**TRƯỜNG TRUNG CẤP NGHỀ ĐÔNG SÀI GÒN**

# GIÁO TRÌNH
## Nguyên lý Hệ điều hành
***(Lưu hành nội bộ)***

*Thành phố Hồ Chí Minh năm 2019*

# CHƯƠNG 1: TỔNG QUAN VỀ HỆ ĐIỀU HÀNH

## 1.1 Khái niệm hệ điều hành

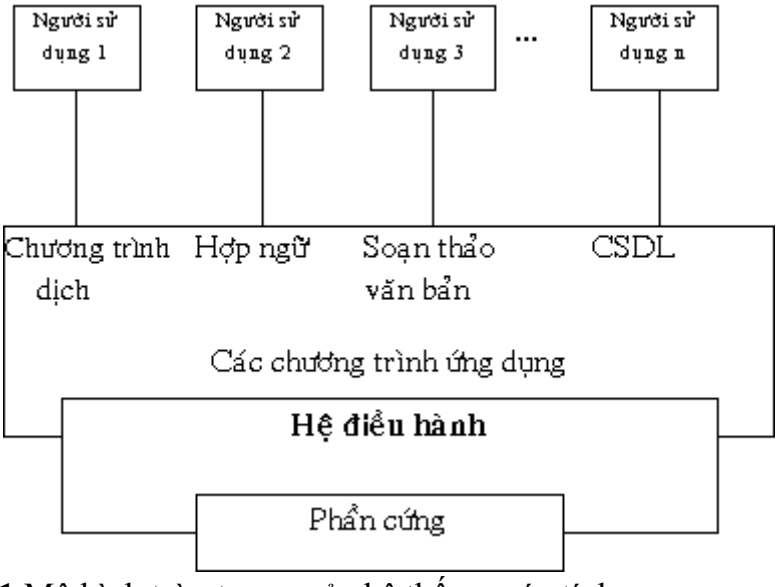
***Hệ điều hành*** là một *hệ thống các chương trình* hoạt động giữa người sử dụng (user) và phần cứng của máy tính. Mục tiêu của hệ điều hành là cung cấp một môi trường để người sử dụng có thể thi hành các chương trình. Nó làm cho máy tính dễ sử dụng hơn, thuận lợi hơn và hiệu quả hơn.

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

Hệ điều hành có thể được coi như là bộ phân phối tài nguyên của máy tính. Nhiều tài nguyên của máy tính như thời gian sử dụng CPU, vùng bộ nhớ, vùng lưu trữ tập tin, thiết bị nhập xuất v.v… được các chương trình yêu cầu để giải quyết vấn đề.

Hệ điều hành hoạt động như một bộ quản lý các tài nguyên và phân phối chúng cho các chương trình và người sử dụng khi cần thiết. Do có rất nhiều yêu cầu, hệ điều hành phải giải quyết vấn đề tranh chấp và phải quyết định ***cấp phát tài nguyên*** cho những yêu cầu theo thứ tự nào để hoạt động của máy tính là hiệu quả nhất. Một hệ điều hành cũng có thể được coi như là một chương trình kiểm soát việc sử dụng máy tính, đặc biệt là các thiết bị nhập xuất.

Tuy nhiên, nhìn chung chưa có định nghĩa nào là hoàn hảo về hệ điều hành. Hệ điều hành tồn tại để giải quyết các vấn đề sử dụng hệ thống máy tính. Mục tiêu cơ bản của nó là giúp cho việc thi hành các chương trình dễ dàng hơn. Mục tiêu thứ hai là hỗ trợ cho các thao tác trên hệ thống máy tính hiệu quả hơn. Mục tiêu này đặc biệt quan trọng trong những hệ thống nhiều người dùng và trong những hệ thống lớn(phần cứng + quy mô sử dụng). Tuy nhiên hai mục tiêu này cũng có phần tương phản vì vậy lý thuyết về hệ điều hành tập trung vào việc tối ưu hóa việc sử dụng tài nguyên của 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 phổ 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)**

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***.

## 1.3. Phân loại hệ thống

### 1.3.1 Hệ thống xử lý theo lô

***B***ộ giám sát thường trực được thiết kế để giám sát việc thực hiện dãy các công việc một cách tự động, chương trình này luôn luôn thường trú trong bộ nhớ chính.

***Hệ điều hành theo lô*** thực hiện các công việc lần lượt theo những chỉ thị định trước.

### 1.3.2 Hệ thống xử lý theo lô đa chương

*Đa chương* (multiprogram) gia tăng khai thác CPU bằng cách tổ chức các công việc sao cho CPU luôn luôn phải trong tình trạng làm việc .

Ý tưởng như sau : hệ điều hành lưu giữ một phần của các công việc ở nơi lưu trữ trong bộ nhớ . CPU sẽ lần lượt thực hiện các phần công việc này. Khi đang thực hiện, nếu có yêu cầu truy xuất thiết bị thì CPU không nghỉ mà thực hiện tiếp công việc thứ hai…

Với hệ đa chương hệ điều hành ra quyết định cho người sử dụng vì vậy, ***hệ điều hành đa chương*** rất tinh vi. Hệ phải xử lý các vấn đề lập lịch cho công việc, lập lịch cho bộ nhớ và cho cả CPU nữa.

### 1.3.3 Hệ thống chia xẻ thời gian

Hệ thống chia xẻ thời gian là một mở rộng logic của hệ đa chương. Hệ thống này còn được gọi là ***hệ thống đa nhiệm*** (multitasking). Nhiều công việc cùng được thực hiện thông qua cơ chế chuyển đổi của CPU như hệ đa chương nhưng thời gian mỗi lần chuyển đổi diễn ra rất nhanh.

Hệ thống chia xẻ được phát triển để cung cấp việc sử dụng bên trong của một máy tính có giá trị hơn. ***Hệ điều hành chia xẻ*** thời gian dùng lập lịch CPU và đa chương để cung cấp cho mỗi người sử dụng một phần nhỏ trong máy tính chia xẻ. Một chương trình khi thi hành được gọi là một tiến trình. Trong quá trình thi hành của một tiến trình, nó phải thực hiện các thao tác nhập xuất và trong khoảng thời gian đó CPU sẽ thi hành một tiến trình khác. Hệ điều hành chia xẻ cho phép nhiều người sử dụng

chia xẻ máy tính một cách đồng bộ do thời gian chuyển đổi nhanh nên họ có cảm giác là các tiến trình đang được thi hành cùng lúc.

Hệ điều hành chia xẻ phức tạp hơn hệ điều hành đa chương. Nó phải có các chức năng : quản trị và bảo vệ bộ nhớ, sử dụng bộ nhớ ảo. Nó cũng cung cấp hệ thống tập tin truy xuất on-line…

Hệ điều hành chia xẻ là kiểu của các hệ điều hành hiện đại ngày nay.

### 1.3.4 Hệ thống song song

Ngoài các hệ thống chỉ có một bộ xử lý còn có các hệ thống có nhiều bộ xử lý cùng chia xẻ hệ thống đường truyền dữ liệu, đồng hồ, bộ nhớ và các thiết bị ngoại vi. Các bộ xử lý này liên lạc bên trong với nhau .

Với sự gia tăng số lượng bộ xử lý, công việc được thực hiện nhanh chóng hơn. Hệ thống với máy nhiều bộ xử lý sẽ tối ưu hơn hệ thống có nhiều máy có một bộ xử lý vì các bộ xử lý chia xẻ các thiết bị ngoại vi, hệ thống lưu trữ, nguồn … và rất thuận tiện cho nhiều chương trình cùng làm việc trên cùng một tập hợp dữ liệu.

Một lý do nữa là độ tin cậy. Các chức năng được xử lý trên nhiều bộ xử lý và sự hỏng hóc của một bộ xử lý sẽ không ảnh hưởng đến toàn bộ hệ thống.

***Hệ thống đa xử lý*** thông thường sử dụng cách ***đa xử lý đối xứng***, trong cách này mỗi bộ xử lý chạy với một bản sao của hệ điều hành, những bản sao này liên lạc với nhau khi cần thiết. Một số hệ thống sử dụng đa xử lý bất đối xứng, trong đó mỗi bộ xử lý được giao một công việc riêng biệt.. Một bộ xử lý chính kiểm soát toàn bộ hệ thống, các bộ xử lý khác thực hiện theo lệnh của bộ xử lý chính hoặc theo những chỉ thị đã được định nghĩa trước. Mô hình này theo dạng quan hệ chủ tớ. Bộ xử lý chính sẽ lập lịch cho các bộ xử lý khác.

Một ví dụ về hệ thống xử lý đối xứng là version Encore của UNIX cho máy tính Multimax. Hệ thống này có hàng tá bộ xử lý. Ưu điểm của nó là nhiều tiến trình có thể thực hiện cùng lúc . Một hệ thống đa xử lý cho phép nhiều công việc và tài nguyên được chia xẻ tự động trong những bộ xử lý khác nhau.

Hệ thống đa xử lý không đồng bộ thường xuất hiện trong những hệ thống lớn, trong đó hầu hết thời gian hoạt động đều dành cho xử lý nhập xuất.

### 1.3.5 Hệ thống phân tán

Hệ thống này cũng tương tự như hệ thống chia xẻ thời gian nhưng các bộ xử lý không chia xẻ bộ nhớ và đồng hồ, thay vào đó mỗi bộ xử lý có bộ nhớ cục bộ riêng. Các bộ xử lý thông tin với nhau thông qua các đường truyền thông như những bus tốc độ cao hay đường dây điện thoại.

Các bộ xử lý trong hệ phân tán thường khác nhau về kích thước và chức năng. Nó có thể bao gồm máy vi tính, trạm làm việc, máy mini, và những hệ thống máy lớn.

Các nguyên nhân phải xây dựng hệ thống phân tán là:

***Chia xẻ tài nguyên*** : hệ thống phân tán cung cấp một cơ chế để chia xẻ tập tin ở vị trí xa, xử lý thông tin trong một cơ sở dữ liệu phân tán, in ấn tại một vị trí xa, sử dụng những thiết bị ở xa để hỗ thực hiện các thao tác.

***Tăng tốc độ tính toán*** : Một thao tác tính toán được chia làm nhiều phần nhỏ cùng thực hiện một lúc. Hệ thống phân tán cho phép phân chia việc tính toán trên nhiều vị trí khác nhau để tính toán song song.

***An toàn*** : Nếu một vị trí trong hệ thống phân tán bị hỏng, các vị trí khác vẫn tiếp tục làm việc.

***Thông tin liên lạc với nhau*** :Có nhiều lúc , chương trình cần chuyển đổi dữ liệu từ vị trí này sang vị trí khác. Ví dụ trong hệ thống Windows, thường có sự chia xẻ và chuyển dữ liệu giữa các cửa sổ. Khi các vị trí được nối kết với nhau trong một hệ thống mạng, việc trao đổi dữ liệu diễn ra rất dễ. Người sử dụng có thể chuyển tập tin hay các E_mail cho nhau từ cùng vị trí hay những vị trí khác.

### 1.3.6 Hệ thống xử lý thời gian thực

***Hệ thống xử lý thời gian thực*** được sử dụng khi có những đòi hỏi khắt khe về thời gian trên các thao tác của bộ xử lý hoặc dòng dữ liệu.

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. 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) 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.
- Cung cấp các cơ chế giao tiếp giữa các tiến trình.
- Cung cấp cơ chế kiểm soát deadlock

**b) Quản lý bộ nhớ chính :**

Trong hệ thống máy tính hiện đại, ***bộ nhớ chính*** là trung tâm của các thao tác, xử lý. Bộnhớ 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.

**c) 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.

- Lập lịch cho đĩa.

**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) Quản lý hệ thống 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.

- Ánh xạ tập tin trên hệ thống lưu trữ phụ.

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

**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) Hệ thống thông dịch lệnh :**

Một trong những phần quan trọng của chương trình hệ thống trong một hệ điều hành là hệ thống thông dịch lệnh, đó là giao tiếp giữa người sử dụng và hệ điều hành. Một số hệ điều hành đặt cơ chế dòng lệnh bên trong hạt nhân, số khác như MS-DOS và UNIX thì xem hệ điều hành như là một chương trình đặt biệt, được thi hành khi các công việc bắt đầu hoặc khi người sử dụng login lần đầu tiên.

Các lệnh đưa vào hệ điều hành thông qua *bộ điều khiển lệnh*. Trong các hệ thống chia xẻ thời gian một chương trình có thể đọc và thông dịch các lệnh điều khiển được thực hiện một cách tự động. Chương trình này thường được gọi là bộ thông dịch điều khiển card, cơ chế dòng lệnh hoặc Shell. Chức năng của nó rất đơn giản đó là lấy lệnh kế tiếp và thi hành.

Mỗi hệ điều hành sẽ có những giao tiếp khác nhau, dạng đơn giản theo cơ chế dòng lệnh, dạng thân thiện với người sử dụng như giao diện của Macintosh có các biểu tượng, cửa sổ thao tác dùng chuột.

Các lệnh có quan hệ với việc tạo và quản lý các tiến trình, kiểm soát nhập xuất, quản lý bộ lưu trữ phụ, quản lý bộ nhớ chính, truy xuất hệ thống tập tin và cơ chế bảo vệ.

## 1.5 Cấu trúc hệ thống

**a) Cấu trúc đơn giản**

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

HĐH là một tập hợp các thủ tục, có thể gọi lẫn nhau. Cấu trúc tối thiểu phân chia các thủ tục trong hệ thống thành 3 cấp độ:

Các thủ tục chính: gọi đến một thủ tục của HĐH, hay còn gọi là lời gọi hệ thống

Các thủ tục dịch vụ: xử lý những lời gọi hệ thống

Các thủ tục tiện ích hỗ trợ các thủ tục dịch vụ xử lý các lời gọi hệ thống

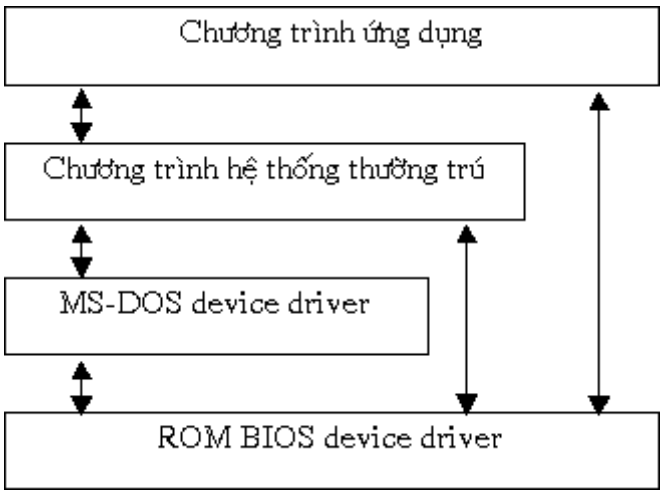
Nhược điểm:

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 mức độ phân chia thủ tục không rõ ràng

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) Cấu trúc phân lớp**

Bằng cách sử dụng kỹ thuật *topdown*, những chức năng và đặc tính của hệ thống được chia làm nhiều thành phần nhỏ. Che dấu thông tin, không cho chương trình của người sử dụng có thể cài đặt những hàm truy xuất cấp thấp , thay vào đó là những lớp giao tiếp bên trong.

Hệ điều hành được chia thành nhiều lớp. Lớp dưới cùng là phần cứng, lớp trên cùng là giao tiếp với người sử dụng. Lớp hệ điều hành được cài đặt thành những đối tượng trừu tượng. Thông thường một lớp của hệ điều hành bao gồm một số cấu trúc dữ

liệu và các hàm có thể được gọi bởi lớp ở trên và bản thân nó gọi những chức năng của lớp bên dưới.

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.

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.

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**

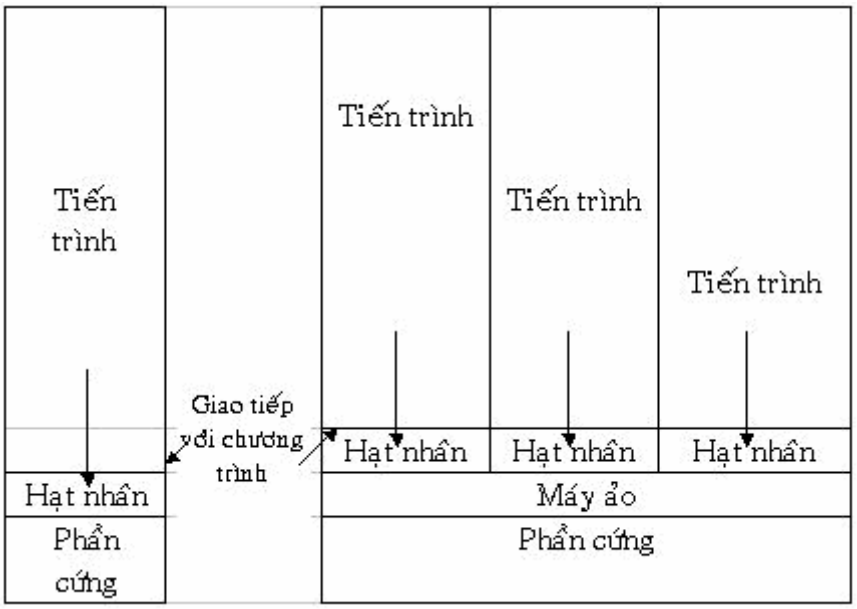
Các máy ảo là những bản sao ảo chính xác các đặc tính phần cứng của máy tính thực sự và cho phép một hệ điều hành khác hoạt động trên đó như trên phần cứng thực sự. Phần nhân hệ thống thực hiện giám sát máy ảo chịu trách nhiệm giao tiếp với phần cứng và cho phép khả năng đa chương bằng cách cung cấp nhiều máy ảo cho các lớp bên trên.

Bằng cách sử dụng lập lịch cho CPU và kỹ thuật bộ nhớ ảo, một hệ điều hành có thể tạo nhiều tiến trình phức ảo, mỗi cái sẽ thực hiện trên một bộ xử lý và bộ nhớ riêng. Những tiến trình này có những đặc điểm riêng như lời gọi hệ thống và hệ thống tập tin không được cung cấp phần cứng trực tiếp.

Tài nguyên của hệ thống được chia xẻ để tạo những máy ảo. Lập lịch CPU chia xẻ CPU cho các người sử dụng. Spooling và hệ thống tập tin được chia thành những card đọc ảo và máy in ảo. Một terminal cung cấp các chức năng tạo các thao tác màn hình ảo.

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.

Mặc dù khái niệm máy ảo rất hữu ích nhưng khó cài đặt. Máy ảo phải thực hiện ở hai dạng: dạng giám sát (monitor) và dạng người sử dụng. Ngoài ra máy ảo còn phải giải quyết các vấn đề về vận chuyển dữ liệu và thời gian.




Hình 1.4 So sánh giữa máy thực và máy ảo

**d) Vi nhân (Microkernels)**

Khi hệ điều hành UNIX được mở rộng, nhân trở nên lớn và khó quản lý. Vào giữa những năm 1980, các nhà nghiên cứu tại đại học Carnegie Mellon phát triển một hệ điều hành được gọi là Match mà module hóa nhân dùng ***tiếp cận vi nhân*** (micro kernel). Phương pháp này định kiến trúc của hệ điều hành bằng xóa tất cả thành phần không quan trọng từ nhân và cài chúng như các chương trình cấp người dùng và hệ thống. Kết quả này làm cho nhân nhỏ hơn. Có rất ít sự nhất trí liên quan đến việc quyết định dịch vụ nào nên để lại trong nhân và dịch vụ nào nên được cài đặt trong không gian người dùng. Tuy nhiên, thường thì các vi nhân điển hình cung cấp quá trình và quản lý bộ nhớ tối thiểu ngoài phương tiện giao tiếp.

Chức năng chính của vi nhân là cung cấp tiện nghi giao tiếp giữa chương trình khách hàng và các dịch vụ khác mà chúng đang chạy trong không gian người dùng. Giao tiếp được cung cấp bằng truyền thông điệp. Thí dụ, nếu chương trình khách hàng muốn truy xuất một tập tin, nó phải giao tiếp với trình phục vụ tập tin (file server).

