

TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI

Khoa Công nghệ Thông tin

Bộ môn công nghệ phần mềm

===== o =====

Bài giảng

HỆ ĐIỀU HÀNH

60 tiết (45 tiết lý thuyết + 15 tiết thực hành)

(Version 0.11)

Biên soạn

Lê Tiến Dũng

Hà nội 8-2001

Chương 1. Các khái niệm cơ bản

LT: 6 Tiết, TH: 0 Tiết

1. Cấu trúc phân lớp và sự phát triển của hệ thống tính toán

a. Lịch sử phát triển của các hệ thống tính toán □ các nguyên lý phát triển của processor và hệ lệnh.

– 1940-1950

Đây là giai đoạn ra đời thế hệ đầu tiên của máy tính điện tử, với những hệ thống máy tính cấu tạo từ các bộ phận cơ khí và đèn điện tử. Các máy tính giai đoạn này vừa cồng kềnh (kích cỡ bằng một toà nhà) vừa phức tạp. Do đó một máy phải dùng nhiều người sử dụng.

Trong giai đoạn này: Phạm vi người sử dụng máy tính điện tử còn bị bó hẹp ở những chuyên gia mà thôi.

Người thiết kế xây dựng chương trình chính là người lập trình và cũng là người vận hành.

Ngôn ngữ làm việc của họ là ngôn ngữ máy là một chuỗi số nhị phân 01 vì vậy công việc của họ rất đơn điệu nhàm chán và việc mắc lỗi là không thể tránh khỏi do đó cần hiệu chỉnh chương trình. Vì thế hiệu suất làm việc thấp, trung bình khoảng 8 câu lệnh/ngày.

Tồn tại công việc cho mọi người, mọi chương trình: Mong muốn của người sử dụng luôn khác xa với khả năng đáp ứng của máy tính. Nó rất phong phú đa dạng, còn khả năng của máy tính trong một thời gian xác định là hầu như không đổi. Bên cạnh đó cũng có những khả năng của máy tính như tốc độ processor người dùng ít khi khai thác triệt để.

Người ta thấy một số công việc là cần thiết cho mọi người, thường xuyên được sử dụng do đó phải xây dựng (các chương trình chuẩn hay chương trình mẫu) Standard Programs. Việc này được thực hiện bởi các nhà lập trình và được cung cấp cùng với máy tính. Từ đó tạo thành một bộ thư viện chương trình mẫu.

{ Standard Programs } = Library of Standard Programs

Ban đầu các chương trình mang tính chất hoàn thiện: từ một đầu vào nhất định (input) sẽ đưa ra một kết quả(output). Tuy nhiên việc này là không thuận lợi và hiệu quả vì có nhiều chức năng con của chương trình sẽ lặp đi lặp lại nhiều lần. Vì vậy người ta nghĩ đến các phương pháp cải tiến hiệu suất làm việc là phải *xây dựng các chương trình con và cải tiến hệ lệnh của processor* tức là giảm bớt các lệnh macroprocessor mà thay vào đó là các phép xử lý tác động lên bit, byte.

Lý do: các hàm cơ sở như lấy căn, sinx, hay lũy thừa không phải ai cũng cần dùng, nhưng ai cũng cần xử lý thông tin, mà cơ sở nhỏ nhất của thông tin nằm trong bit, byte nên khi tăng cường khả năng cho processor trong lĩnh vực xử lý bit thì nó sẽ gần với nhu cầu của mọi người hơn.

- 1951-1960

{Library of Standard Programs} Để đáp ứng nhu cầu người sử dụng, một mặt người ta phải nghiên cứu các thuật toán lấy căn, sinx, cos, \square . trên cơ sở xử lý bit, byte; mặt khác phải xây dựng sẵn các modul đáp ứng yêu cầu này.

Đội ngũ người sử dụng ngày càng lớn => thoả mãn nhu cầu lập trình dưới dạng thư viện ngày càng lớn theo => yêu cầu số thư viện lớn với nội dung đồ sộ => khó. Giải quyết bằng cách phải cung cấp cho người sử dụng công cụ cho phép mô tả những giải thuật cần thiết =>

Đây là thời kỳ ra đời của ngôn ngữ thuật toán với một loạt các chương trình dịch. Các ngôn ngữ ra đời: Assembler, FORTRAN, COBOL.

Đòi hỏi với chúng là kỹ thuật bán dẫn, sản xuất được băng từ cho phép lưu trữ được một số chương trình. Đã có sự phân hoá chức năng giữa người lập trình và thao tác viên (operator).

- + Processor cũng chuyển giao một số chức năng cho các thiết bị khác.
- + Đã bắt đầu việc nhóm các chương trình hay tác vụ (jobs) lại để xử lý theo lô. Đã có những mầm mống của hệ điều hành:
 - o Thực hiện tự động các công việc
 - o Nạp và giải phóng chương trình trong bộ nhớ
 - o Quản lý vào ra: đọc bìa, băng từ, máy in.

=> Mô hình hoạt động của hệ điều hành : người sử dụng tác động trực tiếp lên MTDT hay thông qua thư viện chương trình mẫu hoặc tác động trung gian qua chương trình dịch =>

- + Hệ thống quản lý
 - o Quản lý thư viện
 - o Quản lý Chương trình dịch
 - o Quản lý cung cấp dịch vụ

- 1960-nay

Tiến thêm một bước trong việc cơ sở hoá thư viện chương trình mẫu. Thay vì cung cấp các chương trình hoàn thiện, cung cấp các module giải quyết từng phần giải thuật, phép biến đổi thường gặp => ngoài ra còn cần có những modul chuyên phục vụ điều khiển và tổ chức các module trong thư viện.

Ra đời phát triển mạch tích hợp (kích cỡ nhỏ, tốc độ cao), các đĩa từ với tốc độ truy nhập dữ liệu lớn hơn.

+ Ra đời các máy vi tính IBM PC và MSDOS.

- o Giao diện đồ họa
- o Chức năng đa phương tiện được nhúng vào hệ điều hành.
- o Hệ điều hành đã gắn với mạng và Internet.
- o Các hệ thống song song: nhiều bộ vi xử lý cùng chia sẻ một hệ thống bus, đồng hồ, thiết bị ngoại vi, bộ nhớ.
- o Các hệ thống phân tán: nhiều bộ vi xử lý nhưng dùng bộ nhớ, bus, đồng hồ riêng nhưng liên lạc trao đổi với nhau để thực hiện một nhiệm vụ nào đó.
- o Hệ thống thời gian thực: Các xử lý tính toán bị giới hạn về mặt thời gian thực hiện.

Người ta đưa ra các lớp tùy chọn (option) để đưa máy tính về các môi trường làm việc phù hợp.

b. Phần mềm và vai trò của các lớp chương trình trong hệ thống tính toán

Trong khi phân kỹ thuật thay đổi chậm chạp thì phần chương trình bao quanh MTDT phát triển với tốc độ chóng mặt. Ngày nay giá thành của phần mềm chiếm một tỉ trọng lớn trong giá trị toàn hệ thống.

– So sánh giá thành giữa phần cứng và phần mềm

Ngày 06/06/2001

I. HARDWARE

Server

P/N	Description	Unit Price
X Series 230	IBM Series x230 PIII 1000 Mhz/ 256 Kb Cache	3,475.00
865861Y	IBM Netfinity 128MB SDRAM ECC RDIMM Intergrated dual channel Ultra 160 LVD SCSI Internal storage: 218.4 GB 10/100 Ethernet intergrated 10 bays(6 hot plug, 2 half-high, 40x IDE CDROM) 250w Power Supply Cooling fans: 2	
33L3123	IBM Netfinity 128MB SDRAM ECC RDIMM	305.00
33L3125	IBM Netfinity 256MB SDRAM ECC RDIMM	641.00
37L7205	IBM Netfinity 18.2 GB 10K-4 Wide Ultra160 SCSI	598.00
	IBM E54 Color Monitor, stealth black	178.00
C5647A	HP SureStore Tape 40i GB Internal (40GB)	1,980.00
	TOTAL	7,177.00

II. SOFTWARE		
P/N	Description	Unit Price
00P7778	VISUALAGE JAVA ENTERPRISE EDITION V3.5 PROGRAM PKG	4,349.00
11K7694	WEBSPIHERE STUDIO ADVANCED V3.5 PROGRAM PACKAGE WITH 1 CLIENT ADV	2,899.00
	Oracle 8i Standard Edition Release 3 (with 5 Client)	1,120.00
659-00399	VStudio Pro 6.0 Win32 English Intl CD Refresh	1,107.76
021-02665	Office 2000 win 32 English Intl CD	477.25
021-03851	Office 2000 win 32 English OLP NL	385.65
227-01187	WinNT Svr 4.0 English Intl CD 5 Clt SP4	823.25
	TOTAL	11,161.91

Các chương trình bao quanh phần kỹ thuật tạo thành một môi trường tính toán. Mỗi chương trình muốn được thực hiện phải gắn với môi trường và thừa hưởng ở môi trường mọi khả năng của hệ thống. Làm cho thông tin lưu chuyển dễ dàng giữa các thành phần của hệ thống. Thông tin đầu ra của một module này có thể làm đầu vào cho một module khác. Mọi biến đổi trung gian đều do hệ thống đảm nhiệm và trong suốt với người sử dụng.

Ví dụ: các phần mềm Word, Excel gắn liền với môi trường Windows. Khi các thông số của môi trường thay đổi (Hệ thống font chữ, bàn phím, ngôn ngữ) => sẽ ảnh hưởng lên các phần mềm này.

Dù là một chương trình hay một thành phần của hệ thống thì đều phải hoạt động đồng bộ với toàn hệ thống (các thành phần hệ thống hay chương trình khác). Hệ thống có chức năng đảm bảo đầy đủ các điều kiện vật chất để chương trình chạy được như bộ nhớ, thời gian phục vụ của processor, thiết bị ngoại vi => Tóm lại hệ thống có nhiệm vụ quản lý tài nguyên.

2. Tài nguyên của hệ thống tính toán

- Các tài nguyên chủ yếu

Tài nguyên phân chia làm hai loại cơ bản: không gian và thời gian. Trong khung cảnh mỗi hệ thống thì đó là không gian nhớ và thời gian thực hiện lệnh.

a. Bộ nhớ

- Bộ nhớ là nơi lưu trữ thông tin.
- Đặc trưng bộ nhớ

- + Thời gian truy nhập trực tiếp: thời gian trực tiếp để truy nhập tới địa chỉ bất kỳ trong bộ nhớ.
- + Thời gian truy nhập tuần tự: Khi tồn tại một cách tổ chức lưu trữ kế tiếp.

Bộ nhớ thường được phân cấp theo tốc độ truy nhập trực tiếp hay kế tiếp. Bộ nhớ được gọi là thực hiện nếu processor có thể thực hiện câu lệnh bất kỳ ghi trong đó. Đặc điểm của bộ nhớ này là thời gian truy nhập thực hiện và truy nhập tuần tự là bằng nhau. Bộ nhớ trong bao giờ cũng là bộ nhớ thực hiện.

- + Không gian bộ nhớ
- + Giá thành
- Phân loại bộ nhớ
- + Bộ nhớ trong: Có tốc độ truy nhập cao nhưng không gian bộ nhớ nhỏ
- + Bộ nhớ ngoài: Có không gian bộ nhớ lớn nhưng tốc độ truy nhập thấp. Thời gian truy nhập trực tiếp thường lớn hơn thời gian truy tuần tự. Loại bộ nhớ phổ biến là bộ nhớ đĩa cứng, đĩa mềm, băng từ, đĩa quang.

b. Thời gian processor

Bản thân Processor là tài nguyên quan trọng. Tài nguyên thời gian ở đây là thời gian thực hiện câu lệnh chứ không phải thời gian của cuộc sống hàng ngày. Processor được dùng cho nhiều tiến trình khác nhau do đó việc phân chia thời gian sử dụng processor của mỗi tiến trình phải được tối ưu hoá, đặc biệt là khi chúng còn dùng chung tài nguyên khác: chương trình, dữ liệu, thiết bị vào ra... Nói cách khác, thời gian processor chính là một tài nguyên quan trọng của hệ thống.

c. Thiết bị ngoại vi

- Đa dạng
- Số lượng lớn $\gg 1$
- Tốc độ xử lý \ll tốc độ processor

Các thiết bị tiếp nhận, lưu trữ thông tin ở bộ nhớ ngoài trong thời gian dài được gọi là thiết bị ngoại vi. Máy in, bàn phím, màn hình, chuột, modem, □. Trước đây các thiết bị này thường được đặt xa phòng đặt máy chính nên gọi là thiết bị ngoại vi. Chúng còn có tên gọi khác là thiết bị vào ra. Chúng thường được gắn với MTDT thông qua các thiết bị trung gian: các thiết bị điều khiển.

Tài nguyên có hai loại: Phân chia được và không phân chia được.

Phân chia được: Cho phép nhiều người hay chương trình sử dụng nó một cách đồng thời. Điển hình là bộ nhớ (trong và ngoài): có thể nạp nhiều chương trình vào bộ nhớ trong, hay 1 chương trình sử dụng nhiều tệp trên đĩa cứng.

Không phân chia được: phần lớn các tài nguyên còn lại. Tuy nhiên có thể phân phối việc sử dụng chúng sao cho người sử dụng cảm giác như được phục vụ đồng thời.

3. Định nghĩa HĐH

Hệ điều hành là một phần quan trọng của mọi hệ thống thông tin. Một hệ thống thông tin gồm 4 thành phần: phần cứng, hệ điều hành, chương trình ứng dụng, người sử dụng.

Phần cứng: CPU, bộ nhớ, thiết bị vào ra cung cấp các tài nguyên thông tin cơ sở.

Các chương trình ứng dụng: chương trình dịch, hệ thống cơ sở dữ liệu, trình soạn thảo văn bản. qui định cách sử dụng các tài nguyên đó để giải quyết những vấn đề của người sử dụng.

Hệ điều hành điều khiển và đồng bộ việc sử dụng phần cứng của các chương trình ứng dụng phục vụ các người sử dụng khác nhau với các mục đích sử dụng phong phú đa dạng.

Ta có thể hiểu HĐH là HỆ THỐNG các chương trình đảm bảo các chức năng **giao tiếp người máy** và **quản lý tài nguyên hệ thống tính toán**.

– Tuy nhiên có nhiều người quan sát HĐH dưới các góc độ khác nhau vì thế tồn tại nhiều định nghĩa về HĐH.

Đối với người sử dụng: HĐH là tập hợp các chương trình, phục vụ khai thác hệ thống tính toán một cách dễ dàng, thuận tiện.

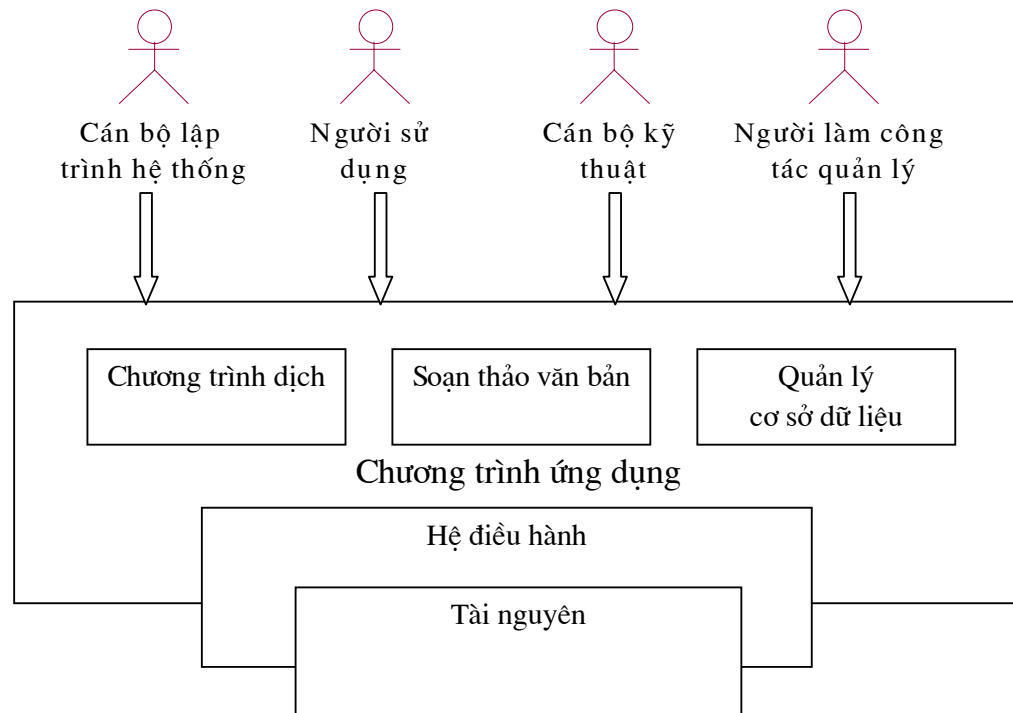
Người sử dụng khi thực hiện một chương trình nào đó trên MTĐT thì chỉ quan tâm đến việc hệ **thống** có đáp ứng được nhu cầu của họ hay không? Có chương trình cần thực hiện, có đủ bộ nhớ để chạy. Họ không quan tâm đến việc hệ điều hành làm gì nhằm mục đích gì, có cấu trúc như thế nào?

Đối với người làm công tác quản lý: HĐH là một tập các chương trình phục vụ quản lý chặt chẽ và sử dụng tối ưu các tài nguyên của hệ thống tính toán.

Đối với cán bộ kỹ thuật: HĐH là hệ thống chương trình bao trùm lên một máy tính vật lý cụ thể để tạo ra một máy logic với những tài nguyên mới và khả năng mới.

Các định **nghĩa** trên phản ánh vị trí quan sát của người nêu. Họ đứng ở ngoài hệ thống và thể hiện điều họ mong đợi và cũng là điều họ nhìn thấy.

Đối với cán bộ lập trình hệ thống: HĐH là hệ thống mô hình hoá, mô phỏng các hoạt động của máy, của người sử dụng và của thao tác viên hoạt động trong các hệ thống đối thoại nhằm tạo môi trường để quản lý chặt chẽ các tài nguyên và tổ chức khai thác chúng một cách thuận tiện và tối ưu.



Đối với các cán bộ lập trình hệ thống, vị trí của họ là ở bên trong hệ điều hành. Họ quan sát các module, các thành phần của hệ thống, quan sát mối quan hệ giữa chúng. Đây là quan điểm của chúng ta trong suốt quá trình khảo sát nghiên cứu hệ điều hành.

- Như vậy HĐH là hệ chuyên gia ra đời sớm nhất và thuộc loại hoàn thiện nhất.

4. Phân loại hệ điều hành

a. Hệ điều hành đơn nhiệm và hệ điều hành đa nhiệm

Dựa vào cách thức đưa chương trình vào bộ nhớ, chọn chương trình có sẵn trong bộ nhớ để processor thực hiện, người ta phân thành: hệ điều hành đơn nhiệm, đa nhiệm.

- Hệ điều hành đơn nhiệm

Tại một thời điểm xác định, khi một chương trình được đưa vào bộ nhớ thì nó chiếm giữ mọi tài nguyên của hệ thống, và vì vậy chương trình khác không thể được đưa vào bộ nhớ trong khi nó chưa kết thúc.

Nhưng do các thiết bị vào ra thường làm việc với tốc độ chậm, người ta dùng kỹ thuật SPOOLING(simultaneous peripheral Operation on line): cho phép tạo ra hiệu ứng song song các thiết bị chỉ cho phép vào ra tuần tự(sẽ đề cập chi tiết ở chương sau).

- Hệ điều hành đa nhiệm

Hệ điều hành cho phép tại một thời điểm có nhiều chương trình ở trong bộ nhớ trong. Chúng có nhu cầu được phân phối thời gian phục vụ CPU, bộ nhớ và thiết bị ngoại vi. Như vậy CPU, bộ nhớ, thiết bị ngoại vi v.v.. là các tài nguyên được chia sẻ cho các chương trình đó. Vấn đề là làm sao đảm bảo tốt nhất tính bình đẳng khi giải quyết vấn đề phân phối tài nguyên.

b. Hệ điều hành đơn chương và hệ điều hành đa chương (MultiUsers)

- Hệ điều hành đơn chương

Tại một thời điểm xác định hệ điều hành chỉ cho phép một người sử dụng thao tác mà thôi.

- Hệ điều hành đa chương

Hệ điều hành cho phép tại một thời điểm có thể phục vụ nhiều người sử dụng.

c. Hệ điều hành tập trung và hệ điều hành phân tán

- Hệ điều hành tập trung

Trên một hệ thống máy tính chỉ có một HĐH duy nhất cài ở máy chủ. Các máy trạm được khởi động nhờ máy chủ và nó chỉ làm chức năng nhập/xuất dữ liệu. Mọi xử lý đều tập trung ở máy chủ.

- Hệ điều hành phân tán

Trên mỗi máy có 1 hệ điều hành khác nhau, máy chủ chịu trách nhiệm cung ứng các dịch vụ để truy nhập đến các tài nguyên chung và điều hành toàn hệ thống, các phép xử lý có thể tiến hành ở máy trạm.

d. Hệ điều hành phân chia thời gian và hệ điều hành thời gian thực

- Hệ điều hành phân chia thời gian (Share time)

Một CPU luôn phiên phục vụ các tiến trình và 1 tiến trình có thể rơi vào trạng thái chờ đợi khi chưa được phân phối CPU.

- Hệ điều hành thời gian thực (Real time)

Một tiến trình khi đã xâm nhập vào hệ thống thì ở bất kỳ lúc nào đều được phân phối CPU.

5. Các tính chất cơ bản của hệ điều hành

a. Tin cậy

- Mọi hoạt động của HĐH đều phải chuẩn xác tuyệt đối.
- Thông tin của HĐH đưa ra phải chính xác và phải ngăn ngừa các sai sót ngẫu nhiên, hạn chế các sai sót cố ý.
- Ví dụ

A:\> copy A:\f1.txt C:

- + Kiểm tra xem có tồn tại các đĩa không (control card)

- + Kiểm tra xem có tồn tại ổ đĩa A:
- + Kiểm tra xem có tồn tại đĩa A
- + Kiểm tra khả năng truy nhập đĩa từ
- + Kiểm tra có tồn tại tệp f1.txt
- + Kiểm tra có đọc được tệp hay không
- + Lặp lại với C:
- HĐH phải có những phương tiện kiểm tra tính đúng đắn của dữ liệu trong khi thao tác.

b. An toàn

- Hệ thống cố gắng bảo vệ thông tin, cố gắng chống các trường hợp truy nhập không hợp thức.
- Chức năng bảo vệ thông tin được chia thành nhiều mức:
 - + Các mức do hệ thống đảm nhiệm: Ví dụ trong các hệ thống UNIX, khi muốn xoá hay sửa đổi nội dung một tệp, người sử dụng phải có quyền xoá sửa đổi với file đó.
 - + Có mức do người sử dụng đảm nhiệm: Lệnh DEL *.* của MSDOS, hệ thống hỏi lại người sử dụng một lần nữa để tránh sai sót vô ý.

c. Khái quát theo thời gian

- HĐH phải có tính kế thừa từ các hệ thống cũ
- HĐH cũng phải có khả năng thích nghi với những thay đổi trong tương lai.

d. Hiệu quả

- Các tài nguyên của hệ thống phải được khai thác tối ưu.
- HĐH phải duy trì đồng độ trong toàn bộ hệ thống.

e. Thuận tiện

- HĐH phải thân thiện với người sử dụng do đó HĐH phải có nhiều hình thái giao tiếp:
 - + Giao tiếp dạng dòng lệnh
 - + Giao tiếp dạng thực đơn (Menu)
 - + Giao tiếp dạng biểu tượng

6. Nguyên lý xây dựng chương trình HĐH

a. Module

- HĐH phải được xây dựng từ các module độc lập nhưng có khả năng liên kết thành một hệ thống có thể thu gọn hoặc mở rộng tùy ý.
- Các module đồng cấp quan hệ với nhau thông qua dữ liệu vào và ra.

- Tồn tại quan hệ phân cấp khi các liên kết các module tạo thành những module có khả năng giải quyết các vấn đề phức tạp hơn.

b. Phủ chức năng

- Một công việc có thể thực hiện bằng nhiều cách khác nhau.
- Ví dụ

Muốn in tệp f1.txt

- + C:\> copy f1.txt prn
- + C:\> type f1.txt >prn
- + C:\> print f1.txt

c. Marco-processor

- Khi có một công việc cụ thể, hệ thống sẽ xây dựng các yêu cầu, liệt kê các bước phải thực hiện từ đó xây dựng chương trình tương ứng, sau đó thực hiện chương trình nói trên.
- Ví dụ: Trong MSDOS ta có các tệp config.sys và autoexec.bat

d. Nguyên lý bảng tham số điều khiển

- Hệ thống không tham chiếu trực tiếp đến thiết bị, đối tượng vật lý mà chỉ làm việc với bảng tham số xác định đặc trưng của thiết bị đó.
- Bảng tham số được hệ thống xây dựng ngay trong quá trình làm việc
- Ví dụ

Bảng tham số của một máy tính PC được lưu trong CMOS 64byte

- Lợi ích của việc sử dụng bảng tham số
 - + Truy nhập thực hiện công việc nhanh với CPU
 - + Không phụ thuộc vào các thiết bị vật lý cụ thể
- Ví dụ:

Bên cạnh bảng tham số được lưu trong CMOS còn có các bảng tham số trong tệp config.sys và autoexec.bat cho phép ta thay đổi giá trị các biến môi trường của MSDOS.

