

BỘ GIÁO DỤC VÀ ĐÀO TẠO
ĐẠI HỌC CÔNG NGHỆ TP.HCM

KIẾN TRÚC MÁY TÍNH
HỆ ĐIỀU HÀNH

Biên soạn:

ThS. Nguyễn Quang Anh

ThS. Hàn Minh Châu

ThS. Nguyễn Lê Văn

ThS. Nguyễn Anh Vinh

www.hutech.edu.vn

KIẾN TRÚC MÁY TÍNH – HỆ ĐIỀU HÀNH

Ấn bản 2018

Các ý kiến đóng góp về tài liệu học tập này, xin gửi về e-mail của ban biên tập :
tailieuhoctap@hutech.edu.vn

MỤC LỤC

MỤC LỤC	I
HƯỚNG DẪN	VII
BÀI 1. CÁC HỆ CƠ SỐ ĐẾM	1
 1.1 GIỚI THIỆU.....	1
1.1.1 Hệ nhị phân	1
1.1.2 Hệ thập phân	1
1.1.3 Hệ bát phân và thập lục phân	2
 1.2 CÁCH CHUYỂN ĐỔI.....	3
 1.3 DẤU CHẨM TĨNH	4
 1.4 SỐ ÂM	4
 1.5 KHOẢNG GIÁ TRỊ BIỂU DIỄN.....	5
 CÂU HỎI VÀ BÀI TẬP.....	5
BÀI 2. TÍNH TOÁN NHỊ PHÂN.....	8
 2.1 CÁC PHÉP TÍNH TOÁN TRÊN NHỊ PHÂN	8
2.1.1 Phép cộng, trừ.....	8
2.1.2 Phép nhân, chia	9
 2.2 DẤU CHẨM ĐỘNG	10
 2.3 CÁC MÃ KHÁC.....	11
 CÂU HỎI VÀ BÀI TẬP.....	12
BÀI 3. ĐẠI SỐ BOOLE VÀ MẠCH SỐ	14
 3.1 GIỚI THIỆU.....	14
3.1.1 Đại số Boolean và các cổng logic	14
3.1.2 Quy trình chế tạo	15
3.1.3 Biểu thức logic.....	17
 3.2 RÚT GỌN BIỂU THỨC LOGIC	18
3.2.1 Giản đồ Karnaugh	19
3.2.2 Quine McCluskey.....	21
 CÂU HỎI VÀ BÀI TẬP.....	22
BÀI 4. CÁC MẠCH SỐ CƠ BẢN	26
 4.1 MẠCH TÍNH TOÁN SỐ HỌC.....	26
4.1.1 Mạch cộng/trừ	26
4.1.2 Mạch nhân	28
 4.2 MẠCH SO SÁNH, ĐA HỢP/GIẢI ĐA HỢP.....	30

4.2.1 Mạch so sánh.....	30
4.2.2 Mạch đa hợp/giải đa hợp	31
4.3 ALU (ARITHMETIC LOGIC UNIT)	32
CÂU HỎI VÀ BÀI TẬP	33
BÀI 5. TỔ CHỨC MÁY TÍNH	34
5.1 BỘ VI XỬ LÝ - CPU	34
5.1.1 Bộ xử lý luận lý số học - ALU	35
5.1.2 Bộ điều khiển - CU	35
5.1.3 Thanh ghi - Register	35
5.1.4 Lịch sử.....	36
5.2 GIAO TIẾP GIỮA CPU VÀ NGOẠI VI.....	38
5.2.1 Phân loại bus	38
5.2.2 Mô hình tổng quát	39
5.3 CẤU TRÚC CỦA VI XỬ LÝ 8088/ 8086.....	40
5.3.1 Các đặc điểm chính.....	40
5.4 CÁCH MÃ HÓA LỆNH CỦA 8088/ 8086.....	45
CÂU HỎI VÀ BÀI TẬP	47
BÀI 6. GIỚI THIỆU HỢP NGỮ	48
6.1 HỢP NGỮ VÀ TRÌNH BIÊN DỊCH.....	48
6.2 CHÚ THÍCH	49
6.3 TỪ KHÓA	49
6.4 THỂ HIỆN GIÁ TRỊ	49
6.5 MÃ ASCII.....	49
6.6 SEGMENT VÀ ĐỊNH ĐỊA CHỈ.....	50
6.7 CÁC THANH GHI ĐA NĂNG	50
6.8 THANH GHI CỜ	51
6.9 QUY TẮC ĐẶT TÊN BIÊN	51
6.10 KIỂU DỮ LIỆU	51
6.11 KHAI BÁO KÍCH THƯỚC	52
6.12 SO SÁNH GIỮA .COM VÀ .EXE.....	52
6.13 KẾT THÚC THỰC THI	53
CÂU HỎI VÀ BÀI TẬP	53
BÀI 7. LẬP TRÌNH ỨNG DỤNG.....	54
7.1 TẬP LỆNH CỦA 80x86	54
7.1.1 Nhóm lệnh di chuyển dữ liệu.....	54
7.1.2 Nhóm lệnh logic và số học	55
7.1.3 Nhóm lệnh dịch chuyển	58

7.1.4 Nhóm lệnh nhảy	59
7.1.5 Lệnh INT 21H	62
7.2 XỬ LÝ MÀN HÌNH.....	63
7.2.1 Tổng quan màn hình	63
7.2.2 Thiết lập con trỏ	63
7.2.3 Đọc vị trí con trỏ.....	64
7.2.4 Xóa và thiết lập màu cho màn hình.....	64
7.2.5 Hiển thị ký tự với thuộc tính tại vị trí con trỏ	65
7.3 THỦ TỤC.....	65
CÂU HỎI VÀ BÀI TẬP.....	66
BÀI 8. QUẢN LÝ TIẾN TRÌNH.....	68
8.1 KHÁI NIỆM TIẾN TRÌNH	68
8.1.1 Tiến trình.....	68
8.1.2 Trạng thái tiến trình	69
8.1.3 Khối quản lý tiến trình	70
8.2 GIAO TIẾP GIỮA CÁC TIẾN TRÌNH	71
8.2.1 Bộ nhớ chia sẻ.....	73
8.2.2 Truyền thông điệp.....	73
8.3 HỆ THỐNG IPC TRONG WINDOWS	78
8.4 GIAO TIẾP TRONG HỆ THỐNG KHÁCH-CHỦ	80
8.4.1 Ổ cắm	80
8.4.2 Cuộc gọi từ xa	81
8.5 LUỒNG	82
8.5.1 Khái niệm	82
8.5.2 Lợi ích	85
8.5.3 Lập trình đa lõi	86
8.5.4 Các mô hình đa luồng	87
8.5.5 Thư viện luồng	88
8.5.6 Luồng trong Windows	88
CÂU HỎI VÀ BÀI TẬP.....	90
BÀI 9. ĐỊNH THỜI CPU	91
9.1 CÁC KHÁI NIỆM CƠ BẢN	91
9.1.1 Các hàng đợi điều phối	91
9.1.2 Các bộ điều phối	93
9.1.3 Chuyển ngữ cảnh.....	95
9.1.4 Chu kỳ CPU-I/O	96
9.1.5 Cơ chế điều phối	96

9.1.6 Bộ phân phát.....	97
9.2 CÁC TIÊU CHUẨN ĐIỀU PHỐI	98
9.3 CÁC GIẢI THUẬT ĐIỀU PHỐI.....	99
9.3.1 Giải thuật đến trước phục vụ trước.....	99
9.3.2 Ưu tiên công việc ngắn nhất.....	100
9.3.3 Điều phối theo độ ưu tiên	103
9.3.4 Điều phối luân phiên	104
9.3.5 Điều phối với hàng đợi nhiều cấp	106
9.3.6 Điều phối hàng đợi phản hồi đa cấp	108
9.4 ĐIỀU PHỐI ĐA BỘ XỬ LÝ.....	110
CÂU HỎI VÀ BÀI TẬP	111
BÀI 10. ĐỒNG BỘ HÓA TIẾN TRÌNH.....	112
10.1 TÀI NGUYÊN GĂNG VÀ ĐOẠN GĂNG	112
10.1.1 Tài nguyên găng.....	112
10.1.2 Đoạn găng.....	114
10.1.3 Yêu cầu của đồng bộ hóa tiến trình	114
10.2 GIẢI PHÁP PETERSON	115
10.3 CÁC GIẢI PHÁP PHẦN CỨNG	117
10.3.1 Giải pháp cầm ngắn.....	117
10.3.2 Dùng chỉ thị TSL.....	118
10.3.3 Nhận xét	119
10.4 SEMAPHORE	120
10.5 MONITORS	122
10.6 GIẢI PHÁP TRAO ĐỔI THÔNG ĐIỆP	125
10.7 VÍ DỤ KINH ĐIỂN	126
10.7.1 Bài toán nhà sản xuất - khách hàng	126
10.7.2 Bài toán đọc/ghi	131
CÂU HỎI VÀ BÀI TẬP	135
BÀI 11. TẮC NGHẼN	136
11.1 KHÁI NIỆM VÀ VÍ DỤ	136
11.2 ĐIỀU KIỆN CẦN CỦA TẮC NGHẼN	137
11.3 NGĂN CHẶN TẮC NGHẼN	138
11.4 TRÁNH TẮC NGHẼN	141
11.4.1 Trạng thái an toàn	141
11.4.2 Giải thuật nhà băng	143
11.4.3 Giải thuật tránh tắc nghẽn	146
11.5 PHÁT HIỆN TẮC NGHẼN	147

11.6 PHỤC HỒI TẮC NGHẼN.....	149
11.6.1 Kết thúc tiến trình	149
11.6.2 Lấy lại tài nguyên.....	150
CÂU HỎI VÀ BÀI TẬP.....	151
BÀI 12. QUẢN LÝ BỘ NHỚ.....	153
12.1 MỞ ĐẦU	153
12.1.1 Địa chỉ vật lý và địa chỉ logic.....	153
12.1.2 Ánh xạ bộ nhớ	154
12.2 CẤP PHÁT LIÊN TỤC	155
12.2.1 Hiện tượng phân mảnh bộ nhớ	155
12.2.2 Ánh xạ và bảo vệ bộ nhớ.....	157
12.3 CẤP PHÁT KHÔNG LIÊN TỤC	158
12.3.1 Kỹ thuật phân trang	158
12.3.2 Kỹ thuật phân đoạn	166
12.3.3 Phân đoạn kết hợp phân trang	171
CÂU HỎI VÀ BÀI TẬP.....	173
BÀI 13. QUẢN LÝ BỘ NHỚ ẢO	174
13.1 MỞ ĐẦU	174
13.1.1 Khái niệm	174
13.1.2 Lợi ích	175
13.2 PHÂN TRANG THEO YÊU CẦU	176
13.2.1 Khái niệm	176
13.2.2 Hỗ trợ phần cứng	176
13.2.3 Xử lý lỗi trang	177
13.3 THAY THẾ TRANG	179
13.3.1 Khái niệm	179
13.3.2 Giải thuật FIFO	180
13.3.3 Giải thuật tối ưu.....	181
13.3.4 Giải thuật LRU	182
13.4 CẤP PHÁT KHUNG TRANG	183
13.4.1 Số khung trang tối thiểu.....	184
13.4.2 Các giải thuật cấp phát	184
13.4.3 Thay thế cục bộ và thay thế toàn cục	185
13.5 TRÌ TRỆ TOÀN HỆ THỐNG	185
13.5.1 Khái niệm và nguyên nhân	185
13.5.2 Mô hình tập làm việc.....	186
13.5.3 Kiểm soát tần suất lỗi trang	187

CÂU HỎI VÀ BÀI TẬP	188
TÀI LIỆU THAM KHẢO	189

HƯỚNG DẪN

MÔ TẢ MÔN HỌC

Môn học này cung cấp cho sinh viên những khái niệm tổng quan về Kiến trúc máy tính và hệ điều hành, chủ yếu nhằm phục vụ cho sinh viên ngành Công Nghệ Thông Tin và ngành Điện tử - Máy Tính.

Nội dung môn học nhấn mạnh đến các nguyên tắc, các chủ đề, các phương pháp tiếp cận và giải quyết vấn đề liên quan đến các công nghệ và kiến trúc cơ bản của lĩnh vực này.

NỘI DUNG MÔN HỌC

Bài 1: CÁC HỆ CƠ SỐ ĐẾM

Cách chuyển đổi giữa các hệ cơ số và khoảng giá trị biểu diễn được của một số nhị phân có số bit xác định

Bài 2: TÍNH TOÁN NHỊ PHÂN

Cách tính một số phép tính cơ bản trên nhị phân và giới thiệu một số mã khác ngoài nhị phân.

Bài 3: ĐẠI SỐ BOOLE VÀ MẠCH SỐ

Cách biến đổi các biểu thức logic và các phương pháp rút gọn

Bài 4: CÁC MẠCH SỐ CƠ BẢN

Cách thiết kế một số các mạch số cơ bản sử dụng các cổng logic.

Bài 5: TỔ CHỨC MÁY TÍNH

Cấu trúc của CPU. Các thành phần bên trong một máy tính, chức năng, nhiệm vụ và cách hoạt động. Khái quát về vi xử lý 8088/8086.

Bài 6: GIỚI THIỆU HỢP NGỮ

Trình bày các khái niệm căn bản cần thiết của lập trình hợp ngữ. Làm quen với cách khai báo, xử lý của một chương trình hợp ngữ

Bài 7: LẬP TRÌNH ỨNG DỤNG

Thực hiện một số ứng dụng đơn giản như hiển thị ký tự, chuỗi ký tự, tính toán, xử lý màn hình...

Bài 8: QUẢN LÝ TIẾN TRÌNH

Trình bày khái niệm tiến trình và luồng, cách thức điều phối tiến trình, giao tiếp giữa các tiến trình, tiến trình đa luồng.

Bài 9: ĐỊNH THỜI CPU

Trình bày các khái niệm, nguyên lý cơ bản về điều phối CPU cho các tiến trình, các giải thuật điều phối.

Bài 10: ĐỒNG BỘ HÓA TIẾN TRÌNH

Trình bày các khái niệm, nguyên lý cơ bản của vấn đề đồng bộ hóa các tiến trình, một số giải thuật đồng bộ hóa

Bài 11: TẮC NGHẼN

Giới thiệu các khái niệm liên quan đến tắc nghẽn hệ thống, một số phương pháp và giải thuật nhằm ngăn chặn, phòng tránh, phát hiện tắc nghẽn.

Bài 12: QUẢN LÝ BỘ NHỚ

Trình bày nguyên lý và ý tưởng một số phương pháp quản lý bộ nhớ như phương pháp phân trang, phương pháp phân đoạn.

Bài 13: QUẢN LÝ BỘ NHỚ ẢO

Trình bày khái niệm và lợi ích của kỹ thuật bộ nhớ ảo, nguyên lý và một số giải thuật phân trang theo yêu cầu và thay thế trang.

KIẾN THỨC TIỀN ĐỀ

Sinh viên có kiến thức căn bản về kiến trúc máy tính, kỹ thuật lập trình, một số hệ điều hành như Windows, Linux.

YÊU CẦU MÔN HỌC

Người học phải dự học đầy đủ các buổi lên lớp và làm bài tập đầy đủ ở nhà.

CÁCH TIẾP CẬN NỘI DUNG MÔN HỌC

Để học tốt môn này, người học cần đọc trước các nội dung chưa được học trên lớp; tham gia đầy đủ và tích cực trên lớp; hiểu các khái niệm, tính chất và ví dụ tại lớp học. Sau khi học xong, cần ôn lại bài đã học, làm các bài tập và câu hỏi. Tìm đọc thêm các tài liệu khác liên quan đến bài học và làm thêm bài tập.

PHƯƠNG PHÁP ĐÁNH GIÁ MÔN HỌC

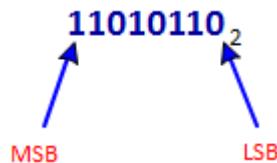
- Điểm quá trình: 50%. Điểm kiểm tra thường xuyên trong quá trình học tập. Điểm kiểm tra giữa học phần, điểm làm bài tập trên lớp, hoặc điểm chuyên cần.
- Điểm thi: 50%. Hình thức bài thi tự luận trong 90 phút, không được mang tài liệu vào phòng thi. Nội dung gồm các câu hỏi và bài tập tương tự như các câu hỏi và bài tập về nhà.

BÀI 1. CÁC HỆ CƠ SỐ ĐÊM

1.1 GIỚI THIỆU

1.1.1 Hệ nhị phân

Hệ nhị phân hay hệ cơ số 2 sử dụng 2 bit “1” và “0” để biểu diễn giá trị. Một giá trị nhị phân là 1 chuỗi bit, với bit nhỏ nhất ngoài cùng bên phải gọi là bit LSB (Least Significant Bit). Bit lớn nhất nằm ngoài cùng bên trái gọi là bit MSB (Most Significant Bit).



Ta có công thức sau để chuyển đổi từ nhị phân sang thập phân.

$$\begin{aligned} A_2 &= a_n a_{n-1} \dots a_1 a_0 = \\ &= a_n 2^n + a_{n-1} 2^{n-1} + \dots + a_1 2^1 + a_0 2^0 \end{aligned}$$

Với a chính là các bit nhị phân có giá trị 0 hoặc 1. Cách viết hệ nhị phân như sau: 1101_2 hoặc 1101_b (b chính là **binary**)

1.1.2 Hệ thập phân

Hệ thập phân là hệ được con người sử dụng từ rất lâu. Hệ thập phân khá gần gũi với con người và được sử dụng hầu như trong tất cả các nền văn minh cổ đại (chỉ khác cách viết). Hệ thập phân ngày nay có tất cả 10 chữ số: từ 0 đến 9. Ta có công thức sau để chuyển đổi từng chữ số thập phân ra giá trị của chính nó:

$$\begin{aligned} A_{10} &= a_n a_{n-1} \dots a_1 a_0 = \\ &= a_n 10^n + a_{n-1} 10^{n-1} + \dots + a_1 10^1 + a_0 10^0 \end{aligned}$$

Với a chính là các chữ số thập phân có giá trị 0 đến 9. Cách viết hệ thập phân như sau: 365_{10} hoặc 365_d (d chính là **decimal**) hoặc 365 (không ghi hệ cơ số thì mặc định là thập phân).

1.1.3 Hệ bát phân và thập lục phân

2 hệ này nhằm mục đích hỗ trợ biểu diễn hệ nhị phân. Hệ bát phân có tất cả 8 chữ số từ 0 đến 7. Hệ thập lục phân có tất cả 16 chữ số từ 0 đến F. Cách viết hệ bát phân như sau: 37_8 hoặc 37_o (o chính là **octal**). Cách viết hệ thập lục phân như sau: AB_{16} hoặc AB_h (h chính là **hexadecimal**).

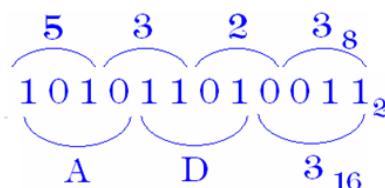
Ta có bảng giá trị như sau:

Thập phân	Nhi phân	Bát phân	Thập lục phân
0	0000	0	0
1	0001	1	1
2	0010	2	2
3	0011	3	3
4	0100	4	4
5	0101	5	5
6	0110	6	6
7	0111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A

11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

1.2 CÁCH CHUYỂN ĐỔI

Do hệ bát phân và thập lục dùng để biểu diễn hệ nhị phân nên cách chuyển đổi khá đơn giản. Một chữ số bát phân có thể chuyển thành 3 bit nhị phân và ngược lại. Tương tự, 1 chữ số thập lục phân có thể chuyển thành 4 bit nhị phân và ngược lại. Ví dụ: ta có chuỗi bit 101011010011_2 , muốn chuyển sang bát phân chỉ việc gom 3 bit thành nhóm lại với nhau từ phải qua trái, mỗi 3 bit nhị phân sẽ chuyển thành 1 chữ số bát phân (5323_8). Tương tự, muốn chuyển sang thập lục phân chỉ việc gom 4 bit nhị phân từ phải qua trái, mỗi 4 bit nhị phân sẽ chuyển thành 1 chữ số thập lục phân ($AD3_{16}$).



Như vậy, 3 hệ cơ số (nhị phân, bát phân, thập lục phân) thực chất là một và việc chuyển đổi giữa 3 hệ này rất nhanh chóng.

Việc chuyển đổi từ nhị phân sang thập phân sẽ như sau: ví dụ ta có 1101_2 . Như vậy thập phân sẽ là: $1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 8 + 4 + 0 + 1 = 13$

Ngược lại, chuyển đổi từ thập phân sang nhị phân có thể thực hiện chia 2 hoặc tính nhẩm. Để tính nhẩm cần học thuộc bảng sau:

$$2^0 = 1$$

$$2^5 = 32$$

$2^1 = 2$	$2^6 = 64$
$2^2 = 4$	$2^7 = 128$
$2^3 = 8$	$2^8 = 256$
$2^4 = 16$	$2^9 = 512$
	$2^{10} = 1024$

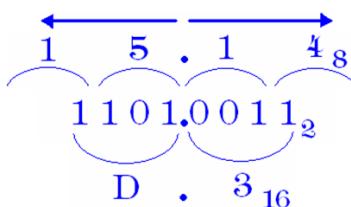
Ví dụ ta có 26_{10} . Như vậy nhị phân sẽ là: $16 + 8 + 2 = 2^4 + 2^3 + 2^1 = 11010_2$

1.3 DẤU CHẤM TĨNH

Một số thực nhị phân có thể biểu diễn dưới dạng dấu chấm tĩnh. Trong đó trước dấu chấm sẽ là $2^0, 2^1, 2^2 \dots$ từ phải qua trái. Sau dấu chấm sẽ là $2^{-1}, 2^{-2}, 2^{-3} \dots$ từ trái qua phải.

$$\begin{aligned} \text{Tương ứng } 101.111_2 &= 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1} + 1 \times 2^{-2} + 1 \times 2^{-3} \\ &= 4 + 0 + 1 + 0.5 + 0.25 + 0.125 \\ &= 5.875_{10} \end{aligned}$$

Cách chuyển đổi một số nhị phân dấu chấm tĩnh sang hệ cơ số bát phân và thập lục phân như sau: Trước dấu chấm (phần nguyên) sẽ gom 3 hoặc 4 bit từ phải qua trái. Sau dấu chấm (phần lẻ) sẽ gom 3 hoặc 4 bit từ trái qua phải. Ví dụ:



1.4 SỐ ÂM

Để biểu diễn số âm trong máy tính người ta sử dụng giá trị bù 2. Bất kỳ một số nhị phân nào ta đều có thể tìm được giá trị bù 2 của nó.

$$\text{Ví dụ: } 1001_2 \xrightarrow{\text{đảo bit}} 0110_2 \xrightarrow{\text{cộng 1}} 0111_2$$

Như vậy ta có 0111_2 chính là bù 2 của 1001_2 . Đặc biệt, nếu ta lấy bù 2 của 0111_2 thì kết quả lại trả về giá trị ban đầu 1001_2 . Có thể nói, cặp số 0111_2 và 1001_2 chính là bù 2 của nhau.

Trong cặp số bù 2 này người ta quy ước như sau:

- + Đây là cặp số có giá trị tuyệt đối bằng nhau nhưng khác dấu
- + Nếu số nào có giá trị MSB = 1 thì là số âm, nếu MSB = 0 là số dương
- + Giá trị tuyệt đối được tính theo số dương

Trở lại ví dụ trên ta tính được đây chính là ± 7 với $0111_2 = +7$, $1001_2 = -7$.

1.5 KHOẢNG GIÁ TRỊ BIỂU DIỄN

Với một số nhị phân có số bit cho trước, ta luôn có thể xác định được khoảng giá trị biểu diễn được cho nó. Do nhị phân có trường hợp không dấu và có dấu nên khoảng giá trị biểu diễn được cũng sẽ khác nhau. Ví dụ: Cho một số nhị phân 3 bit (8 giá trị):

Trường hợp không dấu: từ 0 đến 7 tương ứng $000_2, 001_2, 010_2, 011_2, 100_2, 101_2, 110_2, 111_2$

Trường hợp có dấu: từ -4 đến +3 tương ứng $100_2, 101_2, 110_2, 111_2, 000_2, 001_2, 010_2, 011_2$

CÂU HỎI VÀ BÀI TẬP

1. Nêu ưu và khuyết điểm của dấu chấm tĩnh và dấu chấm động.

2. Thực hiện các chuyển đổi sau trên số nhị phân không dấu

$$59 \rightarrow ?b \rightarrow ?o \rightarrow ?h$$

$$90 \rightarrow ?b \rightarrow ?o \rightarrow ?h$$

$$10101_b \rightarrow ?d \rightarrow ?h \rightarrow ?o$$

$$11111_b \rightarrow ?d \rightarrow ?h \rightarrow ?o$$

3. Thực hiện các chuyển đổi sau trên số nhị phân không dấu (dấu chấm tĩnh)

$7.875 \rightarrow ?b \rightarrow ?o \rightarrow ?h$

$9.75 \rightarrow ?b \rightarrow ?o \rightarrow ?h$

$101.11_b \rightarrow ?d \rightarrow ?h \rightarrow ?o$

$1101.011_b \rightarrow ?d \rightarrow ?h \rightarrow ?o$

4. Thực hiện các chuyển đổi sau trên số nhị phân không dấu (dấu chấm tĩnh)

$7.7 \rightarrow ?b \rightarrow ?o \rightarrow ?h <n.6>$

$5.65 \rightarrow ?b \rightarrow ?o \rightarrow ?h <n.6>$

5. Thực hiện các chuyển đổi sau trên số nhị phân có dấu

$-59 \rightarrow ?b \rightarrow ?o \rightarrow ?h$ (có dấu)

$+19 \rightarrow ?b \rightarrow ?o \rightarrow ?h$ (có dấu)

$0111_b \rightarrow ?d \rightarrow ?h \rightarrow ?o$ (có dấu)

$10101_b \rightarrow ?d \rightarrow ?h \rightarrow ?o$ (có dấu)

6. Thực hiện các chuyển đổi sau trên số nhị phân có dấu (dấu chấm tĩnh)

$-11.25 \rightarrow ?b \rightarrow ?o \rightarrow ?h$ (có dấu)

$-5.35 \rightarrow ?b \rightarrow ?o \rightarrow ?h$ (có dấu) $<n.4>$

$+9.7 \rightarrow ?b \rightarrow ?o \rightarrow ?h$ (có dấu) $<n.4>$

$0111.101_b \rightarrow ?d \rightarrow ?h \rightarrow ?o$ (có dấu)

$101.011_b \rightarrow ?d \rightarrow ?h \rightarrow ?o$ (có dấu)

7. Cho biết khoảng giá trị biểu diễn được của các số nhị phân có số bit sau

	Không dấu	Có dấu
4 bit		
7 bit		
n bit		
<3.2>		
<1.1>		
<n.m>		

BÀI 2. TÍNH TOÁN NHỊ PHÂN

2.1 CÁC PHÉP TOÁN TRÊN NHỊ PHÂN

Trên nhị phân cũng như trên thập phân, ta có một số phép toán cơ bản: cộng, trừ, nhân, chia. Tổng quát cho trường hợp 1 bit, ta có:

Cộng, trừ

$$0 + 0 = 0 \qquad \qquad \qquad 0 - 0 = 0$$

$$0 + 1 = 1 \qquad \qquad \qquad 0 - 1 = 1 \text{ nhớ } 1$$

$$1 + 0 = 1 \qquad \qquad \qquad 1 - 0 = 1$$

$$1 + 1 = 0 \text{ nhớ } 1 \qquad \qquad \qquad 1 - 1 = 0$$

Nhân, chia

$$0 \times 0 = 0$$

$$0 \times 1 = 0 \qquad \qquad \qquad 0 \div 1 = 0$$

$$1 \times 0 = 0 \qquad \qquad \qquad 1 \div 1 = 1$$

$$1 \times 1 = 1$$

2.1.1 Phép cộng, trừ

Do trong nhị phân chia thành số có dấu và không dấu nên trên phép cộng, trừ cũng phải chia làm trường hợp cộng/trừ trên số có dấu và không dấu. Trong trường hợp số nhị phân không dấu, ta có thể cộng/trừ bình thường. Trong trường hợp cộng/trừ có dấu, phép toán phải **bảo toàn số bit**.

Ví dụ: 2 phép toán cộng thực hiện trên số nhị phân không dấu và có dấu

Không dấu $ \begin{array}{r} + \quad \overset{\bullet}{0}110 \quad 6 \\ \underline{-} \quad 1101 \quad 13 \\ \hline 10011 \quad 19 \end{array} $	Có dấu $ \begin{array}{r} + \quad \overset{\bullet}{0}110 \quad +6 \\ \underline{-} \quad 1101 \quad -3 \\ \hline \cancel{1}0011 \quad +3 \end{array} $
--	---

Ví dụ: 2 phép toán trừ thực hiện trên số nhị phân không dấu và có dấu

Không dấu $ \begin{array}{r} - \quad 1110 \quad 14 \\ \underline{-} \quad \overset{\bullet\bullet}{0}011 \quad 3 \\ \hline 1011 \quad 11 \end{array} $	Có dấu $ \begin{array}{r} - \quad 1110 \quad -2 \\ \underline{-} \quad \overset{\bullet\bullet}{0}011 \quad +3 \\ \hline 1011 \quad -5 \end{array} $
--	--

* Các trường hợp bị tràn trên phép toán cộng/trừ: khi ta thực hiện các phép toán cộng và trừ, nếu kết quả có giá trị lớn hơn khoảng giá trị biểu diễn được, phép toán sẽ bị tràn dẫn đến kết quả sai. Có 4 trường hợp bị tràn như sau:

một số dương + một số dương = một số âm

một số âm + một số âm = một số dương

một số dương - một số âm = một số âm

một số âm - một số dương = một số dương

2.1.2 Phép nhân, chia

Trong nhị phân, các phép toán nhân, chia được thực hiện giống như thập phân. Ví dụ 2 phép toán nhân và chia như sau:

$ \begin{array}{r} \times \quad 1010_2 \quad 10 \\ \underline{-} \quad 0111_2 \quad 7 \\ \hline 1010 \quad 70 \end{array} $ $ \begin{array}{r} + \quad \overset{\bullet}{1}010 \\ \underline{-} \quad \overset{\bullet}{1}010 \\ \hline 1000110 \end{array} $	$ \begin{array}{r} - \quad 1111 \quad 101 \quad 15 \div 5 = 3 \\ \underline{-} \quad 101 \quad 11 \\ \hline 101 \\ \underline{-} \quad 101 \\ \hline 0 \end{array} $
--	--

Các phép toán nhân và chia trong nhị phân thường được thực hiện trên số không dấu, nếu thực hiện trên số có dấu, người ta sẽ dùng mạch bù 2 để biến đổi dấu cho các toán hạng và kết quả.

2.2 DẤU CHẤM ĐỘNG

Khi thực hiện các phép toán trên nhị phân, dấu chấm tĩnh là lựa chọn duy nhất. Tuy nhiên, khi lưu trữ trong máy tính, dấu chấm tĩnh sẽ biểu diễn được một khoảng giá trị rất nhỏ. Để cải tiến điều này, người ta đưa ra khái niệm dấu chấm động. Ưu điểm của dấu chấm động là nhằm mục đích tăng khoảng biểu diễn của một số nhị phân với số bit cho trước.

Phần này sẽ trình bày về dấu chấm động 32 bit độ chính xác đơn (IEEE754). Với 32 bit được chia ra như sau:

1	8	23
S	E	M
Sign (Dấu)	Exponent (số mũ)	Mantissa (phần định trị)

Phần bit dấu: quy ước 0 là dương, 1 là âm

$$E = \text{số mũ} + 127$$

$$M = \text{giá trị tính từ sau dấu chấm}$$

Ví dụ:

Tìm dạng dấu chấm động của **0.75**

$$B1 : \text{Đổi sang nhị phân} : 0.75 = 0.11_2$$

$$B2 : \text{Đẩy lên hàng đơn vị một số} 1 : 0.11_2 = 1.1 \times 2^{-1}$$

$$B3 : \text{Số dương nên } S=0$$

$$B4 : E = -1 + 127 = 126 = 0111\ 1110_2$$

$$B5 : M = 100\ 0000\ 0000\ 0000\ 0000\ 0000 \text{ (23 bit)}$$

Ví dụ:

Tìm dạng dấu chấm động của **-9.125**

$$B1 : \text{Đổi sang nhị phân} : 9.125 = 1001.001_2$$

B2 : Đẩy lên hàng đơn vị một số 1 : $1001.001_2 = 1.001001 \times 2^3$

B3 : Số âm nên S=1

B4 : E = 3 + 127 = 130 = 1000 0010₂

B5 : M = 001 0010 0000 0000 0000 0000 (23 bit)

Một số trường hợp đặc biệt:

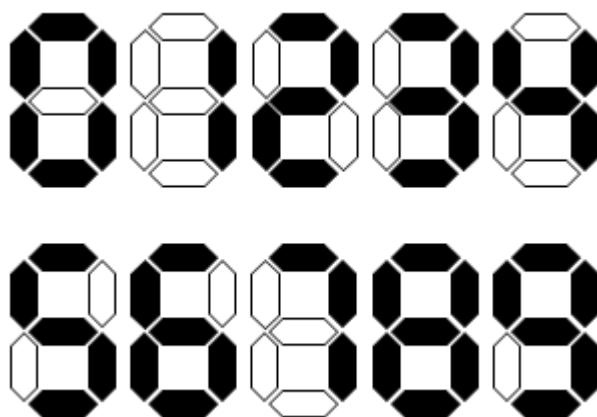
Number	Sign	Exponent	Fraction
0	X	00000000	00000000000000000000000000000000
∞	0	11111111	00000000000000000000000000000000
$-\infty$	1	11111111	00000000000000000000000000000000
NaN	X	11111111	non-zero

2.3 CÁC MÃ KHÁC

Mã BCD dùng để biểu diễn hệ thập phân bằng các bit nhị phân. Mã này thường được sử dụng trước khi qua khôi giải mã led 7 đoạn. Do trong máy tính đều sử dụng hệ nhị phân nhưng trong đời thực lại dùng hệ thập phân, nên khi chuyển các giá trị sau khi tính toán, xử lý trên máy tính ra hiển thị thì phải chuyển qua thập phân.

Mã BCD sử dụng 4 bit nhị phân tương ứng với 1 chữ số thập phân. Ví dụ: $10011_2 = 19_{10} = 0001\ 1001_{BCD}$

Hình 2.1 thể hiện cách giải mã trên led 7 đoạn từ 0 đến 9.



Hình 2.1 Giải mã trên led 7 đoạn

Mã Gray thường được dùng trong các lệnh import/outport. Đặc điểm của mã Gray là 2 số có giá trị liền kề nhau thì khác nhau 1 bit. Ta có bảng mã Gray 3 bit như sau:

Thập phân	Nhị phân	Gray
0	000	000
1	001	001
2	010	011
3	011	010
4	100	110
5	101	111
6	110	101
7	111	100

CÂU HỎI VÀ BÀI TẬP

1. Thực hiện các phép toán cộng/trừ sau (có dấu)

$$1001_2 + 0111_2$$

$$1100_2 - 0011_2$$

$$1100_2 - 1111_2$$

$$1000_2 + 1101_2$$

$$0011_2 + 0011_2$$

$$0011_2 + 0111_2$$

2. Thực hiện các phép toán cộng sau (có dấu)

$$10.010_2 + 01.111_2$$

$$1.101_2 + 0.11_2$$

$$1.001_2 + 1.11_2$$

$$0.11_2 + 01.11_2$$

$$0.11_2 + 0.11_2$$

$$10.001_2 + 1.011_2$$

3. Thực hiện các phép toán trừ sau (có dấu)

$$10.110_2 - 01.101_2$$

$$1.101_2 - 0.11_2$$

$$1.001_2 - 1.111_2$$

$$10.011_2 - 1.111_2$$

$$0.01_2 - 0.11_2$$

$$1.01_2 - 01.11_2$$

4. Tìm dạng dấu chấm động của các giá trị sau: 3.875; -7.625; -0.375

5. Thực hiện các chuyển đổi sau

$$59_{10} \rightarrow ?b \rightarrow ?o \rightarrow ?h \rightarrow ?BCD \rightarrow ?Gray \rightarrow ?IEEE754 \text{ (dấu chấm động)}$$

$$77_{10} \rightarrow ?b \rightarrow ?o \rightarrow ?h \rightarrow ?BCD \rightarrow ?Gray \rightarrow ?IEEE754 \text{ (dấu chấm động)}$$

$$1101101_{gray} \rightarrow ?b \rightarrow ?d \rightarrow ?o \rightarrow ?h \rightarrow ?BCD \rightarrow ?IEEE754 \text{ (dấu chấm động)}$$

BÀI 3. ĐẠI SỐ BOOLE VÀ MẠCH SỐ

3.1 GIỚI THIỆU

3.1.1 Đại số Boole và các cổng logic

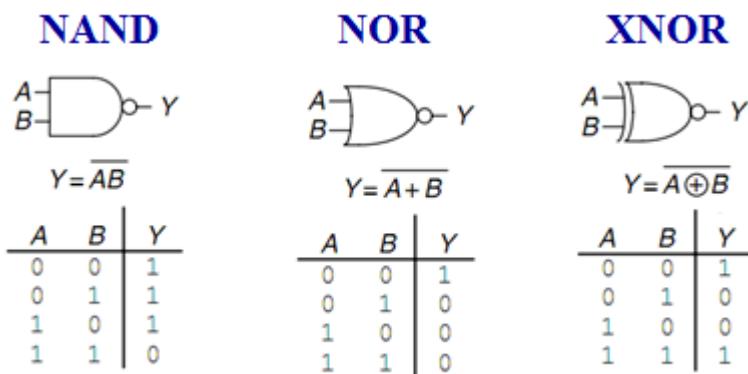
Đại số Boole dựa trên các mệnh đề “đúng” hoặc “sai”. Trong máy tính, trường hợp 1 bit thì “đúng” là 1, “sai” là 0. Nếu có số bit nhiều hơn 1, thì “đúng” là khác 0, “sai” là 0.

Một số các ký hiệu của phép toán trong đại số Boole

- **AND**
- + **OR**
- \bar{A} **NOT**
- \oplus **XOR**

7 cổng logic cơ bản gồm có: NOT, AND, OR, XOR, NAND, NOR, XNOR (hình 3.1).
Toàn bộ các hệ thống số đều dựa trên 7 cổng logic này để thiết kế.

NOT	AND	OR	XOR
			
$Y = \bar{A}$	$Y = AB$	$Y = A + B$	$Y = A \oplus B$
$\begin{array}{ c c } \hline A & Y \\ \hline 0 & 1 \\ 1 & 0 \\ \hline \end{array}$	$\begin{array}{ c c c } \hline A & B & Y \\ \hline 0 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \\ \hline \end{array}$	$\begin{array}{ c c c } \hline A & B & Y \\ \hline 0 & 0 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \\ \hline \end{array}$	$\begin{array}{ c c c } \hline A & B & Y \\ \hline 0 & 0 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ \hline \end{array}$



Hình 3.1 Các cổng logic _ hình vẽ, biểu thức và bảng chân trị

Trong trường hợp tổng quát với cổng logic n ngõ vào, ta có các phát biểu sau:

Cổng AND : lên 1 khi tất cả ngõ vào là 1

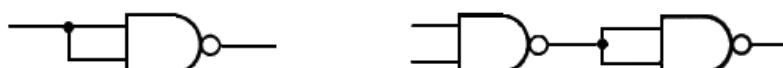
Cổng OR : xuống 0 khi tất cả ngõ vào là 0

Cổng XOR : lên 1 khi số ngõ vào mang giá trị 1 là lẻ

Cổng XNOR : lên 1 khi số ngõ vào mang giá trị 1 là chẵn

Tuy có tất cả là 7 cổng logic, nhưng trong thực tế chế tạo và sản xuất IC. Không bao giờ người ta thực hiện chế tạo 7 cổng logic riêng biệt mà chỉ dựa vào 1 cổng để tạo ra 6 cổng còn lại. Cổng được chọn là NAND hoặc NOR. Ví dụ:

Cổng NOT sử dụng 1 cổng NAND hoặc cổng AND dùng 2 cổng NAND (hình 3.2).



Hình 3.2 Cổng NOT và AND sử dụng cổng NAND

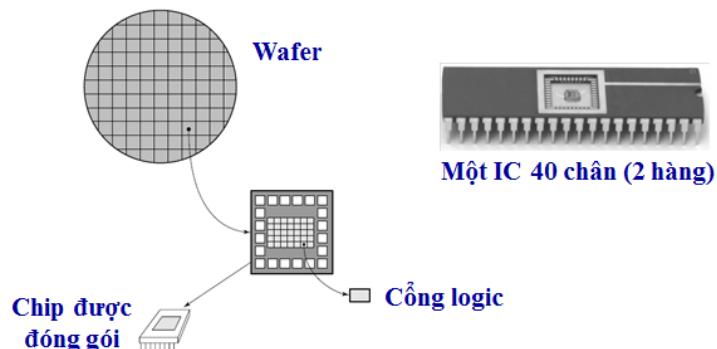
3.1.2 Quy trình chế tạo

Các cổng logic được cấu tạo từ các transistor. Transistor được cấu tạo từ các chất bán dẫn thông qua quá trình chạm, khắc trên bề mặt 1 tấm wafer. Toàn bộ các công đoạn sản xuất chip đều được thực hiện trong phòng sạch (hình 3.3).



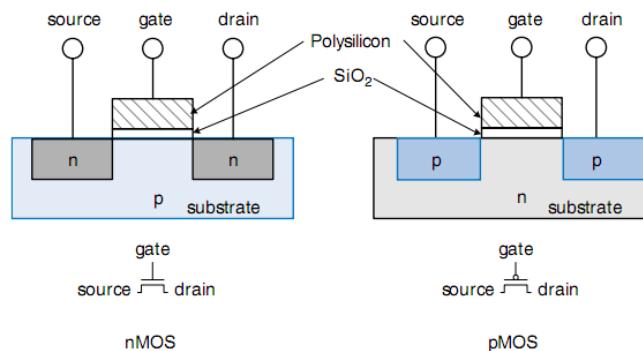
Hình 3.3 Phòng sạch và wafer

Các tấm wafer được cắt ra từ một khối trụ. Tấm wafer hình tròn sẽ được chia ra thành các ô vuông nhỏ, mỗi ô vuông chính là một lõi IP của con chip. Qua quá trình chạm, khắc hoàn chỉnh. Tấm wafer sẽ được cắt nhỏ ra thành các khối vuông nhỏ. Mỗi khối là một lõi IP. Mỗi lõi IP sẽ được gắn vào 1 vỏ nhựa, nối chân ra bên ngoài và hàn lại. Sản phẩm hoàn chỉnh là một con IC. Toàn bộ quá trình được mô tả như hình 3.4.

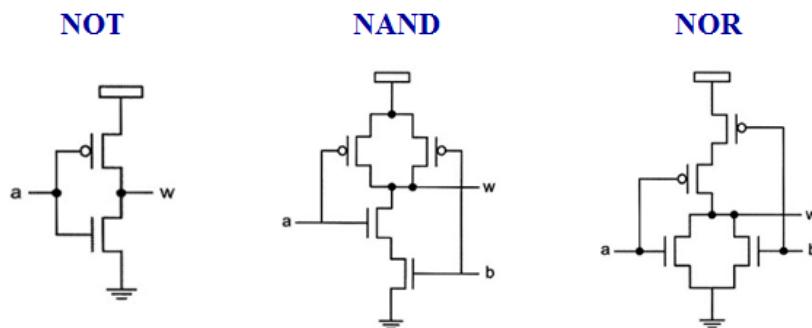


Hình 3.4 Quá trình chế tạo IC

Có 2 loại transistor được sử dụng để chế tạo cổng logic theo công nghệ CMOS. Đó là nMOS và pMOS (hình 3.5). Các đặc tính của 2 transistor này trái ngược nhau: nMOS truyền mass tốt, ngõ vào là 1 thì dẫn, 0 là ngắt; pMOS truyền nguồn tốt, ngõ vào là 0 thì dẫn, 1 là ngắt. Từ những đặc tính cơ bản này, người ta ghép chúng lại để tạo thành các cổng logic như hình 3.6.



Hình 3.5 pMOS và nMOS



Hình 3.6 Cổng logic từ transistor

3.1.3 Biểu thức logic

Các tiên đề của đại số Boole

*** Tính giao hoán**

$$A + B = B + A$$

$$A \cdot B = B \cdot A$$

*** Tính kết hợp**

$$(A + B) + C = A + (B + C) \quad (A \cdot B) \cdot C = A \cdot (B \cdot C)$$

*** Tính phân bố**

$$A \cdot (B + C) = A \cdot B + A \cdot C$$

$$A + B \cdot C = (A + B) \cdot (A + C)$$

+ Một số biểu thức logic cho trường hợp 1 biến:

$$\overline{\overline{A}} = A$$

$$A + A = A \quad \bigcirc \quad A + A = 0 \quad A \cdot A = A$$

$$A + \overline{A} = 1 \quad \bigcirc \quad \overline{A} + A = 1 \quad \overline{A} \cdot A = 0$$

$$A + 1 = 1 \quad \bigcirc \quad A + \overline{1} = A \quad A \cdot 1 = A$$

$$A + 0 = A \quad \bigcirc \quad A + 0 = A \quad A \cdot 0 = 0$$

+ Một số biểu thức logic cho trường hợp nhiều biến:

$$A + A \cdot B = A \quad A \cdot (A + B) = A$$

$$A \cdot B + A \cdot \overline{B} = A \quad (A + B) \cdot (\overline{A} + B) = A$$

$$A + \overline{A} \cdot B = A + B \quad \overline{A} \cdot (A + B) = A \cdot B$$

$$A \cdot B + \overline{A} \cdot C + B \cdot C = A \cdot B + \overline{A} \cdot C$$

$$(A+B) \cdot (\overline{A}+C) \cdot (B+C) = (A+B) \cdot (\overline{A}+C)$$

+ Qui tắc Demorgan

$$\overline{A \cdot B} = \overline{A} + \overline{B}$$

$$\overline{A + B} = \overline{A} \cdot \overline{B}$$

3.2 RÚT GỌN BIỂU THỨC LOGIC

Có 3 phương pháp rút gọn biểu thức logic:

- + Biến đổi đại số Boolean
- + Giản đồ Karnaugh
- + Quine McCluskey

Mỗi phương pháp đều có những ưu khuyết điểm riêng. Dùng cách biến đổi đại số Boolean mang tính chất tính toán và ít được sử dụng nhất. Phương pháp này dựa trên các tiên đề của đại số Boolean để thực hiện việc biến đổi. Phương pháp dùng giản đồ Karnaugh đơn giản, trực quan và cho kết quả nhanh nên được sử dụng chủ yếu. Phương pháp Quine McCluskey chủ yếu dùng để lập trình trên máy tính.

3.2.1 Giản đồ Karnaugh

Giản đồ Karnaugh có những tính chất khá đặc biệt chuyên dùng để rút gọn một biểu thức logic bất kỳ. Bất kỳ 2 ô nào liền kề (chung cạnh) trên giản đồ sẽ khác nhau 1 bit. Đặc biệt, giản đồ Karnaugh là loại giản đồ “không có biên”. Nếu ta đi hết một chiều của giản đồ Karnaugh thì sẽ quay lại.

+ Qui tắc vẽ

$$\text{Số ô} = 2^{\text{số biến}}$$

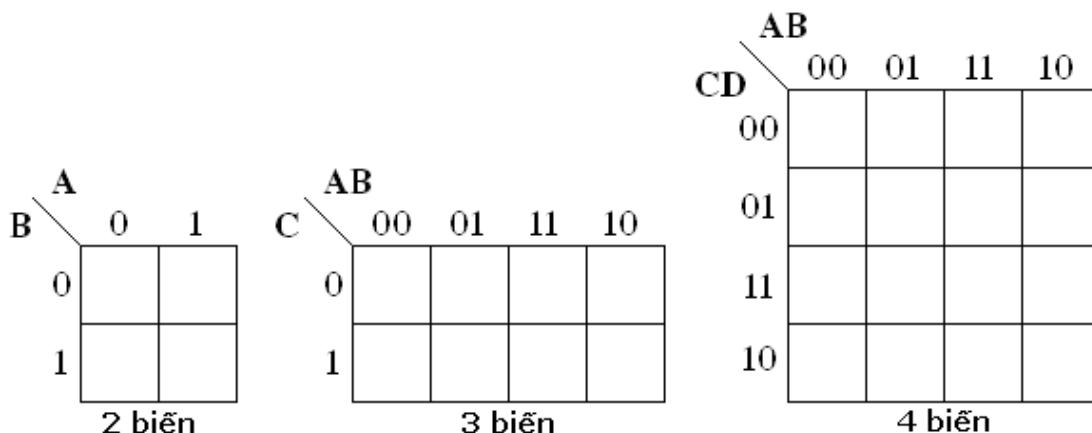
2 ô liền kề nhau thì khác nhau 1 bit

+ Qui tắc khoanh

$$\text{Số ô phải bằng } 2^n$$

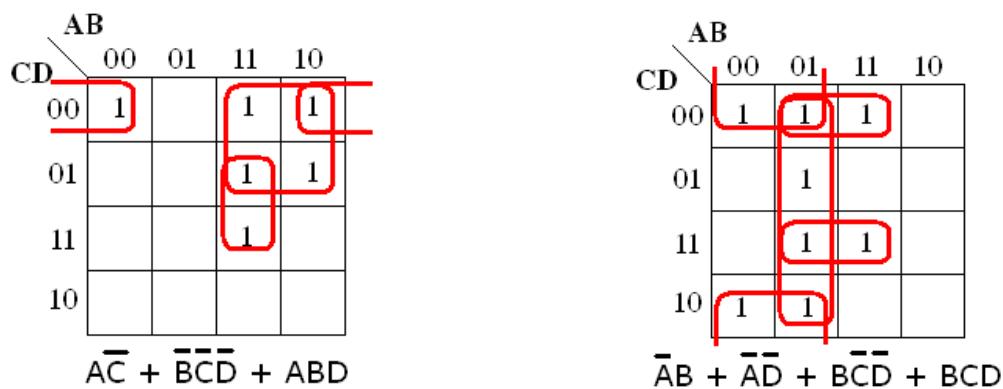
Diện tích khoanh phải lớn nhất

Hình 3.7 là giản đồ Karnaugh cho trường hợp 2 biến, 3 biến và 4 biến.



Hình 3.7 Giản đồ Karnaugh

Hình 3.8 là ví dụ rút gọn biểu thức qua giản đồ Karnaugh.



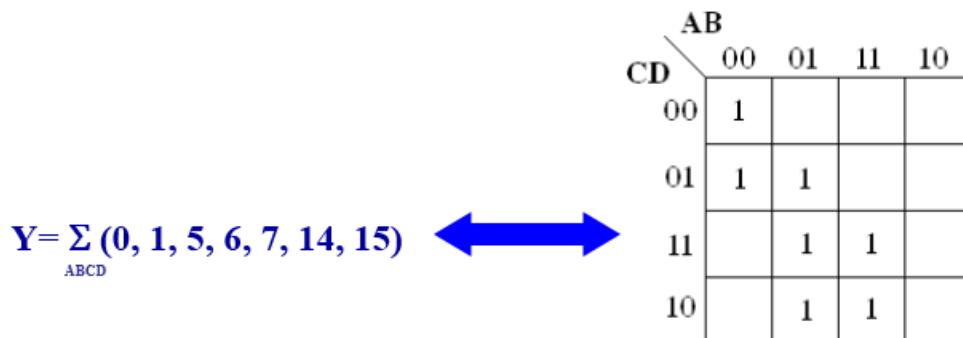
Hình 3.8 Khoanh và rút gọn với giản đồ Karnaugh

Thay vì vẽ hình như trên, ta có thể ghi biểu thức cho giản đồ Karnaugh. Mỗi một ô trên giản đồ Karnaugh tương ứng với một con số. Con số được đổi từ hàng ngang và hàng dọc theo như hình 3.9.

		AB	00	01	11	10
		CD	00			
		00	0	4	12	8
		01	1	5	13	9
		11	3	7	15	11
		10	2	6	14	10

Hình 3.9 Con số tương ứng trên giản đồ Karnaugh 4 biến

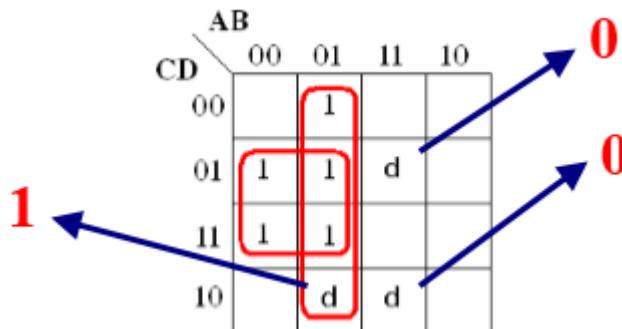
Như vậy thay vì vẽ giản đồ ta có thể viết ra một biểu thức tương ứng. Ví dụ được trình bày ở hình 3.10.



Hình 3.10 Cách viết biểu thức và giản đồ Karnaugh tương đương

* Giản đồ Karnaugh có tín hiệu don't care.

Don't care : không quan trọng, mang giá trị 1 hay 0 đều được. Người ta thường sử dụng các ô mang giá trị don't care để biểu thức logic đơn giản nhất (hình 3.11).



Hình 3.11 Giản đồ Karnaugh có tín hiệu don't care

Như trên hình 3.11 có 3 ô mang giá trị don't care. Do ô don't care mang giá trị 1 hay 0 đều được cho nên ta sẽ chọn giá trị cho ô don't care để biểu thức rút gọn được đơn giản nhất hay số ô khoanh được là lớn nhất. Do đó ta sẽ có 1 ô don't care mang giá trị 1, 2 ô don't care mang giá trị 0.

3.2.2 Quine McCluskey

Phương pháp Quine McCluskey được sử dụng cho các lập trình viên muốn thực hiện trên máy tính. Do máy tính không có khả năng vẽ hình và khoanh như con người. Phương pháp này được thực hiện như sau. Ví dụ ta có biểu thức logic sau:

$$Y = \sum_{ABCD} (2, 3, 6, 7, 12, 13, 14, 15)$$

+ Bước 1: ta sẽ đổi các giá trị ra nhị phân.

2	0010
3	0011
6	0110
7	0111
12	1100
13	1101
14	1110
15	1111

+ Bước 2: tìm những cặp số khác nhau 1 bit. Vị trí khác nhau được đánh dấu bằng dấu “_”.