Chương trình người dùng và dịch vụ không bao giờ giao tiếp trực tiếp. Đúng hơn là chúng giao tiếp gián tiếp bằng cách truyền thông điệp với vi nhân.

Thuận lợi của tiếp cận vi nhân là dễ dàng mở rộng hệ điều hành. Tất cả dịch vụ mới được thêm tới không gian người dùng và do đó không yêu cầu phải hiệu chỉnh nhân. Kết quả là hệ điều hành dễ dàng hơn để chuyển đổi từ thiết kế phần cứng này sang thiết kế phần cứng khác. Vi nhân cũng cung cấp khả năng an toàn và tin cậy hơn vì hầu hết các dịch vụ đang chạy như người dùng –hơn là nhân- các quá trình. Nếu một dịch vụ bị lỗi, phần còn lại của hệ điều hành vẫn không bị ảnh hưởng.

Một số hệ điều hành hiện đại dùng tiếp cận vi nhân. Tru64 UNIX (Digital UNIX trước đây) cung cấp giao diện UNIX tới người dùng, nhưng nó được cài đặt với nhân Mach. Nhân Mach ánh xạ các lời gọi hệ thống vào các thông điệp tới các dịch vụ cấp người dùng tương ứng. Hệ điều hành Apple MacOS Server được dựa trên cơ sở nhân Mach.

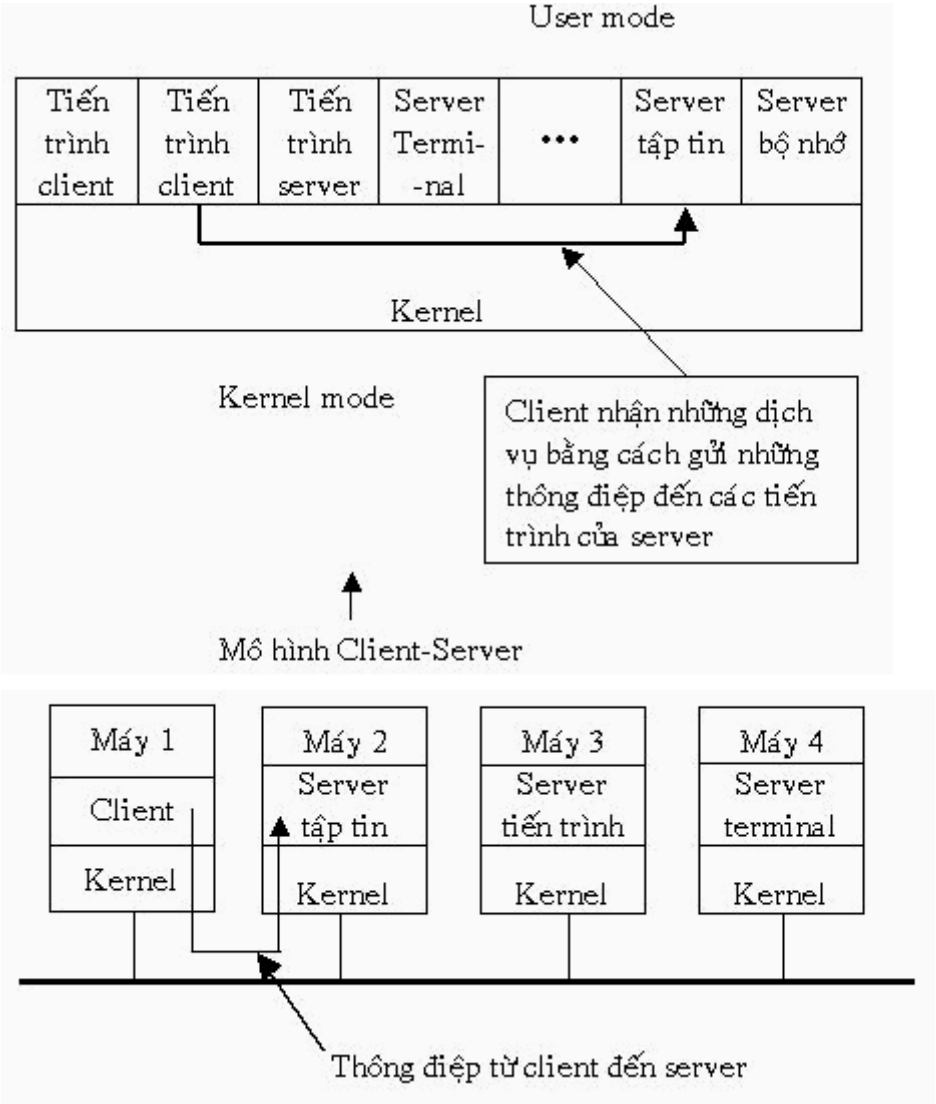
QNX là hệ điều hành thời thực cũng dựa trên cơ sở thiết kế vi nhân. Vi nhân QNX cung cấp các dịch vụ cho việc truyền thông điệp và định thời quá trình. Nó cũng quản lý giao tiếp mạng cấp thấp và các ngắt phần cứng. Tất cả dịch vụ khác trong QNX được cung cấp bởi các quá trình chuẩn chạy bên ngoài nhân trong chế độ người dùng.

Windows NT dùng một cấu trúc tổng hợp. Windows NT được thiết kế để chạy các ứng dụng khác nhau, gồm Win32 (ứng dụng thuần Windows), OS/2, và POSIX (Portable Operating System Interface for uniX). Nó cung cấp một server chạy trong không gian người dùng cho mỗi loại ứng dụng. Các chương trình khách hàng cho mỗi loại ứng dụng chạy trong không gian người dùng. Nhân điều phối việc truyền thông điệp giữa các ứng dụng khách hàng và server ứng dụng.

Khuynh hướng của các hệ điều hành hiện đại là chuyển dần các đoạn mã của hệ thống lên những lớp cao hơn và bỏ dần các chức năng trong hạt nhân, chỉ còn lại một hạt nhân tối thiểu. Cách tiếp cận là cài đặt hầu hết những chức năng của hệ điều hành trong các xử lý của người sử dụng. Để yêu cầu một dịch vụ, như đọc một khối từ tập tin, một xử lý của người sử dụng (còn được gọi là tiến trình client) sẽ gửi những yêu cầu đó cho một xử lý của bộ phận dịch vụ (còn được gọi là tiến trình server). Sau đó, nó sẽ thực hiện và gửi kết quả trở lại.

Trong mô hình này, chức năng của hạt nhân chỉ là kiểm soát quá trình thông tin giữa client và server. Bằng cách chia hệ điều hành thành những phần nhỏ, mỗi phần chỉ kiểm soát một mặt của hệ thống như các dịch vụ về tập tin, tiến trình, terminal, bộ nhớ, mỗi phần sẽ gọn hơn và dể quản lý hơn. Hơn nữa, tất cả server thực hiện như

những tiến trình ở mức độ người dùng (user-mode) không phải ở mức độ hạt nhân (kernel-mode), nên nó không truy xuất trực tiếp phần cứng. Do đó, nếu server tập tin bị lỗi, các dịch vụ về tập tin có thể bị hỏng nhưng nó thường không gây ảnh hưởng đến toàn bộ hệ thống.




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

Một ưu điểm khác của mô hình client-server là nó có thể tương thích dễ dàng với mô hình hệ thống phân tán. Nếu một client giao tiếp với một server bằng cách gửi những thông điệp, họ không biết là khi nào thông điệp đó đang được xử lý cục bộ tại máy hay được gửi vào mạng đến server trên một máy từ xa. Khi client quan tâm đến, một yêu cầu được gửi đi và một trả lời đáp ứng diễn ra như nhau.

## 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 pahỉ 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 đợc khai thác triệt để sao chon gay 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 cso thể cso 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 HĐH

**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 lien 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 đại 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 tị 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 khoa 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 ờang 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.

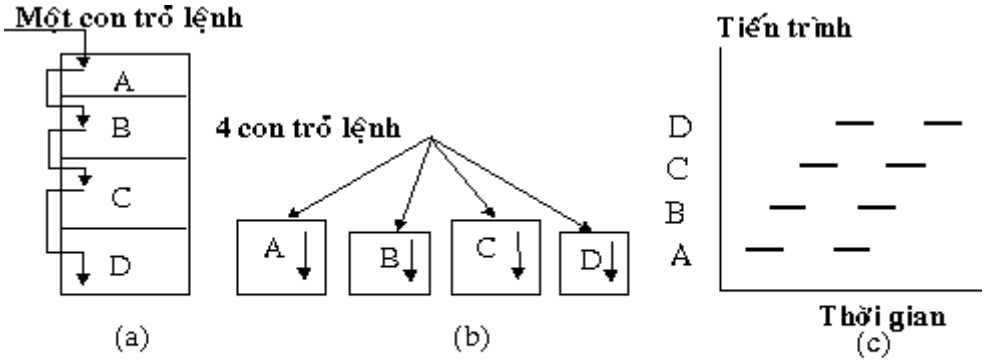
# CHƯƠNG 2 ĐIỀU KHIỂN DỮ LIỆU

## 2.1 Tiến trình

### 2.1.1 Khái niệm về 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

Để hỗ trợ sự đa chương, máy tính phải có khả năng thực hiện nhiều công việc đồng thời. Nhưng việc điều khiển nhiều hoạt động song song ở cấp độ phần cứng là rất khó khăn. Vì thế các nhà thiết kế hệ điều hành đề xuất một mô hình *song song giả lặp* bằng cách chuyển đổi bộ xử lý qua lại giữa các chương trình để duy trì hoạt động của nhiều chương trình cùng lúc, điều này tạo cảm giác có nhiều hoạt động được thực hiện đồng thời.

Về mặt ý niệm, có thể xem như mỗi tiến trình sỡ hữu một bộ xử lý ảo cho riêng nó, nhưng trong thực tế, chỉ có một bộ xử lý thật sự được chuyển đổi qua lại giữa các tiến trình. Sự chuyển đổi nhanh chóng này được gọi là *sự đa chương (multiprogramming)* . Hệ điều hành chịu trách nhiệm sử dụng một thuật toán điều phối để quyết định thời điểm cần dừng hoạt động của tiến trình đang xử lý để phục vụ một

tiến trình khác, và lựa chọn tiến trình tiếp theo sẽ được phục vụ. Bộ phận thực hiện chức năng này của hệ điều hành được gọi là *bộ điều phối (scheduler)*.

**Nhu cầu xử lý đồng hành**

Có 2 động lực chính khiến cho các hệ điều hành hiện đại thường hỗ trợ môi trường đa nhiệm (multitask) trong đó chấp nhận nhiều tác vụ thực hiện đồng thời trên cùng một máy tính :

- Tăng hiệu suất sử dụng CPU

Phần lớn các công việc khi thi hành đều trải qua nhiều chu kỳ xử lý (sử dụng CPU) và chu kỳ nhập xuất (sử dụng các thiết bị nhập xuất) xen kẽ như sau :

| | | | | |
|---|---|---|---|---|
| **CPU** | **IO** | **CPU** | **IO** | **CPU** |

Nếu chỉ có 1 tiến trình duy nhất trong hệ thống, thì vào các chu kỳ IO của công việc, CPU sẽ hoàn toàn nhàn rỗi. Ý tưởng tăng cường số lượng công việc trong hệ thống là để tận dụng CPU : nếu công việc 1 xử lý IO, thì có thể sử dụng CPU để thực hiện công việc 2...

| | | | | |
|---|---|---|---|---|
| **CPU** | **IO** | **CPU** | **IO** | **CPU** |

Công việc 1

| | | | | |
|---|---|---|---|---|
| | **CPU** | **IO** | **CPU** | **IO** |

Công việc 2

Khi đó CPU, bộ nhớ và các tài nguyên khác sẽ được tận dụng tối đa, nâng cao hiệu suất sử dụng tài nguyên.

- Tăng tốc độ xử lý

Một số bài toán có bản chất xử lý song song nếu được xây dựng thành nhiều module hoạt động đồng thời thì sẽ tiết kiệm được thời gian xử lý.

Ví dụ : Xét bài toán tính giá trị biểu thức kq = a*b + c*d . Nếu tiến hành tính đồng thời (a*b) và (c*d) thì thời gian xử lý sẽ ngắn hơn là thực hiện tuần tự.

Trong các trường hợp đó, cần có một mô hình xử lý đồng hành thích hợp. Trên máy tính có cấu hình nhiều CPU, hỗ trợ xử lý song song (multiprocessing) thật sự, điều này sẽ giúp tăng hiệu quả thi hành của hệt thống đáng kể.

### 2.1.2 Khái niệm tiểu trình (Thread) và mô hình đa tiểu trình (Multithread)

Trong hầu hết các hệ điều hành, mỗi tiến trình có một không gian địa chỉ và chỉ có một dòng xử lý. Tuy nhiên, có nhiều tình huống người sử dụng mong muốn có nhiều dòng xử lý cùng chia sẻ một không gian địa chỉ, và các dòng xử lý này hoạt động song song tương tự như các tiến trình phân biệt (ngoại trừ việc chia sẻ không gian địa chỉ).

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.

Chính vì các tình huống tương tự, người ta cần có một cơ chế xử lý mới cho phép có nhiều dòng xử lý trong cùng một tiến trình.

Ngày nay đã có nhiều hệ điều hành cung cấp một cơ chế như thế và gọi là *tiểu trình (threads)*.

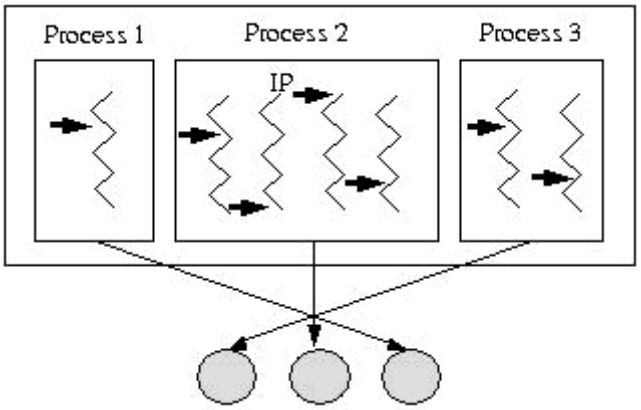
Nguyên lý chung :

*Một tiểu trình là một đơn vị xử lý cơ bản trong hệ thống . Mỗi tiểu trình xử lý tuần tự đoạn code của nó, sỡ hữu một con trỏ lệnh, tập các thanh ghi và một vùng nhớ stack riêng. Các tiểu trình chia sẻ CPU với nhau giống như cách chia sẻ giữa các tiến trình: một tiểu trình xử lý trong khi các tiểu trình khác chờ đến lượt. Một tiểu trình cũng có thể tạo lập các tiến trình con, và nhận các trạng thái khác nhau như một tiến trình thật sự. Một tiến trình có thể sỡ hữu nhiều tiểu trình.*

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, các tiểu trình trong cùng một tiến trình lại chia sẻ một không gian địa chỉ chung , điều này có nghĩa là các tiểu trình có thể chia sẻ các biến toàn cục của tiến trình. Một tiểu trình có thể truy xuất đến cả các stack của những tiểu trình khác trong cùng tiến trình. 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.

Các tiểu trình trong cùng một tiểu trình

| Tiến trình | Tiểu trình |
| --- | --- |
| Không gian địa chỉ<br>Tài nguyên toàn cục<br>Các thông tin thống kê | Con trỏ lệnh + các thanh ghi<br>Stack<br>Tài nguyên cục bộ |




Phân bổ thông tin lưu trữ

Cấu trúc mô tả tiến trình và tiểu trình

### 2.1.3 Phân loại tiến trình

- 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 có quan hệ thông tin

Trao đổi thông tin qua một vùng nhớ được biểu diễn như một hộp thư có thể trao đổi thông tin qua đó.

-Tiến trình độc lập

Hai hay nhiều tiến trình gọi là độc lập khi chúng không có quan hệ thông tin với nhau, hoạt động của tiến trình này không ảnh hưởng đến hoạt động của tiến trình khác và ngược lại.

-Tiến trình cha và tiến trình con

Một tiến trình được sinh ra từ một tiến trình khác thì được gọi là sự phân cấp của tiến trình hay được gọi là tiến trình cha và tiến trình con

-Tiến trình đồng mức

Thể hiện các tiến trình đó truy nhập tài nguyên dung chung theo nguyên tắc lần lượt.

### 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.

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

- Mới tạo : tiến trình đang được tạo lập.

- Running : các chỉ thị của tiến trình đang được xử lý.

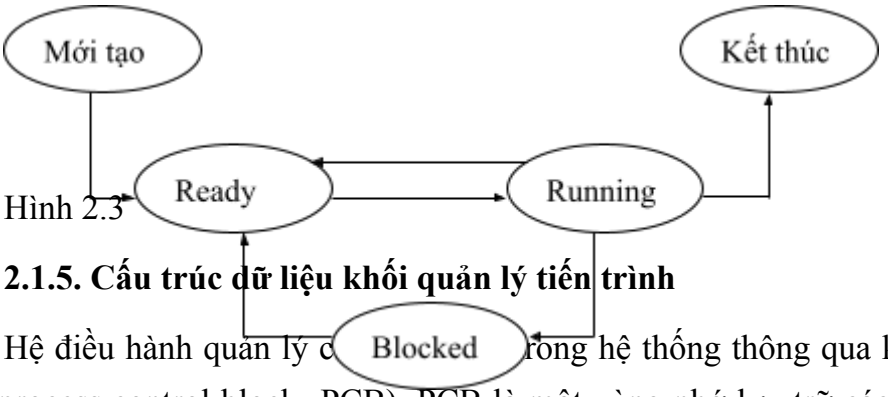
- Blocked : tiến trình chờ được cấp phát một tài nguyên, hay chờ một sự kiện xảy ra (hoàn thành nhập xuất hay nhận một tín hiệu) .

- Ready : tiến trình chờ được cấp phát CPU để xử lý.

- 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

### 2.1.5. Cấu trúc dữ liệu 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

- 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.

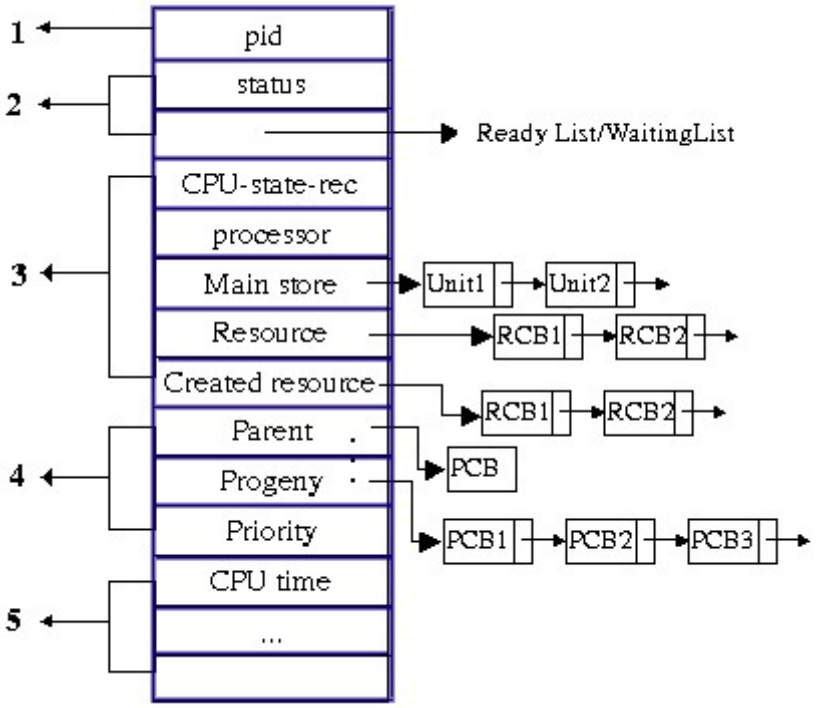
*Tài nguyên sử dụng*: danh sách các tài mguyên hệ thống mà tiến trình đang sử dụng.