Files = Số_tệp_mở_tối_đa

e. Nguyên lý giá trị chuẩn

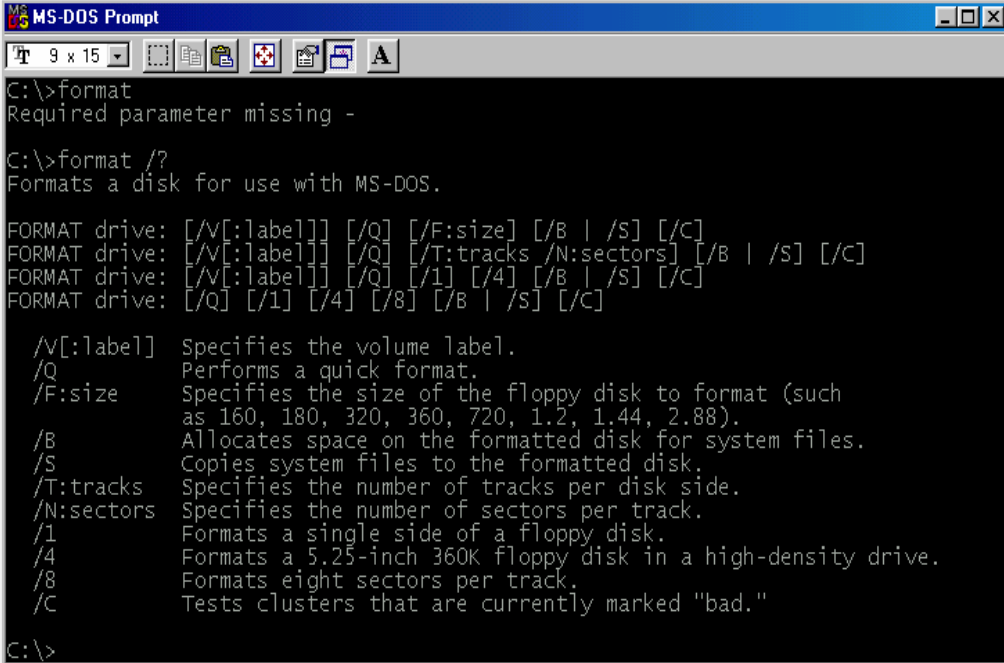
- HĐH chuẩn bị bảng giá trị của các tham số gọi là bảng giá trị chuẩn. Trong trường hợp một module hoặc một câu lệnh có nhiều tham số và người sử dụng không nêu hết các giá trị tham số thì HĐH phải lấy giá trị trong bảng giá trị chuẩn bổ xung vào các tham số thiếu.
- Ví dụ:

C:\>TP70>Dir

- + Đĩa nào? Thường trực: Đĩa C:
- + Thư mục nào? Thường trực: C:\Tp70
- + Xem gì? Xem tất cả các tệp
- + Đưa ra đâu? Đưa ra thiết bị ra chuẩn: Màn hình
- + Đưa ra như thế nào? Đầy đủ thông tin
- Tham số
 - + Tham số vị trí: Xuất hiện theo vị trí và theo dòng tham số.
 - + Tham số khoá được xây dựng theo từ khoá và có thể xuất hiện ở vị trí bất kỳ, trình tự bất kỳ.
- Ví dụ

C:\>format A: /q

- + Lệnh format đĩa
- + Tham số vị trí: Đĩa A
- + Tham số khoá: Format nhanh



```

MS-DOS Prompt
C:\>format
Required parameter missing -

C:\>format /?
Formats a disk for use with MS-DOS.

FORMAT drive: [/V[:label]] [/Q] [/F:size] [/B | /S] [/C]
FORMAT drive: [/V[:label]] [/Q] [/T:tracks /N:sectors] [/B | /S] [/C]
FORMAT drive: [/V[:label]] [/Q] [/1] [/4] [/B | /S] [/C]
FORMAT drive: [/Q] [/1] [/4] [/8] [/B | /S] [/C]

/V[:label] Specifies the volume label.
/Q         Performs a quick format.
/F:size    Specifies the size of the floppy disk to format (such
as 160, 180, 320, 360, 720, 1.2, 1.44, 2.88).
/B         Allocates space on the formatted disk for system files.
/S         Copies system files to the formatted disk.
/T:tracks  Specifies the number of tracks per disk side.
/N:sectors Specifies the number of sectors per track.
/1         Formats a single side of a floppy disk.
/4         Formats a 5.25-inch 360K floppy disk in a high-density drive.
/8         Formats eight sectors per track.
/C         Tests clusters that are currently marked "bad."

C:\>
    
```

f. Nguyên lý bảo vệ nhiều mức

- Chương trình và dữ liệu phải được bảo vệ nhiều mức bằng nhiều khoá.
- Ví dụ trong Linux
 - + Mức 1: Người sử dụng phải có tài khoản mới được sử dụng máy tính.
 - + Mức 2: Chỉ những người sử dụng thuộc nhóm A mới được truy nhập và tệp chung của nhóm A.

7. Thành phần của HĐH và kiến trúc HĐH

a. Thành phần của HĐH

– Ngôn ngữ làm việc và giao tiếp

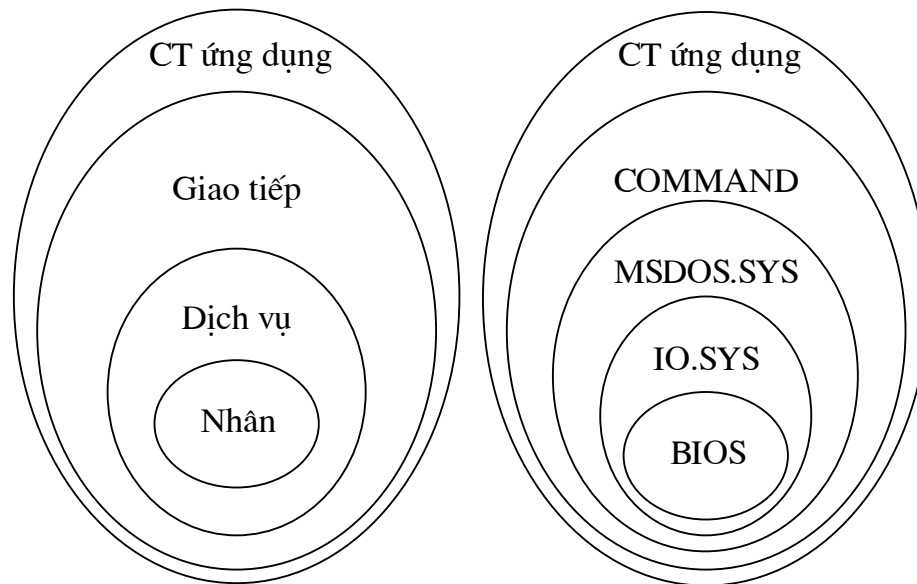
Hệ điều hành phải có ít nhất 3 ngôn ngữ làm việc và giao tiếp phục vụ cho các đối tượng khác nhau.

- + Ngôn ngữ máy: Là ngôn ngữ thực hiện duy nhất của hệ thống vì vậy còn gọi là ngôn ngữ thực hiện. Mọi ngôn ngữ khác đều phải được ánh xạ sang ngôn ngữ này.
- + Ngôn ngữ vận hành: Là ngôn ngữ giúp thao tác viên giao tiếp với hệ thống. Ví dụ: ngôn ngữ MSDOS.
- + Ngôn ngữ thuật toán: Là ngôn ngữ chủ yếu giúp người sử dụng giao tiếp với hệ thống. Ví dụ ngôn ngữ Pascal, C, Visual Basic. Mỗi ngôn ngữ có thể có nhiều chương trình dịch để ánh xạ sang ngôn ngữ máy.

– Hệ thống quản lý tài nguyên: Supervisor

Đây là hệ thống phục vụ phân phối và quản lý tài nguyên.

b. Kiến trúc HĐH



Nhân là phần chính của HĐH làm các nhiệm vụ như quản lý bộ nhớ, quản lý tiến trình, phân chia tài nguyên. Nhân chỉ đảm nhiệm các chức năng cơ bản nhất, có kích thước nhỏ để giảm đến mức tối thiểu lỗi.

Dịch vụ là phần mở rộng các chức năng của HĐH cho phép khai thác tài nguyên hệ thống và hỗ trợ người dùng như quản lý tệp, quản lý thư mục, thư điện tử, truyền tệp.

Giao tiếp là phân tạo ra môi trường giao tiếp giữa người sử dụng và máy tính.

8. Các hình thái giao tiếp

a. Hình thái dòng lệnh

Người sử dụng giao tiếp với HĐH qua các dòng lệnh, mỗi lệnh có các tham số tương ứng.

- Ưu điểm
 - + Dễ xây dựng và giảm công sức cho người xây dựng hệ thống.
 - + Người sử dụng có thể đưa tham số của lệnh một cách chính xác theo mong muốn.
- Nhược điểm
 - + Tốc độ đưa lệnh vào chậm, người sử dụng phải nhớ các tham số.
 - + Đối với các thao tác viên không có kinh nghiệm, thì hình thái giao tiếp này gây cản trở đến hiệu quả làm việc.
 - + Hình thái giao tiếp này bị cản trở bởi hàng rào ngôn ngữ.

b. Hình thái thực đơn

Người sử dụng giao tiếp với HĐH thông qua các thực đơn, các thực đơn thường có dạng trải xuống (Popup). Mỗi một thực đơn con tương ứng với một chức năng. Các tham số có thể được đưa vào thông qua giao tiếp với người sử dụng.

- Ưu điểm
 - + Hình thái này không yêu cầu nhớ lệnh
 - + Người sử dụng có thể truy nhập vào thực đơn qua bàn phím hoặc qua chuột.
- Nhược điểm
 - + Hình thái giao tiếp này bị cản trở bởi hàng rào ngôn ngữ.
 - + Đôi khi các từ trên thực đơn không nêu bật được chức năng của nó

c. Hình thái của sổ □ biểu tượng

Người sử dụng giao tiếp với HĐH thông qua các thanh công cụ và các biểu tượng. Mỗi một biểu tượng tương ứng với một chức năng. Các tham số có thể được đưa vào thông qua giao tiếp với người sử dụng.

- Ưu điểm
 - + Hình thái này không yêu cầu nhớ lệnh
 - + Người sử dụng không bị hàng rào ngôn ngữ gây cản trở.
- Nhược điểm

- + Có thể có rất nhiều biểu tượng do đó gây sự nhập nhằng về chức năng.
- + Không thuận lợi khi thao tác bằng bàn phím

d. Hình thái kết hợp

Hệ điều hành thường kết hợp nhiều hình thái giao tiếp để tạo ra tính thân thiện với người sử dụng. Ví dụ: việc kết hợp thực đơn với các biểu tượng, hoặc kết hợp giữa các biểu tượng và các từ gợi ý (tooltip).

Hình thái giao tiếp kết hợp này khắc phục được các nhược điểm của các hình thái giao tiếp đơn lẻ.

9. Giới thiệu về MSDOS

a. Lịch sử của DOS

- Những năm 1980, hãng Intel cho ra đời bộ vi xử lý 16 bit 8086. Jim Paterson đã bỏ công sức xây dựng hệ điều hành mới cho loại máy tính sử dụng bộ vi xử lý này đó là 86-DOS. HĐH này đã cố gắng khắc phục những điểm yếu của hệ điều hành trước đó là CP/M.
 - Microsoft đã mua lại HĐH của Jim Paterson và phát triển thành hệ điều hành PCDOS hay MSDOS. Phiên bản đầu tiên của MSDOS thế hệ 1.0 ra đời vào 8/1981.
 - Các cải tiến của MSDOS 1.0 so với CP/M
 - + Có thêm loại chương trình chạy EXE bên cạnh các chương trình COM.
 - + HĐH tách bộ xử lý lệnh thành một phần nội trú và một phần ngoại trú.
 - + Để tiện lợi cho việc quản lý đĩa người ta đưa ra bảng File Allocation Table viết tắt là FAT để quản lý đĩa. Mỗi phần tử của bảng FAT tương ứng với 521 byte trên đĩa gọi là sector, chỉ ra sector này đã có dữ liệu hay còn tự do.
 - + MSDOS 1.0 cho phép xử lý lô (batch) một số lệnh của MSDOS bằng cách tạo một tệp batch.
 - + Ngày tháng tạo hay cập nhật tệp cũng được lưu trữ cùng với thông tin của tệp.
- Phiên bản MSDOS 2.0 ra đời vào năm 1983
 Phiên bản MSDOS 3.0 ra đời vào năm 1984
 Phiên bản MSDOS 4.0 ra đời vào năm 1988

b. Các thành phần của MSDOS

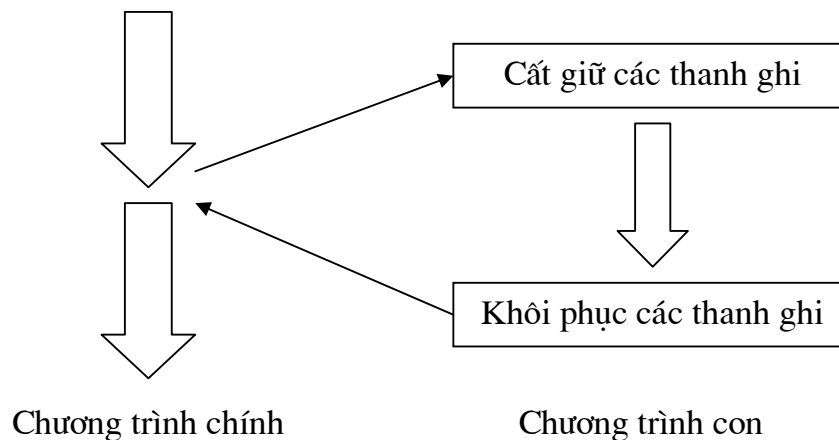
- BIOS: Chứa các chương trình của supervisor và quản lý tệp nhưng chưa kết nối thành hệ thống. Do đó cần chương trình kích hoạt.
- Chương trình môi Boot Strap Loader: nằm ở sector đầu tiên của đĩa từ dùng để kích hoạt toàn bộ chương trình hệ thống.

- IO.SYS: Dưới sự hỗ trợ của BSL bao lấy BIOS, cung cấp các dịch vụ cơ bản nhất như chia sẻ tài nguyên, quản lý bộ nhớ.
- MSDOS.SYS: mở rộng IO.SYS lần nữa
- COMMAND.COM: liên lạc giữa người sử dụng và hệ thống, chứa các lệnh nội trú.
- Các lệnh ngoài: là thành phần mở rộng theo từng lĩnh vực.

Chương 2. Hệ thống xử lý ngắt trong IBM PC

1. Khái niệm về ngắt và xử lý ngắt trong IBM PC

- Ngắt là hiện tượng tạm ngừng thực hiện một tiến trình để chuyển sang thực hiện một tiến trình khác khi có một sự kiện xảy ra trong hệ thống tính toán. Có thể hiểu tạm nghĩa “thực hiện một tiến trình” là thực hiện một chương trình, tiến trình bị ngắt có thể coi là chương trình chính, còn tiến trình xử lý ngắt có thể coi là chương trình con.
- Chương trình con xử lý ngắt là một chương trình ngôn ngữ máy hoàn toàn bình thường. Chương trình này địa chỉ kết thúc bằng lệnh IRET (Interupt RETurn), nó ra lệnh cho bộ xử lý quay về thực hiện tiếp chương trình chính đúng từ chỗ mà nó bị ngắt.



- Đối với các hệ thống tính toán việc gọi ngắt dùng cho việc các bộ phận khác nhau của hệ thống tính toán báo cho processor biết về kết quả thực hiện công việc của mình.

2. Phân loại ngắt

Có nhiều tiêu chí để phân loại ngắt

- Phân loại theo ngắt trong và ngắt ngoài
 - + Ngắt trong là ngắt do các tín hiệu của processor báo cho processor
 - + Ngắt ngoài là ngắt do các tín hiệu bên ngoài báo cho processor
- Phân loại theo sự sử dụng
 - + Ngắt dành cho HĐH sử dụng. Nếu thay đổi xử lý ngắt này sẽ làm thay đổi chức năng của hệ thống.
 - + Ngắt dành cho người sử dụng

- + Ngắt dự trữ, hoặc là để cho HĐH sử dụng sau này hoặc để người sử dụng dùng cho mục đích riêng.
- Phân loại ngắt cứng và ngắt mềm, đây là cách phân loại phổ biến nhất

a. Ngắt mềm

- Là ngắt được gọi bằng một lệnh ở trong chương trình. Lệnh gọi ngắt từ chương trình ngôn ngữ máy là lệnh INT (INTerrupt), các lệnh gọi ngắt từ chương trình ngôn ngữ bậc cao sẽ được dịch thành lệnh INT.

b. Ngắt cứng

- Là ngắt được gọi bởi các chương trình được cứng hoá trong các mạch điện tử.
- Ngắt cứng được chia làm hai loại:

- + Ngắt cứng che được (Maskable Interrupt)

Là ngắt có thể dùng mặt nạ để ngăn cho không ngắt hoạt động. Ta có thể đặt các bit trong mặt nạ bằng lệnh CLI (CLear Interrupt flag).

Ví dụ: Ngắt chuột là ngắt cứng có thể bị che.

- + Ngắt cứng không che được (Non Maskable Interrupt)

Là ngắt không thể dùng mặt nạ che được.

Ví dụ: Ngắt 2 báo hiệu có lỗi trong bộ nhớ.

Ngắt cứng không che được có độ ưu tiên cao nhất và được CPU phục vụ trước tất cả các ngắt khác.

3. Quy trình xử lý ngắt

a. Quy trình xử lý ngắt

Quy trình xử lý ngắt được chia thành 5 bước

- Bước 1:
Lưu đặc trưng sự kiện gây ngắt vào nơi quy định
- Bước 2:
Lưu trạng thái của tiến trình bị ngắt vào nơi quy định
- Bước 3:
Chuyển điều khiển tới chương trình xử lý ngắt
- Bước 4:
Thực hiện chương trình xử lý ngắt, tức là xử lý sự kiện
- Bước 5:
Khôi phục tiến trình bị ngắt

b. Chương trình con và chương trình xử lý ngắt

- Giống nhau

Khi thực hiện xong công việc, hai chương trình đều trở về chương trình ở mức trên nó.

Ba bước thực hiện đầu tiên đều giống nhau

– Khác nhau

Chương trình con	Chương trình xử lý ngắt
Khi chúng ta có lời gọi chương trình con, chúng ta cần biết đích xác chương trình con nằm tại đâu. Chương trình chính và chương trình con được gắn kết với nhau thành một tiến trình duy nhất.	Chương trình bị ngắt và chương trình xử lý ngắt là 2 tiến trình độc lập, 2 tiến trình này không biết thông tin của nhau. Trong chương trình bị ngắt cũng không có lời gọi đến chương trình xử lý ngắt.
Trong chương trình chính ta phải khai báo thư viện các chương trình con.	

4. Bảng vector ngắt

Khi ngắt được tạo ra, nơi phát sinh nó không cần biết địa chỉ của chương trình xử lý ngắt tương ứng mà chỉ cần biết số hiệu ngắt. Số hiệu này chỉ đến một phần tử trong một bảng gọi là bảng các vector ngắt nằm ở vùng có địa chỉ thấp nhất trong bộ nhớ và chứa địa chỉ của chương trình con xử lý ngắt. Địa chỉ bắt đầu của mỗi chương trình con được xác định bởi địa chỉ đoạn và địa chỉ offset được đặt trước đoạn.

Hai địa chỉ này đều là 16 bit (2 byte), như vậy mỗi địa chỉ ngắt chiếm 4 byte trong bộ nhớ. Máy tính PC có 256 ngắt khác nhau được đánh số từ 0 đến 255 do vậy độ dài của cả bảng do vậy sẽ là $256 \times 4 = 1024$. Bảng vector ngắt chiếm các ô nhớ từ địa chỉ 0 đến 3FFh. Số thứ tự của ngắt bằng số thứ tự của vector ngắt. Địa chỉ của chương trình xử lý số i được chứa trong bảng vector ngắt từ địa chỉ offset $4 \times (i-1)$ đến $4 \times (i-1) + 3$.

Một số ngắt thường dùng

STT	Số hiệu ngắt	Chức năng
1	00	Ngắt chia cho 0
2	04	Ngắt tràn số
3	08	Ngắt thời gian
4	09	Ngắt bàn phím

5	10H	Ngắt phục vụ màn hình
6	19H	Ngắt khởi động hệ thống
7	20H	Kết thúc chương trình
8	21H	Gọi các hàm của DOS
9	25H/26H	Đọc/ghi đĩa
10	27H	Kết thúc nhưng thường trú
11	33H	Ngắt phục vụ chuột
12	67H	Quản lý bộ nhớ mở rộng

5. Gọi ngắt trong Assembler

- Ví dụ: Gọi ngắt 10 h ẩn con trỏ

```
Mov AH,1
Mov CX,0100H
INT 10H
```

- Giải thích

Cho CH = 1, dòng đầu tiên của ma trận hiển thị con trỏ

Cho CL = 0, dòng cuối cùng của ma trận hiển thị con trỏ

Như vậy $CH < CL$ vì vậy con trỏ không hiện ra màn hình

- Nhúng ngôn ngữ Assembler vào trong môi trường PASCAL

```
uses crt;
begin
    writeln('...');
    asm
        mov AH,1
        mov CX,$0100
    end;
    readkey;
end.
```

6. Gọi ngắt trong Pascal

Pascal cung cấp hai thủ tục để gọi ngắt trong thư viện Dos

- Thủ tục Intr

```
procedure Intr(IntNo: Byte; var Regs: Registers);
```

để gọi ngắt với một số hiệu ngắt bất kỳ

- Thủ tục MsDos

```
procedure MsDos(var Regs: Registers);
```

để gọi ngắt 21H của DOS.

- Kiểu bản ghi Registers

```
type
```

```
Registers = record
  case Integer of
    0: (AX, BX, CX, DX, BP, SI, DI, DS, ES, Flags: Word);
    1: (AL, AH, BL, BH, CL, CH, DL, DH: Byte);
  end;
```

Chú ý:

- + Đặt các thông số cho các thanh ghi trước khi gọi thủ tục.
- + Lấy ra các giá trị trả về qua các thanh ghi sau khi gọi thủ tục.
- Ví dụ:

Dùng hàm 2AH để xem ngày

```
uses Dos;
var
  date, year, month, day: string;
  regs: Registers;
begin
  regs.ah := $2a; { Hàm $2A để lấy ngày }
  msdos(regs);
  with regs do
    begin
      str(cx, year); { Chuyển giá trị từ số sang xâu }
      str(dh, month);
      str(dl, day);
    end;
  date := day + '/' + month + '/' + year;
  writeln(' Hom nay la ', date);
end.
```

Kết quả

Hom nay la 11/8/2001

7. Bộ thanh ghi của 8088

Bộ vi xử lý 8088 có 14 thanh ghi có độ dài 16 bit và được chia thành 4 loại

a. Các thanh ghi chung

- Thường dùng để lưu trữ trong các phép toán số học.

Gồm có

- + AX (Accumulator): Thanh ghi tích lũy

Thường dùng để lưu trữ các giá trị trong khi thực hiện các phép toán số học.

- + BX (Base): Thanh ghi cơ sở

Thường dùng để trở đến đầu của một bảng dịch chuyển trong bộ nhớ. Nó cũng được dùng để cất giữ phần địa chỉ offset của một địa chỉ theo đoạn.

- + CX (Count): Thanh ghi đếm

Thường dùng như một bộ đếm để điều khiển một vòng lặp hoặc để chuyển lặp dữ liệu.

- + DX (Data): Thanh ghi dữ liệu

Thường dùng để cất giữ các giá trị 16 bit cho các mục đích chung.

Các thanh ghi chung được chia nhỏ thành hai thanh ghi 8 bit là thanh ghi thấp (Low) chứa các bit từ 0 đến 7 và thanh ghi cao (High) chứa các bit từ 8 đến 15. Như vậy ta có 8 thanh ghi nhỏ là AH,AL, BH,BL, CH,CL, DH, DL.

b. Các thanh ghi đoạn

- CS (Code Segment): Thanh ghi đoạn lệnh

Dùng để xác định đoạn lệnh, nơi chứa chương trình đang được thực hiện.

- DS (Data Segment): Thanh ghi đoạn dữ liệu

Dùng để xác định đoạn dữ liệu, nơi chứa dữ liệu của chương trình đang được thực hiện.