(2,3)	001-	(7,15)	-111
(2,6)	0-10	(12,13)	110-
(3,7)	0-11	(12,14)	11-0
(6,7)	011-	(13,15)	11-1
(6,14)	-110	(14,15)	111-

+ Bước 3: tiếp tục tìm trong bước 2 những cặp khác nhau 1 bit

(6,7,14,15)	-11-
(2,3,6,7)	0-1-
(12,13,14,15)	11--

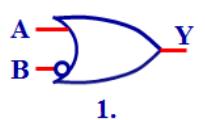
+ Bước 4: lập bảng để đưa ra biểu thức cuối cùng

	2	3	6	7	12	13	14	15
(6,7,14,15)	-11-	BC		*	*		*	*
(2,3,6,7)	0-1-	\overline{AC}	✗	✗	*	*		
(12,13,14,15)	11--	AB			✗	✗	*	*

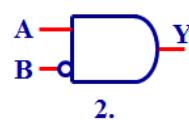
Vậy biểu thức rút gọn được là $Y = \overline{AC} + AB$

CÂU HỎI VÀ BÀI TẬP

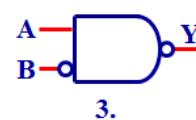
1. Tìm bảng chân trị tương ứng với cổng logic sau.



1.



2.



3.

A	B	Y
0	0	0
0	1	0
1	0	1
1	1	0

a.

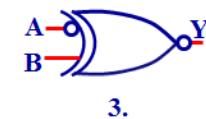
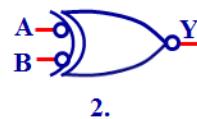
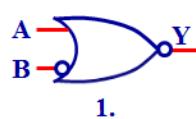
A	B	Y
0	0	1
0	1	1
1	0	0
1	1	1

b.

A	B	Y
0	0	1
0	1	0
1	0	1
1	1	1

c.

2. Tìm bảng chân trị tương ứng với cổng logic sau.



A	B	Y
0	0	1
0	1	0
1	0	0
1	1	1

a.

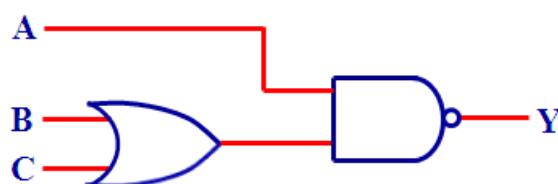
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

b.

A	B	Y
0	0	0
0	1	1
1	0	0
1	1	0

c.

3. Điền vào bảng chân trị



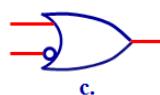
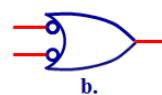
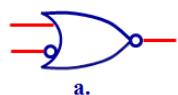
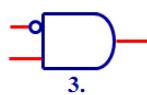
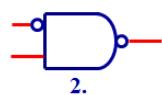
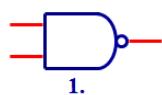
A	B	C	Y
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

4. Khai triển biểu thức logic sau

$$\overline{A \oplus B \oplus C \oplus D} = ?$$

$$\overline{\overline{A \oplus B \oplus C \oplus D}} = ?$$

5. Các cổng logic sau cổng nào tương đương nhau



6. Khoanh và đưa ra biểu thức logic

	AB	00	01	11	10
CD	00	1		1	1
	01		1	1	
	11				
	10	1			1

	AB	00	01	11	10
CD	00		1	1	1
	01		1		
	11			1	
	10	1	1	1	

	AB	00	01	11	10
CD	00	1			1
	01	1			1
	11				1
	10	1	1	1	

	AB	00	01	11	10
CD	00	1	1	1	1
	01	1	1	1	1
	11	1	1		1
	10	1	1	1	1

7. Vẽ giản đồ Karnaugh, khoanh và đưa ra biểu thức logic

$$Y = \sum_{ABCD} (0, 1, 5, 7, 8, 9, 14, 15)$$

$$Y = \sum_{ABCDE} (6, 7, 14, 15, 22, 23, 30, 31)$$

$$Y = \sum_{ABCDE} (1, 4, 5, 17, 20, 21, 29)$$

$$Y = \sum_{ABCDEF} (2, 8, 10, 18, 24, 26, 34, 37, 42, 45, 50, 53, 58, 61)$$

8. Vẽ giản đồ Karnaugh, khoanh và đưa ra biểu thức logic

$$Y = \prod_{ABCD} (0, 2, 3, 7, 8, 10, 11, 15)$$

$$Y = \prod_{ABCD} (2, 3, 5, 7, 10, 11, 13, 15)$$

9. Vẽ giản đồ Karnaugh, khoanh và đưa ra biểu thức logic

$$Y = \sum_{ABCD} (0, 4, 7, 8, 15) + d(2, 10, 14)$$

$$Y = \sum_{ABCD} (1, 3, 7, 15) + d(5, 9, 12)$$

10. Khoanh và đưa ra biểu thức logic

	AB	00	01	11	10
CD	00		1		1
	01	1		1	
	11		1		1
	10	1		1	

	AB	00	01	11	10
CD	00		1	1	
	01	1			1
	11		1	1	
	10	1			1

	AB	00	01	11	10
CD	00	1		1	
	01	1		1	
	11		1		1
	10		1		1

	AB	00	01	11	10
CD	00	1			1
	01		1	1	
	11		1	1	
	10	1			1

11. Rút gọn sử dụng phương pháp Quine McCluskey

$$Y = \sum_{ABCD} (0, 2, 3, 7, 8, 10, 11, 15)$$

$$Y = \sum_{ABCD} (2, 3, 5, 7, 10, 11, 13, 15)$$

$$Y = \sum_{ABCD} (2, 3, 5, 10, 11, 13)$$

BÀI 4. CÁC MẠCH SỐ CƠ BẢN

4.1 MẠCH TÍNH TOÁN SỐ HỌC

Để thiết kế một mạch số đơn giản, có số input từ 2-6, cần trải qua 3 bước:

Bước 1 : Lập bảng trạng thái (hay chân trị)

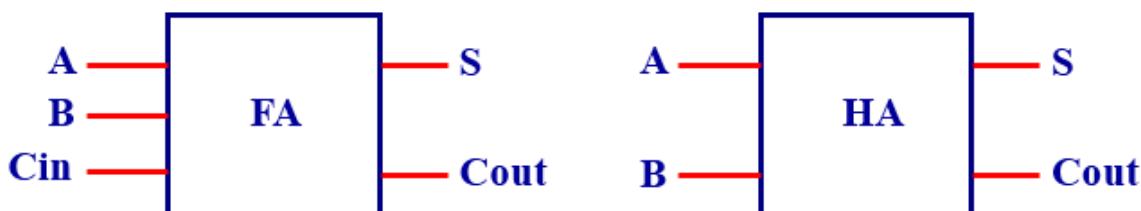
Bước 2 : Rút gọn (thường dùng giản đồ Karnaugh)

Bước 3 : Vẽ hình

Phần sau sẽ đi chi tiết về cách thiết các mạch số cơ bản.

4.1.1 Mạch cộng/trừ

Trong mạch cộng/trừ có thành phần cơ bản nhất đó là mạch Full Adder/Subtractor và mạch Half Adder/Subtractor. Đây chính là các mạch cộng/trừ 1 bit. Sơ đồ khối của mạch FA và HA được vẽ ở hình 4.1. FA có 3 ngõ vào, đây là mạch cộng 3 số 1 bit. HA có 2 ngõ vào, đây là mạch cộng 2 số 1 bit. Kết quả của 2 mạch cộng trên là S, nhớ là Cout.



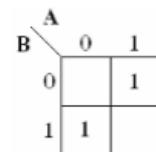
Hình 4.1 FA và HA

Để thiết kế HA, ta sẽ trải qua 3 bước:

+ Bước 1: lập bảng chân trị

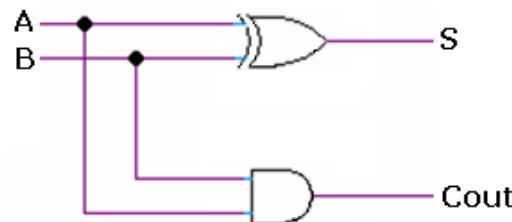
A	B	S	Cout
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

+ Bước 2: Rút gọn dùng giản đồ Karnaugh



$$\begin{aligned} S &= A \oplus B \\ \text{Cout} &= A \cdot B \end{aligned}$$

+ Bước 3: vẽ hình



Tương tự cho FA, cũng trải qua 3 bước

+ Bước 1: lập bảng chân trị

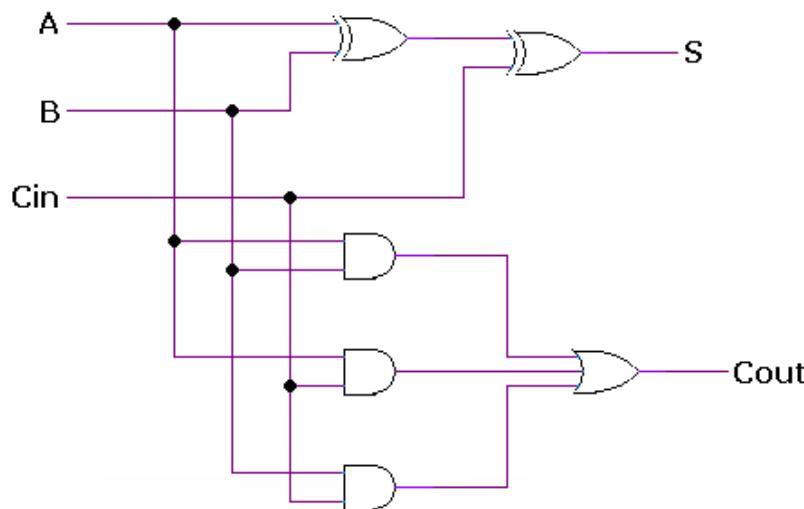
A	B	Cin	S	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

+ Bước 2: Rút gọn dùng giản đồ Karnaugh

	AB	00	01	11	10
Cin	0		1		1
	1	1		1	

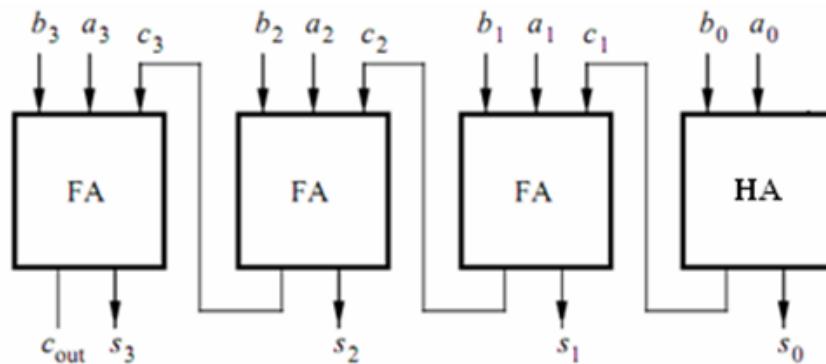
	AB	00	01	11	10
Cin	0			1	
	1		1	1	1

+ Bước 3: vẽ hình



Hoàn toàn tương tự cho cách thiết kế Full Subtractor và Half Subtractor.

Để tạo thành mạch cộng 4 bit, 1 HA và 3 FA sẽ được ghép nối tiếp theo hình 4.2.



Hình 4.2 Mạch cộng 4 bit

Mạch này có tầng đầu tiên (hàng đơn vị) sử dụng HA vì chưa có nhớ, các tầng sau phải sử dụng FA vì đã có nhớ.

4.1.2 Mạch nhân

Ta có 2 số 3 bit A và B nhân nhau, khi ghi chi tiết ta sẽ có biểu thức như sau

$$\begin{array}{r}
 \times \quad A_2 \quad A_1 \quad A_0 \\
 \quad \quad B_2 \quad B_1 \quad B_0 \\
 \hline
 \quad A_2B_0 \quad A_1B_0 \quad A_0B_0 \\
 A_2B_1 \quad A_1B_1 \quad A_0B_1 \\
 \hline
 A_2B_2 \quad A_1B_2 \quad A_0B_2 \\
 \hline
 S_5 \quad S_4 \quad S_3 \quad S_2 \quad S_1 \quad S_0
 \end{array}$$

$$S_0 = A_0 \cdot B_0$$

$$S_1 = A_1 \cdot B_0 + A_0 \cdot B_1$$

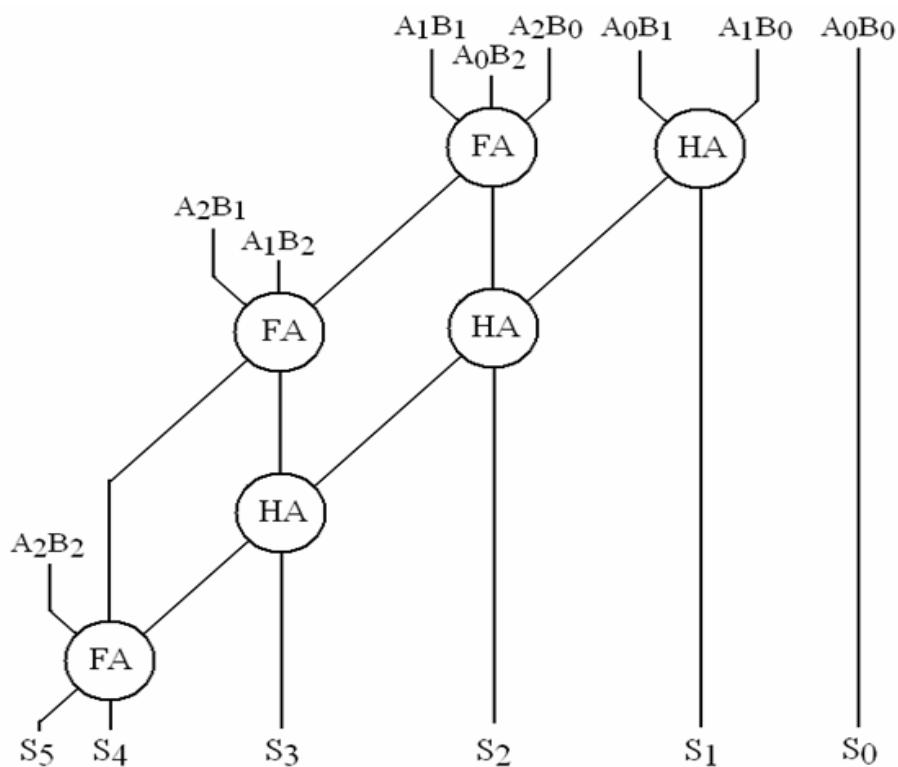
$$S_2 = A_2 \cdot B_0 + A_1 \cdot B_1 + A_0 \cdot B_2$$

$$S_3 = A_2 \cdot B_1 + A_1 \cdot B_2$$

$$S_4 = A_2 \cdot B_2$$

$$S_5 = \text{carry}$$

Như vậy, mạch nhân thực chất được cấu tạo từ cổng AND và FA, HA. Ta có thể vẽ hình mạch nhân như hình 4.3.



Hình 4.3 Mạch nhân 3 bit

Hoàn toàn tương tự khi ta muốn thiết kế các mạch nhân có số bit lớn hơn.

4.2 MẠCH SO SÁNH, ĐA HỢP/GIẢI ĐA HỢP

4.2.1 Mạch so sánh

Với một số bất kỳ, để so sánh bằng ta sẽ so sánh từng chữ số của số đó. Nếu từng chữ số của con số đó bằng nhau nghĩa là 2 số có giá trị bằng nhau. Với so sánh lớn hơn hoặc nhỏ hơn, ta sẽ xét từng chữ số theo thứ tự từ chữ số có giá trị lớn nhất đến nhỏ nhất. Ta có biểu thức sau cho trường hợp so sánh 2 số A và B 4 bit:

$$A = B \Leftrightarrow (A_3=B_3) \text{ và } (A_2=B_2) \text{ và } (A_1=B_1) \text{ và } (A_0=B_0)$$

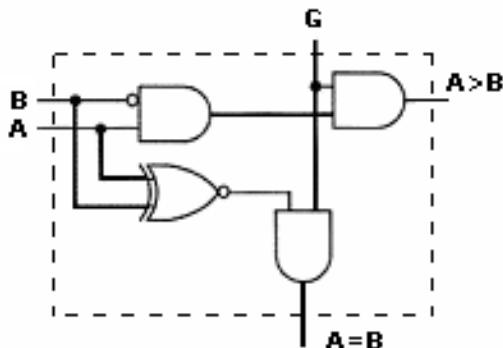
$$A > B \Leftrightarrow (A_3>B_3) \quad \text{hoặc}$$

$$(A_3=B_3) \text{ và } (A_2>B_2) \quad \text{hoặc}$$

$$(A_3=B_3) \text{ và } (A_2=B_2) \text{ và } (A_1>B_1) \quad \text{hoặc}$$

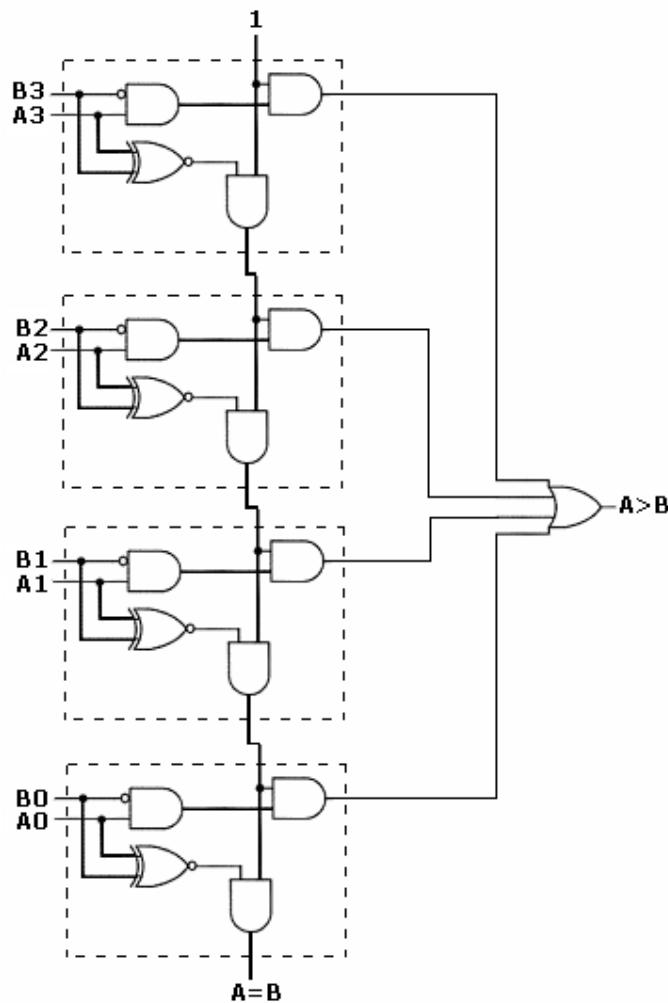
$$(A_3=B_3) \text{ và } (A_2=B_2) \text{ và } (A_1=B_1) \text{ và } (A_0>B_0)$$

Chữ “và” sử dụng cổng AND, chữ “hoặc” sử dụng cổng OR. Như vậy, ta chỉ cần thiết kế mạch so sánh 1 bit như hình 4.4.



Hình 4.4 Mạch so sánh 1 bit

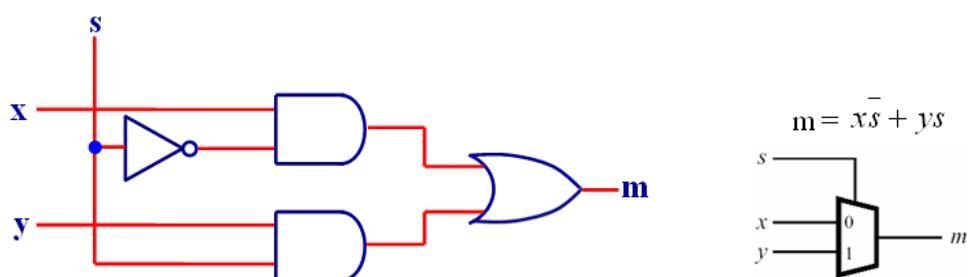
Khi đã có mạch so sánh 1 bit, để tạo thành mạch so sánh 4 bit ta sẽ ghép 4 khối lại với nhau như hình 4.5. Tương ứng, ngõ vào gồm có A và B (4 bit), ngõ ra là “A bằng B” và “A lớn hơn B”.



Hình 4.5 Mạch so sánh 4 bit

4.2.2 Mạch đa hợp/giải đa hợp

Mạch đa hợp được xem là mạch chọn đường cho luồng data di chuyển. Mạch đa hợp thường có **nhiều ngõ vào và chỉ 1 ngõ ra**. Các chân điều khiển sẽ quyết định chọn đường nối từ ngõ vào ra ngõ ra. Hình 4.6 là mạch đa hợp 2-1 1 bit. Mạch này có 2 input là x và y. 1 output là m. 1 tín hiệu điều khiển là s. Nếu s=0 thì x sẽ nối với m, nếu s=1 thì y sẽ nối với m.



Hình 4.6 Mạch đa hợp 1 bit

Mạch giải đa hợp có cấu tạo ngược lại. Mạch giải đa hợp sẽ có **1 ngõ vào và nhiều ngõ ra**. Tín hiệu điều khiển sẽ quyết định ngõ vào kết nối với ngõ ra nào.

4.3 ALU (ARITHMETIC LOGIC UNIT)

CPU cấu tạo chủ yếu gồm 3 thành phần :

ALU (đơn vị luận lý số học): thành phần chịu trách nhiệm tính toán các phép toán số học và logic

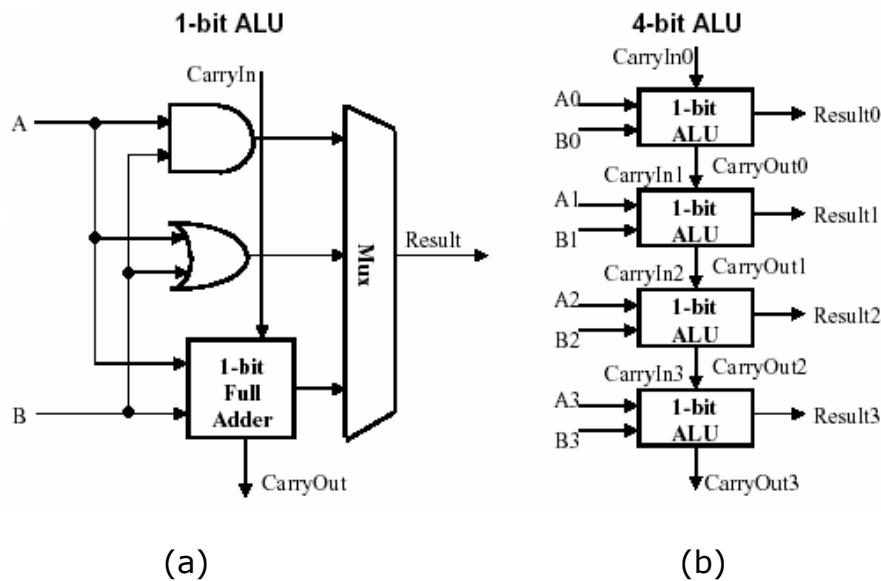
Thanh ghi (Register) : dùng để chứa các toán hạng, kết quả trả về và địa chỉ truy xuất của CPU

Đơn vị điều khiển (CU) : thành phần chịu trách nhiệm điều khiển các trạng thái hoạt động của CPU

Trong 3 thành phần chính này thì ALU đóng vai trò quan trọng nhất, quyết định sức mạnh tính toán của CPU.

Trong hình 4.7(a) là cấu tạo của một bộ ALU 1 bit và hình 4.7(b) là cách ghép các ALU 1 bit tạo thành ALU 4 bit.

ALU thực chất là sự ghép nối của nhiều mạch số cơ bản với nhau tạo thành một mạch số **đa chức năng**, có khả năng thực hiện các phép tính số học và logic. Như trong hình 4.7(a), ALU 1 bit được ghép lại từ 1 cổng AND, 1 cổng OR, 1 bộ Full Adder, 1 bộ đa hợp 3-1. Như vậy bộ ALU 1 bit này có khả năng thực hiện 3 phép toán sau: AND, OR, cộng số học. Để thiết kế các bộ ALU có số bit lớn, ta sẽ ghép các bộ ALU 1 bit này lại với nhau như hình 4.7(b).



Hình 4.7 ALU

CÂU HỎI VÀ BÀI TẬP

1. Thiết kế Full Subtractor và Half Subtractor.
2. Thiết kế mạch bình phương 3 bit.
3. Thiết kế mạch đổi từ nhị phân 4 bit sang mã BCD.
4. Thiết kế mạch đổi từ nhị phân 4 bit sang mã Gray.
5. Thiết kế mạch đổi từ mã Gray 4 bit sang nhị phân.

BÀI 5. TỔ CHỨC MÁY TÍNH

5.1 BỘ VI XỬ LÝ - CPU

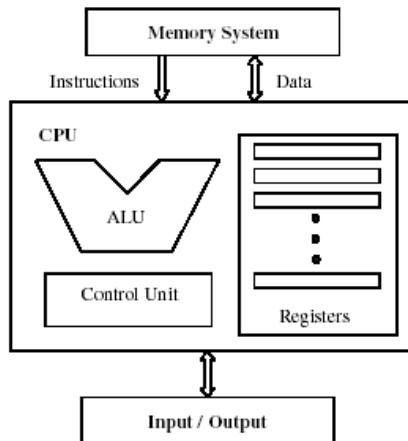
CPU là một vi mạch điện tử có độ tích hợp rất cao và đóng vai trò chủ đạo trong quá trình điều khiển hoạt động của toàn bộ hệ thống. Cấu tạo của nó chính là một mạch tích hợp chứa hàng ngàn, thậm chí hàng triệu transistor (LSI, VLSI) được kết nối với nhau. Các transistor ấy cùng nhau làm việc để lưu trữ và xử lý dữ liệu cho phép bộ vi xử lý có thể thực hiện rất nhiều chức năng khác nhau. CPU đọc mã lệnh dưới dạng các chuỗi bit 0 và 1 (tức dạng nhị phân) từ bộ nhớ, sau đó giải mã lệnh và thực hiện lệnh bằng cách phát ra các tín hiệu điều khiển các bộ phận khác. Bên trong CPU có thanh ghi IP (Instruction Pointer) hoặc PC (Program Counter) chứa địa chỉ của lệnh tiếp theo mà CPU phải thực hiện, thanh ghi lệnh IR (Instruction Register) chứa lệnh đọc được từ bộ nhớ, một số thanh ghi khác dùng làm toán hạng để tính toán, tạo ra địa chỉ trỏ đến dữ liệu... Bên trong CPU còn có ALU (Arithmetic and Logic Unit) chịu trách nhiệm thực hiện các phép toán logic và số học (hình 4.1).

Cấu tạo CPU gồm 3 thành phần :

ALU (đơn vị luận lý số học): thành phần chịu trách nhiệm tính toán các phép toán số học và logic

Thanh ghi : dùng để chứa các toán hạng, kết quả trả về và địa chỉ truy xuất của CPU

CU (đơn vị điều khiển) : thành phần chịu trách nhiệm điều khiển các trạng thái hoạt động của CPU



Hình 5.1 Thành phần của CPU

5.1.1 Bộ xử lý luận lý số học - ALU

Đây là thành phần quan trọng nhất trong CPU. Nó quyết định sức mạnh thật sự của CPU. Việc thực hiện lệnh của CPU chủ yếu sẽ được thực thi bởi bộ ALU. Các chức năng chủ yếu của bộ ALU:

- + Thực hiện các phép toán logic (AND, OR, XOR, NOT) và các phép toán số học (cộng, trừ, nhân, chia)
- + Thực hiện việc chuyển dữ liệu

5.1.2 Bộ điều khiển - CU

Bộ điều khiển chịu trách nhiệm liên quan đến việc giải mã và thực thi các lệnh được lấy ra từ trong bộ nhớ lệnh bằng cách đưa ra các tín hiệu điều khiển và định thời cho bộ ALU và các khối phần cứng khác.

5.1.3 Thanh ghi - Register

Thanh ghi là nơi mà bộ vi xử lý dùng để lưu trữ các giá trị (dữ liệu hoặc địa chỉ) để đưa vào CPU xử lý hoặc chứa các kết quả do CPU trả về. Bộ vi xử lý dùng các thanh ghi để lưu trữ dữ liệu tạm thời trong quá trình thực hiện lệnh. Có rất nhiều loại thanh ghi khác nhau trong CPU như thanh ghi lệnh, thanh ghi dữ liệu, thanh ghi đa năng. Các thanh ghi có thể được truy cập bằng các câu lệnh ngôn ngữ máy (hợp ngữ) thường có chức năng hỗ trợ CPU trong quá trình tính toán. Các thanh ghi điều khiển và các thanh ghi trạng thái được CU dùng để điều khiển việc thực hiện chương trình thì lập trình viên không thể can thiệp, thay đổi được.

Các trạng thái hoạt động cơ bản của CPU

Nạp lệnh

Giải mã lệnh

Thực hiện lệnh

Trả về kết quả

5.1.4 Lịch sử

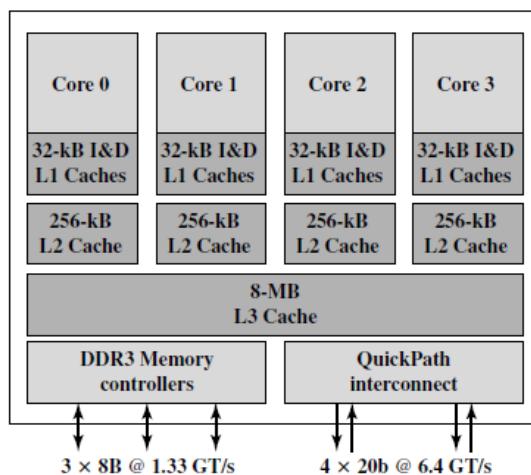
Lịch sử các dòng CPU của Intel

- + 1970 bộ CPU 4004 (4 bit) của Intel trên 1 chip đầu tiên ra đời
- + 1972 CPU Intel 8008 (8 bit)
- + 1974 CPU 8080, 1978 CPU 8086 (16 bit)
- + 1979 CPU 8088 (8 bit)
- + 1981 máy tính IBM PC đầu tiên ra đời trên cơ sở CPU Intel 8088 và hệ điều hành MS DOS

Lịch sử các dòng CPU của Intel

- + 1982 CPU 80286 (16 bit)
- + 1985 CPU 80386 (32 bit), 1989 CPU 80486 . . . (xuất hiện khái niệm đa nhiệm)
- + 1993 Pentium
- + 1997 CPU Pentium II
- + 1999 CPU Pentium III
- + 2000 CPU Pentium IV
- + 2002 Công nghệ siêu phân luồng cho Pentium IV → tăng hiệu suất hoạt động máy tính lên 25%
- + 2003 Pentium IV Extreme Edition
- + 2005 Dual Core
- + 2006 Core 2 Duo → tăng thêm 40% hiệu suất hoạt động

- + Đầu năm 2007, Core 2 Quad với số lõi thực sự là 4
- + Cuối năm 2008, Core i7 ra đời với công nghệ Turbo Boost (hình 5.2)
- + Cuối năm 2009, Core i5
- + Đầu năm 2010, Core i3



Hình 5.2 Sơ đồ khái niệm về bộ vi xử lý Core i7 thế hệ đầu tiên

Các đặc trưng dòng Core i7 (thế hệ đầu tiên)

Tập trung sức mạnh xử lý như chơi game, quản lý đa phương tiện, và hỗ trợ các phần mềm ứng dụng xử lý đa luồng

- + 4 lõi (core) vật lý
- + Bộ nhớ đệm (cache) 3 cấp
- + Sử dụng 2 công nghệ của Intel : Turbo-Boost và Hyper-Threading.
- + Turbo-Boost: tự động điều chỉnh xung nhịp của từng lõi độc lập phù hợp với nhu cầu xử lý → tăng hiệu suất lên 20%
- + Hyper-Threading: cung cấp 2 luồng (thread) xử lý trên mỗi lõi → nhân đôi số tác vụ mà bộ vi xử lý có thể thực thi
- + Ví dụ Turbo-Boost cho Core i7 920XM (hình 5.3)

4 lõi hoạt động: 2.26 GHz

2 lõi hoạt động: 3.06 GHz

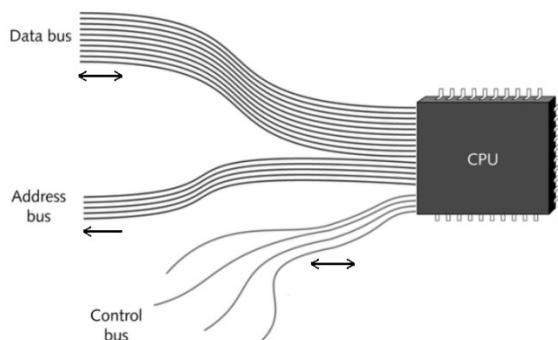
1 lõi hoạt động: 3.20 GHZ



Hình 5.3 Cách hoạt động của Intel Core i

5.2 GIAO TIẾP GIỮA CPU VÀ NGOẠI VI

5.2.1 Phân loại bus



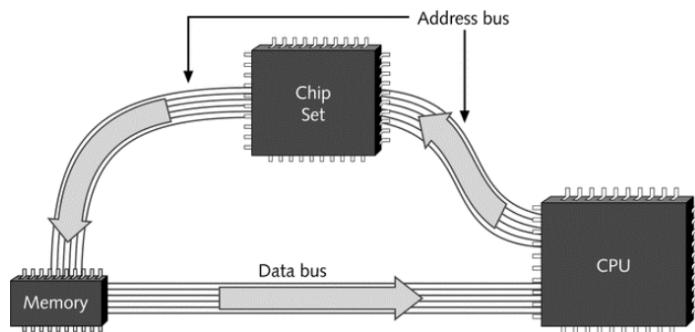
Hình 5.4 Các loại bus

Bus dữ liệu (data bus) : truyền dữ liệu giữa các thành phần trong hệ thống (2 chiều)

Bus địa chỉ (address bus) : để CPU xác định các thành phần trong hệ thống (1 chiều)

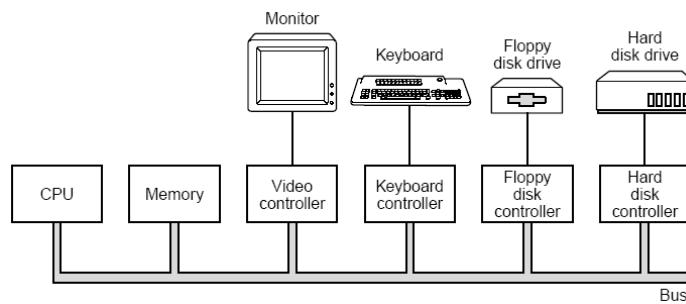
Bus điều khiển (control bus) : để CPU ra lệnh điều khiển cho các thành phần trong hệ thống (2 chiều)

Hình 5.5 miêu tả cách CPU đưa ra tín hiệu địa chỉ lên bus địa chỉ. Tín hiệu này thông qua chipset đi đến bộ nhớ tương ứng. Bộ nhớ, sau khi có địa chỉ từ bus địa chỉ, sẽ trả về dữ liệu cho CPU.



Hình 5.5 Cách CPU truy xuất dữ liệu thông qua chipset

5.2.2 Mô hình tổng quát

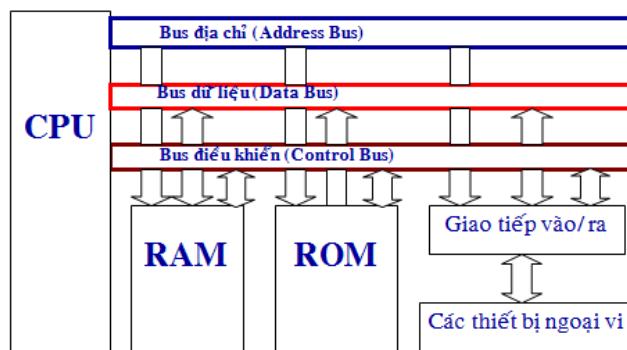


Hình 5.6 Mô hình tổng quát

Bus địa chỉ bao gồm các tín hiệu dùng để chuyển tải địa chỉ (thường được ký hiệu là A ví dụ CPU có 20 tín hiệu địa chỉ thì được ký hiệu từ A0 đến A19). Khi đọc/ghi bộ nhớ (hoặc I/O), CPU đưa ra trên Bus này địa chỉ của bộ nhớ (hoặc I/O) cần đọc/ghi. Như vậy, số lượng tín hiệu địa chỉ sẽ quyết định không gian bộ nhớ (tức là số lượng ô nhớ) mà CPU có thể định vị được. Thí dụ: CPU 8088/8086 có 20 bit tín hiệu địa chỉ thì không gian bộ nhớ của CPU này là $2^{20} = 1M$ ô nhớ, lưu ý rằng các CPU của họ 80x86 định vị theo byte nên không gian bộ nhớ của CPU này là 1Mbytes, tương tự CPU Pentium II có 36 tín hiệu địa chỉ thì không gian bộ nhớ của nó là 64Gbytes.

Bus dữ liệu gồm các tín hiệu dùng để chuyển tải dữ liệu (thường được ký hiệu là D). Số tín hiệu dữ liệu quyết định số bit dữ liệu mà CPU có thể xử lý cùng một lúc. Lưu ý rằng các tín hiệu dữ liệu là hai chiều vì CPU có thể đọc/ghi dữ liệu từ bộ nhớ hoặc I/O.

Bus điều khiển dùng để điều khiển hoạt động của hệ thống như các tín hiệu /WR (Write) để báo hiệu CPU đọc dữ liệu, /RD (Read) để báo hiệu CPU ghi dữ liệu, Ready dùng để báo cho CPU biết bộ nhớ (hoặc I/O) sẵn sàng quá trình trao đổi dữ liệu... Do đó, Bus điều khiển cũng phải là hai chiều.



Hình 5.7 Sơ đồ bus trong hệ thống máy tính

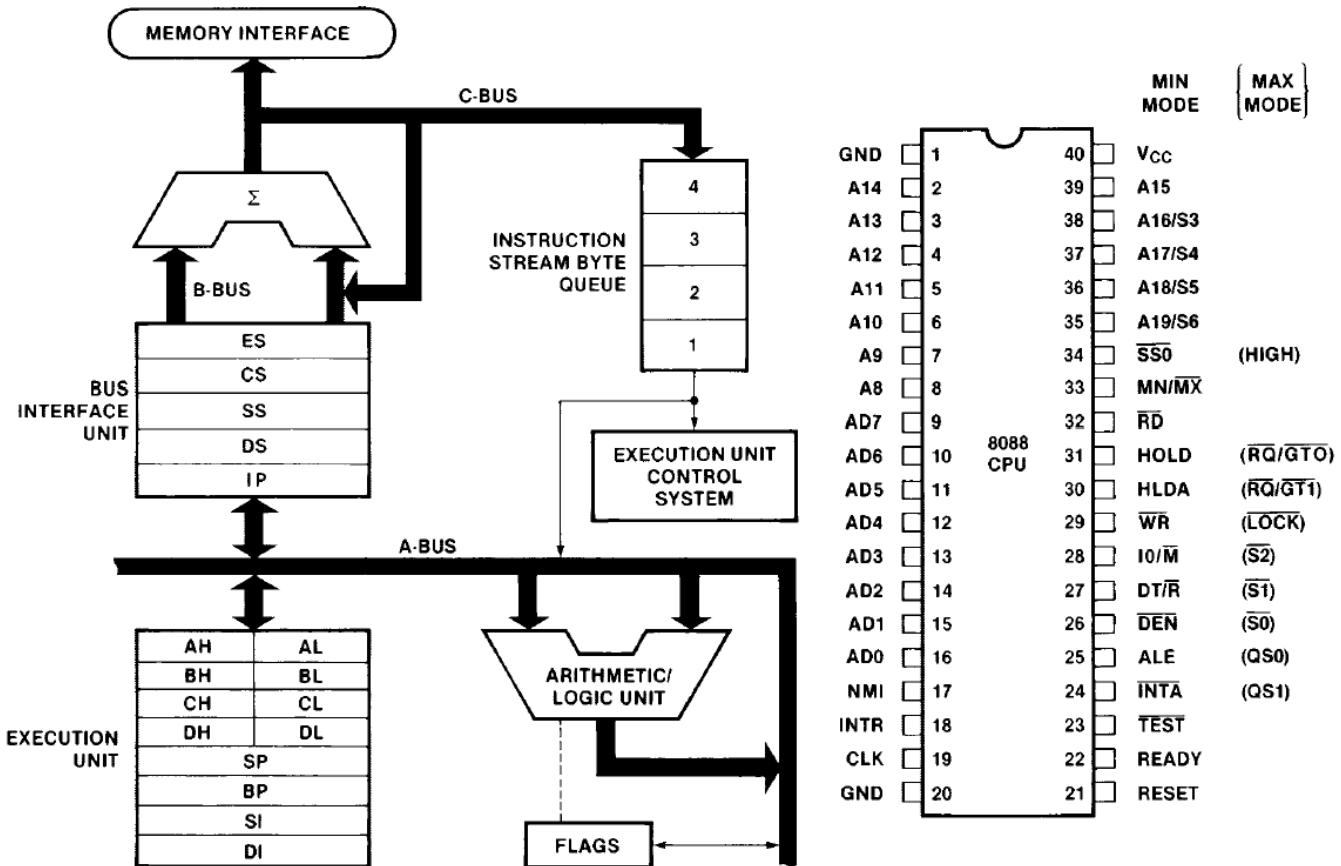
Bộ ghép nối vào ra cho phép ghép nối hệ thống với các thiết bị vào/ ra (I/O) như màn hình, bàn phím, chuột, ổ đĩa... thông qua các địa chỉ cổng vào/ ra (Port) (hình 5.7).

5.3 CẤU TRÚC CỦA VI XỬ LÝ 8088/ 8086

5.3.1 Các đặc điểm chính

Bus địa chỉ có 20 bit, do đó không gian bộ nhớ mà CPU này có thể định vị được là 2^{20} ô nhớ. Trong các CPU của Intel, mỗi ô nhớ có độ dài là 8 bit, nên không gian bộ nhớ tương ứng của CPU 8088/ 8086 là **1 Mega bytes**.

CPU 8088/8086 có các thanh ghi bên trong 16 bit, tổng cộng có 14 thanh ghi. Trong sơ đồ bên dưới các thanh ghi đa năng AX, BX, CX, DX được trình bày ở dạng hai thanh ghi 8 bit tương ứng (hình 5.8).



Hình 5.8 Sơ đồ khái niệm của CPU 8088/8086

Xuất phát từ việc các thanh ghi bên trong CPU là 16 bit mà bus địa chỉ của nó lại là 20 bit. Điều này có nghĩa là CPU sẽ dùng hai thanh ghi 16 bit để tạo thành một địa chỉ 20 bit. Địa chỉ 20 bit được đưa ra trên Bus địa chỉ được gọi là địa chỉ vật lý, còn địa chỉ được ký hiệu bằng hai thanh ghi 16 bit (segment : offset) để hình thành địa chỉ vật lý được gọi là địa chỉ logic. Cách chuyển đổi từ địa chỉ logic sang địa chỉ vật lý được tính như sau:

$$\text{Địa chỉ vật lý} = \text{Segment} \times 16 + \text{Offset} = \text{Trị giá 20 bit}$$

Thí dụ tìm địa chỉ vật lý từ các địa chỉ logic sau:

1234_h:0005_h

$$\text{Địa chỉ vật lý} = 1234 \times 16 + 0005 = 12345$$

Từ đó, ta có thể nhận thấy :

- + Một địa chỉ logic chỉ cho duy nhất một địa chỉ vật lý nhưng một địa chỉ vật lý có thể có nhiều địa chỉ logic.

+ Trong CPU 8088/8086 sẽ có các thanh ghi được dùng làm segment (gọi là các thanh ghi đoạn) và các thanh ghi dùng là offset trong quá trình tính toán địa chỉ.

+ Phạm vi của vùng nhớ ứng với một trị giá segment không đổi và offset thay đổi là 64 KB, đây được gọi là kích thước của một đoạn (segment).

* Các thanh ghi đoạn

Nói chung, khi hoạt động thì thường tồn tại 3 vùng nhớ trong một chương trình: Vùng nhớ chứa mã lệnh của chương trình, vùng nhớ chứa dữ liệu cho chương trình và một vùng nhớ mà người lập trình ngôn ngữ cấp cao ít khi quan tâm đến là vùng nhớ dùng cho ngăn xếp dùng cho các mục đặc biệt như gọi chương trình con, trả về từ chương trình con...

Trong CPU 8088/ 8086 có 4 thanh ghi đoạn bao gồm:

+ CS (Code Segment) là thanh ghi đoạn mã lệnh, trả đến vùng nhớ chứa mã lệnh của chương trình.

+ DS (Data Segment) là thanh ghi đoạn dữ liệu, trả đến vùng nhớ chứa dữ liệu của chương trình.

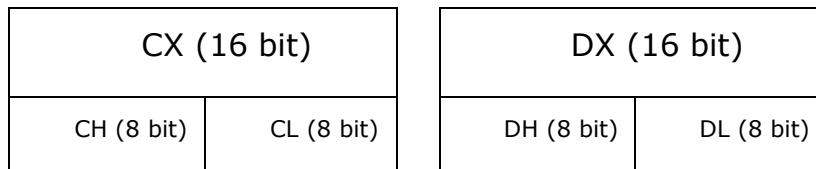
+ ES (Extra Segment) là thanh ghi đoạn dữ liệu phụ, trả đến vùng nhớ chứa dữ liệu của chương trình.

+ SS (Stack Segment) là thanh ghi đoạn ngăn xếp.

Khi hoạt động thì các đoạn có thể nằm riêng lẻ hoặc chồng lấp lên nhau.

Các thanh ghi đa năng trong CPU 8088/8086 gồm AX, BX, CX, và DX có thể chứa nhiều dạng dữ liệu khác nhau. Điều đặc biệt là các thanh ghi này có thể tách ra thành hai thanh ghi 8 bit hoạt động độc lập với nhau. Trong đó, 8 bit cao được gọi tên là H và 8 bit thấp có tên là L. Ví dụ thanh ghi AX tách thành hai thanh ghi 8 bit là AH và AL.

AX (16 bit)		BX (16 bit)	
AH (8 bit)	AL (8 bit)	BH (8 bit)	BL (8 bit)

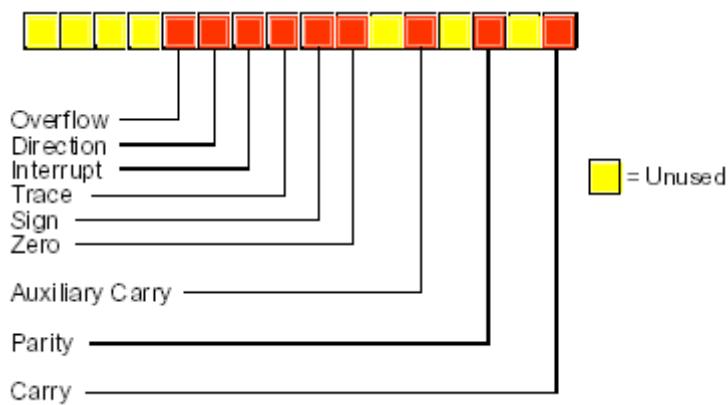


Tuy nhiên, các thanh ghi này còn có chức năng đặc biệt riêng:

- + Thanh ghi AX (Accumulator Register - thanh ghi tích lũy): Dùng trong các phép nhân chia, chứa dữ liệu cho các lệnh in, out.
- + Thanh ghi BX (Base Register - thanh ghi cơ sở): Dùng trong chế độ địa chỉ cơ sở.
- + Thanh ghi CX (Count register - thanh ghi đếm): Dùng để đếm số lần lặp trong các lệnh lặp.
- + Thanh ghi DX (Data Register - thanh ghi dữ liệu): Dùng để chứa địa chỉ cho các lệnh in/ out và làm toán hạng trong các lệnh nhân, chia 16 bit. (DX chứa 16 bit cao).

* Các thanh ghi con trỏ và chỉ số

- + Thanh ghi chỉ số nguồn SI (Source Index) và thanh ghi chỉ số đích DI (Destination Index): Các thanh ghi này có thể được dùng như là các con trỏ để truy xuất gián tiếp đến bộ nhớ. Ngoài ra, các thanh ghi này còn có thể được trong các lệnh chuỗi khi truy xuất đến các chuỗi ký tự.
- + Thanh ghi con trỏ lệnh IP (Instruction Pointer): Trỏ đến lệnh tiếp theo mà CPU sẽ thực hiện, địa chỉ đầy đủ là CS: IP.
- + Thanh ghi con trỏ ngăn xếp SP (Stack Pointer): Trỏ đến ô nhớ nằm trong vùng ngăn xếp, địa chỉ đầy đủ là SS:SP.
- + Thanh ghi con trỏ cơ sở BP (Base Pointer): Được dùng để truy xuất gián tiếp đến bộ nhớ.
- + Thanh ghi cờ (hình 5.9): Thanh ghi cờ bên trong CPU có nhiều chức năng rất quan trọng như phản ánh kết quả sau khi tính toán, thể hiện trạng thái hoạt động của CPU...



Hình 5.9 Thanh ghi cờ

- + Cờ tràn (Overflow Flag - OF): OF = 1 khi kết quả vượt quá khả năng lưu trữ của toán hạng đích (Destination operand) khi xem là số có dấu.
- + Cờ hướng (DirectionFlag - DF): Dùng cho các lệnh chuỗi, khi DF = 0 thì các CPU 80x86 xử lý các phần tử chuỗi theo chiều tăng của địa chỉ. Khi DF = 1 thì xử lý theo chiều ngược lại.
- + Cờ ngắt (Interrupt flag - IF): Cho phép các ngắt che được tác động.
- + Cờ bẫy (Trace Flag - TF): Cho phép chế độ chạy từng bước (dùng cho quá trình gỡ rối (Debug)...).
- + Cờ dấu (Sign Flag - SF): Phản ánh kết quả tính toán dương hay âm. SF = 1 khi kết quả âm
- + Cờ Zero (Zero Flag - ZF): Phản ánh kết quả có bằng Zero hay không. ZF = 1 khi kết quả bằng Zero.
- + Cờ phụ (Auxiliary Flag - AF): Dùng cho các phép toán BCD (Binary Coded Decimal) đặc biệt, trong khi lập trình ít khi sử dụng đến cờ này.
- + Cờ parity (Parity Flag PF): Phản ánh tính chẵn lẻ của kết quả. PF = 1 khi kết quả là số chẵn bit 1.
- + Cờ nhớ (Carry Flag – CF): CF = 1 khi có nhớ hoặc muợn ở MSB của kết quả.

5.4 CÁCH MÃ HÓA LỆNH CỦA 8088/ 8086

Lệnh của vi xử lý được ghi bằng các ký tự ở dạng mã gợi nhớ để thuận tiện cho người lập trình. Đối với vi xử lý thì lệnh tồn tại ở dạng nhị phân (ở dạng các trị giá 0, 1), dạng này còn được gọi là mã máy. Một lệnh có độ dài một hoặc vài byte tùy thuộc vào bộ vi xử lý.

Trong vi xử lý 8088/ 8086 một lệnh có độ dài từ 1 đến 6 byte. Chúng ta sẽ khảo sát lệnh Mov, đây là lệnh đơn giản nhất và thường hay được sử dụng nhất trong vi xử lý. Mã gợi nhớ và mã máy của lệnh MOV được quy định như hình 4.10.

MOV = Move:	76543210	76543210	76543210	76543210
Register/Memory to/from Register	100010 dw	mod reg r/m		
Immediate to Register/Memory	1100011 w	mod 0 0 0 r/m	data	data if w = 1
Immediate to Register	1011 wreg	data	data if w = 1	
Memory to Accumulator	1010000 w	addr-low	addr-high	
Accumulator to Memory	1010001 w	addr-low	addr-high	
Register/Memory to Segment Register	1000111 0	mod 0 reg r/m		
Segment Register to Register/Memory	1000111 00	mod 0 reg r/m		

Hình 5.10 Mã gợi nhớ và mã máy của lệnh MOV

Trong đó:

- + Bit d chỉ hướng đi của dữ liệu, nếu d = 1 thì dữ liệu đi đến thanh ghi cho bởi ba bit của Reg, d = 0 thì dữ liệu đi ra từ ba bit cho bởi Reg.
- + Bit w xác định số byte được truyền trong lệnh MOV, nếu w = 0 thì truyền một byte không thì truyền một word.
- + Hai bit mod và r/m được dùng để chỉ ra chế độ địa chỉ cho các toán hạng của lệnh. Cách mã hóa của các chế độ địa chỉ được trình bày trong bảng sau:

Mod R/m	00	01	10	11	
				w=0 w=1	
000	[BX] + [SI]	[BX] + [SI] + d8	[BX] + [SI] + d16	AL	AX

001	[BX] + [DI]	[BX] + [DI] + d8	[BX] + [SI] + d16	CL	CX
010	[BP + [SI]]	[BP + [SI]] + d8	[BP + [SI]] + d16	DL	DX
011	[BP + [DI]]	[BP + [DI]] + d8	[BP + [DI]] + d8	BL	BX
100	[SI]	[SI] + d8	[SI] + d16	AH	SP
101	[DI]	[DI] + d8	[DI] + d16	CH	BP
110	d16	[BP] + d8	[BP] + d16	DH	SI
111	[BX]	[BX] + d8	[BX] + d16	BH	DI

Các thanh ghi được gán theo bảng sau:

16-Bit (w = 1)	8-Bit (w = 0)	Segment
000 AX	000 AL	00 ES
001 CX	001 CL	01 CS
010 DX	010 DL	10 SS
011 BX	011 BL	11 DS
100 SP	100 AH	
101 BP	101 CH	
110 SI	110 DH	
111 DI	111 BH	

Ví dụ tìm mã máy từ mã gợi nhớ:

Mov AL, [SI]

Ý nghĩa của lệnh này là đưa nội dung của ô nhớ có địa chỉ logic là DS: SI vào thanh ghi AL. Nên lệnh này có dạng:

1	0	0	0	1	0	d	w	Mod(2 bit)	Reg (3 bit)	R/m (3 bit)
1	0	0	0	1	0	1	0	00	000	100

Lệnh này di chuyển dữ liệu 1 byte (w = 0), dữ liệu đi đến thanh ghi (d = 1), thanh ghi tham gia vào lệnh là AL (reg = 000), toán hạng [SI] được xác định với mod = 00 và r/m = 100. Nên mã máy của lệnh này là 8A04H.

Một ví dụ khác

Mov [BX + 9], AL

Có mã máy là 884709H.

CÂU HỎI VÀ BÀI TẬP

1. Trong CPU, thành phần nào quan trọng nhất? Tại sao?
2. Công nghệ siêu phân luồng là gì? Giải thích chi tiết?
3. Công nghệ Turbo Boost là gì? Giải thích chi tiết?
4. Có mấy loại bus?
5. Phân loại các thanh ghi trong vi xử lý 8088/8086?

