TRƯỜNG ĐẠI HỌC NAM CẦN THƠ

KHOA CÔNG KỸ THUẬT CÔNG NGHỆ

TÀI LIỆU GIẢNG DẠY

HỆ ĐIỀU HÀNH

(OPERATING SYSTEM)

Lưu hành nội bộ, 2019

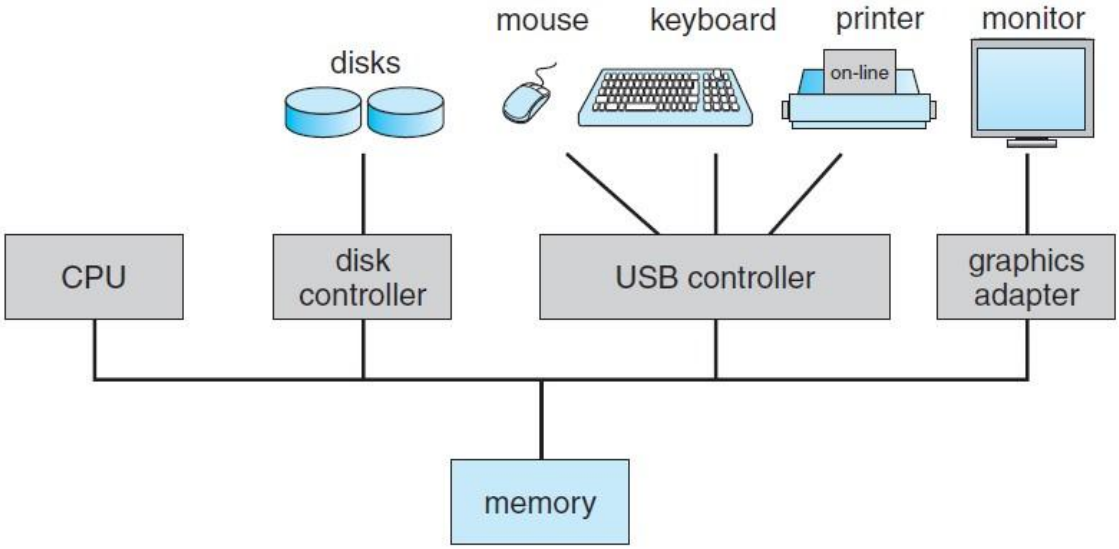
MỤC LỤC

CHƯƠNG 1. KHÁI NIỆM CƠ BẢN HỆ THỐNG MÁY TÍNH VÀ HỆ ĐIỀU HÀNH

1.1 KHÁI NIỆM VỀ HỆ ĐIỀU HÀNH

Một hệ thống máy tính bao gồm:

- Phần cứng (hardware): CPU, bộ nhớ, các thiết bị nhập xuất, đây là những tài nguyên của máy tính.
- Phần mềm (software): những chương trình sử dụng tài nguyên của máy tính để giải quyết các yêu cầu của người sử dụng (chương trình dịch, hệ thống cơ sở dữ liệu, xử lý văn bản, trò chơi, …).



Hình 1.1. Sơ đồ tổ chức phần cứng đơn giản của một hệ thống máy tính

Đối với các chương trình này có hai cách để truy xuất tới phần cứng:

- Truy xuất trực tiếp: đòi hỏi người viết chương trình phải có những kiến thức về phần cứng. Chương trình viết theo cách này sẽ phụ thuộc vào từng phần cứng cụ thể và rất khó viết, không linh động.
- Truy xuất gián tiếp thông qua những chương trình hệ thống. Người lập trình sử dụng phần cứng thông qua những tên gọi, dịch vụ do các chương trình hệ thống cung cấp, việc truy xuất cụ thể phần cứng do các chương trình này đảm nhiệm. Cách này tiết kiệm thời gian, công sức, giúp cho người lập trình tập trung hơn vào việc thiết kế chương trình.

Ngoài ra người dùng cần có một chương trình quản lý tài nguyên máy tính, cung cấp một giao diện thân thiện để sử dụng các chương trình. Một lớp phần mềm được xây dựng để đáp ứng những yêu cầu trên được gọi là hệ điều hành.

1.1.1. Khái niệm

Hệ điều hành là một chương trình đóng vai trò trung gian trong việc giao tiếp giữa người sử dụng và phần cứng của máy tính.

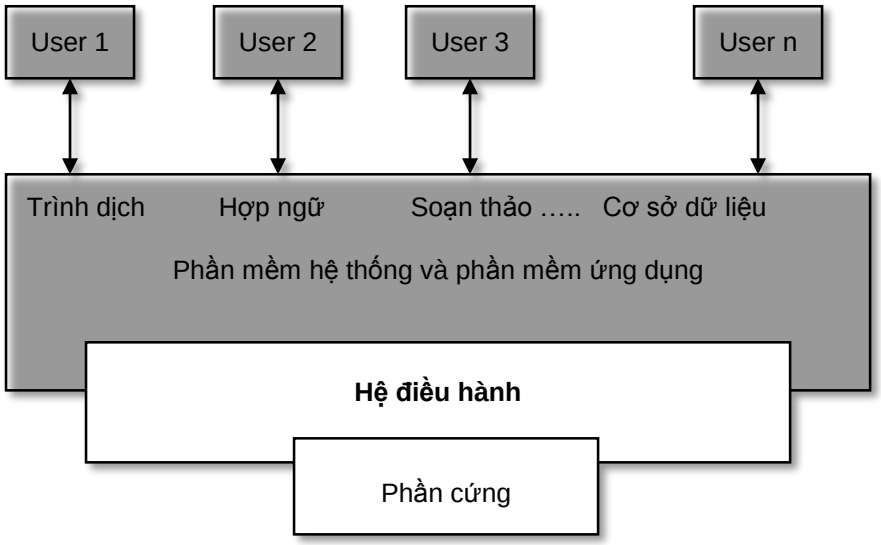
1.1.2. Mục tiêu của hệ điều hành

- Cung cấp một môi trường cho phép người sử dụng phát triển và thực hiện các ứng dụng một cách dễ dàng và hiệu quả.
- Sử dụng và quản lý tốt phần cứng. Thể hiện cho người sử dụng một máy ảo với những đặc trưng khác và dễ sử dụng hơn so với phần vật lý mà nó che dấu.

1.1.3. Chức năng của hệ điều hành

Theo nguyên tắc một hệ điều hành phải thỏa mãn hai chức năng chính yếu sau đây:

- Quản lý, chia sẻ tài nguyên: hệ điều hành cần có các cơ chế và chiến lược thích hợp để quản lý việc phân phối tài nguyên nhằm nâng cao hiệu quả sử dụng, đồng thời cần đảm bảo việc truy xuất đến các tài nguyên này là hợp lệ, không xảy ra sai trái mất đồng nhất dữ liệu.
- Giả lập một máy tính mở rộng: che dấu các chi tiết phần cứng của máy tính và giới thiệu người dùng một máy tính mở rộng có đầy đủ các chức năng của máy tính thực nhưng đơn giản và dễ sử dụng hơn.



Hình 1.2. Mô hình trừu tượng của một hệ thống máy tính

2.1 PHÂN LOẠI HỆ ĐIỀU HÀNH

2.1.1. Hệ thống xử lý theo lô đơn giản (Simple Batch System)

Mỗi thời điểm chỉ có một công việc trong bộ nhớ, khi thực hiện xong một công việc, hệ thống sẽ tự động nạp công việc khác vào và thực thi.

a) Bộ giám sát thường trực (Resident monitor)

Thực hiện tuần tự các công việc theo những chỉ thị định trước. Khi một công việc chấm dứt, hệ thống sẽ thực hiện công việc kế tiếp mà không cần có sự can thiệp của người lập trình, do đó thời gian thực hiện sẽ mau hơn. Một chương trình gọi là bộ 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 thường trú trong bộ nhớ chính.

b) CPU và thao tác nhập xuất

CPU thường hay nhàn rỗi do tốc độ làm việc của các thiết bị nhập xuất (thường là thiết bị cơ) chậm hơn nhiều lần so với các thiết bị điện tử. Cho dù là một CPU chậm nhất, nó cũng nhanh hơn rất nhiều lần so với thiết bị nhập xuất. Do đó phải có các phương pháp để đồng bộ hóa việc hoạt động của CPU và thao tác nhập xuất.

- Xử lý offline: CPU sẽ không truy xuất trực tiếp thiết bị nhập xuất mà dùng bộ lưu trữ trung gian. CPU chỉ thao tác với bộ phận này. Việc đọc hay xuất đều thực hiện trên bộ lưu trữ trung gian.
- Spooling: spool (Simultaneous Peripheral Operations on-line) cơ chế đồng bộ hóa thao tác trên thiết bị ngoại vi online, cho phép CPU lựa chọn công việc kế tiếp thực hiện trên thiết bị trong khi công việc trước đang được thực hiện (sử dụng hàng đợi – queue). Thường được sử dụng cho máy in.

2.1.2. Hệ thống xử lý theo lô đa chương (Multiprogrammed Batch System)

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

Ý tưởng như sau: hệ điều hành lưu trữ một phần của các công việc trong bộ nhớ, CPU sẽ lần lượt thực hiện các 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, …

Ví dụ: Trong bộ nhớ có ba chương trình thực hiện ba công việc. Nếu công việc 1 yêu cầu nhập/xuất thì công việc 1 tạm ngừng, công việc 2 (hoặc 3) sẽ được thực hiện, nếu công việc 2 có nhập/xuất thì sẽ tạm dừng để thực hiện công việc 3, … công việc bị tạm dừng sẽ được chọn để tiếp tục thực hiện khi nào thao tác nhập/xuất đã hoàn tất.

Các chức năng của hệ điều hành trong hệ thống xử lý theo lô đa chương rất phức tạp, bao gồm:

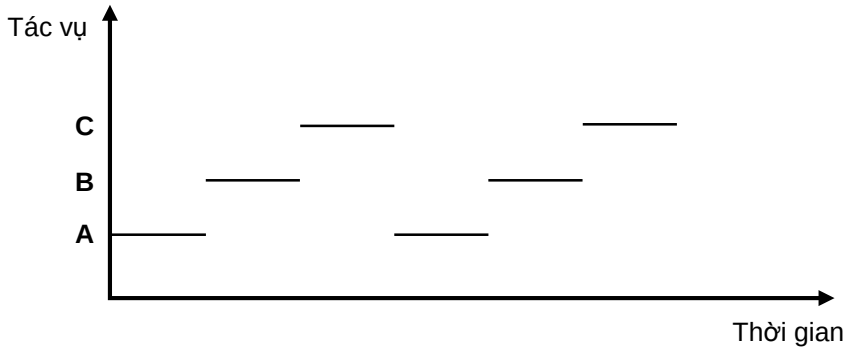
- Lập lịch CPU (CPU Scheduling): chọn một trong những công việc trong bộ nhớ cho thực thi (sử dụng CPU). Khi chọn cần tránh trường hợp một công việc chờ trong bộ nhớ quá lâu.
- Quản lý bộ nhớ (Memory Management): cần phải quản lý phần bộ nhớ nào đã cấp phát và cấp cho công việc nào (bộ nhớ cấp phát cho mỗi công việc phải riêng biệt), phần bộ nhớ nào chưa cấp, khi một công việc thực thi xong cần thu hồi phần nhớ đã cấp cho công việc đó. Nếu một công việc truy xuất đến phần bộ nhớ đã cấp cho công việc khác thì phải ngăn cấm. Nếu bộ nhớ bị phân mảnh quá nhiều thì cần dồn bộ nhớ, …
- Cấp phát thiết bị (Allocation of Devices): tình trạng thiết bị rảnh hay không rảnh, thiết bị đã cấp cho công việc nào, công việc nào cần đưa vào

hàng đợi để chờ. Thiết bị nào có thể dùng chung và tối đa bao nhiêu công việc sử dụng chung thiết bị cùng lúc, thiết bị nào không thể dùng chung, … và phải tránh bị tắc nghẽn (các công việc chờ vô hạn để được cấp tài nguyên).

- Cung cấp các hàm xử lý nhập/xuất (I/O Routines): các hàm nhập/xuất sẽ che dấu sự phức tạp và đa dạng của các thiết bị nhập/xuất, quản lý việc sử dụng chung thiết bị nhập/xuất.

2.1.3. Hệ thống chia sẻ thời gian (Time Sharing System)

Còn được gọi là hệ thống đa nhiệm (multitasking), là một mở rộng của hệ thống xử lý đa chương. Trong hệ thống đa nhiệm, việc chuyển đổi công việc không chỉ chờ công việc đang thực thi có yêu cầu nhập/xuất, mà khi công việc đang thực thi hết thời gian quy định sử dụng CPU thì việc chuyển đổi công việc cũng sẽ xảy ra. Mỗi công việc được thực hiện luân phiên qua cơ chế chuyển đổi CPU.



Hình 1.3. CPU phục vụ các công việc trong một hệ thống chia sẻ thời gian

Hệ điều hành chia sẻ thời gian cho phép nhiều người sử dụng chia sẻ máy tính một cách đồng bộ do thời gian chuyển đổi rất nhanh nên họ có cảm giác là các tiến trình đang được thi hành cùng lúc.

Hệ điều hành chia sẻ 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, cung cấp hệ thống tập tin truy xuất online, …

Hệ điều hành chia sẻ thời gian là kiểu của các hệ điều hành hiện đại ngày nay.

2.1.4. Hệ thống song song (Parallel System)

Hệ thống song song là hệ thống có nhiều bộ xử lý cùng chia sẻ 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.

Ưu điểm của hệ thống đa xử lý:

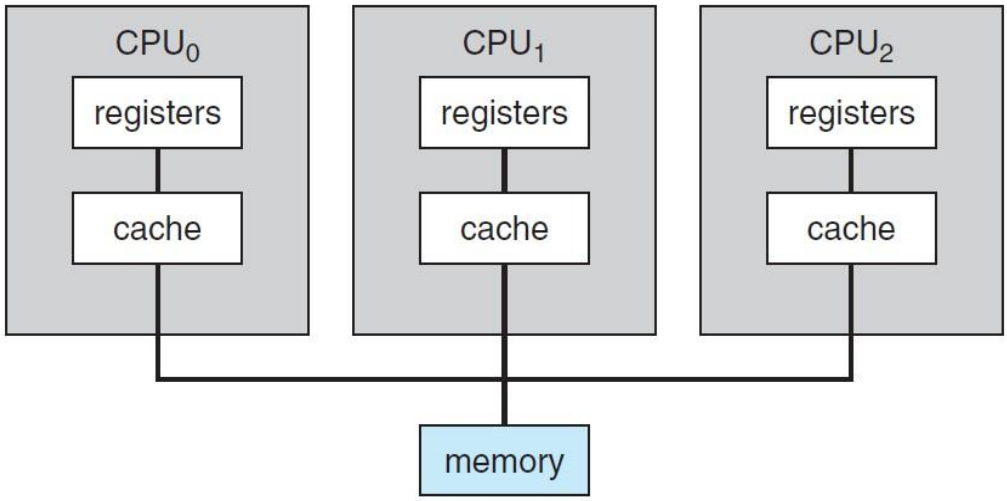
- Tin cậy hơn hệ thống đơn xử lý do sự hỏng hóc của một bộ xử lý sẽ không ảnh hưởng đến toàn hệ thống.

- Hệ thống sẽ thực hiện nhanh hơn do các công việc hoặc các chương trình có thể thực hiện đồng thời trên nhiều bộ xử lý khác nhau. Nhưng không có nghĩa là có n bộ xử lý thì hệ thống hoạt động nhanh hơn n lần.
- Việc liên lạc giữa các công việc dễ dàng do cùng chia sẻ bộ nhớ, thiết bị, …

Hệ thống đa xử lý thường chia thành 2 loại:

a) Đa xử lý đối xứng (Symmetric MultiProcessing)

Mỗi bộ xử lý chạy với một bản sao của hệ điều hành, những bản sao này liên lạc với nhau khi cần thiết. Các hệ điều hành máy cá nhân hiện nay thường hỗ trợ SMP.



Hình 1.4. Mô hình hệ thống song song

b) Đa xử lý bất đối xứng (Asymmetric Multiprocessing)

Mỗi bộ xử lý được giao một công việc riêng biệt, một bộ xử lý chính kiểm soát toàn bộ hệ thống. Mô hình này theo dạng quan hệ chủ tớ (Master – Slave). Bộ xử lý chính sẽ lập lịch cho các bộ xử lý còn lại.

2.1.5. Hệ thống phân tán (Distributed System)

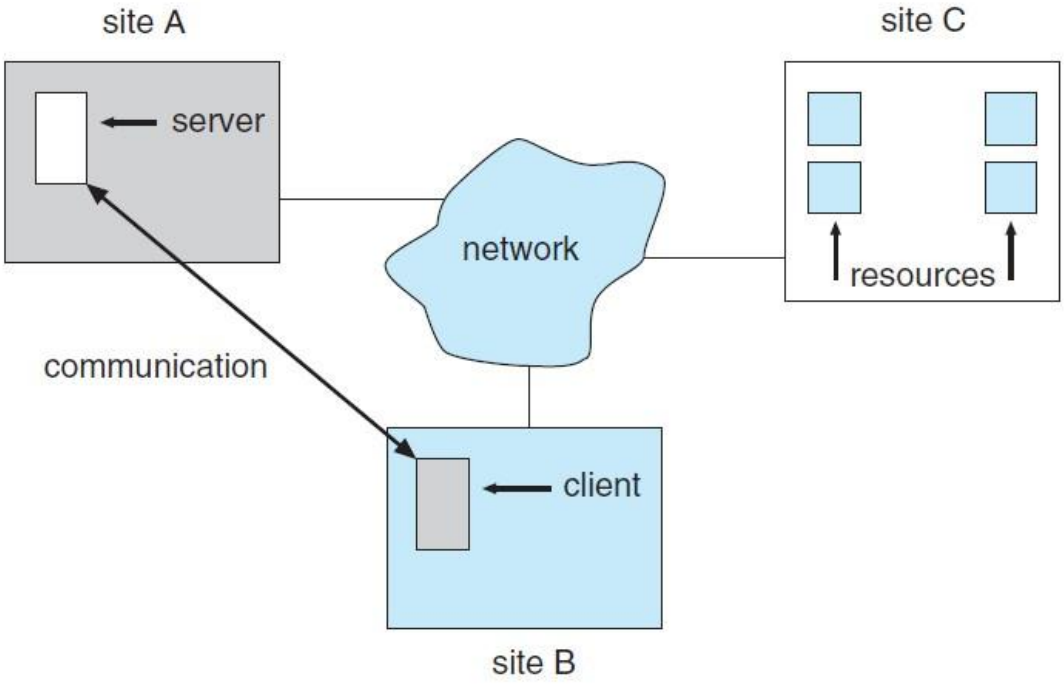
Các bộ xử lý không chia sẻ bộ nhớ và đồng hồ mà mỗi bộ xử lý sẽ có bộ nhớ cục bộ riêng. Các bộ xử lý trong hệ thống phân tán thường khác nhau về kích thước và chức năng. Chúng thực hiện trao đổi thông tin với nhau thông qua các đường truyền bus tốc độ cao hay đường dây điện thoại,…. Công việc xử lý sẻ được phân phối trên các bộ xử lý trong hệ thống phân tán.

Ưu điểm của hệ thống phân tán:

- Chia sẻ tài nguyên
- Tăng tốc độ tính toán
- An toàn
- Thông tin liên lạc

Phân loại hệ thống phân tán:

- Peer – to – peer: hệ thống mạng ngang hàng, các máy tính ngang cấp nhau, không có máy nào đóng vai trò quản lý chung.
- Client – server: có một hoặc nhiều máy đóng vai trò quản lý các tài nguyên dùng chung gọi là máy chủ (server), các máy khác gọi là máy khách (client). Client muốn sử dụng tài nguyên dùng chung phải được server cấp quyền.



Hình 1.5. Mô hình hệ thống phân tán

2.1.6. Hệ thống xử lý thời gian thực (Real Time System)

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, thường được dùng điều khiển thiết bị trong các ứng dụng tận hiến (Dedicated) như: tính toán khoa học, hệ thống hình ảnh y khoa, hệ thống điều khiển công nghiệp,…

Hiện có hai loại hệ thống xử lý thời gian thực là:

a) Hệ thống thời gian thực cứng (Hard Real Time System)

Công việc phải được hoàn tất đúng lúc, dữ liệu thường được lưu trong bộ nhớ ngắn hạn hay ROM.

b) Hệ thống thời gian thực mềm (Soft Real Time System)

Mỗi công việc có độ ưu tiên riêng và sẽ được thi hành theo độ ưu tiên đó. Được sử dụng trong các ứng dụng multimedia, thực tại ảo.

2.1.7. Hệ thống nhúng (Embedded Systems)

Hệ điều hành được nhúng trong các thiết bị: điện gia dụng, máy trò chơi, điện thoại, xe máy, …

Đặc trưng của hệ thống nhúng là tài nguyên hạn chế (bộ nhớ, tốc độ xử lý, kích thước màn hình, …) do đó hệ điều hành loại này cần đơn giản, nhỏ gọn, có tính đặc trưng cho từng thiết bị.

TÓM TẮT CHƯƠNG

- Hệ điều hành là chương trình trung gian giữa phần cứng và người sử dụng.
- Hệ điều hành phải quản lý phần cứng, sau đó cung cấp lại những giao tiếp ảo cho phần mềm ứng dụng
- Mục tiêu của hệ điều hành là phải làm cho máy tính dễ sử dụng hơn.
- Có nhiều loại hệ điều hành
  - o Hệ thống xử lý theo lô đơn giản
  - o Hệ thống xử lý theo lô đa chương
  - o Hệ thống chia sẻ thời gian
  - o Hệ thống song song
  - o Hệ thống phân tán
  - o Hệ thống thời gian thực
  - o Hệ thống nhúng
- Xu hướng hiện tại là các hệ thống song song, phân tán

BÀI TẬP

1) Giải thích sự cần thiết phải có của hệ điều hành trên máy tính.
2) Trình bày mục đích chung của các hệ điều hành.
3) Phân biệt giữa hệ thống xử lý theo lô đa chương và hệ thống chia sẻ thời gian.
4) Phân biệt giữa hệ thống song song và hệ thống phân tán.
5) Mô tả những ứng dụng thực tế của hệ thống thời gian thực.
6) Mô tả những ứng dụng thực tế của hệ thống nhúng.

CHƯƠNG 2. GIAO DIỆN LẬP TRÌNH VÀ XÂY DỰNG HỆ ĐIỀU HÀNH

2.1. CÁC THÀNH PHẦN CỦA HỆ THỐNG

2.1.1. Quản lý tiến trình

Tiến trình là một chương trình đang được thi hành. Hệ điều hành phải tạo lập và duy trì hoạt động của các tiến trình. Để hoàn tất tác vụ, một tiến trình thường đòi hỏi một số tài nguyên nào đó như CPU, bộ nhớ, thiết bị nhập/xuất,… Các tài nguyên này sẽ được cấp phát cho tiến trình vào thời điểm tiến trình được tạo lập hay trong thời gian tiến trình hoạt động. Khi tiến trình kết thúc hệ điều hành cần thu hồi lại các tài nguyên đã cấp phát cho tiến trình để tái sử dụng… Hơn nữa, mỗi tiến trình là một đơn vị tiêu thụ thời gian sử dụng CPU, do vậy trong môi trường đa nhiệm, để đáp ứng nhu cầu xử lý đồng hành, hệ điều hành còn phải đảm nhiệm việc phân phối CPU cho các tiến trình một cách hợp lý. Ngoài ra hệ điều hành cũng cần cung cấp các cơ chế giúp các tiến trình có thể trao đổi thông tin và đồng bộ hóa hoạt động của chúng.

Tóm lại, bộ phận quản lý tiến trình phụ trách các công việc sau đây:

- Tạo lập, hủy bỏ một tiến trình.
- Tạm dừng (suspend), tái kích hoạt (resume) một tiến trình.
- Cung cấp các cơ chế trao đổi thông tin giữa các tiến trình. Cung cấp cơ chế đồng bộ hóa các tiến trình

2.1.2. Quản lý bộ nhớ chính

Bộ nhớ chính là thiết bị lưu trữ duy nhất mà CPU có thể truy xuất trực tiếp. Một chương trình cần được nạp vào bộ nhớ chính và chuyển đổi các địa chỉ thành địa chỉ tuyệt đối để CPU truy xuất trong quá trình xử lý. Để tăng hiệu suất sử dụng CPU, các hệ thống đa nhiệm cố gắng giữ nhiều tiến trình trong bộ nhớ chính tại một thời điểm.

Bộ phận quản lý bộ nhớ cần đảm nhiệm các công việc sau:

- Cấp phát và thu hồi một vùng nhớ cho tiến trình khi cần thiết.
- Ghi nhận tình trạng bộ nhớ chính: phần nào đã được cấp phát, phần nào còn có thể sử dụng,…
- Quyết định tiến trình nào được nạp vào bộ nhớ chính khi có một vùng nhớ trống.