*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 trên 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:

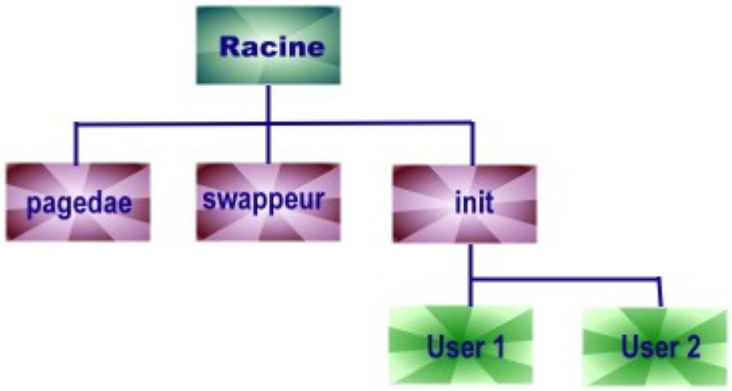
-Người sử dụng chạy một chương trình.

-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 sàngđể 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.

Để thực hiện được mục tiêu này, hệ điều hành phải lựa chọn tiến trình được xử lý tiếp theo. Bộ điều phối sẽ sử dụng một giải thuật điều phối thích hợp để thực hiện nhiệm vụ này. Một thành phần khác của hệ điều hành cũng tiềm ẩn trong công tác điều phối là *bộ phân phối* (dispatcher). Bộ phân phối sẽ chịu trách nhiệm chuyển đổi ngữ cảnh và trao CPU cho tiến trình được chọn bởi bộ điều phối để xử lý.

### 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 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 ( Turnaround Time) :

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

e) Thông lượng tối đa (Throughput ) :

Cực đại hóa số công việc được xử lý trong một đơn vị 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 không độc quyền và điều phối 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** : Ngược với nguyên lý độc quyền, điều phối theo nguyên lý *không độc quyền* cho phép tạm dừng hoạt động của một tiến trình đang sẵn sàng xử lý. Khi một tiến trình nhận được CPU, nó vẫn được sử dụng CPU đến khi hoàn tất hoặc tự nguyện giải phóng CPU, nhưng một tiến trình khác có độ ưu tiên có thể dành quyền sử dụng CPU của tiến trình ban đầu. Như vậy là tiến trình có thể bị tạm dừng hoạt động bất cứ lúc nào mà không được báo trước, để tiến trình khác xử lý. 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.

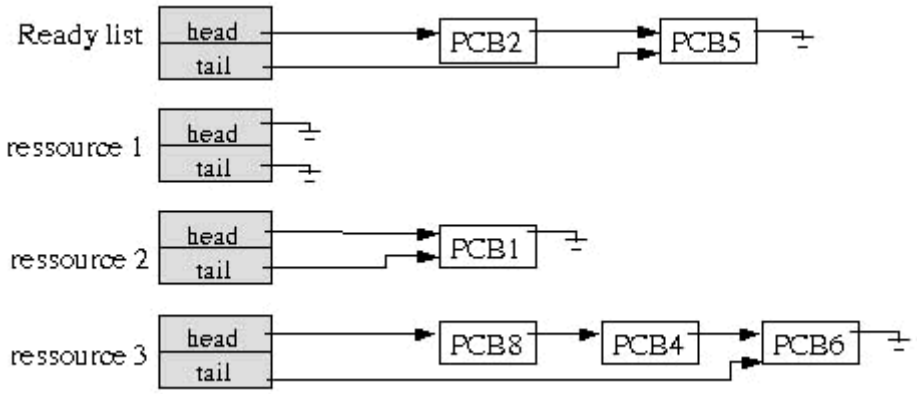
### 2.2.3. Các danh sách sử dụng trong quá trình điều phối.

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)*.

Khi một tiến trình bắt đầu đi vào hệ thống, nó được chèn vào danh sách các tác vụ (job list). Danh sách này bao gồm tất cả các tiến trình của hệ thống. Nhưng chỉ các tiến trình đang thường trú trong bộ nhớ chính và ở trạng thái sẵn sàng tiếp nhận CPU để hoạt động mới được đưa vào *danh sách sẵn sàng*.

Bộ điều phối sẽ chọn một tiến trình trong danh sách sẵn sàng và cấp CPU cho tiến trình đó. Tiến trình được cấp CPU sẽ thực hiện xử lý, và có thể chuyển sang trạng thái chờ khi xảy ra các sự kiện như đợi một thao tác nhập/xuất hoàn tất, yêu cầu tài

nguyên chưa được thỏa mãn, được yêu cầu tạm dừng ...Khi đó tiến trình sẽ được chuyển sang một danh sách chờ đợi.




***Hình 2.7*** *Các danh sách điều phối*

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.

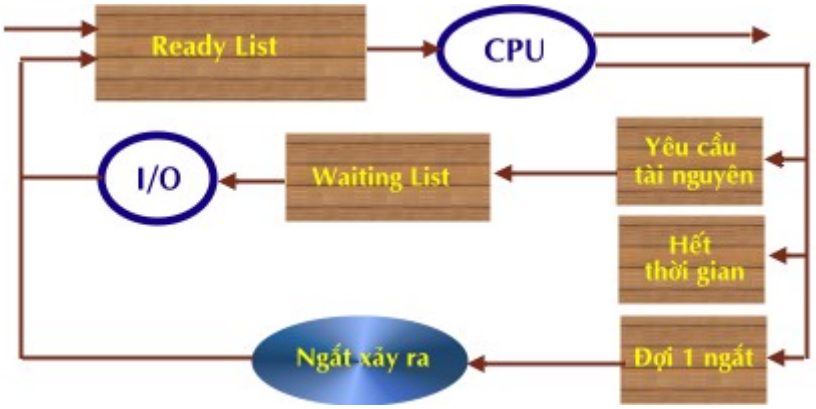
Quá trình xử lý của một tiến trình trải qua những chu kỳ chuyển đổi qua lại giữa danh sách sẵn sàng và danh sách chờ đợi. Sơ đồ dưới đây mô tả sự điều phối các tiến trình dựa trên các danh sách của hệ thống.

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

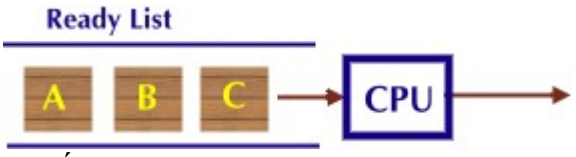


Trong trường hợp đầu tiên, tiến trình cuối cùng sẽ chuyển từ trạng thái blocked sang trạng thái ready và lại được đưa trở vào danh sách sẵn sàng. Tiến trình lặp lại chu kỳ này cho đến khi hoàn tất tác vụ thì được hệ thống hủy bỏ khỏi mọi danh sách điều phối.

### 2.2.4. Các chiến lược điều phối

#### *a). Chiến lược FIFO*

**Nguyên tắc** : CPU được cấp phát cho tiến trình đầu tiên trong danh sách sẵn sàng có yêu cầu, là tiến trình được đưa vào hệ thống sớm nhất. Đây là thuật toán điều phối theo nguyên tắc độc quyền. Một khi CPU được cấp phát cho tiến trình, CPU chỉ được tiến trình tự nguyện giải phóng khi kết thúc xử lý hay khi có một yêu cầu nhập/xuất.




Hình 2.9 Điều phối FIFO

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.

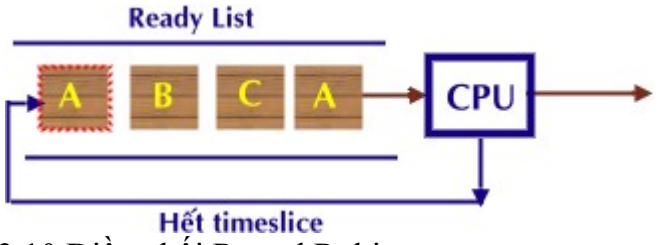
Thời gian chờ trung bình không đạt cực tiểu, và biến đổi đáng kể đối với các giá trị về thời gian yêu cầu xử lý và thứ tự khác nhau của các tiến trình trong danh sách sẵn sàng. Có thể xảy ra hiện tượng tích lũy thời gian chờ, khi các tất cả các tiến trình (có thể có yêu cầu thời gian ngắn) phải chờ đợi một tiến trình có yêu cầu thời gian dài kết thúc xử lý.

Giải thuật này đặc biệt không phù hợp với các hệ phân chia thời gian, trong các hệ này, cần cho phép mỗi tiến trình được cấp phát CPU đều đặn trong từng khoảng thời gian.

***b). Chiến lược phân phối xoay vòng (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 đến trước 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 đợi trung bình sẽ là (0+6+3+5)/3 = 4.66 milisecondes.

Nếu có *n* tiến trìh 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.

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***

**Nguyên tắc** : Mỗi tiến trình được gán cho một độ ưu tiên tương ứng, tiến trình có độ ưu tiên cao nhất sẽ được chọn để cấp phát CPU đầu tiên. Các tiến trình có độ ưu tiên bằng nhau thì tiến trình nào đến trước thì sẽ được cấp trước.Độ ưu tiên có thể được định nghĩa nội tại hay nhờ vào các yếu tố bên ngoài. Độ ưu tiên nội tại sử dụng các đại lượng có thể đo lường để tính toán độ ưu tiên của tiến trình, ví dụ các giới hạn thời gian, nhu cầu bộ nhớ…Độ ưu tiên cũng có thể được gán từ bên ngoài dựa vào các tiêu chuẩn do hệ điều hành như tầm quan trọng của tiến trình, loại người sử dụng sỡ hữu tiến trình…

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, và tiến trình hiện hành vẫn tiếp tục xử lý hết thời gian dành cho nó.

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 |

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

| P1 | P2 | P3 | P1 |
|---|---|---|---|
| 0 | ‘1 | 4 | 7 30 |

**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 trước (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 du :

| 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 |

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 |

**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 :

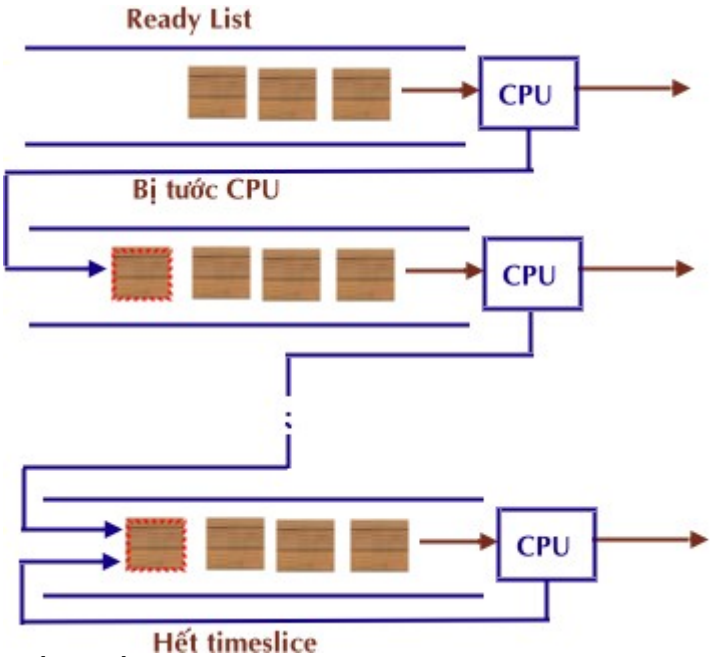
Số lượng các cấp ưu tiên

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 :

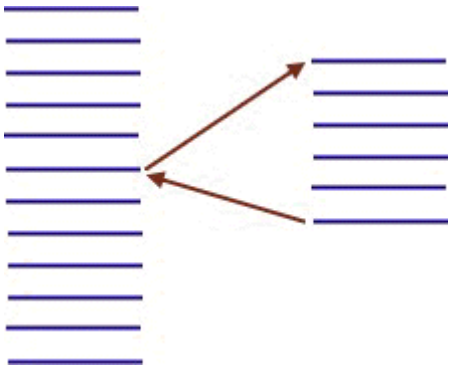
Phần cứng (ví dụ lỗi do các phép tính số học)

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

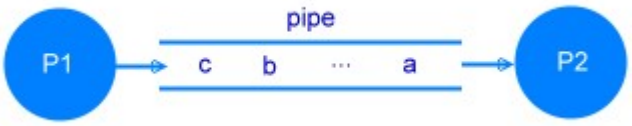
Tiếp nhận tín hiệu và xử lý theo cách đặc biệt của tiến trình.

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 !

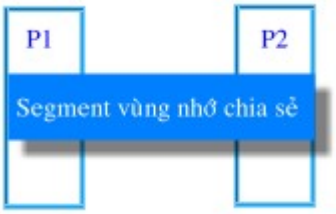
Cơ chế này cho phép truyền dữ liệu với cách thức không cấu trúc.

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 *destination*

**Receive**(source,message) : nhận một thông điệp từ *source*

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-serveur. Trong mô hình này, server sử

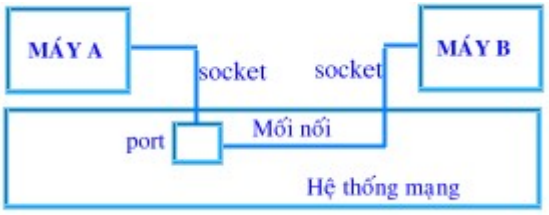
dụng lời gọi hệ thống listen và accept để nối kết với client, sau đó , client và server có thể trao đổi thông tin bằng cách sử dụng các primitive send và receive.

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 một hệ thống cho phép các tiến trình liên lạc với nhau, bao giờ hệ điều hành cũng cần cung cấp kèm theo những cơ chế đồng bộ hóa để bảo đảm hoạt động của các tiến trình đồng hành không tác động sai lệch đến nhau. Truy xuất đồng hành dữ liệu được chia sẻ có thể dẫn tới việc không đồng nhất dữ liệu. Trong phần này chúng ta sẽ thảo luận các cơ chế đảm bảo việc thực thi có thứ tự của các tiến trình hợp tác chia sẻ không gian địa chỉ để tính đúng đắn của dữ liệu luôn được duy trì.

**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 giải quyết vấn đề, cần bảo đảm tiến trình độc quyền truy xuất 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)***

Giả sử có hai tiến trình $P_1$ và $P_2$ thực hiện công việc của các kế toán, 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 ! »);
```

Giả sử trong tài khoản hiện còn 800, $P_1$ muốn rút 500 và $P_2$ muốn rút 400. Nếu xảy ra tình huống 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 !

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)* .

***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 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 vaò 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 nhặt 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:***

Tiếp cận: cho phép tiến trình cấm tất cả các ngắt trước khi vào miền găng, và phục hồi ngắt khi ra khỏi miền găng. Khi đó, ngắt đồng hồ cũng không xảy ra, do vậy hệ thống không thể tạm dừng hoạt động của tiến trình đang xử lý để cấp phát CPU cho

tiến trình khác, nhờ đó tiến trình hiện hành yên tâm thao tác trên miền găng mà không sợ bị tiến trình nào khác tranh chấp.

giải pháp này không được ưa chuộng vì rất thiếu thận trọng khi cho phép tiến trình người dùng được phép thực hiện lệnh cấm ngắt. Hơn nữa, nếu hệ thống có nhiều bộ xử lý, lệnh cấm ngắt chỉ có tác dụng trên bộ xử lý đang xử lý tiến trình, còn các tiến trình hoạt động trên các bộ xử lý khác vẫn có thể truy xuất đến miền găng !

***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

cũng giống như các giải pháp phần cứng khác, chỉ thị TSL giảm nhẹ công việc lập trình để giải quyết vấn đề, nhưng lại không dễ dàng để cài đặt chỉ thị TSL sao cho

được xử lý một cách không thể phân chia, nhất là trên máy với cấu hình nhiều bộ xử lý.

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). Semaphore**

Tiếp cận: Được **Dijkstra** đề xuất vào 1965, một semaphore ***s*** là một *biến* có các thuộc tính sau:

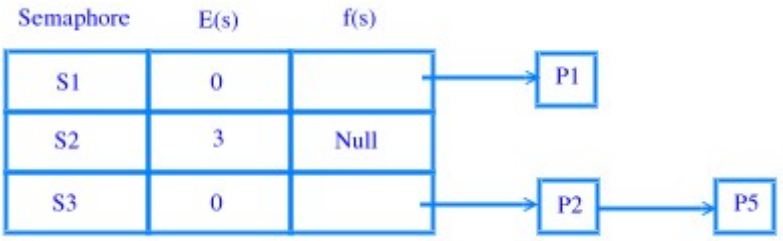
Một giá trị nguyên dương *e(s)*

Một hàng đợi *f(s)* lưu danh sách các tiến trình đang bị khóa (chờ) trên semaphore *s*

Chỉ có hai thao tác được định nghĩa trên semaphore

**Down(s):** giảm giá trị của semaphore *s* đi 1 đơn vị nếu semaphore có trị e(s) > 0, và tiếp tục xử lý. Ngược lại, nếu e(s) ☒ 0, tiến trình phải chờ đến khi e(s) >0.

**Up(s)**: tăng giá trị của semaphore *s* lên 1 đơn vị. Nếu có một hoặc nhiều tiến trình đang chờ trên semaphore *s*, bị khóa bởi thao tác **Down**, thì hệ thống sẽ chọn một trong các tiến trình này để kết thúc thao tác **Down** và cho tiếp tục xử lý.




Hình 2.17 Semaphore s

Cài đặt: Gọi *p* là tiến trình thực hiện thao tác *Down(s)* hay *Up(s)*.

**Down(s):**

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

if e(s) < 0 {

status(P)= **blocked**;

enter(P,f(s));

}

**Up(s)**:

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

if 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 nữa 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. 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 ();
}
```

**Hình 3.11** Cấu trúc một chương trình trong giải pháp semaphore

***Ví dụ:***

| Lần | Tiến trình | Thao tác | E(s) | CS | F(s) |
|---|---|---|---|---|---|
| 1 | A | Down | 0 | A | |
| 2 | B | Down | -1 | A | B |
| 3 | C | Down | -2 | A | B, C |
| 4 | A | Up | -1 | B | C |

***Tổ chức đồng bộ hóa với Semaphores***: với semaphore có thể đồng bộ hóa hoạt động của hai tiến trình trong tình huống một tiến trình phải đợi một tiến trình khác

hoàn tất thao tác nào đó mới có thể bắt đầu hay tiếp tục xử lý. Hai tiến trình chia sẻ một semaphore s, khởi gán e(s) là 0. Cả hai tiến trình có cấu trúc như sau:

**P1:**

while (TRUE) {

**job1();**

Up(s); //đánh thức P2

}

**P2:**

while (TRUE) {

Down(s); // chờ P1

**job2();**

}

**Hình 3.12** Cấu trúc chương trình trong giải pháp semaphore

Nhờ có thực hiện một các không thể phân chia, semaphore đã giải quyết được vấn đề tín hiệu "đánh thức" bị thất lạc. Tuy nhiên, nếu lập trình viên vô tình đặt các primitive Down và Up sai vị trí, thứ tự trong chương trình, thì tiến trình có thể bị khóa vĩnh viễn.

Ví dụ : while (TRUE) {

Down(s)

**critical-section ();**

Noncritical-section ();

}

tiến trình trên đây quên gọi Up(s), và kết quả là khi ra khỏi miền găng nó sẽ không cho tiến trình khác vào miền găng !

Vì thế việc sử dụng đúng cách semaphore để đồng bộ hóa phụ thuộc hoàn toàn vào lập trình viên và đòi hỏi lập trình viên phải hết sức thận trọng.