BÀI 6. GIỚI THIỆU HỢP NGỮ

6.1 HỢP NGỮ VÀ TRÌNH BIÊN DỊCH

Một cách tổng quát, ta có thể phân loại các ngôn ngữ lập trình ra làm 2 dạng: ngôn ngữ lập trình cấp cao và ngôn ngữ lập trình cấp thấp. Các ngôn ngữ lập trình cấp cao như C, Java ... sử dụng các tập lệnh mà trong đó mỗi lệnh bao gồm nhiều lệnh ngôn ngữ máy. Mỗi lệnh ngôn ngữ máy lại chỉ tương đương 1 lệnh trong ngôn ngữ lập trình cấp thấp như hợp ngữ. Như vậy, hợp ngữ là ngôn ngữ lập trình gần với ngôn ngữ máy nhất.

Các ưu điểm cơ bản của hợp ngữ:

- + Dễ dàng điều khiển các yêu cầu của phần cứng
- + tạo ra các chương trình nhỏ và thực thi nhanh

Tuy nhiên, khuyết điểm lớn nhất chính là **độ phức tạp** khi lập trình thực hiện một số chức năng cơ bản ví dụ các phép toán số học so với ngôn ngữ cấp lập trình cấp cao. Điều này làm cho lập trình viên mất nhiều thời gian hơn.

Chương trình hợp ngữ phải qua giai đoạn biên dịch để chuyển sang ngôn ngữ máy.

- + Trình biên dịch (assembler) : là phần mềm biên dịch (hay chuyển) một chương trình viết bằng hợp ngữ sang ngôn ngữ máy.
- + Chương trình nguồn (source program): là chương trình viết bằng hợp ngữ có đuôi là *.ASM.
- + Chương trình đích (object program) là chương trình ở dưới dạng ngôn ngữ máy có đuôi là *.OBJ.
- + Chương trình thực thi có đuôi *.COM hoặc *.EXE.

Sau đây ta sẽ bước vào một số khái niệm cơ bản.

6.2 CHÚ THÍCH

Đây là thành phần quan trọng trong bất kỳ ngôn ngữ lập trình nào. Nó giúp chúng ta có thể đọc và hiểu chương trình dễ dàng hơn khi debug. Chú thích dùng trong hợp ngữ sử dụng dấu chấm phẩy (;). Trình biên dịch sẽ xem tất cả các ký tự sau dấu chấm phẩy là chú thích và sẽ bỏ qua khi biên dịch. Chú thích có thể chứa bất kỳ ký tự nào. Chú thích có thể đứng riêng 1 hàng hay đứng chung trong đoạn code (sau dấu chấm phẩy).

6.3 TỪ KHÓA

Trong bất kỳ ngôn ngữ lập trình nào đều có các từ khóa. Một số từ khóa tiêu biểu như:

- + Lệnh: MOV, ADD..
- + Chỉ dẫn: END, SEGMENT...
- + Các toán tử: FAR, SIZE...

6.4 THỂ HIỆN GIÁ TRỊ

Để chỉ ra một giá trị số theo hệ thập lục phân ta dùng chữ "H" ở phía sau con số. Theo qui ước một số hệ thập lục phân luôn bắt đầu bằng một chữ số thập phân từ $0 \rightarrow 9$, do vậy ta sẽ ghi giá trị AB₁₆ là 0ABH.

Để chỉ ra một giá trị số theo hệ nhị phân ta dùng chữ "B" ở phía sau con số. Ví dụ: 1101₂ sẽ ghi là 1101B.

Đối với số thập phân không cần chữ phía sau. Ví dụ: 79₁₀ sẽ ghi là 79.

6.5 MÃ ASCII

Để biểu diễn ký tự thuận tiện và thống nhất, mã ASCII được xem là mã chuẩn để hiển thị ký tự lên màn hình máy tính. Sau đây là bảng mã:

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	\$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[END OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

6.6 SEGMENT VÀ ĐỊNH ĐỊA CHỈ

Một segment là một vùng nhớ cụ thể trong chương trình. Một segment trong thực tế có thể lên đến 64KB và một chương trình có thể có nhiều segment. Có 3 segment chính trong một chương trình là:

- + Code segment: chứa các lệnh cần thực hiện
- + Data segment: chứa dữ liệu của chương trình, hằng số, vùng hoạt động
- + Stack segment: chứa các dữ liệu, địa chỉ mà ta cần lưu giữ tạm thời trong quá trình tính toán. Nó chính là bộ nhớ hỗ trợ dạng LIFO.

6.7 CÁC THANH GHI ĐA NĂNG

Các thanh ghi đa năng AX, BX, CX, DX cho phép định địa chỉ theo từ (2 byte) hoặc byte. Ví dụ thanh ghi AX (2 byte) có phần byte cao là AH và byte thấp là AL. Tương tự cho các thanh ghi còn lại.

6.8 THANH GHI CỜ

Thanh ghi cờ có 16 bit nhưng chỉ 9 bit được sử dụng để chỉ ra trạng thái hiện tại của hệ thống.

- + Cờ OF: cờ tràn cho biết kết quả có tràn hay không
- + Cờ DF: cờ hướng xác định hướng di chuyển dữ liệu từ phải hay trái
- + Cờ IF: cờ ngắt khi có ngắt ngoài
- + Cờ TF: cờ bẫy cho phép CPU hoạt động theo từng bước nhằm debug
- + Cờ ZF: cờ zero cho biết kết quả phép toán số học/ so sánh có bằng zero hay không
- + Cờ AF: cờ nhớ phụ chứa bit nhớ
- + Cờ PF: cờ parity kiểm tra số bit 1 là chẵn hay lẻ trên dữ liệu
- + Cờ CF: cờ nhớ áp dụng cho phép toán trên số nhị phân không dấu

6.9 QUY TẮC ĐẶT TÊN BIẾN

- + Ký tự chữ hoa hoặc thường (hợp ngữ không phân biệt chữ hoa/thường) từ A → Z
- + Ký tự số 0 → 9 nhưng không được đứng đầu
- + Các ký tự đặc biệt ?, _, \$, @ ...

Ký tự đứng đầu của tên biến phải là ký tự chữ hoặc ký tự đặc biệt. Chiều dài của tên biến tối đa là 31 ký tự. Ví dụ: COUNT, LINE15, \$E10.

6.10 KIỂU DỮ LIỆU

Định nghĩa byte	DB
Định nghĩa từ (word)	DW
Định nghĩa từ kép (double word)	DD
Định nghĩa chuỗi ký tự	DB '<chuỗi ký tự>'

6.11 KHAI BÁO KÍCH THƯỚC

Kích thước chương trình gồm đoạn mã máy và dữ liệu được xác định bằng chỉ dẫn .MODEL <kiểu>

+ TINY: Mã lệnh và dữ liệu nằm chung. Đây là dạng chương trình .COM.

+ SMALL: Mã lệnh và dữ liệu nằm riêng ở những vùng khác nhau. Đây là chương trình dạng .EXE nhỏ nhất.

Ngoài ra còn có các dạng khác như: MEDIUM, COMPACT, LARGE, HUGE, FLAT. Trong nội dung môn học này chỉ cần sử dụng kiểu SMALL là đủ.

Sau đó ta phải khai báo kích thước STACK theo dạng:

.STACK kích thước

Ví dụ:

.STACK 100H ; khai báo bộ nhớ stack gồm 256 byte

.STACK ; khai báo bộ nhớ stack gồm 1024 byte (default)

6.12 SO SÁNH GIỮA .COM VÀ .EXE

+ Đặc điểm của tập tin .COM

Chỉ có duy nhất một đoạn. Mã lệnh (code), dữ liệu (data) và bộ nhớ stack đều dùng chung một đoạn.

Kích thước tối đa của tập tin là 64 KB.

Tập tin dạng .COM nạp vào bộ nhớ và được thực hiện nhanh hơn tập tin dạng .EXE, nhưng nó chỉ áp dụng cho các chương trình nhỏ, chỉ có thể gọi các chương trình con dạng gần, muốn xây dựng các chương trình lớn ta phải viết dưới dạng .EXE.

Tập tin dạng .COM có thể liên kết từ các phần tách rời nhau, mỗi phần tách riêng trên những tập tin khác nhau. Tất cả những phần này phải có cùng tên đoạn, cùng tên với phần chính, phần chính phải chứa điểm bắt đầu của chương trình và phải liên kết trước.

+ Đặc điểm của tập tin .EXE

Chương trình có thể được khai báo nhiều đoạn khác nhau. Mỗi chương trình có thể có nhiều đoạn chương trình, nhiều đoạn dữ liệu.

Kích thước của tập tin có thể lớn tùy ý tuỳ thuộc vào kích thước bộ nhớ máy tính. Thường dạng .EXE dùng để xây dựng các chương trình lớn có kích thước lớn hơn 64 KB.

Tập tin có một header ở đầu tập tin. Header này chứa các thông tin điều khiển về tập tin để DOS có thể nạp nó vào bộ nhớ và thực hiện. Kích thước header phụ thuộc vào số các lệnh trong chương trình cần để định vị lại các địa chỉ đoạn khi chương trình được nạp, nhưng nó luôn là bội số của 256 byte.

6.13 KẾT THÚC THỰC THI

Lệnh INT 21H là lệnh ngắt của DOS sử dụng mã chức năng trong thanh ghi AH để xác định mục đích cần thực hiện. Một số chức năng khác của INT 21H như nhập từ bàn phím, quản lý màn hình, xuất nhập đĩa và xuất ra máy in.

Để kết thúc chương trình một cách bình thường, ta có thể sử dụng đoạn code sau:

```
MOV AX, 4C00H      ; kết thúc bình thường bằng cách chuyển giá trị 4C0016 vào  
                  ; thanh ghi AX
```

```
INT 21H           ; trả về DOS thông qua ngắt
```

Phần tiếp theo sẽ đi vào tập lệnh của họ CPU 80x86. Tập lệnh của 80x86 khá phức tạp, do đó trong từng nhóm lệnh chỉ trình bày một số lệnh có tính cơ bản và thường được sử dụng trong lập trình hợp ngữ, khi hiểu được các lệnh này sinh viên có thể suy luận ra các lệnh khác tương ứng với từng nhóm lệnh.

CÂU HỎI VÀ BÀI TẬP

1. So sánh hợp ngữ với ngôn ngữ C. Ưu và khuyết điểm của từng loại?
2. Người ta thường sử dụng hợp ngữ trong trường hợp nào? Tại sao?

BÀI 7. LẬP TRÌNH ỨNG DỤNG

7.1 TẬP LỆNH CỦA 80X86

7.1.1 Nhóm lệnh di chuyển dữ liệu

Các lệnh di chuyển dữ liệu là những lệnh đơn giản nhất nên sẽ được trình bày đầu tiên.

+ Lệnh MOV

Cú pháp: MOV [đích], [nguồn]

Ý nghĩa: đưa giá trị chứa trong toán hạng nguồn (hoặc một giá trị cụ thể) vào trong toán hạng đích.

Ví dụ: MOV AL, 89H ; đưa giá trị 89_{16} vào trong thanh ghi AL

MOV BL, AL ; đưa giá trị chứa trong thanh ghi AL vào thanh ghi BL

+ Lệnh PUSH

Cú pháp: PUSH [nguồn]

Ý nghĩa : Đưa 1 word từ nguồn vào stack

Ví dụ: PUSH AX ; đưa giá trị thanh ghi AX vào bộ nhớ stack để lưu trữ

+ Lệnh POP

Cú pháp: POP [đích]

Ý nghĩa : Lấy 1 word từ stack đưa vào đích

Ví dụ: POP AX ; đưa giá trị từ đỉnh bộ nhớ stack vào thanh ghi AX

Lưu ý: stack là bộ nhớ LIFO (Last In First Out)

+ Lệnh PUSHF (Push Flag Register to the Stack)

Cú pháp: PUSHF

Ý nghĩa : Cắt thanh ghi cờ vào ngăn xếp. Giảm SP đi 2 ($SP = SP - 2$), đưa 8 bit thấp của thanh ghi cờ vào bộ nhớ.

+ Lệnh POPF

Cú pháp: POPF (pop Word from top of Stack to Flag Register) Lấy 1 từ từ đỉnh của ngăn xếp đưa vào thanh ghi cờ

Ý nghĩa : Đưa ô nhớ có địa chỉ cho bởi SS:SP vào 8 bit thấp của thanh ghi cờ và SS:SP+1 vào 8 bit cao của thanh ghi cờ. Sau đó tăng SP lên 2 ($SP = SP + 2$).

+ Lệnh XCHG (exchange)

Cú pháp: XCHG [toán hạng 1], [toán hạng 2]

Ý nghĩa : hoán đổi giá trị giữa 2 thanh ghi/bộ nhớ với nhau

Ví dụ: XCHG AL, AH ; *hoán đổi nội dung của 2 thanh ghi*

+ Lệnh LEA

Cú pháp: LEA [đích], [nguồn]

Ý nghĩa : nạp địa chỉ offset của nguồn cho thanh ghi đích

Ví dụ: LEA DX, TB ; *DX sẽ chứa địa chỉ offset của biến TB*

7.1.2 Nhóm lệnh logic và số học

+ Lệnh INC (increment)

Cú pháp : INC [toán hạng]

Ý nghĩa : để cộng thêm 1 vào nội dung của một thanh ghi hoặc một vị trí nhớ.

+ Lệnh DEC (decrement)

Cú pháp : DEC [toán hạng]

Ý nghĩa : để giảm bớt 1 khỏi một thanh ghi hoặc 1 vị trí nhớ.

+ Lệnh ADD

Cú pháp: ADD [đích], [nguồn]

Ý nghĩa : cộng 2 toán hạng

Ví dụ: ADD AL, CL ; $AL = AL + CL$

+ Lệnh SUB

Cú pháp: SUB [đích], [nguồn]

Ý nghĩa : trừ 2 toán hạng

Ví dụ: SUB AL, CL ; $AL = AL - CL$

+ Lệnh MUL

Cú pháp: MUL [toán hạng]

Ý nghĩa : Nếu toán hạng có 8 bit thì tích có giá trị 16 bit = AX = AL x [toán hạng]

Ví dụ: MOV BL, 12H

MOV AL,7H

MUL BL ; kết quả $AX = BL \times AL = 12_{16} \times 7_{16} = 7E_{16}$

; Nếu toán hạng là 16 bit thì tích có thị giá 32 bit được lưu ở

; hai thanh ghi DX-AX

+ Lệnh DIV

Cú pháp: DIV [toán hạng]

Ý nghĩa : Chia 2 số không dấu. Nếu toán hạng là 8 bit thì số bị chia là số không dấu 16 bit chứa trong AX, kết quả gồm 2 phần, thương chứa trong AL và số dư chứa trong AH. Nếu toán hạng là 16 bit thì số bị chia là số không dấu 32 bit chứa trong cặp thanh ghi DX-AX gồm 2 phần, thương chứa trong AX và số dư chứa trong DX.

Ví dụ: MOV BL, 3H

MOV AX, 7H

DIV BL ; kết quả $AL = \text{thương} = 2, AH = \text{số dư} = 1$

+ Lệnh AAM

Cú pháp: AAM

Ý nghĩa : chuyển sang mã BCD với giá trị nhị phân chứa trong thanh ghi AL, AAM chia AL cho 10 và lưu phần thương vào AL và phần dư vào AH.

Ví dụ: MOV AL, 9

MOV BL, 7

MUL BL ; cho kết quả AH = 00 và AL = 63

AAM ; điều chỉnh cho kết quả AH = 6 và AL = 3

+ Lệnh CMP

Cú pháp: CMP [tổng hợp 1], [tổng hợp 2]

Ý nghĩa : so sánh 2 tổng hợp

Ví dụ: CMP AL, CL ; Khi AL = CL thì CF = 0; ZF = 1

 ; Khi AL > CL thì CF = 0; ZF = 0

 ; Khi AL < CL thì CF = 1; ZF = 0

+ Lệnh AND

Cú pháp: AND [đích], [nguồn]

Ý nghĩa : Thực hiện phép AND theo từng bit của 2 tổng hợp

Ví dụ: MOV AL, 23H

AND AL, 0FH ; Kết quả AL = 03₁₆

+ Lệnh OR

Cú pháp: OR [đích], [nguồn]

Ý nghĩa : Thực hiện phép OR theo từng bit của 2 tổng hợp

+ Lệnh XOR

Cú pháp: XOR [đích], [nguồn]

Ý nghĩa : Thực hiện phép XOR theo từng bit của 2 tổng hợp

+ Lệnh NOT

Cú pháp: NOT [tổng hợp]

Ý nghĩa : Thực hiện phép đảo bit

+ Lệnh NEG

Cú pháp: NEG [toán hạng]

Ý nghĩa : Thực hiện phép lũy bù 2

7.1.3 Nhóm lệnh dịch chuyển

+ Lệnh SHL

Cú pháp: SHL [đích], 1

Ý nghĩa : Dịch [đích] sang trái 1 lần.

Cú pháp: SHL [đích], CL

Ý nghĩa : Dịch [đích] sang trái , số lần dịch chứa trong thanh ghi CL

+ Lệnh SHR

Cú pháp: SHR [đích], 1

Ý nghĩa : Dịch [đích] sang phải 1 lần.

Cú pháp: SHL [đích], CL

Ý nghĩa : Dịch [đích] sang phải , số lần dịch chứa trong thanh ghi CL

+ Lệnh SAR

Cú pháp: SAR [đích], 1

Ý nghĩa : Dịch phải số học 1 lần. Bit MSB giữ nguyên

Cú pháp: SAR [đích], CL

Ý nghĩa : Dịch phải số học, số lần dịch chứa trong thanh ghi CL

+ Lệnh ROL

Cú pháp: ROL [đích], 1

Ý nghĩa: Quay vòng sang trái 1 lần.

Cú pháp: ROL [đích], CL

Ý nghĩa: Quay vòng sang trái với số lần quay chứa trong thanh ghi CL

+ Lệnh ROR

Cú pháp: ROR [đích], 1

Ý nghĩa: Quay vòng sang phải 1 lần.

Cú pháp: ROR [đích], CL

Ý nghĩa: Quay vòng sang phải với số lần quay chứa trong thanh ghi CL

7.1.4 Nhóm lệnh nhảy

+ Lệnh JMP

Cú pháp: JMP [nhãn]

Ý nghĩa: Nhảy sang dòng lệnh có gán nhãn.

Ví dụ:

MOV AL,8

MOV CL,5

JMP TRU ; nhảy đến nhãn TRU và bỏ qua lệnh phía dưới

ADD AL,CL

TRU:

SUB AL,CL ; Kết quả là AL = 3, CL = 5

+ Lệnh LOOP

Cú pháp: LOOP <nhãn>

Ý nghĩa: CX = CX -1; Nếu CX ≠ 0 thì nhảy đến vị trí được đánh nhãn. Lặp lại đoạn chương trình cho đến khi CX = 0

Ví dụ:

MOV CX,10

MOV AL,0

MOV BL,1

LAP:

ADD AL, BL

LOOP LAP ; Kết quả: AL = 10

+ Lệnh LOOPNZ

Cú pháp: LOOPNZ <nhãn>

Ý nghĩa: CX = CX -1; Nếu CX ≠ 0 thì nhảy đến vị trí được đánh nhãn. Lặp lại đoạn chương trình cho đến khi CX =0 hoặc khi cờ ZF bằng 1.

+ Lệnh nhảy có điều kiện

Cú pháp: J<điều kiện> <nhãn>

Ý nghĩa: Lệnh nhảy có điều kiện trước tiên kiểm tra điều kiện, sau đó nhảy đến vị trí chỉ bởi nhãn nếu điều kiện được thoả hay tiếp tục thực hiện lệnh kế tiếp nếu điều kiện không được thoả.

Lệnh	Ý nghĩa	Cờ kiểm tra	Ghi chú
JB/JNAE	Nhảy nếu nhỏ hơn/không lớn hơn hay bằng	CF = 1	Áp dụng cho số không dấu
JBE/JNA	Nhảy nếu nhỏ hơn hay bằng/ không lớn hơn	CF = 1 hay ZF=1	Áp dụng cho số không dấu
JA/JNBE	Nhảy nếu lớn hơn/ không nhỏ hơn hay bằng	CF = 0 và ZF= 0	Áp dụng cho số không dấu
JAE/ JNB	Nhảy nếu lớn hơn hay bằng/ không nhỏ hơn	CF = 0	Áp dụng cho số không dấu
JE/JZ	Nhảy nếu bằng	ZF = 1	
JNE/JNZ	Nhảy nếu không bằng	ZF = 0	
JL/JNGE	Nhảy nếu nhỏ hơn/ không lớn hơn hay bằng	SF<>OF	Áp dụng cho số có dấu

JLE/JNG	Nhảy nếu nhỏ hơn hay bằng/ không lớn hơn	SF<>OF hay ZF=1	Áp dụng cho số có dấu
JG/JNLE	Nhảy nếu lớn hơn/ không nhỏ hơn hay bằng	SF=OF và ZF = 0	Áp dụng cho số có dấu
JGE/JNL	Nhảy nếu lớn hơn hay bằng/ không nhỏ hơn	SF = OF	Áp dụng cho số có dấu
JP	Nhảy nếu cờ chẵn lẻ được bật lên	PF = 1	
JNP	Nhảy nếu cờ chẵn lẻ không được bật lên	PF = 0	
JS	Nhảy nếu cờ dấu được bật lên	SF = 1	
JNS	Nhảy nếu cờ dấu không được bật lên	SF = 0	
JO	Nhảy nếu cờ tràn được bật lên	OF = 1	
JNO	Nhảy nếu cờ tràn không được bật lên	OF = 0	
JC	Nhảy nếu cờ nhớ được bật lên	CF = 1	
JNC	Nhảy nếu cờ nhớ không được bật lên	CF = 0	
JCXZ	Nhảy nếu CX bằng 0		

+ Lệnh lặp có điều kiện

Cú pháp: LOOP<điều kiện> <nhãn>

Ý nghĩa: Lệnh LOOP chỉ chấm dứt vòng lặp khi thanh ghi CX bằng 0. Muốn chấm dứt vòng lặp trước khi CX bằng 0 ta phải dùng các lệnh vòng lặp có điều kiện.

LOOPE: lặp khi bằng

LOOPZ: giống LOOPE

LOOPNE: lặp khi không bằng

LOOPNZ: giống LOOPNE

7.1.5 Lệnh INT 21H

INT 21h là lệnh gọi một chương trình con của hệ điều hành. Chương trình con này nhiều chức năng khác nhau tuỳ theo giá trị của thanh ghi AH. Sau đây là một số chức năng thông dụng:

+ Nhập ký tự từ bàn phím và hiển thị (AH=01H)

Chức năng AH =01h: thực hiện nhập một ký tự từ bàn phím, mã ASCII của ký tự chứa trong AL (hoặc AL=0 nếu là phím điều khiển).

Ví dụ:

MOV AH,01H ; chọn chức năng 01H

INT 21H ; nhập ký tự và chứa vào AL

+ Xuất ký tự ra màn hình (AH=02H)

Chức năng AH=02h: thực hiện xuất ký tự theo mã ASCII chứa trong thanh ghi DL ra màn hình.

Ví dụ:

MOV AH, 02H

MOV DL, 'A'

INT 21h ; Hiện chữ "A" ra màn hình

+ Xuất chuỗi ký tự ra màn hình (AH=09H)

Chức năng AH =09h: thực hiện xuất chuỗi ký tự có địa chỉ ô chứa trong thanh ghi DX (và địa chỉ đoạn chứa trong DS) ra màn hình. Chuỗi ký tự phải kết thúc bằng dấu \$.

Ví dụ:

```
.DATA
    THONGBAO DB      "Hello HUTECH$"
.CODE
    MOV     AX, @DATA
    MOV     DS, AX
    MOV     AH, 09H
    LEA     DX, THONGBAO
    INT     21h          ; Xuất chuỗi ký tự "Hello HUTECH" ra màn hình
```

7.2 XỬ LÝ MÀN HÌNH

7.2.1 Tổng quan màn hình

Màn hình là một lưới các ô được đánh địa chỉ và một con trỏ có thể được thiết lập ở bất kỳ vị trí nào trên màn hình. Một màn hình tiêu biểu ở chế độ văn bản (trên nền DOS) có 25 hàng (từ 0 → 24) và 80 cột (từ 0 → 79). Một số vị trí tiêu biểu:

Góc trái trên thì hàng =0, cột =0

Góc phải dưới thì hàng =24, cột =79

Giữa màn hình thì hàng =12, cột =39/40

Hệ thống cung cấp cho ta một không gian trong bộ nhớ gọi là vùng hiển thị video hay vùng đệm. Với chế độ văn bản vùng đệm cung cấp các trang màn hình từ 0 đến 3 cho màn hình 80 cột với các byte cho ký tự và thuộc tính ký tự (mỗi ký tự 2 byte).

7.2.2 Thiết lập con trỏ

AH=02H - INT 10H là chỉ thị ngắt dùng để thiết lập con trỏ. Ví dụ thiết lập con trỏ ở hàng 5 cột 15:

```
MOV     AH, 02H
MOV     BH, 00          ; trang 0
MOV     DH, 05          ; hàng 5
```

```
MOV    DL, 15          ; cột 15
INT    10H
```

7.2.3 Đọc vị trí con trỏ

AH=03H - INT 10H là chỉ thị ngắt dùng để xác định hàng, cột và kích thước con trỏ. Giá trị hàng và cột sẽ được lưu trữ trong thanh ghi DX (DH là hàng, DL là cột). Ví dụ:

```
MOV    AH, 03H
MOV    BH, 00          ; trang 0
INT    10H
```

Chỉ thị ngắt sẽ trả về dòng và cột của con trỏ trong thanh ghi DX.

7.2.4 Xóa và thiết lập màu cho màn hình

AH=02H - INT 10H là chỉ thị ngắt dùng để xóa màn hình. Ta có thể xóa 1 phần hoặc cả màn hình từ 1 vị trí bất kỳ và kết thúc ở vị trí có hàng và cột cao hơn.

AL = 00 xóa toàn bộ màn hình

BH = thuộc tính trong đó 4 bit cao là màu nền (background), 4 bit thấp là màu chữ (foreground) (4 bit theo thứ tự là: đậm nhạt - đỏ - xanh lá - xanh da trời)

CX = hàng cột bắt đầu

DX = hàng cột kết thúc

Ví dụ:

```
MOV    AX, 0600H ; xóa cả màn hình
MOV    BH, 71H      ; chữ xanh nền trắng
MOV    CX, 0000H ; bắt đầu từ góc trên trái
MOV    DX, 184FH ; đến góc dưới phải
INT    10H
```

7.2.5 Hiển thị ký tự với thuộc tính tại vị trí con trỏ

AH=09H - INT 10H là chỉ thị ngắn dùng để hiển thị ký tự đi kèm với thuộc tính và số lần lặp lại. Trong đó:

AL chứa mã ASCII của ký tự

BH = số trang

BL = thuộc tính

CX = số lần lặp lại

Ví dụ:

MOV AH, 09H

MOV AL, '-' ; dấu gạch giữa

MOV BH, 00 ; trang 0

MOV BL, 41H ; nền đỏ chữ xanh

MOV CX, 80 ; ghi 80 ký tự

INT 10H

7.3 THỦ TỤC

Thủ tục còn được gọi là chương trình con. Trong tất cả các ngôn ngữ lập trình đều có thủ tục và hàm. Thủ tục hoạt động tương đối độc lập. Nó được viết riêng cho các câu lệnh có tính chất lặp đi lặp lại và cũng làm cho chương trình chính dễ đọc hơn.

Thủ tục được viết theo định dạng sau

<Tên> PROC <kiểu>

<lệnh 1>

<lệnh 2>

<lệnh 3>

RET

<Tên> ENDP

Kiểu NEAR được xem là kiểu mặc định khi thủ tục có cùng đoạn với chương trình chính. Ngược lại kiểu FAR khi thủ tục ở khác đoạn với chương trình chính. Một thủ tục chính (MAIN) sẽ chứa lệnh của chương trình chính. Để thực hiện một chức năng nào đó, thủ tục MAIN hay chương trình chính gọi (**CALL**) một thủ tục con. Thủ tục con cũng có thể gọi một thủ tục con khác. Sau khi thực hiện xong chức năng thì thủ tục trở về thủ tục trước nó bằng lệnh **RET**.

CÂU HỎI VÀ BÀI TẬP

1. Viết chương trình thể hiện 3 câu sau lên màn hình (biết ký tự xuống dòng có giá trị 10 và ký tự trả về đầu dòng có giá trị 13)

Hello !

Day la truong HUTECH

Mon hoc Kien Truc May Tinh

2. Viết chương trình nhập một ký tự từ bàn phím và cho ra kết quả ký tự đứng trước và đứng sau theo thứ tự ABC. Ví dụ:

Nhập ký tự: B

Ký tự trước: A

Ký tự sau: C

3. Viết chương trình nhập một ký tự in hoa từ bàn phím và xuất ra kết quả chuỗi ký tự từ ký tự đó đến 'Z'.

Ví dụ:

Nhập ký tự: X

Chuỗi ký tự: XYZ

4. Viết chương trình nhập một chuỗi ký tự từ bàn phím. Sau khi nhập xong in chuỗi ký tự theo chiều ngược lại.

Ví dụ:

Nhập chuỗi ký tự: ABCDE

Chuỗi ký tự nhận được theo chiều ngược lại: EDCBA

5. Viết chương trình nhập vào 2 chữ số A và B. Xuất ra tổng và hiệu của 2 chữ số đó dưới dạng 1 chữ số (tổng và hiệu không vượt quá 9 và không âm).

6. Như bài tập 5 nhưng chỉ cho phép người dùng nhập số. Nếu người dùng nhập vào một ký tự không phải chữ số sẽ báo lỗi và bắt người dùng nhập lại.

7. Viết chương trình nhập 1 ký tự. In mã ASCII của ký tự đó ra màn hình theo dạng nhị phân.

8. Viết chương trình tính chu vi hình chữ nhật khi nhập vào chiều dài và chiều rộng. Giá trị nhập vào phải từ 2 chữ số trở lên.

9. Viết chương trình hiện chữ "X" ngay điểm giữa màn hình

10. Viết chương trình hiển thị chuỗi "DAI HOC KY THUAT CONG NGHE TP.HCM" ngay giữa màn hình với chữ và nền đổi màu liên tục.

11. Viết chương trình cho chữ "A" chạy xung quanh màn hình.

BÀI 8. QUẢN LÝ TIẾN TRÌNH

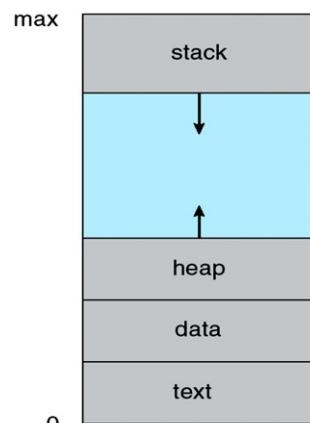
8.1 KHÁI NIỆM TIẾN TRÌNH

Một câu hỏi phát sinh trong thảo luận về các hệ điều hành liên quan đến việc gọi tất cả các hoạt động của CPU là cái gì. Một hệ thống theo lô thực thi các *công việc* (job), trong khi một hệ thống chia sẻ thời gian có các *chương trình người dùng* (user program), hoặc *nhiệm vụ* (task). Ngay cả trên hệ thống một người dùng như Microsoft Windows, người dùng cũng có thể chạy nhiều chương trình cùng một lúc chương trình xử lý văn bản, trình duyệt web và một gói phần mềm e-mail. Ngay cả khi người dùng chỉ thực hiện một chương trình tại một thời điểm, hệ điều hành vẫn cần phải hỗ trợ các hoạt động nội bộ đã được lập trình của nó, chẳng hạn như quản lý bộ nhớ. Trong nhiều khía cạnh, tất cả các hoạt động là tương tự, vì vậy chúng ta gọi tất cả chúng là *tiến trình* (process).

8.1.1 Tiến trình

Một cách không chính thức, như đã đề cập trước đó, *tiến trình là một chương trình đang thực thi*. Một tiến trình không chỉ mã chương trình, nó bao gồm cả các hoạt động hiện tại, được biểu diễn bởi giá trị của con trỏ chương trình (program counter) và nội dung các thanh ghi (resisters) của bộ xử lý. Tiến trình cũng có ngăn xếp (stack) tiến trình, trong đó chứa dữ liệu tạm thời (chẳng hạn như các tham số của hàm, địa chỉ trả lại, và các biến địa phương), và một vùng dữ liệu, trong đó có các biến chung. Tiến trình cũng có thể bao gồm một chồng (heap), đó là phần bộ nhớ được cấp phát động trong suốt thời gian chạy. Cấu trúc của một tiến trình trong bộ nhớ được thể hiện trong hình dưới.

Cần nhấn mạnh rằng chương trình tự nó không phải là tiến trình, chương trình là một thực thể thụ động, chẳng hạn như một tập tin thực thi được lưu trữ trên đĩa, trong khi tiến trình là một thực thể hoạt động. Chương trình sẽ trở thành tiến trình



khi tập tin thực thi được tải vào bộ nhớ. Hai kỹ thuật phổ biến để tải các tập tin thực thi là kích đúp vào biểu tượng đại diện cho tập tin thực thi hoặc nhập tên của tập tin thực thi trên dòng lệnh (như prog.exe hoặc a.com.)

Mặc dù hai tiến trình có thể sinh ra từ cùng một chương trình, chúng vẫn được coi là các chuỗi thực thi riêng biệt. Ví dụ, nhiều người dùng có thể chạy nhiều bản sao của một chương trình mail, hoặc cùng một người dùng có thể gọi nhiều bản sao của chương trình trình duyệt Web. Mỗi tiến trình trong số này là một tiến trình riêng biệt, và mặc dù các phần văn bản là tương đương, các phần dữ liệu, ch่อง, và ngăn xếp là khác nhau. Một điều phổ biến là một tiến trình có thể sinh ra nhiều tiến trình như nó cùng chạy.

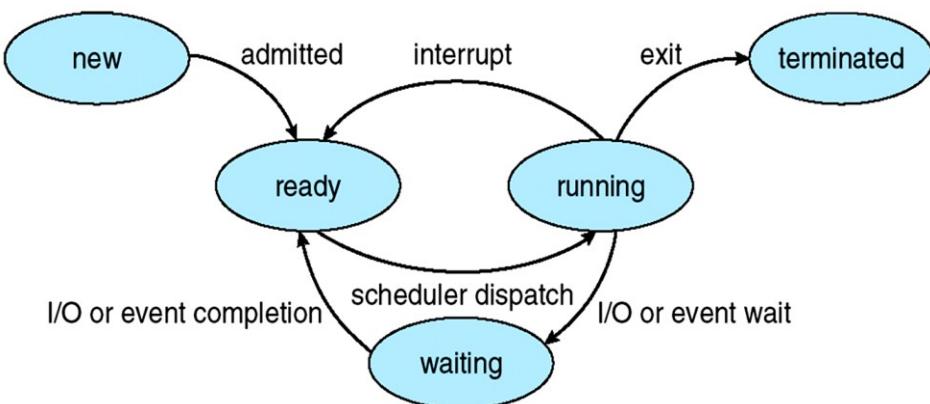
8.1.2 Trạng thái tiến trình

Khi một tiến trình được thực thi, nó thay đổi trạng thái. Trạng thái của một tiến trình được xác định một phần bởi các hoạt động hiện tại của tiến trình đó. Mỗi tiến trình có thể ở một trong các trạng thái sau đây:

- *New* : Tiến trình đang được tạo ra.
- *Running* : Các lệnh đang được thi hành.
- *Waiting* : Tiến trình đang chờ một sự kiện nào đó xảy ra (chẳng hạn như hoàn thành một I/O hoặc nhận một tín hiệu).
- *Ready* : Tiến trình đang chờ được cấp CPU.
- *Terminated* : Tiến trình đã kết thúc việc thực thi.

Các tên gọi trạng thái có thể khác nhau trên các hệ điều hành. Tuy nhiên, các trạng thái mà chúng đại diện được tìm thấy trên tất cả các hệ thống. Một số hệ điều hành cũng phân định trạng thái tiến trình mịn hơn. Điều quan trọng là tại một thời điểm chỉ có một tiến trình có thể chạy trên một CPU. Tuy nhiên, nhiều tiến trình có thể sẵn sàng và chờ đợi.

Sơ đồ trạng thái được



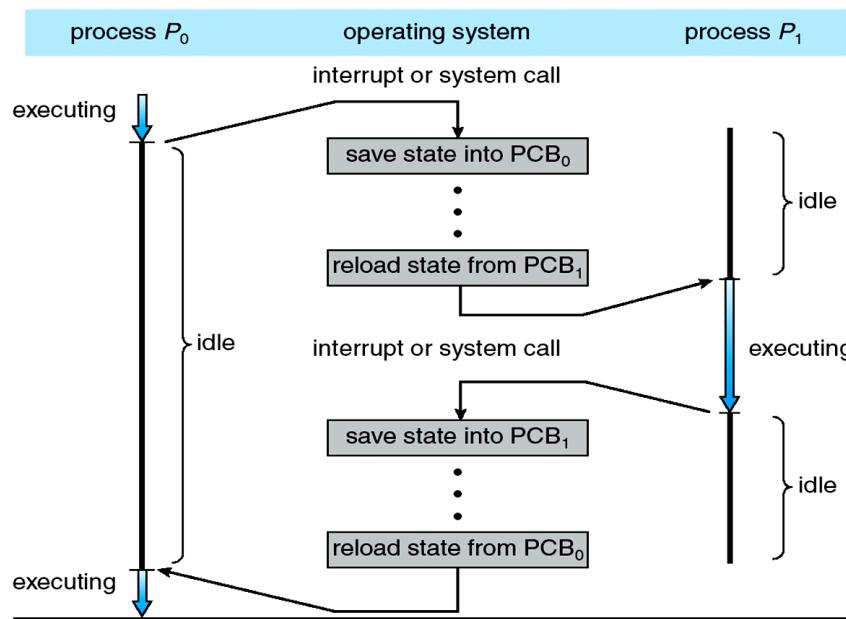
trình bày trong hình dưới.

8.1.3 Khởi quản lý tiến trình

Mỗi tiến trình được thể hiện trong hệ điều hành bởi một khối điều khiển tiến trình (PCB, process control block), còn được gọi là khối điều khiển nhiệm vụ. Một PCB được thể hiện trong hình bên. Nó chứa nhiều mẩu thông tin liên quan đến một tiến trình cụ thể, bao gồm :

- *Trạng thái tiến trình* : Trạng thái có thể là new, ready, running, waiting, terminated.
- *Con trỏ chương trình* : Con trỏ cho biết địa chỉ của lệnh tiếp theo được thực thi của tiến trình.
- *Các thanh ghi CPU* : Tùy thuộc vào kiến trúc máy tính, số lượng và chủng loại các thanh ghi sẽ khác nhau. Chúng bao gồm các thanh ghi tích lũy, thanh ghi chỉ số, con trỏ ngăn xếp, và các thanh ghi đa năng, cộng thêm thông tin mã điều kiện. Cùng với con trỏ chương trình, thông tin về trạng thái tiến trình phải được lưu giữ khi một ngắt (interrupt) xảy ra, để cho phép tiến trình được tiếp tục một cách chính xác sau này.
- *Thông tin điều phối CPU* : Những thông tin này bao gồm độ ưu tiên của tiến trình, con trỏ đến hàng đợi và các thông số điều phối khác.
- *Thông tin quản lý bộ nhớ* : Phần này có thể bao gồm các thông tin như giá trị của thanh ghi cơ sở và thanh ghi giới hạn, bảng trang hoặc bảng phân đoạn, tùy thuộc vào hệ thống bộ nhớ được sử dụng bởi hệ điều hành.
- *Thông tin kế toán* : Những thông tin này bao gồm dung lượng của CPU và thời gian thực được sử dụng, thời hạn, số tài khoản, số hiệu công việc hoặc tiến trình, v.v.
- *Thông tin trạng thái I/O* : Những thông tin này bao gồm danh sách các thiết bị I/O được phân bổ cho tiến trình, danh sách các tập tin mở, v.v.

process state
process number
program counter
registers
memory limits
list of open files
• • •



Tóm lại, PCB chỉ phục vụ như là kho lưu trữ cho bất kỳ thông tin có thể thay đổi tùy theo tiến trình.

8.2 GIAO TIẾP GIỮA CÁC TIẾN TRÌNH

Các tiến trình thực thi đồng thời trong hệ điều hành có thể là các *tiến trình độc lập* hoặc các *tiến trình hợp tác*. Một tiến trình là độc lập nếu không thể ảnh hưởng tới các tiến trình khác hoặc bị ảnh hưởng bởi các tiến trình khác đang thực thi trong hệ thống. Tiến trình không chia sẻ dữ liệu với tiến trình khác là tiến trình độc lập. Một tiến trình không độc lập được gọi là tiến trình hợp tác. Rõ ràng, một tiến trình chia sẻ dữ liệu với các tiến trình khác là một tiến trình hợp tác.

Có nhiều lý do cho việc cung cấp một môi trường cho phép việc hợp tác giữa các tiến trình:

- *Chia sẻ thông tin* : Vì nhiều người sử dụng có thể quan tâm đến cùng một mảnh thông tin (ví dụ, một tập tin chia sẻ), chúng ta phải cung cấp một môi trường để cho phép truy cập đồng thời thông tin đó.
- *Tăng tốc độ tính toán* : Nếu chúng ta muốn một nhiệm vụ cụ thể chạy nhanh hơn, chúng ta phải chia nó thành các nhiệm vụ nhỏ, mỗi nhiệm vụ sẽ được thực thi song song với những nhiệm vụ khác. Chú ý rằng có thể đạt được sự tăng tốc như vậy nếu máy tính có nhiều đơn vị xử lý (ví dụ hạn như nhiều CPU hoặc nhiều kênh I/O).

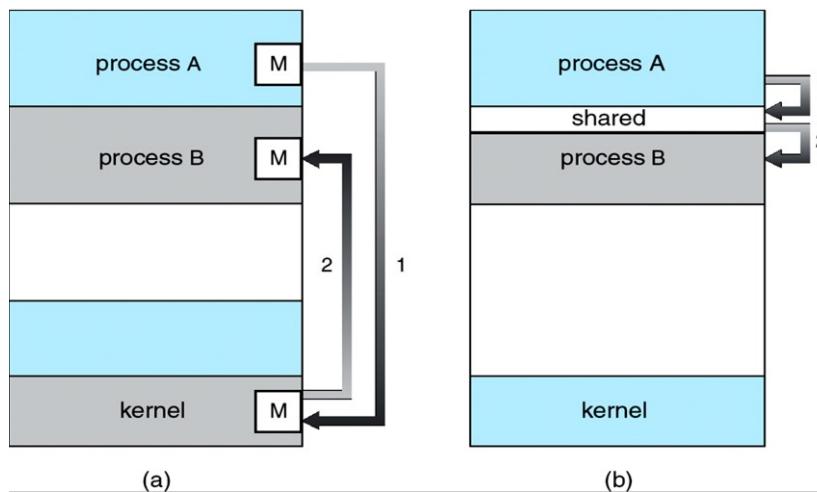
- *Mô đun hóa* : Chúng ta có thể muốn xây dựng hệ thống theo kiểu module, phân chia các chức năng hệ thống thành các tiến trình hoặc luồng riêng biệt.

- *Sự tiện lợi* : Một người dùng riêng rẽ có thể làm nhiều nhiệm vụ cùng một lúc. Ví dụ, người dùng có thể chỉnh sửa, in ấn, và biên dịch song song.

Sự hợp tác giữa các tiến trình đòi hỏi một cơ chế *truyền thông liên tiến trình* (IPC, interprocess communication), cơ chế này cho phép chúng trao đổi dữ liệu và thông tin. Có hai mô hình cơ bản của truyền thông liên tiến trình : *bộ nhớ chia sẻ* (shared memory) và *truyền thông điệp* (message passing). Trong mô hình bộ nhớ chia sẻ, một vùng bộ nhớ, được chia sẻ bởi các tiến trình hợp tác, được thành lập. Sau đó tiến trình có thể trao đổi thông tin bằng cách đọc và ghi dữ liệu vào vùng bộ nhớ chia sẻ. Trong mô hình truyền thông điệp, việc truyền thông diễn ra bằng cách trao đổi thông điệp giữa các tiến trình hợp tác.

Cả hai mô hình trên rất phổ biến trong các hệ điều hành, nhiều hệ thống thực hiện cả hai. Truyền thông điệp hữu ích cho việc trao đổi một lượng nhỏ dữ liệu, bởi vì không có xung đột cần phải tránh. Truyền thông điệp cũng dễ thực hiện hơn bộ nhớ chia sẻ để truyền thông liên máy tính. Bộ nhớ chia sẻ cho phép tốc độ tối đa và tiện lợi. Bộ nhớ chia sẻ nhanh hơn truyền thông điệp, vì các hệ thống truyền thông điệp thường được thực hiện bằng cách sử dụng các lời gọi hệ thống và do đó đòi hỏi tốn thời gian hơn. Ngược lại, trong các hệ thống bộ nhớ chia sẻ, các cuộc gọi hệ thống chỉ được yêu cầu để thiết lập khu vực bộ nhớ chia sẻ. Một khi bộ nhớ chia sẻ được thiết lập, tất cả các truy cập được đối xử như truy cập bộ nhớ thông thường, không có đòi hỏi sự hỗ trợ của hạt nhân. Trong phần còn lại của phần này, chúng ta khám phá các mô hình IPC này chi tiết hơn.

Hình dưới thể hiện hai mô hình vừa thảo luận.



8.2.1 Bộ nhớ chia sẻ

Việc giao tiếp giữa các tiến trình bằng cách sử dụng bộ nhớ chia sẻ yêu cầu các tiến trình giao tiếp thiết lập một khu vực bộ nhớ chia sẻ. Thông thường, khu vực bộ nhớ chia sẻ nằm trong không gian địa chỉ của tiến trình tạo ra vùng bộ nhớ chia sẻ. Các tiến trình khác có nhu cầu giao tiếp bằng vùng bộ nhớ chia sẻ này phải đính kèm nó vào không gian địa chỉ của chúng. Thông thường, hệ điều hành sẽ cố gắng ngăn chặn một tiến trình truy cập bộ nhớ của một tiến trình khác. Bộ nhớ chia sẻ yêu cầu hai hoặc nhiều tiến trình đồng ý loại bỏ hạn chế này. Sau đó chúng có thể trao đổi thông tin bằng cách đọc và viết dữ liệu trong các vùng chia sẻ. Hình thức của dữ liệu và vị trí được xác định bởi các tiến trình và không nằm trong tầm kiểm soát của hệ điều hành. Các tiến trình cũng chịu trách nhiệm đảm bảo rằng chúng không viết vào cùng vị trí cùng một lúc (vì có thể không đảm bảo an toàn dữ liệu).

8.2.2 Truyền thông điệp

Việc truyền thông điệp cung cấp một cơ chế cho phép các tiến trình giao tiếp và đồng bộ các hành động của chúng mà không phải chia sẻ không gian địa chỉ. Phương pháp này đặc biệt hữu ích trong môi trường phân tán, nơi các tiến trình giao tiếp có thể cư trú trên các máy tính khác nhau được nối với nhau qua mạng. Ví dụ, một chương trình chat được sử dụng trên World Wide Web có thể được thiết kế để người tham gia trò chuyện giao tiếp với nhau qua trao đổi tin nhắn.

Cơ chế truyền thông điệp cung cấp ít nhất hai tác vụ: Send(message) và Receive(message). Thông điệp được gửi bởi một tiến trình có thể có kích thước cố định hoặc biến đổi. Nếu chỉ có những thông điệp kích thước cố định được gửi đi, việc

triển khai ở mức hệ thống là đơn giản. Tuy nhiên, hạn chế này làm cho nhiệm vụ lập trình khó khăn hơn. Ngược lại, các thông điệp thay đổi kích thước đòi hỏi việc triển khai ở mức hệ thống phức tạp hơn, nhưng công việc lập trình trở nên đơn giản hơn. Đây là một loại phổ biến của sự cân bằng được thấy trong thiết kế hệ điều hành.

Nếu các tiến trình P và Q muốn giao tiếp, chúng phải gửi và nhận thông điệp của nhau, một liên kết truyền thông phải tồn tại giữa chúng. Liên kết này có thể được thực hiện trong nhiều cách khác nhau. Chúng ta quan tâm ở đây không phải với việc cài đặt vật lý của liên kết (chẳng hạn như bộ nhớ chia sẻ, bus phần cứng, hoặc mạng) mà là với việc cài đặt logic của nó. Dưới đây là một số phương pháp để thực hiện một cách hợp lý một liên kết và các tác vụ send() / receive():

- Giao tiếp trực tiếp hoặc gián tiếp
- Giao tiếp đồng bộ hoặc không đồng bộ
- Đệm (Buffering) tự động hoặc tường minh

Tiếp theo, chúng ta xem xét các vấn đề liên quan các tính năng này

Đặt tên

Tiến trình muốn giao tiếp phải có một cách để tham khảo với nhau. Chúng có thể sử dụng giao tiếp trực tiếp hoặc gián tiếp.

Theo giao tiếp trực tiếp, mỗi tiến trình muốn giao tiếp phải đặt tên một cách rõ ràng tiến trình nhận hoặc tiến trình gửi. Trong sơ đồ này, các hàm send() và receive() được định nghĩa là:

- send(P, message)— Gửi một thông điệp tới tiến trình P.
- receive(Q, message)— Nhận một thông điệp từ tiến trình Q.

Một liên kết giao tiếp trong sơ đồ này có các thuộc tính sau:

1. Liên kết được thiết lập tự động giữa mỗi cặp tiến trình muốn giao tiếp. Các tiến trình chỉ cần phải biết danh tính của nhau để giao tiếp.
2. Liên kết được gắn liền với chính xác hai tiến trình.
3. Giữa mỗi cặp tiến trình, tồn tại chính xác một liên kết.

Sơ đồ này thể hiện tính đối xứng trong việc định địa chỉ, đó là cả tiến trình gửi và tiến trình nhận phải đặt tên tiến trình khác để giao tiếp. Một biến thể của sơ đồ này sử dụng sự bất đối xứng trong việc định địa chỉ. Ở đây, chỉ có tiến trình gửi đặt tên tiến trình nhận và tiến trình nhận không cần thiết đặt tên tiến trình gửi. Trong sơ đồ này, các hàm send() và receive() được định nghĩa như sau:

- send(P, message)— Gửi một thông điệp tới tiến trình P.
- receive(id, message)— Nhận một thông điệp từ tiến trình bất kỳ, biến id được đặt cho tên của tiến trình giao tiếp.

Những bất lợi trong cả hai sơ đồ (đối xứng và bất đối xứng) là tính mô đun bị hạn chế dẫn tới việc xác định tiến trình. Việc thay đổi định danh của một tiến trình có thể dẫn tới việc cần kiểm tra tất cả các đặc tả khác của tiến trình. Tất cả các tham chiếu đến định danh cũ phải được tìm để sửa đổi chúng thành định danh mới. Nói chung, các kỹ thuật mã hóa cứng như vậy, nơi định danh phải được quy định rõ ràng, ít được mong muốn hơn so với các kỹ thuật liên quan đến gián tiếp, như mô tả tiếp theo.

Với giao tiếp gián tiếp, các thông điệp được gửi đến và nhận được từ hộp thư, hoặc cổng. Một hộp thư có thể được xem một cách trừu tượng như một đối tượng mà các tiến trình có thể đặt vào đó hoặc lấy ra từ đó các thông điệp. Mỗi hộp thư có một định danh duy nhất. Ví dụ, hàng đợi tin nhắn POSIX sử dụng một số nguyên để xác định một hộp thư. Trong sơ đồ này, một tiến trình có thể giao tiếp với một số tiến trình khác thông qua một số hộp thư khác nhau. Tuy nhiên, hai tiến trình chỉ có thể giao tiếp nếu chúng có một hộp thư chung. Các hàm send() và receive() được định nghĩa như sau:

- send(A, message) — Gửi một thông điệp tới hộp thư A.
- receive(A, message) — Nhận một thông điệp từ hộp thư A.

Trong sơ đồ này, liên kết truyền thông có các thuộc tính sau:

1. Liên kết được thiết lập giữa một cặp tiến trình chỉ khi cả hai thành viên của cặp có một hộp thư chung.
2. Liên kết có thể liên quan đến hơn hai tiến trình.

3. Giữa mỗi cặp tiến trình giao tiếp, có thể có một số liên kết khác nhau, mỗi liên kết tương ứng với một hộp thư.

Bây giờ giả sử rằng các tiến trình P1, P2, P3 chia sẻ toàn bộ hộp thư A. Tiến trình P1 gửi một thông điệp tới A, trong khi cả hai P2 và P3 thực hiện receive() từ A. Các tiến trình có nhận được các thông điệp gửi bởi P1 không? Câu trả lời phụ thuộc vào các phương pháp chúng ta chọn sau đây :

- Cho phép một liên kết có liên quan nhiều nhất đến hai tiến trình.
- Cho phép chỉ một tiến trình được thực hiện receive() tại một thời điểm.
- Cho phép hệ thống lựa chọn tùy ý tiến trình sẽ nhận được thông điệp (có nghĩa là, P2 hay P3, nhưng không phải cả hai, sẽ nhận được thông điệp).

Một hộp thư có thể được sở hữu bởi một tiến trình hoặc hệ điều hành. Nếu hộp thư được sở hữu bởi một tiến trình (có nghĩa là, các hộp thư là một phần trong không gian địa chỉ của tiến trình này), thì chúng ta phân biệt giữa chủ sở hữu (chỉ có thể nhận tin nhắn qua hộp thư này) và người sử dụng (chỉ có thể gửi tin nhắn đến hộp thư). Vì mỗi hộp thư có một chủ sở hữu duy nhất, không thể có sự nhầm lẫn về tiến trình sẽ nhận được tin nhắn gửi đến hộp thư này. Khi tiến trình sở hữu hộp thư kết thúc, hộp thư biến mất. Bất cứ tiến trình nào sau đó gửi một thông điệp vào hộp thư này phải được thông báo rằng hộp thư không còn tồn tại.

Ngược lại, hộp thư thuộc sở hữu của hệ điều hành có một cuộc sống riêng của mình. Nó độc lập, không gắn liền với bất kỳ tiến trình cụ thể nào. Hệ điều hành sau đó phải cung cấp một cơ chế cho phép tiến trình làm các việc sau:

- Tạo hộp thư mới.
- Gửi và nhận thông điệp qua hộp thư.
- Xóa bỏ hộp thư.

