HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

BÀI GIẢNG MÔN

HỆ ĐIỀU HÀNH

Giảng viên: TS. Đào Thị Thúy Quỳnh

Bộ môn: Khoa học máy tính - Khoa CNTT1

Email: quynhdao.ptit@gmail.com

TÀI LIỆU THAM KHẢO

1. Silberschatz A., Galvin G., Operating systems concepts, 9th ed, John Willey&Sons, 2013
2. Từ Minh Phương, Bài giảng Hệ điều hành
3. Hà Quang Thụy. Nguyên lý các hệ điều hành. Nxb KHKT 2009
4. Nguyễn Thanh Tùng. Giáo trình hệ điều hành. ĐHBK HN 1999

ĐIỂM THÀNH PHẦN

- Điểm chuyên cần: 10%
- Điểm trung bình kiểm tra: 10%
- Điểm thực hành: 10%
- Thi cuối kỳ: 70%

NỘI DUNG CHÍNH

1. Chương 1: Giới thiệu chung
2. Chương 2: Hệ thống file
3. Chương 3: Quản lý bộ nhớ
4. Chương 4: Quản lý tiến trình

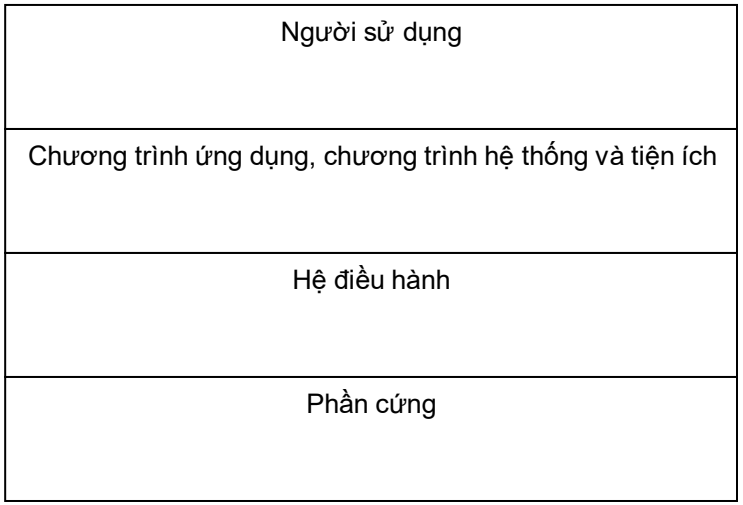
CHƯƠNG 1: GIỚI THIỆU CHUNG

NỘI DUNG

1. Các thành phần của hệ thống máy tính
2. Khái niệm hệ điều hành
3. Các dịch vụ do HDH cung cấp
4. Giao diện lập trình của HDH
5. Quá trình phát triển và một số khái niệm quan trọng
6. Cấu trúc HDH
7. Một số HDH cụ thể

I. CÁC THÀNH PHẦN CỦA HỆ THỐNG MÁY TÍNH

- Phần cứng: cung cấp các tài nguyên cần thiết cho việc tính toán, xử lý dữ liệu
- Phần mềm: các chương trình cụ thể. (phần mềm hệ thống và phần mềm ứng dụng),
- HDH: phần mềm đóng vai trò trung gian giữa **phần cứng** và **người sử dụng** chương trình ứng dụng, làm cho việc sử dụng hệ thống máy tính được *tiện lợi* và *hiệu quả*



II. KHÁI NIỆM HỆ ĐIỀU HÀNH

- Được định nghĩa thông qua mục đích, vai trò, và chức năng trong hệ thống máy tính
- Hệ thống phần mềm đóng vai trò trung gian giữa người sử dụng và phần cứng của máy tính nhằm thực hiện 2 chức năng cơ bản:
  - Quản lý tài nguyên
  - Quản lý việc thực hiện các chương trình

=> Một cách thuận lợi và hiệu quả!

II. KHÁI NIỆM HỆ ĐIỀU HÀNH

1. Quản lý tài nguyên

- Đảm bảo cho tài nguyên hệ thống được sử dụng một cách có ích và hiệu quả
- Các tài nguyên: bộ xử lý (CPU), bộ nhớ chính, bộ nhớ ngoài (các đĩa), các thiết bị vào ra
- Phân phối tài nguyên cho các ứng dụng hiệu quả:
  - Yêu cầu tài nguyên được HDH thu nhận và đáp ứng bằng cách cấp cho chương trình các tài nguyên tương ứng
  - HDH cần lưu trữ tình trạng tài nguyên
- Đảm bảo không xâm phạm tài nguyên cấp cho chương trình khác
- Ví dụ: Lưu trữ thông tin trên đĩa => HDH cần biết những vùng nào trên đĩa chưa được sử dụng để ghi thông tin lên những vùng này. Việc ghi thông tin cũng cần tính toán sao cho quá trình truy cập khi cần có thể thực hiện nhanh nhất.

II. KHÁI NIỆM HỆ ĐIỀU HÀNH

2. Quản lý việc thực hiện các chương trình

- Nhiệm vụ quan trọng nhất của máy tính là thực hiện các chương trình, 1 chương trình đang trong quá trình chạy gọi là tiến trình (process). Chương trình cần được quản lý để thực hiện thuận lợi, tránh lỗi, đồng thời đảm bảo môi trường để việc xây dựng và thực hiện chương trình được thuận lợi.
- Để chạy chương trình cần thực hiện một số thao tác nhất định =>Hdh giúp việc chạy chương trình dễ dàng hơn, người dùng không cần phải thực hiện thao tác
- Để tạo môi trường thuận lợi cho chtr, hđh tạo ra các máy ảo:
  - Là máy logic với các tài nguyên ảo
  - Tài nguyên ảo mô phỏng tài nguyên thực được thực hiện bằng phần mềm
  - Cung cấp các dịch vụ cơ bản như tài nguyên thực
  - Dễ sử dụng hơn, số lượng tài nguyên ảo có thể lớn hơn số lượng tài nguyên thực

III. CÁC DỊCH VỤ DO HDH CUNG CẤP

- Một trong những nhiệm vụ chủ yếu của HDH là tạo ra môi trường thuận lợi cho các chương trình khác thực hiện và giúp người sử dụng hệ thống dễ dàng.
- Các dịch vụ có thể thay đổi theo từng HDH. Một số HDH có thể cung cấp nhiều dịch vụ khi hệ điều hành khác có thể cung cấp ít dịch vụ hơn.
  - Ví dụ như MS-DOS không cung cấp dịch vụ về bảo mật trong khi Windows NT lại rất chú trọng tới dịch vụ này.

III. CÁC DỊCH VỤ DO HDH CUNG CẤP

- **Tải và chạy chương trình:**
- Để thực hiện, chương trình được tải từ đĩa vào bộ nhớ, sau đó được trao quyền thực hiện các lệnh
- Khi thực hiện xong, cần giải phóng bộ nhớ và các tài nguyên
- Toàn bộ quá trình này tương đối phức tạp song lại diễn ra thường xuyên.
- => HDH sẽ thực hiện công việc phức tạp và lặp đi lặp lại này
- Do HDH là chương trình đầu tiên được thực hiện khi khởi động hệ thống nên HDH tự tải mình vào bộ nhớ
- Giao diện với người dùng: cho phép giao tiếp giữa HDH và người dùng: Dưới dạng dòng lệnh, Giao diện đồ họa
- Thực hiện các thao tác vào/ ra dữ liệu

III. CÁC DỊCH VỤ DO HDH CUNG CẤP

- Làm việc với hệ thống file: nhu cầu đọc, ghi, tạo, xóa, chép file hoặc làm việc với thư mục; quản lý quyền truy cập, sao lưu.
- Phát hiện và xử lý lỗi
  - Phát hiện và xử lý kịp thời các lỗi xuất hiện trong phần cứng cũng như phần mềm
  - => Đảm bảo cho hệ thống hoạt động ổn định, an toàn
  - Ví dụ: các lỗi phần cứng như hết bộ nhớ, mất điện, máy in hết mực, hết giấy,..
- Truyền thông:
  - Cung cấp dịch vụ cho phép thiết lập liên lạc và truyền thông tin

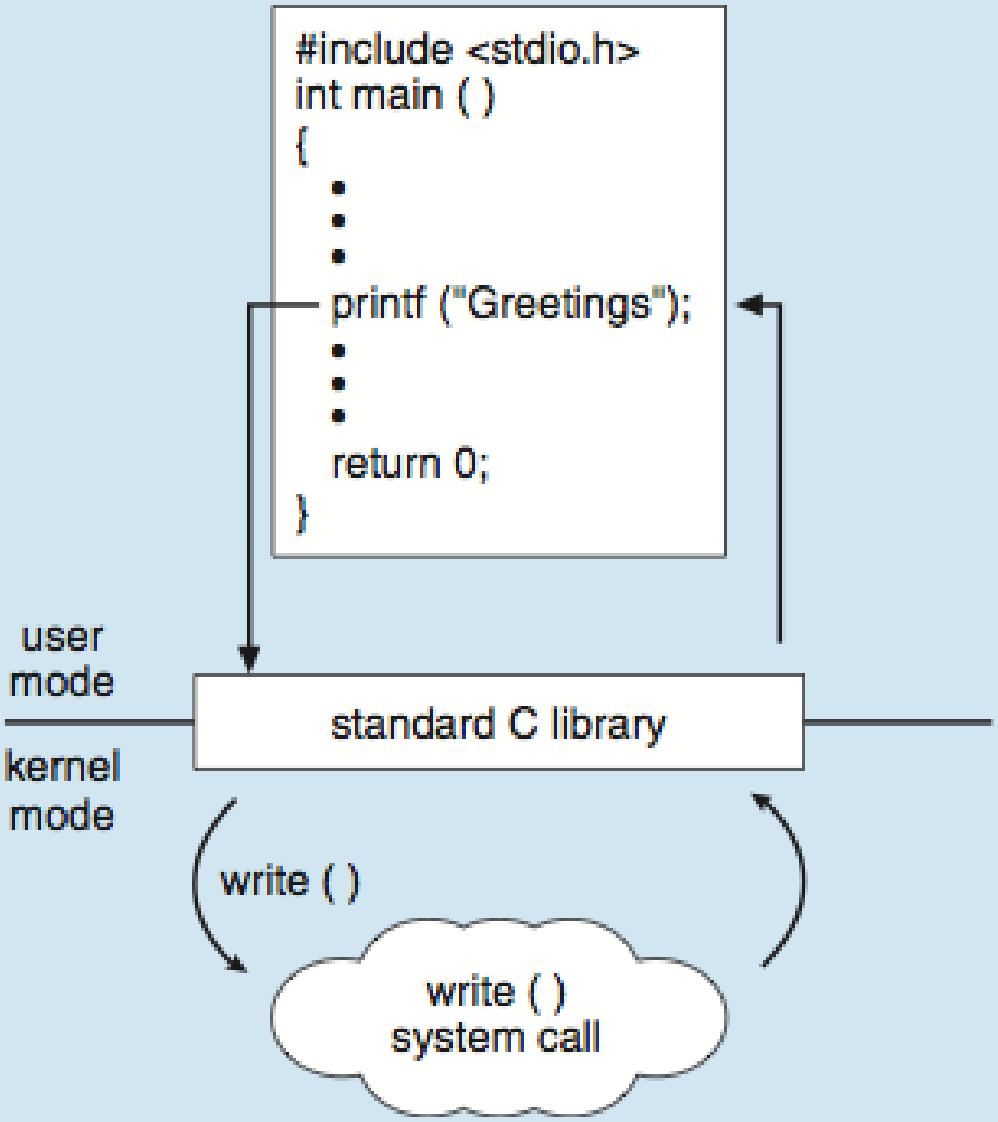
III. CÁC DỊCH VỤ DO HDH CUNG CẤP

- Cấp phát tài nguyên:
  - Trong các hệ thống cho phép nhiều chương trình thực hiện đồng thời cần có cơ chế cấp phát và phân phối tài nguyên hợp lý => người dùng và trình ứng dụng không phải tự thực hiện việc cấp phát tài nguyên mà vẫn đảm bảo cấp ptá công bằng và hiệu quả.
- Dịch vụ an ninh và bảo mật
  - Đối với hệ thống nhiều người dùng thường xuất hiện yêu cầu bảo mật thông tin, tức là người dùng này không tiếp cận được thông tin của người khác nếu không được cho phép => cần đảm bảo để tiến trình không truy cập trái phép tài nguyên (như vùng nhớ, file mở) của tiến trình khác hay chính HDH sẽ thực hiện bằng cách kiểm soát truy cấp tới tài nguyên

IV. GIAO DIỆN LẬP TRÌNH CỦA HDH

- Để các chương trình có thể sử dụng được những dịch vụ, HDH cung cấp giao diện lập trình.
- Giao diện này bao gồm các lời gọi hệ thống (system call) mà chương trình sử dụng yêu cầu một dịch vụ nào đó từ phía HDH.
- Lời gọi hệ thống: các lệnh đặc biệt mà CTUD gọi khi cần yêu cầu HDH thực hiện một việc gì đó
- Lời gọi hệ thống được thực hiện qua những thư viện hàm gọi là thư viện hệ thống. Các hàm này sẽ giúp người lập trình gọi lời gọi hệ thống tương ứng của hệ điều hành.

IV. GIAO DIỆN LẬP TRÌNH CỦA HDH



V. QUÁ TRÌNH PHÁT TRIỂN

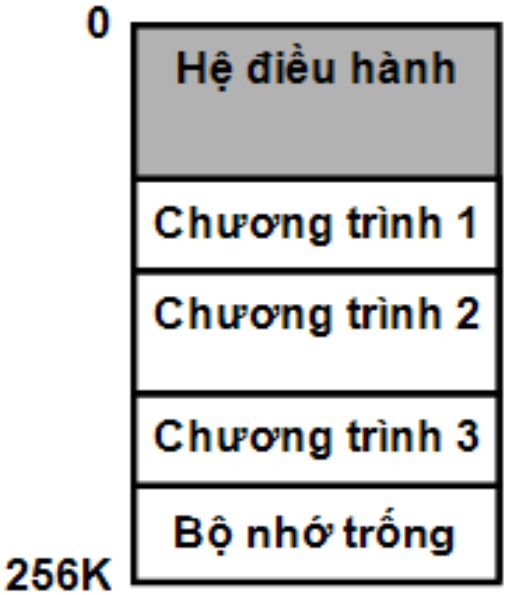
- Các hệ thống đơn giản (những năm 40-50 của thế kỷ trước): tốc độ xử lý của máy tính rất thấp, việc vào/ra được thực hiện thủ công và khó khăn. Việc nạp chương trình được thực hiện nhờ công tắc, mạch hàn sẵn, bìa đục lỗ. Trong thời kỳ này, lập trình viên tương tác trực tiếp với phần cứng.

=> Máy tính thời kỳ này chưa có HDH.

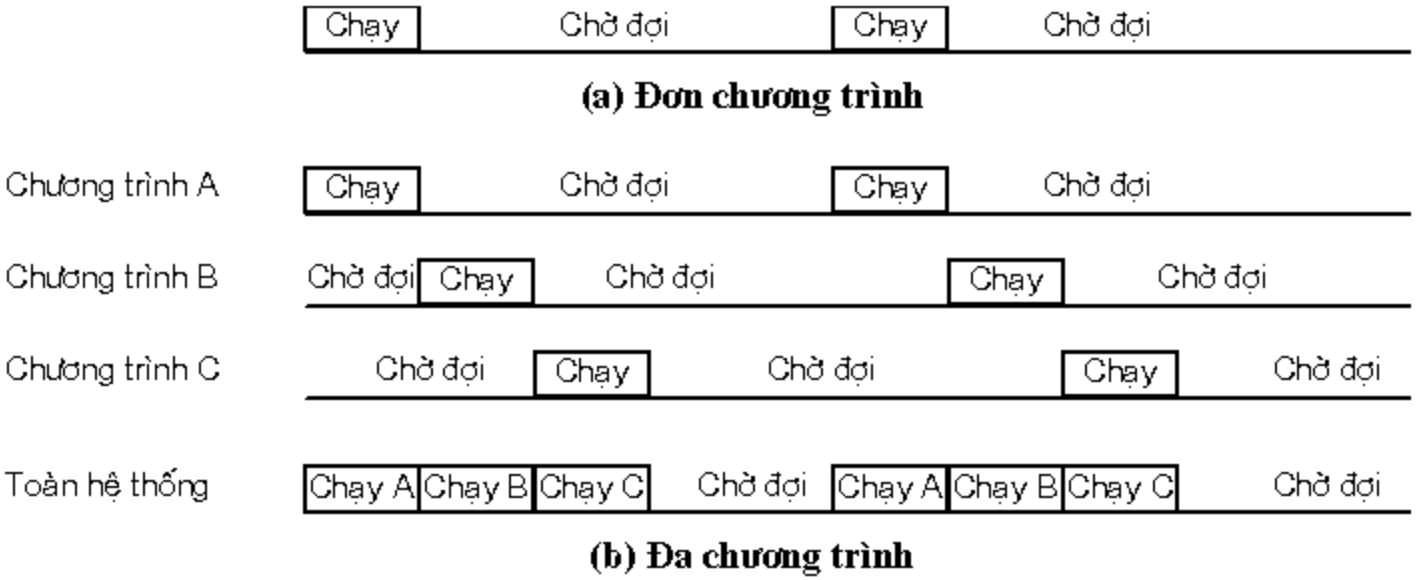
- Xử lý theo mẻ:
  - Chương trình được phân thành các *mẻ*: gồm những chương trình có yêu cầu giống nhau
  - Toàn bộ mẻ được nạp vào băng từ và được tải vào máy để thực hiện lần lượt
- *Chương trình giám sát* (monitor): mỗi khi một chương trình của mẻ kết thúc, chương trình giám sát tự động nạp chương trình tiếp theo vào máy và cho phép nó chạy => Giảm đáng kể thời gian chuyển đổi giữa hai chương trình trong cùng một mẻ

- Trình giám sát là dạng đơn giản nhất của HDH
- Nhược điểm: Mỗi khi có yêu cầu vào/ra, CPU phải dừng việc xử lý dữ liệu để chờ quá trình vào ra kết thúc. Do tốc độ vào/ra thấp hơn tốc độ CPU rất nhiều nên CPU thường xuyên phải chờ đợi 1 thời gian dài.= > hiệu suất CPU thấp.

- Đa chương trình (đa nhiệm):
  - Hệ thống chứa đồng thời nhiều chương trình trong bộ nhớ
  - Khi một chương trình phải dừng lại để thực hiện vào ra, HDH sẽ chuyển CPU sang thực hiện một chương trình khác
  - => Giảm thời gian chạy không tải của CPU



- Đa chương trình:



  - Thời gian chờ đợi của CPU trong chê độ đa chương trình giảm đáng kể so với trong trường hợp đơn chương trình
  - HDH phức tạp hơn rất nhiều so với HDH đơn chương trình
  - Đòi hỏi hỗ trợ từ phần cứng, đặc biệt khả năng vào/ra bằng ngắt và DMA

- Đa chương trình (Hạn chế)
  - Mặc dù đa chương trình cho phép sử dụng hiệu quả CPU và các tài nguyên khác của hệ thống song kỹ thuật này không cho người dùng tương tác với hệ thống.
  - Các máy tính thế hệ sau cho phép máy tính và người dùng làm việc trực tiếp thông quan màn hình và bàn phím.
  - Đối với các hệ thống này thì thời gian từ khi người dùng gõ lệnh cho tới khi máy tính phản xạ lại tương đối nhỏ.
  - Kỹ thuật đa chương trình không đảm bảo được thời gian đáp ứng ngắn như vậy.

- Chia sẻ thời gian:
  - Chia sẻ thời gian có thể coi như đa chương trình cải tiến
  - CPU lần lượt thực hiện các công việc khác nhau trong những khoảng thời gian ngắn gọi là lượng tử thời gian
  - Chuyển đổi giữa các công việc diễn ra với tần số cao và tốc độ CPU lớn
  - => Tất cả người dùng đều có cảm giác máy tính chỉ thực hiện chương trình của mình
  - => CPU được chia sẻ giữa những người dùng khác nhau tương tác trực tiếp với hệ thống

- Quản lý tiến trình:
  - Tạo và xoá tiến trình
  - Tạm treo và khôi phục các tiến trình bị treo
  - Đồng bộ hoá các tiến trình (lập lịch cho các tiến trình .v.v.)
  - Giải quyết các bế tắc, ví dụ như khi có xung đột về tài nguyên
  - Tạo cơ chế liên lạc giữa các tiến trình

- Quản lý bộ nhớ:
  - Quản lý việc phân phối bộ nhớ giữa các tiến trình
  - Tạo ra bộ nhớ ảo và ánh xạ địa chỉ bộ nhớ ảo vào bộ nhớ thực
  - Cung cấp và giải phóng bộ nhớ theo yêu cầu của các tiến trình
  - Quản lý không gian nhớ đã được cấp và không gian còn trống

(Tiến trình: chương trình đang trong thời gian thực hiện)

- Quản lý vào ra:
  - Đơn giản hoá và tăng hiệu quả quá trình trao đổi thông tin giữa các tiến trình với thiết bị vào ra
- Quản lý tệp và thư mục:
  - Tạo, xóa tệp và thư mục
  - Đọc ghi tệp
  - Ánh xạ tệp và thư mục sang bộ nhớ ngoài
- Hỗ trợ mạng và xử lý phân tán
- Giao diện với người dùng
- Các chương trình tiện ích và ứng dụng

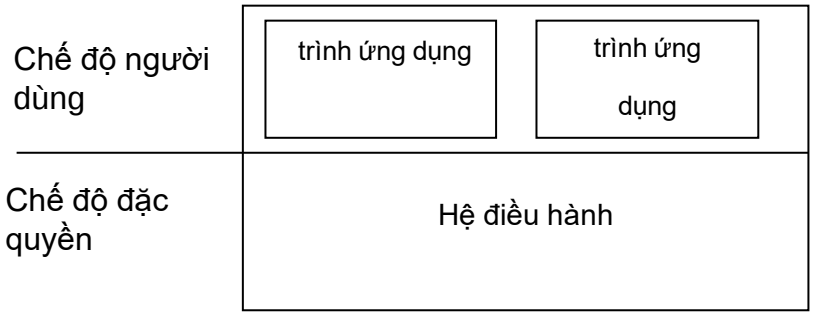
VI. CẤU TRÚC HDH
2. NHÂN CỦA HDH

- HDH gồm rất nhiều thành phần, tuy nhiên độ quan trọng của các tp khác nhau, có những tp không thể thiếu là cơ sở cho toàn hệ thống hoạt động, một số tp của HDH cung cấp chức năng kém quan trọng hơn.
- => chỉ tải những thành phần quan trọng không thể thiếu được vào bộ nhớ gọi là nhân.
- *Nhân (kernel) là phần cốt lõi, thực hiện các chức năng cơ bản nhất, quan trọng nhất của HDH và thường xuyên được giữ trong bộ nhớ*

VI. CẤU TRÚC HDH
2. NHÂN CỦA HDH

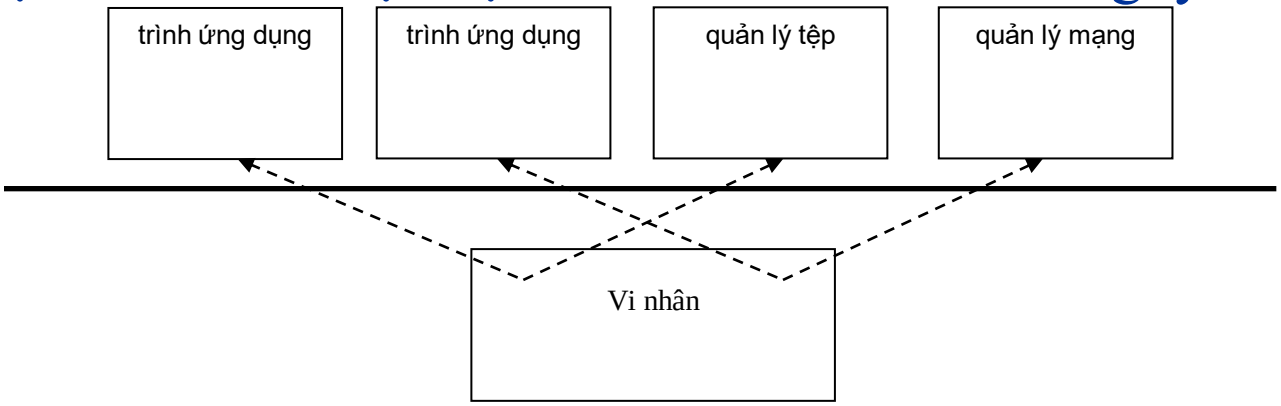
- Máy tính hiện đại thường được thiết kế với hai chế độ thực hiện chương trình.
  - Nhân chạy trong chế độ đặc quyền – chế độ nhân: là chế độ mà chương trình thực hiện trng đó có đầy đủ quyền truy cập và điều khiển phần cứng máy tính.
  - Chế độ người dùng: ch trình thực hiện trong chế độ người dùng bị hạn chế rất nhiều quyền truy cập và sử dụng phần cứng.
- Việc phân biệt chế độ nhân và chế độ người dùng nhằm mục đích ngăn không cho CTUD vô tình hoặc cố ý thực hiện những thao tác làm ảnh hưởng tới hệ thống.

- Cấu trúc nguyên khối
  - Toàn bộ chương trình và dữ liệu của HDH có chung 1 không gian nhớ
  - HDH trở thành một tập hợp các thủ tục hay các chương trình con
  - Ưu điểm: nhanh, không mất thời gian giữa các không gian nhớ
- Nhược điểm: Không an toàn: khi bất kỳ thành phần nào có sự cố thì toàn bộ hệ thống sẽ không hoạt động đc; Ko mềm dẻo và khó sửa đổi, thêm bớt tp nào sẽ ảnh hưởng tới toàn bộ hệ thống, khi có lỗi khó xác định lỗi do tpnào gây ra.
  - Linux



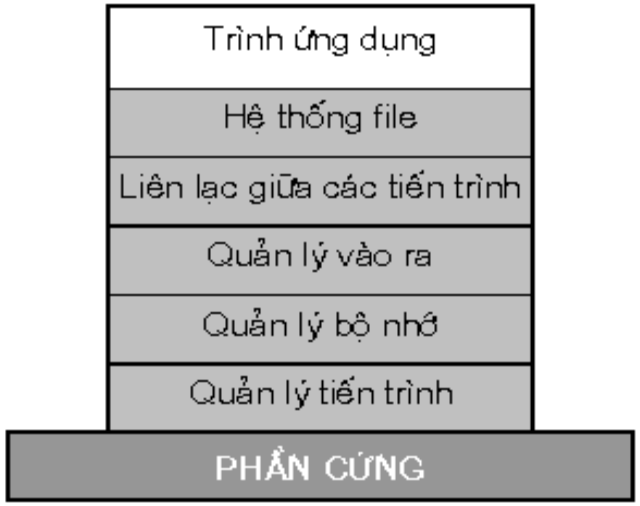
Hình :Cấu trúc nguyên khối

- Cấu trúc vi nhân
  - Nhân có kích thước nhỏ, chỉ chứa các chức năng quan trọng nhất
  - Các chức năng còn lại được đặt vào các modul riêng: chạy trong chế độ đặc quyền hoặc người dùng. Khi có yêu cầu từ ứng dụng, nhân sẽ chuyển cho module tương ứng để xử lý và nhận lại kết quả, nhân chủ yếu đóng vai trò trung gia liên lạc.
  - Ưu điểm: mềm dẻo, an toàn
  - Nhược điểm: tốc độ chậm hơn so với cấu trúc nguyên khối



Hình 1.5 Cấu trúc vi nhân

- Cấu trúc phân lớp
  - Các thành phần được chia thành các lớp nằm chồng lên nhau
  - Mỗi lớp chỉ có thể liên lạc với lớp nằm kề bên trên và kề bên dưới
  - Mỗi lớp chỉ có thể sử dụng dịch vụ do lớp nằm ngay bên dưới cung cấp
  - Ưu điểm: chia nhỏ chứng năng, dễ sử dụng, dễ sửa lỗi
  - Nhược điểm: khó thiết kế do khó sắp xếp các chức năng, tốc độ chậm hơn cấu trúc nguyên khối



VII. MỘT SỐ HDH CỤ THỂ

- UNIX
- MINIX
- LINUX
- MS-DOS
- Windows NT

HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

BÀI GIẢNG MÔN

HỆ ĐIỀU HÀNH

CHƯƠNG 2: HỆ THỐNG FILE

NỘI DUNG

1. Các khái niệm
2. Các phương pháp truy cập file
3. Các thao tác với file
4. Thư mục
5. Cấp phát không gian cho file
6. Quản lý không gian trống trên đĩa
7. Độ tin cậy của hệ thống file
8. Bảo mật cho hệ thống file
9. Hệ thống file FAT

Đặt vấn đề

- Hệ thống máy tính phải có khả năng lưu lại được các thông tin, dữ liệu
- Hệ thống lưu trữ của các loại máy tính cao cấp thường có kích thước rất lớn nên sẽ chứa rất nhiều dữ liệu
- Lượng dữ liệu lưu trữ quá lớn, nếu không khéo trong việc quản lý truy cập sẽ khó khăn và hao tốn thời gian.

