ỦY BAN NHÂN DÂN
THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC SÀI GÒN

Giáo trình

HỆ ĐIỀU HÀNH

Mã số GT2022-04

Tiến sĩ: Phan Tấn Quốc (chủ biên)
Thạc sĩ: Cao Minh Thành
Thạc sĩ: Lương Minh Huấn
Thạc sĩ: Cổ Tồn Minh Đăng

Thành phố Hồ Chí Minh
Ngày 15 tháng 04 năm 2024

Lời cảm ơn

Giáo trình Hệ điều hành, mã số GT2022-04 nhận được nguồn kinh phí tài trợ hoàn toàn từ Trường Đại học Sài Gòn.

Nhóm tác giả trân trọng cảm ơn Ban giám hiệu nhà trường, Ban lãnh đạo khoa Công nghệ thông tin, các nhà khoa học đã tham gia các hội đồng xét duyệt đề cương giáo trình, nghiệm thu giáo trình.

Nhóm tác giả trân trọng cảm ơn các đồng nghiệp khoa Công nghệ thông tin Trường Đại học Sài Gòn đã có nhiều đóng góp quý báu, đã cùng chúng tôi chia sẽ nội dung tập bài giảng, hệ thống các bài tập, các đề thi kết thúc môn học Hệ điều hành trong nhiều năm qua.

Nhóm tác giả trân trọng cảm ơn lãnh đạo và các chuyên viên phòng Quản lý khoa học đã hỗ trợ nhóm tác giả hoàn thành các thủ tục hành chính trong quá trình thực hiện giáo trình này.

Thay mặt các tác giả
Chủ biên

Tiến sĩ. Phan Tấn Quốc

Lời mở đầu

Hệ điều hành là môn học thuộc khối kiến thức cơ sở các ngành học thuộc lĩnh vực Máy tính và Công nghệ thông tin, Toán ứng dụng, Khoa học dữ liệu, Trí tuệ nhân tạo,…

Giáo trình hệ điều hành này được biên soạn để phục vụ sinh viên trường Đại học Sài Gòn trong quá trình học tập và nghiên cứu môn học hệ điều hành. Giáo trình này trình bày ngắn gọn các vấn đề cốt lõi nhất của hệ điều hành bao gồm: chương 1- Tổng quan về hệ điều hành, chương 2- Quản lý tiến trình, chương 3- Quản lý bộ nhớ, chương 4- Quản lý lưu trữ, chương 5- Bảo vệ và an toàn hệ thống và phần Phụ lục có giới thiệu một số đề thi.

Đóng góp chính của chúng tôi trong giáo trình này trình bày các nội dung lý thuyết theo sát nội dung đề cương chi tiết môn học Hệ điều hành chu kỳ đào tạo mới nhất của Trường Đại học Sài Gòn; đề xuất hệ thống ví dụ minh họa, bài tập lý thuyết và thực hành để sinh viên rèn luyện; đề xuất một số đề thi; giáo trình này hỗ trợ sinh viên thuận lợi hơn trong việc ôn tập kiểm tra, thi học kỳ theo đề chung; hỗ trợ giảng viên trong việc giảng dạy và ôn thi.

Các tác giả Phan Tấn Quốc và Cao Minh Thành biên soạn các chương 1, 2, 3; các tác giả Lương Minh Huấn và Cổ Tồn Minh Đăng biên soạn các chương 4, 5 và mục 2.5; phần phụ lục do các tác giả cùng biên soạn.

Chúng tôi trân trọng giới thiệu với bạn đọc các sinh viên Trường Đại học Sài Gòn quyển giáo trình hệ điều hành này và hy vọng rằng nó sẽ giúp cho việc học tập và nghiên cứu môn học hệ điều hành của Trường chúng ta được thuận lợi hơn.

Mặc dù nhóm tác giả đã có nhiều cố gắng, tuy nhiên nội dung giáo trình này cũng không thể tránh khỏi những thiếu sót. Chúng tôi rất mong nhận được ý kiến đóng góp xây dựng của các đồng nghiệp và của bạn đọc để chúng tôi ngày càng hoàn thiện giáo trình này.

Trân trọng cảm ơn.

Thành phố Hồ Chí Minh, ngày 15 tháng 04 năm 2024

Phan Tấn Quốc (chủ biên) –Lương Minh Huấn – Cao Minh Thành – Cổ Tồn Minh Đăng

Mục lục

Danh mục ký hiệu và chữ viết tắt

STT	Ký hiệu chữ viết tắt	Chữ viết đầy đủ
1	CPU	Central Processing Unit
2	MS-DOS	Microsoft Disk Operating System
3	UNIX	UNiplexed Information Computing System
4	VM	Virtual machine
5	VMM	Virtual machine monitor
6	DFS	Distributed File System
7	RTOS	Real-Time Operating System
8	PC	Personal Computer
9	ENIAC	Electronic Numerical Integrator and Compute
10	CTSS	Compatible Time Sharing System
11	MULTICS	Multiplexed Information and Computing Service
12	IC	Integrated Circuit
13	VMS	Virtual Memory System
14	VLSI	Very Large Scale Integration
15	RAM	Random Access Memory
16	ROM	Read Only Memory
17	GUI	Graphical User Interface
18	AWS	Amazon Web Services
19	AI	Artificial Intelligence
20	HDD	Hard Disk Drive
21	SSD	Solid State Drive
22	TFLOPS	Tera FLoating-point Operations Per Second
23	PFLOPS	Peta FLoating-point Operations Per Second
24	PCB	Process Control Block
25	IPC	Inter-Process Communication
26	FCFS	First Come First Served

STT	Ký hiệu chữ viết tắt	Chữ viết đầy đủ
27	SJF	Shortest Job First
28	SRTF	Shortest Remaining Time First
29	RR	Round Robin
30	TCP/IP	Transmission Control Protocol/ Internet Protocol
31	RPC	Remote Producer Call
32	RMI	Remote Method Innovation
33	FIFO	First In First Out
34	CS	Critical Section
35	MMU	Memory – Management Unit
36	LRU	Least Recently Used
37	RPM	Revolutons Per Minute
38	NAS	Network Attached Storage
39	SAN	Storage Area Network
40	DMA	Direct Memory Access
41	DMAC	Direct Memory Access Controller
42	I/O	Input/Output
43	ATA	Advanced Technology Attachment
44	SATA	Serial Advanced Technology Attachment
45	eSATA	extend Serial Advanced Technology Attachment
46	FC	Fibre Channel
47	USB	Universal Serial Bus
48	SLC	Single-level cell
49	MLC	Multi-level cell
50	QLC	Quad-level cell
51	Wi-Fi	Wireless Fidelity
52	LAN	Local Area Network
53	WAN	Wide Area Network
54	IaaS	Infrastructure as a service
55	PaaS	Platform as a service

STT	Ký hiệu chữ viết tắt	Chữ viết đầy đủ
56	SaaS	Software as a Service
57	SSTF	Shortest Seek Time First
58	API	Application Programming Interface
59	NTFS	New Technology File System
60	FAT	File Allocation Table
61	exFAT	Extend File Allocation Table
62	EXT	Extended file system
63	VGA	Video Graphics Array
64	HDMI	High Definition Multimedia Interface
65	DVD	Digital Video Disc
66	PCI	Peripheral Component Interconnect
67	SCSI	Small Computer System Interface
68	SYN	Synchronize
69	MAC	Media Access Control Address
70	IP	Internet Protocol
71	DoS	Denial of Service
72	DDoS	Distributed Denial of Service

Chương 1. Tổng quan về hệ điều hành

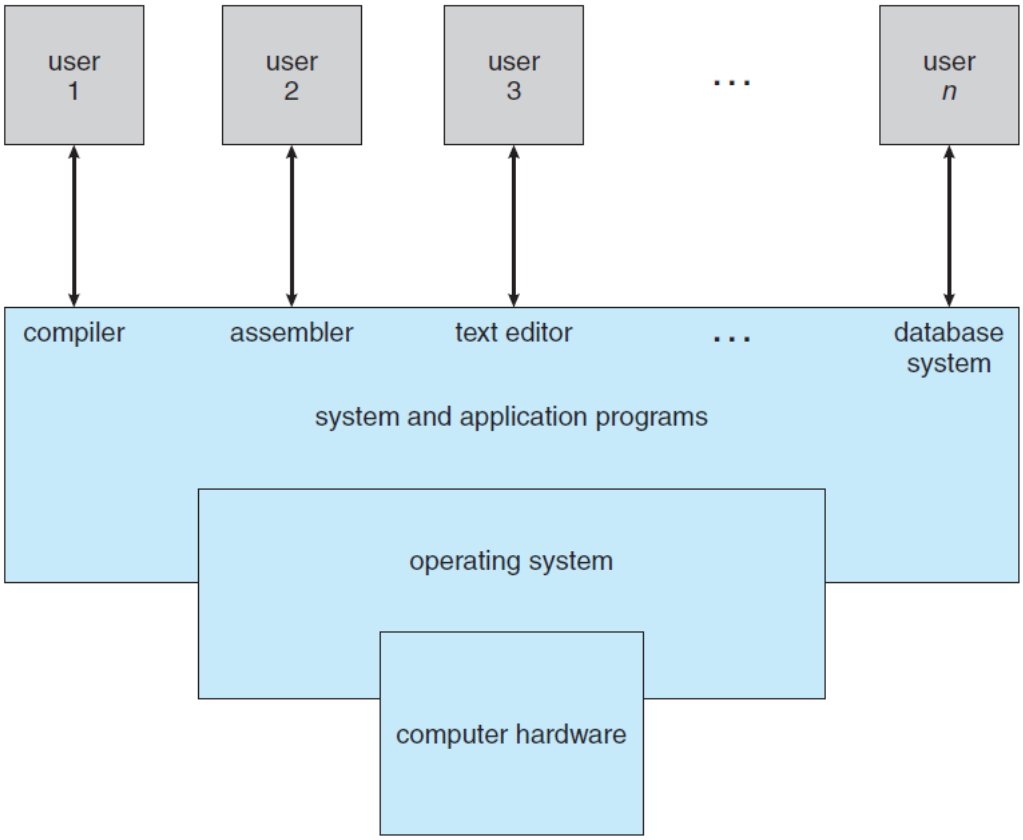
Chương này trình bày các vấn đề: một số khái niệm cơ bản, các thành phần của hệ điều hành, cấu trúc hệ điều hành, phân loại hệ điều hành, lịch sử phát triển các thế hệ máy tính và hệ điều hành.

1.1. Một số khái niệm cơ bản

1.1.1. Cấu trúc hệ thống máy tính

Một hệ thống máy tính gồm phần cứng (hardware), hệ điều hành (operating system), các chương trình ứng dụng (application programs) và người sử dụng/người dùng (users).

- Phần cứng bao gồm các tài nguyên cơ bản của máy tính như CPU, bộ nhớ, các thiết bị nhập/xuất.
- Hệ điều hành điều khiển và phối hợp việc sử dụng phần cứng cho những ứng dụng khác nhau của nhiều người sử dụng khác nhau.
- Chương trình ứng dụng (chương trình dịch, ngôn ngữ lập trình, hệ cơ sở dữ liệu, game,…) sử dụng tài nguyên của máy tính để giải quyết các yêu cầu của người sử dụng.
- Người sử dụng ở đây được hiểu là con người, máy móc hay các máy tính khác.



Hình 1.1. *Các thành phần của một hệ thống máy tính* [3]

1.1.2. Hệ điều hành

Hệ điều hành (operating system) là một chương trình phần mềm quản lý và điều khiển tất cả các tài nguyên và điều khiển hoạt động của máy tính.

Hệ điều hành là lớp trung gian giữa lớp phần cứng (hardware) và lớp ứng dụng (applications), giúp chúng tương tác với nhau một cách hiệu quả.

1.1.3. Chức năng chính của hệ điều hành

Hệ điều hành có hai chức năng chính yếu sau: Quản lý, chia sẻ tài nguyên và giả lặp máy tính mở rộng.

Quản lý, chia sẻ tài nguyên

Tài nguyên hệ thống, đặc biệt là các tài nguyên phần cứng như CPU, bộ nhớ, thiết bị ngoại vi,… bị giới hạn bởi kỹ thuật điện tử. Trong các hệ thống đa nhiệm, nhiều

người sử dụng có thể đồng thời yêu cầu các tài nguyên. Nhằm thỏa mãn nhiều yêu cầu chỉ với một số tài nguyên hữu hạn, cũng như nâng cao hiệu quả sử dụng các tài nguyên, hệ điều hành cần có các cơ chế và chiến lược thích hợp để quản lý việc phân phối tài nguyên.

Bên cạnh đó, nhiều trường hợp người sử dụng cần chia sẻ thông tin (tài nguyên phần mềm) lẫn nhau, khi đó hệ điều hành cần bảo đảm việc truy xuất đến các tài nguyên này là hợp lệ, không để xảy ra các sai lệch có thể gây mất đồng nhất dữ liệu.

Giả lặp một máy tính mở rộng

Hệ điều hành giúp che dấu các chi tiết phần cứng của máy tính và giới thiệu với người dùng một máy tính mở rộng có đầy đủ các chức năng của máy tính thực, nhưng dễ sử dụng hơn.

1.2. Các thành phần của hệ điều hành

Thông thường các hệ điều hành cần có các thành phần sau: Quản lý tiến trình, quản lý bộ nhớ chính, quản lý nhập/xuất, quản lý tập tin, hệ thống bảo vệ, hệ thống dịch lệnh, quản lý mạng.

1.2.1. Quản lý tiến trình

Tiến trình là một chương trình đang thực hiện. Hệ điều hành có nhiệm vụ tạo lập và duy trì hoạt động của các tiến trình. Để hoàn thành các tác vụ, một tiến trình thường đòi hỏi một số tài nguyên nào đó như CPU, bộ nhớ, thiết bị nhập/xuất,…các tài nguyên này sẽ được cấp phát khi tiến trình vào thời điểm được tạo lập hay trong thời gian tiến trình đang hoạt động. Khi tiến trình kết thúc, hệ điều hành cần thu hồi lại các tài nguyên đã cấp phát cho tiến trình để tái sử dụng.

Mỗi tiến trình là một đơn vị tiêu thụ thời gian sử dụng CPU, do vậy trong môi trường đa nhiệm, để đáp ứng nhu cầu xử lý đồng hành, hệ điều hành cần phải đảm nhiệm việc phân phối CPU cho các tiến trình một cách hợp lý. Ngoài ra hệ điều hành cũng cần

cung cấp các cơ chế giúp các tiến trình có thể trao đổi thông tin và đồng bộ hóa hoạt động của chúng.

Tóm lại, bộ phận quản lý tiến trình phụ trách các công việc sau: tạo lập, hủy một tiến trình; tạm dừng, tái kích hoạt một tiến trình; cung cấp các cơ chế trao đổi thông tin liên lạc giữa các tiến trình; cung cấp các cơ chế đồng bộ hóa các tiến trình, cung cấp các cơ chế kiểm soát tắc nghẽn các tiến trình.

Nội dung quản lý tiến trình được đề cập chi tiết trong chương 2 của giáo trình này.

1.2.2. Quản lý bộ nhớ chính

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

Bộ phận quản lý bộ nhớ chính phụ trách các công việc sau: cấp phát và thu hồi vùng nhớ cho tiến trình khi cần thiết; ghi nhận tình trạng bộ nhớ chính: phần nào đã được cấp phát, phần nào còn có thể sử dụng; quyết định tiến trình nào sẽ được nạp vào bộ nhớ chính khi có một vùng nhớ trống.

Nội dung quản lý bộ nhớ chính được đề cập chi tiết trong chương 3 của giáo trình này.

1.2.3. Quản lý nhập/xuất

Một trong những chức năng của hệ điều hành là che dấu các chi tiết phần cứng cụ thể đối với người dùng. Việc điều khiển các thiết bị nhập/xuất là một nhiệm vụ chính của hệ điều hành. Hệ điều hành phải gởi các lệnh điều khiển đến các thiết bị, tiếp nhận các ngắt và xử lý lỗi. Hơn nữa hệ điều hành còn phải cung cấp một giao diện đơn giản và dễ dùng giữa các thiết bị nhập/xuất và phần còn lại của hệ thống; giao diên này cần độc lập với các thiết bị.

Nội dung quản lý nhập/xuất được đề cập chi tiết trong chương 4 của giáo trình này.

1.2.4. Quản lý lưu trữ

Quản lý tập tin

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

Hệ điều hành chịu trách nhiệm về các thao tác liên quan đến tập tin, thư mục sau đây: tạo lập, hủy bỏ tập tin; tạo lập, hủy bỏ thư mục; cung cấp các thao tác xử lý tập tin, thư mục; tạo lập quan hệ tương ứng giữa tập tin và bộ nhớ phụ chứa nó; kiểm soát việc truy cập đồng thời đến cùng một tập tin; phục hồi hệ thống tập tin trên các thiết bị lưu trữ.

Quản lý lưu trữ thứ cấp

Bộ nhớ chính của máy tính thường có dung lượng nhỏ, chỉ đủ chứa chương trình và dữ liệu đang được xử lý; do đó cần có thiết bị khác làm chỗ chứa các chương trình và dữ liệu đã, đang và sẽ xử lý; thiết bị này được gọi là bộ nhớ phụ, nó cần có dung lượng rất lớn. Phương tiện lưu trữ thông dụng là đĩa từ, đĩa quang.

Chương trình được lưu trữ trên bộ nhớ phụ cho tới khi nó được nạp vào bộ nhớ chính và thực hiện sử dụng đĩa để chứa dữ liệu và kết quả xử lý. Có thể sử dụng đĩa để chứa dữ liệu và kết quả xử lý tạm thời (bộ nhớ ảo).

Nhiệm vụ của hệ điều hành trong quản lý lưu trữ thứ cấp là: quản lý vùng trống trên đĩa (free space management), quản lý việc cấp phát vùng lưu trữ theo yêu cầu (storage allocation), quản lý việc lập lịch truy cập đĩa hiệu quả (disk scheduling).

Nội dung quản lý lưu trữ được đề cập chi tiết trong chương 4 của giáo trình này.

1.2.5. Hệ thống bảo vệ

Khi hệ thống cho phép nhiều người sử dụng đồng thời, các tiến trình đồng hành cần phải được bảo vệ lẫn nhau để tránh sự xâm nhập vô tình hay cố ý có thể gây sai lệch toàn bộ hệ thống. Hệ điều hành cần xây dựng các cơ chế bảo vệ cho phép đặc tả sự kiểm soát, và một phương cách để áp dụng các chiến lược bảo vệ thích hợp.

Nội dung quản lý hệ thống bảo vệ được đề cập chi tiết trong chương 5 của giáo trình này.

1.2.6. Hệ thống dịch lệnh

Hệ thống dịch lệnh là một trong những thành phần quan trọng của hệ điều hành, nó đóng vai trò giao diện giữa hệ điều hành và người sử dụng. Các lệnh được chuyển đến hệ điều hành dưới dạng chỉ thị điều khiển. Chương trình - shell – bộ thông dịch lệnh – chỉ có nhiệm vụ đơn giản là nhận lệnh tiếp theo và thông dịch lệnh đó để hệ điều hành có xử lý tương ứng.

1.2.7. Quản lý mạng

Một hệ thống phân bố bao gồm nhiều bộ xử lý không cùng chia sẻ bộ nhớ chung. Mỗi bộ xử lý có một bộ nhớ địa phương độc lập, các tiến trình trong hệ thống có thể được kết nối với nhau thông qua mạng truyền thông.

Hệ điều hành thường xem việc truy xuất đến tài nguyên mạng như một dạng truy xuất tập tin, với các chi tiết kỹ thuật về mạng được chứa đựng trong các trình điều khiển giao tiếp mạng.

1.3. Cấu trúc các hệ điều hành

Một hệ thống lớn và phức tạp như hệ điều hành cần phải được thiết kế kỹ lưỡng để có thể hoạt động đúng và dễ sửa đổi. Hướng tiếp cận hiện nay là phân chia toàn bộ tác vụ của hệ điều hành thành các thành phần nhỏ, một thành phần đảm nhiệm một công

việc chuyên biệt và bao gồm các chức năng được định nghĩa rõ ràng. Cấu trúc của hệ điều hành là cách thức phân chia và kết nối các thành phần này lại với nhau.

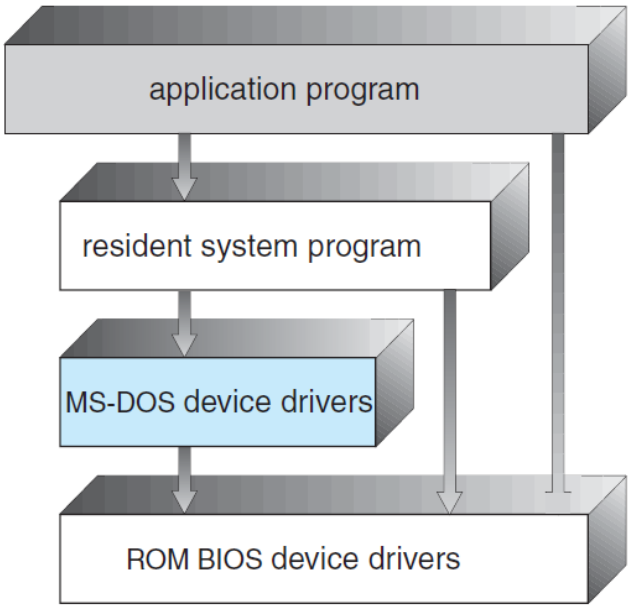
Có nhiều dạng cấu trúc hệ điều hành như: cấu trúc nguyên khối (monolithic structure; còn được gọi là cấu trúc đơn giản), cấu trúc phân lớp (layered structure), cấu trúc máy ảo (virtual machine structure), cấu trúc microkernel, cấu trúc module, cấu trúc hỗn hợp,…

1.3.1. Cấu trúc nguyên khối

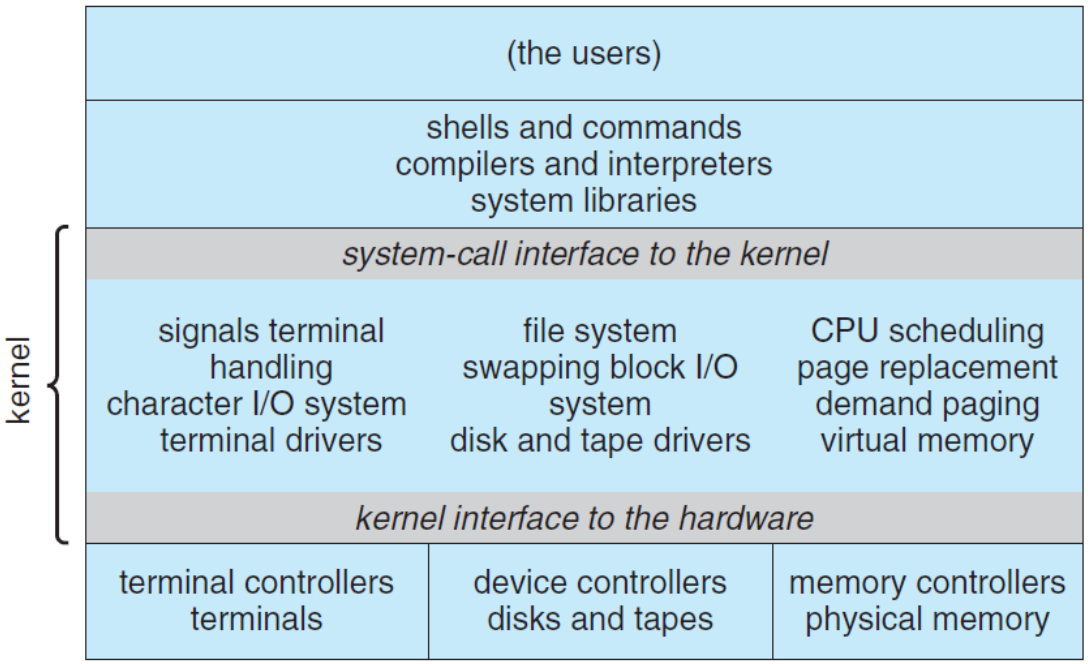
Với cấu trúc nguyên khối (monolithique), hệ điều hành được thiết kế là một tập hợp các thủ tục, có thể gọi lẫn nhau. Một số hệ monolithique cũng có thể có một cấu trúc tối thiểu khi phân chia các thủ tục trong hệ thống thành ba cấp độ:

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

Hệ điều hành MS-DOS, hệ điều hành UNIX phiên bản nguyên thủy thuộc loại cấu trúc nguyên khối này.



Hình 1.2. *Cấu trúc hệ điều hành MS-DOS* [3]



Hình 1.3. *Cấu trúc hệ điều hành UNIX (phiên bản nguyên thủy)* [3]

Ưu điểm:

Hệ điều hành nguyên khối thường cung cấp hiệu suất cao, đặc biệt là trong việc xử lý các lời gọi hệ thống và trao đổi dữ liệu; đơn giản trong thiết kế và triển khai; ổn định và đáng tin cậy; tương thích phần mềm tốt: tối ưu hóa cho phần cứng cụ thể.

Khuyết điểm:

Không có sự che dấu dữ liệu, mỗi thủ tục có thể gọi đến tất cả các thủ tục khác. Các mức độ phân chia thủ tục nếu có cũng không rõ rệt, chương trình ứng dụng có thể truy xuất đến các thủ tục cấp thấp, tác động đến cả phần cứng, do vậy hệ điều hành khó kiểm soát và bảo vệ hệ thống, khó phát triển.

1.3.2. Cấu trúc phân lớp

Có thể đơn thể hóa hệ thống theo nhiều cách, một trong cách tiếp cận đó là cấu trúc hệ điều hành theo kiểu phân lớp (layered structure).

Hệ điều hành được cắt thành một số lớp, mỗi lớp được xây dựng dựa trên các lớp bên trong. Lớp trong cùng (lớp 0 - thường là phần cứng), lớp ngoài cùng (lớp N - thường là giao diện với người sử dụng).

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

Cấu trúc phân lớp lần đầu tiên được áp dụng cho hệ điều hành THE (technische hogeschool eindhoven) được thiết kế vào năm 1968.

Lớp 5: Chương trình ứng dụng

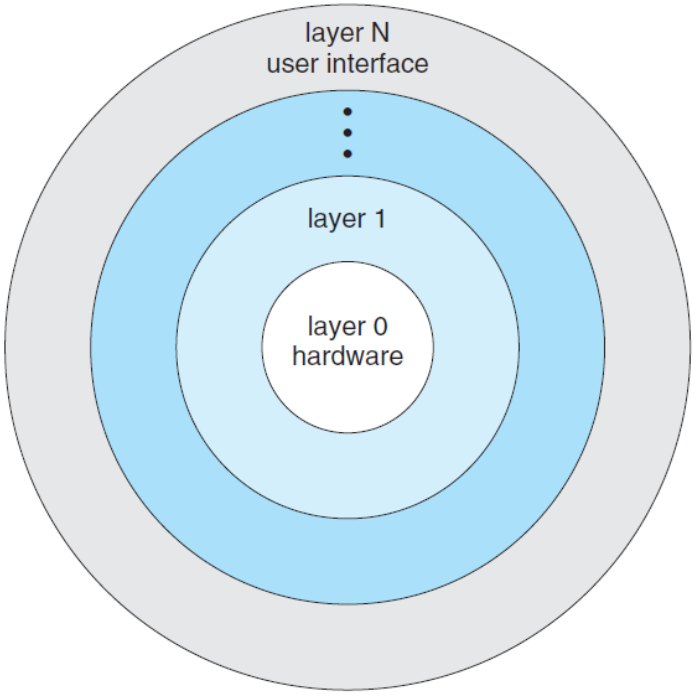
Lớp 4: Quản lý bộ đệm cho hệ thống nhập/xuất

Lớp 3: Trình điều khiển thao tác console

Lớp 2: Quản lý bộ nhớ

Lớp 1: Điều phối CPU

Lớp 0: Phần cứng



Hình 1.4. *Hệ điều hành cấu trúc phân lớp* [3]

Ưu điểm:

Cho phép xây dựng hệ thống mang tính đơn thể, điều này giúp đơn giản hóa việc tìm lỗi và kiểm chứng hệ thống. Nhờ phân chia hệ thống thành các lớp, việc thiết kế và cài đặt trở nên đơn giản hơn. Cấu trúc phân lớp là một trong những cấu trúc phổ biến được sử dụng trong thiết kế hệ điều hành.

Khuyết điểm:

Các hệ theo cấu trúc phân lớp có khuynh hướng hoạt động kém hiệu quả do một lời gọi thủ tục có thể kích hoạt lan truyền các thủ tục khác ở các lớp bên trong, vì thế tổng chi phí để truyền tham số, chuyển đổi ngữ cảnh,…tăng lên. Hậu quả là lời gọi hệ thống trong cấu trúc phân lớp được thực hiện chậm hơn so với các cấu trúc không phân lớp.

1.3.3. Cấu trúc máy ảo

Cấu trúc máy ảo của hệ điều hành thường gồm các thành phần sau:

Hệ điều hành chủ (host operating system): Là hệ điều hành cài đặt trực tiếp trên phần cứng vật lý của máy tính; nó chịu trách nhiệm quản lý tài nguyên phần cứng và cung cấp môi trường để chạy máy ảo.

Hypervisor hoặc Virtual machine monitor (VMM): Là một phần mềm trung gian giữa hệ điều hành chủ và máy ảo; nhiệm vụ chính của nó là quản lý và tạo ra các máy ảo, cũng như phân chia tài nguyên hệ thống giữa chúng.

Máy ảo (virtual machines): Đây là các môi trường máy tính ảo, mỗi cái chứa một hệ điều hành và các ứng dụng tương ứng; các máy ảo này chạy độc lập và cô lập với nhau.

Hệ điều hành khách (guest operating system): Là hệ điều hành chạy bên trong mỗi máy ảo; nó tương tác với tài nguyên ảo được cung cấp bởi máy ảo thay vì tương tác trực tiếp với phần cứng vật lý.

Ứng dụng: Các ứng dụng chạy trên mỗi hệ điều hành khách, tận dụng tài nguyên ảo được cung cấp qua máy ảo.

Tài nguyên ảo: Các tài nguyên như CPU ảo, bộ nhớ ảo, ổ đĩa ảo và thiết bị mạng ảo được cung cấp cho mỗi máy ảo để chúng có thể hoạt động như máy tính độc lập.

Cấu trúc máy ảo trong hệ điều hành giúp tối ưu hóa sử dụng tài nguyên, cung cấp cô lập giữa các máy ảo, và tăng cường linh hoạt trong việc quản lý và triển khai các ứng dụng và hệ điều hành khác nhau.

1.3.4. Cấu trúc Microkernel

Cấu trúc microkernel là một mô hình thiết kế trong đó nhân (kernel) của hệ điều hành chỉ cung cấp các chức năng cơ bản nhất, trong khi các dịch vụ và chức năng hệ thống khác sẽ chạy ở mức người dùng.

Cấu trúc microkernel giúp tăng tính linh hoạt và dễ bảo trì của hệ thống, vì các dịch vụ có thể được thêm vào hoặc gỡ bỏ mà không ảnh hưởng đến kernel chính. Tuy nhiên, cũng có thể xuất hiện hiệu suất thấp hơn so với cấu trúc module hoặc cấu trúc nguyên khối do sự kết nối giữa các dịch vụ.

Cấu trúc microkernel thường mang lại lợi ích về bảo mật, vì mỗi dịch vụ chạy trong không gian người dùng độc lập, giảm khả năng tác động tiêu cực lên kernel.

1.3.5. Cấu trúc module

Với cấu trúc này, hệ điều hành được cấu trúc như một tập hợp các module có thể tải lên và gỡ bỏ một cách linh hoạt.

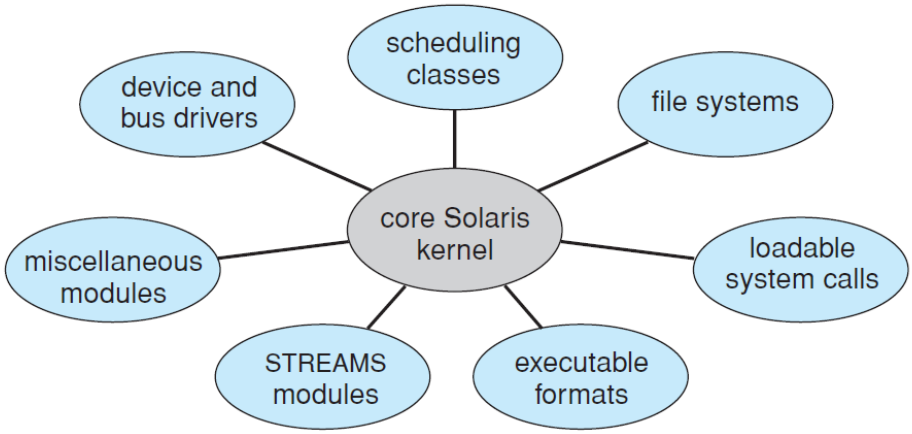
Có thể xem đây là một phương pháp luận tốt để thiết kế các hệ điều hành hiện nay. Kiểu thiết kế này khá phổ biến trong việc triển khai UNIX hiện đại, chẳng hạn như như Solaris, Linux và Mac OS X. Hệ điều hành Windows cũng có cấu trúc thiết kế module và một số đặc điểm của cấu trúc microkernel.

Ưu điểm:

Linh hoạt, dễ mở rộng và cập nhật.

Nhược điểm:

Phụ thuộc vào thiết kế và quản lý module.

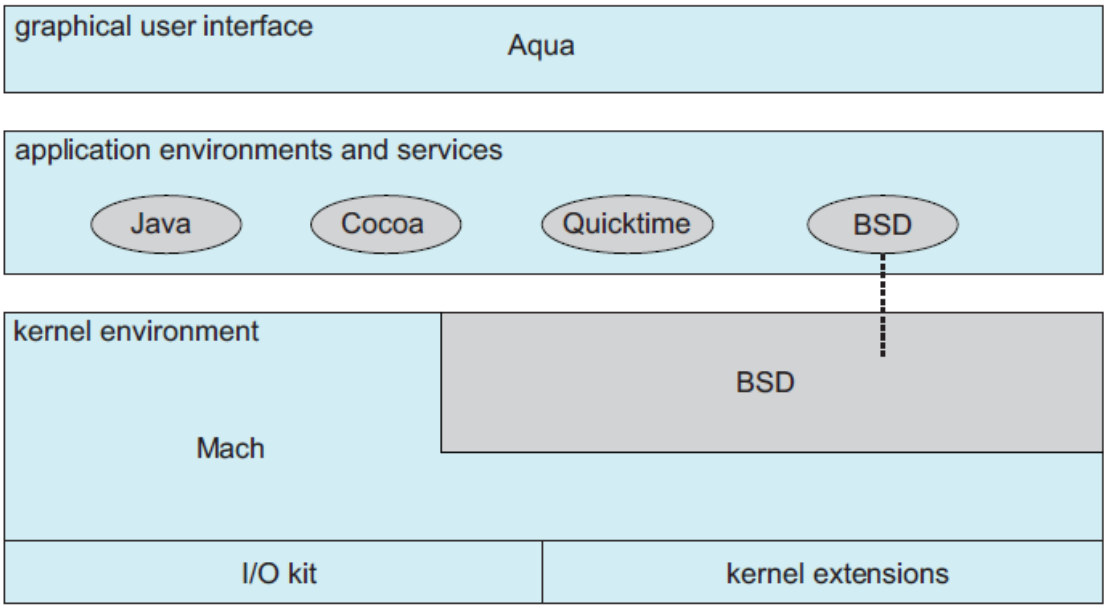


Hình 1.5. *Cấu trúc hệ điều hành Solaris* [3]

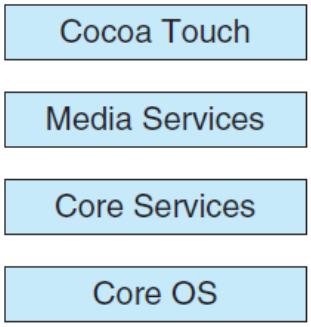
1.3.6. Cấu trúc hỗn hợp

Trong thực tế, rất ít hệ điều hành chỉ áp dụng một cấu trúc duy nhất, mà thường các nhà thiết kế hệ điều hành kết hợp nhiều loại cấu trúc với nhau để tận dụng các ưu điểm đặc trưng của mỗi loại cấu trúc. Tuy nhiên loại cấu trúc hỗn hợp này có thể phức tạp hơn về cả mặt thiết kế và triển khai.

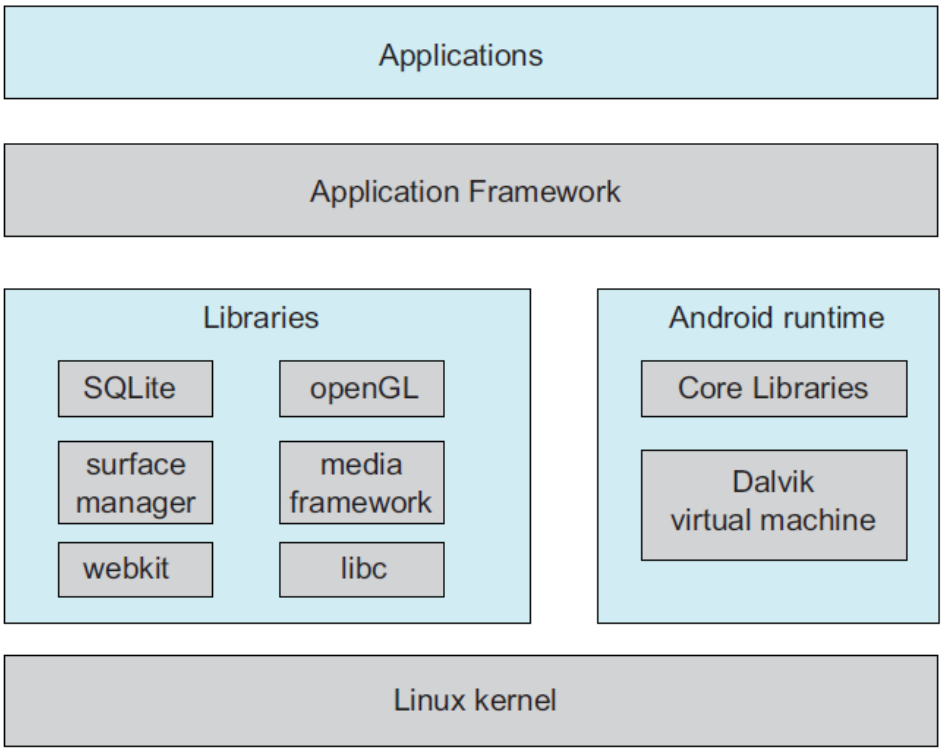
Ví dụ hệ điều hành Linux và Solaris đều có cấu trúc nguyên khối kết hợp với cấu trúc module. Hệ điều hành Apple Mac OS X và hai hệ điều hành di động nổi bật nhất - iOS và Android đều thuộc dạng cấu trúc hỗn hợp.



Hình 1.6. *Cấu trúc hệ điều hành Mac OS X* [3]



Hình 1.7. Cấu trúc hệ điều hành iOS [3]



Hình 1.8. *Cấu trúc hệ điều hành Android* [3]

1.4. Phân loại hệ điều hành

Dưới góc nhìn theo hình thức *xử lý*, có thể phân hệ điều hành gồm các loại sau:

1.4.1. Hệ điều hành xử lý theo lô

Hệ điều hành xử lý theo lô đơn chương (single-program batch operating system) là một loại hệ điều hành đơn giản được thiết kế để xử lý công việc theo lô. Trong mô hình này, hệ thống máy tính chỉ chạy một chương trình tại một thời điểm, từ đầu đến cuối, mà không có sự xen kẽ hoặc chia sẻ thời gian giữa các chương trình khác nhau.

Đặc điểm của hệ điều hành xử lý theo lô đơn chương:

Không có sự đa nhiệm: Máy tính chỉ xử lý một công việc tại một thời điểm. Khi một công việc hoàn thành, hệ thống mới bắt đầu xử lý công việc tiếp theo.

Hiệu quả thấp với người dùng tương tác: Vì máy tính chỉ tập trung vào một công việc duy nhất, người dùng phải chờ đợi cho đến khi công việc hiện tại hoàn thành trước khi công việc của họ được bắt đầu.

Tự động hóa quy trình xử lý: Công việc được tổ chức thành hàng đợi và xử lý tự động mà không cần sự can thiệp thường xuyên của người dùng.

Không cần hệ thống thời gian thực: Do không có yêu cầu về đáp ứng nhanh hay xử lý tương tác, hệ điều hành loại này không cần cung cấp chức năng thời gian thực.

Sử dụng hiệu quả tài nguyên hạn chế: Hệ điều hành này không được thiết kế để tối ưu hóa việc sử dụng tài nguyên máy tính như CPU hay bộ nhớ.

Hệ điều hành xử lý theo lô đơn chương này phổ biến trong thời kỳ đầu của máy tính, khi công nghệ máy tính còn chưa hỗ trợ đa nhiệm hoặc xử lý tương tác. Ngày nay, hệ thống này hầu như ít còn được sử dụng trong các ứng dụng thực tế do hạn chế về khả năng tương tác và hiệu quả.

Hệ điều hành xử lý theo lô đa chương (multiprogramming batch operating system) là một mô hình hệ điều hành trong đó nhiều chương trình (hoặc công việc) được tải và thực thi đồng thời. Mô hình này nhằm tối ưu hóa việc sử dụng của máy tính bằng cách giả định rằng trong quá trình thực thi của một chương trình, các phần khác của hệ thống có thể tiếp tục xử lý các chương trình khác.

Đặc điểm của hệ điều hành xử lý theo lô đa chương:

Đa nhiệm (multiprogramming): Nhiều chương trình có thể được tải và chạy đồng thời trên máy tính. Mục tiêu là giữ CPU hoạt động liên tục, giảm thời gian rỗi và tăng hiệu suất.

