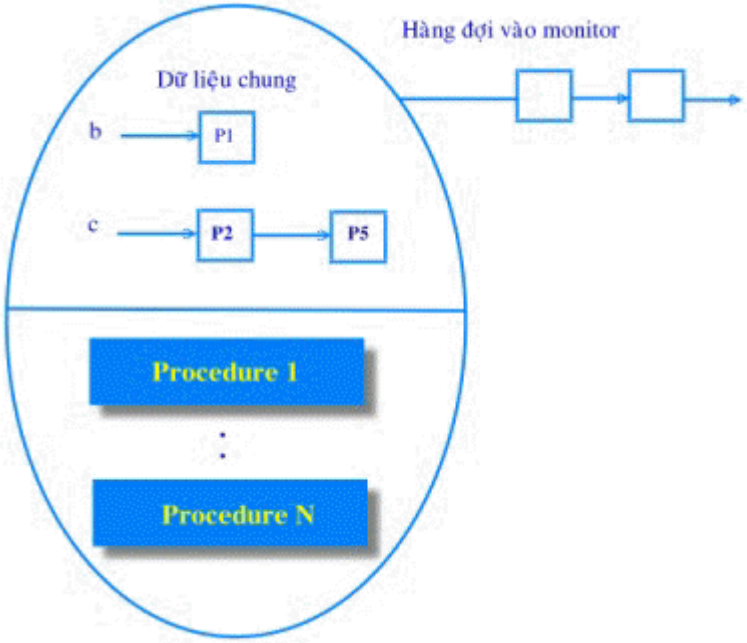
Hai tính chất 1 và 2 tương tự như các tính chất của các đối tượng trong lập trình hướng đối tượng. Như vậy một hệ điều hành hoặc một ngôn ngữ lập trình hướng đối tượng có thể cài đặt monitor như là một đối tượng có các tính chất đặc biệt. Với tính chất 3 monitor có khả năng thực hiện các cơ chế độc quyền, các biến trong monitor có thể được truy xuất chỉ bởi một tiến trình tại một thời điểm. Như vậy các cấu trúc dữ liệu dùng chung bởi các tiến trình có thể được bảo vệ bằng cách đặt chúng bên trong

monitor. Nếu dữ liệu bên trong monitor là tài nguyên găng thì monitor cung cấp sự độc quyền trong việc truy xuất đến tài nguyên găng đó.

Monitor cung cấp các công cụ đồng bộ hóa được định nghĩa như sau: Trong một monitor, có thể định nghĩa các *biến điều kiện* và hai thao tác kèm theo là **Wait** và **Signal(**chỉ có Wait và Signal được tác động đến các biến điều kiện**)** như sau : gọi ***c*** là biến điều kiện được định nghĩa trong monitor:

Wait(c): chuyển trạng thái tiến trình gọi sang blocked , và đặt tiến trình này vào hàng đợi trên biến điều kiện ***c***.

Signal(c): nếu có một tiến trình đang bị khóa trong hàng đợi của ***c,*** tái kích hoạt tiến trình đó, và tiến trình gọi signal sẽ rời khỏi monitor.



Hình 2.18 Monitor và các biến điều kiện

Trình biên dịch chịu trách nhiệm thực hiện việc truy xuất độc quyền đến dữ liệu trong monitor. Để thực hiện điều này, một semaphore nhị phân thường được sử dụng. Mỗi monitor có một hàng đợi toàn cục lưu các tiến trình đang chờ được vào monitor, ngoài ra, mỗi biến điều kiện ***c*** cũng gắn với một hàng đợi ***f(c)*** và hai thao tác trên đó được định nghĩa như sau:

Wait(c) :

status(P)= **blocked**;
enter(P,f(c));

Signal(c) :

if (f(c) != NULL){

exit(Q,f(c)); //Q là tiến trình chờ trên c
statusQ) = **ready**;
enter(Q,ready-list);

}

Sử dụng: Với mỗi nhóm tài nguyên cần chia sẻ, có thể định nghĩa một monitor trong đó đặc tả tất cả các thao tác trên tài nguyên này với một số điều kiện nào đó.:

monitor <tên monitor >

condition <danh sách các biến điều kiện>;

<déclaration de variables>;

procedure $Action_1$();

{

} ....

procedure $Action_n$();

{

}

end monitor;

Hình 3.14 Cấu trúc một monitor

Các tiến trình muốn sử dụng tài nguyên chung này chỉ có thể thao tác thông qua các thủ tục bên trong monitor được gắn kết với tài nguyên:

while (TRUE) {

Noncritical-section ();
<monitor>.Actioni; //critical-section();
Noncritical-section ();

}

Hình 3.15 Cấu trúc tiến trình Pi trong giải pháp monitor

Với monitor, việc truy xuất độc quyền được bảo đảm bởi trình biên dịch mà không do lập trình viên, do vậy nguy cơ thực hiện đồng bộ hóa sai giảm rất nhiều. Tuy nhiên giải pháp monitor đòi hỏi phải có một ngôn ngữ lập trình định nghĩa khái niệm monitor, và các ngôn ngữ như thế chưa có nhiều.

c) Giải pháp trao đổi thông điệp

Khi các tiến trình có sự tương tác với các tiến trình khác, phải yêu cầu cơ bản cần phải được thỏa mãn đó là sự đồng bộ hóa và sự truyền thông. Các tiến trình cần phải thực hiện đồng bộ để thực hiện độc quyền. Các tiến trình hợp tác có thể cần phải trao đổi thông tin. Một hướng tiếp cận để cung cấp cả hai chức năng đó là sự truyền thông điệp. Truyền thông điệp có ưu điểm là có thể thực hiện trên cả hai hệ thống uniprocessor và multiprocessor, khi các hệ thống này hoạt động trên mô hình bộ nhớ chia sẻ.

Giải pháp này dựa trên cơ sở trao đổi thông điệp với hai primitive Send và Receive để thực hiện sự đồng bộ hóa:

Send(destination, message): gởi một thông điệp đến một tiến trình đích.

Receive(source,message): nhận một thông điệp từ một tiến trình nguồn

Tiến trình gọi Receive phải chờ cho đến khi nhận được message từ tiến trình nguồn thì mới có thể tiếp tục được.

Việc sử dụng send và receive để chức đồng bộ hóa các tiến trình được thực hiện như sau:

-Có một tiến trình kiểm soát việc sử dụng tài nguyên găng

-Có nhiều tiến trình khác yêu cầu sử dụng tài nguyên găng này.

- Tiến trình có yêu cầu tài nguyên găng này sẽ gởi một thông điệp đến tiến trình kiểm soát và sau đó chuyển sang trạng thái blocked cho đến khi nhận được một thông điệp chấp nhận cho truy xuất từ tiến trình kiểm soát tài nguyên găng.

-Khi sử dụng xong tài nguyên , tiến trình vừa sử dụng tài nguyên găng gởi một thông điệp khác đến tiến trình kiểm soát để báo kết thúc truy xuất.

-Tiến trình kiểm soát , khi nhận được thông điệp yêu cầu tài nguyên, nó sẽ chờ đến khi tài nguyên găng sẵn sàng để cấp phát thì gởi một thông điệp đến tiến trình đang bị khóa trên tài nguyên đó để đánh thức tiến trình này.

```
while (TRUE) {

Send(process controler, request message);
Receive(process controler, accept message);
critical-section ();
Send(process controler, end message);
Noncritical-section ();

}
```

Giải pháp này thường cài đặt trên các hệ thống mạng máy tính, đặc biệt là trên các hệ thống mạng phân tán. Đây là lợi thế mà semaphore và monitor không làm được

2.5. Tắc nghẽn (Deadlock)

2.5.1. Định nghĩa:

Tất cả các hiện tượng tắc ngẽn đều bắt nguồn từ sự xung đột về tài nguyên của hai hay nhiều tiến trình. Tài nguyên thường là tài nguyên không phân chia được, một số ít trường hợp xảy ra với tài nguyên phân chia được.

Ví dụ 1: Giả sử có hai tiến trình P1, P2 hoạt động đồng thời trong hệ thống. Tiến trình P1 đang giữ tài nguyên R1 và xin được cấp tài nguyên R2 để tiếp tục hoạt động, trong khi đó tiến trình P2 đang chiếm giữ tài nguyên R2 và xin cấp tài nguyên R1 để tiếp tục hoạt động. Như vậy P1, P2 rơi vào trạng thái tắc nghẽn.

Ví dụ 2: Giả sử không gian bộ nhớ còn trống 200KB, và trong hệ thống có 2 tiến trình P1, P2 hoạt động đồng thời. P1 và P2 yêu cầu được sử dụng bộ nhớ như sau:

P1	P2
….	….
Request1 80Kb	Request1 70Kb

…. ….

Request2 30Kb Request2 40Kb

Tắc nghẽn xảy ra khi cả 2 tiến trình cùng yêu cầu bộ nhớ lần thứ hai. Tại thời điểm này không gian bộ nhớ còn trống 50Kb, lớn hơn lượng bộ nhớ mà mỗi tiến trình yêu cầu (30Kb và 40 Kb), nhưng vì cả hai tiến tình đồng thời yêu cầu thêm bộ nhớ, nên hệ thống không thể đáp ứng được, và tắc nghẽn xảy ra.

Như vậy tắc nghẽn là hiện tượng: Trong hệ thống xuất hiện một tập các tiến trình mà mỗi tiến trình trong tập hợp đều chờ đợi được cấp tài nguyên, mà tài nguyên đó được một tiến trình trong tập này chiếm giữ. Và sự đợi này có thể kéo dài vô hạn nếu không có sự tác động từ bên ngoài.

Nói cách khác, mỗi tiến trình trong tập hợp đều chờ được cấp phát một tài nguyên hiện đang bị một tiến trình khác cũng ở trạng thái blocked chiếm giữ. Như vậy không có tiến trình nào có thể tiếp tục xử lý , cũng như giải phóng tài nguyên cho tiến trình khác sử dụng, tất cả các tiến trình trong tập hợp đều bị khóa vĩnh viễn !

Khi hệ thống xảy ra tắc nghẽn nếu hệ điều hành không kịp thời phá bỏ tắc nghẽn thì hệ thống có thể rơi vào tình trạng treo toàn bộ hệ thống.

2.5.2. Điều kiện xuất hiện tắc nghẽn

Coffman, Elphick và Shoshani đã đưa ra 4 điều kiện cần có thể làm xuất hiện tắc nghẽn:

Loại trừ lẫn nhau hay độc quyền sử dụng (Mutual exclusion): Mỗi thời điểm, một tài nguyên không thể chia sẻ được hệ thống cấp phát chỉ cho một tiến trình

Sự chiếm giữ và yêu cầu thêm tài nguyên (Wait for): Các tiến trình tiếp tục chiếm giữ các tài nguyên đã cấp phát cho nó trong khi chờ được cấp phát thêm một số tài nguyên mới.

Không thu hồi tài nguyên từ tiến trình đang giữ chúng (No preemption): Tài nguyên không thể được thu hồi từ tiến trình đang chiếm giữ chúng trước khi tiến trình này sủ dụng chúng xong.

Tồn tại một chu trình trong đồ thị cấp phát tài nguyên (Circular wait):

một tập hợp các tiến trình {P0, P1,…,Pn} đang chờ mà trong đó P0 đang chờ một tài nguyên được giữ bởi P1,

P1 đang chờ tài nguyên đang giữ bởi P2,…,

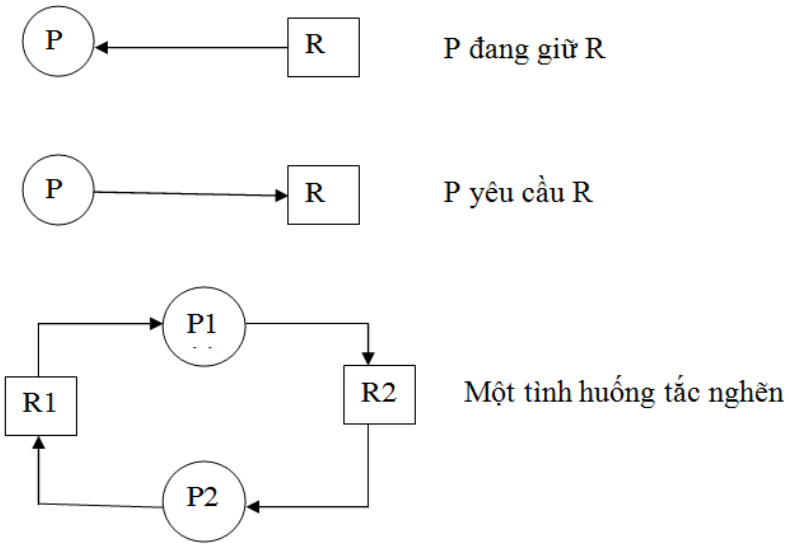
Pn-1 đang chờ tài nguyên đang giữ bởi Pn

Pn đang chờ tài nguyên đang giữ bởi P0

4 điều kiện không hoàn toàn độc lập, các điều kiện là có phụ thuộc lẫn nhau.

Khi có đủ 4 điều kiện này, thì tắc nghẽn xảy ra. Nếu thiếu một trong 4 điều kiện trên thì không có tắc nghẽn.

Có thể sử dụng một đồ thị để mô hình hóa việc cấp phát tài nguyên. Đồ thị này có 2 loại nút : các tiến trình được biểu diễn bằng hình tròn, và mỗi tài nguyên được hiển thị bằng hình vuông



Hình 2.19 Đồ thị cấp phát tài nguyên

2.5.3. Các phương pháp xử lý tắc nghẽn

Chủ yếu có ba hương tiếp cận để xử lý tắc nghẽn :

- Hoàn toàn bỏ qua việc xử lý tắc nghẽn, xem như hệ thống không bao giờ xảy ra tắc nghẽn.

HĐH không có khả năng chống Deadlock

Lý do dung phương pháp này:

Do xác suất xảy ra đealock nhỏ

Giải quyết deadlock đòi hỏi chi phí cao

Xử lý bằng tay do người quản trị hệ thống làm.

Đây là giải pháp của hầu hết các hệ điều hành hiện nay.

- Sử dụng một vài giao thức (protocol) để bảo đảm rằng hệ thống không bao giờ xảy ra tắc nghẽn.

Đây còn gọi là ngăn chặn tắc nghẽn, tập trung vào việc hạn chế quyền truy xuất đến tài nguyên và áp đặt các ràng buộc lên các tiến trình. Điều này có thể ảnh hưởng đến mục tiêu khai thác hiệu quả tài nguyên của hệ điều hành

- Cho phép xảy ra tắc nghẽn, tìm cách phát hiện và khắc phục tắc nghẽn.

Phát hiện tắc nghẽn không giới hạn truy xuất tài nguyên và không áp đặt các ràng buộc lên tiến trình. Để phát hiện tắc nghẽn hệ điều hành thường cài đặt một thuật toán để phát hiện hệ thống có tồn tại một chu trình cấp phát tài nguyên hay không.

Việc kiểm tra để xem thử hệ thống có khả năng xảy ra tắc nghẽn hay không. Có thể được thực hiện liên tục mỗi khi có một yêu cầu tài nguyên, hoặc chỉ thực hiện thỉnh thoảng theo chu kỳ, phụ thuộc vào tắc nghẽn xảy ra như thế nào. Việc kiểm tra tắc nghẽn mỗi khi có yêu cầu tài nguyên sẽ nhận biết được khả năng xảy ra tắc nghẽn nhanh hơn, thuật toán được áp dụng đơn giản hơn vì chỉ dựa vào sự thay đổi trạng thái của hệ thống. Tuy nhiên, hệ thống phải tốn nhiều thời gian cho mỗi lần kiểm tra tắc nghẽn.

2.5.4 Ngăn chặn tắc nghẽn

Ngăn chặn tắc nghẽn là thiết kế một hệ thống sao cho hiện tượng tắc nghẽn bị loại trừ. Để tắc nghẽn không xảy ra, cần bảo đảm tối thiểu một trong 4 điều kiện cần không xảy ra:

-Đối với điều kiện độc quyền: điều kiện này gần như không thể tránh khỏi, vì sự độc quyền là cần thiết đối với tài nguyên thuộc loại phân chia được như các biến

chung, các tập tin chia sẻ, hệ điều hành cần phải hỗ trợ sự độc quyền trên tài nguyên này. Với các tài nguyên không chia được thì rất khó vì bản chất tài nguyên gần như cố định. Tuy nhiên đối với một số tài nguyên về kết xuất, hệ điều hành có thể sử dụng kỹ thuật SPOOL(Smulataneous) để tạo ra tài nguyên ảo cung cấp cho các tiến trình đồng thời.

-Đối với điều kiện sự chiếm giữ và yêu cầu thêm tài nguyên: phải bảo đảm rằng mỗi khi tiến trình yêu cầu thêm một tài nguyên thì nó không chiếm giữ các tài nguyên khác. Có thể áp đặt một trong hai cơ chế truy xuất sau :

+ Buộc các tiến trình yêu cầu tất cả các tài nguyên mà nó cần tại một thời điểm, và tiến trình sẽ bị khóa(blocked) cho đến khi yêu cầu tài nguyên của nó được hệ điều hành đáp ứng

=> phương pháp này có khó khăn là tiến trình khó có thể ước lượng chính xác tài nguyên cần sử dụng vì có thể nhu cầu phụ thuộc vào quá trình xử lý . Ngoài ra nếu tiến trình chiếm giữ sẵn các tài nguyên chưa cần sử dụng ngay thì việc sử dụng tài nguyên sẽ kém hiệu quả. Tiến trình phải đợi trong một thời gian dài để có đủ tài nguyên mới có thể chuyển sang hoạt động được.

+ Khi tiến trình yêu cầu một tài nguyên mới và bị từ chối, nó phải giải phóng các tài nguyên đang chiếm giữ, sau đó lại được cấp phát trở lại khi có đủ tài nguyên.

=> phương pháp này làm phát sinh các khó khăn trong việc bảo vệ tính toàn vẹn dữ liệu của hệ thống.

- Với điều kiện không thu hồi tài nguyên: cho phép hệ thống được thu hồi tài nguyên từ các tiến trình bị khoá và cấp phát cho tiến trình khác và cấp phát trở lại cho tiến trình khi có đủ tài nguyên. Tuy nhiên với một số loại tài nguyên, việc thu hồi sẽ rất khó khăn vì vi phạm sự toàn vẹn dữ liệu.

- Với điều kiện tồn tại một chu trình: tránh tạo chu trình trong đồ thị bằng cách cấp phát tài nguyên theo một sự phân cấp như sau :

gọi R = {R_1, R_2,...,Rm} là tập các loại tài nguyên.

Các loại tài nguyên được phân cấp từ 1-N.

Ví dụ : F(đĩa) = 2, F(máy in) = 12

Các tiến trình khi yêu cầu tài nguyên phải tuân thủ quy định : khi tiến trình đang chiếm giữ tài nguyên Ri thì chỉ có thể yêu cầu các tài nguyên Rj nếu F(Rj) > F(Ri).

2.5.5. Tránh tắc nghẽn

Tránh tắc nghẽn là một trong các phương pháp để giải quyết bài toán tắc nghẽn, nó khác với ngăn chặn tắc nghẽn. Việc tránh tắc nghẽn có thể thực hiện một cách gián tiếp bằng cách ngăn chặn một trong 3 điều kiện cần đầu tiên dẫn đến tắc nghẽn, hoặc một cách trực tiếp bằng cách ngăn chặn không cho xảy ra hiện tượng chờ đợi vòng tròn. Với tránh tắc nghẽn thì một quyết định cấp phát tài nguyên cho tiến trình được thực hiện hay không phụ thuộc vào việc cấp phát đó có nguy cơ dẫn đến tắc nghẽn hay không. Do đó, phương pháp tránh tắc nghẽn cần phải có thông tin về việc sử dụng tài nguyên trong tương lai của các tiến trình. Đây là một vấn đề khó của hệ điều hành, các tiến trình thường không yêu cầu đầy đủ tài nguyên trước khi bắt đầu hoạt động.

Trong công tác phòng tránh tắc nghẽn hệ điều hành phải tuân thủ nghiêm ngặt hai nguyên tắc cơ bản: thứ nhất, không cho phép tiến trình bắt đầu hoạt động nếu yêu cầu tài nguyên của nó có nguy cơ dẫn đến tắc nghẽn. Thứ hai, không được cung cấp thêm tài nguyên cho tiến trình nếu sự cung cấp này có nguy cơ dẫn đến tắc nghẽn.

Một số khái niệm cơ sở

- Trạng thái của hệ thống là sự cấp phát tài nguyên hiện tại cho các tiến trình.

-Trạng thái an toàn : trạng thái A là an toàn nếu hệ thống có thể thỏa mãn các nhu cầu tài nguyên (cho đến tối đa) của mỗi tiến trình theo một thứ tự nào đó mà vẫn ngăn chặn được tắc nghẽn.

Một chuỗi cấp phát an toàn: một thứ tự của các tiến trình $<P_1, P_2,...,Pn>$ là an toàn đối với tình trạng cấp phát hiện hành nếu với mỗi tiến trình Pi nhu cầu tài nguyên của Pi có thể được thỏa mãn với các tài nguyên còn tự do của hệ thống, cộng với các tài nguyên đang bị chiếm giữ bởi các tiến trình Pj khác, với j<i.

Một trạng thái an toàn không thể là trạng thái tắc nghẽn. Ngược lại một trạng thái không an toàn có thể dẫn đến tình trạng tắc nghẽn.

Chiến lược cấp phát : chỉ thỏa mãn yêu cầu tài nguyên của tiến trình khi trạng thái kết quả là an toàn!

Giải thuật xác định trạng thái an toàn

Cần sử dụng các cấu trúc dữ liệu sau :

int Available[NumResources];

/* Available[r]= số lượng các thể hiện còn tự do của tài nguyên r*/

int Max[NumProcs, NumResources];

/*Max[p,r]= nhu cầu tối đa của tiến trình p về tài nguyên r*/

int Allocation[NumProcs, NumResources];

/* Allocation[p,r] = số lượng tài nguyên r thực sự cấp phát cho p*/

int Need[NumProcs, NumResources];

/* Need[p,r] = Max[p,r] - Allocation[p,r]*/

1.Giả sử có các mảng

int Work[NumResources] = Available;

int Finish[NumProcs] = false; với mọi i

2.Tìm i sao cho

Finish[i] == false

Need[i] <= Work[i]

Nếu có i thì sang bước 3

Nếu không có i như thế, đến bước 4.

3. Work = Work + Allocation[i];

  Finish[i] = true;

 Đến bước 2

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

Ngược lại nếu tồn tại i mà finish[i]==false thì hệ thống ở trạng thái không an toàn.

Ví dụ, cho hệ thống sau:

	Max			Allocation			Available		
	R1	R2	R3	R1	R2	R3	R1	R2	R3
P1	6	5	4	2	2	1	2	2	2
P2	5	4	6	3	1	3			
P3	4	3	5	2	1	3			
P4	3	5	2	1	3	1			

Ta tính được bảng Need

	Need		
	R1	R2	R3
P1	4	3	3
P2	2	3	3
P3	2	2	2
P4	2	2	1

B1. Work=(2,2,2)

Finish[i]=false với mọi i từ 1 đến 4.

B2: Tìm thấy tiến trình P3 thỏa mãn 2 điều kiện

Finish[3]=false

$Need_3$<= Work

B3: Work= (2,2,2)+(2,1,3) =(4,3,5)

Finish[3]=true

B2: tìm thấy P1 thỏa mãn 2 điều kiện

B3: Work= (4,3,5)+(2,2,1) =(6,5,6)

Finish[1]=true

B2: tìm thấy P2 thỏa mãn 2 điều kiện

B3: Work= (6,5,6)+(3,1,3) =(9,6,9)

Finish[2]=true

B2: tìm thấy P4 thỏa mãn 2 điều kiện

B3: Work= (9,6,9)+(1,3,1) =(10,9,10)

Finish[4]=true

B2: Không tìm thấy tiến trình nào thỏa mãn điều kiện

B4: finish[i]=true với mọi i từ 1 đến 4 do vậy hệ thống ở trạng thái an toàn.

Giải thuật yêu cầu tài nguyên

Giả sử tiến trình Pi yêu cầu thêm tài nguyên $Request_i$.

1.Nếu $Request_i$. <= Need[i], đến bước 2
Ngược lại, xảy ra tình huống lỗi

2.Nếu $Request_i$. <= Available, đến bước 3
Ngược lại, Pi phải chờ

3.Giả sử hệ thống đã cấp phát cho Pi các tài nguyên mà nó yêu cầu và cập nhật tình trạng hệ thống như sau:

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

Nếu trạng thái kết quả là an toàn, lúc này các tài nguyên trên sẽ được cấp phát thật sự cho Pi

Ngược lại, Pi phải chờ

Ví dụ : Giả sử tình trạng hiện hành của hệ thống được mô tả như sau :

	Max			Allocation			Available		
	R1	R2	R3	R1	R2	R3	R1	R2	R3
P1	3	2	2	1	0	0	4	2	2
P2	6	1	3	2	1	1			
P3	3	1	4	2	1	1			
P4	4	2	2	0	0	2			

Nếu tiến trình P2 yêu cầu 4 cho R1, 1 cho R3. hãy cho biết yêu cầu này có thể đáp ứng mà bảo đảm không xảy ra tình trạng deadlock hay không ? Nhận thấy Available[1] =4, Available[3] =2 đủ để thỏa mãn yêu cầu của P2, ta có

	Need			Allocation			Available		
	R1	R2	R3	R1	R2	R3	R1	R2	R3
P1	2	2	2	1	0	0			
P2	0	0	1	6	1	2			
P3	1	0	3	2	1	1			
P4	4	2	0	0	0	2			

Trạng thái kết quả là an toàn, có thể cấp phát.

Phát hiện tắc nghẽn

Cần sử dụng các cấu trúc dữ liệu sau :

int Available[NumResources];

// Available[r]= số lượng các thể hiện còn tự do của tài nguyên r

int Allocation[NumProcs, NumResources];

// Allocation[p,r] = số lượng tài nguyên r thực sự cấp phát cho p

int Request[NumProcs, NumResources];

// Request[p,r] = số lượng tài nguyên r tiến trình p yêu cầu thêm

Giải thuật phát hiện tắc nghẽn

1. int Work[NumResources] = Available;

   int Finish[NumProcs];

for (i = 0; i < NumProcs; i++)

Finish[i] = (Allocation[i] == 0);

2. Tìm i sao cho

Finish[i] == false

Request[i] <= Work

Nếu không có i như thế, đến bước 4.