2.1.3. Quản lý bộ nhớ phụ

Những chương trình cùng với dữ liệu của chúng phải được đặt trong bộ nhớ chính trong quá trình thi hành. Nhưng bộ nhớ chính quá nhỏ để có thể lưu giữ mọi dữ liệu và chương trình, ngoài ra thông tin còn bị mất khi không còn được cung cấp năng lượng. Hệ thống máy tính ngày nay sử dụng hệ thống

lưu trữ phụ (thường là đĩa). Hầu như tất cả các chương trình đều được lưu trữ trên đĩa cho đến 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. Vì vậy một hệ thống đĩa rất quan trọng cho hệ thống máy tính, tốc độ của hệ thống máy tính phụ thuộc rất nhiều vào tốc độ truy xuất đĩa.

Vai trò của hệ điều hành trong quản lý đĩa:

- Quản lý vùng trống trên đĩa.
- Định vị lưu trữ.
- Lập lịch cho đĩa

2.1.4. Quản lý nhập xuất

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 dễ thao tác hơn.

Một hệ thống nhập xuất bao gồm:

- Hệ thống buffer caching.
- Giao tiếp điều khiển thiết bị (device driver) tổng quát.
- Bộ điều khiển cho các thiết bị phần cứng cụ thể.

Chỉ có device driver mới hiểu đến cấu trúc đặc thù của thiết bị mà nó mô tả.

2.1.5. Quản lý tập tin

Máy tính có thể lưu trữ thông tin trên nhiều loại thiết bị lưu trữ vật lý khác nhau, mỗi thiết bị này có những tính chất và tổ chức vật lý đặc trưng. Nhằm cho phép sử dụng tiện lợi hệ thống thông tin, hệ điều hành đưa ra một khái niệm trừu tượng đồng nhất cho tất cả các thiết bị lưu trữ vật lý bằng cách định nghĩa một đơn vị lưu trữ là một tập tin. Hệ điều hành thiết lập mối liên hệ tương ứng giữa tập tin và thiết bị lưu trữ vật lý chứa nó. Để có thể dễ dàng truy xuất, hệ điều hành còn tổ chức tập tin thành các thư mục. Ngoài ra, hệ điều hành còn trách nhiệm kiểm soát việc truy cập đồng thời đến cùng một tập tin.

Như vậy, hệ điều hành chịu trách nhiệm về các thao tác liên quan đến tập tin như:

- Tạo lập, hủy bỏ một tập tin.
- Tạo lập, hủy bỏ một thư mục.
- Cung cấp thao tác xử lý tập tin và thư mục.
- Tạo lập quan hệ tương ứng giữa tập tin và bộ nhớ phụ chứa nó.

2.1.6. Hệ thống bảo vệ

Khi hệ thống cho phép nhiều người sử dụng đồng thời, các tiến trình đồng hành cần phải được bảo vệ lẫn nhau để tránh sự xâm phạm vô tình hay cố ý có thể gây

sai lạc cho toàn hệ thống. Hệ điều hành cần xây dựng các cơ chế bảo vệ cho phép đặc tả sự kiểm soát và một phương cách để áp dụng các chiến lược bảo vệ thích hợp.

2.1.7. Quản lý mạng

Một hệ thống phân tán bao gồm nhiều bộ xử lý cùng chia sẻ thông tin với nhau. Mỗi bộ xử lý có một bộ nhớ cục bộ, các tiến trình trong hệ thống có thể được kết nối với nhau qua mạng truyền thông. Hệ thống phân tán cho phép người dùng truy cập đến nhiều tài nguyên hệ thống khác nhau.

2.1.8. Hệ thông dịch lệnh

Đây là một trong những bộ phận quan trọng nhất của hệ điều hành, đóng vai trò giao tiếp giữa hệ điều hành và người sử dụng. Các lệnh được chuyển đến hệ điều hành dưới dạng các chỉ thị điều khiển. Chương trình shell – bộ thông dịch lệnh – có nhiệm vụ là nhận lệnh và thông dịch lệnh đó để hệ điều hành có xử lý tương ứng.

2.2. CÁC DỊCH VỤ CỦA HỆ ĐIỀU HÀNH (SYSTEM SERVICES)

Hệ điều hành cung cấp môi trường để thi hành các chương trình, bằng cách cung cấp các dịch vụ cho chương trình và người sử dụng. Các dịch vụ này trên mỗi hệ thống là khác nhau nhưng cũng có những lớp chung. Các dịch vụ này giúp lập trình viên thuận tiện hơn và việc lập trình dễ dàng hơn.

- Thi hành chương trình
  Hệ thống phải có khả năng nạp chương trình vào bộ nhớ và thi hành nó. Chương trình phải chấm dứt thi hành theo cách thông thường hay bất thường (có lỗi).

- Thao tác nhập xuất
  Một chương trình thi hành có thể yêu cầu nhập xuất. Nhập xuất này có thể là tập tin hay thiết bị. Để tăng hiệu quả, người sử dụng không truy xuất trực tiếp các thiết bị nhập xuất mà thông qua cách thức do hệ điều hành cung cấp.

- Thao tác trên hệ thống tập tin
  Chương trình có thế yêu cầu đọc, ghi, xóa, đổi tên, tìm kiếm, cấp quyền cho tập tin, thư mục. Đa số hệ điều hành cung cấp nhiều loại hệ thống file, có thể cho người dùng chọn lựa hoặc bắt buộc tùy theo yêu cầu.

- Thông tin
  Việc trao đổi thông tin có thể được thực hiện giữa các tiến trình trên cùng máy tính hoặc giữa các hệ thống khác nhau trong mạng. Được thực hiện thông qua cơ chế vùng nhớ chia sẻ hoặc chuyển thông điệp.

- Phát hiện lỗi

Hệ điều hành phải đảm bảo việc tính toán chính xác thông qua việc phát hiện các lỗi do phần cứng (CPU, bộ nhớ, thiết bị nhập xuất) hoặc chương trình của người dùng.

- Giao diện người dùng
  Có nhiều dạng: dòng lệnh (CLI – Command-Line Interface), xử lý theo lô (Batch Interface), giao diện đồ họa (GUI – Graphical User Interface). Trong đó dạng GUI được sử dụng nhiều nhất, một số hệ thống còn cung cấp nhiều loại giao diện.

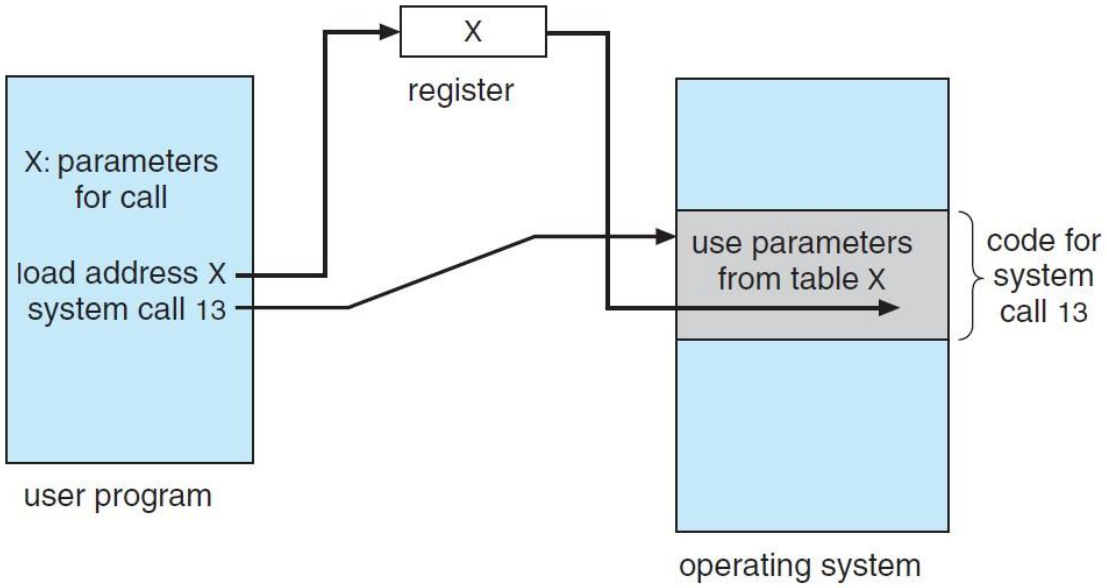
2.3. LỜI GỌI HỆ THỐNG (SYSTEM CALL)

Lời gọi hệ thống cung cấp một giao tiếp giữa tiến trình và hệ điều hành. Lời gọi này cũng như các lệnh hợp ngữ.

Một số hệ thống cho phép lời gọi hệ thống được thực hiện từ cấp lập trình ngôn ngữ cấp cao như các hàm và lời gọi hàm. Nó có thể phát sinh lời gọi từ các thủ tục hay gọi trực tiếp trong dòng (inline).

Có 3 phương pháp được sử dụng để chuyển tham số cho hệ điều hành:

- Chuyển tham số vào thanh ghi.
- Lưu giữ trong một bảng trong bộ nhớ và địa chỉ của bảng này được truyền thông qua thanh ghi.
- Dùng cơ chế Stack.



Hình 2.1. Truyền tham số dạng bảng

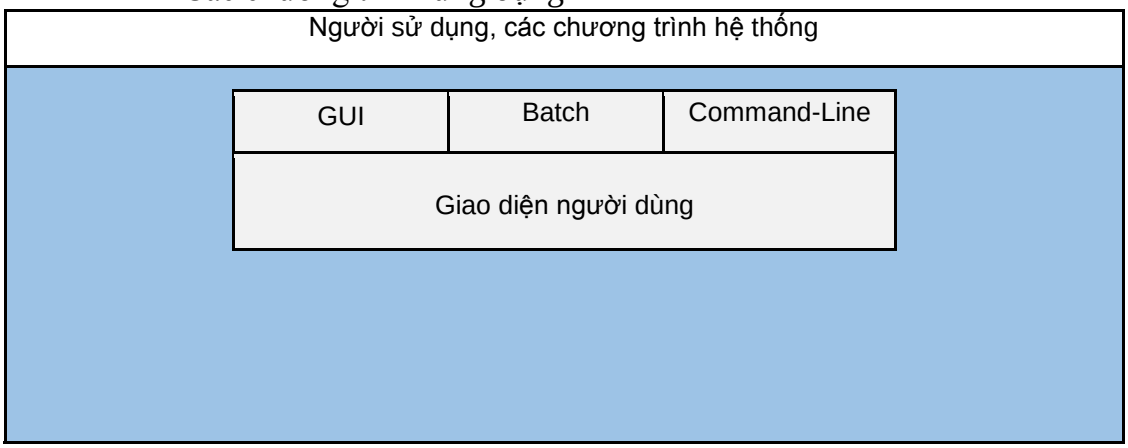
	Windows	Unix
Process Control	CreateProcess() ExitProcess() WaitForSingleObject()	fork() exit() wait()
File Manipulation	CreateFile() ReadFile() WriteFile() CloseHandle()	open() read() write() close()
Device Manipulation	SetConsoleMode() ReadConsole() WriteConsole()	ioctl() read() write()
Information Maintenance	GetCurrentProcessID() SetTimer() Sleep()	getpid() alarm() sleep()
Communication	CreatePipe() CreateFileMapping() MapViewOfFile()	pipe() shm_open() mmap()
Protection	SetFileSecurity() InitlializeSecurityDescriptor() SetSecurityDescriptorGroup()	chmod() umask() chown()

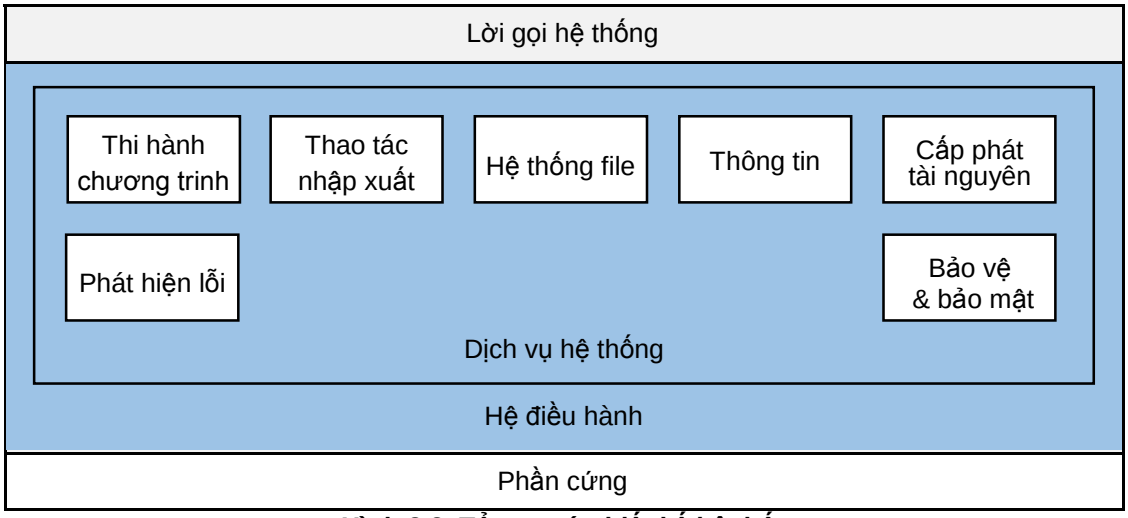
Hình 2.2. Một số lời gọi hệ thống trong Windows và UNIX

2.4. CÁC CHƯƠNG TRÌNH HỆ THỐNG

Các chương trình hệ thống cung cấp một môi trường tiện lợi hơn cho việc xây dựng và thi hành chương trình. Nó có thể chia thành một số loại như:

- Thao tác với tập tin
- Thông tin trạng thái
- Mô tả tập tin
- Hỗ trợ ngôn ngữ lập trình
- Nạp và thi hành chương trình
- Thông tin
- Các chương trình ứng dụng





Hình 2.3. Tổng quát thiết kế hệ thống

TÓM TẮT CHƯƠNG

- Hệ điều hành được thiết kế bao gồm nhiều thành phần hệ thống phối hợp với nhau để hoạt động.
- Có nhiều loại dịch vụ được cung cấp cho người dùng, phần mềm ứng dụng.
- Các dịch vụ được khai thác thông qua những lời gọi hệ thống.

BÀI TẬP

1) Nêu mục đích của bộ nhớ đệm (buffer) trong hệ thống nhập xuất.
2) Trình điều khiển thiết bị (device driver) do hãng nào cung cấp?

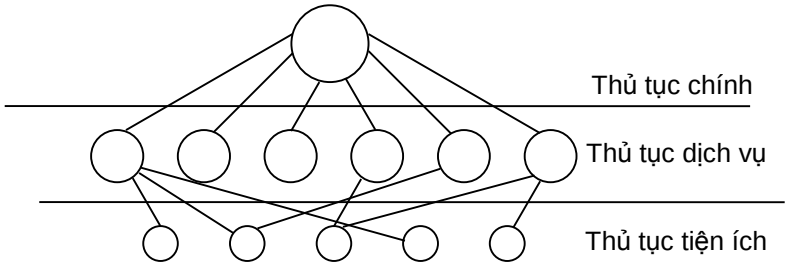
CHƯƠNG 3. CẤU TRÚC HỆ ĐIỀU HÀNH

3.1. CẤU TRÚC HỆ THỐNG

3.1.1. Cấu trúc đơn giản (Monolithique)

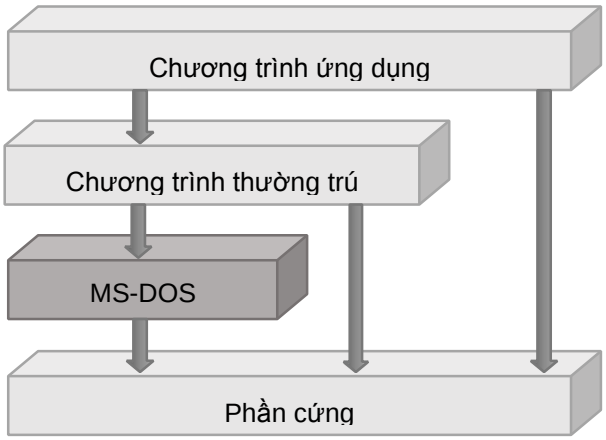
Hệ điều hành là một tập hợp các thủ tục, có thể gọi lẫn nhau. Một số hệ monolithique cũng có thể có một cấu trúc tối thiểu khi phân chia các thủ tục trong hệ thống thành 3 cấp độ:

- Chương trình chính (chương trình của người sử dụng) gọi đến một thủ tục của hệ điều hành, được gọi là lời gọi hệ thống.
- Tập hợp các thủ tục dịch vụ (service) xử lý những lời gọi hệ thống.
- Tập hợp các thủ tục tiện ích (utility) hỗ trợ các thủ tục dịch vụ xử lý những lời gọi hệ thống.



Hình 3.1.Cấu trúc một hệ monolithique

Ví dụ



Hình 3.2. Cấu trúc của hệ điều hành MS-DOS

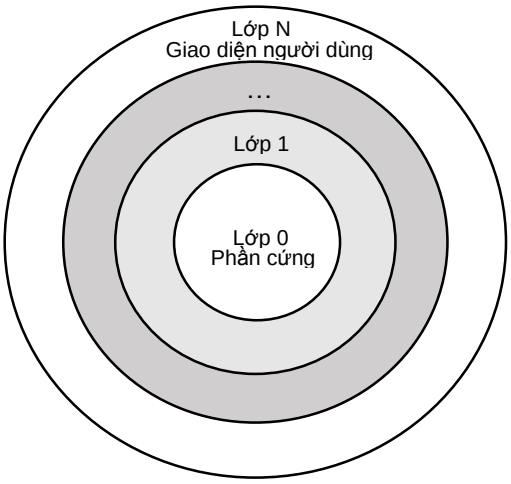
Khuyết điểm

- Không có sự che dấu dữ liệu, mỗi thủ tục có thể gọi đến tất cả các thủ tục khác. Các mức độ phân chia thủ tục nếu có cũng không rõ rệt, chương trình ứng dụng có thể truy xuất đến các thủ tục cấp thấp, tác động đến cả phần cứng, do vậy hệ điều hành khó kiểm soát và bảo vệ hệ thống.
- Hệ thống thủ tục chỉ mang tính chất tĩnh, chỉ được kích hoạt khi cần thiết, do vậy hệ điều hành thiếu chủ động trong việc quản lý môi trường.

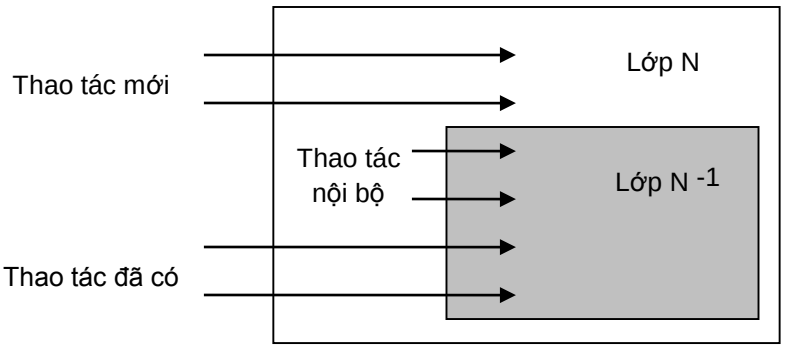
3.1.2. Cấu trúc phân lớp (Layered)

Đơn thể hóa hệ thống bằng cách cắt hệ thống thành một số lớp, mỗi lớp được xây dựng dựa trên các lớp bên trong. Lớp trong cùng (lớp 0) thường là phần cứng, lớp ngoài cùng (lớp N) thường là giao diện với người sử dụng.

Mỗi lớp là một đối tượng trừu tượng bao bọc bên trong nó các dữ liệu và thao tác xử lý các dữ liệu đó. Lớp N chứa đựng một số cấu trúc dữ liệu và một số thủ tục có thể được gọi bởi những lớp bên ngoài và lớp N có thể gọi những thủ tục của các lớp bên trong N.



Hình 3.3. Mô hình cấu trúc



Hình 3.4.Một lớp của hệ điều hành phân lớp

Ví dụ

Cấu trúc của hệ điều hành THE (Technische Hogeschool Eidhoven)

- Lớp 5: chương trình ứng dụng
- Lớp 4: quản lý bộ đệm cho thiết bị nhập xuất
- Lớp 3: trình điều khiển thao tác console
- Lớp 2: quản lý bộ nhớ
- Lớp 1: điều phối CPU
- Lớp 0: phần cứng

Ưu điểm

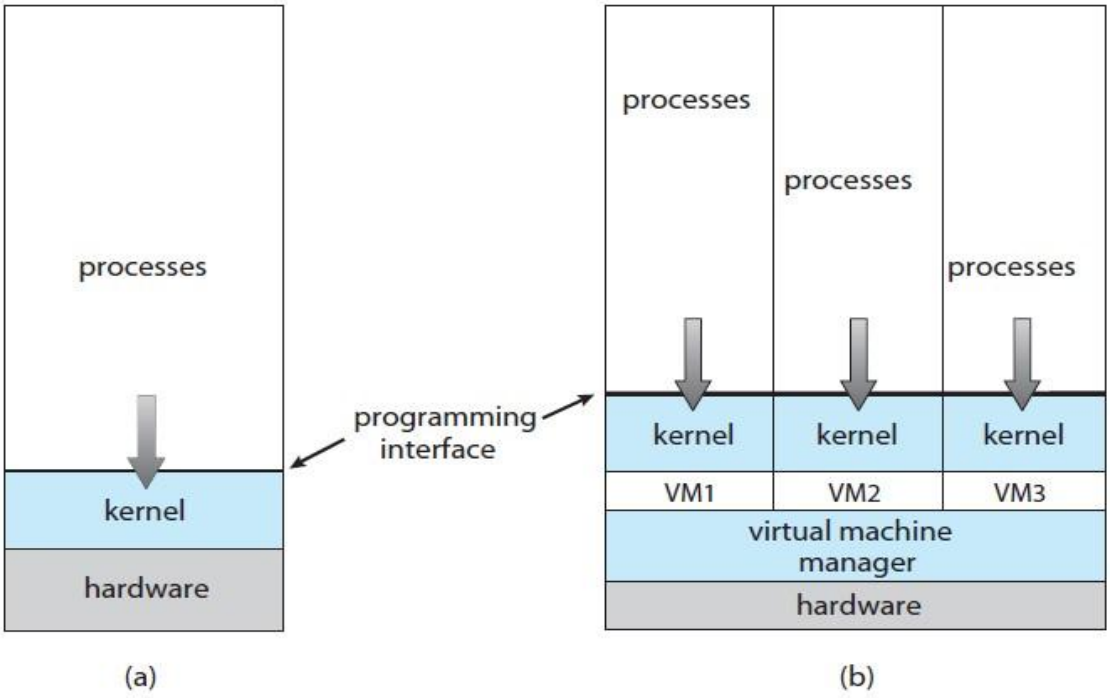
Cho phép xây dựng hệ thống mang tính đơn thể, điều này giúp đơn giản việc tìm lỗi và kiểm chứng hệ thống, việc thiết kế và cài đặt đơn giản.

Khuyết điểm

- Khó có thể xác định các lớp một cách đầy đủ: bao nhiêu lớp? Mỗi lớp đảm nhiệm những chức năng gì? Do mỗi lớp chỉ có thể sử dụng các lớp bên trong nên thứ tự xếp đặt các lớp cần phải được cân nhắc cẩn thận.
- Khi cài đặt thực tế, các hệ thống theo cấu trúc phân lớp có khuynh hướng hoạt động kém hiệu quả do một lời gọi thủ tục có thể kích hoạt lan truyền các thủ tục khác ở các lớp bên trong, vì thế tổng chi phí để truyền tham số, chuyển đổi ngữ cảnh,… tăng lên. Hậu quả là lời gọi hệ thống trong hệ thống phân lớp được thực hiện chậm hơn trong các hệ thống không phân lớp.

3.1.3. Máy ảo (Virtual Machine)

Phần nhân của hệ thống là trình tổ chức giám sát máy ảo (monitor of virtual machine) 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. Các máy ảo không phải là các máy tính mở rộng mà là bản sao (ảo) chính xác các đặc tính phần cứng của máy tính thật sự và cho phép một hệ điều hành khác hoạt động trên nó như trên phần cứng thực sự, các hệ điều hành này mới là thành phần chịu trách nhiệm cung cấp cho người sử dụng một máy tính mở rộng. Hơn nữa, mỗi máy ảo có thể cho phép một hệ điều hành khác nhau hoạt động trên nó, như vậy có thể tạo ra nhiều môi trường đồng thời trên một máy tính thật sự.



Hình 3.5. Mô hình hệ thống máy ảo

Ưu điểm

- Trong môi trường này, các tài nguyên hệ thống được bảo vệ hoàn toàn vì mỗi máy ảo là độc lập với các máy ảo khác.
- Sự phân tách hoàn toàn sự đa chương và máy tính mở rộng dẫn đến một hệ thống mềm dẻo linh động hơn và dễ bảo trì hơn.
- Việc tạo lập các máy ảo đã cung cấp một phương tiện để giải quyết vấn đề tương thích: một chương trình được phát triển trên hệ điều hành X có thể hoạt động trong môi trường máy ảo X trên một máy tính thật sự sử dụng hệ điều hành Y.

Khuyết điểm

Việc cài đặt các phần mềm giả lập phần cứng thường rất khó khăn.