Tiến trình tạo ra một hộp thư mới với chủ sở hữu mặc định. Ban đầu, chủ sở hữu là tiến trình duy nhất có thể nhận thông điệp thông qua hộp thư này. Tuy nhiên, quyền sở hữu và đặc quyền nhận thông điệp có thể được truyền cho các tiến trình khác thông qua các lời gọi hệ thống thích hợp. Tất nhiên, quy định này có thể dẫn đến nhiều tiến trình nhận từ một hộp thư.

Đồng bộ hóa

Giao tiếp giữa các tiến trình diễn ra thông qua các lời gọi tới các hàm send() và receive(). Có những lựa chọn thiết kế khác nhau để thực hiện mỗi hàm này. Truyền thông điệp có thể là khóa (blocking) hay không khóa (nonblocking) - còn được gọi là đồng bộ (synchronous) hay không đồng bộ (nonsynchronous).

- *Gửi đồng bộ* : Tiến trình gửi bị khóa cho đến khi thông điệp được nhận bởi tiến trình nhận hoặc hộp thư.
- *Gửi không đồng bộ* : Tiến trình gửi vẫn tiếp tục hoạt động sau khi gửi thông điệp.
- *Nhận đồng bộ* : Tiến trình nhận bị khóa cho đến khi có thông điệp được gửi tới.
- *Nhận không đồng bộ* : Tiến trình nhận vẫn hoạt động dù không có thông điệp được gửi tới.

Các kết hợp khác nhau của send() và receive() là có thể. Khi cả send() và receive() là đồng bộ, chúng ta có một cuộc gắp gỡ giữa tiến trình gửi và nhận.

Bộ đệm

Cho dù giao tiếp trực tiếp hay gián tiếp, các thông điệp được trao đổi bởi các tiến trình đang giao tiếp cư trú trong một hàng đợi tạm thời. Về cơ bản, hàng đợi như vậy có thể được triển khai theo ba cách:

- *Dung lượng Zero* : Hàng đợi có chiều dài tối đa là 0, do đó, các liên kết không thể có thông điệp nào đang chờ ở đó. Trong trường hợp này, tiến trình gửi phải bị khóa cho đến khi tiến trình nhận nhận được tin nhắn.
- *Dung lượng bị giới hạn* : Hàng đợi có chiều dài hữu hạn n, do đó, nhiều nhất có n thông điệp có thể cư trú trong đó. Nếu hàng đợi là không đầy khi một thông điệp mới được gửi, thông điệp được đặt trong hàng đợi (hoặc thông điệp được sao chép hoặc một con trỏ đến các thông điệp được lưu giữ), và tiến trình gửi có thể tiếp tục thực thi mà không cần chờ đợi. Tuy nhiên, dung lượng của liên kết là hữu hạn. Nếu liên kết đầy, tiến trình gửi bị khóa cho đến khi có không gian trống trong hàng đợi.

- *Dung lượng không bị giới hạn* : Chiều dài của hàng đợi là có khả năng vô hạn, do đó, bất kỳ số lượng thông điệp có thể chờ đợi trong nó. Tiến trình gửi không bao giờ bị khóa.

Trường hợp dung lượng zero đôi khi được gọi là một hệ thống tin nhắn không có đệm, các trường hợp khác được gọi là hệ thống với đệm tự động.

8.3 HỆ THỐNG IPC TRONG WINDOWS

Hệ điều hành Windows là một ví dụ về thiết kế hiện đại, sử dụng tính mô đun để tăng tính chức năng và giảm thời gian cần thiết để triển khai các tính năng mới. Windows cung cấp hỗ trợ cho nhiều môi trường hoạt động, hoặc hệ thống con, với các chương trình ứng dụng giao tiếp thông qua cơ chế truyền thông điệp. Các chương trình ứng dụng có thể được coi là khách hàng của các chương trình chủ hệ thống con Windows XP.

Hạ tầng truyền thông điệp trong Windows được gọi là hạ tầng gọi thủ tục địa phương (LPC - local procedure call). LPC trong Windows dùng để giao tiếp giữa hai tiến trình trên cùng một máy. Nó tương tự như cơ chế RPC tiêu chuẩn được sử dụng rộng rãi, nhưng nó được tối ưu hóa cho Windows. Giống như Mach, Windows sử dụng đối tượng cổng (port) để thiết lập và duy trì một kết nối giữa hai tiến trình. Mỗi khách hàng gọi tới một hệ thống con cần một kênh truyền thông, được cung cấp bởi đối tượng cổng và không bao giờ được thừa kế. Windows sử dụng hai loại cổng: cổng kết nối và các cổng giao tiếp. Chúng thực sự giống nhau nhưng được đặt tên khác nhau theo cách thức chúng được sử dụng.

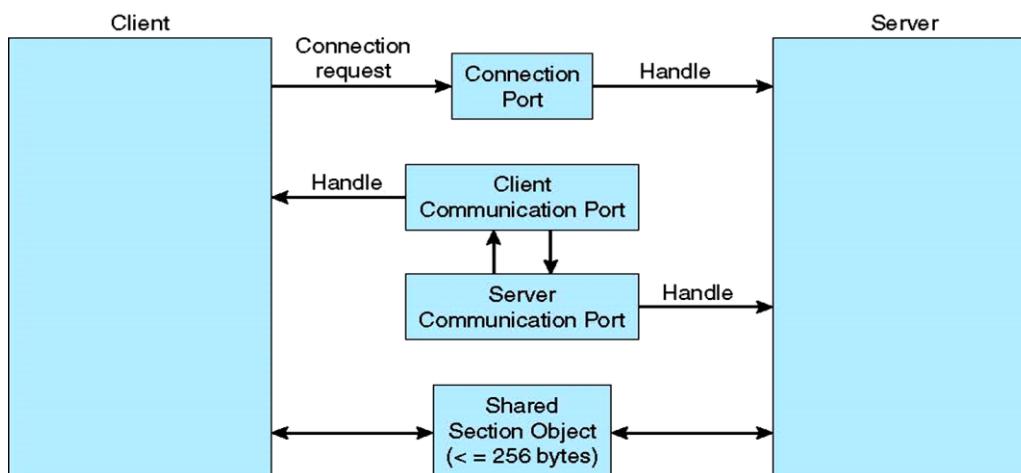
Cổng kết nối là một đối tượng có tên và có thể nhìn thấy tất cả các tiến trình, chúng cung cấp cho các ứng dụng ở xa để thiết lập các kênh giao tiếp. Giao tiếp hoạt động như sau:

- Tiến trình chủ mở một đối tượng cổng kết nối có tên và đợi các tiến trình khách kết nối.
- Tiến trình khách sẽ gửi một yêu cầu kết nối tới cổng có tên của tiến trình chủ bằng một thông điệp kết nối.

- Tiến trình chủ tạo ra hai cổng giao tiếp riêng tư và trả quyền sở hữu của một cổng cho tiến trình khách.
- Tiến trình khách và tiến trình chủ sử dụng cổng tương ứng để gửi thông điệp cho nhau và lắng nghe trả lời.

Windows sử dụng hai loại kỹ thuật truyền thông điệp trên một cổng mà tiến trình khách chỉ định khi thiết lập kênh. Cách đơn giản nhất, được sử dụng cho các thông điệp nhỏ, sử dụng hàng đợi thông điệp của cổng như lưu trữ trung gian và sao chép các thông điệp từ tiến trình này tới tiến trình khác. Theo phương pháp này, thông điệp lên đến 4KB có thể được gửi đi.

Nếu một chương trình khách có nhu cầu gửi tin nhắn lớn hơn, nó truyền thông điệp thông qua một *đối tượng phiên* (secision object), trong đó thiết lập một khu vực bộ nhớ chia sẻ. Khi thiết lập kênh, Chương trình khách quyết định nó có cần phải gửi một thông điệp lớn hay không. Nếu chương trình khách xác định rằng nó muốn gửi thông điệp lớn, nó yêu cầu một đối tượng phiên được tạo ra. Tương tự như vậy, nếu chương trình chủ quyết định rằng thông điệp trả lời là rất lớn, nó tạo ra một đối tượng phiên. Vì vậy mà các đối tượng phiên có thể được sử dụng, một thông báo nhỏ được gửi có chứa một con trỏ và kích thước thông tin về các đối tượng phiên. Phương pháp này là phức tạp hơn so với phương pháp đầu tiên, nhưng nó tránh được việc sao chép dữ liệu. Trong cả hai trường hợp, một cơ chế gọi lại có thể được sử dụng khi chương trình khách hoặc chương trình chủ không thể đáp ứng ngay lập tức với một yêu cầu. Các cơ chế gọi lại cho phép chúng thực hiện xử lý thông điệp không đồng bộ. Cấu trúc của các cuộc gọi thủ tục địa phương trong Windows được thể hiện trong hình dưới.



Điều quan trọng cần lưu ý là các cơ sở LPC trong Windows không phải là một phần của Win32 API và do đó không hiển thị với các lập trình viên ứng dụng. Thay vào đó, các ứng dụng, bằng cách sử dụng Win32 API, viện dẫn các cuộc gọi thủ tục từ xa (RPC, remote procedure calls) tiêu chuẩn. Khi RPC được gọi trên một tiến trình trên cùng một hệ thống, RPC là gián tiếp xử lý thông qua một cuộc gọi thủ tục địa phương. LPCs cũng được sử dụng trong một vài hàm là một phần của Win32 API.

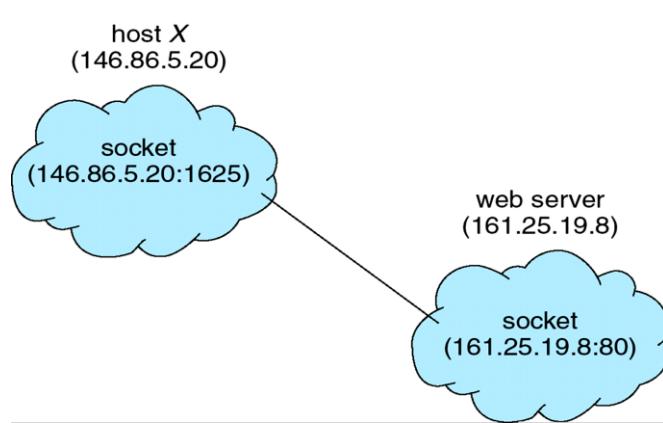
8.4 GIAO TIẾP TRONG HỆ THỐNG KHÁCH-CHỦ

Tại các mục trước, chúng ta đã mô tả các tiến trình có thể giao tiếp bằng cách sử dụng mô hình bộ nhớ chia sẻ và truyền thông điệp. Những kỹ thuật này có thể được sử dụng tốt cho giao tiếp trong hệ thống client-server. Trong phần này, chúng ta khám phá hai chiến lược khác để giao tiếp trong hệ thống client-server: ổ cắm và các cuộc gọi thủ tục từ xa.

8.4.1 Ổ cắm

Một *ổ cắm* (socket) được định nghĩa là một thiết bị đầu cuối để liên lạc. Một cặp tiến trình đang giao tiếp qua mạng sẽ sử dụng một cặp ổ cắm, mỗi tiến trình một ổ. Ổ cắm được xác định bởi một địa chỉ IP cùng với một số hiệu cổng. Nói chung, ổ cắm sử dụng kiến trúc client-server. Chương trình chủ chờ đợi các yêu cầu của chương trình khách đến bằng cách nghe một cổng quy định. Mỗi lần nhận được yêu cầu, máy chủ chấp nhận một kết nối từ ổ cắm của chương trình khách để hoàn thành kết nối. Các chương trình chủ thực hiện các dịch vụ cụ thể (như telnet, FTP và HTTP) sẽ nghe ở các cổng được quy định (chương trình chủ telnet lắng nghe cổng 23, chương trình chủ FTP lắng nghe cổng 21, và chương trình chủ Web, hoặc HTTP, lắng nghe cổng 80). Tất cả các cổng thấp hơn 1024 được coi là các cổng quy định cho các dịch vụ tiêu chuẩn.

Khi một tiến trình khách hàng khởi tạo một yêu cầu cho một kết nối, nó được gán với một cổng của máy chủ của nó. Cổng này là một số tùy ý lớn hơn 1024. Ví dụ, nếu một khách hàng trên máy chủ X với địa chỉ IP 146.86.5.20 mong muốn thiết lập kết nối với một máy chủ web (đang



lắng nghe trên cổng 80) tại địa chỉ 161.25.19.8, máy chủ X có thể được chỉ định cổng 1625. Kết nối sẽ bao gồm một cặp ổ cắm: (146.86.5.20:1625) trên máy chủ X và (161.25.19.8:80) trên máy chủ Web. Tình trạng này được minh họa trong hình trên. Các gói đi du lịch giữa các máy chủ được chuyển giao cho tiến trình thích hợp dựa trên số hiệu cổng đích.

Tất cả các kết nối phải là duy nhất. Do đó, nếu tiến trình khác cũng trên máy chủ X muốn thiết lập một kết nối với máy chủ web, nó sẽ được chỉ định một cổng có số hiệu lớn hơn 1024 và khác 1625. Điều này đảm bảo rằng tất cả các kết nối bao gồm một cặp ổ cắm duy nhất.

Giao tiếp sử dụng ổ cắm, mặc dù phổ biến và hiệu quả, vẫn bị coi là một hình thức cấp thấp của giao tiếp giữa các tiến trình phân tán. Một lý do là ổ cắm chỉ cho phép một dòng không có cấu trúc của byte được trao đổi giữa các luồng giao tiếp. Các ứng dụng khách hoặc chủ có trách nhiệm áp đặt một cấu trúc lên dữ liệu. Trong phần tiếp theo, chúng ta sẽ xem xét *các cuộc gọi thủ tục từ xa* (RPC, Remote Procedure Calls), cung cấp một phương pháp giao tiếp cao cấp hơn.

8.4.2 Cuộc gọi từ xa

Một trong những hình thức phổ biến nhất của dịch vụ từ xa là mô hình cuộc gọi từ xa (RPC - Remote Procedure Calls). RPC được thiết kế như là một cách trừu tượng hóa cơ chế gọi thủ tục để sử dụng giữa các hệ thống với kết nối mạng. Nó tương tự về nhiều khía cạnh với cơ chế IPC và nó thường được xây dựng bên trên một hệ thống như vậy. Tuy nhiên, vì chúng ta đang làm việc với một môi trường trong đó các tiến trình được thực hiện trên các hệ thống riêng rẽ, chúng ta phải sử dụng một sơ đồ giao tiếp dựa trên thông điệp để cung cấp dịch vụ từ xa. Trái ngược với hạ tầng IPC, các thông điệp trao đổi trong RPC giao tiếp được cấu trúc tốt và do đó không còn chỉ là gói dữ liệu. Mỗi thông điệp được định địa chỉ đến một daemon RPC đang nghe trên một cổng của hệ thống từ xa, và mỗi thông điệp có một định danh của một hàm để thực hiện và các tham số để truyền cho hàm đó. Sau đó hàm được thực thi theo yêu cầu và thông tin được gửi trở lại cho tiến trình yêu cầu trong một thông điệp riêng.

Một cổng đơn giản chỉ là một con số có tại đầu một gói tin. Trong khi đó, một hệ thống thường có một địa chỉ mạng, nó có thể có nhiều cổng trong địa chỉ đó để phân biệt nhiều dịch vụ mạng mà nó hỗ trợ. Nếu một tiến trình từ xa cần một dịch vụ, nó

gửi một thông điệp đến đúng cổng. Ví dụ, nếu một hệ thống muốn cho phép các hệ thống khác để có thể liệt kê người dùng hiện tại của nó, nó sẽ có một daemon hỗ trợ như một RPC gắn vào một cổng, ví dụ, cổng 3027. Bất kỳ hệ thống từ xa nào cũng có thể có được những thông tin cần thiết bằng cách gửi thông điệp RPC đến cổng 3027 trên máy chủ, dữ liệu sẽ được nhận trong một thông điệp trả lời.

Ngữ nghĩa của RPC cho phép một tiến trình khách gọi một thủ tục trên một máy chủ từ xa như là nó gọi một thủ tục tại địa phương. Hệ thống RPC giấu các chi tiết cho phép giao tiếp diễn ra bằng cách cung cấp một stub bên phía khách. Thông thường, mỗi stub riêng biệt tồn tại cho một thủ tục từ xa riêng biệt. Khi khách hàng gọi một thủ tục từ xa, hệ thống RPC gọi stub phù hợp, truyền các tham số cung cấp cho thủ tục từ xa. Stub này định vị cổng trên máy chủ và phát đi một thông điệp tới các máy chủ sử dụng truyền thông điệp. Một stub tương tự ở phía máy chủ nhận được thông điệp này và gọi thủ tục trên máy chủ. Nếu cần thiết, giá trị trả lại được truyền lại cho khách hàng bằng cách sử dụng cùng một kỹ thuật.

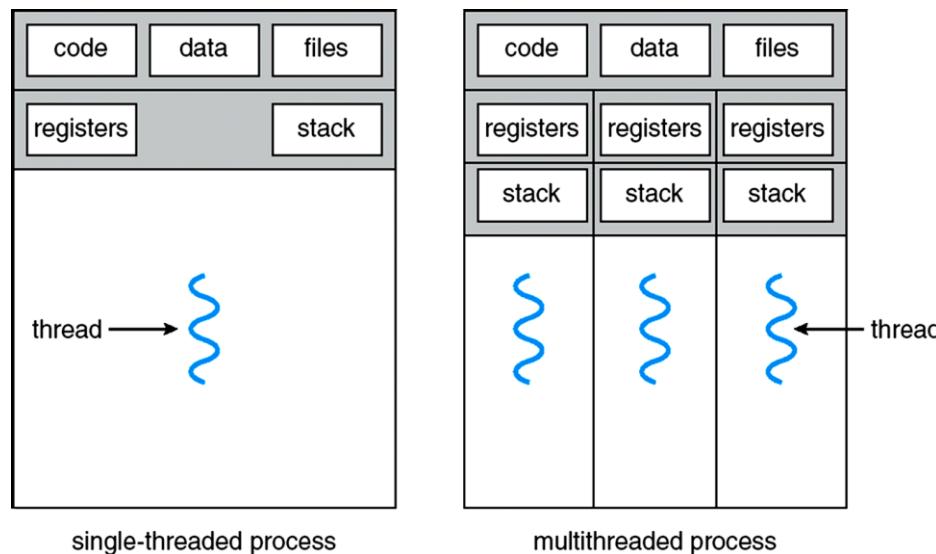
Sơ đồ RPC là hữu ích trong việc triển khai một hệ thống tập tin phân tán. Một hệ thống như vậy có thể được thực hiện như một bộ RPC daemon và khách hàng. Các thông điệp được gửi đến cổng của hệ thống tập tin phân tán trên một máy chủ mà hoạt động tập tin là sẽ diễn ra. Thông điệp có chứa các hoạt động đĩa được thực hiện. Các hoạt động đĩa có thể là đọc, viết, đổi tên, xóa, hay trạng thái, tương ứng với các cuộc gọi hệ thống thông thường liên quan đến file. Các thông điệp phản hồi có chứa bất kỳ dữ liệu kết quả từ cuộc gọi, được thực hiện bởi các daemon DFS thay mặt cho khách hàng. Ví dụ, một thông điệp có thể chứa một yêu cầu chuyển một tập tin toàn bộ cho khách hàng hoặc được giới hạn trong một yêu cầu khởi đơn giản. Trong trường hợp sau, một số yêu cầu như vậy có thể cần thiết nếu một tập tin toàn bộ sắp được chuyển giao.

8.5 LUÔNG

8.5.1 Khái niệm

Mô hình tiến trình được thảo luận cho đến nay đã ngụ ý rằng mỗi tiến trình là một chương trình chỉ có một luồng (thread) thực thi duy nhất. Ví dụ, khi một tiến trình đang chạy một chương trình xử lý văn bản, một luồng lệnh duy nhất được thực thi.

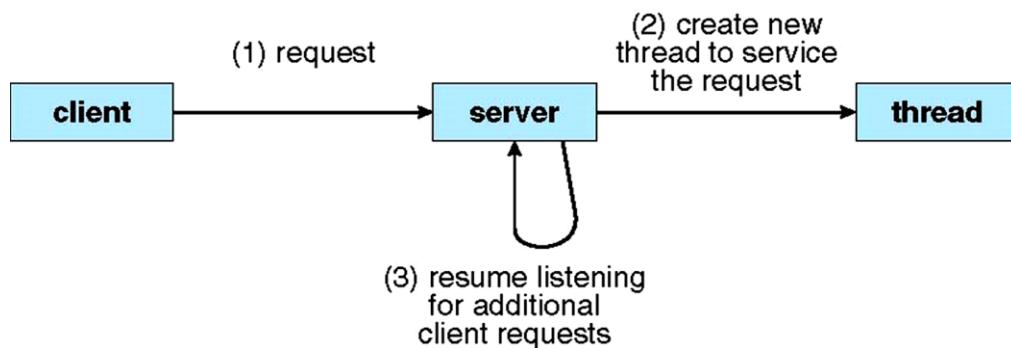
Luồng điều khiển duy nhất này chỉ cho phép tiến trình thực hiện một nhiệm vụ tại một thời điểm. Ví dụ, người dùng không thể đồng thời gõ ký tự và chạy kiểm tra chính tả trong cùng một tiến trình. Nhiều hệ điều hành hiện đại đã mở rộng khái niệm tiến trình để cho phép một tiến trình có nhiều luồng thực thi và do đó thực hiện nhiều nhiệm vụ cùng một lúc. Trên một hệ thống hỗ trợ đa luồng, PCB được mở rộng để bao gồm thông tin cho mỗi luồng. Những thay đổi khác trên toàn hệ thống cũng là cần thiết để hỗ trợ đa luồng.



Hình trên minh họa sự khác biệt giữa một tiến trình đơn luồng truyền thống một tiến trình đa luồng. Luồng bao gồm một ID luồng, một bộ đếm chương trình, một bộ thanh ghi, và một ngăn xếp (stack). Nó chia sẻ với các luồng khác thuộc cùng một tiến trình phần mã của nó, phần dữ liệu, tài nguyên hệ điều hành khác, chẳng hạn như mở các tập tin và tín hiệu. Một tiến trình truyền thống có một luồng điều khiển duy nhất. Nếu một tiến trình có nhiều luồng điều khiển, nó có thể thực hiện nhiều nhiệm vụ cùng một lúc. Động lực

Nhiều gói phần mềm chạy trên máy tính để bàn hiện đại là đa luồng. Một ứng dụng thường được thực hiện như một tiến trình riêng biệt với một số luồng điều khiển. Ví dụ, một trình duyệt web có thể có một luồng hiển thị hình ảnh hoặc văn bản trong khi một luồng khác lấy dữ liệu từ mạng. Một bộ xử lý văn bản có thể có một luồng để hiển thị đồ họa, một luồng để ứng phó với tổ hợp phím từ người dùng, và một luồng thứ ba để thực hiện kiểm tra chính tả và ngữ pháp chạy bên trong.

Trong những tình huống nhất định, một ứng dụng duy nhất có thể được yêu cầu thực hiện một số nhiệm vụ tương tự. Ví dụ, một máy chủ Web chấp nhận yêu cầu của khách hàng cho các trang web, hình ảnh, âm thanh, và vv. Một máy chủ Web bận rộn có thể có nhiều (có lẽ hàng ngàn) khách hàng đồng thời truy cập vào nó. Nếu máy chủ Web chạy như một tiến trình đơn luồng truyền thống, nó sẽ có thể phục vụ chỉ có một khách hàng tại một thời điểm, và một khách hàng có thể phải chờ một thời gian rất dài cho yêu cầu của mình để được phục vụ.



Một giải pháp là phải có server (chương trình phục vụ) chạy như một tiến trình duy nhất chấp nhận các yêu cầu. Khi server nhận được yêu cầu, nó tạo ra một tiến trình riêng biệt để phục vụ yêu cầu. Trong thực tế, phương pháp tạo mới tiến trình này đã được sử dụng phổ biến trước khi luồng trở nên phổ biến. Tuy nhiên, tạo ra tiến trình rất tốn thời gian và cần nhiều tài nguyên. Nếu tiến trình mới sẽ thực hiện các nhiệm vụ tương tự như tiến trình hiện có, tại sao phải chịu tất cả những chi phí trên? Nói chung, sử dụng một tiến trình có chứa nhiều luồng là hiệu quả hơn. Nếu tiến trình Web-server là đa luồng, server sẽ tạo ra một luồng riêng biệt để lắng nghe các yêu cầu của khách hàng. Khi một yêu cầu được thực hiện, thay vì tạo ra tiến trình khác, server sẽ tạo ra một luồng mới để phục vụ yêu cầu và tiếp tục lắng nghe tiếp các yêu cầu. Điều này được minh họa trong hình trên.

Luồng cũng đóng một vai trò quan trọng trong hệ thống lời gọi thủ tục từ xa (RPC). Nhớ lại từ chương 3 là RPC cho phép giao tiếp liên tiến trình bằng cách cung cấp một cơ chế giao tiếp tương tự như lời gọi các hàm hoặc thủ tục thông thường. Thường thường, các RPC server là đa luồng. Khi server nhận được một thông điệp, nó phục vụ thông điệp bằng cách sử dụng một luồng riêng biệt. Điều này cho phép server xử lý nhiều yêu cầu đồng thời.

Cuối cùng, hầu hết hạt nhân hệ điều hành hiện nay đa luồng: một số luồng hoạt động trong hạt nhân, và mỗi luồng thực hiện một nhiệm vụ cụ thể, chẳng hạn như quản lý các thiết bị hoặc xử lý ngắt. Ví dụ, Solaris tạo ra một tập hợp các luồng trong hạt nhân dành cho xử lý ngắt; Linux sử dụng một luồng hạt nhân để quản lý phần bộ nhớ còn trống trong hệ thống.

8.5.2 Lợi ích

Những lợi ích của lập trình đa luồng có thể được chia thành bốn loại chính:

1. *Tính đáp ứng* (Responsiveness) : Đa luồng hóa một ứng dụng tương tác có thể cho phép chương trình tiếp tục chạy ngay cả khi một phần của nó bị khóa (block) hoặc thực hiện một hoạt động kéo dài, do đó làm tăng tính đáp ứng cho người sử dụng. Ví dụ, một trình duyệt web đa luồng có thể cho phép người dùng tương tác trong một luồng trong khi hình ảnh được nạp trong luồng khác.

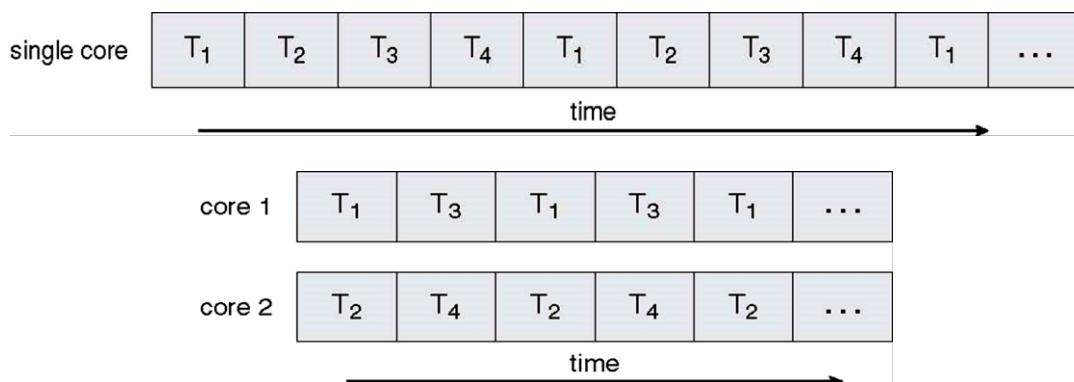
2. *Chia sẻ tài nguyên* : Tiến trình chỉ có thể chia sẻ nguồn lực thông qua các kỹ thuật như bộ nhớ chia sẻ hoặc truyền thông điệp. Kỹ thuật này phải được sắp xếp một cách rõ ràng bởi các lập trình viên. Tuy nhiên, luồng chia sẻ bộ nhớ và tài nguyên của tiến trình mà chúng thuộc về theo mặc định. Lợi ích của việc chia sẻ mã và dữ liệu là nó cho phép một ứng dụng để có nhiều luồng khác nhau của hoạt động trong cùng không gian địa chỉ.

3. *Tiết kiệm* : Việc cấp phát bộ nhớ và các tài nguyên để tạo ra tiến trình rất tốn kém. Vì luồng chia sẻ tài nguyên của tiến trình mà chúng thuộc về, nên việc tạo và chuyển ngữ cảnh luồng là tiết kiệm hơn. Thực nghiệm đo sự khác biệt trong chi phí có thể khó khăn, nhưng nói chung thời gian để tạo và quản lý tiến trình tốn nhiều thời gian hơn luồng. Ví dụ, trong Solaris, tạo ra một tiến trình chậm hơn so với việc tạo ra một luồng ba mươi lần, và chuyển ngữ cảnh chậm hơn khoảng năm lần.

4. *Khả năng mở rộng* : Những lợi ích của đa luồng có thể được tăng lên rất nhiều trong một kiến trúc đa xử, nơi luồng có thể chạy song song trên các bộ xử lý khác nhau. Một tiến trình đơn luồng chỉ có thể chạy trên một bộ xử lý, bất kể đang có sẵn bao nhiêu. Đa luồng trên một máy đa CPU làm tăng tính song song. Chúng ta sẽ tìm hiểu vấn đề này thêm nữa trong phần sau.

8.5.3 Lập trình đa lõi

Một xu hướng gần đây trong thiết kế hệ thống là đặt nhiều lõi tính toán trên một chip duy nhất, trong đó mỗi lõi sẽ xuất hiện như một bộ xử lý riêng biệt cho hệ điều hành. Lập trình đa luồng cung cấp một cơ chế cho việc sử dụng hiệu quả hơn CPU đa lõi và cải thiện tính đồng hành. Hình dưới cho thấy một ứng dụng với bốn luồng. Trên hệ thống máy tính với một lõi duy nhất, đồng hành chỉ có nghĩa là việc thực thi các luồng sẽ được xen kẽ theo thời gian vì lõi xử lý chỉ có khả năng thực thi một luồng tại một thời điểm. Tuy nhiên, trên hệ thống nhiều lõi, đồng hành có nghĩa là các luồng có thể chạy song song vì hệ thống có thể chỉ định một luồng riêng biệt cho mỗi lõi.



Xu hướng các hệ thống đa lõi đã đặt áp lực lên các nhà thiết kế hệ thống cũng như các lập trình viên ứng dụng để sử dụng tốt hơn tính toán đa lõi. Nhà thiết kế hệ điều hành phải viết các thuật toán lập lịch sử dụng xử lý đa lõi cho phép thực hiện song song. Đối với các lập trình viên ứng dụng, có một thách thức là thay đổi chương trình hiện có cũng như thiết kế chương trình mới đa luồng để tận dụng lợi thế của hệ thống đa lõi. Nhìn chung, năm lĩnh vực thể hiện những thách thức trong lập trình cho các hệ thống đa lõi là

1. **Phân chia các hoạt động :** Điều này liên quan đến việc kiểm tra các ứng dụng để tìm khu vực có thể được chia thành các nhiệm vụ riêng biệt, đồng thời và do đó có thể chạy song song trên từng lõi riêng biệt.

2. **Cân bằng :** Trong khi xác định các nhiệm vụ có thể chạy song song, lập trình viên cũng phải đảm bảo rằng các nhiệm vụ thực hiện các công việc có giá trị như nhau. Trong một số trường hợp, một công việc nhất định có thể không đóng góp nhiều giá trị cho tiến trình như các nhiệm vụ tổng thể khác, sử dụng một lõi thực hiện riêng biệt để chạy nhiệm vụ có thể không đáng chi phí.

3. *Tách dữ liệu* : Cũng giống như các ứng dụng được chia thành các nhiệm vụ riêng biệt, truy cập dữ liệu và thao tác bởi các nhiệm vụ phải được chia để chạy trên các lõi riêng biệt.

4. *Phụ thuộc dữ liệu* : Các dữ liệu được truy cập bởi các nhiệm vụ phải được kiểm tra về sự phụ thuộc giữa hai hoặc nhiều nhiệm vụ. Trong trường hợp một nhiệm vụ phụ thuộc vào dữ liệu từ một nhiệm vụ khác, các lập trình viên phải đảm bảo rằng việc thực hiện các nhiệm vụ được đồng bộ hóa để thích ứng với sự phụ thuộc dữ liệu.

5. *Thử nghiệm và gỡ lỗi* : Khi một chương trình đang chạy song song trên nhiều lõi, có nhiều con đường thực hiện khác nhau. Việc thử nghiệm và gỡ lỗi các chương trình đồng hành như vậy là khó khăn hơn việc thử nghiệm và gỡ lỗi các ứng dụng đơn luồng.

Vì những thách thức này, nhiều nhà phát triển phần mềm cho rằng sự ra đời của hệ thống đa lõi sẽ đòi hỏi một cách tiếp cận hoàn toàn mới để thiết kế hệ thống phần mềm trong tương lai.

8.5.4 Các mô hình đa luồng

Thảo luận của chúng ta cho đến nay đã xử lý luồng trong một cảm giác chung. Tuy nhiên, hỗ trợ cho các luồng có thể được cung cấp hoặc ở mức người dùng, cho các luồng người dùng, hoặc bởi hạt nhân, cho các luồng hạt nhân. Các luồng người dùng được hỗ trợ phía bên trên hạt nhân và được quản lý mà không cần sự hỗ trợ của hạt nhân, trong khi các luồng hạt nhân được hỗ trợ và quản lý trực tiếp bởi hệ điều hành. Hầu như tất cả các hệ điều hành hiện đại, bao gồm cả Windows, Linux, Mac OS X, Solaris, và Tru64 UNIX (trước đây là Digital UNIX) đều hỗ trợ các luồng hạt nhân.

Cuối cùng, tồn tại mối quan hệ giữa các luồng người dùng và các luồng hạt nhân. Có ba mô hình quan hệ phổ biến là

- *Mô hình nhiều-một* : Nhiều luồng người dùng được ánh xạ tới một luồng hạt nhân.
- *Mô hình một-một* : Mỗi luồng người dùng được ánh xạ tới một luồng hạt nhân riêng biệt.
- *Mô hình nhiều-nhiều* : Mỗi luồng người dùng được ánh xạ tới nhiều luồng hạt nhân và ngược lại.

8.5.5 Thư viện luồng

Thư viện luồng cung cấp cho các lập trình viên một API dành cho việc tạo và quản lý các luồng. Có hai cách chính triển khai một thư viện luồng. Phương pháp tiếp cận đầu tiên là cung cấp một thư viện hoàn toàn trong không gian người dùng, không có hỗ trợ của hạt nhân. Tất cả các mã và cấu trúc dữ liệu cho thư viện tồn tại trong không gian người dùng. Điều này có nghĩa là việc viện đến một hàm trong thư viện kết quả một lời gọi hàm địa phương trong không gian người dùng và không phải là một lời gọi hệ thống.

Phương pháp thứ hai là triển khai một thư viện cấp hạt nhân được hỗ trợ trực tiếp bởi hệ điều hành. Trong trường hợp này, mã và cấu trúc dữ liệu cho thư viện tồn tại trong không gian hạt nhân. Việc viện đến một hàm trong API cho thư viện thường dẫn đến một lời gọi hệ thống đến hạt nhân.

Ba thư viện luồng chính đang được sử dụng hiện nay: (1) POSIX Pthreads, (2) Win32, và (3) Java. Pthreads là mở rộng các luồng của chuẩn POSIX, có thể được cung cấp như là thư viện cấp người dùng hoặc cấp hạt nhân. Thư viện thread Win32 là một thư viện cấp hạt nhân có sẵn trên hệ thống Windows. Java threads API cho phép luồng được tạo ra và quản lý trực tiếp trong chương trình Java. Tuy nhiên, bởi vì trong hầu hết các trường hợp JVM chạy trên một hệ điều hành máy chủ, nên các luồng Java API thường được triển khai bằng cách sử dụng thư viện luồng có sẵn trên hệ thống máy chủ. Điều này có nghĩa là trên các hệ thống Windows, luồng Java thường được triển khai bằng cách sử dụng Win32 API, các hệ thống UNIX và Linux thường sử dụng Pthreads.

8.5.6 Luồng trong Windows

Windows triển khai Win32 API như API chính của nó. Mỗi ứng dụng Windows chạy như một tiến trình riêng biệt, và mỗi tiến trình có thể chứa một hoặc nhiều luồng. Win32 API dành cho việc tạo luồng được đề cập trong phần 4.3.2. Windows sử dụng ánh xạ một-một mô tả trong phần 4.2.2, trong đó mỗi luồng cấp người dùng ánh xạ tới một luồng cấp hạt nhân có liên quan. Tuy nhiên, Windows cũng cung cấp hỗ trợ cho một thư viện luồng, cung cấp các chức năng của mô hình nhiều-nhiều (mục 4.2.3). Bằng cách sử dụng thư viện luồng, mỗi luồng bất kỳ thuộc về một tiến trình có thể truy cập vào không gian địa chỉ của tiến trình.

Các thành phần chung của một luồng bao gồm:

- Một định danh (ID) luồng duy nhất xác định một luồng
- Một tập thanh ghi biểu diễn tình trạng của bộ xử lý
- Một ngăn xếp người dùng, sử dụng khi luồng đang chạy trong chế độ người dùng, và một ngăn xếp hạt nhân sử dụng khi luồng đang chạy trong chế độ hạt nhân
- Một khu vực lưu trữ riêng được sử dụng bởi các thư viện thời gian chạy khác nhau và các thư viện liên kết động (DLL)

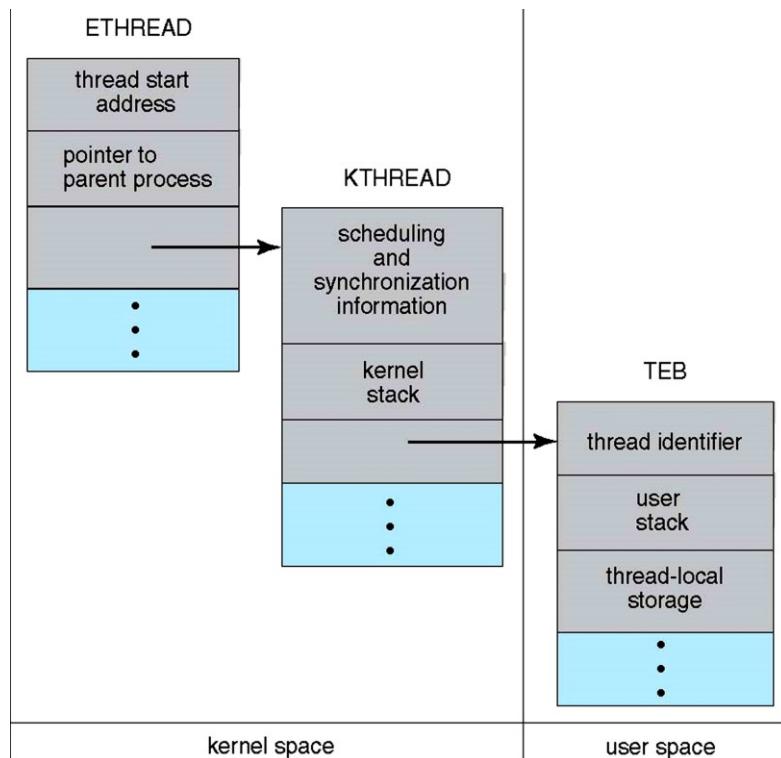
Tập thanh ghi, ngăn xếp, và khu vực lưu trữ riêng tư được gọi là ngữ cảnh của luồng. Cấu trúc dữ liệu cơ bản của một luồng bao gồm:

- ETHREAD – Khối luồng đang thực thi
- KTHREAD – Khối luồng hạt nhân
- TEB – Khối môi trường của luồng

Các thành phần chính của ETHREAD bao gồm một con trỏ đến tiến trình mà luồng thuộc và địa chỉ của thường trình trong đó một luồng bắt đầu điều khiển. ETHREAD cũng chứa một con trỏ đến KTHREAD tương ứng.

KTHREAD bao gồm thông tin lập lịch và đồng bộ hóa cho luồng. Ngoài ra, KTHREAD bao gồm ngăn xếp hạt nhân (sử dụng khi luồng đang chạy trong chế độ hạt nhân) và một con trỏ đến TEB.

Các ETHREAD và KTHREAD tồn tại hoàn toàn trong không gian hạt nhân, điều này có nghĩa là chỉ hạt nhân mới có thể truy cập chúng. TEB là một cấu trúc dữ liệu trong không gian người dùng được truy cập khi luồng đang chạy trong chế độ người dùng. TEB chứa định danh luồng, một ngăn xếp kiểu người dùng, một mảng dành cho dữ liệu luồng cụ thể (mà Windows gọi là lưu trữ địa phương của luồng) và một số trường khác. Cấu trúc của một luồng Windows được minh họa trong hình dưới.



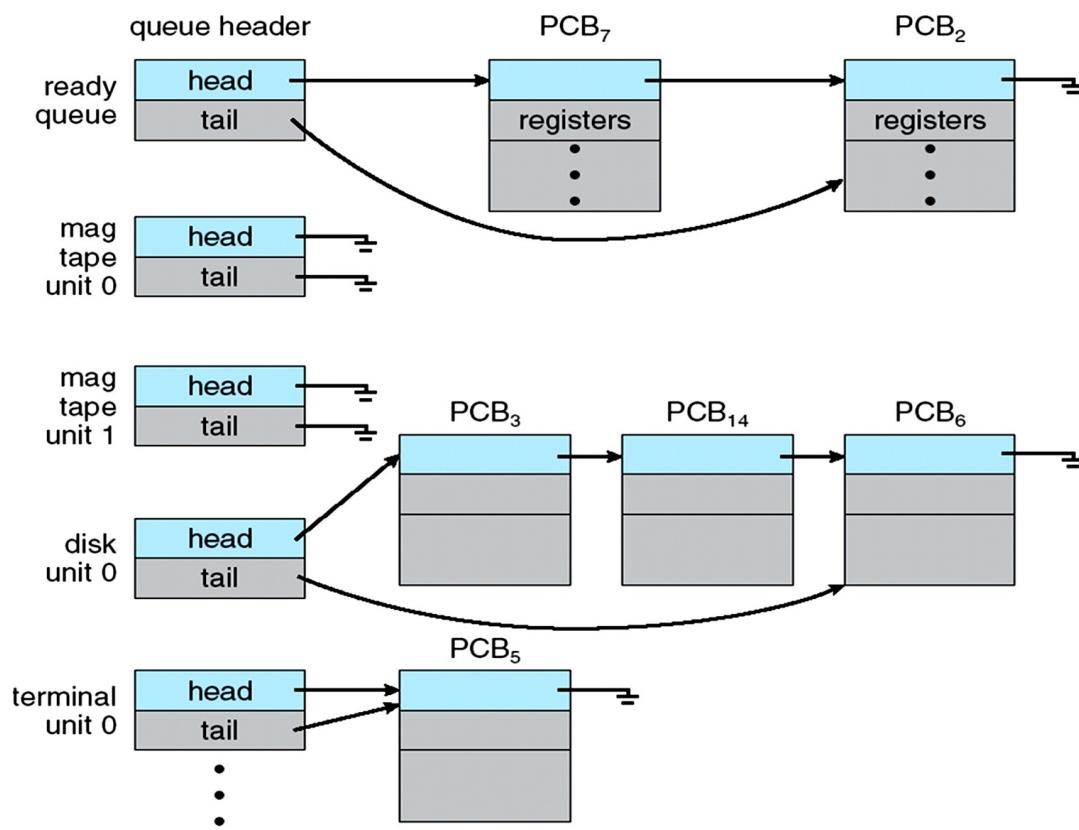
CÂU HỎI VÀ BÀI TẬP

- 1 . Hãy nêu khái niệm tiến trình và luồng.
2. Hãy nêu các lý do để xử lý đồng hành các tiến trình thay cho xử lý tuần tự.
3. Hãy vẽ sơ đồ thể hiện góc nhìn vật lý và góc nhìn logic (góc nhìn của người sử dụng) về xử lý đồng hành.
4. Hãy cho ví dụ chứng tỏ ích lợi của việc xử lý đồng hành.
5. Làm thế nào để xem được danh sách các tiến trình đang hoạt động trong một máy PC chạy hệ điều hành Windows XP.
6. Hãy nêu các trạng thái của tiến trình .
7. Hãy nêu các thao tác quản lý tiến trình
8. Hãy vẽ sơ đồ trạng thái và các thao tác chuyển trạng thái tiến trình.

BÀI 9. ĐỊNH THỜI CPU

9.1 CÁC KHÁI NIỆM CƠ BẢN

9.1.1 Các hàng đợi điều phối

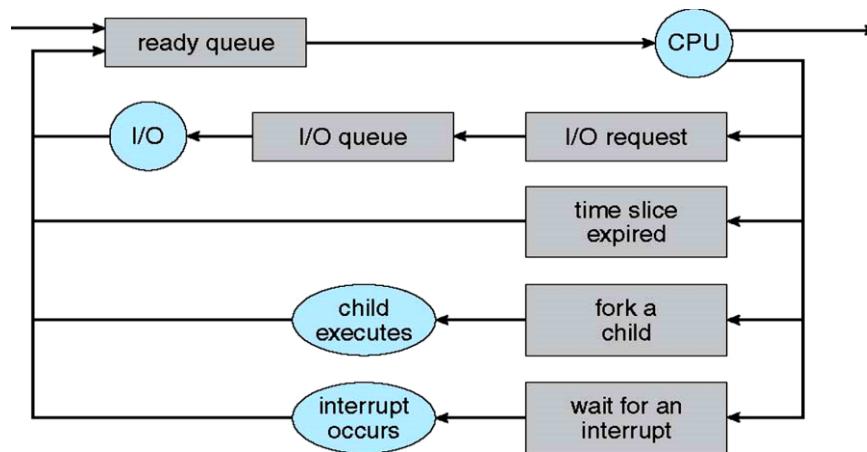


Các tiến trình đang cư trú trong bộ nhớ chính và sẵn sàng chờ đợi để thực thi được lưu giữ trên một danh sách gọi là *hàng đợi sẵn sàng* (ready queue). Hàng đợi này thường được lưu trữ như một danh sách liên kết. Tiêu đề của hàng đợi sẵn sàng chứa con trỏ đến PCBs đầu tiên và cuối cùng trong danh sách. Mỗi PCB chứa một con trỏ chỉ tới PCB tiếp theo trong hàng đợi sẵn sàng.

Hệ thống cũng bao gồm các hàng đợi khác. Khi một tiến trình được phân bổ CPU, nó được thực thi trong một thời gian và kết thúc, hoặc bị ngắt, hoặc chờ đợi sự xuất hiện của một sự kiện cụ thể, chẳng hạn như việc hoàn thành một yêu cầu I/O. Giả sử

tiến trình có yêu cầu I/O để chia sẻ một thiết bị, chẳng hạn như ổ đĩa. Vì có nhiều tiến trình trong hệ thống, ổ đĩa có thể đang bận rộn với các yêu cầu I/O của một số tiến trình khác. Do đó, tiến trình này có thể phải đợi đĩa. Danh sách các tiến trình chờ đợi một thiết bị I/O cụ thể được gọi là một *hàng đợi thiết bị* (device queue). Mỗi thiết bị có hàng đợi thiết bị của mình (hình trên).

Một biểu diễn phổ biến của việc điều phối tiến trình là một biểu đồ hàng đợi, chẳng hạn như trong hình dưới. Mỗi hộp hình chữ nhật đại diện cho một hàng đợi. Có hai loại hàng đợi : hàng đợi sẵn sàng và các hàng đợi thiết bị. Các vòng tròn đại diện cho các nguồn tài nguyên phục vụ cho hàng đợi, và các mũi tên chỉ ra dòng chảy của các tiến trình trong hệ thống.



Tiến trình mới khởi đầu được đưa vào hàng đợi sẵn sàng. Nó chờ đợi ở đó cho đến khi nó được chọn để thực thi. Một khi tiến trình được phân bổ CPU và được thực thi, một trong những sự kiện sau có thể xảy ra:

- Tiến trình có thể sinh ra một yêu cầu I/O và sau đó được đặt vào trong một hàng đợi I/O.
- Tiến trình có thể tạo ra một tiến trình con mới và chờ cho tiến trình con kết thúc.
- Tiến trình có thể bị thu hồi CPU như kết quả của một ngắn, và được đưa trở lại hàng đợi sẵn sàng.

Trong hai trường hợp đầu tiên, tiến trình sẽ chuyển sang trạng thái waiting cho đến khi sự kiện hoàn thành, khi đó nó sẽ chuyển sang trạng thái ready và được đưa trở lại hàng đợi sẵn sàng. Tiến trình tiếp tục chu kỳ này cho đến khi nó kết thúc, lúc đó nó được xóa khỏi tất cả các hàng đợi, PCB và tài nguyên của nó được phân bổ lại.

9.1.2 Các bộ điều phối

Mỗi tiến trình di cư giữa các hàng đợi khác nhau trong suốt cuộc đời của nó. Hệ điều hành phải lựa chọn, cho mục đích điều phối, các tiến trình từ các hàng đợi theo một số cách. Việc lựa chọn tiến trình được thực hiện bởi một *bộ điều phối* (scheduler) phù hợp.

Thông thường, trong một hệ thống xử lý theo lô, số tiến trình được cho phép có thể nhiều hơn số được thực thi ngay lập tức. Các tiến trình này được lưu trữ tạm (spooled) tại một thiết bị lưu trữ dung lượng lớn (thường là đĩa) để thực thi sau. *Bộ điều phối dài hạn* (long-term scheduler), còn gọi là bộ điều phối công việc (job scheduler), chọn các tiến trình từ nơi lưu trữ tạm này và tải chúng vào bộ nhớ để thực thi. *Bộ điều phối ngắn hạn* (short-term scheduler), còn gọi là bộ điều phối CPU (CPU scheduler), lựa chọn và phân bổ CPU cho một trong số các tiến trình trong bộ nhớ để thực thi chúng.

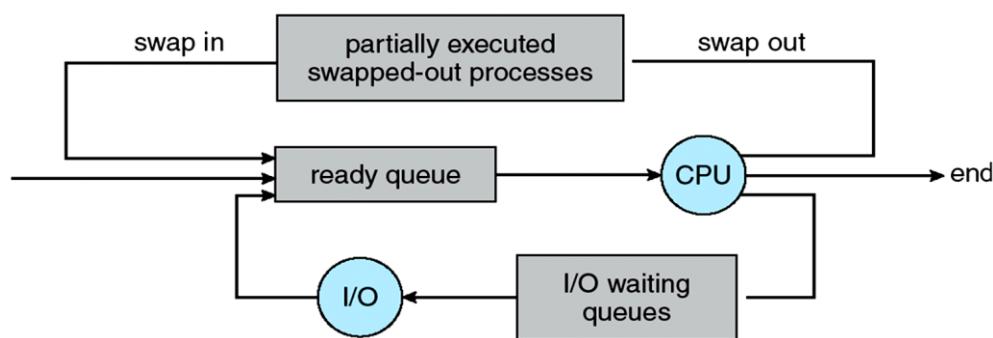
Sự khác biệt chính giữa hai bộ điều phối này nằm ở tần số thực thi. Bộ điều phối ngắn hạn phải thường xuyên chọn một tiến trình mới cho CPU. Một tiến trình có thể được thực thi chỉ một vài mili giây trước khi chờ đợi một yêu cầu I/O. Thông thường, bộ điều phối ngắn hạn thực thi ít nhất một lần mỗi 100 mili giây. Vì thời gian ngắn giữa các chu kỳ thực thi, bộ điều phối ngắn hạn phải làm thật nhanh. Nếu phải mất 10 phần nghìn giây để quyết định thực thi một tiến trình trong 100 mili giây, thì $10/(100 + 10) = 9\%$ thời gian của CPU đang được sử dụng (lãng phí) chỉ cho việc điều phối tiến trình.

Bộ điều phối dài hạn thực hiện với tần số nhỏ hơn nhiều, khoảng cách giữa hai lần tạo ra tiến trình mới có thể là nhiều phút. Bộ điều phối dài hạn kiểm soát mức độ đa chương (số lượng của các tiến trình trong bộ nhớ). Nếu mức độ đa chương là ổn định, thì tốc độ trung bình của việc tạo mới tiến trình phải bằng tốc độ trung bình của việc các tiến trình rời khỏi hệ thống. Như vậy, có thể cần phải gọi đến bộ điều phối dài hạn mỗi khi một tiến trình rời khỏi hệ thống. Vì khoảng thời gian dài giữa các lần thực thi, bộ điều phối dài hạn có thể đủ khả năng dành nhiều thời gian để quyết định chọn tiến trình.

Nói chung, hầu hết các tiến trình có thể được mô tả là hướng I/O hoặc hướng CPU. *Tiến trình hướng I/O* (I/O-bound process) là tiến trình dành nhiều thời gian cho I/O

hơn là cho các tính toán. Ngược lại, *tiến trình hướng CPU* (CPU-bound process) ít có yêu cầu I/O mà sử dụng nhiều thời gian để tính toán. Điều quan trọng là bộ điều phối dài hạn cần kết hợp tốt giữa các tiến trình hướng I/O và hướng CPU khi lựa chọn. Nếu tất cả các tiến trình là hướng I/O, hàng đợi sẵn sàng hầu như rỗng, bộ điều phối ngắn hạn sẽ có ít việc để làm. Nếu tất cả tiến trình là hướng CPU, hàng đợi I/O sẽ hầu như rỗng, các thiết bị sẽ không được sử dụng và một lần nữa hệ thống sẽ không được cân bằng. Do đó, hệ thống với sự kết hợp giữa các tiến trình hướng CPU và hướng I/O sẽ có một hiệu suất tốt nhất.

Trên một số hệ thống, bộ điều phối dài hạn có thể không có hoặc rất nhỏ. Ví dụ, các hệ thống chia sẻ thời gian như UNIX và các hệ thống Microsoft Windows thường không có bộ điều phối dài hạn mà chỉ đặt tất cả các tiến trình mới vào bộ nhớ cho bộ điều phối ngắn hạn. Sự ổn định của các hệ thống này phụ thuộc vào một giới hạn vật lý (ví dụ như số lượng các thiết bị đầu cuối) hoặc theo tính chất tự điều chỉnh của người sử dụng. Nếu hiệu suất giảm đến mức không thể chấp nhận trên một hệ thống đa người dùng, một số người dùng sẽ tự động ra khỏi hệ thống.