- SS (Stack Segmen): Thanh ghi đoạn ngăn xếp

Dùng để xác định đoạn ngăn xếp, là vùng làm việc tạm thời dùng để theo dõi các tham số và các địa chỉ đang được chương trình sử dụng.

- ES (Extra Segment): Thanh ghi đoạn ngoài

Khi vùng nhớ cần sử dụng vượt quá 64K, bộ vi xử lý dùng thanh ES để trở đến một đoạn thêm. Ngoài ra thanh ES còn được sử dụng cho việc chuyển dữ liệu giữa các đoạn.

c. Thanh ghi cờ

Là thanh ghi CF được dùng để lưu trữ các cờ

15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
				OF	DF	IF	TF	SF	ZF		AF		PF		CF

Các cờ được chia làm hai loại

- Các cờ trạng thái

- + CF (Carry Flag): Cờ nhớ chỉ phép toán số học có nhớ
- + OF (Overflow Flag): Cờ tràn chỉ phép toán số học bị tràn
- + ZF (Zero Flag): Cờ zero chỉ kết quả bằng không hoặc so sánh bằng
- + SF (Sign Flag): Cờ dấu chỉ kết quả âm không hoặc so sánh âm
- + PF (Parity Flag): Cờ chẵn lẻ chỉ số chẵn các bit 1

- + AF (Auxiliary Flag): Cờ nhớ phụ dùng trong các phép toán trên các số BCD (Binary-coded Decimal)
- Các cờ điều khiển
 - + DF (Direction Flag): Cờ định hướng điều khiển hướng tiến/lùi trong các phép lặp
 - + IF (Interrupt Flag): Cờ ngắt điều khiển cho phép ngắt
 - + TF (Trap Flag): Cờ bẫy điều khiển việc thực hiện từng lệnh

d. Bộ đếm chương trình

- Là thanh ghi IP được dùng để chứa địa chỉ offset trong đoạn lệnh, nơi mà chương trình đang thực hiện, vì vậy còn được gọi là con trỏ lệnh.

8. Thay đổi ngắt trong hệ thống

a. Các bước khi muốn thay đổi ngắt

- Viết chương trình con xử lý ngắt


```
{Tạo chương trình con xử lý ngắt với địa chỉ gọi 4 byte}
{$F+}
procedure thay_doi_break; interrupt;
begin
    {thực hiện các xử lý ngắt}
end;
{$F-}
```
- Lưu trữ vector ngắt cũ

Sử dụng thủ tục

```
procedure GetIntVec(IntNo: Byte; var Vector: Pointer);
```

Vector thuộc kiểu Pointer dùng để chứa địa chỉ chương trình xử lý ngắt.
- Thay thế vector ngắt mới

Sử dụng thủ tục

```
procedure SetIntVec(IntNo: Byte; Vector: Pointer);
```
- Thực hiện các chức năng khác
- Khôi phục vector ngắt cũ

b. Ví dụ viết chương trình đăng nhập vào máy

```
program Mat_Khau;
uses crt,dos;
var
    p: pointer; { chứa địa chỉ ngắt cũ }
    break_flag: boolean;
    i,j: byte;
    password: string[16];

    procedure pw;
```

```

var i: byte;
begin
password:= '';
{ Password có độ dài bằng 3 }
for i:= 1 to 3 do password:=password + readkey;
end;

{Tạo chương trình con xử lý ngắt với địa chỉ gọi 4 byte}
{$F+}
procedure thay_doi_break; interrupt;
begin
break_flag:=true;
end;
{$F-}

begin
clrscr;
{ lưu ngắt Ctrl + Break vào vùng nhớ được trỏ bởi p }
getintvec($1B,p);
break_flag:=false;

{ thay đổi ngắt Ctrl + Break }
setintvec($1B,addr(thay_doi_break));
i:=0;
write('Cho biet mat khau:');
while (i<3) and (not break_flag) do begin
    pw;
    if(password = 'ABC') then begin
        setintvec($1B,p);
        exit;
    end
    else begin
        i:=i+1;
        write('#7#7#7);
    end;
end;

inline($EA/$00/$00/$FF/$FF);
{ Lệnh JMP FFFF:0000 }
end.

```

9. Một số hàm và thủ tục thường dùng trong lập trình hệ thống

- Các toán tử Logic

```

not
and
or

```



```
xor
shl
shr
```

- Ví dụ:

```
{
not      | Bitwise negation | integer type | integer type
and      | Bitwise and         | integer type | integer type
or       | Bitwise or          | integer type | integer type
xor      | Bitwise xor             | integer type | integer type
shl      | Shift left              | integer type | integer type
shr      | Shift right             | integer type | integer type
}
```

```
procedure WriteBiWord(w: Word);
var i: byte;
begin
  for i := 15 downto 0 do begin
    if((w shr i) mod 2 = 0) then
      Write('0')
    else
      Write('1');
    if(i mod 4 = 0) then Write(' ');
  end;
end;
```

```
procedure WriteHexWord(w: Word);
const
  hexChars: array [0..$F] of Char =
    '0123456789ABCDEF';
begin
  Write(hexChars[Hi(w) shr 4],
        hexChars[Hi(w) and $F],
        hexChars[Lo(w) shr 4],
        hexChars[Lo(w) and $F]);
end;
```

```
procedure WriteWord(w: Word);
begin
  Writeln;
  Write('w = ', w);
  Write(' = '); WriteHexWord(w);
  Write(' = '); WriteBiWord(w);
end;
```

```
procedure TestShift;
var
```

```

    A: Word;
begin
    writeln;
    writeln('-----');
    A := $8FFF;
    WriteWord(A);
    A := A shl 2;
    WriteWord(A);
    A := A shr 4;
    WriteWord(A);
end;

procedure TestNotAndOrXor;
var
    A,B: Word;
begin
    writeln;
    writeln('-----');
    A:= $1234;
    B:= $5678;
    WriteWord(A);
    WriteWord(B);

    WriteWord(NOT A);
    WriteWord(A AND B);
    WriteWord(A OR B);
    WriteWord(A XOR B);
end;

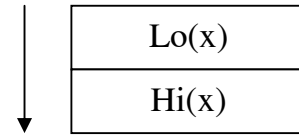
begin
    TestShift;
    TestNotAndOrXor;
end.
{
Ban Dau : w = 36863 =          8FFF = 1000 1111 1111 1111
Sau quay trai: w = 16380 =      3FFC = 0011 1111 1111 1100
Sau quay phai: w = 1023 =       03FF = 0000 0011 1111 1111

-----

A = 4660 =          1234 = 0001 0010 0011 0100
B = 22136 =         5678 = 0101 0110 0111 1000
NOT A = 60875 =      EDCB = 1110 1101 1100 1011
A AND B = 4656 =     1230 = 0001 0010 0011 0000
A OR B = 22140 =     567C = 0101 0110 0111 1100
A XOR B = 17484 =    444C = 0100 0100 0100 1100

```

- Hàm lấy byte cao của một từ (word)
function Hi(X): Byte;
- Hàm lấy byte thấp của một từ (word)
function Lo(X): Byte;
- Hàm hoán đổi nội dung byte thấp và byte cao
function Swap(X): (Same type as parameter);
- Ví dụ:



```

var
  A,B: Word;
  A1,A2: Byte;

procedure WriteHexWord(w: Word);
const
  hexChars: array [0..$F] of Char =
    '0123456789ABCDEF';
begin
  Write(hexChars[Hi(w) shr 4],
        hexChars[Hi(w) and $F],
        hexChars[Lo(w) shr 4],
        hexChars[Lo(w) and $F]);
end;

begin
  A := $1234; { 4660 }
  writeln;
  write('A = ', A, ' = $');WriteHexWord(A);

  { function Hi(X): Byte; }
  A1 := Hi(A); { $12 = 18 }
  writeln;
  write('A1 = ', A1, ' = $');WriteHexWord(A1);

  { function Lo(X): Byte; }
  A2 := Lo(A); { $34 = 52 }
  writeln;
  write('A2 = ', A2, ' = $');WriteHexWord(A2);
  writeln;

  { function Swap(X): (Same type as parameter); }
  B := Swap(A); { $3412 = 13330 }
  write('B = ', B, ' = $');WriteHexWord(B);
end.

- Hàm lấy địa chỉ đoạn của một biến, một thủ tục hay một hàm
function Seg(X): Word;
```

- Hàm lấy địa chỉ offset của một biến, một thủ tục hay một hàm

```
function Ofs(X): Word;
```

- Hàm lấy địa chỉ đoạn của đoạn mã hiện thời

```
function CSeg: Word;
```

- Hàm lấy địa chỉ đoạn của đoạn dữ liệu

```
function DSeg: Word;
```

- Hàm lấy địa chỉ đoạn của đoạn ngăn xếp (stack)

```
function SSeg: Word;
```

- Hàm lấy giá trị của con trỏ stack IP của đoạn ngăn xếp

```
function SPtr: Word;
```

- Ví dụ:

```
{ CSeg, DSeg, SSeg, SPtr, Ofs, and Seg functions.}
{ function Ofs(X): Word; }
```

```
procedure WriteHexWord(w: Word);
const
  hexChars: array [0..$F] of Char =
    '0123456789ABCDEF';
begin
  Write(hexChars[Hi(w) shr 4],
        hexChars[Hi(w) and $F],
        hexChars[Lo(w) shr 4],
        hexChars[Lo(w) and $F]);
end;
var
  i: Integer;
begin
  Write('The current code segment is $');
  WriteHexWord(CSeg); Writeln;

  Write('The global data segment is $');
  WriteHexWord(DSeg); Writeln;

  Write('The stack segment is $');
  WriteHexWord(SSeg); Writeln;

  Write('The stack pointer is at $');
  WriteHexWord(SPtr); Writeln;

  Write('i is at offset $');
  WriteHexWord(Ofs(i));

  Write(' in segment $');
  WriteHexWord(Seg(i));
end.
```

```
{
CSeg : The current code segment is $0DC7
DSeg : The global data segment is $0E6D
SSeg : The stack segment is $0E98
SPtr : The stack pointer is at $3FFE
i is at offset $0062 in segment $0E6D
}
```

- Hàm lấy địa chỉ của một biến, một thủ tục hay một hàm

```
function Addr(X): pointer;
```

- Hàm lấy địa chỉ khi biết địa chỉ đoạn và địa chỉ offset

```
function Ptr(Seg, Ofs: Word): Pointer;
```

- Ví dụ:

```
{ function Addr(X): pointer; }
{ function Ptr(Seg, Ofs: Word): Pointer; }
var
    P1, P2: ^Byte;
    i, j : Byte;
begin
    writeln('-----');
    i := 1;
    P1 := Addr(i);
    Writeln('i = ', P1^);

    j := i + 1;
    P2 := Ptr( seg(j), ofs(j));
    Writeln('j = ', P2^);
end.

{
i = 1
j = 2
}
```

- Thủ tục xin cấp phát và giải phóng bộ nhớ bộ nhớ

```
procedure GetMem(var P: Pointer; Size: Word);
procedure FreeMem(var P: Pointer; Size: Word);
```

- Ví dụ:

```
{ function SizeOf: Integer; }
{ procedure GetMem(var P: Pointer; Size: Word); }
{ The largest block that can be safely allocated
on the heap at one time is 65,528 bytes (64K-$8). }
{ procedure FreeMem(var P: Pointer; Size: Word); }
```

```
type
    NhanVien = record
```

```
Ten: string[40]; { 40 + 1 độ dài ký tự }
DiaChi: string[40]; { 40 + 1 độ dài ký tự }
DienThoai: string[20]; { 20 + 1 độ dài ký tự }
Tuoi: byte; { 1 }
GioiTinh: boolean; { 1 }
end;
var
  P: ^NhanVien;
begin
  GetMem(P, SizeOf(NhanVien));
  Writeln ('Kich thuoc bg la ', SizeOf(NhanVien), ' bytes. ');
  FreeMem (P, SizeOf(NhanVien));
end.
{ Kich thuoc cua ban ghi la 105 bytes. }
```

- Thủ tục sao chép các byte

```
procedure Move(var Source, Dest; Count: Word);
```

- Ví dụ:

```
{ procedure Move(var Source, Dest; Count: Word);
  Copies bytes from source to dest. }
```

```
procedure WriteHexWord(w: Word);
const
  hexChars: array [0..$F] of Char =
    '0123456789ABCDEF';
begin
  Write(hexChars[Hi(w) shr 4],
        hexChars[Hi(w) and $F],
        hexChars[Lo(w) shr 4],
        hexChars[Lo(w) and $F]);
end;
```

```
var
  A: array[1..2] of Byte;
  B: Word;
  i: Byte;
begin
  for i := 1 to 2 do A[i] := i;
  { A[1] = 1; A[2] = 2; => A = $0201 }
  Move(A, B, SizeOf(A));
  { B = 513 = $0201; }
  writeln;
  write('B = ', B);
  write(' = '); WriteHexWord(B);
end.
```

- Thủ tục điền đầy một số các byte với một giá trị kiểu byte hoặc kiểu char

```
procedure FillChar(var X; Count: Word; value);
```

- Truy nhập trực tiếp bộ nhớ

Mem, MemW, MemL

Mem để lấy một byte tại địa chỉ được trả bởi địa chỉ đoạn và địa chỉ offset.

MemW để lấy một word (2 bytes) tại địa chỉ được trả bởi địa chỉ đoạn và địa chỉ offset.

MemL để lấy một longint (4 bytes) tại địa chỉ được trả bởi địa chỉ đoạn và địa chỉ offset.

- Ví dụ:

```
var
  i: Byte;
  j: Word;
  k: Longint;
begin
  i := 1;
  j := 1000;
  k := 1000000;
  writeln;
  writeln('i = ', Mem[seg(i):ofs(i)]);
  writeln('j = ', MemW[seg(j):ofs(j)]);
  writeln('k = ', MemL[seg(k):ofs(k)]);
end.
```

Chương 3. Quản lý thiết bị ngoại vi và tệp

LT: 12 Tiết, TH:

Khi công nghệ thông tin ngày một phát triển, số lượng các thiết bị ngoại vi gắn vào máy tính ngày càng lớn. Tổ chức và quản lý các thiết bị ngoại vi ngày một khó khăn và quan trọng. Nhiệm vụ cơ bản và quan trọng nhất của các HĐH hiện đại là quản lý và khai thác tối ưu các thiết bị ngoại vi. Nhiệm vụ này quyết định hiệu suất chung của toàn hệ thống.

1. Nguyên lý phân cấp trong tổ chức và quản lý thiết bị ngoại vi

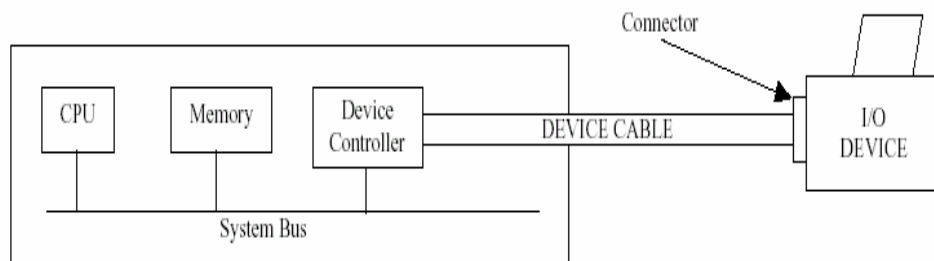
a. Giới thiệu

Thiết bị ngoại vi hết sức đa dạng, phong phú về chủng loại và có thể gắn vào vào hệ thống với số lượng lớn. Ngoài các thiết bị chuẩn có tính chất bắt buộc do người sản xuất cung cấp như bàn phím, màn hình, chuột □ các hệ thống tính toán phải luôn có tính mở tức là khả năng giao tiếp với số lượng tùy ý các thiết bị ngoại vi bổ xung. Điều này là cần thiết vì hệ thống tính toán phải sẵn sàng có thể sử dụng cho một môi trường bất kỳ.

Ví dụ: Máy tính có thể gắn vào máy đọc thẻ để kiểm tra nhân viên có đi làm hay không? để hạn chế sự xâm phạm trái phép.

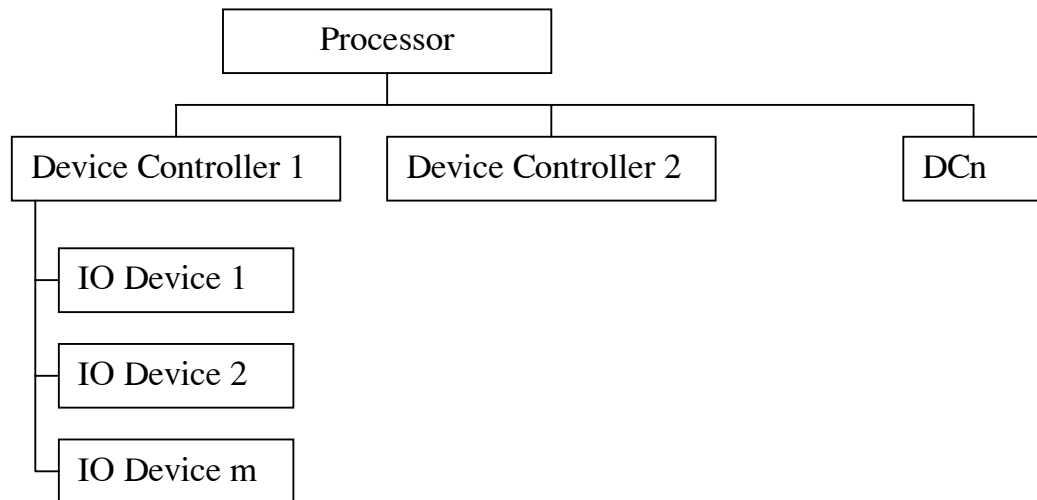
Nói chung cấu tạo và chức năng của các thiết bị ngoại vi là khác nhau. Ví dụ: màn hình là thiết bị ra chuẩn, bàn phím là thiết bị vào chuẩn. Tuy chúng đều là thiết bị chuẩn nhưng rõ ràng cấu tạo và hoạt động của hai thiết bị này khác nhau rất nhiều. Hơn nữa các tín hiệu điều khiển được truyền đến các thiết bị sẽ điều khiển thao tác hiện hành của nó. Rất khó để có các tín hiệu riêng cho từng chức năng riêng của các thiết bị bao gồm cả các thiết bị có thể được thêm vào trong suốt thời gian tồn tại của hệ thống. Ví dụ: 1 tín hiệu của hệ thống đưa đến máy in có thể hiểu là bỏ qua một dòng (dòng trắng) nhưng lại có ý nghĩa là tua lại nếu tín hiệu được đưa đến ổ băng từ.

Chính vì vậy mà processor không thể làm việc trực tiếp với các thiết bị ngoại vi này mà chỉ làm việc với các thiết bị điều khiển (Device Controller).



Các thiết bị điều khiển này sẽ thông dịch tín hiệu điều khiển cho phù hợp với thiết bị ngoại vi gắn với nó và sẽ điều khiển thao tác tương ứng với thiết bị.

Đối với một hệ thống máy tính các thiết bị điều khiển hoạt động như những máy tính chuyên dùng (có hệ lệnh riêng và chương trình riêng). Một máy tính có thể có nhiều thiết bị điều khiển.



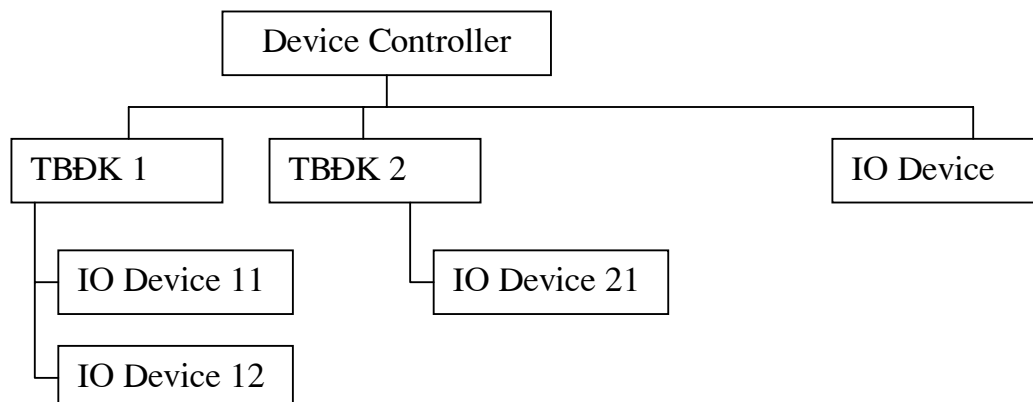
b. Kênh

Một thiết bị điều khiển và các thiết bị ngoại vi do nó điều khiển tạo thành một hệ thống hoạt động độc lập gọi là kênh. Một máy tính có thể có nhiều kênh, các kênh này phải có khả năng liên hệ với processor.

– Kênh đơn và kênh bó:

Nếu thiết bị điều khiển làm việc trực tiếp với thiết bị vào ra ta có kênh đơn.

Kênh bó có kênh có nhiều kênh đơn, tức là thiết bị điều khiển lại điều khiển các thiết bị điều khiển khác.



- Trao đổi vào ra với kênh

Để thực hiện một phép trao đổi vào ra ở một kênh nào đó thì processor phải tạo ra chương trình trên ngôn ngữ của thiết bị điều khiển (còn gọi là chương trình kênh). Tiếp theo processor chuyển giao chương trình kênh cùng với dữ liệu cần thiết cho kênh. Sau đó processor tiếp tục thực hiện công việc của mình.

Như vậy các công việc của kênh được thực hiện song song với công việc của processor điều này làm tăng tốc độ chung của cả hệ thống.

Khi kênh thực hiện xong công việc của mình kênh sẽ báo về cho processor dưới dạng ngắt vào/ra cùng với mã kết quả thực hiện công việc (return code). Tín hiệu ngắt có thể được xử lý ngay lập tức hay phải chờ đợi hoặc thậm chí bị huỷ bỏ nếu processor thấy không cần thiết. Mã kết quả bao giờ cũng được lưu trữ để chờ processor xử lý.

Khi processor dừng công việc của mình để đánh giá công việc thực hiện của kênh (tín hiệu ngắt được xử lý), processor sẽ lấy mã trở về từ kênh. Nếu mã cho biết kênh thực hiện tốt đẹp thì processor có thể giao tiếp công việc mới cho kênh (nếu còn). Ngược lại, processor có thể yêu cầu kênh thực hiện lại công việc. Nếu sau n lần (phụ thuộc vào từng hệ điều hành) vẫn không được ta mới nhận được thông báo lỗi.

Ngôn ngữ kênh có thể được đưa vào hệ thống khi nạp hệ điều hành hoặc có thể được cung cấp cho processor ngay khi hệ thống hoạt động.

Đối với MSDOS giải quyết bằng các đưa các câu lệnh điều khiển thiết bị trong CONFIG.SYS hoặc thực hiện các chương trình cung cấp ngôn ngữ kênh trong khi hệ điều hành đang làm việc.

- Ví dụ

Đối với thiết bị ngoại vi chuột

- + Ta có thể đưa vào tệp config.sys câu lệnh

Device = C:\mouse\mouse.sys

- + Hoặc thực hiện chương trình Mouse.com

- Ví dụ (mở rộng nói thêm): tạo ổ đĩa ảo

Device = c:\windows\Ramdrive.sys 200

Tạo một đĩa ảo (RAM disk) kích thước 200K, nếu chỉ có ổ mềm A: thì ổ đĩa này sẽ là đĩa B:

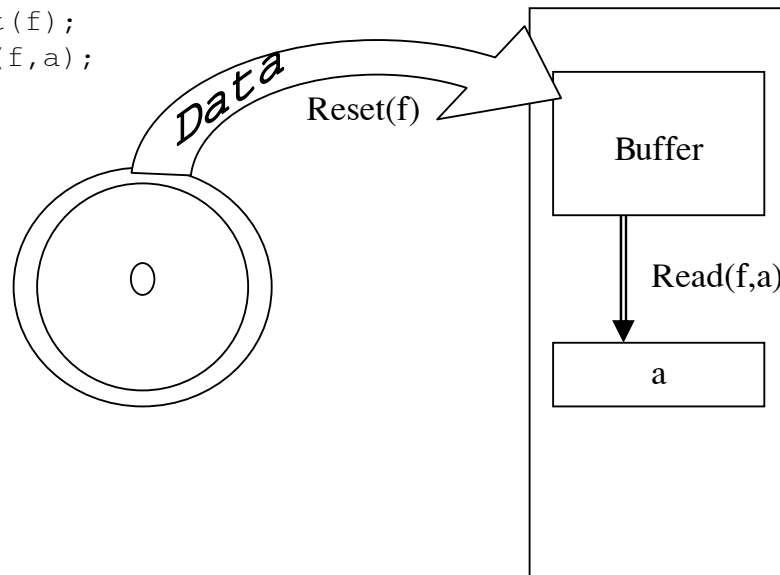
2. Phòng Đệm (Buffer)

Đặc trưng cơ bản của thiết bị ngoại vi là tốc độ hoạt động nhỏ hơn nhiều lần so với tốc độ hoạt động của processor. Để thực hiện một phép vào ra hệ thống phải kích hoạt thiết bị, chờ đợi thiết bị đạt trạng thái thích hợp (Ví dụ như máy in phải chờ nóng □) và sau đó chờ đợi công việc được thực hiện. Chính vì

vậy phần lớn các thiết bị vào ra làm việc với từng khối dữ liệu chứ không phải từng byte riêng lẻ. Để đảm bảo năng suất, hệ thống cần phải

- + Cố gắng thực hiện song song công việc vào ra với các phép xử lý thông tin khác
- + Giảm số lượng các phép trao đổi vào ra vật lý
- + Thực hiện trước các phép nhập dữ liệu
- Như vậy người ta phải sử dụng phòng đệm để nâng cao năng suất
 - + Phòng đệm của hệ điều hành là một vùng nhớ dùng để lưu trữ tạm thời các thông tin phục vụ cho các phép vào ra.
 - + Ngoài ra còn có phòng đệm của thiết bị không phụ thuộc vào hệ điều hành gọi là phòng đệm kỹ thuật. Ví dụ phòng đệm của máy in.
- Ví dụ

```
Assign(f, 'f1.txt');
Reset(f);
Read(f, a);
```



Khi thực hiện Reset(f) thì hệ thống đã đưa dữ liệu từ đĩa lên vùng đệm. Khi chương trình muốn đọc dữ liệu từ tệp vào biến a thì hệ thống chỉ cần lấy dữ liệu từ vùng đệm thay cho việc đọc tệp.