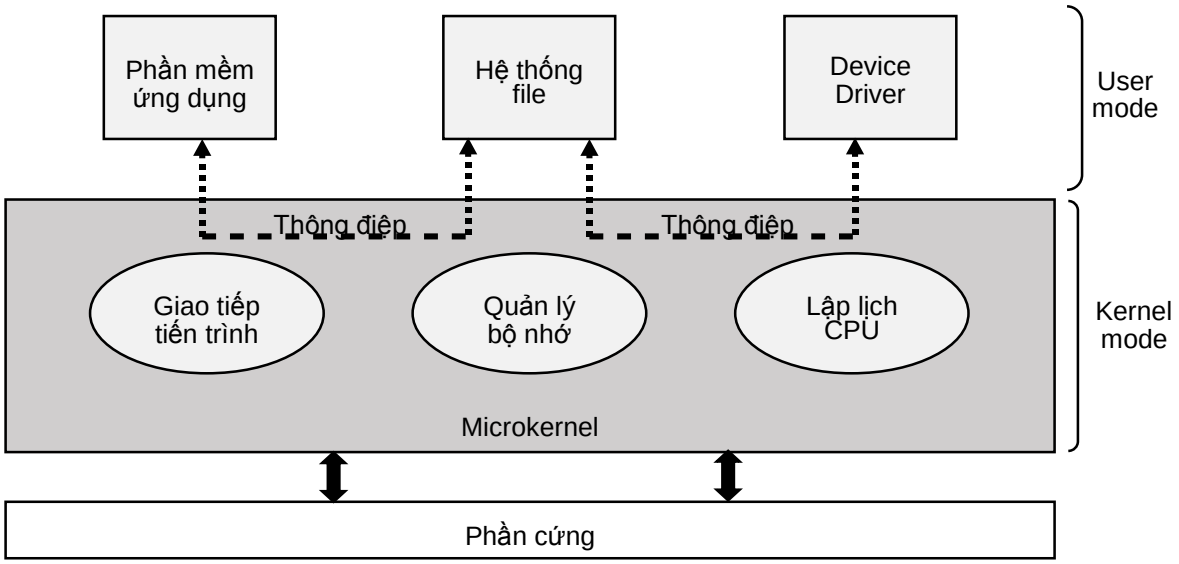
3.1.4. Mô hình Client – Server (Microkernel)

Xu hướng chung của các hệ điều hành hiện đại là chuyển dần các tác vụ của hệ điều hành ra những lớp bên ngoài, thu nhỏ phần cốt lõi của hệ điều hành thành hạt nhân cực tiểu (microkernel) sao cho chỉ phần hạt nhân này là phụ thuộc phần cứng. Một cách tiếp cận phổ biến là cấu trúc hệ thống theo mô hình Client – Server.

Trong mô hình Client-Server, các thành phần không cần thiết sẽ được đưa ra khỏi hạt nhân của hệ điều hành và chạy như một tiến trình server ở chế độ không đặc quyền (user-mode). Phần hạt nhân hệ điều hành sẽ đảm nhiệm việc tạo cơ chế thông tin liên lạc giữa các tiến trình client và tiến trình server. Khi đó các tiến trình trong hệ thống có thể chia thành 2 loại:

- Tiến trình của người dùng (client)
- Tiến trình của hệ điều hành (server)

Khi cần thực hiện một chức năng của hệ thống, tiến trình client sẽ gởi yêu cầu đến tiến trình server tương ứng, tiến trình server sẽ xử lý yêu cầu và hồi đáp cho tiến trình client. Liên lạc giữa 2 tiến trình được thực hiện thông qua hạt nhân của hệ điều hành.



Hình 3.6. Mô hình Client – Server

Ưu điểm

- Hạt nhân rất nhỏ, chỉ gồm các phần cơ bản, nên dễ bảo vệ, nâng cấp.
- Dễ mở rộng và sửa đổi hệ điều hành thông qua việc mở rộng và sửa đổi các tiến trình server.
- Các tiến trình server của hệ điều hành hoạt động trong chế độ không đặc quyền nên không thể truy cập trực tiếp tới phần cứng. Điều này giúp hệ thống được bảo vệ tốt hơn nên tính ổn định và an ninh cao hơn.
- Cấu trúc Client – Server rất phù hợp với hệ thống phân tán. Các tiến trình Server có thể chạy ở những hệ thống khác nhau.

3.2. NGUYÊN LÝ THIẾT KẾ HỆ ĐIỀU HÀNH

- Hệ điều hành cần dễ viết, dễ sửa lỗi, dễ nâng cấp (hệ điều hành nên viết bằng ngôn ngữ cấp cao, chỉ viết bằng hợp ngữ những phần thật cần thiết, vì ngôn ngữ cấp cấp cao dễ viết, dễ bảo trì hơn hợp ngữ).
- Hệ điều hành cần dễ cài đặt, dễ bảo trì, không có lỗi và hiệu quả.
- Hệ điều hành cần dễ sử dụng, dễ học, an toàn, có độ tin cậy và thực hiện nhanh.
- Hệ điều hành cần có tính khả chuyển cao (thực hiện được trên một nhóm các phần cứng khác nhau)
- Hệ điều hành cần có chương trình SYSGEN (System Generation) thu thập thông tin liên qua đến phần cứng để thiết lập cấu hình hệ điều hành phù hợp với mỗi máy tính.

TÓM TẮT CHƯƠNG

- Hệ điều hành được thiết kế bao gồm nhiều thành phần hệ thống phối hợp với nhau để hoạt động.
- Có nhiều loại dịch vụ được cung cấp cho người dùng, phần mềm ứng dụng.
- Các dịch vụ được khai thác thông qua những lời gọi hệ thống.
- Các chương trình hệ thống được cung cấp để đáp ứng những yêu cầu chung của người dùng.
- Mô hình máy ảo giúp triển khai được nhiều hệ điều hành cùng lúc trên cùng một hệ thống.
- Hầu hết các hệ điều hành hiện đại được thiết kế theo mô hình Client – Server.

BÀI TẬP

1) Mô tả những ứng dụng của máy ảo trong thực tế.
2) Tìm hiểu về hệ điều hành MS-DOS.
3) Tìm hiểu cấu trúc của hệ điều hành Windows, Linux

CHƯƠNG 4. FILE VÀ THAO TÁC FILE

4.1. CÁC KHÁI NIỆM CƠ BẢN

4.1.1. Bộ nhớ ngoài

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

- Phải chứa những lượng thông tin rất lớn (giữ vé máy bay, ngân hàng, …).
- Thông tin phải được lưu giữ trong một thời gian dài trước khi xử lý.
- Nhiều tiến trình có thể truy cập cùng một lúc.

Giải pháp là sử dụng các thiết bị lưu trữ bên ngoài gọi là bộ nhớ ngoài.

4.1.2. Tập tin và thư mục

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.

Thư mục

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 quản lý tập tin

Các tập tin được quản lý bởi hệ điều hành với cơ chế gọi là hệ thống quản lý tập tin. Bao gồm: cách hiển thị, các yếu tố cấu thành tập tin, cách đặt tên, cách truy xuất, cách sử dụng và bảo vệ tập tin, các thao tác trên tập tin. Cách tổ chức thư mục, các đặc tính và các thao tác trên thư mục.

4.2. MÔ HÌNH QUẢN LÝ VÀ TỔ CHỨC CÁC TẬP TIN

4.2.1. Mô hình

a) Tập tin

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, hệ thống tập tin có thể có hay không có phân biệt chữ thường và chữ hoa. Nhiều hệ thống tập tin hỗ trợ tên tập tin gồm phần được phân cách bởi dấu "." mà phần sau được gọi là phần mở rộng. Hệ điều hành dùng phần mở rộng để nhận dạng kiểu của tập tin

và các thao tác có thể thực hiện trên kiểu tập tin đó. Ví dụ: tập tin **readme.txt** trong Windows thì hệ điều hành sẽ hiểu là tập tin văn bản và liên kết với một chương trình xử lý văn bản khi mở tập này.

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.
- 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.

Kiểu tập tin

Các hệ điều hành hỗ trợ cho nhiều loại tập tin khác nhau bao gồm các kiểu như: tập tin thường, thư mục, tập tin có ký tự đặc biệt, tập tin khối.

- Tập tin thường: là tập tin text hay tập tin nhị phân chứa thông tin của người sử dụng.
- Thư mục: là những tập tin hệ thống dùng để lưu giữ cấu trúc của hệ thống tập tin.
- Tập tin có ký tự đặc biệt: liên quan đến nhập xuất thông qua các thiết bị nhập xuất tuần tự như màn hình, máy in, mạng, …
- Tập tin khối: dùng để truy xuất trên thiết bị đĩa.

Tập tin thường được chia thành hai loại là tập tin văn bản và tập tin nhị phân.

- Tập tin văn bản: chứa các dòng văn bản, cuối dòng có ký hiệu enter. Mỗi dòng có độ dài có thể khác nhau. Ưu điểm của kiểu tập tin này là nó có thể hiển thị, in hay soạn thảo với một editor thông thường. Đa số các chương trình dùng tập tin văn bản để nhập xuất, nó cũng dễ dàng làm đầu vào và đầu ra cho cơ chế pipeline.
- Tập tin nhị phân: có cấu trúc khác tập tin văn bản. Mặc dù về mặt kỹ thuật, tập tin nhị phân gồm dãy các byte, nhưng hệ điều hành chỉ thực thi tập tin đó nếu nó có cấu trúc đúng.

Truy xuất tập tin

Trước đây chỉ có một kiểu truy xuất tập tin: tuần tự. Tiến trình đọc tất cả các byte trong tập tin theo thứ tự từ đầu. Truy xuất kiểu này thuận lợi cho các loại băng từ. Đối với đĩa, việc truy xuất theo khóa hơn là vị trí, các byte được đọc theo bất kỳ vị trí nào

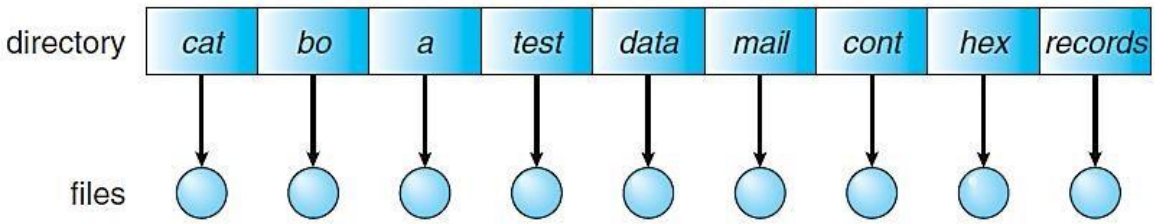
Thuộc tính tập tin

Bao gồm các thuộc tính: bảo vệ, mật khẩu, người tạo, người sở hữu, read only, hidden, system, archive, ASCII/Binary, truy xuất ngẫu nhiên, lock, …

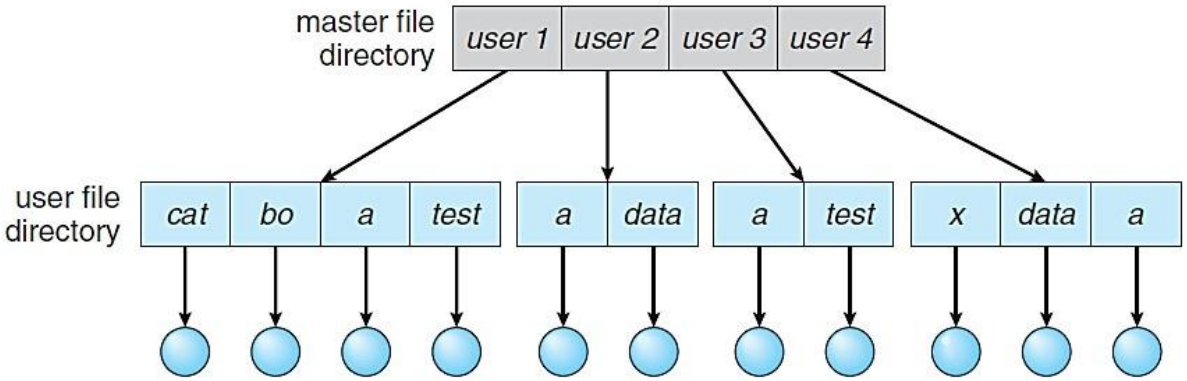
b) Thư mục

Hệ thống thư mục theo cấp bậc

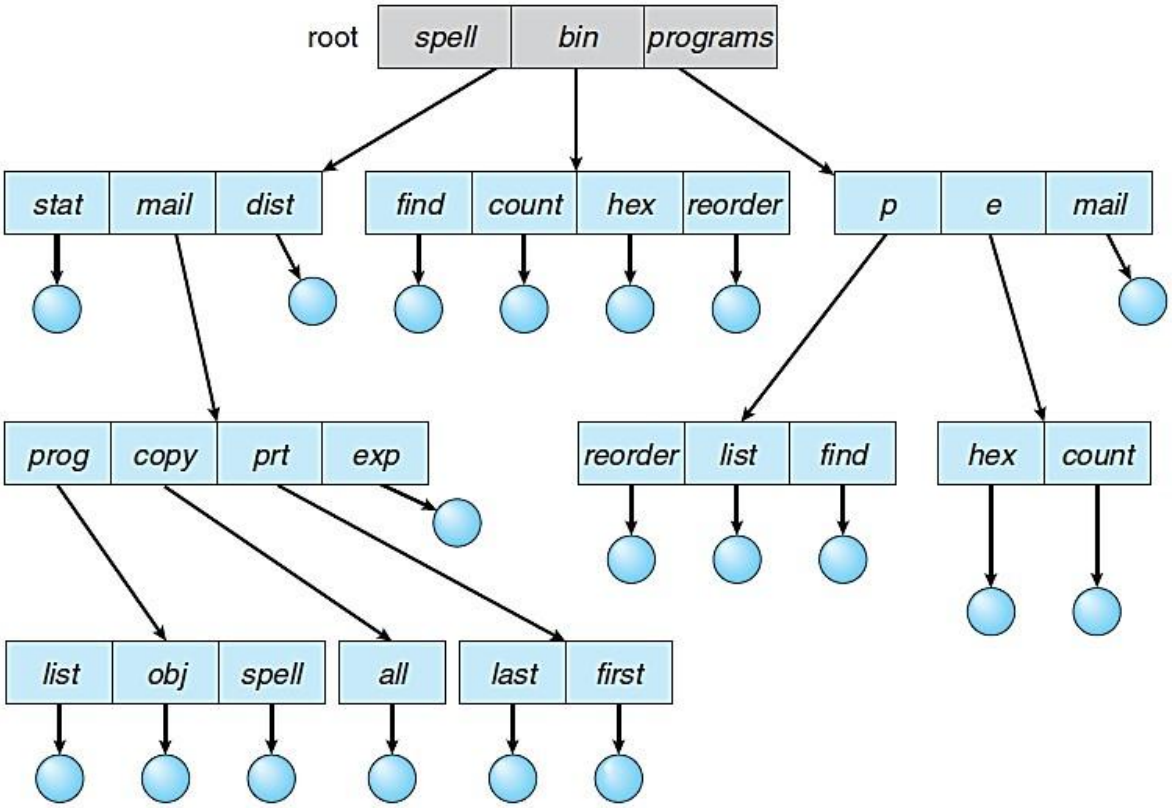
Số lượng thư mục trên mỗi hệ thống là khác nhau. Thiết kế đơn giản nhất là hệ thống chỉ có thư mục đơn chứa tất cả các tập tin của tất cả người dùng. Cách này dễ nhưng sẽ gây ra khó khăn khi có nhiều người sử dụng vì sẽ có nhiều tập tin trùng tên. Cách thứ hai là có một thư mục gốc và trong đó có nhiều thư mục con, trong mỗi thư mục con chứa tập tin của người sử dụng, 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 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.



Hình 4.1. Hệ thống thư mục đơn cấp



Hình 4.2. Hệ thống thư mục 2 cấp



Hình 4.3. 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 tên một 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 ".." . Trong đó "." chỉ thư mục hiện hành còn ".." chỉ thư mục cha.

4.2.2. Các chức năng

a) Tập tin

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 xóa 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 cảu 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 ghi nhưng dữ liệu luôn được ghi vào cuối tập tin.

Tìm: dùng để truy xuất tập tin ngẫu nhiên.

Lấy thuộc tính: lấy thuộc tính của tập tin hiện hành.

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.

b) Thư mục

Tạo: khi một **thư** mục được tạo, nó rỗng, ngoại trừ "." và ".." được đặt tự động bởi hệ thống.

Xóa: xóa một thư mục, chỉ có thư mục rỗng mới bị xóa, thư mục chứa "." và ".." coi như là thư mục rỗng.

Mở: thư mục có thể đọc được. Ví dụ để liệt kê tất cả tập tin trong một thư mục, chương trình liệt kê sẽ mở thư mục và đọc tên của tất cả các tập tin chứa trong đó.

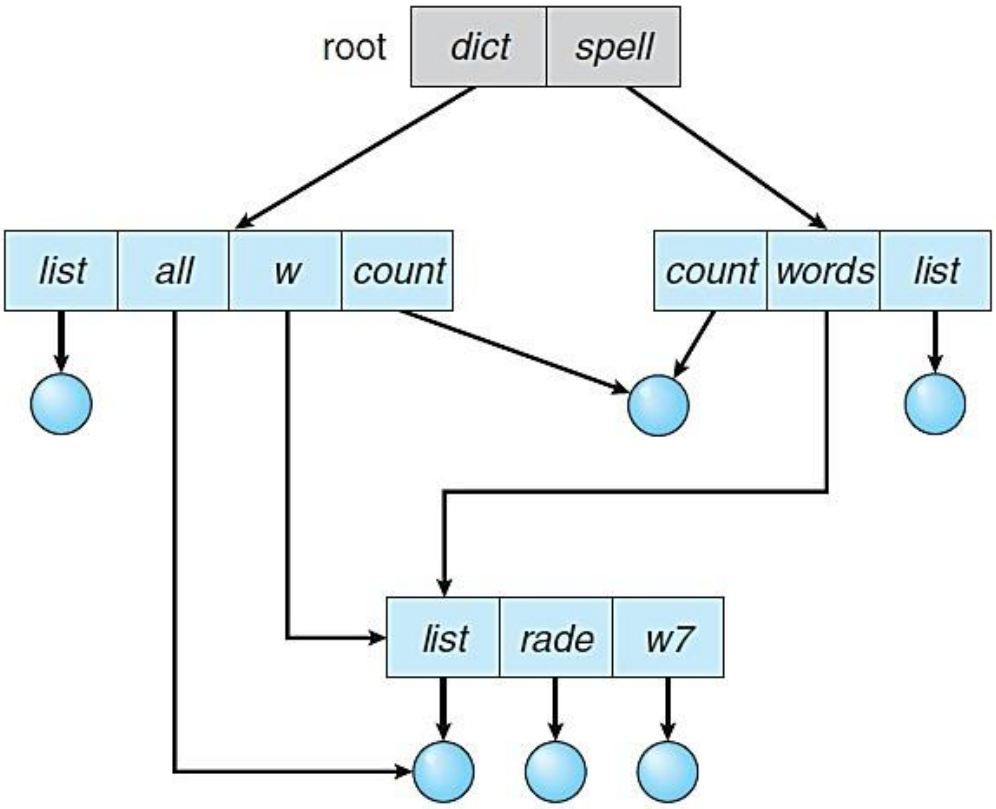
Đóng: khi một thư mục đã được đọc xong, phải đóng thư mục lại để giải phóng vùng nhớ.

Đọc: 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 chỉ bị giảm chỉ số liên kết.



Hình 4.4. Cấu trúc thư mục có sử dụng liên kết

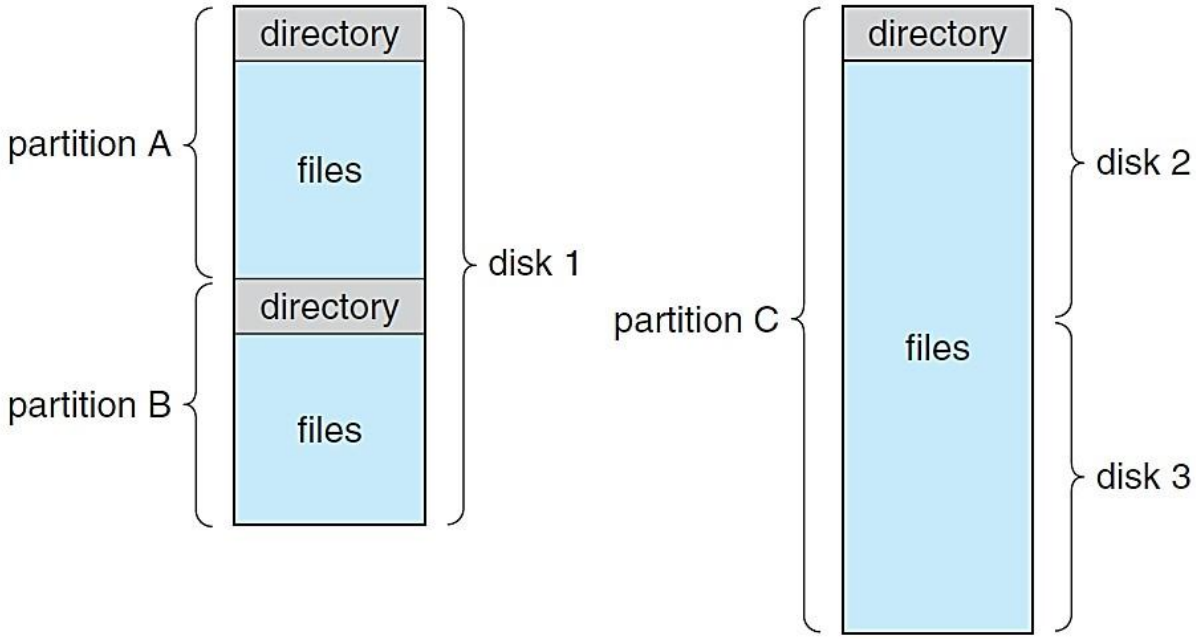
4.3. CÀI ĐẶT HỆ THỐNG QUẢN LÝ TẬP TIN

4.3.1. Giới thiệu

Người sử dụng thì quan tâm đến cách đặt tên tập tin, các thao tác trên tập tin, cây thư mục, … Nhưng đối với người cài đặt thì quan tâm đến tập tin và thư mục được lưu trữ như thế nào, vùng nhớ trên đĩa được quản lý như thế nào và làm sao cho toàn bộ hệ thống làm việc hữu hiệu và tin cậy.

Phân vùng (Partition)

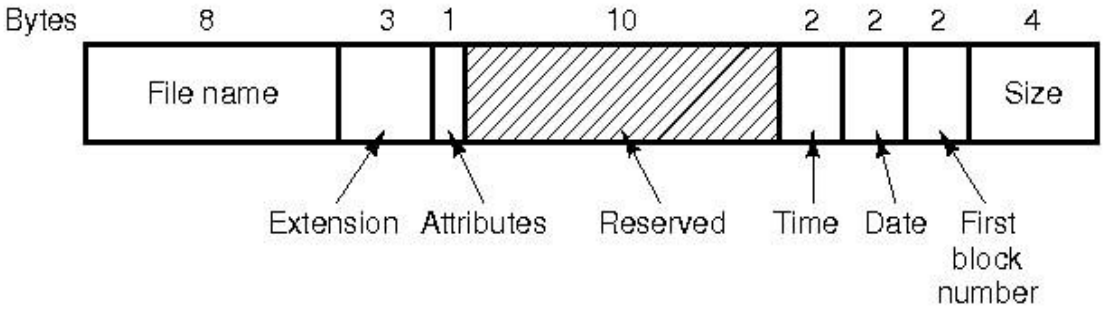
Hệ điều hành có thể chia đĩa cứng thành nhiều phân vùng, mỗi phân vùng gồm nhiều từ trụ (cylinder) liên tiếp, hoặc tập hợp nhiều đĩa cứng thành một phân vùng. Mỗi phân vùng sẽ có cấu trúc thư mục riêng để quản lý các tập tin trong phân vùng đó.



Hình 4.4. Tổ chức phân vùng đĩa

Bảng thư mục (Directory table)

Trước khi tập tin được đọc, tập tin phải được mở. Để mở tập tin, hệ thống phải biết đường dẫn do người sử dụng cung cấp và được định vị trong cấu trúc directory entry. Directory entry cung cấp các thông tin cần thiết để tìm kiếm các khối. Tùy thuộc vào mỗi hệ thống, thông tin là địa chỉ trên đĩa của toàn bộ tập tin, số hiệu của khối đầu tiên, hoặc là số i-node.



Hình 4.6. Một directory entry của FAT (MS-DOS / Windows)

Bảng phân phối vùng nhớ

Nguyên tắc là dùng để lưu trữ các khối trên đĩa cấp phát cho tập tin lưu dữ liệu.