Lập lịch (scheduling): Hệ thống cần có các thuật toán lập lịch để quyết định chương trình nào được thực thi và khi nào.

Quản lý bộ nhớ: Hệ thống phải quản lý bộ nhớ hiệu quả để chứa nhiều chương trình.

Input/Output: Hệ điều hành quản lý việc chuyển đổi giữa các chương trình và các thiết bị ngoại vi để giữ cho CPU hạn chế ở trạng thái rỗi.

Tính hiệu quả và sử dụng tài nguyên: Mục tiêu là tối đa hóa hiệu suất toàn diện của hệ thống và sử dụng tài nguyên máy tính một cách hiệu quả.

Hệ điều hành xử lý theo lô đa chương là một bước tiến so với mô hình xử lý theo lô đơn chương, mang lại khả năng đa nhiệm và tận dụng tốt hơn sức mạnh của máy tính. Điều này thường được thấy trong các hệ thống máy tính đa người dùng hoặc môi trường đa nhiệm.

1.4.2. Hệ điều hành chia sẻ thời gian

Hệ điều hành chia sẻ thời gian (time-sharing operating system) là một loại hệ điều hành được thiết kế để cho phép nhiều người dùng truy cập vào một máy tính đơn lẻ cùng một lúc. Ý tưởng chính ở đây là chia sẻ tài nguyên máy tính, chẳng hạn như CPU, bộ nhớ, và thiết bị lưu trữ, giữa nhiều người dùng một cách hiệu quả và công bằng.

Đặc điểm của hệ điều hành chia sẻ thời gian:

Đa nhiệm (multitasking): Hệ thống cho phép nhiều chương trình hoạt động đồng thời. CPU chuyển đổi nhanh chóng giữa các chương trình, tạo ra ảo giác rằng mỗi chương trình đang có quyền truy cập đến CPU một cách liên tục.

Đa người dùng (multiuser): Nhiều người dùng có thể truy cập vào hệ thống từ các thiết bị khác nhau tại cùng một thời điểm.

Chia sẻ thời gian (time sharing): Hệ thống sử dụng thuật toán lập lịch để phân chia thời gian CPU giữa các người dùng và các chương trình. Mỗi người dùng hoặc chương trình nhận được một khoảng thời gian nhất định để sử dụng CPU, sau đó hệ thống chuyển sang người dùng hoặc chương trình tiếp theo.

Quản lý tài nguyên hiệu quả: Hệ điều hành phải quản lý tài nguyên một cách thông minh để đảm bảo rằng mọi người dùng và chương trình đều nhận được phần tài nguyên cần thiết cho hoạt động của họ mà không làm chậm toàn hệ thống.

Tương tác người dùng: Hệ thống thường hỗ trợ tương tác thời gian thực hoặc gần như thời gian thực giữa người dùng và các ứng dụng.

Các hệ điều hành chia sẻ thời gian hiện đại bao gồm các hệ điều hành như UNIX và các phiên bản của nó, Linux, và thậm chí cả Windows trong một số môi trường làm việc đa người dùng. Hệ thống này giúp tối ưu hóa hiệu suất máy tính khi phục vụ nhiều

người dùng cùng một lúc và là một phần quan trọng trong lịch sử phát triển của máy tính và hệ điều hành.

1.4.3. Hệ điều hành xử lý song song

Hệ điều hành xử lý song song (parallel processing operating system) là một loại hệ điều hành được thiết kế để quản lý và tận dụng các tài nguyên máy tính có khả năng xử lý song song. Trong môi trường xử lý song song, nhiều tác vụ có thể được thực hiện đồng thời, giúp tăng cường hiệu suất và sử dụng hiệu quả các nguồn lực.

Đặc điểm hệ điều hành xử lý song song:

Quản lý nhiều tiến trình: Hệ điều hành này cần có khả năng quản lý nhiều tiến trình chạy đồng thời trên nhiều CPU.

Lập lịch song song: Hệ điều hành xử lý song song cần cung cấp các thuật toán lập lịch hiệu quả để phân phối công việc giữa các CPU.

Giao tiếp và đồng bộ hóa: Cần các cơ chế giao tiếp và đồng bộ hóa giữa các tiến trình để tránh xung đột dữ liệu và đảm bảo tính đúng đắn của các kết quả.

Phân tán tài nguyên: Hệ điều hành cần phân phối và quản lý tài nguyên như bộ nhớ, I/O, và các nguồn lực hệ thống khác một cách hiệu quả giữa các tiến trình.

Tính năng mở rộng: Có khả năng mở rộng để hỗ trợ thêm CPU mà không làm ảnh hưởng đến hiệu suất toàn bộ hệ thống.

Xử lý lỗi đồng thời: Hệ điều hành cần xử lý lỗi một cách hiệu quả khi có sự cố xảy ra trong quá trình thực hiện các công việc song song.

Hệ điều hành xử lý song song thường được sử dụng trong các hệ thống có nhiều CPU, máy tính đa nhiệm và trong các môi trường có nhu cầu xử lý tác vụ lớn và phức tạp. Các ví dụ bao gồm Linux, Windows server, và các hệ điều hành dành cho siêu máy tính và hệ thống phân tán.

1.4.4. Hệ điều hành phân tán

Hệ điều hành xử lý phân tán (distributed operating system) là một loại hệ điều hành được thiết kế để quản lý và điều khiển tài nguyên trên nhiều máy tính và thiết bị

trong một môi trường phân tán. Trong hệ điều hành xử lý phân tán, các tài nguyên như bộ nhớ, lưu trữ, và xử lý có thể phân tán trên nhiều nút (node) trong mạng, và hệ điều hành này cung cấp các cơ chế để quản lý sự tương tác giữa các nút và tài nguyên.

Đặc điểm hệ điều hành xử lý phân tán:

Quản lý tài nguyên phân tán: Hệ điều hành này phải có khả năng quản lý và phân phối tài nguyên trên các máy tính và thiết bị khác nhau trong mạng.

Giao tiếp và đồng bộ hóa: Các cơ chế giao tiếp và đồng bộ hóa giữa các nút cần được quản lý để đảm bảo tính nhất quán và đúng đắn của hệ thống.

Quản lý truy cập đồng thời: Với nhiều người dùng hoặc ứng dụng truy cập cùng một tài nguyên, hệ điều hành xử lý phân tán phải có khả năng quản lý truy cập đồng thời để tránh xung đột dữ liệu.

Bảo mật: Cần có các cơ chế bảo mật để đảm bảo an toàn và bảo mật thông tin khi được truyền qua mạng.

Quản lý sự cố và khôi phục: Hệ điều hành xử lý phân tán cần có khả năng xử lý sự cố và khôi phục một cách hiệu quả, đặc biệt khi một số nút trong mạng gặp sự cố.

Quản lý độ trễ và băng thông: Đối với các ứng dụng đòi hỏi độ trễ thấp và lượng dữ liệu lớn, hệ điều hành xử lý phân tán cần có khả năng quản lý độ trễ và băng thông mạng.

Các ví dụ về hệ điều hành xử lý phân tán bao gồm Google's Chrome OS, Microsoft's Windows Distributed File System (DFS), và Unix-like Distributed Systems. Hệ điều hành xử lý phân tán thường được sử dụng trong các hệ thống máy chủ lớn, môi trường điện toán đám mây, và các ứng dụng yêu cầu khả năng mở rộng và sự phân tán.

1.4.5. Hệ điều hành thời gian thực

Hệ điều hành xử lý thời gian thực (real-time operating system -RTOS) là một loại hệ điều hành được thiết kế để đáp ứng các yêu cầu thời gian thực và đảm bảo rằng các nhiệm vụ và phản ứng xử lý được thực hiện trong khoảng thời gian cố định và đáng tin cậy. Hệ điều hành xử lý thời gian thực thường được sử dụng trong các hệ thống yêu cầu

độ chính xác và độ tin cậy cao trong việc xử lý dữ liệu và điều khiển các thiết bị và quy trình thực tế.

Đặc điểm hệ điều hành xử lý thời gian thực:

Đảm bảo thời gian phản hồi (guaranteed response time): Hệ điều hành xử lý thời gian thực đảm bảo rằng mọi phản ứng của hệ thống đều xảy ra trong một khoảng thời gian cố định và được dự đoán được.

Lập lịch thời gian thực (real-time scheduling): Hệ điều hành này sử dụng các thuật toán lập lịch được thiết kế đặc biệt để đảm bảo rằng các tác vụ quan trọng được thực hiện đúng thời điểm.

Khả năng dự đoán (predictability): Các hệ điều hành xử lý thời gian thực tập trung vào việc đảm bảo rằng thời gian thực hiện các tác vụ là dự đoán được, tránh gặp các độ trễ không mong muốn.

Độ tin cậy cao: Hệ điều hành này cần đảm bảo tính tin cậy cao trong việc thực hiện các nhiệm vụ và xử lý sự cố.

Quản lý nhiệm vụ ưu tiên: Các hệ điều hành xử lý thời gian thực thường quản lý các nhiệm vụ dựa trên độ ưu tiên, đảm bảo rằng các tác vụ ưu tiên cao được xử lý trước.

Ứng dụng của hệ điều hành xử lý thời gian thực rất phong phú và bao gồm các lĩnh vực như điều khiển ô tô, thiết bị y tế, dòng chuyền sản xuất, hệ thống nhúng, và các hệ thống truyền thông.

1.4.6. Các góc nhìn khác để phân loại hệ điều hành

Ngoài góc nhìn *dựa theo góc độ hình thức xử lý ở* trên thì có thể phân loại hệ điều hành dựa trên các góc nhìn khác nữa; chẳng hạn:

Dựa theo góc độ loại máy tính

Hệ điều hành dành cho máy tính cá nhân (personal computer-PC), hệ điều hành dành cho máy nhiều CPU, hệ điều hành dành cho máy chủ (Server), hệ điều hành dành cho máy MainFrame, hệ điều hành dành cho máy chuyên biệt,…

Dựa theo góc độ số chương trình được sử dụng cùng lúc

Hệ điều hành đơn nhiệm

Hệ điều hành đơn nhiệm (single-tasking operating system) là loại hệ điều hành chỉ cho phép thực hiện một công việc duy nhất tại một thời điểm. Người dùng chỉ có thể thực hiện một ứng dụng hoặc chương trình tại một thời điểm, và để chuyển sang công việc khác, họ phải đóng ứng dụng hiện tại và mở ứng dụng mới. Hệ điều hành MS-DOS là một ví dụ điển hình của hệ điều hành đơn nhiệm.

Hệ điều hành đa nhiệm

Hệ điều hành đa nhiệm (multi-tasking operating system) là loại hệ điều hành cho phép thực hiện nhiều công việc cùng một lúc trên một hệ thống. Người dùng có thể mở và chạy nhiều ứng dụng hay chương trình đồng thời, và chuyển đổi giữa chúng mà không cần đóng lại ứng dụng hiện tại. Hệ điều hành Windows, macOS, và nhiều phiên bản của Linux là những ví dụ của hệ điều hành đa nhiệm, cung cấp trải nghiệm đa nhiệm linh hoạt và hiệu quả.

Dựa theo góc độ số lượng người sử dụng

Hệ điều hành một người dùng

Hệ điều hành một người dùng (single-user operating system) là loại hệ điều hành được thiết kế để phục vụ một người dùng duy nhất tại một thời điểm. Trong môi trường này, máy tính được cấu hình để tương tác với và phục vụ nhu cầu của một người dùng cụ thể. Hệ điều hành này chủ yếu được sử dụng trên máy tính cá nhân và các thiết bị di động, nơi một người dùng có toàn quyền kiểm soát và sử dụng tài nguyên hệ thống.

Hệ điều hành nhiều người dùng

Hệ điều hành nhiều người dùng (multi-user operating system) là loại hệ điều hành có khả năng đồng thời phục vụ và hỗ trợ nhiều người dùng truy cập và sử dụng hệ thống. Trong môi trường này, nhiều người dùng có thể đăng nhập và thực hiện các nhiệm vụ riêng biệt mà không ảnh hưởng đến các người dùng khác. Hệ điều hành này thường được sử dụng trong các hệ thống máy chủ và môi trường mạng, nơi nhiều người cần truy cập và chia sẻ tài nguyên hệ thống cùng một lúc. Ví dụ của hệ điều hành nhiều người dùng là UNIX và các hệ điều hành server chuyên dụng.

1.5. Lịch sử phát triển hệ điều hành

Hệ điều hành nằm trên lớp cấp phần cứng nên lịch sử phát triển của hệ điều hành gắn liền với lịch sử phát triển của máy tính. Hơn thế nữa, sự phát triển của hệ điều hành kéo theo sự phát triển của máy tính. Mục này trình bày lịch sử phát triển của các thế hệ máy tính và hệ điều hành điển hình tương ứng với từng thế hệ máy tính này.

1.5.1. Thế hệ máy tính đầu tiên

Thế hệ máy tính đầu tiên: 1940-1956

Máy tính: Sử dụng ống chân không và thiết bị cơ khí.

Hệ điều hành: Không có hệ điều hành chính thức. Người điều khiển máy tính trực tiếp bằng mã máy hoặc các công cụ nguyên thủy.

Thế hệ máy tính thứ nhất đánh dấu sự xuất hiện và phát triển của các máy tính điện tử. Các máy tính trong thời kỳ này thường được thiết kế và xây dựng dựa trên các ống chân không và thiết bị cơ khí.

Đặc điểm thế hệ máy tính đầu tiên

Cơ bản về cấu trúc: Sử dụng ống chân không và thiết bị cơ khí. Các máy tính thế hệ này thường sử dụng ống chân không cho việc lưu trữ và xử lý dữ liệu. Relay điện tử cũng được sử dụng để thực hiện các chức năng logic và điều khiển.

Kích thước và tính năng: Các máy tính trong thế hệ này thường có kích thước lớn, đòi hỏi không gian lớn để đặt và làm việc.

Khả năng xử lý thấp: So với máy tính hiện đại, chúng có khả năng xử lý thấp và thường chỉ thực hiện một nhiệm vụ cụ thể mỗi lần.

Ngôn ngữ lập trình và giao diện: Dữ liệu thường được nhập vào máy tính bằng cách sử dụng các dây đan dữ liệu được đọc bởi máy đọc dây.

Lập trình bằng mã máy: Ngôn ngữ lập trình ở thời kỳ này thường là mã máy, điều này yêu cầu các lập trình viên là những chuyên gia có kiến thức sâu về cấu trúc máy tính cụ thể.

Ứng dụng chính: Các máy tính thế hệ đầu tiên thường được phát triển để phục vụ cho an ninh quốc phòng và nghiên cứu khoa học.

Ví dụ về máy tính thế hệ đầu tiên: ENIAC (electronic numerical integrator and computer): ENIAC, hoàn thành vào năm 1945, là một trong những máy tính đầu tiên của thế giới. Nó sử dụng 17468 ống chân không và có khả năng thực hiện hàng nghìn phép toán mỗi giây.

Thách thức và hạn chế: Các máy tính thế hệ này thường có thiết kế phức tạp và dễ hỏng hóc. Việc duy trì và sửa chữa mất nhiều thời gian.

Chuyển đổi từ số học sang khoa học máy tính: Trong giai đoạn này, máy tính đã chuyển từ việc chỉ xử lý số học đến việc xử lý các dạng thông tin khác nhau, bao gồm cả ký tự và ký hiệu.

Sự phát triển của ngành công nghiệp máy tính: Trong giai đoạn này, các công ty máy tính đầu tiên xuất hiện, tạo nền tảng cho sự phát triển của ngành công nghiệp máy tính.

Thế hệ máy tính thứ nhất đã đặt nền móng cho sự phát triển của công nghiệp máy tính và đã mở ra cánh cửa cho các thế hệ máy tính tiếp theo, với sự tiến bộ vượt bậc trong công nghệ và hiệu suất tính toán.

1.5.2. Thế hệ máy tính thứ hai

Thế hệ máy tính thứ hai: 1956-1963

Máy tính: Sử dụng transistor thay thế cho ống chân không.

Hệ điều hành: Các hệ điều hành đầu tiên xuất hiện, chẳng hạn như IBSYS được phát triển bởi IBM và CTSS (Compatible Time Sharing System) được phát triển bởi MIT. CTSS là một trong những hệ điều hành chia sẻ thời gian đầu tiên, cho phép nhiều người dùng cùng lúc sử dụng máy tính.

Thế hệ máy tính thứ hai đánh dấu một bước tiến quan trọng trong lịch sử phát triển máy tính, chủ yếu nhờ vào sự xuất hiện của transistor.

Đặc điểm thế hệ máy tính thứ hai

Sử dụng transistor: Các máy tính thế hệ thứ hai thay thế ống chân không bằng transistor, giúp chúng nhỏ gọn, tin cậy và hiệu quả hơn.

Tốc độ và hiệu suất: Sự thay thế này làm tăng đáng kể tốc độ và hiệu suất xử lý, đồng thời giảm nhiệt độ và tiêu thụ điện năng.

Lưu trữ và xử lý: Máy tính thế hệ thứ hai sử dụng bộ nhớ lõi từ (magnetic core memory), cho phép lưu trữ dữ liệu và chương trình hiệu quả hơn.

Ngôn ngữ lập trình cao cấp: Thời kỳ này chứng kiến sự phát triển của các ngôn ngữ lập trình cấp cao như FORTRAN (1957), ALGOL (1958), và COBOL (1959).

Ứng Dụng: Máy tính được sử dụng rộng rãi hơn trong các lĩnh vực như doanh nghiệp, khoa học và giáo dục.

Thế hệ máy tính thứ hai là một bước tiến đáng kể so với thế hệ đầu tiên. Việc áp dụng transistor đã giúp máy tính trở nên nhỏ gọn, hiệu quả và mạnh mẽ hơn. Đồng thời, sự ra đời của các ngôn ngữ lập trình cấp cao và hệ điều hành tiên tiến đã mở rộng khả năng và ứng dụng của máy tính, từ đó đặt nền móng cho sự phát triển nhanh chóng của công nghệ máy tính trong các thập kỷ tiếp theo.

1.5.3. Thế hệ máy tính thứ ba

Thế hệ máy tính thứ ba: 1964-1971

Máy tính: Sự xuất hiện của mạch tích hợp.

Hệ điều hành: OS/360, MULTICS (multiplexed information and computing service), UNIX; trong đó UNIX đặt nền móng cho nhiều hệ điều hành hiện đại.

Thế hệ máy tính thứ ba là một giai đoạn quan trọng trong lịch sử phát triển của máy tính, với sự ra đời của mạch tích hợp (IC - Integrated Circuit). Các mạch tích hợp cho phép nhiều thành phần điện tử được tích hợp trên một chip silicon nhỏ, mở đường cho sự xuất hiện của máy tính nhỏ gọn, nhanh chóng và hiệu quả hơn.

Đặc điểm thế hệ máy tính thứ ba

Thay vì sử dụng transistor riêng lẻ, máy tính thế hệ thứ ba sử dụng mạch tích hợp, tăng cường khả năng xử lý và giảm kích thước của máy.

Hiệu suất cao và chi phí thấp: Máy tính trở nên nhỏ gọn, tiêu thụ ít năng lượng hơn, có hiệu suất cao và chi phí sản xuất giảm.

Bộ nhớ chính và bộ nhớ ngoại vi: Sử dụng bộ nhớ lõi từ và bắt đầu sử dụng các thiết bị lưu trữ như đĩa cứng.

Ngôn ngữ lập trình và công cụ phát triển: Phát triển nhiều ngôn ngữ lập trình mới và công cụ phát triển phần mềm, bao gồm BASIC và PL/I.

Ứng dụng rộng rãi: Máy tính bắt đầu được sử dụng rộng rãi trong kinh doanh, chính phủ và giáo dục.

Thế hệ máy tính thứ ba là một bước ngoặt quan trọng, đánh dấu sự chuyển từ công nghệ dựa trên các thành phần riêng lẻ sang công nghệ dựa trên mạch tích hợp. Điều này không chỉ giúp cải thiện đáng kể hiệu suất và giảm kích thước của máy tính mà còn làm giảm chi phí sản xuất, làm cho máy tính trở nên phổ biến hơn trong nhiều lĩnh vực.

1.5.4. Thế hệ máy tính thứ tư

Thế hệ máy tính thứ tư: 1971-1980s

Máy tính: Xuất hiện của vi mạch (microprocessor).

Hệ điều hành: UNIX, MS-DOS, CP/M, Apple DOS và MacOS, VMS (Virtual Memory System).

Thế hệ máy tính thứ tư chứng kiến sự xuất hiện của vi mạch tích hợp (VLSI - Very Large Scale Integration) và máy tính cá nhân. Các máy tính trở nên nhỏ gọn, mạnh mẽ hơn và giá cả phải chăng. Điều này mở ra thời đại của máy tính cá nhân và công nghệ thông tin.

Đặc điểm thế hệ máy tính thứ tư

Sự phát triển của vi mạch tích hợp giúp tích hợp hàng nghìn, thậm chí hàng triệu thành phần điện tử trên một chip nhỏ. Điều này làm tăng đáng kể hiệu suất và giảm chi phí.

Máy tính cá nhân: Xuất hiện của máy tính cá nhân, đặc biệt là sau khi Intel giới thiệu vi xử lý Intel 4004 (1971) và 8080 (1974).

Bộ Nhớ RAM và ROM: Sự phát triển của bộ nhớ RAM và ROM đã cải thiện khả năng xử lý dữ liệu và lưu trữ chương trình.

Ổ đĩa cứng và thiết bị lưu trữ: Sự xuất hiện của ổ đĩa cứng (hard drive) và các thiết bị lưu trữ khác giúp nâng cao khả năng lưu trữ và truy cập dữ liệu.

Mạng máy tính: Sự phát triển của mạng máy tính đã mở ra khả năng kết nối và chia sẻ thông tin giữa các máy tính.

Công nghệ màn hình và giao diện người dùng: Màn hình đồ họa và giao diện người dùng trực quan (GUI) đã xuất hiện, giúp người dùng tương tác với máy tính một cách dễ dàng hơn.

Ngôn ngữ lập trình cấp cao: Sự phát triển của ngôn ngữ lập trình cấp cao như C và Pascal đã giúp việc lập trình trở nên đơn giản và hiệu quả.

Thế hệ máy tính thứ tư chứng kiến sự đổi mới đáng kể với sự ra đời của máy tính cá nhân và công nghệ thông tin phổ cập. Vi mạch tích hợp, máy tính cá nhân, và các hệ điều hành tiêu biểu của thời kỳ này đã mở ra cánh cửa cho cuộc cách mạng số với ảnh hưởng to lớn đến cuộc sống hàng ngày, công việc, và giao tiếp.

1.5.5. Thế hệ máy tính thứ năm

Thế hệ máy tính thứ năm: 1980s-đến nay

Máy tính: Máy tính cá nhân, máy tính hiện tại và tương lai: siêu máy tính, máy tính lượng tử.

Hệ điều hành: Windows, UNIX và Linux, MacOS, iOS và Android; hệ điều hành đám mây như AWS, Azure và Google Cloud Platform, hỗ trợ cho việc xử lý và lưu trữ dữ liệu trên môi trường đám mây; các hệ điều hành chuyên dụng cho các siêu máy tính, máy tính lượng tử.

Thế hệ máy tính thứ năm đánh dấu một bước tiến lớn trong công nghệ máy tính. Giai đoạn này chứng kiến sự phát triển vượt bậc về công nghệ phần cứng và phần mềm, cũng như sự xuất hiện của mạng Internet và công nghệ thông tin di động.

Đặc điểm thế hệ máy tính thứ năm

Phần cứng đa dạng: Sự phát triển của chip vi xử lý với hiệu suất cao, điều này giúp máy tính cá nhân (PC) và máy chủ (server) trở nên mạnh mẽ và đa dạng hơn. Sự ra đời của các thiết bị di động như điện thoại thông minh và máy tính bảng, với vi xử lý chuyên dụng.

Phần mềm tiên tiến và đa dạng: Sự phát triển mạnh mẽ của hệ thống quản lý cơ sở dữ liệu, ứng dụng văn phòng, phần mềm đồ họa và trò chơi điện tử. Sự xuất hiện của trí tuệ nhân tạo (AI) và học máy trong các ứng dụng thực tế.

Mạng và Internet: Internet trở thành nền tảng thông tin toàn cầu, kết nối hàng tỷ thiết bị và người dùng. Sự phát triển của công nghệ mạng không dây và di động.

Lưu trữ và xử lý dữ liệu lớn: Sự phát triển của công nghệ lưu trữ như SSD và các giải pháp lưu trữ đám mây. Sự tăng trưởng của dữ liệu lớn và công nghệ xử lý dữ liệu phân tán.

Thế hệ máy tính thứ năm chứng kiến sự chuyển mình mạnh mẽ của công nghệ thông tin, từ máy tính cá nhân đến di động và đám mây. Sự phát triển của hệ điều hành, từ những phiên bản đầu tiên cho máy tính cá nhân đến các hệ thống di động và đám mây, đã làm thay đổi cách chúng ta tương tác với công nghệ và xử lý thông tin. Công nghệ máy tính ngày càng trở nên tích hợp hơn vào cuộc sống hàng ngày, và hệ điều hành đóng vai trò quan trọng trong việc kết nối và quản lý tất cả các khía cạnh của thế giới số hóa.

Siêu máy tính

Siêu máy tính là một loại máy tính với khả năng xử lý dữ liệu ở quy mô lớn và tốc độ cao hơn đáng kể so với máy tính thông thường. Các siêu máy tính được thiết kế để giải quyết các vấn đề phức tạp trong lĩnh vực nghiên cứu khoa học, mô phỏng, và tính toán số.

Đặc điểm siêu máy tính

Hiệu suất cao: Siêu máy tính có khả năng xử lý hàng trăm nghìn đến hàng triệu tỷ phép toán mỗi giây (TFLOPS), hoặc thậm chí là péta phép toán mỗi giây (PFLOPS).

Kiến trúc đa nhiệm: Các siêu máy tính thường được thiết kế để xử lý đa nhiệm, đồng thời giải quyết nhiều vấn đề phức tạp.

Sử dụng trong nghiên cứu khoa học: Siêu máy tính thường được sử dụng trong các lĩnh vực như dự báo thời tiết, mô phỏng vật lý hạt nhân, nghiên cứu y học, và các ứng dụng nghiên cứu khoa học khác.

Kiến trúc số hóa đặc biệt: Để đáp ứng nhu cầu tính toán độ chính xác cao, siêu máy tính thường sử dụng kiến trúc số hóa chuyên biệt.

Kích thước lớn và yêu cầu năng lượng cao: Siêu máy tính thường có kích thước lớn và yêu cầu năng lượng lớn, cũng như hệ thống làm mát phức tạp để giữ nhiệt độ ổn định.

Máy tính lượng tử

Máy tính lượng tử hoạt động dựa trên hoạt động của cơ học lượng tử. Máy tính lượng tử là máy tính hoạt động trên cơ sở của những hạt lượng tử. Máy tính lượng tử có tốc độ nhanh gấp hàng triệu lần so với siêu máy tính nhanh nhất thế giới hiện nay. Hiện máy tính lượng tử đang được nghiên cứu và phát triển với hy vọng mở ra một kỷ nguyên mới trong tính toán.

Hiện tại các máy tính lượng tử chưa thể thay thế máy tính thông thường mà chúng chỉ có thể hoạt động trong môi trường chuyên biệt, xử lý nhiệm vụ có tính chuyên môn cao như: hóa học lượng tử, tối ưu hóa, học máy, mã hóa, hệ thống tài chính phức tạp,…

Bài tập chương 1

Bài tập 1.1.

Trình bày khái niệm hệ điều hành. Trình bày chức năng hệ điều hành.

Bài tập 1.2.

Cài đặt, tìm hiểu các nhóm câu lệnh của hệ điều hành Window 11 (thực hành).

Bài tập 1.3.

Cài đặt, tìm hiểu các nhóm câu lệnh của hệ điều hành mã nguồn mở Linux/Ubuntu (thực hành).

Bài tập 1.4.

Tạo một máy tính ảo (thực hành).

Bài tập 1.5.

Khảo sát một số hệ điều hành được dùng cho điện thoại di động.

Bài tập 1.6.

Khảo sát một số máy tính thế hệ mới (siêu máy tính, máy tính lượng tử).

Bài tập 1.7.

Khảo sát một số hệ điều hành mã nguồn mở.

Bài tập 1.8.

Khảo sát một số hệ điều hành trong hệ thống phân tán.

Bài tập 1.9.

Khảo sát một số hệ điều hành trong các hệ thống nhúng.

Chương 2. Quản lý tiến trình

2.1. Tiến trình

2.1.1. Mô hình tiến trình

Chương trình máy tính chứa các chỉ thị điều khiển máy tính thực hiện một tác vụ nào đó; chương trình là một thực thể thụ động.

Tiến trình (process) là một chương trình đang thực hiện; tiến trình là một thực thể hoạt động.

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

2.1.2. Các trạng thái của tiến trình

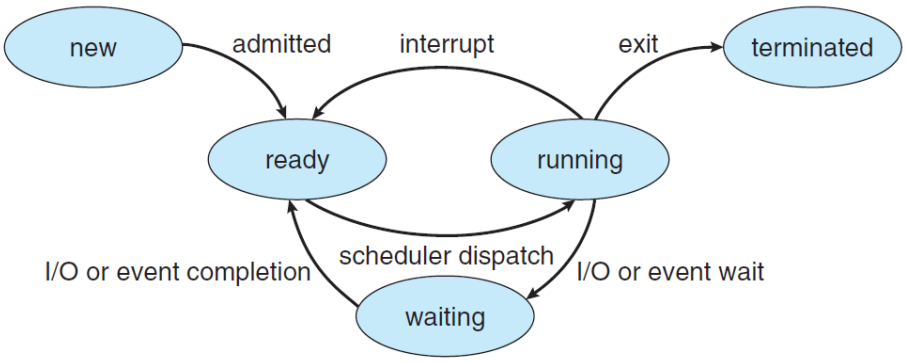
Trạng thái của tiến trình (process state) tại một thời điểm được xác định bởi hoạt động hiện thời của tiến trình tại thời điểm đó. Trong quá trình sống của nó, một tiến trình thay đổi trạng thái do nhiều nguyên nhân như: phải chờ một sự kiện nào đó xảy ra, hay đợi một thao tác nhập/xuất hoàn tất, buộc phải dừng hoạt động do đã hết thời gian xử lý,…

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

Khởi tạo (*new*):	Tiến trình đang được khởi tạo
Thực hiện (*running*):	Các câu lệnh của tiến trình được xử lý
Chờ đợi (*waiting*/*blocked*):	Tiến trình chờ được cấp phát một tài nguyên, hay chờ một sự kiến xảy ra.
Sẵn sàng (*ready*):	Tiến trình chờ được cấp CPU để xử lý.
Kết thúc (*terminated*):	Tiến trình hoàn tất xử lý.

Sơ đồ chuyển trạng thái tiến trình

Hình 2.1. là sơ đồ chuyển trạng thái tiến trình (diagram of process state). Với hệ thống có một processor thì có duy nhất một tiến trình ở trạng thái *running*, có thể có nhiều tiến trình ở trạng thái *waiting*/ *blocked* hoặc *ready*.



Hình 2.1. Diagram of process state

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

(*new, ready):*	Tiến trình mới tạo được đưa vào hệ thống.
(ready, running):	Bộ điều phối cấp phát cho tiến trình một khoảng thời gian sử dụng CPU.
(running, terminated):	Tiến trình kết thúc.
(running, waiting/blocked):	Tiến trình yêu cầu một tài nguyên nhưng chưa được đáp ứng vì tài nguyên chưa sẵn sàng để cấp phát tại thời điểm đó; hoặc tiến trình phải chờ một sự kiện hay thao tác nhập/xuất.
(running, ready):	Bộ điều phối chọn một tiến trình khác để cho xử lý.
(waiting/blocked, ready):	Tài nguyên mà tiến trình yêu cầu trở nên sẵn sàng để cấp phát; hay sự kiện hoặc thao tác nhập/xuất tiến trình đang đợi hoàn tất.

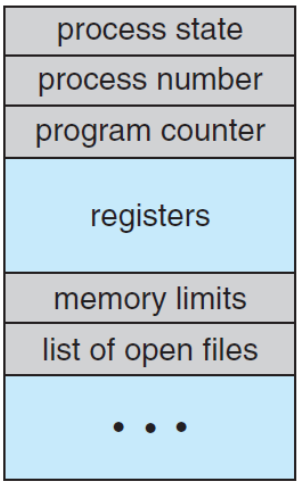
2.1.3. Chế độ xử lý của tiến trình

Để đảm bảo hệ thống hoạt động đúng đắn, hệ điều hành cần phải được bảo vệ khỏi sự xâm phạm của các tiến trình. Bản thân các tiến trình và dữ liệu cũng cần được bảo vệ để

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

2.1.4. Cấu trúc dữ liệu khối quản lý tiến trình

Hệ điều hành quản lý các tiến trình trong hệ thống thông qua **Khối quản lý tiến trình** (process control block -PCB). PCB là một vùng nhớ lưu trữ các thông tin về: trạng thái tiến trình, bộ đếm lệnh, các thanh ghi của CPU, thông tin dùng để điều phối tiến trình, thông tin quản lý bộ nhớ, thông tin tài nguyên có thể sử dụng, thông tin thống kê, con trỏ trỏ đến một PCB khác,…



Hình 2.2. Process control block (PCB) [3]

Một số thành phần chủ yếu của PCB bao gồm:

- **Định danh của tiến trình (1)**: Giúp phân biệt các tiến trình.
- **Trạng thái tiến trình (2)**: Xác định hoạt động hiện hành của tiến trình.

- **Ngữ cảnh của tiến trình (3)**: Mô tả các tài nguyên tiến trình đang trong quá trình, hoặc để phục vụ cho hoạt động hiện tại, hoặc để làm cơ sở phục hồi hoạt động cho tiến trình, bao gồm các thông tin về: Trạng thái CPU, bộ xử lý, bộ nhớ chính, tài nguyên sử dụng, tài nguyên tạo lập.
- **Thông tin giao tiếp (4)**: Phản ánh các thông tin về quan hệ của tiến trình với các tiến trình khác trong hệ thống: Tiến trình cha, tiến trình con, độ ưu tiên.
- **Thông tin thống kê (5)**: Đây là những thông tin thống kê về hoạt động của tiến trình: thời gian đã sử dụng CPU, thời gian chờ,… các thông tin này có thể có ích cho công việc đánh giá tình hình hệ thống và dự đoán các tình huống tương lai.

2.1.5. Thao tác trên tiến trình

Hệ điều hành cung cấp các thao tác chủ yếu sau trên một tiến trình: Tạo lập tiến trình, kết thúc tiến trình, tạm dừng tiến trình, tái kích hoạt tiến trình, thay đổi độ ưu tiên tiến trình.

Trong mục này, giáo trình chỉ trình bày về hai thao tác tạo lập tiến trình và kết thúc tiến trình.

Tạo lập tiến trình

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

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

- Định dạng cho tiến trình mới phát sinh.
- Đưa tiến trình vào danh sách quản lý của hệ thống.
- Xác định độ ưu tiên cho tiến trình
- Tạo PCB cho tiến trình
- Cấp phát các tài nguyên ban đầu cho tiến trình.

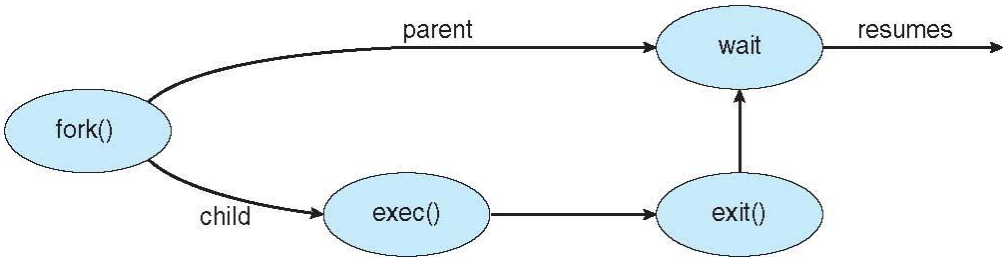
Khi một tiến trình tạo lập một tiến trình con, tiến trình con có thể sẽ được hệ điều hành trực tiếp cấp phát tài nguyên hoặc được tiến trình cha cho thừa hưởng một số tài nguyên ban đầu. Khi một tiến trình tạo tiến trình mới, tiến trình ban đầu có thể xử lý theo một trong hai khả năng sau; tiến trình cha tiếp tục xử lý đồng hành với tiến trình con. Tiến trình cha chờ đến khi một tiến trình con nào đó hoặc tất cả các tiến trình con kết thúc xử lý. Các hệ điều hành khác nhau có thể lựa chọn các cài đặt khác nhau để thực hiện thao tác tạo lập một tiến trình.

Kết thúc tiến trình

Một tiến trình kết thúc xử lý khi nó hoàn tất chỉ thị cuối cùng và sử dụng một lời gọi hệ thống để yêu cầu hệ điều hành hủy bỏ nó. Đôi khi một tiến trình có thể yêu cầu hệ điều hành kết thúc xử lý của một tiến trình khác. Khi một tiến trình kết thúc, hệ điều hành thực hiện các công việc: Thu hồi các tài nguyên hệ thống đã cấp phát cho tiến trình, hủy tiến trình khỏi tất cả các danh sách quản lý của hệ thống, hủy bỏ PCB của tiến trình.
Hầu hết các hệ điều hành không cho phép các tiến trình con tiếp tục tồn tại nếu tiến trình cha tương ứng của nó đã kết thúc. Trong những hệ thống như thế, hệ điều hành sẽ tự động phát sinh một loạt các thao tác kết thúc tiến trình con.

Tiến trình mồ côi và tiến trình zombie

Trong quá trình vận hành, khi tiến trình tạo ra các tiến trình con khác để tạo thành một cây tiến trình. Về nguyên tắc, tiến trình con được tạo ra khi kết thúc tiến trình phải gửi thông tin báo cho tiến trình cha biết về việc kết thúc tiến trình. Cũng như khi kết thúc tiến trình cha, các tiến trình con được tạo ra phải được kết thúc trước.
Ví dụ: trên trình duyệt web (Chrome, FireFox,…) khi trình duyệt web được khởi chạy, lúc này trình duyệt web là 1 tiến trình. Khi ta tạo ra các tab trong trình duyệt để truy cập các websites khác nhau, các tab này chính là các tiến trình con mới được tạo ra. Khi ta tắt trình duyệt, các tab được tạo ra trong trình duyệt cũng sẽ được tắt theo.



Hình 2.3. Tạo lập tiến trình [3]

Tuy nhiên, trên thực tế vận hành, có những trường hợp đặc biệt và có thể gây ra lỗi như:

- Tiến trình cha kết thúc nhưng không tắt tiến trình con. Trạng thái này được gọi là tiến trình mồ côi (orphant process). Điều này có thể làm cho tiến trình con không thể kết thúc tiến trình, ảnh hưởng đến tài nguyên cấp phát của hệ thống. Để khắc phục, hệ điều hành xử lý bằng cách ép tiến trình con nhận một tiến trình mặc định của hệ điều hành để làm tiến trình cha tạm thời và kết thúc tiến trình.
- Tiến trình con kết thúc nhưng không gửi thông tin kết thúc cho tiến trình cha. Trạng thái này được gọi là tiến trình zombie. Điều này dẫn đến tiến trình cha không thể kết thúc tiến trình, ảnh hưởng đến tài nguyên cấp phát của hệ thống. Để khắc phục, người sử dụng phải ra lệnh kết thúc tiến trình cha.

2.2. Luồng

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

2.2.1. Định nghĩa

Một tiểu trình (threads) là một đơn vị xử lý cơ bản trong hệ thống.

Mỗi tiểu trình xử lý tuần tự đoạn code của nó, sở hữu một con trỏ lệnh, tập các thanh ghi và một vùng nhớ stack riêng. Các tiểu trình chia sẻ CPU với nhau giống như cách chia sẻ giữa các tiến trình: Một tiểu trình xử lý trong khi các tiểu trình khác chờ

đến lượt. Một tiểu trình cũng có thể tạo lập các tiến trình con, và nhận các trạng thái khác nhau như một tiến trình thực sự. Một tiến trình có thể sở hữu nhiều tiểu trình.
Các tiến trình tạo thành những thực thể độc lập. Mỗi tiến trình có một tập tài nguyên và một môi trường riêng. Các tiến trình hoàn toàn độc lập với nhau, chỉ có thể liên lạc thông qua các cơ chế thông tin giữa các tiến trình mà hệ điều hành cung cấp. Ngược lại, các tiểu trình trong cùng một tiến trình lại chia sẻ một không gian địa chỉ chung, điều này có nghĩa là các tiểu trình có thể chia sẻ các biến toàn cục của tiến trình.

2.2.2. Thông tin so sánh sự giống nhau và khác nhau của tiến trình và luồng

Trong lĩnh vực hệ điều hành, tiến trình và luồng là hai khái niệm cơ bản nhưng khác nhau. Chúng đều đóng vai trò quan trọng trong việc thực thi và quản lý các chương trình.

Bảng 2.1. Sự giống nhau của tiến trình và luồng

Các tiêu chí so sánh	Nội dung
Đơn vị thực thi	Cả tiến trình và luồng đều là các đơn vị cơ bản mà hệ điều hành sử dụng để quản lý thực thi chương trình.
Chạy đồng thời	Cả hai đều có thể chạy đồng thời, cho phép máy tính thực hiện nhiều tác vụ cùng lúc.
Sử dụng tài nguyên hệ thống	Cả tiến trình và luồng đều sử dụng các tài nguyên của hệ thống như CPU và bộ nhớ.
Lập lịch và quản lý	Hệ điều hành đảm nhận việc lập lịch và quản lý cả tiến trình lẫn luồng.