**b). Monitors**

Tiếp cận: Để có thể dễ viết đúng các chương trình đồng bộ hóa hơn, Hoare(1974) và Brinch & Hansen (1975) đã đề nghị một cơ chế cao hơn được cung

cấp bởi ngôn ngữ lập trình , là *monitor*. Monitor là một cấu trúc đặc biệt bao gồm các thủ tục, các biến và cấu trúc dữ liệu có các thuộc tính sau :

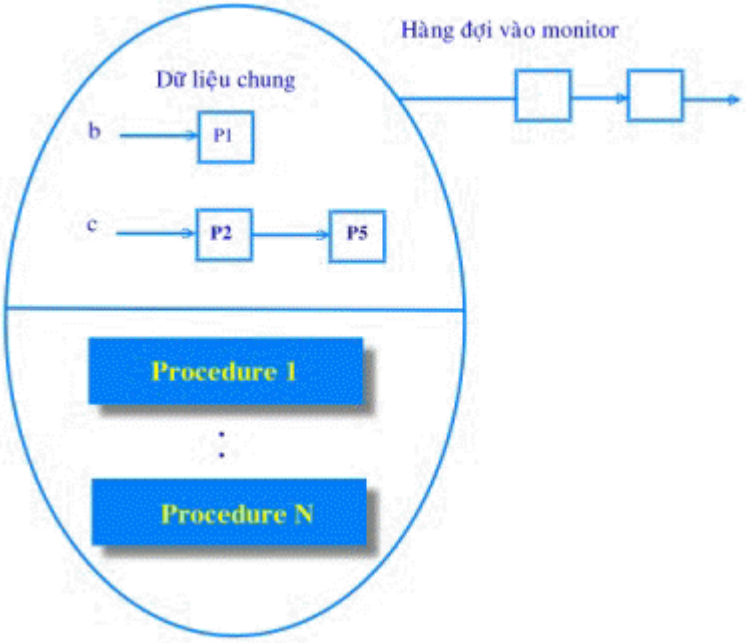
Các biến và cấu trúc dữ liệu bên trong monitor chỉ có thể được thao tác bởi các thủ tục định nghĩa bên trong monitor đó. (*encapsulation*).

Tại một thời điểm, chỉ có một tiến trình duy nhất được hoạt động bên trong một monitor (*mutual exclusive*).

Trong một monitor, có thể định nghĩa các *biến điều kiện* và hai thao tác kèm theo là **Wait** và **Signal** 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 sẽ rời khỏi monitor.




Hình 2.18 Monitor và các biến điều kiện

Cài đặt : 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). Trao đổi thông điệp**

Tiếp cận: 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 hay gởi vào hộp thư.

**Receive(source,message)**: nhận một thông điệp thừ một tiến trình hay từ bất kỳ một tiến trình nào, tiến trình gọi sẽ chờ nếu không có thông điệp nào để nhận.

Sử dụng: Có nhiều cách thức để thực hiện việc truy xuất độc quyền bằng cơ chế trao đổi thông điệp. Đây là một mô hình đơn giản: một tiến trình kiểm soát việc sử dụng tài nguyên và nhiều tiến trình khác yêu cầu tài nguyên này. Tiến trình có yêu cầu tài nguyên 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.Khi sử dụng xong tài nguyên , tiến trình gởi một thông điệp khác đến tiến trình kiểm soát để báo kết thúc truy xuất. Về phần tiến trình kiểm soát , khi nhận được thông điệp yêu cầu tài nguyên, nó sẽ chờ đến khi tài nguyên 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 ();

}

**Hình 3.16** Cấu trúc tiến trình yêu cầu tài nguyên trong giải pháp message

Các primitive semaphore và monitor có thể giải quyết được vấn đề truy xuất độc quyền trên các máy tính có một hoặc nhiều bộ xử lý chia sẻ một vùng nhớ chung. Nhưng các primitive không hữu dụng trong các hệ thống phân tán, khi mà mỗi bộ xử lý sỡ hữu một bộ nhớ riêng biệt và liên lạc thông qua mạng. Trong những hệ thống phân tán như thế, cơ chế trao đổi thông điệp tỏ ra hữu hiệu và được dùng để giải quyết bài toán đồng bộ hóa.

## 2.5. Tắc nghẽn (Deadlock)

### 2.5.1. Định nghĩa:

Một tập hợp các tiến trình được định nghĩa ở trong tình trạng *tắc nghẽn* khi mỗi tiến trình trong tập hợp đều chờ đợi một sự kiện mà chỉ có một tiến trình khác trong tập hợp mới có thể phát sinh được.

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 !

### 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:

Có sử dụng tài nguyên không thể chia sẻ (Mutual exclusion): Mỗi thời điểm, một tài nguyên không thể chia sẻ được hệ thống cấp phát chỉ cho một tiến trình , khi tiến trình sử dụng xong tài nguyên này, hệ thống mới thu hồi và cấp phát tài nguyên cho tiến trình khác.

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 tai một chu kỳ trong đồ thi 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

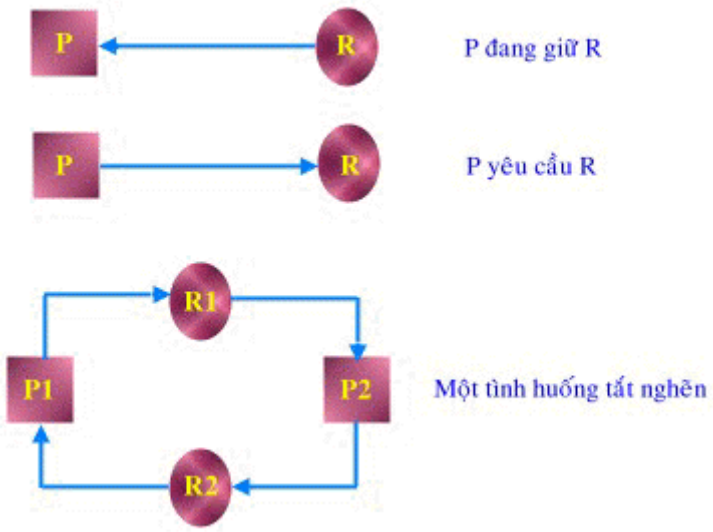
Pn đang chờ tài nguyên đang giữ bởi P0

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.

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 :

- Sử dụng một vài giao thức (protocol) để bảo đảm rằng 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.

- Cho phép xảy ra tắc nghẽn và tìm cách sữa chữa tắc nghẽn.

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

### 2.5.4 Ngăn chặn tắc nghẽn

Để tắc nghẽn không xảy ra, cần bảo đảm tối thiểu một trong 4 điều kiện cần không xảy ra:

Tài nguyên không thể chia sẻ : nhìn chung gần như không thể tránh được điều kiện này vì bản chất tài nguyên gần như cố định. Tuy nhiên đối với một số tài nguyên về kết xuất, người ta có thể dùng các cơ chế spooling để biến đổi thành tài nguyên có thể chia sẻ.

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 :

Tiến trình phải yêu cầu tất cả các tài nguyên cần thiết trước khi bắt đầu xử lý .

=> 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.

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 cùng lần với tài nguyên mới.

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

Không thu hồi tài nguyên: cho phép hệ thống được thu hồi tài nguyên từ các tiến trình bị khoá và cấp phát trở lại cho tiến trình khi nó thoát khỏi tình trạng bị khóa.

Tuy nhiên với một số loại tài nguyên, việc thu hồi sẽ rất khó khăn vì vi phạm sự toàn vẹn dữ liệu .

Tồn tại một chu kỳ: tránh tạo chu kỳ 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

Ngăn cản tắc nghẽn là một mối bận tâm lớn khi sử dụng tài nguyên. Tránh tắc nghẽn là loại bỏ tất cả các cơ hội có thể dẫn đến tắc nghẽn trong tương lai. Cần phải sử dụng những cơ chế phức tạp để thực hiện ý định này.

**Một số khái niệm cơ sở**

**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[NumProcs, NumResources] = Available;

int Finish[NumProcs] = false;

2.Tìm  i sao cho

Finish[i] == false

Need[i] <= Work[i]

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.

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 | | |
| 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 thõa 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 | | |

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

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

| Need | | | Allocation | | | Available | | |
|---|---|---|---|---|---|---|---|---|
| R1 | R2 | R3 | R1 | R2 | R3 | R1 | R2 | R3 |
| P1 | 0 | 0 | 0 | 0 | 0 | 0 | | |
| P2 | 0 | 0 | 0 | 0 | 0 | 0 | | |
| P3 | 0 | 0 | 0 | 0 | 0 | 0 | | |
| P4 | 4 | 2 | 0 | 0 | 0 | 2 | | |

| Need | | | Allocation | | | Available | | |
|---|---|---|---|---|---|---|---|---|
| R1 | R2 | R3 | R1 | R2 | R3 | R1 | R2 | R3 |
| P1 | 0 | 0 | 0 | 0 | 0 | 0 | | |
| P2 | 0 | 0 | 0 | 0 | 0 | 0 | | |
| P3 | 0 | 0 | 0 | 0 | 0 | 0 | | |
| P4 | 0 | 0 | 0 | 0 | 0 | 0 | | |

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

**Giải thuật yêu cầu tài nguyên**

Giả sử tiến trình Pi yêu cầu *k* thể hiện của tài nguyên *r*.

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

2.Nếu k <= Available[i],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[i] = Available[i] - k;
Allocation[i]= Allocation[i]+ k;
Need[i] = Need[i] - k;

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

**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 hai lựa chọn chính để hiệu chỉnh 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

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.

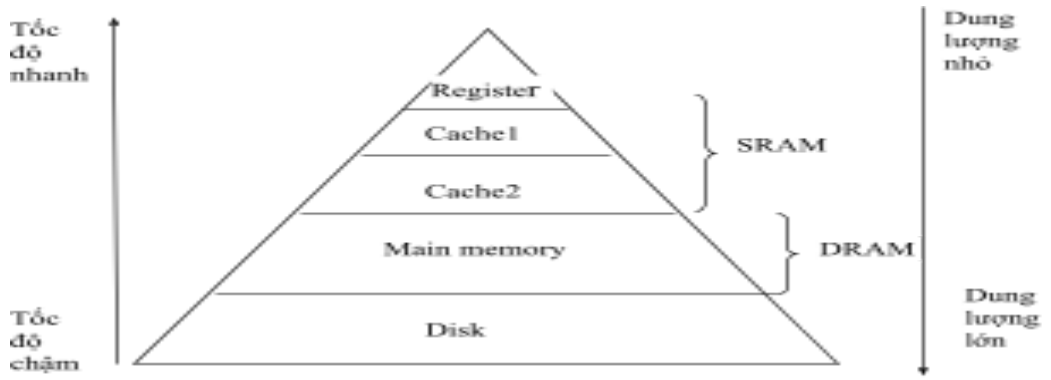
Tình trạng « đói tài nguyên »: làm sao bảo đảm rằng không có một tiến trình luôn luôn bị thu hồi tài nguyên ?

# CHƯƠNG 3 : ĐIỀU KHIỂN BỘ NHỚ

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 thông tin: làm thế nào để cho phép hai tiến trình có thể chia sẻ thông tin trong bộ nhớ?

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

## 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.

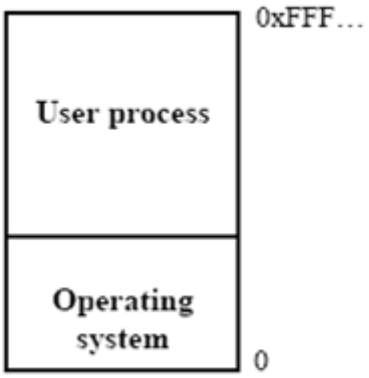
Chương trình của người sử dụng chỉ thao tác trên các địa chỉ ảo, không bao giờ nhìn thấy các địa chỉ vật lý . Địa chỉ thật sự ứng với vị trí của dữ liệu trong bô nhớ chỉ được xác định khi thực hiện truy xuất đến dữ liệu.

## 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

### 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 dung, 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 một ngắt sẽ được phát sinh để báo cho hệ thống về một truy xuất bất hợp lệ.

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.

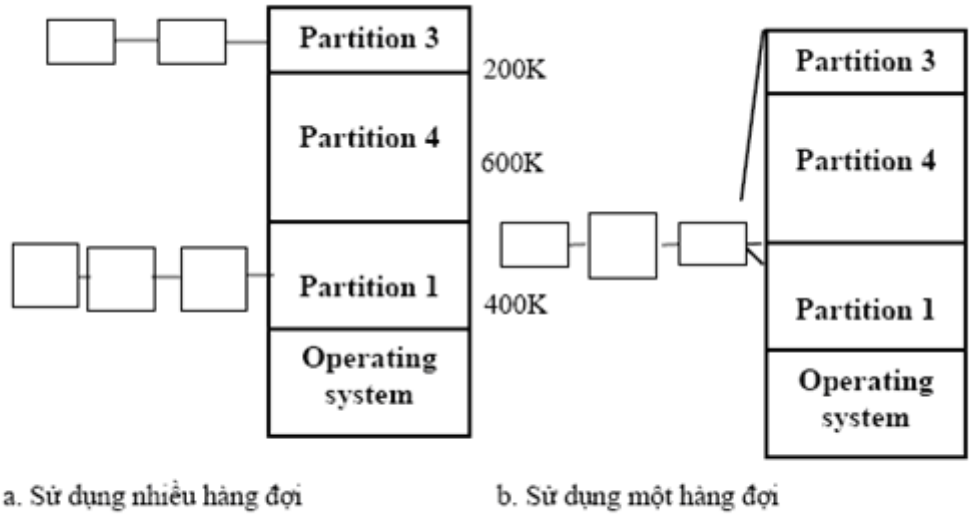
### 3.4.2 Hệ thống đa chương với phân vùng cố định

Một trong những phương pháp đơn giản nhất để cấp phát bộ nhớ là chia bộ nhớ thành những phân vùng có kích thước cố định. Các phân vùng khác nhau có thể có kích thước khác nhau hay bằng nhau. Mỗi phân vùng chỉ có thể chứa một tiến trình. Do đó, cấp độ đa chương được giới hạn bởi số lượng phân vùng. Trong phương pháp đa phân vùng, khi một phân vùng rảnh, một tiến trình được chọn từ hàng đợi nhập và được nạp vào phân vùng trống. 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:

• **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 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**: tất cả các tiến trình được đặt trong hàng đợi duy nhất. Khi có một phân vùng trống, tiến trình đầu tiên trong hàng đợi có kích thước phù hợp sẽ được đặt vào phân vùng và cho xử lý.




Hình 3.3 Cấp phát đa vùng với phân vùng cố định

Trong trường hợp tiến trình đầu tiên có kích thước nhỏ trong khi phân vùng tự do là lớn sẽ dẫn tới lãng phí bộ nhớ.

Giải pháp: Khi có một phân vùng rỗi thì tìm trên toàn bộ hàng đợi này tiến trình lớn nhất đặt vừa rong phân vùng này, nạp tiến trình vào bộ nhớ chính.

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:

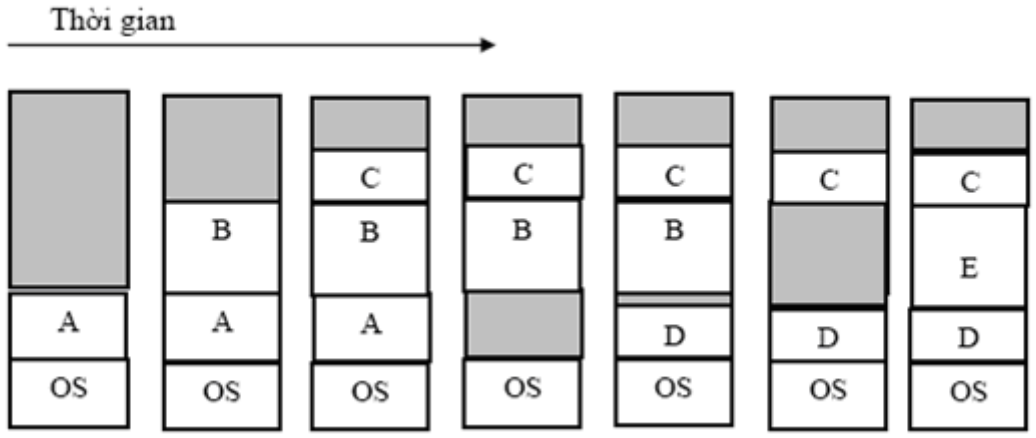
Tổng bộ nhớ nhỏ tự do, rời rạc còn lớn nhưng không thể sử dụng để nạp tiến trình khác.

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.

### 3.4.3 Hệ thống đa chương với phân vùng động

Hệ điều hành giữ một bảng hiển thị những phần nào của bộ nhớ là rỗi và phần nào đang bận. Ban đầu, tất cả bộ nhớ là sẵn dùng cho tiến trình người dùng, và được xem như một khối lớn bộ nhớ sẵn dùng. Khi một tiến trình đến và cần bộ nhớ, hệ điều hành tìm kiếm một vùng trống đủ lớn cho tiến trình này. Nếu tìm thấy, hệ điều hành sẽ cấp phát cho tiến trình phần bộ nhớvừa đúng với kích thước của tiến trình, phần bộ nhớ còn lại dành cho các tiến trình khác.




Hình 3.4 Cấp phát đa vùng với phân vùng động

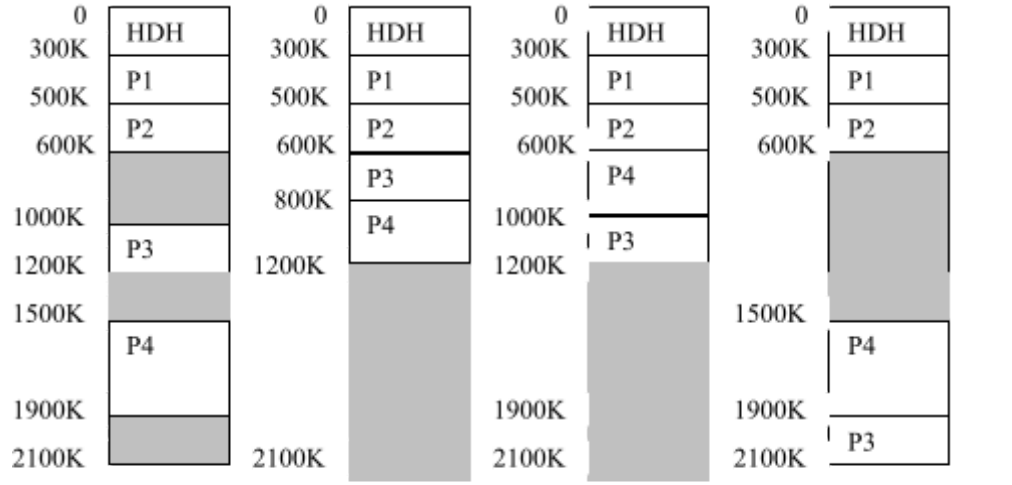
Thông thường, một tập hợp các vùng trống có kích thước khác nhau được phân tán khắp bộ nhớ tại bất cứ thời điểm được cho. Khi một tiến trình đến và yêu cầu bộ nhớ, hệ thống tìm tập hợp này một vùng trống đủ lớn cho tiến trình này. Nếu vùng trống quá lớn, nó được chia làm hai: một phần được cấp cho tiến trình đến; phần còn lại được trả về tập hợp các vùng trống. Nếu vùng trống mới nằm kề với các vùng trống khác, các vùng trống nằm kề này được gom lại để tạo thành một vùng trống lớn hơn.

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 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. 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. Chúng ta phải tìm toàn bộ danh sách, trừ khi danh sách được xếp thứ tự theo kích thước.

