TRƯỜNG ĐẠI HỌC KIẾN TRÚC HÀ NỘI
KHOA CÔNG NGHỆ THÔNG TIN

BÀI GIẢNG

HỆ ĐIỀU HÀNH

GIẢNG VIÊN THỰC HIỆN: ThS. Nguyễn Bá Quảng
ThS. Nguyễn Huy Thịnh
ThS. Bùi Hải Phong

Số tín chỉ: 03

Số tiết giảng: 45 tiết

Lý thuyết: 45 tiết

Hà Nội, 6/ 2017

MỤC LỤC

GIỚI THIỆU MÔN HỌC

I. Giới thiệu chung

Hệ điều hành (Operating system) là thành phần quan trọng và không thể thiếu đối với các hệ thống máy tính. Nó đóng vai trò trung gian giữa người sử dụng hệ thống máy tính và phần cứng máy tính. Hệ điều hành cung cấp môi trường để người dùng có thể sử dụng hệ thống máy tính một cách hiệu quả và dễ dàng.

Kiến thức liên quan tới hệ điều hành có thể chia thành ba dạng chính: thứ nhất đó là các kiến thức và kỹ năng về việc cài đặt, sử dụng, khai thác, đánh giá hệ điều hành một cách hiệu quả. Các kiến thức này cần thiết cho người sử dụng cũng như các kỹ sư, chuyên gia trong việc vận hành, quản lý hệ thống máy tính nói chung. Thứ hai, hệ điều hành được xem xét dưới góc độ thiết kế và xây dựng. Đây là các kiến thức cần thiết cho các chuyên gia về hệ thống cũng như những người tham gia thiết kế, xây dựng hệ điều hành. Thứ ba, là các nguyên lý, các khái niệm về hệ điều hành như là một phần quan trọng của hệ thống máy tính.

Trong chương trình giảng dạy ở bậc đại học, môn học *"Hệ điều hành"* là môn học bắt buộc và đóng vai trò vô cùng quan trọng đối với sinh viên chuyên ngành Công nghệ thông tin.

Để giảng dạy và học tập tốt môn học, tài liệu giảng dạy là yêu cầu tiên quyết. Từ yêu cầu thực tế đó, Khoa Công nghệ thông tin tổ chức biên soạn tài liệu dùng chung "*Hệ điều hành*". Tài liệu được xây dựng phục vụ cho mục đích giảng dạy, học tập, nghiên cứu, tham khảo cho giảng viên, sinh viên khoa Công nghệ thông tin, trường Đại học Kiến Trúc Hà Nội.

II. Mục tiêu môn học

Môn học cung cấp cho sinh viên chuyên ngành Công nghệ thông tin các kiến thức cần thiết về hệ điều hành; giúp sinh viên có lý thuyết và kỹ năng thực hành cần có về hệ điều hành; áp dụng các kiến thức của môn học để quản lý, khai thác, vận hành hệ thống máy tính có hiệu quả cao trong thực tế.

Kết thúc môn học sinh viên có kiến thức vững vàng về nguyên lý hệ điều hành; áp dụng các kiến thức về hệ điều hành để khai thác, vận hành tốt các hệ thống máy tính.

Tài liệu đề cập tới các nguyên lý cơ bản nhất của hệ điều hành: quản lý tiến trình; lập lịch CPU; quản lý bộ nhớ; quản lý thiết bị; các vấn đề liên quan tới bảo vệ, bảo mật hệ điều hành và một số chủ đề nâng cao về hệ điều hành.

III. Phương pháp nghiên cứu

Bài giảng cho môn học *"Hệ điều hành"* được biên soạn dựa trên các cơ sở như sau:

- Nghiên cứu, tham khảo các giáo trình, tài liệu liên quan đến bài giảng.
- Tổng hợp kiến thức cần thiết liên quan tới bài giảng.
- Xây dựng các ứng dụng, đưa ra các ví dụ liên quan tới hệ điều hành của các hệ thống máy tính trong thực tế để minh họa cho lý thuyết môn học.

Bài giảng được biên soạn cho Sinh viên chuyên ngành Công nghệ Thông tin học kỳ 4 Khoa Công nghệ thông tin - Đại học Kiến Trúc Hà Nội sau khi đã hoàn thành các môn học: Tin học đại cương, Nhập môn công nghệ thông tin, Cấu trúc dữ liệu và giải thuật.

Để học tốt môn học, sinh viên cần chú ý:

Kết hợp học lý thuyết và thực hành. Học tới đâu thực hành tới đó, đặc biệt chú trọng vào kỹ năng lập trình giải quyết các bài toán trong học phần.

Có ý thức tự giác học tập, tham gia đầy đủ các buổi học trên lớp.

Chương 1: TỔNG QUAN VỀ HỆ ĐIỀU HÀNH

Nếu không có phần mềm, máy tính chỉ là một thiết bị điện tử thông thường. Với sự hỗ trợ của phần mềm, máy tính có thể lưu trữ, xử lý thông tin và người sử dụng có thể gọi lại được thông tin này. Phần mềm máy tính có thể chia thành nhiều loại: chương trình hệ thống, quản lý sự hoạt động của chính máy tính. Chương trình ứng dụng, giải quyết các vấn đề liên quan đến việc sử dụng và khai thác máy tính của người sử dụng. Hệ điều hành thuộc nhóm các chương trình hệ thống và nó là một chương trình hệ thống quan trọng nhất đối với máy tính và cả người sử dụng. Hệ điều hành điều khiển tất cả các tài nguyên của máy tính và cung cấp một môi trường thuận lợi để các chương trình ứng dụng do người sử dụng viết ra có thể chạy được trên máy tính. Trong chương này chúng ta xem xét vai trò của hệ điều hành trong trường hợp này.

Một máy tính hiện đại có thể bao gồm: một hoặc nhiều processor, bộ nhớ chính, clocks, đĩa, giao diện mạng, và các thiết bị vào/ra khác. Tất cả nó tạo thành một hệ thống phức tạp. Để viết các chương trình để theo dõi tất cả các thành phần của máy tính và sử dụng chúng một cách hiệu quả, người lập trình phải biết processor thực hiện chương trình như thế nào, bộ nhớ lưu trữ thông tin như thế nào, các thiết bị đĩa làm việc (ghi/đọc) như thế nào, lỗi nào có thể xảy ra khi đọc một block đĩa, … đây là những công việc rất khó khăn và quá khó đối với người lập trình. Nhưng rất may cho cả người lập trình ứng dụng và người sử dụng là những công việc trên đã được hệ điều hành hỗ trợ nên họ không cần quan tâm đến nữa. Chương này cho chúng ta một cái nhìn tổng quan về những gì liên quuan đến việc thiết kế cài đặt cũng như chức năng của hệ điều hành để hệ điều hành đạt được mục tiêu: Giúp người sử dụng khai thác máy tính dễ dàng và chương trình của người sử dụng có thể chạy được trên máy tính.

1.1. Các khái niệm cơ bản

1.1.1. Khái niệm và phân loại hệ điều hành

a. Khái niệm hệ điều hành.

Khó có một khái niệm hay định nghĩa chính xác về hệ điều hành, vì hệ điều hành là một bộ phận được nhiều đối tượng khai thác nhất, họ có thể là người sử dụng thông thường, có thể là lập trình viên, có thể là người quản lý hệ thống và tùy theo mức độ khai thác hệ điều hành mà họ có thể đưa ra những khái niện khác nhau về nó. Ở đây ta xem xét 3 khái niệm về hệ điều hành dựa trên quan điểm của người khai thác hệ thống máy tính:

Khái niệm 1: Hệ điều hành là một hệ thống mô hình hoá, mô phỏng hoạt động của máy tính, của người sử dụng và của lập trình viên, hoạt động trong chế độ đối thoại nhằm tạo môi trường khai thác thuận lợi hệ thống máy tính và quản lý tối ưu tài nguyên của hệ thống.

Khái niệm 2: Hệ điều hành là hệ thống chương trình với các chức năng giám sát, điều khiển việc thực hiện các chương trình của người sử dụng, quản lý và phân chia tài nguyên cho nhiều chương trình người sử dụng đồng thời sao cho việc khai thác chức năng của hệ thống máy tính của người sử dụng là thuận lợi và hiệu quả nhất.

Khái niệm 3: Hệ điều hành là một chương trình đóng vai trò như là giao diện giữa người sử dụng và phần cứng máy tính, nó điều khiển việc thực hiện của tất cả các loại chương trình. Khái niệm này rất gần với các hệ điều hành đang sử dụng trên các máy tính hiện nay.

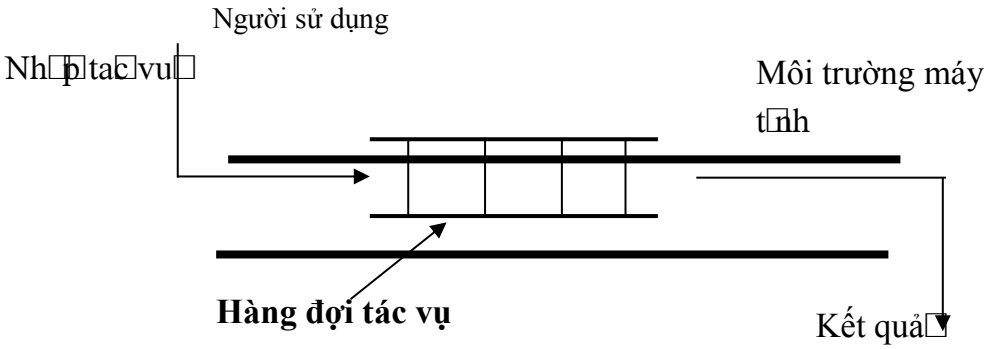
Từ các khái niệm trên chúng ta có thể thấy rằng: Hệ điều hành ra đời, tồn tại và phát triển là để giải quyết vấn đề sử dụng máy tính của người sử dụng, nhằm giúp người sử dụng khai thác hết các chức năng của phần cứng máy tính mà cụ thể là giúp người sử dụng thực hiện được các chương trình của họ trên máy tính.

b. Phân loại hệ điều hành.

Có nhiều cách khác nhau để phân loại hệ điều hành, ở đây chúng tôi dựa vào cách mà hệ điều hành thực hiện các công việc, các tác vụ, các tiến trình của người sử dụng để phân loại hệ điều hành.

Hệ điều hành xử lý theo lô đơn giản.

Hệ điều hành loại này thực hiện các tác vụ lần lượt theo những chỉ thị đã được xác định trước. Khi một tác vụ chấm dứt thì hệ thống sẽ tự động thực hiện tác vụ tiếp theo mà không cần sự can thiệp từ bên ngoài, do đó hệ thống đạt tốc độ thực hiện cao. Để thực hiện được điều này hệ điều hành phải có bộ phận giám sát thường trực để giám sát việc thực hiện của các tác vụ trong hệ thống, bộ phận này thường trú trong bộ nhớ chính.



Hình 1.1. Hoạt động của hệ điều hành lô đơn giản

Trong hệ điều hành này khi hệ thống cần thực hiện một tác vụ thì nó phải lưu chương trình và dữ liệu của các tác vụ vào hành đợi các công việc, sau đó sẽ thực hiện lần lượt từng bộ chương trình và dữ liệu của tác vụ tương ứng trong hàng đời và cho ra lần lượt các kết quả. Hình 1.1. ở trên minh họa cho sự hoạt động của hệ thống theo lô đa chương.

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

Hệ điều hành xử lý theo lô đa chương

Một trong những hạn chế của hệ điều hành xử lý theo lô đơn giản là lãng phí thời gian xử lý của processor khi tác vụ hiện tại truy xuất đến thiết bị vào/ra. Hệ điều hành xử lý theo lô đa chương sẽ khắc phục hạn chế này.

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

Hệ điều hành loại này mang lại hai ưu điểm đó là tiết kiệm được bộ nhớ, vì không nạp hết code và data của các tác vụ vào bộ nhớ, và hạn chế thời gian rỗi của

processor. Tuy nhiên nó phải chi phí cao cho việc lập lịch processor, tức là khi có được processor hệ điều hành phải xem xét nên chuyển nó cho tác vụ nào trong số các tác vụ đang ở trạng thái sẵn sàng. Ngoài ra hệ điều hành còn phải giải quyết việc chia sẻ bộ nhớ chính cho các tác vụ khác nhau. Hệ điều hành MS_DOS là hệ điều hành đơn nhiệm, đa chương.

Hệ điều hành chia sẻ thời gian

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

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

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

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

Hệ điều hành đa vi xử lý

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

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

Đa xử lý bất đối xứng (ASMP: asymmetric): Trong hệ thống này hệ điều

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

Hệ điều hành xử lý thời gian thực

Hệ điều hành này khắc phục nhược điểm của hệ điều hành xử lý theo lô, tức là nó có khả năng cho kết quả tức thời, chính xác sau mỗi tác vụ.

Trong hệ điều hành này các tác vụ cầu thực hiện không được đưa vào hàng đợi mà được xử lý tức thời và trả lại ngay kết quả hoặc thông báo lỗi cho người sử dụng có yêu cầu. Hệ điều hành này hoạt động đòi hỏi sự phối hợp cao giữa phần mềm và phần cứng.

Hệ điều hành mạng

Là các hệ điều hành dùng để điều khiển sự hoạt động của mạng máy tính. Ngoài các chức năng cơ bản của một hệ điều hành, các hệ điều hành mạng còn phải thực hiện việc chia sẻ và bảo vệ tài nguyên của mạng. Hệ điều hành Windows 9x/NT, Windows 200, Linux, là các hệ điều hành mạng máy tính.

Tóm lại: Qua sự phân loại hệ điều hành ở trên ta có thể thấy được quá trình phát triển (evolution) của hệ điều hành. Để khắc phục hạn chế về lãng phí thời gian xử lý của processor trong hệ điều hành theo lô thì hệ điều hành theo lô đa chương ra đời. Để khai thác tối đa thời gian xử lý của processor và tiết kiệm hơn nữa không gian bộ nhớ chính hệ điều hành chia sẻ thời gian ra đời. Chia sẻ thời gian xử lý của processor kết hợp với chia sẻ không gian bộ nhớ chính đã giúp cho hệ điều hành có thể đưa vào bộ nhớ chính nhiều chương trình, tiến trình hơn và các chương trình, tiến trình này có thể hoạt động đồng thời với nhau, nhờ đó mà hiệu suất của hệ thống tăng lên, và cũng từ đây khái niệm hệ điều hành đa chương ra đời. Hệ điều hành đa xử lý và hệ điều hành mạng được phát triển dựa trên hệ điều hành đa nhiệm. Hệ điều hành thời gian thực ra đời là để khắc phục hạn chế của hệ điều hành theo lô và điều khiển các hệ thống thời gian thực. Từ đây chúng ta rút ra một điều rằng: *các hệ điều hành ra đời sau luôn tìm cách khắc phục các hạn chế của hệ điều hành trước đó và phát triển nhiều hơn nữa để đáp ứng yêu cầu ngày càng cao của của người sử dụng và chương trình người sử dụng, cũng như khai thác tối đa các chức năng của phần cứng máy tính để nâng cao hiệu suất của hệ thống. Nhưng chức năng của hệ điều hành càng cao thì chi phí cho nó cũng tăng theo và cấu trúc của hệ điều hành cũng sẽ phức tạp hơn.*

1.1.2. Lịch sử phát triển của hệ điều hành

Thế hệ 1 (1945 - 1955):

Vào những năm 1950 máy tính dùng ống chân không ra đời. Ở thế hệ này mỗi máy tính được một nhóm người thực hiện, bao gồm việc thiết kế, xây dựng chương trình, thao tác, quản lý, ...

Ở thế hệ này người lập trình phải dùng ngôn ngữ máy tuyệt đối để lập trình. Khái niệm ngôn ngữ lập trình và hệ điều hành chưa được biết đến trong khoảng thời gian này.

Thế hệ 2 (1955 - 1965):

Máy tính dùng bán dẫn ra đời, và được sản xuất để cung cấp cho khách hàng. Bộ phận sử dụng máy tính được phân chia rõ ràng: người thiết kế, người xây dựng, người vận hành, người lập trình, và người bảo trì. Ngôn ngữ lập trình Assembly và Fortran ra đời trong thời kỳ này. Với các máy tính thế hệ này để thực hiện một thao tác, lập trình viên dùng Assembly hoặc Fortran để viết chương trình trên phiếu đục lỗ sau đó đưa phiếu vào máy, máy thực hiện cho kết qủa ở máy in.

Hệ thống xử lý theo lô cũng ra đời trong thời kỳ này. Theo đó, các thao tác cần thực hiện trên máy tính được ghi trước trên băng từ, hệ thống sẽ đọc băng từ , thực hiện lần lượt và cho kết quả ở băng từ xuất. Hệ thống xử lý theo lô hoạt động dưới sự điều khiển của một chương trình đặc biệt, chương trình này là hệ điều hành sau này.

Thế hệ 3 (1965 - 1980)

Máy IBM 360 được sản xuất hàng loạt để tung ra thị trường. Các thiết bị ngoại vi xuất hiện ngày càng nhiều, do đó các thao tác điều khiển máy tính và thiết bị ngoại vi ngày càng phức tạp hơn. Trước tình hình này nhu cầu cần có một hệ điều hành sử dụng chung trên tất cả các máy tính của nhà sản xuất và người sử dụng trở nên bức thiết hơn. Và hệ điều hành đã ra đời trong thời kỳ này.

Hệ điều hành ra đời nhằm điều phối, kiểm soát hoạt động của hệ thống và giải quyết các yêu cầu tranh chấp thiết bị. Hệ điều hành đầu tiên được viết bằng ngôn ngữ Assembly. Hệ điều hành xuất hiện khái niệm đa chương, khái niệm chia sẻ thời gian và kỹ thuật Spool. Trong giai đoạn này cũng xuất hiện các hệ điều hành Multics và Unix.

Thế hệ 4 (từ 1980)

Máy tính cá nhân ra đời. Hệ điều hành MS_DOS ra đời gắn liền với máy tính IBM_PC. Hệ điều hành mạng và hệ điều hành phân tán ra đời trong thời kỳ này.

Trên đây chúng tôi không có ý định trình bày chi tiết, đầy đủ về lịch sử hình

thành của hệ điều hành, mà chúng tôi chỉ muốn mượn các mốc thời gian về sự ra đời của các thế hệ máy tính để chỉ cho bạn thấy quá trình hình thành của hệ điều hành gắn liền với quá trình hình thành máy tính. Mục tiêu của chúng tôi trong mục này là muốn nhấn mạnh với các bạn mấy điểm sau đây:

Các ngôn ngữ lập trình, đặc biệt là các ngôn ngữ lập trình cấp thấp, ra đời trước các hệ điều hành. Đa số các hệ điều hành đều được xây dựng từ ngôn ngữ lập trình cấp thấp trừ hệ điều hành Unix, nó được xây dựng từ C, một ngôn ngữ lập trình cấp cao.

Nếu không có hệ điều hành thì việc khai thác và sử dụng máy tính sẽ khó khăn và phức tạp rất nhiều và không phải bất kỳ ai cũng có thể sử dụng máy tính được.

Sự ra đời và phát triển của hệ điều hành gắn liền với sự phát triển của máy tính, và ngược lại sự phát triển của máy tính kéo theo sự phát triển của hệ điều hành. Hệ điều hành thực sự phát triển khi máy tính PC xuất hiện trên thị trường.

Ngoài ra chúng tôi cũng muốn giới thiệu một số khái niệm như: hệ thống xử lý theo lô, hệ thống đa chương, hệ thống chia sẻ thời gian, kỹ thuật Spool, ..., mà sự xuất hiện của những khái niệm này đánh dấu một bước phát triển mới của hệ điều hành. Chúng ta sẽ làm rõ các khái niệm trên trong các chương sau của tài liệu này.

1.1.3. Các khái niệm cơ bản của hệ điều hành

Tiến trình (Process) và tiểu trình (Thread)

Tiến trình là một bộ phận của chương trình đang thực hiện. Tiến trình là đơn vị làm việc cơ bản của hệ thống, trong hệ thống có thể tồn tại nhiều tiến trình cùng hoạt động, trong đó có cả tiến trình của hệ điều hành và tiến trình của chương trình người sử dụng. Các tiến trình này có thể hoạt động đồng thời với nhau.

Để một tiến trình đi vào trạng thái hoạt động thì hệ thống phải cung cấp đầy đủ tài nguyên cho tiến trình. Hệ thống cũng phải duy trì đủ tài nguyên cho tiến trình trong suốt quá trình hoạt động của tiến trình.

Ở đây cần phân biệt sự khác nhau giữa tiến trình và chương trình, chương trình là một tập tin thụ động nằm trên đĩa, tiến trình là trạng thái động của chương trình.

Các hệ điều hành hiện đại sử dụng mô hình đa tiểu trình, trong một tiến trình có thể có nhiều tiểu trình. Tiểu trình cũng là đơn vị xử lý cơ bản trong hệ thống, nó cũng xử lý tuần tự đoạn code của nó, nó cũng sở hữu một con trỏ lệnh, một tập các thanh ghi và một vùng nhớ stack riêng và các tiểu trình cũng chia sẻ thời gian xử lý

của processor như các tiến trình.

Các tiểu trình trong một tiến trình chia sẻ một không gian địa chỉ chung, điều này có nghĩa các tiểu trình có thể chia sẻ các biến toàn cục của tiến trình, có thể truy xuất đến stack của tiểu trình khác trong cùng tiến trình. Như vậy với mô hình tiểu trình, trong hệ thống có thể tồn tại nhiều dòng xử lý cùng chia sẻ một không gian địa chỉ bộ nhớ, các dòng xử lý này hoạt động song song với nhau.

Bộ xử lý lệnh (Shell)

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

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

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

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

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

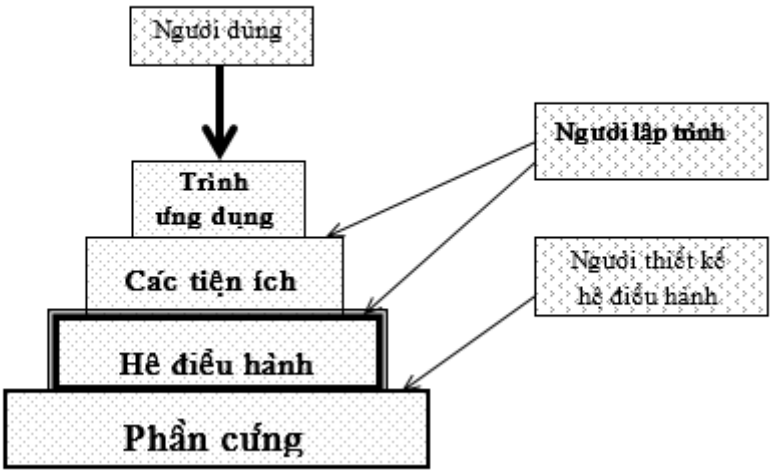
- Trong Assembly, các ứng dụng gọi hàm 4Bh/21h của MS_DOS.
- Trong Pascal, các ứng dụng gọi thủ tục Exec.
- Trong Visual Basic, các ứng dụng gọi hàm/ thủ tục Shell. Ví dụ dòng lệnh sau: Shell "C:\Windows\Notepad.exe" có thể gọi thực hiện chương trình Notepad của Windows.
- Trong Windows 9x/ Windows NT, các ứng dụng gọi hàm ShellExecute.

Sự phân lớp hệ thống (System Layering)

Như đã biết, hệ điều hành là một hệ thống các chương trình bao quanh máy tính

thực (vật lý) nhằm tạo ra một máy tính mở rộng (logic) đơn giản và dễ sử dụng hơn. Theo đó, khi khai thác máy tính người sử dụng chỉ cần tác động vào lớp vỏ bọc bên ngoài của máy tính, mọi sự giao tiếp giữa lớp vỏ bọc này với các chi tiết phần cứng bên trong đều do hệ điều hành thực hiện.

Mỗi người sử dụng khác nhau yêu cầu khai thác hệ điều hành ở những mức độ khác nhau. Người sử dụng thông thường chỉ cần một môi trường thuận lợi để họ thực hiện các ứng dụng, các lập trình viên cần có một môi trường lập trình tốt để họ có thể triển khai các ứng dụng, các chuyên viên lập trình hệ thống cần hệ điều hành cung cấp cho họ các công cụ để họ can thiệp sâu hơn vào hệ thống phần cứng máy tính, ... Để đáp ứng yêu cầu của nhiều đối tượng người sử dụng khác nhau hệ điều



Hình 1.2. Sự phân lớp hệ thống

hành thực hiện phân lớp các chương trình bao quanh máy tính. Các hệ thống như vậy được gọi là hệ thống phân lớp. Hình vẽ 1.2. ở trên minh hoạ cho một hệ thống phân lớp.

Ta có thể hình dung một hệ thống phân lớp được tổ chức như sau:

- Trong cùng là hệ điều hành.
- Tiếp theo là các ngôn ngữ lập trình
- **...**
- **Ngoài cùng là các chương trình ứng dụng .**

Người sử dụng tác động vào lớp trong cùng sẽ gặp nhiều khó khăn hơn khi tác

động vào lớp ngoài cùng.

Tài nguyên hệ thống (System Resources)

Tài nguyên hệ thống là những tồn tại về mặt vật lý tại một thời điểm nhất định hoặc tại mọi thời điểm, và nó có khả năng tác động đến hiệu suất của hệ thống. Một cách tổng quát có thể chia tài nguyên của hệ thống thành hai loại cơ bản:

- Tài nguyên không gian: là các không gian lưu trữ của hệ thống như đĩa, bộ nhớ chính, quan trọng nhất là không gian bộ nhớ chính, nơi lưu trữ các chương trình đang được CPU thực hiện.
- Tài nguyên thời gian: chính là thời gian thực hiện lệnh của processor và thời gian truy xuất dữ liệu trên bộ nhớ.

Sau đây là một vài tài nguyên hệ thống:

Bộ nhớ: Đặc trưng cơ bản của bộ nhớ là thời gian truy cập trực tiếp, thời gian truy cập tuần tự, và dung lượng nhớ. Bộ nhớ được gọi là thực hiện nếu processor có thể thực hiện một câu lệnh trong nó, loại bộ nhớ này có thời gian truy cập trực tiếp và tuần tự là như nhau. Bộ nhớ trong (RAM) của PC là bộ nhớ thực hiện và nó được quản lý bởi hệ thống.

Khi sử dụng bộ nhớ ta cần phân biệt 2 khái niệm: bộ nhớ và truy cập tới bộ nhớ. Bộ nhớ chỉ vùng vật lý chứa dữ liệu, truy cập bộ nhớ là quá trình tìm đến dữ liệu trên bộ nhớ. Có thể xem đây là 2 loại tài nguyên khác nhau vì chúng tồn tại độc lập với nhau.

Processor: Là tài nguyên quan trọng nhất của hệ thống, nó được truy cập ở mức câu lệnh và chỉ có nó mới làm cho câu lệnh thực hiện hay chỉ có Processor mới đưa tiến trình vào trạng thái hoạt động. Trong thực tế khi xem xét về processor người ta chỉ chú ý đến thời gian xử lý của processor.

Tài nguyên ảo/ tài nguyên logic (Virtual Resources): Là loại tài nguyên cung cấp cho chương trình người sử dụng dưới dạng đã được biến đổi, nó chỉ xuất hiện khi hệ thống cần tới nó hoặc khi hệ thống tạo ra nó và nó sẽ tự động mất đi khi hệ thống kết thúc hay chính xác hơn là khi tiến trình gắn với nó đã kết thúc. Tài nguyên ảo có thể là: Đĩa ảo trong môi trường MS_DOS. Điều khiển in trong môi trường mạng của Windows 9x/NT. Nội dung thư mục Spool trong Windows 9x.

Trên khía cạnh cấp phát tài nguyên cho các tiến trình đang hoạt động đồng thời thì tài nguyên hệ thống được chia thành 2 loại:

Tài nguyên phân chia được: là những tài nguyên mà tại một thời điểm nó có thể cấp phát cho nhiều tiến trình khác nhau, các tiến trình song song có thể đồng thời sử dụng các tài nguyên này. Bộ nhớ chính và Processor là 2 tài nguyên phân chia

được điển hình nhất, bởi tại một thời điểm có thể có nhiều tiến trình cùng chia nhau sử dụng không gian lưu trữ của bộ nhớ chính và có thể có nhiều tiến trình thay nhau sử dụng thời gian xử lý của processor.

Tài nguyên không phân chia được: là những tài nguyên mà tại một thời điểm nó chỉ có thể cấp phát cho một tiến trình duy nhất. Máy in là một tài nguyên không phân chia được điển hình nhất.

Vấn đề đặt ra đối với hệ điều hành là phải biến các tài nguyên không phân chia được thành những tài nguyên phân chia được, theo một cách nào đó, để cấp phát cho các tiến trình khi nó có yêu cầu, đặc biệt là các tiến trình hoạt động đồng thời với nhau. Các hệ điều hành đa nhiệm đã cài đặt thành công mục tiêu này. Như chúng ta đã thấy trong môi trường Windows 9x/ NT có thể có nhều tiến trình/ nhiều người sử dụng khác nhau đồng thời sử dụng một máy in.

Ngoài ra hệ điều hành còn phải giải quyết vấn đề tranh chấp tài nguyên giữa các tiến trình đồng thời khi yêu cầu phục vụ của các tiến trình này vượt quá khả năng cấp phát của một tài nguyên kể cả đó là tài nguyên phân chia được.

Lời gọi hệ thống (System Calls)

Để tạo môi trường giao tiếp giữa chương trình của người sử dụng và hệ điều hành, hệ điều hành đưa ra các lời gọi hệ thống. Chương trình của người sử dụng dùng các lời gọi hệ thống để liên lạc với hệ điều hành và yêu cầu các dịch vụ từ hệ điều hành.

Mỗi lời gọi hệ thống tương ứng với một thủ tục trong thư viện của hệ điều hành, do đó chương trình của người sử dụng có thể gọi thủ tục để thực hiện một lời gọi hệ thống. Lời gọi hệ thống còn được thiết dưới dạng các câu lệnh trong các ngôn ngữ lập trình cấp thấp. Lệnh gọi ngắt trong hợp ngữ (Int), và thủ tục gọi hàm API trong windows được xem là một lời gọi hệ thống.

Lời gọi hệ thống có thể được chia thành các loại: quản lý tiến trình, thao tác trên tập tin, thao tác trên thiết bị vào/ ra, thông tin liên tiến trình, ...

Sau đây là một số lời gọi hệ thống của hệ điều hành MS_DOS:

- S = Load_and_exec(processname): tạo tiến trình con và thực hiện nó.
- Fd = Open(filename, mode): mở file để đọc hoặc/và ghi.
- N = Write(Fd, buffer, nbyte): ghi dữ liệu từ đệm vào file.
- Addr = alloc_memory(nbyte): cấp phát một khối nhớ
- Keep_pro(mem_size, status): kết thúc và thường trú chương trình.

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

giao tiếp giữa chương trình người sử dụng và hệ điều hành.

1.2. Các chức năng của hệ điều hành

Một hệ thống máy tính gồm 3 thành phần chính: phần cứng, hệ điều hành và các chương trình ứng dụng và người sử dụng. Trong đó hệ điều hành là một bộ phận quan trọng và không thể thiếu của hệ thống máy tính, nhờ có hệ điều hành mà người sử dụng có thể đối thoại và khai thác được các chức năng của phần cứng máy tính.

Có thể nói hệ điều hành là một hệ thống các chương trình đóng vai trò trung gian giữa người sử dụng và phần cứng máy tính. Mục tiêu chính của nó là cung cấp một môi trường thuận lợi để người sử dụng dễ dàng thực hiện các chương trình ứng dụng của họ trên máy tính và khai thác triệt để các chức năng của phần cứng máy tính.

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

Giả lập một máy tính mở rộng: Máy tính là một thiết bị vi điện tử, nó được cấu thành từ các bộ phận như: Processor, Memory, I/O Device, Bus, ... , do đó để đối thoại hoặc khai thác máy tính người sử dụng phải hiểu được cơ chế hoạt động của các bộ phận này và phải tác động trực tiếp vào nó, tất nhiên là bằng những con số 0,1 (ngôn ngữ máy). Điều này là quá khó đối với người sử dụng. Để đơn giản cho người sử dụng hệ điều hành phải che đậy các chi tiết phần cứng máy tính bởi một máy tính mở rộng, máy tính mở rộng này có đầy đủ các chức năng của một máy tính thực nhưng đơn giản và dễ sử dụng hơn. Theo đó khi cần tác động vào máy tính thực người sử dụng chỉ cần tác động vào máy tính mở rộng, mọi sự chuyển đổi thông tin điều khiển từ máy tính mở rộng sang máy tính thực hoặc ngược lại đều do hệ điều hành thực hiện. Mục đích của chức năng này là: *Giúp người sử dụng khai thác các chức năng của phần cứng máy tính dễ dàng và hiệu quả hơn.*

Quản lý tài nguyên của hệ thống: Tài nguyên hệ thống có thể là: processor, memory, I/O device, printer, file, ..., đây là những tài nguyên mà hệ điều hành dùng để cấp phát cho các tiến trình, chương trình trong quá trình điều khiển sự hoạt động của hệ thống. Khi người sử dụng cần thực hiện một chương trình hay khi một chương trình cần nạp thêm một tiến trình mới vào bộ nhớ thì hệ điều hành phải cấp phát không gian nhớ cho chương trình, tiến trình đó để chương trình, tiến trình đó nạp được vào bộ nhớ và hoạt động được. Trong môi trường hệ điều hành đa nhiệm có thể có nhiều chương trình, tiến trình đồng thời cần được nạp vào bộ nhớ, nhưng không gian lưu trữ của bộ nhớ có giới hạn, do đó hệ điều hành phải tổ chức cấp phát bộ nhớ sao cho hợp lý để đảm bảo tất cả các chương trình, tiến trình khi cần đều được nạp vào bộ nhớ để hoạt động. Ngoài ra hệ điều hành còn phải tổ chức bảo vệ các không gian nhớ

đã cấp cho các chương trình, tiến trình để tránh sự truy cập bất hợp lệ và sự tranh chấp bộ nhớ giữa các chương trình, tiến trình, đặc biệt là các tiến trình đồng thời hoạt động trên hệ thống. Đây là một trong những nhiệm vụ quan trọng của hệ điều hành.

Trong quá trình hoạt động của hệ thống, đặc biệt là các hệ thống đa người dùng, đa chương trình, đa tiến trình, còn xuất hiện một hiện tượng khác, đó là nhiều chương trình, tiến trình đồng thời sử dụng một không gian nhớ hay một tập tin (dữ liệu, chương trình) nào đó. Trong trường hợp này hệ điều hành phải tổ chức việc chia sẻ và giám sát việc truy xuất đồng thời trên các tài nguyên nói trên sao cho việc sử dụng tài nguyên có hiệu quả nhưng tránh được sự mất mát dữ liệu và làm hỏng các tập tin.

Trên đây là hai dẫn chứng điển hình để chúng ta thấy vai trò của hệ điều hành trong việc quản lý tài nguyên hệ thống, sau này chúng ta sẽ thấy việc cấp phát, chia sẻ, bảo vệ tài nguyên của hệ điều hành là một trong những công việc khó khăn và phức tạp nhất. Hệ điều hành đã chi phí nhiều cho công việc nói trên để đạt được mục tiêu: *Trong mọi trường hợp tất cả các chương trình, tiến trình nếu cần được cấp phát tài nguyên để hoạt động thì sớm hay muộn nó đều được cấp phát và được đưa vào trạng thái hoạt động.*

Trên đây là hai chức năng tổng quát của một hệ điều hành, đó cũng được xem như là các mục tiêu mà các nhà thiết kế, cài đặt hệ điều hành phải hướng tới. Các hệ điều hành hiện nay có các chức năng cụ thể sau đây:

- Hệ điều hành cho phép thực hiện nhiều chương trình đồng thời trong môi trường đa tác vụ - **Multitasking Environment**. Hệ điều hành multitasking bao gồm: Windows NT, Windows 2000, Linux và OS/2. Trong hệ thống multasking hệ điều hành phải xác định khi nào thì một ứng dụng được chạy và mỗi ứng dụng được chạy trong khoản thời gian bao lâu thì phải dừng lại để cho các ứng dụng khác được chạy.

- Hệ điều hành tự nạp nó vào bộ nhớ - **It loads itself into memory**: Quá trình nạp hệ điều hành vào bộ nhớ được gọi là quá trình **Booting**. Chỉ khi nào hệ điều hành đã được nạp vào bộ nhớ thì nó mới cho phép người sử dụng giao tiếp với phần cứng. Trong các hệ thống có nhiều ứng dụng đồng thời hoạt động trên bộ nhớ thì hệ điều hành phải chịu trách nhiệm chia sẻ không gian bộ nhớ RAM và bộ nhớ cache cho các ứng dụng này.

- **Hệ điều hành và API: Application Programming Interface**: API là một tập các hàm/thủ tục được xây dựng sẵn bên trong hệ thống, nó có thể thực hiện được nhiều chức năng khác nhau như shutdown hệ thống, đảo ngược hiệu ứng màn hình, khởi động các ứng dụng, … Hệ điều hành giúp cho chương trình của người sử dụng giao tiếp với API hay thực hiện một lời gọi đến các hàm/thủ tục của API.

Nạp dữ liệu cần thiết vào bộ nhớ - **It loads the requied data into memory**: Dữ liệu do người sử dụng cung cấp được đưa vào bộ nhớ để xử lý. Khi nạp dữ liệu vào bộ nhớ hệ điều hành phải lưu lại địa chỉ của bộ nhớ nơi mà dữ liệu được lưu ở đó. Hệ điều hành phải luôn theo dõi bản đồ cấp phát bộ nhớ, nơi dữ liệu và chương trình được lưu trữ ở đó. Khi một chương trình cần đọc dữ liệu, hệ điều hành sẽ đến các địa chỉ bộ nhớ nơi đang lưu trữ dữ liệu mà chương trình cần đọc để đọc lại nó.

- Hệ điều hành biên dịch các chỉ thị chương trình - **It interprets program instructions**: Hệ điều hành phải đọc và giải mã các thao tác cần được thực hiện, nó được viết trong chương trình của người sử dụng. Hệ điều hành cũng chịu trách nhiệm sinh ra thông báo lỗi khi hệ thống gặp lỗi trong khi đang hoạt động.

Hệ điều hành quản lý tài nguyên - **It managers resources**: Nó đảm bảo việc sử dụng thích hợp tất cả các tài nguyên của hệ thống như là: bộ nhớ, đĩa cứng, máy in, …

1.3. Các thành phần và cấu trúc của hệ điều hành

Hệ điều hành là một hệ thống chương trình lớn, thực hiện nhiều nhiệm vụ khác nhau, do đó các nhà thiết kế thường chia hệ điều hành thành nhiều thành phần, mỗi thành phần đảm nhận một nhóm các nhiệm vụ nào đó, các nhiệm vụ này có liên quan với nhau. Cách phân chia nhiệm vụ cho mỗi thành phần, cách kết nối các thành phần lại với nhau để nó thực hiện được một nhiệm vụ lớn hơn khi cần và cách gọi các thành phần này khi cần nó thực hiện một nhiệm vụ nào đó, ... , tất cả các phương thức trên tạo nên cấu trúc của hệ điều hành.

1.3.1. Các thành phần của hệ điều hành

a. Thành phần quản lý tiến trình

Hệ điều hành phải có nhiệm vụ tạo lập tiến trình và đưa nó vào danh sách quản lý tiến trình của hệ thống. Khi tiến trình kết thúc hệ điều hành phải loại bỏ tiến trình ra khỏi danh sách quản lý tiến trình của hệ thống.

Hệ điều hành phải cung cấp đầy đủ tài nguyên để tiến trình đi vào hoạt động và phải đảm bảo đủ tài nguyên để duy trì sự hoạt động của tiến trình cho đến khi tiến trình kết thúc. Khi tiến trình kết thúc hệ điều hành phải thu hồi những tài nguyên mà hệ điều hành đã cấp cho tiến trình.

Trong quá trình hoạt động nếu vì một lý do nào đó tiến trình không thể tiếp tục hoạt động được thì hệ điều hành phải tạm dừng tiến trình, thu hồi tài nguyên mà tiến

trình đang chiếm giữ, sau đó nếu điều kiện thuận lợi thì hệ điều hành phải tái kích hoạt tiến trình để tiến trình tiếp tục hoạt động cho đến khi kết thúc.

Trong các hệ thống có nhiều tiến trình hoạt động song song hệ điều hành phải giải quyết vấn đề tranh chấp tài nguyên giữa các tiến trình, điều phối processor cho các tiến trình, giúp các tiến trình trao đổi thông tin và hoạt động đồng bộ với nhau, đảm bảo nguyên tắc tất cả các tiến trình đã được khởi tạo phải được thực hiện và kết thúc được.

Tóm lại, bộ phận quản lý tiến trình của hệ điều hành phải thực hiện những nhiệm vụ sau đây:

- Tạo lập, hủy bỏ tiến trình.
- Tạm dừng, tái kích hoạt tiến trình.
- Tạo cơ chế thông tin liên lạc giữa các tiến trình.
- Tạo cơ chế đồng bộ hóa giữa các tiến trình.

b. Thành phần quản lý bộ nhớ chính

Bộ nhớ chính là một trong những tài nguyên quan trọng của hệ thống, đây là thiết bị lưu trữ duy nhất mà CPU có thể truy xuất trực tiếp được.

Các chương trình của người sử dụng muốn thực hiện được bởi CPU thì trước hết nó phải được hệ điều hành nạp vào bộ nhớ chính, chuyển đổi các địa chỉ sử dụng trong chương trình thành những địa chỉ mà CPU có thể truy xuất được.

Khi chương trình, tiến trình có yêu cầu được nạp vào bộ nhớ thì hệ điều hành phải cấp phát không gian nhớ cho nó. Khi chương trình, tiến trình kết thúc thì hệ điều hành phải thu hồi lại không gian nhớ đã cấp phát cho chương trình, tiến trình trước đó.

Trong các hệ thống đa chương hay đa tiến trình, trong bộ nhớ tồn tại nhiều chương trình/ nhiều tiến trình, hệ điều hành phải thực hiện nhiệm vụ bảo vệ các vùng nhớ đã cấp phát cho các chương trình/ tiến trình, tránh sự vi phạm trên các vùng nhớ của nhau.

Tóm lại, bộ phận quản lý bộ nhớ chính của hệ điều hành thực hiện những nhiệm vụ sau:

- Cấp phát, thu hồi vùng nhớ.
- Ghi nhận trạng thái bộ nhớ chính.
- Bảo vệ bộ nhớ.
- Quyết định tiến trình nào được nạp vào bộ nhớ.

c. Thành phần quản lý xuất/ nhập

Một trong những mục tiêu của hệ điều hành là giúp người sử dụng khai thác hệ thống máy tính dễ dàng và hiệu quả, do đó các thao tác trao đổi thông tin trên thiết bị xuất/ nhập phải *trong suốt* đối với người sử dụng.

Để thực hiện được điều này hệ điều hành phải tồn tại một bộ phận điều khiển thiết bị, bộ phận này phối hợp cùng CPU để quản lý sự hoạt động và trao đổi thông tin giữa hệ thống, chương trình người sử dụng và người sử dụng với các thiết bị xuất/ nhập.

Bộ phận điều khiển thiết bị thực hiện những nhiệm vụ sau:

- Gởi mã lệnh điều khiển đến thiết bị: Hệ điều hành điều khiển các thiết bị bằng các mã điều khiển, do đó trước khi bắt đầu một quá trình trao đổi dữ liệu với thiết bị thì hệ điều hành phải gởi mã điều khiển đến thiết bị.
- Tiếp nhận yêu cầu ngắt (Interrupt) từ các thiết bị: Các thiết bị khi cần trao đổi với hệ thống thì nó phát ra một tín hiệu yêu cầu ngắt, hệ điều hành tiếp nhận yêu cầu ngắt từ các thiết bị, xem xét và thực hiện một thủ tục để đáp ứng yêu cầu tù các thiết bị.
- Phát hiện và xử lý lỗi: quá trình trao đổi dữ liệu thường xảy ra các lỗi như: thiết bị vào ra chưa sẵn sàng, đường truyền hỏng, ... do đó hệ điều hành phải tạo ra các cơ chế thích hợp để phát hiện lỗi sớm nhất và khắc phục các lỗi vừa xảy ra nếu có thể.

d. Thành phần quản lý bộ nhớ phụ (đĩa)

Không gian lưu trữ của đĩa được chia thành các phần có kích thước bằng nhau được gọi là các block, khi cần lưu trữ một tập tin trên đĩa hệ điều hành sẽ cấp cho tập tin một lượng vừa đủ các block để chứa hết nội dung của tập tin. Block cấp cho tập tin phải là các block còn tự do, chưa cấp cho các tập tin trước đó, do đó sau khi thực hiện một thao tác cấp phát block hệ điều hành phải ghi nhận trạng thái của các block trên đĩa, đặc biệt là các block còn tự do để chuẩn bị cho các quá trình cấp block sau này.

Trong quá trình sử dụng tập tin nội dung của tập tin có thể thay đổi (tăng, giảm), do đó hệ điều hành phải tổ chức cấp phát động các block cho tập tin.

Để ghi/đọc nội dung của một block thì trước hết phải định vị đầu đọc/ ghi đến block đó. Khi chương trình của người sử dụng cần đọc nội dung của một đãy các block không liên tiếp nhau, thì hệ điều hành phải chọn lựa nên đọc block nào trước, nên đọc theo thứ tự nào,..., dựa vào đó mà hệ điều hành di chuyển đầu đọc đến các block thích hợp, nhằm nâng cao tốc độ đọc dữ liệu trên đĩa. Thao tác trên được gọi là lập lịch cho đĩa.

Tóm lại, bộ phận quản lý bộ nhớ phụ thực hiện những nhiệm vụ sau:

- Quản lý không gian trống trên đĩa.
- Định vị lưu trữ thông tin trên đĩa.
- Lập lịch cho vấn đề ghi/ đọc thông tin trên đĩa của đầu từ.

e. Thành phần quản lý tập tin

Máy tính có thể lưu trữ thông tin trên nhiều loại thiết bị lưu trữ khác nhau, mỗi thiết bị lại có tính chất và cơ chế tổ chức lưu trữ thông tin khác nhau, điều này gây khó khăn cho người sử dụng. Để khắc phục điều này hệ điều hành đưa ra khái niệm đồng nhất cho tất cả các thiết bị lưu trữ vật lý, đó là *tập tin* (file).

Tập tin là đơn vị lưu trữ cơ bản nhất, mỗi tập tin có một tên riêng. Hệ điều hành phải thiết lập mối quan hệ tương ứng giữa tên tập tin và thiết bị lưu trữ chứa tập tin. Theo đó khi cần truy xuất đến thông tin đang lưu trữ trên bất kỳ thiết bị lưu trữ nào người sử dụng chỉ cần truy xuất đến tập tin tương ứng thông qua tên của nó, tất cả mọi việc còn lại đều do hệ điều hành thực hiện.

Trong hệ thống có nhiều tiến trình đồng thời truy xuất tập tin hệ điều hành phải tạo ra những cơ chế thích hợp để bảo vệ tập tin trách việc ghi/ đọc bất hợp lệ trên tập tin.

Tóm lại: Như vậy bộ phận quản lý tập tin của hệ điều hành thực hiện những nhiệm vụ sau:

- Tạo/ xoá một tập tin/ thư mục.
- Bảo vệ tập tin khi có hiện tượng truy xuất đồng thời.
- Cung cấp các thao tác xử lý và bảo vệ tập tin/ thư mục.
- Tạo mối quan hệ giữa tập tin và bộ nhớ phụ chứa tập tin.
- Tạo cơ chế truy xuất tập tin thông qua tên tập tin.

f. Thành phần thông dịch lệnh

Đây là bộ phận quan trọng của hệ điều hành, nó đóng vai trò giao tiếp giữa hệ điều hành và người sử dụng. Thành phần này chính là shell mà chúng ta đã biết ở trên. Một số hệ điều hành chứa shell trong nhân (kernel) của nó, một số hệ điều hành khác thì shell được thiết kế dưới dạng một chương trình đặc biệt.

g. Thành phần bảo vệ hệ thống

Trong môi trường hệ điều hành đa nhiệm có thể có nhiều tiến trình hoạt động đồng thời, thì mỗi tiến trình phải được bảo vệ để không bị tác động, có chủ ý hay không chủ ý, của các tiến trình khác. Trong trường hợp này hệ điều hành cần phải có các cơ chế để luôn đảm bảo rằng các File, Memory, CPU và các tài nguyên khác mà hệ điều hành đã cấp cho một chương trình, tiến trình thì chỉ có chương trình tiến trình đó được

quyền tác động đến các thành phần này.

Nhiệm vụ trên thuộc thành phần bảo vệ hệ thống của hệ điều hành. Thành phần này điều khiển việc sử dụng tài nguyên, đặc biệt là các tài nguyên dùng chung, của các tiến trình, đặc biệt là các tiến trình hoạt động đồng thời với nhau, sao cho không xảy ra sự tranh chấp tài nguyên giữa các tiến trình hoạt đồng đồng thời và không cho phép các tiến trình truy xuất bất hợp lệ lên các vùng nhớ của nhau.

Ngoài ra các hệ điều hành mạng, các hệ điều hành phân tán hiện nay còn có thêm thành phần kết nối mạng và truyền thông..

Để đáp ứng yêu cầu của người sử dụng và chương trình người sử dụng các nhiệm vụ của hệ điều hành được thiết kế dưới dạng các dịch vụ:

- Thi hành chương trình: hệ điều hành phải có nhiệm vụ nạp chương trình của người sử dụng vào bộ nhớ, chuẩn bị đầy đủ các điều kiện về tài nguyên để chương trình có thể chạy được và kết thúc được, có thể kết thúc bình thường hoặc kết thúc do bị lỗi. Khi chương trình kết thúc hệ điều hành phải thu hồi tài nguyên đã cấp cho chương trình và ghi lại các thông tin mà chương trình đã thay đổi trong quá trình chạy (nếu có).
- Thực hiện các thao tác xuất nhập dữ liệu: Khi chương trình chạy nó có thể yêu cầu xuất nhập dữ liệu từ một tập tin hoặc từ một thiết bị xuất nhập nào đó, trong trường hợp này hệ điều hành phải hỗ trợ việc xuất nhập dữ liệu cho chương trình, phải nạp được dữ liệu mà chương trình cần vào bộ nhớ.
- Thực hiện các thao tác trên hệ thống tập tin: Hệ điều hành cần cung cấp các công cụ để chương trình dễ dàng thực hiện các thao tác đọc ghi trên các tập tin, các thao tác này phải thực sự an toàn, đặc biệt là trong môi trường đa nhiệm.
- Trao đổi thông tin giữa các tiến trình: Trong môi trường hệ điều hành đa nhiệm, với nhiều tiến trình hoạt động đồng thời với nhau, một tiến trình có thể trao đổi thông tin với nhiều tiến trình khác, hệ điều hành phải cung cấp các dịch vụ cần thiết để các tiến trình có thể trao đổi thông tin với nhau và phối hợp cùng nhau để hoàn thành một tác vụ nào đó.
- Phát hiện và xử lý lỗi: Hệ điều hành phải có các công cụ để chính hệ điều hành và để hệ điều hành giúp chương trình của người sử dụng phát hiện các lỗi do hệ thống (CPU, Memory, I/O device, Program) phát sinh. Hệ điều hành cũng phải đưa ra các dịch vụ để xử lý các lỗi sao cho hiệu quả nhất.