Bảng 2.2. Sự khác nhau của tiến trình và luồng

Các tiêu chí so sánh	Tiến trình (process)	Luồng (thread)
Định nghĩa và cấu trúc	Là một chương trình đang chạy, có không gian bộ nhớ riêng biệt.	Là một đơn vị nhỏ nhất của thực thi, nằm trong một tiến

	Mỗi tiến trình cung cấp môi trường chạy độc lập cho chương trình hoặc tập hợp các chương trình.	trình. Các luồng trong cùng một tiến trình có thể chia sẻ không gian bộ nhớ và tài nguyên.
Tài nguyên và không gian bộ nhớ	Có không gian bộ nhớ riêng biệt; tài nguyên không được chia sẻ giữa các tiến trình mặc định.	Các luồng trong cùng một tiến trình chia sẻ không gian bộ nhớ và tài nguyên, như heap, dữ liệu toàn cục,…
Chi phí tạo và quản lý	Việc tạo và quản lý tiến trình thường tốn kém hơn vì cần phân bổ không gian bộ nhớ riêng và tài nguyên hệ thống.	Tạo và quản lý luồng ít tốn kém hơn so với tiến trình vì chúng chia sẻ tài nguyên với tiến trình mẹ.
Hiệu suất	Chuyển đổi giữa các tiến trình có thể tốn nhiều thời gian hơn do cần lưu và phục hồi nhiều thông tin hơn.	Chuyển đổi giữa các luồng thường nhanh hơn vì chúng chia sẻ một số thông tin.
Độc lập	Mỗi tiến trình hoạt động độc lập và không phụ thuộc trực tiếp vào tiến trình khác.	Luồng phụ thuộc vào tiến trình mà chúng thuộc về; sự cố trong một luồng có thể ảnh hưởng đến các luồng khác trong cùng một tiến trình.
Truyền thông và tương tác	Truyền thông giữa các tiến trình thường phức tạp hơn và cần các kỹ thuật IPC (Inter-Process Communication).	Luồng có thể dễ dàng giao tiếp và chia sẻ dữ liệu với nhau do chung không gian bộ nhớ.

Hiểu biết về sự giống và khác nhau giữa tiến trình và luồng trong lập trình đa tiến trình và đa luồng, giúp tối ưu hóa hiệu suất và độ tin cậy của ứng dụng.

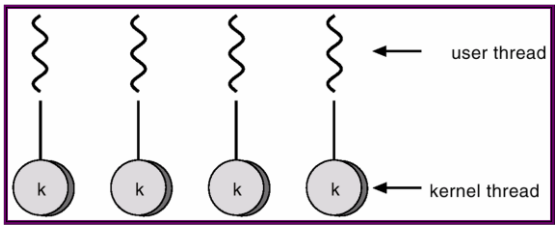
2.2.3. Các mô hình đa luồng

Trong việc quản lý lập trình đa luồng (tiểu trình), có 2 loại luồng:

- Luồng người dùng: người lập trình sử dụng các thư viện để tạo ra các luồng để thực thi chương trình. Thông thường, số lượng luồng sẽ được tạo ra tùy theo người lập trình.
- Luồng ở mức hệ thống: nhân hệ thống (kernel) quản lý các luồng ở mức vật lý. Thông thường, số lượng luồng sẽ luôn cố định ở một con số duy nhất.

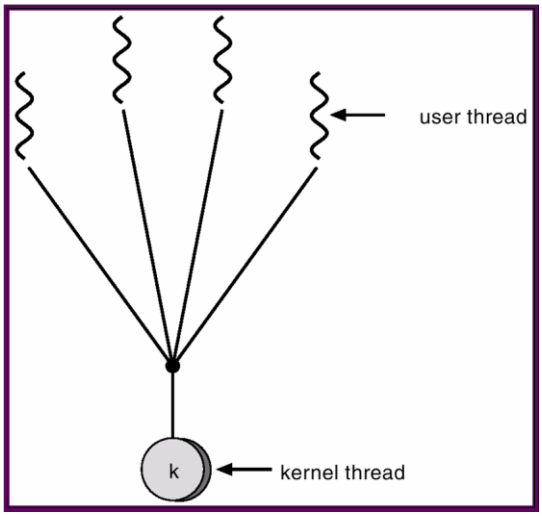
Giữa luồng người dùng và luồng ở mức hệ thống sẽ được hỗ trợ để cùng làm việc với nhau, điều này hình thành nên mô hình đa luồng. Có một số loại mô hình đa luồng phổ biến như sau:

- Mô hình một một: đây là mô hình mà mỗi luồng ở mức người dùng sẽ ánh xạ với một luồng ở mức hệ thống.



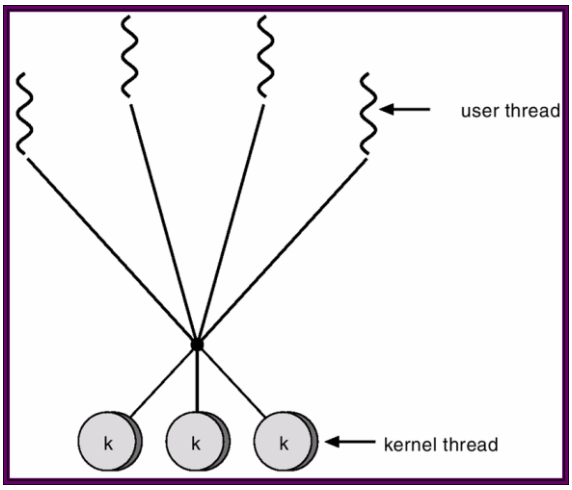
Hình 2.4. Mô hình một - một

- Mô hình nhiều một: đây là mô hình mà nhiều luồng ở mức người dùng sẽ ánh xạ với một luồng ở mức hệ thống.



Hình 2.5. Mô hình nhiều - một

- Mô hình nhiều nhiều: đây là mô hình mà nhiều luồng ở mức người dùng sẽ ánh xạ với nhiều luồng ở mức hệ thống.



Hình 2.6. Mô hình nhiều - nhiều

2.3. Điều phối tiến trình

2.3.1. Mục tiêu điều phối

Mục tiêu của việc điều phối: sự công bằng (fairness), tính hiệu quả (efficiency), thời gian đáp ứng hợp lý (response time), thời gian lưu lại trong hệ thống (turnaround), thông lượng tối đa (throughput). Bản thân các mục tiêu trên có thể có sự mâu thuẩn với nhau, nên chỉ có thể dung hòa theo một mức độ nào đó.

2.3.2. Các đặc điểm của tiến trình

Điều phối hoạt động của các tiến trình là một vấn đề rất phức tạp, đòi hỏi hệ điều hành khi giải quyết phải xem xét nhiều yếu tố khác nhau để có thể đạt được những mục tiêu đề ra. Một số đặc tính của tiến trình cần được quan tâm như sau: tính hướng xuất, nhập của tiến trình, tính hướng xử lý của tiến trình, tiến trình tương tác hay xử lý theo lô, độ ưu tiến của tiến trình, thời gian đã sử dụng CPU của tiến trình, thời gian còn lại tiến trình cần để hoàn tất.

2.3.3. Điều phối độc quyền và điều phối không độc quyền

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

Điều phối độc quyền

Nguyên lý điều phối độc quyền cho phép một tiến trình khi nhận được CPU sẽ có quyền độc chuyến CPU đến khi hoàn tất xử lý hoặc tự nguyện giải phóng CPU. Khi đó quyết định điều phối CPU sẽ xảy ra trong các tình huống sau: Khi tiến trình chuyển từ trạng thái đang xử lý (running) sang trạng thái bị khóa (blocked) hoặc khi tiến trình kết thúc.
Các thuật toán điều phối độc quyền thường đơn giản và dễ cài đặt. Với các hệ thống tổng quát nhiều người dùng, thì chiến lược điều phối độc quyền thường không phù hợp, vì nếu cho phép một tiến trình có quyền xử lý bao lâu tùy ý, có nghĩa là tiến trình này có thể giữ CPU một thời gian không xác định, có thể ngăn cản những tiến trình còn lại trong hệ thống có cơ hội để xử lý.

Điều phối không độc quyền

Điều phối theo nguyên lý không độc quyền cho phép tạm dừng hoạt động của một tiến trình đang sẵn sàng xử lý. Khi một tiến trình nhận được CPU, nó vẫn được sử dụng CPU đến khi hoàn tất hoặc tự nguyện giải phóng CPU, nhưng một tiến trình khác có độ ưu tiên hơn có thể dành quyền sử dụng CPU của tiến trình ban đầu. Như vậy là tiến trình có thể bị tạm dừng hoạt động bất cứ lúc nào mà không được báo trước, để tiến trình khác xử lý.
Các quyết định điều phối xảy ra khi: Khi tiến trình chuyển từ trạng thái đang xử lý (running) sang trạng thái bị khóa (blocked), khi tiến trình chuyển từ trạng thái đang xử lý (running) sang trạng thái ready, khi tiến trình chuyển từ trạng thái chờ (blocked) sang trạng thái ready hoặc khi tiến trình kết thúc.
Các thuật toán điều phối theo nguyên tắc không độc quyền ngăn cản được tình trạng một tiến trình độc chiếm CPU, nhưng việc tạm dừng một tiến trình có thể dẫn đến các mâu

thuẩn trong truy xuất, đòi hỏi phải sử dụng một phương pháp đồng bộ hóa thích hợp để giải quyết.

Trong các hệ thống sử dụng nguyên lý điều phối độc quyền có thể xảy ra tình trạng các tác vụ cần thời gian xử lý ngắn phải chờ tác vụ xử lý với thời gian rất dài hoàn tất. Nguyên lý điều phối độc quyền thường chỉ thích hợp với các hệ xử lý theo lô. Đối với các hệ thống tương tác (time sharing), các hệ thống thời gian thực (real time) cần phải sử dụng nguyên lý điều phối không độc quyền để các tiến trình quan trọng có cơ hội hồi đáp kịp thời. Tuy nhiên thực hiện điều phối theo nguyên lý không độc quyền đòi hỏi những cơ chế phức tạp trong việc phân định độ ưu tiên và phát sinh thêm chi phí khi chuyển đổi CPU qua lại giữa các tiến trình.

2.3.4. Độ ưu tiên của tiến trình

Để có cơ sở điều phối các tiến trình một cách hợp lý, hệ điều hành cần biết độ ưu tiên của từng tiến trình. Độ ưu tiên của một tiến trình là một giá trị giúp phân định tầm quan trọng của tiến trình. Nó có thể được phát sinh tự động bởi hệ thống hoặc được gán tường mình bởi người dùng. Độ ưu tiên có thể tĩnh hay động, và có thể được gán cho tiến trình một cách hợp lý hay ngẫu nhiên.

Độ ưu tiên tĩnh

Là độ ưu tiên được gán sẵn cho tiến trình, và không thay đổi bất kể sự biến động của môi trường. Cơ chế độ ưu tiên tĩnh dễ thực hiện nhưng đôi khi không hợp lý vì môi trường thay đổi có thể ảnh hưởng đến tầm quan trọng của tiến trình.

Độ ưu tiên động

Là độ ưu tiên thay đổi theo thời gian và môi trường xử lý của tiến trình. Tiến trình được khởi động với một độ ưu tiên, độ ưu tiên này thường chỉ giữ giá trị trong một khoảng thời gian ngắn, sau đó hệ thống sẽ sửa đổi giá trị độ ưu tiên trong từng giai đoạn thực hiện của tiến trình cho phù hợp với tình hình cụ thể của hệ thống.

2.3.5. Tổ chức điều phối

Các loại danh sách sử dụng trong quá trình điều phối

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

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

Bộ điều phối sẽ chọn một tiến trình trong danh sách sẵn sàng và cấp CPU cho tiến trình đó. Tiến trình được cấp CPU sẽ thực hiện xử lý, và có thể chuyển qua trạng thái chờ khi xảy ra các sự kiện như đợi một thao tác nhập/xuất hoàn tất, yêu cầu tài nguyên chưa được thỏa mãn, được yêu cầu tạm dừng,… khi đó tiến trình sẽ được chuyển sang một danh sách chờ đợi. Hệ điều hành chỉ sử dụng một danh sách sẵn sàng cho toàn hệ thống, nhưng mỗi một tài nguyên (thiết bị ngoại vi) có một danh sách chờ đợi riêng bao gồm các tiến trình đang chờ được cấp phát tài nguyên đó.

Quá trình xử lý của một tiến trình trãi qua những chu kỳ chuyển đổi qua lại giữa danh sách sẵn sàng và danh sách chờ đợi: Trước hết tiến trình mới được đưa vào danh sách sẵn sàng (ready list), nó sẽ đợi trong danh sách này cho đến khi được chọn để cấp phát CPU và bắt đầu xử lý. Sau đó có thể xảy ra một trong các tình huống sau: Thứ nhất, tiến trình phát sinh một yêu cầu tài nguyên mà hệ thống chưa thể đáp ứng, khi đó tiến trình sẽ được chuyển sang danh sách các tiến trình đang chờ tài nguyên tương ứng. Thứ hai, tiến trình có thể bị bắt buộc tạm dừng xử lý do một ngắt xảy ra, khi đó tiến trình được đưa trở lại vào danh sách sẵn sàng để chờ được cấp CPU cho lượt tiếp theo. Trong trường hợp đầu tiên, tiến trình cuối cùng sẽ chuyển từ trạng thái blocked sang trạng thái ready và lại đưa được trả vào danh sách sẵn sàng. Tiến trình lặp lại chu kỳ này cho đến khi hoàn tất tác vụ thì được hệ thống hủy bỏ khỏi mọi danh sách điều phối.

Các cấp độ điều phối

Việc điều phối được hệ điều hành thực hiện ở hai mức độ: điều phối tác vụ (job scheduling) và điều phối tiến trình (process scheduling).

Điều phối tác vụ

Quyết định lựa chọn tác vụ nào được đưa vào hệ thống, và nạp những tiến trình của tác vụ đó vào bộ nhớ chính để thực hiện. Chức năng điều phối tác vụ quyết định mức độ đa chương của hệ thống (số lượng tiến trình trong bộ nhớ chính). Khi hệ thống tạo lập một tiến trình, hay có một tiến trình kết thúc xử lý thì chức năng điều phối tác vụ mới được kích hoạt. Vì mức độ đa chương tương đối ổn định nên chức năng điều phối tác vụ có tần suất hoạt động thấp.

Để hệ thống hoạt động tốt, bộ điều phối tác vụ cần phân biệt tính chất của tiến trình là hướng nhập/xuất hay hướng xử lý. Một tiến trình được gọi là hướng nhập xuất nếu chủ yếu nó chỉ sử dụng CPU để thực hiện các thao tác nhập xuất. Ngược lại một tiến trình được gọi là hướng xử lý nếu chủ yếu nó chỉ sử dụng CPU để thực hiện các thao tác tính toán. Để cân bằng hoạt động của CPU và các thiết bị ngoại vi, bộ điều phối tác vụ nên lựa chọn các tiến trình để nạp vào bộ nhớ sao cho hệ thống là sự pha trộn hợp lý giữa các tiến trình hướng nhập/xuất và các tiến trình hướng xử lý.

Điều phối tiến trình

Chọn một tiến trình ở trạng thái sẵn sàng (đã được nạp vào bộ nhớ chính và có đủ tài nguyên để hoạt động) và cấp phát CPU cho tiến trình đó thực hiện. Bộ điều phối tiến trình có tần suất hoạt động cao, sau mỗi lần xảy ra ngắt; thường là một lần trong khoảng 100ms. Do vậy để nâng cao hiệu suất của hệ thống, cần phải tăng tốc độ xử lý của bộ điều phối tiến trình. Chức năng điều phối tiến trình (điều phối CPU) là một trong những chức năng cơ bản và quan trọng nhất của hệ điều hành.

2.4. Các chiến lược điều phối tiến trình

2.4.1. Các tiêu chuẩn điều phối CPU

Mục này trình bày một số tiêu chuẩn cơ bản nhất: *thời gian đáp ứng, thời gian hoàn thành, thời gian chờ*; việc điều phối tiến trình nhằm làm cho các giá trị thời gian này là nhỏ nhất có thể.

Thời gian đáp ứng

Thời gian đáp ứng là khoảng thời gian tính từ thời điểm tiến trình được gửi vào hệ thống cho đến khi lần đầu tiên tiến trình đó được sử dụng CPU.

Thời gian hoàn thành

Thời gian hoàn thành là khoảng thời gian tính từ thời điểm tiến trình đến hệ thống cho đến khi tiến trình đó kết thúc; tức bằng thời gian kết thúc của tiến trình – thời gian đến.

Thời gian chờ đợi

Thời gian chờ là tổng thời gian chờ trong hàng đợi sẵn sàng; tức bằng thời gian hoàn thành – thời gian sử dụng CPU.

Lưu ý rằng các thuật toán điều phối CPU không ảnh hưởng đến các tiến trình đang thực hiện hay đang đợi thiết bị nhập xuất. Thời gian chờ có thể là thời gian để đưa tiến trình vào bộ nhớ, thời gian chờ trong hàng đợi sẵn sàng, thời gian chờ trong hàng đợi thiết bị.

Ví dụ 2.1

Cho 3 tiến trình P_1, P_2, P_3 có thời điểm vào hệ thống lần lượt là 0, 1, 2 và có thời gian xử lý lần lượt là 24, 3, 3. Hãy tính thời gian đáp ứng trung bình, thời gian hoàn thành trung bình và thời gian chờ trung bình của các tiến trình.

Hướng dẫn:

Biểu đồ Gantt (thứ tự cấp phát CPU) của các tiến trình là:

P_1	P_2	P_3	
0	24	27	30

Thời gian đáp ứng của từng tiến trình:

P_1: 0 – 0 =0

P_2: 24 – 1 =23

P_3: 27- 2= 25

Suy ra thời gian đáp ứng trung bình của các tiến trình là (0+23+25)/3=48/3=16 (ms).

Thời gian hoàn thành của từng tiến trình:

P_1: 24-0=24

P_2: 27-1=26

P_3: 30-2=28

Suy ra thời gian hoàn thành trung bình của các tiến trình là (24+26+28)/3=78/3=26 (ms).

Thời gian chờ của từng tiến trình:

P_1: 24-24=0

P_2: 26-3=23

P_3: 28-3=25

Suy ra thời gian chờ trung bình của các tiến trình là (0+23+25)/3=48/3=16 (ms).

2.4.2. Chiến lược điều phối "*đến trước phục vụ trước*"

Nguyên tắc:

Chiến lược điều phối "*đến trước phục vụ trước*" (first come first served - FCFS).

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

Ví dụ 2.2

Process	Arrival time	Service time
P_1	0	24
P_2	1	3
P_3	2	3

Thông tin từ arrival time cho thấy thứ tự vào của các tiến trình là P_1, P_2, P_3.

Biểu đồ Gantt của các tiến trình là:

P_1	P_2	P_3	
0	24	27	30

Tính thời gian chờ của các tiến trình:

- Thời gian chờ của tiến trình P_1 là 0
- Thời gian chờ của tiến trình P_2 là 23
- Thời gian chờ của tiến trình P_3 là 25

Thời gian chờ trung bình (average waiting time) của các tiến trình này là (0 + 23 + 25)/3 =16 (ms).

Với chiến lược điều phối FCFS, các tiến trình có thời gian thực hiện ngắn phải chờ đợi như các tiến trình có thời gian thực hiện dài; do vậy thời gian chờ trung bình không tối ưu và thời gian chờ trung bình có sự thay đổi đáng kể khi thay đổi thứ tự thực hiện các tiến trình.

Xem lại **ví dụ 2.2** trên; trong đó có thay đổi thứ tự thực hiện các tiến trình.

Giả sử thứ tự vào của các tiến trình là P_2, P_3, P_1 như bảng sau:

Process	Arrival time	Service time
P_1	2	24
P_2	0	3
P_3	1	3

Biểu đồ Gantt của các tiến trình là:

P_2	P_3	P_1	
0	3	6	30

Tính thời gian chờ của các tiến trình:

- Thời gian chờ của tiến trình P_1 là 4

- Thời gian chờ của tiến trình P_2 là 0
- Thời gian chờ của tiến trình P_3 là 2

Thời gian chờ trung bình (average waiting time) của các tiến trình này là (4 + 0 + 2)/3 = 2(ms).

Chiến lược điều phối này 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.

2.4.3. Chiến lược điều phối "xoay vòng"

Nguyên tắc: Chiến lược điều phối "xoay vòng" (round robin)

Danh sách sẵn sàng được xử lý như một danh sách vòng, bộ điều phối lần lượt cấp phát cho từng tiến trình trong danh sách một khoảng thời gian sử dụng CPU gọi là quantum; đây là một chiến lược điều phối theo nguyên tắc không độc quyền: Khi một tiến trình sử dụng CPU đến hết thời gian quantum dành cho nó, hệ điều hành sẽ 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 chờ đợi được cấp CPU trong lượt kế tiếp.

Ví dụ 2.3

Process	Arrival time	Service time
P_1	0	24
P_2	1	3
P_3	2	3

Nếu sử dụng *quantum time*=4ms, thứ tự cấp phát CPU là:

P_1	P_2	P_3	P_1	P_1	P_1	P_1	P_1	
0	4	7	10	14	18	22	26	30

Tính thời gian chờ của các tiến trình:

- Thời gian chờ của tiến trình P_1 là 0+6=6
- Thời gian chờ của tiến trình P_2 là 3
- Thời gian chờ của tiến trình P_3 là 5

Thời gian chờ trung bình (average waiting time) của các tiến trình này là (6 + 3 + 5)/3 = 4.67 (ms).

Nếu 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. Nếu quantum quá lớn sẽ làm tăng thời gian hồi đáp và giảm khả năng tương tác của hệ thống.

2.4.4. Chiến lược điều phối theo "độ ưu tiên"

Nguyên tắc:

Mỗi tiến trình gắn với một độ ưu tiên tương ứng, tiến trình có độ ưu tiên cao nhất sẽ được chọn để cấp phát CPU đầu tiên. Độ ưu tiên có thể được định nghĩa nội tại hay nhờ vào các yếu tố bên ngoài. Độ ưu tiên nội tại sử dụng các đại lượng có thể đo lường để tính toán độ ưu tiên của tiến trình. Độ ưu tiên cũng có thể được gán từ bên ngoài dựa vào các tiêu chuẩn do hệ điều hành quy định; chẳng hạn như tầm quan trọng của tiến trình, loại người sử dụng sở hữu tiến trình,…

Chiến lược điều phối với độ ưu tiên có thể theo nguyên tắc độc quyền hoặc 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 các tiến trình hiện đang xử lý. Chiến lược điều phối với độ ưu tiên 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. Chiến lược điều phối với độ ưu tiên độc quyền sẽ chỉ đơn giản chèn tiến trình mới vào danh sách sẵn sàng, và tiến trình mới hiện hành vẫn tiếp tục xử lý hết thời gian dành cho nó.

Ví dụ 2.4

Process	Arrival time	Service time	Priority
P_1	0	24	3
P_2	1	3	1
P_3	2	3	2

Áp dụng chiến lược điều phối với độ ưu tiên theo nguyên tắc độc quyền

Biểu đồ Gantt của các tiến trình là:

P_1	P_2	P_3	
0	24	27	30

Thời gian chờ của tiến trình P_1 là 0

Thời gian chờ của tiến trình P_2 là 23

Thời gian chờ của tiến trình P_3 là 25

Thời gian chờ trung bình (average waiting time) của các tiến trình này là (0 + 23 + 25)/3 = 7 (ms).

Kết quả này như chiến lược FCFS.

Áp dụng chiến lược điều phối với độ ưu tiên theo nguyên tắc không độc quyền

Biểu đồ Gantt cho lịch này là:

P_1	P_2	$P3$	P_1	
0	1	4	7	30

Thời gian chờ của tiến trình P_1 là 6

Thời gian chờ của tiến trình P_2 là 0

Thời gian chờ của tiến trình P_3 là 2

Thời gian chờ trung bình (average waiting time) của các tiến trình này là (6 + 0 + 2)/3 = 2.67 (ms).

2.4.5. Chiến lược điều phối "*công việc ngắn nhất*"

Nguyên tắc:

Chiến lược điều phối "*công việc ngắn nhất*" (shortest job first - SJF) là một trường hợp đặc biệt của chiến lược điều phối với độ ưu tiên. Tiến trình có thời gian sử dụng CPU ít nhất sẽ sở hữu CPU. Chiến lược điều phối "*công việc ngắn nhất*" cũng có thể độc quyền (SJF) hoặc không độc quyền (shortest remaining time first - SRTF).

Ví dụ 2.5

Process	Arrival time	Service time
P_1	0	6
P_2	1	8
P_3	2	4
P_4	3	2

Áp dụng chiến lược SJF điều phối theo nguyên tắc độc quyền

Biểu đồ Gantt cho lịch này là:

P_1	P_4	P_3	P_2	
0	6	8	12	20

Tính thời gian chờ của các tiến trình:

- Thời gian chờ của tiến trình P_1 là 0
- Thời gian chờ của tiến trình P_2 là 11
- Thời gian chờ của tiến trình P_3 là 6
- Thời gian chờ của tiến trình P_4 là 3

Thời gian chờ trung bình (average waiting time) của các tiến trình này là (0 + 11 + 6 + 3)/4 = 5 (ms).

Áp dụng chiến lược SJF điều phối theo nguyên tắc không độc quyền

Biểu đồ Gantt cho lịch này là:

P_1	P_4	P_1	P_3	P_2	
0	3	5	8	12	20

(Lưu ý tại thời điểm 2, thời gian sử dụng CPU còn lại của P_1 và P_3 đều bằng 4, nên P_1 vẫn được cho thực hiện tiếp đến thời điểm 3).

Tính thời gian chờ của các tiến trình:

- Thời gian chờ của tiến trình P_1 là 0 + 2 = 2
- Thời gian chờ của tiến trình P_2 là 11
- Thời gian chờ của tiến trình P_3 là 6
- Thời gian chờ của tiến trình P_4 là 0

Thời gian chờ trung bình (average waiting time) của các tiến trình này là (2 + 11 + 6 + 0)/4 = 4.75 (ms).

2.5. Đồng bộ hóa tiến trình

2.5.1. Liên lạc giữa các tiến trình

Nhu cầu liên lạc giữa các tiến trình

Nhu cầu liên lạc giữa các tiến trình (inter-process communication - IPC) trong hệ điều hành xuất phát từ việc các tiến trình cần chia sẻ thông tin, dữ liệu, hoặc cần phối hợp thực hiện các công việc. IPC là một phần quan trọng trong thiết kế và hoạt động của hệ thống phân tán, cơ sở dữ liệu, hệ thống mạng và ứng dụng đa tiến trình.

Khi thực hiện liên lạc giữa các tiến trình, điều này giúp mang lại những ưu điểm như:

- Giúp chia sẻ thông tin cho các tiến trình.
- Tăng tốc độ tính toán của các tiến trình.
- Tạo ra các tiến trình chuyên dụng, thực hiện từng phần chuyên dụng trong một hệ thống lớn.
- Tăng thêm tính tiện lợi cho các tiến trình.

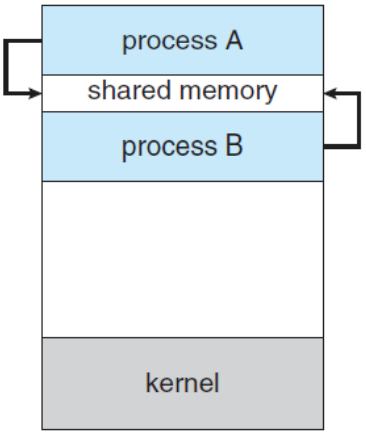
Các vấn đề nảy sinh trong việc liên lạc giữa các tiến trình

- Đồng bộ hóa (synchronization): Đảm bảo rằng các tiến trình không ghi đè lên dữ liệu của nhau khi chia sẻ tài nguyên là một thách thức lớn.
- Tắc nghẽn (deadlock): Các tiến trình có thể rơi vào trạng thái chờ đợi lẫn nhau vô hạn, tạo ra tình trạng tắc nghẽn.
- Không được cấp phát tài nguyên (starvation): Một tiến trình có thể không bao giờ được cấp tài nguyên do ưu tiên thấp hoặc sự không công bằng trong việc phân phối tài nguyên.
- Nhất quán dữ liệu (consistency): Duy trì sự nhất quán của dữ liệu khi nhiều tiến trình cùng truy cập là một thách thức, đặc biệt trong môi trường phân tán.
- Bảo mật (security): Liên lạc giữa các tiến trình mở ra rủi ro về bảo mật, như rò rỉ thông tin hoặc các cuộc tấn công qua kênh liên lạc.

2.5.2. Các cơ chế thông tin liên lạc

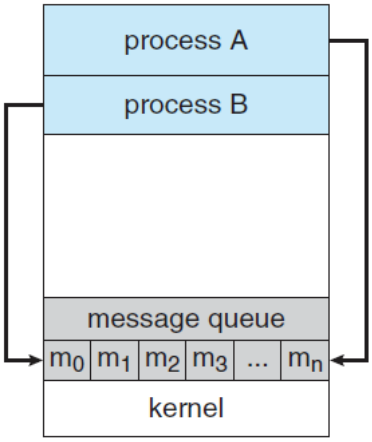
Để thực hiện liên lạc giữa các tiến trình, cần phải có cơ chế để thực hiện. Có 2 cơ chế thực hiện thông tin liên lạc giữa các tiến trình phổ biến là: sử dụng vùng nhớ chia sẻ (share memory) và truyền thông điệp (message passing).

- Sử dụng vùng nhớ chia sẻ là cơ chế cho phép các tiến trình được cấp phát chung một vùng nhớ để trao đổi dữ liệu một cách nhanh nhất và trực tiếp.



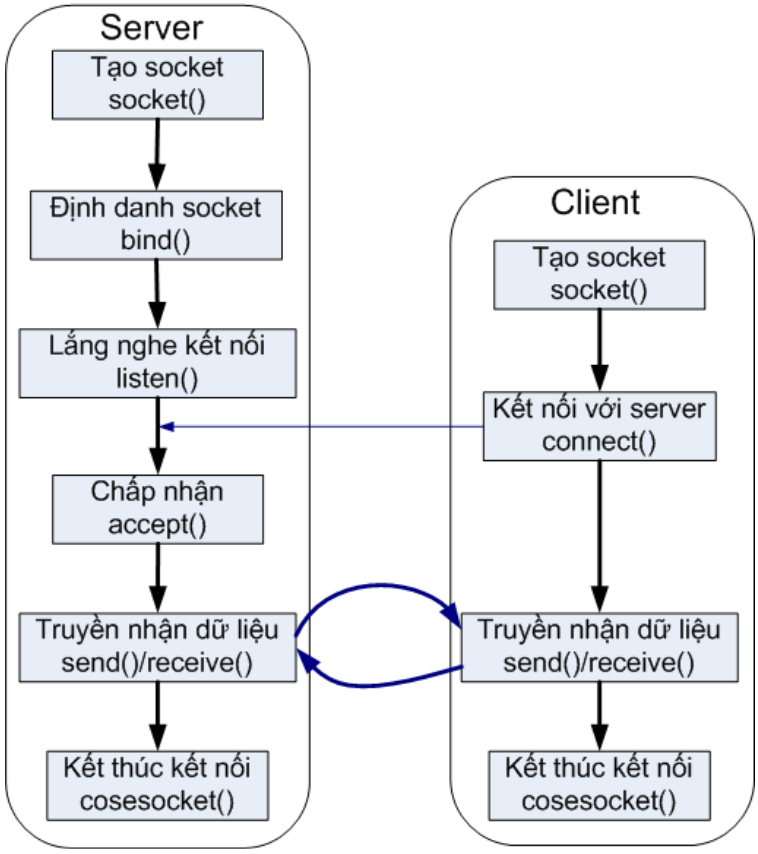
Hình 2.7. Giao tiếp bằng vùng nhớ chia sẻ [3]

- Truyền thông điệp là cơ chế giúp các tiến trình tạo ra các thông điệp thông qua các hàm, thư viện để gửi thông tin cho nhau. Trong việc gửi thông điệp này, có 2 phương pháp là gửi trực tiếp (direction communication) và gửi gián tiếp (indirect communication). Thực hiện gửi trực tiếp của nghĩa là tiến trình A sẽ gửi thông điệp trực tiếp đến tiến trình B, không thông qua bất kỳ tiến trình nào trung gian. Tuy nhiên, nếu một trong hai tiến trình không hoạt động thì việc gửi tiến trình sẽ không thực hiện được. Gửi gián tiếp có nghĩa là các tiến trình sẽ khởi tạo chung một tiến trình trung gian (hộp thư – mailbox), lúc này các tiến trình sẽ gửi lên thông điệp lên hộp thư. Như vậy, khi một trong hai tiến trình không hoạt động, việc thực hiện gửi nhận vẫn diễn ra.



Hình 2.8. Giao tiếp bằng cách truyền thông điệp [3]

Ngoài ra, ngày nay các máy tính có thể kết nối với nhau thông qua hệ thống mạng. Do đó, việc giao tiếp giữa các tiến trình thông qua môi trường mạng cũng rất cấn thiết. Cơ chế truyền thông điệp thông qua socket là một cơ chế cho phép các thiết bị có thể gửi thông tin qua môi trường mạng. Việc thực hiện gửi nhận sẽ thông qua các giao thức mạng như TCP/IP và những giao thức truyền thông khác.



Hình 2.9. Giao tiếp liên mạng bằng socket

Ngoài phương pháp socket, việc giao tiếp liên mạng cũng còn nhiều cách như thực hiện bằng cơ chế RPC (Remote producer call) hay RMI (Remote method innovation) – cơ chế truyền thông tin của Java.

2.5.3. Ví dụ thực thi liên lạc giữa các tiến trình

Liên lạc bằng cách sử dụng vùng nhớ chia sẻ - share memory

Các tiến trình sử dụng chung một vùng không gian nhớ chung. Việc tương tác giữa hai tiến trình sẽ hoàn toàn do việc đọc/ghi trên vùng nhớ chung này. Hệ điều hành không can thiệp vào quá trình này, vì thế vùng nhớ chia sẻ là phương pháp nhanh nhất (xét về mặt tốc độ) để các tiến trình liên lạc với nhau. Tuy nhiên, nhược điểm của phương pháp này là các tiến trình phải tự quản lý việc đọc/ghi dữ liệu, và quản lý việc tranh chấp tài nguyên.

Ví dụ 2.6

Tiến trình con đọc vào 2 số nguyên, ghi vào vùng nhớ chia sẻ, tiến trình cha thực hiện tính tổng và ghi lại vào vùng nhớ chia sẻ. Tiến trình con đọc kết quả và xuất ra màn hình.

```
#include <stdio.h>
#include <unistd.h>
#include <limits.h>
#include <string.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#define SIZE 256
int main(int argc, char* argv[])
{int *shm, shmid, pid;
// Khai báo biến shm là vùng nhớ chia sẻ, shmid là biến lưu lại id
// của vùng nhớ chia sẻ, biến pid là process id khi thực hiện nhân
// bản.
// Thực hiện quá trình tạo ra vùng nhớ chia sẻ.
     key_t key;
     key=ftok(".","a"))==-1)
     shmid = shmget(key, SIZE, IPC_CREAT | 0666);
     shm = (int*) shmat(shmid, 0, 0);
// Thực hiện nhân bản tiến trình.
     pid = fork();
```

```
     if(pid==0) { // Đây là tiến trình con
// ghi 2 giá trị từ đối số vào 2 biến shm[0] và shm[1] thuộc vùng
// nhớ chia sẻ.
          shm[0] = atoi(argv[1]);
          shm[1] = atoi(argv[2]);
// cho tiến trình tạm nghỉ để tiến trình con tính toán.
          sleep(3);
// Xuất thông tin của sau tính toán.
          printf("%d + %d = %d\n", shm[0], shm[1], shm[2]);
// Hủy vùng nhớ dùng chung khỏi tiến trình con.
          shmdt((void*) shm);
// Xóa vùng nhớ chia sẻ sau khi sử dụng.
          shmctl(shmid, IPC_RMID, (struct shmid_ds*) 0);
          return 0;
     }
     else if(pid >0) { // Đây là tiến trình con.
// Tiến trình cha tạm nghỉ để chờ tiến trình con chép dữ liệu lên
// vùng nhớ dùng chung.
          sleep(1);
// Thực hiện tính toán sau khi đã nhận dữ liệu từ tiến trình con.
          shm[2] = shm[1] + shm[0];
// Hủy vùng nhớ dùng chung khỏi tiến trình con.
          shmdt((void*) shm);
// Chương trình tạm nghỉ để chờ tiến trình xóa vùng nhớ chia sẻ.
          sleep(5);
          return 0;
     }
     else { perror("Fork failed."); return 4; }
     return 0;
}
```

Trong ví dụ trên, để hai tiến trình có thể thực hiện tính toán đúng như yêu cầu, ta dùng hàm sleep(), để tiến trình tạm ngừng và chờ kết quả ở tiến trình còn lại.

Liên lạc bằng cách sử dụng đường ống - pipe

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

Khi một pipe được thiết lập giữa hai tiến trình, một trong chúng sẽ ghi dữ liệu vào pipe và tiến trình kia sẽ đọc dữ liệu từ pipe. Thứ tự dữ liệu truyền qua pipe được bảo toàn theo nguyên tắc FIFO. Một pipe có kích thước giới hạn (thường là 4096 ký tự). Trong một pipe việc truyền dữ liệu được thiết lập 1 chiều, có nghĩa là tiến trình đọc và ghi được xác định từ đầu và chỉ thực hiện duy nhất 1 chiều như đã khai báo.

Có 2 loại pipe: unname pipe (đường ống không tên) và name pipe (đường ống có tên).

Unname pipe: là đường ống sử dụng các biến cục bộ. Thường được áp dụng cho các tiến trình được nhân bản, có mối quan hệ cha con.

Name pipe: là đường ống được ghi nhận trên hệ thống tập tin. Có thể sử dụng với những tiến trình hoàn toàn độc lập, không có mối quan hệ cha con.