=> cần có các hình thức tổ chức, sắp xếp dữ liệu một cách hợp lý và xây dựng các thuật toán tối ưu để có thể truy cập nhanh chóng, hiệu quả

I. CÁC KHÁI NIỆM

- *File được định nghĩa như tập hợp các thông tin liên quan đến nhau được đặt tên và được lưu trữ trên bộ nhớ ngoài*
- Thuộc tính của file: để quản lý file ngoài nội dung, HĐH còn định nghĩa các thuộc tính, tính chất. File có các thuộc tính như sau:
  - Tên file
  - Kiểu file
  - Kích thước file
  - Người tạo file, người sở hữu
  - Quyền truy cập file
  - Thời gian tạo file, sửa file, truy cập lần cuối
  - Vị trí file

- Đặt tên cho file:
  - Cho phép xác định file
  - Là thông tin người dùng thường sử dụng nhất khi làm việc với file
  - Quy tắc đặt tên cho file của một số HDH:

Hệ điều hành	Độ dài tối đa	Phân biệt chữ hoa, chữ thường	Cho phép sử dụng dấu cách	Các ký tự cấm
MS-DOS	8 cho tên file 3 cho mở rộng	không	không	Bắt đầu bằng chữ cái hoặc số Không được chứa các ký tự / \ [ ] : ; | = , ^ ? @
Windows NT FAT	255 ký tự cho cả tên file và đường dẫn	không	có	Bắt đầu bằng chữ cái hoặc số Không được chứa các ký tự / \ [] : ; | = , ^ ? @
Windows NT NTFS	255	không	có	Không được chứa các ký tự / \ < > * | :
Linux (EXT3)	256	Có	có (nếu tên file chứa trong ngoặc kép)	Không được chứa các ký tự ! @ # $ % ^ & * ( ) [ ] { } ‘ “ / \ : ; < > `

Cấu trúc file:

- Các thông tin trong file có thể rất khác nhau
- Có những file chứa nhiều thông tin không có cấu trúc: file văn bản. File có cấu trúc như: file CSDL, file excel.
- => Cấu trúc của file cũng rất khác nhau và phụ thuộc vào thông tin chứa trong file
- HDH có cần biết và hỗ trợ các kiểu cấu trúc file?
- Hỗ trợ cấu trúc file ở mức HDH:
  - Ưu điểm:
    - Các thao tác với file sẽ dễ dàng hơn đối với người lập trình ứng dụng
    - HDH có thể kiểm soát được các thao tác với file
  - Nhược điểm:
    - Tăng kích thước hệ thống
    - Tính mềm dẻo của HDH bị giảm
- Thực tế các HDH chỉ coi file là tập hợp các byte không cấu trúc

I. CÁC KHÁI NIỆM

- Đa số HDH không hỗ trợ và quản lý kiểu cấu trúc file
- Cấu trúc file do chương trình ứng dụng và người dùng tự quản lý
- Trong HDH UNIX, DOS, WINDOWS, file được xem như tập hợp các byte.
- Các chương trình ứng dụng khác nhau sẽ tự tạo ra và quản lý cấu trúc file riêng mình.
- Ví dụ: chương trình đồ họa lưu file dưới dạng mã nhị phân đã được giải nén, chương trình hệ thống quản lý dữ liệu sẽ tạo ra file bao gồm các bản ghi.

II. CÁC PHƯƠNG PHÁP TRUY CẬP FILE

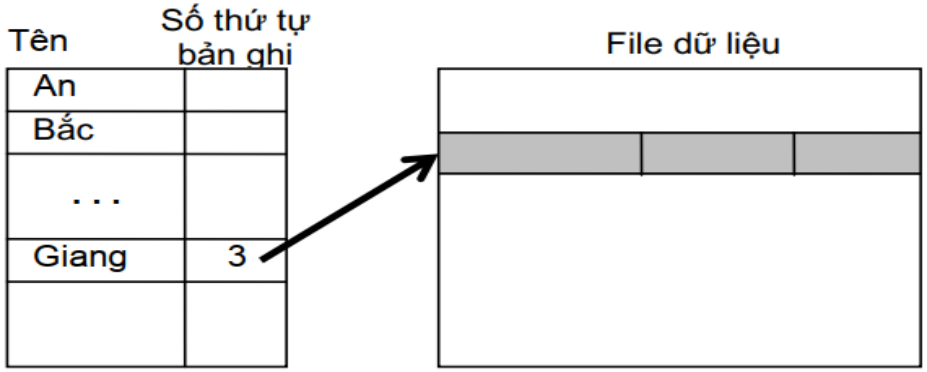
- Để đọc/ghi file hệ điều hành phải quy định cách thức truy cập tới nội dung file. Mỗi HDH có thể hỗ trợ một hoặc nhiều cách truy cập khác nhau.
- Truy cập tuần tự:
  - Thông tin được đọc, ghi theo từng byte/ bản ghi lần lượt từ đầu file
  - Sử dụng 1 con trỏ để định vị vị trí hiện thời trong file
  - Kiểu truy cập này phù hợp với một số thiết bị và một số thiết bị nhớ và một số ứng dụng

II. CÁC PHƯƠNG PHÁP TRUY CẬP FILE

- Truy cập trực tiếp:
  - File được xem như các khối/ bản ghi được đánh số
  - Các khối có thể truy cập theo thứ tự bất kỳ
  - Chẳng hạn ta có thể đọc khối 50 sau đó đọc khối 13 rồi khối 101.
  - Việc truy cập trực tiếp dựa trên đặc điểm của đĩa cho phép truy cập các khối bất kỳ
  - File được chứa trong các khối khác nhau của đĩa do vậy cũng cho phép truy cập không tuần theo thứ tự

II. CÁC PHƯƠNG PHÁP TRUY CẬP FILE

- Truy cập dựa trên chỉ số:
  - Cho phép truy cập tới bản ghi trong file, không theo số thứ tự hoặc vị trí của bản ghi trong file mà theo khóa ứng với bản ghi đó.
  - File chứa 1 chỉ số riêng: gồm các khóa và con trỏ chỉ tới các bản ghi trong file
  - Truy cập: tìm khóa tương ứng trong chỉ mục, sau đó theo con trỏ xác định bản ghi và truy cập trực tiếp tới nó



Hình 4.1: Truy cập theo khối chỉ số

III. CÁC THAO TÁC VỚI FILE

- Tạo file:
  - Tạo file trống chưa có data; được dành 1 chỗ trong thư mục kèm theo một số thuộc tính như thời gian tạo file, tên file, người tạo file,...
- Xóa file:
  - Giải phóng không gian mà dữ liệu của file chiếm trên đĩa
  - Giải phóng chỗ của file trong thư mục
  - Việc giải phóng không gian có thể đơn thuần là đánh dấu không gian như không gian tự do.
- Mở file:
  - Thực hiện trước khi ghi và đọc file
  - Đọc các thuộc tính của file trên đĩa vào bộ nhớ để tăng tốc độ cho thao tác đọc ghi tiếp theo.

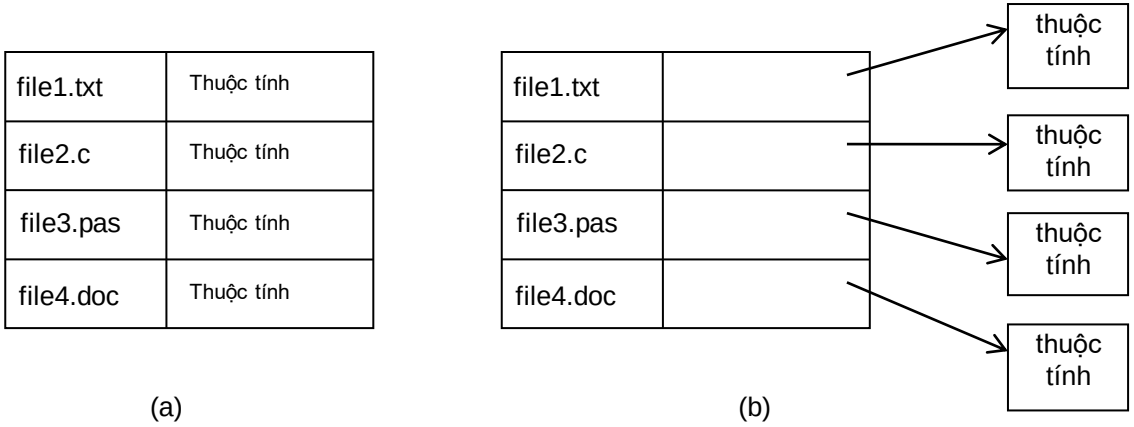
III. CÁC THAO TÁC VỚI FILE

- Đóng file:
  - Xóa các thông tin về file ra khỏi bảng trong bộ nhớ để nhường chỗ cho các file sắp mở.
  - Rất nhiều hệ điều hành hạn chế số lượng file được mởcùng một lúc nên việc đóng các file đã truy cập xong là rất quan trọng.
- Ghi vào file
- Đọc file

IV. THƯ MỤC
1. Khái niệm

- Số lượng file lưu trữ trên đĩa rất lớn => phải tổ chức để dễ dàng quản lý, truy cập files
- Không gian trên đĩa được chia thành các phần (partition/ volume) gọi là đĩa logic
  - Ví dụ: trên máy tính PC có 1 ổ cứng và có thể chia thành các ổ C,D,E => đĩa logic
- Để quản lý file trên các đĩa logic, thông tin về file được lưu trong thư mục của đĩa
- Thư mục = $\sum$ các khoản mục ~ files
- Khoản mục chứa các thông tin về file: tên, kích thước, vị trí, kiểu file,… hoặc con trỏ tới nơi lưu trữ thông tin này
- Coi thư mục như 1 bảng, mỗi dòng là khoản mục ứng với 1 file

- Các cách lưu thông tin về file trong thư mục:
  - Toàn bộ thuộc tính của file được lưu trong thư mục, file chỉ chứa data => kích thước khoản mục, thư mục lớn
  - Một phần thuộc tính được lưu trữ luôn cùng với dữ liệu của file. Thư mục chỉ lưu thông tin tối thiểu cần thiết cho việc tìm kiếm vị trí file trên đĩa => kích thước giảm



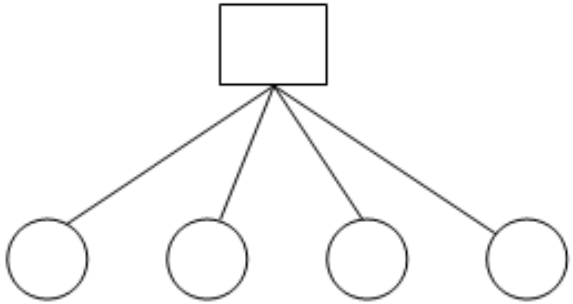
- Mở file:
  - HDH tìm trong thư mục khoản mục ứng với tên file cần mở
  - Đọc các thuộc tính và vị trí dữ liệu của file vào bảng chứa thông tin về các file đang mở
  - Nếu khoản mục trỏ tới CTDL khác chứa thuộc tính file, cấu trúc này sẽ được đọc vào bảng

IV. THƯ MỤC

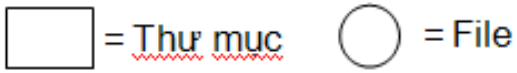
2. Các thao tác với thư mục

- Tìm kiếm file: cấu trúc thư mục phải cho phép tìm kiếm file theo tên file
- Tạo file: tạo khoản mục mới và thêm vào thư mục
- Xóa file: thông tin về file và khoản mục tương ứng bị xóa khỏi thư mục
- Duyệt thư mục: liệt kê các file trong thư mục và thông tin chứa trong khoản mục của file
- Đổi tên file: chỉ cần thực hiện với thư mục chứ không liên quan đến dữ liệu của file

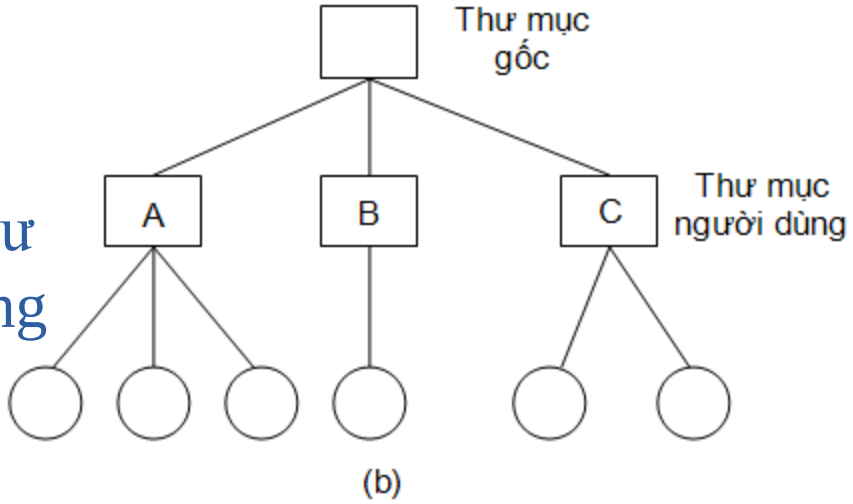
- Thư mục 1 mức:
  - Đơn giản nhất
  - Chỉ có 1 thư mục duy nhất và tất cả các file được giữ trong thư mục này
  - Khó chọn tên cho file
  - Tìm kiếm file khó



(a)

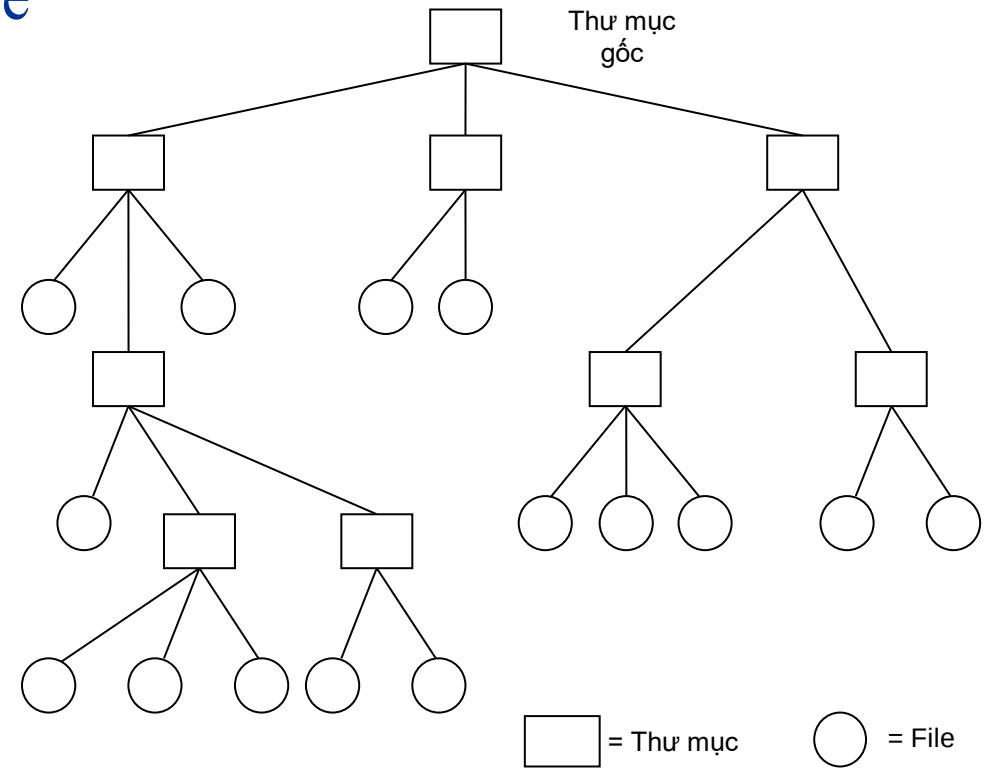


- Thư mục 2 mức:
  - Phân cho mỗi người dùng 1 thư mục riêng (UFD: User File Directory), chứa các file của mình
  - Khi người dùng truy cập file, file sẽ được tìm kiếm trong thư mục ứng với tên người đó
  - => các người dùng khác nhau có thể đặt tên file trùng nhau
  - Cô lập người dùng
  - Các file mà nhiều người dùng truy cập tới => chép vào từng thư mục của từng người dùng => lãng phí



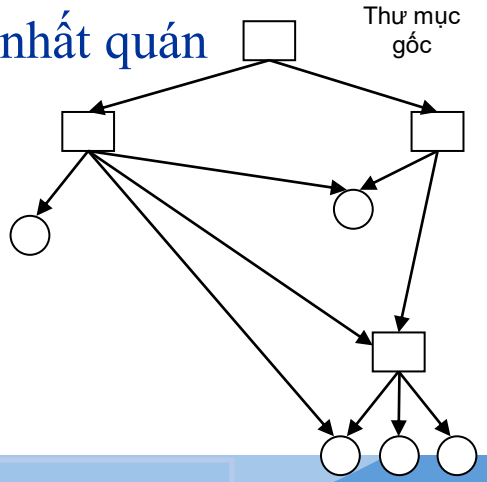
(b)

- Thư mục cấu trúc cây:
  - Thư mục con có thể chứa các thư mục con khác và các files
  - Hệ thống thư mục được biểu diễn phân cấp như 1 cây: cành là thư mục, lá là file



- Thư mục cấu trúc cây (tt):
  - Phân biệt khoản mục file và khoản mục của thư mục con: thêm bit đặc biệt trong khoản mục
    - 1: khoản mục của thư mục mức dưới
    - 0: khoản mục của file
  - Tại mỗi thời điểm, người dùng làm việc với thư mục hiện thời (current directory)
  - Tổ chức cây thư mục cho từng đĩa:
    - Trong hệ thống file như FAT của DOS, cây thư mục được xây cho từng đĩa. Hệ thống thư mục được coi là rừng, mỗi cây trên 1 đĩa
    - Linux: toàn hệ thống chỉ gồm 1 cây thư mục

- Thư mục cấu trúc đồ thị không tuần hoàn (acyclic graph):
  - Chia sẻ files và thư mục để có thể xuất hiện ở nhiều thư mục riêng khác nhau
  - Mở rộng của cấu trúc
  - cây: lá và cành có thể đồng thời thuộc về những cành khác nhau
  - Triển khai:
    - Sử dụng liên kết: con trỏ tới thư mục hoặc file khác
    - Tạo bản sao của file và thư mục cần chia sẻ và chứa vào các thư mục khác nhau => phải đảm bảo tính đồng bộ và nhất quán
  - Mềm dẻo nhưng phức tạp



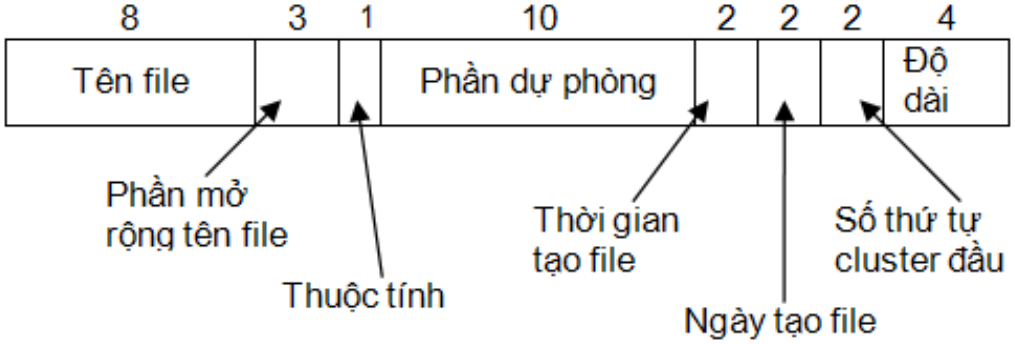
- Mô tả vị trí của file trong thư mục
- Đường dẫn tuyệt đối:
  - Đường dẫn từ gốc của cây thư mục, đi qua các thư mục trung gian, dẫn tới file
  - C:\bc\bin\bc.exe
- Đường dẫn tương đối:
  - Tính từ thư mục hiện thời
  - Thêm 2 khoản mục đặc biệt trong thư mục: “.”, “..”

- Danh sách:
  - Tổ chức thư mục dưới dạng danh sách các khoản mục
  - Tìm kiếm khoản mục được thực hiện bằng cách duyệt lần lượt danh sách
  - Thêm file mới vào thư mục:
    - Duyệt cả thư mục để kiểm tra xem khoản mục với tên file như vậy đã có chưa
    - Khoản mục mới được thêm vào cuối danh sách hoặc 1 ô trong bảng
  - Mở file, xóa file
  - Tìm kiếm trong danh sách chậm
  - Lưu trữ thư mục trong bộ nhớ

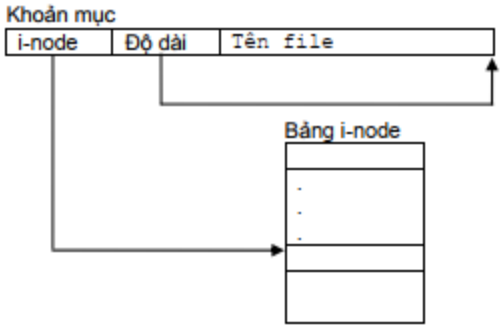
- Cây nhị phân:
  - Tăng tốc độ tìm kiếm nhờ CTDL có hỗ trợ sắp xếp
  - Hệ thống file NTFS của WinNT
- Bảng băm (hash table):
  - Dùng hàm băm để tính vị trí của khoản mục trong thư mục theo tên file
  - Thời gian tìm kiếm nhanh
  - Hàm băm phụ thuộc vào kích thước của bảng băm => kích thước bảng cố định

5. Tổ chức bên trong của thư mục

- Tổ chức thư mục của DOS:
  - Mỗi đĩa logic có cây thư mục riêng, bắt đầu từ thư mục gốc ROOT
  - Thư mục gốc được đặt ở phần đầu của đĩa, ngay sau sector khởi động BOOT và bảng FAT
  - Thư mục gốc chứa files và các thư mục con
  - Thư mục con có thể chứa files và các thư mục cấp dưới nữa
  - Được tổ chức dưới dạng bảng: mỗi khoản mục chiếm 1 dòng trong bảng và có kích thước cố định 32 bytes

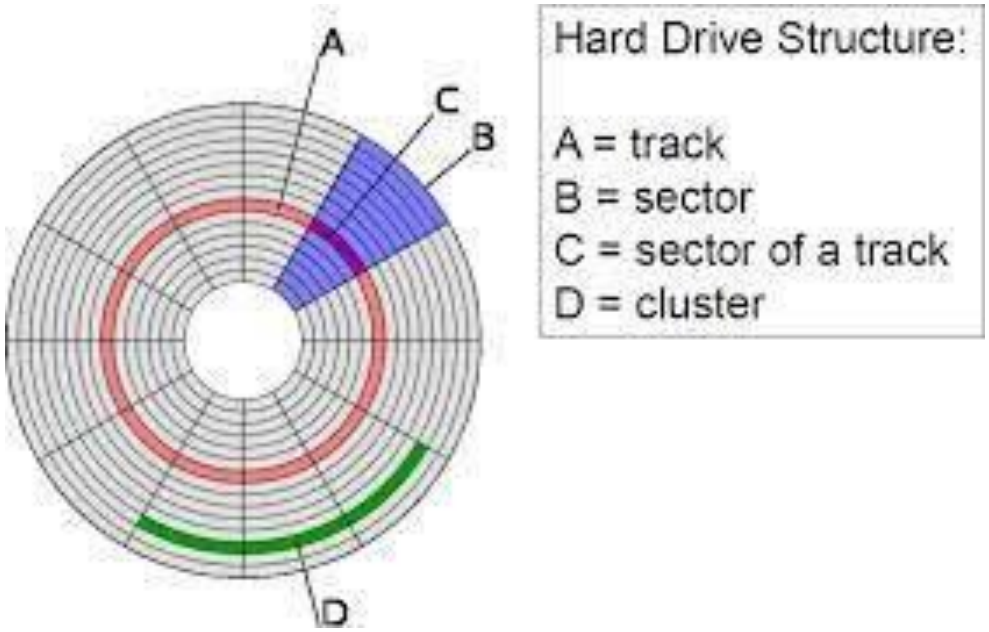


- Tổ chức thư mục của Linux:
  - Thư mục hệ thống file Ext2 của Linux có cách tổ chức đơn giản
  - Khoản mục chứa tên file và địa chỉ I-node
  - Thông tin còn lại về các thuộc tính file và vị trí các khối dữ liệu được lưu trên I-node chứ không phải thư mục
  - Kích thước khoản mục phụ thuộc vào độ dài tên file
  - Phần đầu của khoản mục có trường cho biết kích thước khoản mục



V. CẤP PHÁT KHÔNG GIAN CHO FILE

- Nhiệm vụ quan trọng của HDH trong việc quản lý hệ thống file là cấp phát không gian trên đĩa và các hiết bị nhớ ngoài khác để lưu trữ file và thư mục
- Đồng thời, ghi lại vị trí các khối nhớ đã cấp phát để có thể tiến hành truy cập về sau.



V. CẤP PHÁT KHÔNG GIAN CHO FILE

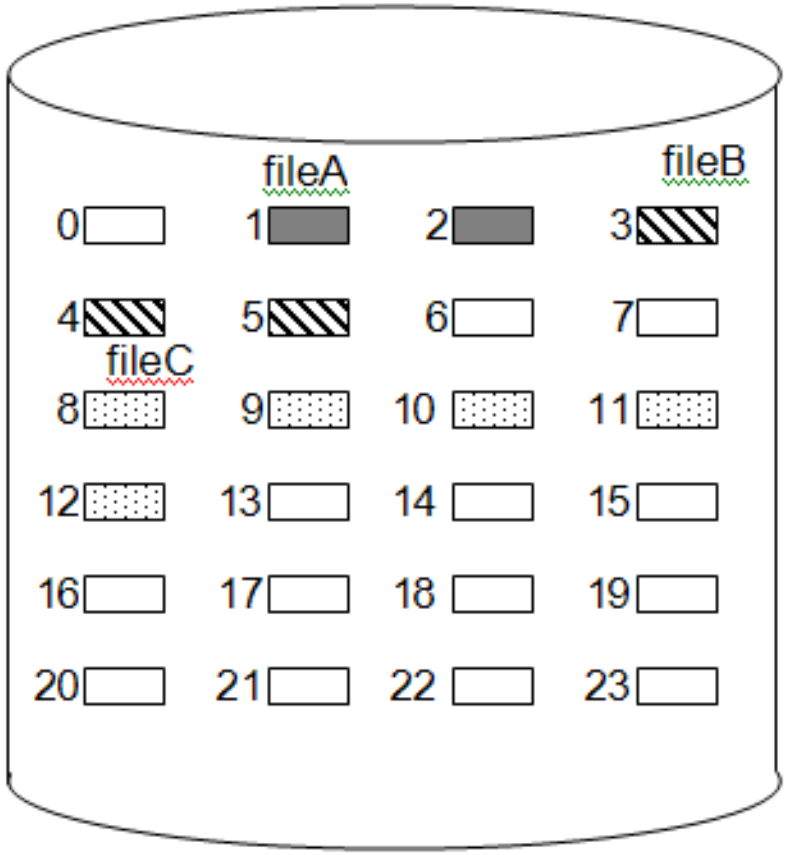
- Phép ánh xạ file: từ tên file có thể chỉ ra vị trí file trên đĩa
- Sơ bộ về tổ chức đĩa:
  - Thông tin được đọc/ghi theo từng khối sector
  - Nhóm các sector thành block hay cluster (khối)
- Trên đĩa: 1 file gồm 1 tập các khối. HDH chịu trách nhiệm cấp phát các khối cho file:
  - Không gian trên đĩa phải được cấp phát cho file
  - Cần theo dõi không gian trống sẵn sàng cho việc cấp phát
- Một số vấn đề:
  - Không gian tối đa yêu cầu cấp phát cho file 1 lần là bao nhiêu?
  - Không gian cấp phát cho file gọi là phần (portion). Kích thước phần ntn?

V. CẤP PHÁT KHÔNG GIAN CHO FILE

1. Cấp phát các khối liên tiếp

- Được cấp phát 1 khoảng không gian gồm các khối liên tiếp trên đĩa (các sector)
- Vị trí file trên đĩa được xác định bởi vị trí khối đầu tiên và độ dài (số khối) mà file đó chiếm
- Khi có yêu cầu cấp phát, HDH sẽ chọn 1 vùng trống có số lượng khối đủ cấp cho file đó
- Bảng cấp phát file chỉ cần 1 khoản mục cho 1 file, chỉ ra khối bắt đầu, và độ dài của file tính = khối
- Là cấp phát trước, sử dụng kích thước phần thay đổi

1. Cấp phát các khối liên tiếp (tt)

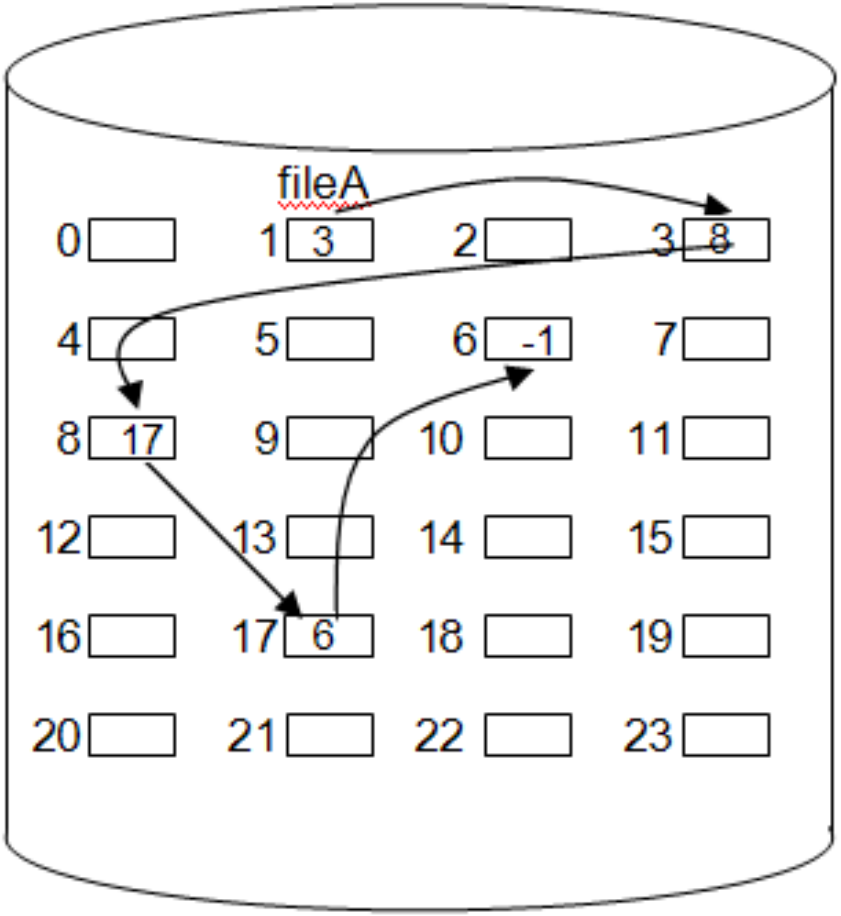


Thư mục

Tên file	Bắt đầu	Độ dài
fileA	1	2
fileB	3	3
fileB	8	5

- Ưu điểm:
  - Cho phép truy cập trực tiếp và tuần tự
  - Đơn giản, tốc độ cao
- Nhược điểm:
  - Phải biết trước kích thước file khi tạo
  - Khó tìm chỗ cho file
  - Gây phân mảnh ngoài: đó là hiện tượng vùng trống còn lại trên đĩa có kích thước quá nhỏ do vậy không thể cấp phát cho file có kích thước lơn hơn.