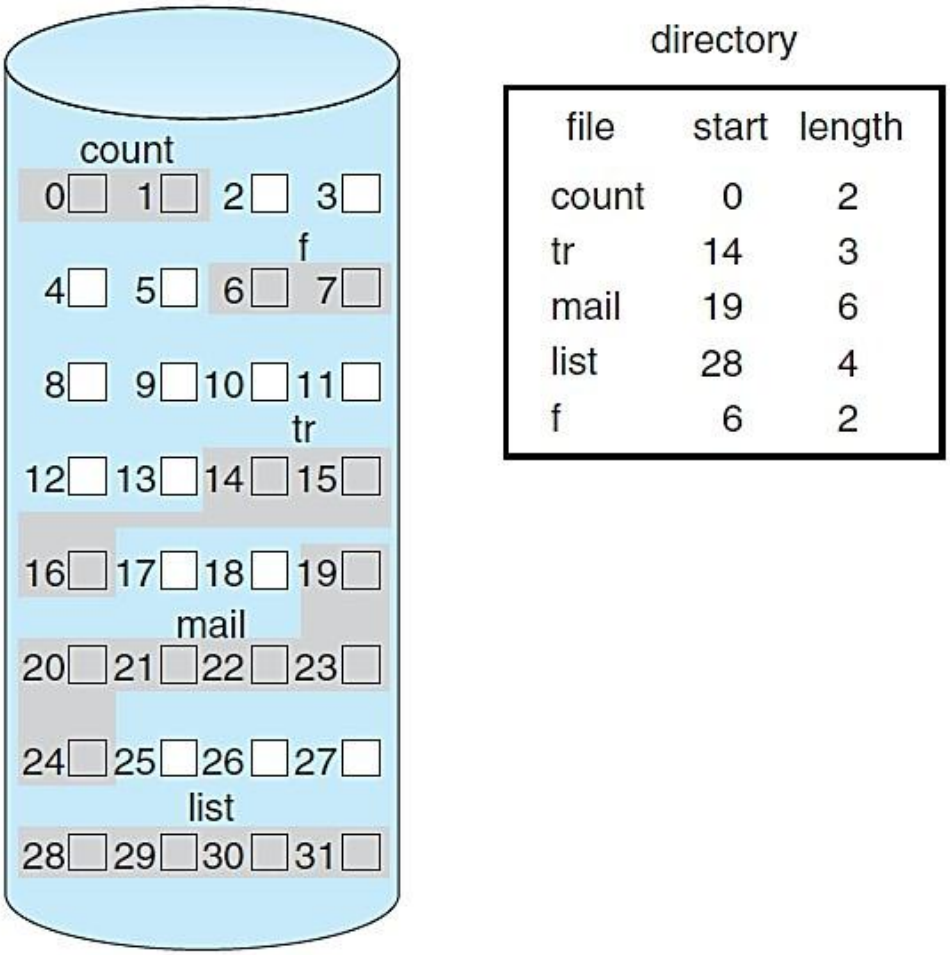
4.3.2. Cài đặt bảng phân phối vùng nhớ

Định vị liên tiếp

Lưu trữ tập tin trên dãy các khối liên tiếp.

Phương pháp này có 2 ưu điểm là dễ dàng cài đặt và dễ thao tác vì toàn bộ tập tin được đọc từ đĩa bằng thao tác đơn giản không cần định vị lại.

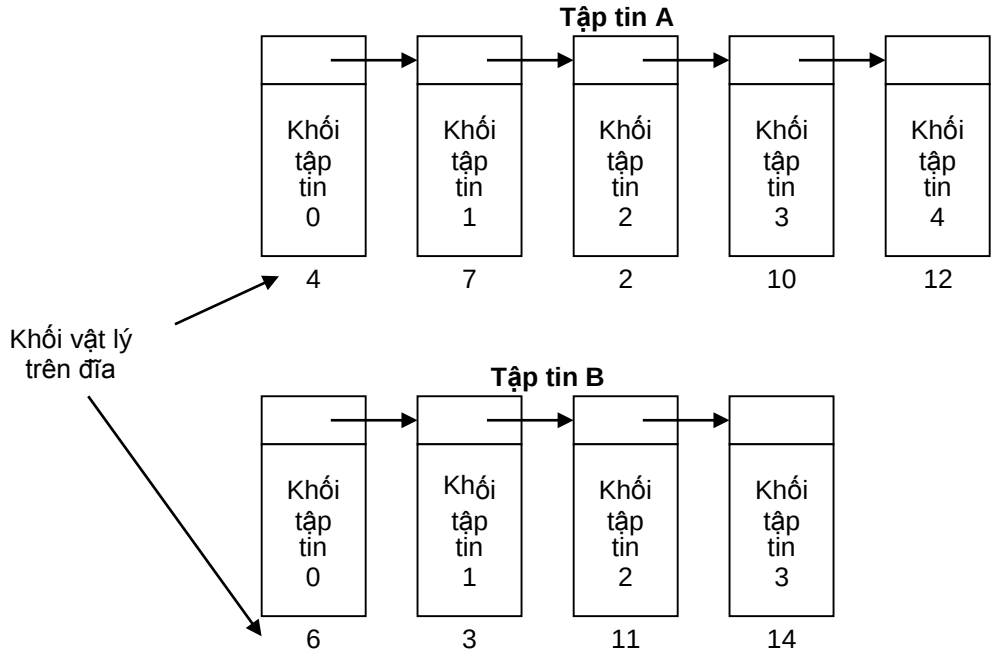
Phương pháp này cũng có 2 khuyết điểm: không linh động trừ khi biết trước kích thước tối đa của tập tin. Gây ra sự phân mảnh trên đĩa, lãng phí lớn.



Hình 4.7. Cấp phát vùng nhớ liên tục

Định vị bằng danh sách liên kết

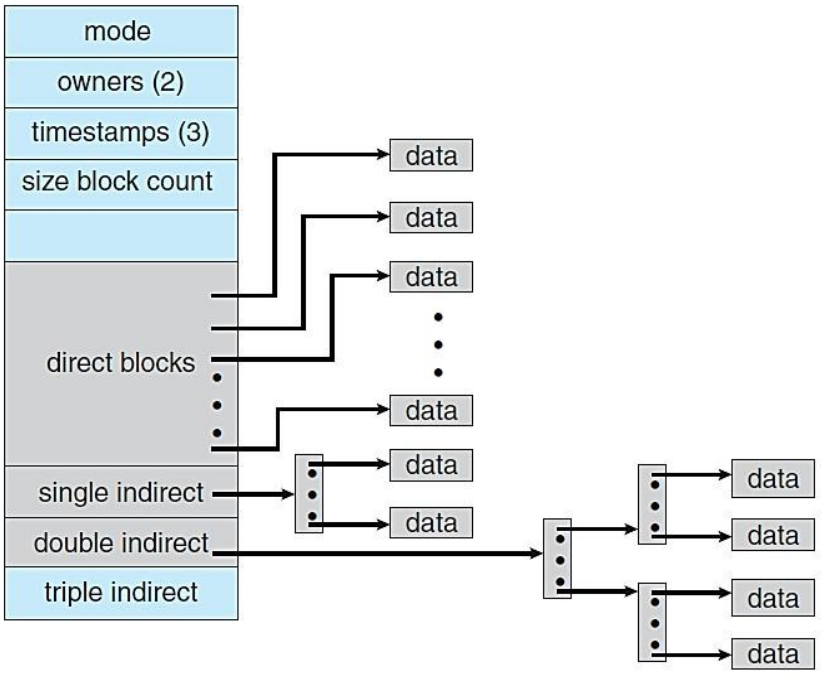
Mọi khối đều được cấp phát, không bị lãng phí trong trường hợp phân mảnh và directory entry chỉ cần chứa địa chỉ của khối đầu tiên. Tuy nhiên khối dữ liệu sẽ bị thu hẹp lại và truy xuất ngẫu nhiên sẽ bị chậm



Hình 4.8.Định vị bằng danh sách liên kết

I-nodes

Một i-node bao gồm hai phần. Phần thứ nhất là thuộc tính của tập tin. Phần thứ hai chứa địa chỉ của khối dữ liệu. Phần này chia thành 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 single indirect khối, chứa địa chỉ của một khối, trong khối đó chứa một bảng có đến 210 phần tử mà mỗi phần tử mới chứa địa chỉ của khối dữ liệu. Phần tử thứ 12 chứa double indirect khối, chứa địa chỉ của bảng các single indirect khối. Phần tử thứ 13 là triple indirect khối.



Hình 4.9. I-node trong UNIX

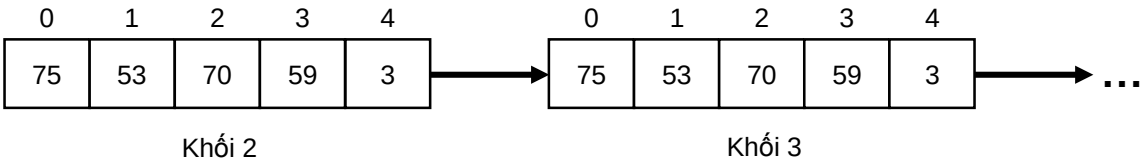
4.3.3. Quản lý các khối trống

Có hai phương pháp mà hệ điều hành thường sử dụng để quản lý các khối trống là sử dụng danh sách liên kết hoặc vector bit

Danh sách liên kết

Mỗi nút trong danh sách liên kết là một khối chứa một bảng gồm các số hiệu khối trống, phần tử cuối của bảng lưu số hiệu khối tiếp theo trong danh sách.

Ví dụ: giả sử khối đầu tiên trong danh sách liên kết là khối 2. Trong khối 2 lưu các số 75, 53, 70, 59 là các số hiệu khối trống. Khối 3 chứa bảng số hiệu các khối trống tiếp theo… Hệ điều hành chỉ cần biết số hiệu khối đầu tiên của danh sách liên kết.



Ví dụ: một đĩa dung lượng 20MB, dùng khối có kích thước 1KB. Để quản lý đĩa này, nếu đĩa hoàn toàn trống thì danh sách liên kết cần bao nhiêu khối?

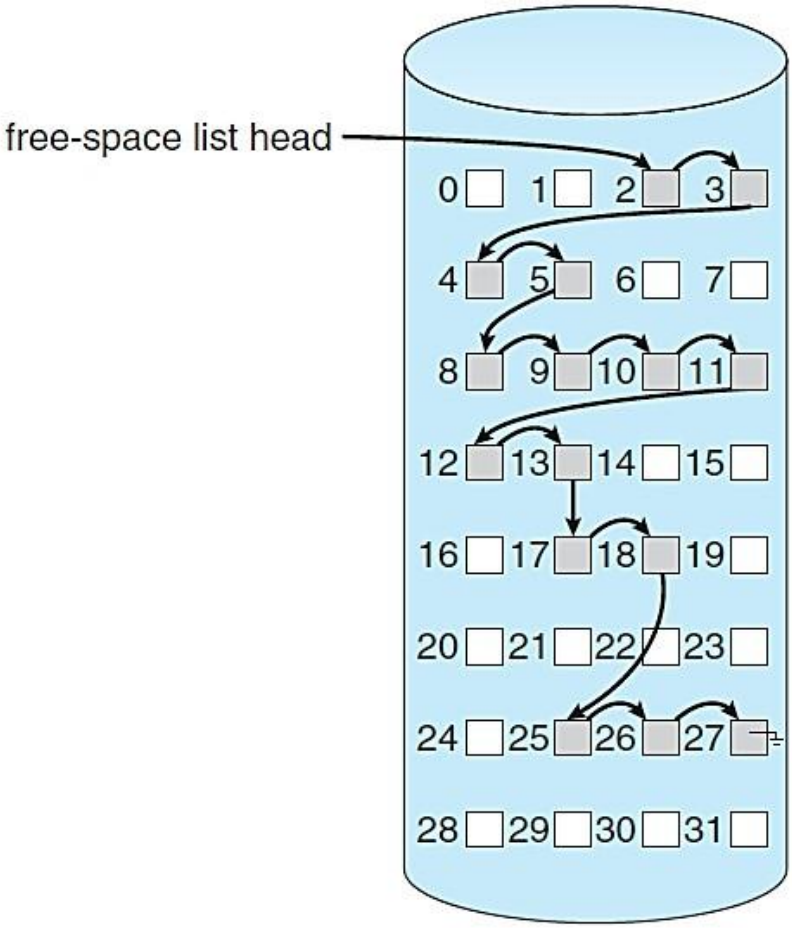
20MB = 20.2^{10} khối ~ 2^{15} khối -> cần dùng 16 bit (2 byte) để lưu một số hiệu khối

⇨ 1 khối = 1024 byte lưu được 511 số hiệu khối trống

⇨ Để quản lý đĩa có 20MB hoàn toàn trống, danh sách liên kết cần

$20 \times 2^{10} / 511 \sim 40$ khối

Nhận xét: tốn khá nhiều khối nhớ cho danh sách liên kết nếu đĩa hoàn toàn trống, nhưng sẽ ít tốn khối nhớ cho danh sách liên kết nếu đĩa gần đầy.



Hình 4.10. Quản lý khối trống sử dụng danh sách liên kết

Dùng vector bit

Là một dãy các bit, trong đó bit thứ i bằng 1 là khối thứ i còn trống, bằng 0 là khối thứ i đã sử dụng. Vector bit được lưu trên một hoặc nhiều khối đĩa, khi cần sẽ được đọc vào bộ nhớ để xử lý nhanh.

Vector bit tốn ít khối nhớ hơn danh sách liên kết nhưng kích thước vector là cố định, hệ điều hành cần đồng bộ vector bit trong bộ nhớ và vector bit trên đĩa.

Ví dụ: đĩa dung lượng 20MB, khối có kích thước 1KB sẽ có vector bit kích thước là 20×2^{10} bit -> chiếm $20 \times 10^{10} / 8 / 2^{10} \sim 3$KB (3 khối).

4.3.4. Quản lý khối hỏng

Có 2 cách:

- Lưu danh sách các sector hỏng trong một file trên đĩa
- Dùng một sector trên đĩa để lưu danh sách các sector hỏng

Khi bộ kiểm soát đĩa thực hiện truy xuất lần đầu tiên, nó đọc danh sách các khối hỏng vào bộ nhớ, từ đó không cho truy cập những khối đó nữa.

TÓM TẮT CHƯƠNG

- Tập tin là đơn vị lưu trữ thông tin cơ bản trên đĩa.
- Tập tin có thể chứa chương trình hay dữ liệu, có thể có cấu trúc hoặc chỉ là một dãy các byte.
- Tập tin có thể được truy xuất tuần tự hoặc ngẫu nhiên
- Thư mục dùng để dể quản lý các tập tin trên đĩa. Thư mục thường có cấu trúc dạng cấp bậc.
- Hệ điều hành có thể chia đĩa cứng thành nhiều phân vùng, mỗi phân vùng sẽ có cấu trúc thư mục riêng để quản lý các tập tin trong phân vùng đó.
- Bảng thư mục là một dạng cài đặt của cấu trúc thư mục. Bảng thư mục có nhiều mục, mỗi mục lưu trữ thông tin mô tả của một file.
- Có nhiều cách cài đặt bảng phân phối vùng nhớ khác nhau: liên tiếp, danh sách liên kết, danh sách liên kết sử dụng chỉ mục, i-nodes, …

BÀI TẬP

1) Giải thích vì sao cần phân chia đĩa cứng thành các phân vùng (partition)?
2) Viết các chương trình mô phỏng giải thuật cấp phát liên tục.
3) Viết các chương trình mô phỏng giải thuật cấp phát theo danh sách liên kết.
4) Viết các chương trình mô phỏng giải thuật cấp phát theo danh sách liên kết sử dụng index.
5) Tìm hiểu cấu trúc bảng FAT. Viết chương trình truy xuất thông tin trực tiếp trên bảng FAT.
6) Viết các chương trình sau (có thể sử dụng Pascal hoặc C):
    a. Mô phỏng lệnh DIR của DOS.
    b. Mô phỏng lệnh COPY của DOS.
    c. Mô phỏng lệnh TREE của DOS.
    d. Mô phỏng lệnh TYPE của DOS.
    e. Mô phỏng lệnh DEL của DOS.
    f. Mô phỏng lệnh MD của DOS.
    g. Mô phỏng lệnh CD của DOS.
    h. Mô phỏng lệnh RD của DOS.
    i. Mô phỏng lệnh MOVE của DOS.

7) Tìm hiểu hệ thống file NTFS. Sử dụng các chức năng của NTFS trong hệ điều hành Windows.
8) Tìm hiểu các hệ thống file EXT của LINUX.

CHƯƠNG 5. TIẾN TRÌNH VÀ DÒNG

5.1. TIẾN TRÌNH (PROCESS)

5.1.1. Khái niệm

Tiến trình là một chương trình đang xử lý. Mỗi tiến trình sở hữu một con trỏ lệnh, tập các thanh ghi và các biến. Để hoàn thành tác vụ 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.

5.1.2. Mô hình tiến trình

Để hỗ trợ sự đa chương, máy tính phải có khả năng thực hiện nhiều tác vụ 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 tiến trình để duy trì hoạt động của nhiều tiến trình cùng lúc, điều này tạo cảm giác có nhiều hoạt động được thực hiện đồng thời.

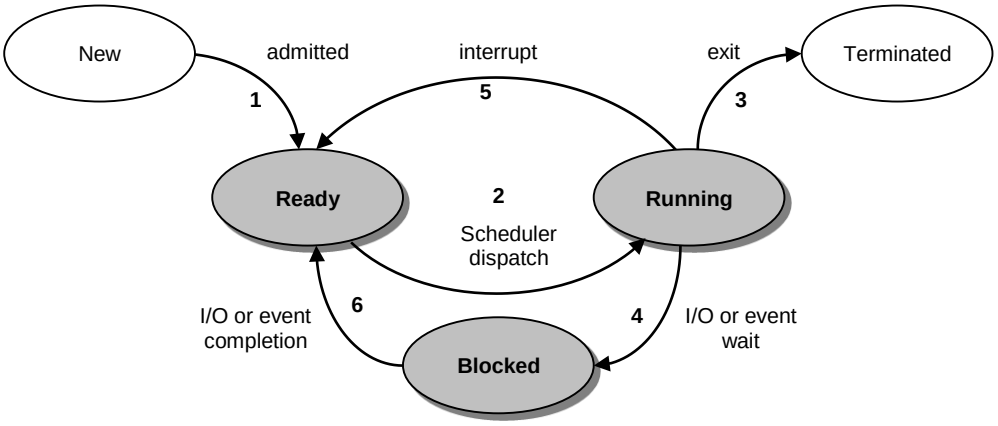
Về mặt ý niệm, có thể xem như mỗi tiến trình sở hữu một bộ xử lý ảo của riêng nó. Nhưng trong thực tế, chỉ có một bộ xử lý thật sự được chuyển đổi qua lại giữa các tiến trình. Sự chuyển đổi nhanh chóng này gọi là sự đa chương (multiprogramming). Hệ điều hành chịu trách nhiệm sử dụng một thuật toán điều 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).

5.1.3. Các trạng thái của tiến trình

Trạng thái của một 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:

- New: tiến trình đang được tạo lập.
- Running: các chị thị của tiến trình đang được thực thi.
- Blocked/waiting: tiến trình chờ được cấp phát tài nguyên, hay chờ một sự kiện xảy ra.
- Ready: tiến trình đang chờ được cấp phát CPU.
- Terminated: tiến trình đã 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.



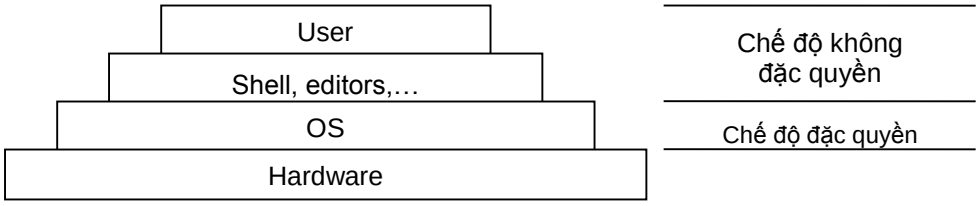
Hình 5.1. Sơ đồ chuyển trạng thái của tiến trình

Các cung chuyển tiếp trong sơ đồ trạng thái biểu diễn sáu sự chuyển trạng thái có thể xảy ra trong các điều kiện sau:

- (1): tiến trình mới được tạo đưa vào hệ thống.
- (2): bộ điều phối cấp phát cho tiến trình một khoảng thời gian sử dụng CPU.
- (3): tiến trình kết thúc.
- (4): tiến trình yêu cầu tài nguyên nhưng chưa được đáp ứng vì tài nguyên chưa sẵn sàng để cấp phát tại thời điểm đó; hoặc tiến trình phải chờ một sự kiện hay một thao tác nhập/xuất.
- (5): bộ điều phối chọn một tiến trình khác để cho xử lý.
- (6): tài nguyên mà tiến trình yêu cầu trở nên sẵn sàng để cấp phát; hay sự kiện hoặc thao tác nhập/xuất tiến trình đang đợi hoàn tất.

5.1.4. Chế độ xử lý của tiến trình

Để đảm bảo hệ thống hoạt động đúng đắn, hệ điều hành cần được bảo vệ khỏi sự xâm phạm của các tiến trình. Bản thân các tiến trình và dữ liệu cũng cần được bảo vệ để tránh các ảnh hưởng sai lạc lẫn nhau. Một cách tiếp cận để giải quyết vấn đề là phân biệt hai chế độ xử lý cho các tiến trình: chế độ đặc quyền và chế độ không đặc quyền nhờ vào sự trợ giúp của cơ chế phần cứng. Tập lệnh của CPU được phân chia thành các lệnh đặc quyền và lệnh không đặc quyền. Cơ chế phần cứng chỉ cho phép các lệnh đặc quyền được thực hiện trong chế độ đặc quyền. Thông thường chỉ có hệ điều hành hoạt động trong chế độ đặc quyền, các tiến trình của người dùng hoạt động trong chế độ không đặc quyền, không thực hiện được các lệnh có nguy cơ ảnh hưởng đến hệ thống. Như vậy hệ điều hành sẽ được bảo vệ. Khi một tiến trình người dùng gọi đến một lời gọi hệ thống, tiến trình của hệ điều hành xử lý lời gọi này sẽ hoạt động trong chế độ đặc quyền, sau khi hoàn tất thì trả quyền điều khiển về cho tiến trình người dùng trong chế độ không đặc quyền.



Hình 5.2. Hai chế độ xử lý của tiến trình

5.1.5. 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

Giúp phân biệt các tiến trình trong hệ thống với nhau.

Trạng thái tiến trình

Xác định hoạt động hiện hành của tiến trình.

Ngữ cảnh của tiến trình

Mô tả các tài nguyên dành cho tiến 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 phục hồi hoạt động của tiến trình đúng như trước lúc 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 phát cho tiến trình.
- Tài nguyên sử dụng: danh sách các tài nguyên 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

Phản ánh các thông tin về quan hệ của tiến trình với các tiến trình khác trong hệ thống, bao gồm:

- Tiến trình cha: tiến trình tạo lập tiến trình này.
- Tiến trình con: các tiến trình do tiến trình này tạo lập.
- Độ ưu tiên: giúp cho bộ điều phối có thông tin để lựa chọn tiến trình được cấp phát CPU.

Thông tin thống kê

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 việc đánh giá tình hình hệ thống và dự đoán các tình huống tương lai.

5.1.6. Thao tác trên tiến trình

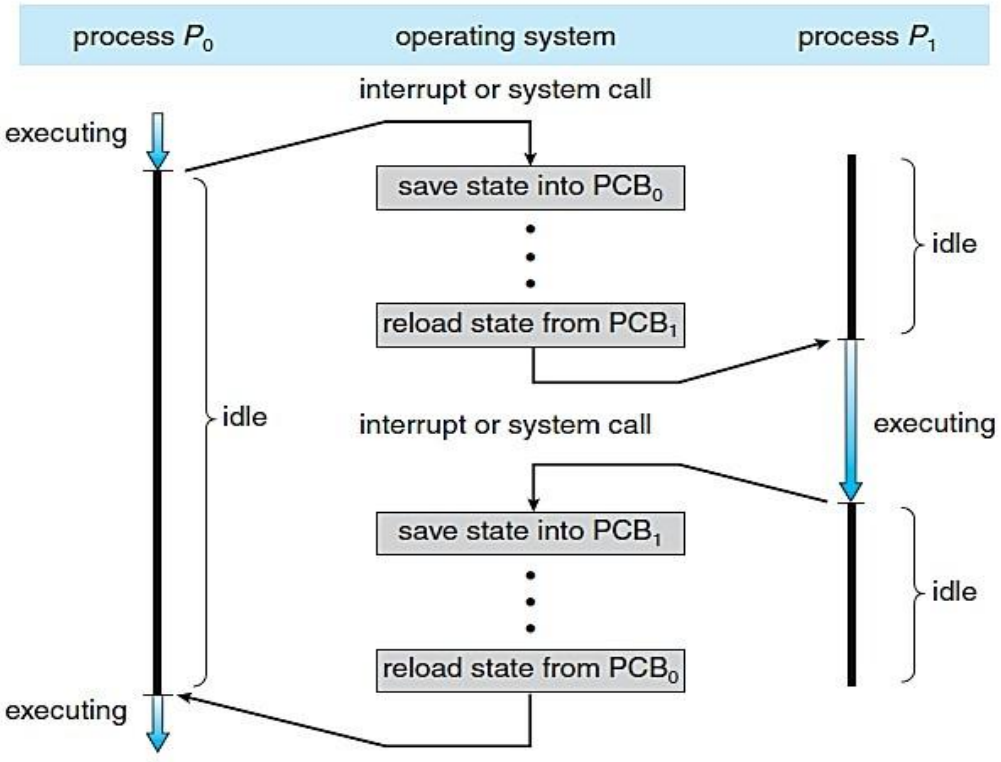
Hệ điều hành cung cấp các thao tác chủ yếu sau đây trên một tiến trình:

- Tạo lập tiến 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

5.1.7. Chuyển đổi ngữ cảnh (Context switch)

Chuyển đổi CPU từ tiến trình này sang tiến trình khác yêu cầu hệ điều hành phải thực hiện lưu trữ trạng thái của tiến trình đang thực hiện và phục hồi trạng thái của tiến trình cần chuyển đến. Quá trình này gọi là chuyển đổi ngữ cảnh.