Giả thiết mỗi lần truy nhập đĩa mất 0,01 giây, kích thước vùng đệm là 512 bytes và thời gian truy nhập vào bộ nhớ là rất nhỏ (so với 0,01)

Số byte cần đọc	Không có vùng đệm	Có vùng đệm
1B	0,01''	0,01''
512B	5'' = 512x0.01	0,01''
5KB	50'' = 10x5	0.1'' = 10x0.01
50KB	8' = 10x50	1'' = 10x0.1

- Phân loại phòng đệm

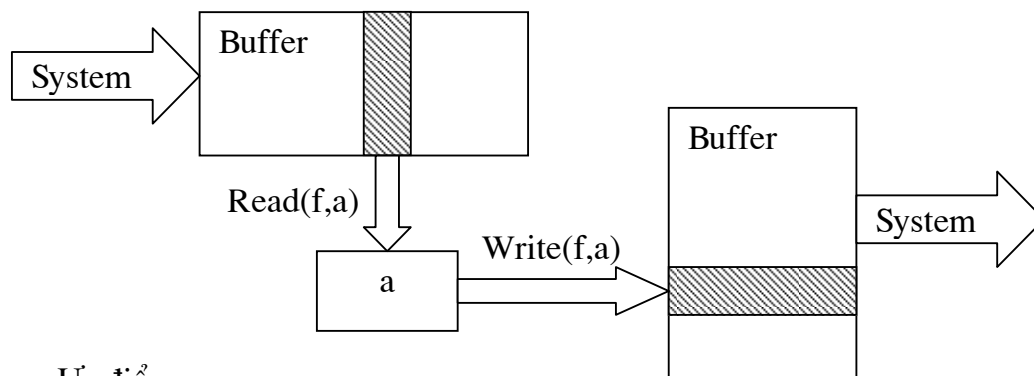
a. Phòng đệm trung chuyển

- Là phòng đệm thuần túy lưu trữ tạm thời các phép phục vụ vào ra.
- Phòng đệm này có hai loại
 - + Phòng đệm vào là phòng đệm chỉ dùng để nhập thông tin. Trong hệ thống sẽ có lệnh để đưa thông tin vào phòng đệm (đọc vật lý).

Khi gặp chỉ thị đọc (READ), thông tin sẽ được tách và chuyển từ phòng đệm vào các địa chỉ tương ứng trong chương trình ứng dụng. Như vậy, mỗi giá trị được lưu trữ ở hai nơi trong bộ nhớ (một ở phòng đệm và một ở vùng bộ nhớ trong chương trình ứng dụng). Khi giá trị cuối cùng của phòng đệm vào được lấy ra thì phòng đệm được giải phóng (rỗng) và hệ thống đưa thông tin mới vào phòng đệm trong thời gian ngắn nhất có thể.

Để giảm thời gian chờ đợi, hệ thống có thể tổ chức nhiều phòng đệm vào, khi hết thông tin ở một phòng đệm, hệ thống sẽ chuyển sang phòng đệm khác.

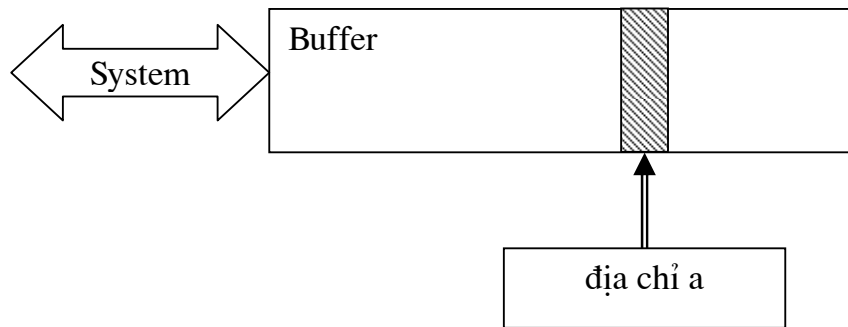
- + Phòng đệm ra là phòng đệm để ghi thông tin. Trong hệ thống có lệnh để giải phóng phòng đệm (ghi vật lý). Khi có chỉ thị ghi (WRITE), thông tin được đưa vào phòng đệm. Khi phòng đệm ra đầy, hệ thống sẽ đưa thông tin ra thiết bị ngoại vi. Hệ thống cũng có thể tổ chức nhiều phòng đệm ra.



- Ưu điểm:
 - + Đơn giản
 - + Có hệ số song song cao vì tốc độ giải phóng vùng đệm lớn
 - + Có tính chất vạn năng, thích ứng với mọi phương pháp truy nhập
- Nhược điểm:
 - + Tốn bộ nhớ
 - + Tốn thời gian để trao đổi thông tin trong bộ nhớ

b. Phòng đệm xử lý

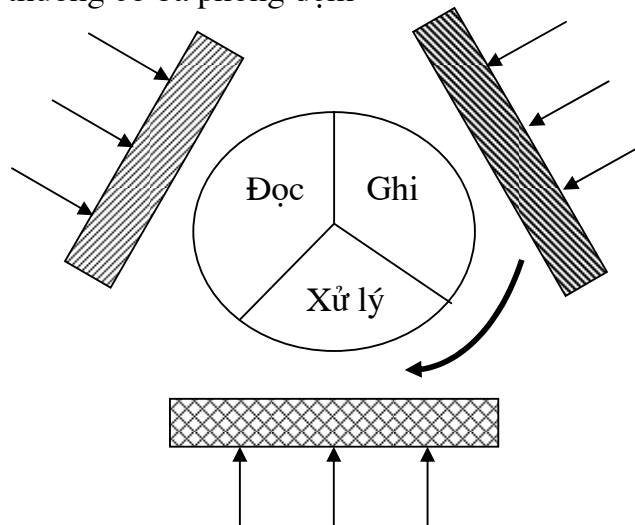
Thông tin được xử lý ngay trong phòng đệm không ghi lại vào nơi khác trong bộ nhớ. Chỉ thị đọc xác định địa chỉ thông tin chứ không cung cấp thông tin chứ không cung cấp giá trị.



- Ưu điểm:
 - + Tiết kiệm bộ nhớ
 - + Không mất thời gian chuyển thông tin ở bộ nhớ trong, thích hợp khi cần kích thước bản ghi dữ liệu lớn.
- Nhược điểm
 - + Tính vạn năng không cao
 - + Hệ số song song thấp

c. Phòng đệm vòng tròn

Phòng đệm vòng tròn thường có ba phòng đệm



- Sau một khoảng thời gian vai trò của ba phòng đệm được thay đổi cho nhau.
- Ưu điểm:

- + Có sự đồng bộ giữa đọc, ghi và xử lý (ba quá trình được thực hiện song song).
- + Thường áp dụng cho hệ cơ sở dữ liệu và hữu dụng nhất khi lượng thông tin vào bằng lượng thông tin ra.
- Nhược điểm (không thấy có, tự nghĩ ra)

3. SPOOL- Simultaneous Peripheral Operation On-Line

Mô phỏng các phép trao đổi vào ra ngay trong lúc thực hiện

- Spool là cơ chế thay một thiết bị ngoại vi bằng một thiết bị trung gian có khả năng dùng chung, có tốc độ cao và sau đó thay trở lại thiết bị trung gian bằng thiết bị cuối khi điều kiện cho phép.
- Ưu điểm:
 - + Có thể mau chóng kết thúc chương trình người sử dụng
 - + Ta giải phóng được các ràng buộc về số lượng thiết bị
 - + Khai thác các thiết bị ngoại vi một cách tối ưu
- Các phương pháp tổ chức SPOOL
 - + Phương pháp 1

Tiến hành đưa thông tin ra thiết bị trung gian ở trên thiết bị mà chúng ta mô phỏng và chúng ta có thể sao chép nguyên văn kết quả ra thiết bị cuối, điều này chỉ làm được khi 2 thiết bị có tính năng tương đương.

- + Phương pháp 2

Bước 1: Tạo chương trình kênh 1 theo yêu cầu của người sử dụng

Ví dụ: người sử dụng cần đưa ra máy in => tạo chương trình kênh đưa ra máy in.

Bước 2: Tạo chương trình kênh 2 có nhiệm vụ lưu trữ chương trình kênh 1 và dữ liệu của nó lên thiết bị trung gian

Xử lý kết thúc: chuyển chương trình kênh đã lưu trữ (chương trình kênh 1) ra kênh của thiết bị cuối.

4. Quản lý màn hình

a. Giới thiệu

Các thành phần của thiết bị màn hình. Trong máy IBM PC, thiết bị màn hình gồm có hai thành phần:

1. Card màn hình (display adapter)
2. Màn hình hiển thị, còn gọi tắt là màn hình (monitor)

Card màn hình nối máy tính với màn hình thông qua một chip là bộ điều khiển (Cathode Ray Tube Controller). Card màn hình có các cổng vào/ra lập trình được, vùng nhớ (để tạo) ký tự ROM và bộ nhớ màn hình RAM chứa thông tin cần đưa ra màn hình hiển thị.

Các cổng vào ra của màn hình:

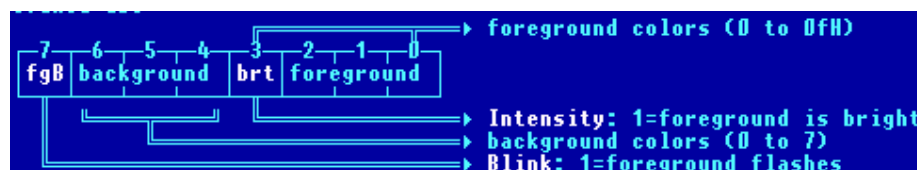
Cổng	Mô tả	Mô tả thêm
2c0-2df	EGA #2	EGA I/O Ports
3b0-3df	Video Graphics Array (VGA)	VGA I/O Ports
3c0-3cf	Enhanced Graphics Adapter (EGA) #1	EGA I/O Ports
3d0-3df	Color/Graphics Adapter (CGA) and EGA	CGA I/O Ports

b. Bộ nhớ màn hình

Bộ nhớ màn hình về mặt logic được coi như một phần của bộ nhớ nằm ở vị trí A0000-BFFFF là nơi lưu trữ thông tin hiển thị cho màn hình ở cả chế độ đồ họa và chế độ văn bản.

- Với màn hình VGA (được sử dụng phần lớn hiện nay), bộ nhớ màn hình văn bản bắt đầu từ vị trí B8000-BFFFF (dài 8000H). Trong chế độ này mỗi ký tự trên màn hình tương ứng với 2 byte trong bộ nhớ.
 - + Byte đầu tiên sẽ lưu trữ mã ASCII của ký tự
 - + Byte tiếp theo lưu trữ thuộc tính của ký tự

7	6	5	4	3	2	1	0
Blink	Red	Green	Blue	Intensity	Red	Green	Blue
Màu nền				Màu ký tự			



Intensity = 1 : màu chữ được tăng độ sáng

Fgb = 1 : màu nền được tô sáng

– Bảng màu

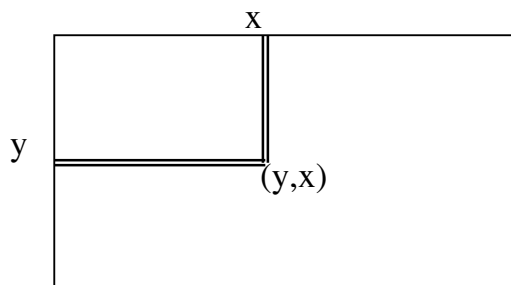
Do ba bit kiểu R,G,B tạo thành các màu

Số thứ tự	Dạng Hex	Màu
1	00H	black
2	01H	blue
3	02H	green
4	03H	cyan
5	04H	red
6	05H	magenta
7	06H	brown
8	07H	white
9	08H	gray
10	09H	bright blue
11	0aH	bright green
12	0bH	bright cyan
13	0cH	bright red
14	0dH	bright magenta
15	0eH	yellow
16	0fH	bright white

Đối với màu nền chỉ có thể có các màu từ 1 — 8 tức là đến màu có mã là 7H (White).

– Hiển thị một ký tự

Ký tự có vị trí (y,x) tính từ (0,0) đến (79,24) trên màn hình thì vị trí của nó trong bộ nhớ là



$$B8000 + 80.2.y + 2.x = B800:160y + 2.x$$

Như vậy tại ô nhớ có địa chỉ $B800:160y + 2.x$ ta đưa mã ký tự cần hiển thị

tại ô nhớ có địa chỉ $B800:160y + 2x + 1$ ta đưa thuộc tính ký tự

- Giải thích khai báo biến dạng absolute

- + Khi ta khai báo biến như sau

```
Var tên_biến : tên_kiểu absolute Seg:Ofs;
```

Thì chương trình sẽ tạo ra một biến với địa chỉ của biến tại địa chỉ được trả bởi seg và ofs. Khai báo biến dạng này cho phép ta truy nhập bộ nhớ một cách thuận tiện hơn.

- + Ví dụ:

```
head: Word absolute $0040:$001A;
```

Chương trình sẽ tạo ra một biến kiểu word tại địa chỉ \$0040:\$001A;

- Ví dụ in một ký tự 'X' tại dòng 10, cột 40 tức là $(y,x) = (39,9)$ với màu nền là màu xanh (có mã là 01H) và chữ màu đỏ (có mã là 04H)

```
uses crt,dos;
var
  A: Array[0..4000] of Byte absolute $B800:$0000;
  x,y: integer;
  ch : char;
  tt : byte;
begin
  clrscr;
  ch := 'X';
  x := 39; y := 9;
  A[160*y + 2*x] := ord(ch); { ký tự X }
  tt := tt OR $14; { mau nen = BLUE, mau chu = RED }
  A[160*y + 2*x + 1] := tt;
  readkey;
end.
```

- Ví dụ: in ra màn hình xâu 'Truong Dai Hoc Bach Khoa'

```
uses crt,dos;
const
  str : string = 'Truong Dai Hoc Bach Khoa';
var
  A: Array[0..4000] of Byte absolute $B800:$0000;
  x,y,i: integer;

begin
  clrscr;
  x := 9; y := 9; { dòng 10 cột 10 }
  for i:= 1 to length(str) do
    A[160*y + 2*(x+i)] := ord(str[i]);
  readkey;
end.
```

- Ví dụ: Khởi tạo màn hình đồ hoạ 256 màu (một pixel chiếm 1 byte) và vẽ một đường thẳng từ (0,0) — (99,99) (**Cần phải đặt lại trang???**)

```
uses Graph,Crt;
var
    gd,gm: integer;
    A: Array[0..4000] of Byte absolute $A000:$0000;
    i: integer;
begin
    gd :=installUserDriver('SVGA256',Nil);
    gm := 2; { 640 x 480 }
    InitGraph(gd,gm,'c:\tp70\bgi');
    for i := 0 to 99 do
        A[i*(getmaxx + 1) + i] := Green;
        { vì getmaxx cho 639 vì vậy phải cộng thêm 1 }
    readkey;
    closegraph;
end.
```

c. Một số hàm phục vụ màn hình của ROM BIOS

Ta sử dụng ngắt 10h của ROM BIOS cho màn hình. Giá trị của hàm được đưa vào thanh ghi AH.

- Đặt kích thước con trỏ: Ta sử dụng hàm 01h

+ Input:

```
AH = 01H
CH = start line (0-1fH; 20H=no cursor)
CL = end line (0-1fH)
```

- Đặt vị trí con trỏ: Ta sử dụng hàm 02h

+ Input:

```
AH = 02H
BH = video page (0-based)
DH,DL = row,column (0-based)
```

+ Ví dụ:

```
uses crt, dos;
var
    r: Registers;
begin
    r.ah := $02; { hàm đặt vị trí con trỏ }
    r.bh := 0; { trang 0 }
    r.dh := 10; { dòng 10 }
    r.dl := 40; { cột 40 }
    intr($10,r);
    readkey;
end.
```

- Đọc con trỏ: Ta sử dụng hàm 03h

+ **Input:**

AH = 03H
BH = video page (0-based)

+ **Output:**

DH,DL = current row,column of cursor
CH,CL = current start,end line of cursor

+ **Ví dụ:**

```
uses crt, dos;
var
  r: Registers;
begin
  clrscr;

  r.ah := $02; { ham dat vi tri con tro }
  r.bh := 0; { trang 0 }
  r.dh := 10; { dong 10 }
  r.dl := 40; { cot 40 }
  intr($10,r);

  r.ah := $02;
  r.bh := 0;
  intr($10,r);
  write('x');

  writeln;
  writeln('vi tri hien thoi cua con tro la (',
          r.dh, ' ', ' ', r.dl, ')');
  writeln('start line = ', r.ch, ' ', end line = ', r.cl);
  readkey;
end.
```

- **Cuộn màn hình lên một số dòng trong phạm vi một cửa sổ: Sử dụng hàm 06h**

+ **Input:**

AH = 06H
CH,CL = row,clm of upper left corner of window (0-based)
DH,DL = row,clm of lower right corner of window
AL = number of blank lines to scroll in (0=blank entire window)
BH = video attribute to be used on blank lines

+ **Ví dụ:**

```
uses crt, dos;
var
  r: Registers;
  i,j: byte;
begin
```

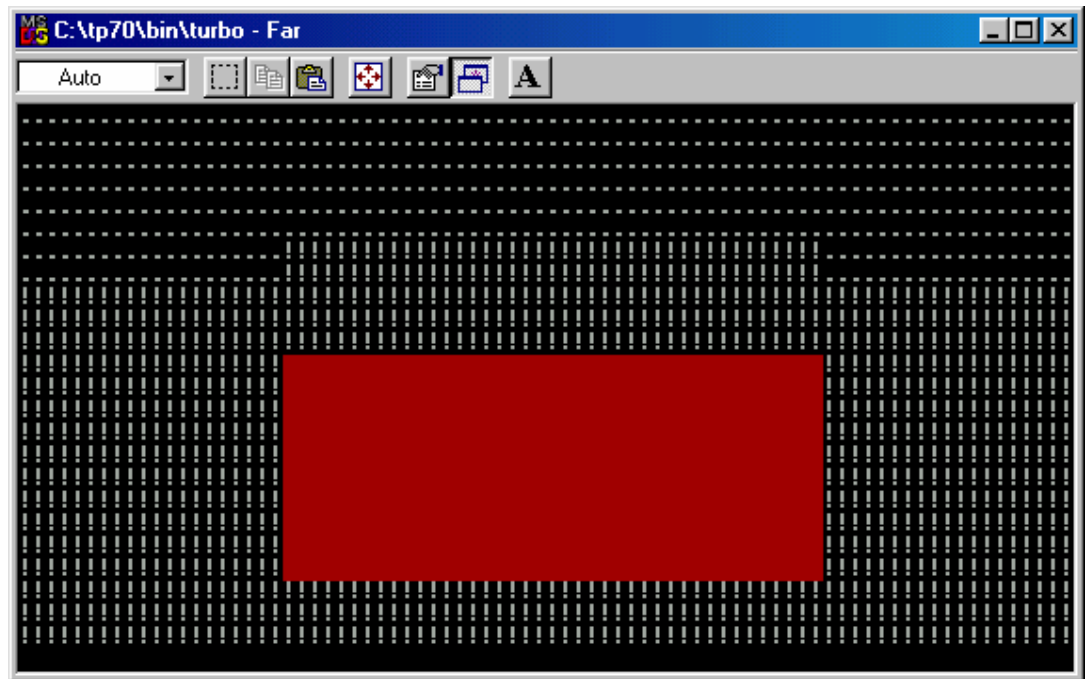
```

clrscr;
for i := 1 to 9 do
  for j := 1 to 80 do
    write('.');
for i := 10 to 25 do
  for j := 1 to 80 do
    write('!');

r.ah := $06; { ham cuon trong cua so }
r.al := 10;
r.ch := 6; { dong 6 goc trai}
r.cl := 20; { cot 20 goc trai}
r.dh := 20; { dong 20 goc phai}
r.dl := 60; { dong 60 goc phai }
r.bh := $40; {màu nền=màu đỏ – xem thêm thuộc tính kt)
intr($10,r);

readkey;
end.

```



- Cuộn màn hình xuống một số dòng trong phạm vi một cửa sổ:
 - + Sử dụng hàm 07h. Các tham số như hàm 06h
- Bài tập: Lưu lại toàn bộ màn hình văn bản ra tệp, sau đó khôi phục lại màn hình khi hoàn thành công việc giải phương trình bậc 2.

- Lời giải

```
uses crt,dos;
type
  MH = Array[0..4000] of Byte;
var
  A: MH absolute $B800:$0000;
  f: file of MH;
begin
  clrscr;
  writeln('Xin chao cac ban da den voi mon HDH');
  writeln('Ban khoe chu');

  assign(f,'mh.hex');
  rewrite(f);
  write(f,A); { dua bo nho man hinh ra tep }
  close(f);
  while keypressed do readkey; readkey;

  clrscr;
  writeln('Bam mot phim bat ky');
  while keypressed do readkey; readkey;

  reset(f);
  read(f,A); { dua tu tep ra bo nho man hinh }
  while keypressed do readkey; readkey;

  close(f);
end.
```

5. Quản lý bàn phím

a. Giới thiệu

Bàn phím được điều khiển thông qua 1 bộ điều khiển bàn phím là bộ Vi Xử Lý 8048 (đối với PC chuẩn) hoặc 8042 (đối với máy AT). Mỗi khi có sự kiện bấm hoặc nhả phím thì bộ điều khiển này có nhiệm vụ báo cho ROM-BIOS biết để xử lý. Nếu một phím được bấm lâu thì bộ điều khiển lặp lại phím này sau những khoảng xác định. Mỗi lần bấm thì các vi mạch của bàn phím tạo ra một số 1 byte gọi là mã quét (scan code) đặc trưng cho phím tương ứng. Bàn phím tạo ra một mã scan khác khi một phím được nhả.

Cụ thể: Khi bấm một phím, bàn phím tạo ra một mã scan, khi nhả phím đó bàn phím tạo ra một mã scan khác bằng mã scan lúc bấm cộng thêm 128 (cho bit 7 của mã scan lúc bấm bằng 1).