- Các khối được kết nối với nhau thành danh sách kết nối; phần đầu mỗi khối chứa con trỏ trỏ tới khối tiếp theo
- Các khối thuộc về 1 file có thể nằm ở vị trí bất kì trên đĩa
- Khoản mục của thư mục chứa con trỏ tới khối đầu tiên của file
- Khi file được cấp thêm khối mới, khối đó được thêm vào cuối danh sách
- HDH đọc lần lượt từng khối và sử dụng con trỏ để xác định khối tiếp theo



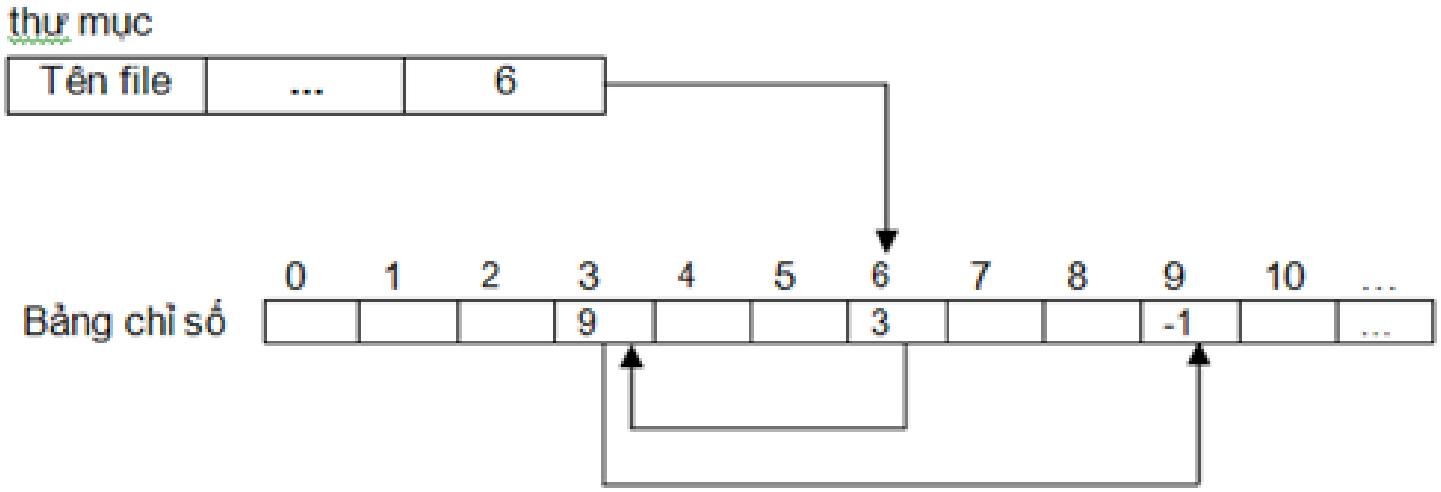
Thư mục

Tên file	Bắt đầu
fileA	1

- Ưu điểm:
  - Không bị phân mảnh ngoài
  - Không yêu cầu biết trước kích thước file lúc tạo
  - Dễ tìm vị trí cho file, khoản mục đơn giản
- Nhược điểm:
  - Không hỗ trợ truy cập trực tiếp
  - Tốc độ truy cập không cao
  - Giảm độ tin cậy và tính toàn vẹn của hệ thống file

3. Sử dụng danh sách kết nối trên bảng chỉ số

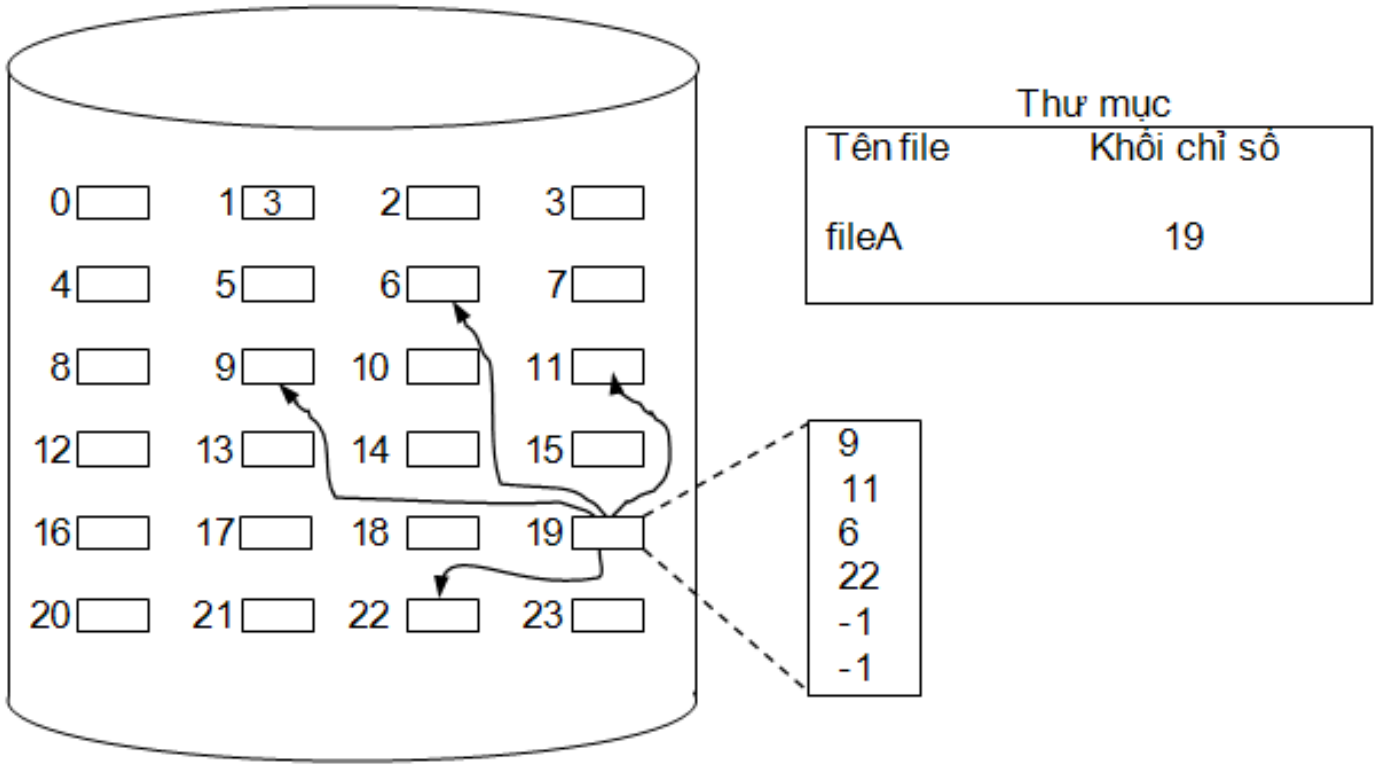
- Bảng chỉ số: mỗi ô của bảng ứng với 1 khối (sector) của đĩa
- Con trỏ tới khối tiếp theo của file được chứa trong ô tương ứng của bảng
- Mỗi đĩa logic có 1 bảng chỉ số được lưu ở vị trí xác định
- Kích thước mỗi ô trên bảng phụ thuộc vào số lượng khối trên đĩa



- Cho phép tiến hành truy cập file trực tiếp: đi theo chuỗi con trỏ chứa trong bảng chỉ mục
- Bảng FAT (File Allocation Table): được lưu ở đầu mỗi đĩa logic sau sector khởi động
- FAT12, FAT16, FAT32: mỗi ô của bảng có kích thước 12, 16, 32 bit

- Tất cả con trỏ tới các khối thuộc về 1 file được tập trung 1 chỗ
- Mỗi file có một mảng riêng của mình chứa trong một khối gọi là khối chỉ mục (I-node)
- Mảng chứa thuộc tính của file và vị trí các khối của file trên đĩa
- Ô thứ i của mảng chứa con trỏ tới khối thứ i của file
- Khoản mục của file trong thư mục chứa con trỏ tới khối chỉ mục này

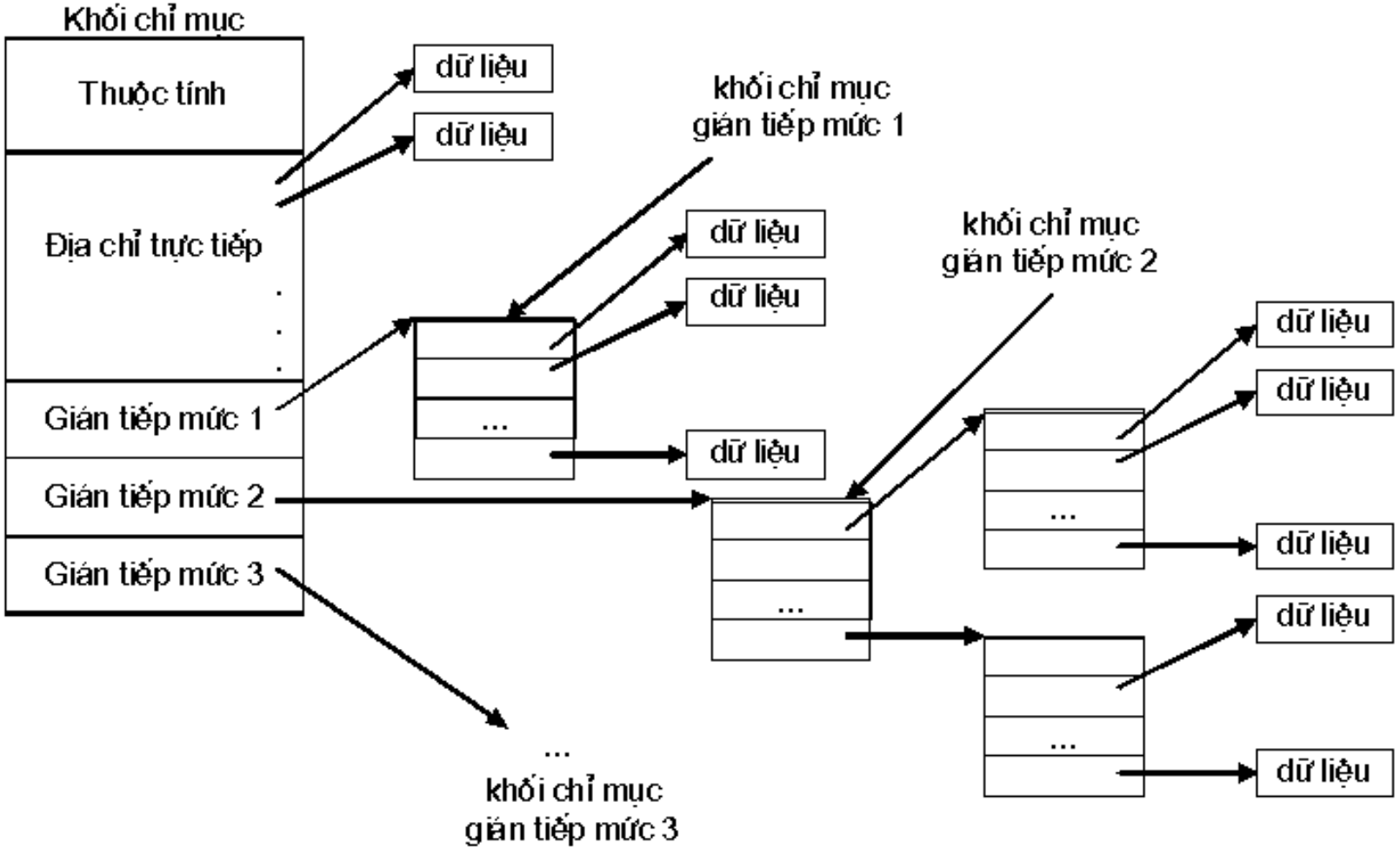
4. Sử dụng khối chỉ mục (index block/ node)



- Chọn kích thước I-node:
  - Nhỏ: tiết kiệm không gian nhưng không đủ con trỏ tới các khối nếu file lớn
  - Lớn: với file nhỏ chỉ chiếm 1 vài ô thì lãng phí
  - Giải pháp:
    - Thay đổi kích thước i-node = sử dụng danh sách kết nối
    - Sử dụng I-node có cấu trúc nhiều mức

4. Sử dụng khối chỉ mục (index block/ node)

- I-node cấu trúc nhiều mức: không trỏ trực tiếp tới ô chứa dữ liệu mà trỏ tới các ô khối chỉ muc sau đó cái ô này mới chứa đia chỉ.



- Ưu điểm:
  - Cho phép truy cập trực tiếp
  - Các khối thuộc 1 file không cần nằm liên tiếp nhau
- Nhược điểm:
  - Tốc độ truy cập file chậm

- Kích thước khối (cluster):
  - Kích thước khối lớn:
    - Giảm kích thước bảng chỉ mục, tăng tốc độ đọc file;
    - Bị phân mảnh trong
  - Kích thước khối nhỏ:
    - Mỗi file chiếm nhiều khối nhớ, nằm rải rác trên đĩa
    - Thời gian đọc file lâu
  - Chọn kích thước khối tùy thuộc:
    - Kích thước đĩa: đĩa lớn, chọn kích thước khối lớn => thời gian truy cập nhanh, đơn giản hóa việc quản lý
    - Kích thước file: hệ thống sử dụng nhiều file lớn, kích thước tăng và ngược
  - Kích thước khối thường là lũy thừa 2 của sector và nằm trong khoảng từ 512B tới 32 KB

1. Bảng bit

- Vector bit là mảng 1 chiều
- Mỗi ô có kích thước 1 bit tương ứng với một khối trên đĩa
- Khối được cấp phát có bít tương ứng là 0, khối trống: 1 hoặc ngược lại
- Dễ tìm 1 hoặc nhóm các khối trống liên tiếp
- Với đĩa có kích thước lớn, đọc toàn bộ vector bit vào MEM có thể đòi hỏi khá nhiều không gian nhớ

2. Danh sách kết nối

- Các khối trống được liên kết với nhau thành danh sách
- Mỗi khối trống chứa con trỏ chỉ tới khối trống tiếp theo
- Địa chỉ khối trống đầu tiên được lưu ở vị trí đặc biệt trên đĩa và được HDH giữ trong MEM khi cần làm việc với các file
- Đòi hỏi truy cập lần lượt khi cần duyệt danh sách này
- HDH có thể cấp phát ngay các khối ở đầu danh sách

3. danh sách vùng trống

- Các khối nằm liền nhau thường được cấp phát và giải phóng đồng thời
- Lưu vị trí khối trống đầu tiên của vùng các khối trống liên tiếp và số lượng các khối trống nằm liền sau đó
- Thông tin trên được lưu vào danh sách riêng

- Phát hiện và loại trừ khối hỏng
  - Phương pháp 1:
    - Một sector trên đĩa được dành riêng chứa danh sách các khối hỏng
    - Một số khối không hỏng được dành riêng để dự trữ
    - Các khối hỏng sẽ được thay thế bởi các khối dự trữ bằng cách thay thế địa chỉ
    - Truy cập tới khối hỏng thành truy cập tới khối dự trữ
  - Phương pháp 2:
    - Tập trung tất cả các khối hỏng thành 1 file
    - => được coi như đã cấp phát và không được sử dụng nữa

VII. ĐỘ TIN CẬY CỦA HỆ THỐNG FILE

2. Sao dự phòng

- Tạo ra một bản sao của đĩa trên một vật mang khác
- Sao lưu toàn bộ (full backup):
  - Ghi toàn bộ thông tin trên đĩa ra vật mang tin khác
  - Chắc chắn nhưng tốn nhiều thời gian
- Sao lưu tăng dần (incremental backup):
  - Được sử dụng sau khi đã tiến hành full backup ít nhất 1 lần
  - Chỉ ghi lại các file đã bị thay đổi sau lần sao lưu cuối cùng
  - Hệ thống lưu trữ thông tin về các lần lưu trữ file
  - DOS: file thay đổi, archive bit =1
- Kết hợp:
  - Full backup: hàng tuần/ tháng
  - Incremental backup: hàng ngày

3. Kiểm tra tính toàn vẹn của hệ thống file

- Hệ thống file chứa nhiều CTDL có mối liên kết => thông tin về liên kết bị hư hại, tính toàn vẹn của hệ thống bị phá vỡ
- Các khối không có mặt trong danh sách các khối trống, đồng thời cũng không có mặt trong một file nào
- Một khối có thể vừa thuộc về một file nào đó vừa có mặt trong danh sách khối trống
- HDH có các chương trình kiểm tra tính toàn vẹn của hệ thống file, được chạy khi hệ thống khởi động, đặc biệt là sau sự cố

- Ví dụ trong hệ UNIX:
  - Tạo hai số đếm cho mỗi khối:
    - Số đếm thứ nhất: số lần khối đó xuất hiện trong danh sách khối trống.
    - Số đếm thứ hai: số lần khối xuất hiện trong file
  - Tất cả số đếm được khởi tạo bằng 0
  - Duyệt danh sách khối trống và toàn bộ i-node của các file
    - Một khối xuất hiện trong danh sách khối trống, số đếm tương ứng thứ nhất được tăng một đơn vị
    - Nếu khối xuất hiện trong i-node của file, số đếm tương ứng thứ hai được tăng một đơn vị
  - Tổng 2 số đếm =1

Số thứ tự khối

Số lần xuất hiện trong danh sách trống	0	1	2	3	4	5
	0	1	0	1	0	1

Số thứ tự khối

Số lần xuất hiện trong file	0	1	2	3	4	5
	0	0	1	1	3	2

- Giao tác (transaction) là một tập hợp các thao tác cần phải được thực hiện trọn vẹn cùng với nhau
- Với hệ thống file: mỗi giao tác sẽ bao gồm những thao tác thay đổi liên kết cần thực hiện cùng nhau
- Toàn bộ trạng thái hệ thống file được ghi lại trong file log
- Nếu giao tác không được thực hiện trọn vẹn, HDH sử dụng thông tin từ log để khôi phục hệ thống file về trạng thái không lỗi trước khi thực hiện giao tác

- Ngăn cản việc truy cập trái phép các thông tin lưu trữ trong file và thư mục
- Hạn chế các thao tác truy cập tới file hoặc thư mục
- Dùng mật khẩu:
  - Người dùng phải nhớ nhiều mật khẩu
  - Mỗi khi thao tác với tài nguyên lại gõ mật khẩu

- Sử dụng danh sách quản lý truy cập ACL (Access Control List)
  - Mỗi file được gán danh sách đi kèm, chứa thông tin định danh người dùng và các quyền người đó được thực hiện với file
  - ACL thường được lưu trữ như thuộc tính của file/ thư mục
  - Thường được sử dụng cùng với cơ chế đăng nhập
  - Các quyền truy cập cơ bản:
    - Quyền đọc (r)
    - Quyền ghi, thay đổi (w)
    - Quyền xóa
    - Quyền thay đổi chủ file (change owner)

IX. HỆ THỐNG FILE FAT

- 3 phiên bản: FAT12, FAT16, FAT32
- Chữ số chỉ kích thước ô bảng FAT tương ứng 12, 16 và 32 bit

1. ĐĨA LOGIC

- Đơn vị cấp phát không gian trên đĩa (khối logic) là cluster (lũy thừa 2 của số lượng sector)

Boot sector và các khối dự phòng	Bảng FAT1	Bảng FAT2	Thư mục gốc (chỉ có trên FAT12 và FAT16)	Phần còn lại cho tới cuối đĩa chứa các file và thư mục của đĩa lô gic

- Boot sector:
  - Sector đầu tiên của đĩa logic
  - Chứa thông tin mô tả cấu trúc đĩa logic: kích thước sector, cluster, kích thước bảng FAT
  - Chứa mã chương trình mồi để tải HĐH nếu đĩa logic là đĩa khởi động
- FAT: bảng chỉ số quản lý cấp phát khối cho file
- Thư mục gốc ROOT
- Vùng dữ liệu: chứa các file và thư mục của đĩa logic

BOOT SECTOR

32 Byte đầu tiên

Vị trí	Độ dài	Ý nghĩa
0	3	Lệnh Jump. Chỉ thị cho CPU bỏ qua phần thông tin và nhảy tới thực hiện phần mã mồi của hệ điều hành nếu đây là đĩa mồi hệ điều hành.
3	8	Tên hãng sản xuất, bổ sung dấu trắng ở cuối cho đủ 8B. Ví dụ: IBM 3.3, MSDOS5.0.v.v.
11	2	Bytes per sector. Kích thước sector tính bằng byte. Giá trị thường gặp là 512 đối với đĩa cứng. Đây cũng là vị trí bắt đầu của Khối Thông số BIOS (BIOS Parameter Block, viết tắt là BPB)
13	1	Sectors per cluster. Số sector trong một cluster, luôn là lũy thừa của 2 và không lớn hơn 128.
14	2	Reserved sectors. Số lượng sector dành cho vùng đầu đĩa đến trước FAT, bao gồm boot sector và các sector dự phòng.
16	1	Số lượng bảng FAT. Thường bằng 2.
17	2	Số khoản mục tối đa trong thự mục gốc ROOT. Chỉ sử dụng cho FAT12 và FAT16. Bằng 0 với FAT32.
19	2	Total sector. Tổng số sector trên đĩa. Nếu bằng không thì số lượng sector được ghi bằng 4 byte tại vị trí 0x20.
21	1	Mô tả loại đĩa. Ví dụ 0xF0 là đĩa mềm 3.5" hai mặt với 80 rãnh trên mỗi mặt, 0xF1 là đĩa cứng .v.v.
22	2	Sectors per FAT. Kích thước FAT tính bằng sector (đối với FAT12/16)
24	2	Sectors per track. Số sector trên một rãnh.
26	2	Number of heads. Số lượng đầu đọc (mặt đĩa được sử dụng)
28	4	Hidden sectors. Số lượng sector ẩn.
32	4	Total sector. Tổng số sector trên đĩa cho trường hợp có nhiều hơn 65535.

Các byte tiếp theo với FAT12/16

Vị trí	Độ dài	Ý nghĩa
36	1	Số thứ tự vật lý của đĩa (0: đĩa mềm, 80h: đĩa cứng .v.v.)
37	1	Dự phòng
38	1	Dấu hiệu của phần mã mồi. Chứa giá trị 0x29 (ký tự ‘)’) hoặc 0x28.
39	4	Số xê ri của đĩa (Volume Serial Number) được tạo lúc format đĩa
43	11	Volume Label. Nhãn của đĩa được tạo khi format.
54	8	Tên hệ thống file FAT, ví dụ"FAT12 ", "FAT16 ".
62	448	Mã mồi hệ điều hành, đây là phần chương trình tải hệ điều hành khi khởi động.
510	2	Dấu hiệu Boot sector (0x55 0xAA)

Các byte tiếp theo với FAT32

Vị trí	Độ dài	Ý nghĩa
36	4	Sectors per FAT. Kích thước FAT tính bằng sector.
0x28	2	Cờ của FAT
0x2a	2	Version. Phiên bản.
0x2c	4	Số thứ tự của cluster đầu tiên của thư mục gốc root.
0x30	2	Số sector của Information Sector. Đây là phần nằm trong số sector dự phòng ngay sau boot sector.
0x32	2	Số thứ tự sector đầu tiên của bản sao của boot sector (nếu có)
0x34	12	Dự phòng
0x40	1	Số thứ tự vật lý của đĩa
0x41	1	Dự phòng
0x42	1	Dấu hiệu của phần mã mồi mở rộng.
0x43	4	Số xê ri của đĩa (Volume Serial Number)
0x47	11	Volume Label
0x52	8	"FAT32 "
0x5a	420	Mã mồi hệ điều hành
0x1FE	2	Dấu hiệu Boot sector (0x55 0xAA)

BẢNG FAT

- Quản lý các cluster trên đĩa và các file theo nguyên tắc:
  - Các khối thuộc cùng 1 file được liên kết thành 1 danh sách
  - Con trỏ được chứa trong ô tương ứng của bảng FAT
- Mỗi ô trong bảng FAT tương ứng với một cluster trên đĩa, chứa 1 trong các thông tin:
  - STT cluster tiếp theo trong danh sách các khối của file
  - Dấu hiệu kết thúc nếu ô tương ứng với cluster cuối cùng của file
  - Dấu hiệu đánh dấu cluster hỏng, không được sử dụng
  - Dấu hiệu đánh dấu cluster dự phòng
  - Bằng 0 nếu cluster trống, chưa cấp phát cho file nào

BẢNG FAT

- Cluster đầu tiên của vùng dữ liệu được đánh STT là 2
- 2 ô đầu tiên của bảng FAT không dùng để quản lý cluster

FAT12	FAT16	FAT32	Ý nghĩa
0x000	0x0000	0x00000000	Cluster trống
0x001	0x0001	0x00000001	Cluster dự phòng, không được sử dụng
0x002–0xFEF	0x0002–0xFFEF	0x00000002–0x0FFFFFEF	Cluster đã được cấp cho file. Chứa số thứ tự cluster tiếp theo của file.
0xFF0–0xFF6	0xFFF0–0xFFF6	0x0FFFFFF0–0x0FFFFFF6	Cluster dự phòng
0xFF7	0xFFF7	0x0FFFFFF7	Cluster hỏng.
0xFF8–0xFFF	0xFFF8–0xFFFF	0x0FFFFFF8–0x0FFFFFFF	Cluster cuối cùng của file

IX. HỆ THỐNG FILE FAT
THƯ MỤC GỐC (ROOT)

- Mỗi thư mục được lưu trong bảng thư mục, thực chất là 1 file đặc biệt chứa các khoản mục của thư mục
- Mỗi khoản mục chứa thông tin về một file hoặc thư mục con của thư mục đang xét
- Với FAT12/16, thư mục trên cùng của đĩa được chứa trong 1 vùng đặc biệt gọi là thư mục gốc
- Các thư mục mức thấp hơn/ thư mục gốc của FAT32 được chứa trong vùng dữ liệu trên đĩa cùng với các file
- Mỗi thư mục gồm các khoản mục 32 byte xếp liền nhau

THƯ MỤC GỐC (ROOT)

Vị trí	Độ dài	Mô tả
0	8	Tên file, thêm bằng dấu trắng ở cuối nếu ngắn hơn 8 byte
8	3	Phần mở rộng, thêm bằng dấu trắng ở cuối nếu ngắn hơn 3 byte
11	1	Byte thuộc tính của file. Các bit của byte này nếu bằng 1 sẽ có ý nghĩa như sau: Bit 0: file chỉ được đọc; Bit 1: file ẩn; Bit 2: file hệ thống; Bit 3: Volume label; Bit 4: thư mục con Bit 5: archive; Bit 6: thiết bị nhớ khác (dùng cho hệ điều hành); Bit 7: không sử dụng Byte thuộc tính bằng 0x0F là dấu hiệu của file tên dài.
12	1	Dự phòng
13	1	Thời gian tạo file tính theo đơn vị 10ms, giá trị từ 0 đến 199
14	2	Thời gian tạo file theo format sau: bit 15-11: giờ (0-23); bit 10-5: phút (0-59); bit 4-0: giây/2 (0-29)
16	2	Ngày tạo file theo format sau. Bit 15-9: năm (0-1980, 127 =2107); bit 8-5: tháng (1-12); bit 4-0: ngày (1-31)
18	2	Ngày truy cập cuối, theo format như ngày tạo file
20	2	2 byte cao của số thứ tự cluster đầu tiên của file trong FAT32
22	2	Thời gian sửa file lần cuối, theo format thời gian tạo file
24	2	Ngày sửa file lần cuôi, theo format như ngày tạo file
26	2	Số thứ tự cluster đầu tiên của file trong FAT12/16.
28	4	Kích thước file tính bằng byte. Bằng 0 với thư mục con

HÀM ĐỌC ĐĨA

- int absread(int drive, int nsects, long lsect, void *buffer)
  - drive: ổ đĩa cần đọc, A: 0, B:1, C:2
  - nsects: số sector cần đọc
  - lsect: vị trí sector bắt đầu đọc
  - buffer: vùng nhớ lưu nội dung thông tin cần đọc

ĐỌC FAT

- Vị trí sector bắt đầu: reserved sector (byte 14, 15 trong bootsector)
- Tổng số sector cần đọc: sectors per FAT (byte 22, 23)

IX. HỆ THỐNG FILE FAT
ĐỌC ROOT

- Vị trí sector bắt đầu: reserved sector + NoOfFATs * sectors per FAT
- Tổng số sector cần đọc: NoOfRootEntries * 32 /BytesPerSector

BÀI TẬP THỰC HÀNH

1. Viết chương trình để hiển thị thông tin boot sector
2. Viết chương trình đọc FAT và in nội dung 100 ô fat đầu tiên lên màn hình
3. Viết chương trình đọc ROOT và in nội dung giống lệnh DIR
4. Cho 1 tên file thuộc ROOT. Viết chương trình tìm tất cả các cluster của file đó
5. Viết chương trình đếm số cluster trống trong 100 cluster đầu tiên của ổ đĩa

Cho 1 tên file thuộc ROOT. Viết chương trình tìm tất cả các cluster của file đó

1. Đọc Boot sector: lưu vào bs
2. Đọc ROOT: lưu trong ROOT
3. Đọc bảng FAT: lưu trong mảng FAT
4. Giả sử tên file đưa vào lưu trong chuỗi tenfile;
5. Kiểm tra xem file "tenfile" có nằm trong thư mục gốc???
    1. - duyệt lần lượt các khoản mục (i từ 0 tới bs->root_entries) xem có khoản mục nào mà ROOT[i]->filename trùng với "tenfile" nhập vào hay không??
    2. Giả sử ROOT[i]->filename =="tenfile" => khoản mục i trong thư mục gốc quản lý "tenfile" đang xét

3. Cluster đầu tiên lưu trữ data cho file “tenfile” là ROOT[i]->first_cluster

6. giả sử ROOT[i]->first_cluster = x;

```
While (x<0xfff8)
{
    cout<<x;
    x= FAT[x];

}
```

Viết chương trình đếm số cluster trống của ổ đĩa

- đọc boot sector absread (2, 1, 0, bs);
- Đoc FAT absread(2, bs->sector_per_FAT, bs->reserverd_sector, FAT);
- Int dem=0;
- For (int I =2; i<(bs->sector_per_FAT * bs->bytes_per_sector)/2; i++)
- If (FAT[i]==0) dem++;

Hệ thống FAT16

Bài 1: Viết đoạn chương trình đếm số cluster trống trong 100 cluster đầu tiên của ổ đĩa D.