Thời gian chuyển đổi ngữ cảnh mặc dù thường rất ngắn (vài ms), nhưng đây là khoảng thời gian lãng phí vì CPU không làm việc có ích.



Hình 5.3. Quá trình chuyển đổi ngữ cảnh

5.1.8. Cấp phát tài nguyên cho tiến trình

Khi có nhiều người sử dụng đồng thời làm việc trong hệ thống, hệ điều hành 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 quả. 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 (CPU, bộ nhớ chính, hệ thống tập tin), trong khi những tài nguyên khác (các thiết bị nhập xuất,…) có thể cần các giải thuật cấp phát và giải phóng tổng quát hơn.

Các mục tiêu của giải 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 thỏa 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.

5.2. TIỂU TRÌNH (THREAD)

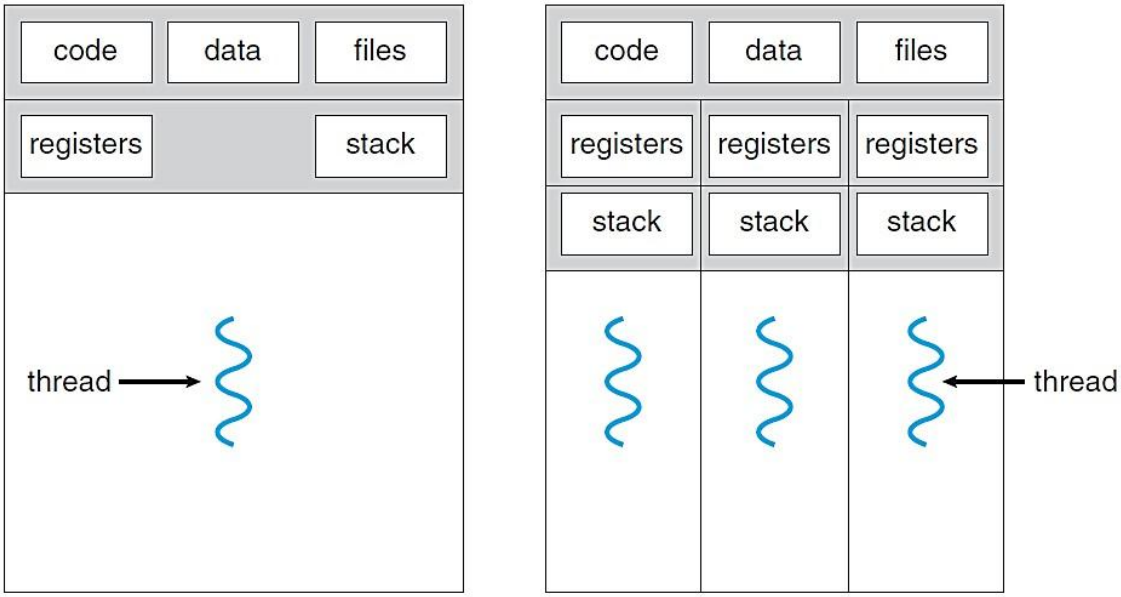
Trong hầu hết các hệ điều hành, mỗi tiến trình có một không gian địa chỉ và chỉ có một dòng xử lý. Tuy nhiên, có nhiều tình huống người sử dụng mong muốn có nhiều dòng xử lý cùng chia sẻ một không gian địa chỉ và các dòng xử lý này hoạt động song song tương tự như các tiến trình phân biệt (ngoại trừ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

5.2.1. Mô hình tiểu trình (Thread Model)

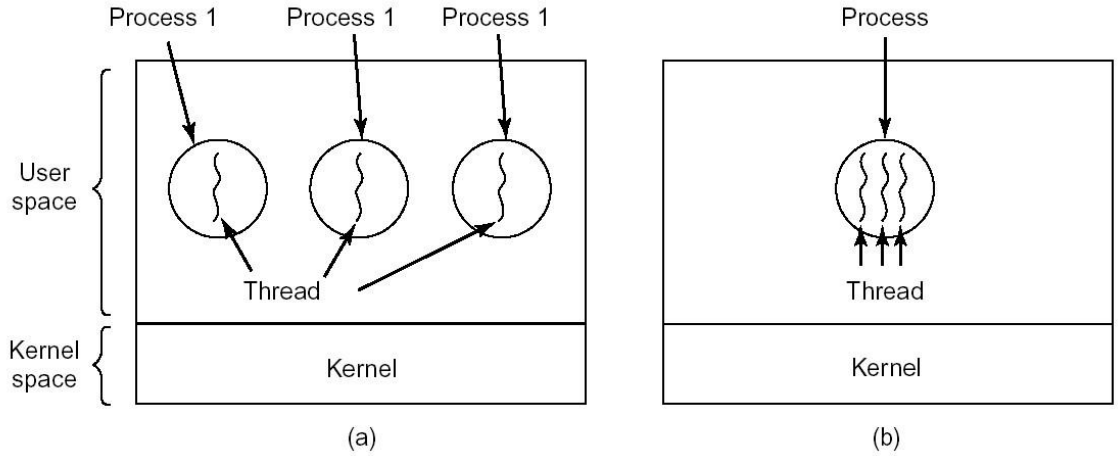
Một tiểu trình là một đơn vị xử lý cơ bản trong hệ thống. Mỗi thread xử lý tuần tự đoạn code của nó, sở hữu một con trỏ lệnh, tập các thanh ghi và một vùng nhớ stack riêng. Các thread chia sẻ

CPU với nhau giống như cách chia sẻ giữa các tiến trình: một thread chờ xử lý trong khi các thread khác chờ đến lượt. Một thread cũng có thể tạo lập các thread con và nhận các trạng thái khác nhau như một tiến trình thật sự. Một tiến trình có thể sở hữu nhiều thread.

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 thread trong cùng một tiến trình lại chia sẻ một không gian địa chỉ chung, điều này có nghĩa là các thread chia sẻ các biến toàn cục của tiến trình. Một thread có thể truy xuất đến cả các stack của những thread khác trong cùng một tiến trình. Cấu trúc này không đề nghị một cơ chế bảo vệ nào, và điều này cũng không thật sự cần thiết vì các thread trong cùng một tiến trình thuộc về cùng một sở hữu chủ đã tạo ra chúng trong ý định cho phép chúng hợp tác với nhau.



Hình 5.4. Mô hình tiến trình đơn luồng và đa luồng

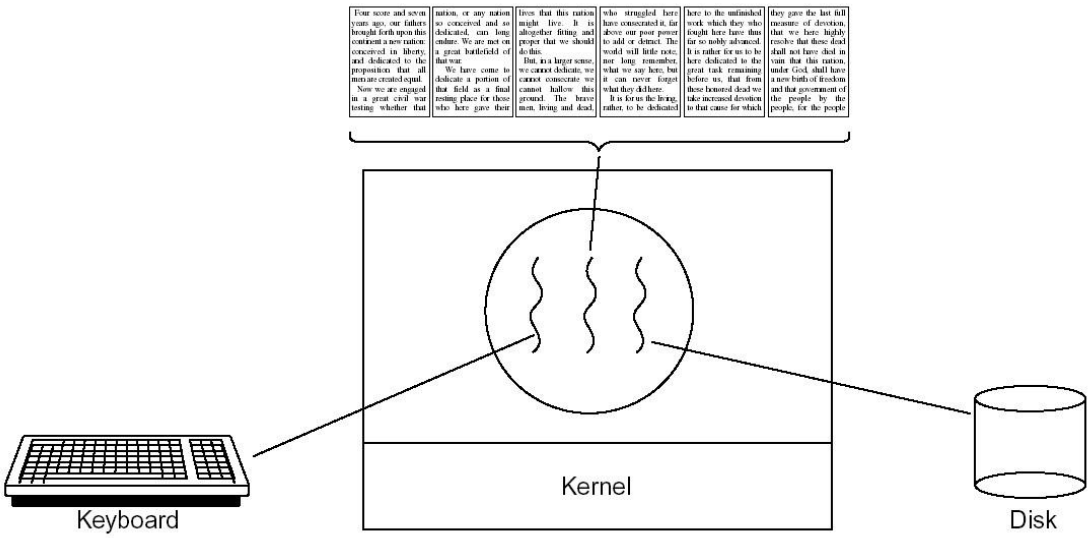


Hình 5.5. a) 3 tiến trình với 1 thread b) 1 tiến trình với 3 thread

Thông tin chia sẻ giữa các thread trong cùng tiến trình	Các thông tin riêng của thread
Không gian địa chỉ	Bộ đếm chương trình
Các biến toàn cục	Các thanh ghi
Các tập tin mở	Ngăn xếp
Các tiến trình con	Trạng thái
Các cảnh báo	
Các tín hiệu và các bộ xử lý tín hiệu	
Thông tin tài khoản	

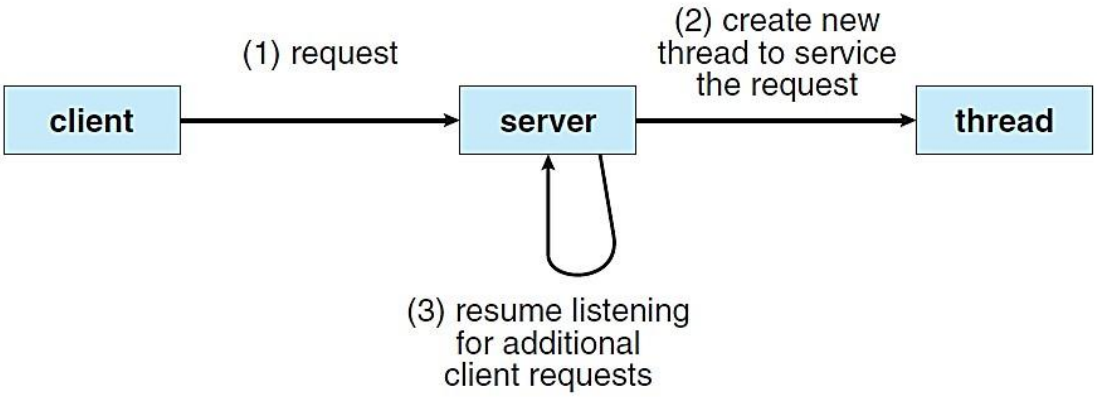
5.2.2. Ví dụ

Một chương trình xử lý văn bản (word processor) có thể chia thành 3 thread: một thread giao tiếp với người dùng, một thread thực hiện định dạng văn bản và một thread thực hiện truy xuất đĩa. Giả sử người dùng xử lý một văn bản dài khoảng 800 trang, khi thực hiện thao tác xóa hoặc chỉnh sửa văn bản chương trình cần phải thực hiện định dạng lại toàn bộ văn bản (thời gian rất dài), khi đó thread định dạng sẽ được thực hiện nền song song với thread giao tiếp, người dùng sẽ không phải chờ đợi trong lúc định dạng. Ngoài ra, việc lưu trữ dự phòng (backup) sẽ được thực hiện ngầm với thread truy xuất đĩa trong lúc vẫn giao tiếp với người dùng. Các tính năng trên không thể thực hiện với nhiều process độc lập vì chúng cần phải chia sẻ vùng nhớ chung (vùng văn bản).



Hình 5.6. Chương trình xử lý văn bản với 3 thread

Một tiến trình dịch vụ (file, web, ftp, …) trên máy chủ (server) có thể sử dụng mô hình đa luồng để đáp ứng yêu cầu từ các máy con. Tiến trình sẽ hoạt động với một luồng nghe (Listener) chuyên nhận lời yêu cầu từ client, khi yêu cầu được chấp nhận, một luồng mới sẽ được tạo ra để phục vụ cho yêu cầu đó.



Hình 5.7. Mô hình đa luồng cho server

5.3. ĐIỀU PHỐI TIẾN TRÌNH (SCHEDULE)

Trong môi trường đa chương, có thể xảy ra tình huống nhiều tiến trình đồng thời sẵn sàng để xử lý. Lúc này, hệ điều hành cần phải thực hiện lựa chọn tiến trình được xử lý tiếp theo. Bộ phận thực hiện công việc này gọi là bộ điều phối (scheduler).

5.3.1. Giới thiệu

Mục tiêu điều phối

Mục tiêu chung:

- Công bằng (Fairness): các tiến trình được chia sẻ CPU một cách công bằng, không có tiến trình phải chờ vô hạn để được cấp phát CPU.

- Tôn trọng chính sách (Policy enforcement): việc điều phối phải tuân thủ theo các chính sách đã đặt ra của hệ thống.
- Cân bằng (Balance): cân bằng hoạt động của tất cả các bộ phận trong hệ thống.

Trong hệ thống theo lô (Batch system):

- Thông lượng (Throughput): cực đại hóa số công việc được xử lý trong một đơn vị thời gian.
- Thời gian lưu lại trong hệ thống (Turnaround time): cực tiểu hóa thời gian hoàn tất một tác vụ.
- Tận dụng CPU (CPU utilization): hệ thống phải tận dụng hiệu quả CPU.

Trong hệ thống tương tác (interactive system):

- Thời gian đáp ứng (Response time): cực tiểu hóa thời gian đáp ứng của tác vụ.
- Cân đối (Proportionality): cân đối thời gian đáp ứng của các tác vụ theo yêu cầu của người sử dụng. *Các đặc tính của tiến trình*
- Tiến trình hướng nhập xuất (I/O-bound process): thực hiện các yêu cầu I/O nhiều hơn sử dụng CPU. Thường các tiến trình dạng này sử dụng CPU trong những khoảng thời gian rất ngắn.
- Tiến trình hướng xử lý (CPU-bound process): sử dụng CPU nhiều, ít thực hiện các yêu cầu nhập xuất.
- Tiến trình tương tác hay xử lý theo lô: người sử dụng theo kiểu tương tác thường yêu cầu được hồi đáp tức thời đối với các yêu cầu của họ, trong khi các tiến trình của tác vụ xử lý theo lô nói chung có thể trì hoãn trong một thời gian chấp nhận được.
- Độ ưu tiên của tiến trình: các tiến trình có thể được phân cấp ưu tiên theo một số tiêu chuẩn nào đó. Các tiến trình có độ ưu tiên cao cần được thực hiện trước.
- Thời gian đã sử dụng CPU của tiến trình: một số quan điểm ưu tiên chọn những tiến trình đã sử dụng CPU nhiều thời gian nhất vì hy vọng chúng sẽ cần ít thời gian nhất để hoàn tất và rời khỏi hệ thống. Tuy nhiên cũng có quan điểm cho rằng các tiến trình nhận được CPU trong ít thời gian là những tiến trình đã phải chờ lâu nhất, do vậy ưu tiên chọn chúng.

  Thời gian còn lại để tiến trình hoàn tất: có thể giảm thiểu thời gian chờ đợi trung bình của các tiến trình bằng cách cho các tiến trình cần ít thời gian nhất để hoàn tất được thực hiện trước. Tuy nhiên thực tế rất hiếm khi biết được tiến trình cần bao nhiêu thời gian nữa để kết thúc xử lý.

Điều phối ưu tiên (preemptive) và không ưu tiên (non-preemptive)

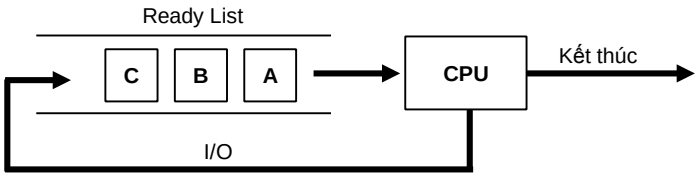
Hệ thống máy tính thường có một đồng hồ ngắt giờ (Interval timer / Interrupt clock), sau những khoảng thời gian *t* sẽ phát sinh một ngắt (interrupt), lúc đó quyền điều khiển được trả về cho chương trình xử lý ngắt của hệ điều hành. Tùy theo cách xử lý của hệ điều hành với ngắt giờ, ta có 2 kiểu điều phối:

- Non-preemptive: khi CPU được bộ điều phối cấp cho tiến trình, nó sẽ được quyền sử dụng đến khi kết thúc hoặc khi nó chuyển sang trạng thái Blocked. Việc điều phối không được thực hiện khi ngắt đồng hồ được kích hoạt.
- Preemptive: khi bộ điều phối cấp CPU cho tiến trình nó sẽ quy định thời gian sử dụng CPU của tiến trình. Khi ngắt đồng hồ được kích hoạt, bộ điều phối sẽ kiểm tra thời gian sử dụng CPU của tiến trình, nếu hết thời gian mà tiến trình chưa giải phóng CPU (kết thúc hoặc chuyển sang trạng thái Blocked) thì tiến trình sẽ được chuyển sang trạng thái Ready, nhường CPU cho tiến trình khác.

5.3.2. Các chiến lược điều phối

a) Chiến lược FIFO (First In First Out)

Nguyên tắc: CPU 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 I/O.



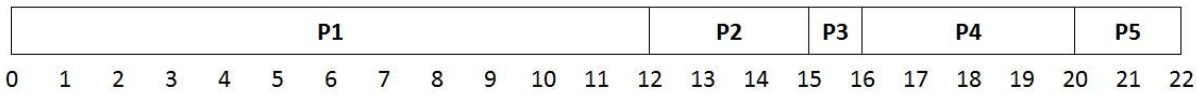
Hình 5.8. Chiến lược FIFO

Ví dụ:

Xét tập các tiến trình sau:

Tiến trình	Thời điểm vào	Thời gian xử lý
P1	0	12
P2	1	3
P3	3	1
P4	6	4
P5	10	2

Thứ tự cấp phát CPU cho tiến trình là:



Thời gian chờ đợi được xử lý của từng tiến trình:

P1 = 0

P2 = 12 - 1 = 11

P3 = 15 - 3 = 12

P4 = 16 - 6 = 10

P5 = 20 - 10 = 10

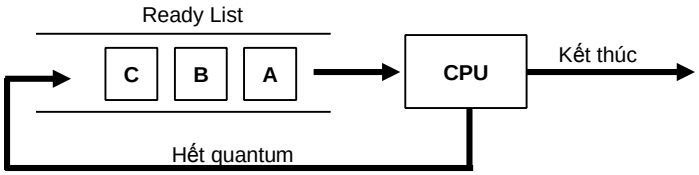
-> Thời gian chờ đợi trung bình của các tiến trình: (0 + 11 + 12 + 10 + 10)/5 = 8.6

Ưu & khuyết điểm:

- Đơn giản, dễ triển khai.
- Không phù hợp với hệ thống tương tác do CPU không được cấp phát đều đặn cho các tiến trình.
- Xảy ra hiện tượng tích lũy thời gian chờ, các tiến trình ngắn phải đợi tiến trình có yêu cầu CPU thời gian dài kết thúc xử lý.

b) Chiến lược 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 sử dụng CPU gọi là quantum. Đây là một giải thuật điều phối preemptive: 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.



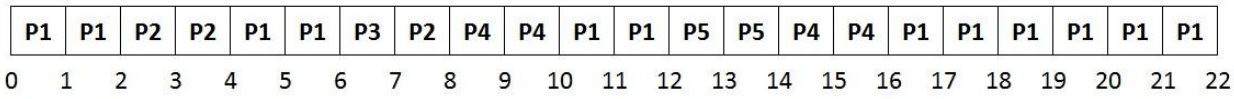
Hình 5.9. Chiến lược Round Robin

Ví dụ: Xét tập các tiến trình sau:

Tiến trình	Thời điểm vào	Thời gian xử lý
P1	0	12
P2	1	3

P3	3	1
P4	6	4
P5	10	2

Nếu sử dụng quantum là 2, thứ tự cấp phát CPU cho tiến trình sẽ là:



Thời gian chờ đợi được xử lý của từng tiến trình:

P1 = 0 + (4 - 2) + (10 - 6) + (16 - 12) = 10

P2 = (2 - 1) + (7 - 4) = 4

P3 = 6 - 3 = 3

P4 = (8 - 6) + (14 - 10) = 6

P5 = 12 - 10 = 2

Thời gian chờ đợi trung bình của các tiến trình: (10 + 4 + 3 + 6 + 2)/5 = 5.0

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

Ưu & khuyết điểm:

- Phù hợp với hệ thống chia sẻ thời gian.
- Khó trong việc xác định quantum hợp lý: nếu quantum quá dài thì trở thành giải thuật FIFO, nếu quantum quá bé sẽ phát sinh nhiều thời gian chuyển đổi ngữ cảnh giữa các tiến trình làm cho việc sử dụng CPU kém hiệu quả.

c) Điều phối với độ ưu tiên (Priority)

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. Độ ưu tiên có thể được định nghĩa nội tại hay nhờ vào các yếu tố bên ngoài.

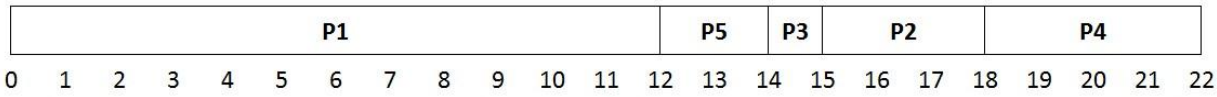
Giải thuật điều phối với độ ưu tiên có thể theo nguyên tắc preemptive hay non-preemptive. Khi một tiến trình được đưa vào danh sách các tiến trình sẵn sàng, độ ưu tiên của nó được so sánh với độ ưu tiên của tiến trình hiện hành đang được xử lý. Giải thuật điều phối với độ ưu tiên preemptive sẽ thu hồi 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. Còn giải thuật non-preemptive sẽ chỉ 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: Xét tập các tiến trình sau:

Tiến trình	Thời điểm vào	Thời gian xử lý	Độ ưu tiên
P1	0	12	3
P2	1	3	2
P3	3	1	1
P4	6	4	4
P5	10	2	0

Qui ước là giá trị biểu diễn càng nhỏ thì độ ưu tiên càng cao.

Kết quả điều phối theo chế độ độc quyền (non-preemptive)



Thời gian chờ đợi được xử lý của từng tiến trình:

P1 = 0

P2 = 15 - 1 = 14

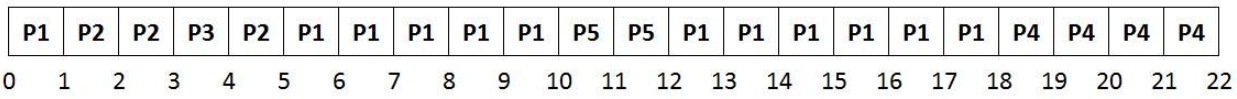
P3 = 14 - 3 = 11

P4 = 18 - 6 = 12

P5 = 12 - 10 = 2

Thời gian chờ đợi trung bình của các tiến trình: (0 + 14 + 11 + 12 + 2)/5 = 7.8

Kết quả điều phối theo chế độ không độc quyền (preemptive)



Thời gian chờ đợi được xử lý của từng tiến trình:

P1 = 0 + (5 - 1) + (12 - 10) = 6

P2 = 0 + (4 - 3) = 1

P3 = 0

P4 = 18 - 6 = 12

P5 = 0

Thời gian chờ đợi trung bình của các tiến trình: (6 + 1 + 0 + 12 + 0)/5 = 3.8

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 sự chuyển đổi quyền sử dụng CPU. Quá trình này gọi là sự lão hóa (aging) tiến trình.

d) Chiến lược SJF (Shortest Job First)

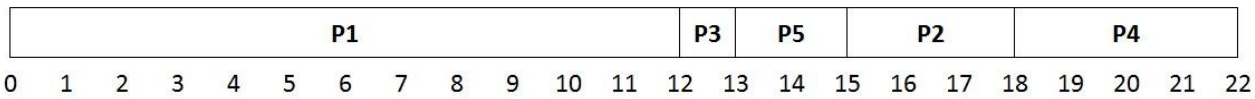
Nguyên tắc: đây là một trường hợp đặt biệt của điều phối với độ ưu tiên. Trong giải thuật này, độ ưu tiên p được gán cho mỗi tiến trình là nghịch đảo của thời gian xử lý t mà tiến trình yêu cầu. CPU sẽ được cấp phát cho tiến trình yêu cầu ít thời gian nhất. Giải thuật này cũng có thể theo nguyên tắc preemptive hay non-preemptive. 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. Nếu thời gian xử lý của tiến trình mới ngắn hơn thời gian yêu cầu xử lý còn lại của tiến trình hiện hành, giải thuật SJF preemptive sẽ dừng hoạt động của tiến trình hiện hành, trong khi giải thuật non-preemptive sẽ cho phép tiến trình hiện hành tiếp tục xử lý.

Thảo luận: giải thuật này cho phép đạt được thời gian chờ trung bình cực tiểu. Khó khăn lớn nhất của giải thuật là xác định thời gian yêu cầu xử lý còn lại của tiến trình. Thường phải thực hiện dự đoán thông qua các khoảng thời gian đã sử dụng CPU trước kia của tiến trình.