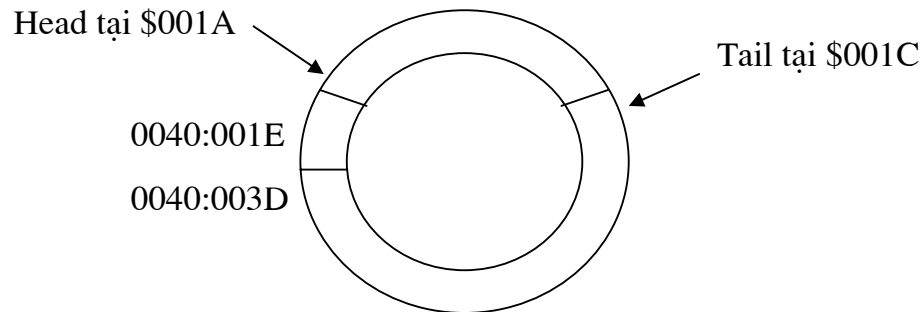
Keyboard Scan Codes											
Hex	Dec	Key	Hex	Dec	Key	Hex	Dec	Key	Hex	Dec	Key
01	1	Esc	12	18	E	23	35	H	34	52	. >
02	2	1 !	13	19	R	24	36	J	35	53	/ ?
03	3	2 @	14	20	T	25	37	K	36	54	Shft(Rt)
04	4	3 #	15	21	Y	26	38	L	37	55	* PrtSc
05	5	4 \$	16	22	U	27	39	; :	38	56	Alt
06	6	5 %	17	23	I	28	40	" '	39	57	spacebar
07	7	6 ^	18	24	O	29	41	` ~	3a	58	CapsLock
08	8	7 &	19	25	P	2a	42	Shft(L)	3b	59	F1
09	9	8 *	1a	26	[{	2b	43	\	3c	60	F2
0a	10	9 (1b	27] }	2c	44	Z	3d	61	F3
0b	11	0)	1c	28	Enter	2d	45	X	3e	62	F4
0c	12	- _	1d	29	Ctrl	2e	46	C	3f	63	F5
0d	13	+ =	1e	30	A	2f	47	V	40	64	F6
0e	14	bksp	1f	31	S	30	48	B	41	65	F7
0f	15	Tab	20	32	D	31	49	N	42	66	F8
10	16	Q	21	33	F	32	50	M	43	67	F9
11	17	W	22	34	G	33	51	, <	44	68	F10

Khi người sử dụng bấm phím, bàn phím không hề biết ý nghĩa của phím được bấm thì chỉ thông báo có tác động phím thông qua ngắt 9H. Ngắt 9H gọi chương trình xử lý ngắt, chương trình này sẽ đọc giá trị ở cổng 60H để biết tác động phím nào đã xảy ra. Sau đó mã scan được bàn phím trao cho ROM-BIOS và được các trình phục vụ bàn phím đổi thành 2 byte. Byte thấp chứa mã ASCII của phím còn byte cao chứa mã scan từ bàn phím. Với các phím chức năng không có mã ASCII nên byte thấp có giá trị là 0. Sau đó ROM-BIOS sẽ đặt 2 byte này vào một hàng đợi nằm trong bộ nhớ.

b. Bộ đệm bàn phím

Bộ đệm bàn phím gồm có 32 byte từ địa chỉ 0040:001E - 0040:003D, chứa tối đa là 16 ký tự (vì mỗi ký tự chiếm 2 byte : 1 cho mã ASCII và 1 cho mã

scan). Để chỉ tới vị trí trong bộ nhớ của ký tự đầu tiên (trong các ký tự còn trong bộ đệm) ta dùng một từ nhớ tại địa chỉ \$0040:\$001A, gọi là con trỏ đầu (Head). Vị trí của ký tự tiếp theo được chỉ bởi nội dung của từ nhớ tại địa chỉ \$0040:\$001C gọi là con trỏ cuối (Tail). Head và Tail chỉ là địa chỉ offset của đoạn có địa chỉ đoạn là \$0040.



- + Khi có cần đưa một ký tự vào bộ đệm thì đưa vào hai byte nhớ được trỏ bởi con trỏ Tail. Sau đó Tail được tăng lên 2 (Tail := Tail + 2, nếu tail > \$003D thì Tail := \$001E). Khi lấy ra một ký tự thì hệ thống lấy hai byte được trỏ bởi con trỏ Head, sau đó Head được tăng lên.
- + Nếu có ký tự trong bộ đệm thì giá trị của head khác giá trị của tail.
Ta có thể thay hàm KeyPressed bằng phép so sánh (head <> tail)
- + Nếu bộ đệm rỗng thì giá trị của head bằng giá trị của tail.
Ta có thể “xoá rỗng” vùng đệm bàn phím bằng cách gán giá trị của head = tail (head := tail) hoặc ngược lại (tail := head).
- Ví dụ: chương trình đọc mã scan và mã ASCII khi bấm phím. Chú ý: đối với một số phím như shift hay caps lock thì không bắt được do chương trình xử lý ngắt không chuyển thành mã hai byte. Muốn bắt được thì dùng chương trình chặn ngắt bàn phím và đọc từ cổng 60H.

```
uses crt,dos;
var
  head: Word absolute $0040:$001A;
  tail: Word absolute $0040:$001C;
  ch1, ch2, i : byte;
begin
  clrscr;
  repeat
    { chờ phím bấm }
    while (head = tail) do; { thay cho Not KeyPressed }

    { đọc mã ascii của ký tự }
    ch1 := Mem[$0040 : head];
```

```

        { đọc mã scan của ký tự }
        ch2 := Mem[$0040 : (head + 1)];
write('Ky tu ', chr(ch1), ' co ma ascii = ', ch1);
write(' va ma scan = ', ch2);      writeln;

        head := tail; { thay cho lệnh readkey; }
until (ch1 = 13); { cho đến khi gặp phím Enter }
end.

```

- **Ví dụ 2: chương trình giả lập bấm phím**

```

{ Dua ra vùng đệm bàn phím lệnh Dir và Enter }
uses crt,dos;
const
  a:array[0..7] of byte=($44,$20,$69,$17,$72,$13,$0D,$1C);
  { gom co lenh Dir va dau Enter (Ascii va scan code) }
var
  head: Word absolute $0000:$041A;
  tail: Word absolute $0000:$041C;

  i : byte;

procedure WriteToKb(ch : byte);
begin { bộ nhớ bàn phím từ 0040:001E - 0040:003D }
  Mem[$0040 : tail] := ch; tail := tail + 1;
  if (tail > $003D) then tail := $001E;
end;

begin
  writeln('Chuong trinh gia lap go phim');
  head := tail;
  for i := 0 to 7 do WriteToKb(a[i]);
  { chay tu dau nhac dos se thay lenh dir duoc thuc hien }
  { hoac thay bang 'while keypressed do write(readkey);' }
end.

```

c. Chuyển đổi các mã scan

Khi ROM-BIOS nhận được mã scan qua cổng 60H thì nó sẽ tiến hành chuyển sang mã 2 byte. Trong quá trình chuyển đổi ROM-BIOS luôn kiểm tra trạng thái các phím SHIFT, CTRL, ALT và các phím Capslock, Numlock để trả kết quả đúng. Trạng thái các phím đặc biệt này được ROM-BIOS lưu trong 2 byte nằm tại địa chỉ \$0040:\$0017 (hay 0417) và \$0040:\$0018 (hay 0418). ROM-BIOS cũng kiểm tra 1 số tổ hợp phím đặc biệt có tác dụng như là các lệnh yêu cầu ROM-BIOS thực hiện một công việc nào đó. Ví dụ như Ctrl — Alt — Del yêu cầu khởi động lại máy.

- Bài tập: Lập trình hiện thị trạng thái của các phím

Caps lock (AND \$40), Num lock (AND \$20), Scroll lock (AND \$10)

```
uses crt,dos;
var
  a: byte absolute $0040:$0017;
const
  tg : array[false .. true] of string = ('Tat','Bat');
begin
  writeln('Trang thai cac phim nhu sau :');
  writeln('Trang thai phim Caps lock la ', tg[(a AND $40) > 0]);
  writeln('Trang thai phim Num lock la ', tg[(a AND $20) > 0]);
  writeln('Trang thai phim Scroll lock la ', tg[(a AND $10) > 0]);
end.
```

d. Một số hàm phục vụ bàn phím của ROM-BIOS

- Trong ROM-BIOS có hai ngắt khác nhau cho bàn phím
 - + Ngắt 9H: Dùng để thu thập dữ liệu từ bàn phím và đặt vào vùng đệm ở địa chỉ thấp trong bộ nhớ.
 - + Ngắt 16H: Đáp ứng các yêu cầu phục vụ bàn phím và truyền dữ liệu từ bộ đệm bàn phím đến các chương trình khác.
- Đọc một ký tự kế tiếp từ bàn phím: Ta sử dụng ngắt 16h, hàm 00h

+ Input:

AH = 00H

+ Output:

AL = ASCII { bằng 0 cho các phím đặc biệt }

AH = Scan Code

- Kiểm tra đã có ký tự trong bộ đệm chưa: Ta sử dụng ngắt 16h, hàm 01h

+ Input:

AH = 01H

+ Output:

ZF = ZR = 1: Nếu không có ký tự nào

ZF = NZ = 0: Nếu trong bộ đệm có ký tự

Thanh ghi cờ

15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
				OF	DF	IF	TF	SF	ZF		AF		PF		CF

ZF := (Flags AND \$40)

e. Thay đổi ngắt bàn phím

- Thay đổi ngắt bàn phím về cơ bản cũng giống như thay đổi các ngắt khác của hệ thống nhưng có một số điểm khác

- + Phải gọi là chương trình xử lý ngắt bàn phím của hệ thống để chuyển dữ liệu từ mã scan về mã 2 byte. Vì vậy ta khai báo một biến kiểu thủ tục và gán địa chỉ của chương trình xử lý ngắt cũ cho địa biến này để sau này ta có thể gọi được thủ tục này.

```
Var kb: procedure;
getintvec($9,@kb);
```

- + Trước khi gọi lại chương trình xử lý ngắt trên phải khôi phục lại thanh ghi cờ bằng lệnh PushF (push flags) hay có mã máy là \$9C.

- Ví dụ: chương trình đọc mã scan bàn phím

```
uses dos,crt;
var
    kb: procedure;
    ch: char;
{$F+}
procedure keyclick; interrupt;
begin
if (Port[$60] < $80) then writeln('Scan = ',Port[$60]);
{ chỉ lay ma scan khi bam phim, khong lay ma nha phimm }
    inline($9C); { PushF push flags }
    kb; { gọi lại thủ tục xử lý ngắt cũ }
end;
{$F-}

begin
    getintvec($9,@kb);
    setintvec($9,addr(keyclick));
    repeat ch := readkey;
    until ch = #27;
    setintvec($9,@kb);
end.
```

- Ví dụ: hoán đổi hai phím 1 và 2 (ở trên hàng phím trên)

```
uses dos,crt;
var
    kb: procedure;
    ch: char;
    sc: byte;

procedure Swap(sc : byte);
var
    tail: Word absolute $0000:$041C;
    head1: word;
    ch: byte;
begin
    { bộ nhớ bàn phím từ 0040:001E - 0040:003D }
```

```

    if (sc = $02) OR (sc = $03) then begin
        if(sc = $02) then { neu bam so 1 tren day phim tren }
            begin ch := $32; { so 2 }
                sc := $03;
            end
        else
            begin ch := $31; { so 1 }
                sc := $02;
            end;
        head1 := tail - 2;
        if (head1 < $001E) then head1 := $003C;
        Mem[$0040 : head1] := ch;
        Mem[$0040 : (head1 + 1) ] := sc;
    end;
end;

{$F+}
procedure keyclick; interrupt;
begin
    sc := Port[$60];
    inline($9C); { PushF push flags }
    kb;
    if (sc < $80) then Swap(sc);
end;
{$F-}

begin
    getintvec($9,@kb);
    setintvec($9,addr(keyclick));
    repeat ch := readkey;
        writeln(' Bam phim ', ch);
    until ch = #27;
    setintvec($9,@kb);
end.

```

- Bài tập: Trong lúc thực hiện công việc giải phương trình bậc 2 (cụ thể trong lúc nhập dữ liệu) lúc người sử dụng bấm ESC thì thoát khỏi chương trình. Biết mã ASCII của ESC = 27 = 1B_h và mã scan = 1. Giả sử thủ tục giải phương trình bậc 2 Giai_PTB2 đã có sẵn.

```

uses dos,crt;
var
    kb: procedure;
    ch: char;
function check : boolean;
var
    tail: Word absolute $0040:$001C;
    x: word;
begin
    { bộ nhớ bàn phím từ 0040:001E - 0040:003D }
    x := tail - 2;
    if (x < $001E) then x := $003C;
    if (Mem[$0040 : x] = $1B) AND
        (Mem[$0040 : (x + 1) ] = $01) then
        check := true
    else
        check := false;
end;

{$F+}
procedure keyclick; interrupt;
begin
    inline($9C); { PushF push flags }
    kb;
    if(check) then
    begin
        writeln('Ban da bam Esc');
        setintvec($9,@kb);
        halt;
    end;
end;
{$F-}

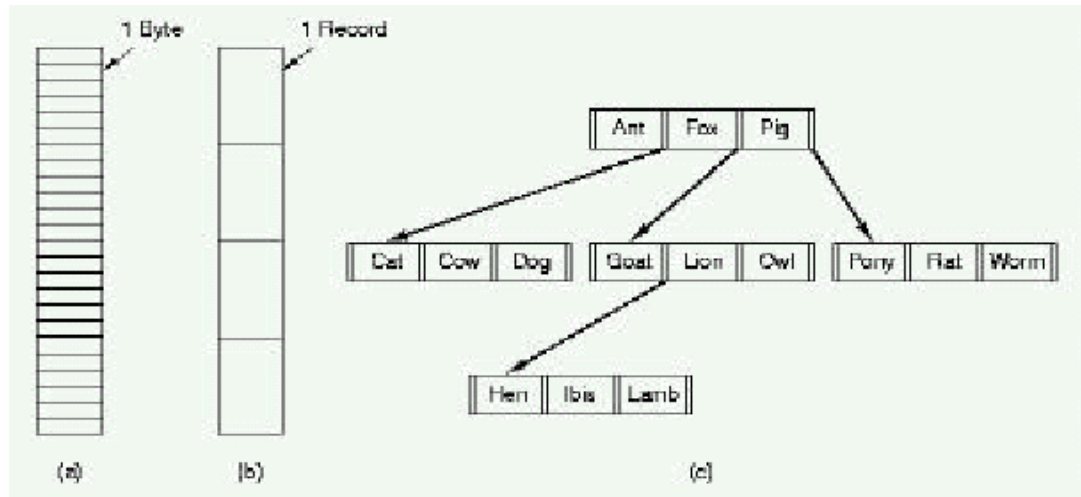
begin
    getintvec($9,@kb);
    setintvec($9,addr(keyclick));
    writeln('Chuong trinh bat phim');
    writeln('Bam Enter hoac ESC de thoat');
    repeat ch := readkey;
        writeln('Bam Enter hoac ESC de thoat');
    until ch = #13;
    writeln('Ban da bam Enter');
    setintvec($9,@kb);
end.

```

6. Quản lý tệp

a. Các khái niệm

- Tệp: Có hai đặc trưng của tệp
 - + Lưu trữ dữ liệu bộ nhớ ngoài: Tồn tại cho tới khi người sử dụng xoá khỏi bộ nhớ ngoài. Không bị mất dữ liệu khi tắt máy tính.
 - + Có nhiều người sử dụng: (nhu cầu chia sẻ tệp)
 - Tệp thực hiện: Nhiều người cùng dùng một ứng dụng nào đó
 - Tệp của nhân hệ điều hành:
 - Tệp văn bản: text, ảnh, âm thanh
 - Thư mục: gồm nhiều tệp
- Tệp gồm có Tên tệp và phần mở rộng (không bắt buộc) để
 - + Người sử dụng có thể hiểu được
 - + Phần mở rộng cho tệp dùng để
 - Nhóm các tệp cùng kiểu theo một quy ước
 - Tệp của hệ điều hành
 - + Biểu tượng cũng là thành phần của phần mở rộng giúp người sử dụng quản lý tốt hơn
- Thuộc tính của tệp
 - + Người sở hữu, nhóm sở hữu
 - + Thuộc tính ẩn, hệ thống, lưu trữ(archive), chỉ đọc
 - + Thời gian lần truy nhập cuối cùng, thời gian sửa đổi cuối cùng
 - + Quyền điều khiển, mật khẩu
 - + Kích thước hiện tại, kích thước tối đa
 - + Các ứng dụng có liên kết, vận hành (operation)
- Tổ chức tệp
 - + Tổ chức tuần tự theo byte: dữ liệu được tổ chức lưu trữ, đọc và ghi một cách tuần tự từng byte. Cách tổ chức này có tính vạn năng, mọi ứng dụng đều có thể sử dụng tệp.
 - + Tổ chức tuần tự theo bản ghi: dữ liệu được tổ chức lưu trữ, đọc và ghi một cách tuần tự từng bản ghi với kích thước cố định.
 - + Tổ chức cây các bản ghi: dữ liệu được tổ chức lưu trữ, đọc và ghi theo cây các bản ghi theo trường khoá.



- Truy nhập tệp
 - + Truy nhập tuần tự: Việc đọc / ghi theo thứ tự từ đầu tệp đến cuối tệp
 - + Truy nhập ngẫu nhiên: Có thể truy nhập tuần tự hoặc đọc / ghi theo bất kỳ trình tự nào.
- Kiểu tệp
 - + Tệp thông thường: Gồm các tệp text (dạng ASCII) và tệp nhị phân dùng để lưu trữ dữ liệu.
 - + Thư mục: Chứa một tập các tệp
 - + Tệp đặc biệt: được truy nhập bởi thiết bị
- Các thao tác với tệp
 - + Tạo tệp
 - + Xóa tệp
 - + Mở tệp
 - + Đóng tệp
 - + Đọc tệp
 - + Ghi dữ liệu (có thể ghi đè lên dữ liệu cũ)
 - + Mở rộng (ghi dữ liệu vào cuối tệp, không làm mất dữ liệu cũ)
 - + Di chuyển con trỏ để đọc/ghi dữ liệu
 - + Đọc/Thiết lập thuộc tính tệp
 - + Đổi tên tệp
- Cấu trúc thư mục
 - + Thư mục gốc: Dos (C:\), Unix (/)
 - + Đường dẫn

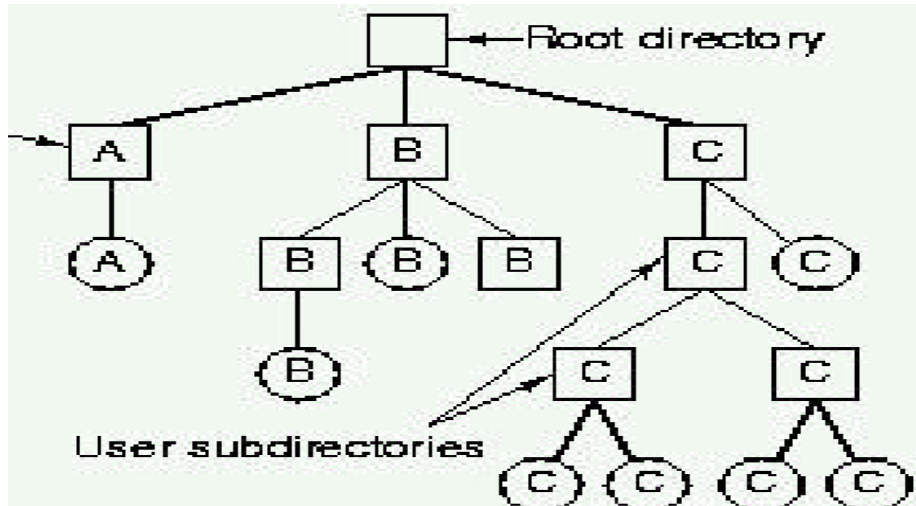
Dùng để chỉ ra nơi lưu trữ tệp

Đường dẫn tuyệt đối: chỉ ra tệp từ thư mục gốc

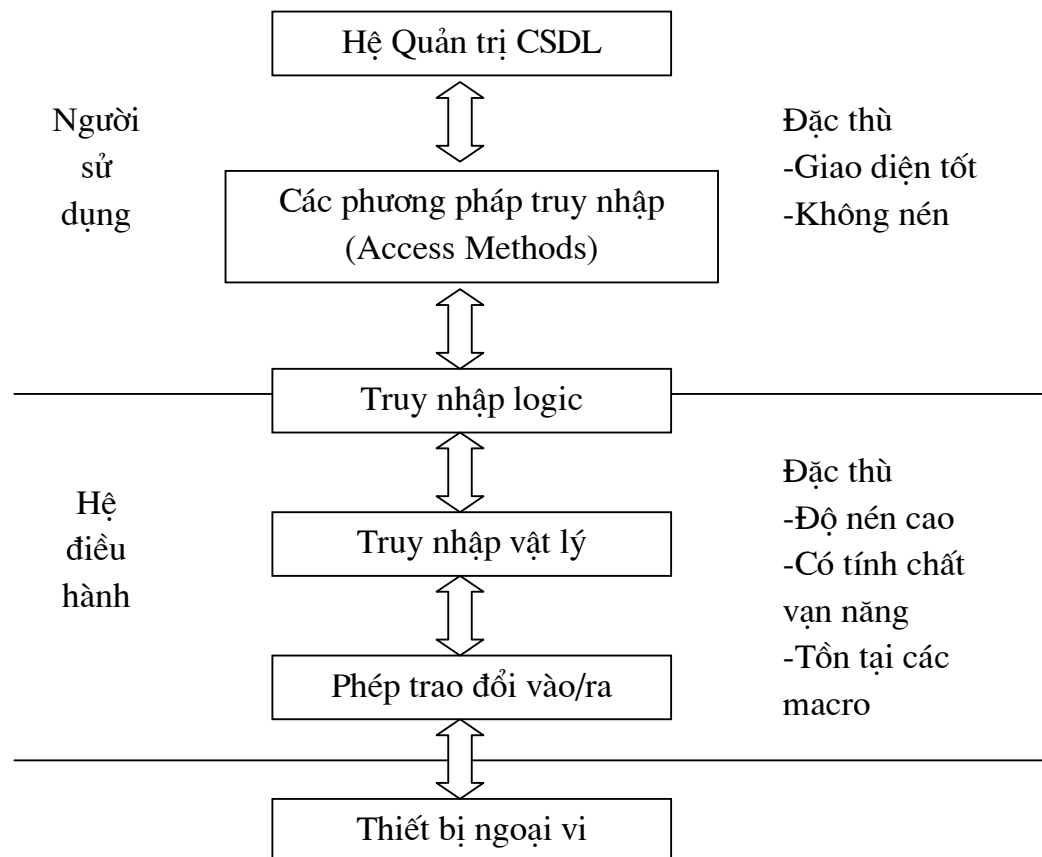
ví dụ: C:\TP\Bin\Turbo.exe

Đường dẫn tương đối: chỉ ra tệp từ thư mục hiện tại

Ví dụ: lệnh DIR ..\BGI (nếu đang ở trong C:\TP\Bin)



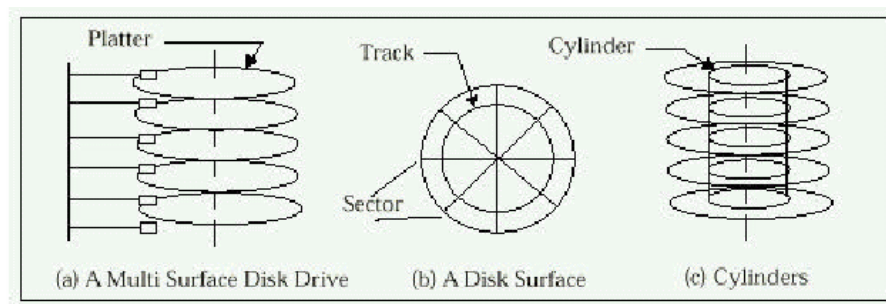
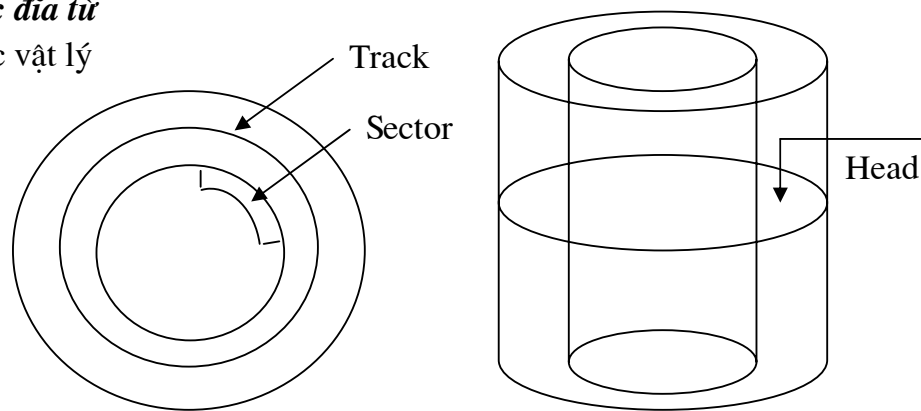
b. Trao đổi dữ liệu với tệp



7. Quản lý tệp trong MSDOS

a. Cấu trúc đĩa từ

- Cấu trúc vật lý



- + Dữ liệu được ghi trên đĩa theo các đường tròn đồng tâm gọi là track (rãnh) khi đĩa có nhiều mặt gọi là cylinder.
- + Mỗi cylinder được chia thành các sector (cung). Mỗi sector lưu trữ được một số các byte.
- + Đĩa có thể có 1, 2 hay nhiều mặt (side). Mỗi mặt được truy nhập bằng một đầu từ (head).
- Dung lượng của đĩa phụ thuộc vào số đầu từ, số cylinder, số sector trên một track và số byte trên một sector
- Ví dụ: đĩa có các thông số sau
 - Số byte / sector = 512
 - Số sector / track = 63
 - Số cylinder = 523
 - Số mặt = 128
 Ta có dung lượng đĩa bằng $523 \times 128 \times 63 \times 512 \approx 2 \text{ GB}$
- Ví dụ: đĩa có các thông số sau (cho bài tập tại lớp)
 - Số byte / sector = 512
 - Số sector / track = 18
 - Số mặt = 2
 - Số cylinder = 80
 Ta có dung lượng đĩa bằng $2 \times 80 \times 18 \times 512 \approx 1.44 \text{ MB}$