1.3.2. Cấu trúc của hệ điều hành

a. Hệ thống đơn khối (monolithic systems)

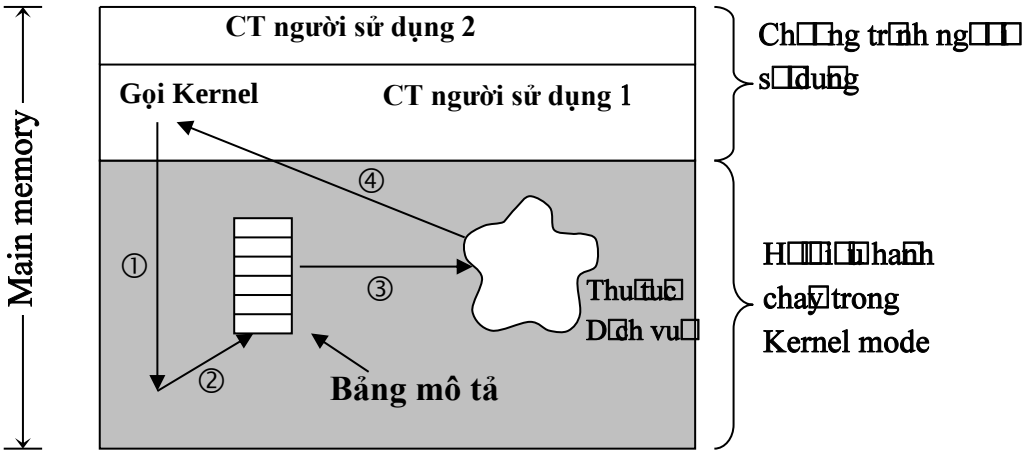
Trong hệ thống này hệ điều hành là một tập hợp các thủ tục, mỗi thủ tục có thể gọi thực hiện một thủ tục khác bất kỳ lúc nào khi cần thiết.

Hệ thống đơn khối thường được tổ chức theo nhiều dạng cấu trúc khác nhau:

Sau khi biên dịch tất cả các thủ tục riêng hoặc các file chứa thủ tục của hệ điều hành được liên kết lại với nhau và được chứa vào một file được gọi là file đối tượng, trong file đối tượng này còn chứa cả các thông tin về sự liên kết của các thủ tục.

Sau khi biên dịch các thủ tục của hệ điều hành không được liên kết lại, mà hệ thống chỉ tạo ra file hoặc một bảng chỉ mục để chứa thông tin của các thủ tục hệ điều hành, mỗi phần tử trong bảng chỉ mục chứa một con trỏ trỏ tới thủ tục tương ứng, con trỏ này dùng để gọi thủ tục khi cần thiết. Ta có thể xem cách gọi ngắt (Interrupt) trong ngôn ngữ lập trình cấp thấp và cách thực hiện đáp ứng ngắt dựa vào bảng vector ngắt trong MS_DOS là một ví dụ cho cấu trúc này.

Hình vẽ sau đây minh họa cho việc đáp ứng một lời gọi dịch vụ từ chương trình của người sử dụng dựa vào bảng chỉ mục.



Hình 1.3. Sơ đồ thực hiện lời gọi hệ thống

Trong đó:

1. Chương trình của người sử dụng gởi yêu cầu đến Kernel.
2. Hệ điều hành kiểm tra yêu cầu dịch vụ.
3. Hệ điều hành xác định (vị trí) và gọi thủ tục dịch vụ tương ứng.
4. Hệ điều hành trả điều khiển lại cho chương trình người sử dụng.

Sau đây là một cấu trúc đơn giản của hệ thống đơn khối, trong cấu trúc này các thủ tục được chia thành 3 lớp:

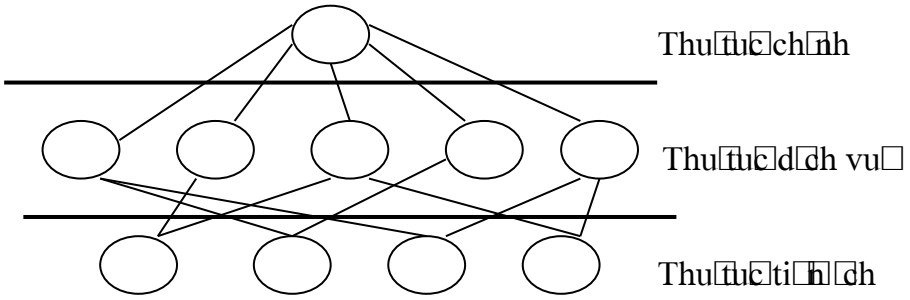
1. Một chương trình chính (chương trình của người sử dụng) gọi đến một

thủ tục dịch vụ của hệ điều hành. Lời gọi này được gọi là lời gọi hệ thống.

2. Một tập các thủ tục dịch vụ (service) để đáp ứng những lời gọi hệ thống từ các chương trình người sử dụng.

3. Một tập các thủ tục tiện ích (utility) hỗ trợ cho các thủ tục dịch trong việc thực hiện cho các lời gọi hệ thống.

Trong cấu trúc này mỗi lời gọi hệ thống sẽ gọi một thủ tục dịch vụ tương ứng. Thủ tục tiện ích thực hiện một vài điều gì đó mà thủ tục dịch vụ cần, chẳng hạn như nhận dữ liệu từ chương trình người sử dụng. Các thủ tục của hệ điều hành được chia vào 3 lớp theo như hình vẽ dưới đây.



Hình 1.4. Cấu trúc đơn giản của một monolithic system

Nhận xét:

- Với cấu trúc này chương trình của người sử dụng có thể truy xuất trực tiếp đến các chi tiết phần cứng bằng cách gọi một thủ tục cấp thấp, điều này gây khó khăn cho hệ điều hành trong việc kiểm soát và bảo vệ hệ thống.
- Các thủ tục dịch vụ mang tính chất tỉnh, nó chỉ hoạt động khi được gọi bởi chương trình của người sử dụng, điều này làm cho hệ điều hành thiếu chủ động trong việc quản lý môi trường.

b. Các hệ thống phân lớp (Layered Systems)

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

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

Ví dụ về một hệ điều hành phân lớp:

Lớp 5: Chương trình ứng dụng

Lớp 4: Quản lý bộ đệm cho các thiết bị xuất nhập

Lớp 3: Trình điều khiển thao tác console

Lớp 2: Quản lý bộ nhớ

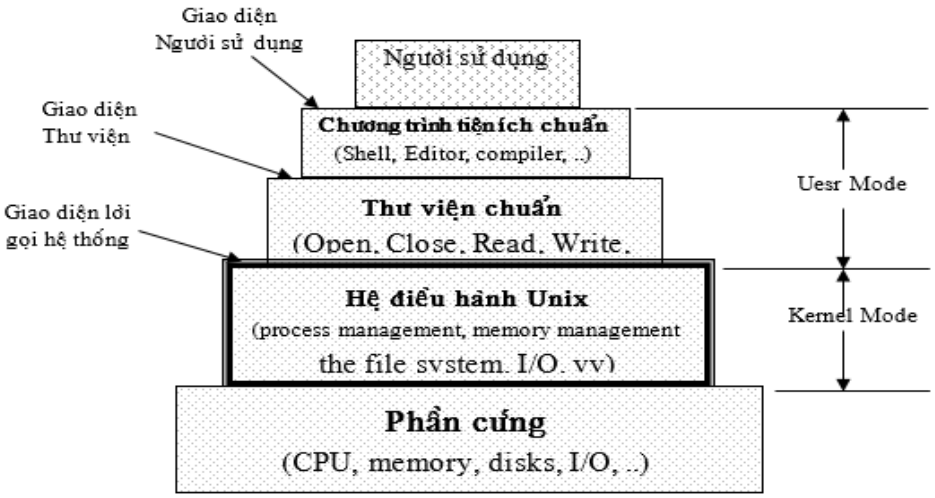
Lớp 1: Điều phối processor

Lớp 0: Phần cứng hệ thống

Hình vẽ 1.5 sau đây cho ta thấy cấu trúc phân lớp trong hệ điều hành Unix.

Nhận xét:

- Khi xây dựng hệ điều hành theo hệ thống này các nhà thiết kế gặp khó khăn trong việc xác định số lượng lớp, thứ tự và chức năng của mỗi lớp.



Hình 1.5. Hệ thống phân lớp của hệ điều hành UNIX

- Hệ thống này mang tính đơn thể, nên dễ cài đặt, tìm lỗi và kiểm chứng hệ thống.
- Trong một số trường hợp lời gọi thủ tục có thể lan truyền đến các thủ tục khác ở các lớp bên trong nên chi phí cho vấn đề truyền tham số và chuyển đổi ngữ cảnh tăng lên, dẫn đến lời gọi hệ thống trong cấu trúc này thực hiện chậm hơn so với các cấu trúc khác.

c. Máy ảo (Virtual Machine)

Thông thường một hệ thống máy tính bao gồm nhiều lớp: phần cứng ở lớp thấp nhất, hạt nhân ở lớp kế trên. Hạt nhân dùng các chỉ thị (lệnh máy) của phần cứng để tạo ra một tập các lời gọi hệ thống. Các hệ điều hành hiện đại thiết kế một lớp các chương trình hệ thống nằm giữa hệ điều hành và chương trình của người sử dụng.

Các chương trình hệ thống có thể sử dụng các lời gọi hệ thống hoặc sử dụng trực tiếp các chỉ thị phần cứng để thực hiện một chức năng hoặc một thao tác nào đó, do đó các chương trình hệ thống thường xem các lời gọi hệ thống và các chỉ thị phần cứng như ở trên cùng một lớp.

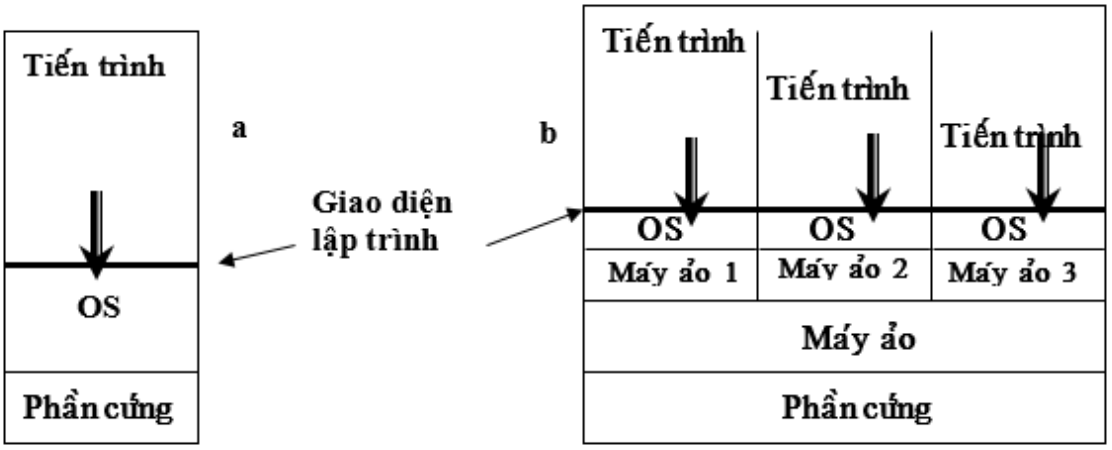
Một số hệ điều hành cho phép các chương trình của người sử dụng có thể gọi dễ dàng các chương trình hệ thống và xem mọi thành phần dưới chương trình hệ thống đều là phần cứng máy tính. Lớp các ứng dụng này sử dụng khái niệm máy ảo.

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

Trong cấu trúc này phần nhân của hệ thống trở thành bộ phận tổ chức giám sát máy ảo, phần này chịu trách nhiệm giao tiếp với phần cứng, chia sẻ tài nguyên hệ thống để tạo ra nhiều máy ảo, hoạt động độc lập với nhau, để cung cấp cho lớp trên.

Ở đây cần phân biệt sự khác nhau giữa máy ảo và máy tính mở rộng, máy ảo là bản sao chính xác các đặc tính phần cứng của máy tính thực sự và cho phép hệ điều hành hoạt động trên nó, sau đó hệ điều hành xây dựng máy tính mở rộng để cung cấp cho người sử dụng.

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



Hình 1.6. Mô hình hệ thống (a) Không có máy ảo (b) Máy ảo

Hình vẽ trên đây cho chúng ta thấy sự khác nhau trong hệ thống không có máy ảo và hệ thống có máy ảo:

Nhận xét:

- Việc cài đặt các phần mềm giả lập phần cứng để tạo ra máy ảo thường rất

khó khăn và phức tạp.

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

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

d. Mô hình Client/ Server (client/ server model)

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

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

- Tiến trình bên ngoài hay tiến trình của chương trình người sử dụng được gọi là các tiến trình Client.
- Tiến trình của hệ điều hành được gọi là tiến trình Server.

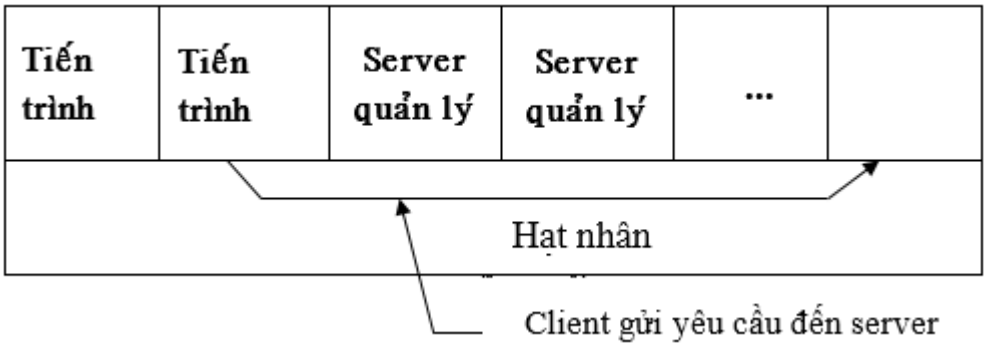
Khi cần thực hiện một chức năng hệ thống các tiến trình Client sẽ gởi yêu cầu tới tiến trình server tương ứng, tiến trình server sẽ xử lý và trả lời kết quả cho tiến trình Client.

Nhận xét:

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

chỉ cần thêm các server mới vào hệ thống.

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



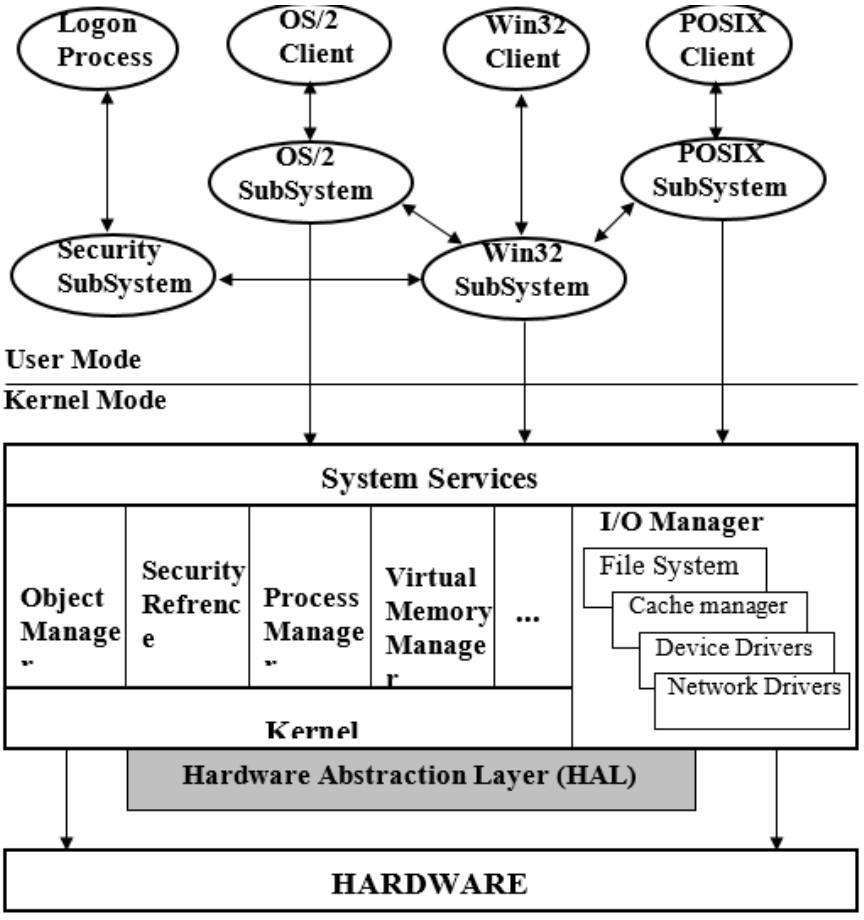
Hình 1.7. Mô hình client-Server

Hình vẽ sau đây cho thấy cấu trúc của hệ điều hành Windows NT. Đây là một cấu trúc phức tạp với nhiều thành phần khác nhau và nó được xây dựng dựa trên mô hình hệ điều hành Client/ Server.

Trong cấu trúc này chúng ta thấy nổi rõ hai điểm sau đây:

- Cấu trúc của windows NT được chia thành 2 mode: Kernel mode và User mode. Các chương trình ứng dụng của người sử dụng chỉ chạy trong User mode, các dịch vụ của hệ điều hành chỉ chạy trong Kernel mode. Nhờ vậy mà việc bảo vệ các chương trình của người sử dụng cũng như các thành phần của hệ điều hành, trên bộ nhớ, được thực hiện dễ dàng hơn.

- Trong User mode của Windows NT có chứa các hệ thống con môi trường như: OS/2 subsystem và POSIX subsystem, nhờ có các hệ thống con môi trường này mà các ứng dụng được thiết kế trên các hệ điều hành khác vẫn chạy được trên hệ điều hành Windows NT. Đây là điểm mạnh của các hệ điều hành Microsoft của từ Windows NT.



Hình 1.8. Cấu trúc của Windows NT

Chương 2: QUẢN LÝ TIẾN TRÌNH

Tất cả các hệ điều hành đa chương, từ các hệ điều hành đơn người sử dụng đến các hệ điều hành có thể hỗ trợ đến hàng ngàn người sử dụng, đều phải xây dụng dựa trên khái niệm tiến trình. Vì thế, một yêu cầu quan trọng trong thiết kế hệ điều hành là thành phần quản lý tiến trình của hệ điều hành phải đáp ứng tất cả những gì liên quan đến tiến trình:

Hệ điều hành phải cho phép thực hiện nhiều tiến trình đồng thời để khai thác tối đa thời gian xử lý của processor nhưng cũng cung cấp được thời gian hồi đáp hợp lý.

Hệ điều hành phải cấp phát tài nguyên để tiến trình hoạt động một cách hiệu quả với một chính sách hợp lý nhưng không xảy ra tình trạng tắc nghẽn trong hệ thống.

Hệ điều hành có thể được yêu cầu để hỗ trợ truyền thông liên tiến trình và người sử dụng tạo ra tiến trình.

Hệ điều hành phải có nhiệm vụ tạo ra tiến trình, điều khiển sự hoạt động của tiến trình và kết thúc tiến trình.

Một số hệ điều hành phân biệt hai khái niệm tiến trình và tiểu trình. Tiến trình liên quan đến quyền sở hữu tài nguyên, tiểu trình liên quan đến sự thực hiện chương trình.

Trong các hệ điều hành đa chương, có nhiều tiến trình tồn tại trên bộ nhớ chính, các tiến trình này luân phiên giữa hai trạng thái: sử dụng processor và đợi thực hiện vào/ra hay một vài sự kiện nào đó xảy ra.

Tất cả những vấn đề trên sẽ được làm sáng tỏ trong chương này.

2.1. Các khái niệm cơ bản

2.1.1. Tiến trình và các loại tiến trình

Tiến trình (process): Trong chương I chúng ta đã có khái niệm về tiến trình: *Tiến trình là một bộ phận của một chương trình đang thực hiện, đơn vị thực hiện tiến trình là processer.* Ở đây chúng tôi nhấn mạnh thêm rằng: Vì tiến trình là một bộ phận của chương trình nên tương tự như chương trình tiến trình cũng sở hữu một con trỏ lệnh, một con trỏ stack, một tập các thanh ghi, một không gian địa chỉ trong bộ nhớ chính và tất cả các thông tin cần thiết khác để tiến trình có thể hoạt động được.

Khái niệm trên đây mang tính trực quan, để thấy được bản chất của tiến trình các chuyên gia về hệ điều hành đã đưa ra nhiều định nghĩa khác nhau về tiến trình, ở đây chúng tôi nêu ra hai định nghĩa để các bạn tham khảo. Định nghĩa của Saltzer: *Tiến trình là một chương trình do một processor logic thực hiện.* Định nghĩa của

Horning & Rendell: *Tiến trình là một quá trình chuyển từ trạng thái này sang trạng thái khác dưới tác động của hàm hành động, xuất phát từ một trạng thái ban đầu nào đó.*

Định nghĩa của Saltzer cho thấy, trên góc độ thực hiện thì tiến trình hoàn toàn tương tự chương trình, chỉ khác ở chỗ: tiến trình do processor logic chứ không phải processor vật lý thực hiện. Điều này sẽ được làm sáng tỏ trong phần mô tả về tiến trình sau đây. Định nghĩa của Horning & Rendell cho thấy trong quá trình hoạt động của tiến trình là quá trình chuyển từ trạng thái này sang trạng thái khác nhưng sự chuyển đổi này không phải do chính bản thân tiến trình mà là do sự tác động từ bên ngoài, cụ thể ở đây là bộ phận điều phối tiến trình của hệ điều hành. Điều này sẽ được làm sáng tỏ trong phần mô tả về các trạng thái tiến trình sau đây.

Các loại tiến trình: Các tiến trình trong hệ thống có thể chia thành hai loại: tiến trình tuần tự và tiến trình song song. Tiến trình tuần tự là các tiến trình mà điểm khởi tạo của nó là điểm kết thúc của tiến trình trước đó. Tiến trình song song là các tiến trình mà điểm khởi tạo của tiến trình này nằm ở thân của các tiến trình khác, tức là có thể khởi tạo một tiến trình mới khi các tiến trình trước đó chưa kết thúc. Tiến trình song song được chia thành nhiều loại:

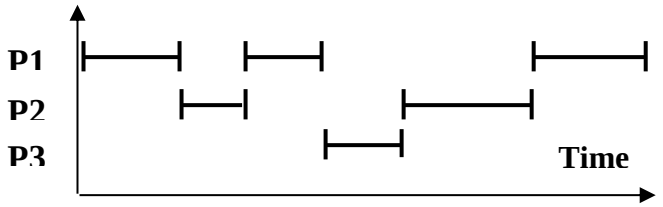
- Tiến trình song song độc lập: là các tiến trình hoạt động song song nhưng không có quan hệ thông tin với nhau, trong trường hợp này hệ điều hành phải thiết lập cơ chế bảo vệ dữ liệu của các tiến trình, và cấp phát tài nguyên cho các tiến trình một cách hợp lý.
- Tiến trình song song có quan hệ thông tin: trong quá trình hoạt động các tiến trình thường trao đổi thông tin với nhau, trong một số trường hợp tiến trình gởi thông báo cần phải nhận được tín hiệu từ tiến trình nhận để tiếp tục, điều này dễ dẫn đến bế tắc khi tiến trình nhận tín hiệu không ở trong trạng thái nhận hay tiến trình gởi không ở trong trạng thái nhận thông báo trả lời.
- Tiến trình song song phân cấp: Trong qua trình hoạt động một tiến trình có thể khởi tạo các tiến trình khác hoạt động song song với nó, tiến trình khởi tạo được gọi là tiến trình cha, tiến trình được tạo gọi là tiến trình con. Trong mô hình này hệ điều hành phải giải quyết vấn đề cấp phát tài nguyên cho các tiến trình con. Tiến trình con nhận tài nguyên ở đâu, từ tiến trình cha hay từ hệ thống. Để giải quyết vấn đề này hệ điều hành đưa ra 2 mô hình quản lý tài nguyên: Thứ nhất, mô hình tập trung, trong mô hình này hệ điều hành chịu trách nhiệm phân phối tài nguyên cho tất cả các tiến trình trong hệ thống. Thứ hai, mô hình phân tán, trong mô hình này hệ điều hành cho phép tiến trình con nhận tài nguyên từ tiến trình cha, tức là tiến trình khởi tạo có nhiệm vụ nhận tài nguyên từ hệ điều hành để cấp phát cho các tiến trình mà nó tạo ra, và nó có nhiệm vụ thu hồi lại tài nguyên đã cấp phát trả về cho hệ điều

hành trước khi kết thúc.

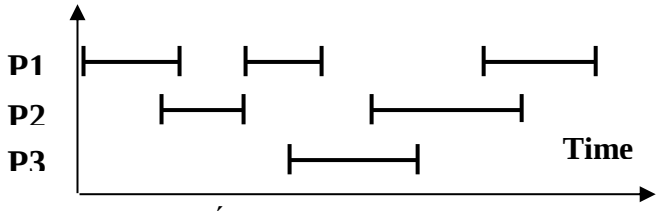
- Tiến trình song song đồng mức: là các tiến trình hoạt động song song sử dụng chung tài nguyên theo nguyên tắc lần lượt, mỗi tiến trình sau một khoảng thời gian chiếm giữ tài nguyên phải tự động trả lại tài nguyên cho tiến trình kia.

Các tiến trình tuần tự chỉ xuất hiện trong các hệ điều hành đơn nhiệm đa chương, như hệ điều hành MS_DOS, loại tiến trình này tồn tại nhiều hạn chế, điển hình nhất là không khai thác tối đa thời gian xử lý của processor. Các tiến trình song song xuất hiện trong các hệ điều hành đa nhiệm đa chương, trên cả hệ thống uniprocessor và multiprocessor. Nhưng sự song song thực, chỉ có ở các hệ thống multiprocessor, trong hệ thống này mỗi processor chịu trách nhiệm thực hiện một tiến trình. Sự song song trên các hệ thống uniprocessor là sự song song giả, các tiến trình song song trên hệ thống này thực chất là các tiến trình thay nhau sử dụng processor, tiến trình này đang chạy thì có thể dừng lại để nhường processor cho tiến trình khác chạy và sẽ tiếp tục lại sau đó khi có được processor. Đây là trường hợp mà ở trên ta cho rằng: điểm khởi tạo của tiến trình này nằm ở thân của tiến trình khác.

Hình vẽ sau đây minh họa sự khác nhau, về mặt thực hiện, giữa các tiến trình song song/ đồng thời trong hệ thống uniprocessor với các tiến trình song song/ đồng thời trong hệ thống multiprocessor.



a. Trong hệ thống uniprocessor



b. Trong hệ thống Multiprocessor

Hình 2.1. Sự thực hiện đồng thời của các tiến trình trong hệ thống uniprocessor (a) và hệ thống multiprocessor (b).

Trong tài liệu này chúng ta chỉ khảo sát sự hoạt động của các tiến trình song

song (hay đồng thời) trên các hệ thống uniprocessor.

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

Mô hình tiến trình

Đa số các hệ điều hành đều muốn đưa sự đa chương, đa nhiệm vào hệ thống. Tức là, trong hệ thống có thể có nhiều chương trình hoạt động đồng thời (concurrence) với nhau. Về nguyên tắc, để thực hiện được điều này thì hệ thống phải có nhiều processor, mỗi processor có nhiệm vụ thực hiện một chương trình, nhưng mong muốn của hệ điều hành cũng như người sử dụng là *thực hiện sự đa chương trên các hệ thống chỉ có một processor*, và trên thực tế đã xuất hiện nhiều hệ điều hành thực hiện được điều này, hệ điều hành windows9x, windowsNT/2000 chạy trên máy tính cá nhân là một ví dụ. Để thực hiện được điều này hệ điều hành đã sử dụng mô hình tiến trình để tạo ra sự song song giả hay tạo ra các processor logic từ processor vật lý. Các processor logic có thể hoạt động song song với nhau, mỗi processor logic chịu trách nhiệm thực hiện một tiến trình.

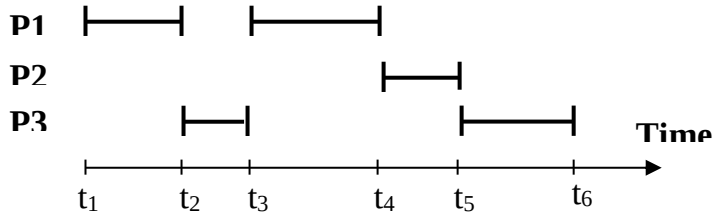
Trong mô hình tiến trình hệ điều hành chia chương trình thành nhiều tiến trình, khởi tạo và đưa vào hệ thống nhiều tiến trình của một chương trình hoặc của nhiều chương trình khác nhau, cấp phát đầy đủ tài nguyên (trừ processor) cho tiến trình và đưa các tiến trình sang trạng thái sẵn sàng. Hệ điều hành bắt đầu cấp processor cho một tiến trình trong số các tiến trình ở trạng thái sẵn sàng để tiến trình này hoạt động, sau một khoảng thời gian nào đó hệ điều hành thu hồi processor của tiến trình này để cấp cho một tiến trình sẵn sàng khác, sau đó hệ điều hành lại thu hồi processor từ tiến trình mà nó vừa cấp để cấp cho tiến trình khác, có thể là tiến trình mà trước đây bị hệ điều hành thu hồi processor khi nó chưa kết thúc, và cứ như thế cho đến khi tất cả các tiến trình mà hệ điều hành khởi tạo đều hoạt động và kết thúc được. Điều đáng chú ý trong mô hình tiến trình này là khoảng thời gian chuyển processor từ tiến trình này sang tiến trình khác hay khoảng thời gian giữa hai lần được cấp phát processor của một tiến trình là rất nhỏ nên các tiến trình có cảm giác luôn được sở hữu processor (logic) hay hệ thống có cảm giác các tiến trình/ chương trình hoạt động song song nhau. Hiện tượng này được gọi là sự song song giả.

Giả sử trong hệ thống có 3 tiến trình sẵn sàng P_1, P_2, P_3 thì quá trình chuyển

processor giữa 3 tiến trình này có thể minh họa như sau:

Thời điểm	Trạng thái các tiến trình	
t_1	P_1: được cấp processor	
t_2	P_1: bị thu hồi processor	(khi chưa kết thúc)
	P_3: được cấp processor	
t_3	P_3: bị thu hồi processor	(khi chưa kết thúc)
	P_1: được cấp processor	
t_4	P_1: kết thúc và trả lại processor	
	P_2: được cấp processor	
t_5	P_2: kết thúc và trả lại processor	
	P_3: được cấp processor	
t_6	P3: kết thúc và trả lại processor	

Hình sau đây minh họa quá trình thực hiện của 3 tiến trình P_1, P_2, P_3 ở trên:



uniprocessor

Hình 2.2. Sự hoạt động "song song" của các tiến trình P_1, P_2, P_3

Chúng ta đều biết, chức năng cở bản của processor là thực hiện các chỉ thị máy (machine instrustion) thường trú trong bộ nhớ chính, các chỉ thị này được cung cấp từ một chương trình, chương trình bao gồm một dãy tuần tự các chỉ thị. Và theo trên, tiến trình là một bộ phận của chương trình, nó cũng sở hữu một tập lệnh trong bộ nhớ chính, một con trỏ lệnh,… Nên xét về bản chất, thì việc chuyển processor từ tiến trình này sang tiến trình khác thực chất là việc điều khển processor để nó thực hiện xen kẽ các chỉ thị bên trong tiến trình. Điều này có thể thực hiện dễ dàng bằng cách thay đổi hợp lý giá trị của con trỏ lệnh, đó chính là cặp thanh ghi CS:IP trong các processor thuộc kiến trúc Intel, để con trỏ lệnh chỉ đến các chỉ thị cần thực hiện trong các tiến trình. Để thấy rõ hơn điều này ta hãy xem ví dụ sau đây:

Giả sử hệ thống cần thực hiện đồng thời 3 tiến trình P_1, P_2, P_3, bắt đầu từ tiến trình P_1. Các chỉ thị của các tiến trình này được nạp vào bộ nhớ tại các địa chỉ như sau:

Tiến trình P_1:	Tiến trình P_2:	Tiến trình P_3:
a + 0	b + 0	c + 0
a + 1	b + 2	c + 1
a + 3	b + 3	c + 4
a + 5		c + 6

Trong đó: a: là địa chỉ bắt đầu của chương trình của tiến trình P_1
b: là địa chỉ bắt đầu của chương trình của tiến trình P_2
c: là địa chỉ bắt đầu của chương trình của tiến trình P_3

Thì giá trị của con trỏ lệnh, chính xác là giá trị cặp thanh ghi CS:IP, lần lượt là: a + 0, b + 0, c + 0, a + 1, b + 2, c + 1, a + 3, b + 3, c + 4, a + 5, c + 6. Tức là, processor thực hiện xen kẽ các chỉ thị của 3 tiến trình P_1, P_2, P_3 từ lệnh đầu tiên đến lệnh cuối cùng, cho đến khi tất cả các chỉ thị của 3 tiến trình đều được thực hiện. Nhưng khoảng thời gian từ khi con trỏ lệnh = a + 0 đến khi = a + 1, hay từ khi = b + 0 đến khi = b + 2, … là rất nhỏ, nên hệ thống có "cảm giác" 3 tiến trình P_1, P_2, P_3 hoạt động đồng thời với nhau.

Ví dụ trên đây cho ta thấy bản chất của việc thực hiện song song (hay đồng thời) các tiến trình trên các hệ thống uniprocessor.

Rõ ràng với mô hình tiến trình hệ thống có được 2 điều lợi:

- Tiết kiệm được bộ nhớ: vì không phải nạp tất cả chương trình vào bộ nhớ mà chỉ nạp các tiến trình cần thiết nhất, sau đó tùy theo yêu cầu mà có thể nạp tiếp các tiến trình khác.
- Cho phép các chương trình hoạt động song song nên tốc độ xử lý của toàn hệ thống tăng lên và khai thác tối đa thời gian xử lý của processor.

Việc chọn thời điểm dừng của tiến trình đang hoạt động (đang chiến giữ processor) để thu hồi processor chuyển cho tiến trình khác hay việc chọn tiến trình tiếp theo nào trong số các tiến trình đang ở trạng thái sẵn sàng để cấp processor là những vấn đề khá phức tạp đòi hỏi hệ điều hành phải có một cơ chế điều phối thích hợp thì mới có thể tạo ra được hiệu ứng song song giả và sử dụng tối ưu thời gian xử lý của processor. Bộ phận thực hiện chức năng này của hệ điều hành được gọi là bộ điều phối (dispatcher) tiến trình.

2.1.2. Tiểu trình và tiến trình

Tiểu trình: Thông thường mỗi tiến trình có một không gian địa chỉ và một dòng xử lý. Nhưng trong thực tế có một số ứng dụng cần nhiều dòng xử lý cùng chia sẻ một không gian địa chỉ tiến trình, các dòng xử lý này có thể hoạt động song song với nhau như các tiến trình độc lập trên hệ thống. Để thực hiện được điều này các hệ điều

hành hiện nay đưa ra một cơ chế thực thi (các chỉ thị trong chương trình) mới, được gọi là tiểu trình.

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

Một tiến trình đơn có thể bao gồm nhiều tiểu trình. Các tiểu trình trong một tiến trình chia sẻ một không gian địa chỉ chung, nhờ đó mà các tiểu trình có thể chia sẻ các biến toàn cục của tiến trình và có thể truy xuất lên các vùng nhớ stack của nhau.

Các tiểu trình chia sẻ thời gian xử lý của processor giống như cách của tiến trình, nhờ đó mà các tiểu trình có thể hoạt động song song (giả) với nhau. Trong quá trình thực thi của tiểu trình nó cũng có thể tạo ra các tiến trình con của nó.

Đa tiểu trình trong đơn tiến trình: Điểm đáng chú ý nhất của mô hình tiểu trình là: có nhiều tiểu trình trong phạm vi một tiến trình đơn. Các tiến trình đơn này có thể hoạt động trên các hệ thống multiprocessor hoặc uniprocessor. Các hệ điều hành khác nhau có cách tiếp cận mô hình tiểu trình khác nhau. Ở đây chúng ta tiếp cận mô hình tiểu trình từ mô hình tác vụ (Task), đây là các tiếp cận của windows NT và các hệ điều hành đa nhiệm khác. Trong các hệ điều hành này tác vụ được định nghĩa như là một đơn vị của sự bảo vệ hay đơn vị cấp phát tài nguyên. Trong hệ thống tồn tại một không gian địa chỉ ảo để lưu giữ tác vụ và một cơ chế bảo vệ sự truy cập đến các file, các tài nguyên Vào/Ra và các tiến trình khác (trong các thao tác truyền thông liên tiến trình).

Trong phạm vị một tác vụ, có thể có một hoặc nhiều tiểu trình, mỗi tiểu trình bao gồm: Một trạng thái thực thi tiểu trình (running, ready,…). Một lưu trữ về ngữ cảnh của processor khi tiểu trình ở trạng thái not running (một cách để xem tiểu trình như một bộ đếm chương trình độc lập hoạt động trong phạm vi tác vụ). Các thông tin thống kê về việc sử dụng các biến cục bộ của tiểu trình. Một stack thực thi. Truy xuất đến bộ nhớ và tài nguyên của tác vụ, được chia sẻ với tất cả các tiểu trình khác trong tác vụ.

Trong các ứng dụng server, chẳng hạn như ứng dụng file server trên mạng cục bộ, khi có một yêu cầu hình thành một file mới, thì một tiểu trình mới được hình thành từ chương trình quản lý file. Vì một server sẽ phải điều khiển nhiều yêu cầu, có thể đồng thời, nên phải có nhiều tiểu trình được tạo ra và được giải phóng trong, có thể đồng thời, một khoảng thời gian ngắn. Nếu server là một hệ thống multiprocessor thì các tiểu trình trong cùng một tác vụ có thể thực hiện đồng thời trên các processor khác nhau, do đó hiệu suất của hệ thống tăng lên. Sự hình thành các tiểu trình này cũng thật sự hữu ích trên các hệ thống uniprocessor, trong trường hợp một chương trình phải thực hiện nhiều chức năng khác nhau. Hiệu quả của việc sử dụng tiểu trình được

thấy rõ trong các ứng dụng cần có sự truyền thông giữa các tiến trình hoặc các chương trình khác nhau.

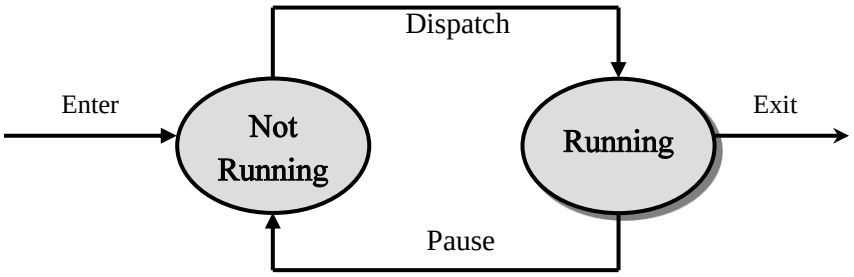
Các thao tác lập lịch và điều phối tiến trình của hệ điều hành thực hiện trên cơ sở tiểu trình. Nhưng nếu có một thao tác nào đó ảnh hưởng đến tấ cả các tiểu trình trong tác vụ thì hệ điều hành phải tác động vào tác vụ.

Vì tất cả các tiểu trình trong một tác vụ chia sẻ cùng một không gian địa chỉ, nên tất cả các tiểu trình phải được đưa vào trạng thái suspend tại cùng thời điểm. Tương tự, khi một tác vụ kết thúc thì sẽ kết thúc tất cả các tiểu trình trong tác vụ đó. Trạng thái suspend sẽ được giải thích ngay sau đây.

Các trạng thái tiến trình

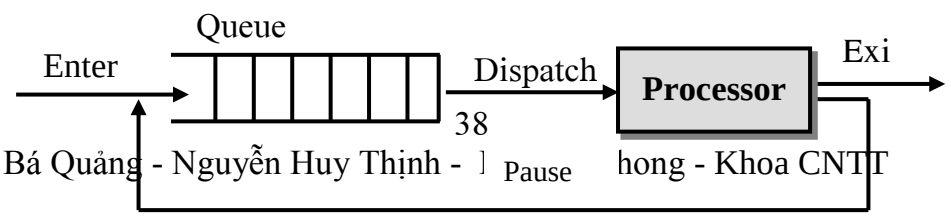
Từ khi được đưa vào hệ thống cho đến khi kết thúc tiến trình tồn tại ở các trạng thái khác nhau. Trạng thái của tiến trình tại một thời điểm được xác định bởi hoạt động hiện thời của tiến trình tại thời điểm đó.

Tiến trình hai trạng thái: Một số ít hệ điều hành chỉ cho phép tiến trình tồn tại ở một trong hai trạng thái: Not Running và Running. Khi hệ điều hành tạo ra một tiến trình mới, hệ điều hành đưa tiến trình đó vào hệ thống ở trạng thái Not Running, tiến trình ở trạng thái này để chờ được chuyển sang trạng thái Running. Vì một lý do nào đó, tiến trình đang thực hiện bị ngắt thì bộ điều phối tiến trình của hệ điều hành sẽ thu hồi lại processor của tiến trình này và chọn một tiến trình ở trạng thái Not running để cấp processor cho nó và chuyển nó sang trạng thái Running. Tiến trình bị thu hồi processor sẽ được chuyển về lại trạng thái Not running.



Hình 2.3.a. Sơ đồ chuyển trạng thái tiến trình

Tại một thời điểm xác định chỉ có duy nhất một tiến trình ở trạng thái Runnig, nhưng có thể có nhiều tiến trình ở trạng thái Not running, các tiến trình ở trạng thái Not running được chứa trong một hàng đợi (Queue). Tiến trình đang ở trạng thái



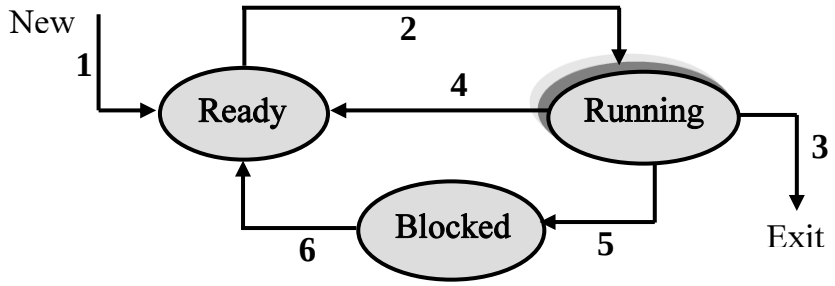
Hình 2.3.b. Sơ đồ chuyển tiến trình vào hàng đợi

Running bị chuyển sang trạng thái Not running sẽ được đưa vào hàng đợi. Hình vẽ sau đây mô tả việc chuyển trạng thái tiến trình trong các hệ điều hành sử dụng 2 trạng thái tiến trình.

Tiến trình ba trạng thái: Đa số hệ điều hành đều cho phép tiến trình tồn tại ở một trong ba trạng thái, đó là: ready, running, blocked:

- Trạng thái Ready (sẵn sàng): Ngay sau khi khởi tạo tiến trình, đưa tiến trình vào hệ thống và cấp phát đầy đủ tài nguyên (trừ processor) cho tiến trình, hệ điều hành đưa tiến trình vào trạng thái ready. Hay nói cách khác, trạng thái ready là trạng thái của một tiến trình trong hệ thống đang chờ được cấp processor để bắt đầu thực hiện.
- Trạng thái Running (thực hiện): Là trạng thái mà tiến trình đang được sở hữu processor để hoạt động, hay nói cách khác là các chỉ thị của tiến trình đang được thực hiện/ xử lý bởi processor.
- Trạng thái Blocked (khoá): Là trạng thái mà tiến trình đang chờ để được cấp phát thêm tài nguyên, để một sự kiện nào đó xảy ra, hay một quá trình vào/ra kết thúc.

Quá trình chuyển trạng thái của các tiến trình trong được mô tả bởi sơ đồ sau:



Hình 2.4.a. Sơ đồ chuyển trạng thái tiến trình

Trong đó:

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

2. (Dispatch) Tiến trình được cấp processor để bắt đầu thực hiện/ xử lý.

3. (Release) Tiến trình hoàn thành xử lý và kết thúc.

4. (Time_out) Tiến trình bị bộ điều phối tiến trình thu hồi processor, do hết thời gian được quyền sử dụng processor, để cấp phát cho tiến trình khác.

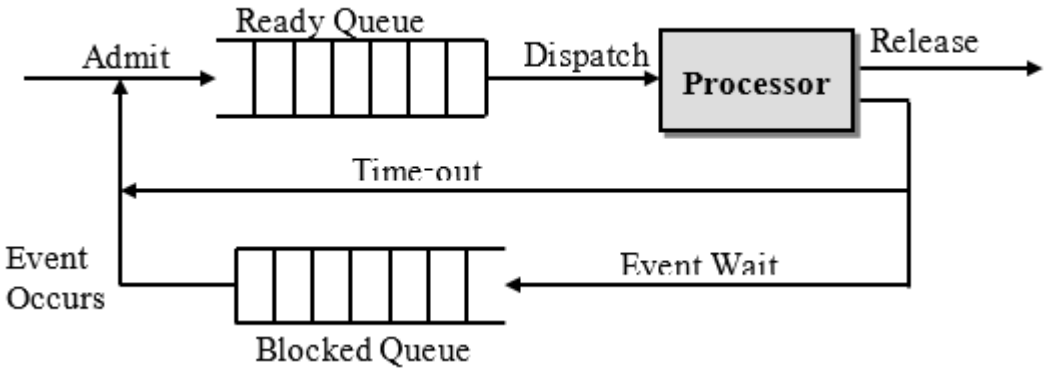
5. (Event wait) Tiến trình đang chờ một sự kiện nào đó xảy ra hay đang chờ một thao vào/ra kết thúc hay tài nguyên mà tiến trình yêu cầu chưa được hệ điều hành đáp ứng.

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

Bộ phận điều phối tiến trình thu hồi processor từ một tiến trình đang thực hiện trong các trường hợp sau:

- Tiến trình đang thực hiện hết thời gian (time-out) được quyền sử dụng processor mà bộ phận điều phối dành cho nó.
- Có một tiến trình mới phát sinh và tiến trình mới này có độ ưu tiên cao hơn tiến trình hiện tại.
- Có một tiến trình mới phát sinh và tiến trình này mới cần một khoảng thời gian của processor nhỏ hơn nhiều so với khoảng thời gian còn lại mà tiến trình hiện tại cần processor.

Tại một thời điểm xác định trong hệ thống có thể có nhiều tiến trình đang ở trạng thái Ready hoặc Blocked nhưng chỉ có một tiến trình ở trạng thái Running. Các tiến trình ở trạng thái Ready và Blocked được chứa trong các hàng đợi (Queue) riêng.

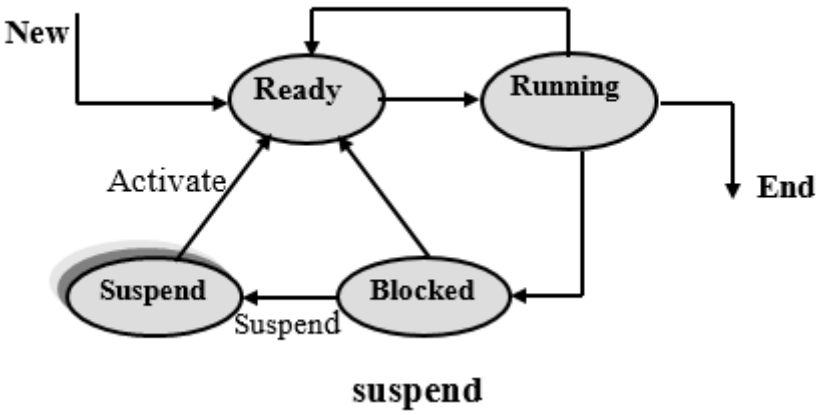


Hình 2.4.b. Sơ đồ chuyển tiến trình vào các hàng đợi

Có nhiều lý do để một tiến trình đang ở trạng thái running chuyển sang trạng thái blocked, do đó đa số các hệ điều hành đều thiết kế một hệ thống hàng đợi gồm nhiều hàng đợi, mỗi hành đợi dùng để chứa những tiến trình đang đợi cùng một sự

kiện nào đó.

Tiến trình 4 trạng thái: Trong môi trường hệ điều hành đa nhiệm thì việc tổ chức các Queue để lưu các tiến trình chưa thể hoạt động là cần thiết, nhưng nếu tồn tại quá nhiều tiến trình trong Queue, hay chính xác hơn trong bộ nhớ chính, sẽ dẫn đến trình trạng lãng phí bộ nhớ, không còn đủ bộ nhớ để nạp các tiến trình khác khi cần thiết. Mặt khác nếu các tiến trình trong Queue đang chiếm giữ tài nguyên của hệ thống, mà những tài nguyên này lại là những tài nguyên các tiến trình khác đang cần, điều này dẫn đến tình trạng sử dụng tài nguyên không hợp lý, làm cho hệ thống thiếu tài nguyên (thực chất là thừa) trầm trọng và có thể làm cho hệ thống tắc nghẽn. Với những lý do trên các hệ điều hành đa nhiệm thiết kế thêm một trạng thái tiến trình mới, đó là trạng thái Suspend (tạm dừng). Trạng thái này rất cần thiết cho các hệ thống sử dụng kỹ thuật Swap trong việc cấp phát bộ nhớ cho các tiến trình. Khái niệm Swap sẽ được đề cập đến trong chương Quản lý bộ nhớ của tài liệu này.