Một số hệ điều hành, chẳng hạn như các hệ thống chia sẻ thời gian, có thể có thêm bộ điều phối trung hạn (medium-term scheduler). Bộ điều phối trung hạn được sơ đồ hóa ở hình dưới. Ý tưởng chính đằng sau bộ điều phối trung hạn là đôi khi có thể thuận lợi hơn nếu loại bỏ một số tiến trình khỏi bộ nhớ (và khỏi sự cạnh tranh hoạt động của CPU) và do đó làm giảm mức độ đa chương. Sau đó, các tiến trình này có thể được đưa trở lại bộ nhớ, và việc thực thi chúng có thể được tiếp tục từ điểm mà chúng bị loại bỏ. Sơ đồ này được gọi là *hoán đổi* (swapping). Tiến trình được hoán đổi ra ngoài, và sau đó được hoán đổi vào trở lại, bởi bộ điều phối trung hạn. Hoán đổi có thể là cần thiết để cải thiện sự pha trộn tiến trình hay vì yêu cầu giảm tải bộ nhớ.

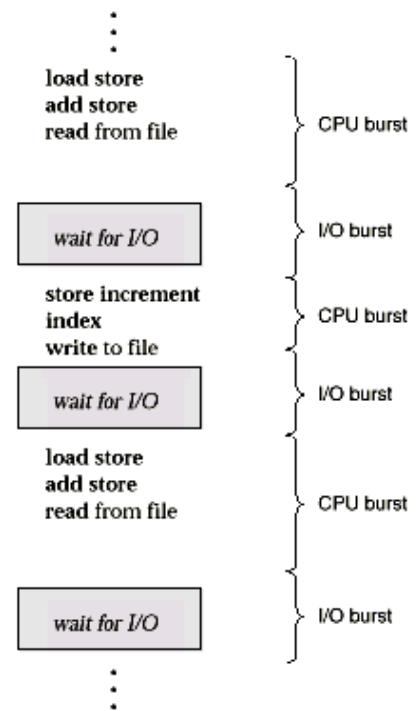
9.1.3 Chuyển ngữ cảnh

Việc luân chuyển CPU từ tiến trình đang thực thi tới tiến trình khác đòi hỏi phải lưu trạng thái tiến trình hiện tại và phục hồi trạng thái của một tiến trình khác. Nhiệm vụ này được gọi là *chuyển ngữ cảnh* (context switch). Khi việc chuyển ngữ cảnh xảy ra, hệ điều hành lưu ngữ cảnh của tiến trình cũ vào PCB và nạp ngữ cảnh đã lưu của tiến trình mới vào CPU. Thời gian chuyển ngữ cảnh hoàn toàn là lãng phí, vì hệ thống không làm việc gì hữu ích trong khi luận chuyển. Tốc độ của nó thay đổi tùy theo máy tính, tùy thuộc vào tốc độ bộ nhớ, số lượng thanh ghi phải được sao chép, và sự tồn tại của các lệnh cụ thể (chẳng hạn như một lệnh duy nhất để nạp hoặc lưu trữ tất cả thanh ghi). Tốc độ thông thường là một vài phần nghìn giây.

Thời gian chuyển ngữ cảnh phụ thuộc nhiều vào sự hỗ trợ của phần cứng. Ví dụ, một số bộ xử lý (như Sun UltraSPARC) cung cấp nhiều tập thanh ghi. Việc chuyển ngữ cảnh ở đây chỉ đơn giản là yêu cầu thay đổi con trỏ đang trỏ vào tập thanh ghi hiện hành. Tất nhiên, nếu số tiến trình đang hoạt động nhiều hơn số tập thanh ghi, hệ thống phải sao chép dữ liệu thanh ghi từ CPU ra bộ nhớ và nạp dữ liệu thanh ghi từ bộ nhớ vào CPU, như mô tả ở phần trên. Ngoài ra, hệ điều hành càng phức tạp, càng nhiều công việc phải được thực hiện trong một lần chuyển ngữ cảnh. Như chúng ta sẽ thấy vào các bài sau, kỹ thuật quản lý bộ nhớ tiên tiến có thể yêu cầu thêm dữ liệu được chuyển với từng ngữ cảnh. Ví dụ, không gian địa chỉ của tiến trình hiện hành phải được bảo quản khi không gian của tiến trình tiếp theo được chuẩn bị sẵn sàng để sử dụng. Làm thế nào không gian địa chỉ được bảo tồn, và số lượng công việc cần thiết để bảo quản nó, chúng phụ thuộc vào phương pháp quản lý bộ nhớ của hệ điều hành.

9.1.4 Chu kỳ CPU-I/O

Sự thành công của việc điều phối CPU phụ thuộc vào thuộc tính được xem xét sau đây của tiến trình. Việc thực thi tiến trình chứa các *chu kỳ* (cycle) thực thi CPU và chờ đợi nhập/xuất như hình dưới. Các tiến trình chuyển đổi giữa hai trạng thái này. Sự thực thi tiến trình bắt đầu với một *chu kỳ CPU* (CPU burst), theo sau bởi một *chu kỳ nhập/xuất* (I/O burst), sau đó là một chu kỳ CPU khác, rồi lại tới một chu kỳ nhập/xuất khác,... chu kỳ CPU cuối cùng sẽ kết thúc việc thực thi tiến trình. Một chương trình hướng nhập/xuất thường có nhiều chu kỳ CPU ngắn. Một chương trình hướng xử lý có thể có nhiều chu kỳ CPU dài. Sự phân bổ này có thể giúp chúng ta chọn giải thuật điều phối CPU hợp lý.



9.1.5 Cơ chế điều phối

Việc điều phối CPU được tiến hành khi một trong bốn trường hợp sau xảy ra:

1. Khi tiến trình chuyển từ trạng thái running sang trạng thái waiting.
 2. Khi tiến trình chuyển từ trạng thái new hoặc running sang trạng thái ready.
 3. Khi tiến trình chuyển từ trạng thái waiting sang trạng thái ready.
 4. Khi tiến trình kết thúc

Có hai loại cơ chế điều phối : *điều phối đặc quyền* (nonpreemptive) và *điều phối không đặc quyền* (preemptive). Trong điều phối đặc quyền, một khi CPU được cấp phát tới một tiến trình, tiến trình giữ CPU cho tới khi nó tự giải phóng CPU do kết thúc công việc hay chuyển tới trạng thái waiting, tức là việc điều phối chỉ xảy ra ở trường hợp 1 và 4. Còn trong điều phối không đặc quyền, việc điều phối xảy ra cả trong bốn trường hợp kể trên.

Phương pháp điều phối đặc quyền được dùng bởi các hệ điều hành Microsoft Windows 3.x. Từ Windows 95 trở về sau, phương pháp điều phối không đặc quyền được sử dụng.

Điều phối không đặc quyền sinh ra một chi phí trong trường chia sẻ dữ liệu. Một tiến trình có thể đang ở giữa giai đoạn cập nhật dữ liệu thì nó bị ngắt CPU cho một tiến trình thứ hai chạy. Tiến trình thứ hai có thể đọc dữ liệu chia sẻ hiện đang ở trong trạng thái thay đổi, điều này có thể dẫn tới việc vi phạm tính toàn vẹn dữ liệu. Do đó, phải có những kỹ thuật mới để kiểm soát việc truy xuất tới dữ liệu được chia sẻ.

Sự không đặc quyền cũng có ảnh hưởng tới thiết kế nhân hệ điều hành. Trong khi xử lý lời gọi hệ thống, nhân có thể chờ một hoạt động dựa theo hành vi của tiến trình. Những hoạt động như thế có thể liên quan với sự thay đổi dữ liệu nhân quan trọng (thí dụ: các hàng đợi nhập/xuất). Điều gì xảy ra nếu tiến trình bị ngắt CPU khi ở trong giai đoạn thay đổi này và nhân (hay trình điều khiển thiết bị) cần đọc hay sửa đổi cùng cấu trúc? Sự lộn xộn chắc chắn xảy ra. Một số hệ điều hành, gồm hầu hết các ấn bản của UNIX, giải quyết vấn đề này bằng cách chờ lời gọi hệ thống hoàn thành hay việc nhập/xuất bị nghẽn, trước khi chuyển đổi ngữ cảnh. Cơ chế này đảm bảo rằng cấu trúc nhân là đơn giản vì nhân sẽ không ngắt một tiến trình trong khi các cấu trúc dữ liệu nhân ở trong trạng thái thay đổi. Tuy nhiên, mô hình thực thi nhân này là mô hình nghèo nàn để hỗ trợ tính toán thời gian thực và đa xử lý.

Trong trường hợp UNIX, các phần mã vẫn là sự rủi ro. Vì các ngắt có thể xảy ra bất cứ lúc nào và vì các ngắt này không thể luôn được bỏ qua bởi nhân, nên phần mã bị ảnh hưởng bởi ngắt phải được đảm bảo từ việc sử dụng đồng thời. Hệ điều hành cần chấp nhận hầu hết các ngắt, ngược lại dữ liệu nhập có thể bị mất hay dữ liệu xuất bị viết chồng. Vì thế các phần mã này không thể được truy xuất đồng hành bởi nhiều tiến trình, chúng vô hiệu hóa ngắt tại lúc nhập và cho phép các ngắt hoạt động trở lại tại thời điểm việc nhập kết thúc. Tuy nhiên, vô hiệu hóa và cho phép ngắt tiêu tốn thời gian, đặc biệt trên các hệ thống đa xử lý.

9.1.6 Bộ phân phát

Một thành phần khác liên quan đến chức năng điều phối CPU là *bộ phân phát* (dispatcher). Bộ phân phát là một module có nhiệm vụ trao điều khiển CPU tới tiến trình được chọn bởi bộ điều phối ngắn hạn. Chức năng này liên quan tới :

1. Việc chuyển ngữ cảnh
2. Việc chuyển chế độ người dùng
3. Việc nhảy tới vị trí hợp lý trong chương trình người dùng để khởi động lại tiến trình

Bộ phân phát cần phải hoạt động rất nhanh, nó được nạp trong mỗi lần chuyển tiến trình. Thời gian mất cho bộ phân phát dừng một tiến trình này và bắt đầu chạy một tiến trình khác được gọi là thời gian trễ cho việc điều phối (dispatch latency).

9.2 CÁC TIÊU CHUẨN ĐIỀU PHỐI

Các giải thuật điều phối khác nhau có các thuộc tính khác nhau và có xu hướng thiên vị cho một loại tiến trình. Trong việc chọn giải thuật, chúng ta phải xét các thuộc tính của các giải thuật khác nhau.

Nhiều tiêu chuẩn được đề nghị để so sánh các giải thuật điều phối. Những đặc điểm được dùng để so sánh có thể tạo sự khác biệt quan trọng trong việc xác định giải thuật tốt nhất. Các tiêu chuẩn gồm:

1. *Việc sử dụng CPU*: Chúng ta muốn giữ CPU bận nhiều nhất có thể. Việc sử dụng CPU có thể từ 0 đến 100%. Trong hệ thống thực, nó nên nằm trong khoảng từ 40% (cho hệ thống được nạp tải nhẹ) tới 90% (cho hệ thống được nạp tải nặng).
2. *Thông lượng*: Nếu CPU bận thực thi các tiến trình thì công việc đang được thực hiện. Thước đo của công việc là số lượng tiến trình được hoàn thành trên một đơn vị thời gian gọi là *thông lượng* (throughput). Đối với các tiến trình dài, tỉ lệ này có thể là 1 tiến trình trên 1 giờ; đối với các giao dịch ngắn, thông lượng có thể là 10 tiến trình trên giây.
3. *Thời gian hoàn thành*: Một chuẩn quan trọng là mất bao lâu để thực thi một tiến trình. Khoảng thời gian từ thời điểm khởi tạo tiến trình tới khi tiến trình hoàn thành được gọi là *thời gian hoàn thành* (turnaround time). Thời gian hoàn thành là tổng các thời gian chờ đưa tiến trình vào bộ nhớ, chờ ở hàng đợi sẵn sàng, thực thi CPU và thực hiện nhập/xuất.

3. *Thời gian chờ*: Giải thuật điều phối CPU không ảnh hưởng lượng thời gian tiến trình thực thi hay thực hiện nhập/xuất; nó chỉ ảnh hưởng lượng thời gian một tiến

trình phải chờ trong hàng đợi sẵn sàng. *Thời gian chờ* (waiting time) là tổng thời gian chờ trong hàng đợi sẵn sàng.

4. *Thời gian đáp ứng*: Trong một hệ thống tương tác, thời gian hoàn thành không phải là tiêu chuẩn tốt nhất. Thông thường, một tiến trình có thể tạo ra dữ liệu xuất tương đối sớm và có thể tiếp tục tính toán các kết quả mới trong khi các kết quả trước đó đang được xuất cho người dùng. Do đó, một thước đo khác là thời gian từ lúc gửi yêu cầu cho tới khi đáp ứng đầu tiên được tạo ra. Thước đo này được gọi là *thời gian đáp ứng* (response time), là lượng thời gian mất đi từ lúc bắt đầu đáp ứng nhưng không là thời gian mất đi để xuất ra đáp ứng đó. Thời gian hoàn thành thường bị giới hạn bởi tốc độ của thiết bị xuất.

Chúng ta muốn tối ưu hóa việc sử dụng CPU và thông lượng, đồng thời tối thiểu hóa thời gian hoàn thành, thời gian chờ và thời gian đáp ứng. Trong hầu hết các trường hợp, chúng ta tối ưu hóa thước đo trung bình. Tuy nhiên, trong một vài trường hợp chúng ta muốn tối ưu giá trị tối thiểu hay giá trị tối đa hơn là giá trị trung bình. Thí dụ, để đảm bảo rằng tất cả người dùng nhận dịch vụ tốt, chúng ta muốn tối thiểu hóa thời gian đáp ứng tối đa.

9.3 CÁC GIẢI THUẬT ĐIỀU PHỐI

Điều phối CPU giải quyết vấn đề quyết định tiến trình nào trong hàng đợi sẵn sàng được cấp phát CPU. Trong phần này chúng ta mô tả nhiều giải thuật điều phối CPU đang có.

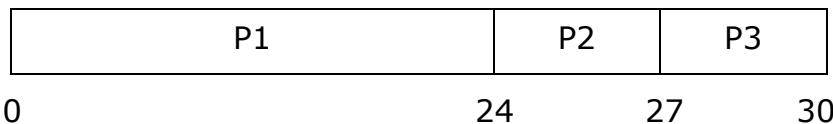
9.3.1 Giải thuật đến trước phục vụ trước

Giải thuật điều phối CPU đơn giản nhất là *đến trước phục vụ trước* (first come first served - FCFS). Với giải thuật này, tiến trình yêu cầu CPU trước sẽ được cấp phát CPU trước. Việc cài đặt giải thuật FCFS được quản lý dễ dàng với hàng đợi FIFO. Khi một tiến trình đi vào hàng đợi sẵn sàng, PCB của nó được liên kết tới đuôi của hàng đợi. Khi CPU rảnh, nó được cấp phát tới một tiến trình tại đầu hàng đợi. Ưu điểm của giải thuật FCFS là đơn giản và dễ hiểu.

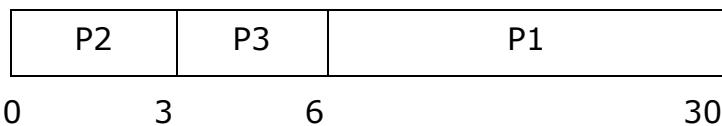
Tuy nhiên, thời gian chờ đợi trung bình của FCFS thường là dài. Xét ví dụ sau bắt đầu từ thời điểm 0, chu kỳ CPU được tính theo mili giây.

Tiến trình	Thời gian xử lý
P1	24
P2	3
P3	3

Nếu các tiến trình đến theo thứ tự P1, P2, P3, chúng ta nhận được kết quả được hiển thị trong *lưu đồ Gantt* như sau:



Thời gian chờ là 0 mili giây cho tiến trình P1, 24 mili giây cho tiến trình P2 và 27 mili giây cho tiến trình P3. Do đó, thời gian chờ đợi trung bình là $(0+24+27)/3=17$ mili giây. Tuy nhiên, nếu các tiến trình đến theo thứ tự P2, P3, P1 thì các kết quả được hiển thị trong lưu đồ Gantt như sau:



Thời gian chờ đợi trung bình bây giờ là $(6+0+3)/3=3$ mili giây. Việc cắt giảm này là quan trọng. Có thể thấy, thời gian chờ đợi trung bình trong FCFS thường không là tối thiểu.

Giải thuật FCSF là giải thuật điều phối đặc quyền CPU. Một khi CPU được cấp phát tới một tiến trình, tiến trình đó giữ CPU cho tới khi nó giải phóng CPU bằng cách kết thúc hay yêu cầu nhập/xuất. Giải thuật FCFS đặc biệt không phù hợp đối với hệ thống chia sẻ thời gian, ở đó mỗi người dùng nhận được sự chia sẻ CPU với những khoảng thời gian đều nhau.

9.3.2 Ưu tiên công việc ngắn nhất

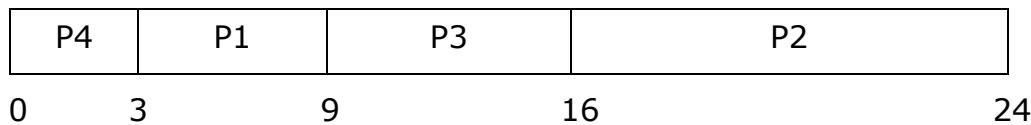
Một tiếp cận khác đối với việc điều phối CPU là giải thuật điều phối *ưu tiên công việc ngắn nhất* (shortest job first - SJF). Giải thuật này gán cho mỗi tiến trình chiều dài của chu kỳ CPU tiếp theo của tiến trình đó. Khi CPU rảnh, nó được cấp phát cho

tiến trình có chu kỳ CPU kế tiếp ngắn nhất. Nếu hai tiến trình có cùng chiều dài chu kỳ CPU kế tiếp, điều phối FCFS được dùng.

Ví dụ, xét tập hợp các tiến trình sau, với chiều dài của thời gian chu kỳ CPU được tính bằng mili giây:

Tiến trình	Thời gian xử lý
P1	6
P2	8
P3	7
P4	3

Dùng điều phối SJF, chúng ta điều phối cho các tiến trình này theo lưu đồ Gantt như sau:



Thời gian chờ đợi là 3 mili giây cho tiến trình P1, 16 mili giây cho tiến trình P2, 9 mili giây cho tiến trình P3, và 0 mili giây cho tiến trình P4. Do đó, thời gian chờ đợi trung bình là $(3+16+9+0)/4 = 7$ mili giây. Nếu chúng ta dùng cơ chế điều phối FCFS thì thời gian chờ đợi trung bình là 10.23 mili giây.

Giải thuật SJF có thể là tối ưu, nó cho thời gian chờ đợi trung bình nhỏ nhất. Bằng cách chuyển một tiến trình ngắn lên trước một tiến trình dài, thời gian chờ đợi của tiến trình ngắn giảm nhiều hơn so với việc tăng thời gian chờ đợi của tiến trình dài. Do đó, thời gian chờ đợi trung bình giảm.

Khó khăn thật sự với giải thuật SJF là làm thế nào để biết chiều dài của yêu cầu CPU tiếp theo. Đối với điều phối dài trong hệ thống theo lô, chúng ta có thể dùng chiều dài như giới hạn thời gian xử lý mà người dùng xác định khi bắt đầu công việc. Do đó, người dùng được cơ động để ước lượng chính xác giới hạn thời gian xử lý vì giá trị thấp hơn có nghĩa là đáp ứng nhanh hơn. Điều phối SJF được dùng thường xuyên trong điều phối dài.

Mặc dù SJF là tối ưu nhưng nó không thể được cài đặt tại cấp điều phối CPU ngắn vì không có cách nào để biết chiều dài chu kỳ CPU tiếp theo. Một tiếp cận khác là điều

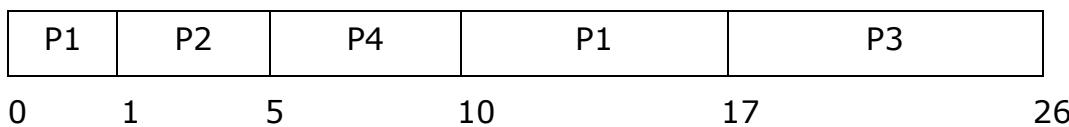
phối SJF gần đúng. Chúng ta có thể không biết chiều dài của chu kỳ CPU kế tiếp nhưng chúng ta có thể đoán giá trị của nó. Chúng ta mong muốn rằng chu kỳ CPU kế tiếp sẽ tương tự chiều dài những chu kỳ CPU trước đó. Do đó, bằng cách tính toán mức xấp xỉ chiều dài của chu kỳ CPU kế tiếp (mà chúng ta sẽ không đi vào chi tiết), chúng ta chọn một tiến trình với chu kỳ CPU được đoán là ngắn nhất.

Giải thuật SJF có thể không đặc quyền hoặc đặc quyền. Chọn lựa này phát sinh khi một tiến trình mới đến tại hàng đợi sẵn sàng trong khi một tiến trình trước đó đang thực thi. Một tiến trình mới có thể có chu kỳ CPU tiếp theo ngắn hơn chu kỳ CPU còn lại của tiến trình đang thực thi. Giải thuật SJF không đặc quyền sẽ ngắt CPU của tiến trình đang thực thi, trong khi giải thuật SJF đặc quyền sẽ cho phép tiến trình đang thực thi thực hiện hết chu kỳ CPU của nó.

Ví dụ, xét 4 tiến trình sau với chiều dài của thời gian chu kỳ CPU được cho tính bằng mili giây

Tiến trình	Thời gian đến	Thời gian xử lý
P1	0	8
2	1	4
P3	2	9
P4	3	5

Nếu các tiến trình đi vào hàng đợi sẵn sàng tại những thời điểm và cần thời gian xử lý được hiển thị trong bảng trên thì kết quả điều phối SJF không đặc quyền được mô tả trong lưu đồ Gantt như sau:



Tiến trình P1 được bắt đầu tại thời điểm 0, vì nó là tiến trình duy nhất trong hàng đợi. Tiến trình P2 đến tại thời điểm 1. Thời gian còn lại cho P1 (7 mili giây) là lớn hơn thời gian được yêu cầu bởi tiến trình P2 (4 mili giây) vì thế tiến trình P1 bị ngắt CPU và tiến trình P2 được cấp phát CPU... Thời gian chờ đợi trung bình cho ví dụ này là: $((10-1) + (1-1) + (17-2) + (5-3))/4 = 6.5$ mili giây. Điều phối SJF đặc quyền cho kết quả thời gian chờ đợi trung bình là 7.75 mili giây.

9.3.3 Điều phối theo độ ưu tiên

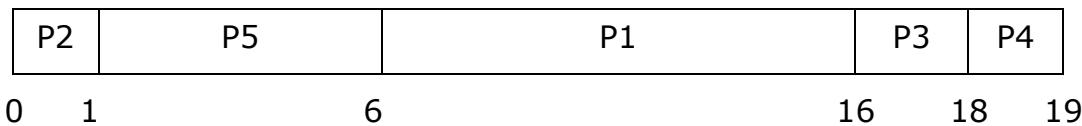
Giải thuật SJF là trường hợp đặc biệt của giải thuật *điều phối theo độ ưu tiên* (priority-scheduling). Độ ưu tiên được gán với mỗi tiến trình và CPU được cấp phát tới tiến trình có độ ưu tiên cao nhất. Các tiến trình có độ ưu tiên bằng nhau được điều phối theo FCFS.

Giải thuật SJF là giải thuật ưu tiên đơn giản ở đó độ ưu tiên là nghịch đảo với chu kỳ CPU được đoán tiếp theo. Chu kỳ CPU lớn hơn có độ ưu tiên thấp hơn và ngược lại.

Ví dụ, xét tập hợp tiến trình sau đến tại thời điểm 0 theo thứ tự P1, P2, ..., P5 với chiều dài thời gian chu kỳ CPU được tính bằng mili giây:

Tiến trình	Thời gian xử lý	Độ ưu tiên
P1	10	3
P2	1	1
P3	2	4
P4	1	5
P5	5	2

Sử dụng điều phối theo độ ưu tiên, chúng ta sẽ điều phối các tiến trình này theo lưu đồ Gantt như sau:



Thời gian chờ đợi trung bình là 8.2 mili giây.

Độ ưu tiên có thể được định nghĩa bên trong hay bên ngoài. Độ ưu tiên được định nghĩa bên trong thường dùng định lượng hoặc nhiều định lượng có thể đo để tính toán độ ưu tiên của một tiến trình. Ví dụ, các giới hạn thời gian, các yêu cầu bộ nhớ, số lượng tập tin đang mở và tỉ lệ của chu kỳ nhập/xuất trung bình với tỉ lệ của chu kỳ CPU trung bình. Các độ ưu tiên bên ngoài được thiết lập bởi các tiêu chuẩn bên ngoài đối với hệ điều hành như sự quan trọng của tiến trình, loại và lượng chi phí đang được trả cho việc dùng máy tính, văn phòng hỗ trợ công việc, ..

Điều phối theo độ ưu tiên có thể không đặc quyền hoặc đặc quyền. Khi một tiến trình đến hàng đợi sẵn sàng, độ ưu tiên của nó được so sánh với tiến trình hiện đang chạy. Giải thuật điều phối theo độ ưu tiên không đặc quyền sẽ ngắt CPU nếu độ ưu tiên của tiến trình mới đến cao hơn độ ưu tiên của tiến trình đang thực thi. Giải thuật điều phối theo độ ưu tiên đặc quyền chỉ đơn giản đặt tiến trình mới tại đầu hàng đợi sẵn sàng.

Vấn đề chính với giải thuật điều phối theo độ ưu tiên là *nghẽn không hạn định* (indefinite blocking) hay *đói CPU* (starvation). Một tiến trình sẵn sàng chạy nhưng thiếu CPU có thể xem như bị nghẽn. Giải thuật điều phối theo độ ưu tiên có thể để lại nhiều tiến trình có độ ưu tiên thấp chờ CPU không hạn định. Trong một hệ thống máy tính tải cao, dòng đều đặn các tiến trình có độ ưu tiên cao hơn có thể ngăn chặn việc nhận CPU của tiến trình có độ ưu tiên thấp. Thông thường, một trong hai trường hợp xảy ra. tất cả các tiến trình sẽ được chạy xong hay hệ thống máy tính sẽ đổ vỡ và mất tất cả các tiến trình có độ ưu tiên thấp chưa được kết thúc.

Một giải pháp cho vấn đề nghẽn không hạn định này là *hoá già* (aging). Hógià là kỹ thuật tăng dần độ ưu tiên của tiến trình chờ trong hệ thống một thời gian dài. Thí dụ, nếu các độ ưu tiên nằm trong dãy từ 127 (thấp) đến 0 (cao), chúng ta giảm độ ưu tiên của tiến trình đang chờ xuống 1 mỗi 15 phút. Cuối cùng, thậm chí một tiến trình với độ ưu tiên khởi đầu 127 sẽ đạt độ ưu tiên cao nhất trong hệ thống và sẽ được thực thi.

9.3.4 Điều phối luân phiên

Giải thuật *điều phối luân phiên* (round robin - RR) được thiết kế đặc biệt cho hệ thống chia sẻ thời gian. Một đơn vị thời gian nhỏ được gọi là *định mức thời gian* (time quantum) hay là *khe thời gian* (time slice) được định nghĩa. Định mức thời gian thường từ 10 đến 100 mili giây. Hàng đợi sẵn sàng được xem như một hàng đợi vòng. Bộ điều phối CPU di chuyển vòng quanh hàng đợi sẵn sàng, cấp phát CPU cho mỗi tiến trình một khoảng thời gian tối đa bằng một định mức thời gian.

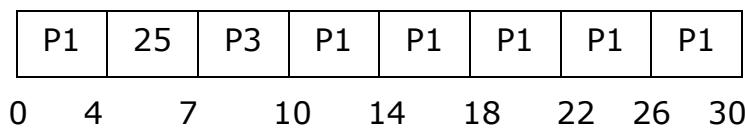
Để cài đặt điều phối RR, chúng ta quản lý hàng đợi sẵn sàng như một hàng đợi FIFO của các tiến trình. Các tiến trình mới được thêm vào đuôi hàng đợi. Bộ điều phối CPU chọn tiến trình đầu tiên từ hàng đợi sẵn sàng, đặt bộ đếm thời gian để ngắt sau 1 định mức thời gian.

Một trong hai trường hợp sẽ xảy ra. Tiến trình có chu kỳ CPU ít hơn một định mức thời gian. Trong trường hợp này, tiến trình sẽ tự giải phóng, bộ điều phối sẽ xử lý tiến trình tiếp theo trong hàng đợi sẵn sàng. Ngược lại, nếu chu kỳ CPU của tiến trình đang chạy dài hơn một định mức thời gian thì độ đếm thời gian sẽ báo và gây ra một ngắt tới hệ điều hành. Việc chuyển đổi ngữ cảnh sẽ được thực thi và tiến trình được đặt trở lại tại đầu của hàng đợi sẵn sàng. Sau đó, bộ điều phối CPU sẽ chọn tiến trình tiếp theo trong hàng đợi sẵn sàng.

Tuy nhiên, thời gian chờ đợi trung bình của RR thường là quá dài. Xét một tập hợp các tiến trình đến tại thời điểm 0 với chiều dài thời gian CPU-burst được tính bằng mili giây:

Tiến trình	Thời gian xử lý
P1	24
P2	3
P3	3

Nếu sử dụng định mức thời gian là 4 mili giây thì tiến trình P1 nhận 4 mili giây đầu tiên. Vì nó yêu cầu 20 mili giây còn lại nên nó bị ngắt CPU sau định mức thời gian đầu tiên và CPU được cấp tới tiến trình tiếp theo trong hàng đợi, tiến trình P2. Vì P2 không cần tới 4 mili giây nên nó kết thúc trước khi định mức thời gian của nó hết hạn. Sau đó, CPU được cho tới tiến trình kế tiếp, tiến trình P3... Kết quả điều phối RR là:



Thời gian chờ đợi trung bình là $17/3=5.66$ mili giây.

Trong giải thuật RR, không tiến trình nào được cấp phát CPU nhiều hơn 1 định mức thời gian trong một lần. Nếu chu kỳ CPU của tiến trình vượt quá 1 định mức thời gian thì tiến trình đó bị ngắt CPU và nó được đặt trở lại hàng đợi sẵn sàng. Giải thuật RR là giải thuật không đặc quyền.

Năng lực của giải thuật RR phụ thuộc nhiều vào kích thước của định mức thời gian. Nếu định mức thời gian rất lớn (lượng vô hạn) thì RR tương tự như FCFS. Nếu định mức thời gian là rất nhỏ (1 mili giây) thì tiếp cận RR được gọi là *chia sẻ bộ xử lý*.

(processor sharing) như thể mỗi tiến trình trong n tiến trình có bộ xử lý riêng của chính nó chạy với $1/n$ tốc độ của bộ xử lý thật.

Chúng ta cũng cần xem xét ảnh hưởng của việc chuyển đổi ngữ cảnh lên năng lực của việc điều phối RR. Chúng ta giả sử rằng chỉ có 1 tiến trình với 10 đơn vị thời gian. Nếu một định mức là 12 đơn vị thời gian thì tiến trình kết thúc ít hơn 1 định mức thời gian, không có chi phí nào khác. Tuy nhiên, nếu định mức là 6 đơn vị thời gian thì tiến trình cần 2 định mức thời gian và một lần chuyển ngữ cảnh. Nếu định mức thời gian là 1 đơn vị thời gian thì có 9 lần chuyển ngữ cảnh, việc thực thi của tiến trình bị chậm lại như được hiển thị trong hình dưới.

Do đó chúng ta muốn có định mức thời gian lớn và thời gian chuyển ngữ cảnh nhỏ. Nếu thời gian chuyển ngữ cảnh chiếm 10% định mức thời gian thì khoảng 10% thời gian CPU sẽ được dùng để chuyển ngữ cảnh.

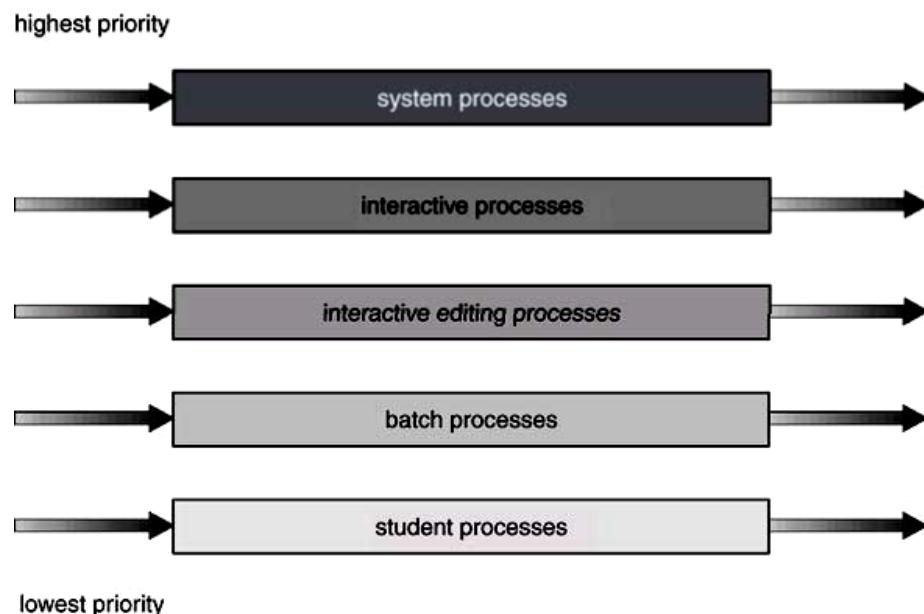
Thời gian hoàn thành cũng phụ thuộc kích thước của định mức thời gian. Nếu định mức thời gian quá lớn thì điều phối RR gần giống với FCFS. Qui tắc là định mức thời gian nên dài hơn 80% chu kỳ CPU của các tiến trình.

9.3.5 Điều phối với hàng đợi nhiều cấp

Một loại giải thuật điều phối khác được tạo ra cho trường hợp các tiến trình được chia thành các nhóm khác nhau. Ví dụ, chia các tiến trình thành hai nhóm, nhóm các tiến trình chạy ở chế độ tương tác (foreground hay interactive) và nhóm các tiến trình chạy ở chế độ nền hay theo lô (background hay batch). Hai nhóm tiến trình này có yêu cầu đáp ứng thời gian khác nhau và vì thế có yêu cầu về điều phối khác nhau. Các tiến trình chạy ở chế độ tương tác có độ ưu tiên cao hơn các tiến trình chạy ở chế độ nền.

Giải thuật điều phối *hàng đợi nhiều cấp* (multilevel queue) chia hàng đợi thành nhiều hàng đợi riêng rẽ (hình dưới). Mỗi tiến trình được gán vĩnh viễn tới một hàng đợi, thường dựa trên thuộc tính của tiến trình như kích thước bộ nhớ, độ ưu tiên tiến trình hay loại tiến trình. Mỗi hàng đợi có giải thuật điều phối của chính nó. Ví dụ: các hàng đợi riêng rẽ có thể được dùng cho các tiến trình ở chế độ nền và chế độ tương tác. Hàng đợi ở chế độ tương tác có thể được điều phối bởi giải thuật RR trong khi hàng đợi ở chế độ nền được điều phối bởi giải thuật FCFS.

Ngoài ra, phải có việc điều phối giữa các hàng đợi, mà thường được cài đặt như điều phối không đặc quyền với độ ưu tiên cố định. Thí dụ, hàng đợi ở chế độ tương tác có độ ưu tiên cao hơn hàng đợi ở chế độ nền.



Xét một ví dụ của giải thuật hàng đợi nhiều mức với năm hàng đợi (hình trên):

- Các tiến trình hệ thống
- Các tiến trình tương tác
- Các tiến trình soạn thảo tương tác
- Các tiến trình theo lô
- Các tiến trình sinh viên

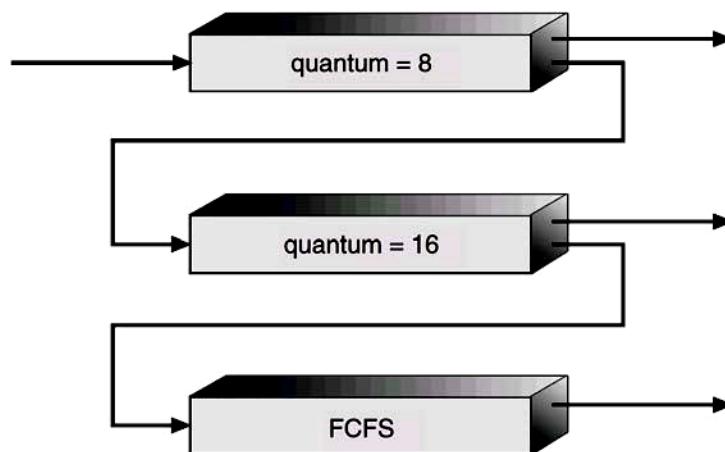
Mỗi hàng đợi có độ ưu tiên tuyệt đối hơn hàng đợi có độ ưu tiên thấp hơn. Thí dụ: không có tiến trình nào trong hàng đợi theo lô có thể chạy trừ khi hàng đợi cho các tiến trình hệ thống, các tiến trình tương tác và các tiến trình soạn thảo tương tác đều rỗng. Nếu một tiến trình soạn thảo tương tác được đưa vào hàng đợi sẵn sàng trong khi một tiến trình theo lô đang chạy thì tiến trình theo lô bị ngắt CPU. Solaris 2 dùng giải thuật này.

Một khả năng khác là khe thời gian giữa hai hàng đợi. Mỗi hàng đợi nhận một khe thời gian CPU xác định, sau đó nó có thể điều phối giữa các tiến trình khác nhau trong hàng đợi của nó. Thí dụ, trong hàng đợi tương tác-nền, hàng đợi tương tác được cho

80% thời gian của CPU cho giải thuật RR giữa các tiến trình của nó, ngược lại hàng đợi nền nhận 20% thời gian CPU cho các tiến trình của nó theo giải thuật FCFS.

9.3.6 Điều phối hàng đợi phản hồi đa cấp

Thông thường, trong giải thuật hàng đợi đa cấp, các tiến trình được gán vĩnh viễn tới hàng đợi khi được đưa vào hệ thống. Các tiến trình không di chuyển giữa các hàng đợi. Nếu có các hàng đợi riêng cho các tiến trình tương tác và các tiến trình nền thì các tiến trình không di chuyển từ một hàng đợi này tới hàng đợi khác vì các tiến trình không tự nhiên thay đổi tính chất giữa tương tác và nền. Cách tổ chức này có ích vì chi phí điều phối thấp nhưng thiếu linh động và có thể dẫn đến tình trạng “đói CPU”.



Tuy nhiên, điều phối *hang doi phan hoi da cap* (multilevel feedback queue) cho phép một tiến trình di chuyển giữa các hàng đợi. Ý tưởng là tách riêng các tiến trình với các đặc điểm chu kỳ CPU khác nhau. Nếu một tiến trình dùng quá nhiều thời gian CPU thì nó sẽ được di chuyển tới hàng đợi có độ ưu tiên thấp. Cơ chế này để lại các tiến trình hướng nhập/xuất và các tiến trình tương tác trong các hàng đợi có độ ưu tiên cao hơn. Tương tự, một tiến trình chờ quá lâu trong hàng đợi có độ ưu tiên thấp hơn có thể được di chuyển tới hàng đợi có độ ưu tiên cao hơn. Đây là hình thức của giải pháp *hóa già* nhằm ngăn chặn việc đói CPU.

Ví dụ, xét một bộ điều phối hàng đợi phản hồi nhiều cấp với ba hàng đợi được đánh số từ 0 tới 2 (như hình trên). Bộ điều phối trước tiên thực thi tất cả tiến trình chứa trong hàng đợi 0. Chỉ khi hàng đợi 0 rỗng nó mới thực thi các tiến trình trong hàng đợi 1. Tương tự, các tiến trình trong hàng đợi 2 sẽ được thực thi chỉ nếu hàng đợi 0 và 1 rỗng. Một tiến trình ở hàng đợi 1 sẽ được ưu tiên hơn tiến trình ở hàng đợi

2. Tương tự, một tiến trình ở hàng đợi 0 sẽ được ưu tiên hơn một tiến trình vào hàng đợi 1.

Tiến trình khi được đưa vào hàng đợi sẵn sàng sẽ được đặt trong hàng đợi 0. Tiến trình trong hàng đợi 0 được cho một định mức thời gian là 8 mili giây. Nếu nó không kết thúc trong thời gian này thì nó sẽ di chuyển vào đuôi của hàng đợi 1. Nếu hàng đợi 0 rỗng thì tiến trình tại đầu của hàng đợi 1 được cho định mức thời gian là 16 mili giây. Nếu nó không hoàn thành thì nó bị ngắt CPU và được đặt vào hàng đợi 2. Các tiến trình trong hàng đợi 2 được chạy trên cơ sở FCFS chỉ khi hàng đợi 0 và 1 rỗng.

Giải thuật điều phối này cho độ ưu tiên cao nhất tới bất cứ tiến trình nào với chu kỳ CPU 8 mili giây hay ít hơn. Một tiến trình như thế sẽ nhanh chóng nhận CPU, kết thúc chu kỳ CPU của nó và bỏ đi chu kỳ I/O kế tiếp của nó. Các tiến trình cần hơn 8 mili giây nhưng ít hơn 24 mili giây được phục vụ nhanh chóng mặc dù với độ ưu tiên thấp hơn các tiến trình ngắn hơn. Các tiến trình dài tự động rơi xuống hàng đợi 2 và được phục vụ trong thứ tự FCFS với bất cứ chu kỳ CPU còn lại từ hàng đợi 0 và 1.

Nói chung, một bộ điều phối hàng đợi phản hồi nhiều cấp được định nghĩa bởi các tham số sau:

- Số lượng hàng đợi
- Giải thuật điều phối cho mỗi hàng đợi
- Phương pháp được dùng để xác định khi nâng cấp một tiến trình tới hàng đợi có độ ưu tiên cao hơn.
- Phương pháp được dùng để xác định khi nào chuyển một tiến trình tới hàng đợi có độ ưu tiên thấp hơn.
- Phương pháp được dùng để xác định hàng đợi nào một tiến trình sẽ đi vào và khi nào tiến trình đó cần phục vụ.

Điều phối dùng hàng đợi phản hồi nhiều cấp trở thành giải thuật điều phối CPU phổ biến nhất. Bộ điều phối này có thể được cấu hình để thích hợp với mỗi hệ thống xác định. Tuy nhiên, bộ điều phối này cũng yêu cầu một vài phương tiện chọn lựa giá trị cho tất cả tham số để xác định bộ điều phối tốt nhất. Mặc dù hàng đợi phản hồi nhiều cấp là cơ chế phổ biến nhất nhưng nó cũng là cơ chế phức tạp nhất.

9.4 ĐIỀU PHỐI ĐA BỘ XỬ LÝ

Phần trên chúng ta tập trung vào những vấn đề điều phối CPU trong một hệ thống có một bộ xử lý. Nếu có nhiều CPU, vấn đề điều phối tương ứng sẽ phức tạp hơn. Nhiều khả năng đã được thử nghiệm và như chúng ta đã thấy với điều phối CPU đơn bộ xử lý, không có giải pháp tốt nhất. Trong phần sau đây, chúng ta sẽ thảo luận vắn tắt một số vấn đề tập trung về điều phối đa bộ xử lý. Chúng ta tập trung vào những hệ thống mà các bộ xử lý của nó được xác định (hay đồng nhất) trong thuật ngữ chức năng của chúng; bất cứ bộ xử lý nào sẵn có thì có thể được dùng để chạy bất cứ tiến trình nào trong hàng đợi. Chúng ta cũng cho rằng truy xuất bộ nhớ là đồng nhất (uniform memory access - UMA). Chỉ những chương trình được biên dịch đối với tập hợp chỉ thị của bộ xử lý được cho mới có thể được chạy trên chính bộ xử lý đó.

Ngay cả trong một bộ đa xử lý đồng nhất đôi khi có một số giới hạn cho việc điều phối. Xét một hệ thống với một thiết bị nhập/xuất được gán tới một đường bus riêng của một bộ xử lý. Các tiến trình muốn dùng thiết bị đó phải được điều phối để chạy trên bộ xử lý đó, ngược lại thiết bị đó là không sẵn dùng.

Nếu nhiều bộ xử lý xác định sẵn dùng thì chia sẻ tài nguyên có thể xảy ra. Nó có thể cung cấp một hàng đợi riêng cho mỗi bộ xử lý. Tuy nhiên, trong trường hợp này, một bộ xử lý có thể rảnh với hàng đợi rỗng, trong khi bộ xử lý khác rất bận. Để ngăn chặn trường hợp này, chúng ta dùng một hàng đợi sẵn sàng chung. Tất cả tiến trình đi vào một hàng đợi và được điều phối trên bất cứ bộ xử lý sẵn dùng nào.

Trong một cơ chế như thế, một trong hai tiếp cận điều phối có thể được dùng. Trong tiếp cận thứ nhất, mỗi bộ xử lý điều phối chính nó. Mỗi bộ xử lý xem hàng đợi sẵn sàng chung và chọn một tiến trình để thực thi. Nếu chúng ta có nhiều bộ xử lý cố gắng truy xuất và cập nhật một cấu trúc dữ liệu chung thì mỗi bộ xử lý phải được lập trình rất cẩn thận. Chúng ta phải đảm bảo rằng hai bộ xử lý không chọn cùng tiến trình và tiến trình đó không bị mất từ hàng đợi. Tiếp cận thứ hai tránh vấn đề này bằng cách để cử một bộ xử lý như bộ điều phối cho các tiến trình khác, do đó tạo ra cấu trúc chủ-tớ (master-slave).

Một vài hệ thống thực hiện cấu trúc này từng bước bằng cách tất cả quyết định điều phối, xử lý nhập/xuất và các hoạt động hệ thống khác được quản lý bởi một bộ

xử lý đơn, một server chủ. Các bộ xử lý khác chỉ thực thi mã người dùng. Đa xử lý không đổi xứng đơn giản hơn đa xử lý đổi xứng vì chỉ một tiến trình truy xuất các cấu trúc dữ liệu hệ thống, làm giảm đi yêu cầu chia sẻ dữ liệu. Tuy nhiên, nó cũng không hiệu quả. Các tiến trình giới hạn nhập/xuất có thể gây thắt cổ chai (bottleneck) trên một CPU đang thực hiện tất cả các hoạt động. Điển hình, đa xử lý không đổi xứng được cài đặt trước trong một hệ điều hành và sau đó được nâng cấp tới đa xử lý đổi xứng khi hệ thống tiến triển.

CÂU HỎI VÀ BÀI TẬP

- 1 . Hãy trình bày mục tiêu của việc điều phối tiến trình.
2. Hãy vẽ sơ đồ tổ chức điều phối tiến trình trong trường hợp các tiến trình tham chiếu tới 3 tài nguyên.
3. Hãy trình bày vắn tắt các chiến lược điều phối tiến trình (không vẽ sơ đồ minh họa).
4. Giả sử trong một hệ thống chia sẻ thời gian với quantum=2, điều phối theo cơ chế không độc quyền, có các tiến trình đang hoạt động như bảng sau :

Tiến trình	P0	P1	P2	P3	P4
Thời gian CPU xử lý	10	1	2	1	5
Thời điểm vào Ready List	0	1	2	3	4
Độ ưu tiên	3	1	3	4	2

Chú ý : Độ ưu tiên 1>2>3

Hãy điều phối các tiến trình trên theo các chiến lược FCFS, Round Robin, SJF, căn cứ vào độ ưu tiên. Điền kết quả vào bảng sau:

Thời điểm cấp CPU	0	1				10	
Tiến trình được cấp CPU	P0	P1					

Tính thời gian chờ đợi chia đều cho từng tiến trình, chờ đợi trung bình.

BÀI 10. ĐỒNG BỘ HÓA TIẾN TRÌNH

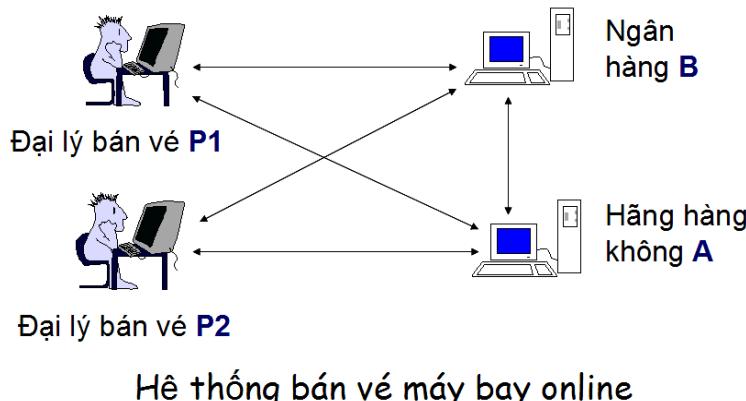
10.1 TÀI NGUYÊN GĂNG VÀ ĐOẠN GĂNG

10.1.1 Tài nguyên găng

Trong hệ điều hành đa chương, việc chia sẻ tài nguyên cho các tiến trình là cần thiết. Nhưng nếu không tổ chức tốt việc sử dụng tài nguyên chung cho các tiến trình hoạt động đồng thời, thì không những không mang lại hiệu quả mà còn làm hỏng dữ liệu của các ứng dụng.

Những tài nguyên có nguy cơ bị hư hỏng, sai lệch khi được hệ điều hành chia sẻ đồng thời cho nhiều tiến trình được gọi là *tài nguyên găng* (critical resource). Tài nguyên găng có thể là tài nguyên phần cứng hoặc tài nguyên phần mềm, có thể là tài nguyên phân chia được hoặc không phân chia được, nhưng đa số thường là tài nguyên phân chia được như là: các biến chung, các file chia sẻ.

Ví dụ sau đây cho thấy hậu quả của việc sử dụng tài nguyên găng trong các chương trình có các tiến trình hoạt động đồng thời. Giả sử có một hệ thống bán vé máy bay của hãng hàng không A nhằm mục đích có thể bán vé máy bay tại nhiều đại lý như hình dưới.



Qui trình bán vé tại đại lý như sau

Bước b1. Hỏi A số chỗ còn trống SCA : N:=SCA

Bước b2. if N=0 then dừng else đi tiếp

Bước b3. Chuyển tiền mua vé T vào tài khoản TKA của A tại B: TKA:=TKA+T

Bước b4. In hóa đơn cho khách hàng

Bước b5. Yêu cầu A ghi vào SCA: SCA:=SCA-1

Xem xét tình huống chuyến bay chỉ còn 1 vé (SCA=1) nhưng lại có hai đại lý P1 và P2 cùng bán vé. Quá trình bán vé của hai đại lý được mô tả trong hình dưới (ta sẽ không chú ý đến bước 4)

Time	P1	P2	A
0			SCA=1
1	b1: N:=SCA		
2	b2: (N=1)	b1: N:=SCA	
3	b3:	b2: (N=1)	
4	b4:	b3:	
5	b5:	b4:	SCA=SCA-1=0
6		b5:	SCA=SCA-1=-1

Hệ thống đã bán dư 1 vé máy bay, dẫn đến tranh chấp chỗ ngồi trên máy bay, hãng hàng không bị mất uy tín và có thể phải bồi thường cho khách hàng. Nguyên nhân sai sót là quá trình truy xuất vào SCA của P1 và P2 trong khi bán vé đan xen lẫn nhau. Trong trường hợp này tài nguyên găng là biến SCA.

Để ngăn chặn các tình huống trên hệ điều hành phải thiết lập cơ chế độc quyền truy xuất trên tài nguyên dùng chung. Nếu có nhiều tiến trình đồng thời yêu cầu truy xuất tài nguyên dùng chung thì chỉ có một tiến trình được chấp nhận truy xuất, các tiến trình khác phải xếp hàng chờ để được truy xuất sau.

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

Các tình huống tương tự như thế có thể xảy ra khi có nhiều tiến trình đọc và ghi dữ liệu trên cùng một vùng nhớ chung được gọi là *tình huống xung đột* (race condition).

10.1.2 Đoạn găng

Trong các tiến trình, đoạn mã có tác động đến các tài nguyên găng được gọi là *đoạn găng* (CriticalSection) hay *miền găng*, ví dụ như các đoạn mã dùng để truy cập đến các vùng nhớ chia sẻ hay các tập tin chia sẻ. Trong ví dụ ở mục trên, đoạn găng là đoạn chương trình từ bước b1 đến bước b5.

Để hạn chế các lỗi có thể xảy ra do sử dụng tài nguyên găng, hệ điều hành phải điều khiển các tiến trình sao cho, tại một thời điểm chỉ có một tiến trình nằm trong đoạn găng, nếu có nhiều tiến trình cùng muốn vào (để thực hiện) đoạn găng thì chỉ có một tiến trình được vào, các tiến trình khác phải chờ, một tiến trình khi ra khỏi đoạn găng phải báo cho hệ điều hành hoặc các tiến trình khác biết để các tiến trình này vào đoạn găng. Vì vậy, cấu trúc chương trình khi có đoạn găng như sau

< noncritical section>	
<enter critical section>	{kiểm tra và xác lập quyền vào đoạn găng}
<critical section>	
<exit critical section>	{xác nhận khi rời đoạn găng}
< noncritical section>	

Mỗi tiến trình trước khi vào đoạn găng có thủ tục <enter critical section> để kiểm tra và thiết lập quyền vào đoạn găng, báo cho các tiến trình khác biết là mình đang ở trong đoạn găng. Để ra khỏi đoạn găng mỗi tiến trình phải thực hiện thủ tục ,<exit critical section> để báo cho hệ điều hành hoặc các tiến trình khác biết là mình đã ra khỏi đoạn găng.

10.1.3 Yêu cầu của đồng bộ hóa tiến trình

Trước hết cần lưu ý rằng, đồng bộ hóa tiến trình là sự phối hợp giữa phần cứng, hệ điều hành, ngôn ngữ lập trình và người lập trình, trong đó nhiệm vụ chính là của hệ điều hành và người lập trình. Phần cứng, hệ điều hành và ngôn ngữ lập trình cung cấp các công cụ để hệ điều hành và người lập trình tổ chức sơ đồ đồng bộ hóa. Hệ điều

hành sẽ giám sát và tổ chức thực hiện các sơ đồ này. Cho dù nhiệm vụ đồng bộ hóa là của thành phần nào, thì cũng phải đạt được các yêu cầu sau:

1. *Loại trừ lẫn nhau* (Mutual Exclusion): Tại một thời điểm không thể có hai tiến trình cùng nắm trong đoạn găng ứng với một tài nguyên găng. Nếu có nhiều tiến trình đồng thời cùng xin được vào đoạn găng thì chỉ có một tiến trình được phép vào đoạn găng, các tiến trình khác phải xếp hàng chờ trong hàng đợi.
2. *Tiến trình* (Progress): Chỉ những tiến trình đang có nhu cầu vào đoạn găng mới có thể tham gia vào việc quyết định tiến trình nào sẽ đi vào đoạn găng.
3. *Chờ đợi có giới hạn* (bounded wait): Không có tiến trình nào được phép ở lâu vô hạn trong đoạn găng và không có tiến trình nào phải chờ quá lâu để vào đoạn găng.