- Địa chỉ vật lý của một sector trên một đĩa được đặc trưng bởi (Cylinder, Head, Sector).
 - + Các cylinder được đánh số từ ngoài vào bắt đầu từ 0.
 - + Số hiệu đầu từ được đánh số từ 0.
 - + Các sector được đánh số từ 1.
- Sector logic

Sector có thể được truy nhập qua địa chỉ logic. Số hiệu sector logic được đánh số từ 0 bắt đầu từ cylinder 0, đầu từ 1, sector vật lý 1. Tiếp tục đánh số theo track và theo đầu từ cho mỗi cylinder

$$\text{relSec} = (\text{CylNo} * \text{SecsPerTrack} * \text{Heads}) + (\text{HeadNo} * \text{SecsPerTrack}) + (\text{SecNo} - 1)$$

$$\text{relSec} = 0 \cdot 63 \cdot 64 + 1 \cdot 63 + (1-1) = 63$$

b. phục vụ đọc/ghi đĩa

Ta sử dụng ngất 13H để đọc/ghi đĩa

- Đọc một Sector: Ta sử dụng ngắt 13h, hàm 02h

+ Input:

DL = Số hiệu đĩa (0 = A...; 80H = HD0; 81H = HD1)

DH = Số hiệu đầu từ

CH = track (cylinder) number ($0 \div n$)

CL = SỐ hiệu sector ($1 \div n$)

AL = Số sector cần đọc

ES:BX => địa chỉ vùng nhớ

Chú ý: Giá trị Sector gồm 6 bit và Cylinder là 10bit:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Cylinder										Sector					

Thủ tục mã hoá Cylinder và Sector

```
Function CylSecEncode(Cylinder, Sector : Word) : Word;
```

Begin

```
CylSecEncode := (Lo(Cylinder) shl 8)
```

```
or (Hi(Cylinder) shl 6) or Sector;
```

End;

+ Output:

Nếu có lỗi: Carry Flag=CY=1 và mã lỗi trong AH

Nếu không lỗi: $AH = 0$ và $ES:BX \Rightarrow$ địa chỉ vùng nhớ

Err#	Description
00H	no error on last operation
01H	bad command: invalid request to controller
02H	bad address mark
03H	write protect: attempted to write on write-protected diskette

```
| 04H | sector ID bad or not found. |
| 05H | reset failed |
| 08H | DMA failure |
| 09H | DMA overrun: attempted to write across a 64K-byte boundary. |
| 0bH | bad track flag encountered |
| 10H | bad CRC: invalid CRC when data checked. |
| 11H | data corrected: recoverable error found/corrected by ECC algorithm |
| 20H | controller failure |
| 40H | bad seek. requested track not found |
| 80H | time out. drive did not respond |
|0aaH | drive not ready |
|0bbH | undefined error |
|0ffH | sense operation failed |
+-----+
```

- Bảng mã lỗi

Giá trị	ý nghĩa
00	Thành công
01	Lệnh không hợp lệ
02	Không tìm thấy dấu địa chỉ trên đĩa.
03	Muốn ghi lên đĩa được bảo vệ chống ghi (M)
04	Không tìm thấy sector
05	Tái lập không được (C)
06	Đĩa mềm đã lấy ra (M)
07	Bảng tham số bị hỏng (C)
08	DMA chạy lỗi (M)
09	DMA ở ngoài phạm vi 46 K
0A	Cờ sector bị lỗi
10	CRC hay ECC lỗi
11	ECC đã điều chỉnh dữ liệu sai (C)
20	Lỗi do bộ điều khiển đĩa.
40	Lỗi không tìm được track
80	Lỗi hết thời gian
AA	ổ đĩa không sẵn sàng (C)
BB	Lỗi không xác định (C)
CC	Lỗi lúc ghi (C)
E0	Lỗi thanh ghi trạng thái (C)
FF	Thao tác dò thất bại

(C = chỉ dùng cho đĩa cứng; M = chỉ dùng cho đĩa mềm)

- Thủ tục đọc một sector

```

Type SectorType = array[0..511] of byte;

Function CylSecEncode(Cylinder, Sector : Word) : Word;
Begin
    CylSecEncode := (Lo(Cylinder) shl 8)
                    or (Hi(Cylinder) shl 6) or Sector;
End;

{ Nếu có lỗi trả về false, nếu không lỗi trả về true }
function ReadSector(var buf: SectorType;
    Drive, Cylinder, Head, Sector: Byte) : Boolean;
var r: registers;
begin
    fillchar(buf, sizeof(buf), $11);
    with r do begin
        dl := Drive;  dh := Head;
        cx := CylSecEncode(Cylinder, Sector);
        al := 1; { so sector }
        ah := 2; { doc o dia }
        bx := ofs(buf);  es := seg(buf);
    end;
    intr($13, r);
    if (r.ah <> 0) then ReadSector := false
    else ReadSector := true;
end;

```

- Ghi một Sector: Ta sử dụng ngắt 13h, hàm 03h, hàm này tương tự hàm 02h.
- Chú ý: Với ổ mềm thì động cơ cần có một thời gian khởi động để đạt tốc độ làm việc, nhưng thời gian chờ đợi của hàm đọc/ghi đĩa là rất nhỏ vì vậy cần phải thực hiện việc đọc/ghi vài lần trước khi khẳng định có lỗi vào/ra.
- Cluster gồm một nhóm sector liên tiếp nhau về mặt logic và phân phối bộ nhớ cho người sử dụng, số lượng sector cho một cluster thường là bộ số của hai và Cluster được đánh số từ 2 trở đi.

c. Cấu trúc thông tin Bảng tham số điều khiển

Bảng tham số điều khiển xác định toàn bộ vùng hệ thống: đối với đĩa mềm thì nằm ở (0, 0, 1) và đối với đĩa cứng thì nằm ở sector đầu tiên của mỗi đĩa logic (partition), ví dụ đối với đĩa logic thứ nhất là (0, 1, 1). Tùy theo khuôn dạng của bảng FAT mà hệ thống dành một số lượng sector khác nhau cho Boot Record. Boot Record của FAT 12 và FAT 16 chiếm 1 sector, còn Boot Record của FAT 32 thường chiếm 3 sector.

- Cấu trúc Boot Record của FAT 12 và FAT 16
 - + Phần 1: chứa tham số của đĩa từ

- + Phần 2: chứa chương trình môi (Boot Strap Loader), cần thiết cho đĩa khởi động

Tham số
Chương trình môi
55AA

Phần này có dấu hiệu là chữ ký gồm 2 byte 55AA.

- Cấu trúc cụ thể như sau:

No	Offset	Leng	ý nghĩa
1	0	3	JMP
2	3	8	Tên hệ thống format đĩa từ
3	B	2	Số byte trên một sector
4	D	1	Số sector / cluster
5	E	2	Ghi địa chỉ của FAT thứ nhất trong đĩa logic
6	10	1	Số lượng bảng FAT
7	11	2	Số phần tử tối đa ở thư mục gốc
8	13	1	= 0; <> 0 : Ghi số lượng sector của đĩa từ nếu < 32 MB
9	15	1	Byte Media: F8 nếu là HD, F0: FDD 1.44
10	16	2	Số sector dành cho một bảng FAT
11	18	2	Số sector dành cho một Track
12	1A	2	Số đầu từ
13	1C	4	Ghi địa chỉ tuyệt đối của Boot Sector trong đĩa vật lý
14	20	4	= 0; <> 0 : tổng số sector của đĩa >= 32 MB
15	24	1	Địa chỉ vật lý ổ đĩa từ A: 00H, C: 80H, D: 81H
16	25	1	Dự trữ
17	26	1	Dấu hiệu 29H
18	27	4	Ghi số hiệu của đĩa từ
19	2B	11 ₁₀	Tên đĩa từ
20	36	8	Dự trữ: FAT16 _{32 32 32} (32 là dấu cách)
21	3Eh	448	Đoạn mã thực hiện
22	1Feh	2	Chữ ký (55h AAh)

- Ví dụ:

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	
00:	EB	3C	90	4D	53	57	49	4E	34	2E	31	00	02	40	01	00	
	JMP				M S W I N 4 . 1								512		64	đ/c FAT	
10:	02	00	02	00	00	F8	D1	00	3F	00	40	00	3F	00	00	00	
	2 FAT	512 Root Dir				HD	209 Sec/ FAT		63 Sec/ Track		64 Head		63 hidden sectors (đ/c Boot Sector)				
20:	41	0C	34	00	80	00	29	1E	13	DA	13	20	20	20	20	20	
	3411009				C:		chữ ký	Serial Number				Tên đĩa từ (rỗng)					
30:	20	20	20	20	20	20	46	41	54	31	36	20	20	20			
	Tên đĩa từ (rỗng)						FAT16										

OEM ID	MSWIN4.1
Bytes per sector	512
Sectors per cluster	64
Reserved sectors at beginning	1
FAT Copies	2
Root directory entries	512
Total sectors on disk	(Unused)
Media descriptor byte	F8 Hex
Sectors per FAT	209
Sectors per track	63
Sides	64
Special hidden sectors	63: Boot Sector nằm (0,1,1)
Big total number of sectors	3411009
Physical drive number	128
Extended Boot Record Signature	29 Hex
Volume Serial Number	13DA131E Hex
Volume Label	(rỗng)
File System ID	FAT16

- Chương trình đọc Boot Sector

+ Kết quả

Thông tin về Boot Sector

Tên HDH : MSWIN4.1

Số byte / sector : 512

Số sector / cluster : 16

Số phần tử tối đa ở thư mục gốc : 512

Số bảng Fat : 2

Số byte / Fat : 0

Số sector / cylinder : 63

Số head : 255

+ Mã nguồn

```

uses crt,dos;
const hexs: array[0..15] of char = '0123456789ABCDEF';
Type SectorType = array[0..511] of byte;

function ReadSector(var buf: SectorType;
                    Drive, Cylinder, Head, Sector: Byte) : Boolean;
begin
end;

var
    buf: SectorType;
    i,j: word;
Begin
    clrscr;
    ReadSector(buf,$80,0,1,1); { Boot Sector }
    Writeln('Thong tin ve Boot Sector');
Write('Ten HDH : ');
for i := $03 to ($03 + 7) do write(chr(buf[i])); Writeln;
Write('So byte / sector : ', buf[$0B] + buf[$0C]*256);
Writeln;
Write('So sector / cluster : ', buf[$0D]); Writeln;
Write('So bang Fat : ', buf[$10]); Writeln;
Write('So phan tu toi da o thu muc goc : ', buf[$11] +
buf[$12]*256); Writeln;
Write('So byte / Fat : ', buf[$16] + buf[$17]*256); Writeln;
Write('So sector / cylinder : ', buf[$18] + buf[$19]*256);
Writeln;
Write('So head : ', buf[$1A] + buf[$1B]*256); Writeln;
    readkey;
end.

```

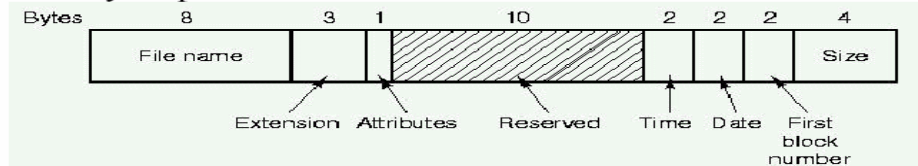
- Bài tập: Sao lưu Boot Sector đĩa A:

- + Lập trình đọc Boot Sector của đĩa A: và lưu ra tệp tại đĩa cứng
- + Khôi phục lại khi Boot Sector của A: khi bị hỏng Boot Sector
- Cấu trúc Boot Record của FAT 32 (cho tự nghiên cứu).

d. Thư mục gốc (Directory)

- Thư mục có hai loại:
 - + Thư mục gốc (Root): là thành phần thiết yếu ở mọi hệ điều hành.
 - + Thư mục con (SubDirectory)
- Thư mục bao gồm các phần tử, mỗi phần tử nếu được sử dụng sẽ tương ứng với một tệp trong thư mục.
- Trong MSDOS thì mỗi phần tử gồm 32 byte được chia thành 8 trường
- Cấu trúc một phần tử trong thư mục (32 byte)

Directory Implementation



No	Offset	Leng	ý nghĩa
1	0	8	Tên tệp (nếu nhỏ 8 ký tự thì lấy ký tự <space> điền vào)
2	8	3	Phần mở rộng
3	B	1	Thuộc tính tệp attribute
4	C	10 ₁₀	Dự trữ
5	16 _H	2	Thời gian (Giờ, Phút, Giây) tạo/cập nhật cuối cùng
6	18	2	Ngày (Năm, Tháng, Ngày) tạo/cập nhật cuối cùng
7	1A	2	Starting Cluster: ghi cluster đầu tiên của tệp
8	1C	4	Kích thước tệp (tính theo byte)

Directory Entry Layout		
Offset	Size	Contents
+0	8	'F' 'I' 'L' 'E' 'N' 'A' 'M' 'E' left-justified, blank-padded
+8	3	'E' 'X' 'T' left-justified, blank-padded
+0bH	1	atr file attribute
+0cH	0aH	reserved
+16H	2	time time created or last modified in filetime format
+18H	2	date date created or last modified in filetime format
+1aH	2	Starting Cluster ghi cluster đầu tiên của tệp (link into FAT)
+1cH	4	file size file size in bytes

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Giờ					Phút					Giây (bước 2)					

+ Thời gian (năm, tháng, ngày)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Số năm kể từ năm 1980						Tháng					Ngày				

```

+-----+
|                                     |
|                                     |
|                                     |
| 1 1 1 1 1 1                      | Load word at DIR_ENTRY+16H into a | | | |
|+5-4-3-2-1-0-9-8-7-6-5-4-3-2-1-0+ | 16-bit register and perform mask  |
| |   hour   |   minute   |   sec/2   | and shift operations to extract   |
|+-----+ +-----+ +-----+ | components of the file time.      |
| +-----+ +-----+ +-----+ |                                     |
| |           |           |           | 5+---> 2-second increments (0-30 valid) (t & 001fH) |
| |           |           |           | 6+-----> minute (0-59 valid) ((t & 07e0H) >> 5) |
| |           |           |           | 5+-----> hour (0-23 valid) ((t & 0f800H) >> 11) |
|                                     |                                     |
| 1 1 1 1 1 1                      | Use word at DIR_ENTRY+18H. After 16-bit |
|+5-4-3-2-1-0-9-8-7-6-5-4-3-2-1-0+ | masks and shifts, add 1980 (07bcH) to year. |
|+-----+ +-----+ +-----+ |                                     |
| |           |           |           | 5+---> day (0-31) (d & 001fH) |
| |           |           |           | 4+-----> month (1-12 valid) ((d & 01e0H) >> 5) |
| |           |           |           | 7+-----> year (0-127 valid) ((d & f800H) >> 9) |
| |           |           |           | => từ 1980 đến 2107 |
|                                     |                                     |
+-----+

```

+ Ví dụ: C0 B2 => Time = C0B2 = 10110 010110 00000

Giờ 10110 = 22 => 10 chiều

Phút 010110 = 22

Giây 00000 = 0

Kết quả cho ta thời gian lần sửa cuối cùng (hoặc tạo) là 10:22 pm

+ Ví dụ: 97 26 => Date = 2697 = 0010011 0100 10111

Năm 1980 + 0010011 = 1980 + 19 = 1999

Tháng 0100 = 4

Ngày 10111 => ngày 23

Kết quả cho ta ngày của lần sửa cuối cùng (hoặc tạo) là 4-23-99

- Bài tập về nhà

Chuyển đổi giá trị thời gian gồm: ngày, tháng, năm, giờ, phút, giây thành 4 byte có định dạng giống định dạng trên và ngược lại.

- Ví dụ:

Đọc thư mục gốc của đĩa A: biết số phân tử tối đa là 224, số sector dành cho thư mục gốc là 14 ($224 \cdot 32 / 512 = 224 / 16 = 14$ hay cho al := 14)

```
uses crt,dos;
```

```
type
```

```

    elmt = record
      filename: array[0..7] of char;
      ext: array[0..2] of char;
      attb: byte;
      reserved: array[0..9] of byte;
      Dt: longint;
      firstCluster: word;
      size: longint;
    end;
  procedure show(e: elmt);
  var
    i: byte;
    t: DateTime;
  begin
    if(e.firstCluster <> 0) then
      { phải kiểm tra vì dưới Win98 thì có sự mở rộng cho tên
      tệp dài, vì thế có nhiều phần tử phụ và có thành phần
      firstCluster = 0 }
      with e do begin
        for i := 0 to 7 do write(filename[i]); write('.');
        for i := 0 to 2 do write(ext[i]); write(' ':8);
        { procedure UnpackTime(DT: Longint; var T: TDateTime); }
        { DateTime = record
          Year,Month,Day,Hour,Min,Sec: Word;
        end;
        }
        UnpackTime(dt,t);
        with t do begin
          write(Day:2, '/', Month:2, '/', Year:4); write(' ':8);
          write(Hour:2, ':', Min:2, ':', Sec:2);
        end;
        write(size: 10); writeln;
      end;
    end;
  end;

  Type RootType = array[0..223] of elmt;

  Function CylSecEncode(Cylinder, Sector : Word) : Word;
  Begin
    CylSecEncode := (Lo(Cylinder) shl 8)
                   or (Hi(Cylinder) shl 6) or Sector;
  End;

  function ReadSector(var buf: RootType;
    Drive, Cylinder, Head, Sector: Byte) : Boolean;
  var r: registers;
  begin

```

```

fillchar(buf,sizeof(buf),$11);
with r do begin
  dl := Drive;  dh := Head;
  cx := CylSecEncode(Cylinder, Sector);
  al := 14; { so sector = 14 vi 224 * 32 = 14 * 512}
  ah := 2; { doc o dia }
  bx := ofs(buf);  es := seg(buf);
end;
intr($13,r);
if(r.ah <> 0) then ReadSector := false
else ReadSector := true;
end;
var
  t: Datetime;
  buf: RootType;
  i,j: word;
{
DateTIme = record
  Year,Month,Day,Hour,Min,Sec: Word;
end;
}
begin
  clrscr;
  writeln('Cho dia A:'); readln;
  ReadSector(buf,0,0,1,2);
  while(buf[i].filename[0] <> #0) do
    begin
      show(buf[i]);
      i := i + 1;
    end;
  readkey;
end.

```

+ Kết quả:

Cho dia A:

A	.TXT	21/ 9/2001	8:46:18	16
FUZZY	.TXT	19/ 6/2001	16:37:38	2444
TMP	.VN	6/ 1/2001	17: 5:20	34

e. FAT (File Allocation Table)

- Vai trò của FAT
 - + Quản lý bộ nhớ phân phối cho từng tệp
 - + Quản lý bộ nhớ tự do
 - + Quản lý vùng bộ nhớ chất lượng kém (không cho hệ thống ghi thông tin vào).
- Bảng FAT bao gồm các phần tử, mỗi phần tử có thể là 12 bit hay 16 bit. Các phần tử từ phần tử thứ hai trở đi, mỗi phần tử tương ứng với một cluster trên đĩa và ngược lại. Trường hợp có một sector bị kém thì hệ thống sẽ đánh dấu các phần tử của cluster đó là FF7 (đối với FAT 12) hoặc là FFF7 (đối với FAT 16). Chỉ cần có một sector có chất lượng kém thì cả cluster bị đánh dấu là có chất lượng kém.

Phần tử 0: Fxx (đối với FAT 12) và FFxx (đối với FAT 16)

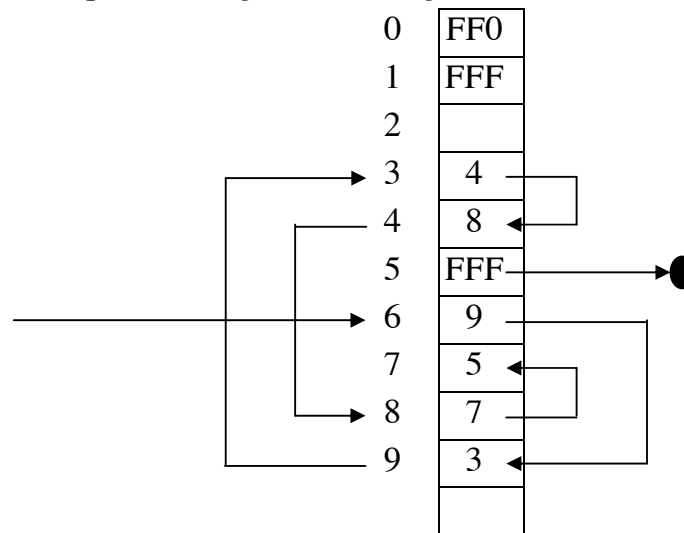
trong đó xx là 1 byte gọi là byte Media cho biết kiểu đĩa từ.

	Media	Kiểu đĩa từ
1	F9	đĩa mềm 1.2 M
2	F0	đĩa mềm 1.4 M
3	F8	đĩa cứng

Nhắc lại phần tử thứ 7 (có địa chỉ offset là 1A với kích thước 2 byte trong 1 phần tử của thư mục) ghi Starting Cluster, tức là cluster đầu tiên của tệp. Từ nội dung của phần tử trong bảng FAT tương ứng ta có thể tìm được phần tử tiếp theo hay chính là cluster tiếp theo dùng để lưu trữ tệp.

Các phần tử trong bảng FAT tạo thành một danh sách móc nối và phần tử cuối cùng của danh sách có giá trị là FFF (đối với FAT 12) và FFFF (đối với FAT 16). Vì các phần tử tạo thành danh sách móc nối nên chúng không nhất thiết phải nằm cạnh nhau.

Ví dụ: tệp f1.txt có giá trị Starting Cluster = 6



Như vậy tệp được lưu ở các cluster sau: 6->9->3->4->8->7->5.

Khi hệ thống muốn đọc nội dung tệp: Hệ thống bắt đầu đọc từ cluster số 6 và đọc phân tử tương ứng trong bảng FAT, nội dung của phân tử này chỉ cho hệ thống biết phải đọc cluster số 9, tiếp tục như vậy cho đến phân tử số 5. Giá trị của phân tử số 5 là FFF đây là dấu hiệu kết thúc và dừng lại.

Để tăng tốc độ truy nhập, ngay lần truy nhập đầu tiên làm việc với đĩa từ, hệ thống sẽ đọc luôn FAT và ROOT vào RAM, như vậy hệ thống chỉ cần truy nhập vào bộ nhớ để lấy thông tin, không cần phải truy nhập lại đĩa từ do vậy tăng được tốc độ và giảm được di chuyển cơ khí của đầu từ.

- Truy nhập bảng FAT 16:

Mỗi phân tử của FAT 16 là một word, vì vậy việc đọc/ghi mỗi phân tử là khá dễ dàng.

0		1		2		3		4		5		6		7	
F8	FF	FF	7F	15	01	FF	FF	17	00	FF	FF	FF	FF	B6	00
				277		<EOF>				<EOF>		<EOF>			
8		9		10		11		12		13		14		15	
25	00	2C	00	14	17	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
37		44		5908		<EOF>		<EOF>		<EOF>		<EOF>		<EOF>	

- Truy nhập bảng FAT 12:

Mỗi phân tử của FAT 12 là một 1.5 word, vì vậy việc đọc/ghi mỗi phân tử là phức tạp hơn. Hai phân tử $2n$ và $2n + 1$ của FAT được lưu trữ trong 3 byte.