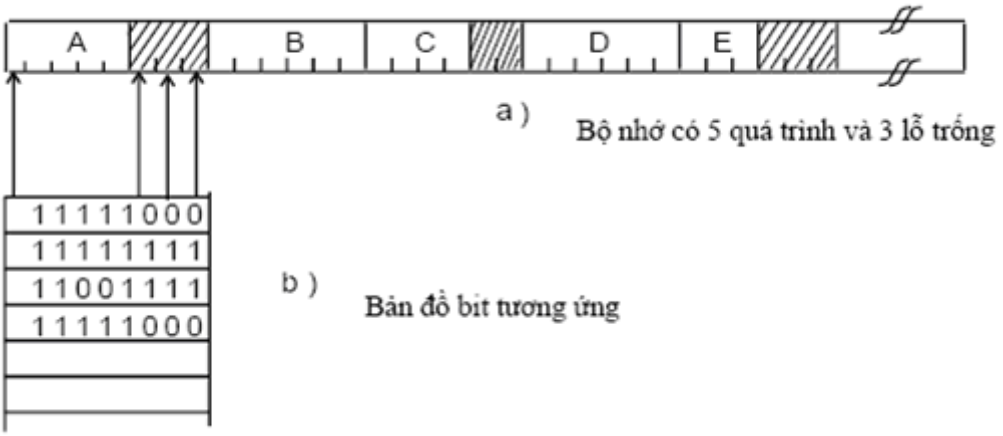
**Worst-fit**: cấp phát vùng nhớ tự do lớn nhất đủ lớn để chứa tiến trình. Chúng ta phải tìm toàn bộ danh sách trừ khi nó được xếp theo thứ tự kích thước.

**Quản lý các khối rỗi bận**

- **Quản lý bằng bản đồ bit**:

Bộ nhớ được chia thành các đơn vị cấp phát, mỗi đơn vị 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.

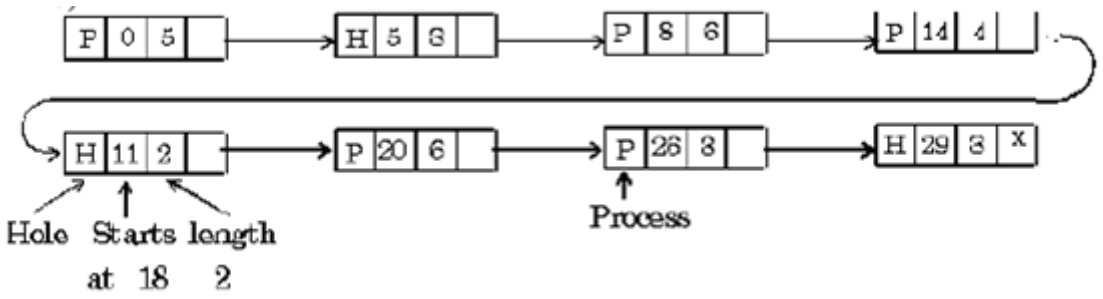
Kích thước của đơn vị cấp phát là vấn đề lớn trong thiết kế. Nếu kích thước đơn vị cấp phát nhỏ sẽ làm tăng kích thước của bản đồ bit. Ngược lại, nếu kích thước đơn vị cấp phát lớn có thể gây hao phí cho đơn vị cấp phát sau cùng. Đây là giải pháp đơn giản nhưng thực hiện chậm nên ít được dùng.




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 1 bit đầu để xác định phân đoạn đó là vùng trống (H) hay một tiến trình (P), sau đó là 3 từ để chỉ địa chỉ bắt đầu, chiều dài và chỉ điểm tới mục kế tiếp. Việc sắp xếp các phân đoạn theo địa chỉ hay theo kích thước tuỳ thuộc vào giải thuật quản lý bộ nhớ.



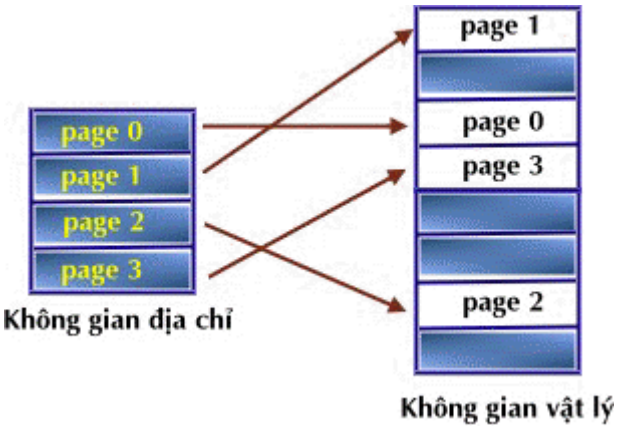

Hình 3.7 Quản lý bộ nhớ bằng danh sách liên kết

## 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 Phân trang ( Paging)

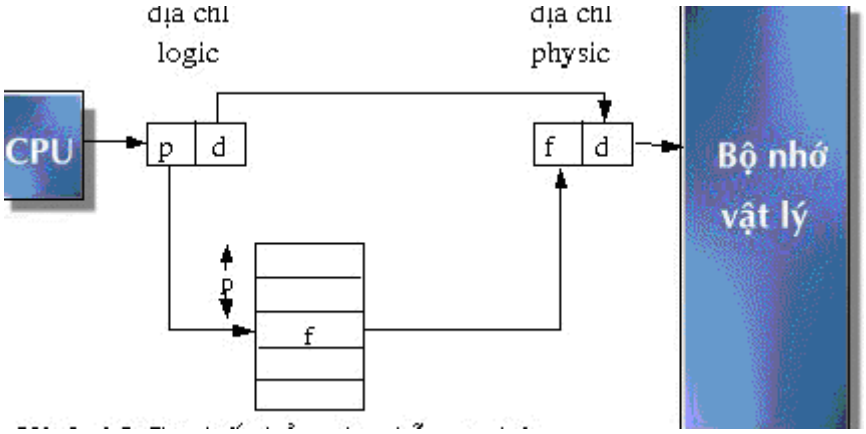
**Ý tưởng**:




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 trìnhsẽ được nạp vào những khung trang còn trống. Một tiến trình kích thước N trang sẽ yêu cầu N khung trang tự do.

**Cơ chế MMU trong kỹ thuật phân trang:**



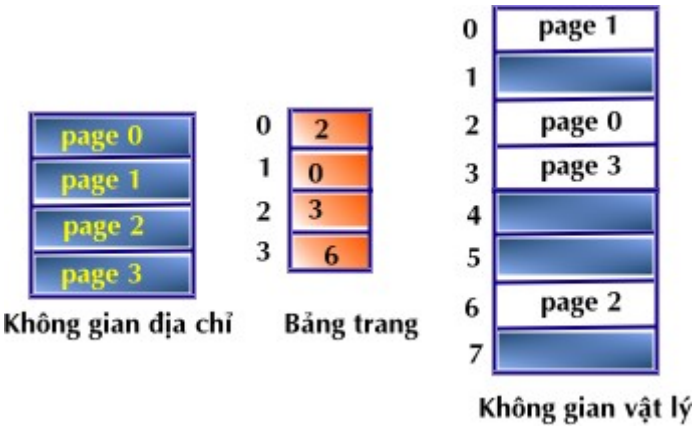

Hình 3.9 Cơ chế phần cứng hỗ trợ phân trang

Cơ chế phần cứng hỗ trợ thực hiện chuyển đổi địa chỉ trong cơ chế phân trang là bảng trang (*pages table*):

Mỗi tiến trình có một bảng trang.

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

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 trang 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.

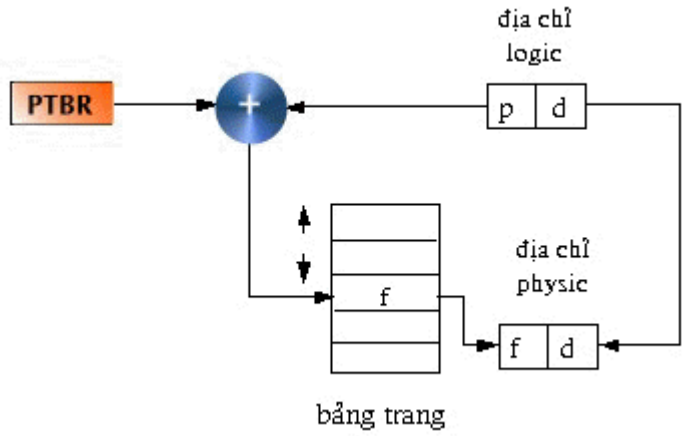
| Số trang | Độ dời trang |
|---|---|
| P | D |
| m –n | N |

**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).

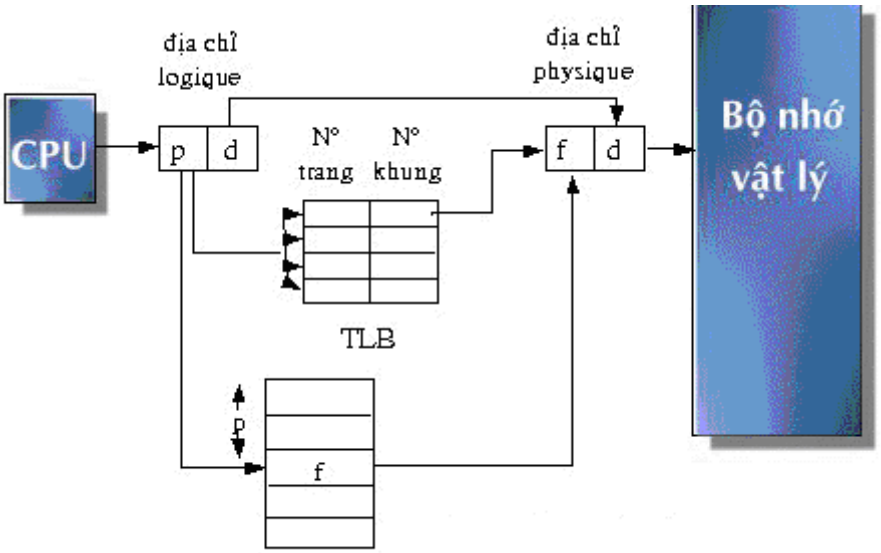
Theo cách tổ chức này, mỗi truy xuất đến dữ liệu hay chỉ thị đều đòi hỏi hai lần truy xuất bộ nhớ : một cho truy xuất đến bảng trang và một cho bản thân dữ liệu, do vậy truy cập chậm.




Hình 3.11 Sử dụng thanh ghi nền trỏ đến bảng trang

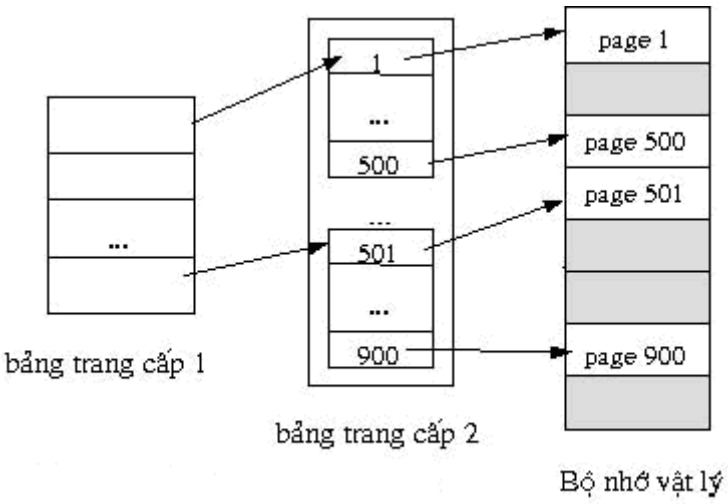
- Để nâng cao tốc độ truy xuất, sử dụng thêm một vùng nhớ đặc biệt , với tốc độ truy xuất nhanh và cho phép tìm kiếm song song, vùng nhớ cache nhỏ này thường được gọi là bộ nhớ kết hợp (translation look-aside buffer TLBs). Mỗi thanh ghi trong bộ nhớ kết hợp chứa số hiệu trang và số hiệu khung trang tương 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

**Tổ chức bảng trang:**




Hình 3.13 Bảng trang 2 cấp

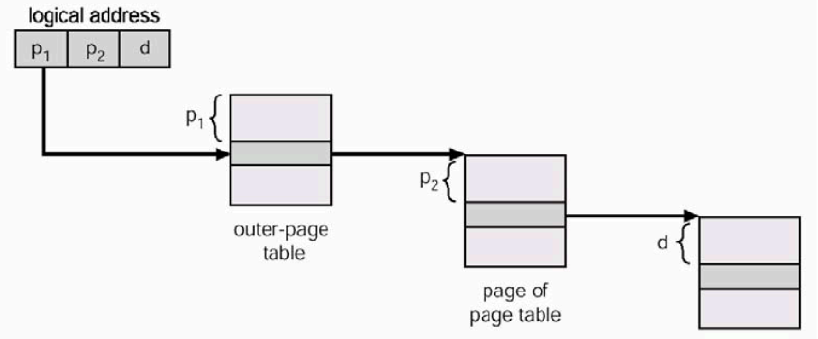
Mỗi hệ điều hành có một phương pháp riêng để tổ chức lưu trữ bảng trang. Đa số các hệ điều hành cấp cho mỗi tiến trình một bảng trang. Tuy nhiên phương pháp này không thể chấp nhận được nếu hệ điều hành cho phép quản lý một không gian địa chỉ có dung lượng quá ($2^{32}$, $2^{64}$): trong các hệ thống như thế, bản thân bảng trang đòi hỏi một vùng nhớ qúa lớn!

Thí dụ, xét một hệ thống với không gian địa chỉ luận lý 32 bit. Nếu kích thước trang 4KB thì bảng trang có thể chứa tới 1 triệu mục từ ($2^{32}/2^{12}$). Giả sử rằng mỗi mục từ chứa 4 bytes, mỗi tiến trình có thể cần tới 4MB không gian địa chỉ vật lý cho một bảng trang. Rõ ràng, chúng ta sẽ không muốn cấp phát bảng trang liên tiếp nhau. Một giải pháp đơn giản cho vấn đề này là chia bảng trang thành những phần nhỏ hơn.

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

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 |

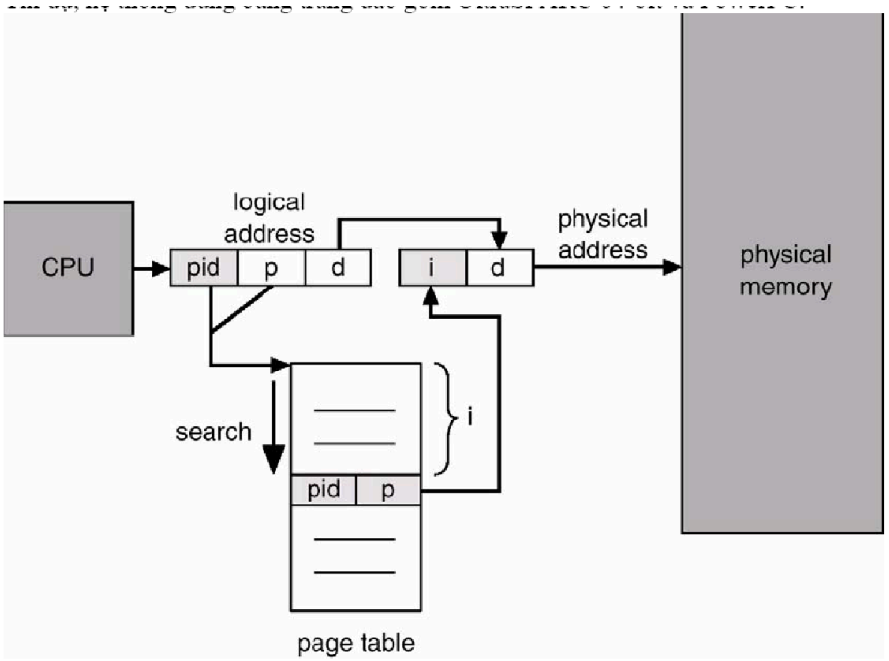
***Bảng trang nghịch đảo*** (inverted page table).

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 <idp, 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.

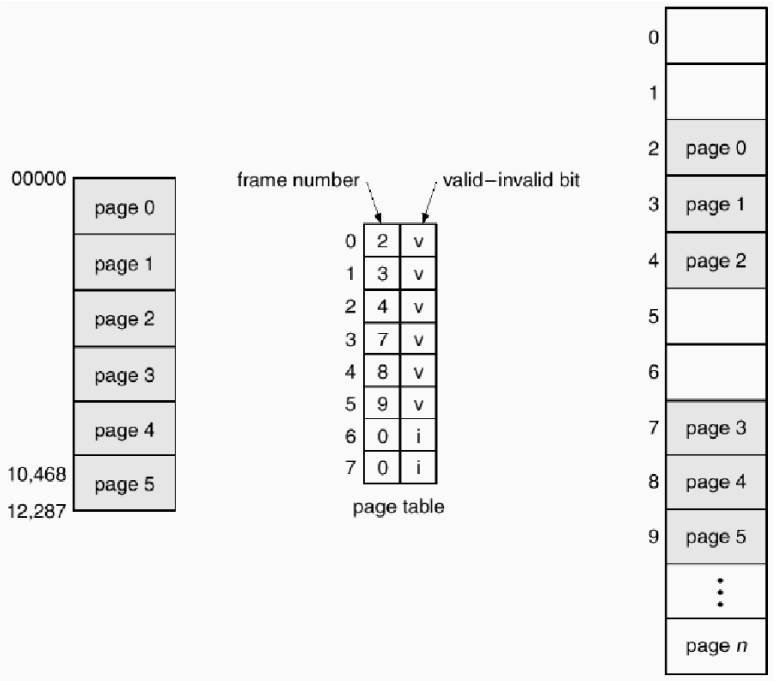
Ngoài ra, một bit phụ trội được thêm vào trong cấu trúc một phần tử của bảng trang : bit hợp lệ-bất hợp lệ (valid-invalid).

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

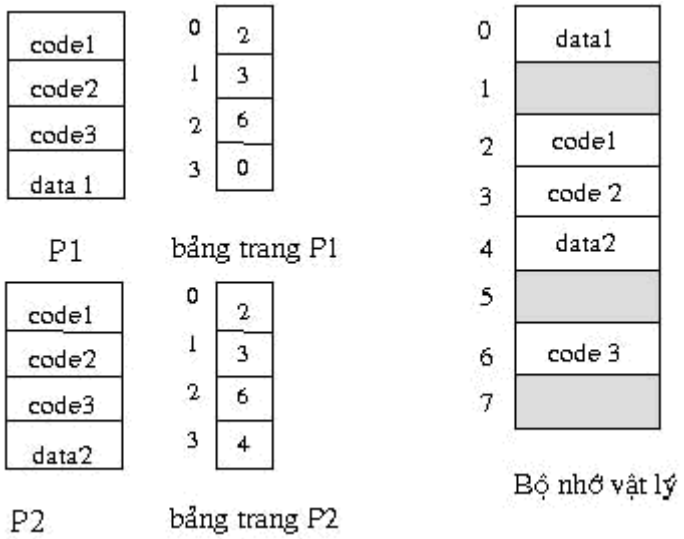
| số hiệu khung trang | bit valid-invalid |
|---|---|

*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.

**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.

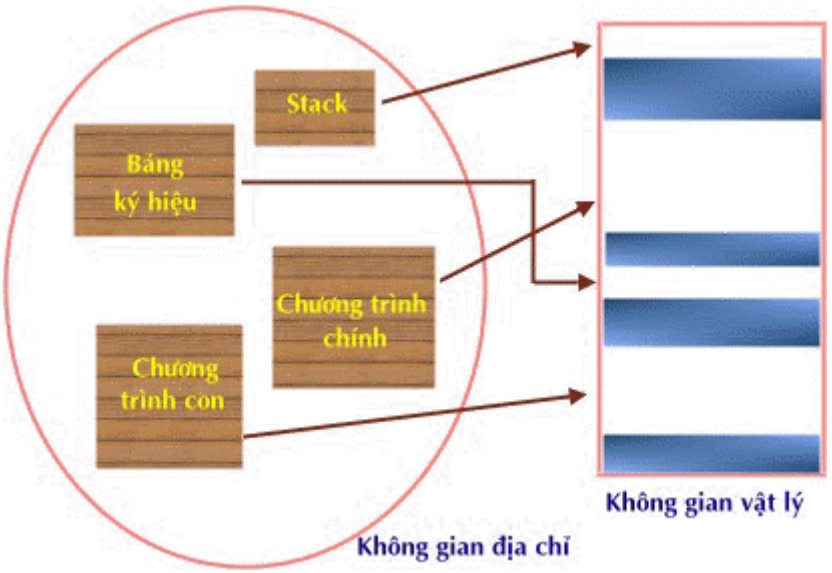
*Góc nhìn của bộ nhớ vật lý:* một tiến trình của người sử dụng được lưu trữ phân tán khắp bộ nhớ vật lý, trong bộ nhớ vật lý đồng thời cũng chứa những tiến trình khác.