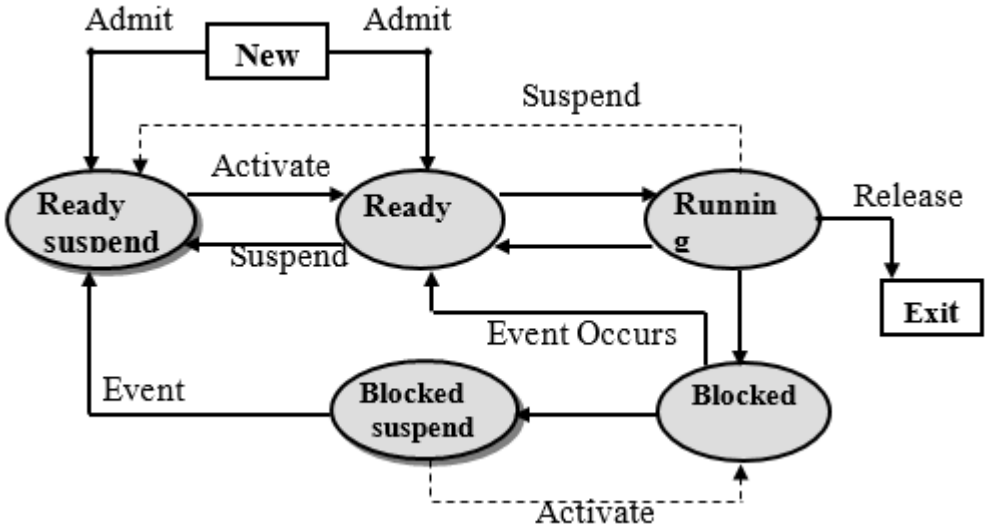
Hình 2.5.a. Sơ đồ chuyển trạng thái tiến trình có 4 trạng thái

Trạng thái Suspend là trạng thái của một tiến trình khi nó đang được lưu trữ trên bộ nhớ phụ, hay chính xác hơn đây là các tiến trình đang ở trong trạng thái blocked và/hoặc ready bị hệ điều hành chuyển ra đĩa để thu hồi lại không gian nhớ đã cấp cho tiến trình hoặc thu hồi lại tài nguyên đã cấp cho tiến trình để cấp cho một tiến trình khác đang rất cần được nạp vào bộ nhớ tại thời điểm hiện tại.

Tiến trình 5 trạng thái: Trong thực tế hệ điều hành thiết kế 2 trạng thái suspend, một trạng thái suspend dành cho các tiến trình từ blocked chuyển đến, trạng thái này được gọi là blocked-suspend và một trạng thái suspend dành cho các tiến trình từ ready chuyển đến, trạng thái này được gọi là ready-suspend.

Tới đây ta có thể hiểu các trạng thái tiến trình như sau:

- Ở trạng thái Ready tiến trình được định vị trong bộ nhớ chính và đang chờ được cấp processor để thực hiện.
- Ở trạng thái Blocked tiến trình được định vị trong bộ nhớ chính và đang đợi một sự kiện hay một quá trình I/O nào đó.
- Ở trạng thái Blocked-suspend tiến trình đang bị chứa trên bộ nhớ phụ (đĩa) và đang đợi một sự kiện nào đó.
- Ở trạng thái Ready-suspend tiến trình đang bị chứa trên bộ nhớ phụ nhưng sẵn sàng thực hiện ngay sau khi được nạp vào bộ nhớ chính.



Hình 2.5.b. Sơ đồ chuyển trạng thái tiến trình với 2 suspend

Sau đây chúng ta xem xét sự chuyển trạng thái tiến trình trong sơ đồ trên:

1. Blocked sang Blocked-suspend: nếu không còn tiến trình ready trong bộ nhớ chính và bộ nhớ chính không còn không gian nhớ trống thì phải có ít nhất một tiến trình blocked bị chuyển ra ngoài, blocked-suspend, để dành bộ nhớ cho một tiến trình không bị khoá (not blocked) khác.

2. Blocked-suspend sang Ready-suspend: một tiến trình đang ở trạng thái blocked-suspend được chuyển sang trạng thái ready-suspend khi sự kiện mà nó đợi đã xảy ra.

3. Ready-suspend sang Ready: có 2 lý do để hệ điều hành chọn khi chuyển một tiến trình ở trạng thái ready-suspend sang trạng thái ready:

- Không còn tiến trình ready trong bộ nhớ chính, hệ điều hành phải nạp một tiến trình mới vào để nó tiếp tục thực hiện
- Nếu có tiến trình ready-suspend có độ ưu tiên cao hơn so với các tiến trình ready hiện tại thì hệ điều hành có thể chuyển nó sang trạng thái ready để nó nhiều cơ hội để được thực hiện hơn.

4. Ready sang Ready suspend: Hệ điều hành thường chuyển các tiến trình blocked sang suspend hơn là các tiến trình ready, vì các tiến trình ở trạng thái blocked không thể thực hiện ngay lập tức nhưng lại chiếm nhiều không gian bộ nhớ chính hơn so với các tiến trình ở trạng thái ready. Tuy nhiên, nếu việc chọn tiến trình để chuyển sang suspend dựa vào 2 điều kiện: chiếm ít không gian bộ nhớ hơn và có độ ưu tiên

thấp hơn thì hệ điều hành có thể chuyển một tiến trình ready sang trạng thái suspend.

Như vậy với việc chuyển tiến trình sang trạng thái suspend hệ điều hành sẽ chủ động hơn trong việc cấp phát bộ nhớ và ngăn chặn các tình huống tắc nghẽn có thể xảy ra do sự tranh chấp về tài nguyên, nhờ vậy mà hệ điều hành tiết kiệm được bộ nhớ, chia sẻ được tài nguyên cho nhiều tiến trình và tăng được mức độ đa chương của hệ thống. Tuy nhiên, để có được những lợi ích trên hệ điều hành đã phải chi phí rất nhiều cho việc tạm dừng tiến trình. Hệ điều hành phải xem xét tiến trình nào được chọn để suspend, khi suspend một tiến trình hệ điều hành phải lưu lại tất cả các thông tin liên quan đến tiến trình đó (con trỏ lệnh, tài nguyên mà tiến trình đã được cấp, ...), hệ điều hành phải lựa chọn thời điển thích hợp để đưa tiến trình ra bộ nhớ ngoài, ... những thao tác đó sẽ làm chậm tốc độ thực hiện của toàn bộ hệ thống. Nhưng hệ điều hành vẫn phải sử dụng trạng thái suspend vì tăng mức độ đa chương của hệ thống là một trong những mục tiêu lớn của hệ điều hành.

2.2. Tài nguyên Găng (Critical Resource) và đoạn Găng (Critical Section)

2.2.1. Tài nguyên găng (Critical Resource)

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

Các tiến trình hoạt động đồng thời thường cạnh tranh với nhau trong việc sử dụng tài nguyên dùng chung. Hai tiến trình hoạt động đồng thời cùng ghi vào một không gian nhớ chung (một biến chung) trên bộ nhớ hay hai tiến trình đồng thời cùng ghi dữ liệu vào một file chia sẻ, đó là những biểu hiện của sự cạnh tranh về việc sử dụng tìa nguyên dùng chung của các tiến trình. Để các tiến trình hoạt động đồng thời không cạnh tranh hay xung đột với nhau khi sử dụng tài nguyên dùng chung hệ điều hành phải tổ chức cho các tiến trình này được độc quyền truy xuất/ sử dụng trên các tài nguyên dùng chung này.

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

chung, các file chia sẻ.

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

Ví dụ 1: Giả sử có một chương trình, trong đó có hai tiến trình P1 và P2 hoạt động đồng thời với nhau. Tiến trình P1 phải tăng biến Count lên 1 đơn vị, tiến trình P2 phải tăng biến Count lên 1 đơn vị, với mục đích tăng Count lên được 2 đơn vị.

Chương trình có thể thực hiện như sau:

1. Tiến trình P1 ghi nội dung biến toàn cục Count vào biến cục bộ L1
2. Tiến trình P2 ghi nội dung biến toàn cục Count vào biến cục bộ L2
3. Tiến trình P1 thực hiện L1:= L1 + 1 và Count := L1
4. Tiến trình P2 thực hiện L2:= L2 + 1 và Count := L2

Như vậy thoạt nhìn ta thấy rằng chắc chắn Count đã tăng được 2 đơn vị, nhưng trong thực tế có thể Count chỉ tăng được 1 đơn vị. Bởi vì, nếu P1 và P2 đồng thời nhận giá trị của Count (giả sử ban đầu Count = 4) vào L1 và L2, sau đó P1 tăng L1 lên 1 và P2 tăng L2 lên 1 (L1 = 5, L2 = 5), rồi sau đó cả P1 và P2 đồng thời ghi giá trị biến L của nó vào lại Count, thì Count chỉ tăng được 1 đơn vị, Count = 6. Đây là điều mà chương trình không mong muốn nhưng cả chương trình và hệ điều hành đều khó có thể phát hiện được.

Nguyên nhân ở trên là do 2 tiến trình P1 và P2 đồng thời truy xuất biến Count, cả khi nhận giá trị của count, lẫn khi ghi giá trị vào Count. Trong trường hợp này nếu hệ điều hành không cho phép hai tiến trình P1 và P2 đồng thời truy xuất Count, hoặc hệ điều hành cho phép mỗi tiến trình được độc quyền truy xuất Count trong đoạn code sau, thì lỗi trên sẽ không xảy ra.

```
P1:   Begin                    P2:   Begin
L1 :=  Count;                  L2 :=  Count;
L1 := L1 + 1;                  L2 := L2 + 1;
Count := L1;                   Count := L2;
      End;                           End;
```

Trong trường hợp này tài nguyên găng là biến count.

Ví dụ 2: Giả sử có một ứng dụng Kế toán, hoạt động trong môi trường đa nhiệm, đa người sử dụng. Mỗi người sử dụng trong môi trường này khi cần thực hiện thao tác rút tiền từ trong tài khoản chung thì phải khởi tạo một tiến trình, tạm gọi là tiến trình rút tiền, tiến trình rút tiền chỉ có thể thực hiện được thao tác rút tiền khi số tiền cần rút nhỏ hơn số tiền còn lại trong tài khoản chung. Trong môi trường này có thể có nhiều người sử dụng đồng thời thực hiện thao tác rút tiền từ tài khoản chung

của hệ thống.

Như vậy các tiến trình rút tiền, giả sử có hai tiến trình rút tiền P1 và P1, có thể hoạt động đồng thời với nhau và cùng chia sẻ không gian nhớ lưu trữ biến **Tài khoản**, cho biết số tiền còn trong tài khoản dùng chung của hệ thống. Và mỗi tiến trình rút tiền khi muốn rút một khoảng tiền từ tài khoản **(Tiền rút)** thì phải thực hiện kiểm tra **Tài khoản** sau đó mới thực hiện việc rút tiền. Tức là mỗi tiến trình rút tiền, khi cần rút tiền đều phải thực hiện đoạn code sau đây:

```
IF   (Tài khoản - Tiền rút >= 0)          {kiểm tra tài khoản}
        Tài khoản := Tài khoản - Tiền rút  {thực hiện rút tiền}
Else
        Thông báo lỗi                      {không thể rút tiền}
EndIf;
```

Nếu tại một thời điểm nào đó:

- Trong tài khoản còn 800 ngàn đồng **(Tài khoản = 800).**
- Tiến trình rút tiền P1 cần rút 500 ngàn đồng **(Tiền rút = 500).**
- Tiến trình rút tiền P2 cần rút 400 ngàn đồng **(Tiền rút = 400).**
- Tiến trình P1 và P2 đồng thời rút tiền.

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

1. P1 được cấp processor để thực hiện việc rút tiền: P1 thực hiện kiểm tra tài khoản: **Tài khoản - Tiền rút = 800 -500 = 300 > 0**, P1 ghi nhận điều này và chuẩn bị rút tiền.
2. Nhưng khi P1 chưa kịp rút tiền thì bị hệ điều hành thu hồi lại processor, và hệ điều hành cấp processor cho P2. P1 được chuyển sang trạng thái ready.
3. P2 nhận được processor, được chuyển sang trạng thái running, nó bắt đầu thực hiện việc rút tiền như sau: kiểm tra tài khoản: **Tài khoản - Tiền rút = 800 - 400 = 500 >= 0**, P2 ghi nhận điều này và thực hiện rút tiền:

   Tài khoản = Tài khoản - Tiền rút = 800 - 400 = 400.
4. P2 hoàn thành nhiệm vụ rút tiền, nó kết thúc xử lý và trả lại processor cho hệ điều hành. Hệ điều hành cấp lại processor cho P1, tái kích hoạt lại P1

để nó tiếp tục thao tác rút tiền.

5. Khi được hoạt động trở lại P1 thực hiện ngay việc rút tiền mà không thực hiện việc kiểm tra tài khoản (vì đã kiểm tra trước đó):

Tài khoản = Tài khoản - Tiền rút = 400 - 500 = -100.

6. P1 hoàn thành nhiệm vụ rút tiền và kết thúc tiến trình.

Như vậy cả 2 tiến trình P1 và P2 đều hoàn thành việc rút tiền, không thông báo lỗi, mà không gặp bất kỳ một lỗi hay một trở ngại nào. Nhưng đây là một lỗi nghiêm trọng đối với ứng dụng, vì không thể rút một khoảng tiền lớn hơn số tiền còn lại trong tài khoản, hay **Tài khoản** không thể nhận giá trị âm.

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

Trong trường hợp này tài nguyên găng là biến Tài khoản.

Ví dụ 3: Giả sử một hệ điều hành đa nhiệm, cung cấp cho các tiến trình của các chương trình người sử dụng một thủ tục Echo. Thủ tục Echo này cho phép các tiến trình nhận một kí tự từ bàn phím rồi đưa kí tự này lên màn hình, mỗi khi gọi nó. Tất cả các tiến trình của chương trình người sử dụng trong hệ thống có thể đồng thời gọi Echo mỗi khi cần đưa một kí tự từ bàn phím lên màn hình. Sau đây là code của thủ tục Echo:

```
Procedure Echo;
Var
    out, in: chracter;
Begin
        Input(In, keyboard);        {Input là hàm nhập, nó nhận kí tự}
        Out:=In;                    {từ bàn phím đưa vào In. Output là}
        Output(Out, Screen);        {hàm xuất, nó đưa kí tự từ biến Out}
End;                                {lên màn hình}
```

Để tiết kiệm bộ nhớ hệ điều hành nạp Echo vào không gian nhớ toàn cục của hệ thống và các tiến trình sẽ chia sẻ không gian nhớ chứa thủ tục Echo này. Sự chia sẻ này là cần thiết và hữu ích, nhưng các tiến trình, hai tiến trình P1 và P2, có thể không đạt được mục tiêu khi gọi Echo, có thể tiến trình P1 gõ kí tự A nhưng màn hình lại xuất hiện kí tự B, B là kí tự của tiến trình P2. Bởi vì hệ thống có thể xảy ra trường hợp sau:

1. Tiến trình P1 gọi thủ tục Echo và bị ngắt ngay lập tức sau khi hàm nhập

Input được thực hiện. Tại thời điểm này, kí tự vừa được nhập gần đây nhất là A, được lưu trữ trong biến In.

2. Tiến trình P2 được kích hoạt và gọi thủ tục Echo, và thủ tục được chạy cho đến khi kết thúc. Giả sử đã nhập và xuất kí tự B ra màn hình.

3. Tiến trình P1 được tiếp tục trở lại. Lúc này giá trị A của biến In đã bị ghi đè, có thể là kí tự B của tiến trình P2, biến In = B. Tiến trình P1 tiếp tục công việc của thủ tục Echo, Out:= In và Out = B. Sau đó hàm xuất Output sẽ đưa giá trị của biến out lên màn hình. Tức là trên màn hình xuất hiện kí tự B. Đây là điều mà tiến trình P1 không hề mong muốn.

Như vậy là kí tự A bị mất, nhưng kí tự B lại xuất hiện hai lần. Bản chất của vấn đề này là nằm ở biến toàn cục In (tài nguyên găng là biến In). Vì hệ điều hành đã để cho nhiều tiến trình hoạt động đồng thời trên hệ thống có quyền truy xuất và truy xuất đồng thời vào biến này. Để tránh lỗi này hệ điều hành cần phải có cơ chế đề bảo vệ biến toàn cục dùng chung và chỉ cho phép một tiến trình duy nhất điều khiển các code truy xuất đến nó. Nếu hệ điều hành chấp nhận quy tắc: tại một thời điểm chỉ có một tiến trình được phép sử dụng thủ tục Echo và thủ tục này phải chạy cho đến khi hoàn thành mới được trao cho tiến trình khác. Thì lỗi trên sẽ không còn xuất hiện nữa. Việc sử dụng thủ tục Echo của các tiến trình P1 và P2 có thể xảy ra theo thứ tự như sau:

1. Tiến trình P1 gọi thủ tục Echo và bị dừng lại ngay sau khi hàm input được thực hiện xong. Giả sử In = A.

2. Tiến trình P2 được kích hoạt và gọi thủ tục Echo. Nhưng vì tiến trình P1 còn đang ở trong thủ tục này, cho dù đang bị treo, nên P2 phải được chuyển sang trạng thái blocked để chờ thủ tục Echo rỗi.

3. Một khoảng thời gian sau, tiến trình P1 được tái kích hoạt trở lại. P1 tiếp tục thủ tục echo cho đến khi hoàn thành. Tức là, đã hiển thị kí tự A lên màn hình.

4. Khi kết thúc P1 trả lại thủ tục echo. Khi đó P2 toàn quyền sử dụng thủ tục Echo để nhập và hiển thị kí tự lên màn hình.

Trường hợp này không xảy ra lỗi là do tiến trình P2 không tiếp tục thủ tục Echo, mặc dù đã gọi, vì nó biết P1 đã đang ở trong thủ tục Echo. Chúng ta nên lưu ý điều này, điều này sẽ được thảo luận trong mục các phương pháp điều độ tiến trình qua đoạn găng ngay sau đây.

Qua các ví dụ trên ta thấy rằng trong các hệ thống đa chương, đa người sử dụng thường xảy ra hiện tượng, nhiều tiến trình đồng thời cùng đọc/ghi dữ liệu vào một vùng nhớ, nơi chứa các biến của chương trình, và nếu không có sự can thiệp của

hệ điều hành thì có thể gây hậu quả nghiêm trọng cho ứng dụng và cho cả hệ thống. Để ngăn chặn các tình huống trên hệ điều hành phải thiết lập cơ chế độc quyền truy xuất trên trên tài nguyên dùng chung. Tức là, tại mỗi thời điểm chỉ có một tiến trình duy nhất được phép truy xuất trên các tài nguyên dung chung. Nếu có nhiều tiến trình hoạt động đồng thời cùng yêu cầu truy xuất tài nguyên dùng chung thì chỉ có một tiến trình được chấp nhận truy xuất, các tiến trình khác phải xếp hàng chờ để được truy xuất sau.

Chúng ta cũng thấy rằng nguyên nhân tiềm ẩn của sự xung đột giữa các tiến trình hoạt động đồng thời khi sử dụng tài nguyên găng là: các tiến trình này hoạt động đồng thời với nhau một cách hoàn toàn độc lập và không trao đổi thông tin với nhau nhưng sự thực thi của các tiến trình này lại ảnh hưởng đến nhau. Trường hợp lỗi trong ví dụ 3 ở trên minh chứng cho điều này.

2.2.3. Đoạn găng (Critical Section)

Đoạn mã lệnh (code) trong các tiến trình đồng thời, có tác động đến các tài nguyên có thể trở thành tài nguyên găng được gọi là đoạn găng hay miền găng. Tức là, các đoạn code trong các chương trình dùng để truy cập đến các vùng nhớ chia sẻ, các tập tin chia sẻ được gọi là các đoạn găng.

Trong ví dụ 2 ở trên, đoạn code sau đây là đoạn găng:

```
{   IF   (Tài khoản - Tiền rút >= 0)
            Tài khoản := Tài khoản - Tiền rút     }
```

Trong ví dụ 1 ở trên có hai đoạn găng là:

{ L1 := Count và Count := L1 }.

Để hạn chế các lỗi có thể xảy ra do sử dụng tài nguyên găng, hệ điều hành phải điều khiển các tiến trình sao cho, tại một thời điểm chỉ có một tiến trình nằm trong đoạn găng, nếu có nhiều tiến trình cùng muốn vào (thực hiện) đoạn găng thì chỉ có một tiến trình được vào, các tiến trình khác phải chờ, một tiến trình khi ra khỏi (kết thúc) đoạn găng phải báo cho hệ điều hành và/hoặc các tiến trình khác biết để các tiến trình này vào đoạn găng, vv. Các công tác điều khiển tiến trình thực hiện đoạn găng của hệ điều hành được gọi là *điều độ tiến trình qua đoạn găng*. Để công tác điều độ tiến trình qua đoạn găng được thành công, thì cần phải có sự phối hợp giữa vi xử lý, hệ điều hành và người lập trình. Vi xử lý đưa ra các chỉ thị, hệ điều hành cung cấp các công cụ để người lập trình xây dựng các sơ đồ điều độ hợp lý, để đảm bảo sự độc quyền trong việc sử dụng tài nguyên găng của các tiến trình.

Trong phần sau đây chúng ta sẽ tìm hiểu về các phương pháp và các sơ đồ điều độ tiến trình qua đoạn găng. Nhưng trước hết ở đây chúng ta chấp nhận một mẫu chương trình được sử dụng trong các sơ đồ điều độ tiến trình. Mẫu chương trình này

mang tính chất trừu tượng, dùng để minh hoạ cho các ý tưởng điều độ. Rất ít ngôn ngữ lập trình hỗ trợ cú phát viết chương trình điều độ này. Mặc dầu đã cung cấp đầy đủ các công cụ điều độ tiến trình cho người lập trình, nhưng các hệ điều hành hiện nay đều tổ chức điều độ tiến trình ngay trong lõi (kernel) của nó nên người lập trình ít quan tâm đến tổ chức điều độ tiến trình khi lập trình. Sau đây là sơ đồ điều độ minh hoạ:

```
Program MultualExclution;
Const
        N = …..    /*số lượng tiến trình */
{-------------------------------------------------}
Procedure P(i: integer);
Begin
        Repeat
                EnterCritical(R);       {kiểm tra và xác lập quyền vào đoạn găng}
                <Đoạn găng của P>;
                ExitCritical(R);                {xác lập khi rời đoạn găng}
                <Đoạn không găng của>;
        Until  .F.
End;
{-------------------------------------------------}
BEGIN                           {*chương trình chính chứa các tiến trình đồng thời*}
        PerBegin
                P(1);
                P(2);
                …..
                P(n);
        ParEnd;
END.
{-------------------------------------------------}
```

Sơ đồ trên tổ chức điều độ cho n tiến trình P, n tiến trình này hoạt đồng đông thời với nhau và chia sẻ tài nguyên dùng chung R. Mỗi tiến trình trong trường hợp này có một đoạn găng với tài nguyên R. Để tổ chức truy xuất độc quyền trên tài nguyên găng, mỗi tiến trình trước khi vào đoạn găng tiến trình phải gọi thủ tục EnterCritical để thiết lập quyền vào đoạn găng, để báo cho các tiến trình biết là tiến

trình hiện tại đang ở trong đoạn găng. Để ra khỏi đoạn găng mỗi tiến trình phải gọi thủ tục ExitCritical, để báo cho các tiến trình khác biết là tiến trình hiện tại đã ra khỏi đoạn găng.

2.2.4. Yêu cầu của công tác điều độ qua đoạn găng

Trước hết chúng ta lưu ý lại rằng, nhiệm vụ điều độ tiến trình phải là sự phối hợp giữ phần cứng vi xử lý, hệ điều hành, ngôn ngữ lập trình và người lập trình, trong đó nhiệm vụ chính là của hệ điều hành và người lập trình. Vi xử lý, hệ điều hành và ngôn ngữ lập trình cung cấp các công cụ để hệ điều hành và/hoặc người lập trình tổ chức sơ đồ điều độ. Hệ điều hành sẽ giám sát và tổ chức thực hiện các sơ đồ điều độ này. Cho dù nhiệm vụ điều độ là của thành phần nào, thì tất cả phải đạt được các yêu cầu sau:

1. Tại một thời điểm không thể có hai tiến trình nằm trong đoạn găng.
2. Nếu có nhiều tiến trình đồng thời cùng xin được vào đoạn găng thì chỉ có một tiến trình được phép vào đoạn găng, các tiến trình khác phải xếp hàng chờ trong hàng đợi.
3. Tiến trình chờ ngoài đoạn găng không được ngăn cản các tiến trình khác vào đoạn găng.
4. Không có tiến trình nào được phép ở lâu vô hạn trong đoạn găng và không có tiến trình phải chờ lâu mới được vào đoạn găng (chờ trong hàng đợi).
5. Nếu tài nguyên găng được giải phóng thì hệ điều hành có nhiệm vụ đánh thức các tiến trình trong hàng đợi ra để tạo điều kiện cho nó vào đoạn găng.

Trước khi tìm hiểu về các giải pháp điều độ tiến trình qua đoạn găng chúng ta cần lưu ý một lần nữa rằng: nguyên lý cơ bản của điều độ là tổ chức truy xuất độc quyền trên tài nguyên găng, nhưng sự bắt buộc độc quyền này còn tồn tại hai hạn chế lớn:

1. Có thể dẫn đến tắc nghẽn (Deadlock) trong hệ thống. Chúng ta sẽ tìm hiểu về tắc nghẽn sau, bây gời chúng ta hãy xem một ví dụ về tắc nghẽn: Giả như có hai tiến trình P1 và P2, và hai tài nguyên găng R1 và R2, mỗi tiến trình đều cần truy xuất đến để mã thực hiện một hàm của nó. Và trường hợp sau đây hoàn toàn có thể xảy ra: R1 đang được giao cho P2, R2 được giao cho P1. Mỗi tiến trình đều chờ đợi được sử dụng tài nguyên thứ hai. Không một tiến trình nào giải phóng tài nguyên mà nó đang sở hữu cho đến khi có nhận được tài nguyên còn lại và thực hiện đoạn găng của nó. Cả hai tiến trình đó đều bị tắc nghẽn.
2. Các tiến trình có thể bị đói (Stravation) tài nguyên: Ví dụ sau đây cho thấy sự đói tài nguyên của các tiến trình trên hệ thống: Giả sử rằng có 3 tiến trình P1, P2, P3, mỗi tiến trình đều cần truy xuất định kỳ đến tài nguyên R. Xét trường hợp P1

đang sở hữu tài nguyên còn hai tiến trình P2, P3 phải chờ đợi tài nguyên đó. Khi mà P1 thoát khỏi đoạn găng của nó, cả P2 lẫn P3 đều có thể được chấp nhận truy xuất đến R. Giả sử rằng P3 được truy xuất R, sau đó trước khi P3 kết thúc đoạn găng của nó P1 lại một lần nữa cần truy xuất, và giả như P1 được truy xuất sau khi P3 kết thúc đoạn găng, và nếu như P1, P3 thay nhau nhận được quyền truy xuất thì P2 hầu như không thể truy cập đến tài nguyên, cho dù không có sự tắc nghẽn nào xảy ra.

Điều độ tiến trình qua đoạn găng

Các giải pháp phần cứng

Dùng cặp chỉ thị STI & CLI

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

Trong sơ đồ điều độ này tiến trình P_i được viết như sau:

```
Procedure P(i: integer);
Begin
        Repeat
                CLI;                    {cấm ngắt trước khi vào đoạn găng}
                <Đoạn găng của P>;
                STI;                    {mở ngắt khi ra khỏi đoạn găng }
                <Đoạn không găng>;
        Until  .F.
End;
{---------------------------------------------}
```

Sơ đồ trên cho thấy, khi tiến trình ở trong đoạn găng nó không hề bị ngắt, do đã cấm ngắt phát sinh, nên nó được độc quyền sử dụng tài nguyên găng cho đến khi ra khỏi đoạn găng.

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

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

Dùng chỉ thị TSL (Test and set)

Trong ví dụ 2 ở trên ta đã thấy, nguyên nhân của lỗi là do hai thao tác kiểm tra tài khoản và rút tiền, bị tách rời nhau. Để tổ chức điều độ cho những trường hợp như vậy, một số vi xử lý 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 đươc, gọi là Test and Set lock (TSL). TSL được định nghĩa như sau :

```
Function  TestAndSetLock(Var I:Integer):Boolean;
    Begin
        IF I = 0 Then
            Begin
                I := 1;                          {hai lệnh này không}
                TestAndSetLock:=True;            {thể tách rời}
            End
        Else
            TestAndSetLock := False
    End;
    {-----------------------------------------------}
```

Để tổ chức điều độ tiến trình với TSL chương trình phải sử dụng biến chia sẻ Lock, khởi gán bằng 0. Theo đó, mỗi tiến trình trước khi vào đoạn găng phải kiểm tra giá trị của Lock. Nếu Lock = 0 thì vào đoạn găng. Nếu Lock = 1 thì phải đợi cho đến khi Lock = 0. Như vậy, trước khi vào đoạn găng tiến trình phải gọi hàm TestAndSetLock, để kiểm tra giá trị trả về của hàm này:

- Nếu bằng False, là đang có một tiến trình trong đoạn găng, thì phải chờ cho đến khi hàm trả về True, có một tiến trình vừa ra khỏi đoạn găng.
- Nếu bằng True, thì tiến trình sẻ vào đoạn găng để sử dụng tài nguyên găng. Khi kết thúc sử dụng tài nguyên găng ra khỏi đoạn găng thì tiến trình phải đặt lại gía trị của Lock, Lock = 0, để các tiến trình khác có thể vào đoạn găng.

Nên nhớ rằng TestAndSetLock là chỉ thị của processor, nên hệ thống đã tổ chức thực hiện độc quyền cho nó. Tức là, các thao tác mà hệ thống phải thực hiện trong chỉ thị này là không thể tách rời nhau.

Trong sơ đồ điều độ này tiến trình P được viết như sau:

```
Procedure P(Lock: integer);
Begin
    Repeat
        While (TestAndSetlock(lock))  DO;
        <Đoạn găng của P>;
        Lock:= 0;
        <Đoạn không găng>;
    Until  .F.
End;
{-----------------------------------------}
```

Sơ đồ này đơn giản, dễ cài đặt nhưng cần phải có sự hỗ trợ của vi xử lý. Ngoài ra nó còn một hạn chế lớn là gây lãng phí thời gian xử lý của processor do tồn tại hiện tượng chờ đợi tích cực trong sơ đồ (While (TestAndSetlock(lock)) DO;). Hiện tượng chờ đợi tích cực là hiện tượng processor chỉ chờ một sự kiện nào đó xảy ra mà không làm gì cả.

Tóm lại: Việc sử dụng các chỉ thị phần cứng đặc biệt để tổ chức điều độ tiến trình qua đoạn găng, hay còn gọi là tổ chức truy xuất độc quyền trên tài nguyên găng, có những thuận lợi và bất lợi sau đây:

Thuận lợi:

- Nó thích hợp với một số lượng bất kỳ các tiến trình cả trên hệ hệ thống Uniprocessor và hệ thống Multiprocessor.
- Nó khá đơn giản cho nên dễ xác định độ chính xác.
- Nó có thể được sử dụng để hỗ trợ cho nhiều đoạn găng; mỗi đoạn găng có thể định nghĩa cho nó một biến riêng.

Bất lợi:

- Trong khi một tiến trình đang chờ đợi được vào đoạn găng thì nó tiếp tục làm tốn thời gian xử lý của processor, mà ta gọi là chờ đợi tích cực.
- Sự đói tài nguyên có thể xảy ra. Khi một tiến trình rời khỏi một đoạn găng, bộ phận điều độ tiến trình phải chọn một tiến trình trong số nhiều tiến trình ngoài đoạn găng để cho nó vào đoạn găng. Việc chọn này có thể dẫn đến hiện tượng có một tiến trình đợi mãi mà không thể vào đoạn găng được.
- Sự tắc nghẽn có thể xảy ra. Hãy xét một tình huống trên một hệ thống uniprocessor. Tiến trình P1 thực thi chỉ thị đặc biệt (TesAndSetLock, Exchange) và vào đoạn găng của nó. P1 sau đó bị ngắt để nhường processor cho P2, P2 là tiến trình

có độ ưu tiên cao hơn. Nếu như P2 cũng định sử dụng tài nguyên như P1, P2 sẽ bị từ chối truy xuất bởi vì cơ chế độc quyền. Do đó P2 sẽ đi vào vòng lặp busy-waitting. Tuy nhiên, P1 sẽ không bao giờ được cấp processor để tiếp tục vì nó có độ ưu tiên thấp hơn so với P2.

Các giải pháp dùng biến khoá

Dùng biến khoá chung

Xuất phát từ nguyên tắc cơ bản của tổ chức độc quyền là, tại mỗi thời điểm chỉ có duy nhất một tiến trình có thể truy xuất đến một vùng nhớ chia sẻ, các hệ điều hành sử dụng biến khoá chung để tổ chức truy xuất độc quyền trên tài nguyên găng. Phương pháp này còn gọi là phương pháp Busy and Waitting (bận và đợi), nó được nhà toán học người Hà Lan tên là Dekker đề xuất.

Với mỗi tài nguyên găng, hệ điều hành dùng một biến chung để điều khiển việc sử dụng tài nguyên này của các tiến trình đồng thời. Tạm gọi là biến chung này là Lock, Lock được chia sẻ cho nhiều tiến trình và được khởi gán = 0.

Theo đó, mỗi tiến trình trước khi vào đoạn găng phải kiểm tra giá trị của Lock:

- Nếu Lock = 1, tức là đã có tiến trình nào đó trong đoạn găng, thì tiến trình phải chờ cho đến khi Lock = 0 (có thể chuyển sang trạng thái blocked để chờ).
- Nếu Lock = 0, tức là không có tiến trình nào trong đoạn găng, thì tiến trình thiết lập quyền vào đoạn găng, đặt Lock = 1, và vào đoạn găng. Tiến trình vừa ra khỏi đoạn găng phải đặt Lock = 0, để các tiến trình khác có thể vào đoạn găng.

Trong sơ đồ điều độ này tiến trình P được viết như sau:

```
Procedure P(Lock: integer);
Begin
        Repeat
                While Lock = 1  DO ;        {đợi cho đến khi Lock = 0}
                Lock :=1;                   {thiết lập quyền vào đoạn găng}
                <Đoạn găng của P>; {vào đoạn găng}
                Lock:= 0;                   {thông báo là đã rời đoạn găng }
                <Đoạn không găng>;
        Until  .F.
End;
{-----------------------------------------------}
```

Sơ đồ điều độ dùng biến khoá chung này đơn giản, dễ xây dựng nhưng vẫn

xuất hiện hiện tượng chờ đợi tích cực, khi chờ cho đến khi Lock = 0 (While Lock = 1 DO;). Hiện tương chờ đợi tích cực gây lãng phí thời gian của processor.

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

Dùng biến khoá riêng

Để khắc phục hạn chế của phương pháp dùng biến chung, các hệ điều hành có thể dùng giải pháp biến riêng để tổ chức điều độ tiến trình. Mỗi tiến trình sử dụng một biến khoá Lock riêng, tương ứng với một tài nguyên găng trong hệ thống. Biến khoá riêng của tất cả các tiến trình đều được khởi gán bằng 0, tức là chưa vào đoạn găng

Theo đó, mỗi tiến trình trước khi vào đoạn găng ứng với một tài nguyên găng nào đó thì trước hết phải kiểm tra biến khoá riêng, tương ứng với tài nguyên găng mà tiến trình muốn truy xuất, của tất cả các tiến trình còn lại:

- Nếu tồn tại một biến khoá riêng của một tiến trình nào đó bằng 1, Lock = 1, tức là đã có một tiến trình nào đó ở trong đoạn găng, thì tiến trình phải chờ ngoài đoạn găng cho đến khi tất cả biến khoá riêng = 0.
- Nếu tất cả các biến khóa riêng của các tiến trình đều = 0, Lock = 0, tức là không có tiến trình nào trong đoạn găng, thì tiến trình thiết lập quyền vào đoạn găng, đặt Lock = 1, và vào đoạn găng. Tiến trình vừa ra khỏi đoạn găng phải đặt Lock = 0, để các tiến trình khác có thể vào đoạn găng.

Sau đây là sơ đồ điều độ dùng biến khoá riêng cho hai tiến trình đồng thời P1 và P2. Hai tiến trình này dùng hai biến khoá riêng là Lock1 và Lock2:

```
Program MultualExclution;
Const       N:2;
Var
      Lock1, Lock2: byte;
BEGIN
      Lock1 = 0; Lock2 = 0;
ParBegin
      P1: Repeat                          {tiến trình P1}
            While Lock2 = 1 Do ;          {P2 đang ở trong đoạn găng }
            Lock1 := 1;                   {P1 thiết lập quyền vào đoạn găng}
            <Đoạn găng của P1>;
            Lock1 := 0;                   {P1 ra khỏi đoạn găng}
            <Đoạn không găng của P1>;
          Until .F.
      P2: Repeat                          {tiến trình P2}
            While Lock1 = 1 Do;           {P1 đang ở trong đoạn găng }
            Lock2 := 1;                   {P2 thiết lập quyền vào đoạn găng}
            <Đoạn găng của P2>;
            Lock2 := 0;                   {P2 ra khỏi đoạn găng}
            <Đoạn không găng của P2>;
          Until .F.
ParEnd
END.
{-------------------------------------------------------}
```

Sơ đồ này đơn giản dễ cài đặt. Một tiến trình nào đó ở ngoài đoạn găng bị blocked sẽ không ngăn cản được các tiến trình khác vào đoạn găng, nhưng nếu tiến trình trong đoạn găng bị lỗi không thể ra khỏi đoạn găng , Lock luôn luôn = 0, thì các tiến trình khác sẽ không được quyền vào đoạn găng.

Phương pháp này vẫn còn tồn tại hiện tượng chờ đợi tích cực và sơ đồ điều độ sẽ trở nên phức tạp khi có nhiều hơn hai tiến trình muốn vào đoạn găng.

Sơ đồ này có thể xảy ra một lỗi nghiêm trọng đó là: Có thể có hai tiến trình cùng nằm trong đoạn găng. Nguyên nhân của lỗi này là do việc kiểm tra quyền vào

đoạn găng và và việc xác lập quyền vào đoạn găng của tiến trình bị tách rời khi thực hiện. Tức là, P1 và P2 có thể bị điều phối thực hiện theo thứ tự sau:

P1 được cấp processor: P1 thực thi vòng lặp While và tìm xem thử Lock2 = 1 không. Khi P1 vừa nhìn thấy Lock2 = 0, thì bị thu hồi processor.

P2 được cấp processor: P2 thực thi vòng lặp While và tìm xem thử Lock1 = 1 không. Khi P2 vừa nhìn thấy Lock1 = 0, thì bị thu hồi processor.

P1 được cấp processor trở lại: P1 không kiểm tra lại Lock2 mà chỉ đặt Lock1 = 1 và vào đoạn găng của nó. Khi vừa vào đoạn găng thì bị thu hồi processor.

P2 được cấp processor trở lại: P2 không kiểm tra lại Lock1 mà chỉ đặt Lock2 = 1 và vào đoạn găng của nó.

Rõ ràng với thực tế này thì cả P1 và P2 đều nằm trong đoạn găng. Và chúng ta đã biết điều gì sẽ xảy ra khi hai tiến trình đồng thời truy xuất tài nguyên găng trong các ví dụ về tài nguyên găng ở trên.

Nhiều nhà thiết kế hệ điều hành đã cải tiến sơ đồ điều độ ở trên để khắc phục hạn chế trên đây và một số hạn chế khác của nó.

Các giải pháp được hỗ trợ bởi hệ điều hành và ngôn ngữ lập trình

Các giải pháp trên tồn tại hiện tượng chờ đợi tích cực, gây lãng phí thời gian xử lý của processor. Điều này có thể khắc phục bằng một nguyên tắc rất cơ bản: nếu một tiến trình khi chưa đủ điều kiện vào đoạn găng thì được chuyển ngay sang trang thái blocked để nó trả lại processor cho hệ thống, để hệ thống cấp cho tiến trình khác. Để thực hiện được điều này cần phải có sự hỗ trợ của hệ điều hành và các ngôn ngữ lập trình để các tiến trình có thể chuyển trạng thái của nó. Hai thủ tục **Sleep** và **Wakeup** được hệ điều hành cung cấp để sử dụng cho mục đích này:

- Khi tiến trình chưa đủ điều kiện vào đoạn găng nó sẽ thực hiện một lời gọi hệ thống để gọi Sleep để chuyển nó sang trạng thái blocked, và tiến trình được gọi này đưa vào hàng đợi để đợi cho đến khi có một tiến trình khác gọi thủ tục Wakeup để giải phóng nó ra khỏi hàng đợi và có thể đưa nó vào đoạn găng.
- Một tiến trình khi ra khỏi đoạn găng phải gọi Wakeup để đánh thức một tiến trình trong hang đợi blocked ra để tạo điều khiện cho tiến trình này vào đoạn găng.

Như vậy giải pháp này được áp dụng trên nhóm các tiến trình hoạt động đồng thời có trao đổi thông tin với nhau, và các tiến trình phải hợp thác với nhau để hoàn thành nhiệm vụ. Các tiến trình này liên lạc với nhau bằng cách gởi tín hiệu cho nhau. Một tiến trình trong hệ thống này có thể bị buộc phải dừng (bị blocked) cho đến khi nhận được một tín hiệu nào đó từ tiến trình bên kia, đó là tiến trình hợp tác với nó.

Thực tế đã chỉ ra được rằng, nếu chỉ dùng hai thủ tục trên thì sơ đồ điều độ sẽ

không đáp ứng được các yêu cầu của công tác điều độ, do đó khi cài đặt các hệ điều hành chỉ sử dụng ý tưởng của Sleep và Wakeup. Sau đây là các giải pháp sử dụng ý tưởng của Sleep và Wakeup.

Giải pháp dùng Semaphore (đèn báo)

Giải pháp này được Dijkstra đề xuất vào năm 1965. Semaphore (sự đánh tín hiệu bằng cờ) được định nghĩa để sử dụng trong các sơ đồ điều độ như sau:

- Semaphore (sự đánh tín hiệu bằng cờ) S là một biến nguyên, khởi gán bằng một giá trị không âm, đó là khả năng phục vụ của tài nguyên găng tương ứng với nó.
- Ứng với S có một hàng đợi F(s) để lưu các tiến trình đang bị blocked trên S.
- Chỉ có hai thao tác Down và Up được tác động đến semaphore (sự đánh tín hiệu bằng cờ) S. Down giảm S xuống một đơn vị, Up tăng S lên một đơn vị.
- Mỗi tiến trình trước khi vào đoạn găng thì phải gọi Down để kiểm tra và xác lập quyền vào đoạn găng. Khi tiến trình gọi Down(S) thì hệ thống sẽ thực hiện như sau: S := S -1, nếu S > = 0 thì tiến trình tiếp tục xử lý và vào đoạn găng, nếu S < 0 thì tiến trình phải vào hàng đợi để chờ cho đến khi S > = 0. Down được cài đặt như sau:

```
Procedure  Down(s);
Begin
        S := S -1;
        If  S < 0 Then                          {S >= 0 thì tiếp tục}
          Begin
              Status(p)= blocked;       {chuyển tiến trình sang blocked}
              Enter(p, F(s));           {đưa tiến trình vào hàng đợi F(S)}
          end;
End;
```

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

```
Procedure  Up(s);
```

```
Begin
        S := S + 1;
        If  S <=  0  Then
            Begin
                Exit(Q,F(s) );                  {đưa tiến trình ra khỏi
F(S)}
                Status(Q) = ready ;       {chuyển tiến trình sang ready}
                Enter(Q, ready-list );      {đưa tiến trình vào ready list}
             End;
End;
```

Sau đây là sơ đồ điều độ dùng Semaphore (sự đánh tín hiệu bằng cờ) cho 2 tiến trình P1 và P2, hai tiến trình này hoạt động đồng thời cùng truy xuất đến tài nguyên găng tương ứng với semaphore (sự đánh tín hiệu bằng cờ) S. Tài nguyên găng này chỉ có thể đáp ứng cho một tiến trình tại một thời điểm nên S được khởi gán bằng 1.

```
Program MultualExclution;
        Const
n = 2;
        Var
s: semaphore (sự đánh tín hiệu bằng cờ);
        {-----------------------------------}
        Procedure P(i:Integer);
        Begin
                Repeat
                        Down(s);                {kiểm tra và xác lập quyền vào đoạn găng}
                        <Đoạn găng>;
                Up(s);                  {rời đoạn găng và kích hoạt tiến trình khác}
                        <Đoạn không găng>;
                Until .F.;
        End;
        {---------------------------------------}
BEGIN
        S := 1;
                ParBegin
                        P(1);
                        P(2);
        ParEnd;
        END.
        {---------------------------------------}
```

Ở đây chúng ta cần lưu ý rằng: Down và Up là các thủ tục của hệ điều hành, nên hệ điều hành đã cài đặt cơ chế độc quyền cho nó, tức là các lệnh bên trong nó không thể tách rời nhau. Nếu điều này không được thực hiện thì sơ đồ này trở nên vô nghĩa. Hai thủ tục Down(S) và Up(S) mà chúng tôi đưa ra ở trên chỉ để minh họa cho nguyên lý hoạt động của Down và Up.

Sử dụng semaphore (sự đánh tín hiệu bằng cờ) để điều độ tiến trình, mang lại những thuận lợi sau:

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

các tiến trình khi chúng ở trong hành đợi.

- Trị tuyệt đối của S cho biết số lượng các tiến trình đang đợi trên F(S).

Chú ý: Down và Up là các thủ tục của hệ điều hành nên sơ đồ điều độ sẽ bị thay đổi khi thay đổi hệ điều hành. Đây là một trở ngại của việc sử dụng semaphore (sự đánh tín hiệu bằng cờ) để tổ chức điều độ tiến trình.

Các ví dụ sau đây thay cho sự giải thích về sơ đồ điều độ ở trên:

Ví dụ 1: Sự thực hiện của hai tiến trình P1 và P2 trong sơ đồ điều độ trên

P thực hiện	**Down/Up**	**S**	**Trạng thái của P1/P2**
		1	
1. **P1**	Down(S)	0	P1 hoạt động
2. **P2**	Down(S)	-1	P2 chờ
3. **P1**	Up(S)	0	P2 hoạt động
4. **P1**	Down(S)	-1	P1 chờ
5. **P2**	Down(S)	0	P1 hoạt động

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

- Có 6 tiến trình yêu cầu sử dụng tài nguyên găng tương ứng với S lần lượt là: **A B C D E F**
- Độ ưu tiên của các tiến trình là (5 là độ ưu tiên cao nhất):

  1 1 2 4 2 5
- Thời gian các tiến trình cần sử dụng tài nguyên găng là:

  4 2 2 2 1 1

Nếu dùng sơ đồ điều độ semaphore (sự đánh tín hiệu bằng cờ) ở trên để tổ chức điều độ cho 6 tiến trình này thì ta có được bảng mô tả sự thực hiện của các tiến trình A, B, C, D, E, F như sau:

T	**Down/**	**Tiến trình**	**S**	**Tiến trình**	**Các tiến trình**

	Up	**thực hiện**		**hoạt động**	**trong hàng đợi**
0	-	-	**1**	-	-
1	Down	A	0	A	-
2	Down	B	-1	A	B
3	Down	C	-2	A	C B
4	Down	D	-3	A	D C B
5	Up	A	-2	D	C B
6	Down	E	-3	D	C E B
7	Up	D	-2	C	E B
8	Down	F	-3	C	F E B
9	Up	C	-2	F	E B
10	Up	F	-1	E	B
11	Up	E	0	B	-
12	Up	B	**1**	-	-

Bảng trên lưu ý với chúng ta hai điều. Thứ nhất, trị tuyệt đối của S cho biết số lượng các tiến trình trong hành đợi F(S). Thứ hai, tiến trình chưa được vào đoạn găng thì được đưa vào hàng đợi và tiến trình ra khỏi đoạn găng sẽ đánh thức tiến trình có độ ưu tiên cao nhất trong hành đợi để đưa nó vào đoạn găng. Tiến trình được đưa vào hàng đợi sau nhưng có độ ưu tiên cao hơn sẽ được đưa vào đoạn găng trước các tiến trình được đưa vào hàng đợi trước nó.

Giải pháp dùng Monitors

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

1. Các biến và cấu trúc dữ liệu bên trong monitor chỉ có thể được thao tác bởi các thủ tục được định nghĩa bên trong monitor đó.
2. Một tiến trình muốn vào monitor phải gọi một thủ tục của monitor đó.
3. Tại một thời điểm, chỉ có một tiến trình duy nhất được hoạt động bên trong monitor. Các tiến trình khác đã gọi monitor phải hoãn lại để chờ monitor rảnh.

Hai tính chất 1 và 2 tương tự như các tính chất của các đối tượng trong lập

trình hướng đối tượng. Như vậy một hệ điều hành hoặc một ngôn ngữ lập trình hướng đối tượng có thể cài đặt monitor như là một đối tượng có các tính chất đặc biệt.

Với tính chất thứ 3 monitor có khả năng thực hiện các cơ chế độc quyền, các biến trong monitor có thể được truy xuất chỉ bởi một tiến trình tại một thời điểm. Như vậy các cấu trúc dữ liệu dùng chung bởi các tiến trình có thể được bảo vệ bằng cách đặt chúng bên trong monitor. Nếu dữ liệu bên trong monitor là tài nguyên găng thì monitor cung cấp sự độc quyền trong việc truy xuất đến tài nguyên găng đó.

Monitor cung cấp các công cụ đồng bộ hoá để người lập trình sử dụng trong các sơ đồ điều độ. Công cụ đồng bộ hoá được định nghĩa để sử dụng trong các sơ đồ điều độ như sau: Trong một monitor, có thể định nghĩa các *biến điều kiện* và hai thao tác kèm theo là Wait và Signal, chỉ có wait và signal được tác động đến các biến điều kiện.