Ví dụ: Xét tập các tiến trình sau:

Tiến trình	**Thời điểm vào**	**Thời gian xử lý**
P1	0	12
P2	1	3
P3	3	1
P4	6	4
P5	10	2

Kết quả điều phối theo chế độ độc quyền (non-preemptive)



Thời gian chờ đợi được xử lý của từng tiến trình:

P1 = 0

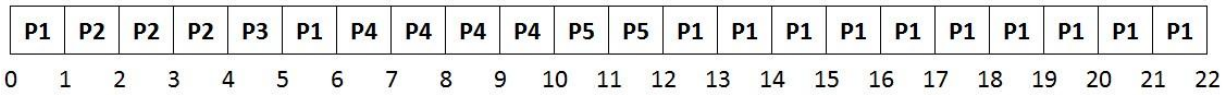
P2 = 15 - 1 = 14 P3 = 12 - 3 = 9

P4 = 18 - 6 = 12

P5 = 13 - 10 = 3

Thời gian chờ đợi trung bình của các tiến trình: (0 + 14 + 9 + 12 + 3)/5 = 7.6

Kết quả điều phối theo chế độ không độc quyền (preemptive)



Thời gian chờ đợi được xử lý của từng tiến trình:

P1 = 0 + (5 - 1) + (12 - 6) = 10

P2 = 0

P3 = 4 - 3 = 1

P4 = 0

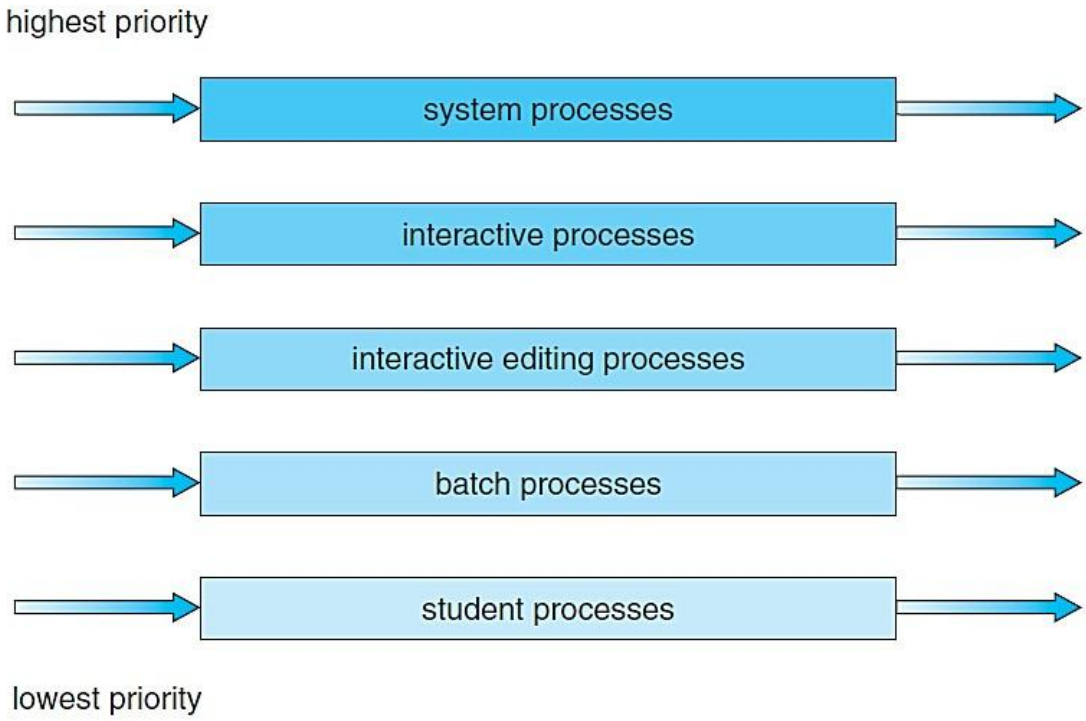
P5 = 0

Thời gian chờ đợi trung bình của các tiến trình: (10 + 0 + 1 + 0 + 0)/5 =

2.2

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 theo 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 thích hợp để điều phối. Ngoài ra, còn có một giải thuật điều phối giữa các nhóm, thường giải thuật này là giải thuật preemptive và sử dụng độ ưu tiên cố định. CPU sẽ luân phiên phục vụ các danh sách theo độ ưu tiên của chúng.



Hình 5.10. Mô hình điều phối nhiều cấp độ ưu tiên

Thảo luận: có thể xảy ra tình trạng đói CPU của các tiến trình ở 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. Tương tự, 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 quyết định 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 đinh một tiến trình mới được đưa vào hệ thống sẽ thuộc danh sách ứng với độ ưu tiên nào.

5.4. ĐỒNG BỘ HÓA TIẾN TRÌNH

5.4.1. Giới thiệu

Nhu cầu đồng bộ hóa

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 vì các lý do sau đây:

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 các 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 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.

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 CPU, … 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.

Miền tranh chấp (Critical Section)

Đoạn chương trình trong đó có khả năng xảy ra các mâu thuẫn truy xuất trên tài nguyên chung được gọi là miền tranh chấp (Critical Section).

Ví dụ: giả sử có hai tiến trình P1 và P2 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(”Không thể rút tiền”);
```

Giả sử trong tài khoản hiện còn 800, P1 muốn rút 500 và P2 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 True, P1 hết thời gian xử lý mà hệ thống cho phép, hệ điều hành cấp phát CPU cho P2.

- P2 kiểm tra cùng điều kiện trên và cũng nhận được kết quả là True (do P1 vẫn chưa rút tiền) và rút 400. Giá trị của taikhoan được cập nhật lại là 400.
    - Khi P1 được tái kích hoạt và tiếp tục xử lý, nó thực hiện tiếp câu lệnh rút tiền và cập nhật lại taikhoan là -100 -> Tình huống lỗi xảy ra.

Các tình huống tương tự như thế có thể xảy ra khi có nhiều tiến trình đọc và ghi dữ liệu trên 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).

Trong ví dụ, trên đoạn mã: if (taikhoan – tienrut >=0) taikhoan = taikhoan - tienrut ; của mỗi tiến trình tạo thành một miền tranh chấp.

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 tranh chấp.

Một phương pháp giải quyết tốt bài toán miền tranh chấp cần thỏa mãn 4 điều kiện sau:

- Không có hai tiến trình cùng ở trong miền tranh chấp cùng một 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 tranh chấp không được ngăn cản các tiến trình khác vào miền tranh chấp.
- Không có tiến trình nào phải chờ vô hạn để được vào miền tranh chấp.

5.4.2. Các giải pháp

Sử dụng biến cờ hiệu

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 tranh chấp 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 tranh chấp. Nếu lock đang nhận giá trị 1, tiến trình phải chờ bên ngoài miền tranh chấp 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 tranh chấp, và lock=1 khi có một tiến trình đang ở trong miền tranh chấp.

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

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

```
        noncritical-
section();      }
```

Giải pháp này có thể vi phạm điều kiện thứ nhất: hai tiến trình có thể cùng ở trong miền tranh chấp tại một thời điểm. Giả sử một tiến trình nhận thấy lock=0 và chuẩn bị vào miền tranh chấp, 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 tranh chấp và đặt lại lock=1. Sau đó tiến trình thứ nhất được tái kích hoạt, nó gán lock=1 lần nữa rồi vào miền tranh chấp. Như vậy tại thời điểm đó cả hai tiến trình đều ở trong miền tranh chấp.

Sử dụng kiểm tra luân phiên

Đây là một giải pháp đề nghị cho hai tiến trình. Hai tiến trình này sử dụng chung biến turn (phản ánh tiến trình nào được vào miền tranh chấp) được khởi động với giá trị 0. Nếu turn=0, tiến trình A được vào miền tranh chấp. 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 tranh chấp, nó đặt giá trị turn về 1 để cho phép tiến trình B đi vào miền tranh chấp.

Cấu trúc tiến trình A:

```
while (TRUE)
        {
    while (turn !=0) ; //wait
        critical-section();
    turn=1;
        noncritical-
section();      }
```

Cấu trúc tiến trình B:

```
while (TRUE)
        {
    while (turn !=1) ; //wait
        critical-section();
    turn=0;
        noncritical-
section();      }
```

Giải pháp này dựa trên việc thực hiện sự kiểm tra nghiêm ngặt đến lượt tiến trình nào được vào miền tranh chấp. Do đó có thể ngăn chặn tình trạng cả hai tiến trình cùng vào một lúc. Nhưng lại có thể vi phạm điều kiện thứ ba: một tiến trình có thể bị

ngăn chặn vào miền tranh chấp bởi một tiến trình khác không ở trong miền tranh chấp. Giả sử tiến trình B ra khỏi miền tranh chấp rất nhanh chóng. Cả hai tiến trình đều ở ngoài miền tranh chấp và turn=0. Tiến trình A vào miền tranh chấp và ra khỏi nhanh chóng, đặt lại giá trị của turn là 1, rồi lại xử lý đoạn lệnh ngoài miền tranh chấp lần nữa. Sau đó, tiến trình A lại kết thúc nhanh chóng đoạn lệnh ngoài miền tranh chấp của nó và muốn vào lại một lần nữa. Tuy nhiên, lúc này B vẫn còn mãi xử lý đoạn lệnh ngoài miền tranh chấp 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.

Giải pháp của Peterson Peterson đưa ra một giả pháp kết hợp ý tưởng của cả hai giải pháp 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 tranh chấp. Khởi đầu, interesse[0]=interesse[1]=FALSE và giá trị của turn được khởi động là 0 hay 1. Để có thể vào được miền tranh chấp, trước tiên tiến trình Pi đặt giá trị interesse[i]=TRUE (xác định rằng tiến trình muốn vào miền tranh chấp), sau đó đặt turn=j (đề nghị thử tiến trình khác vào miền tranh chấp). Nếu tiến trình Pj không quan tâm đến việc vào miền tranh chấp (interesse[j]=FALSE), thì Pi có thể vào miền tranh chấp, nếu không, Pi phải chờ đến khi interesse[j]=FALSE. Khi tiến trình Pi rời khỏi miền tranh chấp, nó lại đặt lại giá trị cho interesse[i]=FALSE.

Cấu trúc của tiến trình Pi trong giải pháp Peterson:

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

Giải pháp này ngăn chặn được tình trạng mâu thuẫn truy xuất: mỗi tiến trình Pi chỉ có thể vào miền tranh chấp khi interesse[j]=FALSE hoặc turn=i. Nếu cả hai tiến trình đều muốn vào miền tranh chấp 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 tranh chấp.

Cấm ngắt

Phần cứng cho phép tiến trình cấm tất cả các ngắt trước khi vào miền tranh chấp và phục hồi ngắt khi ra khỏi miền tranh chấp. 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 tranh chấp 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 vào được miền tranh chấp.

Chỉ thị TSL (Test-and-Set)

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 tranh chấp, nếu lock=FALSE, tiến trình có thể vào miền tranh chấp.

Cấu trúc một chương trình trong giải pháp TSL:

```
while (TRUE)
  {
      while (Test-and-Setlock(lock));
          critical-section();
      lock = FALSE;
          Noncritical-
section();        }
```

Cũng giống như các giải pháp phần cứng khác, chỉ thị TSL chỉ 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.

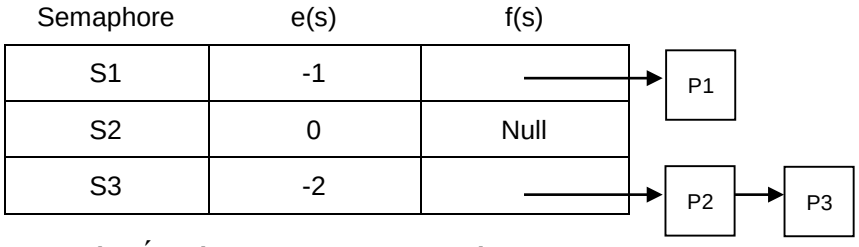
->Tất cả các giải pháp trình bày ở trên đều phải thực hiện một vòng lặp để kiểm tra liệu tiến trình có được phép vào miền tranh chấp. Nếu điều kiện chưa cho phép, tiến trình phải chờ tiếp tục trong vòng lặp kiểm tra này. Các giải pháp buộc tiến trình phải liên tục kiểm tra điều kiện để phát hiện thời điểm thích hợp để vào miền tranh chấp như thế được gọi là 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 hóa là nên tránh các giải pháp "busy waiting".

Semaphore

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

1. Một giá trị nguyên dương e(s).
2. Một hàng đợi f(s) lưu danh sách các tiến trình đang bị khóa (chờ) trên semaphore s.
3. 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 thì tiếp tục xử lý. Ngược lại, nếu e(s) ≤ 0, tiến trình phải chờ cho 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.



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 e(s) <= 0
                {
                                exit(Q, f(s)); // Q là tiến trình
                đang chờ trên s status(Q) = ready;
                   enter(Q, ready-list);
                }
```

Lưu ý cài đặt này có thể đưa đến một giá trị âm cho semaphore, khi đó trị tuyệt đối của semaphore cho biết số tiến trình đang chờ trên semaphore. Điều quan trọng là các thao tác này cần thực hiện một cách không bị phân chia, không bị ngắt nửa 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.

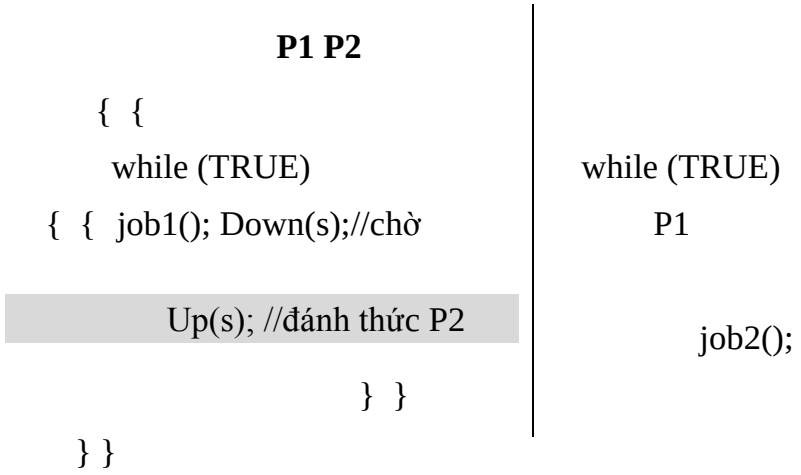
Sử dụng: có thể dùng semaphore để giải quyết vấn đề truy xuất độc quyền hay tổ chức phối hợp giữa các tiến trình.