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)

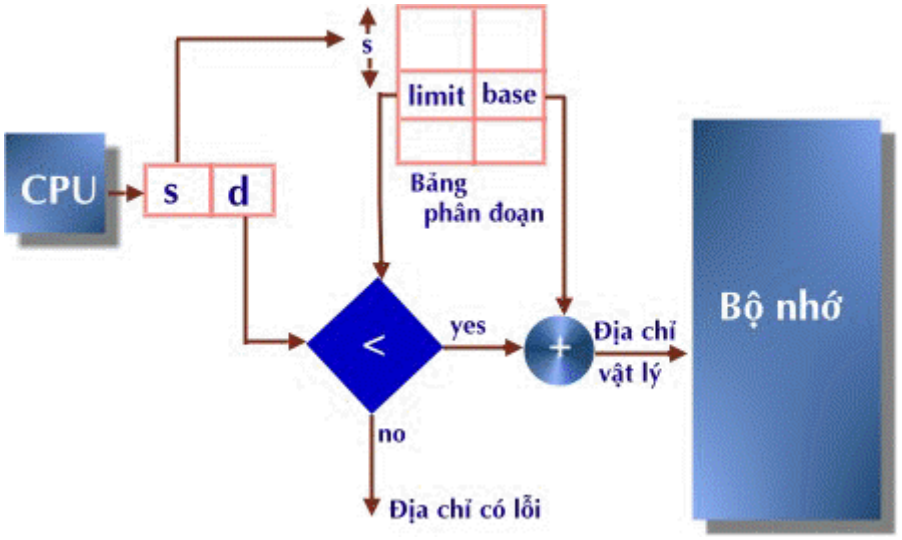
Lưu ý rằng sự phân trang không phản ánh đúng cách thức người sử dụng cảm nhận về bộ nhớ. Người sử dụng nhìn thấy bộ nhớ như một tập các đối tượng của chương trình (segments, các thư viện...) và một tập các đối tượng dữ liệu (biến toàn cục, stack, vùng nhớ chia sẻ...). Vấn đề đặt ra là cần tìm một cách 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

**Ý tưởng**: quan niệm không gian địa chỉ là một tập các *phân đoạn (segments)* – các phân đoạn là những phần bộ nhớ *kích thước khác nhau và có liên hệ logic với nhau*. Mỗi phân đoạn có một tên gọi (số hiệu phân đoạn) và một độ dài. Người dùng sẽ thiết lập mỗi địa chỉ với hai giá trị : <*số hiệu phân đoạn*, *offset*>.




Hình 3.18 Mô hình phân đoạn bộ nhớ

**Cơ chế MMU trong kỹ thuật phân đoạn:**




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

Cần phải xây dựng một ánh xạ để chuyển đổi các địa chỉ 2 chiều được người dùng định nghĩa thành địa chỉ vật lý một chiều. Sự chuyển đổi này được thực hiện qua một *bảng phân đoạn*. Mỗi thành phần trong bảng phân đoạn bao gồm một *thanh ghi nền* và một *thanh ghi giới hạn*. Thanh ghi nền lưu trữ địa chỉ vật lý nơi bắt đầu phân đoạn trong bộ nhớ, trong khi thanh ghi giới hạn đặc tả chiều dài của phân đoạn.

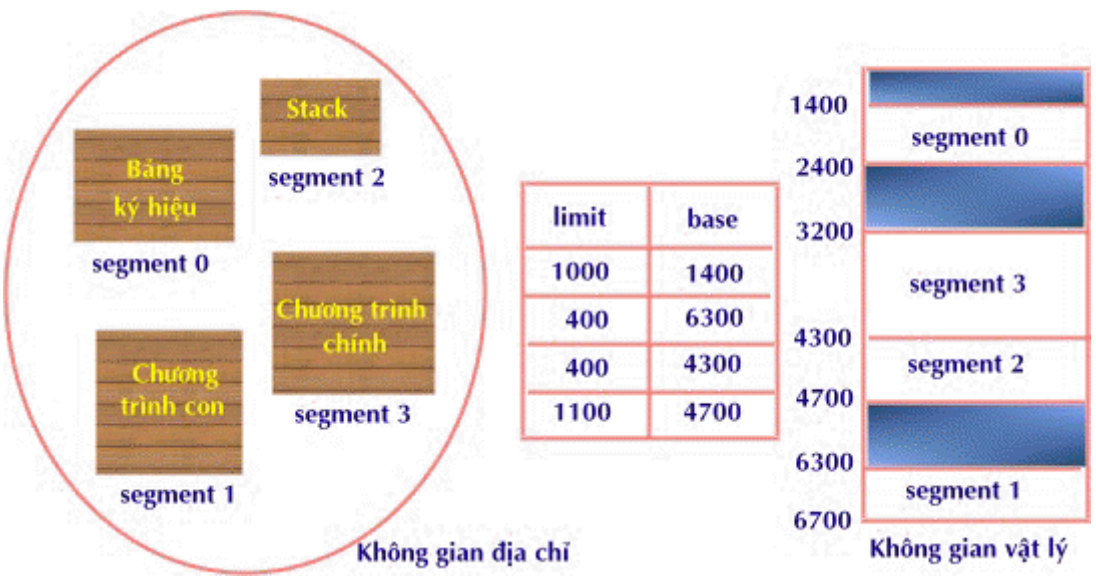
**Chuyển đổi địa chỉ:**

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 trong thanh ghi nền để phát sinh địa chỉ vật lý tương ứng.

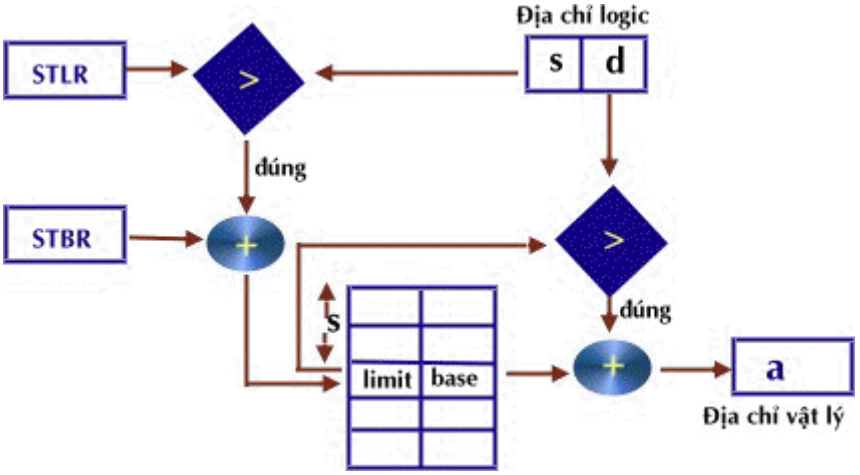
**Cài đặt bảng phân đoạn**:




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 biến động, cần sử dụng thêm một *thanh ghi đặc tả kích thước bảng phân đoạn* (STLR).

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*)

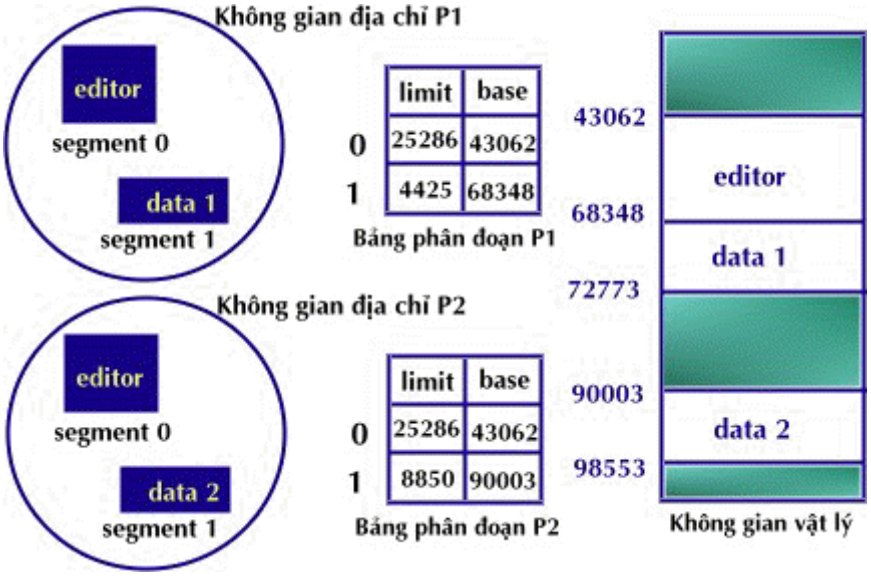


Hình 3.21 Sử dụng STBR, STLR và bảng phân đoạn

**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.

**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.

Kỹ thuật phân đoạn thõa mãn được nhu cầu thể hiện cấu trúc logic của chương trình nhưng nó dẫn đến tình huống phải cấp phát các khối nhớ có kích thước khác nhau cho các phân đoạn trong bộ nhớ vật lý. Điều này làm rắc rối vấn đề hơn rất nhiều so

với việc cấp phát các trang có kích thước tĩnh.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.

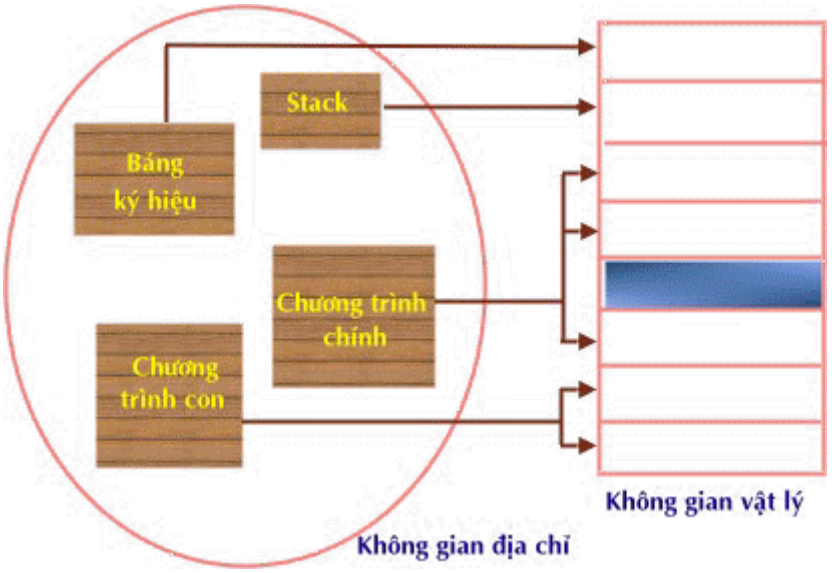
**Cơ chế MMU trong kỹ thuật phân đoạn kết hợp phân trang**:

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***>

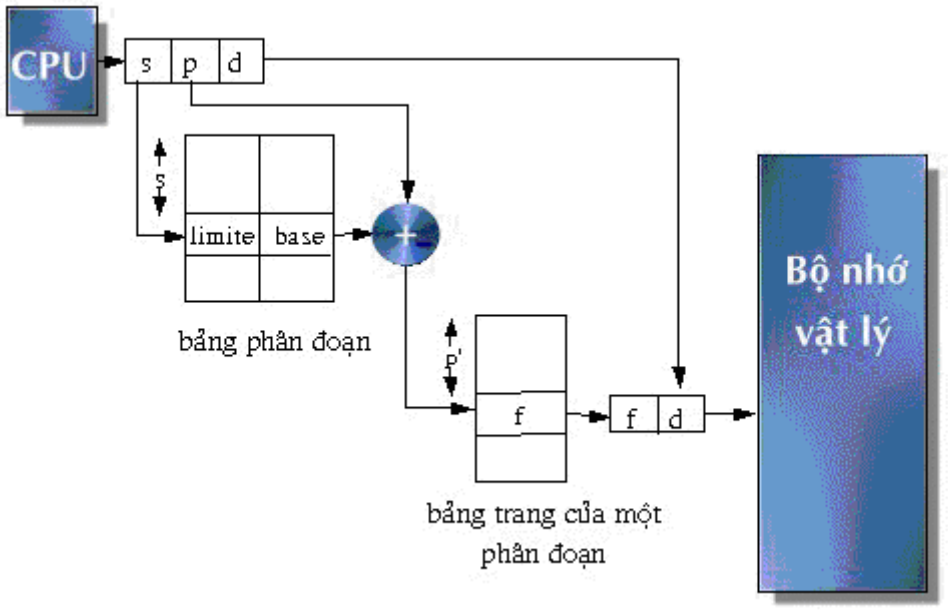
*số hiệu phân đoạn (s)*: sử dụng như chỉ mục đến phần tử tương ứng trong bảng phân đoạn.




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 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 Bộ nhớ ảo (Virtual Memory)

*Bộ nhớ ảo là một kỹ thuật hiện đại giúp cho người dùng được giải phóng hoàn toàn khỏi mối bận tâm về giới hạn bộ nhớ. Ý tưởng, ưu điểm và những vấn đề liên quan đến việc tổ chức bộ nhớ ảo sẽ được trình bày trong bài học này.*

Nếu đặt toàn thể không gian địa chỉ vào bộ nhớ vật lý, thì kích thước của chương trình bị giới hạn bởi kích thước bộ nhớ vật lý.

Thực tế, trong nhiều trường hợp, chúng ta không cần phải nạp toàn bộ chương trình vào bộ nhớ vật lý cùng một lúc, vì tại một thời điểm chỉ có một chỉ thị của tiến trình được xử lý. Ví dụ, các chương trình đều có một đoạn code xử lý lỗi, nhưng đoạn code này hầu như rất ít khi được sử dụng vì hiếm khi xảy ra lỗi, trong trường hợp này, không cần thiết phải nạp đoạn code xử lý lỗi từ đầu.

Từ nhận xét trên, một giải pháp được đề xuất là cho phép thực hiện một chương trình chỉ được nạp từng phần vào bộ nhớ vật lý. Ý tưởng chính của giải pháp này là tại mỗi thời điểm chỉ lưu trữ trong bộ nhớ vật lý các chỉ thị và dữ liệu của chương trình cần thiết cho việc thi hành tại thời điểm đó. Khi cần đến các chỉ thị khác, những chỉ thị mới sẽ được nạp vào bộ nhớ, tại vị trí trước đó bị chiếm giữ bởi các chỉ thị nay không còn cần đến nữa. Với giải pháp này, một chương trình có thể lớn hơn kích thước của vùng nhớ cấp phát cho nó.

Một cách để thực hiện ý tưởng của giải pháp trên đây là sử dụng kỹ thuật *overlay*. Kỹ thuật overlay không đòi hỏi bất kỳ sự trợ giúp đặc biệt nào của hệ điều hành , nhưng trái lại, lập trình viên phải biết cách lập trình theo cấu trúc overlay, và điều này đòi hỏi khá nhiều công sức.

Để giải phóng lập trình viên khỏi các suy tư về giới hạn của bộ nhớ, mà cũng không tăng thêm khó khăn cho công việc lập trình của họ, người ta nghĩ đến các kỹ thuật tự động, cho phép xử lý một chương trình có kích thước lớn chỉ với một vùng nhớ có kích thước nhỏ . Giải pháp được tìm thấy với khái niệm *bộ nhớ ảo (virtual memory)*.

### 3.6.1. Định nghĩa

Bộ nhớ ảo là một kỹ thuật cho phép xử lý một tiến trình không được nạp toàn bộ vào bộ nhớ vật lý. Bộ nhớ ảo mô hình hoá bộ nhớ như một bảng lưu trữ rất lớn và đồng nhất, tách biệt hẳn khái niệm không gian địa chỉ và không gian vật lý. Người sử dụng chỉ nhìn thấy và làm việc trong không gian địa chỉ ảo, việc chuyển đổi sang không gian vật lý do hệ điều hành thực hiện với sự trợ giúp của các cơ chế phần cứng cụ thể.

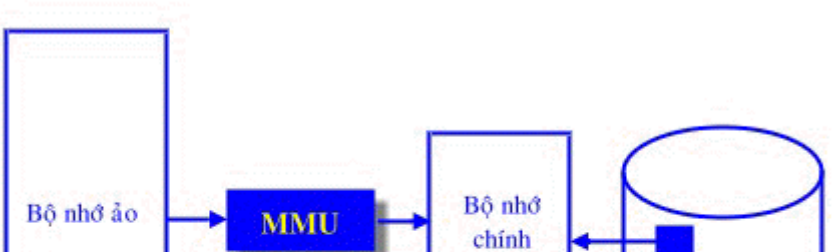
**Thảo luận**:

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

Nhờ việc tách biệt bộ nhớ ảo và bộ nhớ vật lý, có thể tổ chức một bộ nhớ ảo có kích thước lớn hơn bộ nhớ vật lý.

Bộ nhớ ảo cho phép giảm nhẹ công việc của lập trình viên vì họ không cần bận tâm đến giới hạn của vùng nhớ vật lý, cũng như không cần tổ chức chương trình theo cấu trúc overlays.





**Hình 3.25** Bộ nhớ ảo

### 3.6.2. Cài đặt bộ nhớ ảo

Bộ nhớ ảo thường được thực hiện với kỹ thuật *phân trang theo yêu cầu (demand paging)*. Cũng có thể sử dụng kỹ thuật *phân đoạn theo yêu cầu ( demand segmentation)* để cài đặt bộ nhớ ảo, tuy nhiên việc cấp phát và thay thế các phân đoạn phức tạp hơn thao tác trên trang, vì kích thước không bằng nhau của các đoạn.

**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:

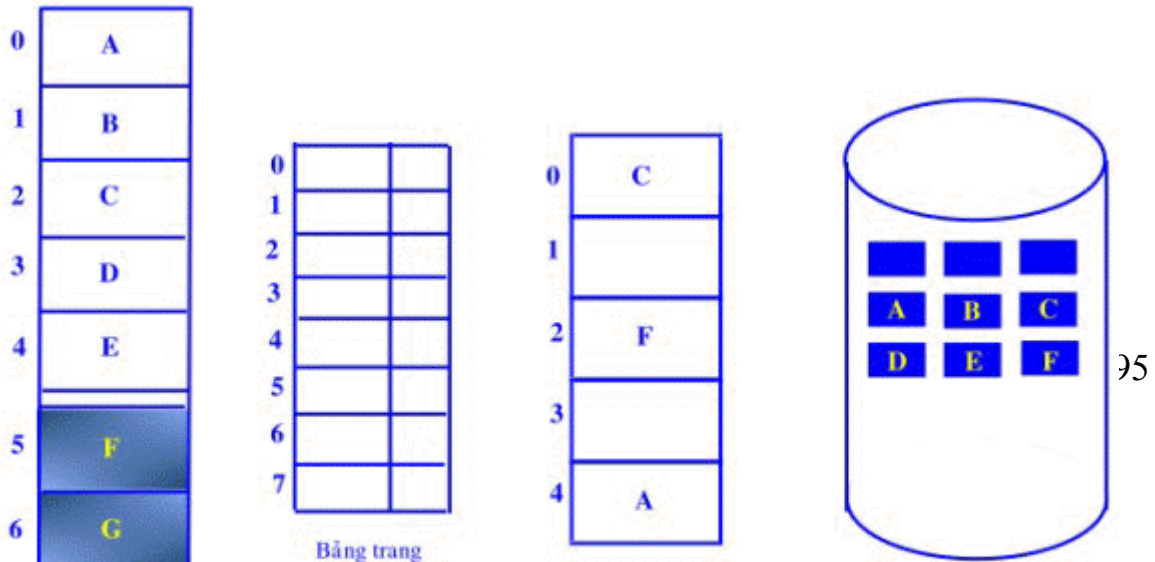
*valid* : trang tương ứng là hợp lệ và đang ở trong bộ nhớ chính .