- Giả sử C là biến điều kiện được định nghĩa trong monitor.
- **Wait(c)**: khi một tiến trình gọi wait, thì wait sẽ chuyển tiến trình gọi sang trạng thái blocked, và đặt tiến trình này vào hàng đợi trên biến điều kiện c. Wait được cài đặt như sau:

```
Procedure   Wait(c);
Begin
        Status(p) = blocked;
        Enter(p,f(c));
End
```

- **Signal(c)**: khi một tiến trình gọi signal, thì signal sẽ kiểm tra trong hàng đợi của c có tiến trình nào hay không, nếu có thì tái kích hoạt tiến trình đó, và tiến trình gọi signal sẽ rời khỏi monitor. Signal được cài đặt như sau:

```
Procedure Signal(c);
  Begin
        If  f(c) <> Null  Then
          Begin
                Exit(Q,f(c));          {Q là tiến trình chờ trên C}
                Status(Q) = ready;
                Enter(Q,ready-lits);
          end;
  End
```

Trình biên dịch chịu trách nhiệm thực hiện việc truy xuất độc quyền đến dữ liệu trong monitor. Để thực hiện điều này, hệ điều hành dùng một semaphore (sự đánh tín hiệu bằng cờ) nhị phân. Mỗi monitor có một hàng đợi toàn cục lưu các tiến trình

đang chờ được vào monitor, ngoài ra mỗi biến điều kiện c cũng gắn với một hàng đợi F(c).

Với mỗi nhóm tài nguyên găng, có thể định nghĩa một monitor trong đó đặc tả tất cả các thao tác trên tài nguyên này với một số điều kiện nào đó. Sau đây là cấu trúc một Monitor. Sau đây là cấu trúc của monitor:

```
Monitor        <Tên monitor>
Condition      <Danh sách các biến điều kiện>;
{-------------------------------------------------}
    Procdure  Action1();              {thao tác i}
    Begin
        ......
    End;
    {-------------------------}
    Procedure   Actionn();            {thao tác n}
    Begin
        ......
    End;
{-------------------------}
End monitor;
```

Mỗi tiến trình muốn sử dụng tài nguyên găng chỉ có thể thao tác thông qua các thủ tục bên trong monitor.

Sau đây là sơ đồ điều độ sử dụng monitor cho 2 tiến trình P1 và P2.

```
Program MultualExclution;
Monitor …….  Endmonitor;                {monitor được định nghĩa như trên}
{----------------------------------}
BEGIN
    ParBegin
        P1: Repeat
              <Đoạn không găng của P1>;
        <monitor>.Action_i;          {Đoạn găng của P1};
        <Đoạn không găng của P1>;
            Until .F.
        P2: Repeat
              <Đoạn không găng của P2>;
        <monitor>.Action_j ;         {Đoạn găng của P2};
        <Đoạn không găng của P2>;
            Until .F.
    Parend
END.
{-------------------------------------------------------------}
```

Với monitor, việc tổ chức truy xuất độc quyền được trình biên dịch thực hiện, nên nó đơn giản hơn cho người lập trình. Tuy nhiên hiện nay rất ít ngôn ngữ lập trình hỗ trợ cấu trúc monitor cho lập trình.

Giải pháp trao đổi Message (thông điệp)

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

Các hệ thống truyền thông điệp có thể có nhiều dạng. Trong phần này chúng tôi giới thiệu một dạng chung nhất mà trong đó đề cập đến các đặc trưng có trong nhiều hệ thống khác nhau. Các hàm của truyền thông điệp trên thực tế có dạng tương tự như hai hàm sau:

- Send(destination, message): gởi thông điệp đến tiến trình đích.
- Receive(source, message): nhận thông điệp từ tiến trình nguồn.

Một tiến trình gởi thông tin dưới dạng một thông điệp (message) đến một tiến trình khác, bằng hàm Send, được nhận biết bởi tham số destination. Một tiến trình nhận thông điệp (message), bằng hàm Receive, từ một tiến trình được nhận biết bởi tham số source. Tiến trình gọi Receive phải chờ cho đến khi nhận được message từ tiến trình source thì mới có thể tiếp tục được.

Việc sử dụng Send và Receive để tổ chức điều độ được thực hiện như sau:

- Có một tiến trình kiểm soát việc sử dụng tài nguyên găng.
- Có nhiều tiến trình khác yêu cầu sử dụng tài nguyên găng này.
- Tiến trình có yêu cầu tài nguyên găng sẽ gởi một thông điệp đến tiến trình kiểm soát và sau đó chuyển sang trạng thái blocked cho đến khi nhận được một thông điệp chấp nhận cho truy xuất từ tiến trình kiểm soát tài nguyên găng.
- Khi sử dụng xong tài nguyên găng, tiến trình vừa sử dụng tài nguyên găng gởi một thông điệp khác đến tiến trình kiểm soát để báo kết thúc truy xuất.
- Tiến trình kiểm soát, khi nhận được thông điệp yêu cầu tài nguyên găng, nó sẽ chờ cho đến khi tài nguyên găng sẵn sàng để cấp phát thì gởi một thông điệp đến tiến trình đang bị khoá trên tài ngyên đó để đánh thức tiến trình này.

Trong sơ đồ điều độ dùng message tiến trình P được viết như sau:

```
Procedure  P(i: Integer);
Begin
        Repeat
                Send(process controler, request message);
                Receive(process controler, accept message );
                <Đoạn găng của P>;
                Send(process controler ,end message);
                <Đoạn không găng của P>;
        Until .F.
End;
{-------------------------------------------------------------------}
```

Giải pháp này thường được cài đặt trên các hệ thống mạng máy tính, đặc biệt là trên các hệ thống mạng phân tán. Đây là lợi thế mà semaphore (sự đánh tín hiệu bằng cờ) và monitor không có được.

Sơ đồ điều độ dùng message phải chú ý sự đồng bộ giữa các tiến trình nhận và giử message, nếu không các tiến trình này sẽ không thoát khỏi trạng thái blocked để tiếp tục được. Điều này cũng có nghĩa là công tác điều độ có thể không thành công mặc dù sơ đồ điều độ đã được tổ chức rất tốt. Sau đây chúng ta sẽ xem xét về sự đồng bộ giữ tiến trình send và tiến trình receiver trong trường hợp này.

Hai bài toán điều phối làm ví dụ

Bài toán 1: Giả sử có một ứng dụng, tạm gọi là ứng dụng Producer/Consumer, trong hệ thống đa nhiệm – đa người sử dụng. Ứng dụng này có hai tiến trình chính đó là, tiến trình người sản xuất (Producer) và tiến trình người tiêu thụ (Consumer), hai tiến trình này hoạt động đồng thời với nhau và cùng chia sẻ một bộ đệm (Buffer) có kích thước giới hạn, chỉ có 3 phần tử. Tiến trình Producer tạo ra dữ liệu và đặt vào Buffer, tiến trình Consumor nhận dữ liệu từ Buffer ra để xử lý. Rõ ràng hệ thống này cần phải có các ràng buộc sau:

1. Hai tiến trình Producer và Consumer không được đồng thời truy xuất Buffer (Buffer là tài nguyên găng).
2. Tiến trình Producer không được ghi dữ liệu vào Buffer khi Buffer đã bị đầy.
3. Tiến trình Consumer không được đọc dữ liệu từ Buffer khi Buffer rỗng.

Hãy dùng các giải pháp Semafore, Monitor, Message để tổ chức điều độ cho các tiến trình Producer và Consumer trong bài toán trên.

Giải pháp dùng Semaphore (sự đánh tín hiệu bằng cờ)

Với giải pháp này sơ đồ điều độ phải sử dụng 3 Semaphore (sự đánh tín hiệu bằng cờ):

- Full: dùng để theo dõi số chỗ đã có dữ liệu trong bộ đệm, nó được khởi gán bằng 0. Tức là, ban đầu Buffer rỗng.
- Empty: dùng để theo dõi số chỗ còn trống trên bộ đệm, nó được khởi gán bằng 3. Tức là, ban đầu Buffer không chứa một phần tử dữ liệu nào.
- Mutex: dùng để kiểm tra truy xuất đồng thời trên bộ đệm, nó được khởi gán bằng 1. Tức là, chỉ có 1 tiến trình được phép truy xuất buffre.

Sơ đồ điều độ sẽ như sau:

```
Program    Producer/Consumer;
Var     Full, Empty, Mutex: Semaphore (sự đánh tín hiệu bằng cờ);
{-------------------------------------------}
Procedure   Producer();
Begin
    Repeat
        < Tạo dữ liệu>;
        Down(empty);            {kiểm tra xem buffer còn chỗ trống ?}
        Down(mutex);            {kiểm tra và xác lập quyền truy xuất
Buffer}
        <Đặt dữ liệu vào Buffer>;
        Up(mutex);              {kết thúc truy xuất buffer}
        Up(Full);               {đã có 1 phần tử dữ liệu trong Buffer}
     Until .F.
End;
{--------------------------------}
Procedure   Consumer();
Begin
    Repeat
        Down(full);             {còn phần tử dữ liệu trong Buffer?}
        Down(mutex);            {kiểm tra và xác lập quyền truy xuất
Buffer}
        <Nhận dữ liệu từ đệm>;
        Up(mutex);              {kết thúc truy xuất buffer}
        Up(empty);              {đã lấy 1 phần tử dữ liệu trong Buffer}
     Until .F.
End;
{-------------------------------------------}
BEGIN
        Full = 0; Empty = 3; Mutex = 1;
        Produc er();
        Consumer();
```

END.

{---}

Giải pháp dùng Monitor

Với giải pháp này người lập trình phải định nghĩa một monitor, có tên là ProducerConsumer, trong đó có hai thủ tục Enter và Remove, dùng để thao tác trên Buffer. Xử lý của các thủ tục này phụ thuộc vào các biến điều kiện full và empty. Full và Emtry được quy định định sử dụng như trong giải pháp semaphore (sự đánh tín hiệu bằng cờ).

Sơ đồ điều độ sẽ như sau:

```
Program    Producer/Consumer;
Monitor ProducerConsumer;
Condition Full, Empty;
Var    Count: Integer;          {để đếm số phần tử dữ liệu được đưa vào Buffer}
       N: Interger;             {số phần tử của Buffer}
{ --------------------------------}
Procedure Enter();
Begin
       If  Count = N  Then  Wait(Full);        {nếu Buffer đầy thì đợi }
       <Đặt dữ liệu vào đệm>;                  {Buffer rỗng}
       Count := Count + 1;
       If  Count = 1  Then  Signal(Empty);     {nếu Buffer không rỗng thì}
End;                                           {báo cho consumer biết}
{-------------------------------------------------}
Procedure Remove();
Begin
       If  Count = 0  Then  Wait(Empty);       {nếu Buffer rỗng thì đợi đầy}
       <Nhận dữ liệu từ đệm>;
       Count := Count - 1;
       If  Count = N - 1  Then  Signal(Full);  {nếu Buffer không đâyf thì}
End;                                           {báo cho producer}
```

```
Endmonitor;
{-------------------------------------------------}
BEGIN
        Count = 0; N = 3;
ParBegin
Procedure Producer();
Begin
    Repeat
        <Tạo dữ liệu>;
        Producer/Consumer.Enter;
     Until .F.
End;
{--------------------------------------}
Procedure  Consumor();
Begin
    Repeat
        Producer/Consumer.Remove;
        <Xử lý dữ liệu>;
    Until .F.
End;
Parend
END.
{-------------------------------------}
```

Giải pháp dùng Message

Với giải pháp này chương trình dùng thông điệp empty. Empty hàm ý có một chỗ trống. Buffer. Khi khởi tạo tiến trình Consumer gởi ngay N thông điệp empty đến tiến trình Producer. Tiến trình Producer tạo ra một dữ liệu mới và chờ đến khi nhận được một thông điệp empty từ consumer thì gởi ngược lại cho Consumer một thông điệp có chứa dữ liệu mà nó tạo ra. Sau khi gởi đi thông điệp Emtry, tiến trình consumer sẽ chờ để nhận thông điệp chứa dữ liệu từ tiến trình producer. Sau khi xử lý xong dữ liệu thì consumer gởi lại một thông điệp empty đến tiến trình producer.

Sơ đồ điều độ sẽ như sau:

```
Program    Producer/Consumer;
Var
        Buffersize: integer;                    {kích thước Buffer}
        M, m': Message;
{ ------------------------------------}
BEGIN
        Buffersize = N;
ParBegin
Procedure  Producer();
Begin
  Repeat
        <Tạo dữ liệu>;
        Receive(Consumer,m);
        <Tạo thông điệp dữ liệu>
        Send(Consumer,m)
   Until .F.
End;
{ ---------------------------------------}
Procedure       Consumer ()
Var  I:integer;
Begin
        For I := 0 to N Do Send(Producer ,m);
        Repeat
                Receive(Producer ,m);
                <Lấy dữ liệu từ thông điệp>
                Send (Producer,m);
                <Xử lý dữ liệu >
        Until .F.
End.
Parend
END.
{-----------------------------------------------------}
```

Bài toán 2: Trong môi trường hệ điều hành đa nhiệm, có thể tồn tại các file chia sẻ, có thể là các file cơ sở dữ liệu. Nhiều tiến trình hoạt động đồng thời trong hệ

thống có thể được chia sẻ sử dụng một file cơ sở dữ liệu này. Tiến trình cần đọc nội dung của file cơ sở dữ liệu được gọi là là tiến trình Reader. Tiến trình cần cập nhật thông tin vào file cơ sở dữ liệu được gọi là là tiến trình Writer. Trong hệ thống này, công tác điều độ tiến trình cần phải thực hiện các ràng buộc sau:

1. Có thể có nhiều tiến trình Reader đồng thời đọc file cơ sở dũ liệu.
2. Không cho phép một tiến trình Writer ghi vào cơ sở dữ liệu khi các tiến trình Reader khác đang đọc cơ sở dữ liệu.
3. Chỉ có duy nhất một tiến trình Writer được phép ghi vào file cơ sở dữ liệu

Hãy dùng các giải pháp Semafore, Monitor, Message để tổ chức điều độ cho các tiến trình Reader và Writer trong bài toán ở trên.

Giải pháp dùng Semaphore (sự đánh tín hiệu bằng cờ)

Giải pháp này sử dụng một biến chung RC và hai semaphore (sự đánh tín hiệu bằng cờ) là Mutex và DB.

- RC (readcount) dùng để ghi nhận số lượng các tiến trình Reader muốn truy xuất file cơ sở dữ liệu, khởi gán bằng 0.
- Mutex: dùng để kiểm soát truy xuất đến RC, khởi gán bằng 1.
- DB: dùng để kiểm tra sự truy xuất độc quyền đến cơ sở dữ liệu, khởi gán bằng 1.

Sau đây là sơ đồ điều độ:

```
Program    Producer/Consumer;
Const
        Mutex: Seamafore = 1;
        Db     : Seamafore = 1;
        Rc     : byte = 0;
{-----------------------------------}
BEGIN
ParBegin
Procedure  Reader();
Begin
    Repeat
        Down(mutex);
        Rc = Rc+1;
```

```
        If Rc = 1 then Down(db);
        Up(mutex);                          {chấm dứt truy xuất Rc}
        <Đọc dữ liệu >;
        Down(mutex)
        Rc = Rc-1
        If Rc = 0 then Up(db);
        Up(mutex);
        < Xử lý dữ liệu đọc được>
      Until .F.
End;
{----------------------------------------}
Procedure  Writer();
Begin
    Repeat
        <Tạo dữ liệu >;
        Down(Db);
        <cập nhận dữ liệu >
        Up(db);
    Until .F.
End;
ParEnd
End.
{----------------------------------------}
```

Giải pháp dùng Monitor

Giải pháp này sử dụng một biến chung RC, để ghi nhận số lượng các tiến trình reader muốn truy xuất cơ sở dữ liệu. Tiến trình Writer phải chuyển sang trạng thái khoá nếu RC > 0. Khi ra khỏi đoạn găng tiến trình Reader cuối cùng sẽ đánh thức tiến trình Write đang bị khoá.

Sau đây là sơ dồ điều độ:

```
Program    Producer/Consumer;
Monitor   Readerwriter
Condition Okwrite,Okread
Var
```

```
        Rc: integer;
        Busy: boolean = False;
{-----------------------------------}
Procedure Beginread()
Begin
        If  (busy) then wait(okread);
        Rc = Rc+1;
        Signal(okread);
End;
Procedure Finishread()
Begin
        Rc = Rc - 1;
        If  Rc = 0 Then Wait(okwrite);
End;
Procedure Beginwrite();
Begin
        Rc = Rc - 1;
        If  (busy) or (Rc <>  0) Then Wait(okwrite);
        Busy = True;
End;
Procedure FinishWrite()
Begin
        Busy = False;
        If  (Okread)  Then Signal(okread)
        Else Signal(okwrite);
End;
Endmonitor.
{-----------------------------------------------------------}
BEGIN
ParBegin
Procedure  Reader ();
Begin
```

```
        Repeat
                ReaderWriter.BeginRead();
                <đọc dữ liệu>
                ReaderWriter.FinishRead();
        Until .F.
End;
Procedure   Writer ();
Begin
        Repeat
                ReaderWriter.BeginWrite();
                <đọc dữ liệu>
                ReaderWriter.FinishWrite();
        Until .F.
End;
Parend
END.
{-----------------------------------------------}
```

Giải pháp dùng Message

Giải pháp này cần phải có một tiến trình Sever điều khiển việc truy xuất cơ sở dữ liệu. Các tiến trình Writer và Reader gửi các thông điệp yêu cầu truy xuất đến server và nhận từ Sever các thông điệp hồi đáp tương ứng.

Sơ đồ điều độ sẽ như sau:

```
Program    Producer/Consumer;
Begin
ParBegin
Procedure   Reader();
Begin
     Repeat
        Send (Sever,Requesread);
        Receive(sever,value);
        Print(value);
    Until .F.
```

```
End;
Procedure  Writer();
Begin
   Repeat
        <Tạo dữ liệu>;
        Send (Sever, Requeswrite,value);
        Receive(sever, okwrite);
    Until .F.
End;
ParEnd
End.
{-----------------------------------------------}
```

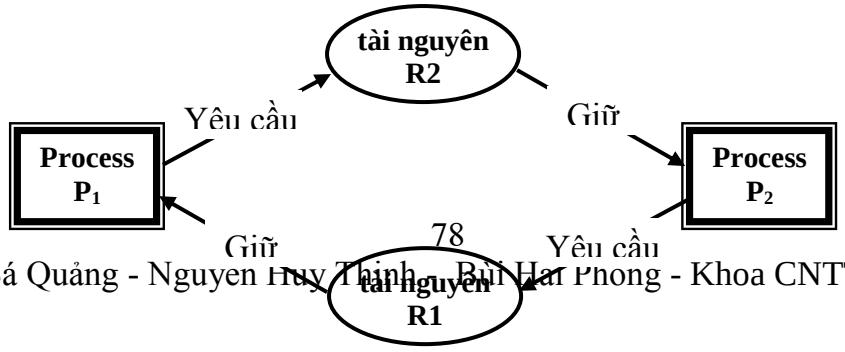
2.3. Hiện tượng bế tắc (Deadlock)

2.3.1. Khái niệm chung

Tắc nghẽn

Tất cả hiện tượng tắc nghẽn đều bắt nguồn từ sự xung đột về tài nguyên của hai hoặc nhiều tiến trình đang hoạt động đồng thời trên hệ thống. Tài nguyên ở đây có thể là một ổ đĩa, một record trong cơ sở dữ liệu, hay một không gian địa chỉ trên bộ nhớ chính. Sau đây là một số ví dụ để minh hoạ cho điều trên.

Ví dụ 1: Giả sử có hai tiến trình P_1 và P_2 hoạt động đồng thời trong hệ thống. Tiến trình P_1 đang giữ tài nguyên R1 và xin được cấp R2 để tiếp tục hoạt động, trong khi đó tiến trình P_2 đang giữ tài nguyên R2 và xin được cấp R1 để tiếp tục hoạt động. Trong trường hợp này cả P_1 và P_2 sẽ không tiếp tục hoạt động được. Như vậy P_1 và P_2 rơi vào trạng thái tắc nghẽn. Ví dụ này có thể được minh hoạ bởi sơ đồ ở hình 2.
Tắc nghẽn thường xảy ra do xung đột về tài nguyên thuộc loại không phân chia được, một số ít trường hợp xảy ra với tài nguyên phân chia được. Ví dụ sau đây là trường hợp tắc nghẽn do xung đột về tài nguyên bộ nhớ, là tài nguyên thuộc loại phân chia được.



Hình 2.6. Chờ đợi vòng tròn

Ví dụ 2: Giả sử không gian bộ nhớ còn trống là 200Kb, và trong hệ thống có hai tiến trình P_1 và P_2 hoạt động đồng thời. P_1 và P_2 yêu cầu được sử dụng bộ nhớ như sau:

P1	P2
….	….
Request1 80Kb	Request1 70Kb
…..	…..
Request2 30Kb	Request2 40Kb
…..	…..

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

Ví dụ 3: Trong các ứng dụng cơ sở dữ liệu, một chương trình có thể khoá một vài record mà nó sử dụng, để dành quyền điều khiển về cho nó. Nếu tiến trình P_1 khoá record R1, tiến trình P_2 khoá record R2, và rồi sau đó mỗi tiến trình lại cố gắng khoá record của một tiến trình khác. Tắc nghẽn sẽ xảy ra.

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

Trong trường hợp của ví dụ 1 ở trên: hai tiến trình P_1 và P_2 sẽ rơi vào trạng thái tắc nghẽn, nếu không có sự can thiệp của hệ điều hành. Để phá bỏ tắc nghẽn này hệ điều hành có thể cho tạm dừng tiến trình P_1 để thu hồi lại tài nguyên R1, lấy R1 cấp cho tiến trình P_2 để P_2 hoạt động và kết thúc, sau đó thu hồi cả R1 và R2 từ tiến trình P_2 để cấp cho P_1 và tái kích hoạt P_1 để P_1 hoạt động trở lại. Như vậy sau một khoảng thời gian cả P_1 và P_2 đều ra khỏi tình trạng tắc nghẽn.

Trong trường hợp của ví dụ 2 ở trên: nếu hai tiến trình này không đồng thời yêu cầu thêm bộ nhớ thì tắc nghẽn không thể xảy ra, hoặc khi cả hai tiến trình đồng thời yêu cầu thêm bộ nhớ thì hệ điều hành phải kiểm tra lượng bộ nhớ còn trống của hệ thống, nếu không đáp ứng cho cả hai tiến trình thì hệ điều hành phải có cơ chế ngăn chặn (từ chối) một tiến trình và chỉ cho một tiến trình được quyền sử dụng bộ nhớ (đáp ứng) thì tắc nghẽn cũng không thể xảy ra. Tuy nhiên để giải quyết vấn đề

tắc nghẽn do thiếu bộ nhớ, các hệ điều hành thường sử dụng cơ chế bộ nhớ ảo. Bộ nhớ ảo là một phần quan trong của hệ điều hành mà chúng ta sẽ khảo sát ở chương *Quản lý bộ nhớ* của tài liệu này.

Khi hệ thống xảy ra tắc nghẽn nếu hệ điều hành không kịp thời phá bỏ tắc nghẽn thì hệ thống có thể rơi vào tình trạng treo toàn bộ hệ thống. Như trong trường hợp tắc nghẽn ở ví dụ 1, nếu sau đó có tiến trình P_3, đang giữ tài nguyên R3, cần R2 để tiếp tục thì P_3 cũng sẽ rơi vào tập tiến trình bị tắc nghẽn, rồi sau đó nếu có tiến trình P_4 cần tài nguyên R1 và R3 để tiếp tục thì P_4 cũng rơi vào tập các tiến trình bị tắc nghẽn như P_3, … cứ thế dần dần có thể dẫn đến một thời điểm tất cả các tiến trình trong hệ thống đều rơi vào tập tiến trình tắc nghẽn. Và như vậy hệ thống sẽ bị treo hoàn toàn.

Điều kiện hình thành tắc nghẽn

Năm 1971, Coffman đã đưa ra và chứng tỏ được rằng, nếu hệ thống tồn tại đồng thời bốn điều kiện sau đây thì hệ thống sẽ xảy ra tắc nghẽn:

1. Loại trừ lẫn nhau (mutual excution) hay độc quyền sử dụng: Đối với các tài nguyên không phân chia được thì tại mỗi thời điểm chỉ có một tiến trình sử dụng được tài nguyên.
2. Giữ và đợi (hold and wait): Một tiến trình hiện tại đang chiếm giữ tài nguyên, lại xin cấp phát thêm tài nguyên mới.
3. Không ưu tiên (No preemption): Không có tài nguyên nào có thể được giải phóng từ một tiến trình đang chiếm giữ nó.

Trong nhiều trường hợp các điều kiện trên là rất cần thiết đối với hệ thống. Sự thực hiện độc quyền là cần thiết để bảo đảm tính đúng đắn của kết quả và tính toàn vẹn của dữ liệu (chúng ta đã thấy điều này ở phần tài nguyên găng trên đây). Tương tự, sự ưu tiên không thể thực hiện một cách tuỳ tiện, đặt biệt đối với các tài nguyên có liên quan với nhau, việc giải phóng từ một tiến trình này có thể ảnh hưởng đên kết quả xử lý của các tiến trình khác.

Sự tắc nghẽn có thể tồn tại với ba điều kiện trên, nhưng cũng có thể không xảy ra chỉ với 3 điều kiện đó. Để chắc chắn tắc nghẽn xảy ra cần phải có điều kiện thư tư

4. Đợi vòng tròn (Circular wait): Đây là trường hợp của ví dụ 1 mà chúng ta đã nêu ở trên. Tức là, mỗi tiến trình đang chiếm giữ tài nguyên mà tiến trình khác đang cần.

Ba điều kiện đầu là điều kiện cần chứ không phải là điều kiện đủ để xảy ra tắc nghẽn. Điều kiện thứ tư là kết quả tất yếu từ ba điều kiện đầu.

2.3.2. Ngăn chặn tắc nghẽn (Deadlock Prevention)

Ngăn chặn tắc nghẽn là thiết kế một hệ thống sao cho hiện tượng tắc nghẽn bị loại trừ. Các phương thức ngăn chặn tắc nghẽn đều tập trung giải quyết bốn điều kiện gây ra tắc nghẽn, sao cho hệ thống không thể xảy ra đồng thời bốn điều kiện tắc nghẽn:

- Đối với điều kiện độc quyền: Điều kiện này gần như không tránh khỏi, vì sự độc quyền là cần thiết đối với tài nguyên thuộc loại phân chia được như các biến chung, các tập tin chia sẻ, hệ điều hành cần phải hỗ trợ sự độc quyền trên các tài nguyên này. Tuy nhiên, với những tài nguyên thuộc loại không phân chia được hệ điều hành có thể sử dụng kỹ thuật SPOOL (Smulataneous Peripheral Operation Online) để tạo ra nhiều tài nguyên ảo cung cấp cho các tiến trình đồng thời.
- Đối với điều kiện giữ và đợi: Điều kiện này có thể ngăn chặn bằng cách yêu cầu tiến trình yêu cầu tất cả tài nguyên mà nó cần tại một thời điểm và tiến trình sẽ bị khoá (blocked) cho đến khi yêu cầu tài nguyên của nó được hệ điều hành đáp ứng. Phương pháp này không hiệu quả. Thứ nhất, tiến trình phải đợi trong một khoảng thời gian dài để có đủ tài nguyên mới có thẻ chuyển sang hoạt động được, trong khi tiến trình chỉ cần một số ít tài nguyên trong số đó là có thể hoạt động được, sau đó yêu cầu tiếp. Thứ hai, lãng phí tài nguyên, vì có thể tiến trình giữa nhiều tài nguyên mà chỉ đến khi sắp kết thúc tiến trình mới sử dụng, và có thể đây là những tài nguyên mà các tiến trình khác đang rất cần. Ở đây hệ điều hành có thể tổ chức phân lớp tài nguyên hệ thống. Theo đó tiến trình phải trả tài nguyên ở mức thấp mới được cấp phát tài nguyên ở cấp cao hơn.
- Đối với điều kiện No preemption: Điều kiện này có thể ngăn chặn bằng cách, khi tiến trình bị rơi vào trạng thái khoá, hệ điều hành có thể thu hồi tài nguyên của tiến trình bị khoá để cấp phát cho tiến trình khác và cấp lại đầy đủ tài nguyên cho tiến trình khi tiến trình được đưa ra khỏi trạng thái khoá.
- Đối với điều kiện chờ đợi vòng tròn: Điều kiện này có thể ngăn chặn bằng cách phân lớp tài nguyên của hệ thống. Theo đó, nếu một tiến trình được cấp phát tài nguyên ở lớp L, thì sau đó nó chỉ có thể yêu cầu các tài nguyên ở lớp thấp hơn lớp L.

Nhận biết tắc nghẽn (Deadlock Detection)

Các phương thức ngăn chặn tắc nghẽn ở trên đều tập trung vào việc hạn chế quyền truy xuất đến tài nguyên và áp đặt các ràng buộc lên các tiến trình. Điều này có thể ảnh hưởng đến mục tiêu khai thác hiệu quả tài nguyên của hệ điều hành, ngăn chặn độc quyền trên tài nguyên là một ví dụ, hệ điều hành phải cài đặt các cơ chế độc quyền để bảo vệc các tài nguyên chia sẻ. Và như đã phân tích ở trên việc cấp phát tài

nguyên một lần cho các tiến trình để ngăn chặn hiện tượng hold and wait cũng tồn tại một vài hạn chế.

Các hệ điều hành có thể giải quyết vấn đề tắc nghẽn theo hướng phát hiện tắc nghẽn để tìm cách thoát khỏi tắc nghẽn. Phát hiện tắc nghẽn không giới hạn truy xuất tài nguyên và không áp đặt các ràng buộc lên tiến trình. Với phương thức phát hiện tắc nghẽn, các yêu cầu cấp phát tài nguyên được đáp ứng ngay nếu có thể. Để phát hiện tắc nghẽn hệ điều hành thường cài đặt một thuật toán để phát hiện hệ thống có tồn tại hiện tượng chờ đợi vòng tròn hay không.

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

Mỗi khi tắc nghẽn được phát hiện, hệ điều hành thực hiện một vài giải pháp để thoát khỏi tắc nghẽn. Sau đây là một vài giải pháp có thể:

1. Thoát tất cả các tiến trình bị tắc nghẽn. Đây là một giải pháp đơn giản nhất, thường được các hệ điều hành sử dụng nhất.

2. Sao lưu lại mỗi tiến trình bị tắc nghẽn tại một vài điểm kiển tra được định nghĩa trước, sau đó khởi động lại tất cả các tiến trình. Giải pháp này yêu cầu hệ điều hành phải lưu lại các thông tin cần thiết tại điểm dừng của tiến trình, đặc biệt là con trỏ lệnh và các tài nguyên tiến trình đang sử dụng, để có thể khởi động lại tiến trình được. Giải pháp này có nguy cơ xuất hiện tắc nghẽn trở lại là rất cao, vì khi tất cả các tiến trình đều được reset trở lại thì việc tranh chấp tài nguyên là khó tránh khỏi. Ngoài ra hệ điều hành thường phải chi phí rất cao cho việc tạm dừng và tái kích hoạt tiến trình.

3. Chỉ kết thúc một tiến trình trong tập tiến trình bị tắc nghẽn, thu hồi tài nguyên của tiến trình này, để cấp phát cho một tiến trình nào đó trong tập tiến trình tắc nghẽn để giúp tiến trình này ra khỏi tắc nghẽn, rồi gọi lại thuật toán kiểm tra tắc nghẽn để xem hệ thống đã ra khỏi tắc nghẽn hay chưa, nếu rồi thì dừng, nếu chưa thì tiếp tục giải phóng thêm tiến trình khác. Và lần lượt như thế cho đến khi tất cả các tiến trình trong tập tiến trình tắc nghẽn đều ra khỏi tình trạng tắc nghẽn. Trong giả pháp này vấn đề đặt ra đối với hệ điều hành là nên chọn tiến trình nào để giải phóng đầu tiên và dựa vào tiêu chuẩn nào để chọn lựa sao cho chi phí để giải phóng tắc nghẽn là thấp nhất.

4. Tập trung toàn bộ quyền ưu tiên sử dụng tài nguyên cho một tiến trình, để tiến trình này ra khỏi tắc nghẽn, và rồi kiểm tra xem hệ thống đã ra khỏi tắc nghẽn hay chưa, nếu rồi thì dừng lại, nếu chưa thì tiếp tục. Lần lượt như thế cho đến khi hệ thống ra khỏi tắc nghẽn. Trong giải pháp này hệ điều hành phải tính đến chuyện tái kích hoạt lại tiến trình sau khi hệ thống ra khỏi tắc nghẽn.

Đối với các giải pháp 3 và 4, hệ điều hành dựa vào các tiêu chuẩn sau đây để chọn lựa tiến trình giải phóng hay ưu tiên tài nguyên: Thời gian xử lý ít nhất; Thời gian cần processor còn lại ít nhất; Tài nguyên cần cấp phát là ít nhất; Quyền ưu tiên là thấp nhất.

Điều phối tiến trình

Trong môi trường hệ điều hành đa nhiệm, bộ phận điều phối tiến trình có nhiệm vụ xem xét và quyết định khi nào thì dừng tiến trình hiện tại để thu hồi processor và chuyển processor cho tiến trình khác, và khi đã có được processor thì chọn tiến trình nào trong số các tiến trình ở trạng thái ready để cấp processor cho nó. Ở đây chúng ta cần phân biệt sự khác nhau giữa điều độ tiến trình và điều phối tiến trình.

Mục tiêu điều phối tiến trình

Các cơ chế điều phối tiến trình: Trong công tác điều phối tiến trình bộ điều phối sử dụng hai cơ chế điều phối: Điều phối độc quyền và điều phối không độc quyền.

- Điều phối độc quyền: Khi có được processor tiến trình toàn quyền sử dụng processor cho đến khi tiến trình kết thúc xử lý hoặc tiến trình tự động trả lại processor cho hệ thống. Các quyết định điều phối xảy ra khi: Tiến trình chuyển trạng thái từ Running sang Blocked hoặc khi tiến trình kết thúc.
- Điều phối không độc quyền: Bộ phận điều phối tiến trình có thể tạm dừng tiến trình đang xử lý để thu hồi processor của nó, để cấp cho tiến trình khác, sao cho phù hợp với công tác điều phối hiện tại. Các quyết định điều phối xảy ra khi: Tiến trình chuyển trạng thái hoặc khi tiến trình kết thúc.

Các đặc điểm của tiến trình: Khi tổ chức điều phối tiến trình, bộ phần điều phối tiến trình của hệ điều hành thường dựa vào các đặc điểm của tiến trình. Sau đây là một số đặc điểm của tiến trình:

- Tiến trình thiên hướng Vào/Ra: Là các tiến trình cần nhiều thời gian hơn cho việc thực hiện các thao tác xuất/nhập dữ liệu, so với thời gian mà tiến trình cần để thực hiện các chỉ thị trong nó, được gọi là các tiến trình thiên hướng Vào/Ra.
- Tiến trình thiên hướng xử lý: Ngược lại với trên, đây là các tiến trình cần nhiều thời gian hơn cho việc thực hiện các chỉ thị trong nó, so với thời gian mà tiến trình để thực hiện các thao tác Vào/Ra.

- Tiến trình tương tác hay xử lý theo lô: Tiến trình cần phải trả lại kết quả tức thời (như trong hệ điều hành tương tác) hay kết thúc xử lý mới trả về kết quả (như trong hệ điều hành xử lý theo lô).

- Độ ưu tiên của tiến trình: Mỗi tiến trình được gán một độ ưu tiên nhất định, độ ưu tiên của tiến trình có thể được phát sinh tự động bởi hệ thống hoặc được gán tường minh trong chương trình của người sử dụng. Độ ưu tiên của tiến trình có hai loại: Thứ nhất, độ ưu tiên tĩnh: là độ ưu tiên gán trước cho tiến trình và không thay đổi trong suốt thời gian sống của tiến trình. Thứ hai, độ ưu tiên động: là độ ưu tiên được gán cho tiến trình trong quá trình hoạt động của nó, hệ điều hành sẽ gán lại độ ưu tiên cho tiến trình khi môi trường xử lý của tiến trình bị thay đổi. Khi môi trường xử lý của tiến trình bị thay đổi hệ điều hành phải thay đổi độ ưu tiên của tiến trình cho phù hợp với tình trạng hiện tại của hệ thống và công tác điều phối tiến trình của hệ điều hành.

- Thời gian sử dụng processor của tiến trình: Tiến trình cần bao nhiêu khoảng thời gian của processor để hoàn thành xử lý.

- Thời gian còn lại tiến trình cần processor: Tiến trình còn cần bao nhiêu khoảng thời gian của processor nữa để hoàn thành xử lý.

Bộ phận điều phối tiến trình thường dựa vào đặc điểm của tiến trình để thực hiện điều phối ở mức tác tụ, hay điều phối tác vụ. Điều phối tác vụ được phải thực hiện trước điều phối tiến trình. Ở mức này hệ điều hành thực hiện việc chọn tác vụ để đưa vào hệ thống. Khi có một tiến trình được tạo lập hoặc khi có một tiến trình kết thúc xử lý thì bộ phận điều phối tác vụ được kích hoạt. Điều phối tác vụ quyết định sự đa chương của hệ thống và hiệu quả cũng như mục tiêu của điều phối của bộ phận điều phối tiến trình. Ví dụ, để khi thác tối đa thời gian xử lý của processor thì bộ phận điều phối tác vụ phải đưa vào hệ thống số lượng các tiến trình tính hướng Vào/Ra cân đối với số lượng các tiến trình tính hướng xử lý, các tiến trình này thuộc những tác vụ nào. Nếu trong hệ thống có quá nhiều tiến trình tính hướng Vào/Ra thì sẽ lãng phí thời gian xử lý của processor. Nếu trong hệ thống có quá nhiều tiến trình tính hướng xử lý thì processor không thể đáp ứng và có thể các tiến trình phải đợi lâu trong hệ thống, dẫn đến hiệu quả tương tác sẽ thấp.

Mục tiêu điều phối: bộ phận điều phối tiến trình của hệ điều hành phải đạt được các mục tiêu sau đây trong công tác điều phối của nó.

- Sự công bằng (Fairness): Các tiến trình đều công bằng với nhau trong việc chia sẻ thời gian xử lý của processor, không có tiến trình nào phải chờ đợi vô hạn để được cấp processor.

- Tính hiệu quả (Efficiency): Tận dụng được 100% thời gian xử lý của

processor. Trong công tác điều phối, khi processor rỗi bộ phận điều phối sẽ chuyển ngay nó cho tiến trình khác, nếu trong hệ thống có tiến trình đang ở trạng thái chờ processor, nên mục tiêu này dễ đạt được. Tuy nhiên, nếu hệ điều hành đưa vào hệ thống quá nhiều tiến trình thiên hướng vào/ra, thì nguy cơ processor bị rỗi là có thể. Do đó, để đạt được mục tiêu này hệ điều hành phải tính toán và quyết định nên đưa vào hệ thống bao nhiêu tiến trình thiên hướng vào/ra, bao nhiêu tiến trình thiên hướng xử lý, là thích hợp.

- Thời gian đáp ứng hợp lý (Response time): Đối với các tiến trình tương tác, đây là khoảng thời gian từ khi tiến trình đưa ra yêu cầu cho đến khi nhận được sự hồi đáp. Một tiến trình đáp ứng yêu cầu của người sử dụng, phải nhận được thông tin hồi đáp từ yêu cầu của nó thì nó mới có thể trả lời người sử dụng. Do đó, theo người sử dụng thì bộ phận điều phối phải cực tiểu hoá thời gian hồi đáp của các tiến trình, có như vậy thì tính tương tác của tiến trình mới tăng lên.

- Thời gian lưu lại trong hệ thống (Turnaround time): Đây là khoảng thời gian từ khi tiến trình được đưa ra đến khi được hoàn thành. Bao gồm thời gian thực hiện thực tế cộng với thời gian đợi tài nguyên (bao gồm cả đợi processor). Đại lượng này dùng trong các hệ điều hành xử lý theo lô. Do đó, bộ phận điều phối phải cực tiểu thời gian hoàn thành (lưu lại trong hệ thống) của các tác vụ xử lý theo lô.

- Thông lượng tối đa (Throughtput): Chính sách điều phối phải cố gắng để cực đại được số lượng tiến trình hoàn thành trên một đơn vị thời gian. Mục tiêu này ít phụ thuộc vào chính sách điều phối mà phụ thuộc nhiều vào thời gian thực hiện trung bình của các tiến trình.

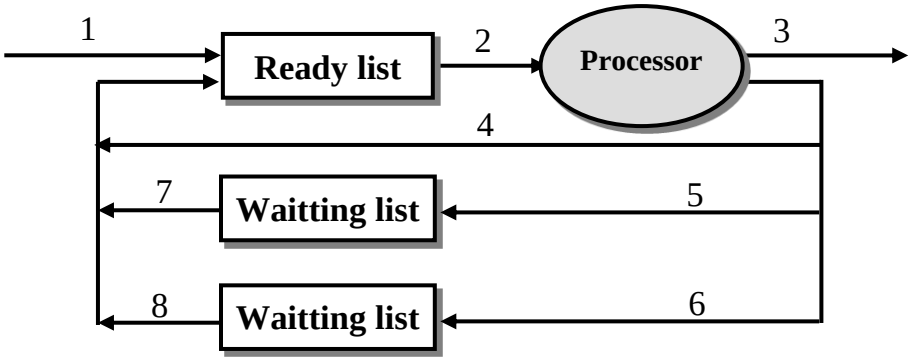
Công tác điều phối của hệ điều hành khó có thể thỏa mãn đồng thời tất cả các mục tiêu trên vì bản thân các mục tiêu này đã có sự mâu thuẫn với nhau. Các hệ điều hành chỉ có thể dung hòa các mục tiêu này ở một mức độ nào đó.

Ví dụ: Giả sử trong hệ thống có bốn tiến trình P_1, P_2, P_3, P_4, thời gian (t) mà các tiến trình này cần processor để xử lý lần lượt là 1, 12, 2, 1. Nếu ban đầu có 2 tiến trình P_1 và P_2 ở trạng thái ready thì chắc chắn bộ phận điều phối sẽ cấp processor cho P_1. Sau khi P_1 kết thúc thì processor sẽ được cấp cho P_2 để P_2 hoạt động (running), khi P_2 thực hiện được 2t thì P_3 được đưa vào trạng thái ready. Nếu để P_2 tiếp tục thì P_3 phải chờ lâu (chờ 8t), như vậy sẽ vi phạm mục tiêu thời gian hồi đáp và thông lượng tối đa (đối với P_3). Nếu cho P_2 dừng để cấp processor cho P_3 hoạt động đến khi kết thúc, khi đó thì P_4 vào trạng thái ready, bộ điều phối sẽ cấp processor cho P_4, và cứ như thế, thì P_2 phải chờ lâu, như vậy sẽ đạt được mục tiêu: thời gian hồi đáp và thông lượng tối đa nhưng vi phạm mục tiêu: công bằng và thời gian lưu lại trong hệ thống (đối với P_2).

Tổ chức điều phối tiến trình

Để tổ chức điều phối tiến trình hệ điều hành sử dụng hai danh sách: Danh sách sẵn sàng (Ready list) dùng để chứa các tiến trình ở trạng thái sẵn sàng. Danh sách đợi (Waiting list) dùng để chứa các tiến trình đang đợi để được bổ sung vào danh sách sẵn sàng.

Chỉ có những tiến trình trong ready list mới được chọn để cấp processor. Các tiến trình bị chuyển về trạng thái blocked sẽ được bổ sung vào waiting list. Hệ thống chỉ có duy nhất một ready list, nhưng có thể tồn tại nhiều waiting list. Thông thường hệ điều hành thiết kế nhiều waitting list, mỗi waitting list dùng để chứa các tiến trình đang đợi được cấp phát một tài nguyên hay một sự kiện riêng biệt nào đó. Hình sau đây minh hoạ cho việc chuyển tiến trình giữa các danh sách:



Hình 2.7. Sơ đồ chuyển tiến trình vào các danh sách

Trong đó:

1. Tiến trình trong hệ thống được cấp đầy đủ tài nguyên chỉ thiếu processor.
2. Tiến trình được bộ điều phối chọn ra để cấp processor để bắt đầu xử lý.
3. Tiến trình kết thúc xử lý và trả lại processor cho hệ điều hành.
4. Tiến trình hết thời gian được quyền sử dụng processor (time-out), bị bộ điều phối tiến trình thu hồi lại processor.
5. Tiến trình bị khóa (blocked) do yêu cầu tài nguyên nhưng chưa được hệ điều hành cấp phát. Khi đó tiến trình được đưa vào danh sách các tiến trình đợi tài nguyên (waiting list 1).
6. Tiến trình bị khóa (blocked) do đang đợi một sự kiện nào đó xảy ra. Khi đó tiến trình được bộ điều phối đưa vào danh sách các tiến trình đợi tài nguyên (waiting list 2).
7. Tài nguyên mà tiến trình yêu cầu đã được hệ điều hành cấp phát. Khi đó tiến trình được bộ điều phối chuyển sang danh sách các tiến trình ở trạng thái sẵn sang (ready list) để chờ được cấp processor để được hoạt động.

8. Sự kiện mà tiến trình chờ đã xảy ra. Khi đó tiến trình được bộ điều phối chuyển sang danh sách các tiến trình ở trạng thái sẵn sang (ready list) để chờ được cấp processor.

Chương 3: LẬP LỊCH CHO CPU

Chương này cung cấp cho người học kiến thức về nguyên lý điều phối các quá trình được thực hiện trên CPU, tối ưu hóa sử dụng tài nguyên CPU, các giải pháp lập lịch mà hệ điều hành thực hiện nhằm điều phối các quá trình được thực hiện trên CPU.

3.1. Các khái niệm cơ bản

Việc điều phối CPU xảy ra chỉ trong chế độ đa chương trình. Trong một số hệ thống máy tính, người ta còn phân biệt trạng thái của CPU: trạng thái bài toán (trạng thái người dùng) hay trạng thái SUPERVISOR (trạng thái kiểm tác) mà điều cốt lõi là trong trạng thái bài toán, không cho phép thực hiện một số lệnh đặc biệt của CPU. Việc phân biệt trạng thái của CPU cho phép phân loại các quá trình theo mức độ thâm nhập hệ thống và như vậy vấn đề an toàn và bảo vệ hệ thống được thuận lợi hơn. Chương trình người dùng chỉ thâm nhập sâu hệ thống chỉ thông qua các chương trình của hệ điều hành.

Chế độ đa chương trình, người sử dụng quan niệm rằng các chương trình " đang được thực hiện" song thực tế CPU của máy tại mỗi thời điểm chỉ phục vụ một chương trình và như vậy ta chỉ có 1 bộ xử lý thực (cho chương trình đã nói); các chương trình đồng thời còn lại "hiện đang" sử dụng CPU "ảo". Tốc độ làm việc của CPU ảo là "nhỏ hơn" tốc độ làm việc của CPU thực sự.

Nếu quan niệm như trên, mỗi quá trình được coi là chiếm giữ CPU suốt trong quá trình thực hiện của mình, do vậy, cần có sự phân biệt khi nào chiếm giữ CPU thực sự và khi nào chiếm giữ CPU ảo.

3.2. Các thuật toán lập lịch

Mục tiêu : Nắm được nguyên lý điều phối các quá trình được thực hiện trên CPU, tối ưu hóa sử dụng tài nguyên CPU.

3.2.1. Nguyên tắc chung

Điều phối chọn trong quá trình đang có mặt trong hàng đợi, ở trạng thái sẵn sàng và có độ ưu tiên cao nhất. Tồn tại rất nhiều quan điểm liên quan đến việc xác định độ ưu tiên, chẳng hạn: thời điểm tạo ra quá trình, thời điểm xuất hiện công việc, thời gian phục vụ, thời gian đã dành cho phục vụ, thời gian trung bình quá trình chưa được phục vụ v.v… Các yếu tố này được tính toán, đánh giá theo các phương pháp khác nhau và do đó tồn tại nhiều nguyên tắc điều phối khác nhau.

Tiêu chuẩn chọn một cách thức điều phối CPU là: cần chú ý tới việc nó ảnh hưởng như thế nào tới *thời gian chờ đợi xử lý*, tức là thời gian chi phí của quá trình

đó trong trạng thái chuẩn bị tới trạng thái sử dụng. Đối với người dùng, các kiểu chờ đợi sau đây của quá trình trong hệ thống là không phân biệt:

-Thời gian trong trạng thái chuẩn bị

-Thời gian trong trạng thái kết khối

-Thời gian quá trình trong đầu vào chờ đợi tài nguyên.

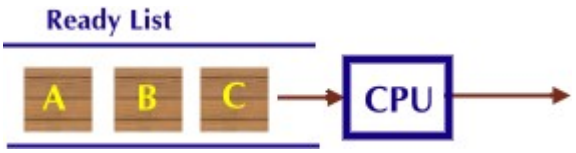
Như đã nói, có nhiều nguyên lý để điều phối; ở đây chỉ xem xét những nguyên lý chung và phổ biến nhất cũng như khảo sát những chiến lược để cài đặt các nguyên lý trên.

3.2.2. Các thuật toán lập lịch

Mục tiêu: Nắm được các giải pháp lập lịch mà hệ điều hành thực hiện nhằm điều phối các quá trình được thực hiện trên CPU.

a. First Come First Served (FCFS)

Tiến trình nào có yêu cầu sử dụng CPU trước thì sẽ được thực hiện trước.
Ưu điểm là thuật thoán đơn giản nhất
Nhược điểm la hiệu quả thuật toán phụ thuộc vào thứ tự của các tiến trình trong hàng đợi.



Hình 3.1. Hàng đợi FCFS

Giả sử có 3 tiến trình P1 , P2 , P3 với thời gian thực hiện tương ứng là 24ms, 3ms, 6ms
Giả sử ba tiến trình xếp hàng theo thứ tự P1, P2, P3
Thời gian chờ các tiến trình là:
P1chờ 0ms, P2 chờ 24ms, P3 chờ 27ms
Thời gian chờ trung bình: (0+24+27)/3=17ms

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

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

b. Shortest Job First (SJF)

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

Ví dụ :

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

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

P1	P4	P3	P2
0	6	8	12 20

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

P1	P4	P1	P3	P2
0	3	5	8	12 20

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

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

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

c. Shortest Remain Time (SRT)

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

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

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

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

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

P1	P2	P3
0	‘24	27 30

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

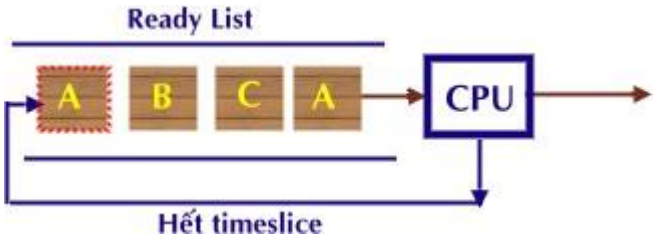
P1	P2	P3	P1
0	‘1	4	7 30

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

d. Round Robin (RR)

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

Ví dụ :



Hình 3.2. Điều phối Round Robin

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

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

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

Thời gian chờ đợi trung bình sẽ là (0+6+3+5)/3 = 4.66 milisecondes.