- Tổ chức truy xuất độc quyền với semaphore: cho phép bảo đảm nhiều tiến trình cùng truy xuất đến miền tranh chấp 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();
                }
```

- Tổ chức đồng bộ hóa với semaphore: ta 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:



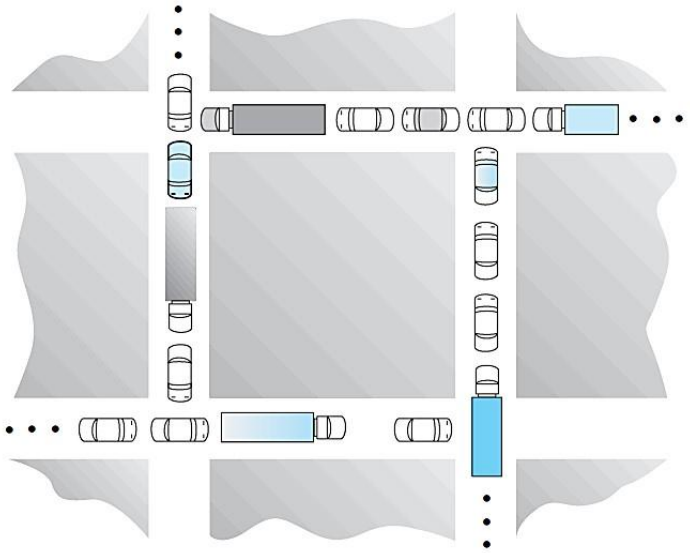
Thảo luận: nhờ có thể thực hiện một cách không thể phân chia, semaphore giải quyết được vấn đề tín hiệu "đánh thức" bị thất lạc. Tuy nhiên, nếu lập trình viên vô tình đặt các primitive Down và Up sai vị trí, thứ tự trong chương trình thì tiến trình có thể bị khóa vĩnh viễn. Vì thế, việc sử dụng đúng 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.

5.5. TẮC NGHẼN (DEADLOCK)

5.5.1. Định nghĩa

Một tập hợp các tiến trình được định nghĩa là 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 đợi 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.



Hình 5.11. Một tình huống giao thông tắc nghẽn

5.5.2. Điều kiện xuất hiện tắc nghẽn

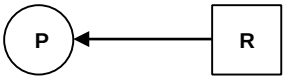
Để xảy ra tắc nghẽn cần có đủ 4 điều kiện sau đây:

1. 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.
2. 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.
3. 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 sử dụng chúng xong.
4. Tồn tại một chu kỳ trong đồ thị cấp phát tài nguyên (Circular wait): có ít nhất hai tiến trình chờ đợi lẫn nhau: tiến trình này chờ được cấp phát tài nguyên đang bị tiến trình kia chiếm giữ và ngược lại.

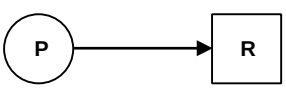
5.5.3. Đồ thị cấp phát tài nguyên

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 biểu thị bằng hình vuông.

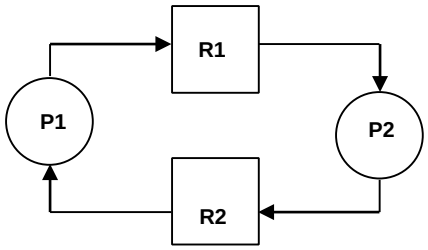
Tiến trình P đang giữ tài nguyên R



Tiến trình P đang yêu cầu tài nguyên R



Một tình huống tắc nghẽn



5.5.4. Các phương pháp xử lý tắc nghẽn

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

- Sử dụng một nghi thức (Protocol) để đảm bảo rằng hệ thống không bao giờ xảy ra tắc nghẽn.
- Cho phép xảy ra tắc nghẽn và tìm cách sửa chữa tắc nghẽn.
- Hoàn toàn bỏ qua việc xử lý tắc nghẽn, xem như hệ thống không bao giờ xảy ra tắc nghẽn.

5.5.5. Tránh tắc nghẽn

Ngăn cản là một mối bận tâm lớn khi sử dụng tài nguyên. Cần phải thực hiện những cơ chế phức tạp để thực hiện ý định này.

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 <P1, P2, …, 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 P_j khác, với j<i.

Một trạng thái an toàn không thể là một 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 các 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]

Bước 1: giả sử có các mảng

int Work = Available;

int Finish[NumProcs] = false;

Bước 2:

Tìm i sao cho:

a) Finish[i] == false

b) Need[i] <=Work[i]

Nếu không có i như thế, đến bước 4

Bước 3:

Work = Work + Allocation[i];

Finish[i] = true;

Đến bước 2

Bước 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:

Tiến trình	**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			
							4	1	2
P3	3	1	4	2	1	1			
P4	4	2	2	0	0	2			

Nếu tiến trình P2 yêu cầu 4 cho R1, 1 cho R3. Hãy cho biết yêu cầu này có thể đáp ứng mà bảo đảm không xảy ra tình trạng deadlock hay không?

Nhận thấy Available[1]=4, Available[3]=2 đủ để thỏa mãn yêu cầu của P2, ta có:

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

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

Tiến trình	Need			Allocation			Available		
	R1	R2	R3	R1	R2	R3	R1	R2	R3
P1	0	0	0	0	0	0	7	2	3
P2	0	0	0	0	0	0			
P3	1	0	3	2	1	1			
P4	4	2	0	0	0	2			

Tiến trình	Need			Allocation			Available		
	R1	R2	R3	R1	R2	R3	R1	R2	R3
P1	0	0	0	0	0	0	9	3	4
P2	0	0	0	0	0	0			
P3	0	0	0	0	0	0			

P4	4	2	0	0	0	2			

Tiến trình	Need			Allocation			Available		
	R1	R2	R3	R1	R2	R3	R1	R2	R3
P1	0	0	0	0	0	0	9	3	6
P2	0	0	0	0	0	0			
P3	0	0	0	0	0	0			
P4	0	0	0	0	0	0			

-> Kết luận: Trạng thái kết quả là an toàn, có thể cấp phát.

5.6. QUẢN LÝ BỘ NHỚ

5.6.1. Giới thiệu

Bộ nhớ chính là thiết bị lưu trữ duy nhất thông qua đó CPU trao đổi dữ liệu 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".

Hệ điều hành có trách nhiệm cấp phát vùng nhớ cho các tiến trình có yêu cầu. Để thực hiện tốt nhiệm vụ này, hệ điều hành cần phải xem xét nhiều khía cạnh:

- Sự tương ứng giữa địa chỉ logic và địa chỉ vật lý (physic): làm cách nào để chuyển đổi một địa chỉ tượng trưng (symbolic) trong chương trình thành một địa chỉ thực trong bộ nhớ chính?
- Quản lý bộ nhớ vật lý: làm cách nào để mở rộng bộ nhớ có sẵn nhằm lưu trữ được nhiều tiến trình đồng thời?
- Chia sẻ thông tin: làm thế nào để cho phép hai tiến trình có thể chia sẻ thông tin trong bộ nhớ?
- Bảo vệ: làm thế nào để ngăn chặn các tiến trình xâm phạm đến vùng nhớ được cấp phát cho tiến trình khác?

Các địa chỉ trong chương trình nguồn là địa chỉ tượng trưng. Vì thế, một chương trình phải trải qua nhiều giai đoạn xử lý để chuyển đổi các địa chỉ này thành các địa chỉ tuyệt đối trong bộ nhớ chính.

Có thể thực hiện kết buộc các chỉ thị và dữ liệu với các địa chỉ bộ nhớ vào một trong những thời điểm sau:

- Thời điểm biên dịch: nếu tại thời điểm biên dịch, có thể biết vị trí mà tiến trình sẽ thường trú trong bộ nhớ, trình biên dịch có thể phát sinh ngay mã với các địa chỉ tuyệt đối. Tuy nhiên, nếu về sau có sự thay đổi vị trí thường trú lúc đầu của chương trình, cần phải biên dịch lại chương trình.
- Thời điểm nạp: nếu tại thời điểm biên dịch, chưa thể biết vị trí mà tiến trình sẽ thường trú trong bộ nhớ, trình biên dịch cần phát sinh mã tương đối (translatable). Sự liên kết địa chỉ được trì hoãn đến thời điểm chương trình được nạp vào bộ nhớ, lúc này các địa chỉ tương đối sẽ được chuyển thành địa chỉ tuyệt đối do đã biết vị trí bắt đầu lưu trữ tiến trình. Khi có sự thay đổi

  vị trí lưu trữ, chỉ cần nạp lại chương trình để tính toán lại các địa chỉ tuyệt đối mà không cần biên dịch lại.
- Thời điểm xử lý: nếu có nhu cầu di chuyển tiến trình từ vùng nhớ này sang vùng nhớ khác trong quá trình tiến trình xử lý, thì thời điểm kết buộc địa chỉ phải trì hoãn đến tận thời điểm xử lý. Để thực hiện kết buộc địa chỉ vào thời điểm xử lý cần sử dụng cơ chế phần cứng đặc biệt.

Một trong những hướng tiếp cận trung tâm nhằm tổ chức quản lý bộ nhớ một cách hiệu quả 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 tất cả các 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 được.
- Không gian địa chỉ: là tập hợp tất cả các địa chỉ ảo phát sinh bởi một chương trình, nói cách khác là vùng địa chỉ mà chương trình có thể sử dụng được.
- 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 bộ nhớ chỉ được xác định khi thực hiện truy xuất đến dữ liệu.

5.6.2. Phân trang (paging)

Ý tưởng:

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

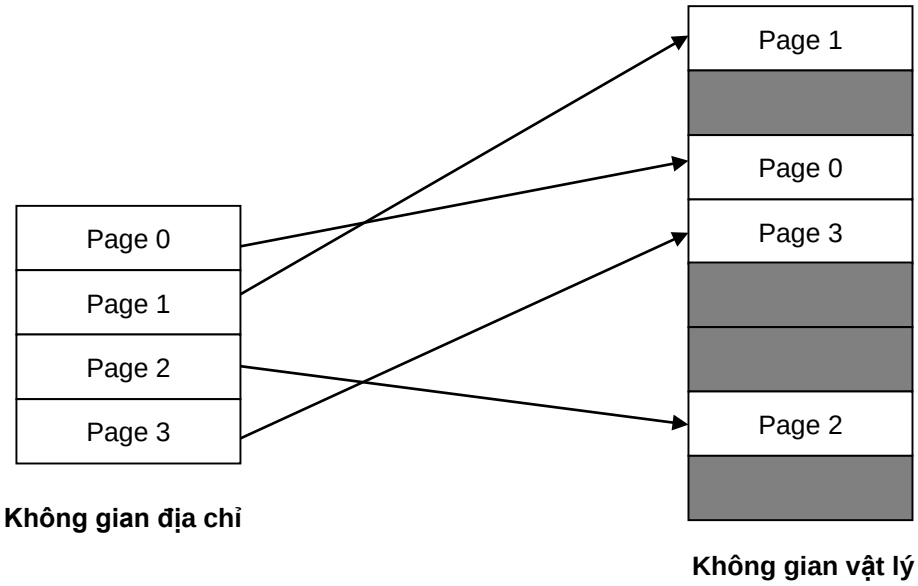
Cơ chế MMU trong kỹ thuật phân trang

Cơ chế phần cứng hỗ trợ thực hiện chuyển đổi địa chỉ trong cơ chế phân trang là bảng trang (pages table). Mỗi phần tử trong bảng trang cho biết các 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).

Chuyển đổi địa chỉ

Mỗi địa chỉ phát sinh bởi CPU được chia thành hai phần:

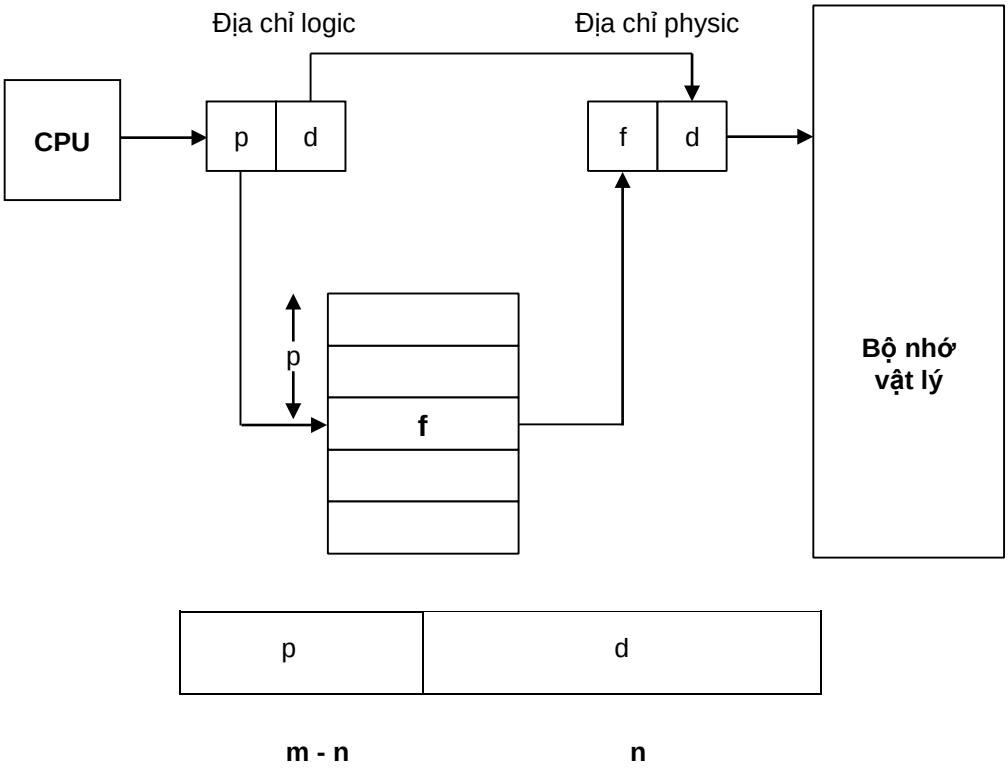
- Số hiệu trang (p): sử dụng như chỉ mục đến phần tử tương ứng trong bảng trang.
- Địa chỉ tương đối trong trang (d): kết hợp với địa chỉ bắt đầu của trang để tạo ra địa chỉ vật lý mà trình quản lý bộ nhớ sử dụng.



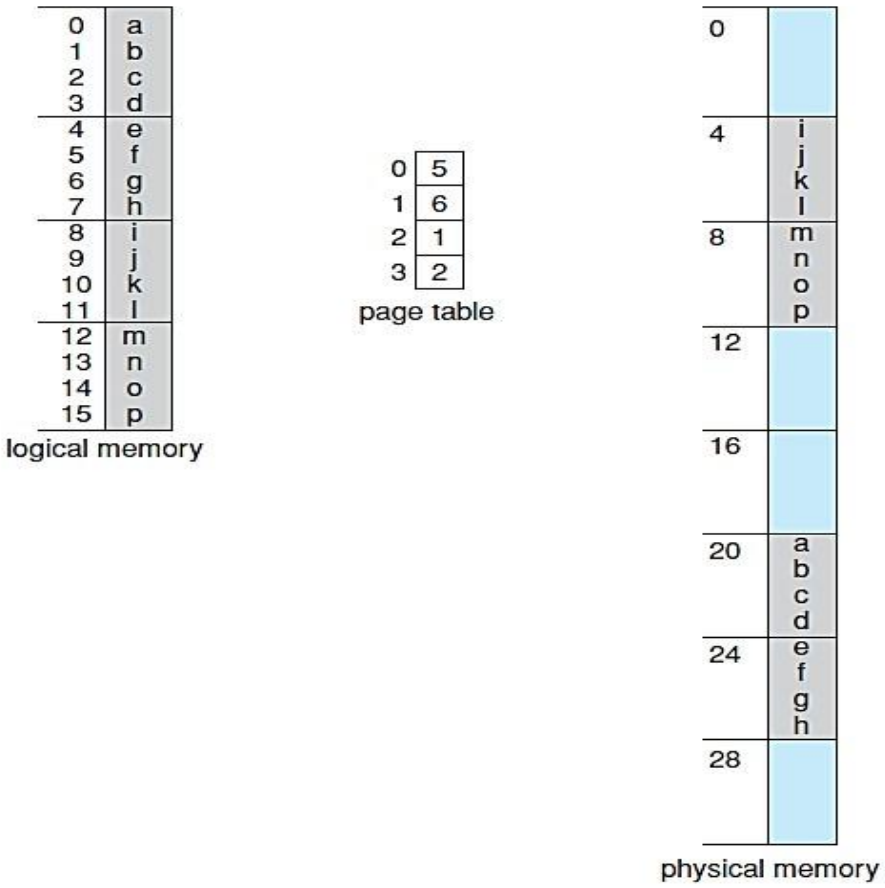
Hình 5.11 . Mô hình bộ nhớ phân trang

Kích thước của trang do phần cứng quy 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 2m và kích thước trang là 2n 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.



Hình 5.12. Cơ chế phần cứng hỗ trợ phân trang



Hình 5.13. Ví dụ phân trang với bộ nhớ 32 byte, kích thước trang là 4 byte

Cài đặt bảng trang

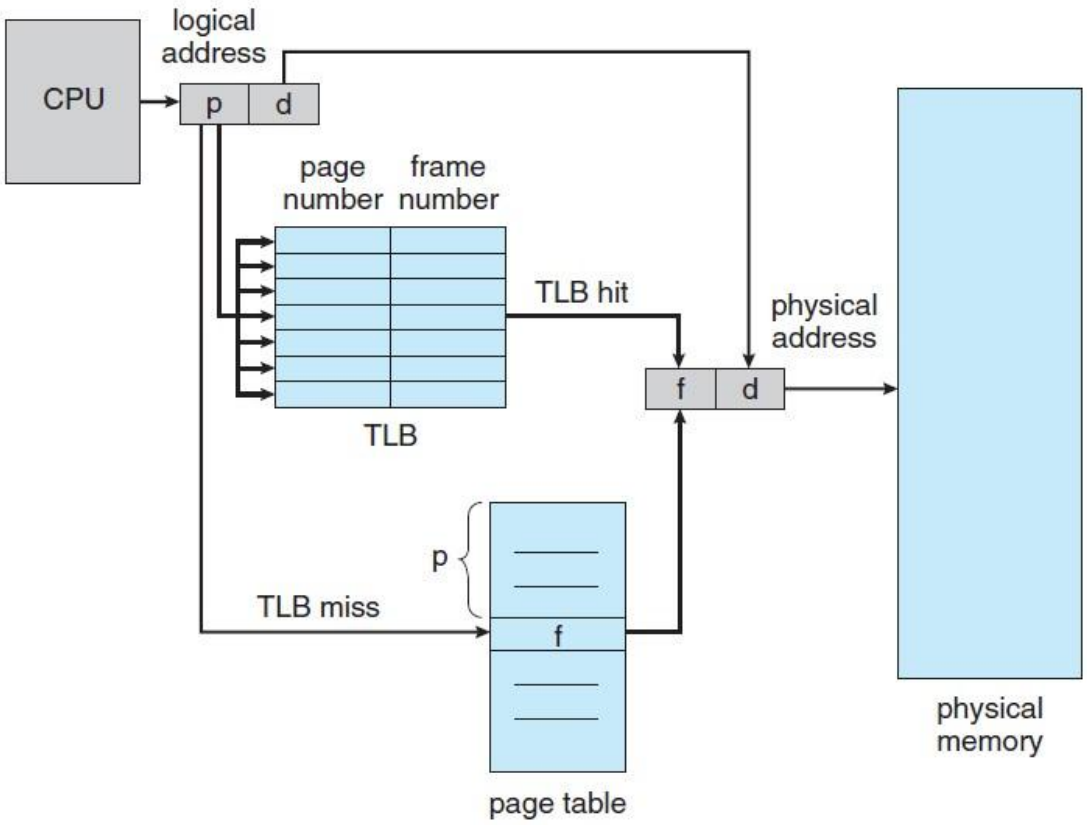
Trong trường hợp đơn giản nhất, bảng trang là một tập các thanh ghi được sử dụng để cài đặt bảng trang. Tuy nhiên việc sử dụng thanh ghi chỉ phù hợp với các bảng trang có kích thước nhỏ, nếu bảng trang có kích thước lớn, nó phải được lưu trữ trong bộ nhớ chính và sử dụng một thanh ghi để lưu địa chỉ bắt đầu lưu trữ bảng trang (PTBR).

Theo cách tổ chức này, mỗi lần 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.

Có thể né tránh bớt việc truy xuất bộ nhớ hai lần bằng cách 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 (TBLs). Mỗi thanh ghi trong bộ nhớ kết hợp gồm một từ khóa và một giá trị, khi đưa đến bộ nhớ kết hợp một đối tượng cần tìm, đối tượng này sẽ được so sánh cùng lúc với các từ khóa trong bộ nhớ kết hợp để tìm ra phần tử tương ứng. 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, TBLs được sử dụng để lưu trữ các trang bộ nhớ được truy cập gần hiện tại nhất. Khi CPU phát sinh một địa chỉ, số hiệu trang

của địa chỉ sẽ được so sánh với các phần tử trong TBLs, nếu có trang tương ứng trong TBLs thì sẽ xác định được ngay số hiệu khung trang tương ứng, nếu không mới cần thực hiện thao tác tìm kiếm trong bảng trang.



Hình 5.14. Phần cứng hỗ trợ phân trang sử dụng TBLs

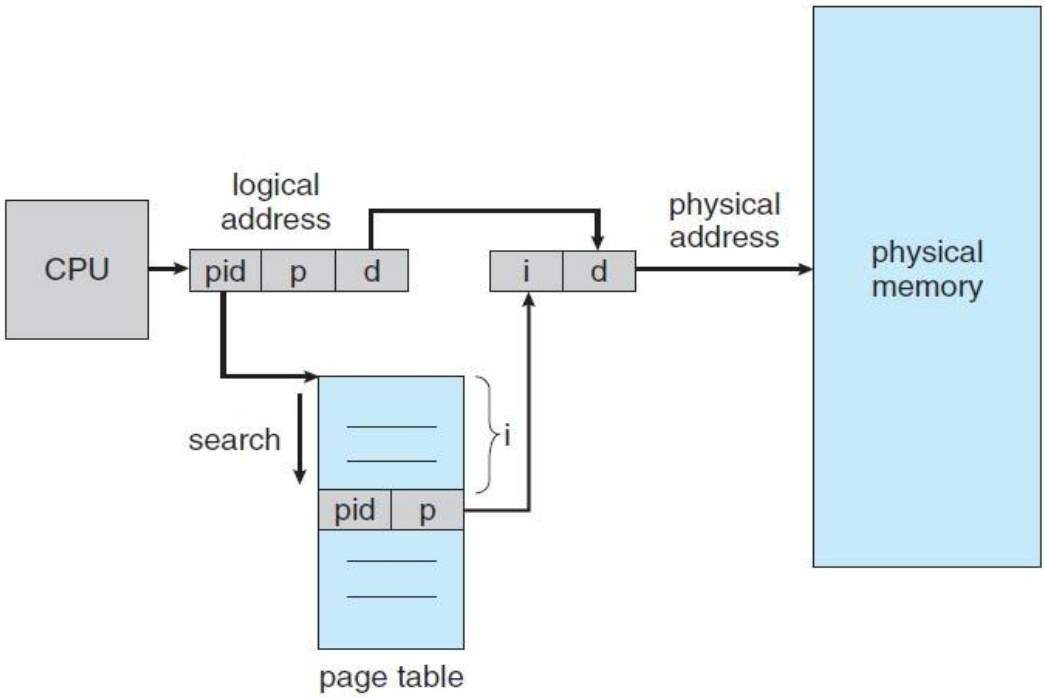
Tổ chức bảng trang

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 quản lý một không gian địa chỉ có dung lượng quá lớn (2^{32}, 2^{64}): trong các hệ thống như thế, bản thân bảng trang đòi hỏi một vùng nhớ quá lớn! Có hai giải pháp cho vấn đề này:

- Phân trang đa cấp: phân chia bảng trang thành các phần nhỏ, bản thân bảng trang cũng sẽ được phân trang.
- 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 sở hữu trang. Mỗi địa chỉ ảo khi đó là một bộ ba <idp, p, d>, trong đó:
    - idp là định danh của tiến trình.
    - p là số hiệu trang.

- d là địa chỉ tương đối trong trang.

Mỗi phần tử trong bảng trang nghịch đảo là một cặp <idp, 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.



Hình 5.15. Bảng trang nghịch đảo

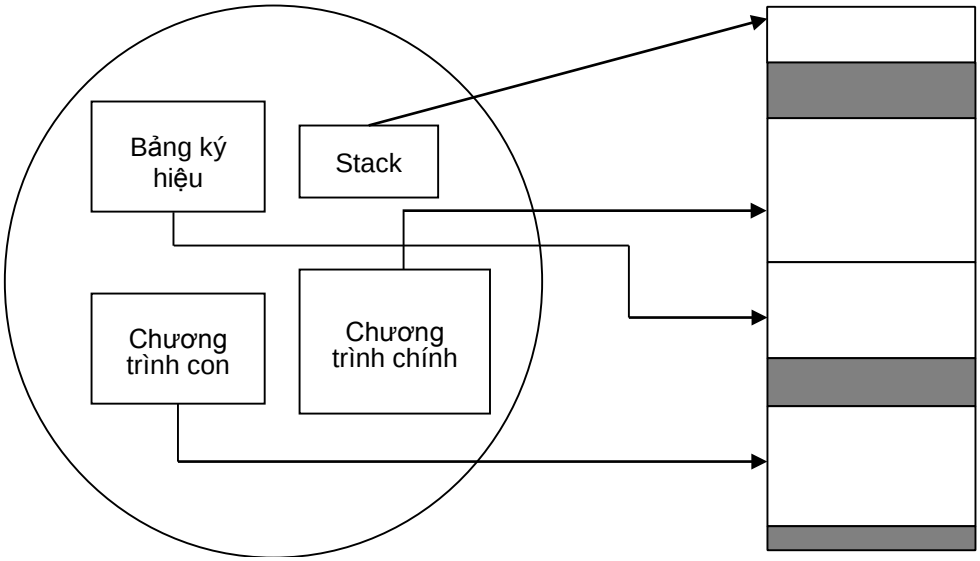
5.6.3. Phân đoạn

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 có một độ dài. Người dùng sẽ thiết lập mỗi địa chỉ với hai giá trị: <số hiệu phân đoạn, offset>

Cơ chế MMU

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.

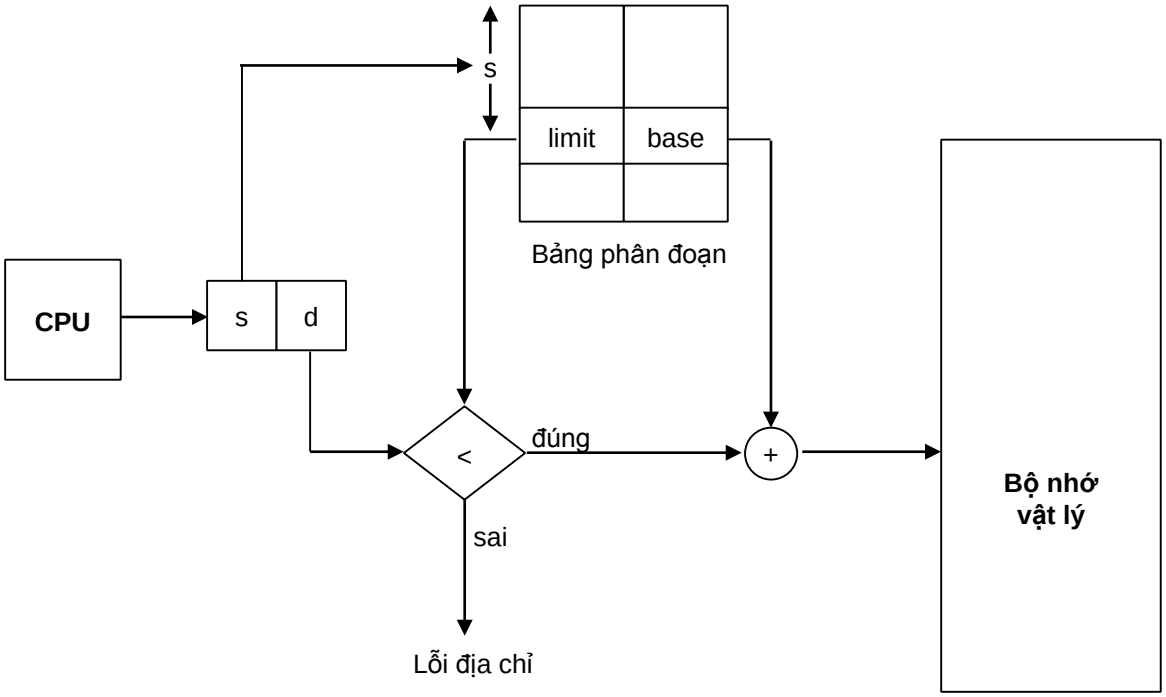


Hình 5.16 Một mô hình phân đoạn bộ nhớ

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 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.



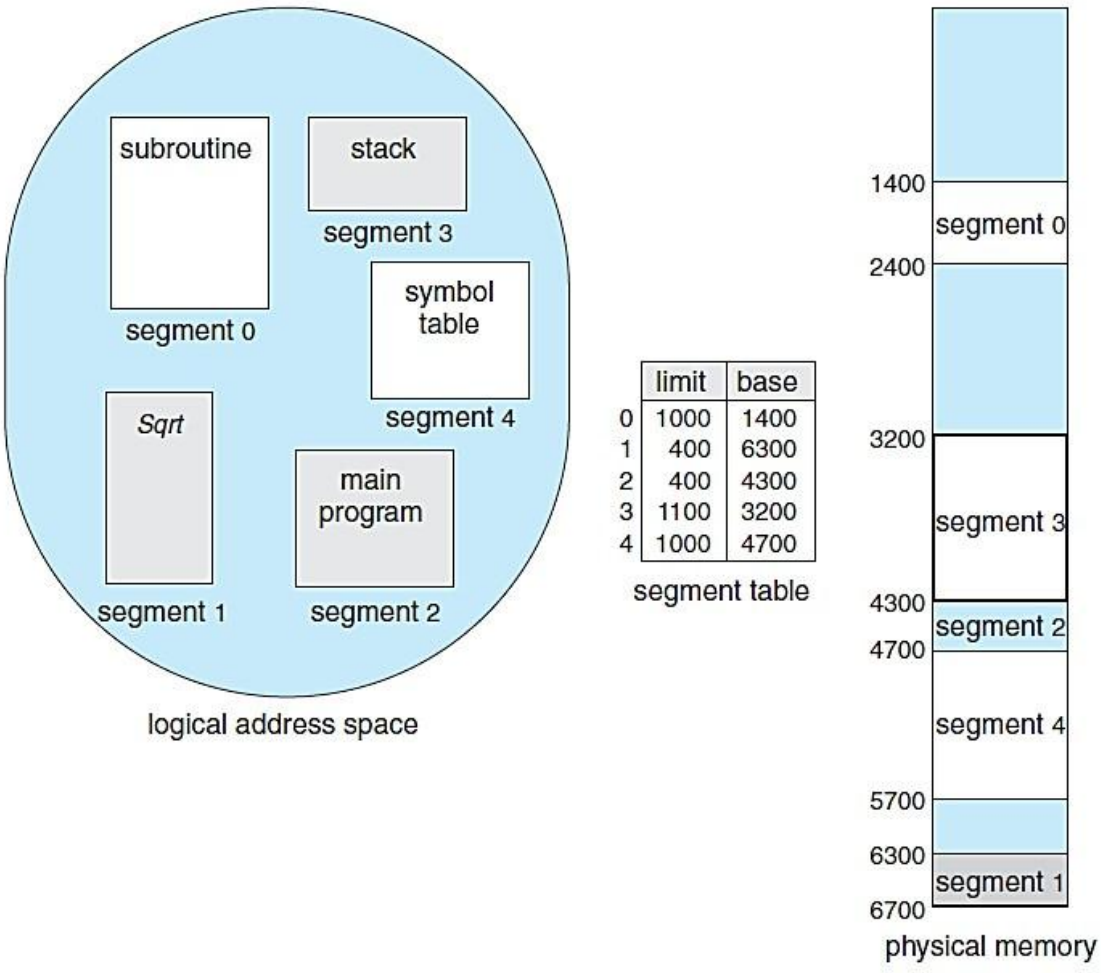
Hình 5.17 Cơ chế phần cứng hỗ trợ phân đoạn

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

Cũng như đối với kỹ thuật phân trang, có thể nâng tốc độ truy xuất bộ nhớ bằng cách sử dụng TBLs để lưu trữ các phần tử trong bảng trang được truy cập gần nhất.



Hình 5.18. Ví dụ một trường hợp phân đoạn

5.6.4. Phân trang kết hợp phân đoạn

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 trang cần thiết để chứa đủ các phân đoạn của tiến trình.

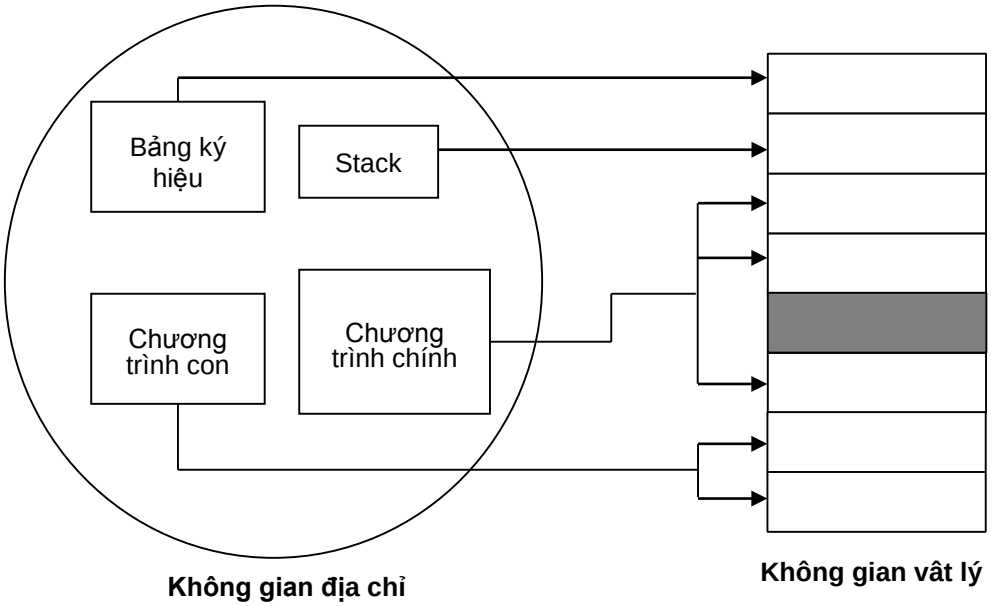
Cơ chế MMU

Để hỗ trợ kỹ thuật phân đoạn, cần phải có một bảng phân đoạn, nhưng giờ đây mỗi phân đoạn cần có một bảng trang riêng 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.
- 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 5.19. Mô hình phân đoạn kết hợp phân trang

Tất cả các mô hình tổ chức bộ nhớ ở trên 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 duy trì nhiều tiến trình cùng lúc trong bộ nhớ, và như vậy khó nâng cao mức độ đa chương của hệ thống.*

5.6.5. Bộ nhớ ảo (virtual memory)

Giới thiệu

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 mỗi thời điểm chỉ có một chỉ thị của tiến trình được xử lý. Ví dụ: các chương trình đều có 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 thể hiện ý tưởng của giải pháp này là 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).

Khái niệm

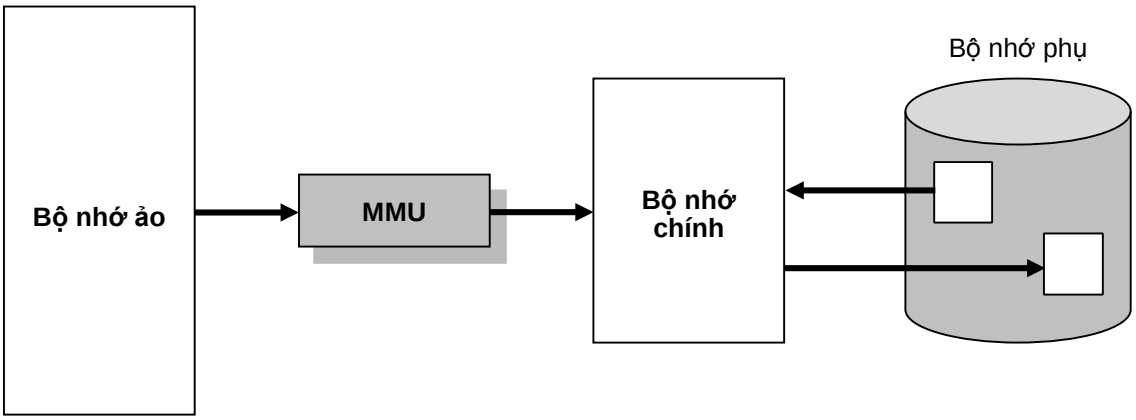
Bộ nhớ ảo là một kỹ thuật cho phép xử lý một tiến trình không được nạp toàn bộ vào bộ nhớ vật lý. Bộ nhớ ảo mô hình hóa bộ nhớ như một bảng lưu trữ rất lớn và đồng nhất, tách biệt hẳn khái niệm không gian địa chỉ và không gian vật lý. Người sử dụng chỉ nhìn thấy và làm việc trong không gian địa chỉ ảo, 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 để 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 overlay.



Hình 5.20. Mô hình bộ nhớ ảo

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.

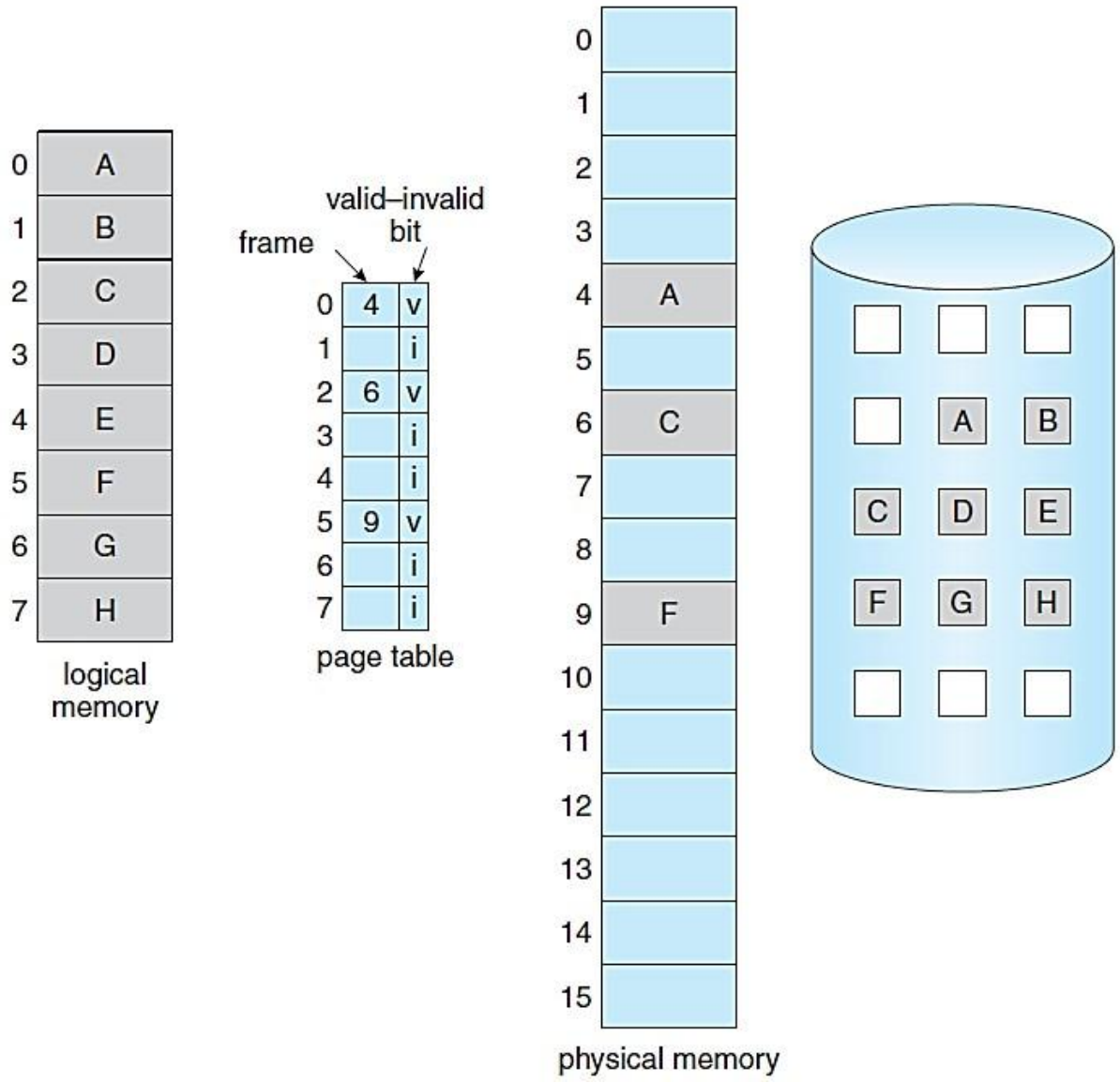
Phân trang theo yêu cầu (demand paging)

Một hệ thống phân trang theo yêu cầu là hệ thống sử dụng kỹ thuật 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 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.



Hình 5.21. Bảng trang của một trường hợp phân trang theo yêu cầu

Cơ chế MMU

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.

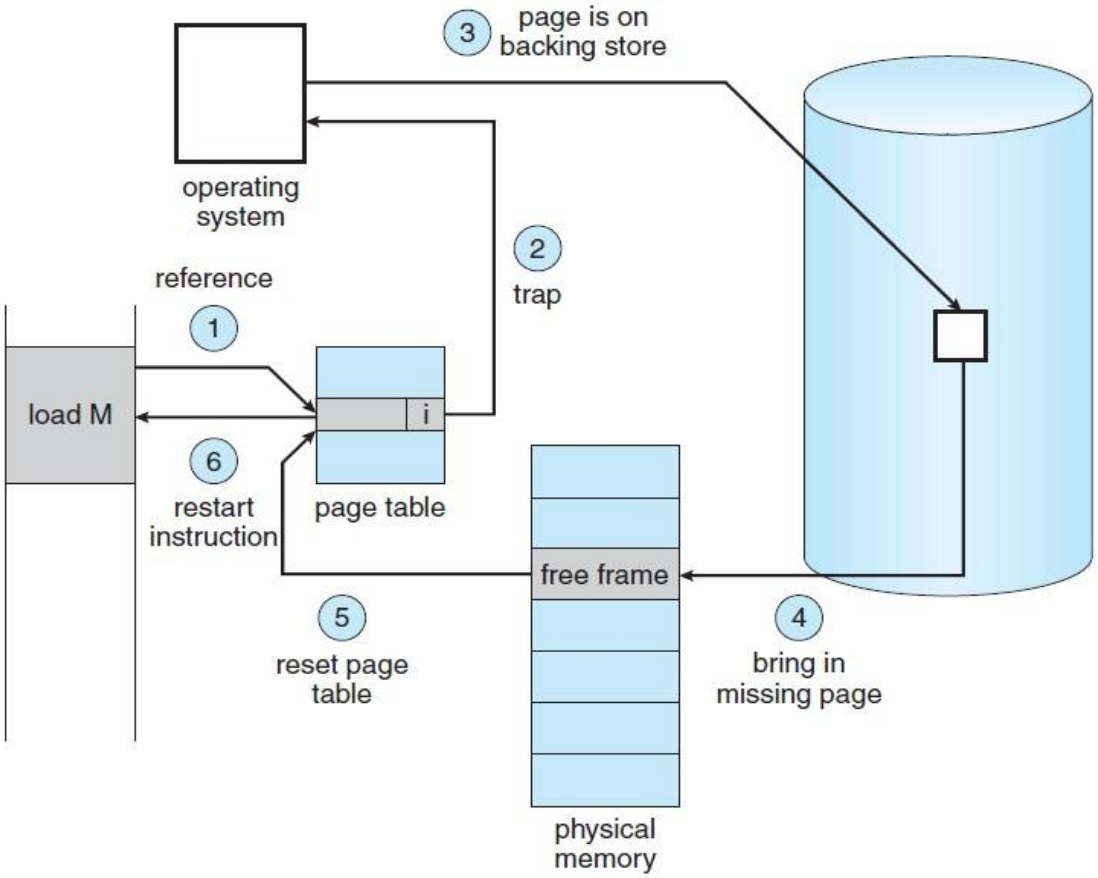
- Bảng trang: cấu trúc bảng trang phải cho phép phản ánh tình trạng của một trang là đang nằm trong bộ nhớ chính hay bộ nhớ phụ.