*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.

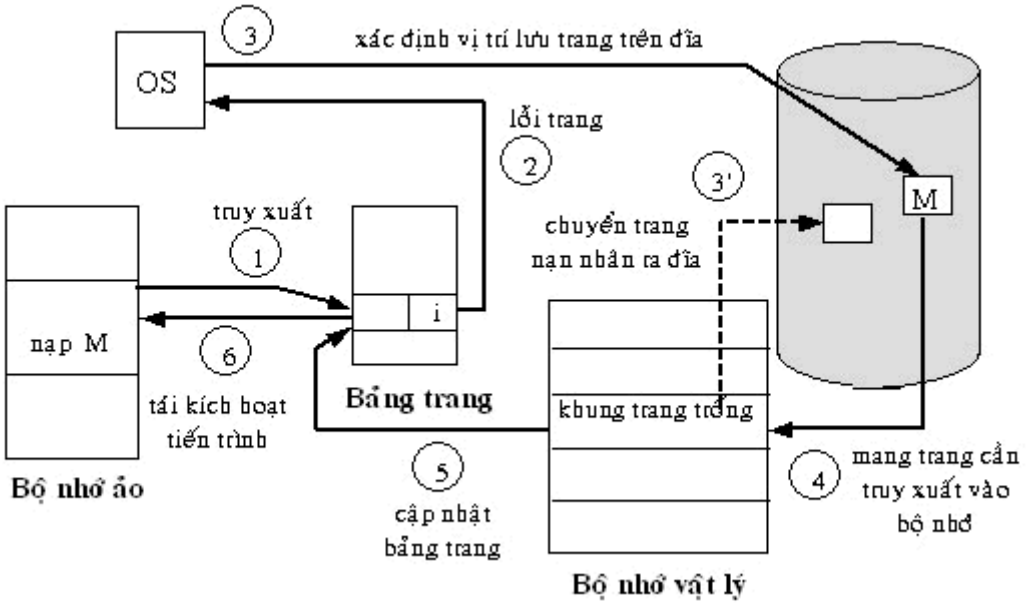
Tìm một khung trang trống trong bộ nhớ chính :

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 sữ 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 they thế trang FIFO dễ hiểu, dễ cài đặt. Tuy nhiên khi thực hiện không phải lúc nào cũng có kết qủa 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.

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 Cấu trúc đĩa cứng

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

Là loại bộ nhớ không thay đổi (Non-Volatile)

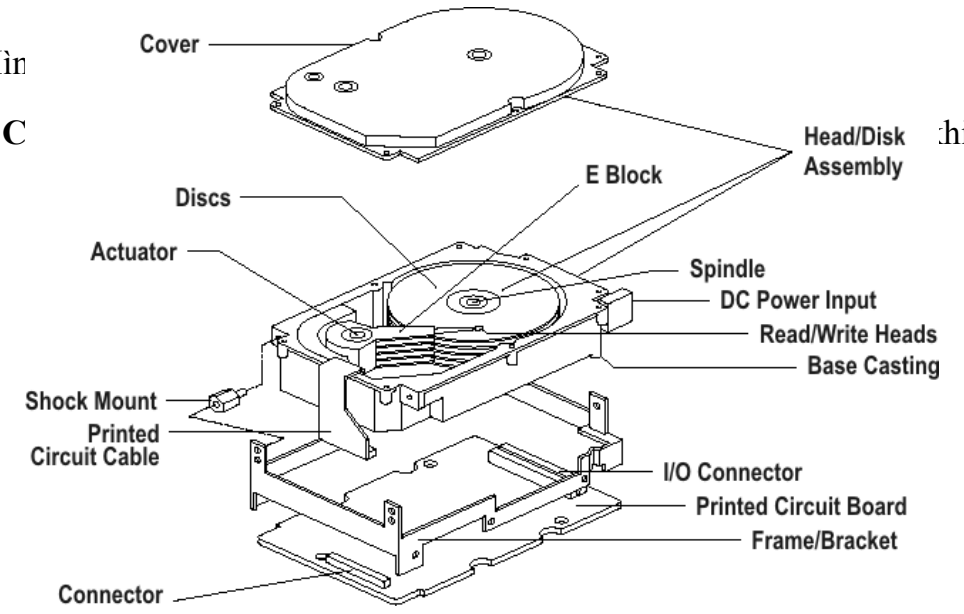
Có vai trò quan trọng trong hệ thống.

Dung lượng ngày càng được nâng lên và kích thước nhỏ đi.

**- Cấu tạo và nguyên lý hoạt động.**

Hìn

**- C** hiển, cổng kết nối.





Hình 4.2

**Vỏ đĩa cứng**: Là bảng gắn linh kiện có chức năng bảo vệ.

**Đĩa từ:** Cấu tạo nhôm hay thủy tinh, gốm,…Bề mặt phủ lớp từ tính. Xếp chồng nhau và dữ liệu ở cả 2 mặt.

Hình 4.3

**Trục quay (Động cơ quay):** Truyền chuyển động quay. Cấu tạo nhẹ, chính xác

**Đầu đọc ghi:** Cấu tạo gồm lõi ferit và cuộn dây. Cấu tạo nhỏ, đọc dữ liệu từ hóa trên mặt đĩa.

Hình 4.4

**Mạch điều khiển:** Điều khiển động cơ đồng trục và cần đọc ghi. Bộ nhớ đệm. Đầu kết nối giao tiếp.

**Cổng kết nối:** Kết nối với mainboard.Các chuẩn ATA, SATA, SATAII,…

Hình 4.5

**- Cấu trúc bề mặt đĩa**

**Track:**

Trên một bề mặt đĩa được chia ra nhiều vòng đồng tâm gọi là track.

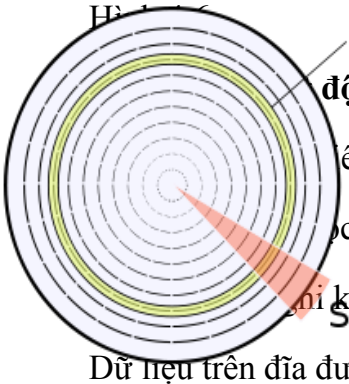
Trên track được chia ra các phần nhỏ bằng các đoạn hướng tâm gọi là Sector (512Byte)

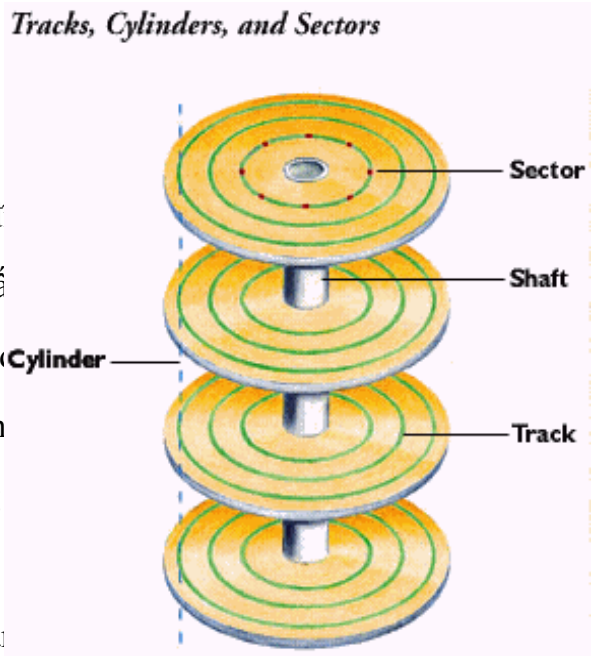
Được định dạng ở cấp thấp (Low format)

**Cylinder:**

Tập hợp các track cùng bán kính (ở các mặt đĩa khác nhau)

Trên một ổ cứng có nhiều cylinder




**động**

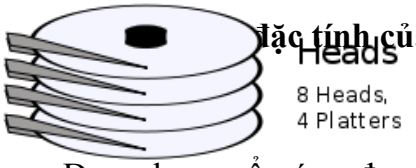
ên dữ liệu trên đ

c/ghi để truy xuấ

i khi đầu đọc đưa

Dữ liệu trên đĩa được lưu dưới dạn

**ặc tính của ổ cứng**




Dung lượng ổ cứng được tính bằ (số cylinder) × (số đầu đọc/ghi).

Theo nhà SX: 1GG = 1000 MB

Tốc độ quay

Số vòng/phút (*revolutions per minute*)

Tốc độ quay tỉ lệ thuận với thời gian truy xuất dữ liệu

3.600 > 15.000

Bộ nhớ đệm

Có nhiệm vụ như RAM, lưu dữ liệu tạm thời suốt thời gian làm việc của ổ cứng.

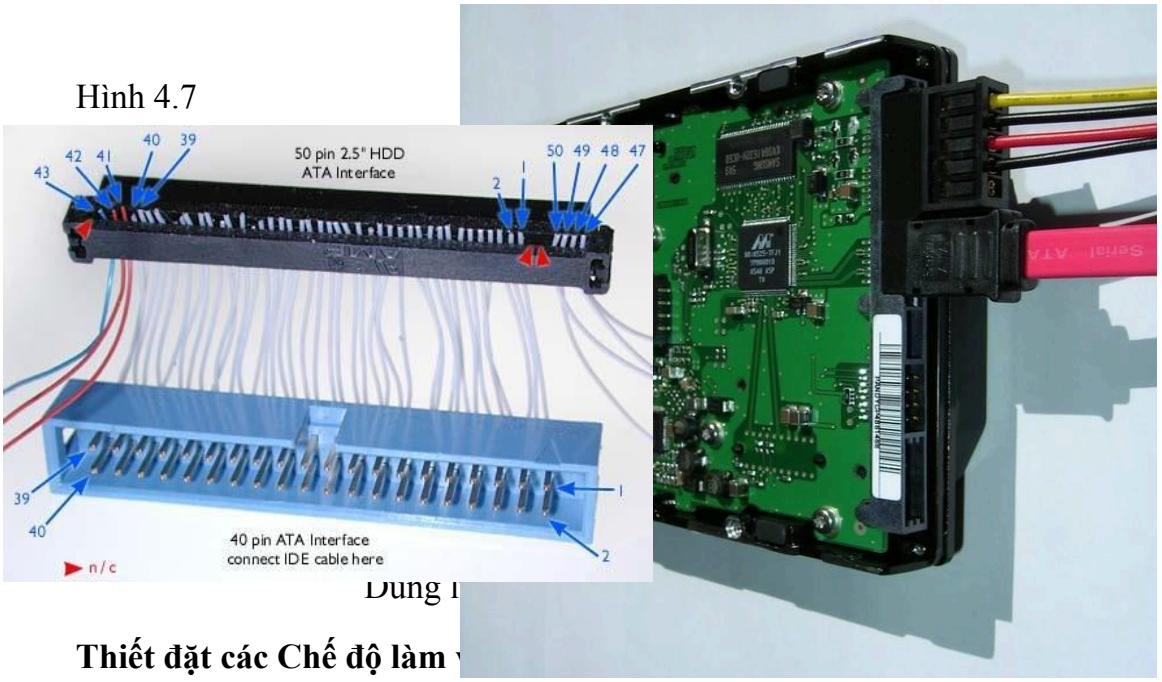
Bộ nhớ đệm càng cao càng tốt (phụ thuộc hiệu suất hoạt động của ổ cứng)

Chuẩn giao tiếp

- Các chuẩn phổ biến ATA, SATA, SCSI, Fibre Channel (máy chủ máy trạm) tốc độ cao

- Chuẩn **SATA II** đã xuất hiện với băng thông **300MB/s (hay** 3Gb/s), gấp đôi so với SATA(150MB/s).

Hình 4.7




Dung l

**Thiết đặt các Chế độ làm v**

Thiết đặt phần cứng

-Thiết đặt kênh

Chuẩn giao tiếp ATA, trên cùng một cáp truyền

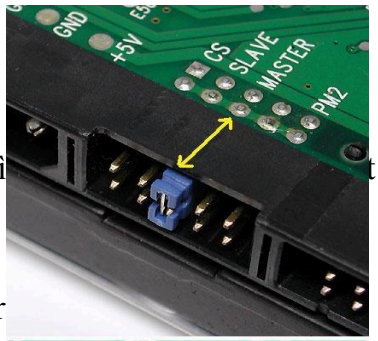
Jumper (cầu đấu)

M = Master

SL = Slave

CS = Cable Select

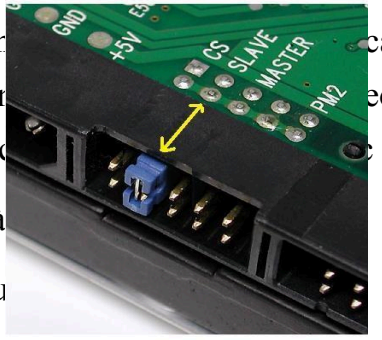
-Thiết đặt chuẩn giao tiếp




Hi t kênh

Thiết đặt phần mềm

-Phân vùng (Par




Phân vùr các vùng ghi nhớ dữ liệu trên các cylinder gầ eo thiết đặt của người sử dụng để sử dụng cho c nhau.

Có thể cà

Hỗ trợ qu

- Định dạng phân vùng

FAT (File Allocation Table) Chuẩn hỗ trợ DOS và các hệ điều hành họ Windows 9X/Me.

FAT32 (File Allocation Table, 32-bit) được hỗ trợ bắt đầu từ hệ điều hành Windows 95 OSR2

NTFS (Windows NT File System) Được hỗ trợ bắt đầu từ các hệ điều hành họ NT/2000/XP/Vista.

- 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))

**Ổ cứng và các lỗi liên quan.**

- Không nhận ổ đĩa khi cắm mới
- Không tìm thấy hệ điều hành

- Thường xuyên bị treo, truy cập ứng dụng chậm

**Khắc phục.**

- Kiểm tra từng phần để loại bỏ từ từ
- Kiểm tra bằng phần mềm SCANDISK
- Phân vùng lại nếu cần

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 |

## 4.2 Hệ thống bảng Fat



Hình 4.9

FAT là vùng chứa thông tin định vị các vùng chóa nội dung các file trên đĩa

a) Bootsector

Bao gồm bảng tham số vật lý của đĩa và chương trình khởi động của HĐH (nếu có)

BPB (BIOS Parameters block)

| offset | size | ý nghĩa |
|---|---|---|
| 0 | 3 byte | lệnh nhẩy |
| 3 | 8 byte | chứa phiên bản của DOS |
| 11 | 2 byte | Kích thước 1 sector |
| 13 | 1 byte | Số sector cho một đơn vị cấp phát |
| 14 | 2 byte | Dự trữ |
| 16 | 1 byte | Số bảng FAT |
| 17 | 2 byte | Số phần tử tối đa có thể có trong thư mục gốc |
| 19 | 2 byte | Chỉ tổng số sector trên đĩa<32M(cũ) |
| 21 | 1byte | Nhận khuôn dạng đĩa (F8: đĩa cứng) |
| 22 | 2 byte | Số sector trong bảng FAT |
| 24 | 2 byte | Số sector trong một rãnh |
| 26 | 2 byte | Số đầu đọc ghi |
| 28 | 2 byte | Số sector ẩn |
| 30 | 2 byte | Dự trữ |
| 32 | 4 byte | Tổng số sector trên đĩa>32M |
| 36 | 1 byte | Số driver vật lý(đĩa mềm:0, đĩa cứng: 80) |
| 37 | 1 byte | Byte dự trữ |
| 38 | 1 byte | chữ ký của Bootsector |
| 39 | 4 byte | Số volume của đĩa |
| 43 | 11 byte | Nhãn đĩa |
| 54 | 8byte | Chứa một đoạn text miêu tả |
| 62 | byte | bắt đầu chương trình mồi |

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

b) FAT

FAT 12 dùng 12 bit biểu diễn 1 FAT entry

FAT 16 dùng 16 bit biểu diễn 1 FAT entry

FAT 32 dùng 32 bit biểu diễn 1 FAT entry

$2^{12}$=(0…4096 FAT entry)

$2^{16}$=(0…65535 FAT entry)

| FAT entry | FAT 12 | FAT 16 | Ý nghĩa |
|---|---|---|---|
| 0 | 0FD | 00FD | Đĩa mềm |
| 1 | FFE | FFFE | Vùng đĩa không sử dụng |
| 2 | 3 | 3 | Vùng lưu giữ tiếp theo |
| 3 | FF7 | FFF7 | Vùng đĩa hỏng |
| 6 | FFF | FFFF | kết thúc File |
| 7 | 000 | 0000 | Vùng đĩa trống |

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

c) Root Directory (DIR)

Từ thư mục gốc cho phép định vị dượctàn bộ cấu trúc logic của đĩa logic.

Lối vào thư mục 32 byte

0-7 Tên file

8-10 Phần mở rộng

11 Thuộc tính của File

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| Dự trữ | Dự trữ | Archive | Director y | volum | system | hiden | readonly |

12-21 Dự trữ của DOS

22-23 Ngày tháng tạo File

24-25 Giờ phút tạo File

26-27 Chỉ ra nơi đầu tiên lưu trữ file

28-31 Chỉ độ lớn của File.

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.

## 4.3 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 |

## 4.4 Các thông số và thuật toán truy nhập đĩa

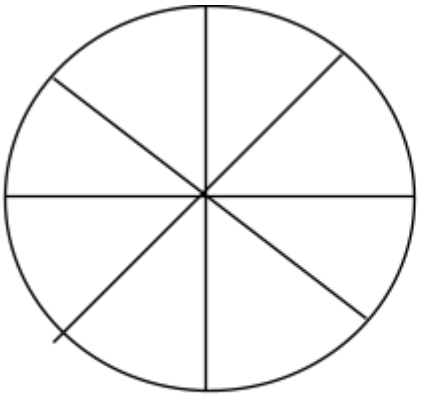
### 4.4.1Các thông số

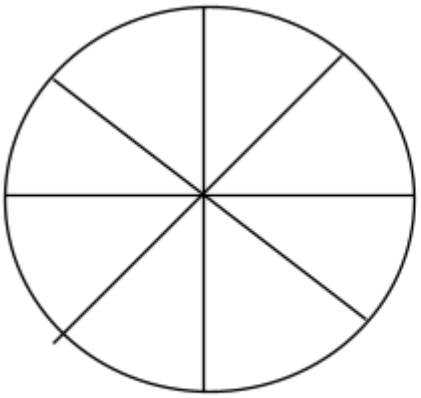
Thời gian truy xuất dữ liệu=thời gian tìm kiếm+thời gian dịch chuyển + thời gian quay trễ

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.





0
3
6
1
0
4
7
2
5
Xen kẽ với hệ số đan xen là 3
0

1
2
3
0
4
5
6
7

Không đan xen

Hình 4.10

### 4.4.2 Các thuật toán đọc đĩa

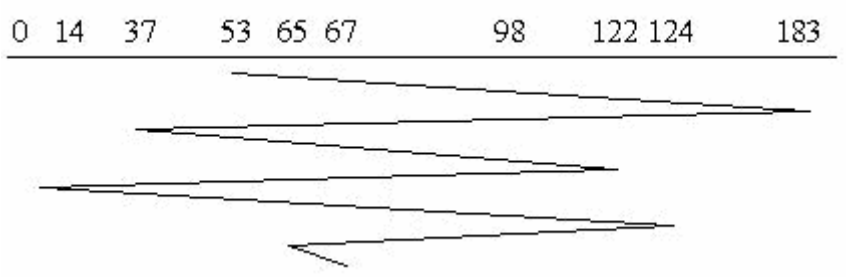
Tất cả mọi công việc đều phụ thuộc vào việc nạp chương trình và nhập xuất tập tin, do đó điều quan trọng là dịch vụ đĩa phải càng nhanh càng tốt. Hệ điều hành có thể tổ chức dịch vụ truy xuất đĩa tốt hơn bằng cách lập lịch yêu cầu truy xuất đĩa.