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

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

e. Multi Level Queue (MLQ)

Để phân lớp các quá trình đang trong trạng thái chuẩn bị và chọn lựa quá trình chuyển sang trạng thái sử dụng có thể sử dụng các thông tin được cho bằng người tạo ra quá trình đó và các thông tin nhận được trong việc điều phối các quá trình. Các thông tin này có thể là:

- Thông tin có sẵn, đã cho trước;
- Thời gian sử dụng thực tế;
- Số nhu cầu vào-ra đã tiến hành…

Với hệ thống tổ chức trang bộ nhớ, tiện lợi nhất là sử dụng một số dòng xếp hàng khác nhau để phân biệt các quá trình ở trạng thái đặt/tách trang với các quá trình chờ đợi sự kết thúc vào/ra.

- Đầu tiên, CPU có quá trình của dòng đợi có độ ưu tiên cao nhất. Quá trình trong mỗi hàng đợi có một lượng tử thời gian: nếu trong thời đoạn của lượng tử thời gian đó nó không hoàn thiện thì nó được xếp vào cuối cùng trong hàng đợi với độ ưu tiên ngay sát nó (ngay cả khi nó đòi hỏi một thời gian nào đó trong trạng thái kết khối). Chỉ có quá trình rơi vào dòng đợi với độ ưu tiên thấp nhất là hoạt động theo chế độ vòng còn các hàng đợi khác hoạt động theo kiểu FCFS.
- Ý nghĩa lôgic của điều phối kiểu này là ở chỗ quá trình đòi hỏi thời gian lâu hơn sẽ kết thúc muộn hơn theo xác xuất. Sự điều phối đa mức đã xem xét với sự liên kết ngược sẽ hiệu quả trong điều kiện tốc độ hoàn thiện của quá trình giảm đi theo lượng thời gian nó đã được phục vụ.

f. Multi Level Feedback Queues (MLFQ)

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

tiến trình thuộc về danh sách ở cấp ưu tiên *i* sẽ chỉ được cấp phát CPU khi các danh sách ở cấp ưu tiên lớn hơn *i* đã trống.

Hình 3.3. Chế độ điều phối nhiều mức độ ưu tiên

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

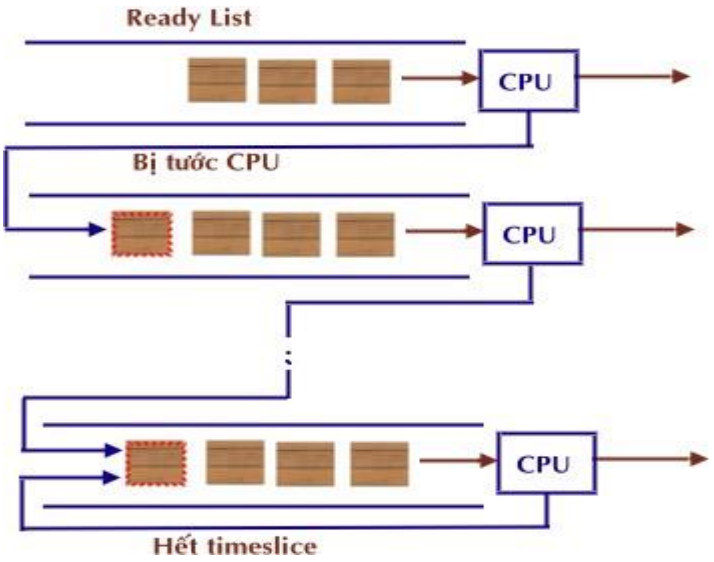
Số lượng các cấp ưu tiên

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

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

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

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



Hình 3.4. Điều phối Multilevel Feedback

3.3. Ngắt

Mục tiêu : Nắm được khái niệm ngắt, phân loại ngắt và quy trình xử lý ngắt.

3.3.1. Khái niệm ngắt

Tồn tại mối quan hệ giữa các bộ phận trong hệ điều hành, ví dụ: *điều phối,* thực hiện quá trình và hệ thống con vào – ra. Thông thường khi hết hạn lượng tử thời gian hay hoàn thiện vào/ra nảy sinh ngắt. Ngắt sinh ra những sự kiện khác và xử lý ngắt là những phương tiện quan trọng của điều khiển CPU. Xem xét chương trình thực hiện các lệnh một cách tuần tự, trong đó có lệnh chuyển điều khiển vô điều kiện và có điều kiện. Ngắt có thể được xác định như là một chương trình gắn vào truyền điều khiển cho một chương trình khác thực hiện tại thời điểm ngắt. Ngắt được coi như cách thức truyền điều khiển cho quá trình xử lý ngắt chưa được biết từ quá trình bị ngắt.

Ngắt được phân chia ra hai lớp cơ bản: ngắt trong và ngắt ngoài.

- Ngắt trong liên quan đến các sự kiện liên kết tới công việc của CPU và để đồng bộ hoạt động của nó. Ví dụ: tràn ô khi cộng hay trừ dấu phẩy động, xuất hiện phép chia cho 0; thực hiện phép toán dấu phẩy động truyền hoặc xóa phần bậc; vi phạm địa chỉ bộ nhớ, thiếu vắng segment hoặc trang, mã lệnh sai…

- Ngắt ngoài: được xảy ra theo các hiện tượng liên quan ngoài thực hiện của CPU: ngắt vào-ra, ngắt do sơ đồ kiểm tra, ngắt từ CPU khác, ngắt do hết lượng tử thời gian.v.v…

3.3.2. Xử lý ngắt

Như vậy, ngắt là một hiện tượng xảy ra có thể độc lập với sự làm việc của CPU. Một vấn đề được đặt ra là thời điểm xử lý ngắt: xử lý ngắt lúc nào là thích hợp nhất khi quan hệ với lệnh máy đang thực hiện. Ngắt xảy ra có thể hoặc do sự thực hiện lệnh, hoặc do tác động từ chính bản thân lệnh. Nếu cơ chế xử lý ngắt không thích hợp sẽ loại bỏ chính lệnh máy đang thực hiện. Thuận lợi hơn cả là xử lý ngắt sau khi thực hiện lệnh và việc ghi nhận ngắt là độc lập với sự thực hiện lệnh. Cơ chế ghi nhận ngắt là nằm ngoài các chương trình xử lý ngắt.

Có rất nhiều phương pháp liên quan đến xử lý ngắt nhưng quy trình chung có thể được mô tả qua các bước:

1. Tại những ô nhớ quy định, ghi nhận các đặc trưng của số hiệu ngắt vừa phát sinh (tùy thuộc vào số liệu được đưa vào ô nhớ tương ứng). Ví dụ với máy IBM 360-370 có các số hiệu để phân biệt các kiểu ngắt như sau:

   -Ngắt vào-ra

   -Ngắt theo chương trình: vi phạm cách thức phương tiện máy: lệnh không chính quy; dữ liệu không chính quy;

   -Ngắt hướng tới supervisor (gọi chương trình supervisor và thay chế độ làm việc của CPU);

   -Ngắt ngoài: có tín hiệu hướng tới CPU, ngắt theo thời gian, ngắt khi có tín hiệu của các bộ xử lý khác…;

   -Ngắt theo sơ đồ kiểm tra.
2. Ghi nhớ trạng thái của quá trình bị ngắt: giá trị bộ đếm lệnh (chú ý từ trạng thái chương trình PSW: Program Status Word, trên bàn điều khiển có một hàng đèn tương ứng với từ máy…)
3. Thanh ghi địa chỉ lệnh hướng tới địa chỉ để xử lý ngắt.
4. Ngắt được xử lý.
5. Quay lại quá trình đã bị ngắt (nếu được)

Các bước 1-3 do các thành phần chức năng của máy tính đảm nhận, bước 4-5 do chương trình xử lý ngắt đảm nhận.

Bước 4. chương trình xử lý ngắt tiến hành các công việc:

Ghi nhớ bổ sung một số thông tin mà do cách thức phương tiện (bước 2) chưa ghi hết, ví dụ, bước 2 ghi PSW còn chương trình xử lý ngắt phải bảo

vệ trạng thái của quá trình bị ngắt bằng việc lưu trữ hệ thống các thanh ghi chung và công việc nói trên đòi hỏi một vùng bộ nhớ nhất định (chẳng hạn, với IBM, EC đòi hỏi vùng 72 bytes cho 16 thanh và 2 địa chỉ chuyển đổi).

Định danh chương trình xử lý ngắt.

Thông tin bước 3 là bộ phận đối với chương trình xử lý ngắt: mỗi loại ngắt có thể do một chương trình ngắt riêng, ví dụ ngắt do vào ra (thiết lập cách thức phương tiện ở bước 1) khác biệt hoàn toàn với ngắt hướng tới supervisor (phân tích tác động tiếp theo supervisor).

- Thực hiện tác động tương ứng với ngắt đã được định danh.

Các tác động này hết sức đơn giản. Ví dụ, chỉ thiết lập dấu hiệu nào đó như trạng thái tràn ô, hoặc quay lại băng từ chuyển sang việc chuẩn bị đọc nếu đã đọc sai.v.v…

Nếu không quá gấp, chương trình xử lý ngắt tương ứng sẽ được ghi vào dòng xếp hàng quá trình ở trạng thái chuẩn bị.

Chương trình xử lý ngắt đảm bảo việc quay về trạng thái bình thường của CPU (chọn quá trình người dùng để thực hiện) tùy thuộc vào:

-Kiểu ngắt;

-Kiểu của chương trình điều phối CPU được sử dụng.

Từ các yếu tố trên sẽ xác định công việc kết khối, về trạng thái chuẩn bị và các công việc được chọn tiếp theo…

Chú ý:

Một số tác động của chương trình xử lý ngắt được thực hiện chậm nếu để ở bộ nhớ ngoài cho nên đưa ra giải pháp một số bộ phận của chương trình xử lý ngắt được đặt thường trực trong bộ nhớ trong như là một phần trong nhân hệ thống. Nếu chương trình xử lý ngắt quá lớn, nó được chia làm hai phần: phần thường trực và phần không thường trực.

Nhiều ngắt có quan hệ đến điều khiển CPU (ngắt theo thời gian, ngắt theo hoạt động thiết bị, ngắt hoàn thiện vào/ra). Quá trình do điều phối làm không chỉ là quá trình người dùng mà còn là những bộ phận khác nhau của hệ điều hành (bao hàm chương trình xử lý ngắt mức 2; chương trình con thống kê; điều phối chính; tải và thậm chí chính cả điều phối).

Ngắt đa mức

Ngắt xảy ra có thể đối với chương trình người dùng, có thể xảy ra chính trong quá trình đang xử lý ngắt. Đây là tình huống được gọi là ngắt đa mức. Xử lý ngắt đa mức ra sao?

-Phân cấp các loại ngắt theo độ ưu tiên, thông thường ngắt liên quan tới cách thức kĩ thuật có độ ưu tiên thấp hơn so với các ngắt có liên quan đến hệ điều hành. Ví dụ: ngắt gọi supervisor có độ ưu tiên cao hơn so với ngắt vào/ra.

-Chọn ngắt nào được xử lý trước tiên: ngắt cũ và ngắt mới, việc đó tùy thuộc vào kiểu của hai ngắt. Ngắt mới hoặc được giải quyết ngay (ngắt trội hơn), hoặc bị hủy bỏ, hoặc chờ để giải quyết tiếp theo.

Xử lý ngắt đa mức theo các độ ưu tiên khác nhau được đảm bảo theo các cách thức phương tiện khác nhau ghi nhận mỗi kiểu ngắt khác nhau trên các ô nhớ khác nhau.

Chương 4: QUẢN LÝ BỘ NHỚ TRONG

Quản lý bộ nhớ là một trong những nhiệm vụ quan trọng và phức tạp nhất của hệ điều hành. Bộ phận quản lý bộ nhớ xem bộ nhớ chính như là một tài nguyên của hệ thống dùng để cấp phát và chia sẻ cho nhiều tiến trình đang ở trong trạng thái hoạt động. Các hệ điều hành đều mong muốn có nhiều hơn các tiến trình trên bộ nhớ chính. Công cụ cơ bản của quản lý bộ nhớ là sự phân trang (paging) và sự phân đoạn (segmentation). Với sự phân trang mỗi tiến trình được chia thành nhiều phần nhỏ có quan hệ với nhau, với kích thước của trang là cố định. Sự phân đoạn cung cấp cho chương trình người sử dụng các khối nhớ có kích thước khác nhau. Hệ điều hành cũng có thể kết hợp giữa phân trang và phân đoạn để có được một chiến lược quản lý bộ nhớ linh hoạt hơn.

4.1. Các khái niệm cơ bản

Trong các hệ thống đơn chương trình (uniprogramming), trên bộ nhớ chính ngoài hệ điều hành, chỉ có một chương trình đang thực hiện. Trong các hệ thống đa chương (multiprogramming) trên bộ nhớ chính ngoài hệ điều hành, có thể có nhiều tiến trình đang hoạt động. Do đó nhiệm vụ quản lý bộ nhớ của hệ điều hành trong hệ thống đa chương trình sẽ phức tạp hơn nhiều so với trong hệ thống đơn chương trình. Trong hệ thống đa chương bộ phận quản lý bộ nhớ phải có nhiệm vụ đưa bất kỳ một tiến trình nào đó vào bộ nhớ khi nó có yêu cầu, kể cả khi trên bộ nhớ không còn không gian trống, ngoài ra nó phải bảo vệ chính hệ điều hành và các tiến trình trên bộ nhớ tránh các trường hợp truy xuất bất hợp lệ xảy ra. Như vậy việc quản lý bộ nhớ trong các hệ thống đa chương là quan trọng và cần thiết. Bộ phận quản lý bộ nhớ phải thực hiện các nhiệm vụ sau đây:

Sự tái định vị (Relocation): Trong các hệ thống đa chương, không gian bộ nhớ chính thường được chia sẻ cho nhiều tiến trình khác nhau và yêu cầu bộ nhớ của các tiến trình luôn lớn hơn không gian bộ nhớ vật lý mà hệ thống có được. Do đó, một chương trình đang hoạt động trên bộ nhớ cũng có thể bị đưa ra đĩa (swap-out) và nó sẽ được đưa vào lại (swap-in) bộ nhớ tại một thời điểm thích hợp nào đó sau này. Vấn đề đặt ra là khi đưa một chương trình vào lại bộ nhớ thì hệ điều hành phải định vị nó vào đúng vị trí mà nó đã được nạp trước đó. Để thực hiện được điều này hệ điều hành phải có các cơ chế để ghi lại tất cả các thông tin liên quan đến một chương trình bị swap-out, các thông tin này là cơ sở để hệ điều hành swap-in chương trình vào lại bộ nhớ chính và cho nó tiếp tục hoạt động. Hệ điều hành buộc phải swap-out một chương trình vì nó còn không gian bộ nhớ chính để nạp tiến trình khác, do đó sau khi

swap-out một chương trình hệ điều hành phải tổ chức lại bộ nhớ để chuẩn bị nạp tiến trình vừa có yêu cầu. Các nhiệm vụ trên do bộ phần quản lý bộ nhớ của hệ điều hành thực hiện. Ngoài ra trong nhiệm vụ này hệ điều hành phải có khả năng chuyển đổi các địa chỉ bộ nhớ được ghi trong code của chương trình thành các địa chỉ vật lý thực tế trên bộ nhớ chính khi chương trình thực hiện các thao tác truy xuất trên bộ nhớ, bởi vì người lập trình không hề biết trước hiện trạng của bộ nhớ chính và vị trí mà chương trình được nạp khi chương trình của họ hoạt động. Trong một số trường hợp khác các chương trình bị swap-out có thể được swap-in vào lại bộ nhớ tại vị trí khác với vị trí mà nó được nạp trước đó.

Bảo vệ bộ nhớ (Protection): Mỗi tiến trình phải được bảo vệ để chống lại sự truy xuất bất hợp lệ vô tình hay có chủ ý của các tiến trình khác. Vì thế các tiến trình trong các chương trình khác không thể tham chiếu đến các vùng nhớ đã dành cho một tiến trình khác để thực hiện các thao tác đọc/ghi mà không được phép (permission), mà nó chỉ có thể truy xuất đến không gian địa chỉ bộ nhớ mà hệ điều hành đã cấp cho tiến trình đó. Để thực hiện điều này hệ thống quản lý bộ nhớ phải biết được không gian địa chỉ của các tiến trình khác trên bộ nhớ và phải kiểm tra tất cả các yêu cầu truy xuất bộ nhớ của mỗi tiến trình khi tiến trình đưa ra địa chỉ truy xuất. Điều này khó thực hiện vì không thể xác định địa chỉ của các chương trình trong bộ nhớ chính trong quá trình biên dịch mà phải thực hiện việc tính toán địa chỉ tại thời điểm chạy chương trình. Hệ điều hành có nhiều chiến lược khác nhau để thực hiện điều này.

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

Chia sẻ bộ nhớ (Sharing): Bất kỳ một chiến lược nào được cài đặt đều phải có tính mềm dẻo để cho phép nhiều tiến trình có thể truy cập đến cùng một địa chỉ trên bộ nhớ chính. Ví dụ, khi có nhiều tiến trình cùng thực hiện một chương trình thì việc cho phép mỗi tiến trình cùng truy cập đến một bản copy của chương trình sẽ thuận lợi hơn khi cho phép mỗi tiến trình truy cập đến một bản copy sở hữu riêng. Các tiến trình đồng thực hiện (co-operating) trên một vài tác vụ có thể cần để chia sẻ truy cập đến cùng một cấu trúc dữ liệu. Hệ thống quản lý bộ nhớ phải điều khiển việc truy cập đến không gian bộ nhớ được chia sẻ mà không vi phạm đến các yêu cầu bảo vệ bộ nhớ. Ngoài ra, trong môi trường hệ điều hành đa nhiệm hệ điều hành phải chia sẻ không gian nhớ cho các tiến trình để hệ điều hành có thể nạp được nhiều tiến trình vào bộ nhớ để các tiến trình này có thể hoạt động đồng thời với nhau.

Tổ chức bộ nhớ logic (Logical organization): Bộ nhớ chính của hệ thống máy

tính được tổ chức như là một dòng hoặc một mảng, không gian địa chỉ bao gồm một dãy có thứ tự các byte hoặc các word. Bộ nhớ phụ cũng được tổ chức tương tự. Mặc dù việc tổ chức này có sự kết hợp chặt chẽ với phần cứng thực tế của máy nhưng nó không phù hợp với các chương trình. Đa số các chương trình đều được chia thành các modun, một vài trong số đó là không thể thay đổi (read only, execute only) và một vài trong số đó chứa dữ liệu là có thể thay đổi. Nếu hệ điều hành và phần cứng máy tính có thể giao dịch một cách hiệu quả với các chương trình của người sử dụng và dữ liệu trong các modun thì một số thuận lợi có thể thấy rõ sau đây:

- Các modun có thể được viết và biên dịch độc lập, với tất cả các tham chiếu từ một modun đến modun khác được giải quyết bởi hệ thống tại thời điểm chạy.
- Các mức độ khác nhau của sự bảo vệ, read-only, execute-only, có thể cho ra các modun khác nhau.
- Nó có thể đưa ra các cơ chế để các modun có thể được chia sẻ giữa các tiến trình.

Công cụ đáp ứng cho yêu cầu này là sự phân đoạn (segmentation), đây là một trong những kỹ thuật quản lý bộ nhớ được trình bày trong chương này.

Tổ chức bộ nhớ vật lý (Physical organization): Như chúng ta đã biết bộ nhớ máy tính được tổ chức theo 2 cấp: bộ nhớ chính và bộ nhớ phụ. Bộ nhớ chính cung cấp một tốc độ truy cập dữ liệu cao, nhưng dữ liệu trên nó phải được làm tươi thường xuyên và không thể tồn tại lâu dài trên nó. Bộ nhớ phụ có tốc độ truy xuất chậm và rẻ tiền hơn so với bộ nhớ chính nhưng nó không cần làm tươi thường xuyên. Vì thế bộ nhớ phụ có khả năng lưu trữ lớn và cho phép lưu trữ dữ liệu và chương trình trong một khoảng thời gian dài, trong khi đó bộ nhớ chính chỉ để giữ (hold) một khối lượng nhỏ các chương trình và dữ liệu đang được sử dụng tại thời điểm hiện tại.

Trong giản đồ 2 cấp này, việc tổ chức luồng thông tin giữa bộ nhớ chính và bộ nhớ phụ là một nhiệm vụ quan trọng của hệ thống. Sự chịu trách nhiệm cho luồng này có thể được gán cho từng người lập trình riêng, nhưng điều này là không hợp lý và có thể gây rắc rối, là do hai nguyên nhân:

- Không gian bộ nhớ chính dành cho các chương trình cùng với dữ liệu của nó thường là không đủ, trong trường hợp này, người lập trình phải tiến hành một thao tác được hiểu như là Overlaying, theo đó chương trình và dữ liệu được tổ chức thành các modun khác nhau có thể được gán trong cùng một vùng của bộ nhớ, trong đó có một chương trình chính chịu trách nhiệm chuyển các modun vào và ra khi cần.
- Trong môi trường đa chương trình, người lập trình không thể biết tại một thời điểm xác định có bao nhiêu không gian nhớ còn trống hoặc khi nào

thì không gian nhớ sẽ trống.

Như vậy nhiệm vụ di chuyển thông tin giữa 2 cấp bộ nhớ phải do hệ thống thực hiện. Đây là nhiệm vụ cơ bản mà thành phần quản lý bộ nhớ phải thực hiện.

4.2. Các kỹ thuật cấp phát bộ nhớ

4.2.1. Kỹ thuật phân vùng cố định (Fixed Partitioning)

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

Trong các hệ thống đơn chương, phần user program được dùng để cấp cho chỉ một chương trình duy nhất, do đó nhiệm vụ quản lý bộ nhớ của hệ điều hành trong trường hợp này sẽ đơn giản hơn, hệ điều hành chỉ kiểm soát sự truy xuất bộ nhớ của chương trình người sử dụng, không cho nó truy xuất lên vùng nhớ của hệ điều hành. Để thực hiện việc này hệ điều hành sử dụng một thanh ghi giới hạn để ghi địa chỉ ranh giới giữa hệ điều hành và chương trình của người sử dụng, theo đó khi chương trình người sử dụng cần truy xuất một địa chỉ nào đó thì hệ điều hành sẽ so sánh địa chỉ này với giá trị địa chỉ được ghi trong thành ghi giới hạn, nếu nhỏ hơn thì từ chối không cho truy xuất, ngược lại thì cho phép truy xuất. Việc so sánh địa chỉ này cần phải có sự hỗ trợ của phần cứng và có thể làm giảm tốc độ truy xuất bộ nhớ của hệ thống nhưng bảo vệ được hệ điều hành tránh việc chương trình của người sử dụng làm hỏng hệ điều hành dẫn đến làm hỏng hệ thống.

Trong các hệ thống đa chương, phần user program lại được phân ra thành nhiều phân vùng (partition) với các biên vùng cố định có kích thước bằng nhau hay không bằng nhau. Trong trường hợp này một tiến trình có thể được nạp vào bất kỳ partition nào nếu kích thước của nó nhỏ hơn hoặc bằng kích thước của partition và partition này còn trống. Khi có một tiến trình cần được nạp vào bộ nhớ nhưng tất cả các partition đều đã chứa các tiến trình khác thì hệ điều hành có thể chuyển một tiến trình nào đó, mà hệ điều hành cho là hợp lệ (kích thước vừa đủ, không đang ở trạng thái ready hoặc running, không có quan hệ với các tiến trình running khác, ...), ra ngoài (swap out), để lấy partition trống đó nạp tiến trình vừa có yêu cầu. Đây là nhiệm vụ phức tạp của hệ điều hành, hệ điều hành phải chi phí cao cho công việc này.

Có hai trở ngại trong việc sử dụng các phân vùng cố định với kích thước bằng nhau:

- Thứ nhất, khi kích thước của một chương trình là quá lớn so với kích thước của một partition thì người lập trình phải thiết kế chương trình theo cấu

trúc overlay, theo đó chỉ những phần chia cần thiết của chương trình mới được nạp vào bộ nhớ chính khi khởi tạo chương trình, sau đó người lập trình phải nạp tiếp các modun cần thiết khác vào đúng partition của chương trình và sẽ ghi đè lên bất kỳ chương trình hoặc dữ liệu ở trong đó. Cấu trúc chương trình overlay tiết kiệm được bộ nhớ nhưng yêu cầu cao ở người lập trình.

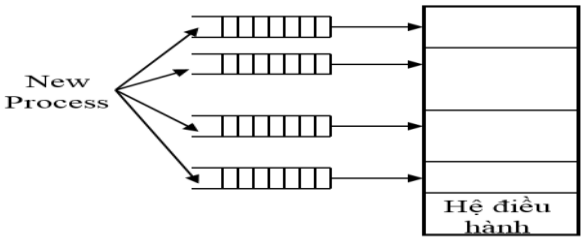
- Thứ hai, khi kích thước của một chương trình nhỏ hơn kích thước của một partition hoặc quá lớn so với kích thước của một partition nhưng không phải là bội số của kích thước một partition thì dễ xảy ra hiện tượng phân mảnh bên trong (internal fragmentation) bộ nhớ, gây lãng phí bộ nhớ. Ví dụ, nếu có 3 không gian trống kích thước 30K nằm rãi rác trên bộ nhớ, thì cũng sẽ không nạp được một modun chương trình có kích thước 12K, hiện tượng này được gọi là hiện tượng phân mảnh bên trong.

Cả hai vấn đề trên có thể được khắc phục bằng cách sử dụng các phân vùng có kích thước không bằng nhau.

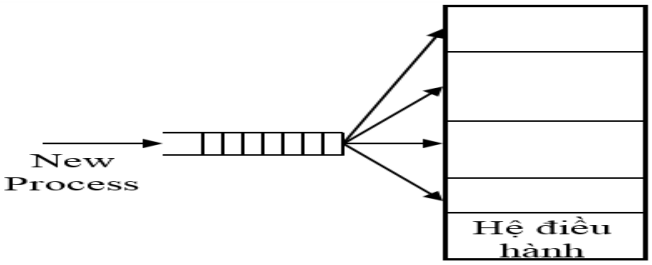
Việc đưa một tiến trình vào partition trong hệ thống đa chương với phân vùng cố định kích thước không bằng nhau sẽ phức tạp hơn nhiều so với trường hợp các phân vùng có kích thước bằng nhau. Với các partition có kích thước không bằng nhau thì có hai cách để lựa chọn khi đưa một tiến trình vào partition:

- Mỗi phân vùng có một hàng đợi tương ứng, theo đó mỗi tiến trình khi cần được nạp vào bộ nhớ nó sẽ được đưa đến hành đợi của phân vùng có kích thước vừa đủ để chứa nó, để vào/để đợi được vào phân vùng. Cách tiếp cận này sẽ đơn giản trong việc đưa một tiến trình từ hàng đợi vào phân vùng vì không có sự lựa chọn nào khác ở đây, khi phân vùng mà tiến trình đợi trống nó sẽ được đưa vào phân vùng đó. Tuy nhiên các tiếp cận này kém linh động vì có thể có một phân vùng đang trống, trong khi đó có nhiều tiến trình đang phải phải đợi để được nạp vào các phân vùng khác, điều này gây lãng phí trong việc sử dụng bộ nhớ.

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



Hình 4.1a. Mỗi Partition có một hàng đợi riêng



Hình 4.1b. Một hàng đợi chung cho tất cả Partition

Mặc dầu sự phân vùng cố định với kích thước không bằng nhau cung cấp một sự mềm dẻo hơn so với phân vùng cố định với kích thước bằng nhau, nhưng cả hai loại này còn một số hạn chế sau đây:

- Số lượng các tiến trình có thể hoạt động trong hệ thống tại một thời điểm phụ thuộc vào số lượng các phân vùng cố định trên bộ nhớ.
- Tương tự như trên, nêu kích thước của tiến trình nhỏ hơn kích thước của một phân vùng thì có thể dẫn đến hiện tượng phân mảnh nội vi gây lãng phí trong việc sử dụng bộ nhớ.

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

4.2.2. Kỹ thuật phân vùng động (Dynamic Partitioning)

Để khắc phục một vài hạn chế của kỹ thuật phân vùng cố định, kỹ thuật phân vùng động ra đời. Kỹ thuật này thường được sử dụng trong các hệ điều hành gần đây như hệ điều hành mainframe của IBM, hệ điều hành OS/MVT,...

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

Để chống lại sự lãng phí bộ nhớ do phân mảnh, thỉnh thoảng hệ điều hành phải thực hiện việc sắp xếp lại bộ nhớ, để các không gian nhớ nhỏ rời rác nằm liền kề lại với nhau tạo thành một khối nhớ có kích thước đủ lớn để chứa được một tiến trình nào đó. Việc làm này làm chậm tốc độ của hệ thống, hệ điều hành phải chi phí cao cho việc này, đặc biệt là việc tái định vị các tiến trình khi một tiến trình bị đưa ra khỏi bộ nhớ và được nạp vào lại bộ nhớ để tiếp tục hoạt động.

Trong kỹ thuật phân vùng động này hệ điều hành phải đưa ra các cơ chế thích hợp để quản lý các khối nhớ đã cấp phát hay còn trống trên bộ nhớ. Hệ điều hành sử dụng 2 cơ chế: Bản đồ bít và Danh sách liên kết. Trong cả 2 cơ chế này hệ điều hành đều chia không gian nhớ thành các đơn vị cấp phát có kích thước bằng nhau, các đơn vị cấp phát liên tiếp nhau tạo thành một khối nhớ (block), hệ điều hành cấp phát các block này cho các tiến trình khi nạp tiến trình vào bộ nhớ.

Trong cơ chế bản đồ bít: mỗi đơn vị cấp phát được đại diện bởi một bít trong bản đồ bít. Đơn vị cấp phát còn trống được đại diện bằng bít 0, ngược lại đơn vị cấp phát được đại diện bằng bít 1.

Trong cơ chế danh sách liên kết: Mỗi block trên bộ nhớ được đại diện bởi một phần tử trong danh sách liên kết, mỗi phần tử này gồm có 3 trường chính: trường thứ nhất cho biết khối nhớ đã cấp phát (P: process) hay đang còn trống (H: Hole), trường thứ hai cho biết thứ tự của đơn vị cấp phát đầu tiên trong block, trường thứ ba cho biết block gồm bao nhiêu đơn vị cấp phát.

Như vậy khi cần nạp một tiến trình vào bộ nhớ thì hệ điều hành phải dựa vào bản đồ bit hoặc danh sách liên kết để tìm ra một block có kích thước đủ để nạp tiến trình. Sau khi thực hiện một thao tác cấp phát hoặc sau khi đưa một tiến trình ra khỏi bộ nhớ thì hệ điều hành phải cập nhật lại bản đồ bít hoặc danh sách liên kết, điều này có thể làm giảm tốc độ thực hiện của hệ thống.

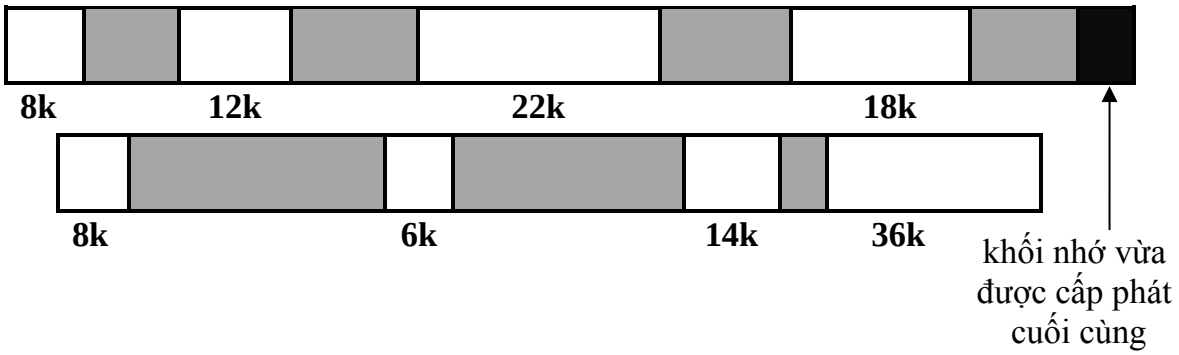
Chọn kích thước của một đơn vị cấp phát là một vấn đề quan trọng trong thiết kế, nếu kích thước đơn vị cấp phát nhỏ thì bản đồ bít sẽ lớn, hệ thống phải tốn bộ nhớ để chứa nó. Nếu kích thước của một đơn vị cấp phát lớn thì bản đồ bít sẽ nhỏ, nhưng sự lãng phí bộ nhớ ở đơn vị cấp phát cuối cùng của một tiến trình sẽ lớn khi kích thước của tiến trình không phải là bội số của một đơn vị cấp phát. Điều vừa trình bày cũng đúng trong trường hợp danh sách liên kết.

Danh sách liên kết có thể được sắp xếp theo thứ tự tăng dần hoặc giảm dần của kích thước hoặc địa chỉ, điều này giúp cho việc tìm khối nhớ trống có kích thước vừa đủ để nạp các tiến trình theo các thuật toán dưới đây sẽ đạt tốc độ nhanh hơn và hiệu quả cao hơn. Một số hệ điều hành tổ chức 2 danh sách liên kết riêng để theo dõi các đơn vị cấp phát trên bộ nhớ, một danh sách để theo dõi các block đã cấp phát và một danh dách để theo dõi các block còn trống. Cách này giúp việc tìm các khối nhớ trống nhanh hơn, chỉ tìm trên danh sách các khối nhớ trống, nhưng tốn thời gian nhiều hơn cho việc cấp nhật danh sách sau mỗi thao tác cấp phát, vì phải thực hiện trên cả hai danh sách.

Khi có một tiến trình cần được nạp vào bộ nhớ mà trong bộ nhớ có nhiều hơn một khối nhớ trống (Free Block) có kích thước lớn hơn kích thước của tiến trình đó,

thì hệ điều hành phải quyết định chọn một khối nhớ trống phù hợp nào để nạp tiến trình sao cho việc lựa chọn này dẫn đến việc sử dụng bộ nhớ chính là hiệu quả nhất. Có 3 thuật toán mà hệ điều hành sử dụng trong trường hợp này, đó là: Best-fit, First-fit, và Next-fit. Cả 3 thuật toán này đều phải chọn một khối nhớ trống có kích thước bằng hoặc lớn hơn kích thước của tiến trình cần nạp vào, nhưng nó có các điểm khác nhau cơ bản sau đây:

- **Best-fit:** chọn khối nhớ có kích thước vừa đúng bằng kích thước của tiến trình cần được nạp vào bộ nhớ.
- **First-fit:** trong trường hợp này hệ điều hành sẽ bắt đầu quét qua các khối nhớ trống bắt đầu từ khối nhớ trống đầu tiên trong bộ nhớ, và sẽ chọn khối nhớ trống đầu tiên có kích thước đủ lớn để nạp tiến trình.



Hình 4.2. Ví dụ về các thuật toán cấp phát bộ nhớ

- **Next-fit:** tương tự như First-fit nhưng ở đây hệ điều hành bắt đầu quét từ khối nhớ trống kế sau khối nhớ vừa được cấp phát và chọn khối nhớ trống kế tiếp đủ lớn để nạp tiến trình.

Hình vẽ 3.4 cho thấy hiện tại trên bộ nhớ có các khối nhớ chưa đươc cấp phát theo thứ tự là: 8k, 12k, 22k, 18k, 8k, 6k, 14k, 36k. Trong trường hợp này nếu có một tiến trình có kích thước 16k cần được nạp vào bộ nhớ, thì hệ điều hành sẽ nạp nó vào:

- khối nhớ 22k nếu theo thuật toán First-fit
- khối nhớ 18k nếu theo thuật toán Best-fit
- khối nhớ 36k nếu theo thuật toán Next-fit

Như vậy nếu theo Best-fit thì sẽ xuất hiện một khối phân mảnh 2k, nếu theo First-fit thì sẽ xuất hiện một khối phân mảnh 6k, nếu theo Next-fit thì sẽ xuất hiện một khối phân mảnh 20k.

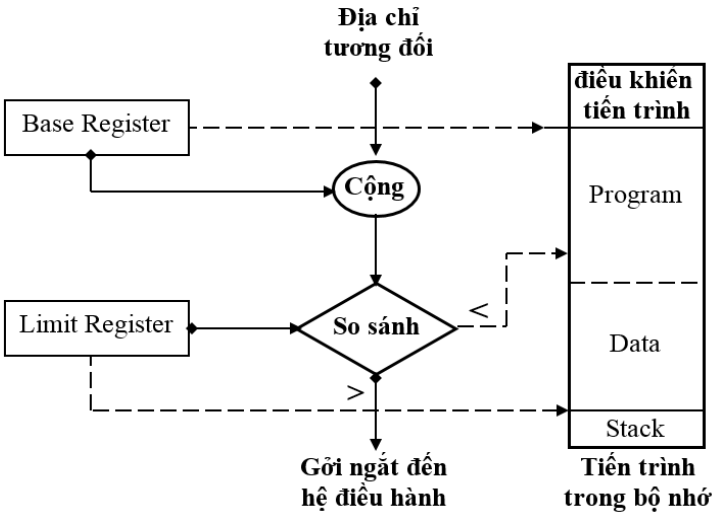
Các hệ điều hành không cài đặt cố định trước một thuật toán nào, tuỳ vào trường hợp cụ thể mà nó chọn cấp phát theo một thuật toán nào đó, sao cho chi phí về việc cấp phát là thấp nhất và hạn chế được sự phân mảnh bộ nhớ sau này. Việc chọn thuật toán này thường phụ thuộc vào thứ tự swap và kích thước của tiến trình. Thuật toán First-fit được đánh giá là đơn giản, dễ cài đặt nhưng mang lại hiệu quả cao

nhất đặc biệt là về tốc độ cấp phát. Về hiệu quả thuật toán Next-fit không bằng First-fit, nhưng nó thường xuyên sử dụng được các khối nhớ trống ở cuối vùng nhớ, các khối nhớ ở vùng này thường có kích thước lớn nên có thể hạn chế được sự phân mảnh, theo ví dụ trên thì việc xuất hiện một khối nhớ trống 20k sau khi cấp một tiến trình 16k thì không thể gọi là phân mảnh được, nhưng nếu tiếp tục như thế thì dễ dẫn đến sự phân mảnh lớn ở cuối bộ nhớ. Thuật toán Best-fit, không như tên gọi của nó, đây là một thuật toán có hiệu suất thấp nhất, trong trường hợp này hệ điều hành phải duyệt qua tất các các khối nhớ trống để tìm ra một khối nhớ có kích thước vừa đủ để chứa tiến trình vừa yêu cầu, điều này làm giảm tốc độ cấp phát của hệ điều hành. Mặt khác với việc chọn kích thước vừa đủ có thể dẫn đến sự phân mảnh lớn trên bộ nhớ, tức là có quá nhiều khối nhớ có kích thước quá nhỏ trên bộ nhớ, nhưng nếu xét về mặt lãng phí bộ nhớ tại thời điểm cấp phát thì thuật toán này làm lãng phí ít nhất. Tóm lại, khó có thể đánh giá về hiệu quả sử dụng của các thuật toán này, vì hiệu quả của nó được xét trong "tương lai" và trên nhiều khía cạnh khác nhau chứ không phải chỉ xét tại thời điểm cấp phát. Và hơn nữa trong bản thân các thuật toán này đã có các mâu thuẩn với nhau về hiệu quả sử dụng của nó.

Do yêu cầu của công tác cấp phát bộ nhớ của hệ điều hành, một tiến trình đang ở trên bộ nhớ có thể bị đưa ra ngoài (swap-out) để dành chỗ nạp một tiến trình mới có yêu cầu, và tiến trình này sẽ được nạp vào lại (swap-in) bộ nhớ tại một thời điểm thích hợp sau này. Vấn đề đáng quan tâm ở đây là tiến trình có thể được nạp vào lại phân vùng khác với phân vùng mà nó được nạp vào lần đầu tiên. Có một lý do khác khiến các tiến trình phải thay đổi vị trí nạp so với ban đầu là khi có sự liên kết giữa các môdun tiến trình của một chương trình thì các tiến trình phải dịch chuyển ngay cả khi chúng đã nằm trên bộ nhớ chính. Sự thay đổi vị trị/địa chỉ nạp này sẽ ảnh hưởng đến các thao tác truy xuất dữ liệu của chương trình vì nó sẽ khác với các địa chỉ tương đối mà người lập trình đã sử dụng trong code của chương trình. Ngoài ra khi một tiến trình được nạp vào bộ nhớ lần đầu tiên thì tất cả các địa chỉ tương đối được tham chiếu trong code chương trình được thay thế bằng địa chỉ tuyệt đối trong bộ nhớ chính, địa chỉ này được xác định bởi địa chỉ cơ sở, nơi tiến trình được nạp. Ví dụ trong chương trình có code truy xuất đến địa chỉ tương đối 100k, nếu chương trình này được nạp vào phân vùng 1 có địa chỉ bắt đầu là 100k thì địa chỉ truy xuất là 200k, nhưng nếu chương trình được nạp vào phân vùng 2 có địa chỉ bắt đầu là 200k, thì địa chỉ truy xuất sẽ là 300k. Để giải quyết vấn đề này hệ điều hành phải thực hiện các yêu cầu cần thiết của công tác tái định vị một tiến trình vào lại bộ nhớ. Ngoài ra ở đây hệ điều hành cũng phải tính đến việc bảo vệ các tiến trình trên bộ nhớ tránh tình trạng một tiến trình truy xuất đến vùng nhớ của tiến trình khác. Trong trường hợp này hệ điều hành sử dụng 2 thanh ghi đặc biệt:

- Thanh ghi cơ sở (base register): dùng để ghi địa chỉ cơ sở của tiến trình tiến trình được nạp vào bộ nhớ.
- Thanh ghi giới hạn (limit register): dùng để ghi địa chỉ cuối cùng của tiến trình trong bộ nhớ.

Khi một tiến trình được nạp vào bộ nhớ thì hệ điều hành sẽ ghi địa chỉ bắt đầu của phân vùng được cấp phát cho tiến trình vào thanh ghi cơ sở và địa chỉ cuối cùng của tiến trình vào thanh ghi giớ hạn. Việc thiết lập giá trị của các thanh ghi này được thực hiện cả khi tiến trình lần đầu tiên được nạp vào bộ nhớ và khi tiến trình được swap in vào lại bộ nhớ. Theo đó mỗi khi tiến trình thực hiện một thao tác truy xuất bộ nhớ thì hệ thống phải thực hiện 2 bước: Thứ nhất, cộng địa chỉ ô nhớ do tiến trình phát ra với giá trị địa chỉ trong thanh ghi cơ sở để có được địa chỉ tuyệt đối của ô nhớ cần truy xuất. Thứ hai, địa chỉ kết quả ở trên sẽ được so sánh với giá trị địa chỉ trong thành ghi giới hạn. Nếu địa chỉ nằm trong phạm vị giới hạn thì hệ điều hành cho phép tiến trình truy xuất bộ nhớ, ngược lại thì có một ngắt về lỗi truy xuất bộ nhớ được phát sinh và hệ điều hành không cho phép tiến trình truy xuất vào vị trí bộ nhớ mà nó yêu cầu. Như vậy việc bảo bệ truy xuất bất hợp lệ được thực hiện dễ dàng ở đây.



Hình 4.3. Tái định vị với sự hỗ trợ của phần cứng

Trong hệ thống đa chương sử dụng sự phân vùng động, nếu có một tiến trình mới cần được nạp vào bộ nhớ, trong khi bộ nhớ không còn chỗ trống và tất cả các

tiến trình trên bộ nhớ đều ở trạng thái khoá (blocked), thì hệ thống phải đợi cho đến khi có một tiến trình được chuyển sang trạng thái không bị khoá (unblocked) để tiến trình này có điều kiện trả lại không gian nhớ mà nó chiếm giữ cho hệ thống: tiến trình hoạt động và kết thúc, tiến trình bị đưa ra khỏi bộ nhớ chính,..., để hệ thống nạp tiến trình vừa có yêu cầu. Sự chờ đợi này làm lãng phí thời gian xử lý của processor. Để tiết kiệm thời gian xử lý của processor trong trường hợp này hệ điều hành chọn ngay một tiến trình đang ở trạng thái khoá để đưa ra ngoài lấy không gian nhớ trống đó cấp cho tiến trình vừa có yêu cầu mà không phải đợi như ở trên. Hệ điều hành sử dụng nhiều thuật toán khác nhau cho việc chọn một tiến trình để thay thế trong trường hợp này, tất cả các thuật toán này đều hướng tới mục dích: tiết kiệm thời gian xử lý của processor, tốc độ thay thế cao, sử dụng bộ nhớ hiệu quả nhất và đặc biệt là không để dẫn đến sự trì trệ hệ thống. Chúng ta sẽ thảo luận rõ hơn về vấn đề này ở phần sau của chương này.

Chú ý: Một nhược điểm lớn của các kỹ thuật ở trên là dẫn đến hiện tượng phân mảnh bộ nhớ bên trong và bên ngoài (internal, external) gây lãng phí bộ nhớ nên hiệu quả sử dụng bộ nhớ kém. Để khắc phục hệ điều hành sử dụng các kỹ thuật phân trang hoặc phân đoạn bộ nhớ.

4.2.3. Kỹ thuật phân trang đơn (Simple Paging)

Trong kỹ thuật này không gian địa chỉ bộ nhớ vật lý được chia thành các phần có kích thước cố định bằng nhau, được đánh số địa chỉ bắt đầu từ 0 và được gọi là các khung trang (page frame). Không gian địa chỉ của các tiến trình cũng được chia thành các phần có kích thước bằng nhau và bằng kích thước của một khung trang, được gọi là các trang (page) của tiến trình.

Khi một tiến trình được nạp vào bộ nhớ thì các trang của tiến trình được nạp vào các khung trang còn trống bất kỳ, có thể không liên tiếp nhau, của bộ nhớ. Khi hệ điều hành cần nạp một tiến trình có n trang vào bộ nhớ thì nó phải tìm đủ n khung trang trống để nạp tiến trình này. Nếu kích thước của tiến trình không phải là bội số của kích thước một khung trang thì sẽ xảy ra hiện tượng phân mảnh nội vi ở khung trang chứa trang cuối cùng của tiến trình. Ở đây không xảy ra hiện tượng phân mảnh ngoại vi. Trên bộ nhớ có thể tồn tại các trang của nhiều tiến trình khác nhau. Khi một tiến trình bị swap-out thì các khung trang mà tiến trình này chiếm giữ sẽ được giải phóng để hệ điều hành có thể nạp các trang tiến trình khác.

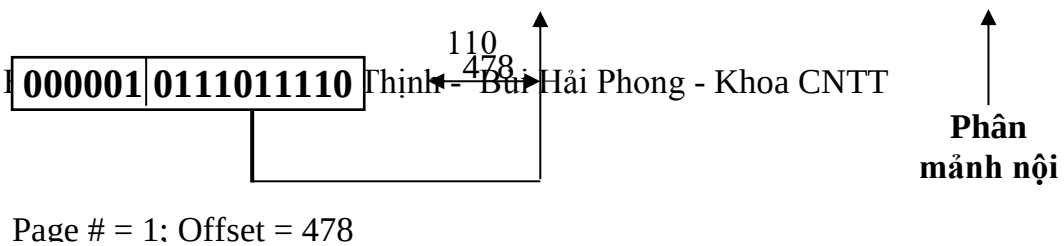
Trong kỹ thuật này hệ điều hành phải đưa ra các cơ chế thích hợp để theo dõi trạng thái của các khung trang (còn trống hay đã cấp phát) trên bộ nhớ và các khung trang đang chứa các trang của một tiến trình của các tiến trình khác nhau trên bộ nhớ. Hệ điều hành sử dụng một danh sách để ghi số hiệu của các khung trang còn trống

trên bộ nhớ, hệ điều hành dựa vào danh sách này để tìm các khung trang trống trước khi quyết định nạp một tiến trình vào bộ nhớ, danh sách này được cập nhật ngay sau khi hệ điều hành nạp một tiến trình vào bộ nhớ, được kết thúc hoặc bị swap out ra bên ngoài.

Hệ điều hành sử dụng các bảng trang (PCT: page control table) để theo dõi vị trí các trang tiến trình trên bộ nhớ, mỗi tiến trình có một bảng trang riêng. Bảng trang bao gồm nhiều phần tử, thường là bằng số lượng trang của một tiến trình mà bảng trang này theo dõi, các phần tử được đánh số bắt đầu từ 0. Phần tử 0 chứa số hiệu của khung trang đang chứa trang 0 của tiến trình, phần tử 1 chứa số hiệu của khung trang đang chứa trang 1 của tiến trình, … Các bảng trang có thể được chứa trong các thanh ghi nếu có kích thước nhỏ, nếu kích thước bảng trang lớn thì nó được chứa trong bộ nhớ chính, khi đó hệ điều hành sẽ dùng một thanh ghi để lưu trữ địa chỉ bắt đầu nơi lưu trữ bảng trang, thanh ghi này được gọi là thanh ghi PTBR: page table base register.

Trong kỹ thuật phân trang này khi cần truy xuất bộ nhớ CPU phải phát ra một địa chỉ logic gồm 2 thành phần: Số hiệu trang (Page): cho biết số hiệu trang tương ứng cần truy xuất. Địa chỉ tương đối trong trang (Offset): giá trị này sẽ được kết hợp với địa chỉ bắt đầu của trang để xác định địa chỉ vật lý của ô nhớ cần truy xuất. Việc chuyển đổi từ địa chỉ logic sang địa chỉ vật lý do processor thực hiện.

Kích thước của mỗi trang hay khung trang do phần cứng quy định và thường là lũy thừa của 2, biến đổi từ 512 byte đến 8192 byte. Nếu kích thước của không gian địa chỉ là 2^m và kích thước của trang là 2^n thì m-n bít cao của địa chỉ logic là số hiệu trang (page) và n bít còn lại là địa chỉ tương đối trong trang (offset). Ví dụ: nếu địa chỉ logic gồm 16 bit, kích thước của mỗi trang là 1K = 1024byte (2^{10}), thì có 6 bít dành cho số hiệu trang, như vậy một chương trình có thể có tối đa 2^6 = 64 trang mỗi trang 1KB. Trong trường hợp này nếu CPU phát ra một giá trị địa chỉ 16 bít là: 0000010111011110 = 1502, thì thành phần số hiệu trang là 000001 = 1, thành phần offset là 0111011110 = 478.