Bài 2: Viết đoạn chương trình in nội dung của 50 ô FAT đầu tiên của ổ đĩa C ra màn hình

Bài 3: Giả sử bảng FAT đã được đọc vào bộ nhớ tại địa chỉ << int *fat>> . Giả sử một file được lưu trữ trên cluster đầu tiên là n. Viết đoạn chương trình liệt kê các cluster thuộc về file đó.

```
struct bootsector
{
        char jump_instruction[3];
        char OEM_ID[8];
        int bytes_per_sector;
        char sector_per_cluster;
        int reserved_sectors;
        char number_of_fats;
        int root_entries;
        int small_sectors;
        char media_descriptor;
        int sectors_per_fat;
        int sectors_per_track;
        int number_of_heads;
        long hidden_sectors;
        long large_sectors;
        char physical_drive_number;
        char reserved;
        char extended_boot_signature;
        char volume_serial_number[4];
        char volume_lable[11];
        char file_system_type[8];
        char bootstrap_code[448];
        char end_of_sector_marker[2];

};
   bootsector *bs= new bootsector();

   absread(2,1,0,bs);
```

HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

BÀI GIẢNG MÔN

HỆ ĐIỀU HÀNH

Bộ môn: Khoa học máy tính- Khoa CNTT1

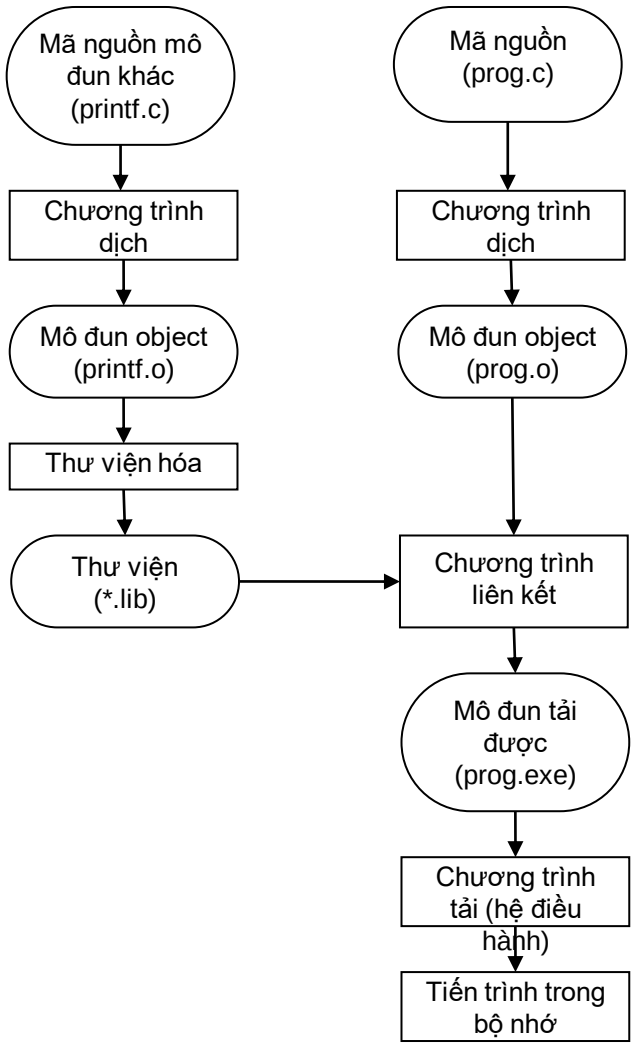
CHƯƠNG 3: QUẢN LÝ BỘ NHỚ

NỘI DUNG

1. Địa chỉ và các vấn đề liên quan
2. Một số cách tổ chức chương trình
3. Các yêu cầu quản lý bộ nhớ
4. Phân chương bộ nhớ
5. Phân trang bộ nhớ
6. Phân đoạn bộ nhớ
7. Bộ nhớ ảo

I. ĐỊA CHỈ VÀ CÁC VẤN ĐỀ LIÊN QUAN

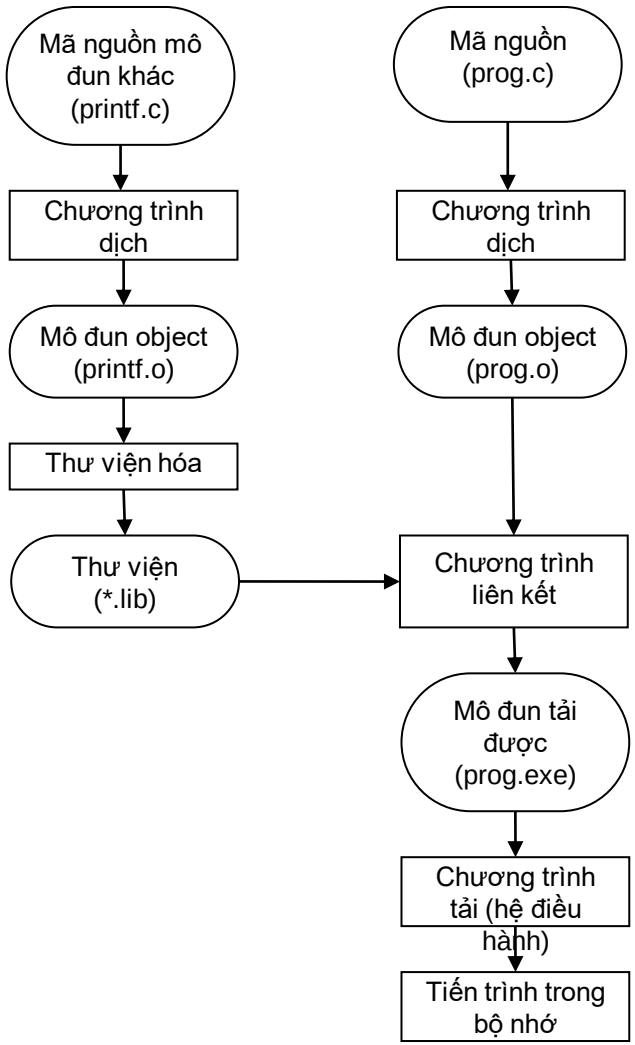
- Vấn đề gán địa chỉ:
  - Chương trình máy tính thường không được viết trực tiếp bởi ngôn ngữ máy, trừ thế hệ máy tính đầu tiên, mà viết trên ngôn ngữ bậc cao hoặc hợp ngữ.
  - Các chương trình phải trải qua một quá trình dịch và liên kết trước khi trở thành chương trình có thể tải vào và thực hiện.



I. ĐỊA CHỈ VÀ CÁC VẤN ĐỀ LIÊN QUAN

- Vấn đề gán địa chỉ:
  - Khi viết chương trình, sử dụng địa chỉ dưới dạng tên (biến, hàm)
  - Khi dịch, chương trình dịch ánh xạ các tên đó theo địa chỉ tương đối tính từ đầu file obj (biến, hàm)
  - Chương trình liên kết ánh xạ tiếp địa chỉ đó thành địa chỉ tương đối tính từ đầu chương trình
- HDH đọc chương trình vào bộ nhớ để thực hiện; vị trí trong bộ nhớ không biết trước

=> HDH cần có khả năng gán địa chỉ



- Địa chỉ logic:
  - Phân biệt với địa chỉ vật lý, CTUD chỉ quan tâm tới địa chỉ địa chỉ logic còn địa chỉ vật lý là do HDH.
  - Gán cho các lệnh và dữ liệu không phụ thuộc vào vị trí cụ thể tiến trình trong bộ nhớ
  - Chương trình ứng dụng chỉ nhìn thấy và làm việc với địa chỉ logic này
  - Là địa chỉ tương đối tức là mỗi phần tử của chương trình được gán một địa chỉ tương đối đối với một vị trí nào đó

- Địa chỉ vật lý:
  - Là địa chỉ chính xác trong bộ nhớ máy tính
  - Các mạch nhớ sử dụng để truy nhập tới chương trình và dữ liệu
- Địa chỉ logic được chuyển thành địa chỉ vật lý nhờ khối ánh xạ địa chỉ. (MMU=Memory Mapping Unit)

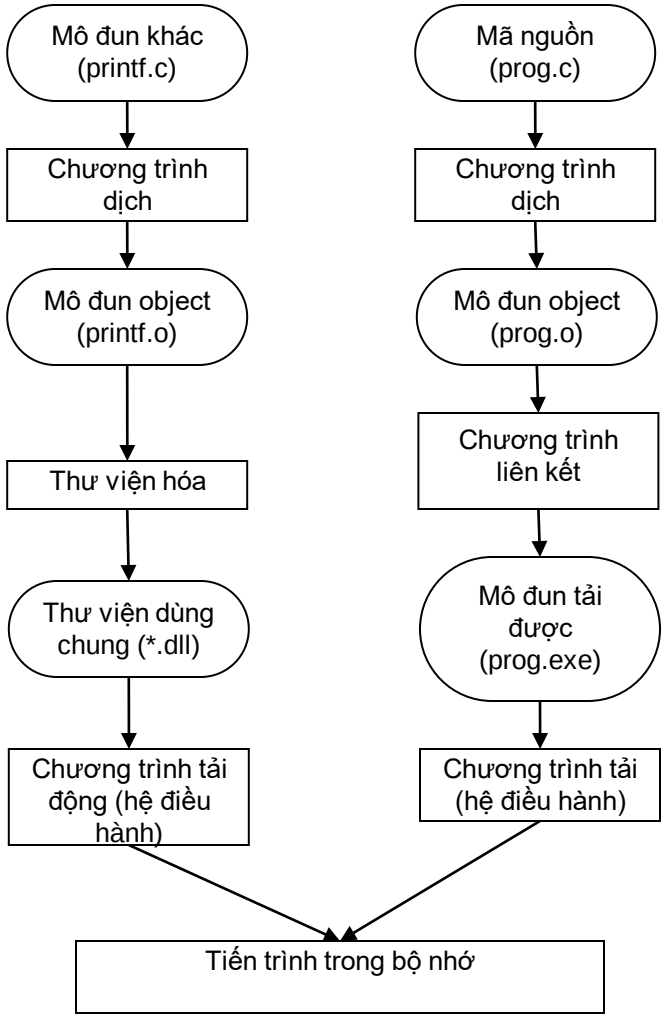
II. MỘT SỐ CÁCH TỔ CHỨC CHƯƠNG TRÌNH

1. Tải trong quá trình thực hiện

- Hàm chưa bị gọi thì chưa tải vào bộ nhớ
- Chương trình chính được load vào bộ nhớ và chạy
- Khi có lời gọi hàm:
  - Chương trình sẽ kiểm tra hàm đó được tải vào chưa.
  - Nếu chưa, chương trình sẽ tiến hành tải sau đó ánh xạ địa chỉ hàm vào không gian chung của chương trình và thay đổi bảng địa chỉ để ghi lại các ánh xạ đó
- Lập trình viên đảm nhiệm, HDH cung cấp các hàm thư viện cho tải động

2. Liên kết động và thư viện dùng chung

- Liên kết tĩnh: các hàm và thư viện được liên kết luôn vào mã chương trình
- Kích thước chương trình khi đó sẽ bằng kích thước chương trình vừa được dịch + kích thước các hàm thư viện
- => các hàm sẽ có mặt lặp đi lặp lại trong các chương trình => lãng phí không gian cả trên đĩa và bộ nhớ trong
- Giải quyết: sử dụng kỹ thuật liên kết động. Trong giai đoạn liên kết, không kết nối các hàm thư viện vào chương trình mà chỉ chèn các thông tin về hàm thư viện đó (stub).



2. Liên kết động và thư viện dùng chung

- Các modul thư viện được liên kết trong quá trình thực hiện:
  - Không giữ bản sao các modul thư viện mà tiến trình giữ đoạn mã nhỏ chứa thông tin về modul thư viện
  - Khi đoạn mã nhỏ được gọi, nó kiểm tra modul tương ứng đã có trong bộ nhớ chưa. Nếu chưa, thì tải phần còn lại và dùng.
  - Lần tiếp theo cần sử dụng, modul thư viện sẽ được chạy trực tiếp
  - Mỗi modul thư viện chỉ có 1 bản sao duy nhất chứa trong MEM
- Cần hỗ trợ từ HDH

1. Cấp phát lại

- Cần có khả năng tráo đổi các tiến trình vào và ra ngoài MEM để tối đa sử dụng vi xử lý
- Không thể yêu cầu tiến trình được chuyển lại vào MEM thì phải vào đúng chỗ nó đã dùng trước khi bị chuyển ra

2. Bảo vệ

- Mỗi tiến trình phải được bảo vệ khỏi các tham chiếu không mong muốn từ các tiến trình khác vào vùng nhớ dành cho mình
- Mọi tham chiếu bộ nhớ của 1 tiến trình phải được kiểm tra lúc chạy
- Khi 1 vùng nhớ đã dùng cho tiến trình này thì nó không cho phép tiến trình khác tham chiếu vào vùng nhớ đang dùng đó.
- HDH không đoán trước được mọi tham chiếu MEM => phần cứng VXL đảm nhiệm

3. Chia sẻ

- Nhiều tiến trình cần và được phép truy cập vào cùng 1 vùng nhớ
- Các tiến trình đang cộng tác cần chia sẻ truy nhập tới 1 cấu trúc dữ liệu
- => Phải cho phép truy cập tới các vùng chia sẻ

- Cấu trúc logic:
  - MEM được cấu trúc 1 cách tuyến tính gồm các byte, còn chương trình được tổ chức thành các modul
  - Phải đáp ứng để:
    - Các modul có thể được viết và thông dịch 1 cách độc lập
    - Mức độ bảo vệ có thể khác nhau
    - Modul có thể được chia sẻ giữa các tiến trình

- Cấu trúc vật lý:
  - 2 mức:
    - Bộ nhớ chính: nhanh; chi phí cao, dung lượng ít
    - Bộ nhớ phụ: dung lượng lớn, cho phép lưu chương trình và dữ liệu trong thời gian dài
  - Hệ thống có trách nhiệm chuyển đổi thông tin giữa 2 mức

IV. KỸ THUẬT PHÂN CHƯƠNG BỘ NHỚ

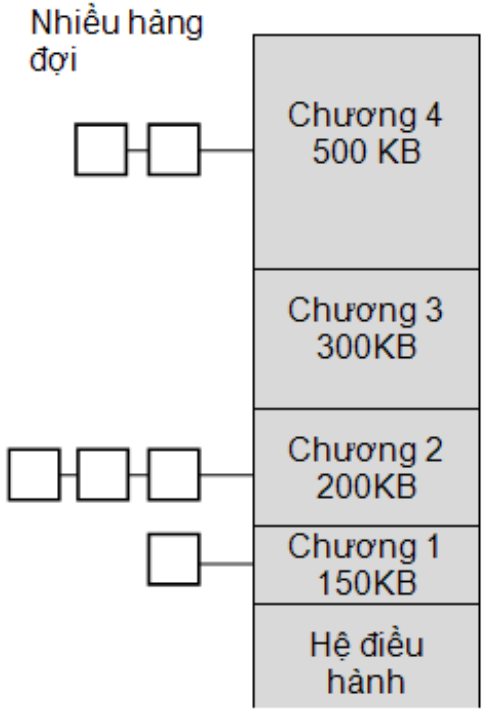
- Để thực hiện tiến trình, HDH cần cấp phát cho tiến trình không gian nhớ cần thiết.
- Việc cấp phát và quản lý vùng nhớ là chức năng quan trọng của HDH.
- Một kỹ thuật cấp phát đơn giản nhất => mỗi tiến trình được cấp một vùng bộ nhớ liên tục
- HDH tiến hành chia bộ nhớ thành các phần liên tục là chương (partition), mỗi tiến trình sẽ được cung cấp một chương để chứa lệnh và dữ liệu của mình.
- Quá trình phân chia bộ nhớ thành chương như vậy gọi là phân chương bộ nhớ.
- Tùy thuộc việc lựa chọn vị trí và kích thước của chương, có thể phân biệt phân chương cố định và phân chương động

1. Phân chương cố định

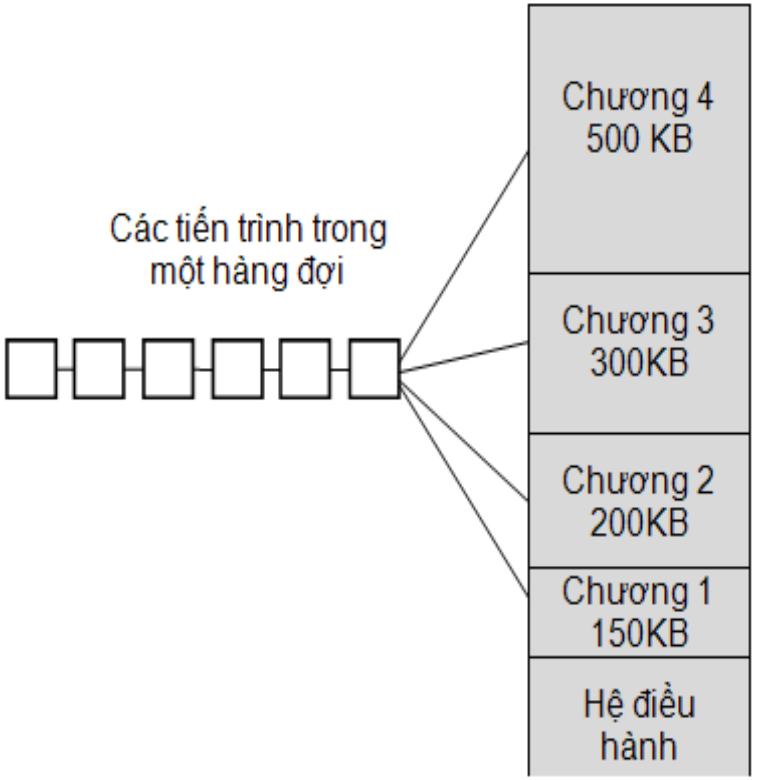
- Chia MEM thành các chương với kích thước cố định ở những vị trí cố định
- Mỗi chương chứa được đúng một tiến trình => số tiến trình tối đa có thể chứa đồng thời bị giới hạn bởi số lượng chương.
- Khi được tải vào, tiến trình được cấp phát một chương. Sau khi tiến trình kết thúc, HDH giải phóng chương và chương có thể được cấp phát cho tiến trình mới.
- Kích thước các chương bằng nhau:
  - Đơn giản
  - Kích thước chương trình > kích thước chương => không thể cấp phát
  - Gây phân mảnh trong

1. Phân chương cố định

- Kích thước các chương khác nhau: Có hai cách lựa chọn chương nhớ để cấp cho tiến trình đang chờ đợi
  - Chọn chương có kích thước nhỏ nhất: cần có hàng đợi lệnh cho mỗi chương.
  - Mỗi chương có một hàng đợi riêng, tiến trình có kích thước phù hợp với chương nào sẽ nằm trong hàng đợi của chương đó
    - Giảm phân mảnh trong, tối ưu cho từng chương => tiết kiệm bộ nhớ.
    - Do mỗi chương có một hàng đợi riêng nên có thời điểm hàng đợi của chương lớn hơn thì rỗng, trong khi hàng đợi của chương nhỏ hơn thì không chứa tiến trình nào => Hệ thống không tối ưu



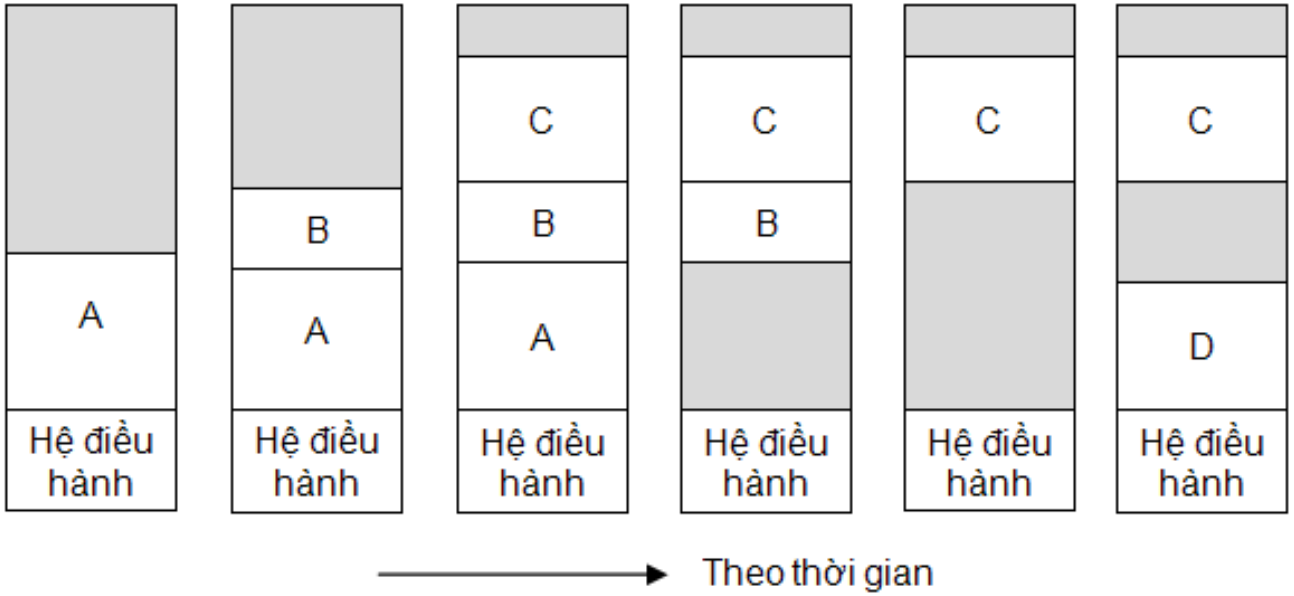
- Kích thước các chương khác nhau:
  - Dùng hàng đợi chung cho tất cả các chương:
    - Mỗi khi có một chương trống, tiến trình nằm gần đầu hàng đợi nhất và có kích thước phù hợp với chương nhất sẽ được tải và thực hiện
    - Khi 1 chương được giải phóng: chọn tiến trình gần đầu hàng đợi nhất và có kích thước phù hợp nhất



- Ưu điểm: đơn giản, ít xử lý
- Nhược điểm:
  - Số lượng chương xác định tại thời điểm tạo hệ thống hạn chế số lượng tiến trình hoạt động
  - Kích thước chương thiết lập trước: không hiệu quả

2. phân chương động

- Kích thước, số lượng và vị trí chương đều có thể thay đổi
- Khi có yêu cầu, HDH cấp cho tiến trình 1 chương có kích thước đúng bằng tiến trình đó đang cần.
- Khi tiến trình kết thúc sẽ tạo vùng trống trong MEM
- Các vùng trống nằm cạnh nhau được nhập lại thành vùng lớn hơn. Các vùng trống bộ nhớ cũng có thể được liên kết thành một danh sách kết nối



2. Phân chương động

- Tránh phân mảnh trong
- Gây phân mảnh ngoài: dồn những vùng trống nhỏ thành lớn (nén)
- Sử dụng các chiến lược cấp chương
  - Chọn vùng thích hợp đầu tiên (first – fit)
  - Vùng thích hợp nhất (best fit)
  - Vùng không thích hợp nhất (worst fit)
  - 250k, 180k, 500k

- Các chương và khối trống có kích thước là lũy thừa của 2^k ($L \le k \le H$): 2^L: kích thước nhỏ nhất của chương; 2^H : kích thước MEM
- Đầu tiên, toàn bộ không gian nhớ là 2^H , yêu cầu cấp vùng nhớ S
  - $2^{H-1} < S \le 2^H$: cấp cả 2^H
  - Chia đôi thành 2 vùng 2^{H-1} :
    - Nếu $2^{H-2} < S \le 2^{H-1}$: cấp 2^{H-1}
    - Tiếp tục chia đôi tới khi tìm được vùng thỏa mãn $2^{k-1} < S \le 2^k$

3. Phương pháp kề cận

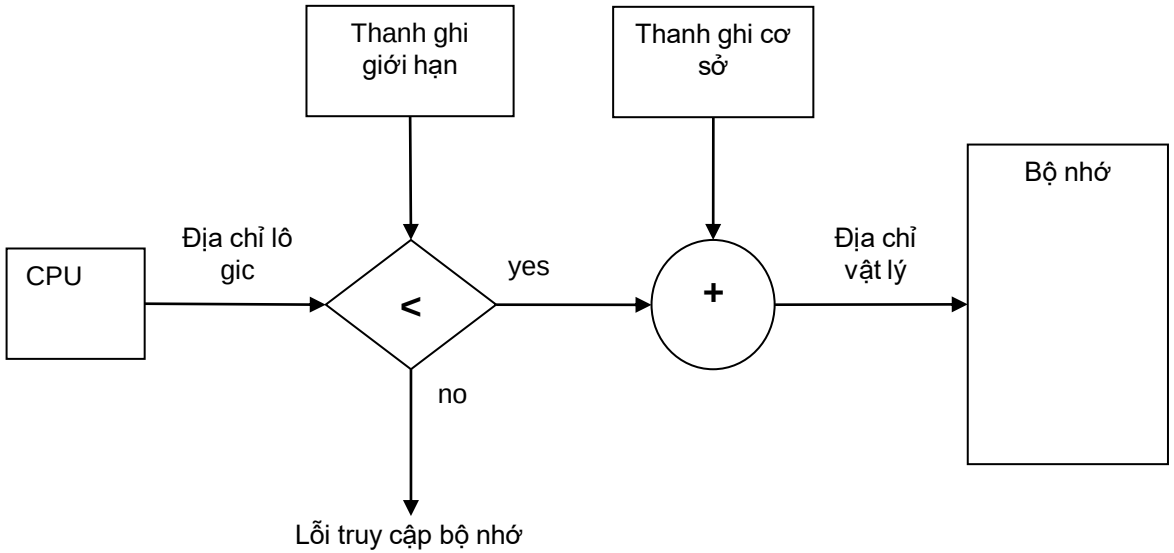
- Sau một thời gian xuất hiện các khối trống có kích thước 2^k
- Tạo danh sách móc nối các vùng có cùng kích thước
- Nếu có 2 khối trống cùng kích thước và kề nhau thì ghép lại thành 1
- Khi cần cấp, sẽ tìm trong danh sách khối phù hợp nhất; nếu không tìm khối lớn hơn và cắt đôi

3. Phương pháp kề cận

Memory

	0	128 K	256 K	384 K	512K	640 K	768 K	896 K	1M
Initially									**1 Hole**
request 70	**A**	**128**			**256**				**3**
request 35	**A**	**B**	**64**		**256**				**3**
request 80	**A**	**B**	**64**	**C**	**128**				**3**
return A	**128**	**B**	**64**	**C**	**128**				**4**
request 60	**128**	**B**	**D**	**C**	**128**				**4**
return B	**128**	**64**	**D**	**C**	**128**				**4**
return D	**256**			**C**	**128**				**3**
return C	**1024**								**1**

- Vị trí các chương thường không biết trước và có thể thay đổi => cần có cơ chế biến đổi địa chỉ logic thành vật lý
- Cấm truy cập trái phép: tiến trình này truy cập tới phần MEM của tiến trình khác
- Ánh xạ địa chỉ do phần cứng đảm nhiệm



- Khi tiến trình được tải vào MEM, CPU dành 2 thanh ghi:
  - Thanh ghi cơ sở: chứa địa chỉ bắt đầu của tiến trình
  - Thanh ghi giới hạn: chứa độ dài chương
- Địa chỉ logic được so sánh với nội dung của thanh ghi giới hạn
  - Nếu lớn hơn: lỗi truy cập
  - Nhỏ hơn: được đưa tới bộ cộng với thanh ghi cơ sở để thành địa chỉ vật lý
- Nếu chương bị di chuyển thì nội dung của thanh ghi cơ sở bị thay đổi chứa địa chỉ vị trí mới

5. Trao đổi giữa bộ nhớ và đĩa (swapping)

- Các tiến trình đang thực hiện có thể bị tạm thời tải ra đĩa nhường chỗ để tải các tiến trình khác vào
- Sau đó lại được tải vào (nếu chưa kết thúc) để thực hiện tiếp
- Xảy ra khi:
  - Một tiến trình đã hết khoảng thời gian sử dụng CPU của mình
  - Nhường chỗ cho một tiến trình khác có thứ tự ưu tiên cao hơn