3. Work = Work + Allocation[i];

Finish[i]= true;

Đến bước 2 4. Nếu Finish[i] == true với mọi i,

thì hệ thống không có tắc nghẽn

Nếu Finish[i] == false với một số giá trị i,

thì các tiến trình mà Finish[i] == false sẽ ở trong tình trạng tắc nghẽn.

2.5.6. Hiệu chỉnh tắc nghẽn

Khi đã phát hiện được tắc nghẽn, có một vài giải pháp để thoát khỏi tắc nghẽn:

Đình chỉ hoạt động của các tiến trình liên quan

Cách tiếp cận này dựa trên việc thu hồi lại các tài nguyên của những tiến trình bị kết thúc. Có thể sử dụng một trong hai phương pháp sau :

Đình chỉ tất cả các tiến trình trong tình trạng tắc nghẽn. Đây là một giải pháp đơn giản nhất, thường được các hệ điều hành sử dụng nhất

Đình chỉ từng tiến trình liên quan cho đến khi không còn chu trình gây tắc nghẽn : để chọn được tiến trình thích hợp bị đình chỉ, phải dựa vào các yếu tố như độ ưu tiên, thời gian đã xử lý, số lượng tài nguyên đang chiếm giữ , số lượng tài nguyên yêu cầu...

Thu hồi tài nguyên

Có thể hiệu chỉnh tắc nghẽn bằng cách thu hồi một số tài nguyên từ các tiến trình và cấp phát các tài nguyên này cho những tiến trình khác cho đến khi loại bỏ được chu trình tắc nghẽn. Cần giải quyết 3 vấn đề sau:

Chọn lựa một nạn nhân: tiến trình nào sẽ bị thu hồi tài nguyên ? và thu hồi những tài nguyên nào ?

Trở lại trạng thái trước tắc nghẽn: khi thu hồi tài nguyên của một tiến trình, cần phải phục hồi trạng thái của tiến trình trở lại trạng thái gần nhất trước đó mà không xảy ra tắc nghẽn.

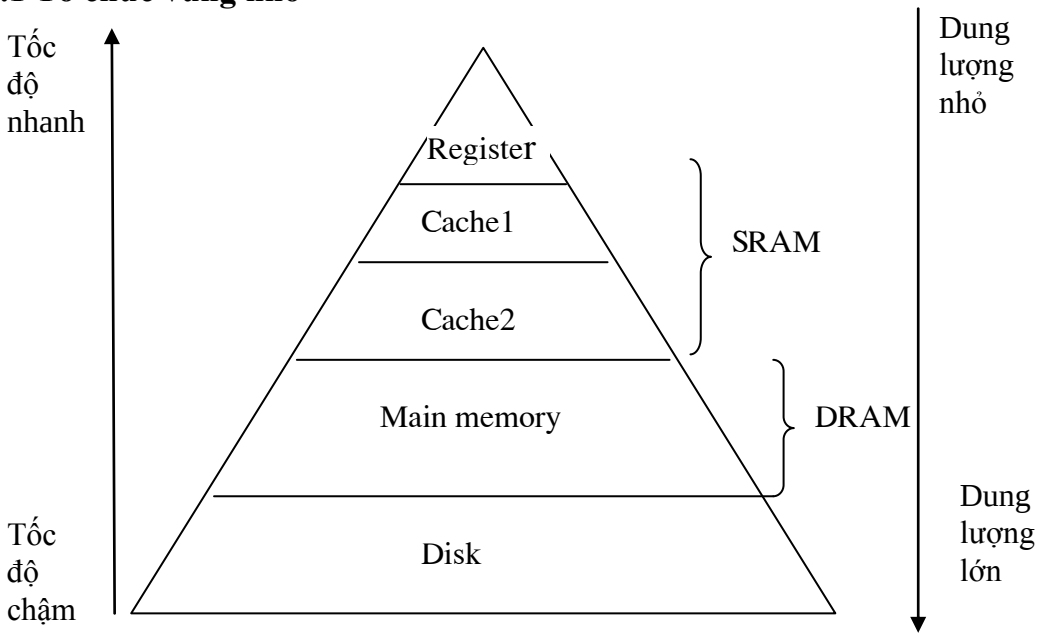
Tình trạng « đói tài nguyên »: làm sao bảo đảm rằng không có một tiến trình luôn luôn bị thu hồi tài nguyên ?

CHƯƠNG 3 :QUẢN LÝ BỘ NHỚ CHÍNH

Bộ nhớ chính là thiết bị lưu trữ duy nhất thông qua đó CPU có thể trao đổi thông tin với môi trường ngoài, do vậy nhu cầu tổ chức, quản lý bộ nhớ là một trong những nhiệm vụ trọng tâm hàng đầu của hệ điều hành . Bộ nhớ chính được tổ chức như một mảng một chiều các từ nhớ (word), mỗi từ nhớ có một địa chỉ . Việc trao đổi thông tin với môi trường ngoài được thực hiện thông qua các thao tác đọc hoặc ghi dữ liệu vào một địa chỉ cụ thể nào đó trong bộ nhớ.

Hầu hết các hệ điều hành hiện đại đều cho phép chế độ đa nhiệm nhằm nâng cao hiệu suất sử dụng CPU. Tuy nhiên kỹ thuật này lại làm nảy sinh nhu cầu chia sẻ bộ nhớ giữa các tiến trình khác nhau . Vấn đề nằm ở chỗ : « *bộ nhớ thì hữu hạn và các yêu cầu bộ nhớ thì vô hạn* ».

3.1 Tổ chức vùng nhớ



Hình 3.1

3.2 Mục tiêu của việc quản lý vùng nhớ

- Cấp phát vùng nhớ cho các tiến trình có yêu cầu và thu hồi vùng nhớ khi tiến trình thực hiện xong. Quản lý được vùng nhớ rỗi, vùng nhớ bận.

- Tại một thời điểm có thể lưu giữ được nhiều tiến trình đồng thời.

- Chuyển đổi giữa địa chỉ *logic* và địa chỉ vật lý (*physic*)

- Chia sẻ bộ nhớ: Bất kỳ một chiến lược nào được cài đặt đều phải có tính mềm dẻo để cho phép nhiều tiến trình có thể truy cập đến cùng một địa chỉ trên bộ nhớ chính. Hệ thống quản lý bộ nhớ phải điều khiển việc truy cập đến không gian bộ nhớ được chia sẻ mà không vi phạm đến các yêu cầu bảo vệ bộ nhớ. Ngoài ra, trong môi trường đa nhiệm hệ điều hành phải chia sẻ không gian nhớ cho các tiến trình để hệ điều hành có thể nạp được nhiều tiến trình vào bộ nhớ để các tiến trình này có thể hoạt động đồng thời với nhau.

- Ngăn chặn các tiến trình xâm phạm đến vùng nhớ được cấp cho tiến trình khác.

Điều quan trọng nhất mà hệ thống quản lý bộ nhớ phải thực hiện là không cho phép các tiến trình của người sử dụng truy cập đến bất kỳ một vị trí nào của chính hệ điều hành, ngoại trừ vùng dữ liệu mà hệ điều hành cung cấp cho người sử dụng.

3.3 Không gian địa chỉ và không gian vật lý

Một trong những hướng tiếp cận trung tâm nhằm tổ chức quản lý bộ nhớ một cách hiệu qủa là đưa ra khái niệm không gian địa chỉ được xây dựng trên không gian nhớ vật lý, việc tách rời hai không gian này giúp hệ điều hành dễ dàng xây dựng các cơ chế và chiến lược quản lý bộ nhớ hữu hiệu :

Địa chỉ logic – còn gọi là *địa chỉ ảo*, là địa chỉ do bộ xử lý tạo ra.

Địa chỉ vật lý - là địa chỉ thực tế mà trình quản lý bộ nhớ nhìn thấy và thao tác.

Không gian địa chỉ – là tập hợp tất cả các địa chỉ ảo phát sinh bởi một chương trình.

Không gian vật lý – là tập hợp tất cả các địa chỉ vật lý tương ứng với các địa chỉ ảo.

Địa chỉ ảo và địa chỉ vật lý là như nhau trong phương thức kết buộc địa chỉ vào thời điểm biên dịch cũng như vào thời điểm nạp. Nhưng có sự khác biệt giữa địa chỉ ảo và địa chỉ vật lý trong phương thức kết buộc vào thời điểm xử lý.

MMU (*memory-management unit*) là một cơ chế phần cứng được sử dụng để thực hiện chuyển đổi địa chỉ ảo thành địa chỉ vật lý vào thời điểm xử lý.

Chương trình của người sử dụng chỉ thao tác trên các địa chỉ ảo, không bao giờ nhìn thấy các địa chỉ vật lý . Địa chỉ thật sự ứng với vị trí của dữ liệu trong bô nhớ chỉ được xác định khi thực hiện truy xuất đến dữ liệu.

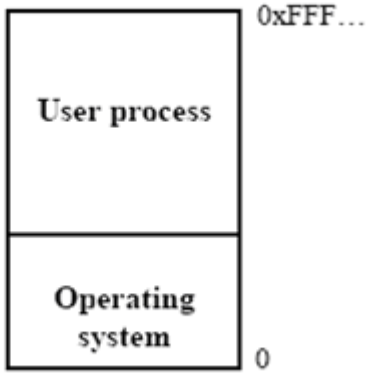
3.4. Cấp phát liên tục

Tiến trình được nạp vào một vùng nhớ liên tục đủ lớn để chứa toàn bộ tiến trình. Không cho phép chương trình khác sử dụng vùng nhớ dành cho chương trình.

Trong kỹ thuật này, bộ nhớ chính được chia thành 2 phần cố định, phần nằm ở vùng dịa chỉ thấp dùng để chứa chính hệ điều hành, phần còn lại, tạm gọi là user program, là sẵn sàng cho việc sử dụng của các tiến trình được nạp vào bộ nhớ chính.

3.4.1 Hệ đơn chương

Trong phương pháp này bộ nhớ được chia sẻ cho hệ điều hành và một tiến trình duy nhất của người sử dụng. Tại một thời điểm, một phần của bộ nhớ sẽ do hệ điều hành chiếm giữ, phần còn lại thuộc về tiến trình người dùng duy nhất trong hệ thống. Tiến trình này được toàn quyền sử dụng bộ nhớ dành cho nó.



Hình 3.2 Tổ chức bộ nhớ trong hệ thống đơn chương

Để bảo vệ hệ điều hành khỏi sự xâm phạm tác động của chương trình người dùng, sử dụng một thanh ghi giới hạn lưu địa chỉ cao nhất của vùng nhớ được cấp cho hệ điều hành. Tất cả các địa chỉ được tiến trình người dung truy xuất sẽ được so sánh với nội dung thanh ghi giới hạn, nếu địa chỉ này lớn hơn giới hạn cho phép thì là hợp lệ, ngược lại thì từ chối không cho truy xuất.

Khi bộ nhớ được tổ chức theo cách thức này, chỉ có thể xử lý một tiến trình tại một thời điểm. Quan sát hoạt động của các tiến trình, có thể nhận thấy rất nhiều tiến trình trải qua phần lớn thời gian để chờ các thao tác nhập/xuất hoàn thành. Trong suốt

thời gian này, CPU ở trạng thái rỗi. Trong trường hợp như thế, hệ thống đơn chương không cho phép sử dụng hiệu quả CPU. Ngoài ra, sự đơn chương không cho phép nhiều người sử dụng làm việc đồng thời theo cơ chế tương tác. Để nâng cao hiệu suất sử dụng CPU, cần cho phép chế độ đa chương mà trong đó các tiến trình chia sẻ CPU với nhau để hoạt động đồng hành.

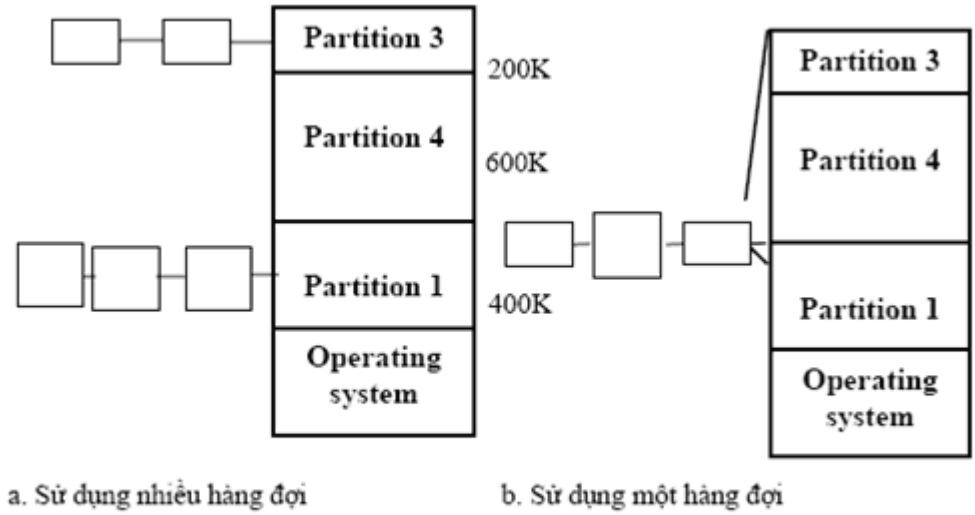
3.4.2 Hệ thống đa chương với phân vùng cố định

Một trong những phương pháp đơn giản nhất để cấp phát bộ nhớ là chia bộ nhớ thành những phân vùng với các biên vùng cố định có kích thước khác nhau hay bằng nhau. Một tiến trình có thể được nạp vào bất kỳ phân vùng nào nêú kích thước của tiến trình này nhỏ hơn hoặc bằng của kích thước của phân vùng và phân vùng này còn trống. Mỗi phân vùng chỉ có thể chứa một tiến trình. Do đó, cấp độ đa chương được giới hạn bởi số lượng phân vùng. Khi tiến trình kết thúc, phân vùng trở nên sẵn dùng cho một tiến trình khác. Có hai tiếp cận để tổ chức hàng đợi, khi có một tiến trình yêu cầu bộ nhớ:

• **Sử dụng nhiều hàng đợi**: mỗi phân vùng sẽ có một hàng đợi tương ứng. Khi một tiến trình mới được tạo lập sẽ được đưa vào hàng đợi của phân vùng có kích thước nhỏ nhất đủ lớn để chứa tiến trình. Cách tiếp cận này sẽ đơn giản trong việc đưa một tiến trình từ hàng đợi vào phân vùng vì không có sự lựa chọn khác ở đây, khi phân vùng mà tiến trình đợi trống

Cách tổ chức này có khuyết điểm trong trường hợp các hàng đợi của một số phân vùng lớn thì trống trong khi các hàng đợi của các phân vùng nhỏ lại đầy, buộc các tiến trình trong những hàng đợi này phải chờ được cấp phát bộ nhớ, do vậy sử dụng không hiệu quả bộ nhớ.

• **Sử dụng một hàng đợi**: Hệ thống dùng một hàng đợi chung cho tất cả các phân vùng, do tất cả các tiến trình muốn được nạp vào phân vùng sẽ được đưa vào hàng đợi chung này. Sau đó nếu có một phân vùng trống thì hệ thống sẽ xem xét để đưa một tiến trình có kích thước vừa đủ vào phân vùng trống đó. Cách tiếp cận này linh động hơn so với việc sử dụng nhiều hàng đợi ở trên, nhưng việc chọn một tiến trình trong hàng đợi vào phân vùng là một việc làm khá phức tạp vì nó phải dựa vào nhiều yếu tố khác nhau như: độ ưu tiên của tiến tình, trạng thái của tiến trình, các mối quan hệ của tiến trình,…



Hình 3.3 Cấp phát đa vùng với phân vùng cố định

Xuất hiện **hiện tượng phân mảnh nội vi (**internal fragmentation): do kích thước tiến trình được nạp nhỏ hơn kích thước của phân vùng chứa tiến trình, phần bộ nhớ không được sử dụng đến trong phân vùng này gọi là phân mảnh nội vi.

Nhận xét:

-Mức độ đa chương của hệ thống bị giới hạn bởi số lượng phân vùng.

- Sử dụng bộ nhớ không hiệu quả:

Tổng bộ nhớ nhỏ tự do, rời rạc còn lớn nhưng không thể sử dụng để nạp tiến trình khác.

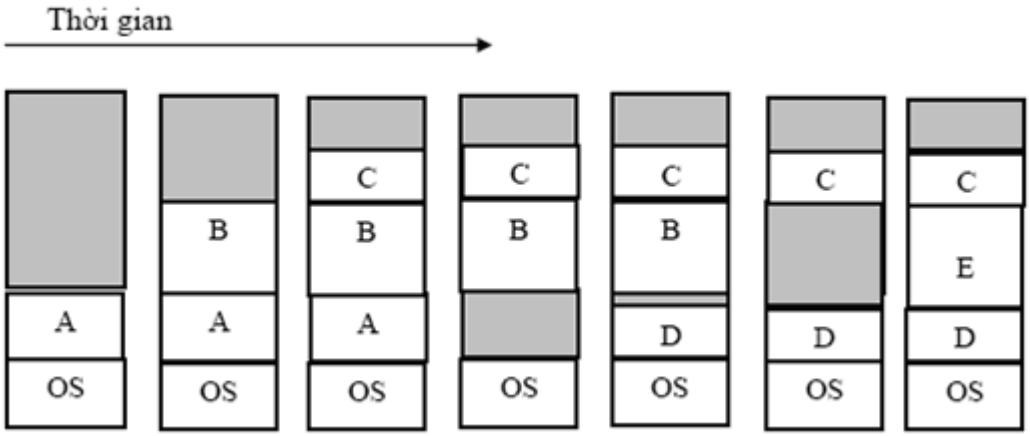
Tiến trình có kích thước lớn hơn phân vùng lớn nhất sẽ không bao giờ được thực hiện.

- Ưu điểm: đơn giản, dễ tổ chức bảo vệ, giảm thời gian tìm kiếm.

Cách tổ chức phân vùng cố định ít được sử dụng trong các hệ điều hành hiện nay.

3.4.3 Hệ thống đa chương với phân vùng động

Kỹ thuật này thường được sử dụng trong các hệ điều hành hiện nay.



Hình 3.4 Cấp phát đa vùng với phân vùng động

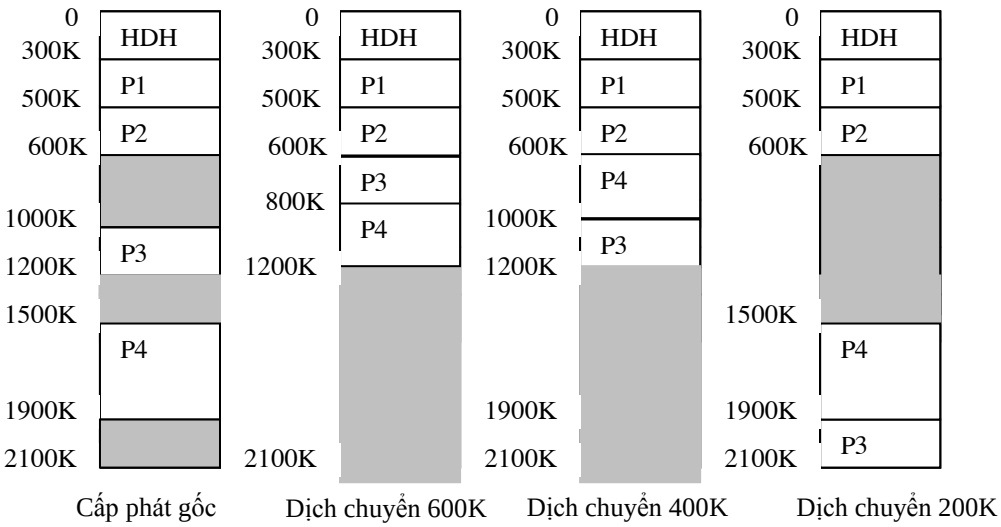
Trong kỹ thuật phân vùng động, số lượng các vùng trên bộ nhớ và kích thước mỗi phân vùng là có thể thay đổi. Tức là phần user program trên bộ nhớ không dược phân chia trước mà nó chỉ được ấn định sau khi dã có một tiến trình được nạp vào bộ nhớ chính. Khi có một tiến trình được nạp vào bộ nhớ nó được hệ điều hành cấp cho nó một không gian vừa đủ, liên tục để chứa tiến trình, phần còn lại để sẵn sàng cấp cho tiến trình khác sau này. Khi một tiến trình kết thúc nó được đưa ra ngoài và phần không gian bộ nhớ mà tiến trình này trả cho hệ điều hành sẽ được hệ điều hành cấp cho tiến trình khác

Xuất hiện hiện tượng phân mảnh ngoại vi(*external fragmentation*) : khi các tiến trình lần lượt vào và ra khỏi hệ thống, dần dần xuất hiện các khe hở giữa các tiến trình. Đây là các khe hở được tạo ra do kích thước của tiến trình mới được nạp nhỏ hơn kích thước vùng nhớ mới được giải phóng bởi một tiến trình đã kết thúc và ra khỏi hệ thống. , không gian bộ nhớ trống bị phân rã thành những mảnh nhớ nhỏ.

Hiện tượng này có thể dẫn đến tình huống tổng vùng nhớ trống đủ để thoả mãn yêu cầu, nhưng các vùng nhớ này lại không liên tục ! Người ta có thể áp dụng kỹ thuật « dồn bộ nhớ » (*memory compaction*) để kết hợp các mảnh bộ nhớ nhỏ rời rạc thành một vùng nhớ lớn liên tục. Tuy nhiên, kỹ thuật này đòi hỏi nhiều thời gian xử lý, ngoài ra, sự kết buộc địa chỉ phải thực hiện vào thời điểm xử lý, vì các tiến trình có thể bị di chuyển trong quá trình dồn bộ nhớ.

Thuật toán đơn giản là dịch chuyển các tiến trình về phía đầu của bộ nhớ.

Ví dụ:



Hình 3.5

Vấn đề cấp phát động

Lựa chọn vùng nhớ tự do trong danh sách các vùng nhớ tự do có kích thước bằng hoặc lớn hơn kích thước của tiến trình để cấp phát cho tiến trình.

Có 3 chiến lược phổ biến nhất được dùng:

• **First-fit**: cấp phát vùng nhớ tự do đầu tiên đủ lớn chứa tiến trình. Tìm kiếm có thể bắt đầu tại đầu từ danh sách tập hợp các vùng trống hay tại điểm kết thúc của tìm kiếm first-fit trước đó. Chúng ta dừng tìm kiếm ngay khi chúng ta tìm thấy một vùng trống đủ lớn.

• **Best-fit**: cấp phát vùng nhớ tự do nhỏ nhất đủ lớn chứa tiến trình. Chúng ta phải tìm toàn bộ danh sách, trừ khi danh sách được xếp thứ tự theo kích thước.

Worst-fit: cấp phát vùng nhớ tự do lớn nhất đủ lớn để chứa tiến trình. Chúng ta phải tìm toàn bộ danh sách trừ khi nó được xếp theo thứ tự kích thước.

Quản lý các khối rỗi bận

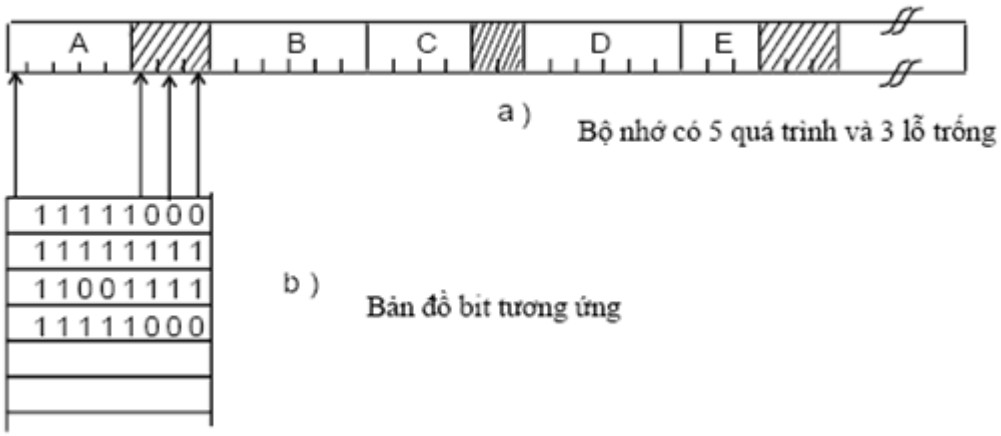
Bộ nhớ được chia thành các đơn vị cấp phát, có kích thước bằng nhau.

- **Quản lý bằng bản đồ bit:**

Mỗi đơn vị cấp phát được ánh xạ tới một bit trong bản đồ bit. Giá trị bit này xác định trạng thái của đơn vị bộ nhớ đó: 0 đang tự do, 1 đã được cấp phát. Khi cần nạp

một tiến trình có kích thước k đơn vị, hệ thống sẽ tìm trong bản đồ bit một dãy k bit có giá trị 0.

Kích thước của đơn vị cấp phát là vấn đề lớn trong thiết kế. Nếu kích thước đơn vị cấp phát nhỏ sẽ làm tăng kích thước của bản đồ bit. Ngược lại, nếu kích thước đơn vị cấp phát lớn có thể gây hao phí cho đơn vị cấp phát sau cùng. Đây là giải pháp đơn giản nhưng thực hiện chậm nên ít được dùng.

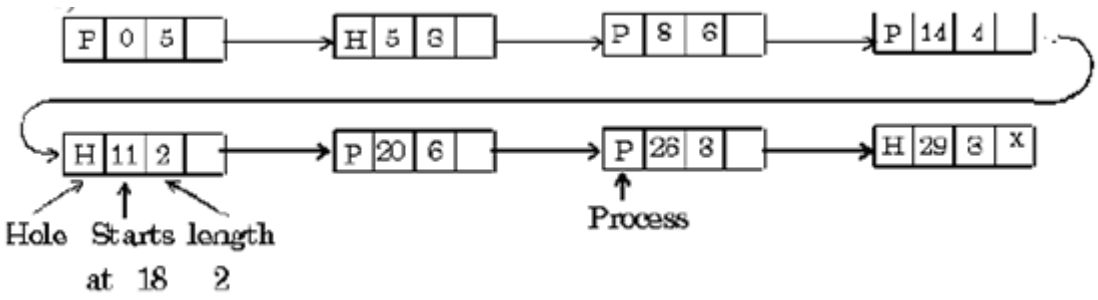


Hình 3.6 Quản lý bộ nhớ bằng bản đồ bit

- Quản lý bằng danh sách liên kết:

Dùng một danh sách liên kết để quản lý các phân đoạn bộ nhớ đã cấp phát và phân đoạn tự do. Danh sách liên kết gồm nhiều nút liên tiếp. Mỗi nút gồm 3 trường chính: trường thứ nhất để xác định khối nhớ đã cấp phát (P: tiến trình) hay đang còn trống (H:Hole), trường thứ hai cho biết thứ tự của đơn vị cấp phát đầu tiên trong khối, trường thứ ba cho biết khối gồm bao nhiêu đơn vị cấp phát.