+ Ví dụ: ghi 2 phân tử

$2n$: $123_h \Rightarrow 0123 \Rightarrow \text{đảo} \Rightarrow 2301$

$2n + 1$: $456_h \Rightarrow 4560 \Rightarrow \text{đảo} \Rightarrow 6045$

ghi vào đĩa 236145

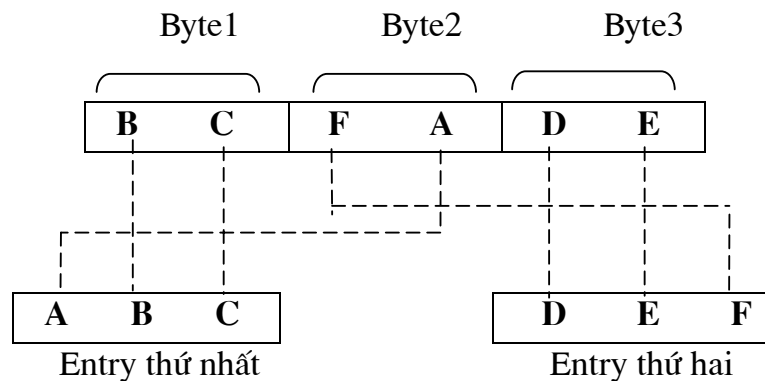
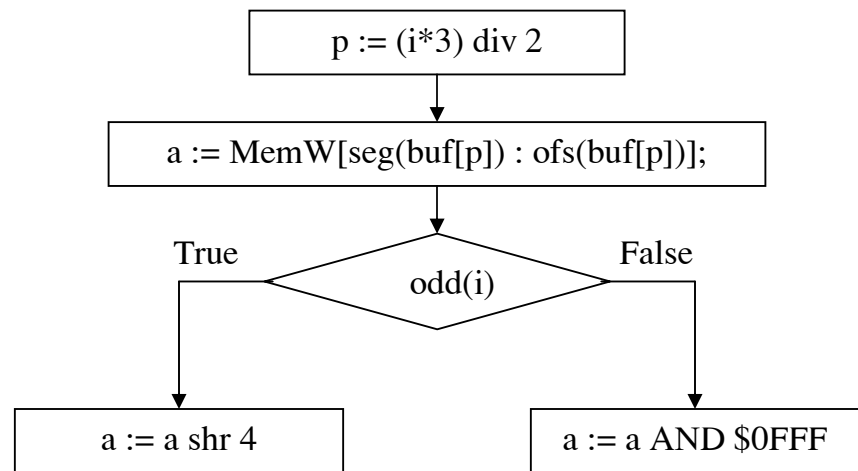
+ Ví dụ: đọc 2 phân tử

đọc 3 byte, ta được 123456

$2n$: $1234 \Rightarrow \text{đảo} \Rightarrow 3412 \Rightarrow \text{AND } 0FFF \Rightarrow 412$

$2n + 1$: $3456 \Rightarrow \text{đảo} \Rightarrow 5634 \Rightarrow \text{AND } FFF0 \Rightarrow 563$

+ Giải thuật đọc phân tử thứ i vào mảng B: `array[0..xxx]` of byte



- Ví dụ:

Giá trị các byte của 50 phân tử đầu tiên của Fat 12 là

F0	FF	FF	03	40	00	05	60	00	07	80	00
09	A0	00	0B	C0	00	0D	E0	00	0F	00	01
11	20	01	13	40	01	15	60	01	17	80	01
19	A0	01	1B	C0	01	1D	E0	01	1F	00	02
21	20	02	23	40	02	25	60	02	27	80	02
29	A0	02	2B	C0	02	2D	E0	02	2F	00	03
31	20	03									

Fat 12 ở hệ 16

FF0	FFF	003	004	005	006	007	008
009	00A	00B	00C	00D	00E	00F	010
011	012	013	014	015	016	017	018
019	01A	01B	01C	01D	01E	01F	020
021	022	023	024	025	026	027	028
029	02A	02B	02C	02D	02E	02F	030
031	032						

Giá trị phần tử thứ 2 và thứ 3 được lưu trữ 3 byte 3,4 và 5 là 03 40 00 tính ra được 003 và 004.

- Ví dụ: Lập trình đọc và đưa ra màn hình 50 phần tử đầu tiên của bảng FAT 12 trên đĩa mềm 1.44 MB

```

uses crt,dos;
const
  hexs: array[0..15] of char = '0123456789ABCDEF';
var
  buf : array[0..511] of byte;
  r: registers;
  i,p,k1,k2,k3,a: word;
begin
  clrscr; fillchar(buf,sizeof(buf),$11);
  writeln('Chương trình đưa ra 50 phần tử đầu tiên của FAT 12');
  writeln('Cho đĩa A: và bấm một phím bất kỳ');
  readkey;
  for i:=1 to 10 do begin
    with r do begin
      dl := 0; { 00H: đĩa A }
      dh := 0; { head = 0 }
      cx := 2; { Fat ở sector 2, cylinder 0 }
      al := 1; { số sector }
      ah := 2; { đọc ở đĩa }
      bx := ofs(buf);
      es := seg(buf);
    end;
    intr($13,r);
  end;

  if(r.ah <> 0) then begin
    writeln('Có lỗi đọc đĩa ');
    exit;
  end;

  writeln('GT các byte của 50 phần tử đầu tiên của Fat 12 là');
  for i:=0 to 74 do { 50 phần tử * 3 / 2 = 75 }
  begin
    if(i mod 8 = 0) then writeln; { 8 * 3/2 = 12 }

```



```

        k1 := buf[i] shr 4;
        k2 := buf[i] and $0F;
        write(hexs[k1]:3, hexs[k2]);
    end;
    writeln;

    writeln('Fat 12 o he 16');
    for i:=0 to 49 do
    begin
        if(i mod 8 = 0) then writeln; { 8 * 3/2= 12 }
        p := i*3 div 2;
        a := MemW[seg(buf[p]) : ofs(buf[p])];
        if odd(i) then a := a shr 4
        else a := a and $0FFF;
        k1 := a shr 8;
        k2 := (a and $00F0) shr 4;
        k3 := a and $000F;
        write(hexs[k1]:3,hexs[k2],hexs[k3]);
    end;

    readkey;
end.
```

f. Master Boot

Là sector đầu tiên của đĩa cứng (cylinder 0, head 0, sector 1) và được nạp vào bộ nhớ tại 0000: 7000 để thực hiện.

- Cấu trúc Master Boot
 - + Phần 1: Chương trình nhận biết cấu trúc
 - + Phần 2: Bảng phân chương (Partition Table)

Chương trình nhận biết cấu trúc	
Bảng phân chương	
	55AA

Phần này có dấu hiệu là chữ ký gồm 2 byte 55AA.

Đĩa cứng được chia thành các phần, mỗi phần được sử dụng như một đĩa từ độc lập gọi là đĩa logic. Thông tin về các phần này nằm trong bảng phân chương.

- Cấu trúc bảng phân chương: bảng phân chương được bắt đầu từ địa chỉ 1BE gồm 4 phần tử mỗi phần tử 16 byte.

Offset	Size	Contents
+0	1beH	code to load and execute boot sector of active partition
+1beH	10H	partition 1 entry (see below)
+1ceH	10H	partition 2 entry
+1deH	10H	partition 3 entry
+1eeH	10H	partition 4 entry
+1feH	2	55 aa partition table signature (0aa55H)

Mỗi phần tử chia thành 4 trường, mỗi trường 4 byte

15-12	11-8	7-4	3-0
Địa chỉ vật lý đầu	Địa chỉ vật lý cuối	Địa chỉ logic đầu	Tổng số sector

- + Địa chỉ vật lý đầu

3	2	1	0
Sys	Head	Sec	Cyl

Sys = 00 thì đây là đĩa logic chứa dữ liệu

Sys = 80 thì đây là đĩa logic chứa hệ thống

Head: đầu từ đầu tiên

2 phần tử Sec và Cyl được phân phối như sau

15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
Cyl				Sec				Cyl							

Như vậy số bit dành cho trường Cyl là 10

và dành cho sec là 6.

Ví dụ: Ta có 2 byte là FF và 4D ta phải tính Sec=?, Cyl=?
 Kết quả cho ta Sec = 63 và Cyl = 845

15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
FF								4D							
1	1	1	1	1	1	1	1	0	1	0	0	1	1	0	1
Cyl		63						Cyl							
Cyl = 11 0100 1101 = 845 ₁₀															

+ Địa chỉ vật lý cuối

3	2	1	0
Type	Head	Sec	Cyl

Type = 00 : cấm truy nhập

Type = 01 : hệ thống Dos 12 (dùng cho FAT 12)

Type = 04 : hệ thống Dos 16 (cho đĩa <= 5M)

Type = 06 : hệ thống Dos 16 (cho đĩa >= 5M)

Type = 0C : hệ thống Dos 32

Type = 50 : hệ thống DM (Disk Management) Readonly

Type = 51: hệ thống DM (Disk Management) Read/Write

Type = 05 : Mở rộng, dùng để tạo ra đĩa logic và cấu trúc của đĩa logic giống như cấu trúc của đĩa vật lý, nhưng các thành phần con của đĩa mở rộng (05) thì không được dùng làm đĩa khởi động.cc

Type = 0F : Mở rộng.

Extended DOS Partition															

_ Each drive table can also contain another type-5 entry															
to point to yet another drive table:															
+-----+ 0M															
+----- master boot sector Volume #1															
+-----															
+-> DOS boot sector (drive C:)															
\ fat, dir & data \															
+----> ----- 32M															
+----- Extension table Volume #2															
+-----															
+-> DOS boot sector (drive D:)															
\ fat, dir & data \															
+----> ----- 64M															
Extension table Volume #3															

```
relSec = (CylNo*SecsPerTrack*Heads)+(HeadNo*SecsPerTrack)+(SecNo - 1)
relSec = 0*63*64 + 1*63 + (1-1) = 63
Dung lượng đĩa 3411009*512 ≈ 1.6 GB
```

```
function ReadSector(var buf: SectorType;
    Drive, Cylinder, Head, Sector: Byte) : Boolean;
{ đã cho chép rồi }
{ phần chương trình chính }
var
    buf: SectorType;
    i,j: word;
    kt, kti: longint;
Begin
    ReadSector(buf,$80,0,0,1);
    kt := 0;
    for i := 0 to 3 do begin
        j := $1CA + i*16;
{ 1CA là vị trí kích thước phần tử đầu tiên }
        kti := MemL[seg(buf[j]):ofs(buf[j])];
        kt := kt + kti;
    end;
    kt := kt div 2048; { MB, 1 sec = 512 byte }
    writeln('Kích thước đĩa tu là ', kt, ' MB');
    readkey;
end.
```

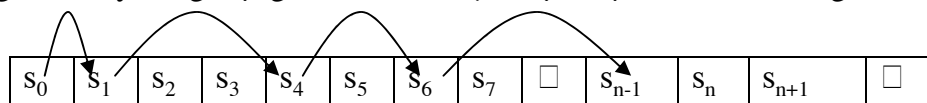

Chương 4. Quản lý tiến trình

1. Định nghĩa tiến trình

Tất cả các máy tính hiện đại đều có thể thực hiện nhiều việc cùng một lúc. Trong khi thực hiện chương trình của người sử dụng, máy tính có thể đọc dữ liệu từ đĩa và đưa ra màn hình hoặc máy in. Trong môi trường đa chương trình (multiprogramming system), một CPU có thể chuyển từ chương trình này sang chương trình khác, thực hiện mỗi chương trình trong khoảng 1% hoặc 1/10 mili giây. Nếu nói chính xác, thì tại một thời điểm, CPU chỉ thực hiện được một chương trình. Nhưng nếu xét trong khoảng thời gian phần trăm giây thì CPU có thể thực hiện nhiều công việc.

– Định nghĩa

Tiến trình là một dãy các trạng thái của hệ thống tính toán và việc chuyển từ trạng thái này sang trạng thái khác được thực hiện theo 1 chương trình nào đó.

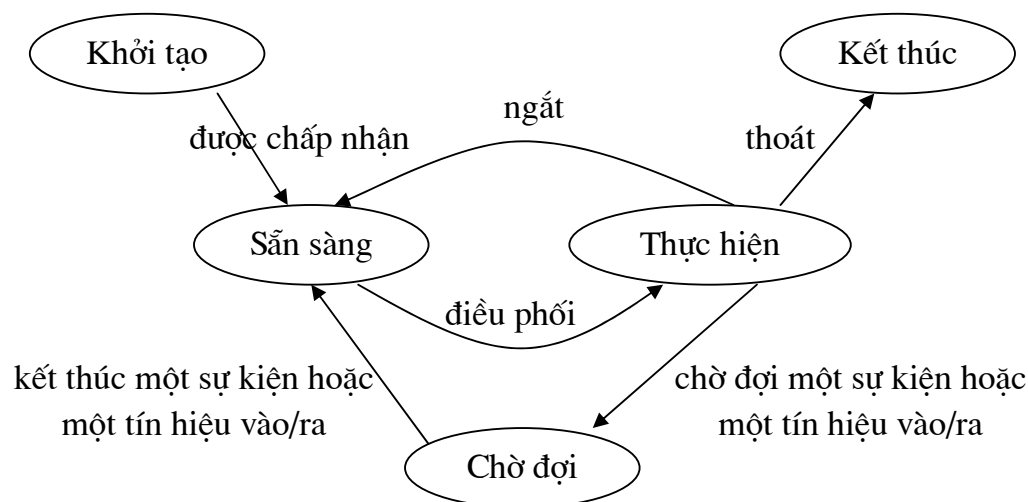


Các trạng thái này không nhất thiết phải liên tiếp nhau.

- + Nếu chương trình của hệ thống thì cho ta tiến trình hệ thống.
- + Nếu chương trình của người sử dụng thì cho ta tiến trình của người sử dụng.

Hiểu một cách thông thường ta có thể coi tiến trình là một chương trình đang được thực hiện.

– Ví dụ:



- + Khởi tạo: Tiến trình đang được tạo ra.
- + Sẵn sàng: Tiến trình chờ để kết nối vào processor.
- + Thực hiện: Các lệnh đang được thực hiện.
- + Chờ đợi: Tiến trình chờ một sự kiện vào/ra hoặc chờ nhận một tín hiệu nào đó.
- + Kết thúc: Tiến trình kết thúc thực hiện.

2. Khối điều khiển tiến trình (Process Control Bloc - PCB)

- Mỗi tiến trình được biểu diễn trong hệ điều hành bởi một khối điều khiển tiến trình gồm có
 - + Trạng thái tiến trình.
 - + Lệnh máy: máy tính chỉ ra địa chỉ lệnh máy đầu tiên trong tiến trình.
 - + Bộ thanh ghi.
 - + Thông tin về lịch trong bộ điều khiển CPU: bao gồm thứ tự ưu tiên của tiến trình, các tham số để lập lịch.
 - + Thông tin về bộ nhớ.
 - + Thông tin tính toán: gồm thời gian chiếm giữ processor, thời gian thực tế, giới hạn về thời gian, số lượng công việc.
 - + Thông tin trạng thái các cổng vào/ra.

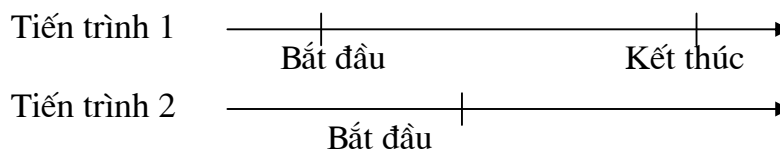
3. Cách thực tiến trình

a. Thực hiện tuần tự

Khi hệ thống kết thúc một tiến trình thì hệ thống mới chuyển sang tiến trình khác. Thực hiện tuần tự không phải là đối tượng nghiên cứu của chúng ta.

b. Thực hiện song song

Hai tiến trình được gọi là song song nếu thời điểm bắt đầu của một tiến trình nằm giữa thời điểm bắt đầu và kết thúc của tiến trình kia.

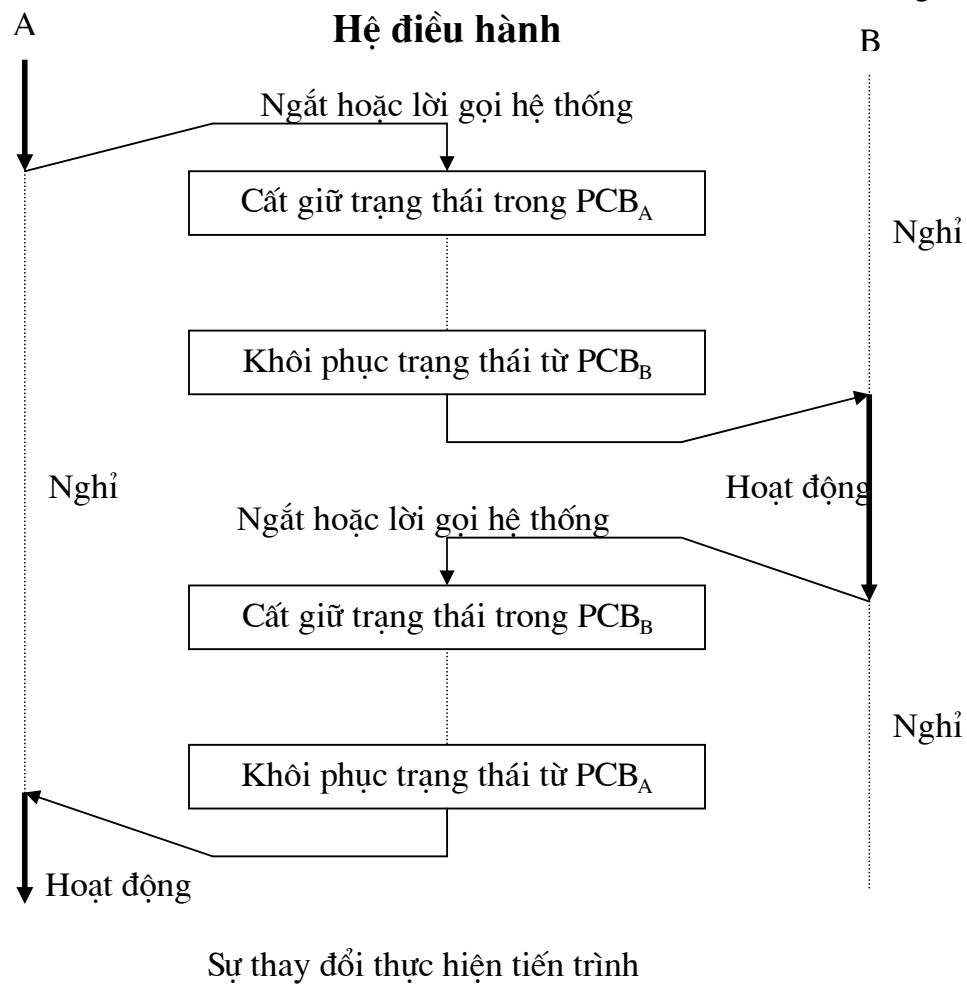
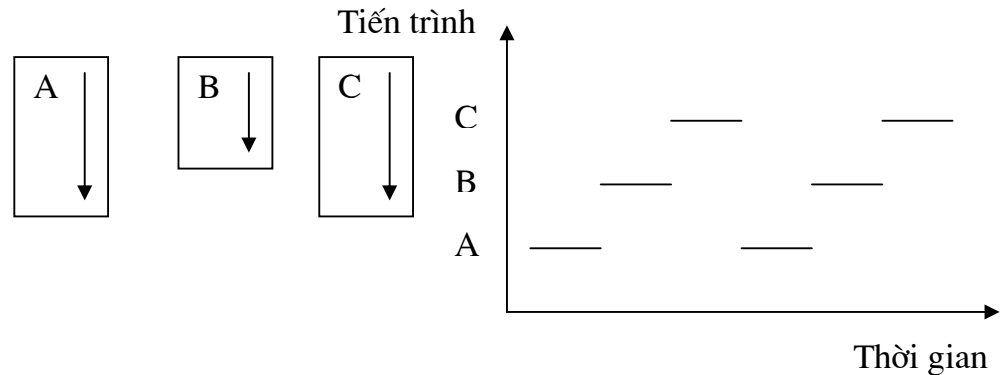


- Thực hiện song song vật lý: cùng một thời điểm 2 tiến trình cùng được thực hiện.

Các điểm cần chú ý:

- + Loại này chỉ có thể thực hiện ở trong chế độ nhiều processor.

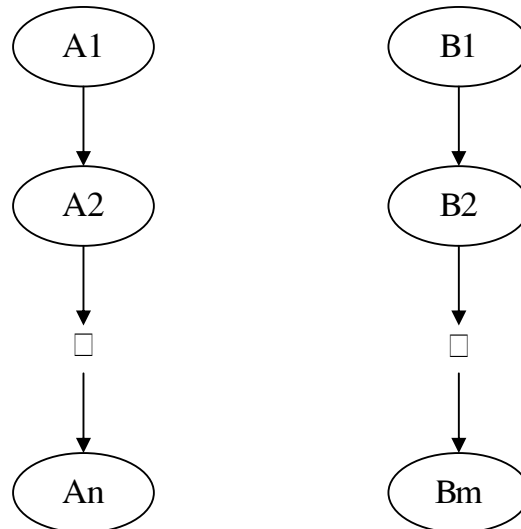
- + Hai tiến trình song song vật lý có thể sử dụng song song thiết bị ngoại vi và processor do đó cách làm việc của hệ thống hoàn toàn khác so với chế độ đơn processor.
- Thực hiện song song đan xen
 - Để nâng cao hiệu quả của processor, các tiến trình lần lượt được phục vụ đan xen lẫn nhau.



4. Phân loại tiến trình song song

a. Độc lập

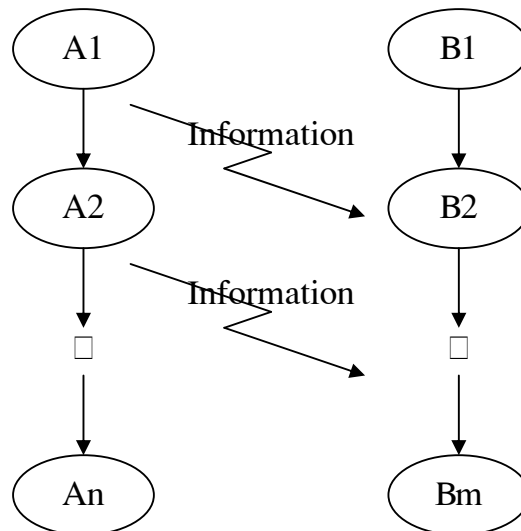
Hai tiến trình song song được thực hiện riêng rẽ không có quan hệ với nhau.



Hệ thống phải có cơ chế bảo vệ để tiến trình này không làm ảnh hưởng đến tiến trình khác.

b. Quan hệ thông tin

Hai tiến trình A và B được gọi là có quan hệ thông tin với nhau nếu tiến trình này có gửi thông báo cho tiến trình kia. Tiến trình gửi thông báo có thể không cần biết tiến trình nhận có tồn tại hay không? ở đâu? và đang ở giai đoạn nào?



Các phương pháp tổ chức lưu trữ các thông báo:

– Sử dụng bộ nhớ

Hệ thống sẽ sử dụng một phần bộ nhớ để lưu trữ các thông báo. Mỗi tiến trình cần nhận thông báo chỉ việc rà soát trong “hòm thư” của hệ thống.

- + Ưu điểm: lưu trữ được lượng thông tin lớn với thời gian lưu trữ lâu.
- + Nhược điểm: tính thụ động cao.

– Gửi thông báo qua cổng vào/ra

- + Ưu điểm: các tiến trình có thể dễ dàng lấy thông tin từ cổng mà không bị hàng rào bộ nhớ ngăn cản.
- + Nhược điểm: dung lượng thông tin chứa ở các cổng không lớn, thời gian lưu trữ thông báo bị hạn chế.

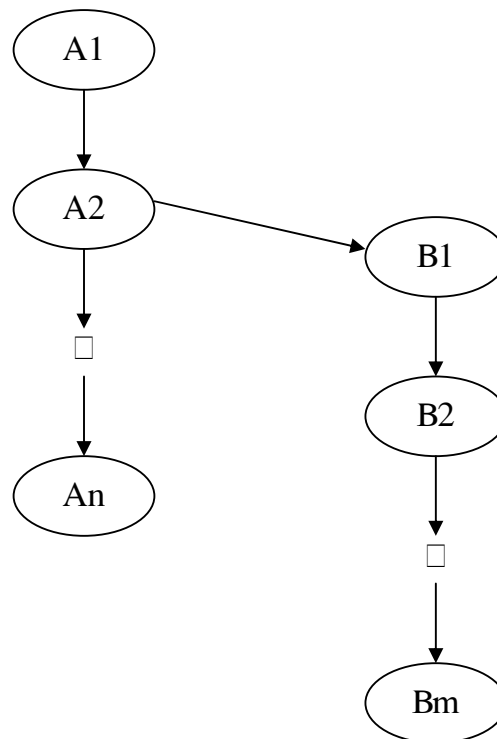
– Sử dụng chương trình thư ký (Monitor)

Chương trình thư ký (Monitor) là chương trình của hệ thống, nó được cung cấp mọi thông tin nhưng không có khả năng điều khiển hệ thống. Thông qua chương trình này, tiến trình có thể dễ dàng xác định được tiến trình kia ở đâu.

- + Ưu điểm: Tính chủ động cao.

c. Loại song song phân cấp

Là loại tiến trình mà trong quá trình hoạt động nó sản sinh ra một tiến trình nữa hoạt động song song với chính nó.



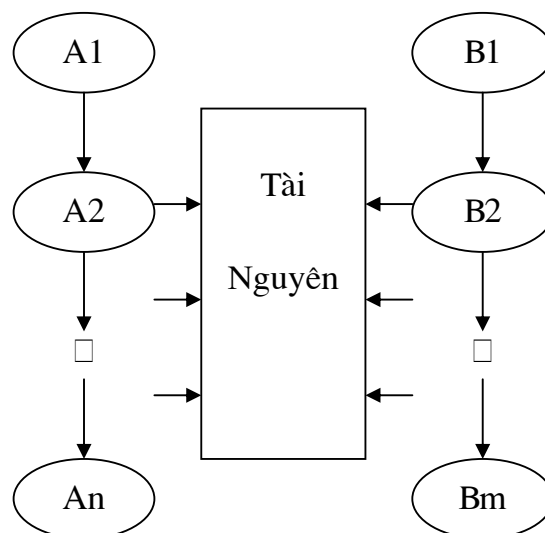
Khi tiến trình con đã hoạt động thì hai tiến trình này không biết gì về nhau

- Tài nguyên của tiến trình con có thể lấy từ vốn tài nguyên của hệ thống hoặc lấy từ vốn tài nguyên của tiến trình chính.
- + Nếu lấy tài nguyên từ vốn tài nguyên của hệ thống thì hệ thống có thể quản lý tài nguyên tập chung. Như vậy sẽ tối ưu hoá được việc sử dụng tài nguyên, nhưng việc quản lý này rất phức tạp.
- + Nếu tiến trình con lấy từ vốn tài nguyên của tiến trình chính thì ta có hệ quản lý tài nguyên phân tán. Loại tài nguyên này đơn giản, nhưng không có khả năng khai thác tối ưu tài nguyên hệ thống.