- Bộ nhớ phụ: bộ nhớ phụ lưu trữ những trang không được nạp vào bộ nhớ chính. Bộ nhớ phụ thường được sử dụng là đĩa và là vùng không gian đĩa dùng để lưu trữ tạm các trang 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:

1. Kiểm tra truy xuất đến bộ nhớ có hợp lệ hay không?
2. Nếu truy xuất bất hợp lệ thì kết thúc tiến trình. Ngược lại thì đến bước 3.
3. Tìm vị trí chứa trang muốn truy xuất trên đĩa.
4. Tìm một khung trang trống trong bộ nhớ chính
   a. Nếu tìm thấy: đến bước 5
   b. 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.
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.
6. Tái kích hoạt tiến trình người sử dụng.



Hình 5.22. Sơ đồ các bước xử lý lỗi trang

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 thì không cần lưu trữ trang trở lại đĩa.

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 xảy ra ít lỗi trang nhất. Có thể đánh giá hiệu quả của một thuật toán bằng cách xử lý trên một chuỗi các địa chỉ cần truy xuất và tính toán số lỗi trang phát sinh.

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

0100, 0423, 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 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 họa 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

Thuật toán FIFO

Giải thuật: 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: ký hiệu * : có lỗi trang

Thảo luận:

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

Thuật toán thay thế trang FIFO dễ hiểu, dễ cài đặt. Tuy nhiên khi thực hiện không phải lúc nào cũng có kết quả tốt: trang được chọn để thay thế có thể là trang chứa nhiều dữ liệu cần thiết, thường xuyên được sử dụng nên được nạp sớm. Do vậy, khi di 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.

Thuật toán tối ưu

Giải thuật: 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 đảm bảo 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.

Thuật toán LRU (Least-Recently-Used)

Giải thuật: 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: 1. Sử dụng bộ đếm

- Thêm vào cấu trúc của mỗi phần tử trong bảng trang một trường ghi nhận thời điểm truy xuất mới nhất, và thêm vào cấu trúc của CPU một bộ đếm.
- Mỗi lần có sự truy xuất bộ nhớ, giá trị của counter tăng lên 1.
- Mỗi lần thực hiện truy xuất đến một trang, giá trị của counter được ghi nhận vào trường thời điểm truy xuất mới nhất của phần tử tương ứng với bảng trang trong bảng trang.
- Thay thế trang có giá trị trường thời điểm truy xuất mới nhất là nhỏ nhất.

2. Sử dụng Stack

- Tổ chức một stack lưu trữ các số hiệu trang.
- Mỗi khi thực hiện một truy xuất đến một trang, số hiệu của trang sẽ được xóa khỏi vị trí hiện hành trong stack và đưa lên đầu stack.
- Trang ở đỉnh stack là trang được truy xuất gần nhất và trang ở đáy stack là trang lâu nhất chưa được sử dụng.

5.6.6. Cấp phát khung và thay thế trang

Với mỗi tiến trình, cần phải cấp phát một số lượng khung trang tối thiểu nào đó để tiến trình có thể hoạt động. Số khung trang tối thiểu này được quy định bởi kiến trúc của máy tính.

Ví dụ: với máy IBM 370, để lệnh MOVE có thể thực hiện thì tối thiểu phải có 2 trang: một trang nguồn (from), một trang đích (to). Khi một lỗi trang xảy ra trước khi chỉ thị hiện hành hoàn tất, chỉ thị đó cần được tái khởi động, lúc đó cần phải có đủ các khung trang để nạp tất cả các trang mà một chỉ thị cần sử dụng.

Số khung trang tối thiểu được quy định bởi kiến trúc máy tính, trong khi số khung trang tối đa được xác định bởi dung lượng bộ nhớ vật lý có thể sử dụng.

Cấp phát số khung trang

Có ba cách cấp phát số lượng khung trang là:

- Cấp phát ngang bằng (Equal Allocation)
- Cấp phát theo tỷ lệ kích thước (Proportional Allocation)

- Cấp phát theo độ ưu tiên (Priority Allocation)

a) Cấp phát ngang bằng

Nếu có m khung trang và n tiến trình, mỗi tiến trình được cấp m/n khung trang. Phương pháp này đơn giản nhưng không hiệu quả.

b) Cấp phát theo tỷ lệ kích thước

Tùy vào kích thước của tiến trình để cấp phát số khung trang. Gọi:

si là kích thước của tiến trình pi

$S = \sum s_i$ là tổng kích thước của tất cả tiến trình m

là số lượng khung trang có thể sử dụng

Khi đó, tiến trình pi sẽ được cấp phát ai khung trang. Với:

$a_i = \frac{s_i}{S} \times m$

Ví dụ: có 2 tiến trình, tiến trình 1 có kích thước 10K, tiến trình 2 là 127K, hệ thống có 62 khung trang trống. Khi đó, có thể cấp cho các tiến trình là:

- Tiến trình 1: 10/137 x 62 ~ 4 khung

- Tiến trình 2: 127/137 x 62 ~ 57 khung

c) Cấp phát theo độ ưu tiên

Số lượng khung trang cấp cho tiến trình phụ thuộc vào độ ưu tiên của tiến trình. Tiến trình có độ ưu tiên cao sẽ được cấp nhiều khung hơn để tăng tốc độ thực hiện.

Thay thế trang

Có hai cách để thay thế trang:

- Thay thế toàn cục (Global Replacement)
- Thay thế cục bộ (Local Replacement)

a) Thay thế toàn cục

Chọn trang "nạn nhân" từ tập tất cả các khung trang trong hệ thống, khung trang đó có thể đang được cấp phát cho một tiến trình khác.

Ví dụ: có thể chọn trang của tiến trình có độ ưu tiên thấp hơn làm trang nạn nhân.

Thuật toán thay thế toàn cục cho phép hệ thống có nhiều khả năng lựa chọn hơn, số khung trang cấp cho một tiến trình có thể thay đổi, nhưng các tiến trình không thể kiểm soát được tỷ lệ phát sinh lỗi trang của mình.

b) Thay thế cục bộ

Chỉ chọn trang thay thế trong tập các khung trang được cấp cho tiến trình phát sinh lỗi trang, khi đó số khung trang cấp cho một tiến trình sẽ không thay đổi.

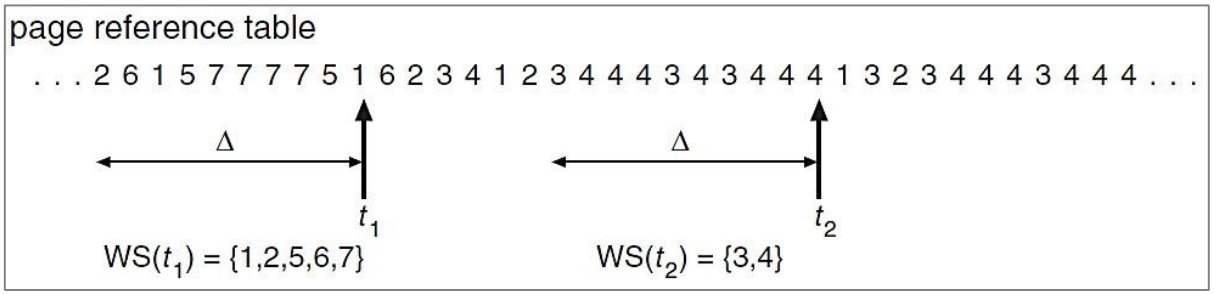
Trì trệ hệ thống (Thrashing)

Khi tiến trình không có đủ các khung trang để chứa những trang cần thiết cho việc xử lý, nó sẽ thường xuyên phát sinh các lỗi trang. Vì thế phải dùng đến rất nhiều thời gian sử dụng CPU để thực hiện thay thế trang. Hệ điều hành thấy hiệu quả sử dụng CPU thấp sẽ tăng mức độ đa chương dẫn đến trì trệ toàn bộ hệ thống.

Để ngăn cản tình trạng trì trệ này xảy ra, cần phải cấp cho tiến trình đủ các khung trang cần thiết để hoạt động. Vấn đề là làm sao biết được mỗi tiến trình cần bao nhiêu trang?

a) Mô hình tập làm việc (Working-Set Model)

Tập làm việc của tiến trình tại thời điểm t là tập các trang được tiến trình truy xuất đến trong Δ lần truy cập cuối cùng tính tại thời điểm t



Hình 5.23. Mô hình tập làm việc

Gọi:

WSSi(Δ, t) là số phần tử của tập working-set của tiến trình Pi tại thời điểm t

m là số khung trang trống

ΣWSSi là tổng số khung trang yêu cầu cho toàn hệ thống

Hệ điều hành giám sát working-set của mỗi tiến trình Pi và tại thời điểm t sẽ cấp phát cho tiến trình Pi số khung trang bằng với số phần tử trong tập làm việc $(WSS_i)(\Delta, t - 1)$

Nếu tổng số khung trang yêu cầu của các tiến trình trong hệ thống vượt quá các khung trang có thể sử dụng (D>m), thì sẽ xảy ra tình trạng hệ thống trì trệ. Khi đó hệ điều hành chọn một tiến trình để tạm dừng, giải phóng các khung trang của tiến trình đã chọn để các tiến trình khác có đủ khung trang hoàn tất công việc.

b) Cấu trúc chương trình

Số lỗi trang có khi phụ thuộc vào ngôn ngữ lập trình, nên khi lập trình ta cần chú ý để chương trình có thể thực hiện nhanh hơn.

Ví dụ: xét chương trình sau int

a[128][128]; for (i=0; i<128;

i++) for (j=0; j<128;

j++) a[i][j]=0;

Giả sử trang có kích thước 128 byte và tiến trình được cấp 2 khung trang: khung trang thứ nhất chứa mã tiến trình, khung trang còn lại được khởi động ở trạng thái trống. trong Pascal, C thì mảng lưu theo hàng, mỗi hàng chiếm 1 trang bộ nhớ nên tổng số lỗi trang phát sinh là 128. Nhưng trong Fortran thì mảng lưu theo cột, do đó số lỗi trang sẽ là 128 x 128 = 16384 .

TÓM TẮT CHƯƠNG

- Tiến trình là một chương trình đang hoạt động.
- Để sử dụng hiệu quả CPU, sự đa chương cần được đưa vào hệ thống.

Sự đa chương được tổ chức bằng cách lưu trữ nhiều tiến trình trong bộ nhớ tại một thời điểm và điều phối CPU qua lại giữa các tiến trình trong hệ thống.

- Trong suốt chu trình sống, tiến trình chuyển đổi qua lại giữa các trạng thái ready, running và blocked.
- Mô hình đa thread cho phép mỗi tiến trình có thể tiến hành nhiều dòng xử lý đồng thời trong cùng một không gian địa chỉ nhằm thực hiện tác vụ hiệu quả hơn trong một số trường hợp.
- Bộ điều phối (scheduler) của hệ điều hành chịu trách nhiệm áp dụng một giải thuật điều phối thích hợp để chọn tiến trình thích hợp được sử dụng CPU, và bộ phân phối (dispatcher) sẽ chuyển giao CPU cho tiến trình này.
- Các giải thuật điều phối thông dụng: FIFO, Round Robin, độ ưu tiên, SJF, đa cấp độ.
- Các giải pháp đồng bộ hóa do lập trình viên xây dựng không được ưa chuộng vì phải tiêu thụ CPU trong thời gian chờ vào miền tranh chấp (“busy waiting”) và khó mở rộng. Thay vào đó, lập trình viên có thể sử dụng các cơ chế đồng bộ do hệ điều hành hay trình biên dịch trợ giúp như semaphore, monitor, trao đổi thông điệp.
- Tắc nghẽn là tình trạng xảy ra trong một tập các tiến trình nếu có hai hay nhiều hơn các tiến trình chờ đợi vô hạn một sự kiện chỉ có thể được phát sinh bởi một tiến trình cũng đang chờ khác trong tập các tiến trình này.
- Có 3 hướng tiếp cận chính trong xử lý tắc nghẽn: o Phòng tránh tắc nghẽn: tuân thủ một vài nghi thức bảo đảm hệ thống không bao giờ lâm vào trạng thái tắc nghẽn. o Phát hiện tắc nghẽn: khi có tắc nghẽn xảy ra, phát hiện các tiến trình liên quan và tìm cách phục hồi. o Bỏ qua tắc nghẽn: xem như hệ thống không bao giờ lâm vào trạng thái tắc nghẽn.

BÀI TẬP

1) Nhiều hệ điều hành không cho phép xử lý đồng hành. Thảo luận về các phức tạp phát sinh khi hệ điều hành cho phép đa chương?

2) Xét tập các tiến trình sau (với thời gian yêu cầu CPU và độ ưu tiên kèm theo):

Tiến trình	**Thời điểm vào RL**	**Thời gian CPU**	**Độ ưu tiên**
P1	0	10	3
P2	1	1	1
P3	2	2	3
P4	3	1	4
P5	4	5	2

   a. Cho biết kết quả điều phối hoạt động của các tiến trình trên theo thuật toán FIFO; SJF; điều phối theo độ ưu tiên (1 > 2 > 3 >…) preemptive và non-preemptive; Round Robin với Quantum=2.

   b. Cho biết thời gian lưu lại trong hệ thống (Turn arround time) của từng tiến trình trong từng thuật toán điều phối ở câu a.

   c. Cho biết thời gian chờ trong hệ thống (waiting time) của từng tiến trình trong từng thuật toán điều phối ở câu a.

   d. Thuật toán điều phối nào trong các thuật toán ở câu a cho thời gian chờ là cực tiểu?

3) Phân biệt sự khác nhau trong cách tiếp cận để ưu tiên cho tiến trình ngắn trong các thuật toán điều phối sau:

   a. FIFO

   b. Round Robin

   c. Điều phối với độ ưu tiên đa cấp.

4) Cho biết hai ưu điểm chính của mô hình đa tiểu trình so với đa tiến trình. Mô tả một ứng dụng thích hợp với mô hình đa tiểu trình và một ứng dụng khác không thích hợp.

5) Mô tả các xử lý hệ điều hành phải thực hiện khi chuyển đổi ngữ cảnh giữa: a. Các tiến trình

   b. Các tiểu trình

6) Xác định thời lượng quantum q là một nhiệm vụ khó khăn. Giả sử chi phí trung bình cho một lần chuyển đổi ngữ cảnh là s, và thời gian trung bình một

tiến trình hướng nhập xuất sử dụng CPU trước khi phát sinh một yêu cầu nhập xuất là t (t >> s). Thảo luận các tác động đến sự thực hiện của hệ thống khi chọn lựa q theo các quy tắc sau:

a. q bất định

b. q lớn hơn 0 một ít.

c. q = s

d. s < q < t

e. q = t

f. q > t

7) Giả sử một hệ điều hành áp dụng giải thuật điều phối multilevel feedback với 5 mức ưu tiên (giảm dần). Thời lượng quantum dành cho hàng đợi cấp 1 là 0.5s. Mỗi hàng đợi cấp thấp hơn sẽ có thời lượng quantum dài gấp đôi hàng đợi với mức ưu tiên cao hơn nó. Một tiến trình khi vào hệ thống sẽ được đưa vào hàng đợi mức cao nhất, và chuyển dần xuống các hàng đợi bên dưới sau mỗi lượt sử dụng CPU. Một tiến trình chỉ có thể bị thu hồi CPU khi đã sử dụng hết thời lượng quantum dành cho nó. Hệ thống có thể thực hiện các tác vụ xử lý theo lô hoặc tương tác và mỗi tác vụ lại có thể hướng xử lý hoặc hướng nhập xuất.

a. Giải thích tại sao hệ thống này hoạt động không hiệu quả?

b. Cần phải thay đổi (tối thiểu) như thế nào để hệ thống điều phối các tác vụ với những bản chất khác biệt như thế tốt hơn?

8) Xét hai tiến trình xử lý đoạn chương trình sau:

process P1 {A1 ; A2}

process P2 {B1 ; B2}

Đồng bộ hóa hoạt động của hai tiến trình này sao cho cả A1 và B1 đều hoàn tất trước khi A2 hay B2 bắt đầu.

9) Cho các tiến trình xử lý đoạn chương trình sau:

process P1 {for (i=1; i<=100; i++) Ai;}

process P2 {for (j=1; j<=100; j++) Bj;}

Đồng bộ hóa hoạt động của hai tiến trình này sao cho cả với k bất kỳ (2<=k<=100), Ak chỉ có thể bắt đầu khi B(k-1) đã kết thúc, và Bk chỉ có thể bắt đầu khi A(k-1) đã kết thúc.

10) Xét trạng thái hệ thống:

	Max			**Allocation**			**Available**		
	R1	R2	R3	R1	R2	R3	R1	R2	R3

P1	3	2	2	1	1	0	4	1	2
P2	7	1	3	2	1	1			
P3	4	1	4	2	1	1			
P4	5	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?