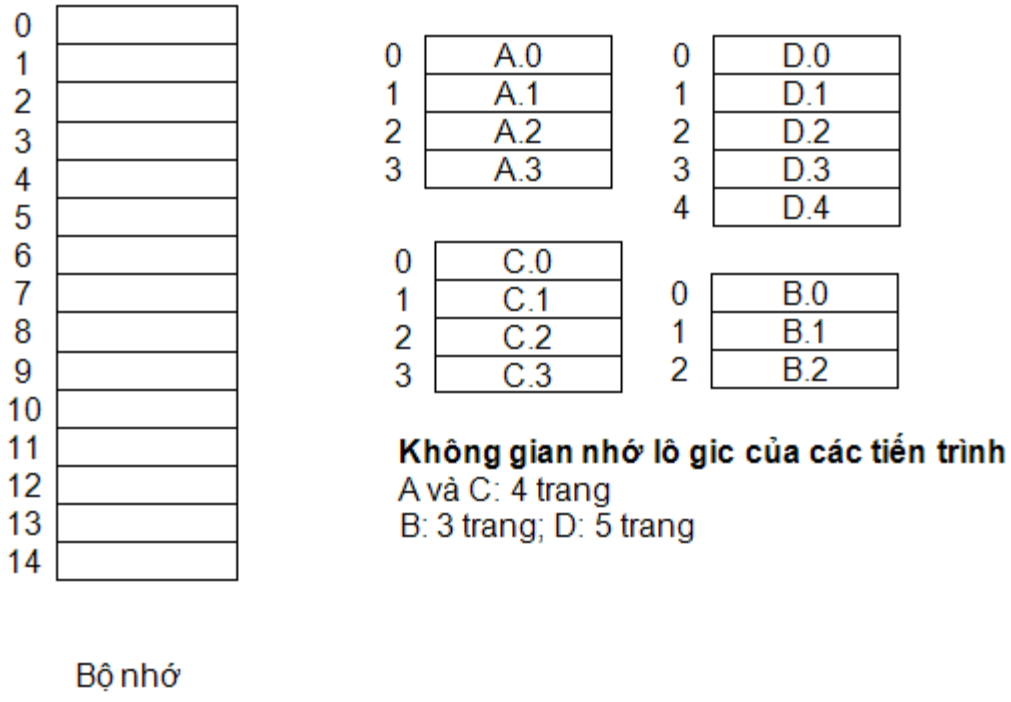
5. Trao đổi giữa bộ nhớ và đĩa (swapping)

- Thời gian tải phụ thuộc vào tốc độ truy cập đĩa, tốc độ truy cập bộ nhớ và kích thước tiến trình
- Khi được tải vào lại, tiến trình có thể được chứa vào chương ở vị trí cũ hoặc được cấp cho một chương địa chỉ hoàn toàn mới
- Các tiến trình bị trao đổi phải ở trạng thái nghỉ, đặc biệt không thực hiện các thao tác vào ra

V. PHÂN TRANG BỘ NHỚ

1. Khái niệm phân trang

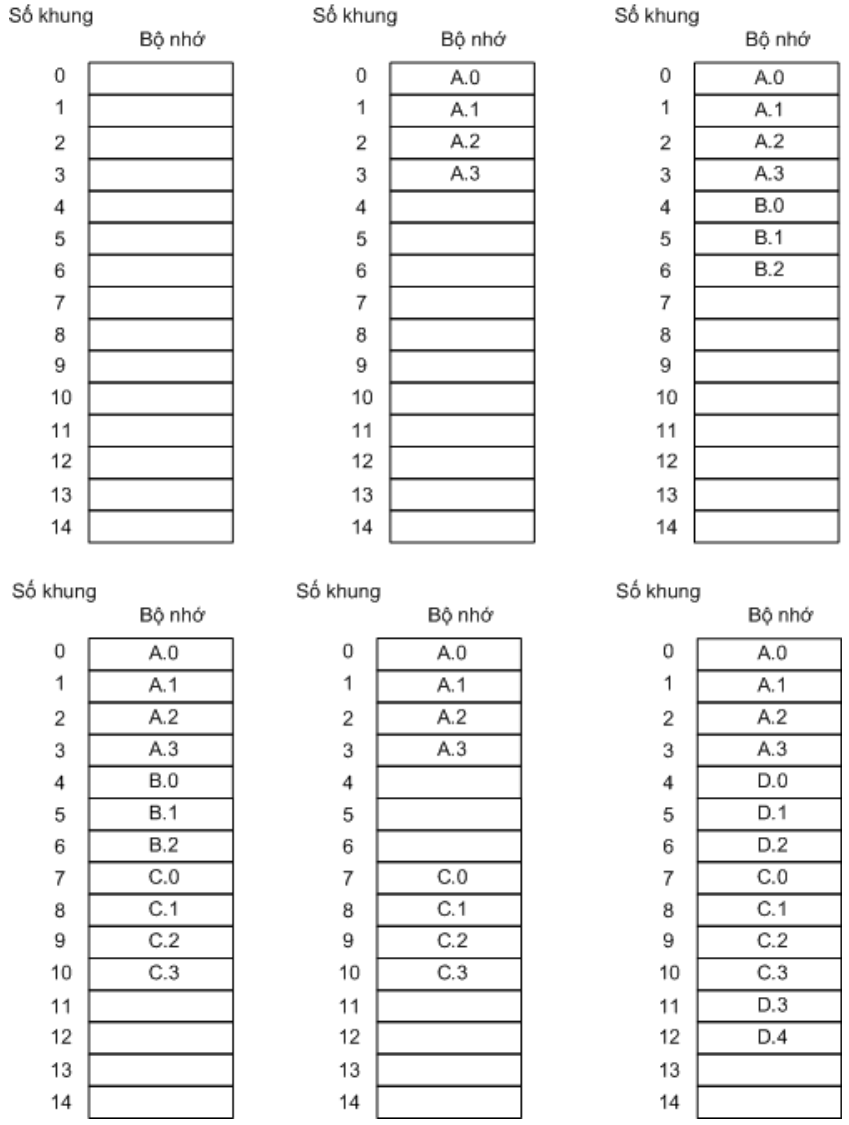
- Bộ nhớ vật lý được chia thành các khối nhỏ, kích thước cố định và bằng nhau gọi là khung trang (page frame)
- Không gian địa chỉ logic của tiến trình được chia thành những khối gọi là trang (page), có kích thước bằng khung



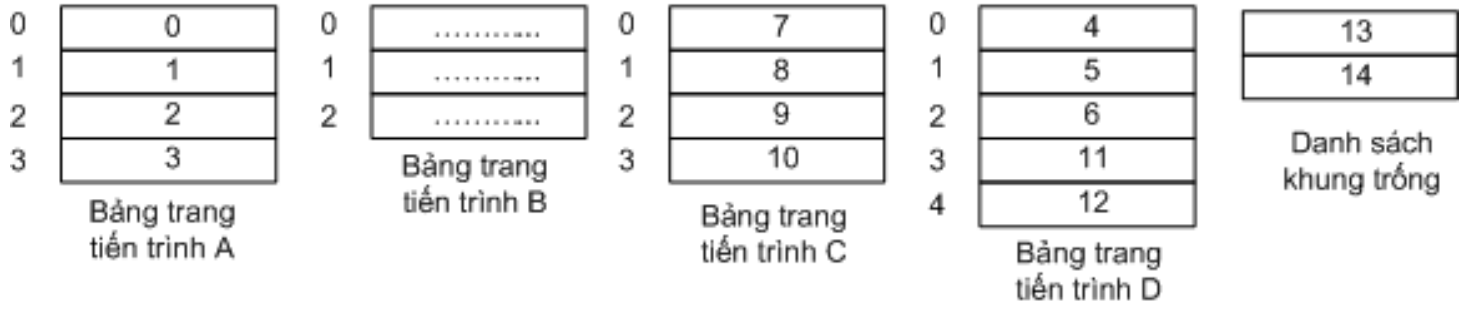
V. PHÂN TRANG BỘ NHỚ

1. Khái niệm phân trang

- Tiến trình được cấp các khung để chứa các trang của mình.
- Các trang có thể chứa trong các khung nằm rải rác trong bộ nhớ



- HDH quản lý việc cấp phát khung cho mỗi tiến trình bằng bảng trang (bảng phân trang): mỗi ô tương ứng với 1 trang và chứa số khung cấp cho trang đó
- Mỗi tiến trình có bảng trang riêng
- Duy trì danh sách các khung trống trong MEM



1. Khái niệm phân trang

- Tương tự như phân chương cố định: khung tương tự chương, kích thước và vị trí không thay đổi
- Tuy nhiên kích thước các phần tương đối nhỏ và các phần cho 1 tiến trình không cần liên tục nhau
- Không có phân mảnh ngoài
- Có phân mảnh trong

2. Ánh xạ địa chỉ

- Để tính toán địa chỉ hiệu quả, kích thước khung được chọn là lũy thừa của 2
- Địa chỉ logic gồm 2 phần:
  - Số thứ tự trang (p)
  - Độ dịch (địa chỉ lệch) của địa chỉ so với đầu trang (o)
- Nếu kích thước trang là 2^n. Biểu diễn địa chỉ logic dưới dạng địa chỉ có độ dài (m + n) bit
  - m bit cao: biểu diễn số thứ tự trang
  - n bit thấp: biểu diễn độ dịch trong trang nhớ

Địa chỉ lô gic	số thứ tự trang (*p*)	độ dịch trong trang (*0*)
Độ dài	*m*	*n*

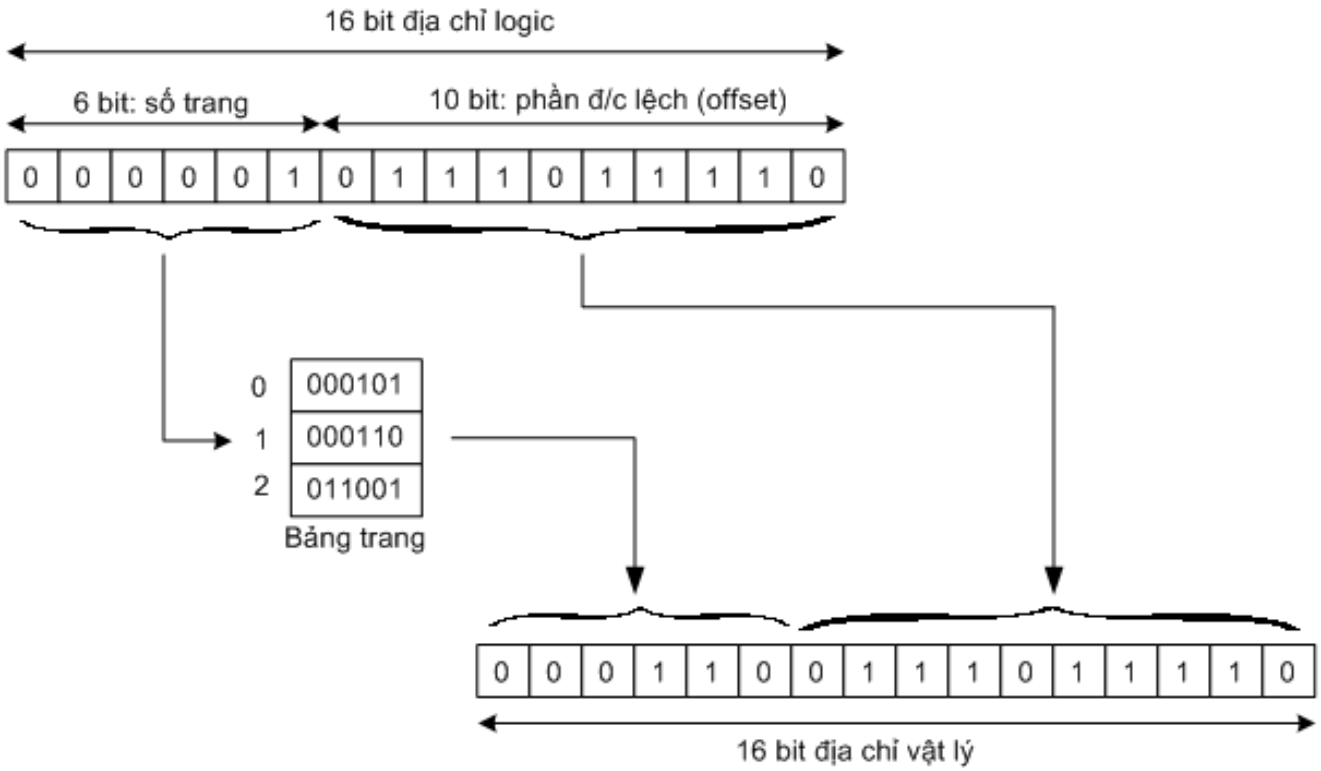
- Quá trình chuyển địa chỉ logic sang địa chỉ vật lý:
  - Lấy m bit cao của địa chỉ => được số thứ tự trang
  - Dựa vào bảng trang, tìm được số thứ tự khung vật lý (k)
  - Địa chỉ vật lý bắt đầu của khung là $k*2^n$
  - Địa chỉ vật lý của byte được tham chiếu là địa chỉ bắt đầu của khung cộng với địa chỉ lệch (độ dịch)

  => Chỉ cần thêm số khung vào trước dãy bit biểu diễn độ lệch

2. Ánh xạ địa chỉ

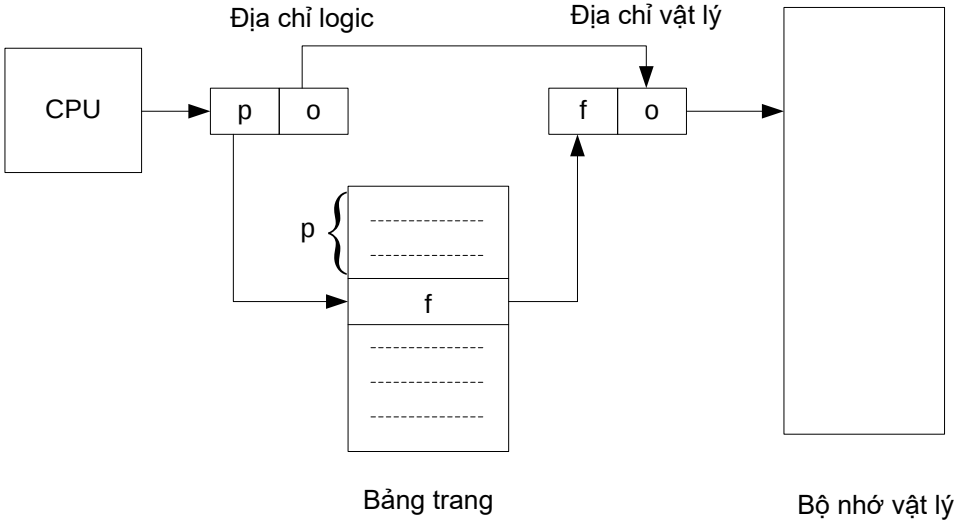
- Kích thước khung là 1KB
- => Sử dụng 10 bit để biểu diễn địa chỉ lệch (n=10)

- Địa chỉ logic 1502
- ↔ byte 478 trong trang 1



V. PHÂN TRANG BỘ NHỚ
2. Ánh xạ địa chỉ

- Quá trình biến đổi từ địa chỉ logic sang địa chỉ vật lý được thực hiện bằng phần cứng
- Kích thước trang là lũy thừa của 2, nằm trong khoảng từ 512B đến 16MB
- Việc tách phần p và o trong địa chỉ logic được thực hiện dễ dàng bằng phép dịch bit



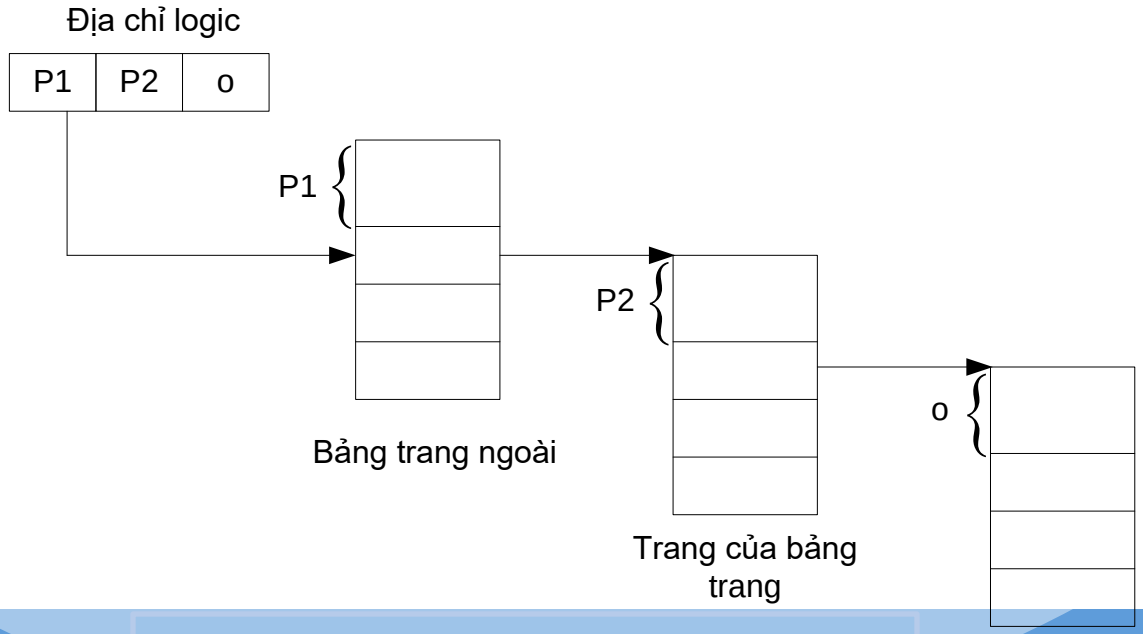
- Phân mảnh trong khi phân trang có giá trị trung bình bằng nửa trang
- => giảm kích thước trang cho phép tiết kiệm MEM
- Kích thước trang nhỏ => số lượng trang tăng => bảng trang to, khó quản lý
- Kích thước trang nhỏ: không tiện cho việc trao đổi với đĩa
- Windows 32bit: kích thước trang 4KB
- Cơ chế ánh xạ giữa hai loại địa chỉ hoàn toàn trong suốt đối với chương trình

3. Tổ chức bảng trang

- Mỗi thao tác truy cập bộ nhớ đều đòi hỏi truy cập bảng phân trang
- => tổ chức bảng phân trang sao cho tốc độ truy cập là cao nhất
- Sử dụng tập hợp các thanh ghi làm bảng phân trang:
  - Tốc độ truy cập rất cao
  - Số lượng thanh ghi hạn chế => không áp dụng được
- Giữ các bảng trang trong MEM:
  - Vị trí mỗi bảng được trỏ bởi thanh ghi cơ sở bảng trang PTBR (Page Table Base Register)
  - Nhiều thời gian để truy cập bảng
  - => sử dụng bộ nhớ cache tốc độ cao

- Không gian địa chỉ logic lớn (2^{32} -> 2^{64}) => kích thước bảng trang tăng
- => cần chia bảng trang thành những phần nhỏ hơn
- Tổ chức bảng trang nhiều mức: Khoản mục của bảng mức trên chỉ tới bảng trang khác

- Ví dụ bảng 2 mức: địa chỉ 32 bit chia thành 3 phần
  - P1: 10 bit cho phép định vị khoản mục trong bảng mức trên => tìm được bảng mức dưới tương ứng
  - P2: định vị khoản mục trong bảng mức dưới (chứa địa chỉ khung tương ứng)
  - O: 12 bit, chứa độ dịch trong trang



VI. PHÂN ĐOẠN BỘ NHỚ

1. Khái niệm

- Chương trình thường được chia thành nhiều phần: dữ liệu, lệnh, ngăn xếp
- Chia chương trình thành các đoạn theo cấu trúc logic
- Mỗi đoạn được phân vào 1 vùng nhớ, có kích thước không bằng nhau
- Mỗi đoạn tương ứng với không gian địa chỉ riêng, được phân biệt bởi tên (STT) và độ dài của mình
- Các vùng nhớ thuộc các đoạn khác nhau có thể nằm ở vị trí khác nhau

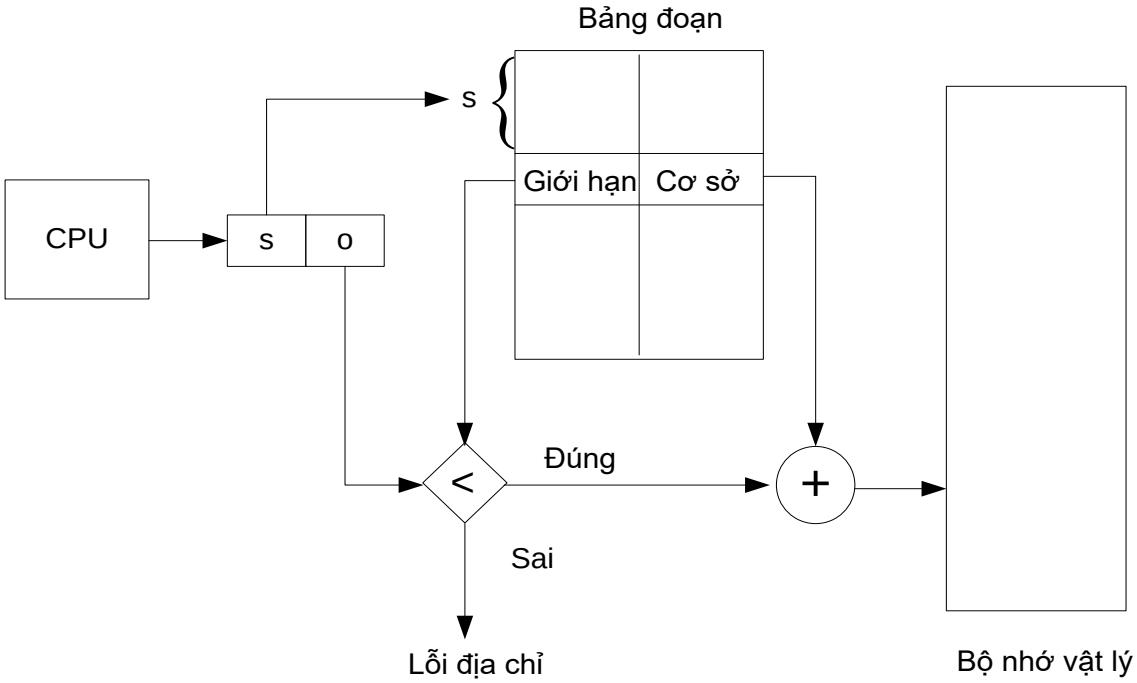
VI. PHÂN ĐOẠN BỘ NHỚ

1. Khái niệm

- Giống phân chương động: bộ nhớ được cấp phát theo từng vùng kích thước thay đổi
- Khác phân chương động: chương trình có thể chiếm nhiều hơn 1 đoạn và không cần liên tiếp nhau trong MEM
- Tránh hiện tượng phân mảnh trong
- Có phân mảnh ngoài
- Dễ sắp xếp bộ nhớ
- Dễ chia sẻ các đoạn giữa các tiến trình khác nhau
- Kích thước mỗi đoạn có thể thay đổi mà không ảnh hưởng tới các đoạn khác

- Sử dụng bảng đoạn cho mỗi tiến trình. Mỗi ô tương ứng với 1 đoạn, chứa:
  - Địa chỉ cơ sở: vị trí bắt đầu của đoạn trong bộ nhớ
  - Địa chỉ giới hạn: độ dài đoạn, sử dụng để chống truy cập trái phép ra ngoài đoạn
- Địa chỉ logic gồm 2 thành phần, (s, o):
  - S: số thứ tự/ tên đoạn
  - O: độ dịch trong đoạn

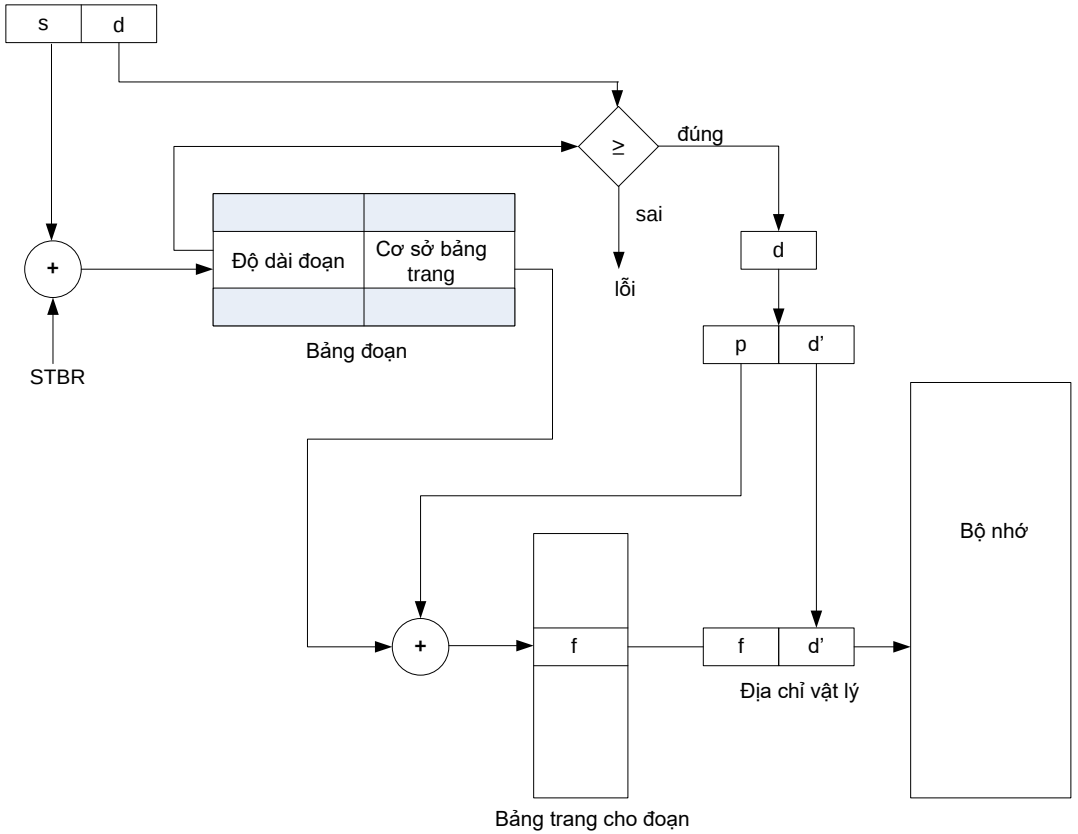
VI. PHÂN ĐOẠN BỘ NHỚ
2. Ánh xạ địa chỉ



- Từ chỉ số đoạn s, vào bảng đoạn, tìm địa chỉ vật lý bắt đầu của đoạn
- So sánh độ dịch o với chiều dài đoạn, nếu lớn hơn => địa chỉ sai
- Địa chỉ vật lý mong muốn là tổng của địa chỉ vật lý bắt đầu đoạn và địa chỉ lệch

3. Kết hợp phân trang và Phân đoạn

- Phân đoạn chương trình, mỗi đoạn sẽ tiến hành phân trang
- Địa chỉ gồm: số thứ tự đoạn, số thự tự trang, độ dịch trong trang
- Tiến trình có 1 bảng phân đoạn, mỗi đoạn có 1 bảng phân trang



V. BỘ NHỚ ẢO

1. Khái niệm

- Tiến trình có thể chia thành các phần nhỏ nằm rải rác trong bộ nhớ
- Tất cả các phép biến đổi là trong suốt với người dùng và người lập trình chỉ làm việc với không gian nhớ logic
- Không phải tiến trình nào khi chạy cũng sử dụng tất cả các lệnh và dữ liệu của mình với tần số như nhau
- => không nhất thiết toàn bộ các trang/ đoạn của một tiến trình phải có mặt đồng thời trong bộ nhớ khi tiến trình chạy
- => Các trang hoặc đoạn có thể được trao đổi từ đĩa vào bộ nhớ khi có nhu cầu truy cập tới

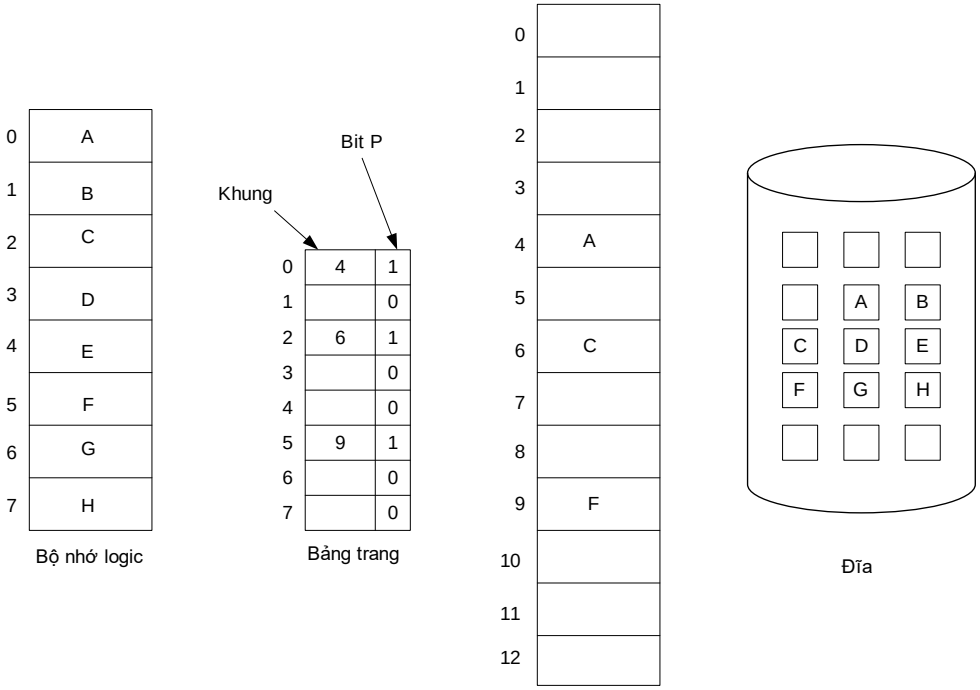
V. BỘ NHỚ ẢO
1. Khái niệm

- Việc thực hiện các tiến trình chỉ nằm một phần trong bộ nhớ có một số ưu điểm:
  - Có thể viết chương trình có kích thước lớn hơn kích thước thực của MEM
  - Cùng 1 lúc nhiều tiến trình cùng được tải vào MEM hơn
- => Bộ nhớ ảo là bộ nhớ lôgic theo cách nhìn của người lập trình và tiến trình và không bị hạn chế bởi bộ nhớ thực.
  - **Bộ nhớ ảo có thể lớn hơn bộ nhớ thực rất nhiều và bao gồm cả không gian trên đĩa**
  - Bộ nhớ ảo thường được xây dựng dựa trên phương pháp phân trang trong đó các trang là đơn vị để nạp từ đĩa vào khi cần