Việc sắp xếp các phân đoạn theo địa chỉ hay theo kích thước tuỳ thuộc vào giải thuật quản lý bộ nhớ.



Hình 3.7 Quản lý bộ nhớ bằng danh sách liên kết

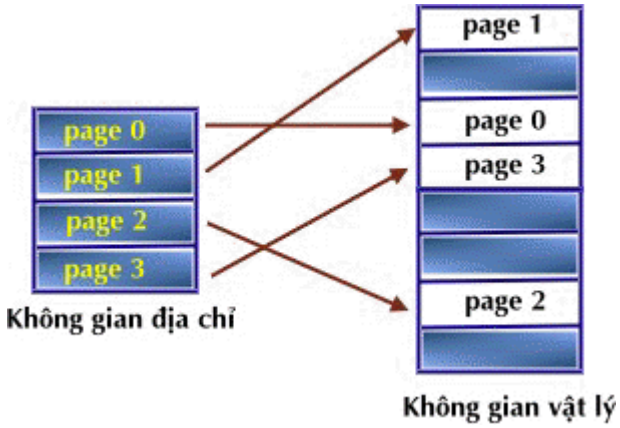
Sau khi thực hiện một thao tác cấp phát hoặc sau khi đưa một tiến trình ra khỏi bộ nhớ thì hệ điều hành phải cập nhật lại bản đồ bit hoặc danh sách liên kết, điều này có thể làm giảm tốc độ thực hiện của hệ thống.

3.5. Cấp phát không liên tục

Cấp phát không liên tục là một chương trình có thể được phân chia thành mộ số đoạn, các đoạn này nằm ở các vùng nhớ rời rạc nhau, giữa các vùng nhớ này có thể có các vùng nhớ được phân phối cho chương trình khác.

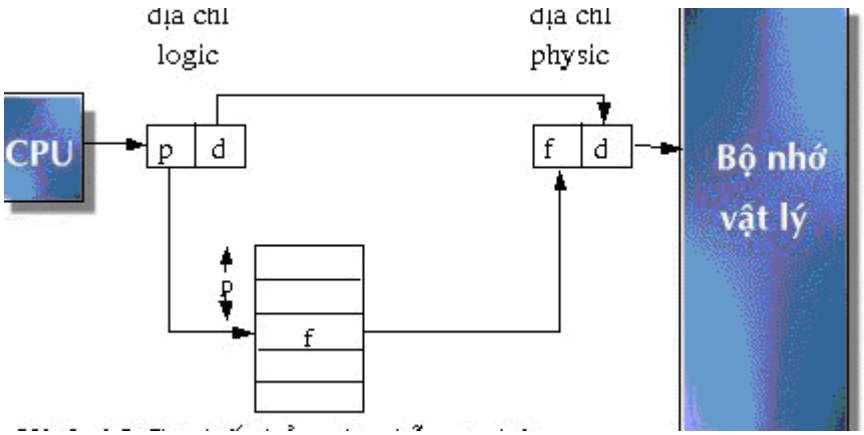
3.5.1 Kỹ thuật phân trang (Paging)

Ý tưởng:



Hình 3.8 Mô hình bộ nhớ phân trang

Phân bộ nhớ vật lý thành các khối (block) có kích thước cố định và bằng nhau, gọi là *khung trang (page frame)*. Không gian địa chỉ cũng được chia thành các khối có cùng kích thước với khung trang, và được gọi là *trang* (*page*). Khi cần nạp một tiến trình để xử lý, các trang của tiến trình sẽ được nạp vào những khung trang còn trống bất kỳ, có thể không liên tiếp nhau. Một tiến trình kích thước N trang sẽ yêu cầu N khung trang tự do.



Hình 3.9 Cơ chế phần cứng hỗ trợ phân trang

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

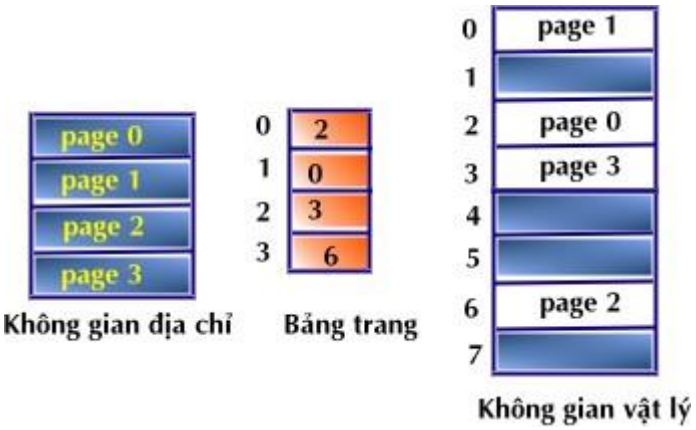
Cơ chế phần cứng hỗ trợ thực hiện chuyển đổi địa chỉ trong cơ chế phân trang là bảng trang (*pages table*). Bảng trang để theo dõi vị trí các trang tiến trình trên bộ nhớ.

Mỗi tiến trình có một bảng trang.

Số phần tử của bảng trang=số trang trong không gian địa chỉ của tiến trình.

Các phần tử được đánh số bắt đầu từ 0

Mỗi phần tử trong bảng trang mô tả một trang cho biết địa chỉ bắt đầu của vị trí lưu trữ trang tương ứng trong bộ nhớ vật lý (số hiệu khung trang trong bộ nhớ vật lý đang chứa trang).



Hình 3.10

Chuyển đổi địa chỉ:

Địa chỉ logic <p, d>

Địa chỉ vật lý <f, d>

số hiệu trang (p): sử dụng như chỉ mục đến phần tử tương ứng trong bảng trang.

địa chỉ tương đối trong trang (d): kết hợp với địa chỉ bắt đầu của khung trang tương ứng để tạo ra địa chỉ vật lý mà trình quản lý bộ nhớ sử dụng.

Số hiệu khung trang (f):địa chỉ bắt đầu của khung trang trong bộ nhớ vật lý.

Kích thước của trang do phần cứng qui định. Để dễ phân tích địa chỉ ảo thành số hiệu trang và địa chỉ tương đối, kích thước của một trang thông thường là một lũy thừa của 2 (biến đổi trong phạm vi 512 bytes và 8192 bytes).

Nếu kích thước của không gian địa chỉ là 2^m và kích thước trang là 2^n, thì *m-n* bits cao của địa chỉ ảo sẽ biểu diễn số hiệu trang, và *n* bits thấp cho biết địa chỉ tương đối trong trang.

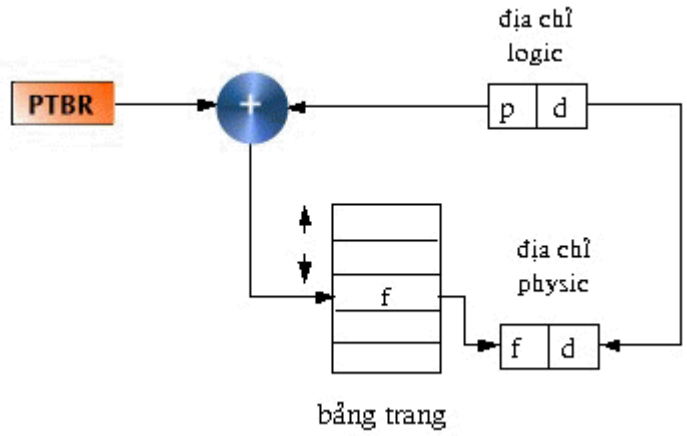
Số trang	Độ dời trang
P	D
m –n	N

Cài đặt bảng trang:

- Với các bảng trang có kích thước nhỏ, trong trường hợp đơn giản nhất, bảng trang được cài đặt trongmột tập các thanh ghi

- Nếu bảng trang có kích thước lớn, nó phải được lưu trữ trong bộ nhớ chính, và sử dụng một thanh ghi để lưu địa chỉ bắt đầu lưu trữ bảng trang (PTBR).

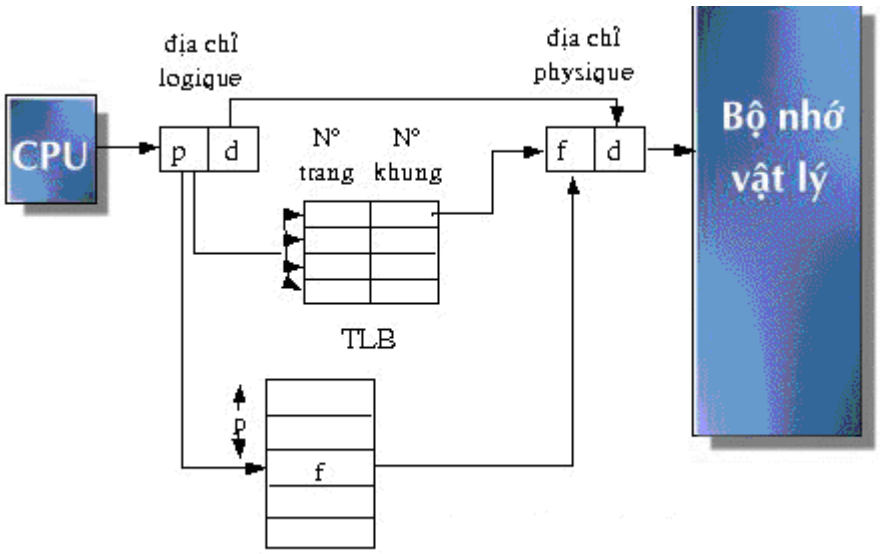
Theo cách tổ chức này, mỗi truy xuất đến dữ liệu hay chỉ thị đều đòi hỏi hai lần truy xuất bộ nhớ : một cho truy xuất đến bảng trang và một cho bản thân dữ liệu, do vậy truy cập chậm.



Hình 3.11 Sử dụng thanh ghi nền trỏ đến bảng trang

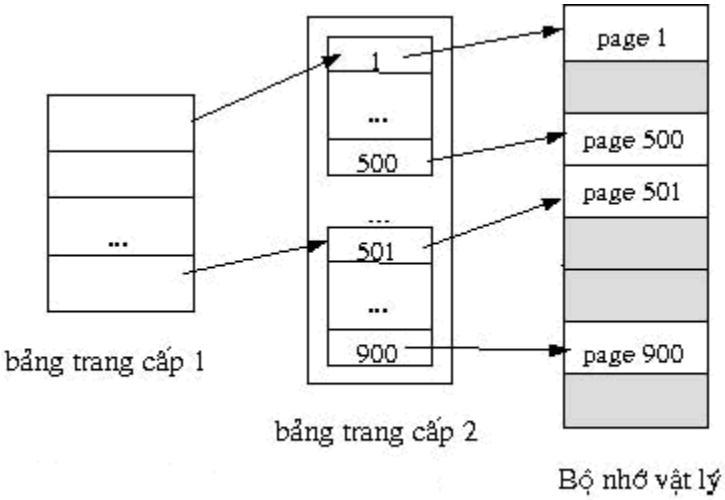
- Để nâng cao tốc độ truy xuất, sử dụng thêm một vùng nhớ đặc biệt , với tốc độ truy xuất nhanh và cho phép tìm kiếm song song, vùng nhớ cache nhỏ này thường được gọi là bộ nhớ kết hợp (translation look-aside buffer TLBs). Mỗi thanh ghi trong bộ nhớ kết hợp chứa số hiệu trang và số hiệu khung trang tương ứng, khi CPU phát sinh một địa chỉ logic, số hiệu trang của địa chỉ sẽ được so sánh cùng lúc với các số hiệu trang trong bộ nhớ kết hợp để tìm ra phần tử tương ứng. Nếu có trang tương ứng trong bộ nhớ kết hợp thì sẽ xác định ngay số hiệu khung trang tương ứng, nếu không mới cần thực hiện thao tác tìm kiếm trong bảng trang. Nhờ đặc tính này mà việc tìm kiếm trên bộ nhớ kết hợp được thực hiện rất nhanh, nhưng chi phí phần cứng lại cao.

Trong kỹ thuật phân trang, TLBs được sử dụng để lưu trữ các trang bộ nhớ được truy cập gần hiện tại nhất.



Hình 3.12 Bảng trang với TLBs

Bảng trang đa cấp:



Hình 3.13 Bảng trang 2 cấp

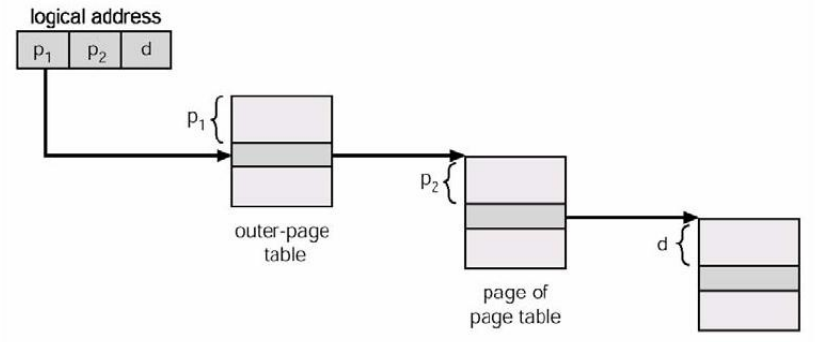
Mỗi hệ điều hành có một phương pháp riêng để tổ chức lưu trữ bảng trang. Đa số các hệ điều hành cấp cho mỗi tiến trình một bảng trang. Tuy nhiên phương pháp này không thể chấp nhận được nếu hệ điều hành cho phép quản lý một không gian địa chỉ có dung lượng quá (2^{31}, 2^{9}): trong các hệ thống như thế, bản thân bảng trang đòi hỏi một vùng nhớ quá lớn(2^{22})!

Phân trang đa cấp: phân chia bảng trang thành các phần nhỏ, bản thân bảng trang cũng sẽ được phân trang

Khi tiến trình thực hiện, chỉ có một phần của bảng trang được nạp vào bộ nhớ chính, đó là phần chứa các phần tử của các trang đang thực hiện tại thời điểm hiện tại.

Ví dụ trong bảng trang 2 cấp cho máy 32 bit với kích thước trang 4KB. Địa chỉ logic được chia thành số trang chứa 20 bit và độ dời trang chứa 12 bit. Vì chúng ta phân trang bảng trang, số trang được chia thành số trang 10 bit và độ dời trang 10-bit. Do đó, một địa chỉ logic như sau:

Số trang		Độ dời trang
P_1	P_2	d
10	10	12



Hình 3.14

Phân trang 3 cấp

Không gian địa chỉ 64 bit, kích thước trang 4KB

Trang bên ngoài cấp 2	Trang bên ngoài	Trang bên trong	Độ dời
P1	P2	P3	D
32	10	10	12

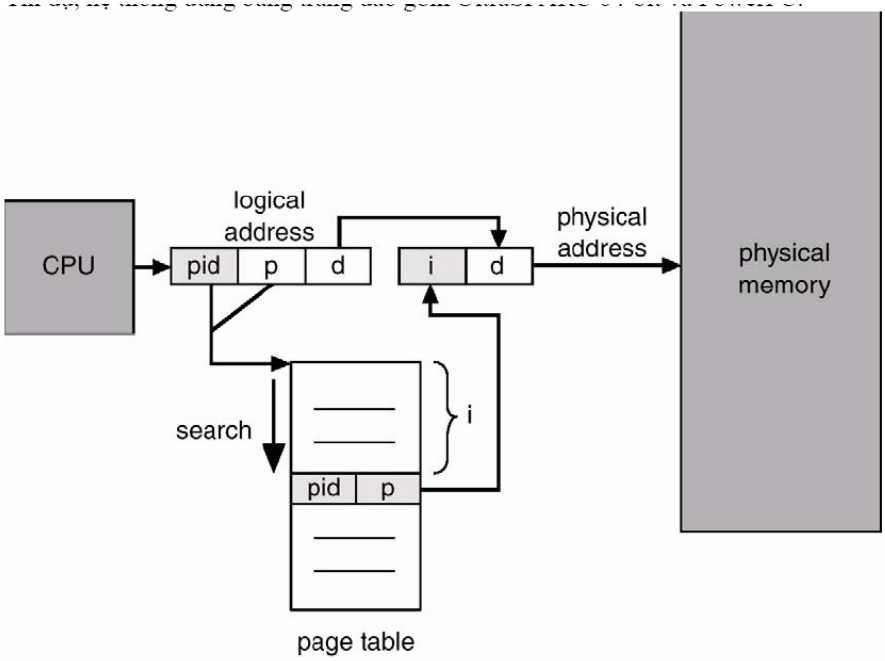
Bảng trang nghịch đảo (inverted page table).

Sử dụng duy nhất một *bảng trang nghịch đảo* cho tất cả các tiến trình. Mỗi phần tử trong *bảng trang nghịch đảo* phản ánh một khung trang trong bộ nhớ bao gồm địa chỉ logic của một trang đang được lưu trữ trong bộ nhớ vật lý tại khung trang này, cùng với thông tin về tiến trình đang được sỡ hữu trang. Mỗi địa chỉ ảo khi đó là một bộ ba <pid,p, d >

Trong đó : pid là định danh của tiến trình

p là số hiệu trang

d là địa chỉ tương đối trong trang



Hình 3.15

Mỗi phần tử trong bảng trang nghịch đảo là một cặp <pid, p >. Khi một tham khảo đến bộ nhớ được phát sinh, một phần địa chỉ ảo là <pid, p > được đưa đến cho trình quản lý bộ nhớ để tìm phần tử tương ứng trong bảng trang nghịch đảo, nếu tìm thấy, địa chỉ vật lý <i,d> sẽ được phát sinh. Trong các trường hợp khác, xem như tham khảo bộ nhớ đã truy xuất một địa chỉ bất hợp lệ.

Bảo vệ:

Cơ chế bảo vệ trong hệ thống phân trang được thực hiện với các bit bảo vệ được gắn với mỗi khung trang. Thông thường, các bit này được lưu trong bảng trang , vì mỗi truy xuất đến bộ nhớ đều phải tham khảo đến bảng trang để phát sinh địa chỉ vật lý, khi đó, hệ thống có thể kiểm tra các thao tác truy xuất trên khung trang tương ứng có hợp lệ với thuộc tính bảo vệ của nó không.

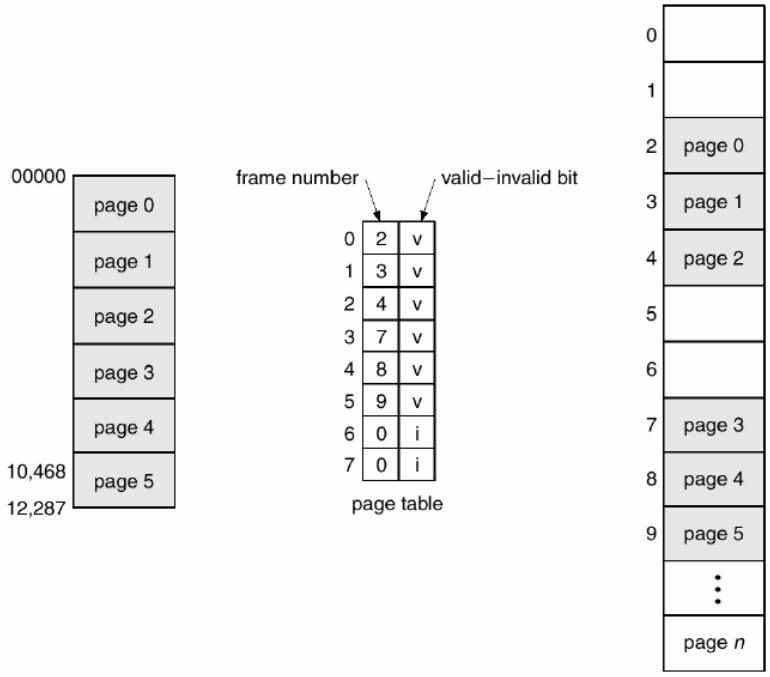
Ngoài ra, một bit phụ trội được thêm vào trong cấu trúc một phần tử của bảng trang : bit hợp lệ-bất hợp lệ (valid-invalid).

Hợp lệ : trang tương ứng thuộc về không gian địa chỉ của tiến trình.

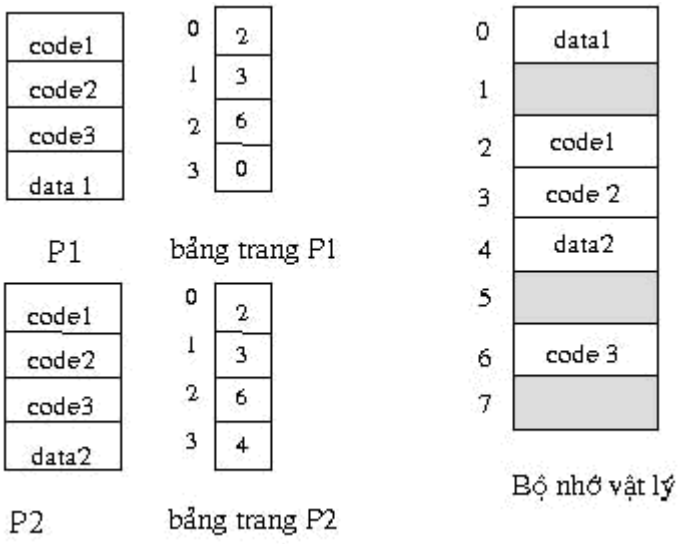
số hiệu khung trang	bit valid-invalid

Bất hợp lệ: trang tương ứng không nằm trong không gian địa chỉ của tiến trình, điều này có nghĩa tiến trình đã truy xuất đến một địa chỉ không được phép.

Hình 3.16 *Cấu trúc một phần tử trong bảng trang*



Chia sẻ bộ nhớ trong cơ chế phân trang:



Hình 3.17 Chia sẻ các trang trong hệ phân trang

Một ưu điểm của cơ chế phân trang là cho phép chia sẻ các trang giữa các tiến trình.Trong trường hợp này, sự chia sẻ được thực hiện bằng cách ánh xạ nhiều địa chỉ logic vào một địa chỉ vật lý duy nhất. Có thể áp dụng kỹ thuật này để cho phép các tiến

trình chia sẻ một vùng code chung*:* nếu có nhiều tiến trình của cùng một chương trình, chỉ cần lưu trữ một đoạn code của chương trình này trong bộ nhớ, các tiến trình sẽ có thể cùng truy xuất đến các trang chứa code chung này. Lưu ý để có thể chia sẻ một đoạn code, đoạn code này phải có thuộc tính cố định và không thay đổi trong quá trình xử lý.

Ưu điểm: tiết kiệm bộ nhớ.

Nhận xét

Kỹ thuật phân trang loại bỏ được hiện tượng phân mảnh ngoại vi : mỗi khung trang đều có thể được cấp phát cho một tiến trình nào đó có yêu cầu. Tuy nhiên hiện tượng phân mảnh nội vi vẫn có thể xảy ra khi kích thước của tiến trình không đúng bằng bội số của kích thước một trang, khi đó, trang cuối cùng sẽ không được sử dụng hết.

Một khiá cạnh tích cực rất quan trọng khác của kỹ thuật phân trang là sự phân biệt rạch ròi góc nhìn của người dùng và của bộ phận quản lý bộ nhớ vật lý:

Góc nhìn của người sử dụng: một tiến trình của người dùng nhìn thấy bộ nhớ như là một không gian liên tục, đồng nhất và chỉ chứa duy nhất bản thân tiến trình này.

Góc nhìn của bộ nhớ vật lý: một tiến trình của người sử dụng được lưu trữ phân tán khắp bộ nhớ vật lý, trong bộ nhớ vật lý đồng thời cũng chứa những tiến trình khác.

Phần cứng đảm nhiệm việc chuyển đổi địa chỉ logic thành địa chỉ vật lý . Sự chuyển đổi này là trong suốt đối với người sử dụng.

3.5.2. Phân đoạn (Segmentation)

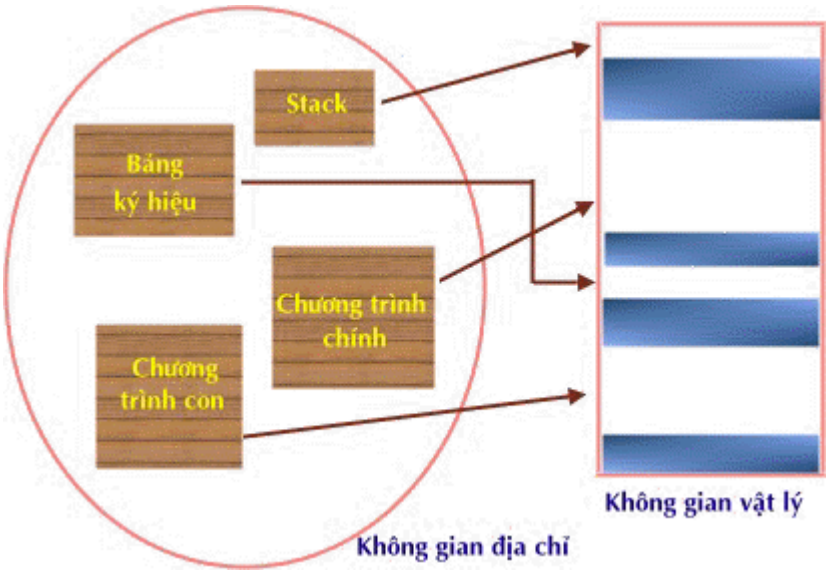
Lưu ý rằng sự phân trang không phản ánh đúng cách thức người sử dụng cảm nhận về bộ nhớ. Người sử dụng nhìn thấy bộ nhớ như một tập các đối tượng của chương trình (chương trình chính, chương trình con, các thư viện...) và một tập các đối tượng dữ liệu (biến toàn cục, stack, vùng nhớ chia sẻ...). Vấn đề đặt ra là cần tìm một cách thức biểu diễn bộ nhớ sao cho có thể cung cấp cho người dùng một cách nhìn gần với quan điểm logic của họ hơn và đó là kỹ thuật phân đoạn

Không gian địa chỉ của các tiến trình kể cả các dữ liệu liên quan cũng được chia thành các đoạn khác nhau và không nhất thiết phải có kích thước bằng nhau, thông

thường mỗi thành phần của một chương trình/tiến trình như: code, data, stack, subprogram…là một đoạn.