Page 0	Page 1	Page 2	

Hình 4.4. Các khung trang của bộ nhớ và địa chỉ logic

Việc chuyển từ địa chỉ logic sang địa chỉ vật lý được thực hiện theo các bước sau:

- Trích ra m-n bít trái nhất (thấp nhất) của địa chỉ logic để xác định số hiệu trang cần truy xuất.
- Sử dụng số hiệu trang ở trên để chỉ đến phần tử tương ứng trong bảng trang của tiến trình, để xác định khung trang tương ứng, ví dụ là k.
- Địa chỉ vật lý bắt đầu của khung trang là k x 2^n , và địa chỉ vật lý của byte cần truy xuất là số hiệu trang cộng với giá trị offset. Địa chỉ vật lý không cần tính toán, nó dễ dàng có được bằng cách nối số hiệu khung trang với giá trị offset.

Nhận xét về kỹ thuật phân trang:

- Có thể thấy sự phân trang được mô tả ở đây tương tự như sự phân vùng cố định. Sự khác nhau là với phân trang các phân vùng có kích thước nhỏ hơn, một chương trình có thể chiếm giữa nhiều hơn một phân vùng, và các phân vùng này có thể không liền kề với nhau.
- Kỹ thuật phân trang loại bỏ được hiện tượng phân mảnh ngoại vi, nhưng vẫn có thể xảy ra hiện tượng phân mảnh nội vi khi kích thước của tiến trình không đúng bằng bội số kích thược của một trang, khi đó khung trang cuối cùng sẽ không được sử dụng hết.
- Khi cần truy xuất đến dữ liệu hay chỉ thị trên bộ nhớ thì hệ thống phải cần một lần truy xuất đến bảng trang, điều này có thể làm giảm tốc độ truy xuất bộ nhớ. Để khắc phục hệ điều hành sử dụng thêm một bảng trang cache, để lưu trữ các trang bộ nhớ vừa được truy cập gần đây nhất. Bảng trang cache này sẽ được sử dụng mỗi khi CPU phát ra một địa chỉ cần truy xuất.
- Mỗi hệ điều hành có một cơ chế tổ chức bảng trang riêng, đa số các hệ điều hành đều tạo cho mỗi tiến trình một bảng trang riêng khi nó được nạp vào bộ nhớ chính. Bảng trang lớn sẽ tốn bộ nhớ để chứa nó.
- Để bảo vệ các khung trang hệ điều hành đưa thêm một bít bảo vệ vào bảng trang. Theo đó mỗi khi tham khảo vào bảng trang để truy xuất bộ nhớ hệ hống sẽ kiểm tra các thao tác truy xuất trên khung trang tương ứng có hợp lệ với thuộc tính bảo vệ của nó hay không.

- Sự phân trang không phản ánh được cách mà người sử dụng nhìn nhận về bộ nhớ. Với người sử dụng, bộ nhớ là một tập các đối tượng chương trình và dữ liệu như các segment, các thư viện, ... và các biến, các vùng nhớ chia sẻ, stack, ... Vấn đề đặt ra là tìm một cách thức biểu diễn bộ nhớ sao cho nó gần với cách nhìn nhận của người sử dụng hơn. Kỹ thuật phân đoạn bộ nhớ có thể thực hiện được mục tiêu này.

4.2.4. Kỹ thuật phân đoạn đơn (Simple Segmentation)

Trong kỹ thuật này không gian địa chỉ bộ nhớ vật lý được chia thành các phần cố định có kích thước không bằng nhau, được đánh số bắt đầu từ 0, được gọi là các phân đoạn (segment). Mỗi phân đoạn bao gồm số hiệu phân đoạn và kích thước của nó. Không gian địa chỉ của các tiến trình kể cả các dữ liệu liên quan cũng được chia thành các đoạn khác nhau và không nhất thiết phải có kích thước bằng nhau, thông thường mỗi thành phần của một chương trình/tiến trình như: code, data, stack, subprogram, ..., là một đoạn.

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

Để theo dõi các đoạn của các tiến trình khác nhau trên bộ nhớ, hệ điều hành sử dụng các bảng phân đoạn (SCT: Segment control Table) tiến trình, thông thường một tiến trình có một bảng phân đoạn riêng. Mỗi phần tử trong bảng phân đoạn gồm tối thiểu 2 trường: trương thứ nhất cho biết địa chỉ cơ sở (base) của phân đoạn mà đoạn chương trình tương ứng được nạp, trường thứ hai cho biết độ dài/giới hạn (length/limit) của phân đoạn, trường này còn có tác dụng dùng để kiểm soát sự truy xuất bất hợp lệ của các tiến trình. Các bảng phân đoạn có thể được chứa trong các thanh ghi nếu có kích thước nhỏ, nếu kích thước bảng phân đoạn lớn thì nó được chứa trong bộ nhớ chính, khi đó hệ điều hành sẽ dùng một thanh ghi để lưu trữ địa chỉ bắt đầu nơi lưu trữ bảng phân đoạn, thanh ghi này được gọi là thanh ghi STBR: Segment table base register. Ngoài ra vì số lượng các đoạn của một chương trình/tiến trình có thể thay đổi nên hệ điều hành dùng thêm thanh ghi STLR:Segment table length register, để ghi kích thước hiện tại của bảng phân đoạn. Hệ điều hành cũng tổ chức một danh sách riêng để theo dõi các segment còn trống trên bộ nhớ.

Trong kỹ thuật này địa chỉ logic mà CPU sử dụng phải gồm 2 thành phần: Số hiệu đoạn (segment): cho biết số hiệu đoạn tương ứng cần truy xuất. Địa chỉ tương đối trong đoạn (Offset): giá trị này sẽ được kết hợp với địa chỉ bắt đầu của đoạn để xác định địa chỉ vật lý của ô nhớ cần truy xuất. Việc chuyển đổi từ địa chỉ logic sang địa chỉ vật lý do processor thực hiện.

Nếu có một địa chỉ logic gồm $n + m$ bít, thì n bít trái nhất là số hiệu segment, m bít phải nhất còn lại là offset. Trong ví dụ minh hoạ sau đây thì $n = 4$ và $m = 12$,

như vậy kích thước tối đa của một segment là 2^{12} = 4096 byte. Sau đây là các bước cần thiết của việc chuyển đổi địa chỉ:

- Trích ra n bít trái nhất của địa chỉ logic để xác định số hiệu của phân đoạn cần truy xuất.
- Sử dụng số hiệu phân đoạn ở trên để chỉ đến phần tử trong bảng phân đoạn của tiến trình, để tìm địa chỉ vật lý bắt đầu của phân đoạn.
- So sánh thành phần offset của địa chỉ logic, được trích ra từ m bít phải nhất của địa chỉ logic, với thành phần length của phân đoạn. Nếu offset > length thì địa chỉ truy xuất là không hợp lệ.
- Địa chỉ vật lý mong muốn là địa chỉ vật lý bắt đầu của phân đoạn cộng với giá trị offset.

Trong sơ đồ ví dụ sau đây, ta có địa chỉ logic là: 0001001011110000, với số hiệu segment là 1, offset là 752, giả định segment này thường trú trong bộ nhớ chính tại địa chỉ vật lý là 0010000000100000, thì địa chỉ vật lý tương ứng với địa chỉ logic ở trên là: 0010000000100000 + 001011110000 = 0010001100010000.

Nhận xét về kỹ thuật phân đoạn:

- Vì các segment có kích thước không bằng nhau nên sự phân đoạn tương tự như sự phân vùng động. Sự khác nhau là với sự phân đoạn một chương trình có thể chiếm giữ hơn một phân vùng, và các phân vùnh này có thể không liền kề với nhau. Sự phân vùng loại trừ được sự phân mảnh nội vi, nhưng như sự phân vùng động nó vẫn xuất hiện hiện tượng phân mảnh ngoại vi.
- Sự phân trang là không tường minh đối với người lập trình, trong khi đó sự phân đoạn là tương minh đối với người lập trình, và nó cung cấp một sự thuận lợi để người lập trình tổ chức chương trình và dữ liệu. Người lập trình hoặc trình biên dịch có thể gán các chương trình và dữ liệu đến các đoạn nhớ khác nhau.
- Tương tự như trong kỹ thuật phân vùng động, kỹ thuật này cũng phải giải quyết vấn đề cấp phát động, ở đây hệ điều hành thường dùng thuật toán best-fit hay first-fit.
- Kỹ thuật phân đoạn thể hiện được cấu trúc logic của chương trình, nhưng nó phải cấp phát các khối nhớ có kích thước khác nhau cho các phân đoạn của chương trình trên bộ nhớ vật lý, điều này phức tạp hơn nhiều so với việc cấp phát các khung trang. Để dung hòa vấn đề này các hệ điều hành có thể kết hợp cả phân trang và phân đoạn.

4.3. Bộ nhớ ảo

Nguyên lý cơ bản của bộ nhớ ảo là vẫn dựa trên 2 kỹ thuật phân trang và phân đoạn, nhưng trong kỹ thuật bộ nhớ ảo:

- Bộ phận quản lý bộ nhớ không nạp tất cả các trang/đoạn của một tiến trình vào bộ nhớ để nó hoạt động, mà chỉ nạp các trang/đoạn cần thiết tại thời điểm khởi tạo. Sau đó, khi cần bộ phận quản lý bộ nhớ sẽ dựa vào PCT hoặc SCT của mỗi tiến trình để nạp các trang/đoạn tiếp theo.
- Nếu có một trang/đoạn của một tiến trình cần được nạp vào bộ nhớ trong tình trạng trên bộ nhớ không còn khung trang/phân đoạn trống thì bộ phận quản lý bộ nhớ sẽ đưa một trang/đoạn không cần thiết tại thời điểm hiện tại ra bộ bộ nhớ ngoài (swap-out), để lấy không gian nhớ trống đó nạp trang/đoạn vừa có yêu cầu. Trang/đoạn bị swap out sẽ được đưa vào tại thời điểm thích hợp hoặc cần thiết sau này (swap-in).

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

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

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

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

- Một lượng không gian bộ nhớ phụ (đĩa) cần thiết đủ để chứa các trang/đoạn bị swap out, không gian đĩa này được gọi là không gian swap.
- Có cơ chế để theo dõi các trang/đoạn của một tiến trình, của tất cả các tiến trình đang hoạt động trên bộ nhớ chính, là đang ở trên bộ nhớ chính hay ở trên bộ nhớ phụ. Trong trường hợp này hệ điều hành thường đưa thêm một bít trạng thái (bit present) vào các phần tử trong PCT hoặc SCT.
- Dựa vào các tiêu chuẩn cụ thể để chọn một trang nào đó trong số các trang đang ở trên bộ nhớ chính để swap out trong trường hợp cần thiết. Các hệ điều hành đã đưa ra các thuật toán cụ thể để phục vụ cho mục đích này.

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

- Hệ điều hành có thể nạp được nhiều tiến trình hơn vào bộ nhớ, trên bộ nhớ tồn tại các trang/đoạn của nhiều tiến trình khác nhau. Hệ thống khó có thể xả ra trường hợp không đủ bộ nhớ để nạp các tiến trình, vì bộ phận quản lý bộ nhớ không nạp tất cả tiến trình vào bộ nhớ và nếu cần có thể swap out các trang/đoạn của một tiến trình nào đó trên bộ nhớ. Lợi ích của việc nạp nhiều tiến trình vào bộ nhớ chúng ta đã biết trong chương Quản lý Tiến trình.
- Có thể nạp vào bộ nhớ một tiến trình có không gian địa chỉ lớn hơn tất cả không gian địa chỉ của bộ nhớ vật lý. Trong thực tế người lập trình có thể

thực hiện việc này mà không cần sự hỗ trợ của hệ điều hành và phần cứng bằng cách thiết kế chương trình theo cấu trúc Overlay, việc làm này là quá khó đối với người lập trình. Với kỹ thuật bộ nhớ ảo người lập trình không cần quan tâm đến kích thước của chương trình và kích thước của bộ nhớ tại thời điểm nạp chương trình, tất cả mọi việc này đều do hệ điều hành và phần cứng thực hiện.

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

Trước khi tìm hiểu về cơ chế cài đặt bộ nhớ ảo của hệ điều hành chúng hãy nhìn lại sự khác biệt giữa các kỹ thuật phân trang, phân đoạn với các kỹ thuật bộ nhớ ảo, thông qua bảng sau đây:

Phân trang đơn	**Phân đoạn đơn**	**Bộ nhớ ảo** (Page + Swap)	**Bộ nhớ ảo** (Segment + Swap)
Bộ nhớ chính được chia thành các phần nhỏ có kích thước cố định, được gọi là các khung trang.	Bộ nhớ chính không được phân vùng trước.	Bộ nhớ chính được chia thành các phần nhỏ có kích thước cố định, được gọi là các khung trang.	Bộ nhớ chính không được phân vùng trước.
Chương trình của người sử dụng được chia thành các trang bởi trình biên dịch hoặc hệ thống quản lý bộ nhớ.	Các đoạn của chương trình được chỉ ra bởi người lập trình và được gởi đến cho trình biên dịch.	Chương trình của người sử dụng được chia thành các trang bởi trình biên dịch hoặc hệ thống quản lý bộ nhớ.	Các đoạn của chương trình được chỉ ra bởi người lập trình và được gởi đến cho trình biên dịch.
Có thể xảy ra	Không xảy ra	Có thể xảy ra phân	Không xảy ra phân

phân mảnh nội vi trong phạm vi các frame. Không xảy ra phân mảnh ngoại vi.	phân mảnh nội vi, nhưng phân mảnh ngoại vi là có thể.	mảnh nội vi trong phạm vi các frame. Không xảy ra phân mảnh ngoại vi.	mảnh nội vi, nhưng phân mảnh ngoại vi là có thể.
Hệ điều hành phải duy trì một bảng trang cho mỗi tiến trình để theo dõi các trang của tiến trình trên bộ nhớ (được nạp vào các khung trang nào)	Hệ điều hành phải duy trì một bảng đoạn cho mỗi tiến trình để theo dõi các đoạn của tiến trình trên bộ nhớ (được nạp vào địa chỉ nào, và độ dài của đoạn)	Hệ điều hành phải duy trì một bảng trang cho mỗi tiến trình để theo dõi các trang của tiến trình trên bộ nhớ (được nạp vào các khung trang nào)	Hệ điều hành phải duy trì một bảng đoạn cho mỗi tiến trình để theo dõi các đoạn của tiến trình trên bộ nhớ (được nạp vào địa chỉ nào, và độ dài của đoạn)
Hệ điều hành phải duy trì một danh sách để theo dõi các khung trang còn trống trên bộ nhớ chính.	Hệ điều hành phải duy trì một danh sách để theo dõi các phần còn trống trên bộ nhớ chính.	Hệ điều hành phải duy trì một danh sách để theo dõi các khung trang còn trống trên bộ nhớ chính.	Hệ điều hành phải duy trì một danh sách để theo dõi các phần còn trống trên bộ nhớ chính.
Processor sử dụng (page number và offset) để tính địa chỉ tuyệt đối.	Processor sử dụng (segment number và offset) để tính địa chỉ tuyệt đối.	Processor sử dụng (page number và offset) để tính địa chỉ tuyệt đối.	Processor sử dụng (segment number và offset) để tính địa chỉ tuyệt đối.
Tất cả các trang của tiến trình phải được nạp vào bộ nhớ chính để chạy trừ khi khi sử dụng các kỹ thuật Overlay.	Tất cả các đoạn của tiến trình phải được nạp vào bộ nhớ chính để chạy trừ khi khi sử dụng các kỹ thuật Overlay.	Không phải nạp tất cả các trang của tiến trình vào các khung trang trên bộ nhớ chính khi tiến trình chay. Các trang có thể được đọc khi cần.	Không phải nạp tất cả các đoạn của tiến trình vào các khung trang trên bộ nhớ chính khi tiến trình chay. Các trang có thể được đọc khi cần.
		Đọc một trang vào	Đọc một trang vào

		bộ nhớ chính có thể cần phải đưa một trang ra đĩa.	bộ nhớ chính có thể cần phải đưa một hoặc đoạn ra đĩa.

Kỹ thuật bộ nhớ ảo

Theo trên thì kỹ thuật bộ nhớ ảo thực chất là kỹ thuật phân trang hoặc phân đoạn theo yêu cầu. Trong các phần trên chúng ta đã tìm hiểu các vấn đề cơ bản của 2 kỹ thuật phân trang đơn và phân đoạn đơn. Trong mục này chúng ta sẽ tìm hiểu lại kỹ hơn về 2 kỹ thuật này, trong bối cảnh của kỹ thuật bộ nhớ ảo.

Sự phân trang:

Trong kỹ thuật phân trang đơn, mỗi tiến trình sở hữu một bảng trang riêng, khi tất cả các trang của tiến trình được nạp vào bộ nhớ chính thì bảng trang của tiến trình được tạo ra và cũng được nạp vào bộ nhớ (nếu lớn), mỗi phần tử trong bảng trang chỉ chứa số hiệu của khung trang mà trang tương ứng được nạp vào. Trong kỹ thuật bộ nhớ ảo cũng vậy, nhưng một phần tử trong bảng trang sẽ chứa nhiều thông tin phức tạp hơn. Bởi vì trong kỹ thuật bộ nhớ ảo chỉ có một vài page của tiến trình được nạp vào bộ nhớ chính, do đó cần phải có một bít để cho biết một page tương ứng của tiến trình là có hay không trên bộ nhớ chính và một bít cho biết page có bị thay đổi hay không so với lần nạp gần đây nhất. Cụ thể là nó phải có thêm các bít điều khiển:

Virtual Address

Page Number	**Offset**

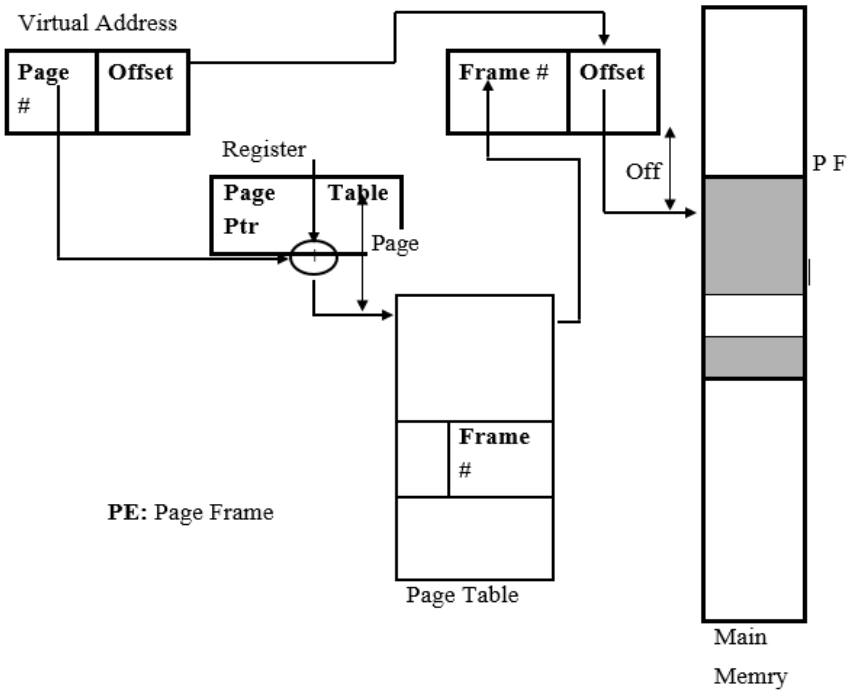
P	**M**	**Các bít điều khiển khác**	**Frame Number**

Hình 4.5. Một phần tử trong bảng Trang

- Bít P (Present): Cho biết trang tương ứng đang ở trên bộ nhớ chính (= 1) hay ở trên bộ nhớ phụ (= 0).
- Bít M (Modify): Cho biết nội dung của trang tương ứng có bị thay đổi hay không so với lần nạp gần đây nhất. Nếu nó không bị thay đổi thì việc phải ghi lại nội dung của một trang khi cần phải đưa một trang ra lại bộ nhớ ngoài là không cần thiết, điều này giúp tăng tốc độ trong các thao tác thay thế trang trong khung trang.
- Các bít điều khiển khác: Các bít này phục vụ cho các mục đích bảo vệ trang và chia sẻ các khung trang.

Chuyển đổi địa chỉ trong hệ thống phân trang:

Chương trình của người sử dụng sử dụng địa chỉ logic hoặc virtual gồm: page number và offset để truy xuất dữ liệu trên bộ nhớ chính. Bộ phận quản lý bộ nhớ phải chuyển địa chỉ virtual này thành địa chỉ vật lý tương ứng bao gồm: page number và offset. Để thực hiện việc này bộ phận quản lý bộ nhớ phải dựa vào bảng trang (PCT).

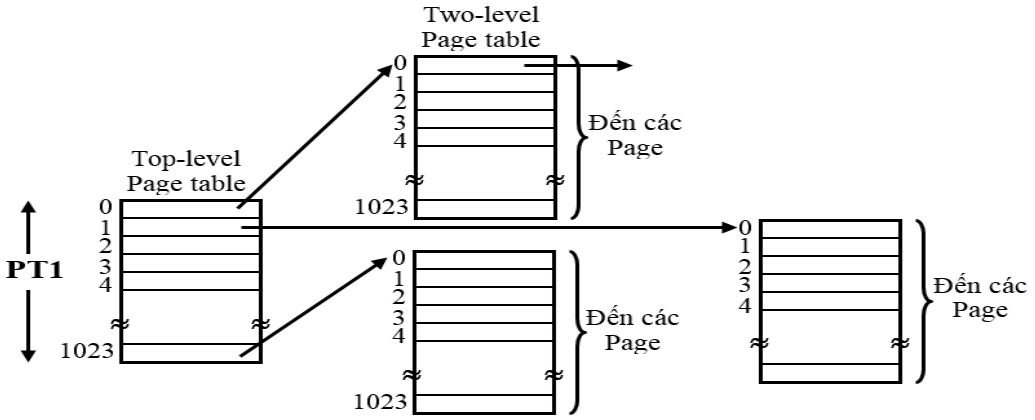


Hình 4.6. Sơ đồ chuyển địa chỉ trong hệ thống phân trang

Vì kích thước của PCT có thể lớn và thay đổi theo kích thước của tiến trình do đó trong kỹ thuật bộ nhớ ảo hệ điều hành thường chứa PCT trong bộ nhớ chính và dùng một thanh ghi để ghi địa chỉ bắt đầu của bộ nhớ nơi lưu trữ PCT của tiến trình khi tiến trình được nạp vào bộ nhớ chính để chạy.

Đa số các hệ điều hành đều thiết kế một bảng trang riêng cho mỗi tiến trình. Nhưng mỗi tiến trình có thể chiếm giữ một không gian lớn bộ nhớ ảo, trong trường hợp này bảng trang rất lớn và hệ thống phải tốn không gian bộ nhớ để chứa nó. Ví dụ, nếu một tiến trình có đến 2^{31} = 2GB bộ nhớ ảo, mỗi trang có kích thước 2^9 = 512 byte, thì tiến trình này phải cần đến 2^{22} phần tử trong bảng trang. Để khắc phục vấn đề này, trong các kỹ thuật bộ nhớ ảo hệ thống lưu trữ bảng trang trong bộ nhớ ảo chứ không lưu trữ trong bộ nhớ thực, và bản thân bảng trang cũng phải được phân trang. Khi tiến trình thực hiện, chỉ có một phần của bản trang được nạp vào bộ nhớ chính, đây là phần chứa các phần tử của các trang đang thực hiện tại thời điểm hiện tại.

Một số processor sử dụng lược đồ hai cấp (two-level) để tổ chức các bảng trang lớn, trong lược đồ này có một thư mục bảng trang (page directory) mà mỗi phần tử trong nó trỏ đến một bảng trang. Trong trường hợp này, nếu chiều dài của thư mục trang là X và chiều dài tối đa của một bảng trang là Y thì tiến trình có thể có X x Y trang. Chiều dài tối đa của một bảng trang chỉ bằng kích thước của một trang. Chúng ta sẽ xem ví dụ minh họa về bảng trang hai cấp sau đây:



Hình 4.7. Cấu trúc bảng trang 2 cấp

Giả sử có một không gian địa chỉ ảo 32 bít, được chia thành 3 trường: PT1 10 bít, PT2 10 bít và Offset 12 bít. Hình vẽ 3.10.c cho thấy cấu trúc của bảng trang 2 cấp tương ứng với không gian địa chỉ ảo 32 bít. Bên trái là top-level của bảng trang (bảng trang cấp 1), nó gồm có 1024 mục vào (tương ứng với 10 bít của PT1), tức là PT1 của địa chỉ ảo dùng để chỉ mục đến một phần tử trong bảng trang cấp 1. Mỗi mục vào dùng để mô tả 4Mb bộ nhớ, vì toàn bộ 4 GB (32 bit) không gian địa chỉ ảo được chia thành 1024 phần. Entry được chỉ mục trong bảng trang cấp 1 từ PT1 sẽ cho ra địa chỉ hoặc số hiệu khung trang của bản trang thứ hai (second-level). Có 1024 bảng trang cấp 2, đánh số từ 0 đến 1023, bảng trang cấp 2 thứ nhất (0) quản lý không gian nhớ 4Mb từ 0Mb đến 4Mb, bảng trang cấp 2 thứ hai (1) quản lý không gian nhớ 4Mb từ 8Mb,…. Trường PT2 bây giờ được dùng để chỉ mục đến bảng trang cấp 2 để tìm ra số hiệu khung trang của page tương ứng. Giá trị tìm được ở đây sẽ được kết hợp với thành phần Offset để có được địa chỉ vật lý của ô nhớ tương ứng với địa chỉ ảo 32 bit được phát sinh ban đầu.

Chúng ta xem lại ví dụ cụ thể sau đây: Có một địa chỉ ảo 32 bít: 0x00403004, đây là địa chỉ tương ứng với PT1 = 1, PT2 = 3 và Offset = 4. Bộ phận MMU sẽ chuyển địa chỉ này thành địa chỉ vật lý như sau: Đầu tiên MMU dùng PT1 để chỉ mục vào bảng trang cấp 1 và đó là entry 1, tương ứng với không gian đại chỉ từ 4Mb đến 8Mb. Sau đó MMU dùng PT2 để chỉ mục vào bảng trang cấp 2 và đó là entry 3, tương ứng với không gian địa chỉ 12292 đến 16383 trong phạm vi 4Mb. Đây là entry chứa số hiệu khung trang của page chứa địa chỉ ảo 0x00403004. Nếu page này có trong bộ nhớ, thì số hiệu khung trang có được từ bảng trang cấp hai sẽ được kết hợp với thành phần Offset để sinh ra địa chỉ vật lý. Địa chỉ này sẽ được đưa lên a_bus và gởi đến bộ nhớ.

Kích thước của trang

Kích thước của một trang do phần cứng quy định, đây là một trong những quyết định quan trọng trong việc thiết kế processor. Nếu kích thước của trang nhỏ thì sự phân mảnh bên trong sẽ nhỏ hơn, vì thế việc sử dụng bộ nhớ chính sẽ được hiệu quả hơn. Nhưng nếu kích thước trang nhỏ thì số lượng trang trên một tiến trình sẽ lớn hơn, bảng trang của tiến trình sẽ lớn, sẽ chiếm nhiều bộ nhớ hơn, và như thế việc sử dụng bộ nhớ chính sẽ kém hiệu quả hơn. Các vi xử lý họ Intel 486 và họ Motorola 68040 chọn kích thước của một trang là 4096 byte.

Ngoài ra kích thước của trang còn ảnh hưởng đến tỉ lệ xảy ra lỗi trang. Ví dụ: khi kích thước của trang là rất nhỏ thì sẽ có một lượng lớn các trang của tiến trình trên bộ nhớ chính, sau một thời gian thì tất cả các trang của bộ nhớ sẽ chứa các tiến trình được tham chiếu gần đây, vì thế tốc độ xảy ra lỗi trang được giảm xuống.

Sự phân đoạn

Sự phân đoạn cho phép người lập trình xem bộ nhớ như bao gồm một tập các không gian nhớ hoặc các đoạn (segment) có địa chỉ được xác định. Với bộ nhớ ảo người lập trình không cần quan tâm đến giới hạn bộ nhớ được đưa ra bởi bộ nhớ chính. Các segment có thể có kích thước không bằng nhau và được ấn định một cách động. Địa chỉ tham chiếu bộ nhớ trong trường hợp này bao gồm: Segment Number và Offset.

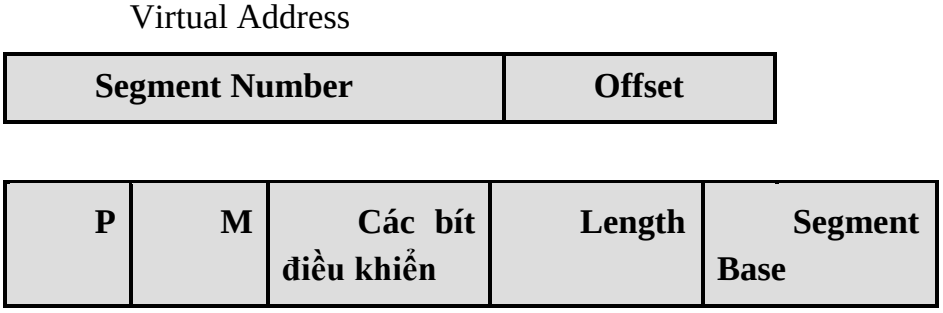
Đối với người lập trình thì sự phân đoạn không gian địa chỉ có một số thuận lợi sau đây so với trường hợp không phân đoạn không gian địa chỉ:

1. Nó đơn giản để điều khiển các cấu trúc dữ liệu lớn dần (growing) trong quá trình hoạt động của hệ thống. Nếu người lập trình không biết trước dữ liệu sẽ lớn đến chừng nào tại thời điểm chạy thì việc ấn định kích thước của động cho segment mang lại nhiều thuận lợi cho người lập trình.
2. Nó cho phép các chương trình không phụ thuộc vào sự thay đổi vào sự biên dịch lại. Nó không yêu cầu thiết lập lại toàn bộ chương trình khi chương trình được liên kết hoặc được nạp trở lại. Việc này chỉ có thể thực hiện bằng cách sử dụng nhiều phân đoạn (Multiple Segment).
3. Nó thích hợp với chiến lược chia sẻ segment giữa các tiến trình. Người lập trình có thể đặt một chương trình tiện ích hoặc một bảng dữ liệu thường sử dụng vào một segment mà có thể được tham chiếu bởi nhiều tiến trình khác nhau.
4. Nó thích hợp với chiến lược bảo vệ bộ nhớ. Bởi vì một segment có thể được sinh ra để chứa một tập xác định các thủ tục hoặc dữ liệu, sau đó người lập trình hoặc người quản trị hệ thống có thể gán quyền truy cập với các độ ưu tiên thích hợp nào đó.

Tổ chức của hệ thống phân đoạn

Trong kỹ thuật phân đoạn đơn, mỗi tiến trình sở hữu một bảng đoạn riêng, khi tất cả các đoạn của tiến trình được nạp vào bộ nhớ chính thì bảng đoạn của tiến trình được tạo ra và cũng được nạp vào bộ nhớ, mỗi phần tử trong bảng đoạn chứa địa chỉ bắt đầu của đoạn tương ứng trong bộ nhớ chính và độ dài của đoạn. Trong kỹ thuật bộ nhớ ảo cũng vậy, nhưng một phần tử trong bảng đoạn sẽ chứa nhiều thông tin phức tạp hơn. Bởi vì trong kỹ thuật bộ nhớ ảo chỉ có một vài segment của tiến trình được nạp vào bộ nhớ chính, do đó cần phải có một bít để cho biết một đoạn tương ứng của tiến trình là có hay không trên bộ nhớ chính và một bít cho biết đoạn có bị thay đổi hay không so với lần nạp gần đây nhất. Cụ thể là nó phải có thêm các bít điều khiển:

- Bít M (Modify): Cho biết nội dung của đoạn tương ứng có bị thay đổi hay không so với lần nạp gần đây nhất. Nếu nó không bị thay đổi thì việc phải ghi lại nội dung của một đoạn khi cần phải đưa một đoạn ra lại bộ nhớ ngoài là không cần thiết, điều này giúp tăng tốc độ trong các thao tác thay thế đoạn.



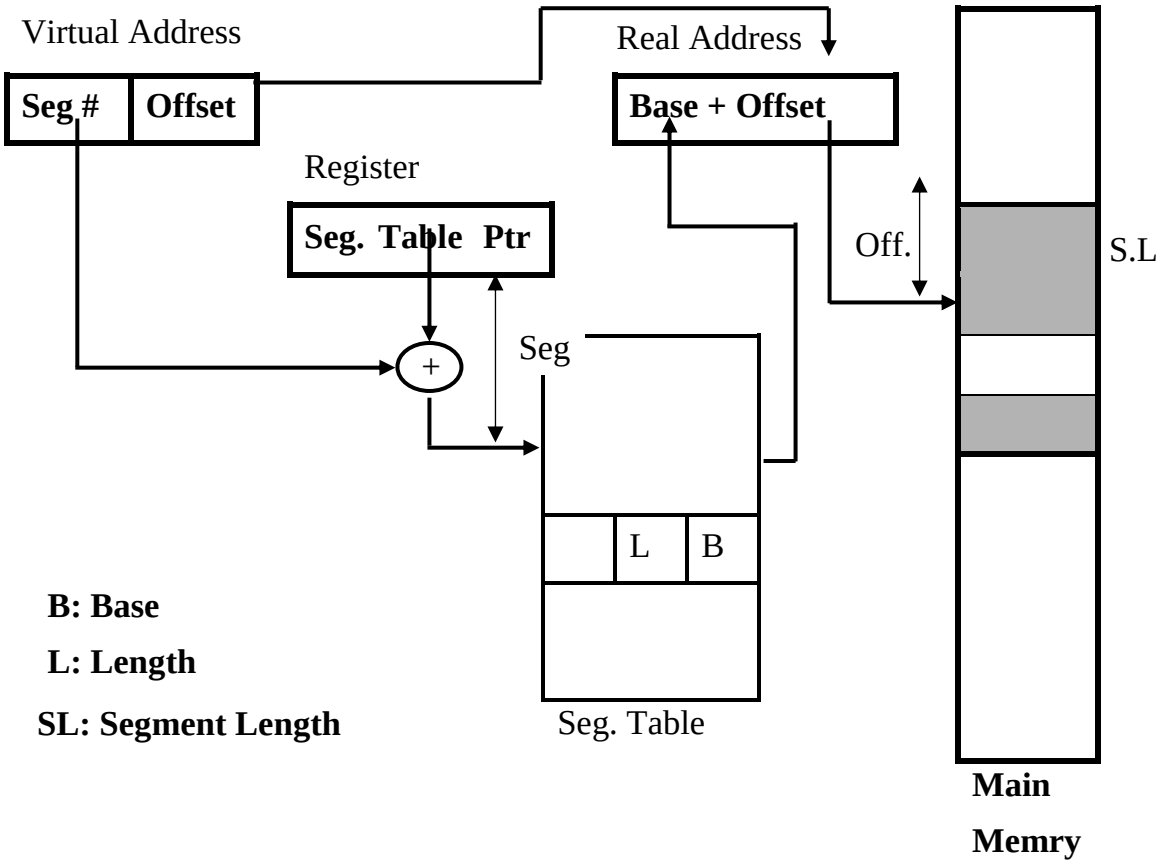
Hình 4.8. Một phần tử trong bảng Đoạn

- Bít P (Present): Cho biết đoạn tương ứng đang ở trên bộ nhớ chính (= 1) hay ở trên bộ nhớ phụ (= 0).
- Các bít điều khiển khác: Các bít này phục vụ cho các mục đích bảo vệ trang và chia sẻ các khung trang.

Chuyển đổi địa chỉ trong hệ thống phân đoạn

Chương trình của người sử dụng sử dụng địa chỉ logic hoặc virtual gồm: segment number và offset để truy xuất dữ liệu trên bộ nhớ chính. Bộ phận quản lý bộ nhớ phải chuyển địa chỉ virtual này thành địa chỉ vật lý tương ứng bao gồm: segment number và offset. Để thực hiện việc này bộ phận quản lý bộ nhớ phải dựa vào bảng đoạn (SCT). Vì kích thước của SCT có thể lớn và thay đổi theo kích thước của tiến trình do đó trong kỹ thuật bộ nhớ ảo hệ điều hành thường chứa SCT trong bộ nhớ chính và dùng một thanh ghi để ghi địa chỉ bắt đầu của bộ nhớ nơi lưu trữ SCT của tiến trình khi tiến trình được nạp vào bộ nhớ chính để chạy. Thành phần segment

number của địa chỉ ảo được dùng để chỉ mục đến bảng đoạn và tìm địa chỉ bắt đầu của segment tương ứng trong bộ nhớ chính. Giá trị này sẽ được cộng với thành phần Offset có trong địa chỉ ảo để có được địa chỉ vật lý thực cần tìm.



Hình 4.9. Sơ đồ chuyển địa chỉ trong hệ thống phân Đoạn

Bảo vệ và chia sẻ trong phân đoạn:

Sự phân đoạn dùng chính nó để cài đặt các chính sách bảo vệ và chia sẻ bộ nhớ. Bởi vì mỗi phần tử trong bảng trang bao gồm một trường length và một trường base address, nên một tiến trình trong segment không thể truy cập đến một vị trí trong bộ nhớ chính mà vị trí này vượt qua giới hạn (length) của segment, ngoại trừ đó là một truy cập dữ liệu đến một segment dữ liệu nào đó.

Để thực hiện việc chia sẻ segment, ví dụ segment A, cho nhiều tiến trình, hệ điều hành cho phép nhiều tiến trình cùng tham chiếu đến segment A, khi đó các thông số (length và base address) của segment A xuất hiện đồng thời ở các bảng segment của các tiến trình cùng chia sẻ segment A. Chiến lược chia sẻ này cũng được áp dụng trong hệ thống phân trang.

Các hệ điều hành cũng có thể sử dụng một chiến lược bảo vệ phức tạp hơn để cài đặt sự bảo vệ các segment, đó là sử dụng cấu trúc vòng bảo vệ (ring protection).

Như đã biết trong hệ thống ring, bao gồm: ring 0, ring 1, ring 2, … thì mỗi ring có một mức đặc quyền truy cập riêng, ring 0 có mức đặc quyền truy cập cao hơn so với ring 1, ring 1 có mức đặc quyền truy cập cao hơn so với ring 2, …, ring thấp nhất được sử dụng cho thành phần kernel của hệ điều hành, các ring cao hơn được sử dụng cho các ứng dụng của người sử dụng. Nguyên lý cơ bản của hệ thống ring là:

- Chương trình chỉ có thể truy cập đến dữ liệu trong cùng một ring hoặc dữ liệu ở ring có mức đặc quyền truy cập thấp hơn.
- Chương trình có thể gọi các dịch vụ trong cùng một ring hoặc ở các ring có mức đặc quyền truy cập cao hơn.

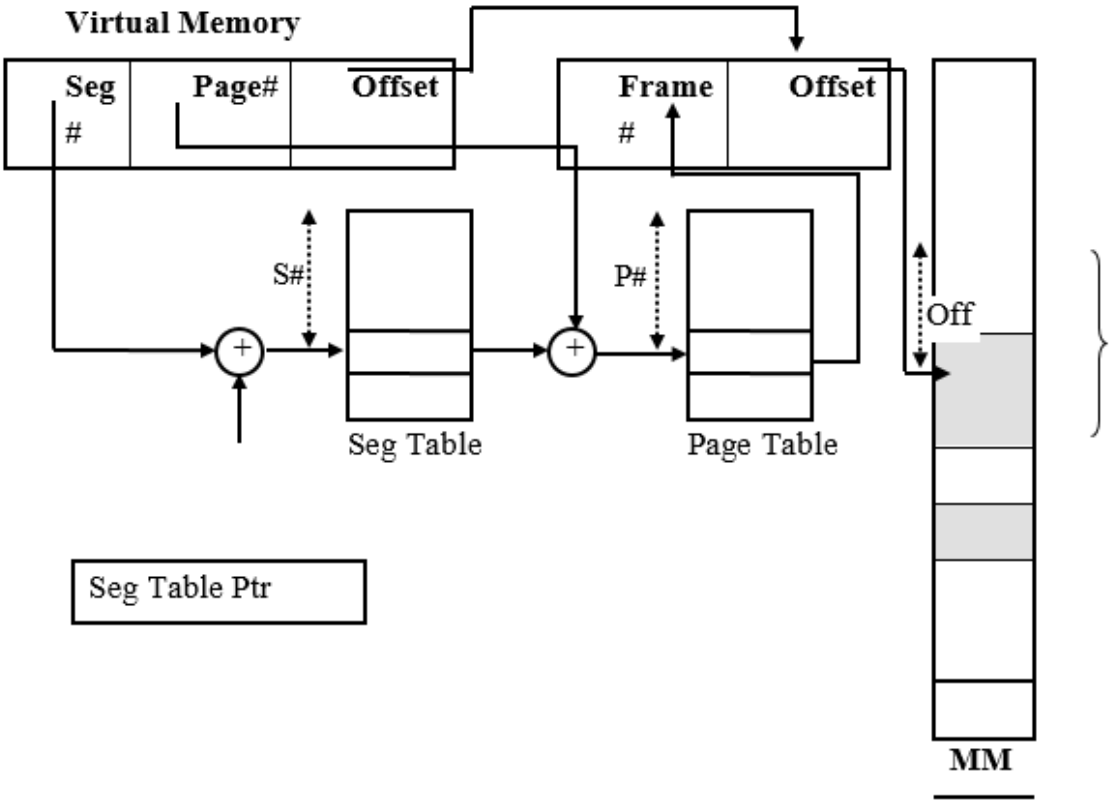
Kết hợp phân trang với phân đoạn:

Cả hai kỹ thuật phân trang và phân đoạn đều có những thế mạnh của nó. Sự phân trang, là trong suốt (transparent) đối với người lập trình, loại bỏ được hiện tượng phân mảnh nội vi. Sự phân đoạn, là thấy được đối với người lập trình, có khả năng điều khiển các cấu trúc dữ liệu lớn dần và hỗ trợ chia sẻ và bảo vệ bộ nhớ. Để kết hợp những thuận lợi của cả hai hệ thống phân trang và phân đoạn, một số hệ thống được trang bị sự hỗ trợ của cả phần cứng processor và phần mềm hệ điều hành để cài đặt kết hợp cả hai kỹ thuật phân trang và phân đoạn.

Trong các hệ thống kết hợp phân trang và phân đoạn, không gian địa chỉ bộ nhớ của người sử dụng được chia thành các đoạn theo ý muốn của người lập trình, sau đó mỗi đoạn lại được chia thành các trang có kích thước cố định bằng nhau. Theo cách nhìn của người lập trình thì địa chỉ logic bao gồm một segment number và một segment offset. Theo cách nhìn của hệ thống thì segment offset được xem như một page number và page offser cho một trang trong phạm vị một segment được chỉ ra.

Trong hệ thống phân trang kết hợp phân đoạn nay, hệ điều hành thiết kế cả bảng trang và bảng đoạn. Hệ điều hành kết hợp với mỗi tiến trình có một bảng đoạn và nhiều bảng trang, mỗi phần tử trong bảng đoạn chỉ đến một bảng trang, bảng trang này quản lý các trang của đoạn tương ứng. Khi một tiến trình riêng biệt chạy, một thanh ghi giữ địa chỉ bắt đầu của bảng đoạn của tiến trình đó. Trong hệ thống này địa chỉ ảo do processor đua ra phải gồm 3 thành phần: Segment Number, Page Number và Offset Segment number chỉ vào bảng đoạn tiến trình để tìm bảng trang của segment đó. Sau đó page number được sử dụng để chỉ mục đến bảng trang và tìm số hiệu khung trang tương ứng, giá trị này sẽ được kết hợp với thành phần Offset trong địa chỉ ảo để có được địa chỉ vật lý thực mong muốn.

Sơ đồ chuyển đổi địa chỉ trong hệ thống phân trang kết hợp phân đoạn:



Hình 4.10. Sơ đồ chuyển địa chỉ trong hệ thống Trang - Đoạn

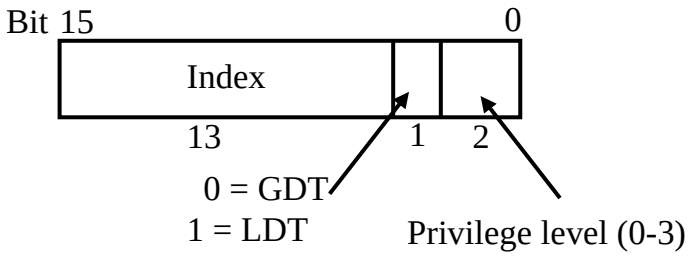
Phân đoạn với phân trang trong Intel 386:

Trong chế độ bảo vệ của 80286 và trong chế độ ảo của 80386 không gian bộ nhớ của hệ thống được chia thành hai loại: không gian bộ nhớ toàn cục và không gian bộ nhớ cục bộ. Không gian nhớ toàn cục được dành cho dữ liệu hệ thống và các tiến trình của hệ điều hành. Mọi chương trình của người sử dụng đều có thể truy cập dữ liệu và các tiến trình ở không gian nhớ toàn cục này. Không gian nhớ cục bộ được dành riêng cho các tiến trình, các tác vụ riêng biệt. Vì vậy, các đoạn mã lệnh và dữ liệu của một tiến trình, một tác vụ nằm trong không gian nhớ cục bộ sẽ được bảo vệ tránh sự truy xuất bất hợp lệ của các tiến trình, các tác vụ khác trong hệ thống.

Trong kỹ thuật bộ nhớ ảo Intel 80386 sử dụng 2 bảng mô tả: Bảng mô tả cục bộ (LDT: Local Descriptor Table), để theo dõi không gian nhớ cục bộ và bảng mô tả toàn cục (GDT: Global Descriptor Table), để theo dõi không gian nhớ toàn cục. Mỗi chương trình sở hữu một LDT riêng, nhưng có một GDT được chia sẻ cho tất cả các chương trình trên hệ thống. LDT mô tả các segment cục bộ cho mỗi chương trình, bao gồm code, data, stack, …, trong khi đó GDT mô tả các segment hệ thống, của chính hệ điều hành. Các LDT, GDT được nạp vào bộ nhớ trong quá trình hoạt động của hệ thống, Intel 80386 dùng thanh ghi GDTR để ghi địa chỉ cơ sở và giới hạn kích thước của GDT và thanh ghi LDTR để ghi địa chỉ cơ sở và giới hạn kích thước của

LDT của tác vụ hiện tại.

Để truy cập một segment, đầu tiên một chương trình chạy trên Intel 386 phải nạp một selector của segment đó vào 1 trong 6 thanh ghi đoạn của Intel 386. Trong quá trình thực hiện chương trình thanh ghi CS giữ selector cho code segment và thanh ghi DS giữ selector cho data segment. Mỗi selector dài 16 bít và được mô tả như sau:

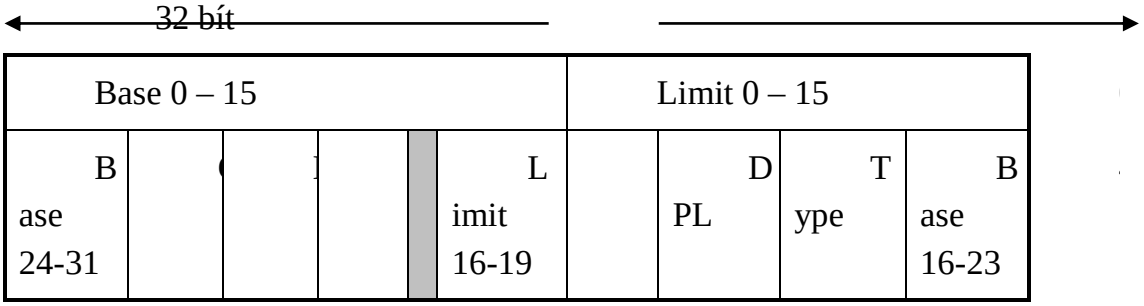


Hình 4.11. Một Selector (bộ chọn đoạn) Intel 386

Trong đó:

- Hai bít đầu tiên cho biết mức đặc quyền truy cập của bộ chọn đoạn, các bít này phục vụ cho công tác bảo vệ bộ nhớ (segment).
- Một bít tiếp theo cho biết segment là cục bộ hay toàn cục.
- Mười ba bít còn lại chỉ đến mục vào (entry) trong LDT hoặc GDT, vì thế mỗi bảng mô tả (Descriptor Table) chỉ lưu giữ được 8k (2^{13}) các bộ mô tả đoạn (segment descriptor). Tức là LDT/GDT có 2^{13} mục vào/ phần tử.

Tại thời điểm một selector được nạp vào một thanh ghi segment, một descriptor tương ứng được nhận từ bảng LDT hoặc GDT và được lưu trữ trong các thanh ghi microprogram, do đó có thể được truy cập nhanh. Một descriptor gồm có 8 byte, gồm có địa chỉ, kích thước, và các thông tin khác của segment. Hình sau đây mô tả một descriptor trong Intel 386:



Hình 4.12. Một descriptor Code segment (bộ mô tả đoạn code) Intel 386

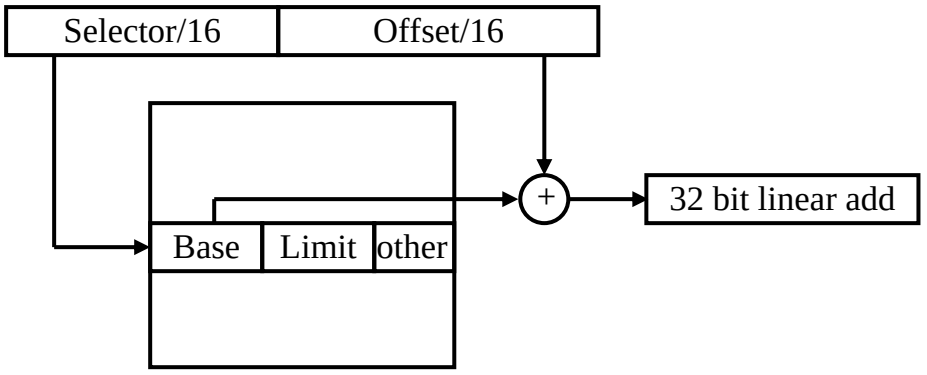
Trong đó:

- Base (24 bit): cho biết vị trí đầu tiên của segment trong không gian địa chỉ tuyến tính 4GB. Bộ xử lý ghép 3 trường địa chỉ cơ sở thành một giá trị địa

chỉ 32 bít duy nhất. Trong thực tế trường Base cho phép mỗi segment bắt đầu tại một vị trí bất kỳ trong không gian địa chỉ tuyến tính 32 bit.