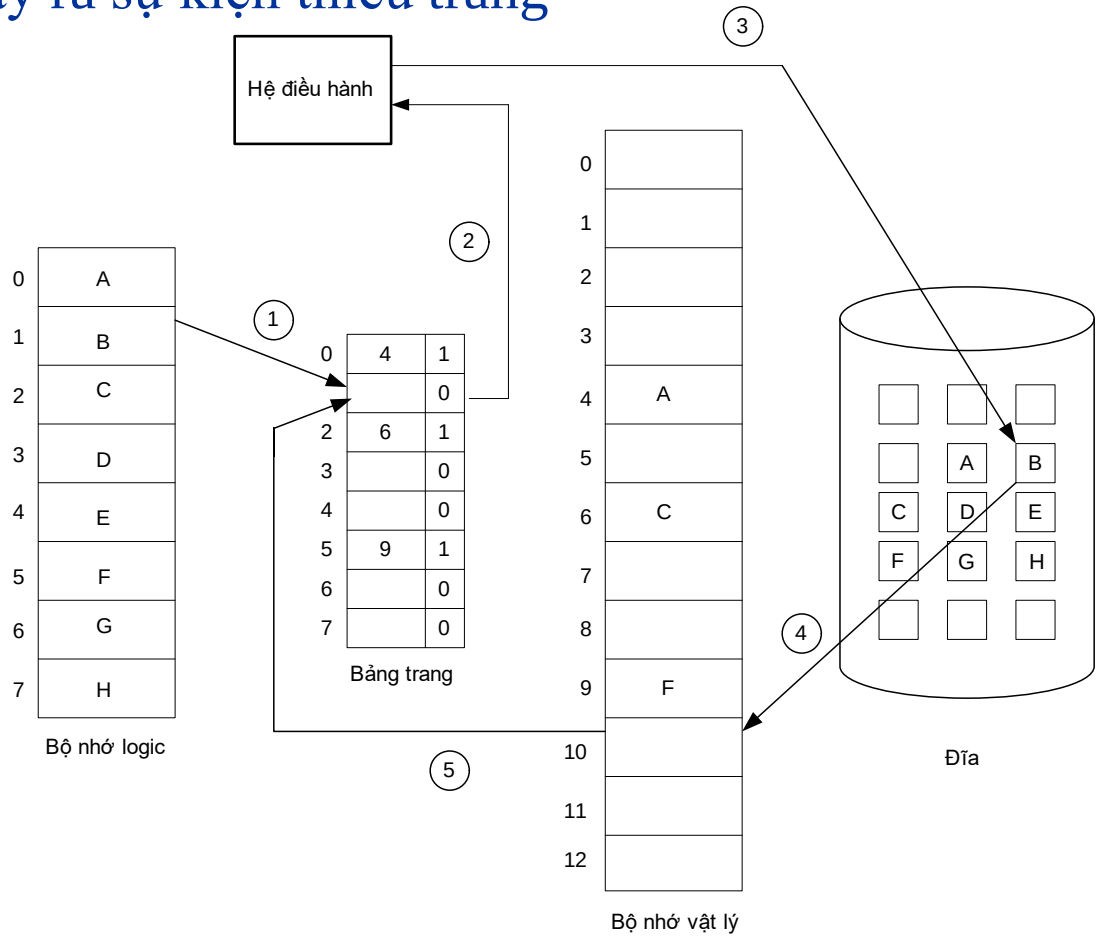
V. BỘ NHỚ ẢO

2. Nạp trang theo nhu cầu

- Tiến trình được phân trang và chứa trên đĩa
- Khi cần thực hiện, nạp tiến trình vào MEM: chỉ nạp những trang cần dùng
- Tiến trình gồm các trang trên đĩa và trong MEM: thêm bit P trong khoản mục bảng trang để phân biệt (P=1: đã nạp vào MEM)



- Quá trình kiểm tra và nạp trang:
  - Tiến trình truy cập tới 1 trang, kiểm tra bit P. Nếu P=1, truy cập diễn ra bình thường. Nếu P=0, xảy ra sự kiện thiếu trang
- Ngắt xử lý thiếu trang:
  - HDH tìm 1 khung trống trong MEM
  - Đọc trang bị thiếu vào khung trang vừa tìm được
  - Sửa lại khoản mục tương ứng trong bảng trang: đổi bit P=1 và số khung đã cấp cho trang
  - Khôi phục lại trạng thái tiến trình và thực hiện tiếp lệnh bị ngắt



- Nạp trang hoàn toàn theo nhu cầu:
  - Bắt đầu một tiến trình mà không nạp bất kỳ trang nào vào bộ nhớ
  - Khi con trỏ lệnh được HDH chuyển tới lệnh đầu tiên của tiến trình để thực hiện, sự kiện thiếu trang sẽ sinh ra và trang tương ứng được nạp vào
  - Tiến trình sau đó thực hiện bình thường cho tới lần thiếu trang tiếp theo
- Nạp trang trước: khác với nạp trang theo nhu cầu
  - Các trang chưa cần đến cũng được nạp vào bộ nhớ
  - Không hiệu quả

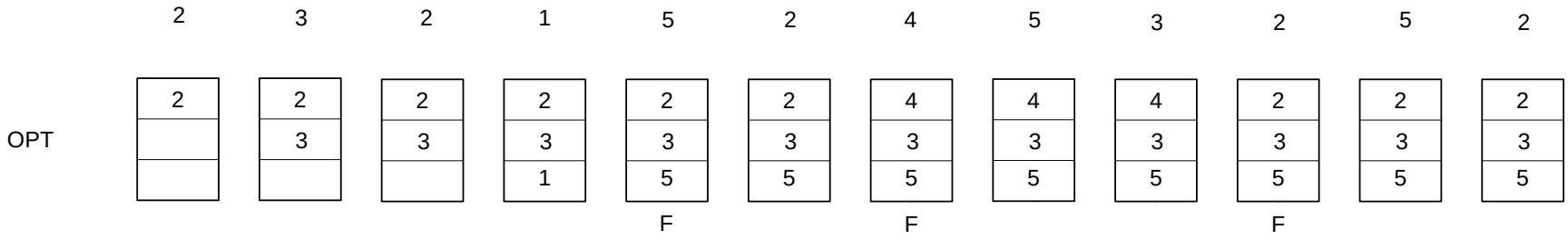
- Bộ nhớ ảo > bộ nhớ thực và chế độ đa chương trình -> có lúc không còn khung nào trống để nạp trang mới

- Giải pháp:
  - Kết thúc tiến trình
  - Trao đổi tiến trình ra đĩa và chờ thời điểm thuận lợi hơn
  - Đổi trang

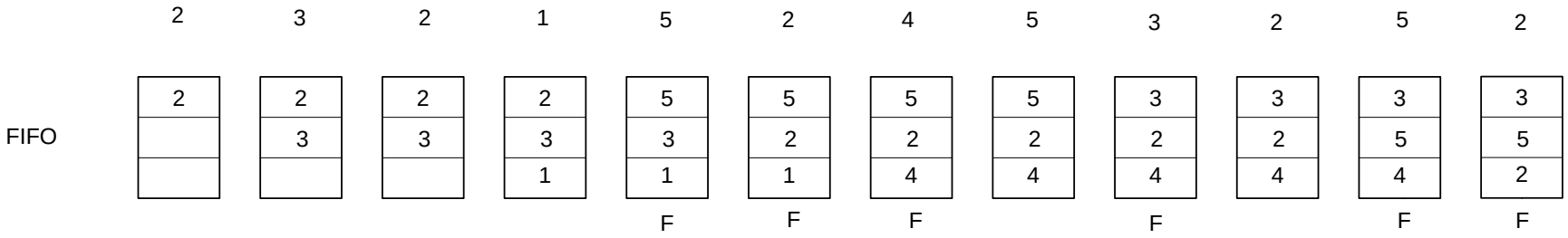
- Nếu không còn khung nào trống, HDH chọn 1 khung đã cấp phát nhưng hiện không dùng tới và giải phóng nó
- Quá trình đổi trang:
  - B1: Xác định trang trên đia cần nạp vào MEM
  - B2: Nếu có khung trống trên MEM thì chuyển sang B4
  - B3:
    - Lựa chọn 1 khung trên MEM để giải phóng, theo 1 thuật toán nào đó
    - Ghi nội dung khung bị đổi ra đĩa (nếu cần), cập nhật bảng trang và bảng khung
  - B4: Đọc trang cần nạp vào khung vừa giải phóng; cập nhật bảng trang và bảng khung
  - B5: Thực hiện tiếp tiến trình từ điểm bị dừng trước khi đổi trang

- Đổi trang có ghi và đổi trang không ghi:
  - Việc ghi trang bị đổi ra đĩa làm tăng đáng kể thời gian nạp trang
  - => nhận biết các trang không thay đổi từ lúc nạp và không ghi ngược ra đĩa
  - Sử dụng thêm bit sửa đổi M trong khoản mục trang để đánh dấu trang đã bị sửa đổi (1) hay chưa (0)
- Các khung bị khóa
  - Một số khung sẽ không bị giải phóng trong quá trình tìm kiếm khung để đổi trang => các khung bị khóa
  - VD: Khung chứa tiến trình nhân của HDH, chứa các cấu trúc thông tin điều khiển quan trọng
  - Nhận biết bởi 1 bit riêng chứa trong khoản mục

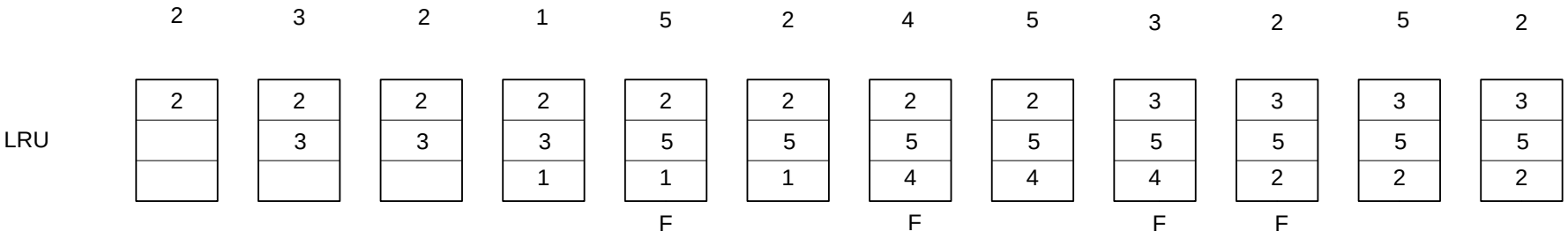
- Đổi trang tối ưu (OPT):
  - Chọn trang sẽ không được dùng tới trong khoảng thời gian lâu nhất để đổi
  - Cho phép giảm tối thiểu sự kiện thiếu trang và do đó là tối ưu theo tiêu chuẩn này
  - HDH không đoán trước được nhu cầu sử dụng các trang trong tương lai
  - => không áp dụng trong thực tế mà chỉ để so sánh với các chiến lược khác



- Vào trước ra trước (FIFO):
  - Trang nào được nạp vào trước thì bị đổi ra trước
  - Đơn giản nhất
  - Trang bị trao đổi là trang nằm lâu nhất trong bộ nhớ



- Đổi trang ít sử dụng nhất trong thời gian cuối (LRU):
  - Trang bị đổi là trang mà thời gian từ lần truy cập cuối cùng đến thời điểm hiện tại là lâu nhất
  - Theo nguyên tắc cục bộ về thời gian, đó chính là trang ít có khả năng sử dụng tới nhất trong tương lai
  - Thực tế LRU cho kết quả tốt gần như phương pháp đổi trang tối ưu



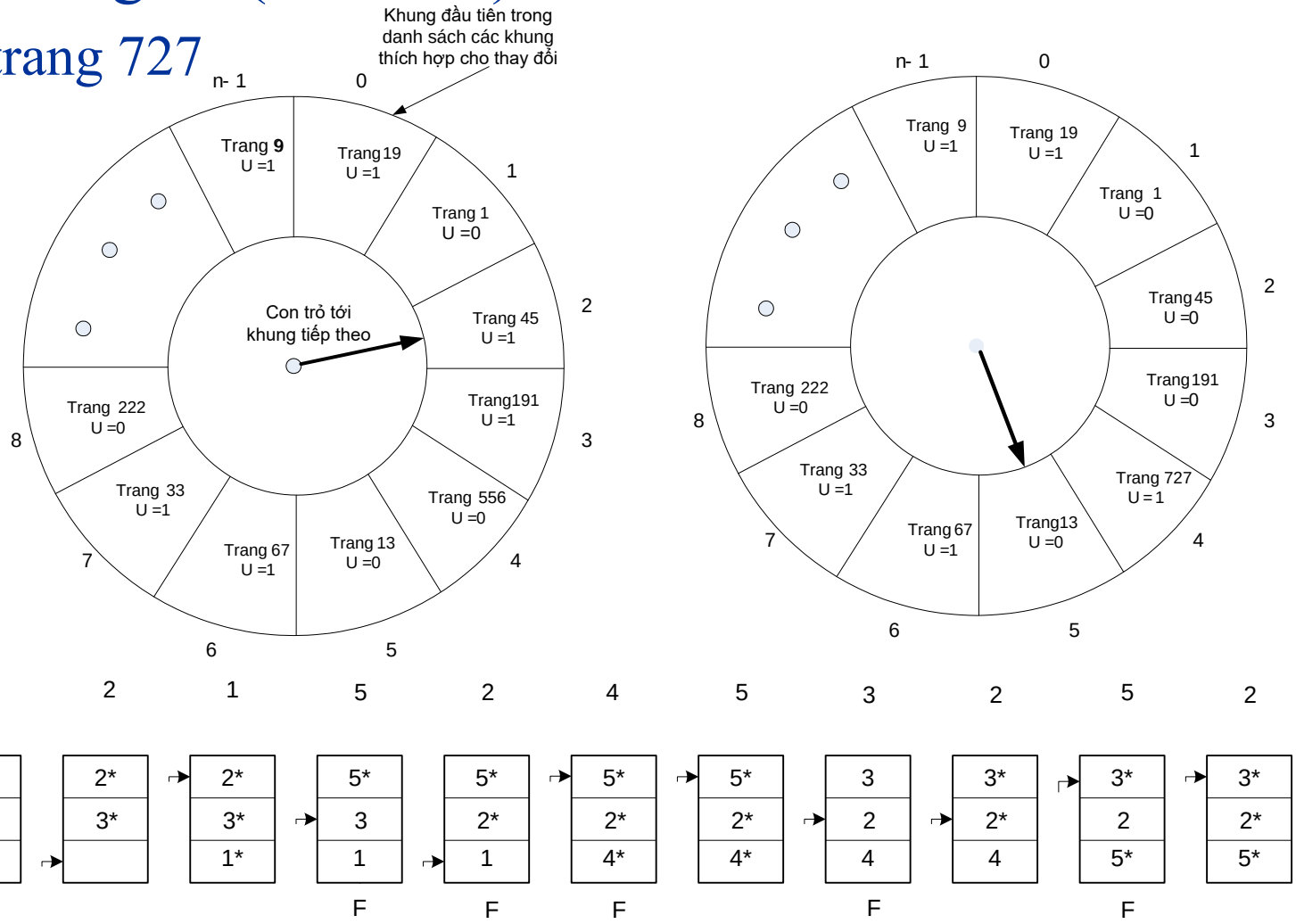
- Đổi trang ít sử dụng nhất trong thời gian cuối (LRU):
  - Xác định được trang có lần truy cập cuối diễn ra cách thời điểm hiện tại lâu nhất?
    - Sử dụng biến đếm:
      - Mỗi khoản mục của bảng phân trang sẽ có thêm một trường chứa thời gian truy cập trang lần cuối - Là thời gian logic
      - CPU cũng được thêm một bộ đếm thời gian lôgic này
      - Chỉ số của bộ đếm tăng mỗi khi xảy ra truy cập bộ nhớ
      - Mỗi khi một trang nhớ được truy cập, chỉ số của bộ đếm sẽ được ghi vào trường thời gian truy cập trong khoản mục của trang đó
      - => trường thời gian luôn chứa thời gian truy cập trang lần cuối
      - => trang bị đổi là trang có giá trị thời gian nhỏ nhất

- Đổi trang ít sử dụng nhất trong thời gian cuối (LRU):
  - Sử dụng ngăn xếp:
    - Ngăn xếp đặc biệt được sử dụng để chứa các số trang
    - Mỗi khi một trang nhớ được truy cập, số trang sẽ được chuyển lên đỉnh ngăn xếp
    - Đỉnh ngăn xếp sẽ chứa trang được truy cập gần đây nhất
    - Đáy ngăn xếp chính là trang LRU, tức là trang cần trao đổi
    - Tránh tìm kiếm trong bảng phân trang
    - Thích hợp thực hiện bằng phần mềm

- Thuật toán đồng hồ (CLOCK):
  - Cải tiến FIFO nhằm tránh thay những trang mặc dù đã được nạp vào lâu nhưng vẫn có khả năng sử dụng
  - Mỗi trang được gắn thêm 1 bit sử dụng U
    - Khi trang được truy cập đọc/ ghi: U = 1
    - => ngay khi trang được đọc vào bộ nhớ: U =1
  - Các khung có thể bị đổi (các trang tương ứng) được liên kết vào 1 danh sách vòng
  - Khi một trang nào đó bị đổi, con trỏ được dịch chuyển để trỏ vào trang tiếp theo trong danh sách

- Thuật toán đồng hồ (CLOCK):
  - Khi có nhu cầu đổi trang, HDH kiểm tra trang đang bị trỏ tới
    - Nếu U=0: trang sẽ bị đổi ngay
    - Nếu U=1: đặt U=0 và trỏ sang trang tiếp theo, lặp lại quá trình
  - Nếu U của tất cả các trang trong danh sách =1 thì con trỏ sẽ quay đúng 1 vòng, đặt U của tất cả các trang =0 và trang hiện thời đang bị trỏ sẽ bị đổi

- Thuật toán đồng hồ (CLOCK):
  - Cần nạp trang 727



- Thuật toán đồng hồ (CLOCK):
  - Căn cứ vào 2 thông tin để đưa ra quyết định đổi trang:
    - Thời gian trang được tải vào, thể hiện qua vị trí trang trong danh sách giống như FIFO
    - Gần đây trang có được sử dụng hay không, thể hiện qua bit U
  - Việc kiểm tra thêm bit U tương tự việc cho trang thêm khả năng được giữ trong bộ nhớ
  - => thuật toán cơ hội thứ 2

- Thuật toán đồng hồ cải tiến:
  - Sử dụng thêm thông tin về việc nội dung trang có bị thay đổi hay không bằng bit M
  - Kết hợp bit U và M, có 4 khả năng:
    - U=0, M=0: trang gần đây không được truy cập và nội dung cũng không bị thay đổi, rất thích hợp để bị đổi ra ngoài
    - U=0, M=1: trang có nội dung thay đổi nhưng gần đây không được truy cập, cũng là ứng viên để đổi ra ngoài
    - U=1, M=0: trang mới được truy cập gần đây và do vậy theo nguyên lý cục bộ về thời gian có thể sắp được truy cập tiếp
    - U=1, M=1: trang có nội dung bị thay đổi và mới được truy cập gần đây, chưa thật thích hợp để đổi

- Thuật toán đồng hồ cải tiến:
  - Các bước thực hiện đổi trang:
    - Bước 1:
      - Bắt đầu từ vị trí hiện tại của con trỏ, kiểm tra các trang
      - Trang đầu tiên có U=0 và M=0 sẽ bị đổi
      - Chỉ kiểm tra mà không thay đổi nội dung bit U, bit M
    - Bước 2:
      - Nếu quay hết 1 vòng mà không tìm được trang có U và M bằng 0 thì quét lại danh sách lần 2
      - Trang đầu tiên có U=0, M=1 sẽ bị đổi
      - Đặt bit U của những trang đã quét đến nhưng được bỏ qua là 0
    - Nếu chưa tìm được thì lặp lại bước 1 và cả bước 2 nếu cần

- HDH dành ra một số khung trống được kết nối thành danh sách liên kết gọi là các trang đệm
- Trang bị đổi như bình thường nhưng nội dung trang này không bị xóa ngay khỏi bộ nhớ
- Khung chứa trang được đánh dấu là khung trống và thêm vào cuối danh sách trang đệm
- Trang mới sẽ được nạp vào khung đứng đầu trong danh sách trang đệm
- Tới thời điểm thích hợp, hệ thống sẽ ghi nội dung các trang trong danh sách đệm ra đĩa

- Kỹ thuật đệm trang cho phép cải tiến tốc độ:
  - Nếu trang bị đổi có nội dung cần ghi ra đĩa, HDH vẫn có thể nạp trang mới vào ngay
    - Việc ghi ra đĩa sẽ được lùi lại tới một thời điểm sau
    - Thao tác ghi ra đĩa có thể thực hiện đồng thời với nhiều trang nằm trong danh sách được đánh dấu trống.
  - Trang bị đổi vẫn được giữ trong bộ nhớ một thời gian:
    - Nếu có yêu cầu truy cập trong thời gian này, trang sẽ được lấy ra từ danh sách đệm và sử dụng ngay mà không cần nạp lại từ đĩa
    - => Vùng đệm đóng vai trò giống như bộ nhớ cache

VI. CẤP PHÁT KHUNG TRANG

1. Giới hạn khung

- HDH cấp phát bao nhiêu khung cho mỗi tiến trình?
- Khi số lượng khung tối đa cấp cho tiến trình giảm tới mức nào đó, lỗi thiếu trang diễn ra thường xuyên

  => Đặt giới hạn tối thiểu các khung cấp phát cho tiến trình
- Khi số lượng khung cấp cho tiến trình tăng tới mức nào đó thì việc tăng thêm khung cho tiến trình không làm giảm đáng kể tần suất thiếu trang nữa
- => Cấp phát số lượng khung cố định và số lượng khung thay đổi

1.1. Cấp phát số lượng khung cố định

- Cấp cho tiến trình một số lượng cố định khung để chứa các trang nhớ
- Số lượng được xác định vào thời điểm tạo mới tiến trình và không thay đổi trong quá trình tiến trình tồn tại
- Cấp phát bằng nhau:
  - Các tiến trình được cấp số khung tối đa bằng nhau
  - Số lượng được xác định dựa vào kích thước MEM và mức độ đa chương trình mong muốn
- Cấp phát không bằng nhau:
  - Các tiến trình được cấp số khung tối đa khác nhau
    - Cấp số khung tỉ lệ thuận với kích thước tiến trình
    - Có mức ưu tiên

- Số lượng khung tối đa cấp cho mỗi tiến trình có thể thay đổi trong quá trình thực hiện
- Việc thay đổi phụ thuộc vào tình hình thực hiện của tiến trình
- Cho phép sử dụng bộ nhớ hiệu quả hơn phương pháp cố định
- => Cần theo dõi và xử lý thông tin về tình hình sử dụng bộ nhớ của tiến trình

2. Phạm vi cấp phát khung

- Cấp phát toàn thể:
  - Cho phép tiến trình đổi trang mới vào bất cứ khung nào (không bị khóa), kể cả khung đã được cấp phát cho tiến trình khác
- Cấp phát cục bộ:
  - Trang chỉ được đổi vào khung đang được cấp cho chính tiến trình đó
- Phạm vi cấp phát có quan hệ mật thiết với số lượng khung tối đa:
  - Số lượng khung cố định tương ứng với phạm vi cấp phát cục bộ
  - Số lượng khung thay đổi tương ứng với phạm vi cấp phát toàn thể

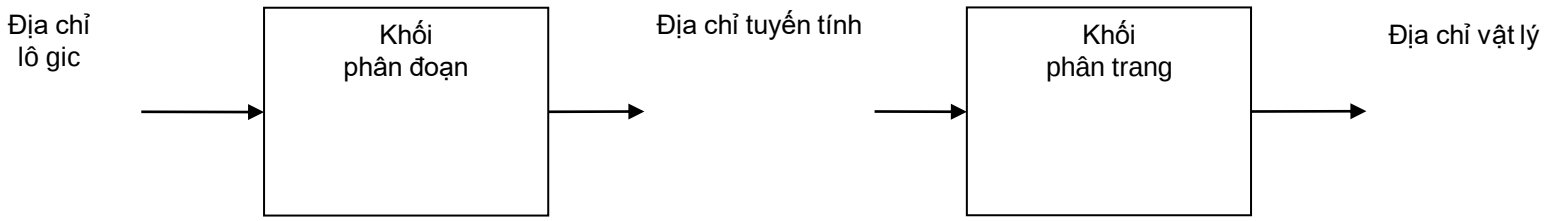
VII. TÌNH TRẠNG TRÌ TRỆ (thrashing)

- Là tình trạng đổi trang liên tục do không đủ bộ nhớ
- Thời gian đổi trang của tiến trình lớn hơn thời gian thực hiện
- Xảy ra khi:
  - Kích thước bộ nhớ hạn chế
  - Tiến trình đòi hỏi truy cập đồng thời nhiều trang nhớ
  - Hệ thống có mức độ đa chương trình cao

VII. TÌNH TRẠNG TRÌ TRỆ (thrashing)
Kiểm soát tần suất thiếu trang

- Khi tiến trình rơi vào tình trạng trì trệ, tần suất thiếu trang tăng đáng kể
- => sử dụng để phát hiện và giải quyết vấn đề trì trệ
- Theo dõi và ghi lại tần suất thiếu trang
- Có thể đặt ra giới hạn trên và giới hạn dưới cho tần suất thiếu trang của tiến trình
  - Tần suất vượt giới hạn trên:
    - Cấp thêm cho tiến trình khung mới
    - Nếu không thể tìm khung để cấp thêm, tiến trình sẽ bị treo hoặc bị kết thúc
  - Tần suất thiếu trang thấp hơn giới hạn dưới: thu hồi một số khung của tiến trình

- Hỗ trợ cơ chế quản lý bộ nhớ: phân đoạn được kết hợp với phân trang
- Không gian nhớ của tiến trình bao gồm nhiều đoạn, mỗi đoạn có thể có kích thước khác nhau và được phân trang trước khi đặt vào bộ nhớ
- Ánh xạ địa chỉ: 2 giai đoạn

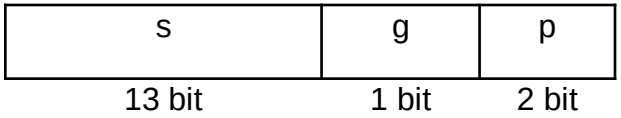


Phân đoạn

- Cho phép tiến trình có tối đa 16KB (hơn 16000) đoạn, mỗi đoạn có kích thước tối đa 4GB
- Không gian nhớ lô gic được chia thành hai phần:
  - Phần 1: dành riêng cho tiến trình, bao gồm tối đa 8KB đoạn
  - Phần 2: dùng chung cho tất cả tiến trình, bao gồm cả HDH, và cũng gồm tối đa 8KB đoạn
- LDT(Local Descriptor Table) & GDT (Global Descriptor Table): chứa thông tin quản lý :
  - Mỗi ô có kích thước 8 byte: chứa địa chỉ cơ sở và giới hạn của đoạn tương ứng

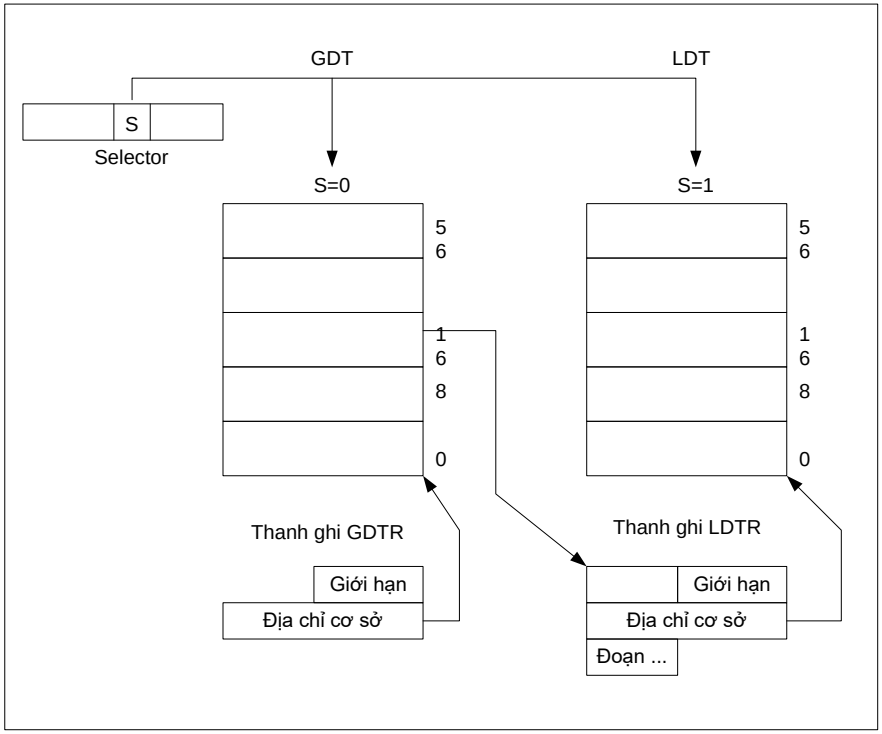
Phân đoạn

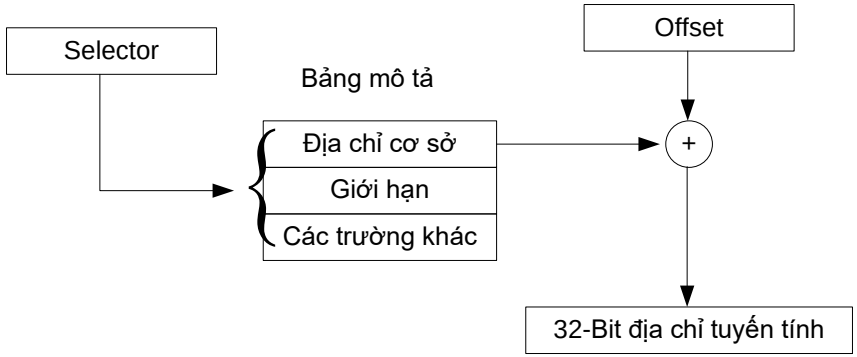
- Có 6 thanh ghi đoạn: cho phép tiến trình truy cập đồng thời 6 đoạn
- Thông tin về đoạn được chứa trong 6 thanh ghi 8 byte
- Địa chỉ logic gồm (selector, offset):
  - Selector: chọn ô tương ứng từ hai bảng mô tả LDT, GDT