Bộ nhớ được tổ chức cấp phát động

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



Hình 3.18 Mô hình phân đoạn bộ nhớ

Hình 3.19 Cơ chế phần cứng hỗ trợ kĩ thuật phân đoạn

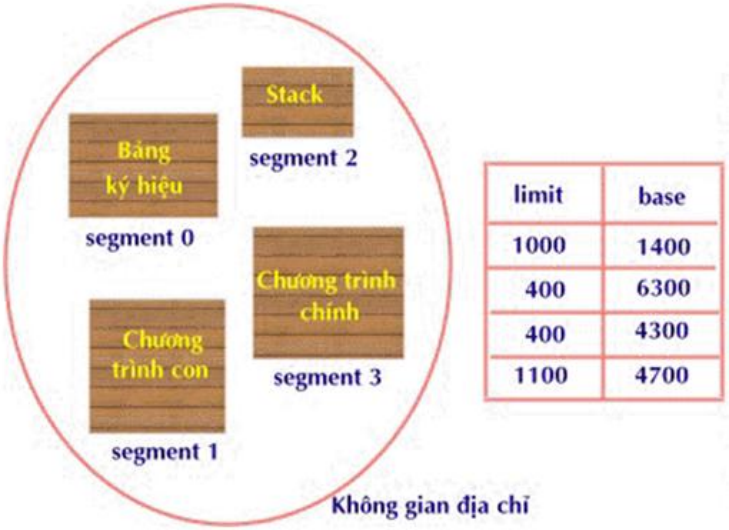
Để theo dõi các đoạn của các tiến trình khác nhau trên bộ nhớ, hệ điều hành sử dụng các bảng phân đoạn tiến trình, thông thường một tiến trình có một bảng phân đoạn riêng. Mỗi phần tử trong bảng phân đoạn gồm tối thiểu 2 trường: trường thứ nhất cho biết địa chỉ cơ sở(base) của phân vùng mà đoạn chương trình tương ứng được nạp, trường thứ 2 cho biết độ dài/giới hạn (limit) của phân đoạn, trường này còn có tác dụng dùng để kiểm soát sự truy xuất bất hợp lệ của các tiến trình.

Chuyển đổi địa chỉ:

Mỗi địa chỉ ảo là một bộ <***s,d***> *:*

số hiệu phân đoạn s : được sử dụng như chỉ mục đến bảng phân đoạn

địa chỉ tương đối d : có giá trị trong khoảng từ 0 đến giới hạn chiều dài của phân đoạn. Nếu địa chỉ tương đối hợp lệ, nó sẽ được cộng với giá trị chứa cơ sở để phát sinh địa chỉ vật lý tương ứng.



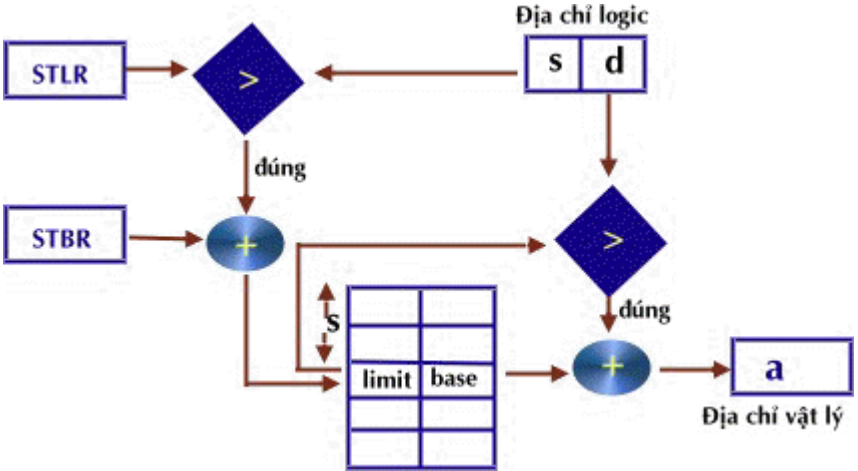
Cài đặt bảng phân đoạn:

Hình 3.20 Hệ thống phân đoạn

Có thể sử dụng các thanh ghi để lưu trữ bảng phân đoạn nếu số lượng phân đoạn nhỏ. Trong trường hợp chương trình bao gồm quá nhiều phân đoạn, bảng phân đoạn phải được lưu trong bộ nhớ chính. Một *thanh ghi nền bảng phân đoạn* (STBR) chỉ đến địa chỉ bắt đầu của bảng phân đoạn. Vì số lượng phân đoạn sử dụng trong một chương trình có thể thay đổi, cần sử dụng thêm một *thanh ghi đặc tả kích thước bảng phân đoạn* (STLR).

Hệ điều hành cũng tổ chức một danh sách riêng để theo dõi các đoạn còn trống trên bộ nhớ.

Với một địa chỉ logic <*s*,*d*>, trước tiên số hiệu phân đoạn *s* được kiểm tra tính hợp lệ (s <STLR). Kế tiếp, cộng giá trị s với STBR để có được địa chỉ địa chỉ của phần



tử thứ *s* trong bảng phân đoạn (STBR+*s*). Địa chỉ vật lý cuối cùng là (STBR+*s* + *d*)

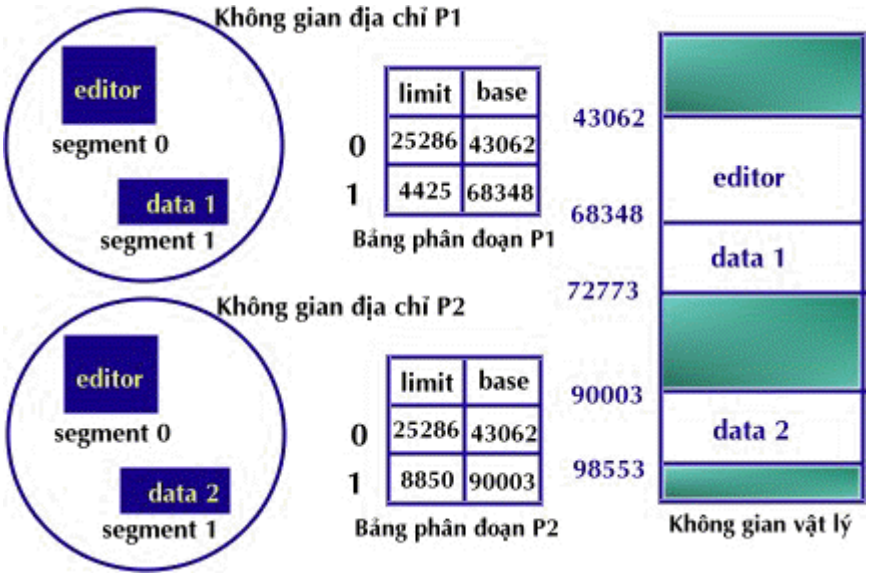
Hình 3.21 Sử dụng STBR, STLR và bảng phân đoạn

Bảo vệ: Một ưu điểm đặc biệt của cơ chế phân đoạn là khả năng đặc tả thuộc tính bảo vệ cho mỗi phân đoạn. Vì mỗi phân đoạn biểu diễn cho một phần của chương trình với ngữ nghĩa được người dùng xác định, người sử dụng có thể biết được một phân đoạn chứa đựng những gì bên trong, do vậy họ có thể đặc tả các thuộc tính bảo vệ thích hợp cho từng phân đoạn.

Cơ chế phần cứng phụ trách chuyển đổi địa chỉ bộ nhớ sẽ kiểm tra các bit bảo vệ được gán với mỗi phần tử trong bảng phân đoạn để ngăn chặn các thao tác truy xuất bất hợp lệ đến phân đoạn tương ứng.

Kỹ thuật phân đoạn dùng chính nó để cài đặt các chính sách bảo vệ và chia sẻ bộ nhớ. Bởi vì mỗi phần tử trong bảng đoạn bao gồm một trường độ dài đoạn và trường chỉ nơi bắt đầu của đoạn, nên một tiến trình trong đoạn không thể truy cập đến một vị trí trong bộ nhớ chính mà vị trí này vượt qua giới hạn của đoạn, ngoiaj trừ đó là truy cập dữ liệu đến một đoạn dữ liệu nào đó.

Chia sẻ phân đoạn:



Hình 3.22 *Chia sẻ code trong hệ phân đoạn*

Một ưu điểm khác của kỹ thuật phân đoạn là khả năng chia sẻ ở mức độ phân đoạn. Nhờ khả năng này, các tiến trình có thể chia sẻ với nhau từng phần chương trình (ví dụ các thủ tục, hàm), không nhất thiết phải chia sẻ toàn bộ

chương trình như trường hợp phân trang. Mỗi tiến trình có một bảng phân đoạn riêng, một phân đoạn được chia sẻ khi các phần tử trong bảng phân đoạn của hai tiến trình khác nhau cùng chỉ đến một vị trí vật lý duy nhất.

Nhận xét:

-Vẫn xuất hiện hiện tượng phân mảnh ngoại vi,

- Sự phân đoạn là tường minh đối với người lập trình, nó cung cấp một sự t huận lợi để người lập trình tổ chức chương trình và dữ liệu. Người lập trình hoặc trình biên dịch có thể gán các chương trình và dữ liệu đến các đoạn nhớ khác nhau.

- Kỹ thuật phân đoạn thể hiện được cấu trúc logic của chương trình, nhưng nó phải cấp phát các khối nhớ có kích thước khác nhau cho các phân đoạn của chương trình trên bộ nhớ vật lý, điều này phức tạp hơn nhiều so với việc cấp phát các khung trang.

Một giải pháp dung hoà là kết hợp cả hai kỹ thuật phân trang và phân đoạn : chúng ta tiến hành *phân đoạn kết hợp phân trang*

3.5.3. Phân đoạn kết hợp phân trang (Paged segmentation)

Ý tưởng: Không gian địa chỉ là một tập các phân đoạn, mỗi phân đoạn được chia thành nhiều trang. Khi một tiến trình được đưa vào hệ thống, hệ điều hành sẽ cấp phát cho tiến trình các khung trang cần thiết để chứa đủ các phân đoạn của tiến trình.

Bộ nhớ vật lý được chia thành các khung trang.

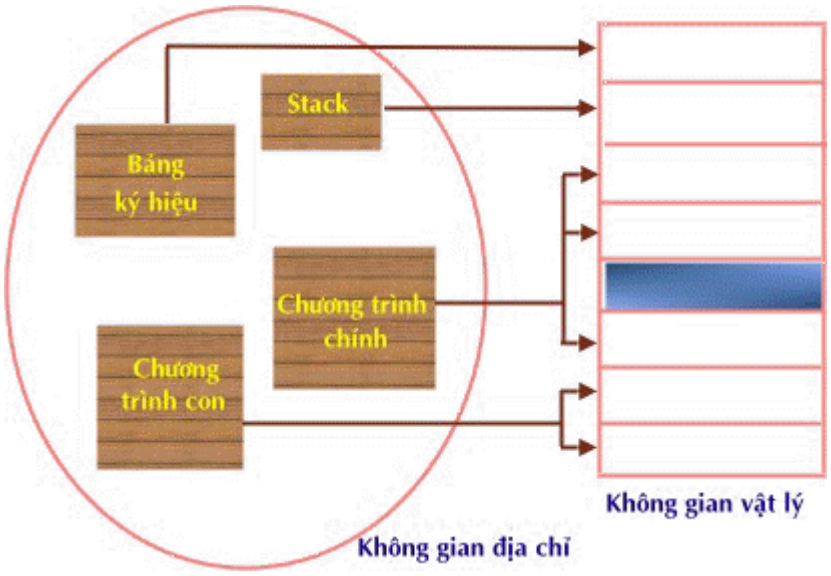
Cơ chế MMU trong kỹ thuật phân đoạn kết hợp phân trang:

Để hỗ trợ kỹ thuật phân đoạn, cần có một *bảng phân đoạn*, nhưng giờ đây mỗi phân đoạn cần có một *bảng trang* phân biệt.

Chuyển đổi địa chỉ:

Mỗi địa chỉ logic là một bộ ba: <***s,p,d***>

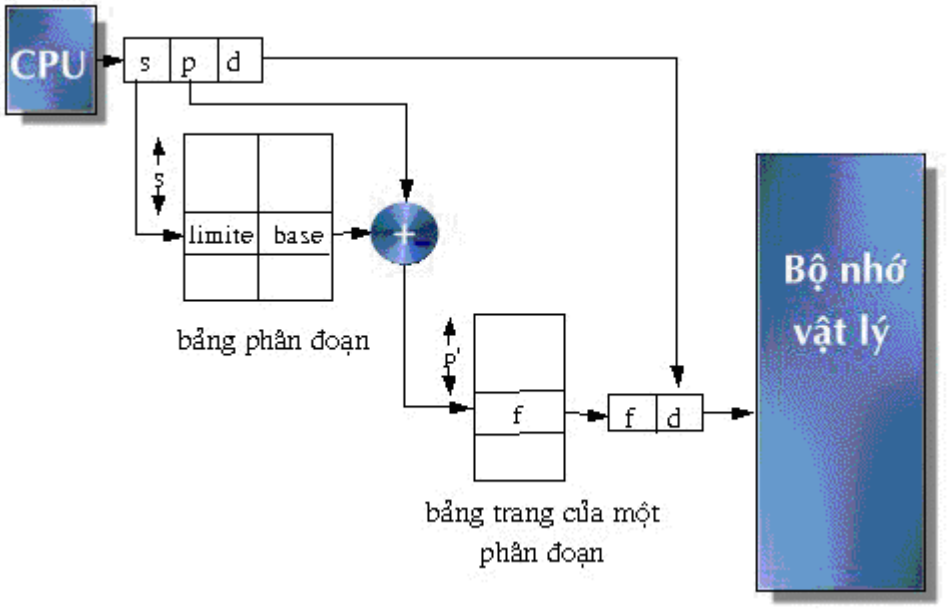
số hiệu phân đoạn (s): sử dụng như chỉ mục đến phần tử tương ứng trong bảng phân đoạn.



Hình 3.23 Mô hình phân đoạn kế hợp phân trang

số hiệu trang (p): sử dụng như chỉ mục đến phần tử tương ứng trong bảng trang của phân đoạn.

địa chỉ tương đối trong trang (d): kết hợp với địa chỉ bắt đầu của khung trang để tạo ra địa chỉ vật lý mà trình quản lý bộ nhớ sử dụng.



Hình 3.24 Cơ chế phần cứng của sự phân đoạn kết hợp phân trang

Tất cả các mô hình tổ chức bộ nhớ trên đây đều có khuynh hướng cấp phát cho tiến trình toàn bộ các trang yêu cầu trước khi thật sự xử lý. Vì bộ nhớ vật lý có kích thước rất giới hạn, điều này dẫn đến hai điểm bất tiện sau :

Kích thước tiến trình bị giới hạn bởi kích thước của bộ nhớ vật lý.

Khó có thể bảo trì nhiều tiến trình cùng lúc trong bộ nhớ, và như vậy khó nâng cao mức độ đa chương của hệ thống.

3.6 Kỹ thuật bộ nhớ ảo (Virtual Memory)

3.6.1. Bộ nhớ ảo

Kỹ thuật phân trang, phân đoạn còn một hạn chế lớn là phải nạp tất cả các trang/đoạn của một tiến trình vào bộ nhớ để tiến trình này hoạt động. Điều này làm cản trở mục tiêu của hệ điều hành là phải nạp được nhiều tiến trình của các chương trình khác nhau vào bộ nhớ để chúng có thể hoạt động đồng thời với nhau, trong thực trạng kích thước của chương trình càng lớn. Ngoài ra việc nạp tất cả các trang/đoạn này đều cần thiết để tiến trình này có thể hoạt động được.

Để khắc phục hạn chế của kỹ thuật phân trang/ phân đoạn, kỹ thuật bộ nhớ ảo ra đời. Nguyên lý cơ bản của bộ nhớ ảo là vẫn dựa trên 2 kỹ thuật phân trang và phân đoạn, nhưng trong kỹ thuật bộ nhớ ảo:

-Bộ phận quản lý bộ nhớ không nạp tất cả các trang/đoạn của một tiến trình vào bộ nhớ để nó hoạt động, mà chỉ nạp các trang/đoạn cần thiết tại thời điểm khởi tạo. Sau đó, khi cần bộ phận quản lý bộ nhớ sẽ dựa vào bảng trang, hoặc bảng đoạn của mỗi tiến trình để nạp các trang/đoạn tiếp theo.

-Nếu có một trang/đoạn của một tiến trình cần được nạp vào bộ nhớ chính trong tình trạng trên bộ nhớ không còn khung trang/phân đoạn trống thì bộ phận quản lý bộ nhớ sẽ đưa một trang/đoạn không cần thiết tại thời điểm hiện tại ra bộ nhớ ngoài, để lấy không gian trống đó nạp trang/đoạn vừa có yêu cầu. Trang/đoạn bị swap out sẽ được đưa vào tại thời điểm thích hợp hoặc cần thiết sau này (swap-in)

Bộ nhớ ảo là một kỹ thuật cho phép xử lý một tiến trình không được nạp toàn bộ vào bộ nhớ vật lý. Các trang/đoạn của một tiến trình, đang ở trên bộ nhớ phụ mà chưa được nạp vào bộ nhớ chính sẽ được định vị tại một không gian nhớ đặc biệt trên bộ nhớ phụ, có thể gọi không gian nhớ này là bộ nhớ ảo của tiến trình. Với sự hỗ trợ phần cứng hệ điều hành đã đưa ra các cơ chế thích hợp để nhận biết một trang/đoạn của tiến trình đang thực hiện là đang ở trên bộ nhớ chính hay bộ nhớ phụ. Như vậy bộ nhớ ảo đã mở rộng (ảo) được không gian bộ nhớ vật lý của hệ thống, chương trình của người sử dụng chỉ nhìn thấy và làm việc trên không gian địa chỉ ảo, việc chuyển đổi từ

địa chỉ ảo sang địa chỉ vật lý thực do bộ phận quản lý bộ nhớ của hệ điều hành và processor thực hiện.

Cần kết hợp kỹ thuật *swapping* đển chuyển các phần của chương trình vào-ra giữa bộ nhớ chính và bộ nhớ phụ khi cần thiết.

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

-Hệ điều hành có thể nạp nhiều tiến trình hơn vào bộ nhớ, trên bộ nhớ tồn tại các trang/đoạn của các tiến trình khác nhau. Hệ thống khó có thể xảy ra trường hợp không đủ bộ nhớ để nạp các tiến trình khác.

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

3.6.2. Cài đặt bộ nhớ ảo

Vì vậy hệ điều hành có thể cài đặt bộ nhớ ảo theo 2 kỹ thuật:

-Phân trang theo yêu cầu: Tức là phân trang kết hợp với swap

-Phân đoạn theo yêu cầu: Tức là phân đoạn kết hợp với swap

Cả hai kỹ thuật trên đều phải có sự hỗ trợ của phần cứng máy tính, cụ thể là processor. Đa số các hệ điều hành đều chọn kỹ thuật phân trang theo yêu cầu, vì nó đơn giản, dễ cài đặt và chi phí thấp hơn

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

-Một lượng không gian bộ nhớ phụ (đĩa) cần thiết đủ để chứa các trang/đoạn bị swap out, không gian đĩa này được gọi là không gian swap.

-Có cơ chế để theo dõi các trang/đoạn của một tiến trình, của tất cả các tiến trình đang hoạt động trên bộ nhớ chính, là đang ở trên bộ nhớ chính hay trên bộ nhớ phụ. Trong trường hợp này hệ điều hành thường đưa thêm một bít trạng thái (present) vào các phần tử trong bảng trang hoặc bảng đoạn.

-Dựa vào các tiêu chuẩn cụ thể để chọn một trang/đoạn nào đó trong số các trang/đoạn đang nằm trong bộ nhớ chính để swap out trong trường hợp cần thiết.

Phân trang theo yêu cầu (demand paging)

Một hệ thống phân trang theo yêu cầu là hệ thống sử dụng kỹ thuật phân trang kết hợp với kỹ thuật swapping. Một tiến trình được xem như một tập các trang, thường trú trên bộ nhớ phụ (thường là đĩa). Khi cần xử lý, tiến trình sẽ được nạp vào bộ nhớ chính. Nhưng thay vì nạp toàn bộ chương trình, chỉ những trang cần thiết trong thời điểm hiện tại mới được nạp vào bộ nhớ. Như vậy một trang chỉ được nạp vào bộ nhớ chính khi có yêu cầu.

Với mô hình này, cần cung cấp một cơ chế phần cứng giúp phân biệt các trang đang ở trong bộ nhớ chính và các trang trên đĩa. Có thể sử dụng lại bit *valid-invalid* nhưng với ngữ nghĩa mới:

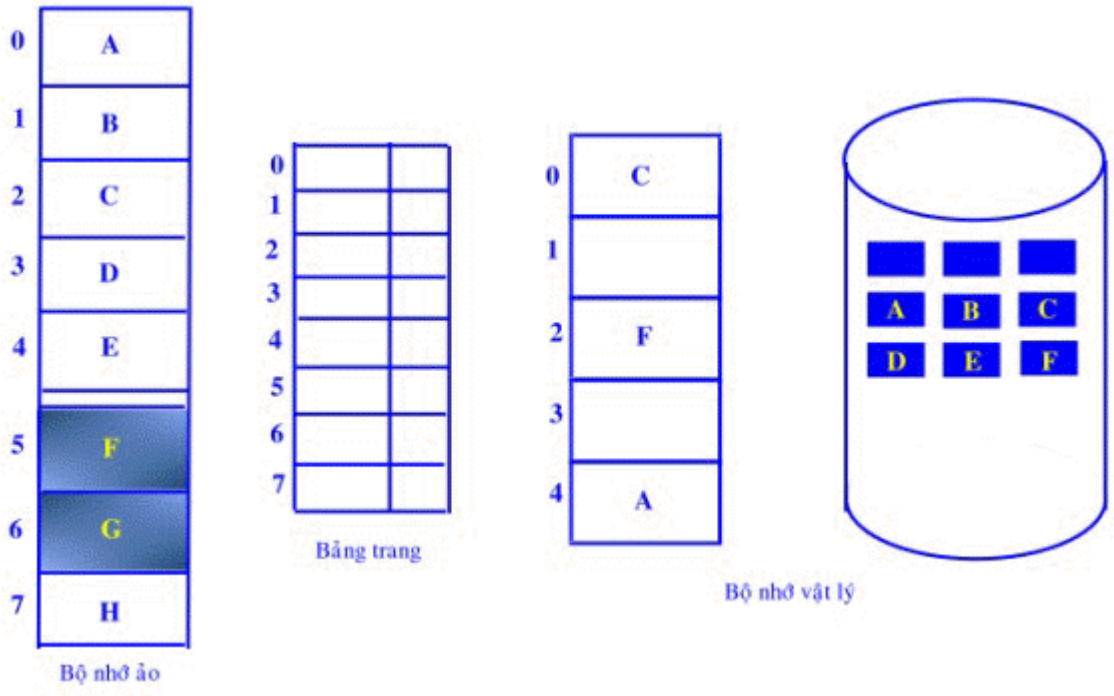
valid : trang tương ứng là hợp lệ và đang ở trong bộ nhớ chính .

invalid : hoặc trang bất hợp lệ (không thuộc về không gian địa chỉ của tiến trình) hoặc trang hợp lệ nhưng đang được lưu trên bộ nhớ phụ.

Một phần tử trong bảng trang mô tả cho một trang không nằm trong bộ nhớ chính, sẽ được đánh dấu *invalid* và chứa địa chỉ của trang trên bộ nhớ phụ.

Cơ chế phần cứng :

Cơ chế phần cứng hỗ trợ kỹ thuật phân trang theo yêu cầu là sự kết hợp của cơ chế hỗ trợ kỹ thuật phân trang và kỹ thuật swapping:



Hình 3.26 Bảng trang với một số trang trên bộ nhớ phụ

Bảng trang: Cấu trúc bảng trang phải cho phép phản ánh tình trạng của một trang là đang nằm trong bộ nhớ chính hay bộ nhớ phụ.

Bộ nhớ phụ: Bộ nhớ phụ lưu trữ những trang không được nạp vào bộ nhớ chính. Bộ nhớ phụ thường được sử dụng là đĩa, và vùng không gian đĩa dùng để lưu trữ tạm các trang trong kỹ thuật swapping được gọi là *không gian swapping.*

Lỗi trang

Truy xuất đến một trang được đánh dấu bất hợp lệ sẽ làm phát sinh một *lỗi trang* (*page fault*). Khi dò tìm trong bảng trang để lấy các thông tin cần thiết cho việc chuyển đổi địa chỉ, nếu nhận thấy trang đang được yêu cầu truy xuất là bất hợp lệ, cơ chế phần cứng sẽ phát sinh một ngắt để báo cho hệ điều hành. Hệ điều hành sẽ xử lý lỗi trang như sau :

Kiểm tra truy xuất đến bộ nhớ là hợp lệ hay bất hợp lệ

Nếu truy xuất bất hợp lệ : kết thúc tiến trình

Ngược lại : đến bước 3

Tìm vị trí chứa trang muốn truy xuất trên đĩa.

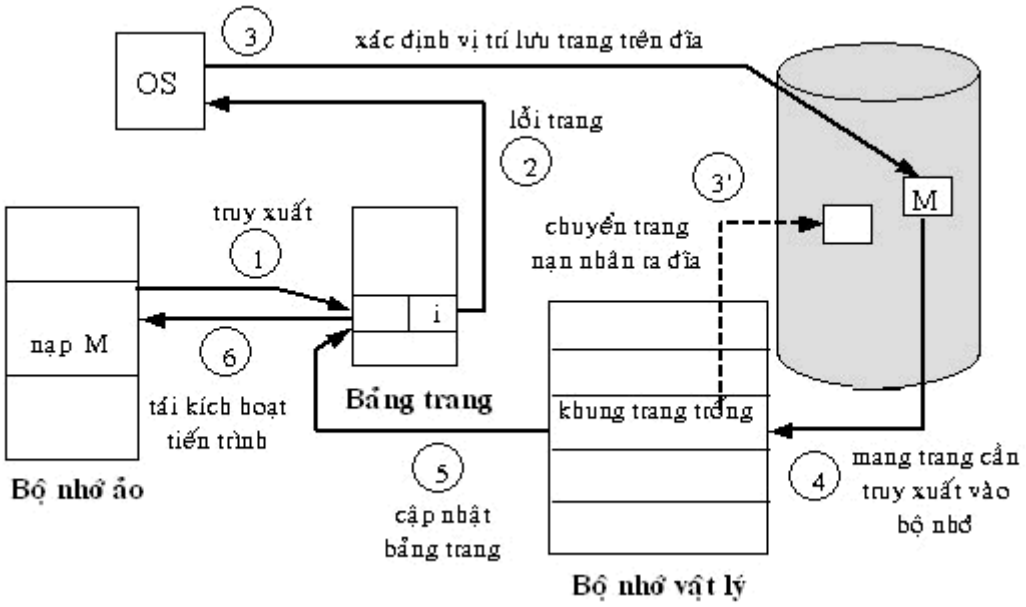
Tìm một khung trang trống trong bộ nhớ chính :

Nếu tìm thấy : đến bước 5

Nếu không còn khung trang trống, chọn một khung trang « nạn nhân » và chuyển trang « nạn nhân » ra bộ nhớ phụ (lưu nội dung của trang đang chiếm giữ khung trang này lên đĩa), cập nhật bảng trang tương ứng rồi đến bước 5

Chuyển trang muốn truy xuất từ bộ nhớ phụ vào bộ nhớ chính : nạp trang cần truy xuất vào khung trang trống đã chọn (hay vừa mới làm trống) ; cập nhật nội dung bảng trang, bảng khung trang tương ứng.