- Limit (20 bit): cho biết kích thước của segment. Bộ xử lý ghép hai trường kích thước thành một giá trị 20 bít. Bộ xử lý tính kích thước theo hai cách dựa vào giá trị của cờ G: G = 0: kích thước đoạn nằm giữa 1B và 1MB, tính theo đơn vị byte. G = 1: kích thước đoạn nằm giữa 4KB và 4GB, tính theo đơn vị 4Kbyte (= 2^{12} = 1page). Như vậy với 20 bít limit thì một segment có thể có kích thước lên đến 2^{32} byte ($2^{12}x2^{20}$).
- Type (5 bit): định nghĩa dạng của đoạn và kiểu truy cập đoạn.
- DPL: Descriptor Privilege Level (2 bit): cho biết mức đặc quyền truy cập của mô tả segment (có 4 mức đặc quyền truy cập: 0-3).
- P: Present (1 bit): cho biết segment này đã được nạp vào bộ nhớ chính (P = 1) hay chưa được nạp vào bộ nhớ chính (P = 0).
- G: Granularity (1 bit): định nghĩa hằng số để nhân với trường kích thước. G = 0: kích thước tính theo đơn vị 1byte. G = 1: kích thước tính theo đơn vị 1page (Một page của Intel 386 có kích thước cố định là 4Kbyte).
- D: Default Operation Sizze (1 bit): cho biết chiều dài của dòng lệnh. D = 1: vi xử lý mặc định 32 bít địa chỉ, 32/8 bít mã lệnh. D = 0: vi xử lý mặc định 16 bít địa chỉ, 32/8 bít mã lệnh.

Sau đây là sơ đồ chuyển địa chỉ gồm 2 thành phần selector và offset thành địa chỉ tuyến tính (linear address) dựa vào bảng mô tả đoạn.



Hình 4.13. Chuyển địa chỉ logic (selector:offset) thành địa chỉ tuyến tính

Nếu sự phân trang (paging) bị cấm thì địa chỉ tuyến tính được biên dịch thành địa chỉ vật lý và gửi đến bộ nhớ để truy xuất dữ liệu. Như vậy khi sự phân trang bị cấm thì trong trườnng hợp này hệ thống chỉ sử dụng sự phân đoạn (segmentation) đơn

thuần, với địa chỉ cơ sở (base address) của segment được cho trong descriptor của nó. Nếu sự phân trang là được phép thì địa chỉ tuyến tính sẽ được biên dịch thành địa chỉ ảo và được ánh xạ thành địa chỉ vật lý bằng cách sử dụng các bảng trang.

Mỗi chương trình có một danh mục bảng trang (page directory) riêng, bao gồm 1024 entry 32 bit, nó được nạp vào bộ nhớ được chỉ bởi một thanh ghi global, mỗi entry trong danh mục bảng trang chỉ đếm một bảng trang (page table), bảng trang cũng chứa 1024 entry 32 bit, một mục vào trong bảng trang lại chỉ đến một khung trang (page frame).

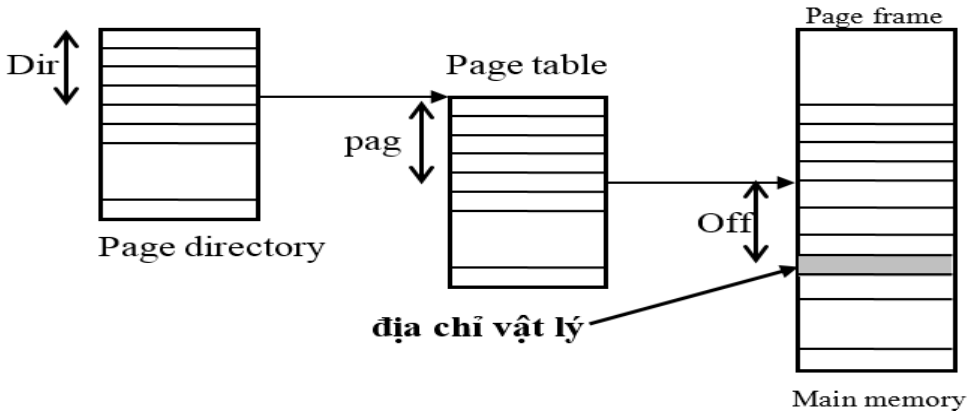
Địa chỉ tuyến tính gồm 3 trường: Dir, Page, Offset. Trường Dir: được sử dụng để chỉ mục vào Page Directory để tìm đến một con trỏ trỏ tới Page Table. Trường Page: được sử dụng để chỉ mục vào Page Table để tìm địa chỉ vật lý của Page Frame. Trường Offset được cộng với địa chỉ vật lý của Page Framre để có được địa chỉ vật lý của ô nhớ chứa dữ liệu cần truy xuất.

Dir: 10 bít	Page: 10 bít	Offset: 12 bít

Hình 4.14. Địa chỉ tuyến tính 32 bít trong Intel 386

Mỗi entry trong page table dài 32 bít, 20 bít chứa số hiệu của page frame, các bít còn lại là các bít truy cập, được thiết lập bởi phần cứng cho các lợi ích của hệ điều hành các bít bảo vệ và các bít tiện ích khác. Mỗi page table có 1024 entry cho các page frame, mỗi page frame có kích thước là 4Kb, nên một page table đơn quản lý được 4Mb bộ nhớ.

Từ địa chỉ tuyến tính ở trên hệ thống sẽ ánh xạ thành địa chỉ vật lý, dựa vào page directorry, page table và page frame. Sau đây là sơ đồ, đơn giản, minh họa sự ánh xạ địa chỉ tuyến tính thành địa chỉ vật lý:



Hình 4.15. Ánh xạ địa chỉ tuyến tính thành địa chỉ vật lý

Trên đây chúng ta đã tìm hiểu về cơ chế bộ nhớ ảo trong Intel 386, bây giờ

chúng ta sẽ tìm hiểu về sự bảo vệ trong cơ chế bộ nhớ ảo của nó.

Công cụ mà 80386 đưa ra để thực hiện nhiệm vụ bảo vệ không gian nhớ chứa các tiến trình và chứa chính hệ điều hành trên bộ nhớ chính là: các mức/ cấp (level) đặc quyền truy cập hay mức ưu tiên được yêu cầu (RPL: Request Privilege Level). Từ vi xử lý 80286 các vi xử lý đã đưa ra 4 mức ưu tiên từ 0 đến 3, *được ghi tại trường Privilege Level của thành phần địa chỉ selector (hình 3.13.a)*. Mức 0 có độ ưu tiên cao nhất, mức 3 có độ ưu tiên thấp nhất. Các segment trên bộ nhớ cũng được gán một mức ưu tiên tương tự, *được ghi tại trường DPL trong bộ mô tả đoạn trong bảng mô tả đoạn (hình 3.13.b)*.

Các ứng dụng có mức ưu tiên cao hơn sẽ được quyền truy cập mã lệnh, dữ liệu tại các đoạn nhớ có mức ưu tiên thấp hơn. Các ứng dụng có mức ưu tiên thấp hơn sẽ không được truy cập mã lệnh, dữ liệu tại các đoạn nhớ có mức ưu tiên cao hơn, trong thực tế thì điều này cũng có thể nếu các ứng dụng biết cách vượt qua các Cổng (Cổng và nguyên tắc hoạt động của Cổng các bạn có thể tìm đọc ở một tài liệu nào đó viết về các vi xử lý của họ Intel). Bốn mức ưu tiên mà 80386 đưa ra là:

- Mức 0: là mức của thành phần kernel của hệ điều hành. Kernel của hệ điều hành được nạp tại segment có mức đặc quyền truy cập là 0.
- Mức 1: là mức của phần mềm hệ thống quản lý thiết bị và cổng phần cứng. Segment nhớ được gán mức này chứa các chương trình hệ thống của BIOS và DOS/ Windows.
- Mức 2: chứa các thủ tục thư viện, có thể chia sẻ cho nhiều chương trình đang chạy. Chương trình của người sử dụng có thể gọi các các thủ tục và đọc dữ liệu ở mức này nhưng không thể modify nó.
- Mức 3: chương trình của người sử dụng chạy tại mức này, đây là mức có độ ưu tiên thấp nhất.

Khi chương trình cần truy xuất vào một đoạn nhớ nào đó trên bộ nhớ thì vi xử lý sẽ dựa vào giá trị mức ưu tiên tại RPL và DPL để quyết định có cho phép chương trình truy xuất vào đoạn nhớ hay không. Trong trường hợp này Intel 80386 công nhận ba mức ưu tiên sau đây:

- Mức ưu tiên hiện tại CPL (Current Privilege Level): là mức ưu tiên của chương trình hay tác vụ đang chạy. CPL được lưu trữ tại bit 0 và bit 1 của các thanh ghi đoạn CS và SS. Thông thường giá trị của CPL bằng giá trị mức ưu tiên của đoạn mã lệnh chứa chương trình đang chạy. Giá trị của CPL có thể bị thay đổi nếu điều khiển chương trình được chuyển đến một đoạn mã lệnh có độ ưu tiên khác.
- Mức ưu tiên bộ mô tả DPL (Descriptor Privilege Level): là mức ưu tiên

của một đoạn. DPL được lưu trữ tại trường DPL của các bộ mô tả đoạn. Nếu chương trình đang chạy tìm cách truy cập một đoạn nhớ, vi xử lý sẽ so sánh DPL với CPL và RPL của bộ chọn đoạn.

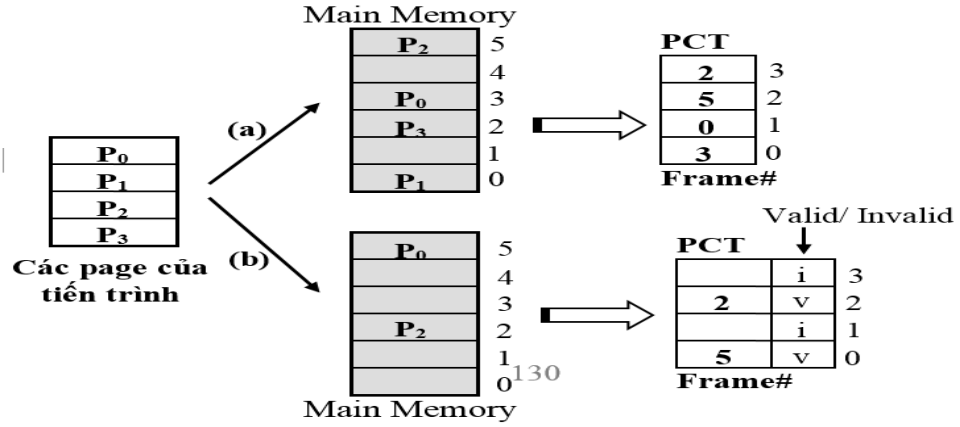
- Mức ưu tiên được yêu cầu RPL (Peqest Privilege Level): là giá trị được ghi tại bít 0 và bít 1 trong bộ chọn đoạn. Vi xử lý sẽ so sánh RPL với CPL để kiểm tra quyền truy cập vào một đoạn. Khi CPL có giá trị cho phép chương trình truy cập một đoạn, nhưng giá trị trong RPL không có mức ưu tiên tương ứng thì chương trình cũng không được phép tuy cập đoạn. Điều này có nghĩa là nếu RPL của bộ chọn đoạn có giá trị lớn hơn CPL, thì RPL sẽ ghi chồng lên CPL.

Khi một chương trình có yêu cầu truy cập vào một đoạn nhớ nào đó, thì bộ phận quản lý bộ nhớ của vi xử lý sẽ so sánh mức ưu tiên ghi trong RPL với mức ưu tiên của đoạn được ghi trong DPL, nếu RPL lớn hoăn hoặc bằng DPL thì vi xử lý cho phép chương trình truy cập vùng nhớ mà nó yêu cầu, nếu ngược lại thì vi xử lý không trao quyền truy cập vùng nhớ cho chương trình, đồng thời phát ra một ngắt thông báo có sự vi phạm quyền truy cập vùng nhớ. Đây chính là cơ chế bảo vệ bộ nhớ của Intel 80386.

Bộ nhớ ảo và lỗi trang (page fault)

Trước khi tiếp tục tìm hiểu về sự cài đặt bộ nhớ ảo và hiện tượng lỗi trang trong bộ nhớ ảo, chúng ta xem lại sơ đồ sau, hình 3.14, để thấy được sự khác nhau giữa kỹ thuật phân trang đơn và kỹ thuật bộ nhớ ảo theo kiểu phân trang.

Sơ đồ này cho thấy: có một tiến trình gồm có 4 trang P_0, P_1, P_2, P_3 cần được nạp vào bộ nhớ chính, không gian địa chỉ bộ nhớ chính gồm có 6 phân trang (paging), còn được gọi là khung trang (frame), còn trống:



Hình 4.16. Sự khác nhau giữa phân trang đơn và bộ nhớ ảo phân trang. a: nạp tất cả các page của tiến trình vào bộ nhớ. b: chỉ nạp page 0 và page 2 vào bộ nhớ.

- Trường hợp a, là trường hợp phân trang đơn: trong trường hợp này tất cả 4 page của tiến trình đều được nạp vào bộ nhớ. Rõ ràng sẽ là lãng phí bộ nhớ nếu biết rằng tiến trình này chỉ cần nạp vào bộ nhớ 2 trang P_0, P_2 là tiến trình có thể khởi tạo và bắt đầu hoạt động được. Và trong trường hợp này nếu bộ nhớ chỉ còn 3 frame còn trống thì tiến trình cũng sẽ không nạp vào bộ nhớ được. PCT trong trường hợp này cho biết các page của tiến trình được nạp vào các frame trên bộ nhớ chính.

- Trường hợp b, là trường hợp bộ nhớ ảo sử dụng kỹ thuật phân trang: trong trường hợp này hệ điều hành không nạp tất cả các page của tiến trình vào bộ nhớ mà chỉ nạp 2 page cần thiết ban đầu để tiến trình có thể khởi tạo và bắt đầu hoạt động được, mặc dầu trên bộ nhớ chính còn một vài frame còn trống. Rõ ràng trong trường hợp này hệ điều hành đã tiết kiệm được không gian bộ nhớ chính và nhớ đó mà hệ điều hành có thể nạp vào bộ nhớ nhiều tiến trình hơn và cho phép các tiến trình này hoạt động đồng thời với nhau. Các page của tiến trình chưa được nạp vào bộ nhớ sẽ được lưu trữ tại một không gian đặc biệt trên đĩa (thường là trên HDD), không gian đĩa này được gọi là không gian bộ nhớ ảo, một cách chính xác thì không gian bộ nhớ ảo này chứa tất cả các page của một tiến trình. Như vậy PCT trong trường hợp này phải có thêm một trường mới, trường này thường được gọi là trường Present. Trường Present chỉ cần một bít, nó cho biết page tương ứng là đã được nạp vào bộ nhớ chính (= 1), hay còn nằm trên đĩa (= 0).

Trong mô hình bộ nhớ ảo khi cần truy xuất đến một page của tiến trình thì trước hết hệ thống phải kiểm tra bít present tại phần tử tương ứng với page cần truy xuất trong PCT, để biết được page cần truy xuất đã được nạp vào bộ nhớ hay chưa. Trường hợp hệ thống cần truy xuất đến một page của tiến trình mà page đó đã được nạp vào bộ nhớ chính, được gọi là truy xuất hợp lệ (v: valid). Trường hợp hệ thống cần truy xuất đến một page của tiến trình mà page đó chưa được nạp vào bộ nhớ chính, được gọi là truy xuất bất hợp lệ (i: invalid). Khi hệ thống truy xuất đến một trang của tiến trình mà trang đó không thuộc phạm vi không gian địa chỉ của tiến trình cũng được gọi là truy xuất bất hợp lệ.

Khi hệ thống truy xuất đến một page được đánh dấu là bất hợp lệ thì sẽ phát sinh một lỗi trang. Như vậy lỗi trang là hiện tượng hệ thông cần truy xuất đến một page của tiến trình mà trang này chưa được nạp vào bộ nhớ, hay không thuộc không gian địa chỉ của tiến trình. Ở đây ta chỉ xét lỗi trang của trường hợp: Page cần truy xuất chưa được nạp vào bộ nhớ chính.

Khi nhận được tín hiệu lỗi trang, hệ điều hành phải tạm dừng tiến trình hiện tại để tiến hành việc xử lý lỗi trang. Khi xử lý lỗi trang hệ điều hành có thể gặp một trong hai tình huống sau:

- Hệ thống còn frame trống (a): Hệ điều hành sẽ thực hiện các bước sau:

1. Tìm vị trí của page cần truy xuất trên đĩa.
2. Nạp page vừa tìm thấy vào bộ nhớ chính.
3. Cập nhật lại bảng trang (PCT) tiến trình.
4. Tái kích hoạt tiến trình để tiến trình tiếp tục hoạt động.

- Hệ thống không còn frame trống (b):
  1. Tìm vị trí của page cần truy xuất trên đĩa.
  2. Tìm một page không hoạt động hoặc không thực sự cần thiết tại thời điểm hiện tại để swap out nó ra đĩa, lấy frame trống đó để nạp page mà hệ thống vừa cần truy xuất. Page bị swap out sẽ được hệ điều hành swap in trở lại bộ nhớ tại một thời điểm thích hợp sau này.
  3. Cập nhật PCT của tiến trình có page vừa bị swap out.
  4. Nạp trang vừa tìm thấy ở trên (bước 1) vào frame trống ở trên (bước 2).
  5. Cập nhật lại bảng trang (PCT) của tiến trình.
  6. Tái kích hoạt tiến trình để tiến trình tiếp tục hoạt động.

Xử lý lỗi trang là một trong những nhiệm vụ quan trọng và phức tạp của hệ thống và hệ điều hành. Để xử lý lỗi trang hệ thống phải tạm dừng các thao tác hiện tại, trong trường hợp này hệ thống phải lưu lại các thông tin cần thiết như: con trỏ lệnh, nội dung của các thanh ghi, các không gian địa chỉ bộ nhớ, …, các thông tin này là cơ sở để hệ thống tái kích hoạt tiến trình bị tạm dừng trước đó khi nó đã hoàn thành việc xử lý lỗi trang.

Khi xử lý lỗi trang, trong trường hợp hệ thống không còn frame trống hệ điều hành phải chú ý đến các vấn đề sau:

- **Nên chọn page nào trong số các page trên bộ nhớ chính để swap out:** Về vấn đề này chúng ta đã biết hệ điều hành sẽ áp dụng một thuật toán thay page cụ thể nào đó, nhưng ở đây cần chú ý thêm rằng đối tượng của các thuật toán thay page là chỉ các page của tiến trình xảy ra lỗi page, hay tất cả các page của các tiến trình đang có trên bộ nhớ chính. Tức là, nên chọn page của tiến trình xảy ra lỗi trang để thay thế (*thay thế cục bộ*), hay chọn một page của tiến trình khác để thay thế (*thay thế toàn cục*). Nếu chọn page của tiến trình xảy ra lỗi trang thì sẽ đơn giản hơn với hệ điều hành và không ảnh hưởng đến các tiến trình khác, nhưng cách này có thể làm cho tiến trình hiện tại lại tiếp tục xảy ra lỗi trang ngay sau khi hệ điều hành vừa xử lý lỗi trang cho nó, vì page mà hệ điều hành vừa chọn để đưa ra (swap out) lại là page cần truy xuất ở thời điểm tiếp theo. Nếu chọn page của tiến trình khác thì tiến trình hiện tại sẽ ít có nguy cơ xảy ra lỗi trang ngay sau đó hơn, nhưng cách này sẽ phức tạp hơn cho hệ điều hành, vì hệ điều hành phải kiểm soát lỗi trang của nhiều tiến trình

khác trong hệ thống, và hệ điều hành khó có thể dự đoán được nguy cơ xảy ra lỗi trang của các tiến trình trong hệ thống. Trong trường hợp này có thể lỗi trang sẽ lan truyền đến nhiều tiến trình khác trong hệ thống, khi đó việc xử lý lỗi trang của hệ điều hành sẽ phức tạp hơn rất nhiều. Đa số các hệ điều hành đều chọn cách thứ nhất vì nó đơn giản và không ảnh hưởng đến các tiến trình khác trong hệ thống.

- **"Neo" một số page:** Trên bộ nhớ chính tồn tại các page của các tiến trình đặc biệt quan trọng đối với hệ thống, nếu các tiến trình này bị tạm dừng thì sẽ ảnh hưởng rất lớn đến hệ thống và có thể làm cho hệ thống ngừng hoạt động, nên hệ điều hành không được đưa các page này ra đĩa trong bất kỳ trường hợp nào. Để tránh các thuật toán thay trang chọn các page này hệ điều hành tổ chức đánh dấu các page này, bằng cách đưa thêm một bít mới vào các phần tử trong các PCT, bít này được gọi là bít neo. Như vậy các thuật toán thay trang sẽ không xem xét đến các page được đánh dấu neo khi cần phải đưa một trang nào đó ra đĩa.

- **Phải tránh được trường hợp hệ thống xảy ra hiện tượng "trì trệ hệ thống":** Trì trệ hệ thống là hiện tượng mà hệ thống luôn ở trong tình trạng xử lý lỗi trang, tức là đa phần thời gian xử lý của processor đều dành cho việc xử lý lỗi trang của hệ điều hành. Hiện tượng này có thể được mô tả như sau: khi xử lý lỗi trang trong trường hợp trên bộ nhớ chính không còn frame trống, trong trường hợp này hệ điều hành phải chọn một page nào đó, ví dụ P_3, để swap out nó, để lấy frame trống đó, để nạp page vừa có yêu cầu nạp, để khắc phục lỗi trang. Nhưng khi vừa khắc phục lỗi trang này thì hệ thống lại xảy ra lỗi trang mới do hệ thống cần truy xuất dữ liệu ở trang P_3, hệ điều hành lại phải khắc phục lỗi trang này, và hệ điều hành phải swap out một page nào đó, ví dụ P_5. Nhưng ngay sau đó hệ thống lại xảy ra lỗi trang mới do không tìm thấy page P_5 trên bộ nhớ chính và hệ điều hành lại phải xử lý lỗi trang, và cứ như thế có thể hệ điều hành phải kéo dài việc xử lý lỗi trang mà không thể kết thúc được. Trong trường hợp này ta nói rằng: hệ thống đã rơi vào tình trạng "trì trệ hệ thống". Như vậy hệ thống có thể xảy ra hiện tượng "trì trệ hệ thống" khi: trên bộ nhớ không còn frame trống, page mà thuật toán thay trang chọn để swap out là một page không được "tốt", xét về khía cạnh dự báo lỗi trang của hệ điều hành.

- **Đánh dấu các trang bị thay đổi:** Khi xử lý lỗi trang, ta thấy hệ điều hành thường phải thực hiện thao tác swap out. Hệ điều hành phải mang một page của một tiến trình tại một khung trang nào đó ra lưu tạm trên đĩa cứng, tại không gian swap. Tức là, hệ điều hành phải tốn thời gian cho thao tác swap out, điều này sẽ làm giảm tốc độ của hệ thống và có thể gây lãng phí thời gian xử lý của processor. Hệ điều hành có thể hạn chế được điều này bằng cách: *không phải lúc nào hệ điều hành cũng phải thực hiện swap out một page để lấy khung trang trống mà hệ điều hành chỉ thực sự swap out một page khi page đó đã bị thay đổi kể từ lần nó được nạp vào bộ nhớ gần*

đây nhất. Khi đã quyết định swap out một page để lấy khung trang trống để nạp một page mới vào bộ nhớ, mà page cần swap này không bị thay đổi kể từ lần nạp gần đây nhất, hệ điều hành sẽ không swap out nó mà hệ điều hành chỉ nạp page mới vào bộ nhớ và ghi đè lên nó, điều này có nghĩa là hệ điều hành đã tiết kiện được thời gian swap out một page tiến trình ra đĩa. Để làm được điều này hệ điều hành phải giải quyết hai vấn đề sau: Thứ nhất, làm thế nào để xác định được một page là đã bị thay đổi hay chưa kể từ lần nạp vào bộ nhớ gần đây nhất. Thứ hai, nếu không swap out một page thì khi cần hệ điều hành sẽ swap in nó từ đâu.

Đối với vấn đề thứ nhất: hệ điều hành chỉ cần thêm một bít, bít modify chẳng hạn, vào phần tử trong bảng trang. Khi một page vừa được nạp vào bộ nhớ thì bit modify bằng 0, nếu sau đó nội dung của page bị thay đổi thì bít modify được đổi thành 1. Hệ điều hành sẽ dựa vào bít modify này để biết được một page có bị thay đổi hay không kể từ lần nạp vào bộ nhớ gần đây nhất.

Đối với vấn đề thứ hai: hệ điều hành có thể swap in một page tại ví trí ban đầu của nó trên đĩa, hoặc tại không gian swap của nó. Trong một số hệ điều hành khi một tiến trình được tạo thì lập tức hệ điều hành sẽ cấp cho nó một không gian swap trên đĩa, bất kỳ khi nào tiến trình bị swap out nó đều được swap đến không gian swap của nó, khi tiến trình kết thúc thì không gian swap của nó sẽ được giải phóng. Như vậy để chuẩn bị cho việc swap in sau này, khi nạp một page của tiến trình vào bộ nhớ hệ điều hành sẽ ghi nội dung của page này vào không gian swap của nó.

Chương 5: QUẢN LÝ BỘ NHỚ NGOÀI

Nội dung của chương cung cấp kiến thức về cách thức tổ chức lưu trữ dữ liệu, các phương pháp quản lý trên bộ nhớ ngoài.

5.1. Các khái niệm cơ bản

Yêu cầu quản lý bộ nhớ ngoài

Khi cần lưu trữ các chương trình hoặc dữ liệu, các hệ thống máy tính bắt buộc phải sử dụng bộ nhớ ngoài (đĩa từ, băng từ,...). Nhiệm vụ chính hệ điều hình phải đảm bảo các chức năng sau:

Quản lý không gian nhớ tự do trên bộ nhớ ngoài (free space manage)

Cấp phát không gian nhớ tự do (allocation methods)

Cung cấp các khả năng định vị bộ nhớ ngoài

Lập lịch cho bộ nhớ ngoài

Cấu trúc vật lý đĩa từ

Đĩa từ bao gồm một hoặc nhiều lá đĩa đặt đồng trục. Mỗi mặt đĩa chia thành các rãnh tròn đồng tâm gọi là track, mỗi track được chia thành các cung gọi là sector. Trên mỗi mặt đĩa có đầu đọc ghi dữ liệu.

Hệ điều hành xem đĩa như mảng một chiều mà thành phần là các khối đĩa (disk block). Mỗi khối đĩa ghi các thông tin về mặt đĩa, track, sector mà hệ điều hành có thể định vị trên đó.

5.2. Các phương pháp quản lý không gian nhớ tự do

Vì không gian trống là giới hạn nên chúng ta cần dùng lại không gian từ các tập tin bị xoá cho các tập tin mới nếu có thể. Để giữ vết của không gian đĩa trống, hệ thống duy trì một danh sách không gian trống. Danh sách không gian trống ghi lại tất cả khối đĩa trống. Để tạo tập tin, chúng ta tìm trong danh sách không gian trống lượng không gian được yêu cầu và cấp phát không gian đó tới tập tin mới. Sau đó, không gian này được xoá từ danh sách không gian trống. Khi một tập tin bị xoá, không gian đĩa của nó được thêm vào danh sách không gian trống. Mặc dù tên của nó là danh sách nhưng danh sách không gian trống có thể không được cài như một danh sách.

a) Bit vector

Thường thì danh sách không gian trống được cài đặt như một bản đồ bit (bit map) hay một vector bit (bit vector). Mỗi khối được biểu diễn bởi 1 bit. Nếu khối là trống, bit của nó được đặt là 1, nếu khối được cấp phát bit của nó được

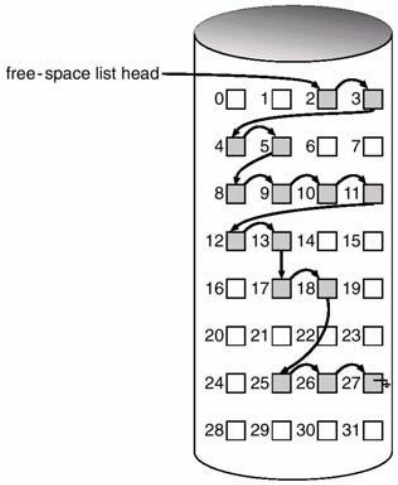
đặt là 0.

Thí dụ, xét một đĩa khi các khối 2, 3, 4, 5, 8, 9, 10, 11, 12, 13, 17, 18, 25, 26, và 27 là trống và các khối còn lại được cấp phát. Bản đồ bit không gian trống sẽ là: 00111100111111000110000011100000…

Lợi điểm chính của tiếp cận này là tính tương đối đơn giản và hiệu quả của nó trong việc tìm khối trống đầu tiên, hay n khối trống tiếp theo trên đĩa.

Một lần nữa, chúng ta thấy các đặc điểm phần cứng định hướng chức năng phần mềm. Tuy nhiên, các vector bit là không đủ trừ khi toàn bộ vector được giữ trong bộ nhớ chính. Giữ nó trong bộ nhớ chính là có thể cho các đĩa nhỏ hơn, như trên các máy vi tính nhưng không thể cho các máy lớn hơn. Một đĩa 1.3 GB với khối 51 bytes sẽ cần một bản đồ bit 332 KB để ghi lại các khối trống. Gom bốn khối vào một nhóm có thể giảm số này xuống còn 83 KB trên đĩa.

b) Danh sách liên kết



Hình 5.1. Danh sách không gian trống được liên kết trên đĩa

Một tiếp cận khác để quản lý bộ nhớ trống là liên kết tất cả khối trống, giữ một con trỏ tới khối trống đầu tiên trong một vị trí đặc biệt trên đĩa và lưu nó trong bộ nhớ. Khối đầu tiên này chứa con trỏ chỉ tới khối đĩa trống tiếp theo,..Trong thí dụ trên, chúng ta có thể giữ một con trỏ chỉ tới khối 2 như là khối trống đầu tiên. Khối 2 sẽ chứa một con trỏ chỉ tới khối 3, khối này sẽ chỉ tới khối 4,…(như hình X-10). Tuy nhiên, cơ chế này không hiệu quả để duyệt danh sách, chúng ta phải đọc mỗi khối, yêu cầu thời gian nhập/xuất đáng kể. Tuy nhiên, duyệt danh sách trống không là hoạt động thường xuyên. Thường thì, hệ điều hành cần một khối trống để mà nó có thể cấp phát khối đó tới một tập tin, vì thế khối đầu tiên trong

danh sách trống được dùng. Phương pháp FAT kết hợp với đếm khối trống thành cấu trúc dữ liệu cấp phát.

c) Nhóm

Thay đổi tiếp cận danh sách trống để lưu địa chỉ của n khối trống trong khối trống đầu tiên. n-1 khối đầu tiên này thật sự là khối trống. Khối cuối cùng chứa địa chỉ của n khối trống khác, …Sự quan trọng của việc cài đặt này là địa chỉ của một số lượng lớn khối trống có thể được tìm thấy nhanh chóng, không giống như trong tiếp cận danh sách liên kết chuẩn.

d) Bộ đếm

Một tiếp cận khác đạt được lợi điểm trong thực tế là nhiều khối kề có thể được cấp phát và giải phóng cùng lúc, đặc biệt khi không gian được cấp phát với giải thuật cấp phát kề hay thông qua nhóm. Do đó, thay vì giữ một danh sách n địa chỉ đĩa trống, chúng ta có thể giữ địa chỉ của khối trống đầu tiên và số n khối kề trống theo sau khối đầu tiên. Mỗi mục từ trong danh sách không gian trống sau đó chứa một địa chỉ đĩa và bộ đếm. Mặc dù mỗi mục từ yêu cầu nhiều không gian hơn một địa chỉ đĩa đơn,

nhưng toàn bộ danh sách sẽ ngắn hơn với điều kiện là bộ đếm lớn hơn 1

5.3. Các phương pháp cấp phát không gian nhớ tự do

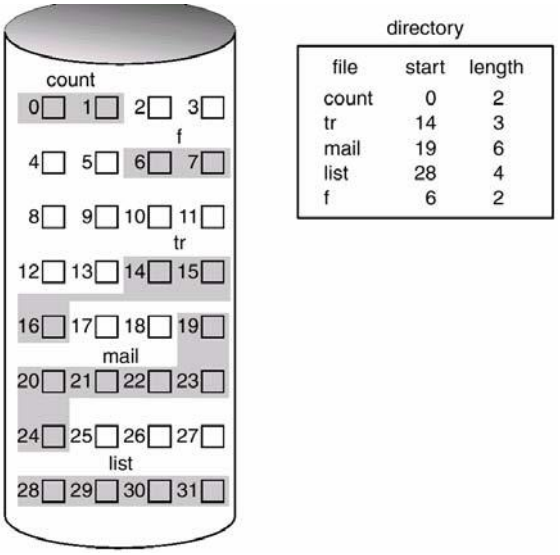
Tính tự nhiên của truy xuất trực tiếp đĩa cho phép chúng ta khả năng linh hoạt trong việc cài đặt tập tin. Trong hầu hết mọi trường hợp, nhiều tập tin sẽ được lưu trên cùng đĩa. Vấn đề chính là không gian cấp phát tới các tập tin này như thế nào để mà không gian đĩa được sử dụng hiệu quả và các tập tin có thể được truy xuất nhanh chóng. Ba phương pháp quan trọng cho việc cấp phát không gian đĩa được sử dụng rộng rãi: cấp phát kề, liên kết và chỉ mục. Mỗi phương pháp có ưu và nhược điểm. Một số hệ thống hỗ trợ cả ba. Thông dụng hơn, một hệ thống sẽ dùng một phương pháp cụ thể cho tất cả tập tin.

a) Cấp phát kề

Phương pháp cấp phát kề yêu cầu mỗi tập tin chiếm một tập hợp các khối kề nhau trên đĩa. Các địa chỉ đĩa định nghĩa một thứ tự tuyến tính trên đĩa. Với thứ tự này, giả sử rằng chỉ một công việc đang truy xuất đĩa, truy xuất khối b+1 sau khi khối b không yêu cầu di chuyển trước. Khi di chuyển đầu đọc được yêu cầu (từ cung từ cuối cùng của cylinder tới cung từ đầu tiên của cylinder tiếp theo), nó chỉ di chuyển một rãnh (track). Do đó, số lượng tìm kiếm đĩa được yêu cầu cho truy xuất kề tới các tập tin được cấp phát là nhỏ nhất.

Cấp phát kề của một tập tin được định nghĩa bởi địa chỉ đĩa và chiều dài (tính

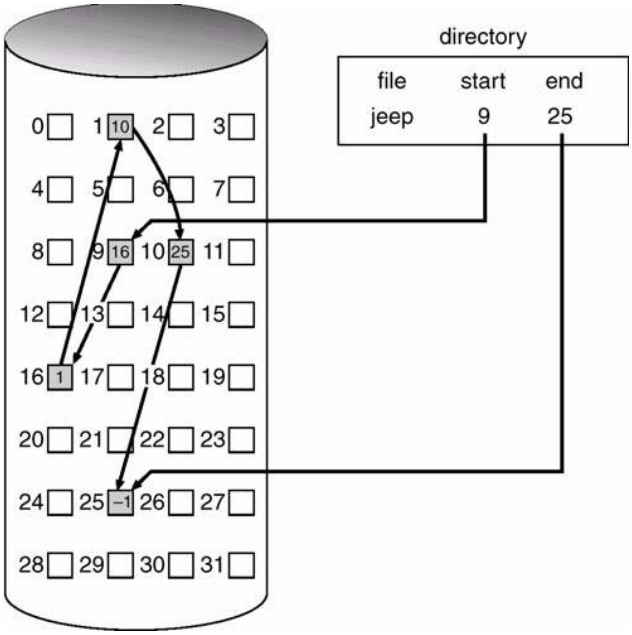
bằng đơn vị khối) của khối đầu tiên. Nếu tập tin có n khối và bắt đầu tại khối b thì nó chiếm các khối b, b+1, b+2,..,b+n-1. Mục từ thư mục cho mỗi tập tin hiển thị địa chỉ của khối bắt đầu và chiều dài của vùng được cấp phát cho tập tin này



Hình 5.2. Danh sách không gian trống được cấp phát kề

b) Cấp phát liên kết

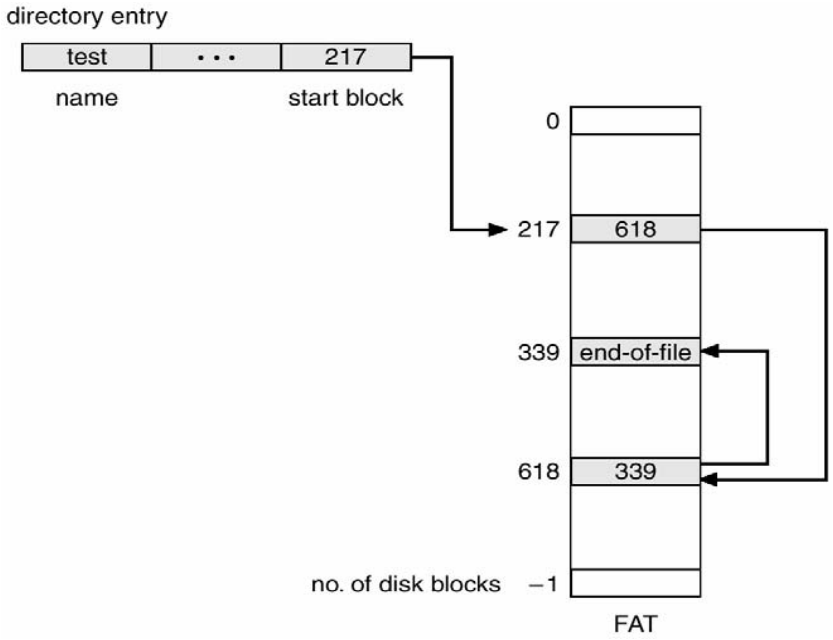
Cấp phát liên kết giải quyết vấn đề của cấp phát kề. Với cấp phát liên kết, mỗi tập tin là một danh sách các khối đĩa được liên kết; các khối đĩa có thể được phân tán khắp nơi trên đĩa. Thư mục chứa một con trỏ chỉ tới khối đầu tiên và các khối cuối cùng của tập tin. Thí dụ, một tập tin có 5 khối có thể bắt đầu tại khối số 9, tiếp tục là khối 16, sau đó khối 1, khối 10 và cuối cùng khối 25 . Mỗi khối chứa một con trỏ chỉ tới khối kế tiếp. Các con trỏ này không được làm sẵn dùng cho người dùng.



Hình 5.3. Danh sách không gian trống được cấp phát liên kết

Một thay đổi quan trọng trên phương pháp cấp phát liên kết là dùng bảng cấp phát tập tin (file allocation table-FAT). Điều này đơn giản nhưng là phương pháp cấp phát không gian đĩa hiệu quả được dùng bởi hệ điều hành MS-DOS và OS/2. Một phần đĩa tại phần bắt đầu của mỗi phân khu được thiết lập để chứa bảng này. Bảng này có một mục từ cho mỗi khối đĩa và được lập chỉ mục bởi khối đĩa. FAT được dùng nhiều như là một danh sách liên kết. Mục từ thư mục chứa số khối của khối đầu tiên trong tập tin. Mục từ bảng được lập chỉ mục bởi số khối đó sau đó chứa số khối của khối tiếp theo trong tập tin. Chuỗi này tiếp tục cho đến khi khối cuối cùng, có giá trị cuối tập tin đặc biệt như mục từ bảng. Các khối không được dùng được hiển thị bởi giá trị bảng 0. Cấp phát một khối mới tới một tập tin là một vấn đề đơn giản cho việc tìm mục từ bảng có

giá trị 0 đầu tiên và thay thế giá trị kết thúc tập tin trước đó với địa chỉ của khối mới. Sau đó, số 0 được thay thế với giá trị kết thúc tập tin. Một thí dụ minh hoạ là cấu trúc FAT của hình X-7 cho một tập tin chứa các khối đĩa 217, 618 và 339.



Hình 5.4. Bảng cấp phát tập tin

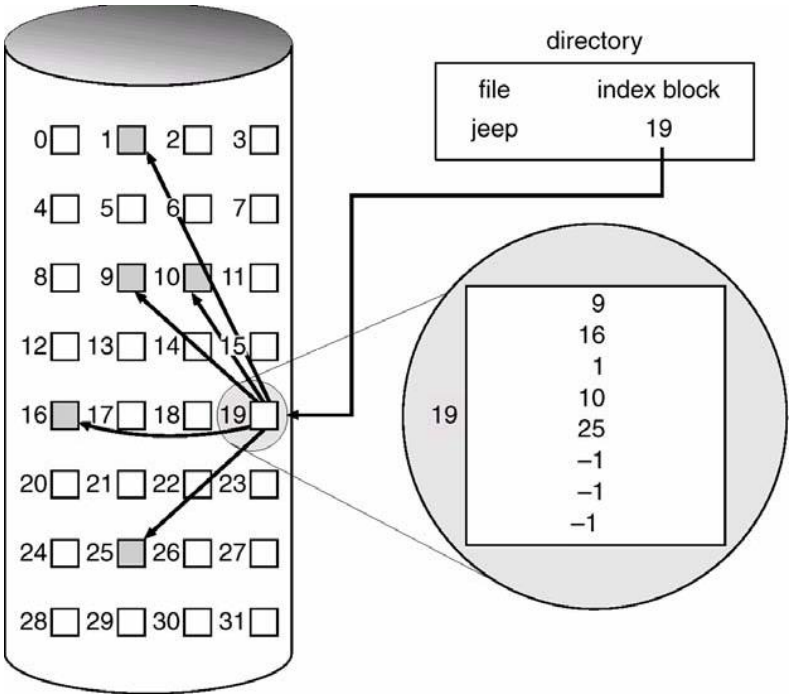
Cơ chế cấp phát FAT có thể dẫn tới số lượng lớn tìm kiếm đầu đọc đĩa nếu FAT không được lưu trữ(cache). Đầu đọc đĩa phải di chuyển tới điểm bắt đầu của phân khu để đọc FAT và tìm vị trí khối sau đó di chuyển tới vị trí của chính khối đĩa đó. Trong trường hợp xấu nhất, cả hai di chuyển xảy ra cho mỗi khối đĩa. Lợi điểm là thời gian truy xuất ngẫu nhiên được cải tiến vì đầu đọc đĩa có thể tìm vị trí của bất cứ khối nào bằng cách đọc thông tin trong FAT.

c) Cấp phát được lập chỉ mục

Cấp phát liên kết giải quyết việc phân mãnh ngoài và vấn đề khai báo kích thước của cấp phát kề. Tuy nhiên, cấp phát liên kết không hỗ trợ truy xuất trực tiếp hiệu quả vì các con trỏ chỉ tới các khối được phân tán với chính các khối đó qua đĩa và cần được lấy lại trong thứ tự. Cấp phát được lập chỉ mục giải quyết vấn đề này bằng cách mang tất cả con trỏ vào một vị trí: khối chỉ mục (index block).

Mỗi tập tin có khối chỉ mục của chính nó, khối này là một mảng các địa chỉ khối đĩa. Mục từ thứ i trong khối chỉ mục chỉ tới khối i của tập tin. Thư mục chứa địa chỉ của

khối chỉ mục (như hình 2.8). Để đọc khối i, chúng ta dùng con trỏ trong mục từ khối chỉ mục để tìm và đọc khối mong muốn. Cơ chế này tương tự như cơ chế phân trang.



Hình 5.5. Cấp phát không gian đĩa được lập chỉ mục

5.4. Lập lịch cho đĩa

Khái niệm về lập lịch cho đĩa

Lập lịch cho đĩa là xây dựng các thuật toán dịch chuyển đầu từ đọc ghi sao cho thời gian truy nhập đĩa là tối ưu nhất.

Một số phương pháp lập lịch

-FCFS

-SSTF

-Scan

-C-Scan

-Look

-C-Look

5.5. Hệ file

Dữ liệu được xử lý trong máy tính được bảo quản lâu dài trên băng từ, đĩa từ, đĩa quang v.v… và dữ liệu được tập hợp lại một cách có tổ chức thành các file dữ liệu theo

mục đích sử dụng. File có thể là chương trình của người dùng, một chương trình của hệ điều hành, một văn bản, một tập hợp dữ liệu. Trong một số hệ điều hành, một số thiết bị ngoại vi cũng được quan niệm như file dữ liệu.

Theo góc độ quan sát của người dùng, dữ liệu trong một file lại được tổ chức thành các bản ghi lôgic (gọi tắt bản ghi), mà mỗi bản ghi lôgic có thể là một byte hoặc một cấu trúc dữ liệu nào đó. Bản ghi chính là đơn vị dữ liệu mà chương trình người dùng quan tâm đến và xử lý theo mỗi nhịp làm việc: file là tập hợp (được người dùng quan niệm là một dãy) các bản ghi có tổ chức. Thông thường, trong file tồn tại một thứ tự giữa các bản ghi, thứ tự đó thể hiện vị trí logic giữa các bản ghi với nhau (chẳng hạn như thứ tự đưa bản ghi vào file).

Chương 6: QUẢN LÝ THIẾT BỊ

Chương này cung cấp kiến thức về thiết bị vào ra chuẩn và thiết bị vào ra mở rộng. Đồng thời, người học nắm được nguyên tắc tổ chức, quản lý thiết bị ngoại vi của hệ điều hành cũng như các kỹ thuật cơ bản được áp dụng trong quản lý thiết bị ngoại vi.

6.1. Nguyên tắc tổ chức và quản lý thiết bị

61.1. Yêu cầu của quản lý thiết bị

Chức năng của các thiết bị ngoại vi là đảm nhiệm việc truyền thông tin qua lại giữa các bộ phận của hệ thống. Do đó, nhiệm vụ của hệ điều hành là tổ chức và truy nhập thông tin trên các thiết bị.

Ngoài các thiết bị ngoại vi chuẩn mang tính chất bắt buộc (màn hình, bàn phím, máy in ...) còn có các thiết bị ngoại vi mở rộng đa dạng và phong phú. Các thiết bị này đa dạng về chủng loại và nguyên tắc hoạt động. Điều này đòi hỏi hệ điều hành phải quản lý, điều khiển và khai thác các thiết bị này một cách hiệu quả.

CPU không làm việc trực tiếp với thiết bị ngoại vi, do đó, hệ điều hành cần phải tổ chức thiết bị ngoại vi sao cho CPU không bị phụ thuộc vào sự biến động của thiết bị ngoại vi.

6.1.2. Nguyên tắc của tổ chức và quản lý thiết bị

Nguyên tắc cơ bản để tổ chức và quản lý thiết bị vào ra: CPU chỉ điều khiển các thao tác vào ra chứ không trực tiếp thực hiện các thao tác này. Để đảm bảo được nguyên tắc này, các thiết bị vào ra không gắn trực tiếp vào CPU mà gắn với các thiết bị đặc biệt - thiết bị quản lý (Control Device). Một thiết bị quản lý có thể kết nối với nhiều thiết bị vào ra.

Thiết bị quản lý đóng vai trò như một máy tính chuyên dụng có nhiệm vụ điều khiển với các thiết bị kết nối với nó gọi là kênh vào ra. Mỗi kênh vào ra có ngôn ngữ và hệ lệnh riêng. Chúng hoạt động độc lập với nhau, độc lập với CPU và độc lập với các thành phần khác của hệ thống.

Một hệ thống máy tính có thể có nhiều kênh vào ra. Mỗi kênh vào ra có thể có những kênh riêng của mình. Để điều khiển các kênh cần có các chương trình riêng gọi là chương trình điều khiển kênh.

6.2. Các kỹ thuật áp dụng trong quản lý thiết bị

6.2.1. Kỹ thuật vùng đệm

a. Khái niệm và mục đích của vùng đệm

Vùng đệm (Buffer) là một vùng nhớ trung gian làm nơi lưu trữ thông tin tạm thời trong các thao tác vào ra.

Để thực hiện một thao tác vào ra, hệ thống cần thực hiện các bước sau:

- Kích hoạt thiết bị
- Chờ thiết bị đạt trạng thái thích hợp
- Chờ thao tác vào ra được thực hiện

Việc chờ đợi các thiết bị vào ra đạt trạng thái thích hợp chiếm khá nhiều thời gian trong tổng thời gian thực hiện thao tác vào ra. Vì vậy, để đảm bảo tốc độ hoạt động chung của toàn hệ thống, cần sử dụng vùng đệm nhắm mục đích:

- Giảm số lượng thao tác vào ra vật lý
- Cho phép thực hiện song song các thao tác vào ra với các thao tác xử lý thông tin khác nhau.
- Cho phép thực hiện trước các phép nhập dữ liệu.

b. Phân loại vùng đệm

Có nhiều phương pháp tổ chức vùng đệm khác nhau nhưng có thể phân chia chúng làm ba loại chính: vùng đệm trung chuyển, vùng đệm xử lý và vùng đệm vòng tròn.

Vùng đệm trung chuyển

Đối với kiểu vùng đệm trung chuyển, hệ thống tổ chức hai vùng nhớ riêng biệt: vùng nhờ vào và vùng nhớ ra.Vùng nhớ vào chỉ dùng để nhập thông tin còn vùng nhớ ra chỉ dùng để ghi thông tin. Tương ứng trong hệ thống có hai lệnh để đưa thông tin vào và lấy thông tin ra (read/write).

Khi gặp lệnh đọc, thông tin sẽ được chuyển từ vùng nhớ vào các địa chỉ tương ứng nêu trong các chương trình ứng dụng. Sau khi giá trị cuối cùng của vùng đệm được lấy ra xử lý, vùng đệm trở nên rỗng và hệ thống tổ chức nhập thông tin mới vào thời điểm sớm nhất có thể được. Để giảm thời gian chờ đợi, hệ thống có thể tổ chức nhiều vùng đệm vào, khi hết thông tin ở một vùng đệm, hệ thống chuyển sang vùng đệm kế tiếp.

Khi gặp lệnh ghi, thông tin được tổ chức tương tự nhưng theo chiều ngược lai. Lệnh ghi không thực hiện trực tiếp trên thiết bị mà ghi vào vùng đệm ra. Khi một vùng đệm ra đầy, hệ thống sẽ chuyển sang làm việc với vùng đệm kế tiếp, đồng thời tổ chức đưa thông tin từ vùng đệm trước ra thiết bị.