Nguyên lý cơ bản của đồng bộ hóa tiến trình là tổ chức truy xuất độc quyền trên tài nguyên găng, nhưng sự độc quyền này có thể dẫn đến hai tình trạng sau.

- *Tắc nghẽn* (Deadlock) trong hệ thống. Chúng ta sẽ tìm hiểu về tắc nghẽn trong các phần sau.
- *Tiến trình bị đói* (Stravation) tài nguyên. Xét ví dụ sau, giả sử có 3 tiến trình P1, P2, P3. Mỗi tiến trình đều cần truy xuất định kỳ đến tài nguyên găng R. Xét trường hợp P1 đang sở hữu tài nguyên R còn hai tiến trình P2 và P3 phải chờ đợi. Khi P1 ra khỏi đoạn găng của nó, cả P2 lẫn P3 đều có thể được chấp nhận truy xuất đến R. Giả sử P3 được truy xuất R, sau đó trước khi P3 kết thúc đoạn găng của nó P1 lại một lần nữa cần truy xuất. Giả sử P1 lại được truy xuất R sau khi P3 kết thúc đoạn găng. Nếu P1, P3 thay nhau nhận được quyền truy xuất R thì P2 hầu như không thể truy cập đến tài nguyên (bị đói tài nguyên), dù không có tắc nghẽn hệ thống.

10.2 GIẢI PHÁP PETERSON

Giải thuật Peterson giải quyết vấn đề đồng bộ giữa hai tiến trình P0 và P1. Các tiến trình chia sẻ hai biến chung:

```
Int turn; // đến phiên ai, turn=i (i=0,1) nghĩa là Pi được quyền vào đoạn găng
Boolean interest[2]; // interest[i]=True (i=0,1) nghĩa là Pi muốn vào đoạn găng.
interest[0]=False; interest[1]=False; turn=0;
```

Để có thể vào được đoạn găng, tiến trình Pi phải đặt giá trị interest[i]=True để xác định Pi muốn vào đoạn găng, sau đó đặt turn=1-i để nhường quyền vào đoạn găng cho tiến trình kia. Nếu tiến trình kia không quan tâm đến việc vào đoạn găng (interest[1-i]=False), thì Pi có thể vào miễn găng, ngược lại, Pi phải chờ đến khi interest[1-i]=False. Khi tiến trình Pi rời khỏi miễn găng, nó đặt lại giá trị cho interest[i]= False. Quá trình trên của Pi được mô phỏng bằng đoạn mã giả sau

```
while (TRUE) {
    interest[i] = False;
    <Noncritical-section>;
    interest[i] = True;
    turn = 1-i;
    while ((turn == 1-i) and (interest[1-i]==True));
    <critical-section>;
    interest[i] = False;
    <Noncritical-section>;
}
```

Giải thuật này đảm bảo cả ba yêu cầu về đồng bộ hóa.

Yêu cầu loại trừ lẫn nhau được đảm bảo : Chú ý rằng Pi chỉ có thể đi vào đoạn găng của mình khi interest[1-i]==false hoặc turn==i, đồng thời interest[i]==True. Giả sử cả P0 và P1 đều đang trong đoạn găng thì suy ra interest[0]==True và interest[1]==True. Từ đẳng thức interest[0]==True suy ra turn==1, đồng thời từ interest[1]==True suy ra turn==0, như vậy turn vừa bằng 1 lại vừa bằng 0, mâu thuẫn. Vậy không thể có trường hợp P0 và P1 cùng ở trong đoạn găng.

Yêu cầu tiến trình được đảm bảo : Rõ ràng Pi chỉ tham gia vào việc quyết định tiến trình nào được vào đoạn găng bằng cách nhường quyền vào đoạn găng cho tiến trình kia (gán turn=1-i), khi đó nó đang có nhu cầu vào đoạn găng (interest[i]==True).

Yêu cầu chờ đợi có giới hạn được đảm bảo : Chú ý rằng Pi chỉ bị kẹt trong vòng lặp while khi interest[1-i]==True và turn==1-i. Khi đó tiến trình kia bắt đầu vào hoặc đang ở trong đoạn găng, ngay khi tiến trình kia ra khỏi đoạn găng, nó gán lại interest[1-i]==False, lúc đó Pi sẽ ra khỏi vòng lặp while và đi vào đoạn găng.

10.3 CÁC GIẢI PHÁP PHẦN CỨNG

10.3.1 Giải pháp cấm ngắt

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

Có thể mô tả quá trình trên bằng đoạn mã giả sau

<i>CLI;</i>	<i>{cấm ngắt trước khi vào đoạn găng}</i>
<i><Đoạn găng của Pi>;</i>	
<i>STI;</i>	<i>{mở ngắt khi ra khỏi đoạn găng }</i>
<i><Đoạn không găng>;</i>	

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

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

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

10.3.2 Dùng chỉ thị TSL

Một số bộ xử lý cung cấp một chỉ thị đặc biệt cho phép kiểm tra và cập nhật nội dung một vùng nhớ trong một thao tác không thể phân chia được, gọi là TSL (Test and Set lock) . TSL được định nghĩa như sau :

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

Để đồng bộ hóa tiến trình với chỉ thị TSL, chương trình phải sử dụng biến chia sẻ Lock, khởi tạo bằng 0. Theo đó, mỗi tiến trình trước khi vào đoạn găng phải kiểm tra giá trị của Lock. Nếu Lock=0 thì vào đoạn găng. Nếu Lock=1 thì phải đợi cho đến khi Lock=0. Như vậy, trước khi vào đoạn găng tiến trình phải gọi hàm TestAndSetLock, để kiểm tra giá trị trả về của hàm này.

Nếu giá trị trả về bằng False, nghĩa là đang có một tiến trình trong đoạn găng, thì phải chờ cho đến khi giá trị trả về bằng True, có một tiến trình vừa ra khỏi đoạn găng.

Nếu giá trị trả về bằng True, thì tiến trình sẽ vào đoạn găng để sử dụng tài nguyên găng. Khi ra khỏi đoạn găng thì tiến trình phải đặt lại giá trị của Lock=0, để các tiến trình khác có thể vào đoạn găng.

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

Có thể mô tả quá trình này bằng đoạn mã giả sau:

```
While (TestAndSetlock(lock)) do;
```

<Đoạn găng của P>;

Lock := 0;

<Đoạn không găng>;

Giải pháp này đơn giản, dễ cài đặt nhưng cần phải có sự hỗ trợ của vi xử lý. Ngoài ra nó còn một hạn chế lớn là gây lãng phí thời gian xử lý của CPU do tồn tại hiện tượng chờ đợi tích cực (CPU hoạt động để kiểm tra điều kiện chờ trong khi chờ đợi mà không làm gì cả) thể hiện qua câu lệnh *While (TestAndSetlock(lock)) do;*.

10.3.3 Nhận xét

Việc sử dụng các chỉ thị phần cứng đặc biệt để tổ chức đồng bộ hóa tiến trình có những thuận lợi và bất lợi sau đây:

Thuận lợi

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

Bất lợi

- Trong khi một tiến trình đang chờ đợi được vào đoạn găng thì nó tiếp tục làm tốn thời gian xử lý của CPU, mà ta gọi là chờ đợi tích cực.
- Việc đói tài nguyên có thể xảy ra. Khi một tiến trình rời khỏi đoạn găng, bộ phận đồng bộ hóa tiến trình phải chọn một trong số nhiều tiến trình đang chờ để vào đoạn găng. Việc chọn này có thể dẫn đến hiện tượng có một tiến trình đợi mãi mà không thể vào đoạn găng được.
- Tắc nghẽn có thể xảy ra. Hãy xét tình huống sau trên hệ thống một CPU. Tiến trình P1 thực thi chỉ thị đặc biệt TestAndSetLock và vào đoạn găng của nó. P1 sau đó bị ngắt để nhường CPU cho P2, P2 là tiến trình có độ ưu tiên cao hơn. Nếu như P2 cũng định sử dụng tài nguyên như P1, P2 sẽ bị từ chối truy xuất bởi cơ chế độc quyền truy xuất tài nguyên. Do đó P2 sẽ đi vào vòng lặp busy-waitting. Tuy nhiên, P1 sẽ không bao giờ được cấp CPU để tiếp tục vì nó có độ ưu tiên thấp hơn so với P2.

10.4 SEMAPHORE

Giải pháp này được Dijkstra đề xuất vào năm 1965. Semaphore (sự đánh tín hiệu bằng cờ) được định nghĩa để sử dụng trong đồng bộ hóa tiến trình như sau:

- Ứng với mỗi tài nguyên găng CR, tạo một biến nguyên S với giá trị khởi đầu là 1 và một waiting list F lưu các tiến trình chờ được cấp quyền truy xuất CR
- Trong HDH, thiết lập 2 thủ tục nguyên tử Down(P,S,F) và Up(S,F) được mô tả tại các slide sau, P là một tiến trình muốn truy xuất CR
- Ứng với mỗi tài nguyên găng CR, Semaphore S là một biến nguyên với giá trị khởi tạo là 1.
- Ứng với S có một hàng đợi F(S) để lưu các tiến trình đang chờ được cấp quyền truy xuất CR.
- Trong HDH, thiết lập 2 thủ tục nguyên tử Down(P,S,F) và Up(S,F). Chỉ có Down và Up được tác động đến semaphore S. Down giảm S xuống một đơn vị, Up tăng S lên một đơn vị. Down và UP được mô tả như sau.

Procedure Down(P,S,F);

Begin

S := S -1;

If S < 0 Then

Begin

<Chuyển P sang trạng thái Waiting>

<Đưa P vào F>

End;

End;

Procedure Up(S,F);

Begin

S := S +1;

If S <= 0 Then

Begin

<Chọn một P trong F>

```

<Chuyển P sang trạng thái Ready>
<Đưa P vào Ready List>
End;
End;

```

Mỗi tiến trình trước khi vào đoạn găng thì phải gọi Down để kiểm tra và xác lập quyền vào đoạn găng. Khi tiến trình ra khỏi đoạn găng phải gọi Up để kiểm tra xem có tiến trình nào đang đợi trong hàng đợi hay không, nếu có thì đưa tiến trình trong hàng đợi vào đoạn găng. Quá trình này được mô tả như sau

```

<noncritical section>
down (P,S,F);
<critical section>
Up(S,F);
<noncritical section>

```

Cần lưu ý rằng Down và Up là các thủ tục của hệ điều hành, nên hệ điều hành đã cài đặt cơ chế độc quyền cho nó, tức là các lệnh bên trong nó không thể tách rời nhau. Sử dụng semaphore để đồng bộ hóa tiến trình, mang lại những thuận lợi sau:

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

Tuy nhiên, vì Down và Up là các thủ tục của hệ điều hành nên việc đồng bộ hóa sẽ bị thay đổi khi thay đổi hệ điều hành. Đây là một trở ngại của việc sử dụng semaphore.

Ví dụ

Bảng dưới đây cho thấy quá trình sử dụng semaphore S để đồng bộ hóa 4 tiến trình A,B,C,D lần lượt muốn truy cập cùng một tài nguyên găng

Time	Process	Down/ Up	S	Ready/Running	Waiting List F
0	-	-	1	-	-
1	A	Down	0	A	-
2	B	Down	-1	A	B
3	C	Down	-2	A	B-C
4	A	Up	-1	B	C
5	D	Down	-2	B	C-D
6	B	Up	-1	C	D
7	C	Up	0	D	-
8	D	Up	1	-	-

10.5 MONITORS

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

- Các biến và cấu trúc dữ liệu bên trong monitor chỉ có thể được thao tác bởi các thủ tục được định nghĩa bên trong monitor đó.
- Một tiến trình muốn vào monitor phải gọi một thủ tục của monitor đó.
- Tại một thời điểm, chỉ có một tiến trình duy nhất được hoạt động bên trong monitor.

Các tiến trình khác đã gọi monitor phải hoãn lại để chờ monitor rảnh.

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

Với tính chất thứ 3 monitor có khả năng thực hiện các cơ chế độc quyền, các biến trong monitor có thể được truy xuất chỉ bởi một tiến trình tại một thời điểm. Như vậy

các cấu trúc dữ liệu dùng chung bởi các tiến trình có thể được bảo vệ bằng cách đặt chúng bên trong monitor. Nếu dữ liệu bên trong monitor là tài nguyên găng thì monitor cung cấp sự độc quyền trong việc truy xuất đến tài nguyên găng đó.

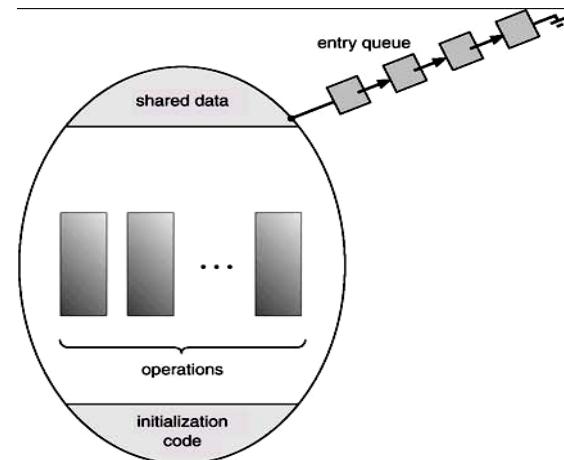
Monitor cung cấp các công cụ đồng bộ hóa để người lập trình sử dụng trong các sơ đồ đồng bộ hóa. Công cụ đồng bộ hóa được định nghĩa như sau: Trong một monitor, có thể định nghĩa các *biến điều kiện* và hai thủ tục kèm theo là *Wait* và *Signal*, chỉ có *wait* và *signal* được tác động đến các biến điều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

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

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

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

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



Trình biên dịch chịu trách nhiệm thực hiện việc truy xuất độc quyền đến dữ liệu trong monitor. Để thực hiện điều này, hệ điều hành dùng một semaphore nhị phân. Mỗi monitor có một hàng đợi toàn cục lưu các tiến trình đang chờ được vào monitor, ngoài ra mỗi biến điều kiện c cũng gắn với một hàng đợi f(c).

Với mỗi tài nguyên gắp, có thể định nghĩa một monitor trong đó đặc tả tất cả các thao tác trên tài nguyên này với một số điều kiện nào đó. Sau đây là cấu trúc một Monitor. Cấu trúc của monitor có thể được mô tả như sau

```

Monitor      <Tên monitor>

Condition    <Danh sách các biến điều kiện>;

Procedure Action1();           {thao tác 1}

Begin
    .....
End;

.....
Procedure Actionk();           {thao tác k}

Begin
    .....
End;

.....
End monitor;

```

Mỗi tiến trình muốn sử dụng tài nguyên gắp chỉ có thể thao tác thông qua các thủ tục bên trong monitor. Đoạn chương trình vào và ra đoạn gắp của tiến trình P được mô tả như sau

```

<Đoạn không gắp của P>;
<monitor>.Action;                  {Đoạn gắp của P};
<Đoạn không gắp của P>;

```

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

10.6 GIẢI PHÁP TRAO ĐỔI THÔNG ĐIỆP

Khi các tiến trình có sự tương tác với tiến trình khác, hai yêu cầu cơ bản cần phải được thỏa mãn đó là: đồng bộ hóa (synchronization) và giao tiếp (communication). Các tiến trình phải được đồng bộ để thực hiện độc quyền truy xuất tài nguyên găng. Các tiến trình hợp tác có thể cần phải trao đổi thông tin. Một hướng tiếp cận để cung cấp cả hai chức năng đó là phương pháp truyền thông điệp (message passing). Truyền thông điệp có ưu điểm là có thể thực hiện được trên cả hai hệ thống một CPU và nhiều CPU, trên cả hệ thống tập trung lẫn hệ thống phân tán, đặc biệt là trong hệ thống client-server.

Các hệ thống truyền thông điệp có thể có nhiều dạng, tuy nhiên chúng ta sẽ chỉ đề cập một dạng chung nhất. Các hàm của truyền thông điệp trên thực tế có dạng tương tự như hai hàm sau:

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

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

Tiến trình gửi thông tin dưới dạng một thông điệp đến một tiến trình khác (được nhận biết bởi tham số destination) bằng hàm Send. Tiến trình nhận thông điệp bằng hàm Receive từ một tiến trình khác được nhận biết bởi tham số source. Tiến trình gọi Receive phải chờ cho đến khi nhận được thông điệp từ tiến trình source thì mới có thể tiếp tục được. Việc sử dụng Send và Receive để tổ chức đồng bộ hóa tiến trình được thực hiện như sau:

- Có một tiến trình (process controller) kiểm soát việc sử dụng tài nguyên găng.
- Có nhiều tiến trình khác yêu cầu sử dụng tài nguyên găng này.
- Tiến trình P có yêu cầu tài nguyên găng sẽ gửi một thông điệp yêu cầu (request message) đến tiến trình kiểm soát và sau đó chuyển sang trạng thái blocked cho đến khi nhận được một thông điệp chấp nhận (accept message) cho truy xuất từ tiến trình kiểm soát tài nguyên găng.

- Khi ra khỏi đoạn găng, tiến trình P gửi một thông điệp kết thúc (end message) đến tiến trình kiểm soát để báo kết thúc việc truy xuất tài nguyên.
- Tiến trình kiểm soát, khi nhận được thông điệp yêu cầu tài nguyên găng, nó sẽ chờ cho đến khi tài nguyên găng sẵn sàng để cấp phát thì gửi thông điệp chấp nhận một trong các tiến trình đang chờ để tiến trình truy xuất tài nguyên.

Quá trình trên được mô tả bằng mã giả như sau

```

Send(process controler, request message);
Receive(process controler, accept message );
<Đoạn găng của P>;
Send(process controler, end message);
<Đoạn không găng của P>;

```

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

10.7 VÍ DỤ KINH ĐIỂN

10.7.1 Bài toán nhà sản xuất - khách hàng

Ứng dụng này có hai tiến trình chính đó là, tiến trình nhà sản xuất (Producer) và tiến trình khách hàng (Consumer), hai tiến trình này hoạt động đồng thời với nhau và cùng chia sẻ một bộ đệm (Buffer) có kích thước giới hạn, chỉ có 3 phần tử. Tiến trình Producer tạo ra dữ liệu và đặt vào Buffer, tiến trình Consumer lấy dữ liệu từ Buffer ra để xử lý. Rõ ràng hệ thống này cần phải có các ràng buộc sau:

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

Hãy dùng các giải pháp Semaphore, Monitor, Message để tổ chức đồng bộ hóa cho các tiến trình Producer và Consumer trong bài toán trên.

Giải pháp dùng Semaphore

Gải pháp này phải sử dụng 3 Semaphore sau

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

Quá trình đồng bộ hóa được mô tả bằng mã giả như sau:

```

Program Producer/Consumer;
Var      Full, Empty, Mutex: Semaphore;
{-----}

Procedure Producer();
Begin
    Repeat
        < Tạo dữ liệu>;
        Down(empty);           {kiểm tra xem buffer còn chỗ trống ?}
        Down(mutex);          {kiểm tra và xác lập quyền truy xuất Buffer}
        <Đặt dữ liệu vào Buffer>;
        Up(mutex);            {kết thúc truy xuất buffer}
        Up(Full);             {đã có 1 phần tử dữ liệu trong Buffer}
        Until .F.

    End;
{-----}

Procedure Consumer();
Begin
    Repeat
        Down(full);           {còn phần tử dữ liệu trong Buffer?}
        Down(mutex);          {kiểm tra và xác lập quyền truy xuất Buffer}
        <Nhận dữ liệu từ đệm>;
        Up(mutex);            {kết thúc truy xuất buffer}
        Up(empty);             {đã lấy 1 phần tử dữ liệu trong Buffer}
    End;
{-----}

```

```

Until .F.

End;

{-----}

BEGIN

Full = 0; Empty = 3; Mutex = 1;
Produc er();
Consumer();

END.

{-----}

```

Giải pháp dùng Monitor

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

Quá trình đồng bộ hóa được mô tả bằng mã giả như sau:

```

Program Producer/Consumer;
Monitor ProducerConsumer;
Condition Full, Empty;
Var Count: Integer;      {để đếm số phần tử dữ liệu được đưa vào Buffer}
    N: Interger;        {số phần tử của Buffer}
{ ----- }

Procedure Enter();
Begin
    If Count = N Then Wait(Full);   {nếu Buffer đầy thì đợi }
    <Đặt dữ liệu vào đệm>;           {Buffer rỗng}
    Count := Count + 1;
    If Count = 1 Then Signal(Empty);   {nếu Buffer không rỗng thì}
                                                {báo cho consumer biết}
End;
{-----}

Procedure Remove();
Begin

```

```

If Count = 0 Then Wait(Empty); {nếu Buffer rỗng thì đợi đầy}
<Nhận dữ liệu từ đệm>;
Count := Count - 1;
If Count = N - 1 Then Signal(Full); {nếu Buffer không đầy thì}
End;                                { báo cho producer}
Endmonitor;
{-----}

BEGIN

Count = 0; N = 3;

ParBegin
Procedure Producer();
Begin
Repeat
<Tạo dữ liệu>;
Producer/Consumer.Enter;
Until .F.

End;
{-----}

Procedure Consumor();
Begin
Repeat
Producer/Consumer.Remove;
<Xử lý dữ liệu>;
Until .F.

End;
Parend
END.

{-----}

```

Giải pháp dùng Message

Với giải pháp này chương trình dùng thông điệp empty. Empty hàm ý có một chỗ trống trên Buffer. Khi khởi tạo, tiến trình Consumer gửi ngay N thông điệp empty đến

tiến trình Producer. Tiến trình Producer tạo ra một dữ liệu mới và chờ đến khi nhận được một thông điệp empty từ Consumer thì gửi ngược lại cho Consumer một thông điệp có chứa dữ liệu mà nó tạo ra. Sau khi gửi đi thông điệp empty, tiến trình Consumer sẽ chờ để nhận thông điệp chứa dữ liệu từ tiến trình Producer. Sau khi xử lý xong dữ liệu thì consumer gửi lại một thông điệp empty đến tiến trình Producer.

Quá trình đồng bộ hóa được mô tả bằng mã giả như sau:

```

Program Producer/Consumer;
Var
    BufferSize: integer;           {kích thước Buffer}
    M, m': Message;
{ ----- }

BEGIN
    BufferSize = N;
    ParBegin
        Procedure Producer();
        Begin
            Repeat
                <Tạo dữ liệu>;
                Receive(Consumer,m);
                <Tạo thông điệp dữ liệu>
                Send(Consumer,m)
            Until .F.
        End;
    { ----- }

    Procedure Consumer ()
    Var I:integer;
    Begin
        For I := 0 to N Do Send(Producer ,m);
        Repeat
            Receive(Producer ,m);
            <Lấy dữ liệu từ thông điệp>
    End;

```

```

Send (Producer,m);
<Xử lý dữ liệu >
Until .F.
End.
Parend
END.

{-----}

```

10.7.2 Bài toán đọc/ghi

Trong môi trường hệ điều hành đa nhiệm, có thể tồn tại các file chia sẻ, ví dụ như các file cơ sở dữ liệu. Nhiều tiến trình hoạt động đồng thời trong hệ thống có thể chia sẻ sử dụng file cơ sở dữ liệu này. Tiến trình cần đọc nội dung của file cơ sở dữ liệu được gọi là tiến trình Reader. Tiến trình cần cập nhật thông tin vào file cơ sở dữ liệu được gọi là tiến trình Writer. Trong hệ thống này, công tác đồng bộ hóa tiến trình cần phải thực hiện các ràng buộc sau:

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

Giải pháp dùng Semaphore

Giải pháp này sử dụng một biến chung RC và hai semaphore là Mutex và DB.

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

Quá trình đồng bộ hóa được mô tả bằng mã giả như sau:

```

Program Producer/Consumer;
Const
    Mutex: Seamafore = 1;

```

```
Db      : Seamafore = 1;
Rc      : byte = 0;
{-----}

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

{-----}

Giải pháp dùng Monitor

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

Quá trình đồng bộ hóa được mô tả bằng mã giả như sau:

```

Program Producer/Consumer;
Monitor Readerwriter
Condition Okwrite,Okread
Var
    Rc: integer;
    Busy: boolean = False;
{-----}
Procedure Beginread()
Begin
    If (busy) then wait(okread);
    Rc = Rc+1;
    Signal(okread);
End;
Procedure Finishread()
Begin
    Rc = Rc - 1;
    If Rc = 0 Then Wait(okwrite);
End;
Procedure Beginwrite();
Begin
    Rc = Rc - 1;
    If (busy) or (Rc <> 0) Then Wait(okwrite);
    Busy = True;
End;
Procedure FinishWrite()
Begin
    Busy = False;
    If (Okread) Then Signal(okread)
    Else Signal(okwrite);
End;
Endmonitor.

```

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

{-----}
```

Giải pháp dùng Message

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

Quá trình đồng bộ hóa được mô tả bằng mã giả như sau:

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

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

CÂU HỎI VÀ BÀI TẬP

1. Hãy kể tên các cơ chế liên lạc giữa các tiến trình.
2. Giao thức truyền thông TCP ứng với cơ chế liên lạc nào? Hãy mô tả quá trình liên lạc giữa tiến trình của một web browser với một tiến trình web server, biết rằng web server sử dụng cổng 80.
3. Hãy trình bày một ví dụ về tình huống xung đột (race condition)
4. Hãy trình bày khái niệm miền găng (critical section), các yêu cầu để giải quyết tốt vấn đề đồng bộ hóa.
5. Hãy trình bày giải pháp Peterson trong trường hợp chỉ có 2 tiến trình P0 và P1 có yêu cầu vào miền găng.
6. Hãy trình bày giải pháp Semaphores.

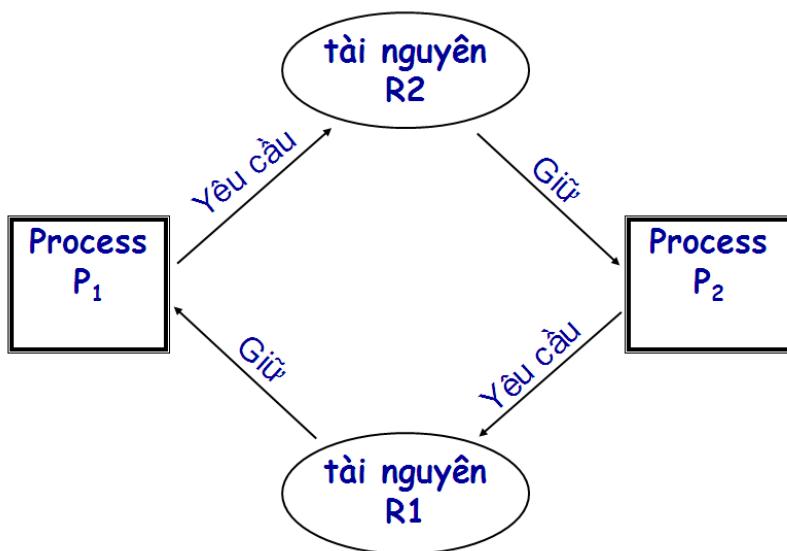
BÀI 11. TẮC NGHẼN

11.1 KHÁI NIỆM VÀ VÍ DỤ

Tắc nghẽn (DeadLock) là trạng thái trong hệ thống có ít nhất hai tiến trình đang dừng chờ lẫn nhau và chúng không thể chạy tiếp được. Sự chờ đợi này có thể kéo dài vô hạn nếu không có sự tác động từ bên ngoài.

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

Ví dụ 1: Giả sử có hai tiến trình P₁ và P₂ hoạt động đồng thời trong hệ thống. Tiến trình P₁ đang giữ tài nguyên gǎng R₁ và xin được cấp tài nguyên gǎng R₂ để tiếp tục hoạt động, trong khi đó tiến trình P₂ đang giữ tài nguyên R₂ và xin được cấp R₁ để tiếp tục hoạt động. Trong trường hợp này cả P₁ và P₂ sẽ không tiếp tục hoạt động được. Như vậy P₁ và P₂ rơi vào trạng thái tắc nghẽn. Ví dụ này có thể được minh họa bởi sơ đồ ở hình dưới.



Ví dụ 2: Trong các ứng dụng cơ sở dữ liệu, một chương trình có thể khóa một vài record mà nó sử dụng để tiến hành cập nhật dữ liệu. Nếu tiến trình P1 khóa record R1, tiến trình P2 khóa record R2, và rồi sau đó mỗi tiến trình lại cố gắng khóa record của một tiến trình kia, tắc nghẽn sẽ xảy ra.

Trong trường hợp của ví dụ 1 ở trên: hai tiến trình P1 và P2 sẽ rơi vào trạng thái tắc nghẽn, nếu không có sự can thiệp của hệ điều hành. Để phá bỏ tắc nghẽn này hệ điều hành có thể cho tạm dừng tiến trình P1 để thu hồi lại tài nguyên R1, lấy R1 cấp cho tiến trình P2 để P2 hoạt động và kết thúc, sau đó thu hồi cả R1 và R2 từ tiến trình P2 để cấp cho P1 và tái kích hoạt P1 để P1 hoạt động trở lại. Như vậy sau một khoảng thời gian cả P1 và P2 đều ra khỏi tình trạng tắc nghẽn.

Khi hệ thống xảy ra tắc nghẽn nếu hệ điều hành không kịp thời phá bỏ tắc nghẽn thì hệ thống có thể rơi vào tình trạng treo toàn bộ hệ thống. Như trong trường hợp tắc nghẽn ở ví dụ 1, nếu sau đó có tiến trình P3, đang giữ tài nguyên R3, cần R2 để tiếp tục thì P3 cũng sẽ rơi vào tình trạng tắc nghẽn, rồi sau đó nếu có tiến trình P4 cần tài nguyên R1 và R3 để tiếp tục thì P4 cũng rơi vào tình trạng tắc nghẽn như P3, ... cứ thế dần dần có thể dẫn đến một thời điểm tất cả các tiến trình trong hệ thống đều rơi vào tình trạng tắc nghẽn. Như vậy hệ thống sẽ bị treo hoàn toàn.

11.2 ĐIỀU KIỆN CẦN CỦA TẮC NGHẼN

Năm 1971, Coffman đã chứng minh rằng khi xảy ra tắc nghẽn thì bốn điều kiện sau sẽ xuất hiện đồng thời.

- Độc quyền sử dụng (Mutual Exclusion):** Các tiến trình có giữ ít nhất một tài nguyên không thể chia sẻ, tức là loại tài nguyên mà tại mỗi thời điểm chỉ có một tiến trình được sử dụng. Nếu có một tiến trình khác yêu cầu tài nguyên đó thì nó phải chờ cho đến khi tài nguyên được giải phóng.
- Giữ và đợi (Hold and Wait):** Có ít nhất một tiến trình đang chiếm giữ một số tài nguyên, lại xin cấp phát thêm tài nguyên mới mà tài nguyên này đang bị chiếm giữ bởi một tiến trình khác.
- Không thể thu hồi (No preemption):** Trong các tài nguyên bị các tiến trình chiếm giữ, có tài nguyên mà hệ điều hành không có quyền thu hồi tài nguyên đó từ tiến trình đang chiếm giữ nó.

4. *Đợi luẩn quẩn* (Circular wait): Có một dãy các tiến trình P1, P2,..., Pn mà P1 đang đợi tài nguyên do P2 chiếm giữ, P2 đang đợi tài nguyên do P3 chiếm giữ, ... Pn đang đợi tài nguyên do P1 chiếm giữ. Tức là các tiến trình đợi tài nguyên do tiến trình khác giữ tạo thành một vòng luẩn quẩn.

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

Bốn điều kiện trên được áp dụng để phòng chống tắc nghẽn. Nói chung, có thể giải quyết vấn đề tắc nghẽn theo một trong ba hướng

- Sử dụng một giao thức để ngăn chặn hay tránh tắc nghẽn, đảm bảo rằng hệ thống sẽ không bao giờ đi vào trạng thái tắc nghẽn.
- Cho phép hệ thống đi vào trạng thái tắc nghẽn, phát hiện nó và phục hồi hệ thống.
- Bỏ qua vấn đề này, coi như tắc nghẽn không bao giờ xảy ra trong hệ thống. Giải pháp này được dùng trong nhiều hệ điều hành, kể cả UNIX và Windows, khi đó lập trình viên ứng dụng phải viết các đoạn chương trình quản lý tắc nghẽn.

11.3 NGĂN CHẶN TẮC NGHẼN

Ngăn chặn tắc nghẽn (Deadlock Prevention) là cung cấp các phương thức đảm bảo rằng ít nhất một trong bốn điều kiện cần của tắc nghẽn không xảy ra. Các phương thức ngăn chặn tắc nghẽn được trình bày trong mục này đều khó thực hiện và có thể làm cho việc sử dụng tài nguyên bị chậm và thông lượng hệ thống bị giảm.

Ngăn chặn điều kiện loại trừ lẫn nhau

Gần như không thể ngăn chặn điều kiện độc quyền sử dụng vì sự độc quyền truy xuất là cần thiết đối với một số tài nguyên như các biến chung, các tập tin chia sẻ. Điều này phụ thuộc vào bản chất của tài nguyên chứ không phụ thuộc vào hệ thống.

Tuy nhiên, đối với một số tài nguyên, ví dụ như máy in, hệ điều hành có thể sử dụng kỹ thuật SPOOL (Simultaneous Peripheral Operation Online) để tạo ra nhiều tài nguyên ảo cung cấp đồng thời cho các tiến trình.

Tất nhiên, có những tài nguyên không đòi hỏi đòi hỏi điều kiện độc quyền truy xuất, ví dụ như các tập tin chỉ đọc, những tài nguyên này không thể gây ra tình trạng tắc nghẽn. Nếu nhiều tiến trình cố gắng mở một tập tin chỉ đọc tại cùng một thời điểm thì chúng có thể được gán truy xuất tập tin cùng lúc.

Ngăn chặn điều kiện giữ và đợi

Để ngăn chặn điều kiện giữ-và-đợi không xuất hiện trong hệ thống, phải đảm bảo rằng bất cứ khi nào một tiến trình yêu cầu tài nguyên, nó phải không giữ bất cứ tài nguyên nào khác.

Một giao thức có thể được dùng là đòi hỏi mỗi tiến trình phải yêu cầu và được cấp phát tất cả tài nguyên trước khi nó bắt đầu thực thi. Chúng ta có thể cài đặt sự giao thức này bằng cách yêu cầu các lời gọi hệ thống về yêu cầu tài nguyên cho một tiến trình được thi hành trước tất cả các lời gọi hệ thống khác.

Một giao thức khác có thể được sử dụng là cho phép một tiến trình yêu cầu tài nguyên chỉ khi tiến trình này không có tài nguyên nào nhưng tiến trình có thể chỉ yêu cầu một số tài nguyên (không phải yêu cầu tất cả tài nguyên) và dùng chúng. Tuy nhiên, trước khi nó có thể yêu cầu bổ sung tài nguyên, nó phải giải phóng tất cả tài nguyên mà nó hiện đang được cấp phát.

Để thấy rõ sự khác nhau giữa hai giao thức, chúng ta xét một tiến trình chép dữ liệu từ DVD tới tập tin đĩa, sắp xếp tập tin đĩa và sau đó in kết quả ra máy in. Nếu tất cả tài nguyên phải được yêu cầu cùng một lúc thì khởi đầu tiến trình phải yêu cầu ổ DVD, tập tin đĩa và máy in. Nó sẽ giữ máy in trong toàn thời gian thực thi của nó mặc dù nó cần máy in chỉ ở giai đoạn cuối.

Giao thức thứ hai cho phép ban đầu tiến trình chỉ yêu cầu ổ DVD và tập tin đĩa. Nó chép dữ liệu từ băng từ tới đĩa, rồi giải phóng cả băng từ và đĩa. Sau đó, tiến trình phải yêu cầu lại tập tin đĩa và máy in. Sau khi in xong tập tin đĩa, nó giải phóng hai tài nguyên này và kết thúc công việc.

Các giao thức này có hai nhược điểm chủ yếu. Thứ nhất, việc sử dụng tài nguyên có thể chậm vì nhiều tài nguyên có thể được cấp nhưng không được sử dụng trong thời gian dài. Thứ hai, việc đói tài nguyên có thể xảy ra. Một tiến trình cần nhiều tài nguyên có thể phải đợi vô hạn định vì một tài nguyên mà nó cần luôn được cấp phát cho tiến trình khác.

Ngăn chặn điều kiện không thể thu hồi

Để ngăn chặn điều kiện này không xảy ra, có thể dùng giao thức sau. Nếu một tiến trình đang giữ một số tài nguyên và yêu cầu tài nguyên khác mà không được cấp phát ngay cho nó (nghĩa là tiến trình phải chờ) thì tất cả tài nguyên hiện đang giữ được đòi lại. Nói cách khác, những tài nguyên này được hệ điều hành thu hồi. Những tài nguyên bị đòi lại được thêm vào danh sách các tài nguyên mà tiến trình đang chờ. Tiến trình sẽ được khởi động lại khi nó có thể nhận lại tài nguyên cũ của nó cũng như các tài nguyên mới mà nó đang yêu cầu.

Có một sự chọn lựa khác, nếu một tiến trình yêu cầu một số tài nguyên, đầu tiên chúng ta kiểm tra chúng có sẵn không. Nếu tài nguyên có sẵn, chúng ta cấp phát chúng. Nếu tài nguyên không có sẵn, chúng ta kiểm tra chúng có được cấp phát cho một số tiến trình khác đang chờ tài nguyên bổ sung hay không. Nếu đúng như thế, chúng ta lấy lại tài nguyên đó từ tiến trình đang đợi và cấp chúng cho tiến trình đang yêu cầu. Nếu tài nguyên không sẵn có hay được giữ bởi một tiến trình đang đợi, tiến trình đang yêu cầu phải chờ. Trong khi nó đang chờ, một số tài nguyên của nó có thể được đòi lại nếu tiến trình khác yêu cầu chúng. Một tiến trình có thể được khởi động lại khi nó được cấp các tài nguyên mới mà nó đang yêu cầu và phục hồi bất cứ tài nguyên nào đã bị lấy lại trong khi nó đang chờ.

Giao thức này thường được áp dụng tới tài nguyên mà trạng thái của nó có thể được lưu lại dễ dàng và phục hồi, như các thanh ghi CPU và không gian bộ nhớ. Có những tài nguyên không thể áp dụng được giao thức này như máy in.

Ngăn chặn điều kiện chờ đợi luẩn quẩn

Một cách để ngăn chặn điều kiện này không xảy ra là sắp xếp thứ tự cho tất cả loại tài nguyên (Ví dụ : $\text{đĩa}=2$, $\text{máy in}=12$) và đòi hỏi mỗi tiến trình phải yêu cầu tài

nguyên theo thứ tự đó. Ví dụ, một tiến trình muốn dùng ổ đĩa và máy in tại cùng một lúc thì trước tiên phải yêu cầu ổ đĩa, nếu được cấp ổ đĩa thì mới yêu cầu máy in.

11.4 TRÁNH TẮC NGHẼN

Tránh tắc nghẽn (Deadlock Avoidance) đòi hỏi hệ điều hành có trước thông tin bổ sung liên quan đến tài nguyên mà một tiến trình sẽ yêu cầu và sử dụng trong suốt cuộc đời của nó. Với kiến thức bổ sung này, hệ điều hành có thể quyết định thỏa mãn ngay yêu cầu yêu cầu tài nguyên của tiến trình hay để tiến trình phải chờ đợi. Để quyết định xem các yêu cầu hiện tại có thể được đáp ứng ngay hay phải trì hoãn, hệ điều hành phải xem xét các tài nguyên hiện có, các tài nguyên hiện phân bổ cho từng tiến trình và các yêu cầu phát sinh của mỗi tiến trình trong tương lai.

Mô hình đơn giản và hữu ích nhất yêu cầu mỗi tiến trình khai báo số lượng tối đa mỗi loại tài nguyên mà nó cần. Từ đó có thể xây dựng một giải thuật kiểm tra xem việc cấp phát tài nguyên có dẫn tới tắc nghẽn hay không, từ đó tránh được tắc nghẽn.

Tiếp theo chúng ta sẽ trình bày một giải thuật né tránh tắc nghẽn.

11.4.1 Trạng thái an toàn

Trạng thái an toàn là trạng thái trong đó hệ thống có thể thỏa mãn các nhu cầu tài nguyên của mỗi tiến trình theo một thứ tự nào đó mà vẫn không xảy ra tắc nghẽn. Thứ tự này được gọi là *thứ tự an toàn*.

Giả sử thứ tự cấp phát tài nguyên cho các tiến trình $\langle P_1, P_2, \dots, P_n \rangle$ là một thứ tự an toàn, khi đó các tài nguyên mà P_i yêu cầu có thể được thỏa mãn bởi các tài nguyên hiện có cộng với các tài nguyên đang được các tiến trình trước P_i giữ. Trong trường hợp xấu nhất, tiến trình P_i có thể chờ cho đến khi tất cả các tiến trình trước nó hoàn thành và giải phóng các tài nguyên. Lúc đó hệ thống sẽ có đủ tài nguyên rảnh rỗi để cấp cho P_i .

Có thể mô tả trạng thái của hệ thống qua một *bảng trạng thái* như hình dưới đây

	Max		Chiếm		Còn	
	R1	R2	R1	R2	R1	R2
P1	3	2	1	0	4	1

P2	6	1	2	1		
P3	3	1	2	1		

Trong bảng này, ta có ba tiến trình P1, P2, P3 và có hai tài nguyên không chia sẻ R1, R2. Cột *Max* chỉ số lượng tối đa của mỗi loại tài nguyên mà mỗi tiến trình yêu cầu. Cột *Chiếm* chỉ số lượng mỗi loại tài nguyên mà mỗi tiến trình đang chiếm giữ (tức là đã được cấp). Cột *Còn* chỉ số lượng mỗi loại tài nguyên hiện đang còn rảnh rỗi trong hệ thống, có thể cấp ngay cho các tiến trình có yêu cầu.

Để minh họa, chúng ta xét một hệ thống với 12 ổ băng từ và 3 tiến trình: P1, P2, P3. P1 yêu cầu 10 ổ băng từ, P2 cần 4 và P3 cần tới 9 ổ băng từ. Giả sử rằng tại thời điểm hiện tại, tiến trình P1 đang giữ 5 ổ băng từ, tiến trình P2 giữ 2 và tiến trình P3 giữ 2 ổ băng từ. Do đó, có 3 ổ băng từ còn rảnh. Ta có thể mô tả trạng thái hiện tại của hệ thống bằng bảng trạng thái dưới đây. Trong trường hợp này chỉ có một tài nguyên duy nhất R là băng từ.

	Max	Chiếm	Còn
	R	R	R
P1	10	5	3
P2	4	2	
P3	9	2	

Tại thời điểm hiện tại, hệ thống ở trạng thái an toàn vì có thể thỏa mãn nhu cầu tài nguyên của các tiến trình theo thứ tự $\langle P2, P1, P3 \rangle$ mà không xảy ra tắc nghẽn. Thực vậy, hệ thống có thể cấp thêm ngay cho P2 hai trong số ba băng từ đang còn rảnh rỗi để P2 bắt đầu chạy. Sau khi P2 kết thúc và giải phóng tài nguyên, hệ thống sẽ có 5 ổ băng từ sẵn dùng, lúc này có thể cấp thêm cho P1 tất cả 5 ổ băng từ đó để P1 bắt đầu chạy. Sau khi P1 kết thúc và giải phóng tài nguyên, hệ thống sẽ có 10 ổ băng từ sẵn dùng, lúc này có thể cấp thêm cho P3 bảy trong số mười ổ băng từ đó để P1 bắt đầu chạy.

Một hệ thống có thể đi từ trạng thái an toàn tới một trạng thái không an toàn. Giả sử rằng tại thời điểm hiện tại, tiến trình P3 được cấp thêm 1 ổ băng từ nữa, khi đó trạng thái của hệ thống sẽ như bảng dưới đây.

	Max	Chiếm	Còn
	R	R	R
P1	10	5	
P2	4	2	2
P3	9	3	

Hệ thống không còn trong trạng thái an toàn. Tại thời điểm này, chỉ có thể cấp cho P2 tất cả ổ băng từ mà nó yêu cầu. Sau khi P2 kết thúc và giải phóng tài nguyên, hệ thống có 4 ổ băng từ rảnh rồi. Vì P1 cần được cấp phát thêm 5 ổ băng từ, P3 cần thêm 6 ổ băng từ mới chạy được, như vậy hệ thống không thể cấp đủ tài nguyên cho bất cứ tiến trình nào trong hai tiến trình P1 và P2, cả hai tiến trình đều phải chờ đợi, dẫn đến tình trạng tắc nghẽn. Sai lầm ở đây là đã cấp thêm ngay cho P3 cho 1 ổ băng từ nữa.

Với khái niệm trạng thái an toàn, chúng ta có thể định nghĩa các giải thuật tránh tắc nghẽn. Ý tưởng là đảm bảo hệ thống sẽ luôn ở trong trạng thái an toàn. Khoi đầu, hệ thống ở trong trạng thái an toàn. Bất cứ khi nào một tiến trình yêu cầu một tài nguyên hiện có, hệ thống phải quyết định cấp phát ngay hay để tiến trình phải chờ. Yêu cầu cấp phát sẽ được thỏa mãn ngay nếu sau khi cấp phát hệ thống vẫn trong trạng thái an toàn.

11.4.2 Giải thuật nhà băng

Khi một tiến trình mới đưa vào hệ thống, nó phải khai báo số lượng tối đa các thể hiện của mỗi loại tài nguyên mà nó cần. Số này phải không vượt quá tổng số tài nguyên trong hệ thống. Khi một tiến trình yêu cầu cấp phát tài nguyên, hệ thống phải xác định sau khi cấp phát các tài nguyên này hệ thống có vẫn ở trong trạng thái an toàn hay không. Nếu trạng thái hệ thống sẽ vẫn là an toàn, tài nguyên sẽ được cấp, ngược lại, tiến trình phải chờ cho tới khi một vài tiến trình giải phóng đủ tài nguyên.

Giải thuật nhà băng dùng để xác định trạng thái hiện tại có an toàn hay không được mô tả như sau

Bước 1 : Từ bảng trạng thái lập *bảng nhu cầu* trong đó thay cột *Max* bằng cột *Cần* với công thức tính toán *Cần* = *Max* – *Chiếm*. Cột *Cần* thể hiện số lượng mỗi loại tài nguyên cần cung cấp thêm cho mỗi tiến trình.

Bước 2 :

While $\exists i : (\text{Cần}(Pi) < > 0)$ and $(\text{Cần}(Pi) \leq \text{Còn})$

Begin

$\text{Còn} = \text{Còn} + \text{Chiếm}(Pi);$

$\text{Cần}(Pi) = 0; \text{Chiếm}(Pi) = 0;$

End

If $\forall i : \text{Cần}(Pi) = 0$

Then "Trạng thái an toàn"

Else "Trạng thái không an toàn"

Trong đó *Cần(Pi)*, *Còn*, *Chiếm(Pi)* được xem xét như là các véc-tơ trong không gian véc-tơ n chiều. Véc-tơ X được coi là nhỏ hơn hay bằng \leq véc-tơ Y (ký hiệu $X \leq Y$) khi và chỉ khi mọi phần tử của X đều nhỏ hơn hay bằng mọi phần tử tương ứng của Y.

Ví dụ 1 : Trở lại với ví dụ ở mục trước, từ bảng trạng thái

	Max		Chiếm		Còn	
	R1	R2	R1	R2	R1	R2
P1	3	2	1	0		
P2	6	1	2	1	4	1
P3	3	1	2	1		

Bước 1 : Ta lập ra bảng nhu cầu sau

	Cần		Chiếm		Còn	
	R1	R2	R1	R2	R1	R2
P1	2	2	1	0	4	1

P2	4	0	2	1		
P3	1	0	2	1		

Bước 2 : Biến đổi bảng nhu cầu theo giải thuật đã nêu, ta có

Vì $\exists i=2$: ($Cần(Pi) <> 0$) and ($Cần(Pi) \leq Còn$) đúng nên biến đổi dòng ứng với P2

	Cần		Chiếm		Còn	
	R1	R2	R1	R2	R1	R2
P1	2	2	1	0		
P2	0	0	0	0	6	2
P3	1	0	2	1		

Vì $\exists i=1$: ($Cần(Pi) <> 0$) and ($Cần(Pi) \leq Còn$) đúng nên biến đổi dòng ứng với P1

	Cần		Chiếm		Còn	
	R1	R2	R1	R2	R1	R2
P1	0	0	0	0		
P2	0	0	0	0	7	2
P3	1	0	2	1		

Vì $\exists i=3$: ($Cần(Pi) <> 0$) and ($Cần(Pi) \leq Còn$) đúng nên biến đổi dòng ứng với P3

	Cần		Chiếm		Còn	
	R1	R2	R1	R2	R1	R2
P1	0	0	0	0		
P2	0	0	0	0	9	3
P3	0	0	0	0		

Đến đây ta có điều kiện $\exists i$: ($Cần(Pi) <> 0$) and ($Cần(Pi) \leq Còn$) sai nên ra khỏi vòng lặp while, kiểm tra điều kiện if ta có điều kiện $\forall i$: $Cần(Pi)=0$ đúng nên kết luận trạng thái an toàn.

Ví dụ 2 : Xét tiếp ví dụ về hệ thống với các ổ băng từ, trường hợp tiến trình P3 được cấp thêm 1 ổ băng từ nữa, khi đó trạng thái của hệ thống sẽ như bảng dưới đây.

	Max	Chiếm	Còn
	R	R	R
P1	10	5	
P2	4	2	2
P3	9	3	

Bước 1 : Ta lập ra bảng nhu cầu sau

	Cần	Chiếm	Còn
	R	R	R
P1	5	5	
P2	2	2	2
P3	6	3	

Bước 2 : Biến đổi bảng nhu cầu theo giải thuật đã nêu, ta có

Vì $\exists i=2$: ($Cần(P_i) < 0$) and ($Cần(P_i) \leq Còn$) đúng nên biến đổi dòng ứng với P2

	Cần	Chiếm	Còn
	R	R	R
P1	5	5	
P2	0	0	4
P3	6	3	

Đến đây ta có điều kiện $\exists i$: ($Cần(P_i) < 0$) and ($Cần(P_i) \leq Còn$) sai nên ra khỏi vòng lặp while, kiểm tra điều kiện if ta có điều kiện $\forall i$: $Cần(P_i) = 0$ sai nên kết luận trạng thái không an toàn.

11.4.3 Giải thuật tránh tắc nghẽn

Cho $Request(P)$ là vector yêu cầu tài nguyên của tiến trình P. Ta cần thực hiện các hoạt động sau.

if Not($Request(P) \leq Còn$)

Then "Không cấp được"

Else

Begin

1. Lập bảng trạng thái sau khi cấp tài nguyên cho P:

Còn = Còn – Request(P);

Chiếm(P) = Chiếm(P) + Request(P);

Max(P) = Max(P);

Các số liệu ứng với các tiến trình khác giữ nguyên;

2. Kiểm tra trạng thái trên có an toàn không

3. If (An toàn) then "Cấp ngay" else "Không cấp ngay"

end

Trong ví dụ 2 ở mục trên, sau khi nhận được yêu cầu cấp thêm 1 ổ băng từ của tiến trình P3, hệ thống sẽ tiến hành kiểm tra theo giải thuật vừa trình bày và có quyết định "Không cấp ngay", tránh được tình trạng tắc nghẽn.

11.5 PHÁT HIỆN TẮC NGHẼN

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

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

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

1. Thoát tất cả các tiến trình bị tắc nghẽn. Đây là một giải pháp đơn giản nhất, thường được các hệ điều hành sử dụng nhất.
2. Sao lưu lại mỗi tiến trình bị tắc nghẽn tại một vài điểm kiểm tra được định nghĩa trước, sau đó khởi động lại tất cả các tiến trình. Giải pháp này yêu cầu hệ điều hành phải lưu lại các thông tin cần thiết tại điểm dừng của tiến trình, đặc biệt là con trỏ lệnh và các tài nguyên tiến trình đang sử dụng, để có thể khởi động lại tiến trình được. Giải pháp này có nguy cơ xuất hiện tắc nghẽn trở lại là rất cao, vì khi tắt cả các tiến trình đều được reset trở lại thì việc tranh chấp tài nguyên là khó tránh khỏi. Ngoài ra hệ điều hành thường phải chi phí rất cao cho việc tạm dừng và tái kích hoạt tiến trình.
3. Chỉ kết thúc một tiến trình trong tập tiến trình bị tắc nghẽn, thu hồi tài nguyên của tiến trình này, để cấp phát cho một tiến trình nào đó trong tập tiến trình tắc nghẽn để giúp tiến trình này ra khỏi tắc nghẽn, rồi gọi lại thuật toán kiểm tra tắc nghẽn để xem hệ thống đã ra khỏi tắc nghẽn hay chưa, nếu rồi thì dừng, nếu chưa thì tiếp tục giải phóng thêm tiến trình khác. Và lần lượt như thế cho đến khi tắt cả các tiến trình trong tập tiến trình tắc nghẽn đều ra khỏi tình trạng tắc nghẽn. Trong giải pháp này vẫn đề đặt ra đối với hệ điều hành là nên chọn tiến trình nào để giải phóng đầu tiên và dựa vào tiêu chuẩn nào để chọn lựa sao cho chi phí để giải phóng tắc nghẽn là thấp nhất.
4. Tập trung toàn bộ quyền ưu tiên sử dụng tài nguyên cho một tiến trình, để tiến trình này ra khỏi tắc nghẽn, và rồi kiểm tra xem hệ thống đã ra khỏi tắc nghẽn hay chưa, nếu rồi thì dừng lại, nếu chưa thì tiếp tục. Lần lượt như thế cho đến khi hệ thống ra khỏi tắc nghẽn. Trong giải pháp này hệ điều hành phải tính đến chuyện tái kích hoạt lại tiến trình sau khi hệ thống ra khỏi tắc nghẽn.

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

Khi nào thì chúng ta nên nạp giải thuật phát hiện tắc nghẽn? Câu trả lời phụ thuộc vào hai yếu tố: Tắc nghẽn có khả năng xảy ra thường xuyên như thế nào? Bao nhiêu tiến trình sẽ bị ảnh hưởng bởi tắc nghẽn khi nó sẽ ra?

Nếu tắc nghẽn xảy ra thường xuyên thì giải thuật phát hiện nên được nạp thường xuyên. Những tài nguyên được cấp phát để các tiến trình bị tắc nghẽn sẽ rảnh cho đến khi tắc nghẽn có thể bị phá vỡ. Ngoài ra, số lượng tiến trình liên quan trong chu trình tắc nghẽn có thể tăng lên.