    - S: là số thứ tự đoạn
    - G: cho biết đoạn thuộc GDT (g=0) hay LDT(g=1)
    - P: cho biết chế độ bảo vệ (p=0 là chế độ nhân, p=3 là chế độ người dùng)
  - Offset: độ dịch trong đoạn, kích thước 32bit

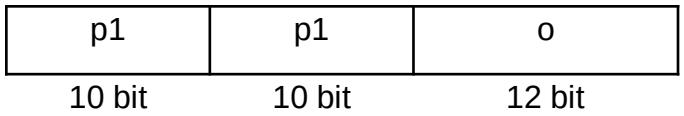
- Biến đổi địa chỉ logic thành địa chỉ tuyến tính:





Phân trang

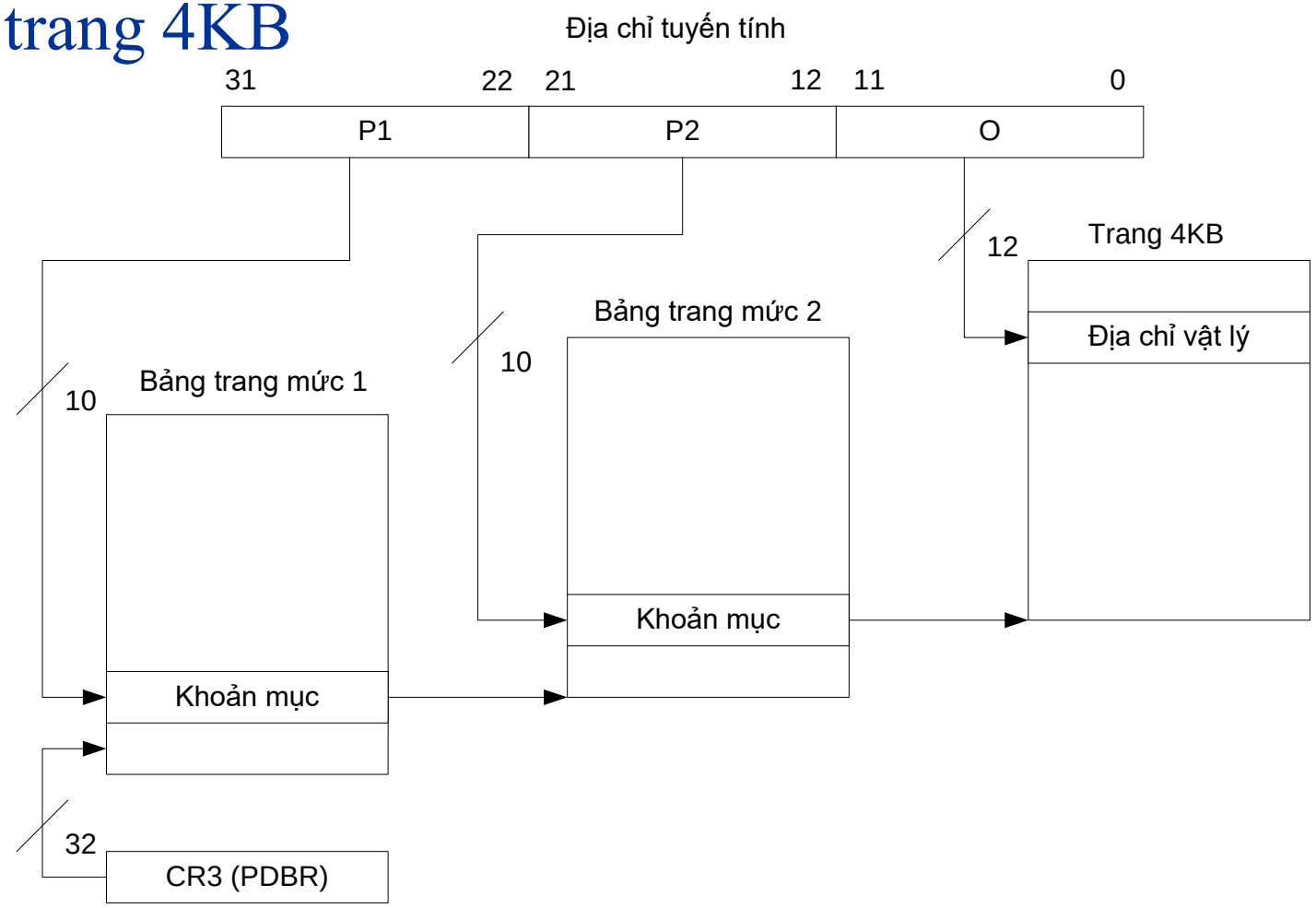
- Hỗ trợ kích thước trang 4KB hoặc 4MB, tùy thuộc vào giá trị cờ kích thước trang
- Trang kích thước 4KB: tổ chức bảng trang thành 2 mức
  - Địa chỉ tuyến tính có kích thước 32 bit



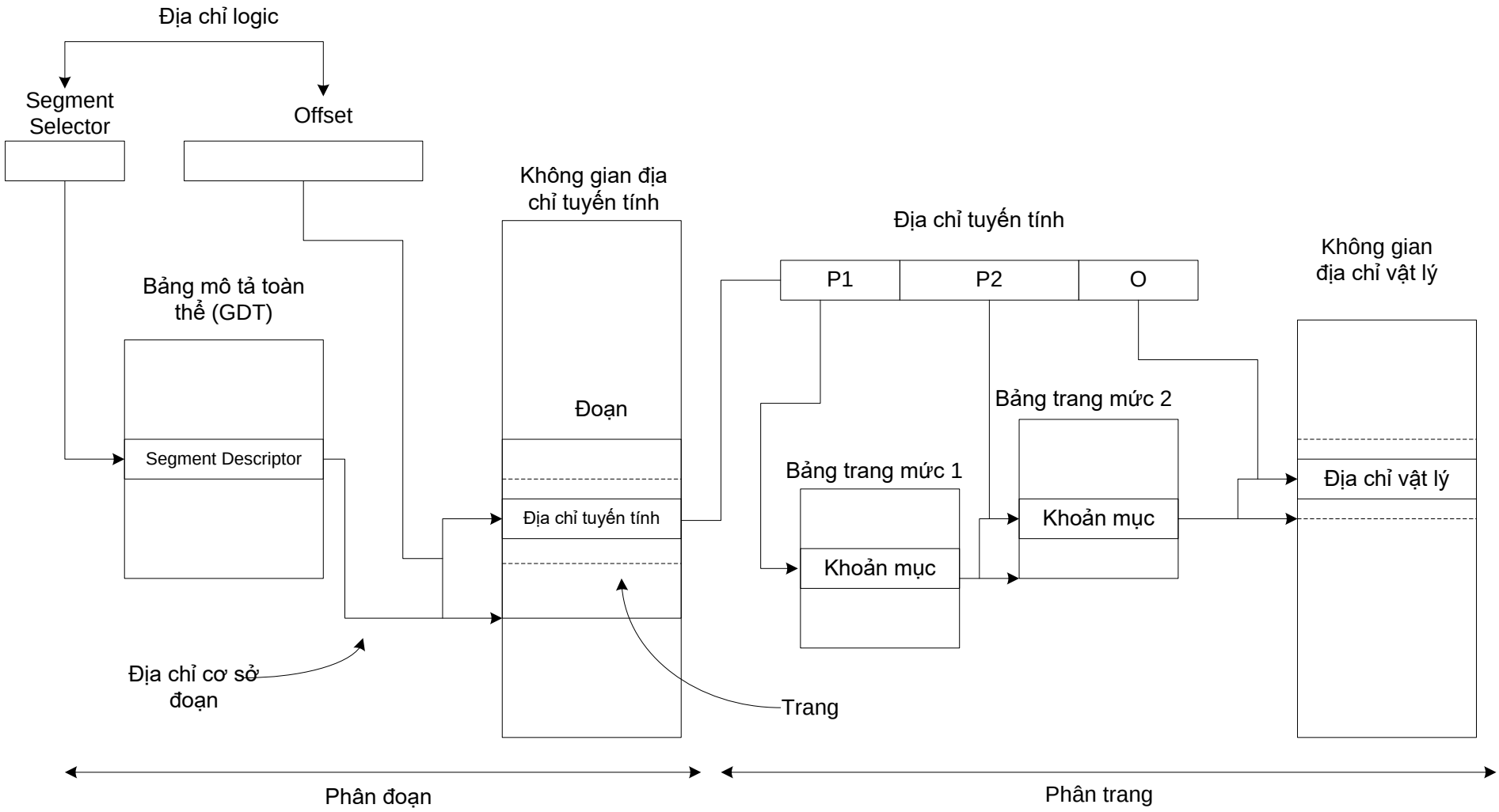
  - P1: cho phép tìm bảng trang mức hai
  - P2: tìm ô tương ứng trong bảng trang mức 2 kết hợp với độ dịch o tạo ra địa chỉ vật lý
- Trang kích thước 4MB: Bảng trang chỉ có một mức
    - P :10bit
    - O: độ dịch, kích thước 22bit cho phép trỏ tới vị trí cụ thể trong trang nhớ 4MB

Phân trang

- Biến đổi địa chỉ tuyến tính thành địa chỉ vật lý với kích thước trang 4KB



Ánh xạ địa chỉ



IX. QUẢN LÝ BỘ NHỚ TRONG WINDOWS XP

- Cho phép tiến trình sử dụng bộ nhớ ảo tới 4GB
  - 2GB được dùng riêng cho tiến trình
  - 2GB sau được dùng chung cho hệ thống
- Bộ nhớ ảo thực hiện bằng kỹ thuật nạp trang theo nhu cầu và đổi trang
  - Kích thước trang nhớ 4KB
  - Tổ chức bảng trang 2 mức
- Nạp trang theo cụm: khi xảy ra thiếu trang, nạp cả cụm gồm 1 số trang nằm sau trang bị thiếu

IX. QUẢN LÝ BỘ NHỚ TRONG WINDOWS XP

- Kiểm soát số lượng trang: gán cho mỗi tiến trình số lượng trang tối đa và tối thiểu
- Số lượng trang tối đa và tối thiểu cấp cho tiến trình được thay đổi tùy vào tình trạng bộ nhớ trống
- HDH lưu danh sách khung trống, và sử dụng một ngưỡng an toàn
  - Số khung trống ít hơn ngưỡng: HDH xem xét các tiến trình đang thực hiện.
  - Tiến trình có số trang lớn hơn số lượng tối thiểu sẽ bị giảm số trang cho tới khi đạt tới số lượng tối thiểu của mình.
- Tùy vào vi xử lý, Windows XP sử dụng thuật toán đổi trang khác nhau

KIỂM TRA

Bài 1: Kích thước khung bộ nhớ là 4096 bytes. Hãy chuyển địa chỉ logic 8207, 4300 sang địa chỉ vật lý biết rằng bảng trang như sau:

Số trang	Số khung
0	3
1	5
2	
3	30
4	22
5	14

Bài 2: Không gian địa chi logic của tiến trình gồm 11 trang, mỗi trang có kích thước 2048B được ánh xạ vào bộ nhớ vật lý có 20 khung

a. Để biểu diễn địa chỉ logic cần tối thiểu bao nhiêu bit

b. Để biểu diễn địa chỉ vật lý cần bao nhiêu bit

Bài 3: Bộ nhớ vật lý có 4 khung. Thứ tự truy cập các trang là 1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6. Vẽ sơ đồ cấp phát bộ nhớ và Có bao nhiêu sự kiện thiếu trang xảy ra nếu sử dụng:

- Thuật toán tối ưu
- FIFO
- LRU
- Đồng hồ

Bài 4: Bộ nhớ có kích thước 1MB. Sử dụng phương pháp kề cận (buddy system) để cấp phát cho các tiến trình lần lượt với kích thước như sau: A: 112KB, B: 200KB, C: 150KB, D: 50KB

HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

BÀI GIẢNG MÔN

HỆ ĐIỀU HÀNH

Bộ môn: Khoa học máy tính- Khoa CNTT1

CHƯƠNG 4: QUẢN LÝ TIẾN TRÌNH

NỘI DUNG

1. Các khái niệm liên quan đến tiến trình
2. Luồng (thread)
3. Điều độ tiến trình
4. Đồng bộ hóa các tiến trình đồng thời
5. Tình trạng bế tắc và đói

1. Tiến trình là gì?

- *Tiến trình là một chương trình đang trong quá trình thực hiện*

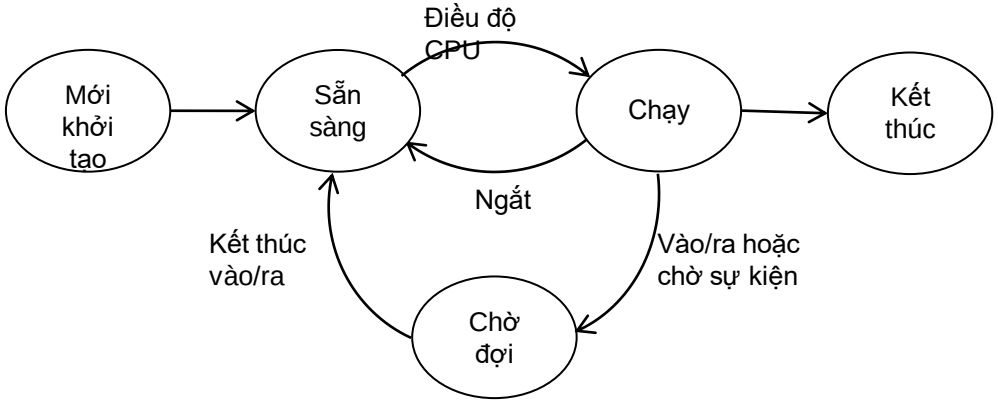
Chương trình	Tiến trình
Thực thể tĩnh	Thực thể động
Không sở hữu tài nguyên cụ thể	Được cấp một số tài nguyên để chứa tiến trình và thực hiện lệnh

- Tiến trình được sinh ra khi chương trình được tải vào bộ nhớ để thực hiện
  - Tiến trình người dùng
  - Tiến trình hệ thống

2. Trạng thái của tiến trình

- Phân biệt theo 2 trạng thái: chạy và không chạy
- => Không phản ánh đầy đủ thông tin về trạng thái tiến trình
- => Mô hình 5 trạng thái: *mới khởi tạo, sẵn sàng, chạy, chờ đợi, kết thúc*

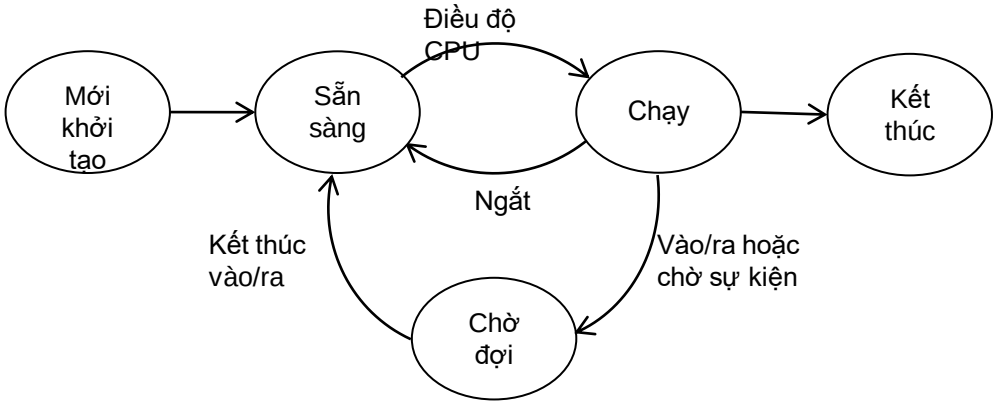
- **Mới khởi tạo:** tiến trình đang được tạo ra
- **Sẵn sàng**: tiến trình chờ được cấp CPU để thực hiện lệnh của mình
- **Chạy**: lệnh của tiến trình được CPU thực hiện
- **Chờ đợi**: tiến trình chờ đợi một sự kiện gì đó xảy ra (blocked)
- **Kết thúc:** tiến trình đã kết thúc việc thực hiện nhưng vẫn chưa bị xóa



Sơ đồ chuyển đổi trạng thái của tiến trình

2. Trạng thái của tiến trình

- Khởi tạo -> sẵn sàng: tiến trình khởi tạo xong và đã được tải vào bộ nhớ, chỉ chờ được cấp Cpu để chạy.
- Sẵn sàng -> chạy: do kết quả điều độ CPU của hđh, tiến trình được HDH cấp phát CPU và chuyển sang trạng thái chạy
- Chạy -> sẵn sàng: HĐH cấp phát CPU cho tiến trình khác, do kết quả điều độ/do ngắt xảy ra, tiến trình hiện thời chuyển sang trạng thái sẵn sàng và chờ được cấp CPU để chạy tiếp.
- Chạy -> chờ đợi: Khi tiến trình đó chỉ chạy khi có 1 sự kiện nào đó xảy ra, chuyển sang tt chờ được phân phối Cpu để chạy tiếp.
- Chạy -> end: khi TT đã thực hiện xong.



Sơ đồ chuyển đổi trạng thái của tiến trình

3. Thông tin mô tả tiến trình

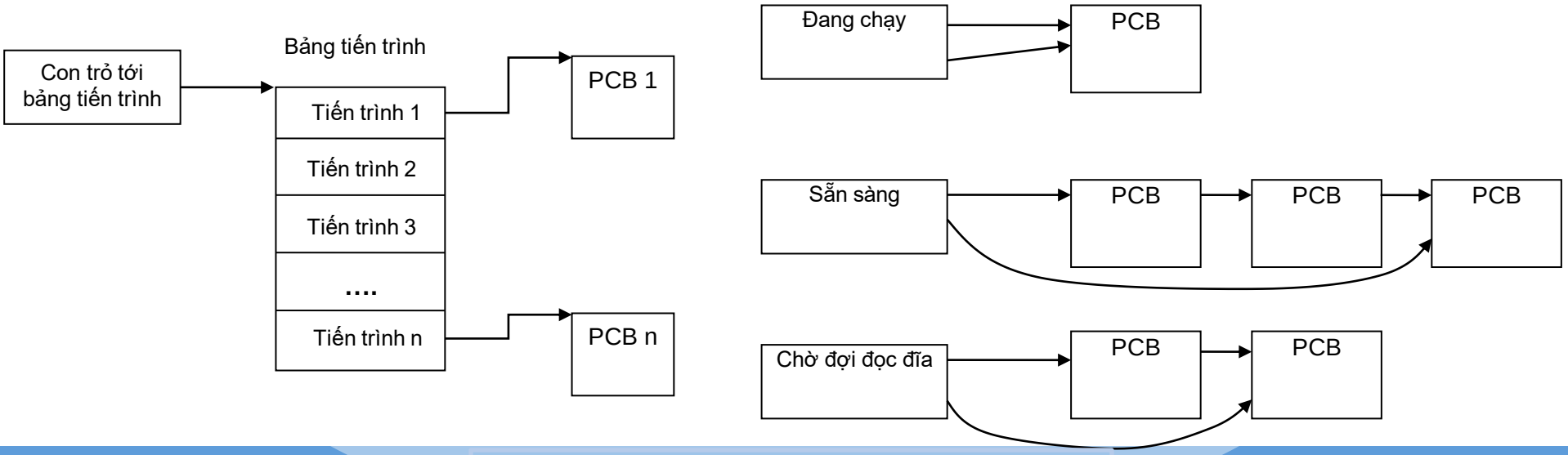
- Để có thể quản lý tiến trình, HĐH cần có các thông tin về tiến trình đó.
- Thông tin của tiến trình được lưu trong một cấu trúc dữ liệu gọi là khối quản lý tiến trình - PCB (Process Control Block)
- Các thông tin chính trong PCB:
  - Số định danh của tiến trình (PID) Trạng thái tiến trình: một trong năm trạng thái
  - Nội dung một số thanh ghi CPU:
    - Thanh ghi con trỏ lệnh: trỏ tới lệnh tiếp theo
    - Thanh ghi con trỏ ngăn xếp: lưu tham số/tình trạng hàm khi thực hiện lời gọi hàm/thủ tục của chương trình
    - Các thanh ghi điều kiện và trạng thái: chứa trạng thái sau khi thực hiện phép toán logic/số học
    - Các thanh ghi đa năng

3. Thông tin mô tả tiến trình

- Lý do phải lưu lại các thanh ghi này trong PCB là do tiến trình có thể bị chuyển khỏi trạng thái chạy để nhường chỗ cho tiến trình khác. Khi tiến trình quay trời lại, HĐH sẽ dùng thông tin từ PCB để khôi phục lại nội dung các thanh ghi, cho phép tiến trình thực hiện lại từ trạng tháitrước lúc dừng.
- PCB:
  - Thông tin phục vụ điều độ tiến trình: mức độ ưu tiên của tiến trình, vị trí trong hàng đợi, …
  - Thông tin về bộ nhớ của tiến trình
  - Danh sách các tài nguyên khác: các file đang mở, thiết bị vào ra mà tiến trình sử dụng
  - Thông tin thống kê phục vụ quản lý: thời gian sử dụng CPU, giới hạn thời gian

4. Bảng và danh sách tiến trình

- Để quản lý, HĐH cần biết vị trí các PCB. HĐH sử dụng bảng tiến trình chứa con trỏ tới PCB của toàn bộ tiến trình có trong hệ thống
- Ngoài ra, PCB của các tiến trình cùng trạng thái hoặc cùng chờ 1 tài nguyên nào đó được liên kết thành 1 danh sách, mỗi danh sách gồm một số tiến trình cùng trạng thái.



3. Các thao tác với tiến trình

Hoạt động quản lý tiến trình bao gồm một số công việc như tạo mới, kết thúc tiến trình, chuyển đổi giữa các tiến trình, điều độ, đồng bộ hóa, đảm bảo việc liên lạc giữa các tiến trình.

1. Tạo mới tiến trình:

Để tạo ra một tiến trình mới, HDH thực hiện một số bước như sau:

- Gán số định danh cho tiến trình được tạo mới và tạo một ô trong bảng tiến trình
- Tạo không gian nhớ cho tiến trình và PCB
- Khởi tạo PCB
- Liên kết PCB của tiến trình vào các danh sách quản lý

2. Kết thúc tiến trình:

- Kết thúc bình thường: yêu cầu HDH kết thúc mình bằng cách gọi lời gọi hệ thống exit()
- Bị kết thúc:
  - Bị tiến trình cha kết thúc
  - Do các lỗi
  - Yêu cầu nhiều bộ nhớ hơn so với số lương hệ thống có thể cung cấp
  - Thực hiện lâu hơn thời gian giới hạn
  - Do quản trị hệ thống hoặc hệ điều hành kết thúc

3. Chuyển đổi giữa các tiến trình:

- Trong quá trình thực hiện, CPU có thể được chuyển tử tiến trình hiện thời sang thực hiện tiến trình khác.
- Thông tin về tiến trình hiện thời (chứa trong PCB) được gọi là ngữ cảnh (context) của tiến trình. Việc chuyển giữa tiến trình còn được gọi là chuyển đổi ngữ cảnh
- Việc chuyển đổi tiến trình xảy ra khi:
  - Có ngắt: ngắt do đồng hồ/ngắt vào/ra
  - Tiến trình gọi lời gọi hệ thống
- Trước khi chuyển sang thực hiện tiến trình khác, ngữ cảnh được lưu vào PCB
- Khi được cấp phát CPU thực hiện trở lại, ngữ cảnh được khôi phục từ PCB vào các thanh ghi và bảng tương ứng.

3. Chuyển đổi giữa các tiến trình:

- Thông tin nào phải được cập nhật và lưu vào PCB khi chuyển tiến trình? => Tùy từng trường hợp:
- TH đơn giản: Hệ thống chuyển sang thực hiện ngắt vào/ra rồi quay lại thực hiện tiếp tiến trình:
  - Ngữ cảnh gồm thông tin có thể bị hàm xử lý ngắt thay đổi
  - => nội dung thanh ghi, trạng thái CPU

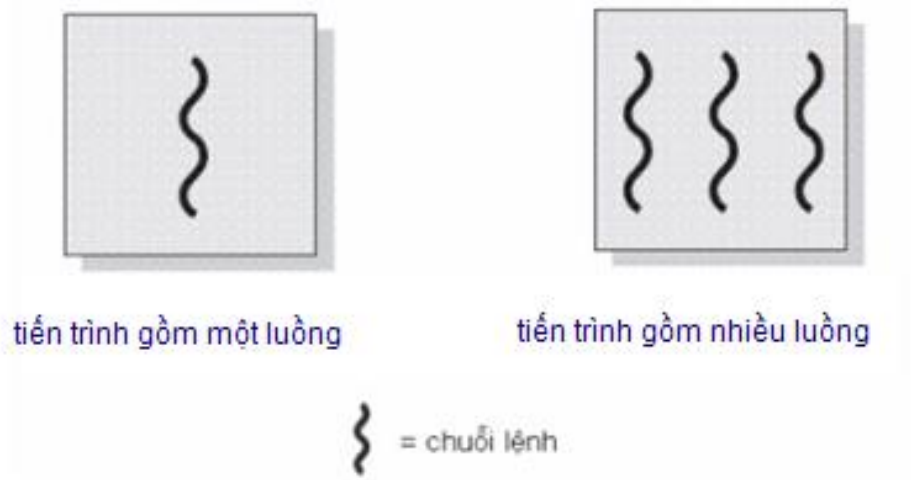
- TH phức tạp: Sau khi thực hiện ngắt, hệ thống thực hiện tiến trình khác
  - Thay đổi trạng thái tiến trình
  - Cập nhật thông tin thống kê trong PCB
  - Chuyển liên kết PCB của tiến trình vào danh sách ứng với trạng thái mới
  - Cập nhật PCB của tiến trình mới được chọn
  - Cập nhật nội dung thanh ghi và trạng thái CPU
- => Chuyển đổi tiến trình đòi hỏi thời gian

- Tiến trình được xem xét từ 2 khía cạnh:
  - Tiến trình là 1 đơn vị sở hữu tài nguyên
  - Tiến trình là 1 đơn vị thực hiện công việc tính toán xử lý
- Các HDH trước đây: mỗi tiến trình chỉ tương ứng với 1 đơn vị xử lý duy nhất
- => Tiến trình không thể thực hiện nhiều hơn một công việc cùng một lúc

II. LUỒNG THỰC HIỆN

1. Khái niệm (tt)

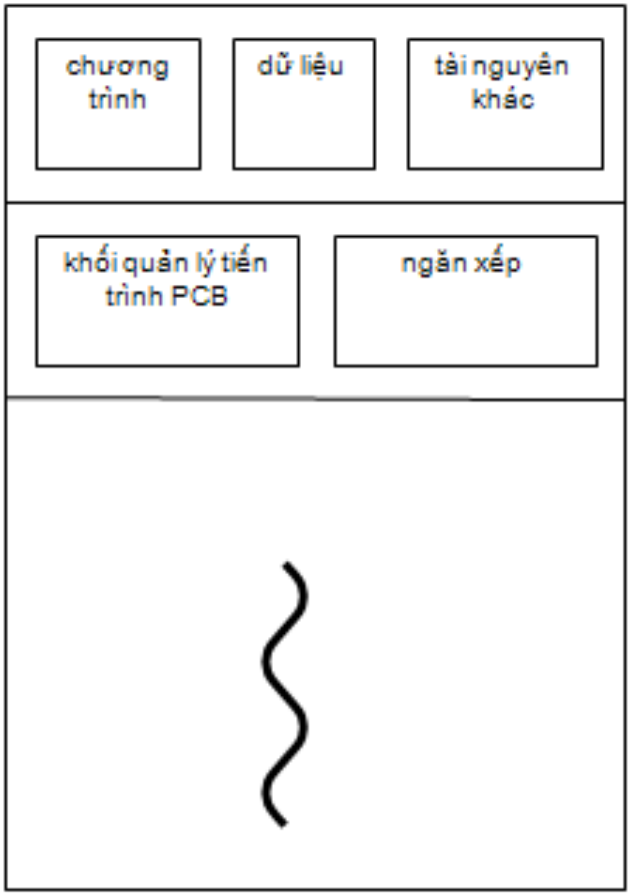
- HDH hiện đại: cho phép tách riêng vai trò thực hiện lệnh của tiến trình
- **Mỗi đơn vị thực hiện lệnh của tiến trình, tức là 1 chuỗi lệnh được cấp phát CPU để thực hiện độc lập được gọi là một *luồng thực hiện***
- HDH hiện nay thường hỗ trợ đa luồng (multithreading) => cho phép nhiều chuỗi lệnh được thực hiện cùng một lúc



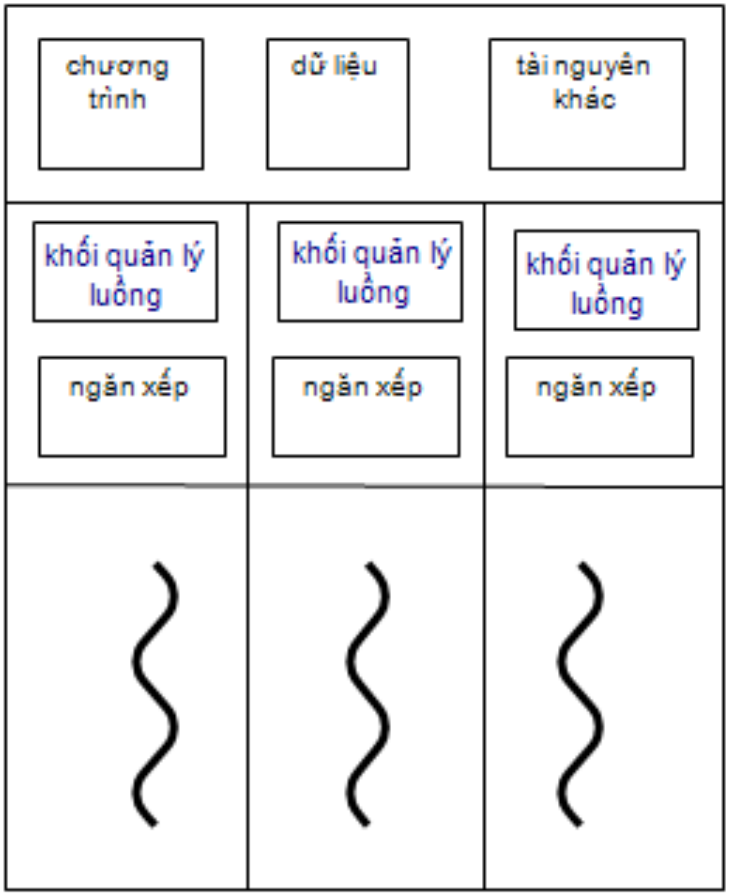
- Trong hệ thống cho phép đa luồng, tiến trình vẫn là 1 đơn vị để HDH phân phối tài nguyên
- Mỗi tiến trình sở hữu chung một số tài nguyên:
  - Không gian nhớ của tiến trình (logic): chứa chương trình (các lệnh), phần dữ liệu cuả tiến trình.
  - Các tài nguyên khác: các file đang mở, thiết bị hoặc cổng vào/ra