Ưu điểm của vòng đệm trung chuyển là có hệ số song song cao, phổ dụng, cách thức tổ chức đơn giản. Nhược điểm là tốn bộ nhớ, kéo dài thời gian trao đổi thông tin ở bộ nhớ trong.

Vùng đệm xử lý

Trong vùng đệm xử lý, cả thông tin vào và thông tin ra cùng được xử lý trong một vùng bộ nhớ, thông tin không cần phải lưu trữ ở nhiều nơi khác nhau trong bộ nhớ. Trong trường hợp này, lệnh đọc xác định địa chỉ thông tin chứ không cần cung cấp giá trị thông tin như trong vùng đệm trung chuyển.

Phương pháp này có ưu điểm là tiết kiệm không gian nhớ, rút ngắn thời gian trao đổi thông tin ở bộ nhớ trong nhưng tốc độ giải phóng vùng đệm chậm vì vậy hệ số song song thấp hơn so với vùng đệm trung chuyển. Mặt khác, không phải thao tác vào ra nào cũng thực hiện được theo phương pháp này. Đây là một phương pháp tổ chức vùng đệm phức tạp.

Vùng đệm vòng tròn

Trong cách tổ chức này, hệ thống làm việc với ba vùng đệm: một vùng đệm đưa thông tin vào, một vùng đệm đưa thông tin ra và một vùng đệm xử lý. Sau một khoảng thời gian nhất định thì vai trò của các vòng đệm được trao đổi cho nhau.

Loại vùng đệm này có thể gắn với từng file cụ thể hoặc gắn với toàn hệ thống. Trong chế độ gắn với file, vùng đệm được xây dựng khi mở file, xóa khi đóng file và chỉ phục vụ riêng cho file đó. Phương pháp này đặc biệt phù hợp khi mỗi file có kích thước bản ghi vật lý riêng. Trong trường hợp các file có kích thước gần giống nhau, phương pháp vùng đệm chung sẽ được sử dụng cho toàn bộ hệ thống.

Vùng đệm có thể trở thành tài nguyên găng khi có nhiều file được mở cùng một lúc. Để giảm khả năng cạnh tranh xảy ra ở vùng đệm, có thể tăng số lượng vùng đệm từ khi nạp hệ thống nhưng như vậy sẽ chiếm dụng nhiều bộ nhớ và làm tăng thời gian dịch vụ của hệ thống.

6.2.2. Kỹ thuật kết khối

Để giảm số lần truy nhập vật lý, hệ thống còn xử lý kỹ thuật kết khối tức là ghép nhiều bản ghi logic thành bản ghi vật lý và việc trao đổi thông tin tiến hành theo bản ghi vật lý.

Thông thường, có các cách tổ chức kết khối sau:

- Mỗi bản ghi vật lý chứa một số nguyên lần bản ghi logic và giá trị này là như nhau với mọi bản ghi vật lý.
- Mỗi bản ghi vật lý chứa một số nguyên lần bản ghi logic và giá trị này khác nhau với mỗi bản ghi vật lý.
- Bản ghi vật lý có độ dài cố định, không phụ thuộc vào bản ghi logic.

- Bản ghi vật lý chỉ chứa một phần bản ghi logic, do đó cần ghép nhiều bản ghi vật lý mới được bản ghi logic.

Thao tác kết khối sẽ kéo theo các chi phí bổ sung như: cần phải có bộ nhớ lưu trữ các chương trình phục vụ kết khối và mở khối, tốn thời gian xử lý bản ghi, đặc biệt khi một bản ghi logic nằm trên nhiều bản ghi vật lý khác nhau. Tuy vậy, ưu điểm đặc biệt của phương pháp này là giảm đáng kể số lần truy nhật vật lý.

6.2.3. Xử lý lỗi

Bất kỳ thành phần nào của hệ thống cũng có thể hoạt động không chính xác. Điều này xảy ra cả với kỹ thuật phần cứng và phần mềm. Ngay cả với những chương trình điều khiển được xây dựng cẩn thận, chu đáo vẫn có thể có lỗi. Trong thực tế, khả năng xảy ra trong quá trình thao tác với các thiết bị vào ra là rất cao. Nguyên nhân là do quá trình thao tác với thiết bị vào ra phụ thuộc nhiều vào môi trường bên ngoài: bị hao mòn trong quá trình sử dụng, bị nhiễm từ ...

Phương pháp chủ yếu để chống lỗi là giao trách nhiệm phát hiện lỗi cho hệ thống chứ không phải cho người dùng. Vì có nhiều nguyên nhân phát sinh lỗi nên hệ thống phải sử dụng linh hoạt các chiến lược kiểm tra thiết bị.

Khi phát hiện lỗi, hệ thống cố gắng khác phục bằng cách thực hiện nhiều lần thao tác vào ra. Nếu vẫn có lỗi ổn định thì hệ thống cố gắng khôi phục thông tin ban đầu, trong trường hợp không khắc phục được thì hệ thống thông báo cho người dùng tự quyết định.

Việc kiểm tra và xử lý lỗi là một quá trình phức tạp liên quan chặt chẽ với đặc trưng của từng thiết bị cụ thể. Tuy vậy, mỗi thiết bị đều cung cấp một mã trả về cho hệ thống để chương trình xử lý kết quả phân tích, đánh giá.

Để công việc phân tích, kiểm tra lỗi không chiếm nhiều thời gian xử lý của CPU, ảnh hưởng tới tốc độ của hệ thống thì các thiết bị thường có xu hướng cục bộ hóa lỗi (phân tích, xử lý ngay tại thiết bị).

Ví dụ về nguyên tắc hoạt động của cơ chế kiểm tra chẵn lẻ (Parity checking)

Phương pháp đơn giản nhất là VRC (Vertiacl Redundancy Check). Trong phương pháp này, mỗi chuỗi bit biểu diễn một ký tự trong dữ liệu cần kiểm tra được thêm vào một bit kiểm tra (gọi là parity bit). Bit này có giá trị bằng 0 nếu số lượng các bit 1 trong chuỗi là chẵn và ngược lại. Hệ thống sẽ căn cứ vào đó để phát hiện lỗi.

Vị trí bit trong ký tự	Chuỗi ký tự cần kiểm tra				
	A	S	C	I	I
1	1	1	1	1	1
2	0	0	0	0	0
3	0	1	0	0	0
4	0	0	0	1	1
5	0	0	0	0	0
6	0	1	1	0	0
7	1	1	1	1	1
VRC	0	0	1	1	1

Bảng 6.1. Kiểm tra VRC

Nhược điểm của VRC là không định vị được bit bị lỗi hoặc nếu có một số lần chẵn các bit bị lỗi thì giá trị của parity bit không thay đổi.

Để khắc phục có thể áp dụng thêm kỹ thuật LRC (Logitudinal Redundancy Check). LRC kiểm tra parity bit cho từng khối ký tự. Nếu kết hợp cả hai phương pháp VRC-LRC sẽ cho phương pháp kiểm tra lỗi hai chiều, nâng cao hiệu quả rõ rệt.

Vị trí bit trong ký tự	Chuỗi ký tự cần kiểm tra					LRC
	A	S	C	I	I	
1	1	1	1	1	1	1
2	0	0	0	0	0	0
3	0	1	0	0	0	1
4	0	0	0	1	1	0
5	0	0	0	0	0	0
6	0	1	1	0	0	0
7	1	1	1	1	1	1
VRC	0	0	1	1	1	1

Bảng 6.2. Kiểm tra lỗi kết hợp VRC-LRC

6.2.4. SPOOL (Simultaneous Peripheral Operations Online - hệ thống mô phỏng các phép trao đổi ngoại vi trong chế độ trực tiếp)

Thông thường, các thiết bị vào ra được xem như những công cụ kỹ thuật để các chương trình kênh và dữ liệu, đồng thời là nơi gửi các mã trạng thái cho hệ thống phân tích. Nhưng trên thực tế, mọi chương trình và dữ liệu của nó hoạt động như thiết bị vào ra có thực nên có thể dùng tiến trình để mô phỏng hoạt động vào ra và ngược lại, mọi thiết bị có thể coi như tiến trình. Trong thực tế, nhiều trường hợp được hệ thống mô phỏng vào ra bằng chương trình. Các chương trình này có thể hoạt động song song và tuân thủ theo nguyên tắc của tiến trình.

Việc mô phỏng thiết bị ngoại vi đã làm xuất hiện thiết bị ảo. Mỗi thiết bị ngoại vi cộng với chương trình mô phỏng tương ứng sẽ tạo ra một thiết bị hoàn toàn khác đối với người sử dụng.

Ngoài mục đích mô phỏng quá trình điều khiển thiết bị, thiết bị ảo còn có hai ứng dụng khác:

- Mô phỏng quá trình điều khiển và quản lý thiết bị mới đang trong quá trình chế tạo hoặc chưa lắp đặt.
- Tạo ra các SPOOL (Simultaneous Peripheral Operation On Line) hệ thống mô phỏng các thao tác trao đổi ngoại vi trực tiếp.

Nhiệm vụ của SPOOL là tạo ra hiệu ứng sử dụng song song, các thiết bị chỉ được phép khai thác trong chế độ tuần tự.

Chương 7: BẢO VỆ VÀ AN TOÀN HỆ THỐNG

Chương này cung cấp các kiến thức về mục đích và phương pháp bảo vệ hệ thống, tránh được sự can thiệp bất hợp pháp từ bên ngoài vào hệ thống và các nguy cơ tiềm ẩn từ bên trong. Đồng thời, chương này trang bị kiến thức về cơ chế an toàn hệ thống, virus và phòng tránh virus.

7.1. Bảo vệ hệ thống

7.1.1. Mục tiêu của bảo vệ hệ thống

Khi các tiến trình hoạt động đồng thời thì một tiến trình gặp lỗi có thể ảnh hưởng tới tiến trình khác và ảnh hưởng tới toàn bộ hệ thống. Hệ điều hành cần phải phát hiện, ngăn chặn không cho lỗi lan truyền và đặc biệt là phát hiện các lỗi tiềm ẩn để tăng cường độ tin cậy cho hệ thống.

Mặt khác, mục tiêu của bảo vệ hệ thống còn là chống sự truy nhập bất hợp lệ, đảm bảo cho các tiến trình của hệ thống sử dụng tài nguyên hợp lệ.

Bảo vệ hệ thống cần cung cấp cơ chế, chiến lược để quản trị việc sử dụng tài nguyên, quyết định những đối tượng nào trong hệ thống cần được bảo vệ và quy định các thao tác thích hợp trên các đối tượng này.

7.1.2. Miền bảo vệ

a. Khái niệm miền bảo vệ

Một hệ thống máy tính được xem như một tập các đối tượng *(objects)*. Một đối tượng có thể là một bộ phận phần cứng (CPU, bộ nhớ, ổ đĩa...) hay một thực thể phần mềm (tập tin, chương trình, semaphore...). Mỗi đối tượng có một định danh duy nhất để phân biệt với các đối tượng khác trong hệ thống, và chỉ được truy xuất đến thông qua các thao tác được định nghĩa chặt chẽ và được qui định ngữ nghĩa rõ ràng. Các thao tác có thể thực hiện được trên một đối tượng được xác định cụ thể tùy vào đối tượng.

Để có thể kiểm soát được tình hình sử dụng tài nguyên trong hệ thống, hệ điều hành chỉ cho phép các tiến trình được truy xuất đến các tài nguyên mà nó có quyền sử dụng, hơn nữa tiến trình chỉ được truy xuất đến các tài nguyên cần thiết trong thời điểm hiện tại để nó hoàn thành tác vụ (nguyên lý*need-to-know*) nhằm hạn chế các lỗi truy xuất mà tiến trình có thể gây ra trong hệ thống.

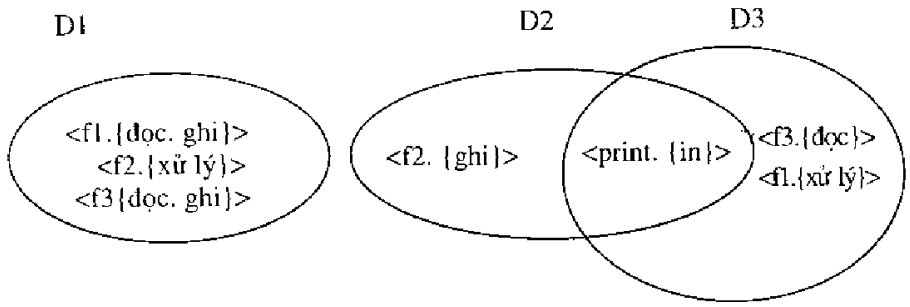
Mỗi tiến trình trong hệ thống đều hoạt động trong một miền bảo vệ *(protection domain)* nào đó. Một miền bảo vệ sẽ xác định các tài nguyên (đối tượng) mà những tiến

trình hoạt động trong miền bảo vệ này có thể sử dụng, và các thao tác hợp lệ các tiến trình này có thể thực hiện trên những tài nguyên đó.

b. Cấu trúc miền bảo vệ

Các khả năng thao tác trên một đối tượng được gọi là quyền truy xuất *(access right).* Một *miền bảo vệ* là một tập các quyền truy xuất, mỗi quyền truy xuất được định nghĩa bởi một bộ hai thứ tự *<đối tượng, {quyền thao tác} >*.

Các miền bảo vệ khác nhau có thể giao nhau một số quyền truy nhập.



Hình 7.1. Ví dụ một miền bảo vệ

Mối liên kết giữa một tiến trình và một miền bảo vệ có thể tĩnh hay động :

Liên kết tĩnh : trong suốt thời gian sống của tiến trình, tiến trình chỉ hoạt động trong một miền bảo vệ . Trong trường hợp tiến trình trải qua các giai đoạn xử lý khác nhau, ở mỗi giai đoạn tiến trình có thể thao tác trên những tập tài nguyên khác nhau bằng các thao tác khác nhau. Tuy nhiên, nếu sử dụng liên kết tĩnh, rõ ràng là ngay từ đầu miền bảo vệ đã phải đặc tả tất cả các quyền truy xuất qua các giai đoạn cho tiến trình , điều này có thể khiến cho tiến trình có dư quyền trong một giai đoạn nào đó, và vi phạm nguyên lý need-to-know. Để có thể tôn trọng nguyên lý này, khi đó cần phải có khả năng cập nhật nội dung miền bảo vệ để có thể phản ánh các quyền tối thiểu của tiến trình trong miền bảo vệ tại một thời điểm!

Liên kết động : cơ chế này cho phép tiến trình chuyển từ miền bảo vệ này sang miền bảo vệ khác trong suốt thời gian sống của nó. Để tiếp tục tuân theo nguyên lý **need-to-know**, thay vì sửa đổi nội dung của miền bảo vệ, có thể tạo ra các miền bảo vệ mới với nội dung thay đổi qua từng giai đoạn xử lý của tiến trình, và chuyển tiến trình sang hoạt động trong miền bảo vệ phù hợp theo từng thời điểm.

Một miền bảo vệ có thể được xây dựng cho:

Một người sử dụng : trong trường hợp này, tập các đối tượng được phép truy xuất phụ thuộc vào định danh của người sử dụng, miền bảo vệ được chuyển khi thay đổi người sử dụng.

Một tiến trình : trong trường hợp này, tập các đối tượng được phép truy xuất phụ thuộc vào định danh của tiến trình, miền bảo vệ được chuyển khi quyền điều khiển được chuyển sang tiến trình khác.

Một thủ tục : trong trường hợp này, tập các đối tượng được phép truy xuất là các biến cục bộ được định nghĩa bên trong thủ tục, miền bảo vệ được chuyển khi thủ tục được gọi.

c. Ma trận quyền truy nhập

Một cách trừu tượng, có thể biểu diễn mô hình bảo vệ trên đây như một ma trận quyền truy xuất (access matrix). Các dòng của ma trận biểu diễn các miền bảo vệ và các cột tương ứng với các đối tượng trong hệ thống. Phần tử ***acess[i,j]*** của ma trận xác định các quyền truy xuất mà một tiến trình hoạt động trong miền bảo vệ ***Di*** có thể thao tác trên đối tượng ***Oj***.

Object / Subject	F1	F2	F3	Printer
D1	đọc/ ghi	xử lý	đọc/ ghi	
D2		ghi		in
D3	xử lý		đọc	in

Hình 7.2. Ma trận quyền truy nhập

Cơ chế bảo vệ được cung cấp khi ma trận quyền truy nhập được cài đặt (với đầy đủ các thuộc tính ngữ nghĩa đả mô tả trên lý thuyết), lúc này người sử dụng có thể áp dụng các chiến lược bảo vệ bằng cách đặc tả nội dung các phần tử tương ứng trong ma trận _ xác định các quyền truy xuất ứng với từng miền bảo vệ , và cuối cùng, hệ điều hành sẽ quyết định cho phép tiến trình hoạt động trong miền bảo vệ thích hợp.

Ma trận quyền truy nhập cũng cung cấp một cơ chế thích hợp để định nghĩa và thực hiện một sự kiểm soát nghiêm nhặt cho cả phương thức liên kết tĩnh và động các tiến trình với các miền bảo vệ :

Có thể kiểm soát việc chuyển đổi giữa các miền bảo vệ nếu quan niệm miền bảo vệ cũng là một đối tượng trong hệ thống, và bổ sung các cột mô tả cho nó trong ma trận quyền truy nhập.

d. Các phương pháp cài đặt ma trận quyền truy nhập

Bảng toàn cục

Cách đơn giản nhất để cài đặt ma trận truy xuất là sử dụng một bảng bao gồm các bộ ba thứ tự < miền bảo vệ, đối tượng, các quyền truy xuất >. Mỗi khi thực hiện thao tác *M* trên đối tượng *Oj* trong miền bảo vệ *Di*, cần tìm trong bảng toàn cục một bộ ba < *Di, Oj, Rk* > mà M ∈ *Rk.* Nếu tìm thấy, thao tác *M* được phép thi hành, nếu không, xảy ra lỗi truy xuất.

Danh sách quyền truy nhập.

Có thể cài đặt mỗi cột trong ma trận quyền truy xuất như một danh sách quyền truy xuất đối với một đối tượng. Mỗi đối tượng trong hệ thống sẽ có một danh sách bao gồm các phần tử là các bộ hai thứ tự <miền bảo vệ, các quyền truy xuất>, danh sách này sẽ xác định các quyền truy xuất được qui định trong từng miền bảo vệ có thể tác động trên đối tượng. Mỗi khi thực hiện thao tác *M* trên đối tượng *Oj* trong miền bảo vệ *Di*, cần tìm trong danh sách quyền truy xuất của đối tượng *Oj*một bộ hai < *Di,Rk* > mà M ∈ *Rk.* Nếu tìm thấy, thao tác *M* được phép thi hành, nếu không, xảy ra lỗi truy xuất.

Ví dụ : Một miền bảo vệ trong hệ thống UNIX được xác định tương ứng với một người sử dụng (uid) trong một nhóm (gid) nào đó. Giả sử có 4 người dùng : A,B,C,D thuộc các nhóm tương ứng là system, staff, student, student. Khi đó các tập tin trong hệ thống có thể có các ACL như sau :

File0 : (A,*,RWX)

File1 : (A,system,RWX)

File2 : (A,*,RW-),(B,staff,R--),(D,*,RW-)

File3 : (*,student,R--)

File4 : (C,*,---),(*,student,R--)

Thực tế, hệ thống tập tin trong UNIX được bảo vệ bằng cách mỗi tập tin được gán tương ứng 9 bit bảo vệ , từng 3 bit sẽ mô tả quyềntruy xuất R(đọc), W(ghi) hay X(xử lý) của các tiến trình trên tập tin này theo thứ tự : tiến trình sỡ hữu các tiến trình cùng nhóm với tiến trình sỡ hữu, các tiến trình khác. Đây là một dạng ACL nhưng được nén thành 9 bit.

Danh sách khả năng

Mỗi dòng trong ma trận quyền truy xuất tương ứng với một miền bảo vệ sẽ được tổ chức thành một danh sách tiềm năng *(capabilities list)* :

Một danh sách tiềm năng của một miền bảo vệ là một danh sách các đối tượng và các thao tác được quyền thực hiện trên đối tượng khi tiến trình hoạt động trong miền bảo vệ này.

Một phần tử của C-List được gọi là một tiềm năng (capability) là một hình thức biểu diễn được định nghĩa một cách có cấu trúc cho một đối tượng trong hệ thống và các quyền truy xuất hợp lệ trên đối tượng này.

Cơ chế khóa và chìa

Đây là cách tiếp cận kết hợp giữa danh sách quyền truy xuất và danh sách khả năng. Mỗi đối tượng sỡ hữu một danh sách các mã nhị phân , được gọi là « khoá » *(lock)*. Cũng như thế, mỗi miền bảo vệ sẽ sỡ hữu một danh sách mã nhị phân gọi là « chìa » *(key)*. Một tiến trình hoạt động trong một miền bảo vệ chỉ có thể truy xuất đến một đối tượng nếu miền bảo vệ sỡ hữu một chìa tương ứng với một khóa trong danh sách của đối tượng.

Cũng như C_List, danh sách « khóa » và « chìa » được hệ điều hành quản lý, người sử dụng không thể truy xuất trực tiếp đến chúng để thay đổi nội dung.

e. Thu hồi quyền truy nhập

Trong một hệ thống bảo vệ động, đôi khi hệ điều hành cần thu hồi một số quyền truy xuất trên các đối tượng được chia sẻ giữa nhiều người sử dụng. Khi đó đặt ra một số vấn đề như sau :

Thu hồi tức khắc hay trì hoãn, trì hoãn đến khi nào ?

Nếu loại bỏ một quyền truy xuất trên một đối tượng, thu hồi quyền này trên tất cả hay chi một số người sử dụng?

Thu hồi một số quyền hay toàn bộ quyền trên một đối tượng ?

Thu hồi tạm thời hay vĩnh viển một quyền truy xuất ?

Đối với các hệ thống sử dụng danh sách quyền truy xuất, việc thu hồi có thể thực hiện dễ dàng : tìm và hủy trên ACL quyền truy xuất cần thu hồi, như vậy việc thu hồi được htực hiện tức thời, có thể áp dụng cho tất cả hay một nhóm người dùng, thu hồi toàn bộ hay một phần, và thu hồi vĩnh viễn hay tạm thời đều được.

Tuy nhiên trong các hệ sử dụng C_List, vấn đề thu hồi gặp khó khăn vì các tiềm năng được phân tán trên khắp các miền bảo vệ trong hệ thống, do vậy cần tìm ra chúng trước khi loại bỏ. Có thể giải quyết vấn đề này theo nhiều phương pháp :

Tái yêu cầu (Reacquisiton): loại bỏ các tiềm năng ra khỏi mỗi miền bảo vệ sau từng chu kỳ, nếu miền bảo vệ vẫn còn cần tiềm năng nào, nó sẽ tái yêu cầu tiềm năng đó lại.

Sử dụng các con trỏ đến tiềm năng (Back-pointers) : với mỗi đối tượng, lưu trữ các con trỏ đến những tiềm năng tương ứng trên đối tượng này. Khi cần thu hồi quyền truy xuất nào trên đối tượng, lần theo các con trỏ để cập nhật tiềm năng tương ứng.

Sử dụng con trỏ gián tiếp (Indirection) : các tiềm năng không trực tiếp trỏ đến các đối tượng, mà trỏ đến một bảng toàn cục do hệ điều hành quản lý. KHi cần thu hồi quyền, sẽ xoá phần tử tương ứng trong bảng này.

Khóa (Key) : nếu sử dụng cơ chế khóa và chìa, khi cần thu hồi quyền, chỉ cần thay đổi khóa và bắt buộc tiến trình hay người dùng yêu cầu chìa mới.

7.2. An toàn hệ thống

Bảo vệ hệ thống *(protection)* là một cơ chế kiểm soát việc sử dụng tài nguyên của các tiến trình hay người sử dụng để đối phó với các tình huống lỗi có thể phát sinh từ trong hệ thống . Trong khi đó khái niệm an toàn hệ thống *(security)* muốn đề cập đến mức độ tin cậy mà hệ thống duy trì khi phải đối phó không những với các vấn đề nội bộ, mà còn cả với những tác hại đến từ môi trường ngoài.

7.2.1. Các vấn đề của an toàn hệ thống

Hệ thống được gọi là an toàn nếu các tài nguyên được sử dụng đúng như quy ước trong mọi hoàn cảnh. Kém may mắm là điều này hiếm khi đạt được trong thực tế ! Thông thường, an toàn bị vi phạm vì các nguyên nhân vô tình hay cố ý phá hoại. Việc chống đỡ các phá hoại cố ý là rất khó khăn và gần như không thể đạt hiệu quả hoàn toàn. Bảo đảm an toàn hệ thống ở cấp cao chống lại các tác hại từ môi trường ngoài như hoả hoạn, mất điện, phái hoại...cần được thực hiện ở 2 mức độ *vật lý* (trang bị các thiết bị an toàn cho vị trí đạt hệ thống...) và *nhân sự* (chọn lọc cẩn thận những nhân viên làm việc trong hệ thống...). Nếu an toàn môi trường được bảo đảm khá tốt, an toàn của hệ thống sẽ được duy trì tốt nhờ các cơ chế của hệ điều hành (với sự trợ giúp của phần cứng).

Lưu ý rằng nếu bảo vệ hệ thống có thể đạt độ tin cậy 100%, thì các cơ chế an toàn hệ thống được cung cấp chỉ với hy vọng ngăn chặn bớt các tình huống bất an hơn là đạt đến độ an toàn tuyệt đối.

7.2.2. Các cơ chế an toàn hệ thống

Kiểm định danh tính

Để đảm bảo an toàn, hệ điều hành cần giải quyết tốt vấn đề chủ yếu là *kiểm định danh tính (authentication)*. Hoạt động của hệ thống bảo vệ phụ thuộc vào khả năng xác định các tiến trình đang xử lý. Khả năng này, đến lượt nó, lại phụ thuộc vào việc xác định được *người dùng* đang sử dụng hệ thống để có thể kiểm tra người dùng này được cho phép thao tác trên những tài nguyên nào.

Cách tiếp cận phổ biến nhất để giải quyết vấn đề là sử dụng *password* để kiểm định đúng danh tính của người dùng. Mỗi khi người dùng muốn sử dụng tài nguyên, hệ thống sẽ kiểm tra password của người dùng nhập vào với password được lưu trữ, nếu đúng, người dùng mới được cho phép sử dụng tài nguyên. Password có thể đuợc để bảo vệ từng đối tượng trong hệ thống, thậm chí cùng một đối tượng sẽ có các password khác nhau ứng với những quyền truy xuất khác nhau.

Cơ chế password rất dễ hiểu và dễ sử dụng do vậy được sử dụng rộng rãi, tuy nhiên yếu điểm nghiêm trọng của phương pháp này là khả năng bảo mật password rất khó đạt được sự hoàn hảo, những tác nhân tiêu cực có thể đoán ra password của người khác nhờ nhiều cách thức khác nhau.

Ngăn chặn nguyên nhân từ phía tiến trình

Trong môi trường mà một chương trình được tạo lập bởi người này lại có thể được người khác sử dụng, có thể xảy ra các tình huống sử dụng không đúng, từ đó dẫn đến những hậu qủa khó lường. Hai trường hợp điển hình là :

Ngựa thành Troy

Khi một người dùng A cho một chương trình do B viết hoạt động dưới danh nghĩa của mình (trong miền bảo vệ được gán tương ứng cho người dùng A), chương trình này có thể trở thành một « con ngựa thành Troy » vì khi đó các đoạn lệnh trong chương trình có thể thao tác trên các tài nguyên với những quyền tương ứng của người A (mà có thể người B vốn bị cấm!), nhiều chương trình như thế đã « lợi dụng hoàn cảnh » để gây ra các tác hại đáng tiếc.

Trap Door

Một mối đe dọa đặc biệt nguy hiểm và khó chống đỡ đến từ sự vô tình hay ý nghĩ bất chính của các lập trình viên. Khi xây dựng chương trình, các lập trình viên có thể để lại một « cánh cửa nhỏ » trong phần mềm mà chỉ có họ là có khả năng sử dụng , qua đó thâm nhập và phá hoại hệ thống (ví dụ làm tròn các số lẻ trong những tài khoản, và thu lợi riêng

từ phần dư này...). Vấn đề này rất khó đối phó vì cần phải tiến hành phân tích chương trình nguồn để tìm ra chỗ sơ hở.

7.3. Virus máy tính

7.3.1. Khái niệm Virus máy tính

Trong khoa học máy tính, virus máy tính (thường được người sử dụng gọi tắt là virus) là những chương trình hay đoạn mã được thiết kế để tự nhân bản và sao chép chính nó vào các đối tượng lây nhiễm khác (file, ổ đĩa, máy tính,...).

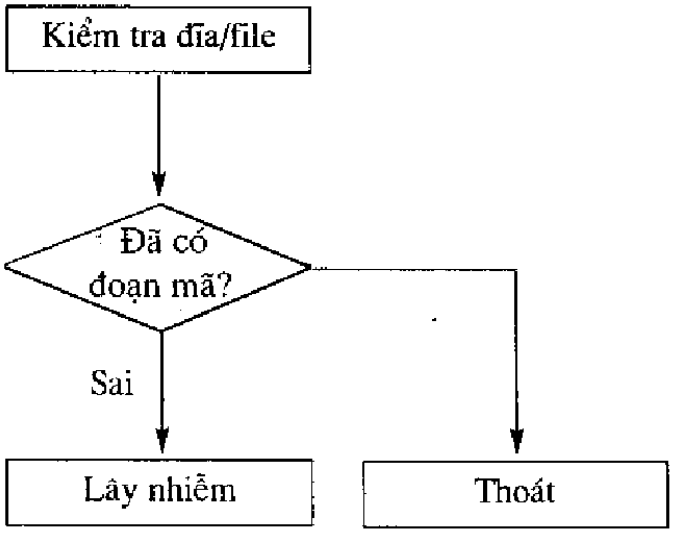
Trước đây, virus thường được viết bởi một số người am hiểu về lập trình muốn chứng tỏ khả năng của mình nên thường virus có các hành động như: cho một chương trình không hoạt động đúng, xóa dữ liệu, làm hỏng ổ cứng,... hoặc gây ra những trò đùa khó chịu.

Những virus mới được viết trong thời gian gần đây không còn thực hiện các trò đùa hay sự phá hoại đối máy tính của nạn nhân bị lây nhiễm nữa, mà đa phần hướng đến việc lấy cắp các thông tin cá nhân nhạy cảm (các mã số thẻ tín dụng) mở cửa sau cho tin tặc đột nhập chiếm quyền điều khiển hoặc các hành động khác nhằm có lợi cho người phát tán virus.

Chiếm trên 90% số virus đã được phát hiện là nhắm vào hệ thống sử dụng hệ điều hành họ Windows chỉ đơn giản bởi hệ điều hành này được sử dụng nhiều nhất trên thế giới. Do tính thông dụng của Windows nên các tin tặc thường tập trung hướng vào chúng nhiều hơn là các hệ điều hành khác. Cũng có quan điểm cho rằng Windows có tính bảo mật không tốt bằng các hệ điều hành khác (như Linux) nên có nhiều virus hơn, tuy nhiên nếu các hệ điều hành khác cũng thông dụng như Windows hoặc thị phần các hệ điều hành ngang bằng nhau thì cũng lượng virus xuất hiện có lẽ cũng tương đương nhau.

7.3.2. Cơ chế hoạt động của Virus

Chúng ta có thể hình dung quá trình hoạt động của Virus như sau:
Khi đọc một đĩa hoặc thi hành một chương trình bị nhiễm Virus, nó sẽ tạo ra một bản sao của mã và nằm thường trú trong bộ nhớ máy tính. Khi đọc một đĩa hoặc thực hiện một chương trình, đoạn mã chứa Virus nằm trong bộ nhớ sẽ kiểm tra đĩa/file đó đã tồn tại đoạn mã hay chưa? Nếu chưa thì tạo một bản sao khác lây nhiễm vào đĩa/file.



Hình 7.1. Cơ chế hoạt động của Virus

7.3.3. Phòng, tránh Virus

Không thể khẳng định chắc chắn bảo vệ an toàn 100% cho máy tính trước hiểm hoạ virus và các phần mềm hiểm độc, nhưng chúng ta có thể hạn chế đến tối đa có thể và có các biện pháp bảo vệ dữ liệu của mình.

Sử dụng phần mềm diệt Virus

Bảo vệ bằng cách trang bị thêm một phần mềm diệt virus có khả năng nhận biết nhiều loại virus máy tính và liên tục cập nhật dữ liệu để phần mềm đó luôn nhận biết được các virus mới.

(Để biết cách sử dụng phần mềm diệt virus hiệu quả, xem thêm tại "phần mềm diệt virus")

Trên thị trường hiện có rất nhiều phần mềm diệt virus.

Trong nước (Việt Nam): Bkav, CMC.

Của nước ngoài: Avira, Kaspersky, AVG.

Sử dụng tưởng lửa

Tường lửa (Firewall) không phải một cái gì đó quá xa vời hoặc chỉ dành cho các nhà cung cấp dịch vụ internet (ISP) mà mỗi máy tính cá nhân cũng cần phải sử dụng tường lửa để bảo vệ trước virus và các phần mềm độc hại. Khi sử dụng tường lửa, các thông tin vào

và ra đối với máy tính được kiểm soát một cách vô thức hoặc có chủ ý. Nếu một phần mềm độc hại đã được cài vào máy tính có hành động kết nối ra Internet thì tường lửa có thể cảnh báo giúp người sử dụng loại bỏ hoặc vô hiệu hoá chúng. Tường lửa giúp ngăn chặn các kết nối đến không mong muốn để giảm nguy cơ bị kiểm soát máy tính ngoài ý muốn hoặc cài đặt vào các chương trình độc hại hay virus máy tính.

Sử dụng tường lửa bằng phần cứng nếu người sử dụng kết nối với mạng Internet thông qua một modem có chức năng này. Thông thường ở chế độ mặc định của nhà sản xuất thì chức năng "tường lửa" bị tắt, người sử dụng có thể truy cập vào modem để cho phép hiệu lực (bật). Sử dụng tường lửa bằng phần cứng không phải tuyệt đối an toàn bởi chúng thường chỉ ngăn chặn kết nối đến trái phép, do đó kết hợp sử dụng tường lửa bằng các phần mềm.

Sử dụng tường lửa bằng phần mềm: Ngay các hệ điều hành họ Windows ngày nay đã được tích hợp sẵn tính năng tường lửa bằng phần mềm, tuy nhiên thông thường các phần mềm của hãng thứ ba có thể làm việc tốt hơn và tích hợp nhiều công cụ hơn so với tường lửa phần mềm sẵn có của Windows. Ví dụ bộ phần mềm ZoneAlarm Security Suite của hãng ZoneLab là một bộ công cụ bảo vệ hữu hiệu trước virus, các phần mềm độc hại, chống spam, và tường lửa.

Cập nhật các bản sửa lỗi của Windows

Hệ điều hành Windows (chiếm đa số) luôn luôn bị phát hiện các lỗi bảo mật chính bởi sự thông dụng của nó, tin tặc có thể lợi dụng các lỗi bảo mật để chiếm quyền điều khiển hoặc phát tán virus và các phần mềm độc hại. Người sử dụng luôn cần cập nhật các bản vá lỗi của Windows thông qua trang web Microsoft Update (cho việc nâng cấp tất cả các phần mềm của hãng Microsoft) hoặc Windows Update (chỉ cập nhật riêng cho Windows). Cách tốt nhất hãy đặt chế độ nâng cấp (sửa chữa) tự động (Automatic Updates) của Windows. Tính năng này chỉ hỗ trợ đối với các bản Windows mà Microsoft nhận thấy rằng chúng hợp pháp.

Bảo vệ dữ liệu máy tính

Nếu như không chắc chắn 100% rằng có thể không bị lây nhiễm virus máy tính và các phần mềm hiểm độc khác thì bạn nên tự bảo vệ sự toàn vẹn của dữ liệu của mình trước khi dữ liệu bị hư hỏng do virus (hoặc ngay cả các nguy cơ tiềm tàng khác như sự hư hỏng của các thiết bị lưu trữ dữ liệu của máy tính). Trong phạm vi về bài viết về virus máy tính, bạn có thể tham khảo các ý tưởng chính như sau:

Sao lưu dữ liệu theo chu kỳ là biện pháp đúng đắn nhất hiện nay để bảo vệ dữ liệu. Bạn có thể thường xuyên sao lưu dữ liệu theo chu kỳ đến một nơi an toàn như: các thiết bị

nhớ mở rộng (ổ USB, ổ cứng di động, ghi ra đĩa quang...), hình thức này có thể thực hiện theo chu kỳ hàng tuần hoặc khác hơn tuỳ theo mức độ cập nhật, thay đổi của dữ liệu của bạn.

Tạo các dữ liệu phục hồi cho toàn hệ thống không dừng lại các tiện ích sẵn có của hệ điều hành (ví dụ System Restore của Windows Me, XP...) mà có thể cần đến các phần mềm của hãng thứ ba, ví dụ bạn có thể tạo các bản sao lưu hệ thống bằng các phần mềm ghost, các phần mềm tạo ảnh ổ đĩa hoặc phân vùng khác.

Thực chất các hành động trên không chắc chắn là các dữ liệu được sao lưu không bị lây nhiễm virus, nhưng nếu có virus thì các phiên bản cập nhật mới hơn của phần mềm diệt virus trong tương lai có thể loại bỏ được chúng.

Chương 8: HỆ ĐIỀU HÀNH ĐA XỬ LÝ

Nội dung chương này khái quát được xu thế sử dụng hệ thống đa xử lý hiện nay, giúp người học hiểu được những nét cơ bản về hệ điều hành đa xử lý nhằm trang bị khả năng tự nghiên cứu trong tương lai.

8.1. Tổng quan về hệ điều hành đa xử lý

8.1.1. Tổng quan

Hiện nay, từ sự phát triển với tốc độ nhanh của công nghệ, máy tính ngày càng được phổ dụng trong xã hội. Mức độ thâm nhập của máy tính vào cuộc sống càng cao thì yêu cầu nâng cao năng lực của máy tính lại ngày càng trở nên cấp thiết. Bộ nhớ chính ngày càng rộng lớn; đĩa từ có dung lượng càng rộng, tốc độ truy nhập ngày càng cao; hệ thống thiết bị ngoại vi càng phong phú, hình thức giao tiếp người – máy ngày càng đa dạng. Như đã nói, CPU là một tài nguyên thể hiện chủ yếu nhất năng lực của hệ thống máy tính, vì vậy một trong những vấn đề trọng tâm nhất để tăng cường năng lực của hệ thống là tăng cường năng lực của CPU. Về vấn đề này, nảy sinh giải pháp theo hai hướng:

Giải pháp *tăng cường năng lực của một CPU riêng* cho từng máy máy tính: công nghệ vi mạch ngày càng phát triển vì vậy năng lực của từng CPU cũng ngày nâng cao, các dự án các vi mạch VLSI với hàng triệu, hàng chục triệu transitor. Tuy nhiên giải pháp này cũng nảy sinh những hạn chế về kĩ thuật: tốc độ truyền thông tin không vượt qua tốc độ ánh sáng; khoảng cách gần nhất giữa hai thành phần không thể giảm thiểu quá nhỏ v.v…

Song song với giải pháp tăng cường năng lực của CPU là giải pháp *liên kết nhiều CPU để* tạo ra một hệ thống chung có năng lực đáng kể: việc đưa xử lý song song tạo ra nhiều lợi điểm. Thứ nhất, chia các phần nhỏ công việc cho mỗi CPU đảm nhận, năng suất tăng không chỉ theo tỷ lệ thuận với một hệ số nhân mà còn cao hơn do không mất thời gian phải thực hiện những công việc trung gian.

Giải pháp này còn có lợi điểm tích hợp các hệ thống máy đã có để tạo ra một hệ thống mới với sức mạnh tăng gấp bội.

Trong chương này, xem xét việc chọn giải pháp đa xử lý theo nghĩa một hệ thống tính toán được tổ hợp không chỉ một CPU mà nhiều CPU trong một máy tính hoặc nhiều máy tính trong một hệ thống thống nhất. Gọi chung các hệ có nhiều CPU như vậy là *hệ đa xử lý.*

8.1.2. Phân loại các hệ đa xử lý

Có một số cách phân loại các hệ đa xử lý:

Ví dụ về hệ đa xử lý tập trung là tập các xử lý trong một siêu máy tính (supercomputer). Đặc trưng của hệ thống này là các CPU được liên kết với nhau trong một máy tính duy nhất;

Ví dụ về hệ đa xử lý phân tán là các mạng máy tính: mạng gồm nhiều máy tính liên kết và được đặt ở những vị trí khác nhau, với một khoảng cách có thể coi là xa tùy ý.

Phân loại theo đặc tính của các CPU thành phần: hệ đã xử lý thuần nhất hoặc hệ đa xử lý không thuần nhất v.v…

Một ví dụ dễ quen thuộc là trong các máy vi tính từ 80486 trở đi trong đó có hai CPU (80x86 và 80x87) là hai CPU không thuần nhất.

Siêu máy tính ILLIAC-IV gồm nhiều CPU có đặc trưng giống nhau là một ví dụ về thuần nhất.

Phân loại theo cách các CPU thành phần tiếp nhận và xử lý dữ liệu. Trong cách phân loại này bao gồm cả những máy tính đơn xử lý thông thường:

- *Đơn câu lệnh, đơn dữ liệu (SISD: single data single instruction)* được thể hiện trong máy tính thông thường; Mỗi lần làm việc, CPU chỉ xử lý "một dữ liệu" và chỉ có một câu lệnh được thực hiện.
- *Đơn câu lệnh, đa dữ liệu (SIMD: single instruction multiple data):*

Các bộ xử lý trong cùng một nhịp thời gian chỉ thực hiện cùng một câu lệnh. Có thể lấy ví dụ từ việc cộng hai vector cho trước: Các CPU thành phần đều thực hiện các phép cộng; đổi số tương ứng đã có từng CPU; sau đó, chọn tiếp lệnh (chỉ thị) mới để điều khiển công việc này. Thông thường có một hệ chọn câu lệnh chung và mọi CPU thành phần cùng thực hiện: siêu máy tính ILLIAC-IV sử dụng cách thức này, có một máy tính con có tác dụng lưu giữ hệ điều hành để điều khiển ILLIAC.IV (bộ xử lý ma trận).

- Đa câu lệnh, đơn dữ liệu (MISD: multiple instruction single data)

Trong các máy tính thuộc loại này, hệ thống gồm nhiều CPU, các CPU liên kết nhau một cách tuần tự: output của bộ xử lý này là input của bộ xử lý tiếp theo (ví dụ CRAY-1: Bộ xử lý vector). Các CPU kết nối theo kiểu này được gọi là kết nối "dây chuyền".

- Đa dữ liệu, đa câu lệnh (MIMD)

Mỗi bộ xử lý có bộ phân tích chương trình riêng; câu lệnh và dữ liệu do chính mỗi CPU phải đảm nhận; có thể hình dung các CPU này hoạt động hoàn toàn "độc lập nhau". Các hệ điều hành mạng, hệ điều hành phân tán là những ví dụ về đa dữ liệu, đa câu lệnh.

Trong nội dung ở chương này, xem xét cách phân loại dạng tập trung/phân tán song thực chất chỉ quan tâm đến hệ đa xử lý tập trung còn với hệ đa xử lý phân tán, sẽ có những chuyên đề riêng đáp ứng.

Chú ý, một xu thế nghiên cứu và triển khai các hệ thống tính toán đa xử lý thời sự là nghiên cứu về tính toán cụm trong đó các mô hình SIMD, MISD và MIMD tương ứng được phát triển.

8.2. Hệ điều hành đa xử lý tập trung

Hệ đa xử lý tập trung hoạt động trên các máy tính có nhiều CPU mà điển hình là các siêu máy tính: CRAY-1,ILLIAC-IV, Hitachi và các máy tính nhiều xử lý hiện nay (máy tính của khoa CNTT, trường ĐHKHTN-ĐHQGHN có hai bộ xử lý). Các tài nguyên khác CPU có thể được phân chia cho các CPU. Trong các hệ điều hành đa xử lý, hai bài toán lớn nhất có thể kể đến là phân phối bộ nhớ và phân phối CPU.

8.2.1. Phân phối bộ nhớ

Các quá trình xuất hiện trong bộ nhớ chung. Việc phân phối bộ nhớ được tiến hành cho quá trình theo các chế độ điều khiển bộ nhớ đã cài đặt: phân phối theo chế độ mẻ hay phân phối gián đoạn.

Để tăng tốc độ làm việc với bộ nhớ (bài toán xử lý con trỏ ngoài v.v.) có thể gắn với mỗi CPU một cache nhớ. Phân ra hai loại thâm nhập cache: tĩnh và động. Thâm nhập tĩnh: mỗi CPU chỉ thâm nhập cache tương ứng, không thâm nhập dữ liệu tại vùng cache của các CPU khác. Thâm nhập động cho phép CPU của máy này có thể thâm nhập các cache của CPU khác.

8.2.2. Bài toán điều khiển CPU

Có nhiều CPU, việc điều khiển CPU được phân ra một số cách như sau:

Toàn bộ các CPU dành cho một quá trình : một quá trình được phân phối CPU, song tự quá trình nói trên nảy sinh các quá trình con; mỗi quá trình con được giải quyết trên mỗi CPU. Các quá trình con có thể được coi như một tính toán hết sức đơn giản nào đó: Máy tính đa xử lý vector chia các công đoạn của quá trình và mỗi CPU thực hiện một quá trình con (một công đoạn) trong quá trình đó. Máy tính đa xử lý ma trận cho phép mọi CPU cùng thực hiện một thao tác.

Về dòng xếp hàng có thể xem xét theo hai mô hình dưới đây:

Mô hình tĩnh: Hoặc mỗi CPU có một dòng xếp hàng riêng; mỗi bài toán được gắn với từng dòng xếp hàng, việc điều khiển mỗi dòng xếp hàng như đã được chỉ ra độc lập với các dòng xếp hàng khác, mỗi quá trình được phát sinh gắn với một dòng xếp hàng nào đó;

Mô hình động: toàn bộ hệ thống gồm một hay một vài dòng xếp hàng, các quá trình được xếp lên các CPU khi rỗi (có thể sử dụng kiểu dữ liệu semaphore nhiều giá trị để phân phối CPU cho các quá trình này).

8.3. Hệ phân tán

8.3.1. Khái niệm của hệ đa xử lý phân tán

Trong phần phân loại hệ thống đa xử lý, chú ý cách phân loại theo vị trí đặt các CPU (tập trung và phân tán) thì hệ phân tán được xây dựng từ các " máy tính" rời rạc nhau: mỗi vị trí là một máy tính nguyên vẹn, có đầy đủ chức năng xử lý, lưu trữ và truyền dữ liệu.

Hệ tập trung cho phép xử lý song song theo thao tác hoặc theo quá trình, trong khi đó, hệ phân tán chỉ có thể xử lý song song theo quá trình: các quá trình con được xử lý trên các máy tính khác nhau. Việc phân chia quá trình cho các CPU thành phần hoặc theo chức năng của CPU đó (server/client) hoặc theoo một lịch được phân công của một hệ thống chung.

Do phân tán nên vấn đề truyền dẫn dữ liệu đóng vai trò quan trọng trong các hệ phân tán. Đây cũng là một trong những lí do điển hình nhất để cách thức xử lý song song trên các hệ phân tán là theo quá trình mà không phải theo phép toán.

8.3.2. Đặc điểm hệ phân tán

Hệ thống phân tán (kéo theo sự hình thành các hệ điều hành phân tán) được phát sinh do các nhu cầu hết sức tự nhiên về việc nâng cao năng lực tài nguyên hệ thống (sức mạnh của hệ thống tính toán và cơ sở dữ liệu chung v.v..). Giải pháp phân tán có tác dụng phát huy năng lực chung của toàn bộ hệ thống khi giải quyết bài toán với kích thước bài toán tăng lên và vẫn đảm bảo hoạt động bình thường của các máy tính thành viên. Hệ thống phân tán được thiết lập hoặc là một hệ thống mới hoàn toàn được thiết kế theo mô hình phân tán hoặc xây dựng một hệ phân tán dựa trên các tài nguyên địa phương (máy tính, cơ sở dữ liệu) sẵn có.

Một trong các trường hợp điển hình, các hệ phân tán được dùng để quản trị các hệ thống cơ sở dữ liệu lớn. Trong các hệ cơ sở dữ liệu phân tán, tính dư thừa thông tin lại được quan tâm chú ý không chỉ tới khía cạnh gây khó khăn khi tính đến tính nhất quán dữ liệu mà còn tới khía cạnh thuận lợi về vấn đề an toàn: lưu trữ kép (ngoài bản chính còn một số bản sao) để có thể phục hồi khi xảy ra sự cố đối với hệ thống. Để đảm bảo tính nhất quán của hệ thống định kỳ "làm tươi" các thông tin do hệ thống quản lý.

Như đã biết, bài toán lập lịch cho hệ thống chung là phức tạp ngay cả đối với máy tính với một CPU, vì vậy trong các hệ phân tán, bài toán nói trên là hết sức phức tạp (ngay cả các hệ đồng nhất) cho nên người ta thường chọn các phương án đơn giản nhất.

Các nội dung kiến thức về hệ thống phân tán được trình bày chi tiết hơn trong giáo trình chuyên đề" mạng và các hệ phân tán". Bản chất của hệ điều hành trong các mô hình phân tán là một hệ điều hành đa chương trình. Do tính chất không thuần nhất của các máy tính địa phương và có liên quan chặt chẽ đến đường truyền thông, bài toán lập lịch và các hệ thống chương trình điều khiển là phức tạp. Các thuật toán điều khiển được chọn lựa là đủ đơn giản và vẫn luôn là bài toán thời sự đang được nghiên cứu.

Tài liệu tham khảo:

[1]. [Đặng Vũ Hùng], Giáo trình Nguyên lý hệ điều hành. Nhà xuất bản Hà Nội.

[2]. [Nguyễn Thanh Tùng], Hệ điều hành. Khoa Công nghệ thông tin Đại học Bách Khoa Hà Nội, 1996.

[3]. [Nguyễn Mạnh Hùng], [Phạm Tiến Dũng], [Quách Tuấn Ngọc], [Nguyễn Phú Tiến] (biên dịch). Cẩm nang lập trình hệ thống cho máy vi tính IBM-PC tập 1, tác giả Michael Tischer, 1996.

[4]. [Silberschatz A.& Galvin G], Operating systems concepts, John Willey 7 Sons 2002.

[5]. Thư viện Học liệu Mở Việt Nam https://voer.edu.vn