Tái kích hoạt tiến trình người sử dụng.



Hình 3.27 Các giai đoạn xử lý lỗi trang

3.6.3.Các thuật toán thay thế trang

Khi xảy ra một lỗi trang, cần phải mang trang vắng mặt vào bộ nhớ . Nếu không có một khung trang nào trống, hệ điều hành cần thực hiện công việc *thay thế trang* – chọn một trang đang nằm trong bộ nhớ mà không được sử dụng tại thời điểm hiện tại và chuyển nó ra *không gian swapping* trên đĩa để giải phóng một khung trang dành chỗ nạp trang cần truy xuất vào bộ nhớ.

Như vậy nếu không có khung trang trống, thì mỗi khi xảy ra lỗi trang cần phải thực hiện hai thao tác chuyển trang : chuyển một trang ra bộ nhớ phụ và nạp một trang khác vào bộ nhớ chính. Có thể giảm bớt số lần chuyển trang bằng cách sử dụng thêm một bit *cập nhật* (dirty bit). Bit này được gắn với mỗi trang để phản ánh tình trạng trang có bị cập nhật hay không : giá trị của bit được cơ chế phần cứng đặt là 1 mỗi lần có một từ được ghi vào trang, để ghi nhận nội dung trang có bị sửa đổi. Khi cần thay thế một trang, nếu bit cập nhật có giá trị là 1 thì trang cần được lưu lại trên đĩa, ngược lại, nếu bit cập nhật là 0, nghĩa là trang không bị thay đổi, thì không cần lưu trữ trang trở lại đĩa.

số hiệu trang	bit valid-invalid	dirty bit

Hình 3.28 Cấu trúc một phần tử trong bảng trang

Sự thay thế trang là cần thiết cho kỹ thuật phân trang theo yêu cầu. Nhờ cơ chế này, hệ thống có thể hoàn toàn tách rời bộ nhớ ảo và bộ nhớ vật lý, cung cấp cho lập trình viên một bộ nhớ ảo rất lớn trên một bộ nhớ vật lý có thể bé hơn rất nhiều lần.

Sự thi hành phân trang theo yêu cầu

Việc áp dụng kỹ thuật phân trang theo yêu cầu có thể ảnh hưởng mạnh đến tình hình hoạt động của hệ thống.

Gỉa sử *p* là xác suất xảy ra một lỗi trang (0<= p <=1):

p = 0 : không có lỗi trang nào

p = 1 : mỗi truy xuất sẽ phát sinh một lỗi trang

Thời gian thật sự cần để thực hiện một truy xuất bộ nhớ (TEA) là:

TEA = (1-p)ma + p (tdp) [+ swap out] + swap in + tái kích hoạt

Trong công thức này, *ma* là thời gian truy xuất bộ nhớ, *tdp* thời gian xử lý lỗi trang.

Có thể thấy rằng, để duy trì ở một mức độ chấp nhận được sự chậm trễ trong hoạt động của hệ thống do phân trang, cần phải duy trì *tỷ lệ phát sinh lỗi trang* thấp.

Hơn nữa, để cài đặt kỹ thuật phân trang theo yêu cầu, cần phải giải quyết hai vấn đề chính yếu : xây dựng một *thuật toán cấp phát khung trang*, và *thuật toán thay thế trang*.

Các thuật toán thay thế trang

Vấn đề chính khi thay thế trang là chọn lựa một trang « nạn nhân » để chuyển ra bộ nhớ phụ. Có nhiều thuật toán thay thế trang khác nhau, nhưng tất cả cùng chung một mục tiêu : chọn trang « nạn nhân » là trang mà sau khi thay thế sẽ gây ra ít lỗi trang nhất.

Có thể đánh giá hiệu qủa của một thuật toán bằng cách xử lý trên một *chuỗi các địa chỉ cần truy xuất* và tính toán số lượng lỗi trang phát sinh.

Ví dụ: Giả sử theo vết xử lý của một tiến trình và nhận thấy tiến trình thực hiện truy xuất các địa chỉ theo thứ tự sau :

0100, 0432, 0101, 0162, 0102, 0103, 0104, 0101, 0611, 0102, 0103,0104, 0101, 0610, 0102, 0103, 0104, 0101, 0609, 0102, 0105

Nếu có kích thước của một trang là 100 bytes, có thể viết lại *chuỗi truy xuất* trên giản lược hơn như sau :

1, 4, 1, 6, 1, 6, 1, 6, 1

Để xác định số các lỗi trang xảy ra khi sử dụng một thuật toán thay thế trang nào đó trên một chuỗi truy xuất cụ thể, còn cần phải biết số lượng khung trang sử dụng trong hệ thống.

Để minh hoạ các thuật toán thay thế trang sẽ trình bày, chuỗi truy xuất được sử dụng là :

7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

a) Thuật toán FIFO

Tiếp cận: Ghi nhận thời điểm một trang được mang vào bộ nhớ chính. Khi cần thay thế trang, trang ở trong bộ nhớ lâu nhất sẽ được chọn

Ví dụ : sử dụng 3 khung trang , ban đầu cả 3 đều trống :

7	**0**	**1**	**2**	**0**	**3**	**0**	**4**	**2**	**3**	**0**	**3**	**2**	**1**	**2**	**0**	**1**	**7**	**0**	**1**
7	7	7	2	2	2	2	4	4	4	0	0	0	0	0	0	0	7	7	7
	0	0	0	0	3	3	3	2	2	2	2	2	1	1	1	1	1	0	0
		1	1	1	1	0	0	0	3	3	3	3	3	2	2	2	2	2	1
*	*	*	*		*	*	*	*	*	*			*	*			*	*	*

Ghi chú : * : có lỗi trang

Thảo luận:

Để áp dụng thuật toán FIFO, thực tế không nhất thiết phải ghi nhận thời điểm mỗi trang được nạp vào bộ nhớ, mà chỉ cần tổ chức quản lý các trang trong bộ nhớ trong một danh sách FIFO, khi đó trang đầu danh sách sẽ được chọn để thay thế.

Thuật toán thay thế trang FIFO dễ hiểu, dễ cài đặt. Tuy nhiên khi thực hiện không phải lúc nào cũng có kết quả tốt : trang được chọn để thay thế có thể là trang chức nhiều dữ liệu cần thiết, thường xuyên được sử dụng nên được nạp sớm, do vậy khi bị chuyển ra bộ nhớ phụ sẽ nhanh chóng gây ra lỗi trang.

Số lượng lỗi trang xảy ra sẽ tăng lên khi số lượng khung trang sử dụng tăng. Hiện tượng này gọi là *nghịch lý Belady*.

Ví dụ: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

Sử dụng 3 khung trang , sẽ có 9 lỗi trang phát sinh

1	2	3	4	1	2	5	1	2	3	4	5
1	1	1	4	4	4	5	5	5	5	5	5
	2	2	2	1	1	1	1	1	3	3	3
		3	3	3	2	2	2	2	2	4	4
*	*	*	*	*	*	*			*	*	

Sử dụng 4 khung trang , sẽ có 10 lỗi trang phát sinh

1	2	3	4	1	2	5	1	2	3	4	5
1	1	1	1	1	1	5	5	5	5	4	4
	2	2	2	2	2	2	1	1	1	1	5
		3	3	3	3	3	3	2	2	2	2
			4	4	4	4	4	4	3	3	3
*	*	*	*			*	*	*	*	*	*

b) Thuật toán tối ưu

Tiếp cận: Thay thế trang sẽ lâu được sử dụng nhất trong tương lai.

Ví dụ : sử dụng 3 khung trang, khởi đầu đều trống:

7	**0**	**1**	**2**	**0**	**3**	**0**	**4**	**2**	**3**	**0**	**3**	**2**	**1**	**2**	**0**	**1**	**7**	**0**	**1**
7	7	7	2	2	2	2	2	2	2	2	2	2	2	2	2	2	7	7	7
	0	0	0	0	0	0	4	4	4	0	0	0	0	0	0	0	0	0	0
		1	1	1	3	3	3	3	3	3	3	3	1	1	1	1	1	1	1
*	*	*	*		*		*			*			*				*		

Thảo luận:

Thuật toán này bảo đảm số lượng lỗi trang phát sinh là thấp nhất, nó cũng không gánh chịu nghịch lý Belady, tuy nhiên, đây là một thuật toán không khả thi trong thực tế, vì không thể biết trước chuỗi truy xuất của tiến trình!

c) Thuật toán « Lâu nhất chưa sử dụng » (Least-recently-used LRU)

Tiếp cận: Với mỗi trang, ghi nhận thời điểm cuối cùng trang được truy cập, trang được chọn để thay thế sẽ là trang lâu nhất chưa được truy xuất.

Trang được hệ điều hành chọn để thay thế là trang có khoảng thời gian từ lúc nó được truy xuất gần đây nhất đến thời điểm hiện tại là dài nhất, so với các trang đang trên bộ nhớ chính.

Ví dụ: sử dụng 3 khung trang, khởi đầu đều trống:

7	**0**	**1**	**2**	**0**	**3**	**0**	**4**	**2**	**3**	**0**	**3**	**2**	**1**	**2**	**0**	**1**	**7**	**0**	**1**
7	7	7	2	2	2	2	4	4	4	0	0	0	1	1	1	1	1	1	1
	0	0	0	0	0	0	0	0	3	3	3	3	3	3	0	0	0	0	0
		1	1	1	3	3	3	2	2	2	2	2	2	2	2	2	7	7	7
*	*	*	*		*		*	*	*	*			*		*		*		

Thảo luận:

Thuật toán FIFO sử dụng thời điểm nạp để chọn trang thay thế, thuật toán tối ưu lại dùng thời điểm trang *sẽ* được sử dụng, vì thời điểm này không thể xác định trước nên thuật toán LRU phải dùng thời điểm cuối cùng trang được truy xuất – dùng quá khứ gần để dự đoán tương lai.

Thuật toán này đòi hỏi phải được cơ chế phần cứng hỗ trợ để xác định một thứ tự cho các trang theo thời điểm truy xuất cuối cùng. Có thể cài đặt theo một trong hai cách

Chương 4 QUẢN LÝ VÙNG NHỚ PHỤ

Máy tính phải sử dụng thiết bị có khả năng lưu trữ trong thời gian dài (long-time) vì:

Phải chứa những lượng thông tin rất lớn (giữ vé máy bay, ngân hàng…).

Thông tin phải được lưu trữ một thời gian dài trước khi xử lý.

Nhiều tiến trình có thể truy cập thông tin cùng lúc.

Giải pháp là sử dụng các thiết bị lưu trữ bên ngoài gọi là bộ nhớ ngoài. Bao gồm: ổ cứng, đĩa mềm, Đĩa CD,Flash disk

4.1 Một số khái niệm dùng quản lý đĩa

Lưu trữ dữ liệu trên bề mặt các đĩa phủ vật liệu từ tính.

Track (rãnh từ): Là các vòng đường tròn đồng tâm được tạo ra trên bề mặt đĩa, đây sẽ là nơi chứa dữ liệu sau này. Các track được đánh số bắt đầu từ 0. Số track trên mỗi mặt đĩa phụ thuộc vào từng loại đĩa.

Sector (cung từ): Các track được chia thành các khối có kích thước cố định bằng nhau và được đánh địa chỉ, các khối này được gọi là các sector. Các sector được đánh địa chỉ bắt đầu từ 1 trên mỗi track, như vậy trên đĩa sẽ tồn tại nhiều sector có cùng số hiệu địa chỉ, cách đánh địa chỉ này gây khó khăn nhiều người lập trình.

Kích thước của sector, số byte dữ liệu có thể chứa trên một sector, phụ thuộc vào phần cứng. Trên các họ processor x86, kích thước sector trên đĩa cứng thường là 512 byte, kích thước sector trên đĩa CD_ROM thường là 2048 byte.

Các sector được đánh địa chỉ theo kiểu trên được gọi là các sector vật lý. Trong thực tế lập trình, các hệ điều hành chỉ sử dụng sector logic, theo đó địa chỉ các sector được đánh số bắt đầu từ 0 kể từ track 0 của mặt 0 trên đĩa thứ nhất. Như vậy trên đĩa không có các sector có cùng số hiệu địa chỉ. Bảng sau đây cho thấy sự tương ứng giữa các sector vật lý với các sector logic trên một đĩa mềm:

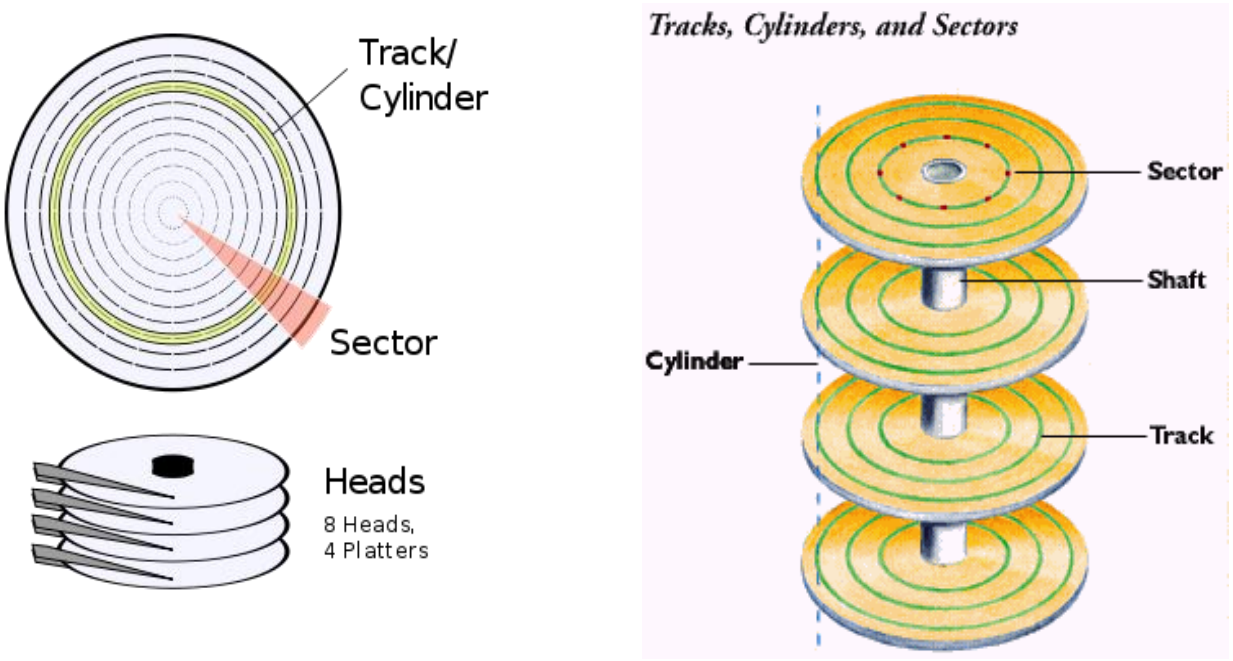
Mặt đĩa	Track	Sector	Sector logic	Thông tin lưu trữ
0	0	1	0	Boot record
0	0	2-5	1-4	FAT
0	0	6-9	5-8	Thư mục gốc
1	0	1-3	9-11	Thư mục gốc
1	0	4-9	12-17	Dữ liệu
0	1	1-9	18-26	Dữ liệu

Bảng: Tương ứng giữa sector vật lý và sector logic trên đĩa mềm

Trên bề mặt đĩa tồn tại các sector mà hệ điều hành không thể ghi dữ liệu vào đó hoặc không thể đọc dữ liệu từ đó. Các sector này được gọi là các bad sector. Trong quá trình định dạng đĩa hệ điều hành đánh dấu loại bỏ các bad sector này.

Cylinder(từ trụ): Tập hợp các track có cùng số hiệu trên các mặt đĩa khác nhau của một hệ thống đĩa tạo thành một cylinder. Như vậy mặt đĩa có bao nhiêu track thì có bấy nhiêu cylinder. Cylinder chỉ có trên các ổ dĩa cứng.

Trên một ổ cứng có nhiều cylinder



Hình 4.6

Cluster (liên cung): Một nhóm gồm 2, 4 hoặc 6sector liên tiếp nhau tạo thành một cluster. Kích thước của cluster thường là bội số kích thước một sector. Các cluster được đánh địa chỉ bắt đầu từ 0. Số sector trên một cluster phụ thuộc vào từng loại đĩa. Một số hệ điều hành cho phép người sử dụng quy định số sector trên một cluster. Các hệ điều hành thường tổ chức lưu trữ dữ liệu, nội dung các tập tin trên đĩa theo từng cluster. Trên bề mặt đĩa cũng tồn tại các bad cluster, đó là các cluster có chứa bad sector.

Partition (phân khu): Partition là một tập các sector liền kề trên một đĩa. Mỗi partition có một bảng partition hoặc một cơ sở dữ liệu quản lý đĩa riêng, dùng để lưu trữ sector đầu tiên, kích thước và các đặc tính khác của partition.

Volume: Một volume tương tự một partition logic trên một đĩa, và nó được tạo khi ta định dạng một đĩa hoặc một phần của đĩa theo hệ thống File NTFS. Trong hệ điều hành windowsNT/2000 ta có thể tạo ra một volume trải dài trên nhiều đĩa vật lý khác nhau. Một đĩa có thể có một hoặc nhiều volume. NTFS điều khiển mỗi volume sao cho không phụ thuộc vào các volume khác.

Một volume bao gồm một tập các file cùng với bất kỳ một không gian chưa được cấp phát còn lại trên partition đĩa. Trong hệ thống file FAT, một volume cũng chứa các vùng đặc biệt được định dạng cho việc sử dụng của hệ thống file, như là bitmap, directory và cả system bootstrap trên các file.

Simple volume: là các đối tượng đại diện cho các sector từ một partition đơn, mà các trình điều khiển hệ thống file, quản lý nó như một đơn vị.

Multipartition volume: là các đối tượng đại diện cho các sector từ nhiều partition khác nhau, mà các trình điều khiển hệ thống file quản lý nó như một đơn vị. Các multipartition volume có các đặc tính mà các simple volume không có được như: hiệu suất cao, độ tin cậy cao và khả năng mở rộng kích thước.

Metadata: là một dạng dữ liệu đặc biệt, được lưu trữ trên đĩa, nó hỗ trợ cho các thành phần quản lý các dạng thức hệ thống file khác nhau, dữ liệu của nó có thể là vị trí của các tập tin/thư mục trên ổ đĩa. Metadata không được sử dụng trong các ứng dụng.

File system (hệ thống file): Các dạng thức hệ thống file định nghĩa cách mà dữ liệu file được lưu trữ trên thiết bị lưu trữ và sự tác động của hệ thống file đến các file. Một dạng thức hệ thống file cũng có thể đưa ra các giới hạn về kích thước của các file và các thiết bị lưu trữ mà hệ thống file hỗ trợ. Một vài hệ thống file hỗ trợ cho các file lớn nhỏ, hoặc cả các đĩa lớn và nhỏ.

Một hệ thống file thường bao gồm các thành phần: Sector khởi động, bảng định vị File, bảng thư mục gốc, một tập các file các thư mục và các công cụ quản lý các thành phần này. Các thành phần này có thể có cấu trúc hoặc phương thức tổ chức khác nhau trên các dạng hệ thống file khác nhau. Người ta thường dùng tên của FAT trong hệ thống File để gọi tên của hệ thống file đó.

Hệ điều hành MS_DOS sử dụng hệ thống File FAT 12 và FAT16, hệ điều hành Windows9x sự dụng hệ thống file FAT32 và CDFS, hệ điều hành WindowsNT và Windows 2000 sử dụng các hệ thống file FAT12, FAT16, FAT32, CDFS (CD_ROM File System) UDF (Universal Disk Format) và NTFS(New Technology File System)

- Format

Format cấp thấp (low format) định dạng lại các track, sector, cylinder

Format thông thường: định dạng mức cao (high-level format)

Format nhanh (xóa các kí tự lưu trữ đầu tiên của hdh hay phần mềm)

Format thường (Xóa dữ liệu và kiểm tra khối hư hỏng (bad block))

Đĩa mềm

Mặt	Rãnh	Sector	ý nghĩa
0	0	1	Bootsector
0	0	2,3	FAT1(File Allocation Table)
0	0	4, 5	FAT2(dành trường hợp FAT1 hỏng)
0	0	6,7,8,9	Root directory
1	0	1,2,3	Root directory

Đĩa cứng

Mặt	Rãnh	Sector	ý nghĩa
0	0	1	Bootsector
1	0	1	Cung khởi động

Bảng các phân khu được tạo

Offset	Nội dung	Kích thước
1BEh	Partition1 entry	16 byte
1CEh	Partition1 entry	16 byte
1DEh	Partition1 entry	16 byte
1EEh	Partition1 entry	16 byte

Nội dung 16 byte

địa chỉ	Kích thước	Nội dung
00	1 byte	địa chỉ khởi động
01	3 byte	địa chỉ đầu phân khu
05	3 byte	địa chỉ cuối phân khu
08	4 byte	Số cung trước phân khu
0C	4 byte	Số cung trong phân khu
04	1 byte	Chỉ thị hệ thống

Quản lý không gian đĩa

Để tổ chức lưu trữ nội dung các file trên đĩa, các hệ điều hành đều chia không gian lưu trữ của đĩa thành các phần có kích thước bằng nhau được gọi là khối(block) lưu trữ. Nội dung của file cũng được chia thành các block có kích thước bằng nhau, trừ block cuối cùng, và bằng các kích thước block đĩa. Khi cần lưu trữ File trên đĩa hệ điều hành cấp cho mỗi tập tin một số lượng block vừa đủ để chứa hết nội dung của tập tin, các block đĩa này có thể nằm tại vị trí bất kỳ trên đĩa. Trong quá trình sử dụng file kích thước của file có thể thay đổi, tăng lên hay giảm xuống, do hệ điều hành phải tổ chức cấp phát động các block đĩa cho các file. Khi kích thước của file tăng lên thì hệ điều hành phải cấp phát them block cho nó, khi kích thước file giảm xuống hoặc khi file bị xóa khỏi đĩa thì hệ điều hành phải thu hồi lại các block đĩa đã cấp cho nó để cấp cho các file khác sau này.

Để tổ chức cấp phát động các block đĩa cho file, hệ điều hành phải quản lý được trạng thái của các block, còn tự do hay đã cấp phát trên đĩa. Trong trường hợp này các hệ điều hành có thể sử dụng 2 kỹ thuật: Dùng bảng bit và/hoặc dùng danh sách liên kết.

4.2 Hệ thống bảng FAT

4.2.1 Quản lý file trên đĩa của MS_DOS

Trong quá trình định dạng một đĩa mềm, hoặc một đĩa logic trên các phân khu trên đĩa cứng, hệ điều hành chia không gian lưu trữ của đĩa thành 2 vùng: vùng dữ liệu(Data Area) và vùng hệ thống (System Area), đồng thời ghi những thông tin cần thiết vào vùng hệ thống để chuẩn bị cho việc quản lý lưu trữ sau này.

-Vùng dữ liệu: bao gồm các block có kích thước bằng nhau và được đánh địa chỉ (12 bit hoặc 16 bit) để phân biệt, đây chính là các cluster trên đĩa mà ta đã nói ở trên. Nội dung của tập tin cũng được chia thành các block có kích thước bằng kích thước của một cluster. Các cluster trên đĩa dùng để chứa nội dung của các tập tin trên đĩa. Các thông tin liên quan đến một tập tin trên đĩa được chứa ở vùng hệ thống.

-Vùng hệ thống: bao gồm các đoạn chương trình, các thông tin hệ thống, các thông tin liên quan đến các tập tin/thư mục trên đĩa mà hệ điều hành dùng để quản lý việc lưu trữ tập tin/thư mục trên đĩa sau này. Cụ thể nó bao gồm các thành phần sau đây: Boot sector, FAT1, FAT2 và Root Directory.

Sau đây chúng ta sẽ khảo sát các thành phần trong vùng hệ thống, để thấy được cách mà DOS quản lý các file và các file và các thư mục được lưu trữ trên đĩa.