Ví dụ 2.7

Sử dụng đường ống không tên, tiến trình con đọc dữ liệu từ đối số truyền, ghi vào pipe. Tiến trình cha đọc từ pipe và xuất ra màn hình.

```
#include <stdio.h>
#include <unistd.h>
#include <string.h>
int main(int argc, char* argv[])
{
     char result[100]; // biến dùng để lưu trữ dữ liệu
     int fp[2]; // khai báo đường ống không tên.
     int pid;  // khai báo process id để nhân bản tiến trình.
     if(argc<2) {
          printf("Doi so thieu.\n");
          return -1;
     }
//Tạo đường ống bằng hàm pipe
     if(pipe(fp)==0) {
// Thực hiện nhân bản tiến trình
          pid = fork();
          if(pid<0) {printf("Fork failed\n"); return -1;}
          else if(pid==0) {
// Thực hiện nhận dữ liệu từ tiến trình con.
               printf("Data from child: %s\n", argv[1]);
// Khai báo tiến trình con là đầu ghi và ghi dữ liệu lên đường ống.
               close(fp[0]);
               write(fp[1], argv[1], strlen(argv[1]));
               }
          else {
// Thực hiện xuất dữ liệu ở tiến trình cha.
// Khai báo tiến trình cha là đầu đọc và nhận dữ liệu từ đường
// ống.
               close(fp[1]);
               read(fp[0], result, strlen(argv[1]));
// Xuất dữ liệu từ biến result sau khi lấy dữ liệu từ đường ống.
               printf("Read from child: %s\n", result);
               }
          }
          else {printf("Pipe failed\n"); return -2;}
}
```

Ví dụ 2.8

Sử dụng đường ống có tên, cho phép người dùng nhập vào một chuỗi, gửi chuỗi này qua tiến trình thứ 2. Tiến trình thứ 2 cho phép nhập lại 1 chuỗi khác, gửi ngược lại tiến trình đầu tiên.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/errno.h>
// Khai biến FIFO1 và FIFO2 là 2 đường ống có tên.
#define FIFO1 "/tmp/ff.1"
#define FIFO2 "/tmp/ff.2"
#define PM 0666
extern int errno;
// Định nghĩa độ lớn của đường ống là 4096 bytes
#define PIPE_BUF 4096
int main(int argc, char* argv[])
{
// Khai báo 2 biến s1 và s2 là 2 biến string dùng để xử lý dữ
// liệu.
     char s1[PIPE_BUF], s2[PIPE_BUF];
// Khai báo biến childpid để nhân bản tiến trình.
// Biến readfd và writerd để quản lý quyền đọc ghi của tiến
// trình.
     int childpid, readfd, writefd;
// Kiểm tra 2 file FIFO1 và FIFO2 có khởi tạo thành công?
     if((mknod(FIFO1, S_IFIFO | PM, 0)<0)&&(errno!=EEXIST)){
          printf("Fail to create FIFO 1. Aborted.\n);
          return -1;
     }
     if((mknod(FIFO2, S_IFIFO | PM, 0)<0)&&(errno!=EEXIST)){
          unlink(FIFO1);
          printf("Fail to create FIFO 2. Aborted.\n);
          return -1;
     }
// Thực hiện nhân bản tiến trình
     childpid=fork();
     if(childpid==0){
// Tiến trình con
// Kiểm tra có thể mở 2 file FIFO1, FIFO2.
// Thiết lập quyền đọc của tiến trình con trên đường ống FIFO1
// Thiết lập quyền ghi của tiến trình con trên đường ống FIFP2
          if((readfd=open(FIFO1, 0))<0)
               perror("Child cannot open readFIFO.\n");
          if((writefd=open(FIFO2, 1))<0)
               perror("Child cannot open writeFIFO.\n");
// Đọc dữ liệu từ đường ống FIFO1 vào biến s2 và in dữ liệu.
          read(readfd, s2, PIPE_BUF);
```

```
          printf("Child read from parent: %s\n", s2);
          printf("Enter response: ");
// Cho nhập dữ liệu vào s1 và ghi lên đường ống FIFO2.
          gets(s1);
          write(writefd, s1, strlen(s1));
// Đóng cả 2 đường ống FIFO1 và FIFO2.
          close(readfd);
          close(writefd);
          return 1;
     }
     else if(childpid>0) {
// Đây là tiến trình cha.
// Kiểm tra có mở thành công 2 file FIFO1, FIFO2.
// Thiết lập quyền đọc của tiến trình con trên đường ống FIFO2
// Thiết lập quyền ghi của tiến trình con trên đường ống FIFO1
          if((writefd=open(FIFO1, 1))<0)
               perror("Parent cannot open writeFIFO.\n");
          if((readfd=open(FIFO2, 0))<0)
               perror("Child cannot open readFIFO.\n");
// Nhập dữ liệu vào biến s1 và chép lên đường ống FIFO1
          printf("Enter data to FIFO1: ");
          gets(s1);
          write(writefd, s1, strlen(s1));
// Đọc dữ liệu từ đường ống FIFO2 vào biến s2 và in dữ liệu.
          read(readfd, s2, PIPE_BUF);
          printf("Parent read from child: %s\n", s2);
// Chờ tiến trình con hoàn thành.
          while(wait((int*) 0)!=childpid);
// Đóng cả 2 đường ống FIFO1 và FIFO2.
          close(readfd);
          close(writefd);
// Hủy cả 2 đường ống sau khi thực hiện.
          if(unlink(FIFO1)<0)
               perror("Cannot remove FIFO1.\n");
          if(unlink(FIFO2)<0)
               perror("Cannot remove FIFO2.\n");
          return 1;
     }
     else { printf("Fork failed\n"); return -1;}
}
```

Liên lạc bằng cách sử dụng tin nhắn – message queue

Message queue là một hộp thư, cho phép các thành phần/service trong một hệ thống (hoặc nhiều hệ thống), gửi thông tin cho nhau. Sở dĩ gọi nó là queue (hàng đợi) vì nó thực hiện việc lấy message theo cơ chế FIFO – First In First Out. Trong các hệ thống dùng kiến trúc microservice, ta sử dụng message queue để giúp các service liên

hệ với nhau **một cách bất đồng bộ**. *Service A* làm xong việc có thể gửi message queue để *service B* biết mà xử lý, **không cần phải chờ** service B làm xong.

Ví dụ 2.9

Tạo 2 tiến trình hoàn toàn độc lập, cho phép gửi thông điệp từ tiến trình này sang tiến trình khác:

```
// Ta tạo ra tiến trình writer để gửi dữ liệu:
#include <stdio.h>
#include <sys/ipc.h>
#include <sys/msg.h>

// Định nghĩa cấu trúc của message queue
struct mesg_buffer {
    long mesg_type; // giá trị để gửi và nhận tin nhắn
    char mesg_text[100]; // nội dung tin nhắn
} message;

int main()
{
// Khai báo biến key và msgid để tạo ra message
    key_t key;
    int msgid;
// Khởi tạo message queue
    key = ftok(".", 1);
    msgid = msgget(key, 0666 | IPC_CREAT);
    message.mesg_type = 1;
// Nhập dữ liệu vào mesg_text trong cấu trúc message
    printf("Write Data : ");
    gets(message.mesg_text);
// Thực hiệu gửi message
    msgsnd(msgid, &message, sizeof(message), 0);
// Xuất thông tin của message
    printf("Data send is : %s \n", message.mesg_text);
    return 0;
}
```

Sau khi thực hiện xong chương trình writer, ta tạo thêm 1 chương trình gọi là reader để nhận dữ liệu.

```
// Ta tạo ra tiến trình reader để nhận dữ liệu:
#include <stdio.h>
#include <sys/ipc.h>
#include <sys/msg.h>

// Định nghĩa cấu trúc của message queue
struct mesg_buffer {
    long mesg_type; // giá trị để gửi và nhận tin nhắn
    char mesg_text[100]; // nội dung tin nhắn
} message;
int main()
```

```
{
// Khai báo biến key và msgid để tạo ra message
    key_t key;
    int msgid;
// Khởi tạo message queue
    key = ftok(".", 1);
    msgid = msgget(key, 0666 | IPC_CREAT);
// Nhận dữ liệu từ tin nhắn và chép vào cấu trúc message.
    msgrcv(msgid, &message, sizeof(message), 1, 0);
// Xuất thông tin của message nhận được.
    printf("Data Received is : %s \n", message.mesg_text);
// Hủy message queue
    msgctl(msgid, IPC_RMID, NULL);
     return 0;
}
```

Trong ví dụ trên, 2 chương trình khi chạy sẽ thực hiện độc lập và có thể gửi dữ liệu cho nhau theo cấu trúc message đã được xây dựng.

2.5.4. Giải pháp đồng bộ hóa tiến trình

Đồng bộ hoá tiến trình là một kỹ thuật trong lập trình hệ thống, giúp đảm bảo các tiến trình xử lý song song không tác động sai lệch đến nhau. Điều này được thực hiện bằng cách đảm bảo hai yêu cầu sau:

- Yêu cầu độc quyền truy xuất (Mutual exclusion): Tại một thời điểm, chỉ có một tiến trình được quyền truy xuất một tài nguyên không thể chia sẻ.
- Yêu cầu phối hợp (Synchronization): Các tiến trình cần hợp tác với nhau để hoàn thành công việc.

Các tiến trình khi thực hiện liên lạc với nhau sẽ phải truy cập vào tài nguyên găng (Critical Resource). Tài nguyên găng là tài nguyên được chia sẻ giữa các tiến trình. Tài nguyên găng có thể là thiết bị vật lý hoặc các dữ liệu dùng chung giữa các tiến trình. Việc truy cập vào tài nguyên găng nếu không được quản lý chính xác sẽ dẫn đến những sai sót trong tính toán. Do đó, cần phải có các giải pháp để đảm bảo rằng các tiến trình không truy cập vào tài nguyên găng cùng một lúc, và đồng thời đảm bảo rằng các tiến trình không bị chặn khi chờ đợi tài nguyên.

i = 0;

Thread a:	**Thread b:**

while (I < 10)
i = i+1;
print “A won!”
	while (I > -10)
	i = i-1;
	print “B won!”

Hình 2.10. Vấn đề khi sử dụng tài nguyên găng

Trong hình trên, biến i là biến dùng chung giữa 2 thread. Biến i chính là tài nguyên găng của 2 thread. Cả hai thread đều có thể tác động và nhận được sự thay đổi biến i ngay lập tức. Điều này sẽ dẫn đến vấn đề trong vòng lặp giữa 2 thread như trong hình trên.

Các giải pháp đồng bộ hoá tiến trình cũng có thể được chia thành hai loại: giải pháp phần mềm và giải pháp phần cứng. Giải pháp phần mềm bao gồm semaphores, mutexes, và monitors, trong khi giải pháp phần cứng bao gồm các cơ chế như hardware locks và transactional memory.

Tuy nhiên, việc đồng bộ hoá tiến trình cũng có thể gây ra hiệu suất giảm do các tiến trình phải chờ đợi tài nguyên được giải phóng.

Giải pháp phần mềm – sử dụng biến cờ hiệu (flag)

Trong giải pháp này, một biến cờ hiệu được tạo ra. Biến cờ hiệu này là biến được dùng chung giữa các tiến trình. Khi một tiến trình muốn truy cập vào các đoạn code sử dụng tài nguyên găng – còn gọi là miền găng (Critical Section – CS) – tiến trình bắt buộc phải kiểm tra giá trị của biến cờ hiệu có cho phép truy cập hay không. Nếu biến cờ hiệu phát tín hiệu cho phép truy cập, lúc này tiến trình có thể truy cập vào miền găng, đồng thời phải đặt trạng thái khóa cho biến cờ hiệu, để đảm bảo các tiến trình khác không được truy cập vào miền găng. Sau khi hoàn thành xong các đoạn code ở miền găng, tiến trình sẽ thoát khỏi miền găng, và đặt lại trạng thái mở của biến cờ hiệu để các tiến trình khác có thể truy cập vào miền găng.

int lock = 0;

P_0	**P_1**
NonCS;	NonCS;

while (lock == 1); //wait
lock = 1;
	while (lock == 1); //wait
	lock = 1;

CS;	CS;
lock = 0;	lock = 0;
NonCS;	NonCS;

Hình 2.11. Sử dụng biến cờ hiệu

Trong hình 2.11, P_0 và P_1 có những miền găng (CS), 2 tiến trình này chia sẻ biến lock là biến cờ hiệu để quản lý quyền truy cập miền găng. Mặc định biến lock = 0 sẽ ở trạng thái mở, nếu biến lock = 1 thì có nghĩa là có tiến trình đang truy cập vào miền găng. Các tiến trình khác phải chờ đợi.

Giải pháp biến cờ hiệu có thể đảm bảo cho việc mở rộng ra nhiều tiến trình, nhưng không thể đảm bảo đươc mutual exclusion. Vì bản thân biến lock cũng là một tài nguyên găng và việc truy cập biến lock không hề có sự kiểm tra để truy cập vào miền găng.

Giải pháp phần mềm – sử dụng biến luân phiên

Trong giải pháp này, một biến dùng chung được tạo ra giữa các tiến trình. Khi một tiến trình muốn truy cập vào miền găng. Tiến trình này sẽ kiểm tra biến luân phiên để xem có đến phiên truy cập của mình hay không. Nếu đúng phiên truy cập thì tiến trình sẽ được truy cập vào miền găng. Ngược lại tiến trình phải chờ đợi.

int turn = 1;

P_0	**P_1**
NonCS;	NonCS;
while (turn != 0); //wait	while (turn != 1); //wait
CS;	CS;
turn = 1;	turn = 0;
NonCS;	NonCS;

Hình 2.12. Sử dụng biến kiểm tra luân phiên

Trong hình 2.12 biến turn là biến kiểm tra luân phiên. Mặc định biến turn =1 cho phép tiến trình P_1 được quyền truy cập vào tiến trình. Sau khi P_1 hoàn thành, P_1 sẽ đặt lại turn = 0 để P_0 có thể truy cập vào miền găng.

Đối với giải pháp kiểm tra biến luân phiên có thể đảm bảo được mutual exclusion. Tuy nhiên, rất khó để có thể mở rộng cho nhiều tiến trình. Ngoài ra, một vấn đề quan trọng là khi tiến trình P_1 không vận hành thì tiến trình P_0 cũng không vận hành. Điều này vi phạm nguyên tắc không chặn tiến trình khác truy cập vào miền găng.

Giải pháp phần mềm – Peterson

Giải pháp Peterson là giải pháp kết hợp ý tưởng của 2 giải pháp biến luân phiên và biến cờ hiệu. Ý tưởng của giải pháp là các tiến trình sẽ có biến kiểm tra luân phiên và biến nhu cầu truy cập miền găng, đây là các biến dùng chung. Khi một tiến trình muốn truy cập vào miền găng, tiến trình phải báo nhu cầu truy cập miền găng. Đồng thời phải kiểm tra xem có đến lượt truy cập hay không. Nếu thỏa mãn đủ cả hai thì tiến trình được truy cập miền găng. Ngược lại thì phải chờ đợi. Sau khi hoàn thành, tiến trình phải đặt lại nhu cầu truy cập ở trạng thái không vào miền găng.

P_i **NonCS;**

j = 1 – i ;
interest[i] = TRUE;
turn =j;
while (turn == j && interest[j] == TRUE);

CS;
interest[i] = FALSE;
NonCS;

P_j **NonCS;**

i = 1 – j ;
interest[j] = TRUE;
turn =i;
while (turn == i && interest[i] == TRUE);

CS;
interest[j] = FALSE;
NonCS;

Hình 2.13. Sử dụng giải pháp Peterson

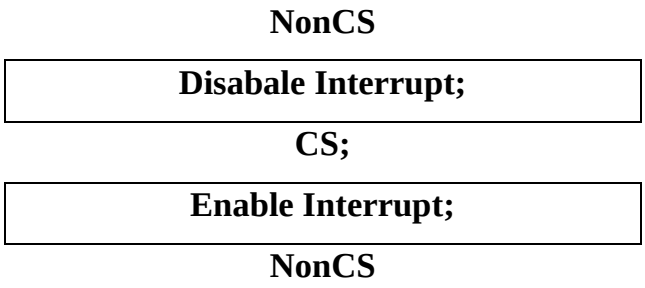
Trong hình 2.13, tiến trình P_i và P_j có biến interest, turn là các biến dùng chung. Tiến trình P_i khi thực hiện vào miền găng thì phải kiểm tra xem tiến trình P_j có nhu cầu truy cập hay không, và có đến lượt của P_j không. Nếu thỏa mãn thì tiến trình P_i phải chờ. Ngược lại một trong hai điều kiện không thỏa mãn thì tiến trình P_i được phép truy cập vào miền găng. Sau khi hoàn thành thì tiến trình P_i phải thiết lập lại interest[i] thành

FALSE để cho biết tiến trình P_i không còn nhu cầu truy cập miền găng. Tương tự với tiến trình P_j.

Giải pháp Peterson có thể đảm bảo mutual exclusion và không ngăn tiến trình khác truy cập vào miền găng. Tuy nhiên, rất khó để mở rộng ra nhiều tiến trình, chỉ thực hiện tốt khi có 2 tiến trình.

Giải pháp phần cứng

Để thực hiện giải pháp phần cứng, yêu cầu bắt buộc là phần cứng phải hỗ trợ việc thực hiện động bộ hóa. Các giải pháp phần cứng có thể kể đến như cấm ngắt (disable interupt) hay test and set lock. Các giải pháp này sử dụng các hàm trong các thư viện được phần cứng hỗ trợ. Khi đó, các tiến trình muốn truy cập vào miền găng sẽ nhờ các hàm này kiểm tra và xác lập quyền truy cập vào miền găng.



Hình 2.14. Sử dụng giải pháp disable interupt

Trong hình 2.14, các tiến trình sử dụng giải pháp cấm ngắt. Khi 1 tiến trình đi vào miền găng thì tiến trình sẽ thực hiện cấm ngắt để không cho CPU có thể ngắt sự hoạt động của tiến trình, cho đến khi tiến trình hoàn thành miền găng.

Các giải pháp phần cứng nhìn chung có thể mở rộng dễ dàng cho nhiều tiến trình, tuy nhiên cần phải có được sự hỗ trợ của phần cứng. Ngoài ra, một vấn đề là việc thực hiện câu lệnh ở miền găng phải rất thận trọng. Vì nếu xảy ra vấn đề xử lý ở miền găng, dẫn đến lặp vô tận chẳng hạn, thì tiến trình sẽ không được kết thúc do đã thực hiện cấm ngắt.

Giải pháp semaphore

Semaphore là một giải pháp rất quan trọng dùng trong việc đồng bộ các tiến trình. Giải pháp này được Dijkstra đề xuất vào năm 1965. Ý tưởng của giải pháp là semaphore sử dụng 1 biến nguyên dương s>=0. Biến này được thao tác bởi 2 hàm Up và Down.

- Hàm Down khi được gọi sẽ giảm biến s 1 giá trị. Nếu biến s <0 tiến trình sẽ được đưa vào hàng chờ và đưa vào trạng thái sleep.
- Hàm Up khi được gọi sẽ tăng biến s lên 1 giá trị. Nếu biến s<=0 tiến trình sẽ được đưa ra khỏi hàm chờ và được đánh thức khỏi trạng thái sleep để hoạt động.

Khi thực hiện semaphore, tùy theo nhu cầu đồng bộ mà phối hợp 2 hàm Up và Down. Khi muốn quản lý việc truy cập tài nguyên dùng chung giữa các tiến trình để đảm bảo rằng 1 thời điểm chỉ có 1 tiến trình truy cập. Việc thiết lập semaphore có thể được thiết lập như sau:

Tiến trình P_i

Thiết lập giá trị Semaphore ban đầu s=1

Thực hiện kiểm tra quyền truy cập ở hàm Down và Up.

Down(s)

Truy cập vào tài nguyên dùng chung

Up(s)

Với cách thiết lập trên, giả sử có 2 tiến trình cùng hoạt động, khi đó cả 2 tiến trình đều gọi hàm Down(s). Tiến trình nào đến sớm hơn, sẽ giảm đi s 1 giá trị, lúc này s=0. Khi đó, hàm Down không được thực thi vì điều kiện kiểm tra là s<0. Khi đó, tiến trình thứ 2 sẽ giảm tiếp s 1 giá trị, lúc này s=-1. Như vậy hàm Down được thực thi và đưa tiến trình thứ 2 vào hàng chờ và ép vào trạng thái sleep. Sau khi tiến trình thứ nhất hoàn thành, nó sẽ dùng hàm Up(s) để tăng 1 giá trị của s, lúc này s=0, đồng thời đánh thức tiến trình thứ 2 ra khỏi hàng chờ và truy cập vào miền găng.

Xét đoạn code sau để hiểu rõ hơn về cách sử dụng semaphore.

Ví dụ 2.10

```
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>
sem_t sem; //khai báo biến semaphore là sem
// Định nghĩa công việc của thread
void* thread(void* arg)
{
// Hàm sem_wait thục hiện công việc của hàm Down như ý tưởng giải
// pháp.
     sem_wait(&sem);
     printf("\nEntered..\n");
//Truy cập vào miền găng
```

```
      sleep(4);
      printf("\nJust Exiting...\n");
// Hàm sem_post thực hiện công việc của hàm Up như ý tưởng giải
// pháp.
      sem_post(&sem);
}
int main()
{
      sem_init(&sem, 0, 1); // khởi tạo biến sem với giá trị bằng 1
// Định nghĩa ra 2 luồng và khởi chạy 2 luồng
      pthread_t t1,t2;
      pthread_create(&t1,NULL,thread,NULL);
      pthread_create(&t2,NULL,thread,NULL);
      pthread_join(t1,NULL);
      pthread_join(t2,NULL);
// Hủy semaphore sau khi sử dụng.
      sem_destroy(&sem);
      return 0;
}
```

Trong cách sử dụng trên, tại 1 thời điểm duy nhất, chỉ có 1 luồng duy nhất truy cập vào tài nguyên dùng chung.

Ngoài cách quản lý như đề cập ở trên, semaphore còn có thể phối hợp giữa 2 tiến trình để cùng thực hiện công việc. Xem xét đoạn chương trình dưới đây để hiểu rõ cách thức phối hợp hành động của 2 tiến trình bằng giải pháp monitor.

Ví dụ 2.11

```
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>
sem_t sem1,sem2; // Khai báo 2 biến semaphore sem1 và sem2
void* inchan(void* arg)
{
      int i;
      for(i=2;i<13;i=i+2)
      {
// Sử dụng semaphore để phối hợp giữa 2 luồng
            sem_wait(&sem1);
            printf("thread 1: %d \n",i);
            sem_post(&sem2);
      }
}
void* inle(void* arg)
{  int i;
   for(i=1;i<13;i=i+2)
      {
// Sử dụng semaphore để phối hợp giữa 2 luồng
            sem_wait(&sem2);
            printf("thread 2: %d \n",i);
```