**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

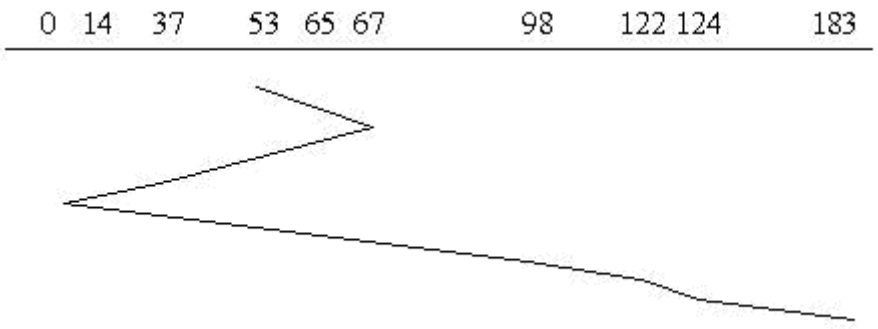
**b)Lập lịch SSTF (shortest-seek-time-first)**

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

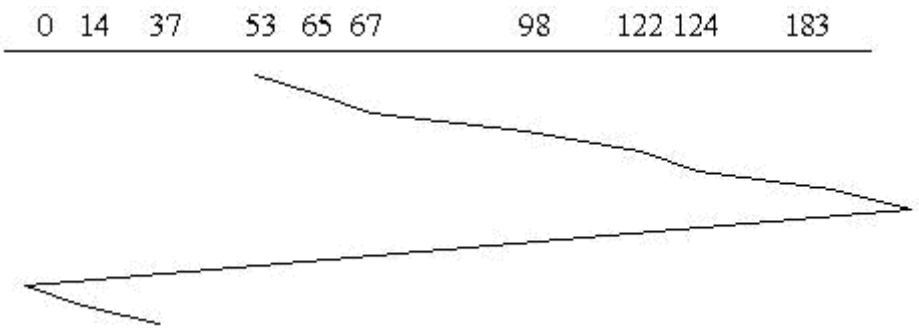
Giả sử hiện tại đầu đọc đang ở vị trí 53. Như vậy đầu đọc lần lượt đi qua các khối

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**

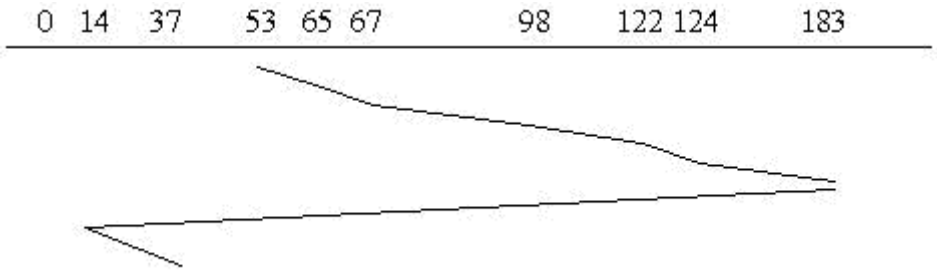
Thuật toán này tương tự như thuật toán SCAN, chỉ khác là khi nó di chuyển đến một đầu nào đó của đĩa, nó sẽ lập tức trở về đầu bắt đầu của đĩa. Lấy lại ví dụ trên, khi đó thứ tự truy xuất các khối sẽ là : 53, 65, 67, 98, 122, 124, 183, 199, 0, 14, 37 như hình sau :




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.

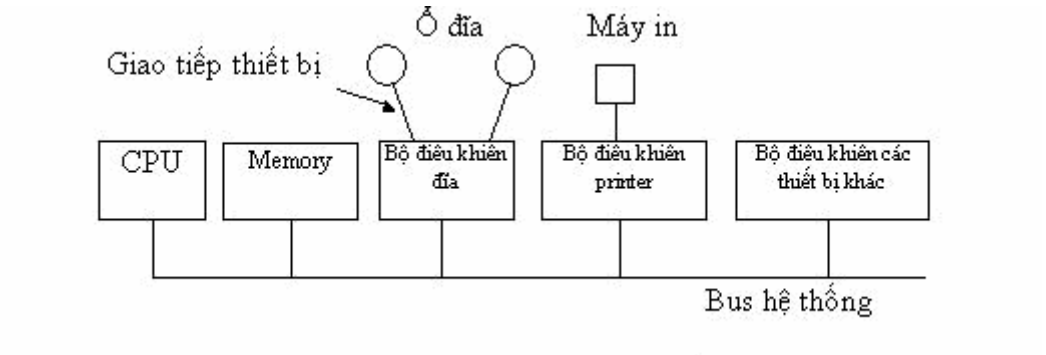
Sử dụng cơ chế DMA, bộ xử lý truy xuất những dữ liệu I/O trực tiếp trong bộ nhớ chính.

### 5.2.3 Bộ điều khiển thiết bị

Một đơn vị bị 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 dể 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 khái niệm liên quan

**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 cuối cùng | Ngày và giờ truy xuất tập tin gần nhất |
| 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.

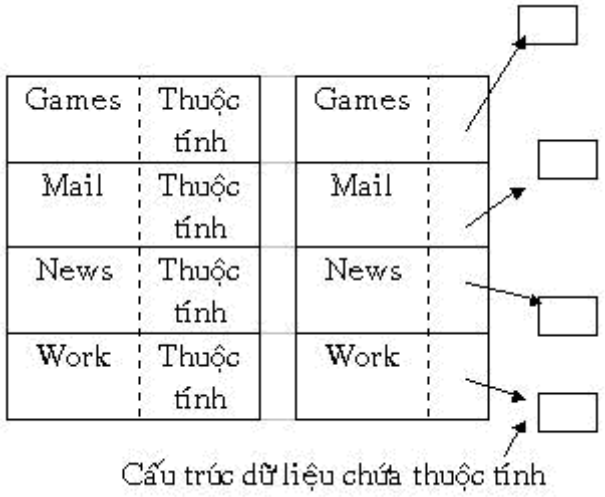
Khi một tập tin được mở, hệ điều hành tìm trên thư mục của nó cho tới khi tìm thấy tên của tập tin được mở. Sau đó nó sẽ xác định thuộc tính cũng như địa chỉ lưu trữ trên đĩa và đưa vào một bảng trong bộ nhớ. Những truy xuất sau đó thực hiện trong bộ nhớ chính.

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.

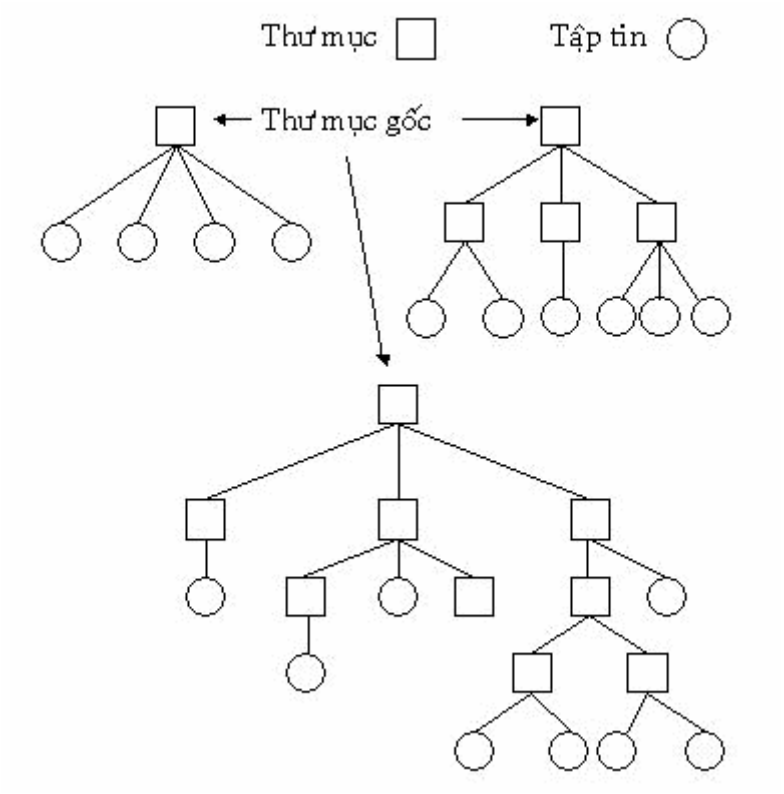
Cách thứ hai là có một thư mục gốc và trong đó có nhiều thư mục con, trong mỗi thư mục con chứa tập tin của người sử dụng (còn gọi là thư mục hai cấp), cách này tránh được trường hợp xung đột tên nhưng cũng còn khó khăn với người dùng có nhiều tập tin. Người sử dụng luôn muốn nhóm các ứng dụng lại một 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

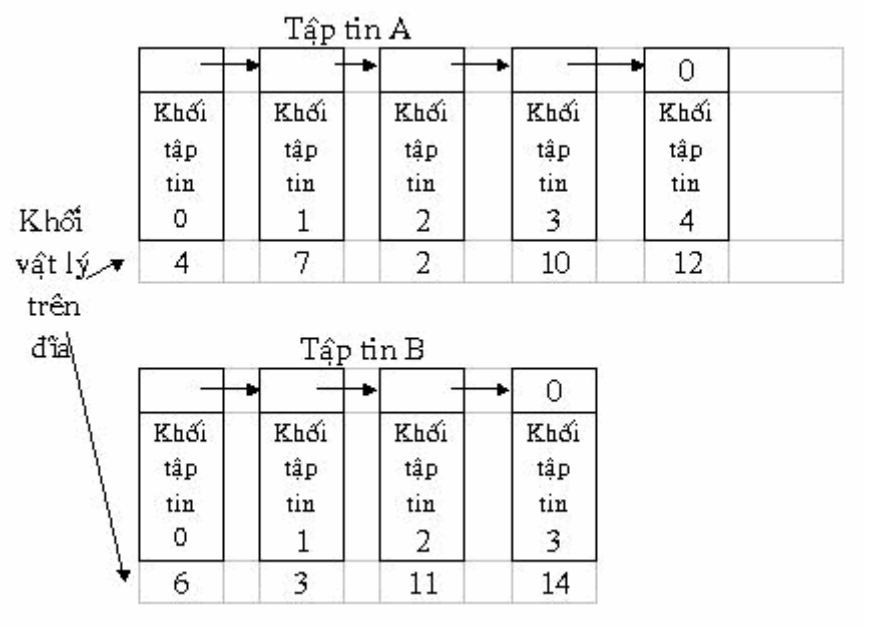
**Định vị liên tiếp :**

Lưu trữ tập tin trên dãy các khối liên tiếp.

Phương pháp này có 2 ưu điểm : thứ nhất, dể dàng cài đặt. Thứ hai, dể 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 :**



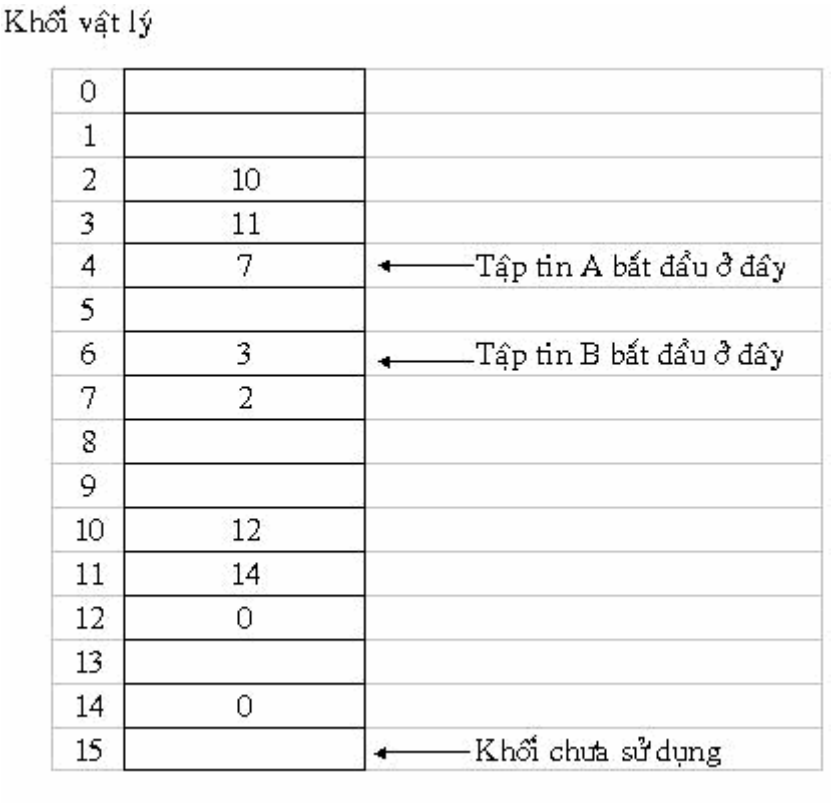

Hình 6.3 Định vị bằng danh sách liên kết

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 .

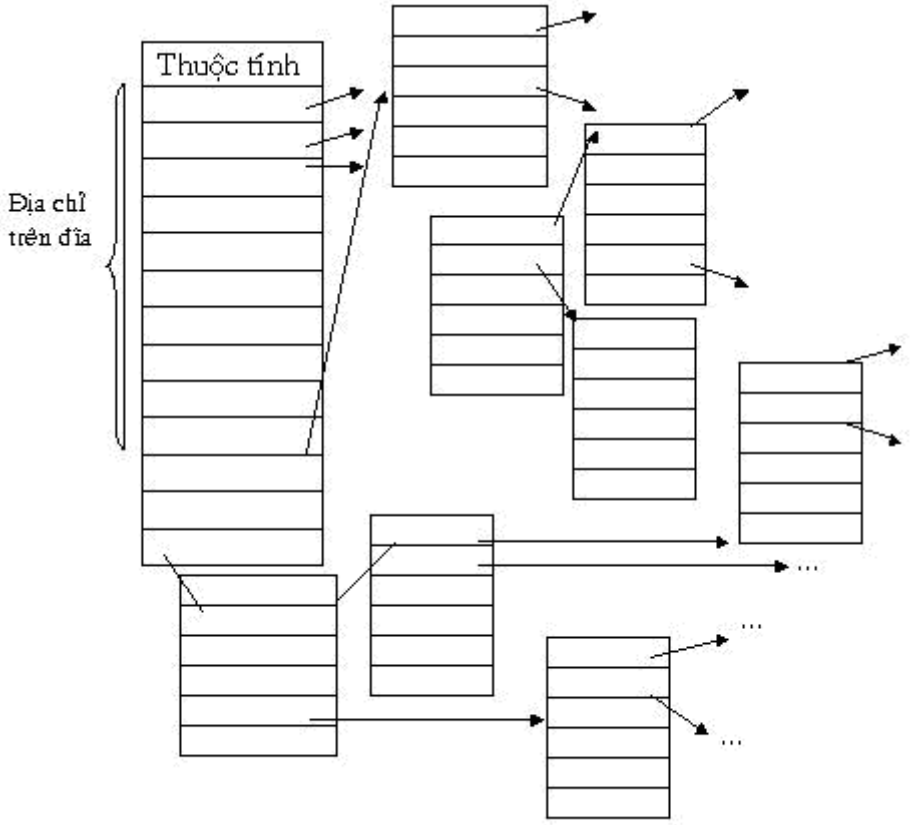



Hình 6.4 Bảng chỉ mục của danh sách liên kết

**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 Các yêu cầu quản lý file.

- Người sử dụng có thể dễ dàng tạo, sửa, thêm, bớt, xoá file.

- 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.

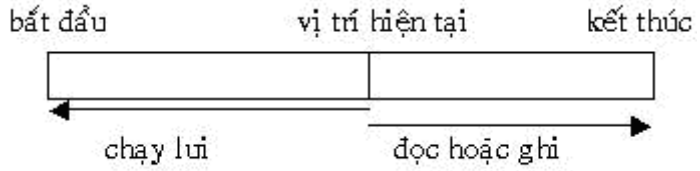
Cấu trúc cây : gồm cây của những record, không cần thiết có cùng độ dài, mỗi record có một trường khóa giúp cho việc tìm kiếm nhanh hơn.

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.

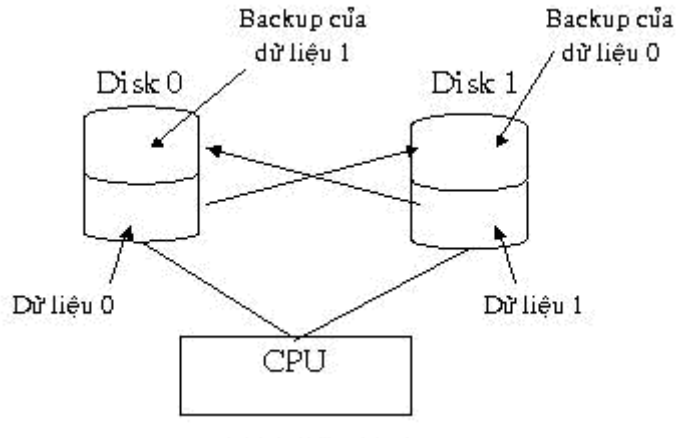
Phần mềm là hệ thống tập tin xây dựng một tập tin chứa các khối hỏng. Kỹ thuật này loại trừ chúng ra khỏi danh sách các khối trống, do đó nó sẽ không được cấp phát cho tập tin.

**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.

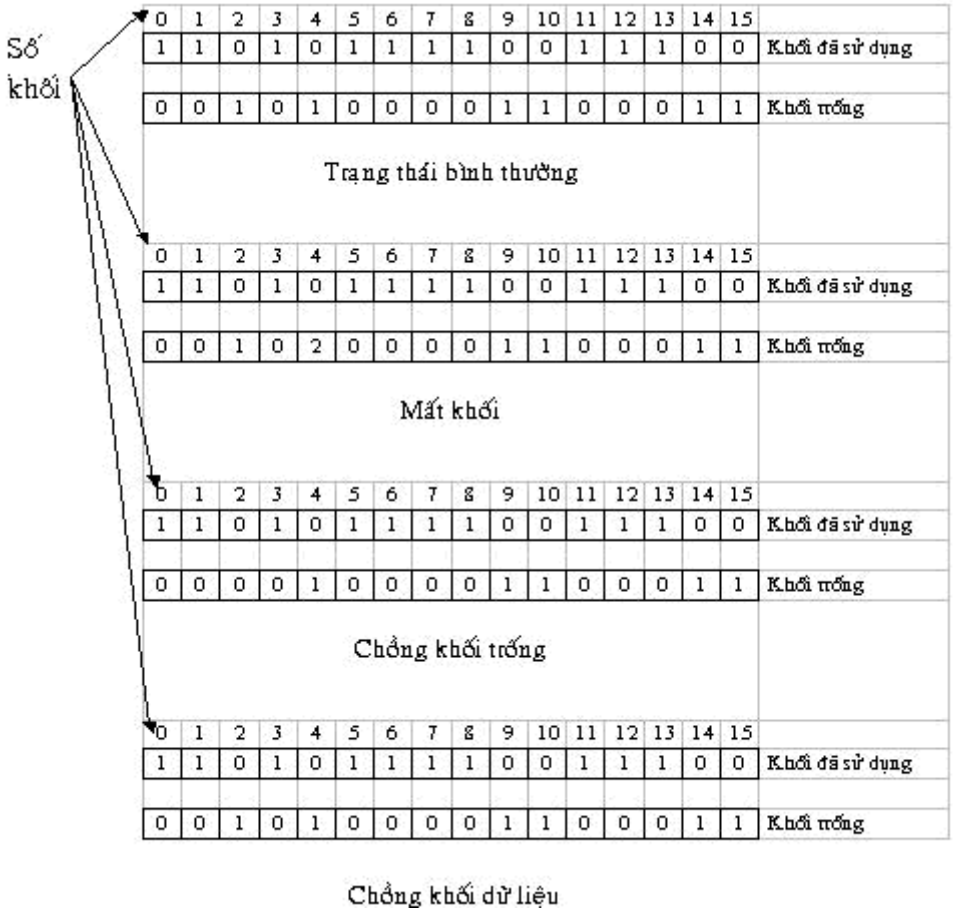



Hình 6.7 Backup

**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*".
[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.