-Bootsector: còn được gọi là boot record (bản ghi khởi động), dài 512 byte (1 sector) được đặt tại sector logic 0 trên đĩa mềm hay sector logic đầu tiên của partition (đĩa logic) trên ổ đĩa cứng. Tất cả các đĩa (FDD và đĩa logic trên đĩa cứng) sau khi được định dạng đều có boot record và đều có chứa các thông tin liên quan về đĩa trong đó, nhưng chỉ có đĩa được định dạng là đĩa khởi động mới có chứa một đoạn code Bootstrap Loader. Bootstrap Loader thực hiện việc nạp thành phần cốt lõi của DOS như io.sys, msdos.sys, command.com vào bọ nhớ RAM (chính xác hơn là chỉ nạp io.sys vào Ram sau đó io.sys sẽ tìm nạp các tập tin tiếp theo) trong quá trình khởi động máy tính. Chính vì vậy Bootstrap Loader còn được gọi là chương tình mồi khởi động. Bảng sau đây cho thấy vị trí, độ lớn và nội dung của các trường trong bootsector

BPB (BIOS Parameters block)

Offset	size	ý nghĩa
00h	3 byte	lệnh JUMP, nhẩy về Bootstrap Loader
03h	8 byte	Tên nhà sản xuất và số phiên bản
0Bh	2 byte	Số byte trên một sector
0Dh	1 byte	Số sector trên một cluster
0Eh	2 byte	Số sector dành cho bootsector
10h	1 byte	Số bảng FAT
11h	2 byte	Số phần tử trong thư mục gốc
13h	2 byte	Chỉ tổng số sector trên một tập đĩa (volume)
15h	1byte	Nhận khuôn dạng đĩa (F8: đĩa cứng)
16h	2 byte	Số sector dành cho bảng FAT
18h	2 byte	Số sector trên một track
1Ah	2 byte	Số mặt (đầu từ)
1Ch	4 byte	Số sector dự trữ
1Eh	4 byte	Số sector nếu kích thước >32Mb
22h	1 byte	Số hiệu ổ đĩa (đĩa mềm:0, đĩa cứng: 80)
23h	1 byte	Byte dự trữ
24 h	1 byte	chữ ký của Bootsector mở rộng
25h	4 byte	Số serial của đĩa được tạo ra lúc format
29h	11 byte	Nhãn đĩa
34h	8 byte	Loại FAT: “FAT 12” hoặc “FAT 16”
3Ch-200h	452 byte	Code của chương trình bootstrap loader

VD Bootsector đĩa cứng

EB 3C 90 4D 53 57 49 4E 34 2E 31 M S W I N 4 1 0 0 0 2

Như vậy, ngay sau khi quyền điều khiển được trả về cho bootsector thì hệ thống thì hệ thống sẽ thực hiện lệnh nhẩy (Jmp) ở đầu bootsector (offset 00), để nhảy đến thực hiện đoạn code bootstrap loader ở cuối boot sector (từ offset 3Ch đến offset 200h). Và bootstrap loader sẽ thực hiện nhiệm vụ của nó.

b) FAT

Nội dung của một File cần lưu trữ trên đĩa được chia thành các phần có kích thước bằng nhau và bằng kích thước của một cluster, được gọi là các block file. Các block file của các file được lưu trữ tại các cluster xác định trên đĩa, các cluster chứa nội dung một file có thể không nằm kề nhau. Để theo dõi danh sách các cluster đang chứa nội dung của một file của tất cả các file đang lưu giữ trên đĩa hệ điều hành DOS dùng bảng FAT, hay còn gọi là bảng định vị File. Bảng Fat còn dùng để ghi nhận trạng thái của các cluster trên đĩa: còn trống, đã cấp phát cho các file, bị bad không thể sử dụng hay dành riêng vào hệ điều hành. Trong quá trình khởi động máy tính hệ điều hành nạp bảng FAT vào bộ nhớ để chuẩn bị cho việc đọc/ghi các file sau này.

Khi cần ghi nội dung của một file vào đĩa hoặc đọc nội dung của một file trên đĩa hệ điều hành phải dựa vào bảng FAT, nếu bảng FAT bị hỏng thì hệ điều hành không thể ghi/đọc các file trên đĩa. Do đó, hệ điều hành DOS tạo ra hai bảng FAT hoàn toàn giống nhau là FAT1, FAT2, DOS sử dụng bảng FAT1 và dự phòng FAT2, nếu FAT1 bị hỏng thì DOS sẽ dùng FAT2 để khôi phục lại FAT1. Điều này không đúng với hệ thống file FAT32, FAT32 vẫn tạo ra 2 FAT của DOS, nhưng nếu FAT1 bị hỏng thì hệ điều hành sẽ chuyển sang sử dụng FAT2, sau đó mới khôi phục FAT1, và ngược lại.

Hệ điều hành DOS tổ chức cấp phát động các cluster các cluster cho các file trên đĩa, sau mỗi thao tác cấp phát/thu hồi cluster thì hệ điều hành phai cập nhật nội dung cho cả FAT1 và FAT2. Có thể hệ điều hành chỉ thực hiện cấp phát động cluster cho các file dữ liệu (có kích thước thay đổi), còn đối với các file chương trình, file thư viện, file liên kết động…(có kích thước không thể thay đổi) thì hệ điều hành sẽ thực hiện cấp tĩnh cluster cho nó.

Bảng FAT bao gồm nhiều phần tử (điểm nhập/mục vào), các phần tử được đánh địa chỉ bắt đầu từ 0 để phân biệt. Giá trị dữ liệu tại một phần tử trong bảng FAT cho biết trạng thái của một cluster tương ứng trên vùng dữ liệu. Hệ điều hành DOS có thể định dạng hệ thống File theo một trong 2 loại FAT là FAT12 và FAT16. Mỗi phần tử trong FAT12 rộng 12bit (1.5 byte), mỗi phần tử trong FAT16 rộng 16 bit(2 byte)

FAT 12	FAT 16	Ý nghĩa
FF7h	FFF7h	Cluster tương ứng bị hỏng
FF8h-FFFh	FFF8h-FFFFh	Cluster cuối cùng trong dãy các cluster chứa file
000h	0000h	Cluster tương ứng trống
FF0h- FF6h	FFF0h-FFF6h	Cluster tương ứng dành riêng cho hệ điều hành.
002h-FFEh	0002h-FFFEh	Đây là số hiệu của cluster trong bảng FAT, nó cho biết cluster tiếp theo trong dãy các cluster chứa nội dung một file.

Trong bảng FAT, hai phần tử đầu tiên (00 và 01) không dùng cho việc theo dõi trạng thái cluster và ghi nhận bản đồ cấp phát file, mà nó được sử dụng để chứa một giá trị nhận biết khuôn dạng đĩa, được gọi là byte định danh của đĩa, đây là byte đầu tiên trong bảng FAT. Đối với đĩa cứng thì byte ID=F8h.

Như vậy để đọc được nội dung của một file trên đĩa thì trước hết hệ điều hành phải tìm được dãy các cluster chứa nội dung của một file. Nhưng bảng Fat chỉ cho biết số hiệu các cluster từ cluster thứ hai đến cluster cuối cùng trong dãy nói trên. Cluster

đầu tiên trong dãy các cluster chứa nội dung của một file trên đĩa được tìm thấy trong bảng thư mục gốc.

Lưu giữ File theo FAT

Các phần tử trong bảng FAT tạo thành danh sách móc nối và phần tử cuối cùng của danh sách có giá trị FFF(FFFF). Vì các phần tử tạo thành danh sách móc nối nên chúng không nhất thiết phải nằm cạnh nhau.

Ví dụ : Tệp f1.txt có giá trị Starting cluster=6

0	FF0
1	FFF
2	
3	4
4	8
5	FFF
6	9
7	5
8	7
9	3

Tệp được lưu ở các cluster sau:

6→9→3→4→8→7→5

Thao tác đọc File như trên của DOS là kém hiệu quả, vì ngoài việc đọc nội dung của file tại các cluster trên vùng data của đĩa, hệ điều hành còn phải đọc và phân tích bảng FAT để dò tìm ra dãy các cluster chứa nội dung của một file. Hệ thống File NTFS trong WindowsNT/2000 khắc phục điều này bằng cách lưu danh sách các cluster chứa nội dung của một file vào một vị trí cố định nào đó, nên khi dọc file hệ điều hành chỉ cần đọc nội dung của các cluster trên đĩa theo danh sách ở trên, mà không phải tốn thời gian cho việc dò tìm dãy các cluster chứa nội dung của file của hệ thống file FAT trong DOS.

c) Root Directory (DIR)

Để quản lý thông tin của các file và các thư mục (thư mục con của thư mục gốc) đang được lưu trữ trên thư mục gốc của đĩa mềm hoặc đĩa logic trên đĩa cứng, hệ điều hành DOS sử dụng bảng thư mục gốc.

Bảng thư mục gốc gồm nhiều phần tử, số lượng phần tử trong bảng thư mục gốc được DOS quy định trước trong quá trình format đĩa và được ghi tại word tại offset 11h trong bootsector, giá trị này không thể thay đổi. Do đó tổng số file và thư mục con mà người sử dụng có thể chứ trên thư mục gốc của đĩa là có giới hạn, đây là

một hạn chế của DOS. Trong hệ thống file FAT32 và NTFS số phần tử trong bảng thư mục gốc không bị giới hạn, có thể thay đổi được và có thể được định vị tại một vị trí bất kỳ trên đĩa hoặc chứa trong một tập tin nào đó.

Mỗi phần tử trong bảng thư mục gốc dùng để chứa thông tin về một file hay thư mục nào đó đang được lưu trên thư mục gốc của đĩa. Khi có một file hoặc một thư mục nào được tạo ra trên thư mục gốc của đĩa thì hệ điều hành dùng một phần tử trong bảng thư mục gốc để chứa các thông tin liên quan của nó, khi một file hoặc một thư mục bị xóa/di chuyển khỏi thư mục gốc thì hệ điều hành sẽ thu hồi lại phần tử này để chuẩn bị cấp cho các file thư mục khác sau này.

Mỗi phần tử trong thư mục gốc dài 32 byte, chứa các thông tin sau:

Offset	Nội dung	Độ lớn
00h	Tên chính của file	8 byte
08h	Phần mở rộng của tên file	3 byte
0Bh	Thuộc tính file	1 byte
0Ch	Dự trữ, chưa được sử dụng	10 byte
16h	Giờ thay đổi tập tin cuối cùng	2 byte
18h	Ngày thay đổi tập tin cuối cùng	2 byte
1Ah	Cluster đầu tiên của file	2 byte
1Ch	Kích thước của file	4 byte

Thuộc tính của File

7	6	5	4	3	2	1	0
Dự trữ	Dự trữ	Archive	Directory	volum	system	hiden	readonly

d) Data area

Vùng data chính là nơi trên đĩa logic chứa nội dung các file có trên đĩa. Vùng này đưcợ chia thành các cluster (khối) và một file chứa nguyên vẹn một số cluster.

4.2.2 Hệ thống NTFS (New Technology File System)

Bảng Partition (mặt 0, rãnh 0, cung 1)

Boot sector (mặt 1, rãnh 0, cung 1)

Win2000 NTFS nhìn thấy FAT32

FAT32 không nhìn thấy NTFS

Bootsector

Byte 0x00-0x0A: lệnh nhẩy và các thông tin khác

Byte 0x0B-0x53: Các thông số BPB và BPB mở rộng

Byte phần còn lại Đoạn mã khởi động

MFT(Master File Table)

chiếm 12% dung lượng đĩa

Tương tự FAT (MFT1, MFT2)

MFT chứa thông tin: cso 16 điểm vào đầu tiên(16 bản ghi)

Bản ghi 1: MFT1

Bản ghi 2: MFT2

$logFile: chỉ ra sự thay đổi của danh mục

$bitmap:

$badcluster: danh sách khối hỏng

$secure: thông tin bảo vệ

$dùng cho file nhỏ

16 bản ghi

System file	File name	MFT record	Ý nghĩa
MFT	$MFT	0	Chứa một file cơ sở ghi nhớ mã file và thư mục trên NTFS
MFT	$MFT mirr	1	Bản sao
LogFile	$log File	2	Danh sách các thao tác khôi phục khi xoá
Volumn	$volumn	3	Chứa thông tin về ổ đĩa
Attribute	$Att Def	4	Bảng tên số lượng va mô tả thuộc tính
Rootfile name index	$RF	5	Thư mục gốc
Cluster bitmap	$bitmap	6	Chỉ các cluster đã sử dụng
Bootsector	$Boot	7	Chứa PDB
Bad cluster	$badclus	8	Chứa cluster hỏng
Secure File	$sercure	9	chứa các danh sách an toàn
Upcase Table	$Upcase	10	Chuyển đổi ký tự
NTFS extension	$Extend	11	Mở rộng
		12-15	Dự trữ

4.3 Các thông số và thuật toán truy nhập đĩa

4.3.1Các thông số

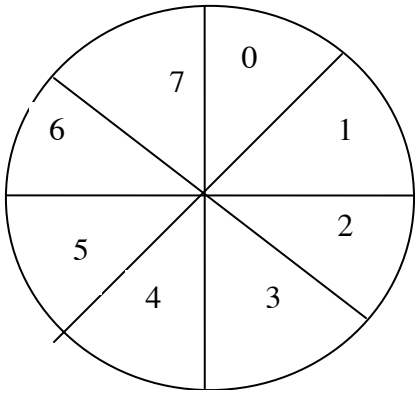
Thời gian truy xuất dữ liệu=thời gian tìm kiếm+thời gian dịch chuyển + thời gian quay trễ

Tốc độ đĩa bao gồm ba phần. Để truy xuất các khối trên đĩa, trước tiên phải di chuyển đầu đọc đến track hay cylinder thích hợp, thao tác này gọi là seek và thời gian để hoàn tất gọi là *seek time(thời gian tìm kiếm)*. Một khi đã đến đúng track, còn phải chờ cho đến khi khối cần thiết đến dưới đầu đọc. Thời gian chờ này gọi là *latency time(thời gian quay trễ)*. Cuối cùng là vận chuyển dữ liệu giữa đĩa và bộ nhớ chính gọi là *transfer time(thời gian dịch chuyển)*. Tổng thời gian cho dịch vụ đĩa chính là tổng của ba khoảng thời gian trên. Trong đó *seek time* và *latency time* là mất nhiều thời gian nhất, do đó để giảm thiểu thời gian truy xuất hệ điều hành đưa ra các thuật toán lập lịch truy xuất.

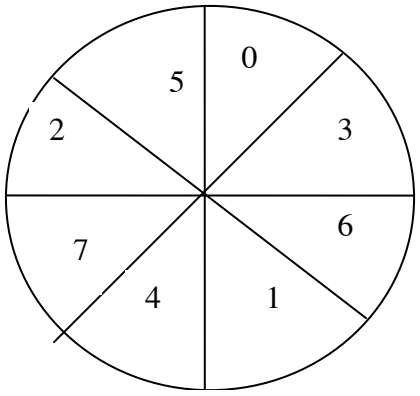
Hệ số đan xen

Hệ số đan xen của các sector nhằm làm khớp tốc độ quay nhanh của đĩa với tốc độ mà đầu từ có thể xử lý dữ liệu chậm khi chúng đi qua hết một sector. Nếu dữ liệu của một tệp được ghi trên nhiều cung liên tiếp của một rãnh đầu từ phải đợi vòng quay tới để đọc cung tiếp theo do vậy làm tăng thời gian truy nhập. Để tối ưu hoá quá trình

này, cung có thể được định địa chỉ xen kẽ khiến nhiều cung được đọc cùng một lúc trong một vòng quay của đĩa.



Không đan xen



Xen kẽ với hệ số đan xen là 3

Hình 4.10

4.3.2 Các thuật toán đọc đĩa

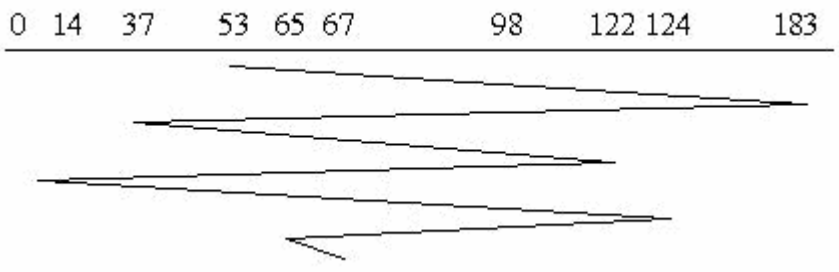
Tất cả mọi công việc đều phụ thuộc vào việc nạp chương trình và nhập xuất tập tin, do đó điều quan trọng là dịch vụ đĩa phải càng nhanh càng tốt. Hệ điều hành có thể tổ chức dịch vụ truy xuất đĩa tốt hơn bằng cách lập lịch yêu cầu truy xuất đĩa.

a) Lập lịch FCFS :

Phương pháp lập lịch đơn giản nhất là FCFS(first-come,first-served). Thuật toán này rất dể lập trình nhưng không cung cấp được một dịch vụ tốt. Ví dụ : cần phải đọc các khối theo thứ tự như sau :

98, 183, 37, 122, 14, 124, 65, và 67

Giả sử hiện tại đầu đọc đang ở vị trí 53. Như vậy đầu đọc lần lượt đi qua các khối 53, 98, 183, 37, 122, 14, 124, 65, và 67 như hình sau :



Hình 4.11 Phương pháp FCFS

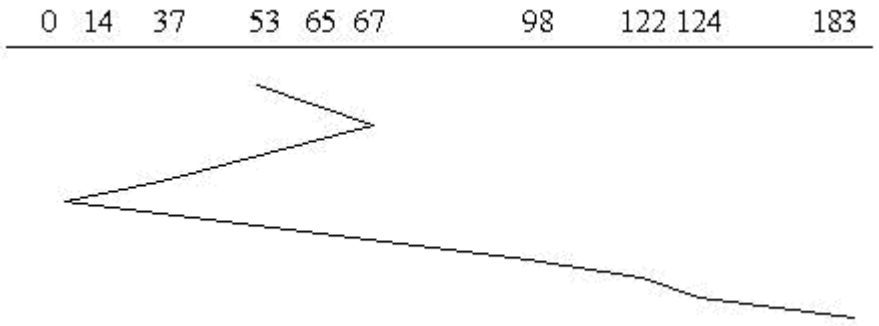
b)Lập lịch SSTF (shortest-seek-time-first)

Thuật toán này sẽ di chuyển đầu đọc đến các khối cần thiết theo vị trí lần lượt gần với vị trí hiện hành của đầu đọc nhất. Ví dụ : cần đọc các khối như sau :

98, 183, 37, 122, 14, 124, 65, và 67

Giả sử hiện tại đầu đọc đang ở vị trí 53. Như vậy đầu đọc lần lượt đi qua các khối

53, 65, 67, 37, 14, 98, 122, 124 và 183 như hình sau :



Hình 4.12 Phương pháp SSTF

Với ví dụ này, thuật toán SSTF làm giảm số khối mà đầu đọc phải di chuyển là 208 khối.

***c)*Lập lịch SCAN**

Theo thuật toán này, đầu đọc sẽ di chuyển về một phía của đĩa và từ đó di chuyển qua phía kia. Ví dụ : cần đọc các khối như sau :

98, 183, 37, 122, 14, 124, 65, và 67

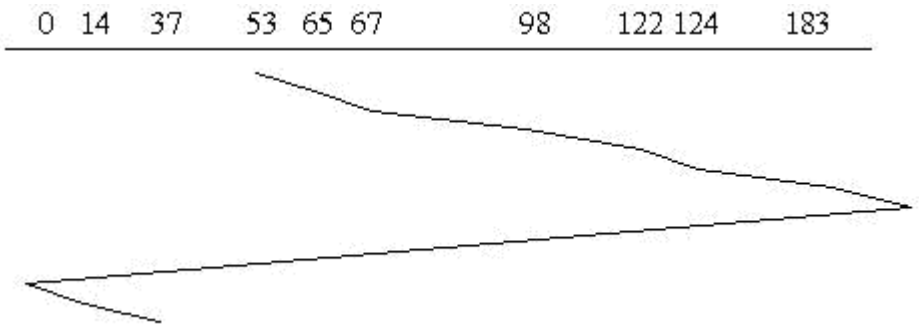
Giả sử hiện tại đầu đọc đang ở vị trí 53. Như vậy đầu đọc lần lượt đi qua các khối

53, 37, 14, 0 , 65, 67, 98, 122, 124 và 183 như hình sau :

Thuật toán này còn được gọi là thuật toán thang máy. Hình ảnh thuật toán giống như hình ảnh của một người quét tuyết, hay quét lá.

d)Lập lịch C-SCAN

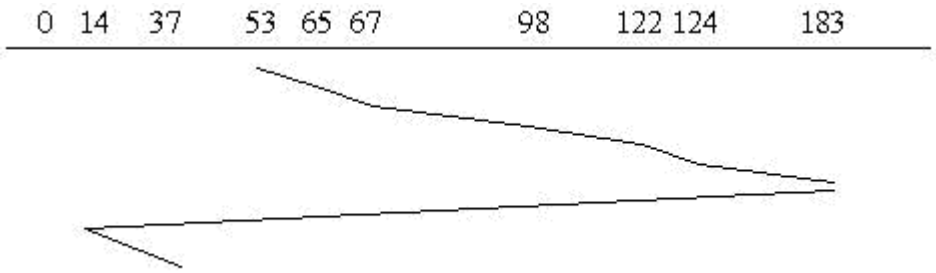
Thuật toán này tương tự như thuật toán SCAN, chỉ khác là khi nó di chuyển đến một đầu nào đó của đĩa, nó sẽ lập tức trở về đầu bắt đầu của đĩa. Lấy lại ví dụ trên, khi đó thứ tự truy xuất các khối sẽ là : 53, 65, 67, 98, 122, 124, 183, 199, 0, 14, 37 như hình sau :



Hình 4.13 Phương pháp C-SCAN

e)Lập lịch LOOK:

Nhận xét rằng cả hai thuật toán lập lịch SCAN và C-SCAN luôn luôn chuyển đầu đọc của đĩa từ đầu này sang đầu kia. Nhưng thông thường thì đầu đọc chỉ chuyển đến khối xa nhất ở mỗi hướng chứ không đến cuối. Do đó SCAN và C-SCAN được chỉnh theo thực tế và gọi là lập lịch LOOK. Như hình sau :



Hình 4.14 Phương pháp LOOK

Lựa chọn thuật toán lập lịch :

Với những thuật toán lập lịch, vấn đề là phải lựa chọn thuật toán nào cho hệ thống. Thuật toán SSTF thì rất thông thường. Thuật toán SCAN và C-SCAN thích hợp cho những hệ thống phải truy xuất dữ liệu khối lượng lớn. Với bất kỳ thuật toán lập lịch nào, điều quan trọng là khối lượng về số và kiểu khối cần truy xuất. Ví dụ , nếu số khối cần truy xuất là liên tục thì FCFS là thuật toán tốt.

Quản lý lỗi

Đĩa là đối tượng mà khi truy xuất có thể gây nhiều lỗi. Một trong số các lỗi thường gặp là

Lỗi lập trình : yêu cầu đọc các sector không tồn tại.

Lỗi lập trình xảy ra khi yêu cầu bộ điều khiển tìm kiếm cylinder không tồn tại, đọc sector không tồn tại, dùng đầu đọc không tồn tại, hoặc vận chuyển vào và ra bộ nhớ không tồn tại. Hầu hết các bộ điều khiển kiểm tra các tham số và sẽ báo lỗi nếu không thích hợp.

Lỗi checksum tạm thời : gây ra bởi bụi trên đầu đọc.

Bụi tồn tại giữa đầu đọc và bề mặt đĩa sẽ gây ra lỗi đọc. Nếu lỗi tồn tại, khối có thể bị đánh dấu hỏng bởi phần mềm.

Lỗi checksum thường trực : đĩa bị hư vật lý trên các khối.

Lỗi tìm kiếm : ví dụ đầu đọc đến cylinder 7 trong khi đó phải đọc 6.

Lỗi điều khiển : bộ điều khiển từ chối thi hành lệnh.

Chương 5 QUẢN LÝ VÀO RA

Một trong những chức năng chính của hệ điều hành là quản lý tất cả những thiết bị nhập/xuất của máy tính. Hệ điều hành phải ra các chỉ thị điều khiển thiết bị, kiểm soát các ngắt và lỗi. Hệ điều hành phải cung cấp một cách giao tiếp đơn giản và tiện dụng giữa các thiết bị và phần còn lại của hệ thống và giao tiếp này phải độc lập với thiết bị.

5.1 Khái niệm về hệ thống quản lý vào/ ra

Hệ thống quản lý nhập/xuất được tổ chức theo từng lớp, mỗi lớp có một chức năng nhất định và các lớp có giao tiếp với nhau như sơ đồ sau :

CÁC LỚP CHỨC NĂNG VÀO/RA

Xử lý của người dùng	Tạo lời gọi nhập/xuất, định dạng nhập/xuất
Phần mềm độc lập thiết bị	Đặt tên, bảo vệ, tổ chức khối, bộ đệm, định vị
Điều khiển thiết bị	Thiết lập thanh ghi thiết bị, kiểm tra trạng thái
Kiểm soát ngắt	Báo cho driver khi nhập/xuất hoàn tất
Phần cứng	Thực hiện thao tác nhập/xuất

Ví dụ: Trong một chương trình ứng dụng, người dùng muốn đọc một khối từ một tập tin, hệ điều hành được kích hoạt để thực hiện yêu cầu này. Phần mềm độc lập thiết bị tìm kiếm trong cache, nếu khối cần đọc không có sẵn, nó sẽ gọi chương trình điều khiển thiết bị gửi yêu cầu đến phần cứng. Tiến trình bị ngưng lại cho đến khi thao tác đĩa hoàn tất. Khi thao tác này hoàn tất, phần cứng phát sinh một ngắt. Bộ phận kiểm soát ngắt kiểm tra biến cố này, ghi nhận trạng thái của thiết bị và đánh thức tiến trình bị ngưng để chấm dứt yêu cầu I/O và cho tiến trình của người sử dụng tiếp tục thực hiện.

5.2 Phần cứng vào/ra

5.2.1 Các thiết bị vào/ra

Các thiết bị nhập xuất có thể chia tương đối thành hai loại là thiết bị khối (block device)và thiết bị tuần tự(character device).

- Thiết bị khối là thiết bị mà thông tin được lưu trữ trong những khối có kích thước cố định và được định vị bởi địa chỉ. Kích thước thông thường của một khối là khoảng từ 128 bytes đến 1024 bytes. Đặc điểm của thiết bị khối là chúng có thể được

truy xuất (đọc hoặc ghi) từng khối riêng biệt, và chương trình có thể truy xuất một khối bất kỳ nào đó. Đĩa là một ví dụ cho loại thiết bị khối.

- Một dạng thiết bị thứ hai là thiết bị tuần tự. Ở dạng thiết bị này, việc gửi và nhận thông tin dựa trên là chuỗi các bits, không có xác định địa chỉ và không thể thực hiện thao tác seek được. Màn hình, bàn phím, máy in, card mạng, chuột, và các loại thiết bị khác không phải dạng đĩa là thiết bị tuần tự.

Việc phân chia các lớp như trên không hoàn toàn tối ưu, một số các thiết bị không phù hợp với hai lớp trên, ví dụ : đồng hồ, bộ nhớ màn hình v.v...không thực hiện theo cơ chế tuần tự các bits. Ngoài ra, người ta còn phân loại các thiết bị I/O dưới một tiêu chuẩn khác :

- Thiết bị tương tác được với con người : dùng để giao tiếp giữa người và máy. Ví dụ : màn hình, bàn phím, chuột, máy in ...

- Thiết bị tương tác trong hệ thống máy tính là các thiết bị giao tiếp với nhau. Ví dụ : đĩa, băng từ, card giao tiếp...

- Thiết bị truyền thông : như modem...

Những điểm khác nhau giữa các thiết bị I/O gồm :

Tốc độ truyền dữ liệu , ví dụ bàn phím : 0.01 KB/s, chuột 0.02 KB/s ...

Công dụng.

Đơn vị truyền dữ liệu (khối hoặc ký tự).