Trong mọi trường hợp nếu tài nguyên lấy ở đâu thì phải trả về đó, vì vậy tiến trình chính thường sử dụng các lệnh chờ POS hoặc WAIT để các tiến trình con kịp trả lại tài nguyên.

d. Tiến trình đồng mức

Hai tiến trình được gọi là đồng mức nếu có thể sử dụng chung tài nguyên theo nguyên tắc lần lượt.



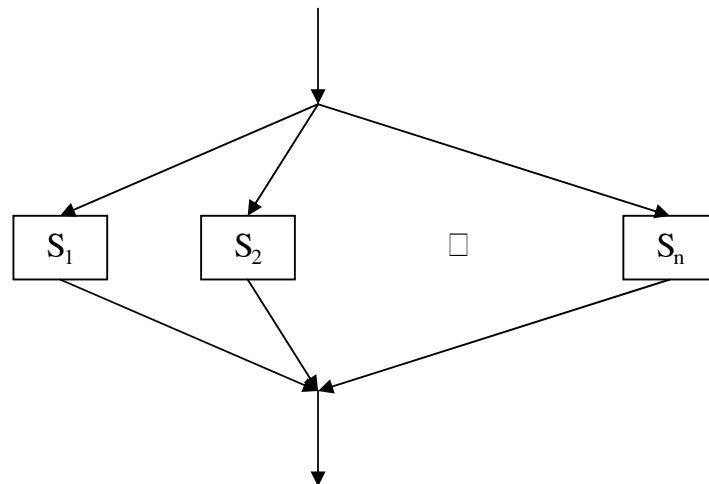
Hai tiến trình này không phân biệt tiến trình chính và tiến trình con, mà là hai tiến trình độc lập. Mỗi tiến trình sau khi sử dụng tài nguyên thì phải trả lại cho hệ thống và tiếp tục hoạt động độc lập.

Ví dụ: chương trình chơi cờ: Tài nguyên chung là bàn cờ. Giả sử đến lượt tiến trình thứ nhất, tiến trình thứ nhất chiếm tài nguyên để chơi, khi ra quyết định xong thì trả lại bàn cờ cho hệ thống. Tiến trình thứ hai phải kiểm tra xem tiến trình thứ nhất đã đi chưa? nếu xong rồi thì mới đến lượt nó (thực hiện như tiến trình thứ nhất).

5. Mô tả tiến trình song song

Ta dùng ký pháp nhân tạo

Giả sử cần thực hiện một tập các khối lệnh song song s_1, s_2, \dots, s_n



Ta đưa vào trong một khối lệnh được bắt đầu bởi từ khoá ParBegin (Parallel Begin) và kết thúc bởi từ khoá ParEnd (Parallel End).

```

ParBegin
    S1;
    S2;
    ...
    Sn;
ParEnd;
  
```

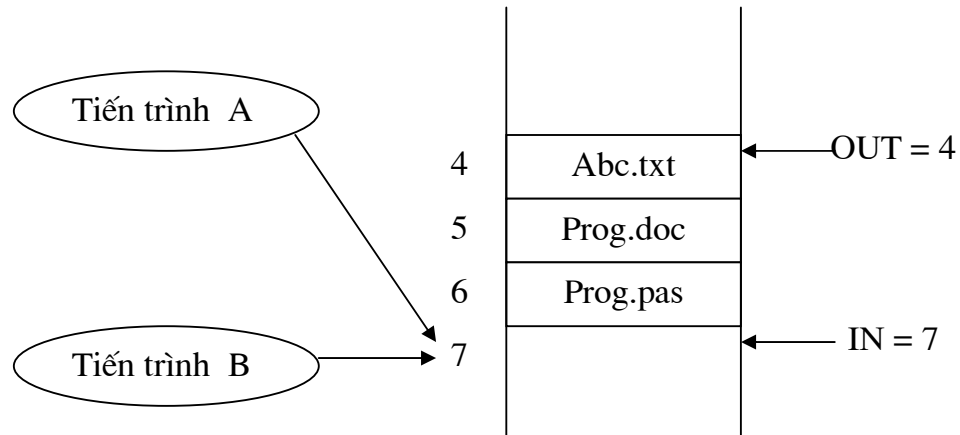
6. Tài nguyên căng và đoạn căng

- Tài nguyên căng là tài nguyên mà trong một khoảng thời gian nhất định thì chỉ phục vụ hợp lý cho một số hữu hạn các tiến trình.
- Đoạn chương trình sử dụng tài nguyên căng gọi là đoạn căng hay chỗ hẹp trong tiến trình.
- Hệ điều hành phải tổ chức cho mọi tiến trình đi qua chỗ hẹp một cách hợp lý, công việc này gọi là điều độ tiến trình qua đoạn căng.
- Sự cần thiết phải điều độ

Ta xem xét ví dụ khi 2 tiến trình cùng muốn in ra máy in.

- + Khi một tiến trình cần in một tệp ra máy in, nó đưa tên tệp vào thư mục spool. Một tiến trình điều khiển in khác kiểm tra định kỳ nếu có tệp nào cần in, nếu tìm thấy thì in tệp nó và loại tên tệp khỏi thư mục spool. Giả sử thư mục spool có số lượng phần tử rất lớn (mỗi phần tử chứa một tên tệp). Ta có hai biến dùng để chỉ tệp tiếp theo cần in và IN để chỉ vị trí rỗng tiếp theo dùng để chứa tên tệp cần in.
- + Ta giả sử vị trí 0 — 3 rỗng (các tệp đã được in), vị trí 4 — 6 đang bận (chứa tên tệp cần in).

Như vậy biến $OUT = 4$ và $IN = 7$



- + Giả sử tiến trình A cần in một tệp a.txt, khi đó tiến trình A sẽ đọc biến IN và đưa vào biến cục bộ IN_A , như vậy $IN_A = 7$. Lúc đó có tín hiệu ngắt đồng hồ và CPU quyết định tiến trình A đã chạy đủ thời gian và chuyển sang thực hiện tiến trình B. Đến lượt mình, tiến trình B cũng muốn in tệp b.txt. Tiến trình B đọc biến IN và đưa vào biến cục bộ IN_B , như vậy $IN_B = 7$. Tiến trình B đưa tên tệp b.txt vào vị trí thứ 7 trong thư mục spool và cập nhật biến $IN = IN_B + 1 = 8$, sau đó làm các việc khác.
- + Khi CPU chuyển sang thực hiện tiến trình A, không may tiến trình A vẫn giữ nguyên biến $IN_A = 7$. Tiến trình A đưa tên tệp a.txt vào vị trí thứ 7 và cập nhật biến $IN = IN_A + 1 = 8$.
- + Tiến trình điều khiển in không được thông báo là có sự cố và tiếp tục thực hiện nhiệm vụ.
- + Như vậy tệp b.txt đã bị đổi thành tệp a.txt và sẽ không được in ra máy in.
- Các công cụ điều độ phải thỏa mãn các yêu cầu sau:
 - + Phải đảm bảo sao cho tiến trình không chiếm giữ tài nguyên gần vô hạn
 - + Nếu có một tiến trình xếp hàng chờ tài nguyên gần thì sớm hay muộn nó phải vào được đoạn gần của mình (được phục vụ tài nguyên gần).
 - + Nếu có tiến trình xếp hàng chờ đợi tài nguyên gần và nếu tài nguyên gần đó được giải phóng thì nó phải được phục vụ trong các tiến trình đang chờ đợi.
- Các công cụ điều độ: Chia làm ba lớp chính
 - + Phương pháp khoá trong: là loại giải thuật không yêu cầu gì về thiết bị hoặc hệ thống. Phương pháp này có tính chất vạn năng ứng với mọi ngôn ngữ, mọi loại máy.
 - + Kiểm tra và xác lập

Xác lập dựa vào thiết bị, thiết bị có những lệnh đặc biệt phục vụ cho riêng công tác điều độ.

- + Kỹ thuật đèn báo: dựa vào công cụ đặc biệt của từng hệ điều hành.

7. Phương pháp khoá trong

- Nguyên lý:

Dùng thêm các biến với tư cách là tài nguyên chung để chứa các cờ cho biết tiến trình vào đoạn găng hay ra khỏi đoạn găng.

- Giả thiết:

- + Có hai tiến trình song song cùng sử dụng 1 tài nguyên găng chung và khả năng phục vụ của tài nguyên găng là 1.
- + Mỗi tiến trình chỉ có một đoạn găng nằm ở đầu tiến trình.
- + Các tiến trình này lặp vô hạn, nếu có kết thúc thì ở đâu đó ngoài đoạn găng.

- Sử dụng một biến IS_USED có giá trị bằng 1 để chỉ ra tài nguyên găng đang bị một tiến trình nào đó chiếm giữ và ngược lại, khi IS_USED = 0 chỉ ra tài nguyên găng đang sẵn sàng phục vụ. Khi một tiến trình thấy biến IS_USED = 0, nó phải đặt biến IS_USED = 1 trước khi sử dụng tài nguyên găng. Tuy nhiên ta dễ dàng tiến biến IS_USED lại trở thành tài nguyên găng. Giả sử tiến trình 1 kiểm tra thấy biến IS_USED = 0, trước lúc nó đặt biến này lên 1 thì tiến trình 2 lại kiểm tra biến này và tất nhiên khi đó biến IS_USED = 0. Như vậy cả hai tiến trình đều vào đoạn găng và đều sử dụng tài nguyên găng. Nói cách khác vấn đề điều độ chưa được giải quyết

- Sử dụng biến TURN để chỉ đến lượt tiến trình nào được sử dụng tài nguyên găng.

- + Sơ đồ nguyên lý

```

Var turn : integer;
Begin
    turn := 1;
ParBegin
    { Hai khối lệnh trong từ khoá ParBegin và ParEnd được
thực hiện song song với nhau }
    TT1:
    REPEAT
        while (turn <> 1) do ;
        vao_doan_gang_1; { đoạn găng của tiến trình 1 }
        turn := 2; { chuyển tài nguyên găng cho tt2)
        thuc_hien_viec_khac_1;
        { phần còn lại của tiến trình 1 }
    UNTIL FALSE;

```

```

TT2:
REPEAT
    while (turn <> 2) do ;
    vao_doan_gang_2; { đoạn găng của tiến trình 2 }
    turn := 1; { chuyển tài nguyên găng cho tt1)
    thuc_hien_viec_khac_2;
    { phần còn lại của tiến trình 2 }
UNTIL FALSE;
ParEnd;
End.

```

- + Giải thích: Ban đầu TURN = 1, tức là tiến trình 1 được phép sử dụng tài nguyên găng. Khi tiến trình 1 dùng xong tài nguyên găng thì đặt TURN = 2, để cho phép tiến trình 2 sử dụng tài nguyên găng. Khi tiến trình 2 sử dụng xong tài nguyên găng thì lại đặt TURN = 1, để chỉ đến lượt tiến trình 1 sử dụng.
- + Tuy nhiên ta giả sử tiến trình 1 dùng xong tài nguyên găng, sau khi đặt TURN = 2 sang thực hiện thủ tục `thuc_hien_viec_khac_1`, thủ tục này khá ngắn, tiến trình 1 quay lại đoạn găng. Nhưng lúc này tiến trình 2 đang bận thực hiện các công việc khác trong thủ tục `thuc_hien_viec_khac_2`. Tiến trình 2 vẫn chưa vào đoạn găng vì vậy biến TURN vẫn có giá trị bằng 2. Vì vậy mặc dù tài nguyên găng không được sử dụng nhưng do TURN = 2 mà tiến trình 1 không thể sử dụng được tài nguyên găng.
- Để khắc phục nhược điểm này người ta đưa ra cách thức dùng hai biến `c1` và `c2` cho hai tiến trình như sau:

```

+ Sơ đồ nguyên lý
Var c1,c2 : integer;
Begin
    c1 := 0;
    c2 := 0;
ParBegin
    { Hai khối lệnh trong từ khoá ParBegin và ParEnd được
thực hiện song song với nhau }
    TT1:
    REPEAT
        while (c2 > 0) do ;
        c1 := 1;
        vao_doan_gang_1; { đoạn găng của tiến trình 1 }
        c1 := 0;
        thuc_hien_viec_khac_1;
        { phần còn lại của tiến trình 1 }
    UNTIL FALSE;

```

```

TT2:
REPEAT
    while (c1 > 0) do ;
    c2 := 1;
    vao_doan_gang_2; { đoạn găng của tiến trình 2 }
    c2 := 0;
    thuc_hien_viec_khac_2;
    { phần còn lại của tiến trình 2 }
UNTIL FALSE;
ParEnd;
End.

```

- Giải thích

C1 và C2 đại diện cho việc sử dụng tài nguyên găng thứ nhất và tài nguyên găng thứ hai.

- + Ban đầu cả hai biến đều có giá trị bằng 0 thể hiện tài nguyên găng đang ở trạng thái sẵn sàng phục vụ.
- + Giả sử tiến trình 1 được phục vụ trước, tiến trình 1 bỏ qua việc chờ đợi
`while (c2 > 0) do ;`
 và chiếm lấy tài nguyên găng đồng thời đặt $C1 = 1$;
- + $C1 = 1$ có nghĩa là tiến trình 1 đang sử dụng tài nguyên găng. Trong lúc tài nguyên găng đang bị tiến trình 1 chiếm giữ thì tiến trình 2 phải chờ đợi
`while (c1 > 0) do ;`
 Khi tiến trình 1 dùng xong tài nguyên găng thì đặt lại biến $C1 = 0$.
- + Khi $C1 = 0$ và tiến trình 2 kết thúc việc chờ đợi
`while (c1 > 0) do ; { được kết thúc do $c1 = 0$ }`
 lúc này tiến trình 2 chiếm giữ tài nguyên găng và đặt $C2 = 1$;
 Khi tiến trình 2 dùng xong tài nguyên găng thì đặt lại $C2 = 0$;
- + Quá trình như vậy được lặp đi lặp lại, cho đến khi kết thúc cả hai tiến trình (lệnh kết thúc ở trong đoạn chương trình không phải là đoạn găng).
- Trong trường hợp tồi nhất, cả hai tiến trình đều vào đoạn găng và đặt biến C1 và C2 bằng 1, và cả hai tiến trình đều không vào được đoạn găng và gây ra hiện tượng chờ đợi vòng tròn. Lý do là việc xác lập vào đoạn găng và khả năng xem xét có được vào đoạn găng của hai đoạn trên không có quan hệ với nhau.
- Vì vậy người ta đưa ra một phương pháp khác phối hợp hai phương pháp trên, phương pháp này phối hợp Xác lập — Kiểm tra — Xác lập, do Delker công bố năm 1968 như sau:

```

Var c1, c2, tt: integer;
Begin
    c1:=0; c2:=0; tt:=1;

```



```

ParBegin
  TT1:
  REPEAT
    c1:=1;
    while(c2=1) do Begin
      if(tt = 2) then Begin
        c1:=0;
        while(tt =2) do;
          c1:=1;
        End;
      End;
      vao_doan_gang_1;
      c1:=0; tt:=2;
      thuc_hien_viec_khac_1;
    UNTIL FALSE;
  TT2:
  REPEAT
    c2:=1;
    while(c1=1) do Begin
      if(tt = 1) then Begin
        c2:=0;
        while(tt =1) do;
          c2:=1;
        End;
      End;
      vao_doan_gang_2;
      c2:=0; tt:=1;
      thuc_hien_viec_khac_2;
    UNTIL FALSE;
  ParEnd;
End.

```

- Ưu điểm:

- + Giải thuật này có tính chất vạn năng áp dụng cho mọi công cụ và mọi hệ thống.
- + Tận dụng, phát huy khả năng tối đa tài nguyên găng.

- Nhược điểm

- + Độ phức tạp tỷ lệ với số lượng tiến trình và số tài nguyên găng.
- + Tồn tại hiện tượng chờ đợi tích cực. Mặc dù không làm gì cả nhưng vẫn chiếm thời gian processor.

Nguyên nhân là do mỗi tiến trình phải làm việc với nhiều biến, trong đó có nhiều biến không phải của mình (ví dụ: muốn xác lập biến c1 phải kiểm tra biến c2 và biến tt).

8. Phương pháp Kiểm tra và Xác lập (Test and Set)

- Trong hệ lệnh của máy tính tồn tại lệnh cho thực hiện nhiều công việc liên tục. Các công việc này tạo thành một hệ lệnh nguyên tố, không thể chỉ thực hiện một công việc. Thủ tục Test_And_Set có thể được định nghĩa như sau:

```

Procedure TS(var local: integer);
Begin
    local:=global;
    global:=1;
End;
```

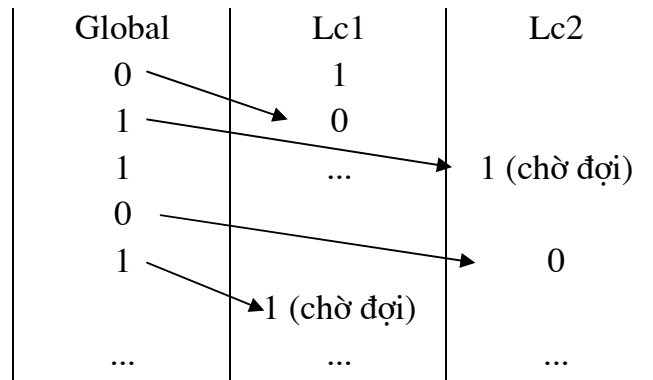
- + Chú ý: hai lệnh trên phải được thực hiện liên tục không bị chia rẽ.
- + Mỗi tiến trình sẽ sử dụng hai biến là biến local của mình và biến global của toàn chương trình.
- Sơ đồ điều độ

```

Var
    lc1, lc2: integer;
    global: integer;

Procedure TS(var local: integer);
Begin
    local:=global;
    global:=1;
End;

Begin
    gl:=0;
    ParBegin
        TT1:
            REPEAT
                lc1:=1;
                while lc1=1 do TS(lc1);
                vao_doan_gang_1;
                gl:=0;
                thuc_hien_viec_khac_1;
            UNTIL FALSE;
        TT2:
            REPEAT
                lc2:=1;
                while lc2=1 do TS(lc2);
                vao_doan_gang_2;
                gl:=0;
                thuc_hien_viec_khac_2;
            UNTIL FALSE;
        ParEnd;
    End.
```



– Ưu điểm:

Khắc phục được độ phức tạp của thuật toán, độ phức tạp thuật toán không phụ thuộc vào số lượng tiến trình.

– Nhược điểm:

Vẫn còn hiện tượng chờ đợi tích cực.

9. Kỹ thuật đèn báo

Đây là công cụ phụ thuộc vào hệ thống do Dijkstra đề xuất, với tư tưởng như sau:

- Hệ thống sử dụng biến đèn báo nguyên đặc biệt (Semaphore) s . Ban đầu s nhận một giá trị bằng khả năng phục vụ của tài nguyên găng. Hệ thống có hai phép để thao tác trên s là $P(s)$ và $V(s)$.

P : Proberen (tiếng Hà Lan) có nghĩa là giảm

V : Verhogen có nghĩa là kiểm tra

- Nội dung của $P(s)$ như sau:

+ Giảm s đi một:

$s := s - 1$

+ Kiểm tra xem nếu $s < 0$ đưa tiến trình vào xếp hàng

If ($s < 0$) then xếp_hàng;

- Nội dung của $V(s)$ như sau:

+ Tăng s lên một:

$s := s + 1;$

+ Kiểm tra nếu $s \leq 0$ thì kích hoạt một tiến trình ra hoạt động

If ($s \leq 0$) then hoạt_dong;

- Đặc điểm quan trọng là 2 phép P và V là liên tục, trong quá trình thực hiện P hoặc V thì processor không bị ngắt để chuyển sang công việc khác.
- Tuy nhiên các phép xử lý này có thể không tồn tại trên các máy vì P và V phải làm việc với dòng xếp hàng và thông tin lưu trữ khá lớn. Để khắc phục

điều này người ta xây dựng các thủ tục procedure để thực hiện các phép xử lý này.

- + Đầu của thân thủ tục bao giờ cũng ra lệnh cấm ngắt tức là chặn mọi tín hiệu vào processor CLI, trừ những tín hiệu bắt buộc (ngắt không che được).
- + Cuối thân thủ tục có lệnh giải phóng ngắt (STI).
- Sơ đồ điều độ

```

Var
    s: integer;
Begin
    s:=1;
    ParBegin
        TT1:
            REPEAT
                P(s);
                vao_doan_gang_1;
                V(s);
                thuc_hien_viec_khac_1;
            UNTIL FALSE;
        TT2:
            REPEAT
                P(s);
                vao_doan_gang_2;
                V(s);
                thuc_hien_viec_khac_2;
            UNTIL FALSE;
    ParEnd;
End.

```

S	TT1	TT2
1	P(s)	
0	Làm TT1	P(s)
-1		chờ đợi
-1	TT1 xong	chờ đợi
0	V(s)	Làm TT2
0		TT2 xong
1		V(s)
Vì s>0 nên không còn tiến trình nào cần tài nguyên gang		

- Ưu điểm:
Chống được hiện tượng chờ đợi tích cực.

Lý do: mỗi tiến trình chỉ phải kiểm tra điều kiện vào đoạn găng một lần, nếu không vào được sẽ có một tiến trình khác kích hoạt tiến trình này vào thời điểm thích hợp.

- Quản lý bộ nhớ

10. LT: 6 Tiết, TH:

Quản lý tiến trình LT: 12 Tiết, TH:Tài liệu tham khảo

- Tài liệu tham khảo

- [1] **Nguyễn Thanh Tùng.** *Giáo trình Hệ điều hành*, 1995.
- [2] **A.S. Tanenbaum.** *Operating Systems Design and Implementation*, 1997.
- [3] **Abraham Silberschatz.** *Principes des systèmes d'exploitation*, 1994.
- [4] **Peter Norton.** *Cẩm nang lập trình hệ thống cho IBM PC*, 1992.

- [5] **Phạm Văn Ất.** *Lập trình C cơ sở và nâng cao*, 1997.
- [6] **Lê Đức Trung, Lê Đăng Hưng, Nguyễn Thanh Thủy.** *Ngôn ngữ Lập trình C*, 1996.
- [7] **Lê Đăng Hưng, Tạ Tuấn Anh.** *Lập trình hướng đối tượng với C++*, Khoa CNTT 1999.
- [8] **Bruce Eckel.** *Thinking in C++*, 1999.

Chương 1. Các khái niệm cơ bản.....	2
1. Cấu trúc phân lớp và sự phát triển của hệ thống tính toán.....	2
2. Tài nguyên của hệ thống tính toán.....	5
3. Định nghĩa HĐH.....	7
4. Phân loại hệ điều hành.....	8
5. Các tính chất cơ bản của hệ điều hành.....	9
6. Nguyên lý xây dựng chương trình HĐH.....	10
7. Thành phần của HĐH và kiến trúc HĐH.....	13
8. Các hình thái giao tiếp.....	14
9. Giới thiệu về MSDOS.....	15
Chương 2. Hệ thống xử lý ngắt trong IBM PC.....	17
1. Khái niệm về ngắt và xử lý ngắt trong IBM PC.....	17
2. Phân loại ngắt.....	17
3. Quy trình xử lý ngắt.....	18
4. Bảng vector ngắt.....	19
5. Gọi ngắt trong Assembler.....	20
6. Gọi ngắt trong Pascal.....	20
7. Bộ thanh ghi của 8088.....	21
8. Thay đổi ngắt trong hệ thống.....	23
9. Một số hàm và thủ tục thường dùng trong lập trình hệ thống.....	24
Chương 3. Quản lý thiết bị ngoại vi và tệp.....	32
1. Nguyên lý phân cấp trong tổ chức và quản lý thiết bị ngoại vi.....	32
2. Phòng Đệm (Buffer).....	34
3. SPOOL- Simultaneous Peripheral Operation On-Line.....	38
4. Quản lý màn hình.....	39
5. Quản lý bàn phím.....	46
6. Quản lý tệp.....	54
7. Quản lý tệp trong MSDOS.....	57
Chương 4. Quản lý tiến trình.....	78
1. Định nghĩa tiến trình.....	78
2. Khối điều khiển tiến trình (Process Control Bloc - PCB).....	79
3. Cách thực tiến trình.....	79
4. Phân loại tiến trình song song.....	81
5. Mô tả tiến trình song song.....	83
6. Tài nguyên căng và đoạn căng.....	84

7. Phương pháp khoá trong	86
8. LT: 6 Tiết, TH:	94