```
            sem_post(&sem1);
        }
    }
int main()
{
    sem_init(&sem1, 0, 0); // khởi tạo sem1 = 0
    sem_init(&sem2, 0, 1); // khởi tạo sem2 = 1
// Tạo ra 2 luồng và gọi công việc tương ứng.
    pthread_t t1,t2;
    pthread_create(&t2,NULL,inle,NULL);
    pthread_create(&t1,NULL,inchan,NULL);
    pthread_join(t1,NULL);
    pthread_join(t2,NULL);
//Hủy sem1 và sem2
    sem_destroy(&sem1);
    sem_destroy(&sem2);
    return 0;
}
```

Chương trình trên khi khởi chạy sẽ lần lượt gọi luồng inle rồi đền luồng inchan, thực hiện 1 cách tuần tự để tiến trình sẽ in ra kết quả lần lượt là 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12.

Ngoài giải pháp Semaphore, người ta còn quản lý đồng bộ bằng giải pháp Monitor, là một cơ chế đồng bộ hóa do ngôn ngữ lập trình hỗ trợ với những ý tưởng và cách thức thực hiện tương tự như semaphore.

2.6. Tắc nghẽn

2.6.1. Định nghĩa

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

2.6.2. Điều kiện xảy ra tắc nghẽn

Các điều kiện cần có thể làm xuất hiện tắc nghẽn:

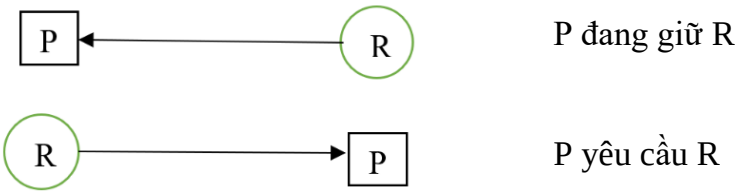
- Điều kiện 1: Có sử dụng tài nguyên không thể chia sẻ
- Điều kiện 2: Sự chiếm giữ và yêu cầu thêm tài nguyên
- Điều kiện 3: Không thể thu hồi tài nguyên từ tiến trình đang giữ chúng.

- Điều kiện 4: Tồn tại một chu trình trong đồ thị cấp phát tài nguyên

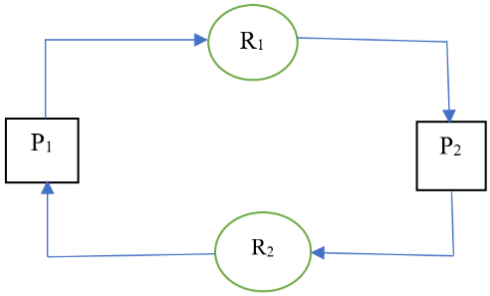
Khi có đủ 4 điều kiện này thì tắc nghẽn xảy ra; nếu thiếu một trong 4 điều kiện này thì không có tắc nghẽn.

2.6.3. Đồ thị cấp phát tài nguyên

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



Sau đây là một ví dụ về một tình huống tắc nghẽn



Đồ thị này gọi là đồ thị cấp phát tài nguyên (resource allocation graph)

2.6.4. Các phương pháp xử lý tắc nghẽn

Chủ yếu có 3 hướng tiếp cận sau để xử lý tắc nghẽn:

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

2.6.5. Trạng thái an toàn

Tránh tắc nghẽn

Ngăn cản tắc nghẽn là mối bận tâm khi sử dụng tài nguyên. Tránh tắc nghẽn là loại bỏ tất cả các cơ hội có thể dẫn đến tắc nghẽn trong tương lai. Hệ điều hành sử dụng những cơ chế phức tạp để giải quyết vấn đề này.

Chuỗi cấp phát an toàn (safe sequence)

Một thứ tự các tiến trình <P_1, P_2,…,P_n> là an toàn (hay còn được gọi là **Chuỗi cấp phát an toàn**) đối với tình trạng cấp phát hiện hành nếu với mỗi tiến trình P_i, nhu cầu tài nguyên của Pi có thể được thỏa mãn với các tài nguyên còn tự do của hệ thống, cộng với các tài nguyên đang bị chiếm giữ bởi các tiến trình P_j khác, với j<i.

Trạng thái an toàn (safe)

Một trạng thái của hệ thống được gọi là *an toàn* nếu tồn tại một *chuỗi an toàn*.

Trạng thái không an toàn (unsafe)`

Một trạng thái của hệ thống được gọi là *không an toàn* nếu nó không tồn tại một chuỗi an toàn.

Chiến lược cấp phát:

Chỉ thỏa mãn tài nguyên yêu cầu của tiến trình khi trạng thái kết thúc là an toàn.

2.6.6. Thuật toán Banker

Giải thuật xác định trạng thái an toàn

Cần sử dụng các cấu trúc dữ liệu sau:

int available[numresources]

available[r]: số lượng các thể hiện còn tự do của tài nguyên r.

int max[numprocs, numresources]

max[p,r]: nhu cầu tối đa của tiến trình p đối với tài nguyên r.

int allocation[numprocs, numresources]

allocation[p,r]:số tài nguyên r thực sự cấp phát cho tiến trình p.

int need[numprocs, numresources]

need[p,r] = max[p,r]- allocation[p,r];

Giải thuật xác định trạng thái an toàn

Bước 1:

Giả sử có các mảng:

int work [numprocs, numresources]= available;

int finish[numprocs]=false;

Bước 2:

Tìm i sao cho:

finish[i]==false và need[i] <= work[i]

Nếu không tìm thấy i thỏa mãn thì chuyển đến bước 4;

Bước 3:

work= work+ allocation[i];

finish[i]=true;

Chuyển đến bước 2;

Bước 4:

Nếu finish[i]==true với mọi *i* thì hệ thống ở trạng thái an toàn.

Ví dụ 2.12

Hệ thống gồm 5 tiến trình P_0, P_1, P_2, P_3, P_4 và 3 loại tài nguyên R_1, $R2$, R_3; trong đó R_1 có 10 thể hiện, R_2 có 5 thể hiện, R_3 có 7 thể hiện.

Giả sử trạng thái của hệ thống tại thời điểm T_0 được mô tả như sau:

Process	Max			Allocation			Available		
	R_1	R_2	R_3	R_1	R_2	R_3	R_1	R_2	R_3
P_0	7	5	3	0	1	0	3	3	2
P_1	3	2	2	2	0	0			
P_2	9	0	2	3	0	2			
P_3	2	2	2	2	1	1			
P_4	4	3	3	0	0	2			

Tại thời điểm T_0 hệ thống có ở trạng thái an toàn hay không ?

Process	Need		
	R_1	R_2	R_3
P_0	7	4	3
P_1	1	2	2
P_2	6	0	0
P_3	0	1	1
P_4	4	3	1

Process	Work		
	R_1	R_2	R_3
	3	3	2
P_1	5	3	2
P_3	7	4	3
P_4	7	4	5
P_0	7	5	5
P_2	10	5	7

Hệ thống trên ở trạng thái an toàn vì nó tồn tại thứ tự an toàn P_1,P_3,P_4,P_0,P_2.

Giải thuật phát hiện tắc nghẽn

Cần sử dụng các cấu trúc dữ liệu sau:

int available[numresources]

available[r]: số lượng các thể hiện còn tự do của tài nguyên r.

int allocation[numprocs, numresources]

allocation[p,r]: số tài nguyên r thực sự cấp phát cho tiến trình p.

int request[numprocs, numresources]

request[p,r]: số tài nguyên r tiến trình p yêu cầu thêm.

Giải thuật phát hiện tắc nghẽn

Bước 1:

int work [numresources]=available;

int finish[numprocs];

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

finish[i]=(allocation[i]==0);

Bước 2:

Tìm *i* sao cho finish[i]==false và request[i]<=work;

Nếu không có i thỏa mãn, chuyển đến bước 4;

Bước 3:

work= work+allocation[i];

finish[i]=true;

Chuyển đến bước 2;

Bước 4:

Nếu finish[i]==true với mọi *i* thì hệ thống không có tắc nghẽn;

Nếu finish[i]==false với một giá trị *i* thì các tiến trình mà finish[i]==false sẽ ở trong tình trạng tắc nghẽn.

Ví dụ 2.13

Cho một hệ thống gồm 5 tiến trình P_0,P_1,P_2,P_3,P_4 và 3 loại tài nguyên R_1,R_2,R_3; trong đó R_1 có 7 thể hiện, R_2 có 2 thể hiện, R_3 có 6 thể hiện.

Giả sử trạng thái của hệ thống tại thời điểm T_0 được mô tả như sau:

	Request			Allocation			Available		
	R_1	R_2	R_3	R_1	R_2	R_3	R_1	R_2	R_3
P_0	0	0	0	0	1	0	0	0	0
P_1	2	0	2	2	0	0			
P_2	0	0	0	3	0	3			
P_3	1	0	0	2	1	1			
P_4	0	0	2	0	0	2			

Tại thời điểm T_0 hệ thống có ở trạng thái an toàn hay không ?

Process	Work		
	R_1	R_2	R_3
	0	0	0
P_0	0	1	0
P_2	3	1	3
P_3	5	2	4
P_4	5	2	6
P_1	7	2	6

Vậy hệ thống không bị deadlock (hệ thống an toàn) vì tồn tại một thứ tự an toàn P_0,P_2,P_3,P_4,P_1.

Giải thuật yêu cầu tài nguyên

Giải thuật yêu cầu tài nguyên

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

Bước 1:

Nếu *k*<=need[i], chuyển đến bước 2;

ngược lại, xảy ra tình huống lỗi;

Bước 2:

Nếu *k*<=available [i], chuyển đến bước 3;

ngược lại, P_i phải chờ;

Bước 3:

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

available [i]= available [i]-k;

allocation[i[=allocation[i[+k;

need[i]= need[i]-k;

Nếu trạng thái kết thúc là an toàn, lúc này các tài nguyên trên sẽ được cấp phát thật sự cho P_i; ngược lại, P_i phải chờ.

Ví dụ 2.14

Hệ thống gồm 5 tiến trình P_0, P_1, P_2, P_3, P_4 và 3 loại tài nguyên R_1, R_2, R_3; trong đó R_1 có 10 thể hiện, R_2 có 5 thể hiện, R_3 có 7 thể hiện.

Giả sử trạng thái của hệ thống tại thời điểm T_0 được mô tả như sau:

Process	Max			Allocation			Available		
	R_1	R_2	R_3	R_1	R_2	R_3	R_1	R_2	R_3
P_0	7	5	3	0	1	0	3	3	2
P_1	3	2	2	2	0	0			
P_2	9	0	2	3	0	2			
P_3	2	2	2	2	1	1			
P_4	4	3	3	0	0	2			

Giả sử P_1 yêu cầu cấp phát với số thể hiện (1, 0, 2). Hỏi yêu cầu này có được thỏa mãn hay không ?

Tương tự với P_4(3, 3, 0), P_0(0, 2, 0), P_3(0, 2, 1).

Tiến trình P_1 yêu cầu thêm (1,0,2) thể hiện, ta cần cập nhật lại trạng thái allocation của P_1 là (3,0,2).

Kiểm tra các điều kiện:

Request[1](1,0,2) <= Available (3,3,2) - >True

Request[1](1,0,2) <= Need[1] (1,2,2) - >True

(Lưu ý: Nếu một trong hai điều kiện trên là sai thì kết luận việc yêu cầu cấp phát thêm không thành công; nếu cả hai điều kiện đều đúng thì mới tiến hành tính các ma trận Need và Work).

Tính ma trận Need: giữ nguyên giá trị cho các tiến trình P_0, P_2, P_3, P_4; chỉ cập nhật thông tin của P_1.

Process	Need		
	R_1	R_2	R_3
P_0	7	4	3
P_1	0	2	0
P_2	6	0	0
P_3	0	1	1
P_4	4	3	1

Process	Work		
	R_1	R_2	R_3
	2	3	0
P_1	5	3	2
P_3	7	4	3
P_4	7	4	5
P_0	7	5	5
P_2	10	5	7

Tồn tại chuỗi an toàn P_1,P_3,P_4,P_0,P_2. Vậy yêu cầu cấp phát thêm của P_1 là thành công.

Bài tập chương 2

Bài tập 2.1.

Trình bày định nghĩa tiến trình. Nêu các trạng thái của tiến trình. Trình bày sơ đồ chuyển trạng thái tiến trình.

Bài tập 2.2.

Cho bảng tình trạng sau:

Process	Arrival time	Service time
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5

a. Tìm thứ tự cấp phát CPU, thời gian chờ trung bình theo thuật toán SJF áp dụng điều phối theo nguyên tắc độc quyền.

b. Tìm thứ tự cấp phát CPU, thời gian chờ trung bình theo thuật toán SRTF áp dụng điều phối theo nguyên tắc không độc quyền.

Bài tập 2.3.

Sử dụng các giải thuật FCFS, SJF, SRTF, Priority để tính các giá trị thời gian chờ trung bình, thời gian đáp ứng trung bình và thời gian hoàn thành trung bình của bảng tình trạng sau:

Process	Burst time	Priority
P_1	2	2
P_2	1	1
P_3	8	4
P_4	4	2
P_5	5	3

Bài tập 2.4.

Sử dụng các thuật toán FCFS, SJF, SRTF, Priority để tính các giá trị thời gian chờ trung bình, thời gian đáp ứng trung bình và thời gian hoàn thành trung bình của bảng tình trạng sau:

Process	Priority	Burst	Arrival time
P_1	40	20	0
P_2	30	25	25
P_3	30	25	30
P_4	35	15	60
P_5	5	10	100
P_6	10	10	105

Bài tập 2.5.

Sử dụng các giải thuật FCFS, SJF, SRTF, Priority -Pre, RR (10) để tính các giá trị thời gian chờ trung bình, thời gian đáp ứng trung bình, thời gian hoàn thành trung bình và vẽ biểu đồ Gaint

Process	Arrival time	Burst	Priority
P_1	0	20	20
P_2	25	25	30
P_3	20	25	15
P_4	35	15	35
P_5	10	35	5
P_6	15	50	10

Bài tập 2.6.

Cho 5 tiến trình với thời gian vào và thời gian cần CPU tương ứng như bảng sau:

Process	Arrival time	Burst
P_1	0	10
P_2	2	29
P_3	4	3
P_4	5	7
P_5	7	12

Vẽ sơ đồ Gantt và tính thời gian đợi trung bình, thời gian đáp ứng trung bình và thời gian lưu lại trong hệ thống (turnaround time) trung bình cho các thuật toán sau:

a. FCFS

b. SJF preemptive (trưng dụng CPU, tức không độc quyền)

c. RR với quantum time = 10

Bài tập 2.7.

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

Process	Arrival time	Burst	Priority
P_1	0	10	3
P_2	1	3	2
P_3	2	2	1
P_4	3	1	2
P_5	4	5	4

Vẽ sơ đồ Gantt và tính thời gian chờ trung bình và thời gian lưu lại trong hệ thống trung bình (turnaround time) cho các thuật toán sau:

a. SFJ Preemptive

b. RR với quantum time = 2

c. Điều phối theo độ ưu tiên độc quyền (độ ưu tiên 1 > 2 >...)

Bài tập 2.8.

Tất cả process đều đến ở thời điểm 0 theo thứ tự từ P_1 đến P_5. Vẽ sơ đồ Gantt và tính thời gian chờ trung bình và thời gian lưu lại trong hệ thống (turnaround time) trung bình cho các thuật toán sau:

Process	Burst time
P_1	10
P_2	29
P_3	3
P_4	7
P_5	12

a. FCFS, SFJ

b. RR với quantum time = 10

Bài tập 2.9.

Cho 4 tiến trình và thời gian vào (Arrival Time) tương ứng

Process	Arrival time	Burst
P_1	0	12
P_2	2	7
P_3	3	5
P_4	5	9

Vẽ sơ đồ Gannt và tính thời gian chờ trung bình (average wait time) và thời gian xoay vòng (average turnaround time) trung bình cho các thuật toán định thời CPU sau;

a. Shortest Remaining Time First (SRTF)

b. Round Robin (RR) với quantum time = 4

Bài tập 2.10.

Cho 5 tiến trình P_1, P_2, P_3, P_4, P_5 với thời gian vào Ready List vào thời gian cần CPU tương ứng như bảng sau:

Process	Arrival time	CPU Burst time
P_1	0	8
P_2	2	19
P_3	4	3
P_4	5	6
P_5	7	12

Vẽ sơ đồ Gannt và tính thời gian chờ trung binh, thời gian đáp ứng trung bình và thời gian lưu lại trong hệ thống (turnaround time) trung bình cho các thuật toán sau:

a. FCFS

b. SJF preemptive

c. RR với quantum time = 6

Bài tập 2.11.

Giả sử tình trạng của hệ thống được mô tả như sau:

Max			
Process	R_1	R_2	R_3
P_1	3	2	2
P_2	6	1	3
P_3	3	1	4
P_4	4	2	2

Allocation			
Process	R_1	R_2	R_3
P_1	1	0	0
P_2	2	1	1
P_3	2	1	1
P_4	0	0	2

Availabe		
R_1	R_2	R_3
4	1	2

Tiến trình P_2 yêu cầu 4 cho R_1, 1 cho R_3. Hãy cho biết yêu cầu này có thể đáp ứng mà bảo đảm không xảy ra tình trạng deadlock hay không ?

Bài tập 2.12.

Cho một hệ thống có 4 tiến trình P_1,P_2,P_3,P_4 và 3 loại tài nguyên R_1 (3), R_2 (2), R_3 (2); trong đó P_1 giữ 1 tài nguyên R_1 và yêu cầu 1 tài nguyên R_2, P_2 giữ 2 tài nguyên R_2 và yêu

cầu 1 tài nguyên R_1 và 1 tài nguyên R_3, P_3 giữ 1 tài nguyên R_1 và yêu cầu 1 tài nguyên R_2, P_4 giữ 2 tài nguyên R_3 và yêu cầu 1 tài nguyên R_1.

a. Hãy vẽ đồ thị tài nguyên cho hệ thống này.

b. Hỏi có xảy ra tình trạng deadlock hay không ?

c. Hãy chỉ một chuỗi an toàn (nếu có).

Bài tập 2.13.

Giả sử tình trạng của hệ thống được mô tả như sau:

Max				
Process	*A*	*B*	*C*	*D*
P_0	0	0	1	2
P_1	1	7	5	0
P_2	2	3	5	6
P_3	0	6	5	2
P_4	0	6	5	6

Allocation				
Process	*A*	*B*	*C*	*D*
P_0	0	0	1	2
P_1	1	0	0	0
P_2	1	3	5	4
P_3	0	6	3	2
P_4	0	0	1	4

Availabe			
A	*B*	*C*	*D*
1	5	2	0

a. Tìm bảng Need

b. Hệ thống có an toàn không ?

c. Nếu P_1 yêu cầu (0,4,2,0) thì có thể cấp phát cho nó ngay không ?

d. Nếu P_1 yêu cầu (0,4,3,0) thì có thể cấp phát cho nó ngay không ?

Bài tập 2.14.

Sử dụng thuật toán Banker xem các trạng thái sau có an toàn hay không ? Nếu có thì đưa ra chuỗi thực thi an toàn, nếu không thì nêu rõ lý do không an toàn ?

Max				
Process	*A*	*B*	*C*	*D*
P_0	5	1	1	7
P_1	3	2	1	1
P_2	3	3	2	1
P_3	4	6	1	2
P_4	6	3	2	5

Allocation				
Process	*A*	*B*	*C*	*D*
P_0	3	0	1	4
P_1	2	2	1	0
P_2	3	1	2	1
P_3	0	5	1	0
P_4	4	2	1	2

a. Available = (0,3,0,1)

b. Available = (1,0,0,2)

Bài tập 2.15.

Trả lời các câu hỏi sau sử dụng giải thuật Banker:

a. Hệ thống có an toàn không ? Đưa ra chuỗi an toàn nếu có ?

b. Nếu P_1 yêu cầu (1,1,0,0) thì có thể cấp phát cho nó ngay không ?

c. Nếu P_4 yêu cầu (0,0,2,0) thì có thể cấp phát cho nó ngay không ?

Max				
Process	*A*	*B*	*C*	*D*
P_0	4	2	1	2
P_1	5	2	5	2
P_2	2	3	1	6
P_3	1	4	2	4
P_4	3	6	6	5

Allocation				
Process	*A*	*B*	*C*	*D*
P_0	2	0	0	1
P_1	3	1	2	1
P_2	2	1	0	3
P_3	1	3	1	2
P_4	1	4	3	2

Availabe			
A	*B*	*C*	*D*
3	3	2	1

Bài tập 2.16.

a. Một hệ thống có 3 tiến trình P_1, P_2, P_3 và 3 ổ băng từ với trạng thái cấp phát tài nguyên ở thời điểm T_i thể hiện bằng véc-tơ Allocation = (1, 0, 1) và Max = (1, 2, 2). Hãy chứng minh trạng thái {P_1, P_2, P_3} an toàn.

b. Một hệ thống có 3 tiến trình P_1, P_2, P_3 và 3 ổ băng từ với trạng thái cấp phát tài nguyên tại thời điểm *Ti* thể hiện bằng các véc-tơ Allocation=(0, 2, 1) và Max=(2,2, 2). Hãy chứng minh trạng thái P_1, P_2, P_3 an toàn.

Bài tập 2.17.

Cài đặt các thuật toán điều phối tiến trình

Bài tập 2.18.

Cài đặt các dạng của thuật toán Banker

Bài tập 2.19.

Cho 4 tiến trình P_1, P_2, P_3, P_4 có thời điểm vào (arrival time) và thời gian xử lý của CPU (burst time; đơn vị milliseconds) như bảng sau:

Process	arrival time	burst time
P_1	0	7
P_2	1	5
P_3	2	1
P_4	3	2

Hãy vẽ biểu đồ Gantt (Gantt chart) để tìm thứ tự cấp phát CPU cho các tiến trình và cho biết thời gian chờ trung bình (average waiting time) của các tiến trình theo chiến lược điều phối "*công việc ngắn nhất được thực hiện trước*" với hai dạng sau:

a. Shortest job first scheduling (SJF); SJF làm việc theo nguyên tắc độc quyền.
b. Shortest remaining time first scheduling (SRTF); SRTF làm việc theo nguyên tắc không độc quyền.

Bài tập 2.20.

Một hệ thống có 5 tiến trình P_0, P_1, P_2, P_3, P_4 và 3 loại tài nguyên *A*, *B*, *C*; trong đó tài nguyên *A* có 9 thể hiện (instances), tài nguyên *B* có 8 thể hiện, tài nguyên *C* có 7 thể hiện.

Giả sử trạng thái của hệ thống tại thời điểm T_0 như sau:

Process	Allocation			Max			Available		
	A	*B*	*C*	*A*	*B*	*C*	*A*	*B*	*C*
P_0	2	1	2	4	2	3	1	1	1
P_1	1	2	1	2	2	1			
P_2	3	1	1	7	1	2			
P_3	1	2	0	6	4	2			
P_4	1	1	2	2	1	2			

Sử dụng thuật toán *Banker* hãy thực hiện các công việc sau:

a. Tìm bảng Need.

b. Xác định xem hệ thống tại thời điểm T_0 có ở trạng thái an toàn hay không ?

Bài tập 2.21.

Cho 4 tiến trình P_1, P_2, P_3, P_4 có thời điểm vào (arrival time) và thời gian xử lý của CPU (burst time; đơn vị milliseconds) như bảng sau:

Process	arrival time	burst time
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5

Hãy vẽ biểu đồ Gantt (Gantt chart) để tìm thứ tự cấp phát CPU cho các tiến trình và cho biết thời gian chờ trung bình (average waiting time) của các tiến trình theo chiến lược điều phối "*công việc ngắn nhất được thực hiện trước*" với hai dạng sau:

a. Shortest job first scheduling (SJF); SJF làm việc theo nguyên tắc độc quyền.

b. Shortest remaining time first scheduling (SRTF); SRTF làm việc theo nguyên tắc không độc quyền.

Bài tập 2.22.

Một hệ thống có 5 tiến trình P_0, P_1, P_2, P_3, P_4 và 3 loại tài nguyên *A*, *B*, *C*; trong đó tài nguyên *A* có 7 thể hiện (instances), tài nguyên *B* có 2 thể hiện, tài nguyên *C* có 6 thể hiện.

Giả sử trạng thái của hệ thống tại thời điểm T_0 như sau:

Process	Allocation			Max			Available		
	A	*B*	*C*	*A*	*B*	*C*	*A*	*B*	*C*
P_0	0	1	0	0	1	0	0	0	0
P_1	2	0	0	4	0	2			
P_2	3	0	3	3	0	3			
P_3	2	1	1	3	1	1			
P_4	0	0	2	0	0	4			

Sử dụng thuật toán *Banker* hãy thực hiện các công việc sau:

a. Tìm bảng Need.

b. Xác định xem hệ thống tại thời điểm T_0 có ở trạng thái an toàn hay không ?

Chương 3. Quản lý bộ nhớ

3.1. Không gian địa chỉ và không gian vật lý

3.1.1. Các loại địa chỉ

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

Địa chỉ logic (còn gọi là địa chỉ ảo)

Là tất cả các địa chỉ do bộ xử lý tạo ra.

Địa chỉ vật lý (địa chỉ thực)

Là địa chỉ mà trình quản lý bộ nhớ nhìn thấy và thao tác.

Không gian địa chỉ

Là tập hợp tất cả các địa chỉ ảo phát sinh bởi một chương trình;

Không gian vật lý

Là tập hợp tất cả các địa chỉ vật lý tương ứng với các địa chỉ ảo.

Địa chỉ ảo và địa chỉ vật lý là như nhau trong phương thức kết buộc địa chỉ vào thời điểm biên dịch cũng như thời điểm nạp chương trình vào bộ nhớ; nhưng có sự khác biệt giữa địa chỉ ảo và địa chỉ vật lý trong phương thức kết buôc vào thời điểm xử lý.

MMU (memory – management unit) là một cơ chế phần cứng được sử dụng để thực hiện chuyển đổi địa chỉ ảo thành địa chỉ vật lý vào thời điểm xử lý. Chương trình của người sử dụng chỉ thao tác trên các địa chỉ ảo, mà không làm việc trên địa chỉ vật lý.

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

3.1.2. Các vấn đề cần xem xét trong việc quản lý bộ nhớ

Bộ nhớ chính là thiết bị lưu trữ duy nhất thông qua đó CPU có thể trao đổi thông tin với môi trường bên ngoài. Bộ nhớ chính được tổ chức như một mảng một chiều các từ nhớ (word), mỗi từ nhớ có một địa chỉ. Việc trao đổi thông tin với môi trường bên ngoài được thực hiện thông qua các thao tác đọc hoặc ghi dữ liệu vào một địa chỉ cụ thể nào đó trong bộ nhớ.

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

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

Sự tương ứng giữa địa chỉ logic và địa chỉ vật lý (physis):

Làm thế nào để chuyển đổi một địa chỉ tượng trưng trong chương trình thành một địa chỉ thực trong bộ nhớ chính.

Quản lý bộ nhớ vật lý:

Làm cách nào để mở rộng bộ nhớ có sẵn nhằm lưu trữ được nhiều tiến trình đồng thời ?

Chia sẻ thông tin:

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

Bảo vệ:

Làm thế nào để ngăn chặn các tiến trình xâm phạm đến vùng nhớ được cấp phát cho tiến trình khác ?

Các giải pháp quản lý bộ nhớ phụ thuộc rất nhiều vào đặc tính phần cứng và trãi qua nhiều giai đoạn cải tiến để trở thành những giải pháp thỏa đáng như hiện này.

Bộ nhớ là tài nguyên quan trọng của hệ thống máy tính, bộ nhớ được đặc trưng bởi kích thước và tốc độ truy cập. Quản lý bộ nhớ là một trong những nhiệm vụ trọng tâm hàng đầu của hệ điều hành.

3.1.3. Các thời điểm kết buộc các chỉ thị và dữ liệu với các địa chỉ bộ nhớ

Thông thường, một chương trình được lưu trữ trên đĩa như một tập tin nhị phân có thể xử lý. Để thực hiện chương trình, cần nạp chương trình vào bộ nhớ chính, tạo lập tiến trình tương ứng để xử lý.

Hàng đợi nhập hệ thống là tập hợp các chương trình trên đĩa đang chờ được nạp vào bộ nhớ để tiến hành xử lý. Các địa chỉ trong chương trình nguồn là địa chỉ tượng trưng, Do vậy, một chương trình phải trải qua nhiều giai đoạn xử lý để chuyển đổi các địa chỉ này thành các địa chỉ tuyệt đối trong bộ nhớ chính.

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

Thời điểm biên dịch

Nếu tại thời điểm biên dịch, có thể biết vị trí mà tiến trình sẽ thường trú trong bộ nhớ, trình biên dịch có thể phát sinh ngay mã với địa chỉ tuyệt đối. Tuy nhiên, nếu về sau có sự thay đổi vị trí thường trú lúc đầu của chương trình, cần phải biên dịch lại chương trình.

Thời điểm chương trình được nạp vào bộ nhớ

Nếu tại thời điểm biên dịch, chưa thể biết vị trí mà tiến trình sẽ thường trú trong bộ nhớ, trình biên dịch cần phát sinh mã tương đối. Sự liên kết địa chỉ được trì hoãn đến thời điểm chương trình được nạp vào bộ nhớ, lúc này các địa chỉ tương đối sẽ được chuyển thành địa chỉ tuyệt đối do đã biết vị trí ban đầu lưu trữ tiến trình. Khi có sự thay đổi vị trí lưu trữ, chỉ cần nạp lại chương trình để tính toán lại các địa chỉ tuyệt đối, mà không cần biên dịch lại.

Thời điểm xử lý

Nếu có nhu cầu di chuyển tiến trình từ vùng nhớ này sang vùng nhớ khác trong quá trình tiến trình xử lý, thì thời điểm kết buộc địa chỉ phải trì hoãn đến tận thời điểm xử lý. Để thực hiện kết buộc địa chỉ vào thời điểm xử lý, cần sử dụng cơ chế phần cứng đặc biệt.

3.2. Các chiến lược cấp phát liên tục

3.2.1. Các hệ đơn chương

Ý tưởng

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

Thảo luận

- Khi bộ nhớ được tổ chức theo cách thức này, chỉ có thể xử lý một chương trình tại một thời điểm.
- Đa số các tiến trình trải qua phần lớn thời gian để chờ các thao tác nhập, xuất hoàn thành; khi đó CPU nhàn rỗi; như vậy hệ thống đơn chương làm cho việc

sử dụng CPU không đạt hiệu suất cao. Hơn nữa, các hệ đơn chương không cho phép nhiều người sử dụng làm việc đồng thời theo chế độ tương tác. Để nâng cao hiệu suất sử dụng CPU, cần cho phép chế độ đa chương; trong đó các tiến trình chia sẻ CPU với nhau để hoạt động đồng hành.

3.2.2. Các hệ thống đa chương với phân vùng cố định

Ý tưởng

Bộ nhớ được chia thành n phân vùng (chương) (partition) có kích thước cố định; trong đó các phân vùng khác nhau có thể có kích thước khác nhau hay bằng nhau. Các tiến trình có nhu cầu bộ nhớ sẽ được lưu trữ trong hàng đợi: có thể sử dụng một hoặc nhiều hàng đợi.

- Sử dụng một hàng đợi: tất cả các tiến trình được đặt trong một hàng đợi duy nhất. Khi có một phân vùng tự do, tiến trình đầu tiên trong hàng đợi có kích thước phù hợp sẽ được đặt vào phân vùng này và cho xử lý.
- Sử dụng nhiều hàng đợi: mỗi phân vùng sẽ có một hàng đợi tương ứng, một tiến trình mới được tạo lập sẽ được đưa vào hàng đợi của phân vùng có kích thước nhỏ nhất đủ thỏa mãn nhu cầu chứa nó. Cách tổ chức này để lộ khuyết điểm trong trường hợp các hàng đợi của một số phân vùng thì trống nhưng hàng đợi của một số phân vùng khác lại đầy, buộc các tiến trình trong các hàng đợi này phải chờ được cấp phát bộ nhớ.

Thảo luận

- Nếu kích thước của tiến trình không vừa đúng bằng kích thước phân vùng chứa nó, phần bộ nhớ không sử dụng đến trong phân vùng sẽ bị lãng phí. Hiện tượng này được gọi là phân mảnh nội vi (internal fragmentation).
- Mức độ đa chương của hệ thống bị giới hạn bởi số lượng phân vùng.
- Khi cho phép đa chương, cần giải quyết hai vấn đề: sự tái định vị (relocation) và sự bảo vệ các tiến trình. Trong các hệ thống đa chương, các tiến trình khác nhau sẽ thi hành trên các vùng địa chỉ vật lý khác nhau. Khi liên kết một chương trình (kết hợp chương trình chính với các module vào cùng một không

gian địa chỉ), trình liên kết (linker) cần phải biết chương trình sẽ được nạp vào địa chỉ nào trong bộ nhớ. Đây là vấn đề tái định vị. Ngoài ra, khi có nhiều tiến trình đồng thời trong bộ nhớ, làm thế nào để ngăn chặn các tiến trình xâm phạm lẫn nhau, đây là vấn đề bảo vệ.

3.2.3. Các hệ thống đa chương với phân vùng động

Ý tưởng

Khi một tiến trình được đưa vào hệ thống, cấp phát cho tiến trình một vùng nhớ vừa đúng với kích thước của tiến trình, phần bộ nhớ còn lại dành cho các tiến trình khác. Khi một tiến trình kết thúc, vùng nhớ đã cấp phát cho nó sẽ được giải phóng và có thể được cấp phát cho các tiến trình có thước động, và vị trí bắt đầu của các phân vùng cũng là động.

Thảo luận

Không còn hiện tượng phân mảnh nội vi nhưng xuất hiện hiện tượng phân mảnh ngoại vi (external fragmentation): Khi các tiến trình lần lượt vào và ra khỏi hệ thống, dần dần xuất hiện các khe hỡ giữa các tiến trình.

Vấn đề nãy sinh khi kích thước của tiến trình tăng trưởng trong quá trình xử lý mà không còn vùng nhớ trống gần kề để mở rộng vùng nhớ cho tiến trình. Khi đó có thể giải quyết theo hai hướng: Dời chỗ tiến trình cấp phát dư vùng nhớ cho tiến trình.

Cần giải quyết vấn đề cấp phát động: Làm thế nào để thỏa mãn một yêu cầu vùng nhớ kích thước N ? Cần phải chọn vùng nhớ nào trong danh sách vùng nhớ tự do để cấp phát ? Như vậy cần phải ghi nhớ hiện trạng bộ nhớ để có thể cấp phát đúng. Khi đó có hai phương pháp quản lý chủ yếu sau: Quản lý bằng một bảng các bit và quản lý bằng danh sách.

Một tiến trình cần được nạp vào bộ nhớ để xử lý. Trong các phương thức tổ chức trên đây, một tiến trình luôn được lưu trữ trong bộ nhớ suốt quá trình xử lý của nó. Tuy nhiên, trong trường hợp tiến trình bị khóa, hoặc tiến trình sử dụng hết thời gian CPU dành cho nó, nó có thể được chuyển tạm thời ra bộ nhớ phụ và sau này được nạp trở lại vào bộ nhớ chính để tiếp tục xử lý.

Các thuật toán thông dụng để chọn một phân đoạn tự do (khối nhớ trống) trong danh sách để cấp phát cho tiến trình:

First-fit:

Cấp phát phân đoạn tự do đầu tiên đủ lớn để nạp tiến trình.

Best-fit:

Cấp phát phân đoạn tự do nhỏ nhất nhưng đủ lớn để thỏa mãn nhu cầu để nạp tiến trình.

Worst-fit:

Cấp phát phân đoạn tự do lớn nhất.

Next-fit:

Tương tự như chiến lược First-fit, nhưng sẽ 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.

Last-fit:

Chọn khối nhớ trống cuối cùng có kích thước đủ lớn để nạp tiến trình.

Ví dụ 3.1

Cho 4 phân vùng bộ nhớ có thứ tự và kích thước lần lượt là 480 KB, 450 KB, 250 KB, 300 KB và 4 tiến trình có thứ tự và kích thước lần lượt là 255 KB, 435 KB, 215 KB, 452 KB.

Hãy cấp phát bộ nhớ cho các tiến trình trên lần lượt theo các thuật toán first-fit, best-fit, worst-fit, next-fit, last-fit; từ đó hãy cho biết thuật toán nào cấp phát bộ nhớ hiệu quả nhất trong trường hợp trên ?

Giải:

Algorithm	480 KB	450 KB	250 KB	300 KB
first-fit	255 KB 215 KB	435 KB		
best-fit	452 KB	435 KB	215 KB	255 KB
worst-fit	255 KB	435 KB		215 KB
next-fit	255 KB	435 KB	215 KB	
last-fit	452 KB	435 KB	215 KB	255 KB

Theo kết quả trên, các thuật toán best-fit và last-fit sử dụng bộ nhớ hiệu quả nhất.

3.2.4. Các hệ thống đa chương với kỹ thuật "Swapping"

Ý tưởng

Một tiến trình đang ở trong trạng thái chờ một thời gian tương đối dài sẽ được tạm thời chuyển ra bộ nhớ phụ để giải phóng vùng nhớ chính cho một tiến trình khác hoạt động. Khi tiến trình ban đầu kết thúc việc chờ, nó có thể được mang trở vào bộ nhớ chính để tiếp tục xử lý.

Thảo luận

Có thể nâng cao mức độ đa chương vì tăng số lượng tiến trình đồng hành. Khi tiến trình được mang trở lại vào bộ nhớ chính, nó sẽ được nạp vào bộ nhớ nào ? Nếu sự kết buộc địa chỉ được thực hiện vào thời điểm nạp, tiến trình cần được cấp phát lại đúng vùng nhớ nó đã chiếm giữ trước đó. Nếu kết buộc địa chỉ vào thời điểm xử lý, thì có thể nạp tiến trình vào một vùng nhớ bất kỳ.

Kỹ thuật Swapping cần sử dụng một bộ nhớ phụ, bộ nhớ này phải đủ lớn để lưu trữ các tiến trình bị khóa, và phải cho phép truy xuất trực tiếp đến các tiến trình này.

Trong các hệ Swapping, thời gian chuyển đổi giữa các tác vụ cần được quan tâm. Mỗi tiến trình cần được phân chia một khoảng thời gian sử dụng CPU đủ lớn để không thấy quá rõ sự chậm trễ do các thao tác swap gây ra. Nếu không, hệ thống sẽ dùng phân lớn thời gian để dời chuyển các tiến trình vào ra bộ nhớ chính, CPU như vậy sẽ không được sử dụng hiệu quả.

3.3. Các chiến lược cấp phát không liên tục

Các cách tổ chức bộ nhớ trên đây gọi chung là cách "cấp phát liên tục" vì chúng đều tiếp cận theo kiểu cấp phát một vùng nhớ liên tục cho tiến trình. Tiếp theo là các cách tiếp cận theo hướng cho phép không gian địa chỉ vật lý của tiến trình không liên tục, nghĩa là có thể cấp phát cho tiến trình những vùng nhớ tự do bất kỳ, không cần liên tục.

3.3.1. Kỹ thuật phân trang

Ý tưởng

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

Cơ chế MMU trong kỹ thuật phân trang:

Cơ chế phần cứng hỗ trợ thực hiện chuyển đổi địa chỉ trong cơ chế phân trang là bảng trang (page table). Mỗi phần tử trong bảng trang cho biết các địa chỉ bắt đầu của vị trí lưu trữ trang tương ứng trong bộ nhớ vật lý.

Chuyển đổi địa chỉ:

Mỗi địa chỉ phát sinh bởi CPU được chia thành 2 phần, bao gồm số hiệu trang (p): sử dụng như chỉ mục đến phần tử tương ứng trong bảng trang và địa chỉ tương đối trong trang (d): kết hợp với địa chỉ bắt đầu của trang để tạo ra địa chỉ vật lý mà trình quản lý bộ nhớ sử dụng.

Kích thước của trang do phần cứng quy định. Để thuận lợi trong việc phân tích địa chỉ ảo thành số hiệu trang và địa chỉ tương đối, kích thước của một trang thông

thường là một lũy thừa của 2 (biến đổi trong phạm vi 2^8 bytes..2^{13} bytes). Nếu kích thước của không gian địa chỉ là 2^m và kích thước trang là 2^n, thì m-n bit cao của địa chỉ ảo sẽ biểu diễn số hiệu trang, còn n bit thấp cho biết địa chỉ tương đối trong trang.

3.3.2. Kỹ thuật phân đoạn

Ý tưởng

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

Cơ chế MMU trong kỹ thuật phân đoạn:

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

Chuyển đổi địa chỉ:

Mỗi địa chỉ ảo là một bộ <s,d> bao gồm: số hiệu phân đoạn s: được sử dụng như chỉ mục đến bảng phân đoạn và địa chỉ tương đối d: có giá trị trong khoảng từ 0 đến giới hạn chiều dài của phân đoạn. Nếu địa chỉ tương đối hợp lệ, nó sẽ được cộng với giá trị chứa trong thanh ghi nền để phát sinh địa chỉ vật lý tương ứng.

Kỹ thuật phân đoạn thỏa mãn được nhu cầu thể hiện cấu trúc logic của chương trình nhưng nó dẫn đến tình huống phải cấp phát các khối nhớ có kích thước khác nhau cho các phân đoạn trong bộ nhớ vật lý. Điều này làm rắc rối vấn đề hơn rất nhiều so với việc cấp phát các trang có kích thước tĩnh. Một giải pháp dung hòa là kết hợp cả hai kỹ

thuật phân trang và phân đoạn (paged segmentation): chúng ta tiến hành phân trang các phân đoạn!

3.3.3. Kỹ thuật phân đoạn kết hợp với phân trang

Ý tưởng

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

Cơ chế MMU trong kỹ thuật phân trang:

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

Chuyển đổi địa chỉ:

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

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

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

Các mô hình tổ chức bộ nhớ trên đều có khuynh hướng cấp phát cho tiến trình toàn bộ các trang yêu cầu trước khi thật sự xử lý. Vì bộ nhớ vật lý có kích thước giới hạn, điều này dẫn đến hai điểm cần lưu ý sau: kích thước tiến trình bị giới hạn bởi kích thước của bộ nhớ vật lý và khó có thể bảo trì nhiều tiến trình cùng lúc trong bộ nhớ, và như vậy khó nâng cao mức độ đa chương của hệ thống.

Ví dụ 3.2

Chuyển đổi địa chỉ logic sang địa chỉ vật lý
Cho bảng phân trang như sau:

page	frame
0	3
1	2
2	6
3	4

Với kích thước mỗi trang là 1KB, hãy chuyển địa chỉ 1240 sang địa chỉ vật lý.

Giải:

1KB=1024B

1240 div 1024=1 (Lưu ý: nếu phép chia lấy nguyên này cho kết quả không có trong cột thứu nhất của bảng trên thì kết luận địa chỉ này không hợp lê).

Từ bảng trên, suy ra frame tương ứng là 2.

2 x 1024 + (1240 % 1024)=2048 + 216=2264

Vậy địa chỉ vật lý của địa chỉ logic 1240 là 2264.

Tương tự: 3580, 5540

Ví dụ 3.3

Chuyển đổi địa chỉ vật lý sang địa chỉ logic

Cho địa chỉ vật lý 6578 sẽ được chuyển thành địa chỉ logic là bao nhiêu? Biết rằng kích thước mỗi frame là 1K bytes và bảng ánh xạ địa chỉ logic như hình sau:

0	6
1	4
2	5
3	7
4	1
5	9

Địa chỉ vật lý: 6578

mỗi frame là 1Kbytes = 1024 B

có 6578 div 1024=6

suy ra: page: 0

địa chỉ logic: 0 x 1024 + (6578- 6 x 1024)=434

Vậy địa chỉ logic tương ứng với địa chỉ vật lý 6578 là 434.

Ví dụ 3.4

Địa chỉ vật lý & địa chỉ luận lý trong phân trang

Xét một không gian địa chỉ luận lý có 8 trang, mỗi trang có kích thước 1KB. Ánh xạ vào bộ nhớ vật lý có 32 khung trang.

a. Địa chỉ logic gồm bao nhiêu bit ?

b. Địa chỉ physic gồm bao nhiêu bit ?

c. Bảng trang có bao nhiêu mục? Mỗi mục trong bảng trang cần bao nhiêu bit?

Giải

Số bit của địa chỉ logic: $8 \times 2^{10} = 2^{13}$, suy ra 13 bít địa chỉ luận lý.

Số bit của địa chỉ vật lý: $32 \times 2^{10} = 2^{15}$, suy ra 15 bít địa chỉ vật lý.

Bảng trang có 8 mục, bộ nhớ vật lý 32 byte. Lưu trữ từ 0..31, suy ra mỗi mục cần 5 bit.

3.4. Bộ nhớ ảo

3.4.1. Giới thiệu

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

Một cách để thực hiện ý tưởng của giải pháp trên là sử dụng kỹ thuật overlay. Kỹ thuật overlay không đòi hỏi bất kỳ sự trợ giúp đặc biệt nào của hệ điều hành, nhưng trái lại, lập trình viên phải biết cách lập trình theo cấu trúc overlay, và điều này đòi hỏi khá nhiều công sức. Để giải phóng lập trình viên khỏi các suy tư về giới hạn của bộ nhớ, mà cũng không tăng thêm khó khăn cho công việc lập trình của họ, người ta nghĩ đến các kỹ thuật tự động, cho phép xử lý một chương trình có kích thước lớn chỉ với một vùng

nhớ có kích thước nhỏ hơn. Giải pháp được tìm thấy với khái niệm bộ nhớ ảo (virtual memory).

3.4.2. Định nghĩa

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

3.4.3. Cài đặt bộ nhớ ảo

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

Phân trang theo yêu cầu

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

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

- valid: trang tương ứng là hợp lệ và đang ở trong bộ nhớ chính.
- Invalid: hoặc trang bất hợp lệ hoặc trang hợp lệ nhưng đang được lưu trên bộ nhớ phụ.

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

Cơ chế phần cứng

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

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

Lỗi trang

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

Bước 1: Kiểm tra truy xuất đến bộ nhớ có hợp lệ ?

Bước 2: Nếu truy xuất bất hợp lệ: kết thúc tiến trình; ngược lại: đến bước 3.

Bước 3: Tìm vị trí chứa trang muốn truy xuất trên đĩa.

Bước 4: Tìm một khung trang trống trong bộ nhớ chính:

a.Nếu tìm thấy: đến bước 5

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

Bước 5: Chuyển trang muốn truy xuất từ bộ nhớ phụ vào bộ nhớ chính: nạp trang cần truy xuất vào khung trang trống đã chọn (hay vừa mới làm trống); cập nhật nội dung bảng trang, bảng khung trang tương ứng.

Bước 6: Tái kích hoạt tiến trình người sử dụng.

3.4.4. Các thuật toán thay thế trang

Vấn đề thay thế trang

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

Sự thay thế trang là cần thiết cho kỹ thuật phân trang theo yêu cầu. Nhờ cơ chế này, hệ thống có thể hoàn toàn tách rời bộ nhớ ảo và bộ nhớ vật lý, cung cấp cho lập trình viên một bộ nhớ ảo rất lớn trên một bộ nhớ vật lý có thể bé hơn rất nhiều lần.
Để duy trì ở một mức độ chấp nhận được sự chậm trễ trong hoạt động của hệ thống do phân trang, cần phải duy trì tỉ lệ phát sinh lỗi trang thấp. Hơn nữa, để cài đặt kỹ thuật phân trang theo yêu cầu, cần phải giải quyết hai vấn đề chính yếu: xây dựng một thuật toán cấp phát khung trang, và thuật toán thay thế trang.

Vấn đề chính khi thay thế trang là chọn lựa một trang "nạn nhân" để chuyển ra bộ nhớ phụ. Có nhiều thuật toán thay thế trang khác nhau, nhưng tất cả cùng chung một mục tiêu: chọn trang "nạn nhân" là trang mà sau khi thay thế sẽ gây ra ít lỗi trang nhất. Có thể đánh giá hiệu quả của một thuật toán bằng cách xử lý trên một chuỗi các địa chỉ cần truy xuất và tính toán số lượng lỗi trang phát sinh. Để xác định số các lỗi trang xảy ra khi sử dụng một thuật toán thay thế trang nào đó trên một chuỗi truy xuất cụ thể, cần phải biết số lượng khung trang sử dụng trong hệ thống.

Để minh họa các thuật toán thay thế trang sẽ trình bày, chuỗi truy xuất được sử dụng là: 7, 5, 0, 2, 0, 2, 3, 2, 3, 1, 0, 2, 3, 6, 2, 7, 6, 7, 2, 4.

Thuật toán FIFO

Ý tưởng

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

Ví dụ 3.5

Cho chuỗi truy xuất (reference string) bộ nhớ sau:

7, 5, 0, 2, 0, 2, 3, 2, 3, 1, 0, 2, 3, 6, 2, 7, 6, 7, 2, 4. Giả sử sử dụng 3 khung trang và ban đầu các khung trang đều trống. Hãy tìm số lỗi trang (page faults) xảy ra khi sử dụng thuật toán thay thế trang FIFO (first in first out page replacement algorithm).

Giải:

Thuật toán thay thế trang FIFO

Page frames	**7**	**5**	**0**	**2**	**0**	**2**	**3**	**2**	**3**	**1**	**0**	**2**	**3**	**6**	**2**	**7**	**6**	**7**	**2**	**4**
I	7	7	7	2	2	2	2	2	2	2	0	0	0	6	6	6	6	6	6	4
II		5	5	5	5	5	3	3	3	3	3	2	2	2	2	7	7	7	7	7
III			0	0	0	0	0	0	0	1	1	1	3	3	3	3	3	3	2	2
page faults	*	*	*	*			*			*	*	*	*	*		*			*	*

Tổng số lỗi trang:13

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

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

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

(bạn đọc hãy kiểm chứng nghịch lý Belady khi sử dụng 4,5,6,7 khung trang với chuỗi truy xuất trên).

Thuật toán tối ưu

Ý tưởng

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

Ví dụ 3.6

Cho chuỗi truy xuất (reference string) bộ nhớ sau:

7, 5, 0, 2, 0, 2, 3, 2, 3, 1, 0, 2, 3, 6, 2, 7, 6, 7, 2, 4. Giả sử sử dụng 3 khung trang và ban đầu các khung trang đều trống. Hãy tìm số lỗi trang (page faults) xảy ra khi sử dụng thuật toán thay thế trang tối ưu (optimal page replacement algorithm).

Giải:

Page frames	**7**	**5**	**0**	**2**	**0**	**2**	**3**	**2**	**3**	**1**	**0**	**2**	**3**	**6**	**2**	**7**	**6**	**7**	**2**	**4**
I	7	7	7	7	7	7	3	3	3	1	1	**1**	3	6	6	6	6	6	6	4
II		5	5	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
III			0	0	0	0	0	0	0	0	0	0	0	0	0	7	7	7	7	7
page faults	*	*	*	*			*			*			*	*		*				*

Tổng số lỗi trang: 9

Thuật toán này bảo đảm số lượng lỗi trang phát sinh là thấp nhất, nó cũng không gánh chịu nghịch lý Belady (trong thực tế, việc thu thập thông tin về các trang sẽ được dùng trong tương lai cũng không tạo ra độ trễ lớn cũng như tốn kém nhiều tài nguyên. Do đó, các thuật toán thực tế thường dựa vào các chiến lược ước lượng để quyết định trang nào sẽ được thay thế).

Ghi chú rằng khi cần thay thế một trang (ở chuỗi truy xuất) mà khi đó trong các khung trang có nhiều hơn một trang không xuất hiện trong phần tương lai của chuỗi truy xuất, thì thuật toán sẽ chọn trang để thay thế là trang ở khung trang có chỉ số khung trang nhỏ hơn (chẳng hạn như trường hợp khi chuyển trang 3 ở vị trí 13 của chuỗi truy xuất vào khung trang I).

Thuật toán Least recently used – LRU

Ý tưởng

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

Ví dụ 3.7

Cho chuỗi truy xuất (reference string) bộ nhớ sau:

7, 5, 0, 2, 0, 2, 3, 2, 3, 1, 0, 2, 3, 6, 2, 7, 6, 7, 2, 4. Giả sử sử dụng 3 khung trang và ban đầu các khung trang đều trống. Hãy tìm số lỗi trang (page faults) xảy ra khi sử dụng thuật toán thay thế trang LRU (least recently used page replacement algorithm).

Giải:

Page frames	**7**	**5**	**0**	**2**	**0**	**2**	**3**	**2**	**3**	**1**	**0**	**2**	**3**	**6**	**2**	**7**	**6**	**7**	**2**	**4**
I	7	7	7	2	2	2	2	2	2	2	0	0	0	6	6	6	6	6	6	4
II		5	5	5	5	5	3	3	3	3	3	2	2	2	2	2	2	2	2	2
III			0	0	0	0	0	0	0	1	1	1	3	3	3	7	7	7	7	7
page faults	*	*	*	*			*			*	*	*	*	*		*				*

Tổng số lỗi trang:12

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

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

3.4.5. Một số vấn đề khác

Cấp phát khung trang

Vấn đề đặt ra là làm thế nào để cấp phát một vùng nhớ tự do có kích thước cố định cho các tiến trình khác nhau ?

Trong trường hợp đơn giản nhất của bộ nhớ ảo là hệ đơn nhiệm, có thể cấp phát cho tiến trình duy nhất của người dùng tất cả các khung trang trống.

Vấn đề nãy sinh khi kết hợp kỹ thuật phân trang theo yêu cầu đối với sự đa chương: cần phải duy trì nhiều tiến trình trong bộ nhớ cùng lúc, vậy mỗi tiến trình sẽ được cấp bao nhiêu khung trang ? Số khung trang tối thiểu được quy định bởi kiến trúc

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

Tần suất xảy ra lỗi trang

Tần suất lỗi trang rất cao khiến tình trạng trì trệ hệ thống có thể xảy ra. Khi tần suất lỗi trang quá cao, tiến trình cần thêm một số khung trang. Khi tần suất lỗi trang quá thấp, tiến trình có thể sở hữu nhiều khung trang hơn mức cần thiết.

Có thể thiết lập một giá trị chặn trên và chặn dưới cho tần suất xảy ra lỗi trang, và trực tiếp ước lượng và kiểm soát tần suất lỗi trang để ngăn chặn tình trạng trì trệ xảy ra: Nếu tần suất lỗi trang vượt quá chặn trên, cấp cho tiến trình thêm một khung trang. Nếu tần suất lỗi trang thấp hơn chặn dưới, thu hồi bớt một khung trang từ tiến trình.

Kích thước trang

Kích thước trang thông thường được xác định bởi phần cứng. Không có một lựa chọn lý tưởng cho kích thước trang: Kích thước trang càng lớn thì kích thước bảng trang càng giảm, kích thước trang càng bé thì cho phép tổ chức nhóm trang cục bộ tốt hơn. Thời gian nhập xuất nhỏ khi kích thước trang lớn. Kích thước trang bé thì có thể giảm số lượng thao tác nhập xuất cần thiết, vì có thể xác định các nhóm trang cục bộ chính xác hơn, kích thước trang lớn sẽ giảm tần suất lỗi trang. Đa số các hệ thống chọn kích thước trang là 4KB.

Bài tập chương 3

Bài tập 3.1.

Giả sử bộ nhớ chính được phân thành các phân vùng có kích thước là 600K, 500K, 200K, 300K (theo thứ tự), cho biết các tiến trình có kích thước 212K, 417K, 112K và 426K (theo thứ tự) sẽ được cấp phát bộ nhớ như thế nào, nếu sử dụng :

a. Thuật toán First fit

b. Thuật toán Best fit

c. Thuật toán Worst fit

Hãy cho biết thuật toán nào cho phép sử dụng bộ nhớ hiệu qủa nhất trong trường hợp trên ?

Bài tập 3.2.

Xet chuỗi truy xuất bộ nhớ sau:

1, 2 , 3 , 4 , 2 , 1 , 5 , 6 , 2 , 1 , 2 , 3 , 7 , 6 , 3 , 2 , 1 , 2 , 3 , 6

Hỏi có bao nhiêu lỗi trang xảy ra khi sử dụng các thuật toán thay thế sau đây, giả sử có 1, 2,3, 4, 5, 6, 7 khung trang ?

a. LRU

b. FIFO

c. Chiến lược tối ưu

Bài tập 3.3.

Xét bảng phân đoạn sau đây:

Segment	Base	length
0	219	600
1	2300	14
2	90	100
3	1327	580
4	1952	96

Hãy cho biết địa chỉ vật lý tương ứng với các địa chỉ logic sau đây:

a. 0, 430

b. 1, 10

c. 2, 500

d. 3, 400

e. 4, 112

Bài tập 3.4.

Hệ thống phân trang, kích thước trang là 1024B. Bảng phân trang hiện thời như sau (page, frame):

3 0

2 4

1 x

0 1

Hãy tính địa chỉ vật lý của các địa chỉ logic sau:

a. 1020

b. 2060

Bài tập 3.5.

Cho bảng phân trang page 0,1,2,3; frame tương ứng là 3,2,6,4. Với kích thước mỗi trang là 1KB thì địa chỉ logic 2021 ứng với địa chỉ vật lý nào ?

Bài tập 3.6.

Cho địa chỉ vật lý là 4100 sẽ được chuyển thành địa chỉ ảo bao nhiêu? Biết rằng kích thước mỗi frame là 1K bytes, và bảng ánh xạ địa chỉ ảo như hình.

Page	frame
0	6
1	4
2	5
3	7
4	1
5	9

Bài tập 3.7.

Cho địa chỉ vật lý là 4604 sẽ được chuyển thành địa chỉ logic bao nhiêu? Biết rằng kích thước mỗi frame là 1K bytes và bảng ánh xạ địa chỉ logic như hình sau:

Page	frame
0	6
1	4
2	5
3	7
4	1
5	9

Bài tập 3.8.

Giả sử trong quá trình quản lý bộ nhớ ảo dạng phân đoạn, hệ điều hành duy trì segment table

Segment	base	limit
0	300	700
1	1200	500
2	2000	600

Hãy tính địa chỉ vật lý cho mỗi địa chỉ logic sau: (1,200); (1,0); (0,700); (2,0); (2,600)

Bài tập 3.9.

Một hệ thống có bộ nhớ chính kích thước 512 MB. Hệ thống sử dụng địa chỉ logic 64 bit. Kích thước của 1 page được sử dụng là 8KB. Hãy xác định các thông số sau:

a. Cho biết 1 frame có kích thước là bao nhiêu? Tại sao?
b. Tổng số pages mà hệ thống có thể cấp phát?
c. Tổng số frames trong bộ nhớ chính?
d. Cho địa chỉ logic: 202212. Hãy đổi địa chỉ này sang dạng <p,d>

Bài tập 3.10.

Trong bộ nhớ ảo, tiến trình P đang chạy được chia làm 8 pages (0, 1, ….., 7), hệ thống cấp cho tiến trình P 3 frame để thực hiện. Ta có dãy truy cập đến page như sau:

7 0 1 2 3 4 5 3 5 1 0 6 2 1 4 7 2 1 0 3 2

a. Hãy lập bảng cấp phát và thay thế khung trang đối với dãy truy cập page như trên, sử dụng thuật toán thay thế trang FIFO.
b. Hãy đếm số page faults trong tình huống trên.

Bài tập 3.11.

Cài đặt các thuật toán đã được trình bày trong chương này.

Bài tập 3.12.

Cho 4 phân vùng bộ nhớ có thứ tự và kích thước lần lượt là 585 KB, 450 KB, 150 KB, 250 KB và 4 tiến trình có thứ tự và kích thước lần lượt là 212 KB, 417 KB, 112 KB, 426 KB. Hãy cấp phát bộ nhớ cho các tiến trình trên lần lượt theo các thuật toán first-fit, best-fit, và worst-fit; từ đó hãy cho biết thuật toán nào cấp phát bộ nhớ hiệu quả nhất trong trường hợp trên ?

Bài tập 3.13.

Cho chuỗi truy xuất (reference string) bộ nhớ sau:

7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 5, 6. Giả sử sử dụng 3 khung trang và ban đầu các khung trang đều trống. Hãy tìm số lỗi trang (page faults) xảy ra khi sử dụng các thuật toán thay thế trang sau:

a. Thuật toán thay thế trang FIFO (first in first out page replacement algorithm).
b. Thuật toán thay thế trang LRU (least recently used page replacement algorithm).

CHƯƠNG 4. QUẢN LÝ LƯU TRỮ

Để lưu trữ dữ liệu trong máy tính, người ta dùng các thiết bị lưu trữ như ổ đĩa HDD, SSD hay các thiết bị lưu trữ mạng NAS, Cloud. Để có thể điều khiển các loại thiết bị lưu trữ, hệ điều hành sử dụng hệ thống nhập xuất để đọc - ghi dữ liệu. Đồng thời, hệ điều hành sử dụng hệ thống tập tin để quản lý tập tin, thư mục. Điều này giúp người dùng có thể quản lý dữ liệu của mình một cách dễ dàng hơn.

4.1. Cấu trúc Mass – Storage

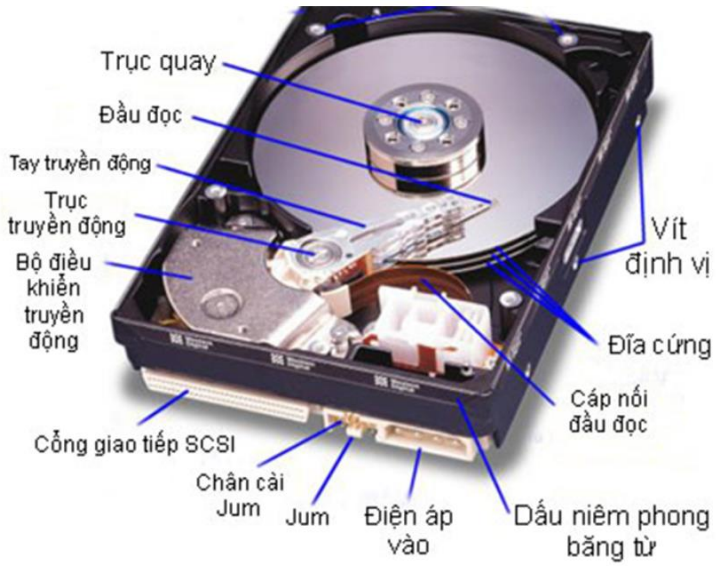
Mass – Storage là một loại hệ thống dùng để lưu trữ lượng lớn dữ liệu. Thông thường, người ta sử dụng các loại ổ đĩa ngoài hoặc các loại lưu trữ mạng để lưu trữ. Ngày nay, các loại thiết bị dùng để lưu trữ có thể kể đến như:

- Ổ đĩa từ (magnetic disks), còn gọi là ổ đĩa HDD.
- Ổ đĩa thể rắn (SSD).
- Lưu trữ trên mạng như: NAS, SAN,…

4.1.1. Ổ đĩa HDD

Ổ đĩa HDD sử dụng các miếng đĩa tròn làm bằng nhôm, hoặc thủy tinh được phủ các vật liệu từ tính lên trên. Các phiến đĩa này được sắp chồng lên nhau với tâm của đĩa được gắn một loại động cơ. Khi hoạt động, các phiến đĩa sẽ được quay bởi động cơ để đọc và ghi dữ liệu. Ổ đĩa HDD là loại ổ đĩa không mất dữ liệu khi mất điện.

Ổ đĩa HDD có ưu điểm: dung lượng lưu trữ lớn (có thể đến hàng chục TB), chi phí tương đối thấp hơn so với các loại ổ đĩa SSD. Tuy nhiên, ổ đĩa HDD cũng có nhược điểm là: tốc độ truy xuất còn thấp do phụ thuộc vào tốc độ quay của đĩa, gây ồn và tỏa nhiệt cao khi hoạt động, khả năng mất dữ liêu cao do bị sốc trong quá trình sử dụng.



Hình 4.1. Hình ảnh bên trong của một ổ cứng

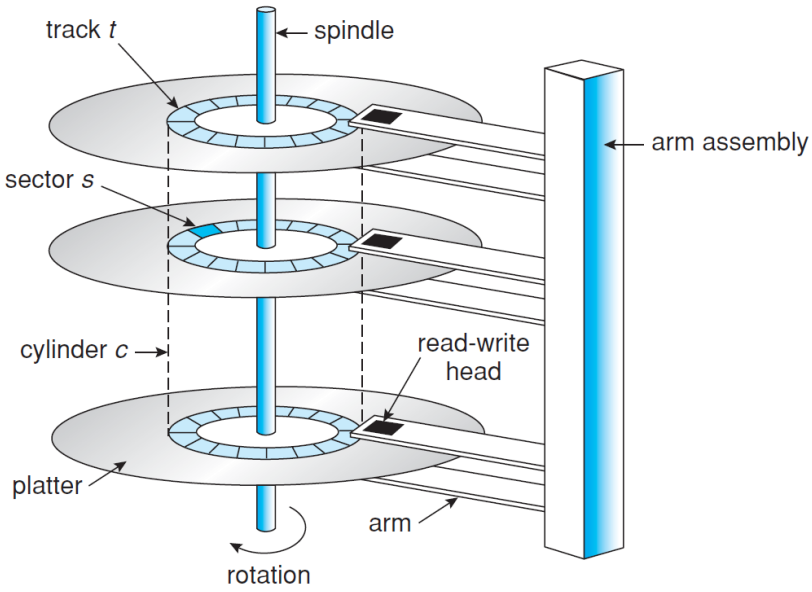
Trong hình trên, đĩa từ là một đĩa kim loại hình tròn được gắn bên trong ổ cứng. Có nhiều đĩa từ được xếp chồng đồng tâm lên nhau, gắn vào một động cơ trục chính để tạo nhiều bề mặt lưu trữ dữ liệu trong một không gian nhỏ hơn.

Để duy trì việc lưu trữ và truy xuất dữ liệu có tổ chức, các đĩa từ được sắp xếp thành các cấu trúc cụ thể. Các cấu trúc cụ thể này bao gồm các track (rãnh), sector và cluster.

- *Track*: Mỗi đĩa từ được chia thành hàng ngàn vòng tròn đồng tâm được đóng gói chặt chẽ, được gọi là track. Tất cả các thông tin được lưu trữ trên ổ cứng đều được ghi trên các track.
- *Sector*: Mỗi track được chia thành các đơn vị nhỏ hơn gọi là sector. Sector là đơn vị lưu trữ cơ bản dữ liệu trên ổ đĩa cứng.
- *Cluster*: Các sector thường được nhóm lại với nhau để tạo thành các cluster.

Đầu đọc ghi (read – write head) là thành phần dùng để chuyển thông tin ở dạng xung từ sang dạng bit khi đọc dữ liệu và thực hiện ngược lại khi ghi dữ liệu. Trục chuyển động giúp cho đầu đọc ghi có thể di chuyển ra vào giữa mặt đĩa.

Toàn bộ đĩa cứng được gắn trong vỏ kín được thiết kế để bảo vệ nó khỏi không khí bên ngoài. Phía dưới ổ đĩa gọi là phần đế. Các cơ chế truyền động được đặt trong phần đế và phần nắp đậy, được đặt trên đầu để đảm bảo độ kín cho đầu đọc và đĩa từ.



Hình 4.2. Nguyên lý đọc ghi dữ liệu của ổ HDD [3]

Khi ổ HDD muốn đọc hay ghi dữ liệu, ổ HDD nhờ vào cơ chế quay của trục quay và trục chuyển động của đầu đọc ghi để đưa đầu đọc ghi đến đúng vị trí cần đọc hay ghi dữ liệu. Trong bề mặt từ, các vị trí được biểu diễn bằng bộ 3 (cylinder, track, sector). Cylinder là tập hợp các tracks có cùng bán kính ở các mặt đĩa khác nhau. Khi đã đến đúng vị trí, đầu đọc ghi sẽ tác động xung từ để chuyển đổi sang dạng dữ liệu bit cho các thao tác đọc hay ghi.

Khi ổ HDD đọc ghi dữ liệu, tốc độ của ổ đĩa phụ thuộc vào số vòng quay trên 1 phút (RPM – revolutons per minute). Thông thường, tốc độ vòng quay của ổ HDD là 5400 rpm, 7200 rpm, 10.000 rpm và 15.000 rpm. Thời gian đọc ghi dữ liệu sẽ chịu ảnh hưởng bởi 3 thành phần:

- Seek time: thời gian di chuyển đầu đọc để định vị đúng track/cylinder, phụ thuộc tốc độ/cách di chuyển của đầu đọc.
- Rotational delay (latency): thời gian đầu đọc chờ đến đúng sector cần đọc, phụ thuộc tốc độ quay của đĩa.
- Transfer time: thời gian chuyển dữ liệu từ đĩa vào bộ nhớ hoặc ngược lại, phụ thuộc băng thông kênh truyền giữa đĩa và bộ nhớ.

Disk I/O time = seek time + rotational delay + transfer time.

Ổ HDD được gắn vào máy tính thông qua một bộ giao tiếp gọi là I/O bus. Các I/O bus thông dụng hiện nay là ATA, SATA, eSATA, USB và fibre channel (FC). Các

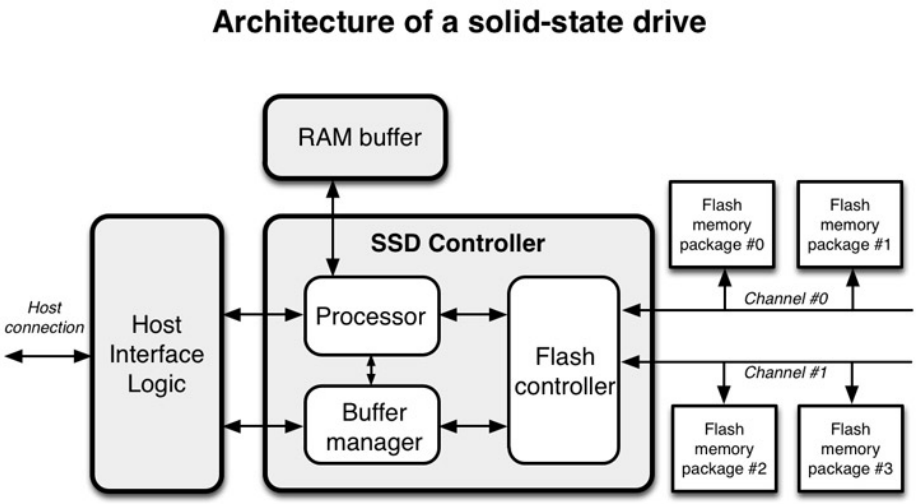
dữ liệu được truyền trên các bus dữ liệu thông qua một thiết bị điều khiển điện tử gọi là controller. Host controller được thiết lập trên máy tính, trong khi disk controller được đặt tại ổ đĩa. Khi cần đọc ghi dữ liệu, host controller sẽ chuyển lệnh đến disk controller. Từ đây, disk controller sẽ xử lý trên ổ cứng, sau đó chuyển dữ liệu đến host controller.

4.1.2. Ổ đĩa SSD

Ổ đĩa SSD (solid state drive), còn gọi là ổ cứng thể rắn. Ổ cứng SSD cũng dùng để lưu trữ dữ liệu giống như ổ HDD, không bị mất dữ liệu khi mất điện. Tuy nhiên, khác với ổ HDD, ổ SSD không dùng các đĩa từ để lưu trữ dữ liệu mà sử dụng các chip nhớ flash để lưu trữ dữ liệu. Điều này giúp ổ SSD có thể đọc và ghi dữ liệu nhanh chóng hơn. Tuy nhiên, giá thành của ổ SSD cao hơn so với ổ HDD, và dung lượng lưu trữ của ổ SSD vẫn còn thấp hơn so với ổ HDD.

Hình 4.3. Ổ cứng SSD

SSD được xây dựng lên từ nhiều chip nhớ flash NOR và bộ nhớ NAND flash (bộ nhớ điện tĩnh). SSD được làm hoàn toàn bằng linh kiện điện tử và không có bộ phận chuyển động vật lý như trong ổ đĩa HDD. Những con chip flash sẽ được lắp cố định trên một bo mạch chủ khoảng từ 10-60 NAND của hệ thống. Các chip nhớ được lắp vào trong một chiếc hộp có hình dạng và kích thước giống như ổ cứng nhưng nhỏ hơn. Ngoài ra, một vài ổ đĩa SSD được thiết kế để gắn trực tiếp vào bên trong Bus hệ thống thông qua cổng PCI.



Hình 4.4. Cấu trúc bên trong của ổ SSD

Trong cấu trúc của ổ cứng SSD bao gồm một bộ điều khiển SSD (SSD controller) dùng để điều khiển hoạt động đọc ghi dữ liệu. Đồng thời thực hiện xử lý các lỗi khi lưu dữ liệu hay đọc dữ liệu.

Ổ SSD cũng sử dụng RAM buffer là bộ nhớ RAM sử dụng chuẩn giao tiếp DDR3 làm thành phần lưu trữ tạm, hỗ trợ quá trình đọc ghi dữ liệu.

Các Flash memory package là các NAND flash. Đây là nơi lưu trữ dữ liệu của ổ SSD. Các flash này bao gồm nhiều Cell. Mỗi Cell có thể lưu trữ từ một đến bốn bit dữ liệu, tùy vào chip set SLC, MLC, TLC, hay QLC.

Hình 4.5. Bên trong ổ SSD

Ổ cứng SSD sử dụng một tấm các ổ điện để đọc ghi dữ liệu. Những tấm này được phân chia thành các phần được gọi là "trang" và là nơi lưu trữ dữ liệu. Các trang này được nhóm lại với nhau tạo thành các "khối". Ổ SSD chỉ có thể ghi vào một trang trống

trong một khối. Trong ổ cứng HDD, dữ liệu có thể được ghi ở bất cứ vị trí nào trên đĩa vào bất cứ lúc nào. Điều này có nghĩa là dữ liệu sẽ bị ghi đè dễ dàng hơn. Tuy nhiên, các ổ SSD không thể ghi đè trực tiếp dữ liệu lên từng trang riêng lẻ như vậy, chúng chỉ ghi dữ liệu lên trang trống trong một khối. Khi các trang trong một khối được đánh dấu là không sử dụng, ổ SSD chép dữ liệu của toàn bộ khối vào bộ nhớ RAM buffer, sau đó xóa toàn bộ khối đó và chép lại dữ liệu từ bộ nhớ trở lại khối trong khi để trống các trang không sử dụng. Lưu ý, việc xóa khối không có nghĩa là dữ liệu hoàn toàn biến mất.

Ổ SSD sẽ không ghi đè dữ liệu lên các trang đã ghi, do đó, khi muốn ghi dữ liệu mới, ổ SSD thực hiện qua những bước như sau:

1) Tìm một khối có đủ các trang được đánh dấu là "không sử dụng".
2) Ghi lại các trang trong khối đó mà vẫn còn sử dụng vào RAM buffer.
3) Reset tất cả các trang trong khối đó.
4) Chép lại các trang đã lưu trước đó vào khối đã được reset.
5) Điền các trang còn lại bằng dữ liệu mới

Nhìn chung, ngày nay ổ cứng SSD cũng được sử dụng phổ biến trên thị trường do ưu điểm lưu trữ dữ liệu nhanh chóng, khả năng chịu lỗi cao. Tuy nhiên, chi phí của ổ SSD vẫn cao hơn so với ổ HDD.

4.1.3. Lưu trữ trên mạng

NAS (Network Attached Storage) hay còn gọi là ổ cứng mạng. Đây là cách thức lưu trữ sử dụng một thiết bị để lưu trữ toàn bộ dữ liệu của chúng ta trên thiết bị này. Sau đó, khi người dùng cần truy cập, sẽ sử dụng các kết nối mạng để kết nối đến thiết bị và lấy dữ liệu cần thiết.

Hình 4.6. NAS và cách kết nối

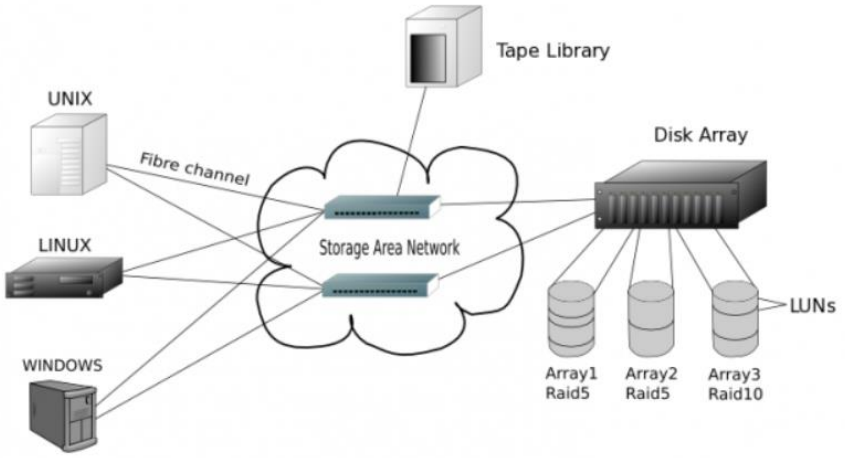
NAS có kiến trúc tương tự như một máy tính với bộ vi xử lý CPU và thường có sẵn hệ điều hành, có thể là một phiên bản rút gọn dựa trên Linux và có khả năng kết nối qua mạng có dây hoặc không dây Wi-Fi.

NAS có những lợi ích như dữ liệu được quản lý tập trung, có thể truy cập mọi lúc, mọi nơi. Đối với doanh nghiệp, NAS giúp tiết kiệm chi phí, thay vì phải sử dụng những hệ thống server lưu trữ đắt đỏ, đòi hỏi phải quản lý, bảo trì phức tạp với chi phí cao. NAS nâng cấp, quản trị dễ dàng, các bước sử dụng đơn giản và không yêu cầu những kỹ năng cao cấp.

Hiển nhiên, dung lượng lưu trữ của NAS phải dựa vào các ổ đĩa lưu trữ được gắn vào thiết bị. Do đó, nếu nhu cầu lưu trữ càng lớn thì phải đầu tư thiết bị NAS có chi phí cao để đáp ứng đủ các tính năng. Đồng thời, phải đầu tư thêm các ổ đĩa để gắn vào trong NAS dùng để lưu trữ.

SAN (Storage Area Network) hay còn gọi là mạng lưu trữ. Đây là một cách thức lưu trữ sử dụng một hệ thống mạng chuyên dụng, kết nối giữa nhiều server, nhiều thiết bị lưu trữ nhằm mục đích truyền tải dữ liệu giữa tất cả các thiết bị trong mạng với nhau. SAN hoàn toàn tách biệt với các mạng LAN và WAN. Mạng SAN có thể nối kết tất cả các tài nguyên liên quan đến lưu trữ trong mạng lại với nhau.

Là một một mạng có tốc độ cao dành riêng cho việc lưu và quản trị dữ liệu, SAN giúp việc sử dụng tài nguyên lưu trữ hiệu quả hơn, dễ dàng hơn trong công việc quản trị, quản lý tập trung các thao tác tăng độ an toàn, sao lưu, khôi phục khi có sự cố.



Hình 4.7. Hệ thống SAN

Hệ thống SAN thường có 3 thành phần chính:

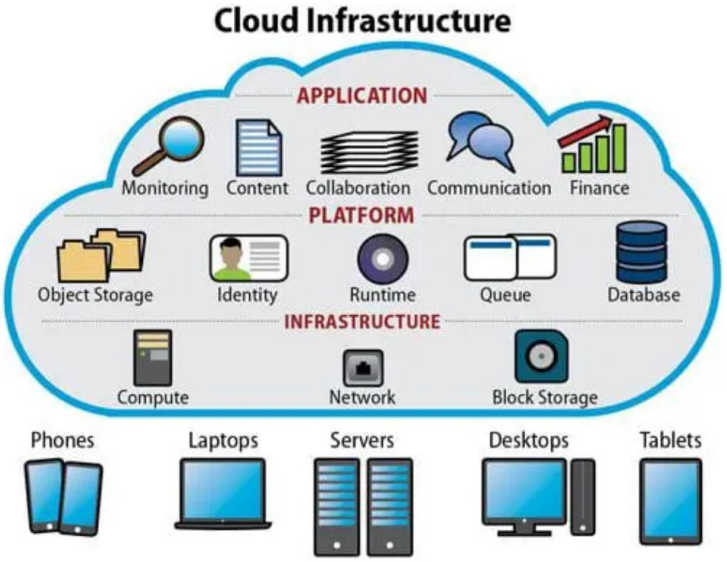
- Thiết bị lưu trữ: là nơi đặt các thiết bị lưu trữ như ổ đĩa có dung lượng lớn, khả năng truy xuất nhanh, có hỗ trợ các chức năng RAID, … Đây là nơi chứa dữ liệu chung cho toàn bộ hệ thống. Hiển nhiên, các thiết bị lưu trữ này phải được đặt trong các thiết bị chuyên dụng để thực hiện kết nối khi sử dụng.
- Thiết bị chuyển mạch SAN: đó là các SAN Switch thực hiện việc kết nối các máy chủ đến thiết bị lưu trữ.
- Các máy chủ (server) hoặc máy trạm (client) có nhu cầu lưu trữ dữ liệu, được kết nối đến SAN Switch bằng các loại cable hoặc wifi.

Nhìn chung, hệ thống SAN thường được áp dụng cho các doanh nghiệp có nhu cầu lưu trữ dữ liệu cao. Vì có nhiều ưu điểm như tốc độ truy cập, khả năng bảo mật, khả năng truy cập ở mọi nơi, khả năng mở rộng dung lượng lưu trữ dễ dàng,… Tuy nhiên, SAN cũng có những khó khăn đó là chi phí đầu tư cao, đòi hỏi khả năng cấu hình và quản lý phức tạp.

Cloud hay còn gọi là điện toán đám mây. Đây là một hệ thống phân phối các tài nguyên công nghệ thông tin theo nhu cầu qua internet với chính sách thanh toán theo mức sử dụng. Thay vì mua, sở hữu và bảo trì các trung tâm dữ liệu và máy chủ vật lý, người dùng có thể tiếp cận các dịch vụ công nghệ, như năng lượng điện toán, lưu trữ và cơ sở dữ liệu thông qua các công ty chuyên cung cấp hệ thống cloud.

Ngày nay, hệ thống cloud được sử dụng rộng rãi, cloud không chỉ đơn thuần là nơi lưu trữ dữ liệu, đó là nơi cung cấp rất nhiều tiện ích. Thông thường, có 3 loại cloud

chính: Cơ sở hạ tầng dưới dạng dịch vụ (IaaS), nền tảng dưới dạng dịch vụ (PaaS) và phần mềm dưới dạng dịch vụ (SaaS).



Hình 4.8. Cấu trúc Cloud

4.1.4. Disk Formatting

Ổ đĩa cứng mới sản xuất chỉ đơn giản là các platter chất liệu ghi từ. Do đó, để sử dụng được ổ đĩa phải thông qua 2 giai đoạn format.

Giai đoạn 1: Low level formatting, đây là giai đoạn do nhà sản xuất thiết bị thực hiện. Giai đoạn này xác định các chỉ số như free sector, logical block, và những chỉ số liên quan đến việc đọc ghi bên dưới ổ đĩa.

Giai đoạn 2: Logical formatting. Đây là giai đoạn do người dùng thực hiện. Người dùng sẽ tùy theo nhu cầu của mình mà phân chia ổ đĩa ra thành các partition, và chọn hệ thống tập tin (file system) cho từng partition.

Sau khi đã thực hiện xong giai đoạn 2, lúc này người dùng có thể lưu trữ dữ liệu lên ổ đĩa một cách bình thường.

Các hệ điều hành xem partition như một ổ đĩa riêng biệt; có nghĩa là muốn lưu trữ dữ liệu phải định nghĩa partition, khởi tạo hệ thống tập tin lên partition. Tuy nhiên, một số hệ điều hành cho phép chương trình đặc biệt sử dụng disk partition như một chuỗi các logical blocks, bỏ qua hệ thống tập tin. Điều này được gọi là RAW disk. Thông thường được áp dụng cho các server lưu trữ cơ sở dữ liệu (database).

4.2. Lập lịch đĩa

Hệ điều hành có chức năng phối hợp các thiết bị phần cứng để làm việc hiệu quả. Đối với các loại ổ đĩa HDD sử dụng đĩa từ để lưu trữ dữ liệu, việc thực hiện lập lịch đĩa (disk sheduling) là việc xây dựng các thuật toán để điều khiển đầu từ đọc ghi sao cho thời gian truy nhập đĩa là tối ưu nhất.