Biểu diễn dữ liệu, điều này tùy thuộc vào từng thiết bị cụ thể.

Tình trạng lỗi : nguyên nhân gây ra lỗi, cách mà chúng báo về...

Một thiết bị giao tiếp với một hệ thống máy tính bằng cách gửi các tín hiệu qua dây cáp hay thậm chi qua không khí. Các thiết bị giao tiếp với máy tính bằng một điểm nối kết(cổng-port) như cổng tuần tự. Nếu một hay nhiều thiết bị dung một tập hợp dây dẫn, nối kết được gọi là bus. Một bus là một tập hợp dây dẫn và giao thức được định nghĩa chặt chẽ để xác định tập hợp thông điệp có thể được gửi qua dây. Trong thuật ngữ điện tử, các thông điệp được truyền bởi các mẫu điện thế điện tử được áp dụng tới các dây dẫn với thời gian được xác định. Khi thiết bị A có một cáp gán vào thiết bị B, thiết bị B có một cáp gán vào thiết bị C và thiết bị C gán vào một cổng máy tính, sự

sắp xếp này được gọi là chuỗi nối tiếp. Một chuỗi nối tiếp thường điều hành như một bus.

5.2.2 Tổ chức của chức năng I/O

Có ba cách để thực hiện I/O :

Một là, bộ xử lý phát sinh một lệnh I/O đến các đơn vị I/O, sau đó, nó chờ trong trạng thái "busy" cho đến khi thao tác này hoàn tất trước khi tiếp tục xử lý.

Hai là, bộ xử lý phát sinh một lệnh I/O đến các đơn vị I/O, sau đó, nó tiếp tục việc xử lý cho tới khi nhận được một ngắt từ đơn vị I/O báo là đã hoàn tất, nó tạm ngưng việc xử lý hiện tại để chuyển qua xử lý ngắt.

Ba là, sử dụng cơ chế DMA (như được đề cập ở sau)

Các bước tiến hóa của chức năng I/O :

Bộ xử lý kiểm soát trực tiếp các thiết bị ngoại vi.

Hệ thống có thêm bộ điều khiển thiết bị. Bộ xử lý sử dụng cách thực hiện nhập xuất thứ nhất. Theo cách này bộ xử lý được tách rời khỏi các mô tả chi tiết của các thiết bị ngoại vi.

Bộ xử lý sử dụng thêm cơ chế ngắt.

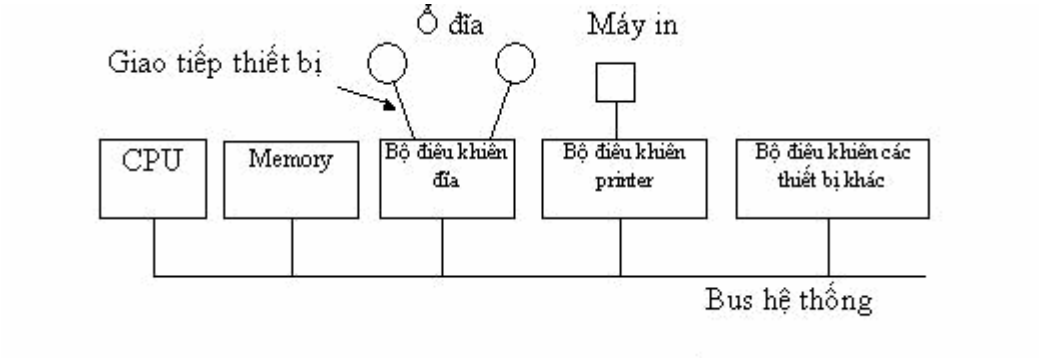
Sử dụng cơ chế DMA, bộ xử lý truy xuất những dữ liệu I/O trực tiếp trong bộ nhớ chính.

5.2.3 Bộ điều khiển thiết bị

Một đơn vị nhập xuất thường được chia làm hai thành phần chính là thành phần cơ và thành phần điện tử. Thành phần điện tử được gọi là bộ phận điều khiển thiết bị hay bộ tương thích, trong các máy vi tính thường được gọi là card giao tiếp. Thành phần cơ chính là bản thân thiết bị.

Một bộ phận điều khiển thường có bộ phận kết nối trên chúng để có thể gắn thiết bị lên đó. Một bộ phận điều khiển có thể quản lý được hai, bốn hay thậm chí tám thiết bị khác nhau. Nếu giao tiếp giữa thiết bị và bộ phận điều khiển là các chuẩn như ANSI, IEEE hay ISO thì nhà sản xuất thiết bị và bộ điều khiển phải tuân theo chuẩn đó, ví dụ : bộ điều khiển đĩa được theo chuẩn giao tiếp của IBM.

Giao tiếp giữa bộ điều khiển và thiết bị là giao tiếp ở mức thấp.



Hình 5.1 Sự kết nối giữa CPU, bộ nhớ, bộ điều khiển và các thiết bị nhập xuất

Chức năng của bộ điều khiển là giao tiếp với hệ điều hành vì hệ điều hành không thể truy xuất trực tiếp với thiết bị. Việc thông tin thông qua hệ thống đường truyền gọi là bus.

Công việc của bộ điều khiển là chuyển đổi dãy các bit tuần tự trong một khối các byte và thực hiện sửa chửa nếu cần thiết. Thông thường khối các byte được tổ chức thành từng bit và đặt trong buffer của bộ điều khiển. Sau khi thực hiện checksum nội dung của buffer sẽ được chuyển vào bộ nhớ chính. Ví dụ : bộ điều khiển cho màn hình đọc các byte của ký tự để hiển thị trong bộ nhớ và tổ chức các tín hiệu để điều khiển các tia của CRT để xuất trên màn ảnh bằng cách quét các tia dọc và ngang. Nếu không có bộ điều khiển, lập trình viên hệ điều hành phải tạo thêm chương trình điều khiển tín hiệu analog cho đèn hình. Với bộ điều khiển , hệ điều hành chỉ cần khởi động chúng với một số tham số như số ký tự trên một dòng, số dòng trên màn hình và bộ điều khiển sẽ thực hiện điều khiển các tia.

Mỗi bộ điều khiển có một số thanh ghi để liên lạc với CPU. Trên một số máy tính, các thanh ghi này là một phần của bộ nhớ chính tại một địa chỉ xác định gọi là ánh xạ bộ nhớ nhập xuất. Hệ máy PC dành ra một vùng địa chỉ đặc biệt gọi là địa chỉ nhập xuất và trong đó được chia làm nhiều đoạn, mỗi đoạn cho một loại thiết bị như sau :

Bộ điều khiển nhập/xuất	Địa chỉ nhập/xuất	Vectơ ngắt
Đồng hồ	040 - 043	8
Bàn phím	060 - 063	9
RS232 phụ	2F8 - 2FF	11
Đĩa cứng	320 - 32F	13
Máy in	378 - 37F	15

Màn hình mono	380 - 3BF	-
Màn hình màu	3D0 - 3DF	-
Đĩa mềm	3F0 - 3F7	14
RS232 chính	3F8 - 3FF	12

Hệ điều hành thực hiện nhập xuất bằng cách ghi lệnh lên các thanh ghi của bộ điều khiển. Ví dụ : bộ điều khiển đĩa mềm của IBMPC chấp nhận 15 lệnh khác nhau như : READ, WRITE, SEEK, FORMAT, RECALIBRATE, một số lệnh có tham số và các tham số cũng được nạp vào thanh ghi. Khi một lệnh đã được chấp nhận, CPU sẽ rời bộ điều khiển để thực hiện công việc khác. Sau khi thực hiện xong, bộ điều khiển phát sinh một ngắt để báo hiệu cho CPU biết và đến lấy kết quả được lưu giữ trong các thanh ghi.

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

Đa số các loại thiết bị, đặc biệt là các thiết bị dạng khối, hỗ trợ cơ chế DMA (direct memory access). Để hiểu về cơ chế này, trước hết phải xem xét quá trình đọc đĩa mà không có DMA. Trước tiên, bộ điều khiển đọc tuần tự các khối trên đĩa, từng bit từng bit cho tới khi toàn bộ khối được đưa vào buffer của bộ điều khiển. Sau đó máy tính thực hiện checksum để đảm bảo không có lỗi xảy ra. Tiếp theo bộ điều khiển tạo ra một ngắt để báo cho CPU biết. CPU đến lấy dữ liệu trong buffer chuyển về bộ nhớ chính bằng cách tạo một vòng lặp đọc lần lượt từng byte. Thao tác này làm lãng phí thời gian của CPU. Do đó để tối ưu, người ta đưa ra cơ chế DMA.

Cơ chế DMA giúp cho CPU không bị lãng phí thời gian. Khi sử dụng, CPU gửi cho bộ điều khiển một số các thông số như địa chỉ trên đĩa của khối, địa chỉ trong bộ nhớ nơi định vị khối, số lượng byte dữ liệu để chuyển.

Sau khi bộ điều khiển đã đọc toàn bộ dữ liệu từ thiết bị vào buffer của nó và kiểm tra checksum. Bộ điều khiển chuyển byte đầu tiên vào bộ nhớ chính tại địa chỉ được mô tả bởi địa chỉ bộ nhớ DMA. Sau đó nó tăng địa chỉ DMA và giảm số bytes phải chuyển. Quá trình này lập cho tới khi số bytes phải chuyển bằng 0, và bộ điều khiển tạo một ngắt. Như vậy không cần phải copy khối vào trong bộ nhớ, nó đã hiện hữu trong bộ nhớ.

5.3 Phần mềm vào/ra

Mục tiêu chung của thiết bị logic là dễ biểu diễn. Thiết bị logic được tổ chức thành nhiều lớp. Lớp dưới cùng giao tiếp với phần cứng, lớp trên cùng giao tiếp tốt, thân thiện với người sử dụng. Khái niệm then chốt của thiết bị logic là độc lập thiết bị,

ví dụ : có thể viết chương trình truy xuất file trên đĩa mềm hay đĩa cứng mà không cần phải mô tả lại chương trình cho từng loại thiết bị. Ngoài ra, thiết bị logic phải có khả năng kiểm soát lỗi. Thiết bị logic được tổ chức thành bốn lớp : Kiểm soát lỗi, điều khiển thiết bị, phần mềm hệ điều hành độc lập thiết bị, phần mềm mức người sử dụng.

5.3.1 Kiểm soát ngắt

Ngắt là một hiện tượng phức tạp. Nó phải cần được che dấu sâu trong hệ điều hành, và một phần ít của hệ thống biết về chúng. Cách tốt nhất để che dấu chúng là hệ điều hành có mọi tiến trình thực hiện thao tác nhập xuất cho tới khi hoàn tất mới tạo ra một ngắt. Tiến trình có thể tự khóa lại bằng cách thực hiện lệnh WAIT theo một biến điều kiện hoặc RECEIVE theo một thông điệp.

Khi một ngắt xảy ra, hàm xử lý ngắt khởi tạo một tiến trình mới để xử lý ngắt. Nó sẽ thực hiện một tín hiệu trên biến điều kiện và gửi những thông điệp đến cho các tiến trình bị khóa. Tổng quát, chức năng của ngắt là làm cho một tiến trình đang bị khóa được thi hành trở lại.

5.3.2 Điều khiển thiết bị (device drivers)

Tất cả các đoạn mã độc lập thiết bị đều được chuyển đến device drivers. Mỗi device drivers kiểm soát mỗi loại thiết bị, nhưng cũng có khi là một tập hợp các thiết bị liên quan mật thiết với nhau.

Device drivers phát ra các chỉ thị và kiểm tra xem chỉ thị đó có được thực hiện chính xác không. Ví dụ, driver của đĩa là phần duy nhất của hệ điều hành kiểm soát bộ điều khiển đĩa. Nó quản lý sectors, tracks, cylinders, head, chuyển động, interleave, và các thành phần khác giúp cho các thao tác đĩa được thực hiện tốt.

Chức năng của device drivers là nhận những yêu cầu trừu tượng từ phần mềm nhập/xuất độc lập thiết bị ở lớp trên, và giám sát yêu cầu này thực hiện. Nếu driver đang rảnh, nó sẽ thực hiện ngay yêu cầu, ngược lại, yêu cầu đó sẽ được đưa vào hàng đợi.

Ví dụ, bước đầu tiên của yêu cầu nhập/xuất đĩa là chuyển từ trừu tượng thành cụ thể. Driver của đĩa phải biết khối nào cần đọc, kiểm tra sự hoạt động của motor đĩa, xác định vị trí của đầu đọc đã đúng chưa v.v…

Nghĩa là device drivers phải xác định được những thao tác nào của bộ điều khiển phải thi hành và theo trình tự nào. Một khi đã xác định được chỉ thị cho bộ điều

khiển, nó bắt đầu thực hiện bằng cách chuyển lệnh vào thanh ghi của bộ điều khiển thiết bị. Bộ điều khiển có thể nhận một hay nhiều chỉ thị liên tiếp và sau đó tự nó thực hiện không cần sự trợ giúp của hệ điều hành. Trong khi lệnh thực hiện. Có hai trường hợp xảy ra : Một là device drivers phải chờ cho tới khi bộ điều khiển thực hiện xong bằng cách tự khóa lại cho tới khi một ngắt phát sinh mở khóa cho nó. Hai là, hệ điều hành chấm dứt mà không chờ, vì vậy driver không cần thiết phải khóa.

Sau khi hệ điều hành hoàn tất việc kiểm tra lỗi và nếu mọi thứ đều ổn driver sẽ chuyển dữ liệu cho phần mềm độc lập thiết bị. Cuối cùng nó sẽ trả về thông tin về trạng thái hay lỗi cho nơi gọi và nếu có một yêu cầu khác ở hàng đợi, nó sẽ thực hiện tiếp, nếu không nó sẽ khóa lại chờ đến yêu cầu tiếp theo.

5.3.3 Phần mềm nhập/xuất độc lập thiết bị

Mặc dù một số phần mềm nhập/xuất mô tả thiết bị nhưng phần lớn chúng là độc lập với thiết bị. Ranh giới chính xác giữa drivers và phần mềm độc lập thiết bị là độc lập về mặt hệ thống, bởi vì một số hàm mà được thi hành theo kiểu độc lập thiết bị có thể được thi hành trên drivers vì lý do hiệu quả hay những lý dó khác nào đó.

Giao tiếp đồng nhất cho device drivers
Đặt tên thiết bị
Bảo vệ thiết bị
Cung cấp khối độc lập thiết bị
Tổ chức buffer
Định vị lưu trữ trên thiết bị khối
Cấp phát và giải phóng thiết bị tận hiến
Báo lỗi

Chức năng cơ bản của phần mềm nhập/xuất độc lập thiết bị là những chức năng chung cho tất cả các thiết bị và cung cấp một giao tiếp đồng nhất cho phần mềm phạm vi người sử dụng.

Trước tiên nó phải có chức năng tạo một ánh xạ giữa thiết bị và một tên hình thức. Ví dụ đối với UNIX, tên /dev/tty0 dành riêng để mô tả I-node cho một file đặc biệt, và I-node này chứa chứa số thiết bị chính, được dùng để xác định driver thích hợp và số thiết bị phụ, được dùng để xác định các tham số cho driver để cho biết là đọc hay ghi.

Thứ hai là bảo vệ thiết bị, là cho phép hay không cho phép người sử dụng truy xuất thiết bị. Các hệ điều hành có thể có hay không có chức năng này.

Thứ ba là cung cấp khối dữ liệu độc lập thiết bị vì ví dụ những đĩa khác nhau sẽ có kích thước sector khác nhau và điều này sẽ gây khó khăn cho các phần mềm người sử dụng ở lớp trên. Chức năng này cung cấp các khối dữ liệu logic độc lập với kích thước sector vật lý.

Thứ tư là cung cấp buffer để hỗ trợ cho đồng bộ hóa quá trình hoạt động của hệ thống. Ví dụ buffer cho bàn phím.

Thứ năm là định vị lưu trữ trên các thiết bị khối.

Thứ sáu là cấp phát và giải phóng các thiết bị tận hiến.

Cuối cùng là thông báo lỗi cho lớp bên trên từ các lỗi do device driver báo về.

5.3.4 Phần mềm vào/ra phạm vi người sử dụng

Hầu hết các phần mềm nhập/xuất đều ở bên trong của hệ điều hành và một phần nhỏ của chúng chứa các thư viện liên kết với chương trình của người sử dụng ngay cả những chương trình thi hành bên ngoài hạt nhân.

Lời gọi hệ thống, bao gồm lời gọi hệ thống nhập/xuất thường được thực hiện bởi các hàm thư viện. Ví dụ khi trong chương trình C có lệnh

count = write(fd, buffer, nbytes) ;

Hàm thư viện write được địch và liên kết dưới dạng nhị phân và nằm trong bộ nhớ khi thi hành. Tập hợp tất cả những hàm thư viện này rõ ràng là một phần của hệ thống nhập/xuất.

Không phải tất cả các phần mềm nhập/xuất đều chứa hàm thư viện, có một loại quan trọng khác gọi là hệ thống spooling dùng để khai thác tối đa thiết bị nhập/xuất trong hệ thống đa chương.

Các hàm thư viện chuyển các tham số thích hợp cho lời gọi hệ thống và hàm thư viện thực hiện việc định dạng cho nhập và xuất như lệnh printf trong C. Thư viện nhập/xuất chuẩn chứa một số hàm có chức năng nhập/xuất và tất cả chạy như chương trình người dùng.

Chức năng của spooling là tránh trường hợp một tiến trình đang truy xuất thiết bị, chiếm giữ thiết bị nhưng sau đó không làm gì cả trong một khoảng thời gian và như vậy các tiến trình khác bị ảnh hưởng vì không thể truy xuất thiết bị đó. Một ví dụ của

spooling device là line printer. Spooling còn được sử dụng trong hệ thống mạng như hệ thống e-mail chẳng hạn.

Chương 6: HỆ THỐNG QUẢN LÝ FILE

Trong hầu hết các ứng dụng, tập tin là thành phần chủ yếu. Cho dù mục tiêu của ứng dụng là gì nó cũng phải bao gồm phát sinh và sử dụng thông tin. Thông thường đầu vào của các ứng dụng là tập tin và đầu ra cũng là tập tin cho việc truy xuất của người sử dụng và các chương trình khác sau này.

6.1 File và các thuộc tính của file

Tập tin

Máy tính có thể lưu giữ thông tin trên các thiết bị lưu trữ khác nhau như đĩa từ, băng từ, đĩa quang…Để cho máy tính trở nên thuận tiện và dễ sử dụng, HĐH cung cấp một cách lưu giữ thông tin logic như nhau trên các thiết bị lưu giữ đó là file (tập tin)

Tập tin là đơn vị lưu trữ thông tin của bộ nhớ ngoài.

Các tiến trình có thể đọc hay tạo mới tập tin nếu cần thiết. Thông tin trên tập tin là vững bền không bị ảnh hưởng bởi các xử lý tạo hay kết thúc các tiến trình, chỉ mất đi khi user thật sự muốn xóa. Tập tin được quản lý bởi hệ điều hành.

Các thuộc tính file

- *Tên tập tin :*

Tập tin là một cơ chế trừu tượng và để quản lý mỗi đối tượng phải có một tên. Khi tiến trình tạo một tập tin, nó sẽ đặt một tên, khi tiến trình kết thúc tập tin vẫn tồn tại và có thể được truy xuất bởi các tiến trình khác với tên tập tin đó.

Cách đặt tên tập tin của mỗi hệ điều hành là khác nhau, đa số các hệ điều hành cho phép sử dụng 8 chữ cái để đặt tên tập tin như ctdl, caycb, tamhghau v.v…, thường thường thì các ký tự số và ký tự đặc biệt cũng được sử dụng như baitap2…,

Hệ thống tập tin có thể có hay không phân biệt chữ thường và chữ hoa. Ví dụ : UNIX phân biệt chữ thường và hoa còn MS-DOS thì không phân biệt.

Nhiều hệ thống tập tin hỗ trợ tên tập tin gồm 2 phần được phân cách bởi dấu '.' mà phần sau được gọi là phần mở rộng. Ví dụ : vidu.txt. Trong MS-DOS tên tập tin có từ 1 đến 8 ký tư, phần mở rộng có từ 1 đến 3 ký tự. Trong UNIX có thể có nhiều phân cách như prog.c.Z. Một số kiểu mở rộng thông thường là :

.bak, .bas, .bin, .c, .dat, .doc, .ftn, .hlp, .lib, .obj, .pas, .tex, .txt.

Trên thực tế phần mở rộng có hữu ích trong một số trường hợp, ví dụ như có những trình dịch C chỉ nhận biết các tập tin có phần mở rộng là .C

Loại File: được thể hiện ở phần mở rộng và cách thực hiện trên file

Loại File	Phần mở rộng	Chức năng
Excutable	Exe, com, bin	sẵn sàng để chạy ngôn ngữ máy
Object	Obj,o	dịch ngôn ngữ máy, không liên kết
Source code	C, pas, asm	Mã nguồn
Batch	Bat, sh	xử lý theo lô
Text	Txt, doc	đocữ liệu văn bản, tài liệu
Library	Lib,a	Thư viện
Print or view	Ps, pdf, gif	In ấn và hiển thị
Archive	Arc, zip, tar	Nhóm các file trong một file, lưu giữ.

Ngoài tên và dữ liệu, hệ điều hành cung cấp thêm một số thông tin cho tập tin gọi là thuộc tính.

Các thuộc tính thông dụng trong một số hệ thống tập tin :

Tên thuộc tính	Ý nghĩa
Bảo vệ	Ai có thể truy xuất được và bằng cách nào
Mật khẩu	Mật khẩu cần thiết để truy xuất tập tin
Người tạo	Id của người tạo tập tin
Người sở hữu	Người sở hữu hiện tại
Chỉ đọc	0 là đọc ghi, 1 là chỉ đọc
ẩn	0 là bình thường, 1 là không hiển thị khi liệt kê
Hệ thống	0 là bình thường, 1 là tập tin hệ thống
Lưu trữ	0 đã được backup, 1 cần backup
ASCII/binary	0 là tập tin văn bản, 1 là tập tin nhị phân
Truy xuất ngẫu nhiên	0 truy xuất tuần tự, 1 là truy xuất ngẫu nhiên
Temp	0 là bình thường, 1 là bị xóa khi tiến trình kết thúc
Khóa	0 là không khóa, khác 0 là khóa
Độ dài của record	Số byte trong một record
Vị trí khóa	Offset của khóa trong mỗi record
Giờ tạo	Ngày và giờ tạo tập tin
Thời gian truy cập	Ngày và giờ truy xuất tập tin gần nhất

cuối cùng	
Thời gian thay đổi cuối cùng	Ngày và giờ thay đổi tập tin gần nhất
Kích thước hiện thời	Số byte của tập tin
Kích thước tối đa.	Số byte tối đa của tập tin

Hình 8.3 Một số thuộc tính thông dụng của tập tin

6.2 Thư mục: khái niệm, hệ thống thư mục, tổ chức bên trong

Thư mục

Thư mục là nơi để lưu giữ tập các file

Để lưu trữ dãy các tập tin, hệ thống quản lý tập tin cung cấp thư mục, mà trong nhiều hệ thống có thể coi như là tập tin.

Hệ thống thư mục theo cấp bậc

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

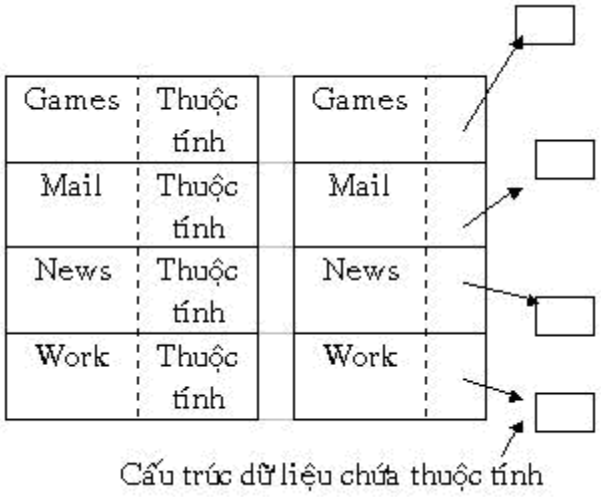
Khi một tập tin được mở, hệ điều hành tìm trên thư mục của nó cho tới khi tìm thấy tên của tập tin được mở. Sau đó nó sẽ xác định thuộc tính cũng như địa chỉ lưu trữ trên đĩa và đưa vào một bảng trong bộ nhớ. Những truy xuất sau đó thực hiện trong bộ nhớ chính.

Số lượng thư mục trên mỗi hệ thống là khác nhau. Thiết kế đơn giản nhất là hệ thống chỉ có thư mục đơn(còn gọi là thư mục một cấp), chứa tất cả các tập tin của tất cả người dùng, cách này dễ tổ chức và khai thác nhưng cũng dễ gây ra khó khăn khi có nhiều người sử dụng vì sẽ có nhiều tập tin trùng tên. Ngay cả trong trường hợp chỉ có một người sử dụng, nếu có nhiều tập tin thì việc đặt tên cho một tập tin mới không trùng lắp là một vấn đề khó.

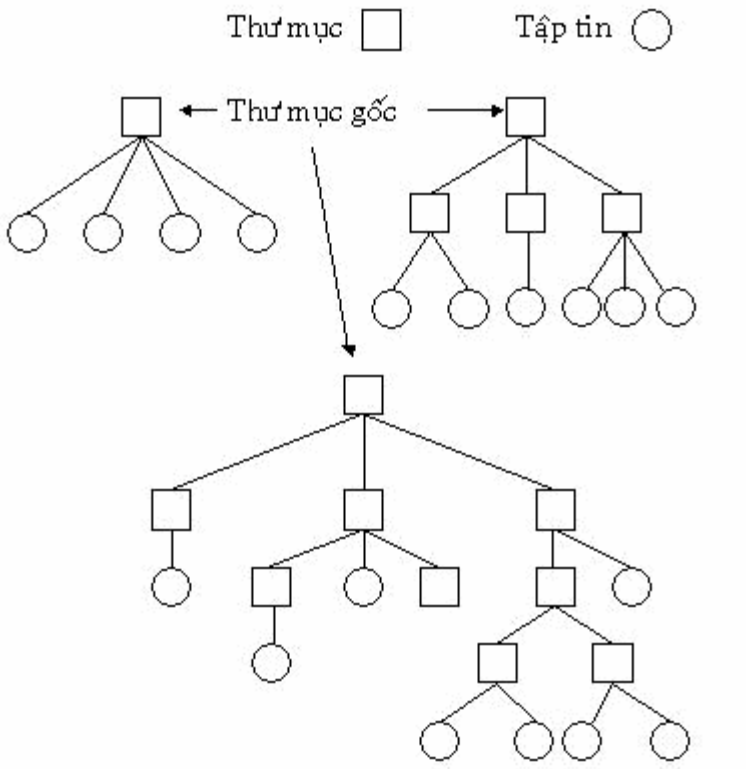
Cách thứ hai là có một thư mục gốc và trong đó có nhiều thư mục con, trong mỗi thư mục con chứa tập tin của người sử dụng (còn gọi là thư mục hai cấp), cách này tránh được trường hợp xung đột tên nhưng cũng còn khó khăn với người dùng có nhiều tập tin. Người sử dụng luôn muốn nhóm các ứng dụng lại một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

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

Ngoài ra, trong một số hệ điều hành nhiều người dùng, hệ thống còn xây dựng các hình thức khác của cấu trúc thư mục như cấu trúc thư mục theo đồ thị có chu trình và cấu trúc thư mục theo đồ thị tổng quát. Các cấu trúc này cho phép các người dùng trong hệ thống có thể liên kết với nhau thông qua các thư mục chia sẻ.