Tắc nghẽn xảy ra chỉ khi một số tiến trình thực hiện yêu cầu mà không được cấp tài nguyên tức thì. Yêu cầu này có thể là yêu cầu cuối hoàn thành một chuỗi các tiến trình đang yêu cầu. Ngoài ra, chúng ta có thể nạp giải thuật phát hiện một khi một yêu cầu cho việc cấp phát không thể được cấp tức thì. Trong trường hợp này, chúng ta không chỉ định nghĩa tập hợp các tiến trình bị tắc nghẽn, mà còn xác định tiến trình đã gây ra tắc nghẽn.

Dĩ nhiên, nạp giải thuật phát hiện tắc nghẽn cho mỗi yêu cầu có thể gây ra một chi phí về thời gian tính toán. Có thể nạp giải thuật ít thường xuyên hơn, ví dụ, một lần một giờ hay bất cứ khi nào việc sử dụng CPU rơi xuống thấp hơn 40%.

11.6 PHỤC HỒI TẮC NGHẼN

Khi giải thuật phát hiện xác định rằng tắc nghẽn tồn tại, nhiều thay đổi tồn tại. Một khả năng là thông báo người điều hành rằng tắc nghẽn xảy ra và để người điều hành giải quyết tắc nghẽn bằng thủ công. Một khả năng khác là để hệ thống tự động phục hồi. Có hai tùy chọn cho việc phá vỡ tắc nghẽn. Một là huỷ bỏ một hay nhiều tiến trình để phá vỡ việc tồn tại chu trình trong đồ thị cấp phát tài nguyên. Hai là lấy lại một số tài nguyên từ một hay nhiều tiến trình bị tắc nghẽn.

11.6.1 Kết thúc tiến trình

Để xóa tắc nghẽn bằng cách hủy bỏ tiến trình, chúng ta dùng một trong hai phương pháp. Trong cả hai phương pháp, hệ thống lấy lại tài nguyên được cấp phát đối với tiến trình bị kết thúc.

- Huỷ bỏ tất cả tiến trình bị tắc nghẽn: phương pháp này rõ ràng sẽ phá vỡ chu trình tắc nghẽn, nhưng chi phí cao; các tiến trình này có thể đã tính toán trong thời gian dài, và các kết quả của các tính toán từng phần này phải bị bỏ đi và có thể phải tính lại sau đó.

- Hủy bỏ một tiến trình tại thời điểm cho đến khi chu trình tắc nghẽn bị xóa: phương pháp này chịu chi phí có thể xem xét vì sau khi mỗi tiến trình bị hủy bỏ, một giải thuật phát hiện tắc nghẽn phải được nạp lên để xác định có tiến trình nào vẫn đang bị tắc nghẽn.

Hủy bỏ tiến trình có thể không dễ. Nếu một tiến trình đang ở giữa giai đoạn cập nhật một tập tin, kết thúc nó sẽ để tập tin đó trong trạng thái không phù hợp. Tương tự, nếu tiến trình đang ở giữa giai đoạn in dữ liệu ra máy in, hệ thống phải khởi động lại trạng thái đúng trước khi in công việc tiếp theo.

Nếu phương pháp kết thúc một phần được dùng thì với một tập hợp các tiến trình tắc nghẽn được cho, chúng ta phải xác định tiến trình nào (hay các tiến trình nào) nên được kết thúc trong sự cố gắng để phá vỡ tắc nghẽn. Việc xác định này là một quyết định chính sách tương tự như các vấn đề lập thời biểu CPU. Câu hỏi về tính kinh tế là chúng ta nên hủy bỏ tiến trình nào thì sự kết thúc của tiến trình đó sẽ chịu chi phí tối thiểu. Tuy nhiên, thuật ngữ chi phí tối thiểu là không chính xác. Nhiều yếu tố có thể xác định tiến trình nào được chọn bao gồm:

- Độ ưu tiên của tiến trình là gì.
- Tiến trình đã được tính toán bao lâu và bao lâu nữa tiến trình sẽ tính toán trước khi hoàn thành tác vụ được chỉ định của nó.
- Bao nhiêu và loại tài nguyên gì tiến trình đang sử dụng.
- Bao nhiêu tài nguyên nữa tiến trình cần để hoàn thành
- Bao nhiêu tiến trình sẽ cần được kết thúc.
- Tiến trình là giao tiếp hay dạng bó

11.6.2 Lấy lại tài nguyên

Để xóa tắc nghẽn ta sử dụng việc thu hồi tài nguyên, chúng ta đòi tài nguyên từ một số tiến trình và cấp các tài nguyên này cho các tiến trình khác cho đến khi chu trình tắc nghẽn bị phá vỡ. Nếu việc thu hồi được yêu cầu để giải quyết tắc nghẽn thì có ba vấn đề cần được xác định:

- Chọn nạn nhân: những tài nguyên nào và những tiến trình nào bị đòi lại? Trong khi kết thúc tiến trình, chúng ta phải xác định thứ tự đòi lại để giảm thiểu chi phí. Các

yếu tố chi phí có thể gồm các tham số như số lượng tài nguyên một tiến trình tắc nghẽn đang giữ, lượng thời gian một tiến trình tắc nghẽn dùng khi thực thi.

- Trở lại trạng thái trước tắc nghẽn: Nếu chúng ta đòi lại tài nguyên từ một tiến trình, điều gì nên được thực hiện với tiến trình đó? Rõ ràng, nó không thể tiếp tục việc thực thi bình thường; nó đang bị mất một số tài nguyên được yêu cầu. Chúng ta phải phục hồi tiến trình tới trạng thái an toàn và khởi động lại từ trạng thái gần nhất trước đó. Thông thường, rất khó để xác định trạng thái gì là an toàn vì thế giải pháp đơn giản nhất là phục hồi toàn bộ: hủy tiến trình và sau đó khởi động lại nó. Tuy nhiên, hữu hiệu hơn để phục hồi tiến trình chỉ đủ xa cần thiết để phá vỡ tắc nghẽn. Ngoài ra, phương pháp này yêu cầu hệ thống giữ nhiều thông tin hơn về trạng thái của tất cả các tiến trình đang chạy.
- Đói tài nguyên: chúng ta đảm bảo như thế nào việc đói tài nguyên không xảy ra? Nghĩa là, chúng ta có thể đảm bảo rằng tài nguyên sẽ không luôn bị đòi lại từ cùng một tiến trình.

Trong một hệ thống việc chọn nạn nhân ở đâu dựa trên cơ sở các yếu tố chi phí, nó có thể xảy ra cùng tiến trình luôn được chọn như là nạn nhân. Do đó, tiến trình này không bao giờ hoàn thành tác vụ được chỉ định của nó, một trường hợp đói tài nguyên cần được giải quyết trong bất kỳ hệ thống thực tế. Rõ ràng, chúng ta phải đảm bảo một tiến trình có thể được chọn như một nạn nhân chỉ một số lần xác định (nhỏ). Giải pháp chung nhất là bao gồm số lượng phục hồi trong yếu tố chi phí.

CÂU HỎI VÀ BÀI TẬP

1. Hãy định nghĩa trạng thái tắc nghẽn của hệ thống
2. Hãy nêu 4 điều kiện cần để xuất hiện trạng thái tắc nghẽn
3. Hãy cho một ví dụ về trạng thái tắc nghẽn.
4. Hãy định nghĩa trạng thái an toàn (hiểu theo nghĩa không gây ra tình trạng tắc nghẽn) của hệ thống.
5. Hãy dùng giải thuật nhà băng để xác định xem trạng thái sau có an toàn hay không?

	Max		Chiếm		Còn	
	R1	R2	R1	R2	R1	R2
P1	4	10	1	6	2	1
P2	6	3	4	2		
P3	8	5	6	1		

BÀI 12. QUẢN LÝ BỘ NHỚ

12.1 MỞ ĐẦU

Bộ nhớ chính là thiết bị lưu trữ duy nhất mà thông qua đó CPU có thể trao đổi thông tin với môi trường ngoài, do vậy quản lý bộ nhớ là một trong những nhiệm vụ quan trọng và phức tạp nhất của hệ điều hành. Hầu hết các hệ điều hành hiện đại đều cho phép chế độ đa nhiệm nhằm nâng cao hiệu suất sử dụng CPU. Khi đó bộ nhớ là một tài nguyên của hệ thống dùng để cấp phát và chia sẻ cho nhiều tiến trình. Có thể chia việc quản lý bộ nhớ thành các nhiệm vụ chính như sau.

- Tổ chức và quản lý bộ nhớ vật lý
- Tổ chức và quản lý bộ nhớ logic
- Định vị và tái định vị các tiến trình
- Chia sẻ bộ nhớ cho các tiến trình
- Bảo vệ vùng nhớ của các tiến trình

Công cụ cơ bản của quản lý bộ nhớ là sự phân trang (paging) và sự phân đoạn (segmentation). Với sự phân trang mỗi tiến trình được chia thành nhiều phần nhỏ có quan hệ với nhau, với kích thước của trang là cố định. Sự phân đoạn cung cấp cho chương trình người sử dụng các khối nhớ có kích thước khác nhau. Hệ điều hành cũng có thể kết hợp giữa phân trang và phân đoạn để có được một chiến lược quản lý bộ nhớ linh hoạt hơn.

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

12.1.1 Địa chỉ vật lý và địa chỉ logic

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 khái niệm này giúp hệ điều hành dễ dàng xây dựng các cơ chế và chiến lược quản lý bộ nhớ hữu hiệu.

Địa chỉ vật lý và không gian nhớ vật lý

Bộ nhớ chính (memory) được tổ chức như là mảng một chiều của các *từ nhớ* (word), mỗi từ nhớ có một địa chỉ, địa chỉ này được gọi là *địa chỉ vật lý* (physical address). Việc trao đổi thông tin với CPU và môi trường ngoài được thực hiện thông qua các thao tác đọc hoặc ghi dữ liệu vào một địa chỉ vật lý cụ thể nào đó trong bộ nhớ. Tất cả các địa chỉ vật lý tạo thành *không gian nhớ vật lý* (gọi tắt là không gian vật lý).

Địa chỉ logic và không gian địa chỉ

Địa chỉ logic (logical address, địa chỉ ảo) là địa chỉ được tạo ra khi biên dịch từ chương trình nguồn ra mã máy (ví dụ như các file .com, .exe,...) để tham chiếu đến các vị trí nhớ trong chương trình, nó độc lập với cấu trúc và tổ chức vật lý của bộ nhớ. Thông thường, địa chỉ logic được biểu diễn như là vị trí tương đối so với một điểm xác định nào đó trong chương trình mã máy (ví dụ như điểm bắt đầu chương trình) nên còn được gọi là *địa chỉ tương đối* (relative address).

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

Khi chương trình được nạp vào trong bộ nhớ vật lý và trở thành tiến trình, mỗi vị trí nhớ trong chương trình sẽ lưu trú tại một từ nhớ trong bộ nhớ vật lý, tức là mỗi địa chỉ logic sẽ ứng với một địa chỉ vật lý.

12.1.2 Ánh xạ bộ nhớ

CPU chỉ làm việc với các địa chỉ logic, vì vậy để có thể các tham chiếu được đến các vị trí nhớ của chương trình trong bộ nhớ vật lý, địa chỉ logic phải được chuyển đổi thành địa chỉ vật lý. Thao tác này gọi là *ánh xạ bộ nhớ* hay *kết buộc địa chỉ*.

Có thể thực hiện ánh xạ bộ nhớ vào một trong những thời điểm sau :

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

trình, cần phải biên dịch lại chương trình. Các file chương trình .COM của hệ điều hành DOS sử dụng phương thức này.

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

Thời điểm thực thi : Nếu có nhu cầu di chuyển tiến trình từ vùng nhớ này sang vùng nhớ khác trong khi tiến trình đang thực thi, thì thời điểm kết buộc địa chỉ phải trì hoãn đến tận thời điểm xử lý. Khi đó, thao tác chuyển đổi này thường do một cơ chế phần cứng là MMU (*memory-management unit*) thực hiện. Hầu hết các hệ điều hành hiện đại sử dụng phương thức này.

12.2 CẤP PHÁT LIÊN TỤC

Trong phương thức *cấp phát liên tục*, mỗi tiến trình được nạp vào một vùng nhớ liên tục đủ lớn để chứa toàn bộ tiến trình. Ưu điểm của phương thức này là đơn giản, việc chuyển đổi địa chỉ logic thành địa chỉ vật lý và ngược lại chỉ cần dựa vào một công thức đơn giản $\langle\text{địa chỉ vật lý}\rangle = \langle\text{địa chỉ bắt đầu}\rangle + \langle\text{địa chỉ logic}\rangle$.

12.2.1 Hiện tượng phân mảnh bộ nhớ

Nhược điểm lớn nhất của phương thức cấp phát liên tục là không sử dụng hiệu quả bộ nhớ do *hiện tượng phân mảnh bộ nhớ*. Đây là hiện tượng khi các tiến trình lăn lướt vào và ra khỏi hệ thống, dần dần xuất hiện các khe hở trống giữa các tiến trình. Đây là các khe hở được tạo ra do kích thước của tiến trình mới được nạp nhỏ hơn kích thước vùng nhớ mới được giải phóng bởi tiến trình đã kết thúc. Hiện tượng này có thể dẫn đến tình huống không thể nạp được một tiến trình nào đó do không có một vùng nhớ trống liên tục đủ lớn trong khi tổng kích thước các vùng nhớ trống đủ để thỏa mãn yêu cầu. Ví dụ, nếu bộ nhớ có ba vùng nhớ trống liên tục với kích thước 1Mbytes, 3Mbytes, 5Mbytes thì không thể nào nạp một chương trình có kích thước 6Mbytes mặc dù tổng kích thước bộ nhớ trống là 9Mbytes.

Để giải quyết vấn đề này, có thể theo một trong các hướng sau

1. Đề ra các chiến lược cấp phát hợp lý nhằm giảm thiểu kích thước vùng nhớ trống không sử dụng được. Các chiến lược phổ biến nhất là
 - First-fit : Cấp phát vùng nhớ trống liên tục đầu tiên đủ lớn.
 - Best-fit : Cấp phát vùng nhớ trống liên tục nhỏ nhất đủ lớn. Chiến lược này tạo ra lỗ trống nhỏ nhất còn thừa lại.
 - Worst-fit : cấp phát vùng nhớ trống liên tục lớn nhất đủ lớn. Chiến lược này tạo ra lỗ trống còn lại lớn nhất mà có thể có ích hơn lỗ trống nhỏ từ tiếp cận best-fit.
2. Áp dụng kỹ thuật tái định vị các tiến trình để kết hợp các mảnh bộ nhớ trống nhỏ rời rạc thành một vùng nhớ trống lớn liên tục. Tuy nhiên, kỹ thuật này đòi hỏi nhiều thời gian xử lý, ngoài ra sự kết buộc địa chỉ phải thực hiện vào thời điểm xử lý vì các tiến trình có thể bị di chuyển trong quá trình dồn bộ nhớ.
3. Sử dụng kỹ thuật hoán vị (swaping) : Hệ điều hành chuyển tạm một vài tiến trình đang trong trạng thái blocked hoặc ready ra bộ nhớ phụ, giải phóng bộ nhớ chính để có vùng nhớ trống liên tục đủ lớn cho việc nạp chương trình mới. Sau này chương trình bị chuyển tạm ra bộ nhớ phụ sẽ được nạp trở lại vào bộ nhớ chính để tiếp tục thực thi. Tuy nhiên giải pháp này cũng làm hệ thống bị chậm lại do thời gian hoán vị một chương trình cũng khá lớn.
4. Sử dụng kỹ thuật phủ lấp (overlay) để giảm thiểu kích thước bộ nhớ trống cần để nạp chương trình. Ngay khi viết và biên dịch chương trình, ta phải chia chương trình thành phần chính và các phần overlay. Khi nạp chương trình chỉ nạp phần chính cộng thêm một vùng nhớ đủ lớn để có thể nạp một phần overlay bất kỳ. Khi cần thực thi các lệnh trong overlay nào thì nạp overlay đó vào vùng bộ nhớ dành sẵn. Tuy nhiên kỹ thuật này đòi hỏi sự hỗ trợ của ngôn ngữ lập trình và người lập trình phải quan tâm đến kích thước bộ nhớ ngay khi lập trình.

Hiện nay đã có những kỹ thuật tốt hơn để giải quyết vấn đề phân mảnh bộ nhớ và chạy những chương trình lớn, đó là *kỹ thuật cấp phát không liên tục* và *kỹ thuật bộ nhớ ảo* mà ta sẽ xem xét sau này.

12.2.2 Ánh xạ và bảo vệ bộ nhớ

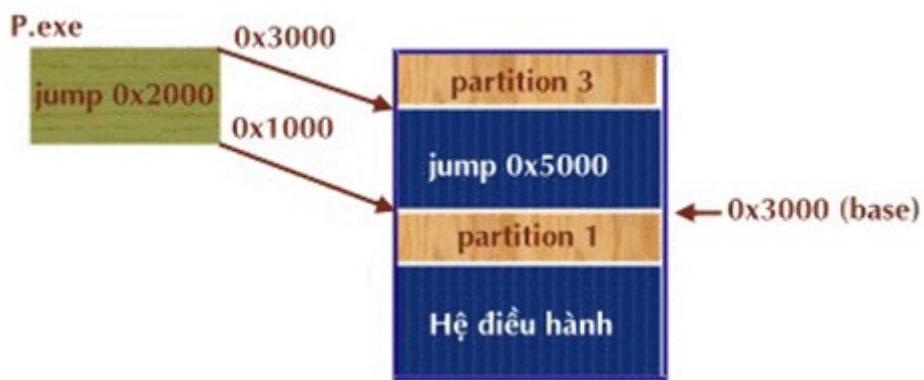
Trong phương thức cấp phát liên tục, khi biên dịch chương trình, các địa chỉ tham chiếu bên trong vẫn là địa chỉ logic. Việc ánh xạ bộ nhớ được thực hiện tại thời điểm nạp hoặc tại thời điểm thực thi chương trình.

Ánh xạ bộ nhớ tại thời điểm nạp chương trình

Hệ điều hành sẽ trả về địa chỉ bắt đầu nạp tiến trình và thay các địa chỉ tham chiếu trong tiến trình (đang là địa chỉ logic) bằng địa chỉ vật lý theo công thức

$$(\text{địa chỉ vật lý}) = (\text{địa chỉ bắt đầu}) + (\text{địa chỉ logic})$$

Thao tác này được mô tả trong hình dưới đây



Mô hình này có tên gọi là *mô hình Linker_Loader*, nó có các nhược điểm sau

- Sau khi nạp không thể tái định vị tiến trình, do vậy không thể áp dụng các kỹ thuật giảm thiểu hiện tượng phân mảnh và thiếu bộ nhớ được trình bày ở trên.
- Không có khả năng kiểm soát địa chỉ mà các tiến trình truy cập, do vậy một tiến trình có thể truy cập bất hợp pháp vào vùng nhớ được cấp cho tiến trình khác.

Ánh xạ bộ nhớ tại thời điểm thực thi chương trình

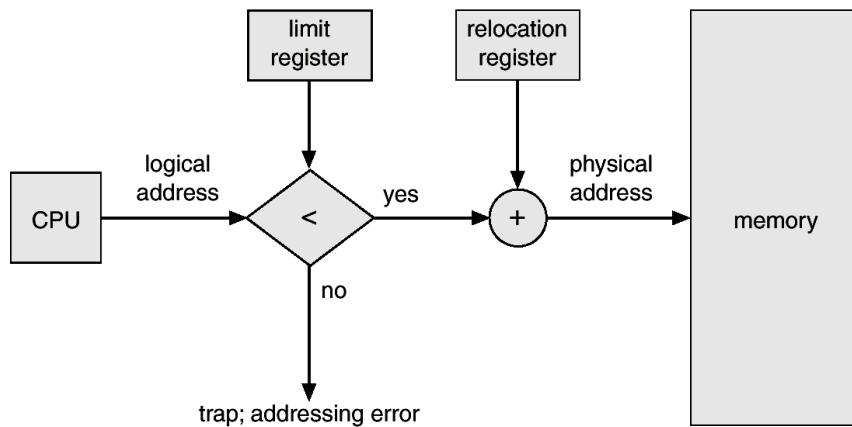
Trong cấu trúc phần cứng máy tính có một thanh ghi nền (base register, relocation register, thanh ghi tái định vị) và một thanh ghi giới hạn (limit register, bound register). Khi tiến trình được cấp phát vùng nhớ, hệ điều hành nạp vào thanh ghi nền địa chỉ bắt đầu của vùng nhớ được cấp phát cho tiến trình và nạp vào thanh ghi giới hạn kích thước của tiến trình. Khi CPU cần tham chiếu tới một địa chỉ, MMU sẽ tiến hành theo giải thuật sau.

```

if (địa chỉ logic) < (giá trị thanh ghi giới hạn)
then (địa chỉ vật lý) = (giá trị thanh ghi nền) + (địa chỉ logic)
else báo lỗi

```

Mô hình này có tên gọi là mô hình Base and Bound. Giải thuật trên được mô tả như hình sau



So với việc ánh xạ bộ nhớ tại thời điểm nạp chương trình, việc ánh xạ bộ nhớ tại thời điểm thực thi chương trình có các ưu điểm sau

- Có thể tái định vị tiến trình, do vậy có thể áp dụng các kỹ thuật giảm thiểu hiện tượng phân mảnh và thiếu bộ nhớ được trình bày ở trên. Mỗi khi tiến trình được tái định vị, chỉ cần nạp lại giá trị cho thanh ghi nền, các địa chỉ tuyệt đối sẽ được phát sinh lại mà không cần cập nhật các địa chỉ tương đối trong chương trình
- Có khả năng kiểm soát địa chỉ mà tiến trình truy cập, do vậy tiến trình không thể truy cập vào vùng nhớ cấp cho tiến trình khác.

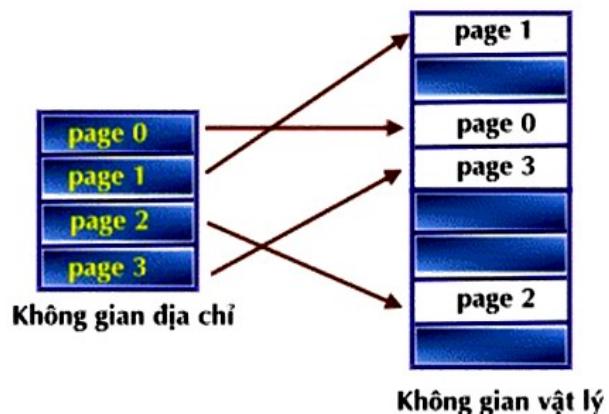
12.3 CẤP PHÁT KHÔNG LIÊN TỤC

Cấp phát không liên tục là cấp phát cho một tiến trình những vùng bộ nhớ trống không liên tục, do đó cho phép giảm thiểu hiện tượng phân mảnh, sử dụng bộ nhớ hiệu quả hơn, tuy nhiên kỹ thuật quản lý phức tạp hơn, đòi hỏi cấu hình phần cứng mạnh hơn so với phương pháp cấp phát liên tục. Hai kỹ thuật cơ bản của cấp phát không liên tục là kỹ thuật phân trang và kỹ thuật phân đoạn.

12.3.1 Kỹ thuật phân trang

Ý tưởng của kỹ thuật phân trang

Ý tưởng chính của kỹ thuật phân trang là như sau: Chia bộ nhớ vật lý thành các khung (block) có kích thước bằng nhau, gọi là *khung trang* (page frame). Chia bộ nhớ logic (của một tiến trình) thành các khung có cùng kích thước với khung trang, gọi là *trang* (page). Khi nạp một tiến trình, các trang của tiến trình sẽ được nạp vào những khung trang còn trống. Hình dưới là mô hình của kỹ thuật phân trang.



Cơ chế hoạt động của kỹ thuật phân trang

Về truyền thống, hỗ trợ phân trang được quản lý bởi phần cứng. Tuy nhiên, những thiết kế gần đây cài đặt phân trang bằng cách tích hợp chật chẽ phần cứng và hệ điều hành, đặc biệt trên các bộ xử lý 64 bit.

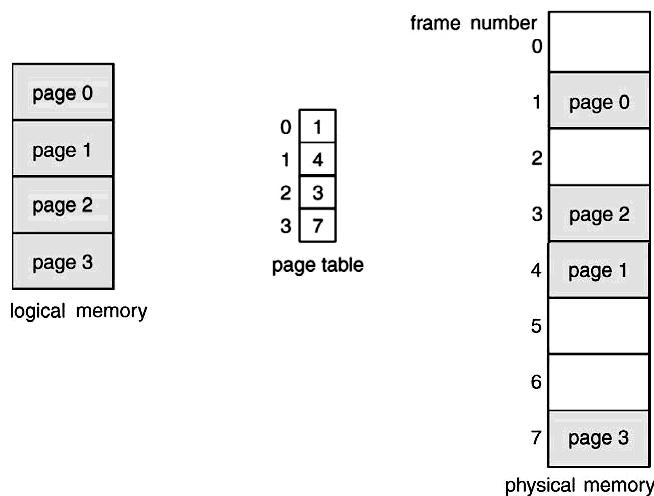
Hệ thống quản lý khung trang thông qua một cấu trúc dữ liệu được gọi là *bảng khung* (Frame Table) với số hiệu khung trang bắt đầu từ 0. Bảng khung chỉ có một mục từ cho mỗi khung trang vật lý, hiển thị khung trang đó đang rảnh hay được cấp phát. Nếu khung trang được cấp phát, thì xác định trang nào của tiến trình nào được cấp.

Cơ chế phần cứng hỗ trợ việc chuyển đổi giữa địa chỉ logic và địa chỉ vật lý trong kỹ thuật phân trang là *bảng trang* (page table) với số hiệu trang bắt đầu từ 0. Số hiệu trang được dùng như chỉ mục của bảng trang. Sau khi trang được nạp vào bộ nhớ, bảng trang chứa địa chỉ nền của trang trong bộ nhớ vật lý, tức là số hiệu của khung trang (frame) mà trang đó được nạp vào.

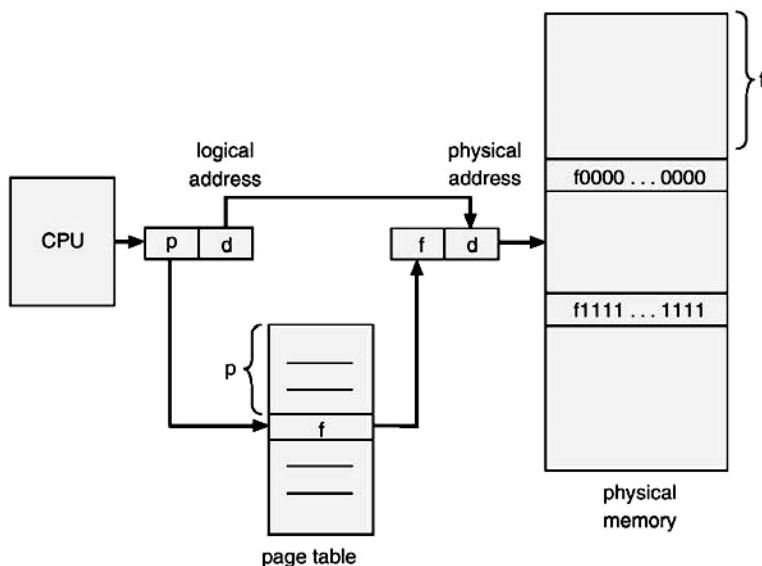
Mỗi địa chỉ logic được chia thành hai phần: *số hiệu trang* (page) và *bước nhảy* (displacement, offset). Bước nhảy là khoảng cách từ địa chỉ đầu tiên của trang đến địa chỉ logic. Ta có các công thức

$$(địa chỉ logic) = (số hiệu trang) \times (\text{kích thước trang}) + (\text{bước nhảy})$$

$$(địa chỉ vật lý) = (số hiệu khung trang) \times (\text{kích thước khung trang}) + (\text{bước nhảy})$$



Mô hình chuyển đổi địa chỉ trong kỹ thuật phân trang được mô phỏng trong hình sau. Trong mô hình này, p là số hiệu trang và cũng là chỉ mục của bảng trang, f là số hiệu khung trang, d là bước nhảy.



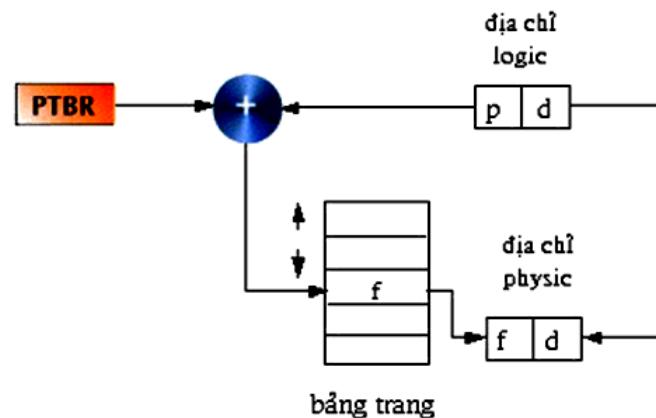
Kích thước trang và khung trang được xác định bởi phần cứng. Kích thước của một trang điển hình là lũy thừa của 2, từ 512 bytes đến 16MB một trang, phụ thuộc vào kiến trúc máy tính. Việc chọn kích thước trang là lũy thừa của 2 cho phép chuyển địa chỉ logic thành số hiệu trang và bước nhảy rất dễ dàng vì máy tính làm việc theo cơ số 2 (tức là bit). Nếu kích thước không gian địa chỉ là 2^m và kích thước trang là 2^n đơn vị địa chỉ (byte hay word), thì $m-n$ bit cao của địa chỉ logic chỉ số trang, n bit thấp chỉ bước nhảy.

Hỗ trợ phần cứng trong kỹ thuật phân trang

Chúng ta thấy cứ mỗi lần thực thi một lệnh, hệ thống phải chuyển đổi địa chỉ logic ra địa chỉ vật lý, muốn vậy phải tra cứu bảng trang. Vì vậy tốc độ tra cứu bảng trang có ảnh hưởng rất lớn đến tốc độ làm việc của hệ thống. Tốc độ tra cứu bảng trang phụ thuộc vào hai yếu tố : tốc độ truy xuất của phần cứng tạo ra bảng trang và cấu trúc logic của bảng trang. Trong phần này chúng ta sẽ xem xét việc sử dụng các phần cứng có tốc độ truy xuất nhanh và phần tiếp theo sẽ xem xét các cấu trúc để tăng tốc độ tra cứu bảng trang.

Cài đặt phần cứng của bảng trang có thể được thực hiện trong nhiều cách. Cách đơn giản nhất, bảng trang được cài đặt như tập hợp các thanh ghi tận hiến. Các thanh ghi này có tốc độ truy xuất rất cao nên việc chuyển đổi địa chỉ cũng diễn ra rất nhanh.

Vì các thanh ghi rất đắt nên việc sử dụng chúng cho bảng trang chỉ phù hợp nếu bảng trang có kích thước nhỏ (ví dụ: 256 mục từ). Trong khi đó, hầu hết các máy tính hiện nay cần bảng trang rất lớn (ví dụ, 1 triệu mục từ) nên việc sử dụng các thanh ghi để cài đặt bảng trang là không khả thi. Trong trường hợp này, bảng trang được giữ trong bộ nhớ chính và thanh ghi nền bảng trang (page-table base register - PTBR) chỉ tới địa chỉ bắt đầu bảng trang.

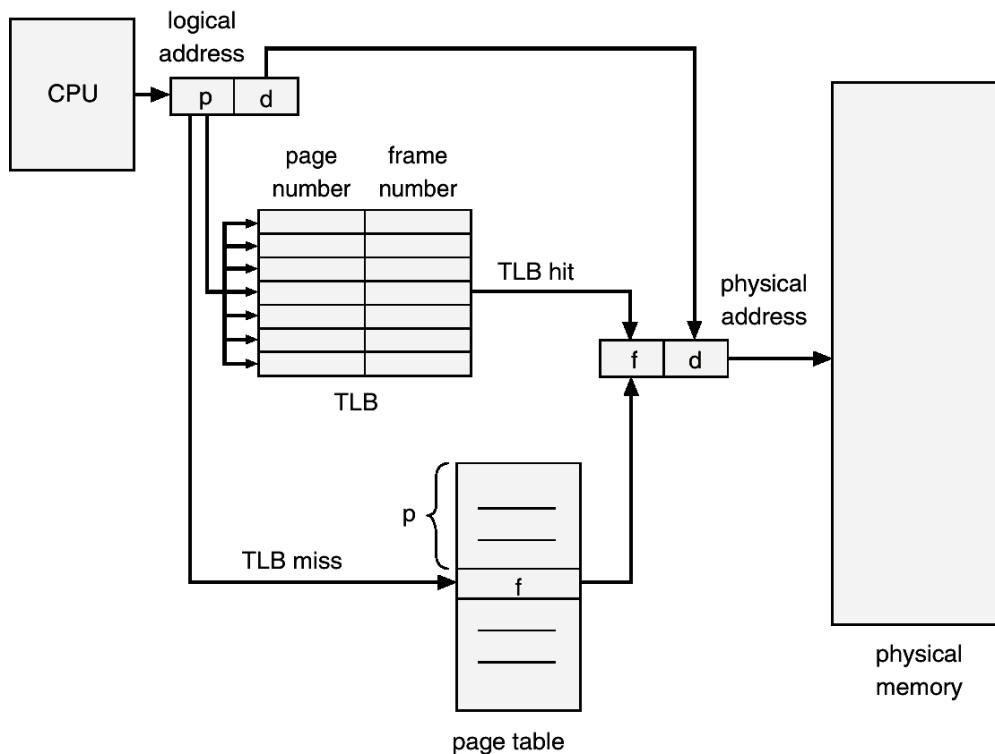


Theo cách tổ chức này, mỗi truy xuất đến dữ liệu hay chỉ thị đều đòi hỏi hai lần truy xuất bộ nhớ thông thường : một cho truy xuất đến bảng trang để chuyển đổi địa chỉ logic thành địa chỉ vật lý và một để lấy dữ liệu dữ liệu. Vì tốc độ truy xuất bộ nhớ thông thường không phải là nhanh nên việc này làm giảm tốc độ của hệ thống.

Có thể giảm bớt việc truy xuất bộ nhớ thông thường hai lần nói ở trên bằng cách dùng một phần cứng đặc biệt là bộ nhớ cache tốc độ cao có khả năng tìm kiếm nhanh, được gọi là TLB (translation look-aside buffer). Mỗi mục từ trong TLB chứa hai phần: số hiệu trang và số hiệu khung trang tương ứng. Tuy nhiên TLB đắt tiền nên

thường chỉ dùng với kích thước nhỏ, ví dụ số lượng mục từ trong TLB thường chỉ từ 64 đến 1024.

Trong kỹ thuật phân trang, TLBs được sử dụng để lưu trữ số hiệu các trang và khung trang hay được truy cập nhất, chẳng hạn như các trang được truy cập gần hiện tại nhất và các trang của một số tiến trình hạt nhân. Khi CPU phát sinh một địa chỉ, số hiệu trang của địa chỉ sẽ được so sánh với các phần tử trong TLBs, nếu có trang tương ứng trong TLBs, thì sẽ xác định được ngay số hiệu khung trang tương ứng, nếu không mới cần thực hiện thao tác tìm kiếm trong bảng trang (ở trong bộ nhớ thông thường). Quá trình trên được biểu diễn trong sơ đồ sau. Toàn bộ tác vụ có thể mất ít hơn 10% thời gian so với dùng bảng trang thông thường.



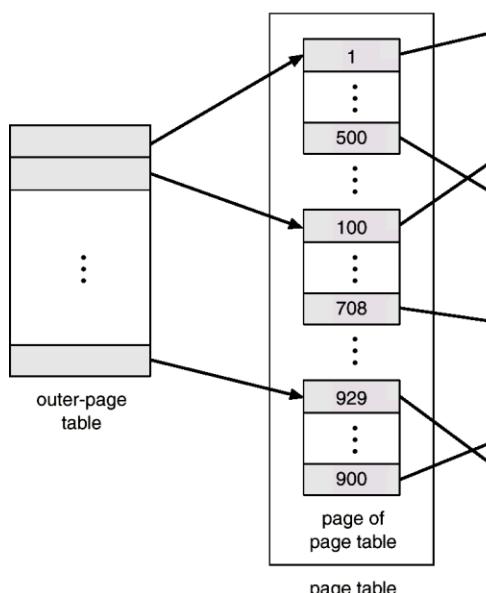
Để thấy tác dụng của TLB, ta hãy xem xét ví dụ sau. Giả sử mỗi truy xuất của TLB mất 20 nano giây, còn mỗi truy xuất bộ nhớ thông thường mất 100 nano giây. Nếu số hiệu khung trang được tìm thấy trong TLB thì mỗi lần truy xuất dữ liệu theo yêu cầu của CPU sẽ mất $20+100=120$ nano giây, nếu số hiệu khung trang không tìm thấy trong TLB thì sẽ mất $20+100+100=220$ nano giây, còn nếu sử dụng bảng trang thông thường thì sẽ mất $100+100=200$ nano giây. Nếu xác suất tìm thấy khung trang trong TLB là 80% thì thời gian trung bình cho một lần truy xuất dữ liệu theo yêu cầu của

CPU là $0.80 \times 120 + 0.2 \times 220 = 140$ nano giây, nghĩa là nhanh hơn được 60 nano giây so với dùng bảng trang thông thường.

Cấu trúc bảng trang

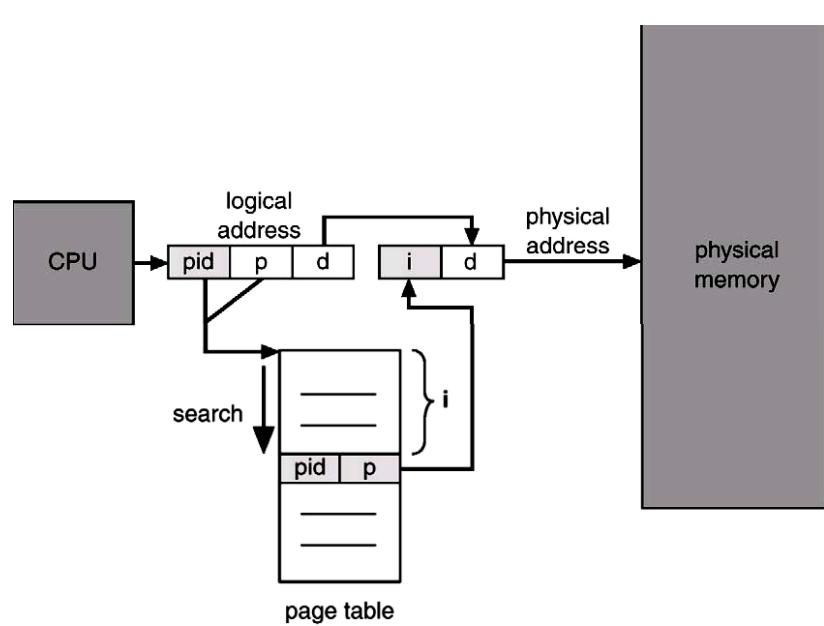
Hầu hết các hệ thống máy tính hiện đại hỗ trợ một không gian địa chỉ logic lớn (từ 2^{32} tới 2^{64} byte). Trong môi trường như thế, bảng trang trở nên quá lớn. Ví dụ, xét một hệ thống với không gian địa chỉ logic 32 bit (2^{32} byte). Nếu kích thước trang là 4096 byte (2^{12} byte) thì bảng trang có thể chứa tới khoảng 1 triệu mục từ ($2^{32}/2^{12}$). Giả sử rằng mỗi mục từ chứa 4 bytes, khi đó mỗi tiến trình có thể cần tới 4MB không gian địa chỉ vật lý cho một bảng trang. Có nhiều giải pháp cho vấn đề này, ví dụ như:

1. Dùng bảng trang phân cấp : Chia bảng trang thành các phần nhỏ, có quan hệ phân cấp, bản thân bảng trang cũng sẽ được phân trang



2. Dùng bảng trang nghịch đảo

: Sử dụng duy nhất một bảng trang nghịch đảo cho tất cả các tiến trình. Mỗi phần tử trong bảng trang nghịch đảo phản ánh một khung trang trong bộ nhớ bao gồm địa chỉ



logic của một trang đang được lưu trữ trong bộ nhớ vật lý tại khung trang này, cùng với thông tin về tiến trình đang được sở hữu trang. Mỗi địa chỉ ảo khi đó là một bộ ba $\langle idp, p, d \rangle$, trong đó idp là định danh của tiến trình, p là số hiệu trang và d là bước nhảy. Mỗi phần tử trong bảng trang nghịch đảo là một cặp $\langle idp, p \rangle$. Khi một tham khảo đến bộ nhớ được phát sinh, một phần địa chỉ ảo là $\langle idp, p \rangle$ được đưa đến cho trình quản lý bộ nhớ để tìm phần tử tương ứng trong bảng trang nghịch đảo, nếu tìm thấy, địa chỉ vật lý $\langle i, d \rangle$ sẽ được phát sinh. Trong các trường hợp khác, xem như tham khảo bộ nhớ đã truy xuất một địa chỉ bất hợp lệ.

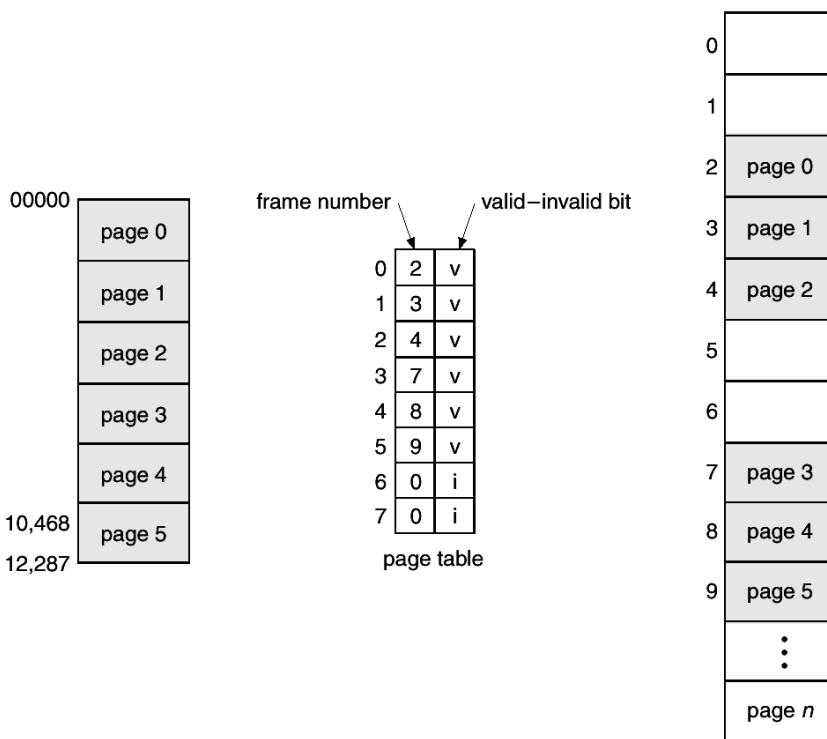
Bảo vệ bộ nhớ trong kỹ thuật phân trang

Bảo vệ bộ nhớ trong môi trường phân trang được thực hiện bởi các bit bảo vệ. Thông thường, các bit này được giữ trong bảng trang. Một bit có thể định nghĩa một trang để đọc-viết hay chỉ đọc. Khi chuyển đổi địa chỉ logic ra địa chỉ vật lý, các bit bảo vệ có thể được tham chiếu để kiểm tra liệu thao tác viết nào đang được thực hiện tới trang chỉ đọc. Cố gắng viết tới một trang chỉ đọc sẽ gây ra một ngắt phần cứng tới hệ điều hành.

Có thể mở rộng tiếp cận này để cung cấp một cấp độ bảo vệ chi tiết hơn với các chế độ *chỉ đọc, đọc viết, chỉ thực thi*. Bằng cách cung cấp các bit bảo vệ riêng cho mỗi chế độ, có thể kết hợp các chế độ này, mọi cố gắng không hợp lệ sẽ gây ra ngắt cứng tới hệ điều hành.

Một bit nữa thường được gán tới mỗi mục từ trong bảng trang là *bit hợp lệ- không hợp lệ* (valid-invalid). Giá trị “hợp lệ” cho biết trang đó là trang của tiến trình, có thể truy xuất. Giá trị “không hợp lệ” cho biết trang đó không có trong tiến trình (mục từ này là mục từ dư ra của bảng trang), không thể truy xuất. Ví dụ, trong hình dưới. Địa

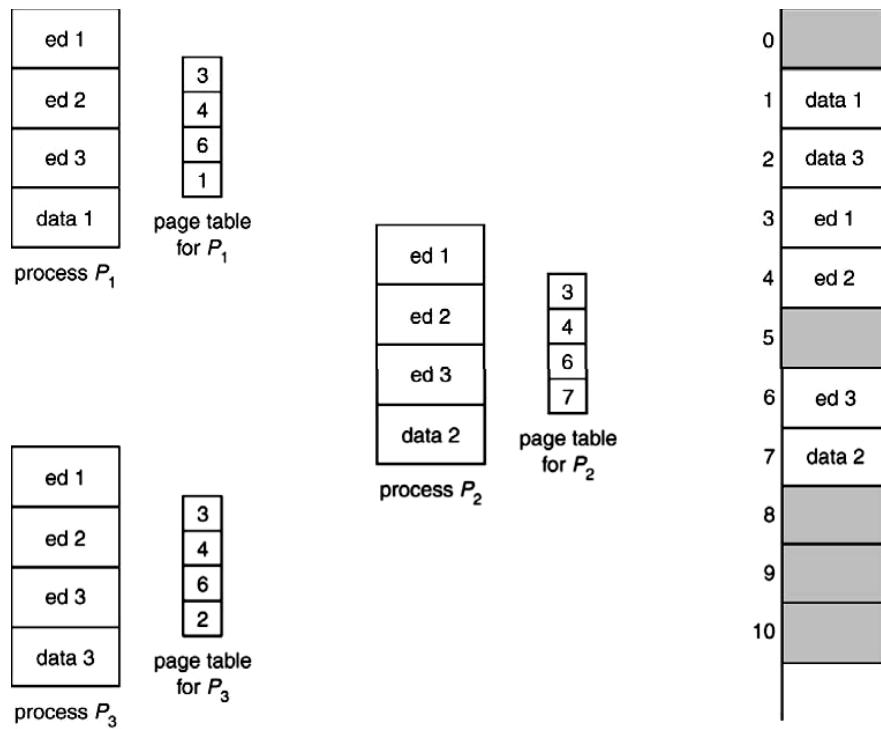
chỉ trong các trang từ 0 tới 5 là hợp lệ, còn các trang 6 và 7 là không hợp lệ.



Chia sẻ bộ nhớ trong kỹ thuật phân trang

Một ưu điểm của cơ chế phân trang là cho phép chia sẻ các trang giữa các tiến trình. Trong trường hợp này, sự chia sẻ được thực hiện bằng cách ánh xạ nhiều địa chỉ logic vào một địa chỉ vật lý duy nhất. Có thể áp dụng kỹ thuật này để cho phép có tiến trình chia sẻ một vùng code chung: nếu có nhiều tiến trình của cùng một chương trình, chỉ cần lưu trữ một đoạn code của chương trình này trong bộ nhớ, các tiến trình sẽ có thể cùng truy xuất đến các trang chứa code chung này. Lưu ý để có thể chia sẻ một đoạn code, đoạn code này phải là mã tái sử dụng (reentrant code), cho phép một bản sao của chương trình được sử dụng đồng thời bởi nhiều tác vụ.

Trong hình dưới đây, chỉ một bản sao của bộ soạn thảo được giữ trong bộ nhớ vật lý. Mỗi bảng trang của người dùng ánh xạ tới cùng một bản sao vật lý của bộ soạn thảo nhưng các trang dữ liệu được ánh xạ tới các khung khác nhau. Do đó, để hỗ trợ 40 người dùng, chúng ta cần chỉ một bản sao của bộ soạn thảo cộng với 40 bản sao của không gian dữ liệu cho mỗi người dùng. Nếu kích thước của bộ soạn thảo là 150KB và kích thước không gian dữ liệu của mỗi người dùng là 50KB, toàn bộ không gian được yêu cầu là $150 + 40 \times 50 = 2150$ KB, còn nếu không chia sẻ đoạn code thì sẽ cần một không gian là $40 \times (150 + 50) = 8000$ KB.



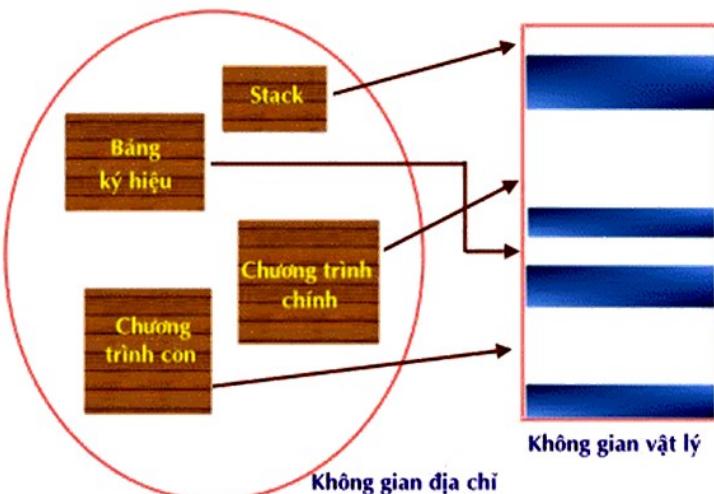
Nhận xét

- Kỹ thuật phân trang loại bỏ được hiện tượng phân mảnh ngoại vi, mỗi khung trang đều có thể được cấp phát cho một tiến trình nào đó. Tuy nhiên hiện tượng phân mảnh nội vi vẫn có thể xảy ra khi kích thước của tiến trình không đúng bằng bội số của kích thước một trang, khi đó trang cuối cùng sẽ không được sử dụng hết.
- Kỹ thuật phân trang phân biệt rạch ròi góc nhìn của người dùng và góc nhìn nhớ vật lý. Người dùng thấy bộ nhớ cấp cho tiến trình như là một không gian liên tục. Góc nhìn vật lý cho thấy một tiến trình được lưu trữ phân tán rời rạc khắp bộ nhớ vật lý.
- Việc chuyển đổi địa chỉ logic thành địa chỉ vật lý do phần cứng đảm nhiệm. Sự chuyển đổi này là trong suốt đối với người sử dụng.

12.3.2 Kỹ thuật phân đoạn

Kỹ thuật phân trang không phản ánh đúng cách thức người dùng cảm nhận về cấu trúc một tiến trình. Người dùng thường nghĩ cấu trúc một tiến trình giống như cấu trúc một chương trình trước khi biên dịch ra ngôn ngữ máy. Ví dụ, một chương trình Pascal có thể tạo các phân đoạn riêng có chiều dài khác nhau như hình dưới

1. Bảng ký hiệu để lưu các biến toàn cục.
2. Ngăn xếp để lưu các tham số và các biến riêng của các chương trình con mỗi khi chúng được gọi.
3. Phần mã của chương trình con.
4. Phần mã của chương trình chính.



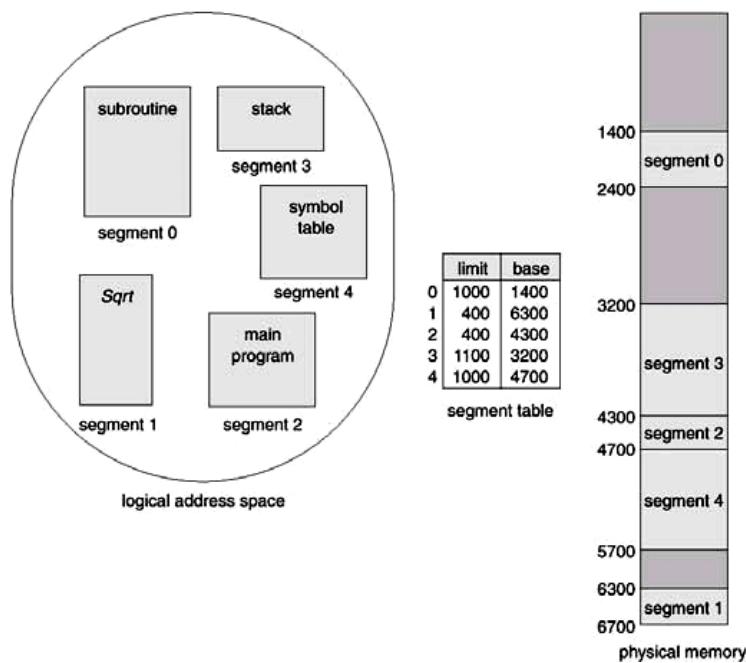
Vấn đề đặt ra là tìm một cách thức biểu diễn bộ nhớ sao cho nó gần với cách nhìn nhận của người sử dụng hơn. Kỹ thuật phân đoạn bộ nhớ có thể thực hiện được mục tiêu này.

Ý tưởng của kỹ thuật phân đoạn

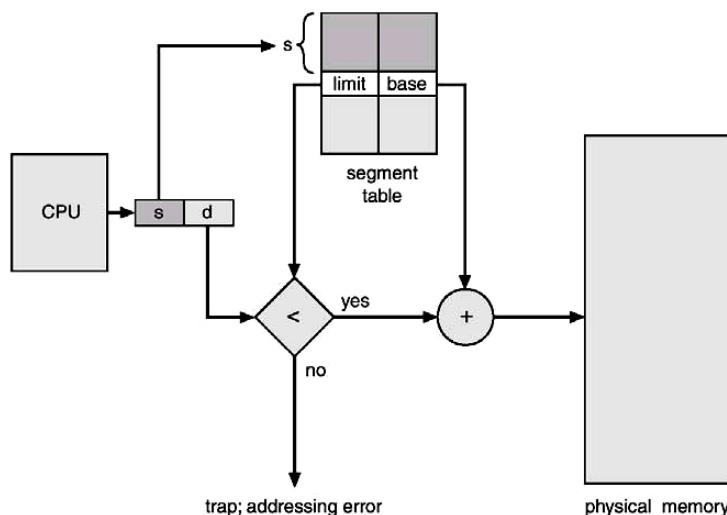
Kỹ thuật phân đoạn là một cơ chế quản lý bộ nhớ hỗ trợ góc nhìn bộ nhớ của người dùng. Không gian địa chỉ logic là một tập hợp các phân đoạn (segment), mỗi phân đoạn có tên (dựa vào chức năng của phân đoạn) và chiều dài khác nhau. Để đơn giản việc cài đặt, các phân đoạn được đánh số và được tham chiếu bởi *số hiệu phân đoạn*. Mỗi phần tử trong một phân đoạn được xác định bởi *bước nhảy* (offset, displacement) của chúng từ điểm bắt đầu của phân đoạn. Do đó, địa chỉ logic là một bộ đôi <số hiệu phân đoạn, bước nhảy>.