Thời gian truy cập ổ đĩa sẽ phụ thuộc vào 3 yếu tố:

- Thời gian di chuyển đầu từ đọc ghi đến track thích hợp (còn gọi là seek time).
- Thời gian chờ để trục quay thực hiện xoay đĩa từ đến đúng khối thích hợp ở dưới đầu từ (còn gọi là latency time).
- Thời gian để chuyển dữ liệu giữa đĩa và bộ nhớ chính (còn gọi là tranfer time)

Khi một tiến trình cần đọc hoặc ghi dữ liệu, tiến trình sẽ sử dụng lời gọi hệ thống (system call) gọi đến hệ điều hành để thực hiện việc xử lý đọc ghi dữ liệu lên đĩa. Lúc này hệ điều hành sẽ chuyển yêu cầu đến bộ điều khiển của ổ đĩa (disk controller), nếu ổ đĩa sẵn sàng phục vụ, disk controller sẽ thực hiện truy cập ngay lập tức. Nếu ổ đĩa đang trong trạng thái busy, những yêu cầu (request) sẽ được đưa vào hàng đợi (queue). Lúc này, việc xây dựng các thuật toán để bảo đảm các yêu cầu được thực hiện nhanh nhất chính là xây dựng các thuật toán lập lịch đĩa.

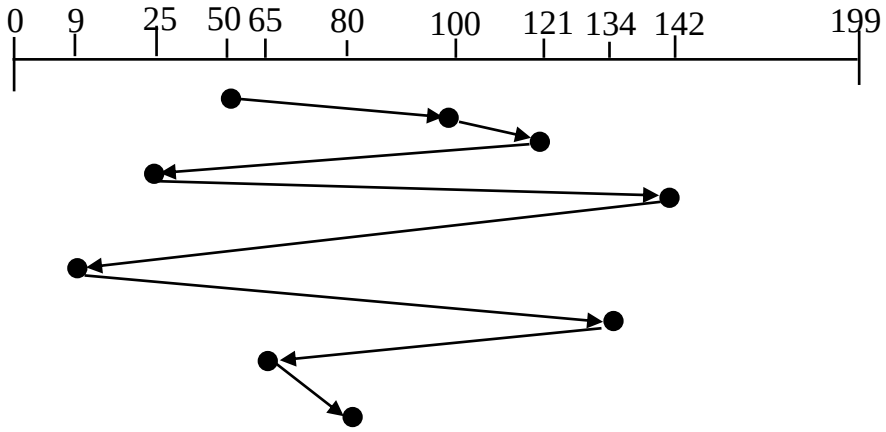
Có khá nhiều thuật toán lập lịch đĩa. Trong đó có thể kể đến những thuật toán phổ biến như sau:

- Thuật toán FCFS.
- Thuật toán SSTF.
- Thuật toán SCAN.
- Thuật toán C-SCAN.
- Thuật toán LOOK.
- Thuật toán C-LOOK.

4.2.1. Thuật toán FCFS

Thuật toán FCFS (first come first served). Các yêu cầu nào được gọi đến trước sẽ được thực hiện trước. Thuật toán này nhìn chung là công bằng với các yêu cầu nhưng nó không xử lý nhanh nhất khi đọc ghi dữ liệu.

Ví dụ 4.1: Giả sử rằng tiến trình yêu cầu đọc ghi dữ liệu ở ổ đĩa, các yêu cầu này được đưa vào hàng đợi của đĩa như sau: 100, 121, 25, 142, 9, 134, 65, 80. Các chỉ số này chính là địa chỉ đến từng block trên ổ đĩa để đầu đọc ghi phải di chuyển đến lấy dữ liệu. Hiện đầu đọc ghi đang ở vị trí thứ 50. Như vậy, thuật toán FCFS sẽ thực hiện đọc ghi dữ liệu như sau:



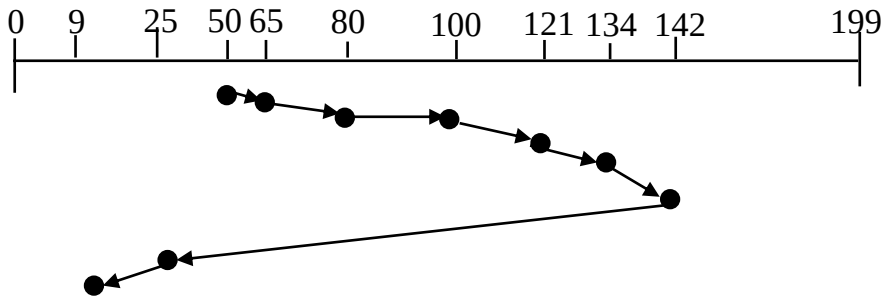
Hình 4.9. Thuật toán FCFS

Trong ví dụ trên, các chỉ số đi từ 0 – 199 là các vòng tròn đồng tâm trên cùng mặt đĩa. Đầu đọc ghi đang ở vòng tròn thứ 50, lúc này theo thuật toán, những yêu cầu nào đến trước sẽ được xử lý trước bằng cách di chuyển đầu đọc ghi đến đúng thứ tự vòng tròn để đọc ghi dữ liệu. Thứ tự di chuyển của đầu đọc sẽ là: 50, 100, 121, 25, 142, 9, 134, 65, 80.

4.2.2. Thuật toán SSTF

Thuật toán SSTF (shortest seek time first): thuật toán này sẽ thực hiện bằng cách lựa chọn những yêu cầu đang ở trong hàng đợi sao cho thời gian dịch chuyển từ vị trí của đầu đọc ghi hiện tại đến yêu cầu đó là ngắn nhất.

Ví dụ 4.2: cũng xét các yêu cầu như ví dụ ở thuật toán FCFS, các yêu cầu bao gồm: 100, 121, 25, 142, 9, 134, 65, 80. Đầu đọc ghi cũng đang ở vị trí 50, lúc này thuật toán SSTF sẽ thực hiện như sau:



Hình 4.10. Thuật toán SSTF

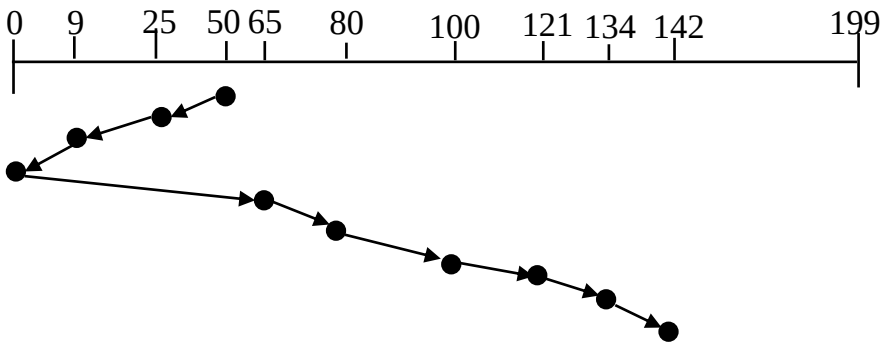
Trong hình trên, ta thấy thứ tự của yêu cầu lần lượt là 100, 121, 25, 142, 9, 134, 65, 80. Tuy nhiên, do đầu đọc ghi đang ở vị trí 50, nên thuật toán lựa chọn yêu cầu ở vị trí 65 là gần với đầu đọc ghi nhất, sau khi đã xử lý xong ở ví trí 65, sẽ lựa chọn tiếp vị trí 80, rồi đến vị trí 100, cứ như vậy thuật toán tiếp tục cho đến khi hoàn thành hết tất cả yêu cầu. Thứ tự hoàn chỉnh của giải thuật sẽ là: 50, 65, 80, 100, 121, 134, 142, 25, 9.

Trong thuật toán này, rất rõ ràng đã có sự cải tiến so với thuật toán FCFS, tuy nhiên, nếu các yêu cầu đi vào ngày càng nhiều và liên tục. Điều này sẽ dẫn đến khả năng có những yêu cầu sẽ chờ đợi lâu hơn dù được yêu cầu từ sớm.

4.2.3. Thuật toán SCAN

Thuật toán SCAN còn được gọi là thuật toán thang máy, vì thuật toán thực hiện theo nguyên tắc di chuyển đầu đọc ghi về một phía (vào bên trong hoặc bên ngoài), sau khi đã di chuyển đến tận cùng của đĩa sẽ di chuyển ngược lại. Và cứ thực hiện liên tục như thế.

Ví dụ 4.3: cũng lấy cùng một yêu cầu như trên, hàng đợi lần lượt là: 100, 121, 25, 142, 9, 134, 65, 80. Đầu đọc ghi cũng đang ở vị trí 50. Lúc này thuật toán SCAN sẽ thực hiện như sau:



Hình 4.11. Thuật toán SCAN

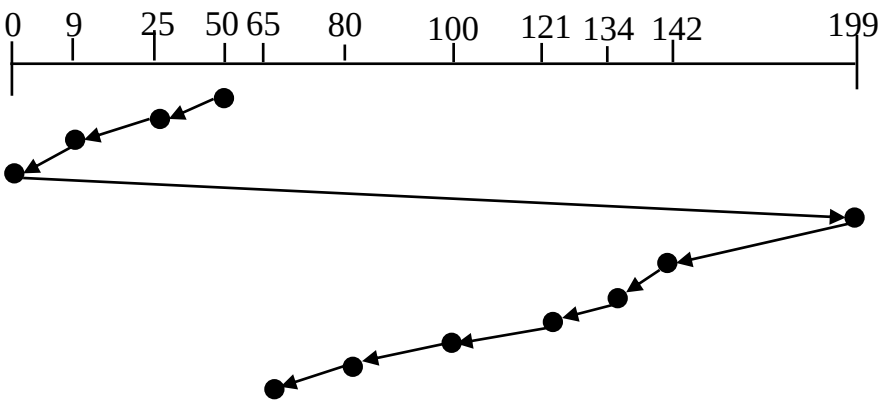
Trong ví dụ ở trên, ta thấy đầu đọc ghi đang ở vị trí 50, đầu đọc ghi sẽ dịch chuyển vào bên trong ổ đĩa, trên đường di chuyển sẽ thực hiện đọc ghi dữ liệu ở các vị trí trong hàng đợi. Cho đến khi di chuyển đến vị trí 0 (vị trí trong cùng) sẽ di chuyển ngược ra ngoài để thực đọc ghi cho phần dữ liệu còn lại. Thứ tự hoàn chỉnh của giải thuật sẽ là: 50, 25, 9, 0, 65, 80, 100, 121, 134, 142.

Đối với thuật toán SCAN, nếu những yêu cầu nào gọi đến các vị trí trên đường di chuyển theo hướng của đầu đọc, yêu cầu đó sẽ được xử lý ngay. Tuy nhiên, nếu yêu cầu đứng ở phía ngược lại hướng di chuyển của đầu đọc ghi, yêu cầu này phải chờ khá lâu để đầu đọc ghi quay ngược lại. Dĩ nhiên, thuật toán cũng khá tốn thời gian trong trường hợp đầu đọc ghi đã xử lý xong các yêu cầu, nhưng đầu đọc ghi vẫn phải đi đến cuối vị trí đĩa mới quay lại, dù không còn bất cứ yêu cầu nào sau đó.

4.2.4. Thuật toán C-SCAN

Thuật toán C-SCAN là thuật toán có ý tưởng giống như thuật toán SCAN, đầu đọc ghi cũng phải di chuyển từ một hướng đến hướng còn lại. Tuy nhiên, khác biệt là khi đầu đọc ghi di chuyển đến điểm cuối cùng, đầu đọc ghi sẽ được đưa trở lại đầu ngược lại của ổ đĩa để di chuyển tiếp.

Ví dụ 4.4: Cũng xét ví dụ như trên.



Hình 4.12. Thuật toán C-SCAN

Trong ví dụ trên, ta thấy rằng khi đầu đọc ghi di chuyển đến vị trí trong cùng (0) sẽ được di chuyển đến đầu ngược lại của ổ đĩa (vị trí 199), từ đó mới tiếp tục xử lý các yêu cầu còn lại. Thứ tự hoàn chỉnh của giải thuật là: 50, 25, 9, 0, 199, 142, 134, 121, 100, 80, 65.

4.2.5. Thuật toán LOOK

Thuật toán LOOK là thuật toán có ý tưởng và cách thức hoạt động giống như thuật toán SCAN, tuy nhiên điểm khác biệt đó là khi thuật toán quy định khi xử lý đến yêu cầu cuối cùng của một phía, nó sẽ quay ngược lại các yêu cầu theo hướng ngược lại, chứ không cần đi đến điểm cuối cùng của ổ đĩa.

Thuật toán C-LOOK cũng tương tự như thuật toán C-SCAN, tuy nhiên điểm khác biệt là thuật toán quy định, khi xử lý xong yêu cầu cuối cùng của một phía, nó sẽ di chuyển về yêu cầu gần nhất với đầu mốc còn lại của ổ đĩa. Một cách đơn giản là thuật toán sẽ không di chuyển về phía điểm cuối cùng của ổ đĩa rồi mới quay lại, mà sẽ quay lại ngay sau khi hoàn thành yêu cầu sau cùng.

Nhìn chung các thuật toán lập lịch đĩa đều có những ưu điểm, nhược điểm của từng thuật toán. Trên thực tế, các thuật toán lập lịch đĩa được các nhà sản xuất ổ đĩa tích hợp vào Disk Controller của ổ đĩa. Điều này sẽ giúp tăng hiệu năng xử lý của ổ đĩa. Hệ điều hành sẽ tập trung vào xử lý lập lịch độ ưu tiên của các service.

Các thuật toán lập lịch được xét ở trên áp dụng đối với các ổ đĩa sử dụng đĩa từ để lưu dữ liệu như ổ HDD. Đối với các ổ đĩa SSD, bản thân ổ đĩa SSD không có đầu

đọc ghi, cũng không có các bề mặt đĩa từ. Ổ đĩa SSD lưu trữ dữ liệu dựa trên các chip điện tử. Do đó, đối với thuật toán lập lịch trên ổ đĩa SSD, hầu như đều dùng thuật toán FCFS. Bất kỳ yêu cầu nào đến trước sẽ được gọi vào xử lý trước.

4.3. Hệ thống tập tin

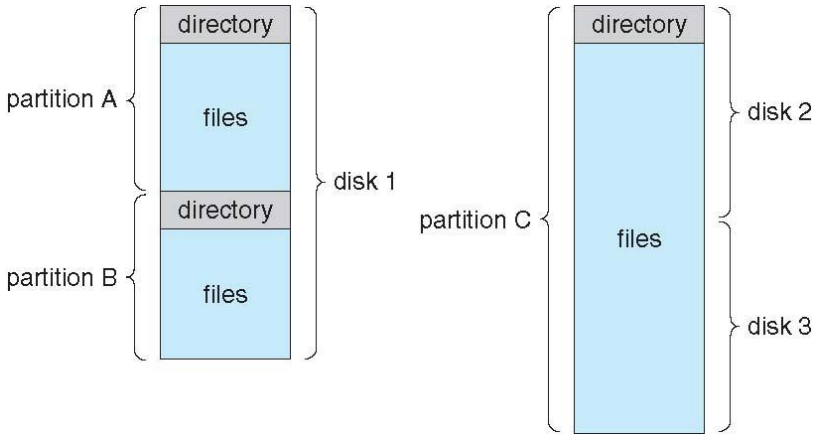
4.3.1. Giới thiệu hệ thống tập tin

Người dùng khi sử dụng máy tính sẽ không muốn quan tâm đến cách thức ổ đĩa lưu trữ dữ liệu. Họ cũng không cần quan tâm đến những khái niệm như sectors, cylinder, blocks,… Khi sử dụng máy tính, họ chỉ cần quan tâm đến đường dẫn đến tập tin và thư mục mà họ làm việc. Để thực hiện được điều này, trong hệ thống máy tính, hệ thống tập tin (file system) được thiết lập.

Hệ thống tập tin được sử dụng để kiểm soát dữ liệu, lưu trữ và phục hồi dữ liệu. Nếu không có hệ thống tập tin, thông tin được lưu trong những khối lớn sẽ không có cách nào để tìm thấy vị trí bắt đầu và vị trí kết thúc, vị trí tiếp theo của dữ liệu. Khi đã thiết lập hệ thống tập tin, dữ liệu sẽ được chia thành các mảnh và mỗi mảnh được đặt tên, vì thế thông tin dễ dàng được đánh dấu và xác thực.

Khái niệm hệ thống tập tin được hiểu là các phương pháp và cấu trúc dữ liệu mà một hệ điều hành sử dụng để theo dõi các tập tin trên ổ đĩa hoặc các phân vùng. Hệ thống tập tin cũng kiểm soát cách lưu trữ và truy xuất dữ liệu.

Có rất nhiều loại hệ thống tập tin. Mỗi loại đều có cấu trúc và logic khác nhau, tốc độ, khả năng bảo mật, khả năng lưu trữ,.. cũng đều khác nhau. Một số hệ thống tập tin được thiết kế để sử dụng cho những ứng dụng đặc biệt. Ví dụ, ISO 9660 file system được thiết đặc biệt cho đĩa quang. Hệ thống tập tin có thể sử dụng nhiều trên loại thiết bị lưu trữ và các loại phương tiện truyền thông khác nhau. Nổi bật và thông dụng nhất là các loại ổ đĩa cứng.



Hình 4.13. Tổ chức hệ thống tập tin thông thường [3]

Đối với kiến trúc của hệ thống tập tin, thông thường một hệ thống tập tin bao gồm hai hoặc ba lớp. Đôi khi các lớp được phân tách rõ ràng và đôi khi các chức năng được kết hợp lại với nhau.

- Lớp hệ thống tập tin logic (logical file system) chịu trách nhiệm tương tác với ứng dụng của người dùng. Nó cung cấp giao diện chương trình ứng dụng (API) cho các thao tác với tập tin như OPEN, CLOSE, READ,... và chuyển thao tác được yêu cầu đến lớp bên dưới để xử lý. Hệ thống tập tin logic quản lý việc mở các mục trong bảng tập tin (file table) và các file descriptor của từng tiến trình (process).
- Lớp thứ hai là hệ thống tập tin ảo (virtual file system). Lớp này hỗ trợ nhiều đối tượng cùng gọi đồng thời của các hệ thống tập tin vật lý, mỗi phiên đối tượng khi đó gọi là hệ thống tập tin thực thi (file system implementation).
- Lớp thứ ba là hệ thống tập tin vật lý (physical file system). Lớp này liên quan đến hoạt động vật lý của thiết bị lưu trữ. Nó xử lý các khối vật lý đang được đọc hoặc ghi, bộ đệm, quản lý bộ nhớ và chịu trách nhiệm cho việc đặt các khối vật lý vào các vị trí cụ thể trên phương tiện lưu trữ. Hệ thống tập tin vật lý tương tác với trình điều khiển thiết bị hoặc với kênh để điều khiển thiết bị lưu trữ.

Thông thường, người ta thường sử dụng một hệ thống tập tin trên toàn bộ thiết bị lưu trữ. Tuy nhiên, trong một số trường hợp, vẫn có thể có một số hệ thống tập tin khác nhau có thể cùng được sử dụng. Ví dụ dễ thấy nhất là ảo hóa, user có thể chạy định dạng ext4 của Linux trên máy ảo, mà máy ảo đó được lưu trữ trên định dạng NTFS Windows.

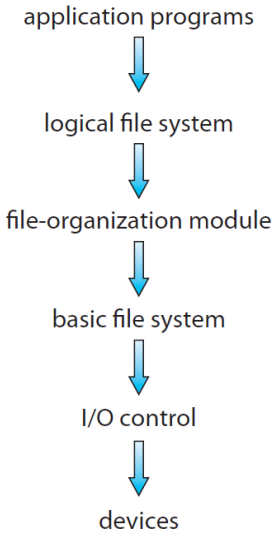
Để sử dụng hệ thống tập tin dễ dàng với người dùng cần phải có một giao diện được cung cấp bởi hệ điều hành. Giao diện này có thể là các dòng lệnh như shell unix chẳng hạn. Giao diện này cũng có thể là giao diện đồ họa như file explorer trong windows. Điều này giúp người dùng sử dụng dễ dàng mà không quá quan tâm đến những kiến trúc và cách thức xử lý phức tạp bên dưới.

Ngày nay, tương ứng với các hệ điều hành khác nhau sẽ có nhiều hệ thống tập tin khác nhau. Điển hình như:

- Windows có các hệ thống tập tin: FAT, NTFS, exFAT.
- Linux có các hệ thống tập tin: EXT, XFS, JFS,…
- MAC OS có các hệ thống tập tin: APFS, HFS Plus.
- Ngoài ra, còn có nhiều hệ thống tập tin để hỗ trợ cho các hệ điều hành đặc thù với từng loại thiết bị riêng biệt.

4.3.2. Thực thi hệ thống tập tin

Khi người dùng sử dụng máy tính, người ta sử dụng các hệ thống tập tin để làm các công việc liên quan đến quản lý tập tin và thư mục. Tuy nhiên, về mặt bản chất, khi người dùng thực hiện các thao tác đến tập tin và thư mục, quá trình phải trải qua nhiều thành phần hơn. Quá trình từ thao tác ở tập tin và thư mục đến khi xử lý việc đọc ghi dữ liệu ở ổ đĩa có thể mô tả ở hình sau:



Hình 4.14. Các tầng xử lý của hệ thống tập tin [3]

Người dùng máy tính sẽ thực hiện các công việc của mình thông qua các chương trình ứng dụng (application programs, ví dụ như phần mềm microsoft word, powerpoint,..). Khi người dùng bắt đầu thực hiện lưu trữ dữ liệu hoặc đọc các dữ liệu từ máy tính. Lúc này các chương trình ứng dụng sẽ gọi đến hệ thống tập tin logic để xử lý tiếp.

Hệ thống tập tin logic sẽ quản lý các thông tin siêu dữ liệu (metada information). Các thông tin siêu dữ liệu này bao gồm tất cả cấu trúc của hệ thống tập tin, nhưng không chứa các thông tin về dữ liệu (như nội dung của tập tin). Hệ thống tập tin logic sẽ quản lý cấu trúc thư mục, để cung cấp các thông tin về cách truy cập đến các tập tin và thư mục con bên trong. Đối với tập tin, hệ thống tập tin logic sẽ chứa các thông tin file control blocks. File control blocks là nơi lưu trữ tất cả thông tin về quyền sở hữu (ownership, permissions, và nơi lưu trữ nội dung tập tin). Có thể nói, hệ thống tập tin logic sẽ quản lý những thông tin về tập tin và thư mục của người dùng, nhưng không chứa dữ liệu thật sự của tập tin và thư mục.

Sau khi hệ thống tập tin logic thông qua file control blocks để biết được nơi lưu trữ tập tin thông qua các blocks. Lúc này file – organization module sẽ chuyển đổi các thông tin về blocks này thành các địa chỉ vật lý tương ứng. Do các blocks vật lý thường chứa dữ liệu một cách rời rạc, không liên tục, do đó phải có một bản dịch để sắp xếp lại cho phù hợp để sử dụng. File – organization module cũng quản lý thông tin về các khối chưa được sử dụng, và cung cấp khối này khi được yêu cầu lưu trữ dữ liệu.

Lúc này, hệ thống tập tin cơ bản sẽ thực hiện ra các chỉ thị như đọc ghi các khối vật lý được chuyển từ file – organization moudule để đưa các lệnh này xuống cho ổ đĩa thực hiện.

Để ổ đĩa tiếp nhận các lệnh từ hệ thống tập tin cơ bản, I/O control sẽ sử dụng các driver của thiết bị để kết nối và chuyển thông tin từ máy tính đến ổ đĩa. Mỗi thiết bị sẽ được kết nối với máy tính thông qua I/O port khác nhau như: USB, VGA, HDMI, LAN,…

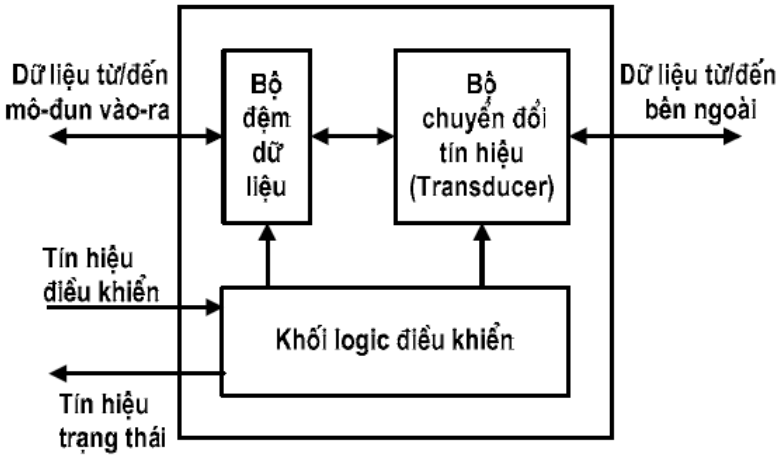
Lúc này dữ liệu đã được gửi đến các thiết bị để xử lý đọc ghi theo yêu cầu. Hiển nhiên, mỗi thiết bị cũng sẽ có một bộ controller để điều khiển việc đọc ghi và kết nối với máy tính.

Đối với việc phân chia các tầng để thực thi hệ thống tập tin như trên đôi khi có thể dẫn đến sự quá tải của hệ điều hành, điều này làm dẫn đến giảm hiệu năng của máy tính. Tuy nhiên, việc phân chia ra thành nhiều tầng giúp cho việc phát triển của các thiết bị sẽ không ảnh hưởng đến hệ điều hành và các chương trình ứng dụng của người dùng.

Ví dụ 4.5: khi các hãng sản xuất thiết bị lưu trữ phát triển một công nghệ lưu trữ mới, xử lý dữ liệu nhanh hơn, dung lượng lưu trữ lớn hơn. Lúc này thiết bị chỉ cần cài đặt driver để có thể kết nối đến máy tính. Từ đây, các chương trình ứng dụng của người dùng hoàn toàn có thể giao tiếp với thiết bị lưu trữ một cách dễ dàng. Hay ngược lại, khi một chương trình ứng dụng mới được xây dựng, chương trình này sẽ thông qua các lời gọi hệ thống để chuyển các thông tin xử lý dữ liệu mà chương trình cần đến hệ thống tập tin. Từ đây, tiếp tục xử lý theo để chuyển yêu cầu đến thiết bị lưu trữ. Khi một hệ thống tập tin phát triển mới cũng vậy, chỉ cần đảm bảo có thể đáp ứng các yêu cầu truyền tải thông tin mà hệ điều hành có thể hiểu thì việc phát triển hệ thống tập tin mới vẫn có thể bảo đảm kết nối với các thiết bị lưu trữ hay hỗ trợ người dùng sử dụng các chương trình ứng dụng một các bình thường.

4.4. Hệ thống I/O

Hệ thống máy tính khi làm việc cần kết nối với rất nhiều loại thiết bị. Các thiết bị đó được gọi chung là thiết bị ngoại vi (chuột, bàn phím, loa, màn hình, USB, DVD,…). Mỗi loại thiết bị sẽ có cách vận hành và xử lý dữ liệu hoàn toàn riêng biệt. Để máy tính có thể kết nối đến thiết bị này một cách dễ dàng, hệ thống I/O được thiết lập để làm nhiệm vụ kết nối giữa máy tính và các thiết bị ngoại vi.

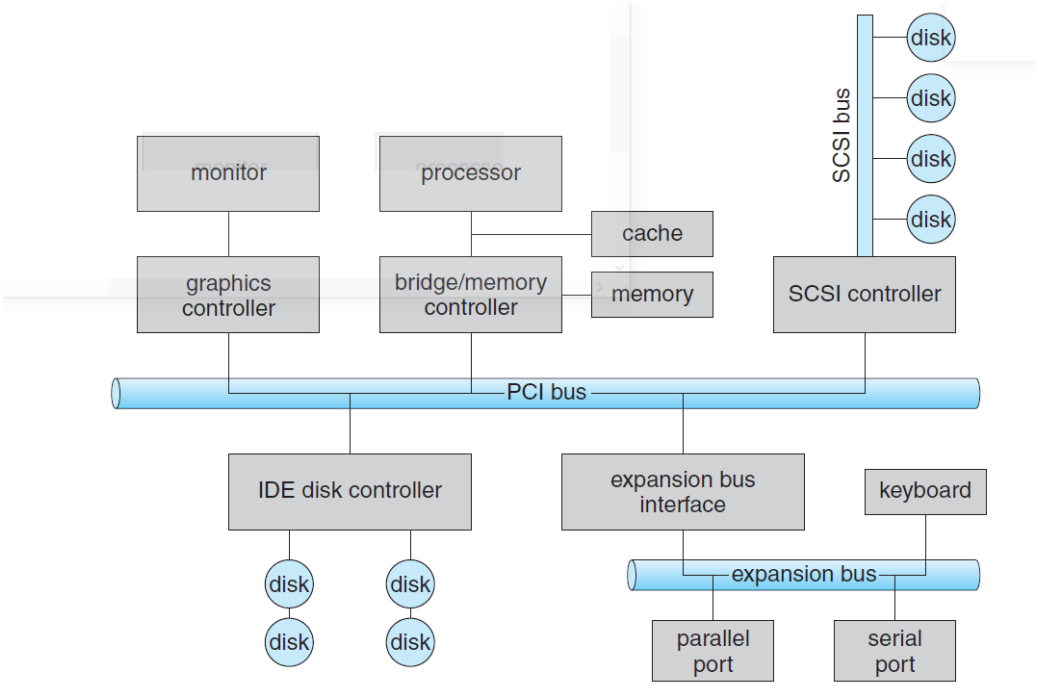


Hình 4.15. Cấu trúc chung của các thiết bị ngoại vi

Driver là một chương trình hoặc một tập các lệnh để giúp thông dịch qua lại giữa hệ điều hành trên máy tính và thiết bị. Thông qua driver, hệ điều hành có thể nhận biết thiết bị, kết nối và xử lý thông tin trên thiết bị. Ví dụ: khi một máy in được thực hiện kết nối vào máy tính, lúc này máy tính vẫn chưa thể thực hiện in dữ liệu lên máy in được. Máy tính cần cài đặt driver để máy tính có thể giao tiếp được với thiết bị đó. Sau khi đã cài đặt driver tương ứng với máy in, lúc này ta có thể in dữ liệu lên máy in đang có. Mỗi loại thiết bị khác nhau bắt buộc phải có driver khác nhau để kết nối đến thiết bị.

Để kết nối các thiết bị vào máy tính, thiết bị phải kết nối thông qua các điểm kết nối, được gọi là I/O Port (cổng Serial, USB, VGA, HDMI,…). Các thiết bị sẽ sử dụng những đường truyền tín hiệu để truyền dữ liệu, các lệnh điều khiển từ thiết bị đến máy tính thông qua hệ thống bus. Có nhiều loại bus như: PCI, SCSI, PCIe (PCI express).

Để có thể vận hành I/O port, bus,.. các thiết bị cần có một bộ điều khiển để có thể vận hành toàn bộ quá trình của thiết bị.



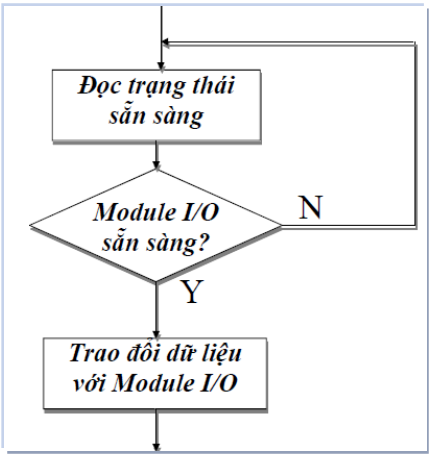
Hình 4.16. Hệ thống bus thông dụng [3]

Trong hình trên, khi Processor gửi một thông điệp đến các thiết bị ngoại vi để xử lý. Vấn đề đặt ra là làm thế nào để dữ liệu hoặc lệnh gửi đến đúng thiết bị hay đúng cổng I/O đang kết nối dến thiết bị. Có 2 cách để thực hiện kết nối để trao đổi dữ liệu giữa processor và thiết bị ngoại vi:

- Port – mapped I/O : Loại này dùng các CPU Instruction đặc biệt, được thiết kế đặc biệt cho các loại ngoại vi. Có nghĩa là mỗi loại thiết bị ngoại vi sẽ được thiết lập các lời gọi hệ thống đặc biệt riêng, khi cần gọi đến thiết bị nào sẽ sử dụng tương ứng đến lời gọi hệ thống đó. Các kiến trúc Intel dùng loại này.
- Memory – mapped I/O : Loại này không cần bất kỳ Instruction đặc biệt nào. Mỗi thanh ghi được gán 1 địa chỉ bộ nhớ trong không gian địa chỉ bộ nhớ của CPU. Được thực hiện bởi các lệnh load/store riêng của CPU. Kiến trúc ARM dùng loại này.

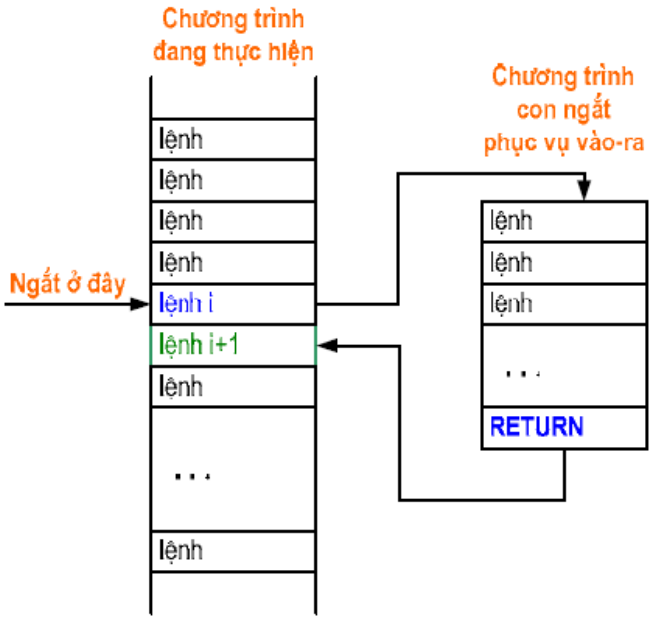
Hai cách trên giúp Processor có thể kết nối đúng cổng mà thiết bị ngoại vi đang kết nối. Đôi khi có những hệ thống sử dụng kết hợp cả hai cách như trên. Vấn đề được đặt ra tiếp theo là, khi đã kết nối được đến đúng cổng, vậy máy tính và thiết bị ngoại vi thực hiện trao đổi vào ra (I/O) như thế nào? Thông thường có 3 phương pháp để giúp máy tính thực hiện I/O với các thiết bị ngoại vi:

- Thực hiện I/O bằng chương trình (I/O Programme): Processor sẽ điều khiển I/O trực tiếp bằng chương trình. Do đó, cần phải lập trình các vấn đề I/O trên chương trình vận hành.



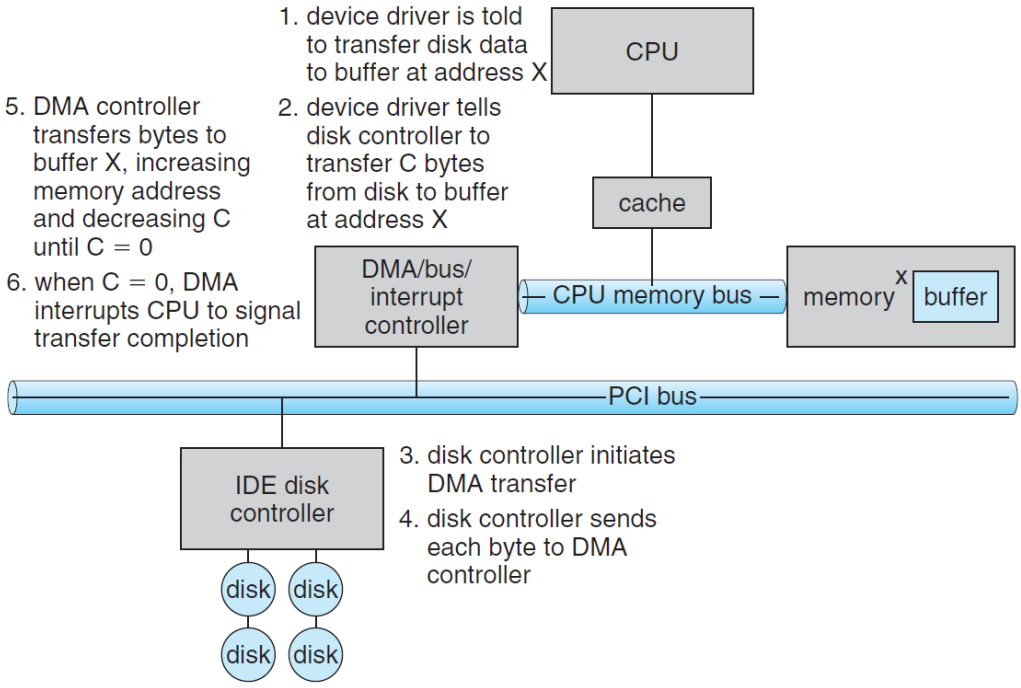
Hình 4.17. Thực hiện I/O bằng chương trình

- Thực hiện I/O bằng ngắt (interrupts): CPU phải được hỗ trợ phần cứng (hardware) được gọi là interrupt - request line. Khi CPU phát hiện ra tín hiệu trên interrupt – request line báo, CPU sẽ thực hiện lưu lại trạng thái mà chương trình đang vận hành để chuyển sang trạng thái ngắt. Lúc này, việc thực hiện I/O giữa máy tính và thiết bị ngoại vi được thực hiện. Sau khi hoàn thành, CPU sẽ khôi phục trở lại trạng thái trước khi bắt đầu ngắt để thực hiện tiếp chương trình.



Hình 4.18. Thực hiện I/O bằng ngắt

- Thực hiện I/O bằng DMA (Direct Memory Access – Truy cập trực tiếp): đây là các thức cho phép truyền dữ liệu trực tiếp giữa thiết bị I/O và bộ nhớ của hệ thống mà không cần sự can thiệp của CPU. Điều này giúp tối ưu hóa hiệu suất của hệ thống và giảm thiểu tải cho vi xử lý. Để thực hiện được thì trên BUS của máy tính cần trang bị thêm một bộ phần cứng là DMA Controller (DMAC). Lúc này CPU khi muốn trao đổi dữ liệu với thiết bị ngoại vi, CPU chỉ cần thông báo đến cho DMAC, sau đó CPU sẽ tiếp tục thực hiện công việc khác mà không cần phải chờ. Trong khi đó, DMAC sẽ thay CPU để tiếp tục xử lý truyền và xử lý thông tin với các thiết bị ngoại vi.



Hình 4.19. Cách thức DMA chuyển tải dữ liệu [3]

Nhìn chung đối với phương pháp I/O bằng chương trình hay bằng ngắt thì đều cần CPU xử lý, do đó tốn thời gian và tốn tài nguyên của máy tính. Đối với phương pháp sử dụng DMA sẽ cho phép truyền dữ liệu trực tiếp giữa I/O và bộ nhớ của hệ thống mà không thông qua vi xử lý, DMA giúp tối ưu hóa hiệu suất và giảm thiểu tải cho CPU.

Bài tập chương 4

Bài tập 4.1.

Giả sử đầu đọc đang ở vị trí cylinder 57, cần đọc các cylinder: 98, 189, 41, 121, 15, 127, 64, 67 (xét trong miền 0..199). Hãy liệt kê lần lượt các cylinder mà đầu đọc đi qua khi sử dụng các thuật toán lập lịch đĩa (disk scheduling algorithm) sau đây; từ đó hãy tính tổng khoảng cách di chuyển của đầu đọc ứng với mỗi thuật toán.

a. Thuật toán lập lịch đĩa SSTF (shortest seek time first algorithm).
b. Thuật toán lập lịch đĩa SCAN (cần giải quyết cho cả hai trường hợp: đầu đọc di chuyển về hướng bên trái, đầu đọc di chuyển về hướng bên phải).

Bài tập 4.2.

Giả sử đầu đọc đang ở vị trí cylinder 53, cần đọc các cylinder: 98, 184, 37, 122, 12, 124, 65, 67 (xét trong miền 0..199). Hãy liệt kê lần lượt các cylinder mà đầu đọc đi qua khi sử dụng các thuật toán lập lịch đĩa (disk scheduling algorithm) LOOK, C-SCAN, C-LOOK; từ đó hãy tính tổng khoảng cách di chuyển của đầu đọc ứng với mỗi thuật toán.

Bài tập 4.3.

Cho 1 ổ đĩa có 5000 Cylinder, đánh số từ 0 – 4999. Lúc này đầu đọc đang ở vị trí 2150, đầu đọc vừa mới di chuyển đến vị trí 2150 từ vị trí 1805. Trong hàng chờ của đầu đọc đang có những request như sau: 2069, 1212, 2296, 2280, 544, 1618, 356, 1523. Hãy sử dụng các giải thuật FIFO, SSTF, SCAN, LOOK, C-SCAN, C-LOOK để mô tả quá trình thực hiện của ổ đĩa. Với mỗi giải thuật hãy tính xem đầu đọc phải di chuyển qua tổng cộng tất cả bao nhiêu Cylinder để hoàn thành. Từ đó so sánh độ hiệu quả của các giải thuật.

Bài tập 4.4.

Hãy thực hiện lập trình cho các giải thuật FIFO, SSTF, SCAN, LOOK, C-SCAN, C-LOOK. Với yêu cầu: cho người dùng nhập thông tin của vị trí đầu đọc ghi hiện tại và vị trí trước đó. Các yêu cầu trong hàng đợi được nhập từ file. Chương trình sẽ cho biết các vị trí nào

được thực hiện theo thứ tự. Đồng thời tính khoảng cách các Cylinder mà đầu đọc ghi đã đi qua.

Bài tập 4.5.

Hãy so sánh hệ thống tập tin NTFS và FAT. Cho biết khi nào thì nên dùng FAT, khi nào nên dùng NTFS.

Bài tập 4.6.

Hãy liệt kê các phiên bản của hệ thống tập EXT trên Linux. Cho biết mỗi phiên bản có đặc điểm như thế nào?

Chương 5. BẢO VỆ VÀ AN TOÀN HỆ THỐNG

Trong việc sử dụng máy tính, an toàn và bảo mật được hiểu như thế nào là chính xác? Trong vấn đề bảo mật, vấn đề bảo mật (protection) và vấn đề an toàn (security) cần được phân biệt như thế nào cho chính xác? Ngày nay, những phương pháp tấn công máy tính phổ biến như DoS, Sniffing hay Spoofing có thể gây ra những nguy hiểm thế nào?

5.1. Các khái niệm cơ bản

Ngày nay, với sự phát triển nhanh chóng của Internet, các thiết bị kết nối và làm việc với nhau thành các hệ thống rộng rãi trên toàn thế giới. Tuy nhiên, vấn đề đặt ra là sự an toàn của các hệ thống này cần được bảo vệ như thế nào ? Thực tế cho thấy, ngày nay, các doanh nghiệp, tổ chức, chính phủ đều rất chú trọng về vấn đề an toàn và bảo mật trên hệ thống máy tính. Bất kỳ các cuộc tấn công hay xâm nhập của giới tội phạm công nghệ cao đều để lại những hậu quả rất lớn cho các doanh nghiệp, tổ chức, chính phủ trên thế giới.

Khái niệm về vấn đề bảo mật là rất rộng, phức tạp, đa dạng và có nhiều phạm trù liên quan đến các khía cạnh khác nhau của an ninh thông tin và hệ thống. Dưới đây là một số phạm trù trong vấn đề bảo mật:

- Bảo mật máy tính.
- Bảo mật dữ liệu.
- An ninh mạng.
- Bảo mật ứng dụng.
- Bảo mật các thiết bị IoT.
- …

Có thể nói có rất nhiều phạm trù xoay quanh vấn đề bảo mật ngày nay. Trong chương này, chúng ta sẽ tập trung tìm hiểu về các vấn đề bảo mật và an toàn trên máy tính. Có thể hiểu một cách đơn giản thì: bảo mật máy tính là một tập hợp các biện pháp và quy trình được thiết kế để bảo vệ thông tin, dữ liệu, hệ thống, và các tài nguyên liên quan trên máy tính khỏi những nguy cơ, mối đe dọa và việc truy cập trái phép. Mục tiêu của bảo mật máy tính là đảm bảo tính riêng tư, toàn vẹn và sẵn sàng của các thông tin và tài nguyên quan trọng trên hệ thống, đồng thời ngăn chặn hoặc giảm thiểu rủi ro từ các hoạt động tấn công, phá hoại hoặc lừa đảo.

Trong vấn đề bảo mật, ta cần phải phân biệt giữa 2 vấn đề là bảo vệ và an toàn.

Bảo vệ (Protection) là các cơ chế để cho phép truy cập vào chương trình, tiến trình hay tài nguyên người dùng trên máy tính. Hệ thống bảo vệ cung cấp phương thức để phân biệt giữa truy cập được phép và truy cập bị cấm.

An toàn (Security) định nghĩa các cách thức xác thực người dùng để bảo đảm tính toàn vẹn của thông tin trên hệ thống (bao gồm cả dữ liệu và cả những đoạn code của chương trình), cũng như các tài nguyên vật lý của máy tính. An toàn không chỉ đòi hỏi phải có hệ thống bảo vệ phù hợp, mà còn phải giám sát tác động từ môi trường bên ngoài. Bảo vệ sẽ trở nên vô ích nếu người tấn công có thể điều khiển hệ thống hoặc dễ dàng xóa bỏ file được lưu trữ trên ổ đĩa, băng từ,...). An toàn còn thuộc về chính sách quản lý, chứ không chỉ là vấn đề của riêng hệ điều hành.

5.2. Bảo vệ hệ thống

Đối với vấn đề bảo vệ hệ thống, các hệ điều hành thường áp dụng nguyên tắc: đặc quyền tối thiểu (principle of least privilege). Điều này có nghĩa là: hệ điều hành sẽ cung cấp các lời gọi hệ thống, các dịch vụ để các chương trình ứng dụng có thể thông qua những công cụ này nhằm truy cập vào các thành phần bên dưới của máy tính. Hiển nhiên, chỉ khi chương trình hay tiến trình cần thiết thì mới được hệ điều hành cho quyền truy cập, sau khi thực hiện xong công việc, quyền truy cập sẽ bị thu hồi. Điều này giúp bảo vệ cho hệ thống tốt hơn. Có thể hiểu một cách đơn giản như sau: một người bảo vệ chỉ được phép cầm chìa khóa của một khu vực nhất định trong phiên trực của anh ta. Nếu người bảo vệ có tất cả chìa khóa của toàn bộ khu vực thì việc mất mát tài sản sẽ

khó kiểm soát. Dĩ nhiên, khi đã hết phiên trực thì người bảo vệ phải trả lại chìa khóa để bảo đảm an toàn. Nguyên tắc đặc quyền tối thiểu cũng tương tự như vậy.

Vấn đề tiếp theo ta tìm hiểu bảo vệ hệ thống sẽ bảo vệ những mục tiêu nào? Và một số cách thức mà nó thực hiện.

5.2.1. Mục tiêu của bảo vệ hệ thống

Mục tiêu của việc bảo vệ hệ thống là:

- **Bảo vệ chống lỗi của tiến trình**: khi có nhiều tiến trình cùng hoạt động, lỗi của một tiến trình phải được ngăn chặn không cho lan truyền trên hệ thống làm ảnh hưởng đến các tiến trình khác. Đặc biệt, qua việc phát hiện các lỗi tiềm ẩn trong các thành phần của hệ thống có thể tăng cường độ tin cậy của hệ thống (reliability).
- **Chống sự truy xuất bất hợp lệ**: Bảo đảm các bộ phận tiến trình sử dụng tài nguyên theo một cách thức hợp lệ được qui định trong việc khai thác các tài nguyên này.

Vai trò của bảo vệ trong hệ thống là cung cấp cơ chế để áp dụng các chiến lược quản trị trong việc sử dụng tài nguyên.

- **Cơ chế (mechanism):** xác định làm thế nào để thực hiện việc bảo vệ, có thể có các cơ chế phần mềm hoặc cơ chế phần cứng.
- **Chiến lược (policy):** quyết định việc bảo vệ được áp dụng như thế nào: những đối tượng nào trong hệ thống cần được bảo vệ, và các thao tác thích hợp trên các đối tượng này.

Để hệ thống có tính tương thích cao, cần phân tách các cơ chế và chiến lược được sử dụng trong hệ thống. Các chiến lược sử dụng tài nguyên là khác nhau tùy theo ứng dụng, và thường dễ thay đổi. Thông thường, các chương trình ứng dụng sẽ được tích hợp các chiến lược của riêng mình để bảo vệ và thực hiện ứng dụng thuận tiện và chính xác hơn. Trong khi đó, hệ thống sẽ cung cấp các chiến lược để bảo vệ cho hệ thống và người dùng hệ thống.

5.2.2. Kiểm chứng

Kiểm chứng (verification) là một thành phần quan trọng của bảo vệ hệ thống. Kiểm chứng là quá trình xác nhận tính toàn vẹn, độ tin cậy và an toàn của một hệ thống máy tính. Mục tiêu của kiểm chứng là đảm bảo rằng hệ thống hoạt động đúng theo thiết kế và không bị tác động bởi các yếu tố độc hại hay sai sót. Việc kiểm chứng trong an toàn máy tính giúp đảm bảo rằng hệ thống hoạt động như mong đợi, giảm thiểu nguy cơ tấn công, đánh cắp thông tin hoặc phá hoại dữ liệu.

Để thực hiện việc kiểm chứng, người ta thường xem xét những phạm trù: Miền bảo vệ, ma trận quyền truy xuất, quyền truy cập.

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,...). 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.

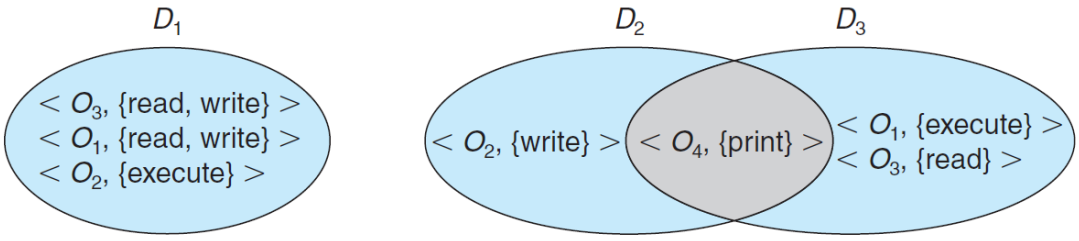
Mỗi tiến trình trong hệ thống đều hoạt động trong một miền bảo vệ (protection domain) nào đó. Một miền bảo vệ sẽ xác định các tài nguyên (đối tượng) mà những tiến trình hoạt động trong miền bảo vệ này có thể sử dụng, và các thao tác hợp lệ mà tiến trình này có thể thực hiện trên những tài nguyên đó.

Ví dụ 5.1 : <File F, {read, write}>: có nghĩa là đối tượng File F, có thể được các đối tượng khác thao tác thông qua thao tác đọc (read) hoặc ghi (write).

Cấu trúc của 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 xuất.



Hình 5.1. Hệ thống với ba miền bảo vệ [3]

Trong hình trên, D_1, D_2, D_3 là các miền bảo vệ của hệ thống; O_1, O_2, O_3, O_4 là các đối tượng; các thao tác bao gồm read, write, execute. Mỗi đối tượng với thao tác tương ứng sẽ thuộc vào một hoặc nhiều miền bảo vệ. D_2 và D_3 giao nhau ở <O_4, {print}>.

Mỗi tiến trình có thể liên kết với một miền bảo vệ, liên kết này có 2 dạng: tĩnh hoặc động.

Liên kết tĩnh: trong suốt thời gian làm việc 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 trinh 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:

Người dùng (user): trong trường hợp này, tập các đối tượng được phép truy xuất phụ thuộc vào định danh (UID) 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 (PID), 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.

Ma trận quyền truy xuất (access matrix)

Mô hình miền bảo vệ có thể được biểu diễn thành ma trận quyền truy xuất. 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 D_i co thể thao tác trên đối tượng O_j.

Bảng 5.1. Ma trận quyền truy xuất

Đối tượng / Miền bảo vệ	F_1 (file)	F_2 (file)	F_3 (file)	P (printer)
D_1	Read		Read	
D_2				Print
D_3		Read	Execute	
D_4	Read, Write		Read, Write	

Trong bảng trên, ta xét ma trận gồm 4 miền và 4 đối tượng: ba file (F_1, F_2, F_3) và máy in laser (P). Khi thực thi trong miền D_1, tiến trình có thể đọc file F_1 và F_3. Tiến trình thực thi trong miền D_3 có quyền đọc file F_2, nhưng không có quyền đọc file F_1 hay F_3. Chú ý, chỉ tiến trình thực thi trong miền D_2 mới in được trên máy in laser.

Thông qua ma trận truy cập, hệ thống có thể cài đặt nhiều loại chính sách khác nhau. Cơ chế bảo vệ ở đây gồm hai phần: cài đặt ma trận và đảm bảo các nguyên tắc an ninh của hệ thống. Hệ thống phải đảm bảo tiến trình thực thi trong miền D_i chỉ có thể thực hiện được đúng các thao tác đã liệt kê tại các phần tử ở hàng i. Chính sách bảo vệ

xác định nội dung các phần từ của ma trận quyền truy cập. Người dùng có thể quyết định nội dung các phần từ của ma trận truy cập. Khi người dùng tạo mới đối tượng O_J, cột O_j được thêm vào ma trận với các giá trị khởi tạo ban đầu do người dùng thiết lập. Khi cần thiết, người dùng có thể thêm quyền vào các ô ở cột *j* của ma trận.

Bảng 5.2. Ma trận quyền truy xuất với thao tác chuyển miền

Object / Domain	F_1 (file)	F_2 (file)	F_3 (file)	P (printer)	D_1	D_2	D_3	D_4
D_1	Read		Read			Switch		
D_2				Print			Switch	Switch
D_3		Read	Execute					
D_4	Read, Write		Read, Write		Switch			

Trong bảng trên những thao lác có thế thực hiện được trên các đối tượng đặc biệt (miền bảo vệ và ma trận truy cập) và làm thế nào dế các tiến trình có thế thực thi được các thao tác đó. Tiến trình có thể chuyển từ miền bảo vệ D_i sang D_j khi và chỉ khi thao tác switch (chuyển) nằm trong phần từ (i, j) của ma trận quyền truy cập - nghĩa là switch $\in$ access(i. j). Trong bảng trên tiến trình ở miền D_2 có thể chuyển sang miền D_3 và D_4.

Ta định nghĩa thêm các thao tác: Sao chép (copy), Sở hữu (owner) và Kiểm soát (control). Đây là các thao tác thay đổi giá trị các phần tử trong ma trận quyền truy cập - tức là thay đổi quyền cho miền bảo vệ trên từng đối tượng. Ta có 2 bảng sau:

Bảng 5.3a. Ma trận trước sao chép quyền

Object / Domain	F_1	F_2	F_3
D_1	Execute		Write*
D_2	Execute	Read*	Execute
D_3	Execute		

Bảng 5.3b. Ma trận sau sao chép quyền

Object / Domain	F_1	F_2	F_3
	Execute		Write*
	Execute	Read*	Execute
	Execute	Read	

Trong bảng trên, Thao tác được đánh dấu (*) nghĩa là miền bảo vệ tương ứng (hàng trong ma trận) có thế sao chép thao tác đó sang các miền (hàng) khác. Quyền copy

cho phép sao chép quyền truy cập trong phạm vi một cột (nghĩa là cho cùng một đối tượng).

Ví dụ 5.2: trong bảng bên trái (bảng 5.3.a) tiến trình chạy trong miền D_2 có thể sao chép quyền read gắn với F_2 cho miền D_3. Khi đó, ma trận ở bảng bên phải (bảng 5.3.b) thể hiện sau khi áp dụng quyền sao chép giữa hai miền D_2 và D_3 trên F_2.

Các thao tác sửa đổi nội dung ma trận được phép thực hiện bao gồm : sao chép quyền (copy), chuyển quyền (transfer), quyền sở hữu (owner), và quyền kiểm soát (control).

Copy: nếu một quyền truy xuất *R* trong access[i,j] được đánh dấu là R^* thì có thể sao chép nó sang một phần tử access[k,j] khác.

Transfer: nếu một quyền truy xuất *R* trong access[i,j] được đánh dấu là R^+ thì có thể chuyển nó sang một phần tử access[k,j] khác.

Owner: nếu access[i,j] chứa quyền truy xuất owner thì tiến trình hoạt động trong miền bảo vệ D_i có thể thêm hoặc xóa các quyền truy xuất trong bất kỳ phần tử nào tren cột j.

Control: nếu access[i,j] chứa quyền truy xuất control thì tiến trình hoạt động trong miền bảo vệ D_i có thể xóa bất kỳ quyền truy xuất nào trong các phần tử trên dòng *j*.

Quyền truy cập (access controll)

Quyền truy cập đảm bảo rằng chỉ những người dùng được phép có thể truy cập vào dữ liệu và các chức năng cần thiết cho công việc của họ, và ngăn chặn những người không được phép khỏi việc truy cập vào các tài nguyên nhạy cảm hoặc thực hiện các hành động không hợp lệ.

Có ba loại cơ bản của quyền truy cập:

- Quyền truy cập hệ thống (System access): Đây là quyền truy cập chung vào hệ thống máy tính. Nó kiểm soát việc đăng nhập và đăng xuất, cũng như các quyền khác nhau của người dùng như người dùng thông thường, quản trị viên hệ thống và người quản lý.
- Quyền truy cập tài nguyên (Resource access): Đây là quyền truy cập vào các tài nguyên cụ thể trong hệ thống như tệp tin, thư mục, ổ đĩa, máy in, cổng kết

nối và các dịch vụ mạng. Người dùng được cấp quyền truy cập như đọc, ghi, thực thi hoặc quản lý tài nguyên này dựa trên vai trò và yêu cầu của họ.

- Quyền truy cập dịch vụ (Service access): Đây là quyền truy cập vào các dịch vụ và ứng dụng trong hệ thống. Ví dụ, một người dùng có thể có quyền truy cập vào dịch vụ lưu trữ đám mây hoặc quyền truy cập vào cơ sở dữ liệu.

Quyền truy cập	**Ma trận truy cập**
Quyền truy cập và ma trận truy cập đều liên quan đến việc quản lý quyền truy cập vào tài nguyên trong hệ thống máy tính, nhưng tiếp cận và phạm vi của chúng có sự khác biệt. Quyền truy cập là các quyền cụ thể được cấp cho người dùng hoặc nhóm người dùng để thực hiện các hoạt động cụ thể trên tài nguyên (file, thư mục, dịch vụ, ứng dụng, v.v.). Quyền truy cập được xác định bởi các hành động (action) mà người dùng có thể thực hiện, chẳng hạn như read, write, execute, delete, v.v. Đối với mỗi tài nguyên, mỗi người dùng hoặc nhóm người dùng sẽ được gán các quyền truy cập riêng biệt, điều này quyết định khả năng của họ trong việc truy cập và tương tác với tài nguyên đó.	Ma trận truy cập là một biểu đồ hai chiều (dạng bảng) thể hiện mối quan hệ giữa các đối tượng (người dùng) và các tài nguyên (resource) trong hệ thống máy tính. Trong ma trận truy cập, hàng đại diện cho các đối tượng và cột đại diện cho các tài nguyên. Mỗi phần tử trong ma trận đại diện cho các quyền truy cập mà đối tượng tương ứng có đối với tài nguyên tương ứng. Ma trận truy cập giúp trình bày một cách rõ ràng và cụ thể quyền truy cập của từng đối tượng đối với từng tài nguyên trong hệ thống.

Mối tương quan giữa quyền truy cập và ma trận truy cập nằm trong việc ma trận truy cập là công cụ hữu ích để biểu thị và quản lý quyền truy cập của người dùng vào tài nguyên. Nó cho phép quản trị viên hệ thống dễ dàng hiểu và điều chỉnh các quyền truy cập của người dùng và nhóm người dùng một cách hợp lý, giúp đảm bảo an toàn và bảo mật cho hệ thống. Việc sử dụng ma trận truy cập giúp quản lý quyền truy cập một

cách cụ thể và hạn chế các rủi ro về bảo mật mà người dùng không được phép truy cập vào các tài nguyên không phù hợp.

5.3. An toàn hệ thống

Bảo vệ hệ thống 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 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.

Hệ thống được gọi là an toàn nếu các tài nguyên được sử dụng đúng như quy ước trong mọi hoàn cảnh. Tuy nhiên, điều này hầu như không đạt được trong thực tế. Vì trong một hệ thống, ngoài các cơ chế, chính sách bảo mật ra thì người sử dụng hệ thống là con người. Điều này dẫn đến khả năng gây ra lỗi hệ thống từ những lỗi trong công việc của con người. Ngoài yếu tố con người, các tác hại từ môi trường bên ngoài như vấn đề mất điện, hỏa hoạn,.. cũng là những yếu tố có thể ảnh hưởng đến an toàn hệ thống. Do đó, để bảo đảm an toàn hệ thống, người ta cần thực hiện ở cả hai khía cạnh: vật lý (các vấn đề an toàn cho trang thiết bị trên hệ thống), con người (tuyển chọn và huấn luyện nhân viên để đảm bảo vận hành không sai sót trên hệ thống).

5.3.1. Xác thực danh tính

Để đảm bảo an toàn, hệ điều hành cần giải quyết tốt vấn đề chủ yếu là xác thực 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.

Cơ chế xác thực người dùng dựa trên sự kết hợp của ba yếu tố sau:

- Vật sở hữu của người dùng (khóa hoặc thẻ);
- Kiến thức mà người dùng biết (định danh, mật khẩu);

- Đặc trưng người dùng (dấu vân tay, võng mạc hay chữ ký).

Cách tiếp cận phổ biến nhất để giải quyết vấn đề là sử dụng password (mật khẩu) để xác thực 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 được dùng để bảo vệ từng đối tượng trong hệ thống, thậm chi cùng một đối tượng sẽ có các password khác nhau ứng với những quyền truy xuất khác nhau.

Password tuy được sử dụng đơn giản, tuy nhiên, password có thể bị lộ do nhiều nguyên nhân: bị đoán, vô tình để lộ, … Để tránh những vấn đề bị lộ password này, các hệ thống có những cách thức như:

- Bắt buộc phải đặt password với độ phức tạp cao, nhiều ký tự.
- Định kỳ bắt buộc phải thay đổi password.
- Password chỉ sử dụng một lần trong phiên làm việc, hết phiên làm việc thì password không còn hiệu lực (mật khẩu OTP hiện nay).

Để bảo đảm password được lưu trữ và không bị lộ, ngày nay vấn đề mã hóa password được thực hiện với rất nhiều giải thuật và công cụ tiên tiến. Việc mã hóa password là cách thức sử dụng các thuật toán để chuyển những password mà người dùng nhập vào thành những ký tự, văn bản có độ phức tạp cao, nhằm tránh bị lộ password. Tuy nhiên, việc mã hóa này vẫn có thể bị dò ra password trong một số trường hợp. Do đó, để bảo đảm an toàn trong hệ thống máy tính, ta phải kết hợp nhiều phương pháp để đảm bảo việc xác thực người dùng là chính xác, an toàn.

5.3.2. Những nguy hiểm từ chương trình

Trong thực tế, ta thường sử dụng các chương trình do người khác viết. Điều này hoàn toàn có thể dẫn đến trường hợp là những chương trình này có chưa mã độc và gây hại cho máy tính của chúng ta. Hai trường hợp điển hình là Trojan Horse và Trapdoor.

Trojan Horse là một dạng mã độc gắn kèm vào những chương trình khác, để khi người dùng thực hiện chương trình. Đoạn mã độc này sẽ lợi dụng những quyền truy cập của chương trình mà nó dính vào. Từ đó phát tán và gây hại cho máy tính. Thông thường có 2 loại trojan:

- Toàn bộ mã đều là Trojan (loại này dễ phân tích và phát hiện).
- Mã Trojan được cài đặt thêm vào chương trình gốc, bổ sung thêm một vài tính năng (có thể là backdoor hoặc rootkit). Loại này chủ yếu xuất hiện trong hệ thống sử dụng mã nguồn mở vì có thể dễ dàng xâm nhập vào các mã nguồn có sẵn.

Ví dụ 5.3: trong một login script, khi người dùng cung cấp user id, password để đăng nhập, ngoài việc chuyển những thông tin đó cho tiến trình thực hiện đăng nhập, Trojan lưu những thông tin đó lại để phục vụ những mục đích sau này. Người dùng thấy đăng nhập thành công nên không mấy nghi ngờ rằng mình đã bị mất thông tin cá nhân.

Trapdoor là "điểm vào" một module nhưng không được công bố. Trapdoor được người phát triển ứng dụng thêm vào trong quá trình phát triển phần mềm để thử nghiệm module phần mềm. Nguyên nhân là do các phần mềm ngày nay thường rất phức tạp, gồm nhiều module kết hợp với nhau. Mỗi module lại do một nhóm người phụ trách, nên trong quá trình phát triển, họ phải tự tạo ra điểm vào cho module của mình để kiểm nghiệm một số chức năng nào đó. Vì lý do an ninh, hầu hết Trapdoor sẽ bị xóa bỏ khi hoàn thành phần mềm. Có ba nguyên nhân chính dẫn tới việc còn tồn tại Trapdoor:

- Quên không xóa đi.
- Để lại với mục đích bảo trì.
- Cố tình để lại nhằm có những truy cập bất hợp pháp sau này.

Nhìn chung Trapdoor rất khó bị phát hiện, vì để phát hiện ra nó phải phân tích toàn bộ thành phần của hệ thống. Điều này là bất khả thi. Do đó, để đảm bảo an toàn trong việc sử dụng hệ thống máy tính, ta phải bảo đảm sử dụng các chương trình có nguồn gốc đảm bảo. Trên thực tế, ở rất nhiều công ty, việc nhân viên tự động download phần mềm và các file crack để bẻ khóa phần mềm là bị cấm hoàn toàn. Nhân viên sẽ đối diện với nhiều chế tài nếu cố tình thực hiện điều này. Điều này giúp giảm thiểu nguy cơ về vấn đề Trapdoor.

Ngoài hai vấn đề trên thì hiện nay, Virus máy tính cũng là một vấn đề phức tạp cho những người sử dụng máy tính. Virus máy tính (computer virus) là một loại phần mềm độc hại được thiết kế để tự nhân bản và lây lan từ máy tính này sang máy tính khác, gây hại đến hệ thống hoặc ảnh hưởng đến hoạt động của máy tính mà không được sự cho phép của người dùng.

Các tính chất chính của virus máy tính bao gồm:

- Tự nhân bản: Virus máy tính có khả năng sao chép và nhân bản chính nó bằng cách gắn bản sao của mình vào các tệp hoặc các phần mềm khác trong hệ thống, làm cho nó lây lan từ máy tính này sang máy tính khác qua các phương tiện truyền thông như đĩa mềm, email, mạng LAN, USB, v.v.
- Tấn công và lây nhiễm: Virus máy tính thường tấn công và lây nhiễm các tệp thực thi (executable files), dữ liệu hoặc phần mềm hệ thống, cho phép nó kích hoạt và thực thi khi người dùng khởi động hoặc chạy các tệp bị nhiễm.
- Gây hại: Virus máy tính có thể thực hiện các hành động độc hại như xóa hoặc làm hỏng dữ liệu, kiểm soát máy tính từ xa, gửi thông tin cá nhân đến người tấn công, làm chậm hoạc làm đứng máy tính, v.v.
- Ẩn danh: Đôi khi Virus máy tính sẽ ẩn nấp trong hệ thống hoặc thay đổi chính mình để tránh bị phát hiện bởi các phần mềm diệt virus hoặc biện pháp bảo mật.

Virus máy tính thường là một trong những loại Malware (phần mềm độc hại) phổ biến nhất và gây ra nhiều vấn đề về bảo mật và hiệu suất máy tính. Để đối phó với Virus máy tính, người dùng cần sử dụng phần mềm diệt Virus và thực hiện các biện pháp bảo mật phù hợp để ngăn chặn sự lây lan và gây hại của chúng.

Ngày nay, Virus máy tính có thể gây ra thiệt hại rất nhiều về mặt kinh tế, cũng như ảnh hưởng đến công việc, cuộc sống của người dùng, những tổ chức, công ty… Trường hợp Virus WannaCry là một ví dụ. Virus không chỉ ảnh hưởng đến các vấn đề về kinh tế, tiến độ công việc, mất mát dữ liệu, nó còn làm mất niềm tin và uy tín đối với nhiều tổ chức và hệ thống. Các tổ chức không đảm bảo an ninh mạng và không cập nhật hệ thống đều bị đặt dấu hỏi về khả năng bảo vệ dữ liệu của khách hàng và đối tác.

Nhìn chung, ngày nay, sau rất nhiều cuộc tấn công của Virus, tấn công mạng,… để lại những tác hại vô cùng nghiêm trọng cho cá nhân, tổ chức, doanh nghiệp,… Điều này góp phần tăng nhận thức về vấn đề phải giữ an toàn và bảo mật trên hệ thống máy tính, hệ thống mạng của các cá nhân, tổ chức và doanh nghiệp trên toàn thế giới. Các vấn đề về an toàn máy tính, an ninh mạng, an ninh thông tin,… đều được nghiên cứu kỹ lưỡng và quan tâm trên toàn thế giới.

5.4. Một số phương pháp tấn công máy tính

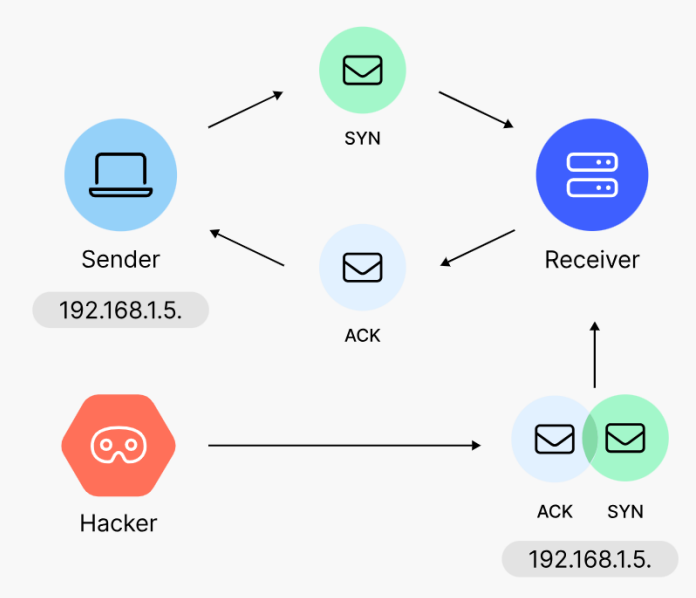
Ngày nay, các vấn đề về an ninh mạng, an toàn hệ thống, an toàn máy tính đều được quan tâm rất nhiều. Tội phạm mạng đã phát triển rất nhiều phương pháp tấn công máy tính, cũng như tấn công một hệ thống mạng. Việc tìm hiểu các phương pháp tấn công này rất quan trọng, vì chỉ khi ta hiểu rõ từng phương pháp thì mới có thể giúp chống lại các cuộc tấn công này một cách hữu hiệu. Sau đây là một số phương pháp tấn công khá phổ biến ngày nay.

Các phương pháp tấn công có thể chia thành một số kiểu như sau:

- Tấn công chủ động: Bao gồm các phương pháp tấn công từ chối dịch vụ, tấn công từ chối dịch vụ phân tán, Tấn công tràn bộ đệm, tấn công qua ký tự điều khiển đồng bộ SYN, giả mạo, người trung gian, giả mạo giao thức điều khiển truyền tin qua Internet, tự động kết nối đường truyền.
- Tấn công bị động: Bao gồm quét, bắt trộm và nghe trộm các gói tin.
- Tấn công mật khẩu: Bao gồm việc dự đoán, so sánh và tra mật khẩu thông qua một bộ từ điển mật khẩu.
- Tấn công mã nguồn: Bao gồm các phương pháp cửa sau (BackDoor), Virus, Trojans, Worms, các khóa mật mã yếu và thuật toán.

5.4.1. Tấn công chủ động

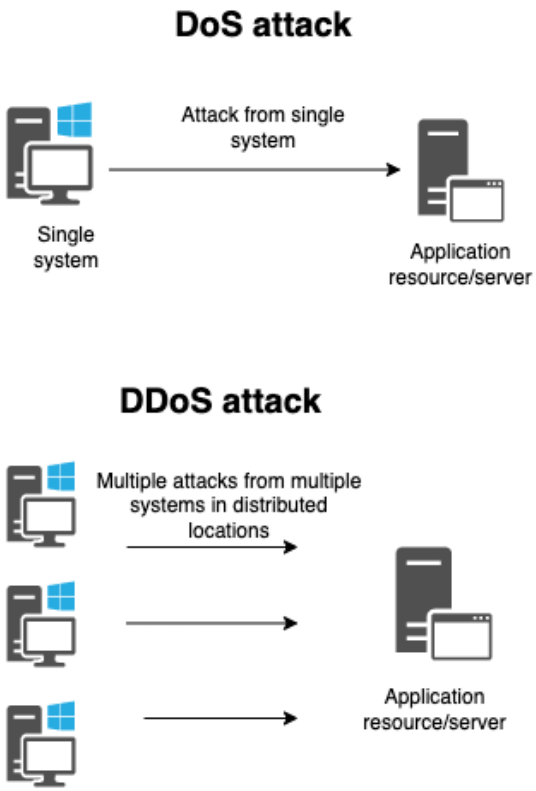
Tấn công giả mạo (Spoofing): là tình huống một người/một chương trình giả mạo thành công là một người khác hoặc một chương trình khác bằng cách làm sai lệch dữ liệu và do đó đạt được một lợi thế không chính đáng. Có nhiều loại tấn công giả mạo như: giả mạo IP, giả mạo MAC, giả mạo địa chỉ email,…



Hình 5.2. Tấn công giả mạo địa chỉ IP

Tấn công từ chối dịch vụ (DoS – Denial of Service): là một loại tấn công mạng nhằm làm cho dịch vụ hoặc tài nguyên mạng trở nên không khả dụng cho người dùng hợp lệ. Mục tiêu của tấn công DoS là làm cho hệ thống bị quá tải, không thể hoạt động đúng cách hoặc tài nguyên sẵn có không thể truy cập được. Khi xảy ra tấn công DoS, người dùng hợp lệ sẽ gặp khó khăn hoặc không thể truy cập được vào tài nguyên, dịch vụ hoặc trang web bị tấn công. Có rất nhiều kiểu tấn công DoS được thực triển khai như: Ping of Death, SYN Flooding,… Đây là kiểu tấn công đã xuất hiện từ rất lâu, tuy nhiên, nó lại là kiểu tấn công khá hiệu quả và gây nhiều hậu quả nghiêm trọng.

Tấn công DDoS (Distributed Denial of Service) là một biến thể phổ biến của tấn công DoS, trong đó tấn công được thực hiện từ nhiều máy tính khác nhau, đồng thời hoạt động như một lực lượng đồng thuận, làm cho tấn công trở nên khó phát hiện và khó ngăn chặn.

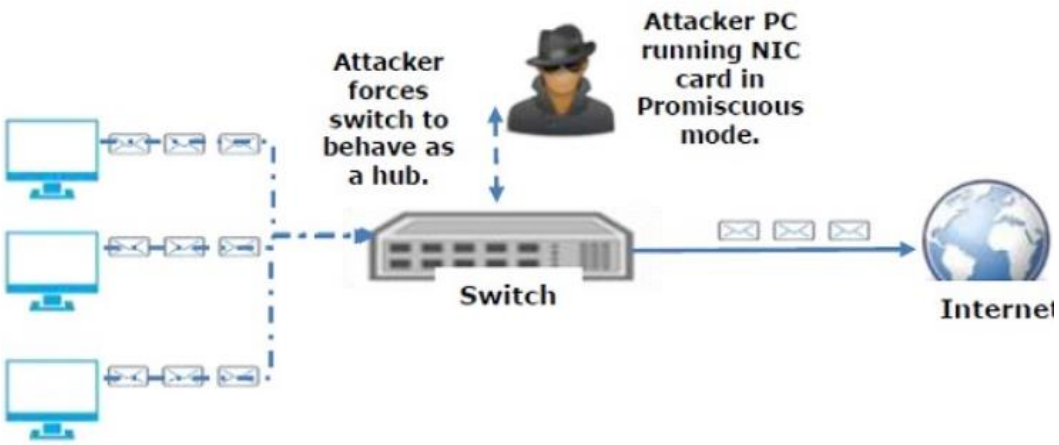


Hình 5.3. Tấn công DoS và DDoS

5.4.2. Tấn công thụ động

Tấn công nghe lén (Sniffing): là một loại tấn công mạng nhằm giám sát và thu thập thông tin truyền qua mạng một cách bí mật và trái phép. Kỹ thuật này thường được sử dụng để ghi lại các gói dữ liệu đang truyền qua mạng mà không cần phải can thiệp vào dữ liệu đó. Mục tiêu của tấn công nghe lén là thu thập thông tin nhạy cảm như tên đăng nhập, mật khẩu, thông tin tài khoản ngân hàng, hay bất kỳ dữ liệu quan trọng nào mà người dùng gửi qua mạng.

Các mạng không được bảo mật mạnh hoặc sử dụng giao thức truyền dữ liệu không được mã hóa có thể trở thành mục tiêu dễ dàng cho tấn công nghe lén. Một khi người thực hiện tấn công đã có được thông tin quan trọng từ việc nghe lén, họ có thể sử dụng thông tin này để thực hiện các cuộc tấn công tiếp theo, chẳng hạn như tấn công hệ thống hoặc truy cập trái phép vào tài khoản của người dùng.



Hình 5.4. Tấn công Sniffing

Trong hình trên, người tấn công (Attacker PC) sẽ lắng nghe các gói tin từ các máy tính nạn nhân. Sau đó thu thập và lấy thông tin một cách bất hợp pháp.

Phần tấn công mật khẩu và tấn công mã độc được đề cập ở phần trên (Phần: 1. xác thực và 2. Những nguy hiểm từ chương trình)

Một cách tổng quát, các phương pháp tấn công ngày càng được phát triển và tinh vi hơn. Hậu quả của những cuộc tấn công đem đến những thất thoát rất nghiêm trọng. Do đó, người dùng máy tính phải tuân thủ theo những khuyến cáo bảo mật của các cơ quan an ninh mạng. Cũng như luôn liên tục cập nhật kiến thức của mình về vấn đề bảo mật để hạn chế những nguy hiểm mà các mối hiểm họa có thể gây ra.

Bài tập chương 5

Bài tập 5.1.

Hãy trình bày các vấn đề về bảo vệ và an toàn nổi bật nhất đối với hệ điều hành trong thời điểm hiện tại .

Bài tập 5.2.

Hãy khảo sát và so sánh các kĩ thuật bảo vệ máy tính trên môi trường internet.

Bài tập 5.3.

Hãy tìm hiểu các nhà cung cấp về điện toán đám mây hiện này ở Việt Nam. Tìm hiểu nguyên lý vận hành và tính an toàn khi sử dụng các dịch vụ của những nhà cung cấp dịch vụ điện toán đám mây.

Bài tập 5.4.

Hãy liệt kê các loại Virus máy tính trong giai đoạn hiện nay và cách phòng chống.

Bài tập 5.5.

Trình bày các công nghệ tạo ra máy ảo, máy chủ ảo hiện nay. So sánh các công nghệ này.

Bài tập 5.6.

Trình bày các tiêu chuẩn của trung tâm dữ liệu (data center). Hiện nay, có những trung tâm dữ liệu nào hoạt động ở Việt Nam. Trình bày các vấn đề bảo mật mà các trung tâm dữ liệu thông thường phải đối mặt.

Bài tập 5.7.

Trình bày nguyên lý thiết kế một hệ điều hành cơ bản. So sánh những điểm ưu và nhược của hệ điều hành Windows và Ubuntu (hoặc một hệ điều hành mã nguồn mở khác).

Tài liệu tham khảo

[1]. Trần Hạnh Nhi, Lê Khắc Nhiên Ân,"*Hệ điều hành*", Trường Đại học khoa học tự nhiên, Đại học Quốc gia Thành phố Hồ Chí Minh, 2003.

[2]. Trần Hạnh Nhi, "*Hệ điều hành nâng cao*", Trường Đại học khoa học tự nhiên, Đại học Quốc gia Thành phố Hồ Chí Minh, 2005.

[3]. Abraham Silberschatz, Peter Baer Galvin, Greg Gagne (2018), Operating System Concepts Essentials 9th Edition, publisher Wiley.

[4]. Alan Clements (2018), Principles of Computer Hardware, publisher Oxford University Press.

[5]. Andrew S. Tanenbaum, Herbert Bos (2014), Modern Operating Systems 4th Edition, publisher Pearson Education.

[6]. Andrew S. Tanenbaum, Albert S. Woodhull (2014), Operating Systems: Design and Implementation 3rd Edition, publisher Pearson.

[7]. Dhananjay M. Dhamdhere (2010), Operating Systems: A Concept-Based Approach, publisher McGraw-Hill.

[8]. Gary Nutt (2014), Operating Systems: A Modern Perspective 3rd Edition, publisher Pearson.

[9]. J. Stanley Warford (2016), Computer Systems: An Integrated Approach to Architecture and Operating Systems 2nd Edition, publisher Pearson.

[10]. Remzi H. Arpaci-Dusseau, Andrea C. Arpaci-Dusseau (2015), Operating Systems: Three Easy Pieces 1st Edition, publisher Arpaci-Dusseau Books.

Phụ lục

ĐỀ SỐ 1

(Thời gian 90 phút, sinh viên không sử dụng tài liệu)

Câu 1. (2,0 điểm)

Cho 4 tiến trình P_1, P_2, P_3, P_4 có thời điểm vào (arrival time) lần lượt là 0, 2, 4, 5 và thời gian xử lý của CPU (burst time) lần lượt là 8, 4, 1, 5 (đơn vị milliseconds). Hãy vẽ biểu đồ Gantt (Gantt chart) để tìm thứ tự cấp phát CPU cho các tiến trình và cho biết thời gian chờ trung bình (average waiting time) của các tiến trình theo chiến lược điều phối "*công việc ngắn nhất được thực hiện trước*" với hai dạng sau:

a. Shortest job first scheduling (SJF); SJF làm việc theo nguyên tắc độc quyền.

b. Shortest remaining time first scheduling (SRTF); SRTF làm việc theo nguyên tắc không độc quyền.

Câu 2. (2,0 điểm)

Một hệ thống có 5 tiến trình P_0, P_1, P_2, P_3, P_4 và 3 loại tài nguyên *A*, *B*, *C*; trong đó tài nguyên *A* có 10 thể hiện (instances), tài nguyên *B* có 5 thể hiện và tài nguyên *C* có 7 thể hiện.

Giả sử trạng thái của hệ thống tại thời điểm T_0 như sau:

Process	Allocation			Max			Available		
	A	*B*	*C*	*A*	*B*	*C*	*A*	*B*	*C*
P_0	0	1	0	7	5	3	3	3	2
P_1	2	0	0	3	2	2			
P_2	3	0	2	9	0	2			
P_3	2	1	1	2	2	2			
P_4	0	0	2	4	3	3			

Sử dụng thuật toán *Banker*

a. Hãy tìm bảng Need.

b. Hãy xác định xem hệ thống tại thời điểm T_0 có ở trạng thái an toàn hay không ?

c. Hãy xác định xem có nên đáp ứng yêu cầu cấp phát (0, 2, 0) của tiến trình P_0 hay không ?

Câu 3. (2,0 điểm)

Cho 6 phân vùng bộ nhớ có kích thước lần lượt là 300 KB, 600 KB, 350 KB, 200 KB, 750 KB, 125 KB và 6 tiến trình có kích thước lần lượt là 115 KB, 500 KB, 358 KB, 200 KB, 375 KB, 320 KB.

Hãy cấp phát bộ nhớ cho các tiến trình trên lần lượt theo các thuật toán first-fit, best-fit, và worst-fit; từ đó hãy cho biết thuật toán nào cấp phát bộ nhớ hiệu quả nhất trong trường hợp trên ?

Câu 4. (2,0 điểm)

Cho chuỗi truy xuất (reference string) bộ nhớ sau:

1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6. Giả sử sử dụng 4 khung trang và ban đầu các khung trang đều trống. Hãy tìm số lỗi trang (page faults) xảy ra khi sử dụng các thuật toán thay thế trang sau:

a. Thuật toán thay thế trang FIFO (first in first out page replacement algorithm).

b. Thuật toán thay thế trang LRU (least recently used page replacement algorithm).

Câu 5. (2,0 điểm)

Giả sử đầu đọc đang ở vị trí cylinder 53, cần đọc các cylinder: 98, 183, 37, 122, 14, 124, 65, 67 (xét trong miền 0..199). Hãy liệt kê lần lượt các cylinder mà đầu đọc đi qua khi sử dụng thuật toán lập lịch đĩa (disk scheduling algorithm) shortest seek time first algorithm (SSTF); từ đó hãy tính tổng khoảng cách di chuyển của đầu đọc.

Hết

ĐỀ SỐ 2

(Thời gian 90 phút, sinh viên không sử dụng tài liệu)

Câu 1. (2,0 điểm)

Cho 4 tiến trình P_1, P_2, P_3, P_4 có thời điểm vào (arrival time) lần lượt là 0, 1, 2, 3 và thời gian xử lý của CPU (burst time) lần lượt là 6, 8, 5, 2 (đơn vị milliseconds). Hãy vẽ biểu đồ Gantt (Gantt chart) để tìm thứ tự cấp phát CPU cho các tiến trình và cho biết thời gian chờ trung bình (average waiting time) của các tiến trình theo chiến lược điều phối "*công việc ngắn nhất được thực hiện trước*" với hai dạng sau:

a. Shortest job first scheduling (SJF); SJF làm việc theo nguyên tắc độc quyền.

b. Shortest remaining time first scheduling (SRTF); SRTF làm việc theo nguyên tắc không độc quyền.

Câu 2. (2,0 điểm)

Một hệ thống có 5 tiến trình P_0, P_1, P_2, P_3, P_4 và 4 loại tài nguyên *A*, *B*, *C*, *D*; trong đó tài nguyên *A* có 3 thể hiện (instances), tài nguyên *B* có 14 thể hiện, tài nguyên *C* có 12 thể hiện và tài nguyên *D* có 12 thể hiện.

Giả sử trạng thái của hệ thống tại thời điểm T_0 như sau:

Process	Allocation				Max				Available			
	A	*B*	*C*	*D*	*A*	*B*	*C*	*D*	*A*	*B*	*C*	*D*
P_0	0	0	1	2	0	0	1	2	1	5	2	0
P_1	1	0	0	0	1	7	5	0				
P_2	1	3	5	4	2	3	5	6				
P_3	0	6	3	2	0	6	5	2				
P_4	0	0	1	4	0	6	5	6				

Sử dụng thuật toán *Banker*.

a. Hãy tìm bảng Need.

b. Hãy xác định xem hệ thống tại thời điểm T_0 có ở trạng thái an toàn hay không ?

c. Hãy xác định xem có nên đáp ứng yêu cầu cấp phát (0, 4, 3, 0) của tiến trình P_1 hay không ?

d. Hãy xác định xem có nên đáp ứng yêu cầu cấp phát (0, 4, 2, 0) của tiến trình P_4 hay không ?

Câu 3. (2,0 điểm)

Cho 4 phân vùng bộ nhớ có kích thước lần lượt là 600 KB, 500 KB, 200 KB, 300 KB và 4 tiến trình có kích thước lần lượt là 212 KB, 417 KB, 112 KB, 426KB.

Hãy cấp phát bộ nhớ cho các tiến trình trên lần lượt theo các thuật toán first-fit, best-fit, và worst-fit; từ đó hãy cho biết thuật toán nào cấp phát bộ nhớ hiệu quả nhất trong trường hợp trên ?

Câu 4. (2,0 điểm)

Cho chuỗi truy xuất (reference string) bộ nhớ sau:

7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1. Giả sử sử dụng 3 khung trang và ban đầu các khung trang đều trống. Hãy tìm số lỗi trang (page faults) xảy ra khi sử dụng các thuật toán thay thế trang sau:

a. Thuật toán thay thế trang FIFO (first in first out page replacement algorithm).

b. Thuật toán thay thế trang LRU (least recently used page replacement algorithm).

Câu 5. (2,0 điểm)

Giả sử đầu đọc đang ở vị trí cylinder 53, cần đọc các cylinder: 88, 198, 37, 122, 14, 124, 65, 67 (xét trong miền 0..199). Hãy liệt kê lần lượt các cylinder mà đầu đọc đi qua khi sử dụng thuật toán lập lịch đĩa (disk scheduling algorithm) shortest seek time first algorithm (SSTF); từ đó hãy tính tổng khoảng cách di chuyển của đầu đọc.

Hết

ĐỀ SỐ 3

(Thời gian 90 phút, sinh viên không sử dụng tài liệu)

Câu 1. (2 điểm)

Hãy nêu ý tưởng cơ bản của giải thuật điều phối CPU Round Robin?

Cho tập các tiến trình như sau:

Tiến trình	Thời điểm vào Ready list	Thời gian CPU lần 1	IO lần 1		Thời gian CPU lần 2	IO lần 2	
			Thời gian	Thiết bị		Thời gian	Thiết bị
P1	0	3	1	R1	1	0	Null
P2	2	5	4	R2	3	2	R1
P3	5	2	5	R2	4	3	R2
P4	6	6	3	R1	4	0	Null

Lưu ý rằng: mỗi loại thiết bị IO chỉ có duy nhất 1 thể hiện. Trong mỗi chu kỳ IO, mỗi tiến trình yêu cầu 1 thể hiện duy nhất của mỗi loại thiết bị.

Hãy vẽ sơ đồ (hoặc bảng) điều phối CPU sử dụng chiến lược Round Robin, với q=4.

Hãy tính thời gian chờ của mỗi tiến trình và thời gian chờ trung bình của tất cả tiến trình trên.

Câu 2. (2 điểm)

Một hệ thống có 3 loại tài nguyên (A, B, C) và 5 tiến trình (P1, P2, P3, P4, P5) kèm theo các thông số được mô tả trong bảng sau:

	Allocation			Max			Available		
	A	B	C	A	B	C	A	B	C
P1	2	0	1	3	4	3	2	3	4
P2	1	0	0	3	7	5			
P3	0	1	0	2	3	2			
P4	2	3	0	2	6	0			
P5	1	1	1	2	2	2			

Hãy cho biết tổng số tài nguyên mỗi loại của hệ thống.

Tiến trình trên đang ở trong một trạng thái an toàn (safe state). Nếu tiến trình P3 yêu cầu thêm tài nguyên (2,1,0), hãy áp dụng giải thuật Banker để xét xem có nên cấp phát cho P3 hay không ?

Câu 3. (2 điểm)

Một hệ thống có bộ nhớ chính kích thước 512 MB. Hệ thống sử dụng địa chỉ logic 64 bit. Kích thước của 1 page được sử dụng là 8KB. Hãy xác định các thông số sau:

- Cho biết 1 frame có kích thước là bao nhiêu? Tại sao?
- Tổng số pages mà hệ thống có thể cấp phát?
- Tổng số frames trong bộ nhớ chính?
- Cho địa chỉ logic: 202212. Hãy đổi địa chỉ này sang dạng <p,d>

Câu 4. (2 điểm)

Hãy mô tả ý tưởng thuật toán thay thế trang FIFO trong kỹ thuật quản lý bộ nhớ ảo.

Trong bộ nhớ ảo, tiến trình P đang chạy được chia làm 8 pages (0, 1, ….., 7), hệ thống cấp cho tiến trình P 3 frame để thực hiện. Ta có dãy truy cập đến page như sau:

7 0 1 2 3 4 5 3 5 1 0 6 2 1 4 7 2 1 0 3 2

Hãy lập bảng cấp phát và thay thế khung trang đối với dãy truy cập page như trên, sử dụng thuật toán thay thế trang FIFO.

Hãy đếm số page faults trong tình huống trên.

Câu 5 (2 điểm)

Cho đoạn code như sau:

```
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>
void* inchan(void* arg)
{
        int i;
        for(i=2;i<13;i=i+2)
        {
                printf("thread 1: %d \n",i);
```

```
        }
 }
 void* inle(void* arg)
{ int i;
  for(i=1;i<13;i=i+2)
        {
                printf("thread 2: %d \n",i);
        }
}
int main()
{
   pthread_t t1,t2;
   pthread_create(&t2,NULL,inle,NULL);
   pthread_create(&t1,NULL,inchan,NULL);
   pthread_join(t1,NULL);
   pthread_join(t2,NULL);
   return 0;
}
```

Hãy thiết kế Semaphore để các tiểu trình (thread) phối hợp với nhau để xuất đúng thứ tự của số. Viết lại toàn bộ chương trình trên với semaphore đã thiết kế theo yêu cầu ở trên.

Hết

ĐỀ SỐ 4

(Thời gian 90 phút, sinh viên không sử dụng tài liệu)

Câu 1. (2 điểm)

Hãy nêu ý tưởng cơ bản của giải thuật điều phối CPU Shortest Job First (SJF – no pre-emptive) ?

Cho tập các tiến trình như sau:

Tiến trình	Thời điểm vào Ready list	Thời gian CPU lần 1	IO lần 1		Thời gian CPU lần 2	IO lần 2	
			Thời gian	Thiết bị		Thời gian	Thiết bị
P1	0	3	1	R1	1	0	Null
P2	2	5	2	R2	3	2	R1
P3	2	2	5	R2	4	3	R2
P4	4	4	3	R1	2	3	R1

Lưu ý rằng: mỗi loại thiết bị IO chỉ có duy nhất 1 thể hiện. Trong mỗi chu kỳ IO, mỗi tiến trình yêu cầu 1 thể hiện duy nhất của mỗi loại thiết bị.

Hãy vẽ sơ đồ (hoặc bảng) điều phối CPU sử dụng chiến lược SJF.

Hãy tính thời gian chờ của mỗi tiến trình và thời gian chờ trung bình của tất cả tiến trình trên.

Câu 2. (2 điểm)

Một hệ thống có 3 loại tài nguyên (A, B, C) và 5 tiến trình (P1, P2, P3, P4, P5) kèm theo các thông số được mô tả trong bảng sau:

	Allocation			Max			Available		
	A	B	C	A	B	C	A	B	C
P1	2	0	1	3	4	3	2	3	4
P2	1	0	0	3	7	5			
P3	0	1	0	2	3	2			
P4	2	3	0	2	6	0			
P5	1	1	1	2	2	2			

Hãy cho biết các tiến trình trong bảng trên có ở trong trạng thái an toàn (safe state) không? Tại sao ?

Nếu tiến trình P2 yêu cầu thêm tài nguyên (2,2,1), hãy áp dụng giải thuật Banker để xét xem có nên cấp phát cho P2 hay không?

Câu 3. (2 điểm)

Một hệ thống có bộ nhớ chính kích thước 512 MB. Hệ thống sử dụng địa chỉ logic 32 bit. Kích thước của 1 page được sử dụng là 4KB. Hãy xác định các thông số sau:

- Cho biết 1 frame có kích thước là bao nhiêu? Tại sao?
- Số bit làm offset?
- Tổng số frames trong bộ nhớ chính?
- Cho địa chỉ logic: 202212. Hãy đổi địa chỉ này sang dạng <p,d>

Câu 4. (2 điểm)

Hãy mô tả ý tưởng thuật toán thay thế trang FIFO trong kỹ thuật quản lý bộ nhớ ảo.

Trong bộ nhớ ảo, tiến trình P đang chạy được chia làm 8 pages (0, 1, ….., 7), hệ thống cấp cho tiến trình P 4 frame để thực hiện. Ta có dãy truy cập đến page như sau:

7 0 1 2 0 4 2 3 5 1 5 6 2 1 4 7 2 1 0 3 2

Hãy lập bảng cấp phát và thay thế khung trang đối với dãy truy cập page như trên, sử dụng thuật toán thay thế trang FIFO.

Hãy đếm số page faults trong tình huống trên.

Câu 5 (2 điểm)

Cho đoạn code như sau:

```
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>
void* thread(void* arg)
{
        printf("\nEntered..\n");
        sleep(4);
        printf("\nJust Exiting...\n");
}
```

```
int main()
{
        pthread_t t1,t2;
        pthread_create(&t1,NULL,thread,NULL);
        pthread_create(&t2,NULL,thread,NULL);
        pthread_join(t1,NULL);
        pthread_join(t2,NULL);
        return 0;
}
```

Chương trình trên khi chạy sẽ cho ra kết quả như thế nào? Tại sao ?

Nếu muốn chương trình trên chạy ra kết quả như sau:

```
Enter..
Just Exiting..
Enter..
Just Exiting..
```

Hãy thiết kế semaphore và viết lại toàn bộ chương trình để có thể chạy ra kết quả như mô tả ở trên.

Hết

ĐỀ SỐ 5

(Thời gian 90 phút, sinh viên không sử dụng tài liệu)

Câu 1. (2,0 điểm)

Cho 4 tiến trình P_1, P_2, P_3, P_4 có thời điểm vào (arrival time) lần lượt là 0, 2, 4, 5 và thời gian xử lý của CPU (burst time) lần lượt là 7, 4, 1, 6 (đơn vị milliseconds). Hãy vẽ biểu đồ Gantt (Gantt chart) để tìm thứ tự cấp phát CPU cho các tiến trình và cho biết thời gian chờ trung bình (average waiting time) của các tiến trình theo chiến lược điều phối "*công việc ngắn nhất được thực hiện trước*" với hai dạng sau:

a. Shortest job first scheduling (SJF); SJF làm việc theo nguyên tắc độc quyền.

b. Shortest remaining time first scheduling (SRTF); SRTF làm việc theo nguyên tắc không độc quyền.

Câu 2. (2,0 điểm)

Một hệ thống có 5 tiến trình P_0, P_1, P_2, P_3, P_4 và 3 loại tài nguyên *A*, *B*, *C*; trong đó tài nguyên *A* có 10 thể hiện (instances), tài nguyên *B* có 5 thể hiện và tài nguyên *C* có 7 thể hiện.

Giả sử trạng thái của hệ thống tại thời điểm T_0 như sau:

Process	Allocation			Max			Available		
	A	*B*	*C*	*A*	*B*	*C*	*A*	*B*	*C*
P_0	0	1	0	7	5	3	3	3	2
P_1	2	0	0	3	2	2			
P_2	3	0	2	9	0	2			
P_3	2	1	1	2	2	2			
P_4	0	0	2	4	3	3			

Sử dụng thuật toán *Banker* hãy thực hiện các công việc sau:

a. Tìm bảng Need.

b. Xác định xem hệ thống tại thời điểm T_0 có ở trạng thái an toàn hay không ?

c. Xác định xem có nên đáp ứng yêu cầu cấp phát (0, 2, 0) của tiến trình P_0 hay không ?

Câu 3. (2,0 điểm)

Cho 4 phân vùng bộ nhớ có thứ tự và kích thước lần lượt là 450 KB, 650 KB, 325 KB, 185 KB và 7 tiến trình có thứ tự và kích thước lần lượt là 180 KB, 460 KB, 440 KB, 300 KB, 130 KB, 56 KB, 20 KB.

Hãy cấp phát bộ nhớ cho các tiến trình trên lần lượt theo các thuật toán first-fit, best-fit, và worst-fit; từ đó hãy cho biết thuật toán nào cấp phát bộ nhớ hiệu quả nhất trong trường hợp trên ?

Câu 4. (2,0 điểm)

Cho chuỗi truy xuất (reference string) bộ nhớ sau:

7, 2, 3, 1, 2, 5, 3, 4, 6, 7, 7, 1, 0, 5, 4, 6, 2, 3, 0, 1. Giả sử sử dụng 3 khung trang và ban đầu các khung trang đều trống. Hãy tìm số lỗi trang (page faults) xảy ra khi sử dụng các thuật toán thay thế trang sau:

a. Thuật toán thay thế trang FIFO (first in first out page replacement algorithm).

b. Thuật toán thay thế trang LRU (least recently used page replacement algorithm).

Câu 5. (2,0 điểm)

Giả sử đầu đọc đang ở vị trí cylinder 70, cần đọc các cylinder: 72, 75, 97, 80, 189, 65, 168, 14, 30 (xét trong miền 0..199). Hãy liệt kê lần lượt các cylinder mà đầu đọc đi qua khi sử dụng các thuật toán lập lịch đĩa (disk scheduling algorithm) sau đây; từ đó hãy tính tổng khoảng cách di chuyển của đầu đọc ứng với mỗi thuật toán.

a. Thuật toán lập lịch đĩa SSTF (shortest seek time first algorithm).

b. Thuật toán lập lịch đĩa SCAN (cần giải quyết cho cả hai trường hợp: đầu đọc di chuyển về hướng bên trái, đầu đọc di chuyển về hướng bên phải).

Hết

ĐỀ SỐ 6

(Thời gian 90 phút, sinh viên không sử dụng tài liệu)

Câu 1. (2,0 điểm)

Cho 4 tiến trình P_1, P_2, P_3, P_4 có thời điểm vào (arrival time) lần lượt là 0, 2, 4, 5 và thời gian xử lý của CPU (burst time) lần lượt là 8, 4, 1, 5 (đơn vị milliseconds). Hãy vẽ biểu đồ Gantt (Gantt chart) để tìm thứ tự cấp phát CPU cho các tiến trình và cho biết thời gian chờ trung bình (average waiting time) của các tiến trình theo chiến lược điều phối "*công việc ngắn nhất được thực hiện trước*" với hai dạng sau:

a. Shortest job first scheduling (SJF); SJF làm việc theo nguyên tắc độc quyền.

b. Shortest remaining time first scheduling (SRTF); SRTF làm việc theo nguyên tắc không độc quyền.

Câu 2. (2,0 điểm)

Một hệ thống có 4 tiến trình P_0, P_1, P_2, P_3 và 3 loại tài nguyên *A*, *B*, *C*; trong đó tài nguyên *A* có 9 thể hiện (instances), tài nguyên *B* có 3 thể hiện, tài nguyên *C* có 6 thể hiện.

Giả sử trạng thái của hệ thống tại thời điểm T_0 như sau:

Process	Allocation			Max			Available		
	A	*B*	*C*	*A*	*B*	*C*	*A*	*B*	*C*
P_0	1	0	0	3	2	2	4	1	2
P_1	2	1	1	6	1	3			
P_2	2	1	1	3	1	4			
P_3	0	0	2	4	2	2			

Sử dụng thuật toán *Banker* hãy thực hiện các công việc sau:

a. Tìm bảng Need.

b. Xác định xem hệ thống tại thời điểm T_0 có ở trạng thái an toàn hay không ?

c. Xác định xem có nên đáp ứng yêu cầu cấp phát (4, 0, 1) của tiến trình P_1 hay không ?

Câu 3. (2,0 điểm)

Cho 6 phân vùng bộ nhớ có thứ tự và kích thước lần lượt là 300 KB, 600 KB, 350 KB, 200 KB, 750 KB, 125 KB và 6 tiến trình có thứ tự và kích thước lần lượt là 115 KB, 500 KB, 358 KB, 200 KB, 375 KB, 320 KB.

Hãy cấp phát bộ nhớ cho các tiến trình trên lần lượt theo các thuật toán first-fit, best-fit, và worst-fit; từ đó hãy cho biết thuật toán nào cấp phát bộ nhớ hiệu quả nhất trong trường hợp trên ?

Câu 4. (2,0 điểm)

Cho chuỗi truy xuất (reference string) bộ nhớ sau:

1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6. Giả sử sử dụng 4 khung trang và ban đầu các khung trang đều trống. Hãy tìm số lỗi trang (page faults) xảy ra khi sử dụng các thuật toán thay thế trang sau:

a. Thuật toán thay thế trang FIFO (first in first out page replacement algorithm).

b. Thuật toán thay thế trang LRU (least recently used page replacement algorithm).

Câu 5. (2,0 điểm)

Giả sử đầu đọc đang ở vị trí cylinder 53, cần đọc các cylinder: 77, 182, 37, 122, 15, 124, 65, 67 (xét trong miền 0..199). Hãy liệt kê lần lượt các cylinder mà đầu đọc đi qua khi sử dụng các thuật toán lập lịch đĩa (disk scheduling algorithm) sau đây; từ đó hãy tính tổng khoảng cách di chuyển của đầu đọc ứng với mỗi thuật toán.

a. Thuật toán lập lịch đĩa SSTF (shortest seek time first algorithm).

b. Thuật toán lập lịch đĩa SCAN (cần giải quyết cho cả hai trường hợp: đầu đọc di chuyển về hướng bên trái, đầu đọc di chuyển về hướng bên phải).

Hết