Hình 6.1



Hình 6.2 Hệ thống thư mục theo cấp bậc

Đường dẫn :

Khi một hệ thống tập tin được tổ chức thành một ***cây thư mục***, có hai cách để xác định một tên tập tin. Cách thứ nhất là ***đường dẫn tuyệt đối***, mỗi tập tin được gán một đường dẫn từ thư mục gốc đến tập tin. Ví dụ : /usr/ast/mailbox.

Dạng thứ hai là ***đường dẫn tương đối***, dạng này có liên quan đến một khái niệm là ***thư mục hiện hành*** hay thư mục làm việc. Người sử dụng có thể quy định một thư mục là thư mục hiện hành. Khi đó đường dẫn không bắt đầu từ thư mục gốc mà liên quan đến thư mục hiện hành. Ví dụ, nếu thư mục hiện hành là /usr/ast thì tập tin với đường dẫn tuyệt đối /usr/ast/mailbox có thể được dùng đơn giản là mailbox.

Trong phần lớn hệ thống, mỗi tiến trình có một thư mục hiện hành riêng, khi một tiến trình thay đổi thư mục làm việc và kết thúc, không có sự thay đổi để lại trên hệ thống tập tin. Nhưng nếu một hàm thư viện thay đổi đường dẫn và sau đó không đổi lại thì sẽ có ảnh hưởng đến tiến trình.

Hầu hết các hệ điều hành đều hỗ trợ hệ thống thư mục theo cấp bậc với hai entry đặc biệt cho mỗi thư mục là "." và "..". "." chỉ thư mục hiện hành, ".." chỉ thư mục cha.

Các thao tác trên thư mục :

Tạo : một thư mục được tạo, nó rỗng, ngoại trừ "." và ".." được đặt tự động bởi hệ thống.

Xóa :xoá một thư mục, chỉ có thư mục rỗng mới bị xóa, tư mục chứa "." và ".." coi như là thư mục rỗng.

Mở thư mục :thư mục có thể được đọc. Ví dụ để liệt kê tất cả tập tin trong một thư mục, chương trình liệt kê mở thư mục và đọc ra tên của tất cả tập tin chứa trong đó. Trước khi thư mục được đọc, nó phải được mở ra trước.

Đóng thư mục :khi một thư mục đã được đọc xong, phải đóng thư mục để giải phóng vùng nhớ.

Đọc thư mục :Lệnh này trả về entry tiếp theo trong thư mục đã mở. Thông thường có thể đọc thư mục bằng lời gọi hệ thống READ, lệnh đọc thư mục luôn luôn trả về một entry dưới dạng chuẩn .

Đổi tên :cũng như tập tin, thư mục cũng có thể được đổi tên.

Liên kết :kỹ thuật này cho phép một tập tin có thể xuất hiện trong nhiều thư mục khác nhau. Khi có yêu cầu, một liên kết sẽ được tạo giữa tập tin và một đường dẫn được cung cấp.

Bỏ liên kết :Nếu tập tin chỉ còn liên kết với một thư mục, nó sẽ bị loại bỏ hoàn toàn khỏi hệ thống, nếu nhiều thì nó bị giảm chỉ số liên kết.

6.3 Các phương pháp lưu giữ file

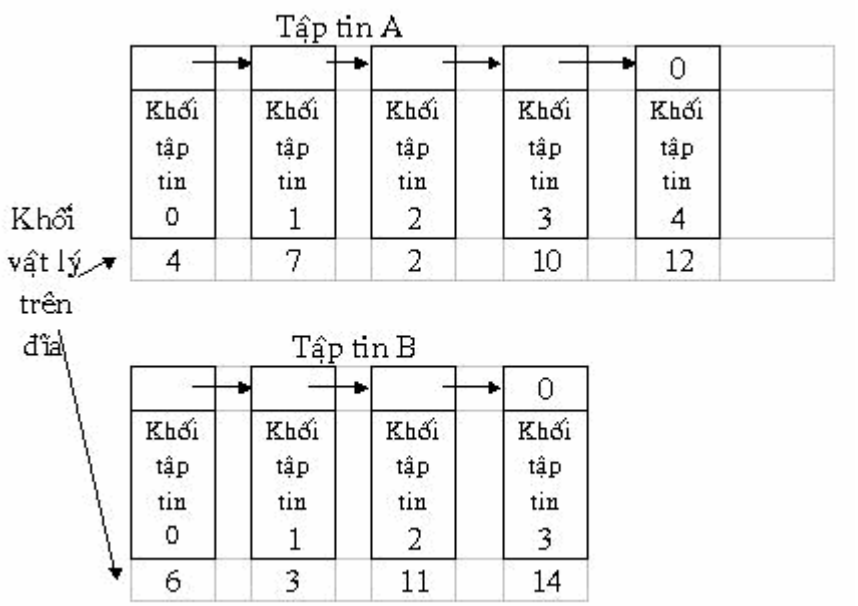
Định vị liên tiếp :

Lưu trữ tập tin trên dãy các khối liên tiếp.

Phương pháp này có 2 ưu điểm : thứ nhất, dễ dàng cài đặt. Thứ hai, dễ dàng thao tác vì toàn bộ tập tin được đọc từ đĩa bằng thao tác đơn giản không cần định vị lại.

Phương pháp này cũng có 2 khuyết điểm : không linh động trừ khi biết trước kích thước tối đa của tập tin. Sự phân mảnh trên đĩa, gây lãng phí lớn.

Định vị bằng danh sách liên kết :



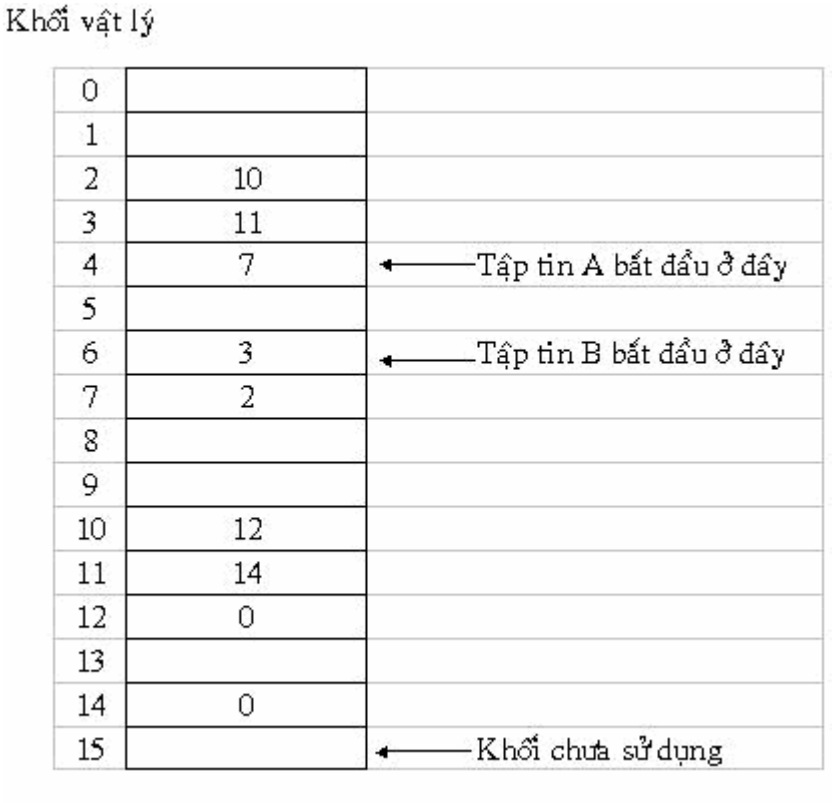
Hình 6.3 Định vị bằng danh sách liên kết

Mọi khối đều được cấp phát, không bị lãng phí trong trường hợp phân mảnh và directory entry chỉ cần chứa địa chỉ của khối đầu tiên.

Tuy nhiên khối dữ liệu bị thu hẹp lại và truy xuất ngẫu nhiên sẽ chậm.

Danh sách liên kết sử dụng index :

Tương tự như hai nhưng thay vì dùng con trỏ thì dùng một bảng index. Khi đó toàn bộ khối chỉ chứa dữ liệu. Truy xuất ngẫu nhiên sẽ dễ dàng hơn. Kích thước tập tin được mở rộng hơn. Hạn chế là bản này bị giới hạn bởi kích thước bộ nhớ .



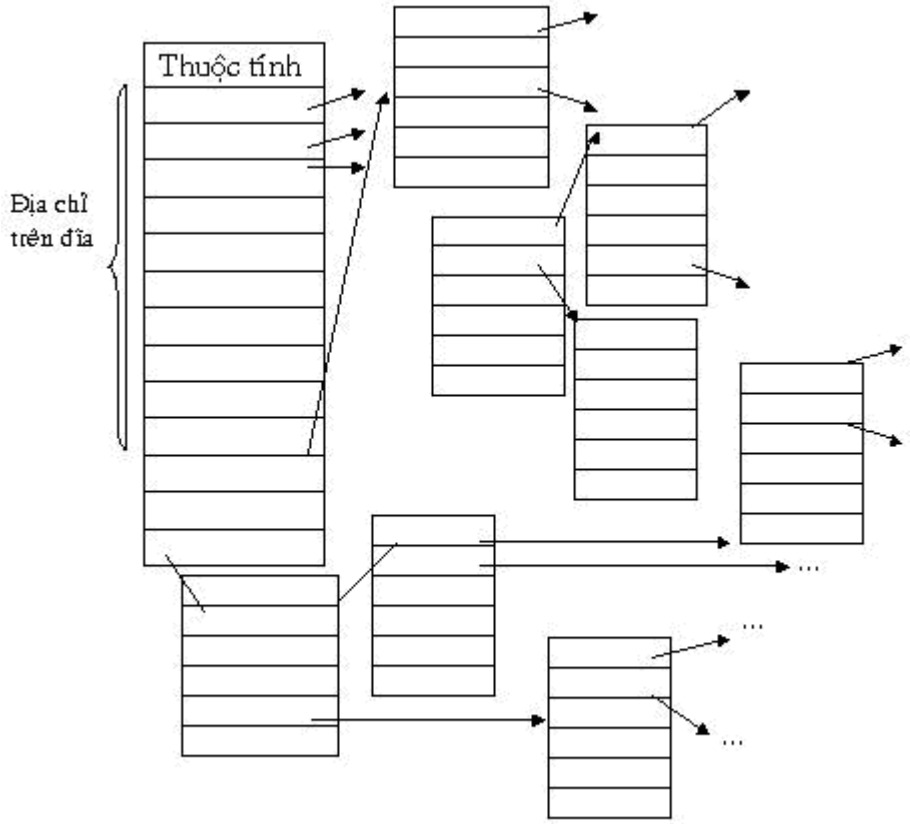
Hình 6.4 Bảng chỉ mục của danh sách liên kết

I-nodes :

Một I-node bao gồm hai phần. Phần thứ nhất là thuộc tính của tập tin. Phần này lưu trữ các thông tin liên quan đến tập tin như kiểu, người sở hữu, kích thước, v.v...Phần thứ hai chứa địa chỉ của khối dữ liệu. Phần này chia làm hai phần nhỏ. Phần nhỏ thứ nhất bao gồm 10 phần tử, mỗi phần tử chứa địa chỉ khối dữ liệu của tập tin. Phần tử thứ 11 chứa địa chỉ gián tiếp cấp 1 (single indirect), chứa địa chỉ của một khối, trong khối đó chứa một bảng có thể từ 2^{10} đến 2^{32} phần tử mà mỗi phần tử mới chứa địa chỉ của khối dữ liệu. Phần tử thứ 12 chứa địa chỉ gián tiếp cấp 2 (double indirect),

chứa địa chỉ của bảng các khối single indirect. Phần tử thứ 13 chứa địa chỉ gián tiếp cấp 3 (double indirect), chứa địa chỉ của bảng các khối double indirect.

Cách tổ chức này tương đối linh động. Phương pháp này hiệu quả trong trường hợp sử dụng để quán lý những hệ thống tập tin lớn. Hệ điều hành sử dụng phương pháp này là Unix (Ví dụ : BSD Unix)



Hình 6.5 Cấu trúc của I-node

6.4 Hệ thống quản lý tập tin (File management system).

Hệ thống quản lý tập tin là một tập các dịch vụ mà hệ điều hành cung cấp cho người sử dụng và chương trình người sử dụng để các đối tượng này sử dụng các tập tin trên hệ thống. Người sử dụng và chương trình của người sử dụng chỉ có thể truy xuất đến các tập tin thông qua hệ thống tập tin. Hệ thống quản lý tập tin của hệ điều hành phải đáp ứng các mục tiêu cơ bản sau đây.

- Đáp ứng các yêu cầu về lưu trữ dữ liệu của người sử dụng, bao gồm: khả năng lưu trữ, độ tin cậy và hiệu suất.

- Cực tiểu hay loại bỏ các nguy cơ có thể dẫn đến hỏng hoặc mất dữ liệu.

-Cung cấp sự hỗ trợ vào/ra cho nhiều người sử dụng trong các hệ thống đa người sử dụng.

-Cung cấp một tập chuẩn các thủ tục giao diện vào/ra.

Đối với người sử dụng thì hệ thống quản lý tập tin của một hệ điều hành phải đáp ứng các yêu cầu tối thiểu sau đây:

- Người sử dụng có thể dễ dàng tạo, sửa, thêm, bớt, xoá file.

- Người sử dụng có thể được điều khiển để truy cập đến các tập tin của người sử dụng khác.

- Người sử dụng phải có thể di chuyển dữ liệu giữa các tập tin.

- Người sử dụng phải có thể di chuyển dữ liệu giữa các tập tin.

- Người sử dụng phải có thể truy cập đến các tập tin của họ thông qua tên tượng trưng của tập tin.

- Người sử dụng có thể dự phòng và khôi phục lại các tập tin của họ trong trường hợp hệ thống bị hỏng.

- Người sử dụng có thể chia sẻ file cho người sử dụng khác

- Người sử dụng có thể truyền thông tin giữa các file lẫn nhau.

- Người sử dụng có thể lấy lại, lưu giữ file một cách dễ dàng.

- Người sử dụng có thể ngăn chặn các hành vi xâm phạm đến file và thực hiện được chế độ bảo vệ file.

- Có giao diện sử dụng file dễ dàng.

6.5 Các thao tác file

Tạo : một tập tin được tạo chưa có dữ liệu. Mục tiêu của chức năng này là thông báo cho biết rằng tập tin đã tồn tại và thiết lập một số thuộc tính.

Xóa :khi một tập tin không còn cần thiết nữa, nó được xóa để tăng dung lượng đĩa. Một số hệ điều hành tự động xoá tập tin sau một khoảng thời gian n ngày.

Mở : trước khi sử dụng một tập tin, tiến trình phải mở nó. Mục tiêu của mở là cho phép hệ thống thiết lập một số thuộc tính và địa chỉ đĩa trong bộ nhớ để tăng tốc độ truy xuất.

Đóng : khi chấm dứt truy xuất, thuộc tính và địa chỉ trên đĩa không cần dùng nữa, tập tin được đóng lại để giải phóng vùng nhớ. Một số hệ thống hạn chế tối đa số tập tin mở trong một tiến trình.

Đọc : đọc dữ liệu từ tập tin tại vị trí hiện thời của đầu đọc, nơi gọi sẽ cho biết cần bao nhiêu dữ liệu và vị trí của buffer lưu trữ nó.

Ghi : ghi dữ liệu lên tập tin từ vị trí hiện thời của đầu đọc. Nếu là cuối tập tin,kích thước tập tin sẽ tăng lên, nếu đang ở giữa tập tin, dữ liệu sẽ bị ghi chồng lên.

Thêm : gần giống như WRITE nhưng dữ liệu luôn được ghi vào cuối tập tin.

Tìm :dùng để truy xuất tập tin ngẫu nhiên. Khi xuất hiện lời gọi hệ thống, vị trí con trỏ đang ở vị trí hiện hành được di chuyển tới vị trí cần thiết. Sau đó dữ liệu sẽ được đọc ghi tại vị trí này.

Lấy thuộc tính :lấy thuộc tính của tập tin cho tiến trình

Thiết lập thuộc tính :thay đổi thuộc tính của tập tin sau một thời gian sử dụng.

Đổi tên :thay đổi tên của tập tin đã tồn tại.

6.6 Tổ chức file, truy nhập file

Cấu trúc của tập tin :

Gồm 3 loại :

Dãy tuần tự các byte không cấu trúc : hệ điều hành không biết nội dung của tập tin:MS-DOS và UNIX sử dụng loại này.

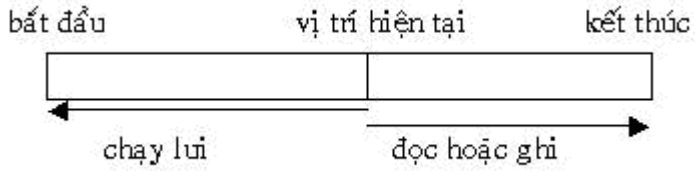
Dãy các record có chiều dài cố định.

Cấu trúc cây : gồm cây của những record, không cần thiết có cùng độ dài, mỗi record có một trường khóa giúp cho việc tìm kiếm nhanh hơn.

Tập tin lưu trữ các thông tin. Khi tập tin được sử dụng, các thông tin này được đưa vào bộ nhớ của máy tính. Có nhiều cách để truy xuất chúng. Một số hệ thống cung cấp chỉ một phương pháp truy xuất, một số hệ thống khác, như IBM chẳng hạn cho phép nhiều cách truy xuất.

Kiểu truy xuất tập tin đơn giản nhất là truy xuất tuần tự . Tiến trình đọc tất cả các byte trong tập tin theo thứ tự từ đầu. Các trình soạn thảo hay trình biên dịch cũng truy xuất tập tin theo cách này. Hai thao tác chủ yếu trên tập tin là đọc và ghi. Thao tác đọc sẽ đọc một mẫu tin tiếp theo trên tập tin và tự động tăng con trỏ tập tin. Thao tác ghi cũng tương tự như vậy. Tập tin có thể tự khởi động lại từ vị trí đầu tiên và trong một số hệ thống tập tin cho phép di chuyển con trỏ tập tin đi tới hoặc đi lui n mẫu tin.

Truy xuất kiểu này thuận lợi cho các loại băng từ và cũng là cách truy xuất khá thông dụng. Truy xuất tuần tự cần thiết cho nhiều ứng dụng. Có hai cách truy xuất. Cách truy xuất thứ nhất thao tác đọc bắt đầu ở vị trí đầu tập tin, cách thứ hai có một thao tác đặc biệt gọi là SEEK cung cấp vị trí hiện thời làm vị trí bắt đầu. Sau đó tập tin được đọc tuần tự từ vị trí bắt đầu.



Hình 6.6 Truy xuất tuần tự trên tập tin

Một kiểu truy xuất khác là truy xuất trực tiếp. Một tập tin có cấu trúc là các mẫu tin logic có kích thước bằng nhau, nó cho phép chương trình đọc hoặc ghi nhanh chóng mà không cần theo thứ tự. Kiểu truy xuất này dựa trên mô hình của đĩa. Đĩa cho phép truy xuất ngẫu nhiên bất kỳ khối dữ liệu nào của tập tin. Truy xuất trực tiếp được sử dụng trong trường hợp phải truy xuất một khối lượng thông tin lớn như trong cơ sở dữ liệu chẳng hạn. Ngoài ra còn có một số cách truy xuất khác dự trên kiểu truy xuất này như truy xuất theo chỉ mục ...

6.7 Độ an toàn của hệ thống file

Một hệ thống tập tin bị hỏng còn nguy hiểm hơn máy tính bị hỏng vì những hư hỏng trên thiết bị sẽ ít chi phí hơn là hệ thống tập tin vì nó ảnh hưởng đến các phần mềm trên đó. Hơn nữa hệ thống tập tin không thể chống lại được như hư hòng do phần cứng gây ra, vì vậy chúng phải cài đặt một số chức năng để bảo vệ.

Quản lý khối bị hỏng

Đĩa thường có những khối bị hỏng trong quá trình sử dụng đặc biệt đối với đĩa cứng vì khó kiểm tra được hết tất cả.

Có hai giải pháp : phần mềm và phần cứng.

Phần cứng là dùng một sector trên đĩa để lưu giữ danh sách các khối bị hỏng. Khi bộ kiểm soát tực hiện lần đầu tiên, nó đọc những khối bị hỏng và dùng một khối thừa để lưu giữ. Từ đó không cho truy cập những khối hỏng nữa.

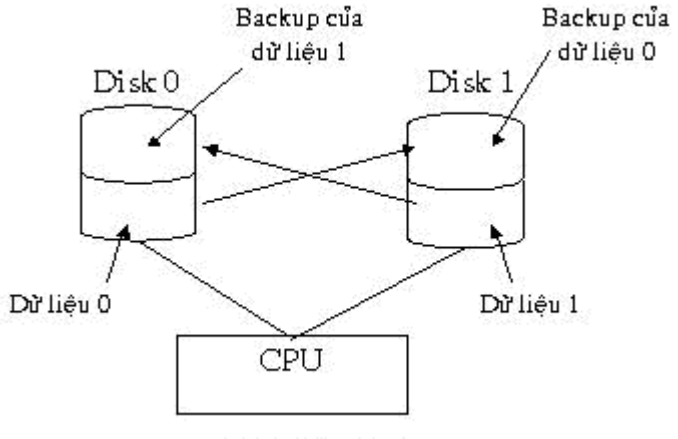
Phần mềm là hệ thống tập tin xây dựng một tập tin chứa các khối hỏng. Kỹ thuật này loại trừ chúng ra khỏi danh sách các khối trống, do đó nó sẽ không được cấp phát cho tập tin.

Backup

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

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

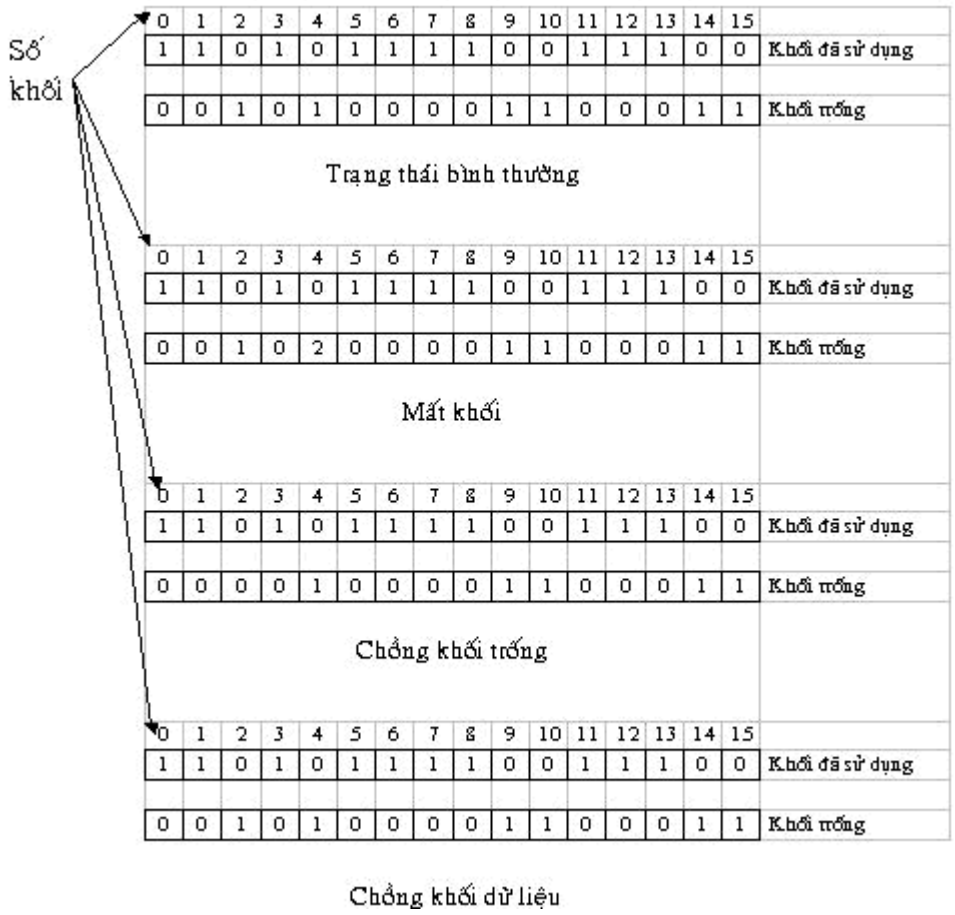
Đối với các đĩa cứng lớn, việc backup thường được tiến hành ngay trên nó. Một chiến lược dể cài đặt nhưng lãng phí một nữa đĩa là chia đĩa cứng làm hai phần một phần dữ liệu và một phần là backup. Mỗi tối, dữ liệu từ phần dữ liệu sẽ được chép sang phần backup.



Hình 6.7 Backup

Tính không đổi của hệ thống tập tin

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



Hình 6.8 Trạng thái của hệ thống tập tin

Tài liệu tham khảo:

[1] "*Tập slide bài giảng*". Bộ môn Các hệ thống thông tin.

[2] "*Operating System Concepts*". A. Silberschatz, P. B. Galvin, G. Gagne, Wiley & Sons, 2002.

[3] "*Operating Systems*", H. M. Deitel, P. J. Deitel and D. R. Choffnes, Pearson Education International, 2004.

[4] "*Modern Operating Systems*". Andrew S. Tanenbaum, Prentice-Hall International, 2001.

[5] "*Operating Systems – Internals and Design Principles*". William Stallings, Pearson Education International, 2005.