Cơ chế hoạt động của kỹ thuật phân đoạn

Cần phải chuyển đổi các địa chỉ 2 chiều <số hiệu phân đoạn, bước nhảy> thành địa chỉ vật lý một chiều. Sự chuyển đổi này được thực hiện qua một cấu trúc *bảng phân đoạn* (segment table). Mỗi mục từ của bảng phân đoạn bao gồm hai giá trị, *nền phân đoạn* (segment base) và *giới hạn phân đoạn* (segment limit). Nền phân đoạn chứa địa chỉ vật lý nơi bắt đầu của phân đoạn trong bộ nhớ, giới hạn phân đoạn xác định chiều dài của phân đoạn. Hình dưới mô tả mô tả một trường hợp nạp tiến trình theo kỹ thuật phân đoạn.



Cơ chế chuyển đổi địa chỉ được mô tả như trong hình dưới đây. Mỗi địa chỉ logic có hai phần: số hiệu phân đoạn s và bước nhảy d . Số hiệu phân đoạn được dùng như chỉ mục trong bảng phân đoạn. Bước nhảy d của địa chỉ logic phải ở trong khoảng từ 0 tới giới hạn phân đoạn, nếu không hệ thống sẽ gây ra một ngắt cứng để báo lỗi địa chỉ vật lý vượt qua điểm cuối của phân đoạn. Nếu bước nhảy này là hợp lệ thì nó được cộng thêm giá trị nền của phân đoạn để tạo ra địa chỉ trong bộ nhớ vật lý ứng với địa chỉ logic.

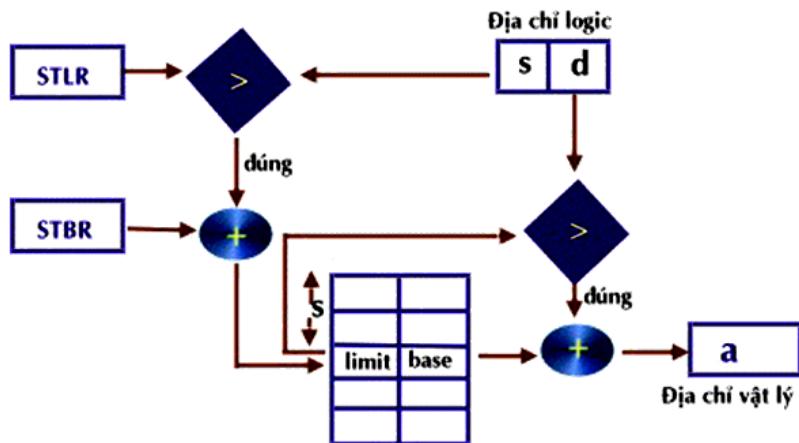


Hỗ trợ phần cứng trong kỹ thuật phân đoạn

Để tăng tốc độ tra cứu bảng phân đoạn, có thể sử dụng các thanh ghi để lưu trữ bảng phân đoạn nếu số lượng phân đoạn nhỏ. Trong trường hợp chương trình bao

gồm quá nhiều phân đoạn, bảng phân đoạn phải được lưu trong bộ nhớ chính, tuy nhiên trường hợp này không nhiều. Khi đó phải có một thanh ghi nền bảng phân đoạn (STBR, segment table base register) chỉ đến địa chỉ bắt đầu của bảng phân đoạn trong bộ nhớ. Vì số lượng phân đoạn sử dụng trong một chương trình biến động, cần sử dụng thêm một thanh ghi độ dài bảng phân đoạn (STLR, segment table length register), độ dài tức là số mục từ trong bảng phân đoạn.

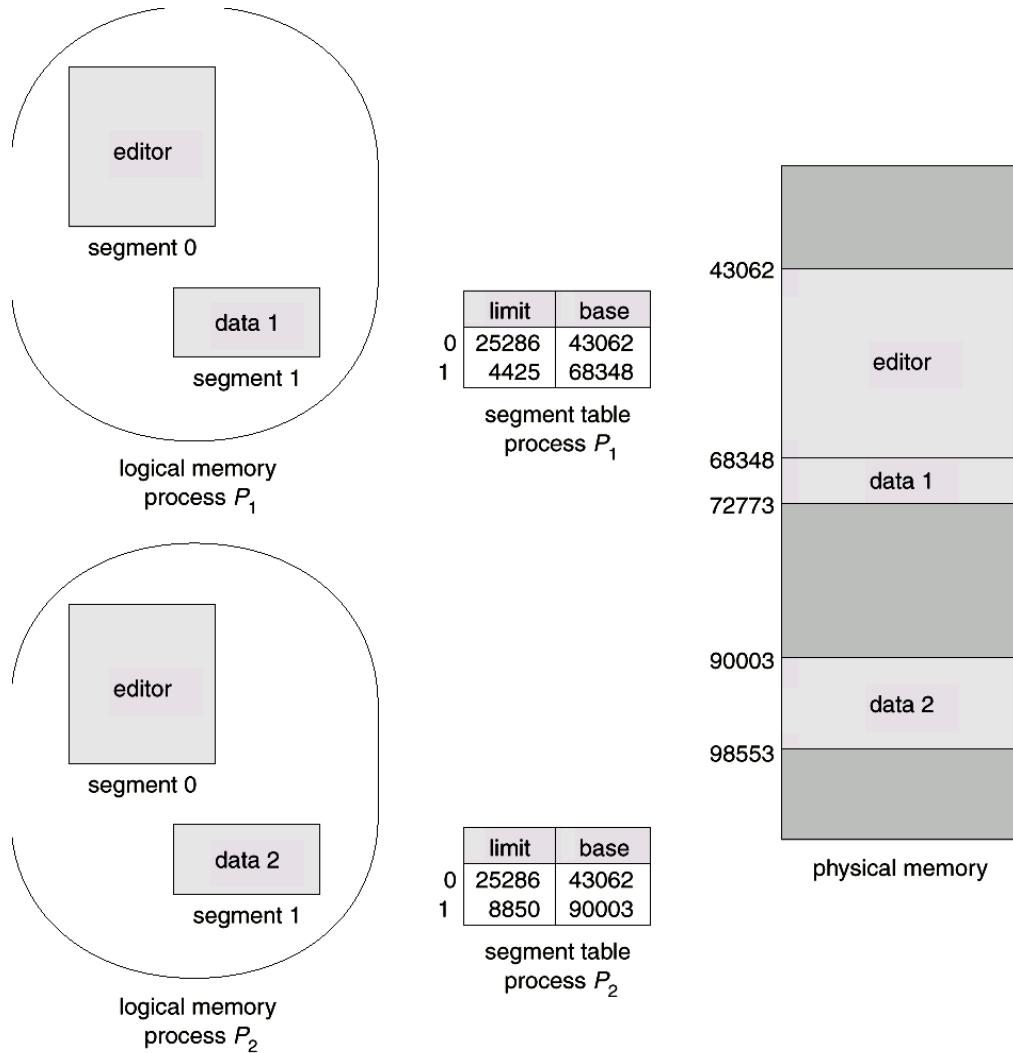
Với một địa chỉ logic $<s,d>$, trước tiên số hiệu phân đoạn s được kiểm tra tính hợp lệ (STLR> s). Kế tiếp, cộng giá trị s với STBR để có được địa chỉ địa chỉ của mục từ thứ s trong bảng phân đoạn (STBR+ s). Địa chỉ vật lý cuối cùng là (STBR+ s + d). Quá trình chuyển đổi này được mô tả trong hình dưới đây.



Bảo vệ bộ nhớ trong kỹ thuật phân đoạn

Trong sơ đồ hoạt động của kỹ thuật phân trang, chúng ta đã thấy địa chỉ logic được kiểm tra tính hợp lệ trước khi chuyển đổi. Lợi điểm đặc biệt của kỹ thuật phân đoạn là sự gắn liền việc bảo vệ với các phân đoạn. Các phân đoạn biểu diễn một phần xác định của chương trình, trong đó một số phân đoạn là chỉ thị, một số phân đoạn khác là dữ liệu. Trong một kiến trúc hiện đại, các chỉ thị không hiệu chỉnh chính nó, vì thế các phân đoạn chỉ thị có thể được xác định là chỉ đọc hoặc chỉ thực thi. Phần cứng chuyển đổi địa chỉ sẽ kiểm tra các bit bảo vệ được gắn với mỗi mục từ trong bảng phân đoạn để ngăn chặn các truy xuất không hợp lệ tới bộ nhớ, như cố gắng viết vào một phân đoạn chỉ đọc hay truy xuất vào một phân đoạn chỉ thực thi như truy xuất dữ liệu. Do đó, nhiều lỗi chương trình sẽ được phát hiện bởi phần cứng trước khi chúng có thể gây ra tác hại lớn.

Chia sẻ bộ nhớ trong kỹ thuật phân đoạn



Một lợi điểm khác của kỹ thuật phân đoạn liên quan đến việc chia sẻ mã hay dữ liệu của chương trình. Mỗi tiến trình có một bảng phân đoạn gắn với nó. Các phân đoạn được chia sẻ khi các mục từ trong bảng phân đoạn của hai tiến trình khác nhau chỉ tới cùng một vị trí vật lý như hình trên.

Trong sơ đồ trên, một trình soạn thảo văn bản được sử dụng bởi nhiều người dùng trong một hệ thống chia sẻ thời gian. Trình soạn thảo hoàn chỉnh có thể rất lớn, được hình thành từ nhiều phân đoạn có thể được chia sẻ giữa tất cả người dùng, giới hạn địa chỉ vật lý được yêu cầu hỗ trợ các tác vụ soạn thảo. Thay vì cần nhiều bản sao của trình soạn thảo, chúng ta chỉ cần một bản sao. Đối với mỗi người dùng, chúng ta vẫn

cần các phân đoạn riêng để lưu các biến cục bộ. Dĩ nhiên, các phân đoạn này sẽ không được chia sẻ.

Sự phân mảnh trong kỹ thuật phân đoạn

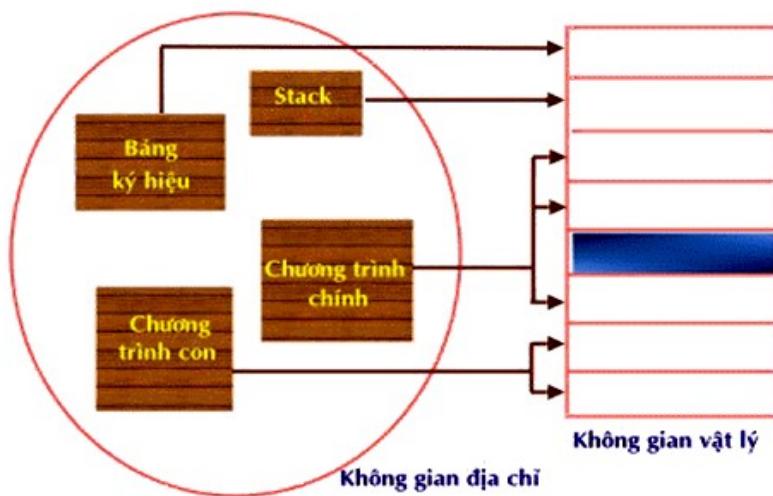
Do các phân đoạn có kích thước khác nhau và mỗi phân đoạn được cấp phát một khối bộ nhớ liên tục nên kỹ thuật phân đoạn có thể gây ra sự phân mảnh ngoại vi, tương tự như kỹ thuật cấp phát liên tục. Có thể giảm thiểu hiện tượng phân mảnh bằng các chiến lược cấp phát như *first fit*, *best fit*, *worst fit*. Khi tất cả khối bộ nhớ trống là quá nhỏ để chứa một phân đoạn, tiến trình có thể phải chờ cho đến khi có một khối bộ nhớ trống liên tục đủ lớn. Vì kỹ thuật phân đoạn có thể tái định vị các đoạn nên có thể tiến hành gom bộ nhớ bất cứ khi nào, tất nhiên là cũng chiếm khá nhiều thời gian.Thêm nữa, trong khi tiến trình lớn chờ đợi, bộ điều phối CPU có thể tìm một tiến trình nhỏ hơn để chạy.

12.3.3 Phân đoạn kết hợp phân trang

Cả hai kỹ thuật phân đoạn và phân trang đều có những lợi điểm và nhược điểm riêng. Có thể kết hợp hai kỹ thuật để tận dụng lợi điểm của chúng. Sự kết hợp này được thể hiện tốt nhất bởi kiến trúc của Intel 386.

Ý tưởng chung của việc kết hợp này là :

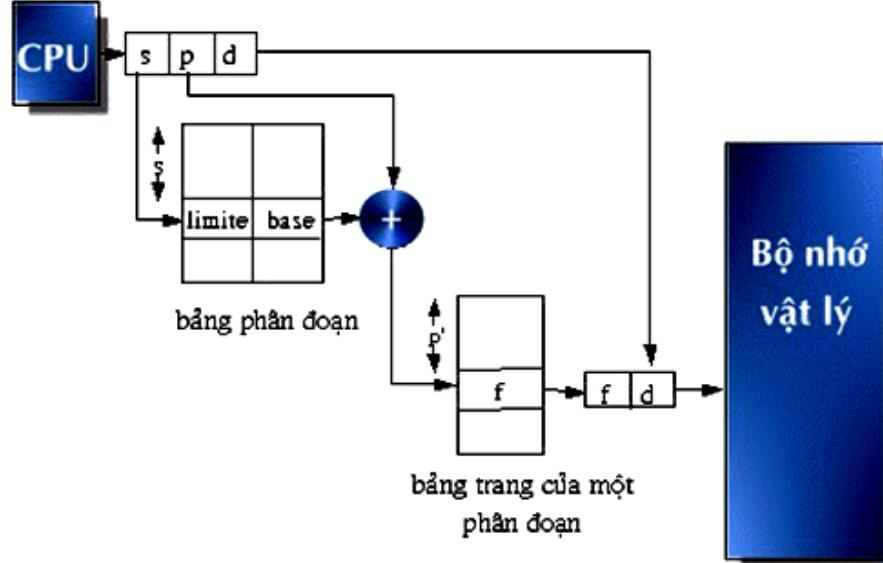
- Bộ nhớ vật lý được chia thành các khung trang có kích thước bằng nhau.
- Bộ nhớ logic của tiến trình được chia thành nhiều đoạn (segment), có thể căn cứ vào chức năng của từng đoạn. Mỗi đoạn được chia thành nhiều trang (page).
- Khi nạp tiến trình vào hệ thống, hệ điều hành sẽ cấp phát các khung trang trống để chứa đủ các đoạn của tiến trình, như mô tả trong hình dưới đây.



Cơ chế chuyển đổi địa chỉ trong kỹ thuật hết hợp phân đoạn với phân trang được mô tả trong hình dưới.

Để hỗ trợ kỹ thuật phân đoạn, mỗi tiến trình cần có một *bảng phân đoạn*, nhưng mỗi phân đoạn lại cần có một *bảng trang* riêng. Khi đó, mỗi địa chỉ logic là một bộ ba: $\langle s, p, d \rangle$, trong đó :

- s là số hiệu phân đoạn, được sử dụng như chỉ mục đến mục từ tương ứng trong bảng phân đoạn.
- p là số hiệu trang, được sử dụng như chỉ mục đến mục từ tương ứng trong bảng trang của phân đoạn.



- d là bước nhảy, chỉ địa chỉ tương đối trong trang của địa chỉ logic, được dùng kết hợp với địa chỉ nền của trang trong bộ nhớ vật lý để tạo ra địa chỉ vật lý tương ứng với địa chỉ logic.

CÂU HỎI VÀ BÀI TẬP

1. Hãy trình bày khái niệm địa chỉ logic và địa chỉ vật lý. Hãy cho biết tên tiếng Anh của bộ phận phần cứng chịu trách nhiệm chuyển đổi địa chỉ logic sang địa chỉ vật lý.
2. Vẽ sơ đồ ánh xạ và bảo vệ bộ nhớ trong kỹ thuật cấp phát bộ nhớ liên tục.
3. Giả sử giá trị của thanh ghi nền là 8192, thanh ghi giới hạn là 16384. Hãy tính địa chỉ logic của ô nhớ có địa chỉ vật lý là 12288
4. Cho biết sự khác nhau giữa phương pháp cấp phát liên tục và phương pháp cấp phát không liên tục. Ưu và nhược điểm của phương pháp cấp phát không liên tục so với kỹ thuật cấp phát liên tục.
5. Hãy trình bày phương pháp cấp phát theo kỹ thuật phân trang. Hãy vẽ sơ đồ mô hình cơ chế chuyển đổi địa chỉ logic sang địa chỉ vật lý trong kỹ thuật phân trang.
6. Hãy trình bày mục đích, ý tưởng và sơ đồ phương pháp cấp phát TLB trong kỹ thuật phân trang.
7. Hãy trình bày ý tưởng và sơ đồ mô hình của bảng trang nghịch đảo.
8. Hãy trình bày phương pháp cấp phát kết hợp kỹ thuật phân trang và kỹ thuật phân đoạn.

BÀI 13. QUẢN LÝ BỘ NHỚ ẢO

13.1 MỞ ĐẦU

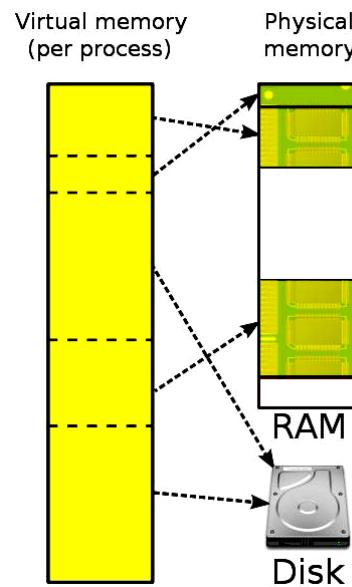
13.1.1 Khái niệm

Bộ nhớ ảo (virtual memory) là một kỹ thuật giúp giải quyết vấn đề thiếu bộ nhớ vật lý (bộ nhớ thật) khi chạy đồng thời nhiều tiến trình hoặc tiến trình có kích thước lớn hơn bộ nhớ vật lý. Nguyên lý cơ bản của bộ nhớ ảo là vẫn dựa trên hai kỹ thuật phân trang và phân đoạn, nhưng trong kỹ thuật bộ nhớ ảo:

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

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

Có thể cài đặt kỹ thuật bộ nhớ ảo theo kỹ thuật *phân trang theo yêu cầu* (demand paging) hoặc theo kỹ thuật *phân đoạn theo yêu cầu* (demand segmentation). Cả hai kỹ thuật trên đều phải có sự hỗ trợ của phần cứng máy tính, cụ thể là CPU. Tuy nhiên, các giải thuật thay thế đoạn phức tạp hơn các giải thuật thay thế trang vì các đoạn có kích thước khác nhau. Vì vậy, đa số các hệ điều hành chọn kỹ thuật phân



trang theo yêu cầu. Chúng ta sẽ không đề cập đến kỹ thuật phân đoạn theo yêu cầu trong tài liệu này. Khi cài đặt kỹ thuật bộ nhớ ảo hệ điều hành cần phải :

- Có một lượng không gian bộ nhớ phụ (đĩa) cần thiết đủ để chứa các trang/đoạn bị hoán vị ra, không gian đĩa này được gọi là *không gian hoán vị*.
- Có cơ chế để theo dõi các trang/đoạn của mỗi tiến trình đang hoạt động, xem chúng đang ở trên bộ nhớ chính hay ở trên bộ nhớ phụ. Trong trường hợp này, hệ điều hành thường đưa thêm một bit trạng thái (bit present) vào các mục từ trong bảng trang hoặc bảng đoạn.
- Giảm thiểu số lần hoán vị trang/đoạn bằng cách có giải thuật tốt để chọn một trang nào đó trong số các trang đang ở trên bộ nhớ chính để hoán vị ra trong trường hợp cần thiết.

13.1.2 Lợi ích

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

- Hệ điều hành có thể nạp được nhiều tiến trình hơn vào bộ nhớ vì hệ thống không nạp tất cả tiến trình vào bộ nhớ và nếu cần có thể hoán vị ra các trang/đoạn của một tiến trình nào đó trên bộ nhớ. Lợi ích của việc nạp nhiều tiến trình vào bộ nhớ chúng ta đã biết trong bài trước.
- Có thể nạp vào bộ nhớ một tiến trình có không gian địa chỉ lớn hơn tất cả không gian địa chỉ của bộ nhớ vật lý. Trong thực tế người lập trình có thể thực hiện việc này mà không cần sự hỗ trợ của hệ điều hành và phần cứng bằng cách thiết kế chương trình theo cấu trúc Overlay, việc làm này là quá khó đối với người lập trình. Với kỹ thuật bộ nhớ ảo người lập trình không cần quan tâm đến kích thước của chương trình và kích thước của bộ nhớ tại thời điểm nạp chương trình, tất cả mọi việc này đều do hệ điều hành và phần cứng thực hiện.
- Thời gian dành cho việc hoán vị sẽ ít hơn so với việc hoán vị cả chương trình như ở bài trước vì mỗi khi hoán vị chỉ hoán vị một trang/đoạn chứ không phải cả chương trình. Do vậy, hệ thống sẽ chạy nhanh hơn.

13.2 PHÂN TRANG THEO YÊU CẦU

13.2.1 Khái niệm

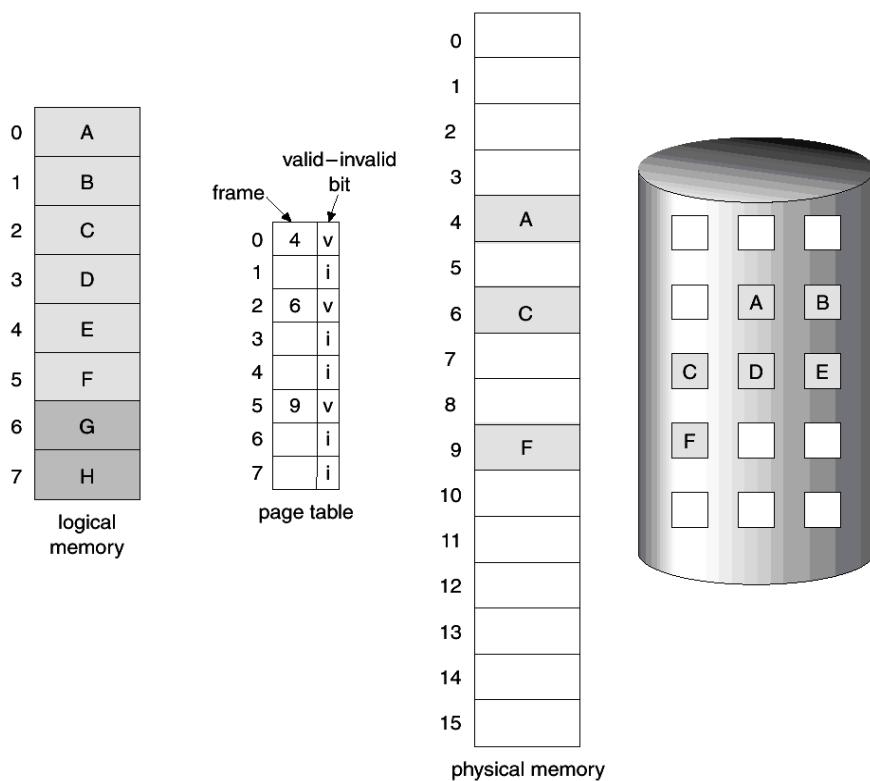
Hệ thống phân trang theo yêu cầu là hệ thống sử dụng kỹ thuật phân trang kết hợp với kỹ thuật hoán vị. Mỗi tiến trình được xem như một tập các trang, thường trú trên bộ nhớ phụ (thường là đĩa). Khi cần xử lý, tiến trình sẽ được nạp vào bộ nhớ chính. Nhưng thay vì nạp toàn bộ chương trình, hệ thống chỉ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

13.2.2 Hỗ trợ 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 cơ chế hỗ trợ kỹ thuật hoán vị, bao gồm :

- *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 hoán vị được gọi là không gian hoán vị.

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 đang ở trên đĩa. Có thể sử dụng lại bit valid-invalid đã nêu trong bài trước nhưng với ngữ nghĩa mới: valid nghĩa là trang tương ứng là hợp lệ (tức là trang của tiến trình) và đang ở trong bộ nhớ chính, invalid nghĩa là trang không hợp lệ hoặc đang được lưu trên bộ nhớ phụ. Mục từ trong bảng trang ứng với trang không nằm trong bộ nhớ chính sẽ được đánh dấu invalid và chứa địa chỉ của trang trên bộ nhớ phụ. Cơ chế hỗ trợ phần cứng cho kỹ thuật phân trang theo yêu cầu được mô tả trong hình sau.



13.2.3 Xử lý lỗi trang

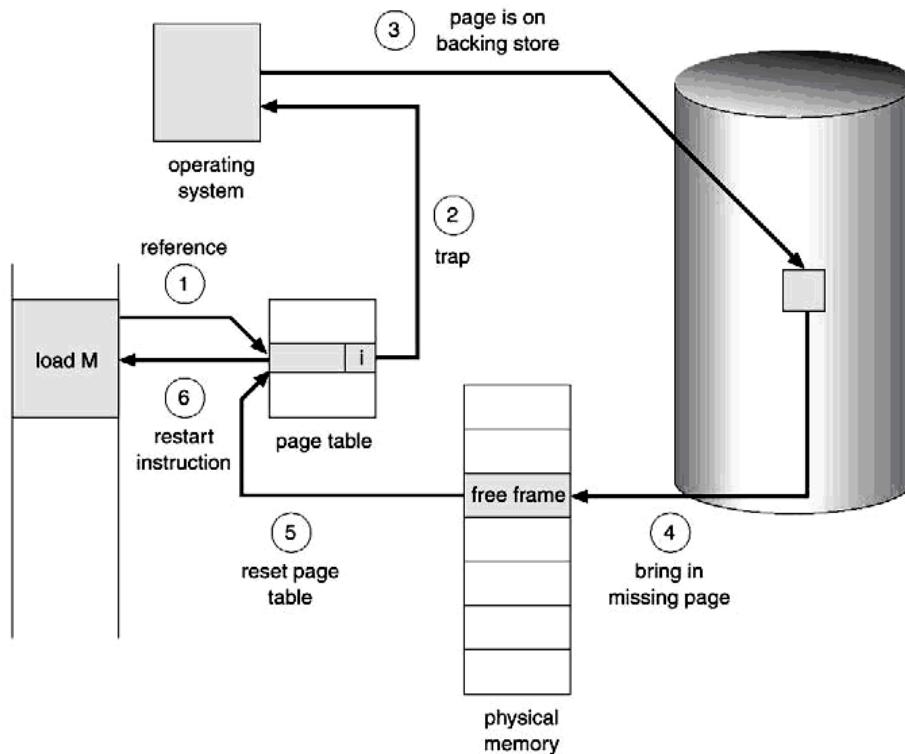
Lỗi trang (page fault) là việc CPU yêu cầu truy xuất đến một trang được đánh dấu invalid. Cơ chế phần cứng sẽ phát sinh một ngắt để báo cho hệ điều hành. Hệ điều hành sẽ xử lý theo sơ đồ sau.

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

1. Kiểm tra bảng bên trong của tiến trình (thường được giữ với khối điều khiển tiến trình) để xác định tham chiếu là truy xuất bộ nhớ hợp lệ hay không hợp lệ.
2. Nếu tham chiếu là không hợp lệ thì kết thúc tiến trình. Nếu nó là hợp lệ nhưng không ở trong bộ nhớ chính thì tiến hành các bước tiếp theo để nạp trang đó vào bộ nhớ chính.
3. Tìm khung trang trống. Nếu không có khung trang trống thì phải tiến hành hoán vị ra một trang để có khung trang trống, thao tác này sẽ được trình bày trong phần nói về thay thế trang.

4. Nạp trang từ bộ nhớ phụ vào khung trang trống.
5. Cập nhật lại bảng trang để hiển thị rằng trang bấy giờ ở trong bộ nhớ.
6. Khởi động lại chỉ thị bị ngắt bởi tham chiếu không hợp lệ.

Hình dưới mô tả các bước kể trên



Chú ý rằng thời gian truy xuất trang khi phải xử lý lỗi trang lớn hơn rất nhiều so với thời gian truy xuất khi không bị lỗi trang vì phải có thêm rất nhiều thao tác như đã trình bày ở trên.

Gọi m là thời gian truy xuất nếu không bị lỗi trang, n là thời gian truy xuất khi phải xử lý lỗi trang, p là xác suất xảy ra lỗi trang, ta có công thức tính thời gian trung bình cho một truy xuất trang t là : $t = (1-p)*m + p*n$

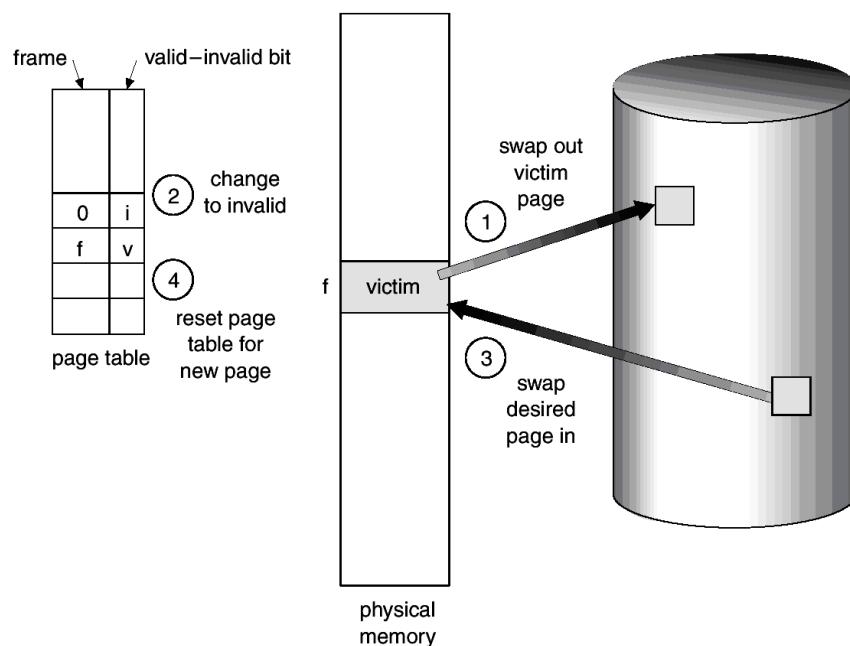
Do n lớn hơn m rất nhiều nên p càng nhỏ thì t càng nhỏ. Ví dụ, nếu $m=20$ nano giây, $n=200$ nano giây, khi $p=50\%$ thì $t=0.5*20+0.5*200=110$ nano giây, khi $p=10\%$ thì $t=0.9*20+0.1*200=21.8$ nano giây. Như vậy, để hệ thống không bị chậm, phải làm sao cho xác suất lỗi trang càng thấp càng tốt. Vấn đề này liên quan đến các giải thuật thay thế trang sẽ được trình bày ở phần dưới.

13.3 THAY THẾ TRANG

13.3.1 Khái niệm

Thay thế trang là việc chọn một trang đang nằm trong bộ nhớ mà không được sử dụng tại thời điểm hiện tại, chuyển nó ra không gian hoán vị trên bộ nhớ phụ để giải phóng một khung trang, nạp trang cần truy xuất (đang ở bộ nhớ phụ) vào khung trang vừa được giải phóng chính để truy xuất. Đây là trường hợp được đề cập tới ở bước 3 trong quá trình xử lý lỗi trang. Như vậy, thay thế trang gồm hai thao tác chính: 1-sao chép một trang (trang nạn nhân) từ bộ nhớ chính ra bộ nhớ phụ (hoán vị ra), 2-sao chép một trang từ bộ nhớ phụ vào bộ nhớ chính (hoán vị vào). Như đã nói ở trên, thời gian xử lý lỗi trang khi không phải thay thế trang đã lớn, trong trường hợp phải thay thế trang lại càng lớn vì có thêm hai thao tác trên.

Hình dưới mô tả các bước thay thế trang.



Có thể giảm bớt số lần hoán vị ra bằng cách sử dụng thêm một bit cập nhật (dirty bit). Dirty bit được gắn với mỗi trang đang trong bộ nhớ chính để biết sau khi được nạp vào bộ nhớ chính, nội dung trang có bị thay đổi không. Nếu nội dung không bị thay đổi thì không cần sao chép trang từ bộ nhớ chính ra bộ nhớ phụ mà chỉ cần sao chép trang mới vào khung trang được giải phóng, đè lên nội dung cũ.

Như đã nói ở trên, để hệ thống có thể hoạt động nhanh, cần phải làm cho xác suất lỗi trang càng thấp càng tốt. Như vậy cần phải đưa ra giải thuật chọn trang đưa ra ngoài khi thay thế trang để có xác suất lỗi trang thấp. Trong các phần tiếp theo, chúng ta sẽ xem xét các giải thuật thay thế trang phổ biến như giải thuật FIFO, giải thuật tối ưu, giải thuật LRU (Least-recently-used)...

13.3.2 Giải thuật FIFO

Giải thuật thay thế trang đơn giản nhất là giải thuật FIFO (first in first out). Ý tưởng của giải thuật FIFO là *thay thế trang đã ở trong bộ nhớ vật lý lâu nhất*. Có thể tạo một hàng đợi FIFO để quản lý các trang đang ở trong bộ nhớ. Khi cần thay thế trang, chọn trang ở đầu hàng đợi và xóa nó ra khỏi hàng đợi. Khi trang được nạp vào bộ nhớ chính, đưa nó vào đuôi của hàng đợi.

Xem xét ví dụ sau. Giả sử một tiến trình có 8 trang, đánh số từ 0 đến 7, tiến trình được cấp 3 khung trang. Thứ tự truy xuất các trang của tiến trình như sau 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1. Ba lần truy xuất đầu tiên (7, 0, 1) gây ra lỗi trang và được mang vào các khung rỗng, không có thay thế trang. Lần truy xuất thứ tư (2) sẽ dẫn đến việc thay thế trang 7 vì trang 7 được nạp vào bộ nhớ chính trước tiên. Lần truy xuất thứ năm (0) không có lỗi trang vì 0 đã ở trong bộ nhớ chính. Lần truy xuất thứ sáu (3) sẽ dẫn đến việc thay thế trang 0 vì nó là trang đầu tiên của 3 trang trong bộ nhớ (0, 1, 2) được nạp vào bộ nhớ chính...

Kết quả của việc thay thế trang theo giải thuật FIFO được thể hiện như bảng sau, dòng "Trang" cho biết trang được yêu cầu truy xuất, dòng "Khung trang" cho biết trang được nạp vào khung trang nào, dòng "Lỗi" cho biết lần truy xuất đó có xảy ra lỗi trang hay không. Kết quả cho thấy có 15 lỗi trang trên 20 lần truy xuất, trong đó có 12 lần thay thế trang.

Giải thuật thay thế trang FIFO dễ hiểu, dễ cài đặt. Tuy nhiên giải thuật này không phải là một giải thuật tốt, nó không làm giảm xác suất lỗi trang, thậm chí nó còn có thể dẫn đến nghịch lý Belady : Khi tăng số lượng khung trang cấp cho tiến trình, xác suất lỗi trang có thể không giảm mà lại tăng lên. Xét ví dụ sau, giả sử có chuỗi truy xuất trang : 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5. Nếu sử dụng 3 khung trang, sẽ có 9 lỗi trang phát sinh. Nếu sử dụng 4 khung trang, sẽ có 10 lỗi trang phát sinh. Kết quả này thể hiện trong hai bảng sau.

Trang	1	2	3	4	1	2	5	1	2	3	4	5
Khung trang	1	1	1	4	4	4	5	5	5	5	5	5
		2	2	2	1	1	1	1	1	3	3	3
			3	3	3	2	2	2	2	2	4	4
Lỗi	*	*	*	*	*	*	*			*	*	

Trang	1	2	3	4	1	2	5	1	2	3	4	5
Khung trang	1	1	1	1	1	1	5	5	5	5	4	4
		2	2	2	2	2	2	2	1	1	1	5
			3	3	3	3	3	3	2	2	2	2
				4	4	4	4	4	4	3	3	3
Lỗi	*	*	*	*			*	*	*	*	*	*

13.3.3 Giải thuật tối ưu

Giải thuật thay thế trang tối ưu có xác suất lỗi trang thấp nhất trong tất cả các giải thuật và không gặp phải nghịch lý Belady. Ý tưởng của giải thuật này là *thay thế trang sẽ không được dùng trong một khoảng thời gian lâu nhất trong tương lai*. Tuy nhiên, trong thực tế, giải thuật thay thế trang tối ưu là khó cài đặt vì nó yêu cầu thông tin về tương lai của chuỗi truy xuất. Do đó, giải thuật tối ưu được dùng chủ yếu cho nghiên cứu.

Xét ví dụ ở mục trước nhưng sử dụng giải thuật thay thế trang tối ưu. Ba lần truy xuất đầu tiên (7, 0, 1) gây ra lỗi trang và được mang vào các khung rỗng, không có thay thế trang. Lần truy xuất thứ tư (2) sẽ dẫn đến việc thay thế trang 7 vì trang 7 sẽ không được dùng cho tới lần truy xuất thứ 18, trong khi trang 0 sẽ được dùng tại lần truy xuất thứ 5 và trang 1 sẽ được dùng tại lần truy xuất thứ 14. Lần truy xuất thứ năm (0) không có lỗi trang vì 0 đã ở trong bộ nhớ chính. Lần truy xuất thứ sáu (3) sẽ dẫn đến việc thay thế thay thế trang 1 vì trang 1 sẽ là trang cuối cùng của ba trang trong bộ nhớ được truy xuất lần nữa...Với chỉ 9 lỗi trang, thay thế tối ưu là tốt hơn nhiều so với giải thuật FIFO, có 15 lỗi. Nếu chúng ta bỏ qua 3 lỗi đầu mà tất cả các giải thuật đều phải gặp thì thay thế tối ưu tốt gấp 2 lần thay thế FIFO. Kết quả của ví dụ được thể hiện trong bảng sau.

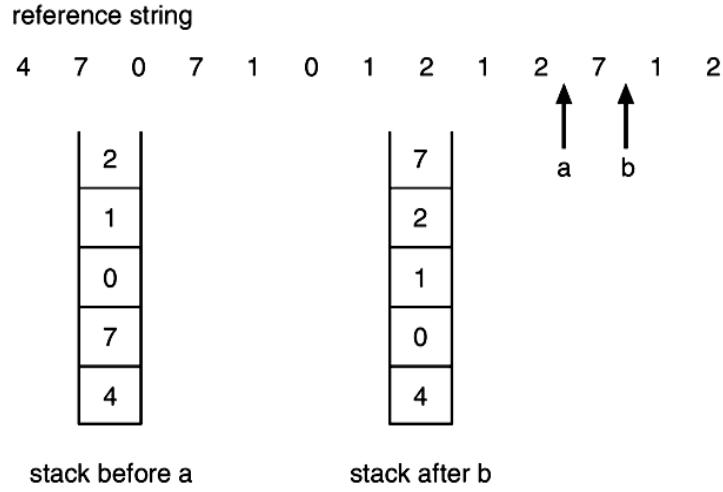
Trang	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
Khung trang	7	7	7	2	2	2	2	2	2	2	2	2	2	2	2	2	2	7	7	7
	0	0	0	0	0	0	0	4	4	4	0	0	0	0	0	0	0	0	0	0
	1	1	1	3	3	3	3	3	3	3	3	3	3	1	1	1	1	1	1	1
	*	*	*	*	*	*	*		*		*		*			*			*	

13.3.4 Giải thuật LRU

Giải thuật thay thế trang LRU (least-recently-used, ít được dùng gần đây nhất) sử dụng quá khứ gần như một xấp xỉ của tương lai gần. Ý tưởng của giải thuật LRU là khi phải thay thế một trang, sẽ *chọn trang đã không được dùng trong một khoảng thời gian lâu nhất* trong quá khứ. Đây này là giải thuật thay thế trang tối ưu tìm kiếm lùi theo thời gian, trong khi giải thuật tối ưu là tìm kiếm tới trước. Nếu gọi SR là chuỗi ngược của chuỗi truy xuất S thì tỉ lệ lỗi trang cho giải thuật tối ưu trên S là tương tự như tỉ lệ lỗi trang cho giải thuật LRU trên SR. Tương tự, tỉ lệ lỗi trang đối với giải thuật LRU trên S là giống như tỉ lệ lỗi trang cho giải thuật tối ưu trên SR.

Xét ví dụ ở mục trước nhưng sử dụng giải thuật thay thế LRU, bảng trên cho thấy giải thuật thay thế LRU có 12 lỗi, tuy nhiều lỗi hơn giải thuật thay thế tối ưu (9 lỗi) nhưng vẫn tốt hơn giải thuật thay thế FIFO (15 lỗi).

Giải thuật LRU được áp dụng nhiều trong thực tế. Vấn đề chính là cách cài đặt, giải thuật này đòi hỏi phải có cơ chế phần cứng hỗ trợ để xác định thứ tự cho các trang trong bộ nhớ chính theo thời điểm truy xuất cuối cùng. Có thể cài đặt theo một trong hai cách sau.



13.4 CẤP PHÁT KHUNG TRANG

Trong hệ điều hành đơn nhiệm, có thể cấp phát cho tiến trình người dùng duy nhất tất cả các khung trang, nghĩa là tiến trình duy nhất đó có thể sử dụng toàn bộ năng lực bộ nhớ cho dù số lượng khung trang đó là thừa so với nhu cầu của tiến trình. Vấn đề nảy sinh khi hệ điều hành là hệ đa chương, cần phải chia sẻ các khung trang cho

nhiều tiến trình. Trong hệ đa chương có hai nhu cầu mâu thuẫn nhau: vừa muốn thực thi cùng lúc nhiều tiến trình lại vừa muốn đảm bảo tốc độ hoạt động của hệ thống (tức là muốn số lỗi trang trên toàn hệ thống là ít nhất). Để có thể cho phép nhiều tiến trình hoạt động cùng lúc, cần phải cấp cho mỗi tiến trình chỉ một số ít khung trang, nhưng khi đó số lỗi trang sẽ tăng lên, dẫn tới tốc độ hoạt động của hệ thống bị giảm xuống, thậm chí dẫn tới hiện tượng trì trệ toàn hệ thống. Ngược lại, để hệ thống có tốc độ hoạt động tốt, cần phải giảm số lỗi trang, như vậy cần phải cấp nhiều khung trang cho mỗi tiến trình, nhưng điều này lại dẫn tới việc phải giảm số lượng tiến trình hoạt động đồng thời trong hệ thống. Thông thường, trong hệ đa chương, dung lượng bộ nhớ chính là thiếu so với nhu cầu, nên không thể cấp thừa cho tiến trình. Vậy sẽ cấp khung trang cho mỗi tiến trình như thế nào để hệ thống có thể vận hành một cách tối ưu, dung hòa được hai nhu cầu mâu thuẫn nhau vừa kể trên.

13.4.1 Số khung trang tối thiểu

Để có thể thực hiện được một chỉ thị (lệnh máy), cần phải có đủ trong bộ nhớ tất cả các trang mà chỉ thị đó cần tham chiếu tới. Ví dụ, một chỉ thị nằm ở trang 10 có thể tham chiếu tới một địa chỉ biến con trỏ nằm ở trang 3, biến này lại tham chiếu gián tiếp tới một địa chỉ ở trang 20, khi đó cả ba trang trên phải đang nằm trong bộ nhớ chính thì mới có thể thực hiện được chỉ thị. Trong trường hợp này, số khung trang tối thiểu cần phải cấp cho tiến trình không thể dưới ba. Số khung trang tối thiểu cần cấp cho một tiến trình được qui 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. Có thể có nhiều lựa chọn giữa hai con số đó.

13.4.2 Các giải thuật cấp phát

Như đã nói ở trên, mục tiêu của các giải thuật cấp phát khung trang là cố gắng dung hòa các nhu cầu mâu thuẫn nhau của hệ thống. Nhìn chung là có hai hướng tiếp cận để cấp phát khung trang.

1. Cấp phát cố định : Số lượng khung trang được cấp phát cho mỗi tiến trình là một số được tính toán ngay từ đầu, không thay đổi trong quá trình hoạt động của tiến trình. Kiểu cấp phát này có hai loại.

- *Cấp phát công bằng* : Nếu có hệ thống có m khung trang và n tiến trình, mỗi tiến trình được cấp m/n khung trang.
- *Cấp phát theo tỷ lệ* : Dựa vào kích thước của tiến trình để cấp phát số khung trang. Nếu s là kích thước của tiến trình p, S là tổng kích thước của tất cả các tiến trình, m là tổng số khung trang có thể sử dụng. Khi đó số khung trang cấp phát cho tiến trình p là $m*s/S$

2. Cấp phát theo độ ưu tiên : Sử dụng ý tưởng cấp phát theo tỷ lệ, nhưng số lượng khung trang cấp cho tiến trình phụ thuộc vào độ ưu tiên của tiến trình. Nếu một tiến trình cần thay thế trang, có thể chọn một trong các khung trang của nó hoặc chọn một khung trang của một tiến trình khác với độ ưu tiên thấp hơn để thay thế.

13.4.3 Thay thế cục bộ và thay thế toàn cục

Có thể phân các thuật toán thay thế trang thành hai lớp chính :

- *Thay thế cục bộ* : Yêu cầu chỉ được chọn trang thay thế trong tập các khung trang được cấp cho tiến trình phát sinh lỗi trang.
- *Thay thế toàn cục* : Khi lỗi trang xảy ra với một tiến trình, có thể chọn trang thay thế từ tất cả các khung trang trong hệ thống, kể cả khung trang đang được cấp phát cho một tiến trình khác. Một khuyết điểm của giải thuật thay thế toàn cục là các tiến trình không thể kiểm soát được tỷ lệ phát sinh lỗi trang của mình. Vì thế, tuy giải thuật thay thế toàn cục nhìn chung cho phép hệ thống có nhiều khả năng xử lý hơn, nhưng nó có thể dẫn hệ thống đến tình trạng trì trệ toàn bộ hệ thống (thrashing).

13.5 TRÌ TRỆ TOÀN HỆ THỐNG

13.5.1 Khái niệm và nguyên nhân

Sự trì trệ (thrashing) là hiện tượng tiến trình thường xuyên phát sinh lỗi trang và vì thế phải dùng rất nhiều thời gian sử dụng CPU để thực hiện việc thay thế trang. Nguyên nhân là do tiến trình không có đủ các khung trang để chứa những trang cần thiết cho xử lý công việc. Hiện tượng này ảnh hưởng nghiêm trọng đến hoạt động hệ thống, xét tình huống sau:

1. Hệ điều hành giám sát việc sử dụng CPU.
2. Nếu hiệu suất sử dụng CPU quá thấp, hệ điều hành sẽ nâng mức độ đa chương bằng cách đưa thêm một tiến trình mới vào hệ thống.
3. Hệ thống có thể sử dụng giải thuật thay thế toàn cục, chọn trang thay thế thuộc một tiến trình bất kỳ để có chỗ nạp tiến trình mới.
4. Mỗi tiến trình sẽ có ít khung trang hơn và do đó phát sinh nhiều lỗi trang hơn.
5. Các tiến trình phải dùng nhiều thời gian hơn để chờ việc thay thế trang hoàn tất, lúc đó hiệu suất sử dụng CPU lại giảm.
6. Hệ điều hành lại quay trở lại bước 1.

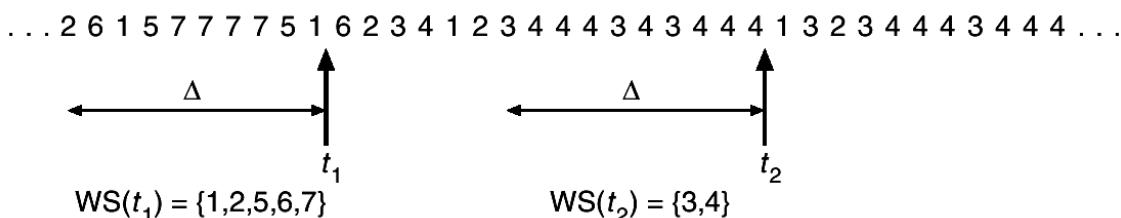
Như vậy, hệ thống sẽ lâm vào tình trạng luẩn quẩn của việc giải phóng các trang để cấp phát thêm khung trang cho một tiến trình, các tiến trình khác lại thiếu khung trang. Lỗi trang tăng cao, các tiến trình phải dành nhiều thời gian để chờ thay thế trang. Cuối cùng, thời gian dành cho xử lý công việc còn rất ít. Khi tình trạng này xảy ra, hệ thống gần như mất khả năng xử lý công việc. Tốc độ phát sinh lỗi trang tăng rất cao, không công việc nào có thể kết thúc vì tất cả tiến trình đều bận rộn với việc thay thế trang. Đây chính là tình trạng *trì trệ toàn bộ hệ thống*.

Để ngăn cản tình trạng trì trệ này xảy ra, cần phải cấp cho tiến trình đủ các khung trang cần thiết để hoạt động. Vấn đề cần giải quyết là làm sao đoán nhận được tiến trình đang thiếu hay là thừa khung trang. Sau đây là một số giải pháp.

13.5.2 Mô hình tập làm việc

Một trong các giải pháp cho tình trạng trì trệ toàn bộ hệ thống là *mô hình tập làm việc* (working set model). Mô hình này sử dụng tham số Δ để định nghĩa cửa sổ tập làm việc. Tập các trang được tiến trình truy xuất trong Δ lần truy xuất cuối cùng của tiến trình P được gọi là *tập làm việc của tiến trình P* tại thời điểm hiện tại. Nếu trang A đang được tiến trình P truy xuất tại thời điểm hiện tại, A sẽ nằm trong tập làm việc của tiến trình P. Nếu trong Δ lần truy xuất tiếp theo, tiến trình P không truy xuất A (mà chỉ truy xuất các trang khác) thì A sẽ bị loại khỏi tập làm việc của tiến trình P. Hình dưới đây mô tả khái niệm tập làm việc.

page reference table



Hệ điều hành kiểm soát tập làm việc của mỗi tiến trình và cấp phát cho mỗi tiến trình một số lượng tối thiểu các khung trang đủ để chứa tập làm việc của nó. Nếu có đủ khung trang trống bổ sung thì khởi tạo thêm tiến trình. Nếu tổng kích thước các tập làm việc (tức là tổng số khung trang được yêu cầu) của các tiến trình đang hoạt động vượt quá tổng số khung trang sẵn có, hệ thống có thể sẽ lâm vào tình trạng trì trệ. Khi đó, để làm giảm tổng kích thước các tập làm việc của các tiến trình đang hoạt động, hệ điều hành sẽ tạm dừng một số tiến trình. Những trang của tiến trình này được lưu ra đĩa và các khung trang của nó được cấp phát lại cho các tiến trình còn lại. Sau này, các tiến trình bị tạm dừng có thể được khởi động lại.

Mô hình tập làm việc có thể ngăn chặn sự trì trệ toàn hệ thống trong khi vẫn giữ cấp độ đa chương cao nhất có thể. Do đó, nó tối ưu hóa được việc sử dụng CPU. Khó khăn với mô hình tập làm việc là giữ vết của tập làm việc. Để giữ vết của tập làm việc người ta sử dụng khái niệm cửa sổ tập làm việc. Cửa sổ tập làm việc là một cửa sổ trượt. Cứ sau mỗi lần tham chiếu bộ nhớ, cửa sổ tập làm việc lại trượt tới một nấc trong chuỗi tham chiếu bộ nhớ của tiến trình. Một trang sẽ ở trong tập làm việc nếu nó được tham chiếu tại một thời điểm nào đó trong cửa sổ tập làm việc.

13.5.3 Kiểm soát tần suất lỗi trang

Một giải pháp khác cho tình trạng trì trệ toàn hệ thống là *kiểm soát tần suất lỗi trang*. Tần suất lỗi trang rất cao khiến tình trạng trì trệ hệ thống xảy ra. Khi tần suất lỗi trang quá cao, tiến trình cần thêm một số khung trang. Ngược lại, khi tần suất lỗi trang quá thấp, tiến trình có thể đang sở hữu nhiều khung trang hơn mức cần thiết.

Có thể thiết lập một giá trị cận trên và cận dưới cho tần suất xảy ra lỗi trang. 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 cận trên, cấp thêm khung trang cho tiến trình. Nếu tần suất lỗi trang thấp hơn cận dưới, thu hồi bớt khung trang của tiến trình.

Nếu tỉ lệ lỗi trang tăng và không có trang nào trống, phải chọn một số tiến trình và tạm dừng nó. Những khung trang được giải phóng sẽ được phân phối lại cho các tiến trình có tỉ lệ lỗi trang cao. Sau này các tiến trình bị tạm dừng sẽ được khởi động lại.

CÂU HỎI VÀ BÀI TẬP

1. Hãy trình bày khái niệm, ý tưởng và lợi ích của kỹ thuật bộ nhớ ảo.
 2. Hãy trình bày khái niệm lỗi trang và các bước xử lý lỗi trang. Vẽ sơ đồ xử lý lỗi trang.
 3. Công thức tính thời gian trung bình truy xuất cho một truy xuất trang. Vì sao phải giảm thiểu xác suất xảy ra lỗi trang.
 4. Hãy trình bày giải thuật tối ưu cho việc thay thế trang. Vì sao giải thuật này không áp dụng được trong thực tế.
 5. Hãy trình bày giải thuật LRU cho việc thay thế trang.
 6. Hãy thực hiện các giải thuật thay thế trang FIFO, LRU, giải thuật tối ưu cho chuỗi truy xuất trang sau (vẽ lại và điền kết quả vào bảng sau):

TÀI LIỆU THAM KHẢO

1. David A. Patterson, John L. Hennessy, Peter J. Ashenden, James R. Larus, Daniel J. Sorin, *Computer Organization and Design: The Hardware/Software Interface*, Fifth Edition, Morgan Kaufmann 2014.
2. David Money Harris, Sarah L. Harris, *Digital Design and Computer Architecture*, Morgan Kaufmann 2013.
3. PGS.TS. Nguyễn Hữu Phương, *Mạch số*, Nhà xuất bản Thống Kê 2001.
4. Tống Văn On, Hoàng Đức Hải, *Hợp ngữ và lập trình ứng dụng*, Nhà xuất bản Giáo Dục 2001
5. Abraham Silberschatz, Peter Baer Galvin, Greg Gagne (2011), *Operating System Concepts Essentials*. John Wiley & Sons, Inc.
6. Andrew S. Tanenbaum (2001), *Modern Operating Systems*, Second Edition. Prentice Hall PTR.
7. Nguyễn Kim Tuấn (2004), *Lý thuyết hệ điều hành*, Đại học Huế.
8. Nguyễn Phú Trường (2004), *Giáo trình môn hệ điều hành*, Đại học Cần thơ