- Đến đây, có sự khác biệt giữa tiến trình đơn luồng và tiến trình đa luồng.
- Mô hình đơn luồng:
  - Tiến trình có khối quản lý PCB chứa đầy đủ thông tin trạng thái tiến trình, giá trị thanh ghi
  - Ngăn xếp chứa tham số, trạng thái hàm/ thủ tục/ chương trình con
  - Khi tiến trình thực hiện, nó sẽ làm chủ nội dung các thanh ghi và con trỏ lệnh

- Mô hình đa luồng:
  - Mỗi luồng cần có khả năng quản lý con trỏ lệnh, nội dung thanh ghi
  - Luồng cũng có trạng thái riêng
  - => chứa trong khối quản lý luồng
  - Luồng cũng cần có ngăn xếp riêng
  - Tất cả các luồng của 1 tiến trình chia sẻ không gian nhớ và tài nguyên
  - Các luồng có cùng không gian địa chỉ và có thể truy cập tới dữ liệu của tiến trình



Tiến trình đơn luồng



Tiến trình nhiều luồng

3. Ưu điểm của mô hình đa luồng

- Tăng hiệu năng và tiết kiệm thời gian
- Dễ dàng chia sẻ tài nguyên và thông tin
- Tăng tính đáp ứng
- Tận dụng được kiến trúc xử lý với nhiều CPU
- Thuận lợi cho việc tổ chức chương trình

- Có thể tạo và quản lý luồng ở 2 mức:
  - Mức người dùng
  - Mức nhân
- Luồng mức người dùng: được tạo ra và quản lý không có sự hỗ trợ của HDH
- Luồng mức nhân: được tạo ra và quản lý bởi HDH

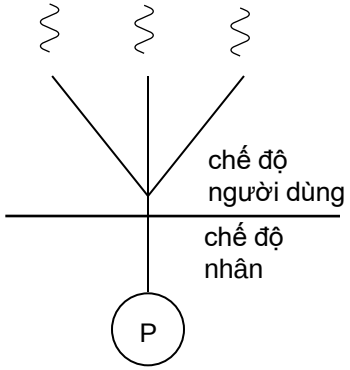
- Do trình ứng dụng tự tạo ra và quản lý
- HĐH không biết về sự tồn tại của những dòng như vậy
- Sử dụng thư viện do ngôn ngữ lập trình cung cấp
- HDH vẫn coi tiến trình như một đơn vị duy nhất với một trạng thái duy nhất
- Việc phân phối CPU được thực hiện cho cả tiến trình

- Ưu điểm:
  - Việc chuyển đổi luồng không đòi hỏi chuyển sang chế độ nhân => tiết kiệm thời gian
  - Trình ứng dụng có thể điều độ theo đặc điểm riêng của mình, không phụ thuộc vào cách điều độ của HDH
  - Có thể sử dụng trên cả những HDH không hỗ trợ đa luồng

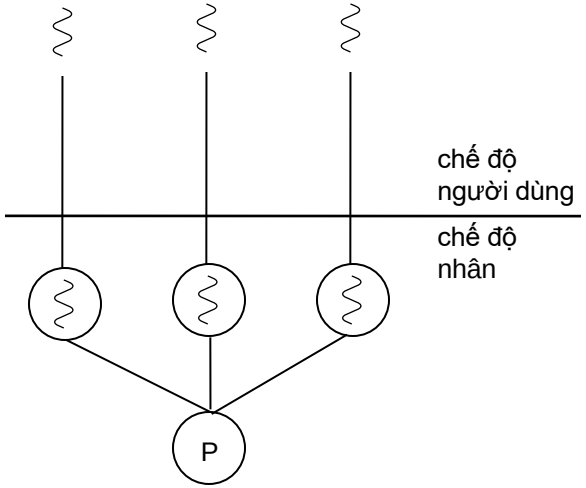
- Nhược điểm:
  - Khi một luồng gọi lời gọi hệ thống và bị phong tỏa thì toàn bộ tiến trình bị phong tỏa
  - => không cho phép tận dụng ưu điểm về tính đáp ứng của mô hình đa luồng
  - Không cho phép tận dụng kiến trúc nhiều CPU do HĐH phân phối CPU cho cả tiến trình chứ không phải từng dòng cụ thể

- Luồng mức nhân được HĐH tạo ra và quản lý
- HĐH cung cấp giao diện lập trình: gồm các lời gọi hệ thống mà trình ứng dụng có thể yêu cầu tạo/ xóa luồng và thay đổi tham số liên quan quản lý dòng
- Tăng tính đáp ứng và khả năng thực hiện đồng thời của các luồng trong cùng tiến trình
- Tạo và chuyển đổi luồng thực hiện trong chế độ nhân => tốc độ chậm
- HĐH Windows và Linux hỗ trợ luồng mức nhân

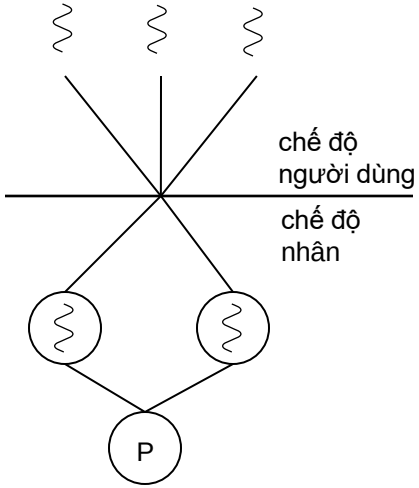
- Có thể sử dụng dòng mức người dùng và dòng mức nhân
- Theo các tổ chức này:
  - Luồng mức người dùng được tạo ra trong chế độ người dùng nhờ thư viện cuả trình ứng dụng
  - Luồng mức người dùng được ánh xạ lên số lượng tương ứng hoặc ít hơn các luồng mức nhân
  - Số lượng dòng mức nhân phụ thuộc vào hệ thống cụ thể, chẳng hạn hệ thống nhiều CPU sẽ có nhiều dòng mức nhân hơn.



a) Mô hình mức người dùng



b) Mô hình mức nhân



c) Mô hình kết hợp

III. ĐIỀU ĐỘ TIẾN TRÌNH

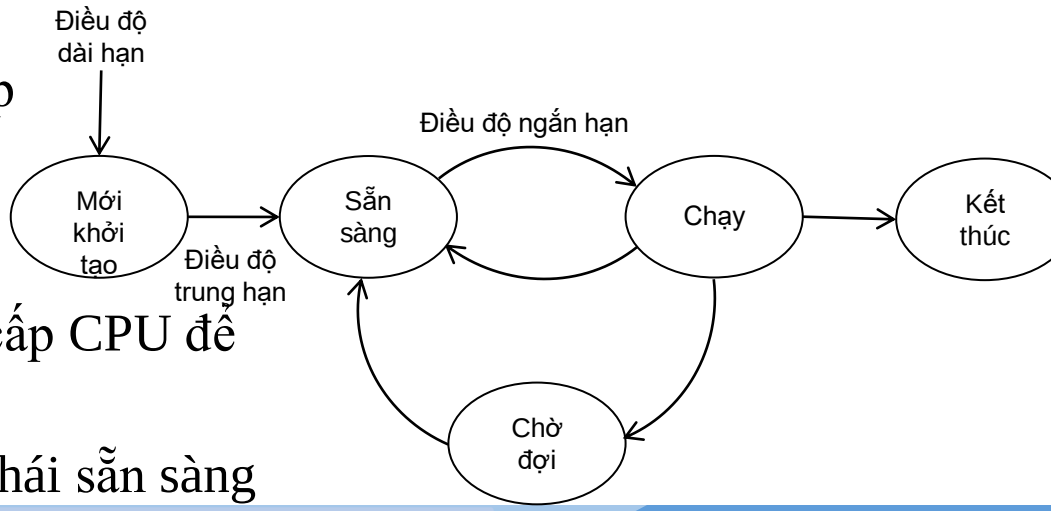
1. Khái niệm điều độ

- *Trong hệ thống cho phép đa chương trình, nhiều tiến trình có thể tồn tại và thực hiện cùng lúc.*
- *Kỹ thuật đa chương trình có nhiều ưu điểm do cho cho phép sử dụng CPU hiệu quả, đáp ứng nhu cầu tính toán của người dùng*
- *Tuy nhiên, đặt ra nhiều vấn đề phức tạp hơn đối với HĐH*

- *Điều độ* (scheduling) hay *lập lịch* là quyết định tiến trình nào được sử dụng tài nguyên phần cứng khi nào, trong thời gian bao lâu
- Tập trung vào vấn đề điều độ đối với CPU
- => Quyết định thứ tự và thời gian sử dụng CPU
- Điều độ tiến trình và điều độ dòng:
  - Hệ thống trước kia: tiến trình là đơn vị thực hiện chính => điều độ thực hiện với tiến trình
  - Hệ thống hỗ trợ luồng: luồng mức nhân là đơn vị thực hiện được HDH cấp CPU chứ không phải tiến trình.
  - => Sử dụng thuật ngữ điều độ tiến trình rộng rãi ⇔ điều độ luồng

1. Điều độ dài hạn và ngắn hạn

- Điều độ dài hạn:
  - Thực hiện khi mới tạo ra tiến trình
  - HDH quyết định tiến trình có được thêm vào danh sách đang hoạt động?
  - Nếu được chấp nhận, hệ thống sẽ có thêm tiến trình mới. Ngược lại, chờ tới thời điểm khác để tạo và thực hiện
  - Ảnh hưởng tới mức độ đa chương trình
- Điều độ trung hạn:
  - Quyết định tiến trình có được cấp MEM để thực hiện?
- Điều độ ngắn hạn:
  - Quyết định tiến trình nào được cấp CPU để thực hiện
  - Thực hiện với tiến trình ở trạng thái sẵn sàng



2. Điều độ có phân phối lại và không phân phối lại:

- Điều độ có phân phối lại (preemptive):
  - HDH có thể sử dụng cơ chế ngắt để thu hồi CPU của một tiến trình đang trong trạng thái chạy
- Điều độ không phân phối lại (nonpreemptive):
  - Tiến trình đang ở trạng thái chạy sẽ được sử dụng CPU cho đến khi xảy ra một trong các tình huống sau:
    - Tiến trình kết thúc
    - Tiến trình phải chuyển sang trạng thái chờ đợi do thực hiện I/O
  - => Điều độ hợp tác: chỉ thực hiện được khi tiến trình hợp tác và nhường CPU
  - Nếu tiến trình không hợp tác, dùng CPU vô hạn => các tiến trình khác không được cấp CPU (Windows 96, NT)

2. Điều độ có phân phối lại:

- So với điều độ không phân phối lại, điều độ có phân phối lại có nhiều ưu điểm hơn.
- HDH chủ động hơn, không phụ thuộc vào hoạt động của tiến trình
- Đảm bảo chia sẻ thời gian thực sự
- Đòi hỏi phần cứng có bộ định thời gian và một số hỗ trợ khác
- Vấn đề quản lý tiến trình phức tạp hơn

Một số tiêu chí thường được sử dụng:

1. Lượng tiến trình được thực hiện xong:
   - Số lượng tiến trình thực hiện xong trong 1 đơn vị thời gian
   - Đo tính hiệu quả của hệ thống
2. Hiệu suất sử dụng CPU
   - Cố gắng để CPU càng ít phải nghỉ càng tốt
3. Thời gian vòng đời trung bình của tiến trình:
   - Từ lúc có yêu cầu tạo tiến trình đến khi kết thúc
4. Thời gian chờ đợi:
   - Tổng thời gian tiến trình nằm trong trạng thái sẵn sàng và chờ cấp CPU
   - Ảnh hưởng trực tiếp của thuật toán điều độ tiến trình

5. Thời gian đáp ứng
   - Đây là tiêu chí hướng tới người dùng và thường được sử dụng trong hệ thống tương tác trực tiếp.
6. Tính dự đoán được:
   - Vòng đời, thời gian chờ đợi, thời gian đáp ứng phải ổn định, không phụ thuộc vào tải của hệ thống
7. Tính công bằng
   - Các tiến trình cùng độ ưu tiên phải được đối xử như nhau

1. Thuật toán đến trước phục vụ trước (FCFS):

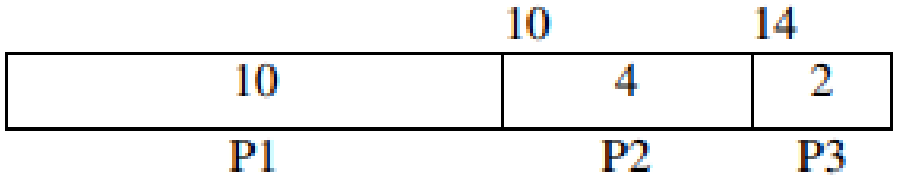
- Tiến trình yêu cầu CPU trước sẽ được cấp trước
- HDH xếp các tiến trình sẵn sàng vào hàng đợi FIFO
- Tiến trình mới được xếp vào cuối hàng đợi
- Đơn giản, đảm bảo tính công bằng
- Thời gian chờ đợi trung bình lớn
- => Ảnh hưởng lớn tới hiệu suất chung của toàn hệ thống
- Thường là thuật toán điều độ không phân phối lại, sau khi tiến trình được cấp CPU, tiến trình đó sẽ sử dụng CPU đến khi kết thúc hoặc phải dừng lại để chờ kết quả vào ra.

1. Thuật toán đến trước phục vụ trước (FCFS):

Cho 3 tiến trình với thứ tự xuất hiện và độ dài chu kỳ CPU như sau:

Tiến trình	Độ dài chu kỳ CPU
P1	10
P2	4
P3	2

Kết quả điều độ theo thuật toán FCFS thể hiện trên hình sau:



Thời gian chờ đợi của P1, P2, P3 lần lượt là 0, 10, và 14.

Thời gian chờ đợi trung bình = (0 + 10 + 14)/3 = 8.

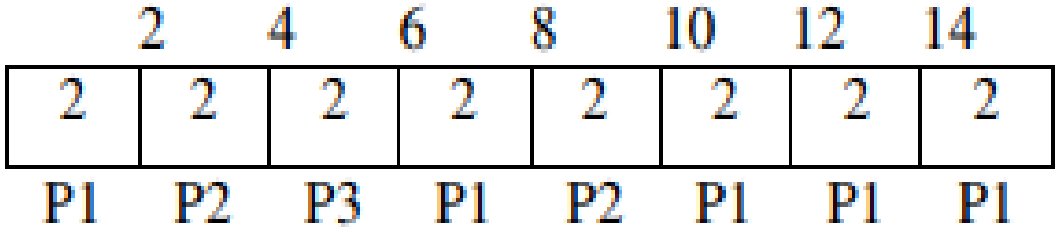
2. Điều độ quay vòng (RR: round robin):

- Sửa đổi FCFS dùng cho các hệ chia sẻ thời gian
- Có thêm cơ chế phân phối lại bằng cách sử dụng ngắt của đồng hồ
- Hệ thống xác định những khoảng thời gian nhỏ gọi là *lượng tử/ lát cắt thời gian t*
- Khi CPU được giải phóng, HDH đặt thời gian của đồng hồ bằng độ dài lượng tử, lấy tiến trình ở đầu hàng đợi và cấp CPU cho nó
- Tiến trình kết thúc trước khi hết thời gian t: trả quyền điều khiển cho HDH

2. Điều độ quay vòng (tt)

- Hết lượng tử thời gian mà tiến trình chưa kết thúc:
  - Đồng hồ sinh ngắt
  - Tiến trình đang thực hiện bị dừng lại
  - Quyền điều khiển chuyển cho hàm xử lý ngắt của HDH
  - HDH chuyển tiến trình về cuối hàng đợi, lấy tiến trình ở đầu và tiếp tục
- Cải thiện thời gian đáp ứng so với FCFS
- Thời gian chờ đợi trung bình vẫn dài
- Lựa chọn độ dài lượng tử thời gian?

2. Điều độ quay vòng (tt)



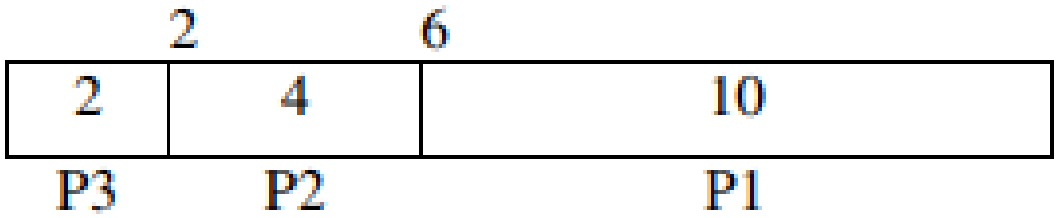
Thời gian chờ đợi của P1, P2, P3 lần lượt là 6, 6, và 4.

Thời gian chờ đợi trung bình = (6 + 6 + 4)/3=5,33.

3. Điều độ ưu tiên tiến trình ngắn nhất (SPF)

- Chọn trong hàng đợi tiến trình có chu kỳ sử dụng CPU tiếp theo ngắn nhất để phân phối CPU
- Nếu có nhiều tiến trình với chu kỳ CPU tiếp theo bằng nhau, chọn tiến trình đứng trước
- Thời gian chờ đợi trung bình nhỏ hơn nhiều so với FCFS
- Khó thực hiện vì phải biết độ dài chu kỳ CPU tiếp:
  - Trong các hệ thống xử lý theo mẻ: dựa vào thời gian đăng kí tối đa do lập trình viên cung cấp
  - Dự đoán độ dài chu kỳ CPU tiếp theo: dựa trên độ dài TB các chu kỳ CPU trước đó
- Không có phân phối lại

3. Điều độ ưu tiên tiến trình ngắn nhất (SPF)



Thời gian chờ đợi trung bình = (6 + 2 +0)/3 = 2,67.

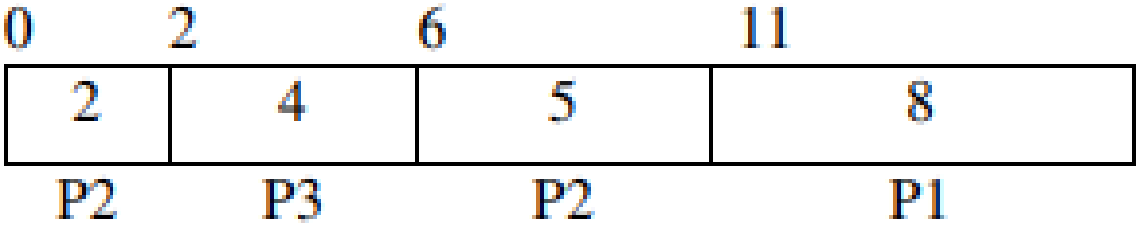
4. Điều độ ưu tiên thời gian còn lại ngắn nhất

- SPF có thêm cơ chế phân phối lại (SRTF)
- Khi 1 tiến trình mới xuất hiện trong hàng đợi, HDH so sánh thời gian còn lại của tiến trình đang chạy với thời gian còn lại của tiến trình mới xuất hiện
- Nếu tiến trình mới xuất hiện có thời gian còn lại ngắn hơn, HDH thu hồi CPU của tiến trình đang chạy, phân phối cho tiến trình mới
- Thời gian chờ đợi trung bình nhỏ
- HDH phải dự đoán độ dài chu kỳ CPU của tiến trình
- Việc chuyển đổi tiến trình ít hơn so với RR

4. Điều độ ưu tiên thời gian còn lại ngắn nhất (SRTF)

Tiến trình	Thời điểm xuất hiện	Độ dài chu kỳ CPU
P1	0	8
P2	0	7
P3	2	2

Kết quả điều độ sử dụng SRTF được thể hiện trên biểu đồ sau:

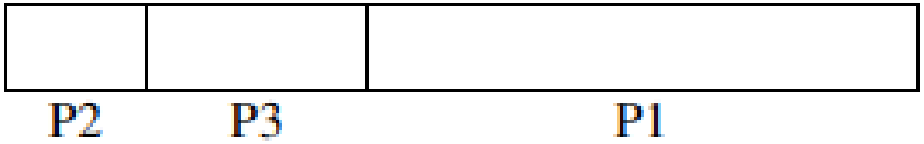


5. Điều độ có mức ưu tiên

- Mỗi tiến trình có 1 mức ưu tiên
- Tiến trình có mức ưu tiên cao hơn sẽ được cấp CPU trước
- Các tiến trình có mức ưu tiên bằng nhau được điều độ theo FCFS
- Mức ưu tiên được xác định theo nhiều tiêu chí khác nhau

5. Điều độ có mức ưu tiên

Tiến trình	Mức ưu tiên
P1	4
P2	1
P3	3



IV. ĐỒNG BỘ HÓA CÁC TIẾN TRÌNH ĐỒNG THỜI

- Những tiến trình cùng tồn tại được gọi *là tiến trình đồng thời /tiến trình tương tranh*
- Quản lý tiến trình đồng thời là vấn đề quan trọng:
  - Liên lạc giữa các tiến trình
  - Cạnh tranh và chia sẻ tài nguyên
  - Phối hợp và đồng bộ hóa các tiến trình
  - Vấn đề bế tắc
  - Đói tài nguyên (starvation)

1. Tiến trình cạnh tranh tài nguyên với nhau:

- Đảm bảo ***loại trừ tương hỗ*** (mutual exclusion):
  - Khi các tiến trình cùng truy nhập tài nguyên mà khả năng chia sẻ của tài nguyên đó là có hạn
  - => phải đảm bảo tiến trình này đang truy cập tài nguyên thì tiến trình khác không được truy cập
  - Tài nguyên đó gọi là ***tài nguyên nguy hiểm***. Đoạn chương trình có yêu cầu sử dụng tài nguyên nguy hiểm gọi là ***đoạn nguy hiểm*** (critical section)
  - Mỗi thời điểm chỉ có 1 tiến trình nằm trong đoạn nguy hiểm => loại trừ lẫn nhau

1. Tiến trình cạnh tranh tài nguyên với nhau (tt):

- Không để xảy ra ***bế tắc*** (deadlock):
  - Bế tắc: tình trạng hai hoặc nhiều tiến trình không thể thực hiện tiếp do chờ đợi lẫn nhau
- Không để ***đói*** tài nguyên (starvation):
  - Tình trạng chờ đợi quá lâu mà không đến lượt sử dụng tài nguyên

2. Tiến trình hợp tác với nhau qua tài nguyên chung:

- Có thể trao đổi thông tin bằng cách chia sẻ vùng nhớ dùng chung (biến toàn thể)
- Đòi hỏi đảm bảo loại trừ tương hỗ
- Xuất hiện tình trạng bế tắc và đói
- Yêu cầu đảm bảo tính nhất quán dữ liệu
- ***Điều kiện chạy đua*** (race condition): tình huống mà một số dòng /tiến trình đọc, ghi dữ liệu sử dụng chung và kết quả phụ thuộc vào thứ tự các thao tác đọc, ghi
- => Đặt thao tác truy cập và cập nhật dữ liệu dùng chung vào đoạn nguy hiểm

3. Tiến trình có liên lạc nhờ gửi thông điệp:

- Có thể trao đổi thông tin trực tiếp với nhau bằng cách gửi thông điệp (message passing)
- Không có yêu cầu loại trừ tương hỗ
- Có thể xuất hiện bế tắc và đói

2. Yêu cầu với giải pháp cho đoạn nguy hiểm

- Loại trừ tương hỗ
- Tiến triển
- Chờ đợi có giới hạn
- Các giả thiết:
  - Giải pháp không phụ thuộc vào tốc độ của các tiến trình
  - Không tiến trình nào được phép nằm quá lâu trong đoạn nguy hiểm
  - Thao tác đọc và ghi bộ nhớ là thao tác nguyên tử (atomic) và không thể bị xen ngang giữa chừng

- Các giải pháp được chia thành 3 nhóm chính:
  - Nhóm giải pháp phần mềm
  - Nhóm giải pháp phần cứng
  - Nhóm sử dụng hỗ trợ của HDH hoặc thư viện ngôn ngữ lập trình

3. Giải thuật peterson

- Giải pháp thuộc nhóm phần mềm
- Đề xuất ban đầu cho đồng bộ 2 tiến trình
- P0 và P1 thực hiện đồng thời với một tài nguyên chung và một đoạn nguy hiểm chung
- Mỗi tiến trình thực hiện vô hạn và xen kẽ giữa đoạn nguy hiểm với phần còn lại của tiến trình
- Yêu cầu 2 tiến trình trao đổi thông tin qua 2 biến chung:
  - Turn: xác định đến lượt tiến trình nào được vào đoạn nguy hiểm
  - Cờ cho mỗi tiến trình: flag[i]=true nếu tiến trình thứ i yêu cầu được vào đoạn nguy hiểm

3. Giải thuật peterson (tt)

bool flag[2]; int turn;

```
Void P0() {
    for(;;)  { //lặp vô hạn
        flag[0] = true;
        turn = 1;
        while(flag[1] && turn ==1) ;
//lặp đến khi điều kiện không thỏa
          <đoạn nguy hiểm>
          flag[0] = false;
          <phần còn lại của tiến trình>
    }
}
```

```
Void P1() {
    for(;;)  {  //lặp vô hạn
        flag[1] = true;
        turn = 0;
        while(flag[0] && turn ==0) ;
//lặp đến khi điều kiện không thỏa
          <đoạn nguy hiểm>
          flag[1] = false;
          <phần còn lại của tiến trình>
    }
}
```

```
Void main() {
        flag[0] = flag[1] = false;  turn =0;
//tắt tiến trình chính, chạy đồng thời 2 tiến trình P0 và P1
        startProcess (P0); startProcess(P1);
}
```

3. Giải thuật peterson (tt)

- Thỏa mãn các yêu cầu:
  - Điều kiện loại trừ tương hỗ
  - Điều kiện tiến triển:
    - P0 chỉ có thể bị P1 ngăn cản vào đoạn nguy hiểm nếu flag[1] = true và turn =1 luôn đúng
    - Có 2 khả năng với P1 ở ngoài đoạn nguy hiểm:
      - P1 chưa sẵn sàng vào đoạn nguy hiểm => flag[1] = false, P0 có thể vào ngay đoạn nguy hiểm
      - P1 đã đặt flag[1]=true và đang trong vòng lặp while => turn = 1 hoặc 0
        - Turn = 0: P0 vào đoạn nguy hiểm ngay
        - Turn = 1: P1 vào đoạn nguy hiểm, sau đó đặt flag[1] = false => quay lại khả năng 1
  - Chờ đợi có giới hạn

3. Giải thuật peterson (tt)

- Sử dụng trên thực tế tương đối phức tạp
- Đòi hỏi tiến trình đang yêu cầu vào đoạn nguy hiểm phải nằm trong trạng thái ***chờ đợi tích cực***
- Chờ đợi tích cực: tiến trình vẫn phải sử dụng CPU để kiểm tra xem có thể vào đoạn nguy hiểm?
- => Lãng phí CPU

1. Cấm các ngắt:

- Tiến trình đang có CPU: thực hiện cho đến khi tiến trình đó gọi dịch vụ hệ điều hành hoặc bị ngắt
- => cấm không để xẩy ra ngắt trong thời gian tiến trình đang ở trong đoạn nguy hiểm để truy cập tài nguyên
- Đảm bảo tiến trình được thực hiện trọn vẹn đoạn nguy hiểm và không bị tiến trình khác chen vào
- Đơn giản
- Giảm tính mềm dẻo của HDH
- Không áp dụng với máy tính nhiều CPU

2. Sử dụng lệnh máy đặc biệt:

- Phần cứng được thiết kế có một số lệnh máy đặc biệt
- 2 thao tác kiểm tra và thay đổi giá trị cho một biến, hoặc các thao tác so sánh và hoán đổi giá trị hai biến, được thực hiện trong cùng một lệnh máy
- => Đảm bảo được thực hiện cùng nhau mà không bị xen vào giữa – thao tác nguyên tử (atomic)
- Gọi là lệnh "kiểm tra và xác lập" – Test_and_Set

2. Sử dụng lệnh máy đặc biệt (tt):

- Logic của lệnh Test_and_Set:

```
Bool Test_and_Set(bool & val)
{
        bool temp = val;
        val = true;
        return temp;
}
```

2. Sử dụng lệnh máy đặc biệt (tt):

```
const int n; //n là số lượng tiến trình
bool lock;
void P(int i){    //tiến trình P(i)
    for(;;){    //lặp vô hạn
        while(Test_and_Set(lock));//lặp đến khi điều kiện không thỏa
        <Đoạn nguy hiểm>
        lock = false;
        <Phần còn lại của tiến trình>
    }
}
void main(){
            lock = false;
//tắt tiến trình chính, chạy đồng thời n tiến trình
            StartProcess(P(1)); ….    StartProcess(P(n));
}
```

2. Sử dụng lệnh máy đặc biệt (tt):

- Ưu điểm:
  - Việc sử dụng tương đối đơn giản và trực quan
  - Có thể dùng để đồng bộ nhiều tiến trình
  - Có thể sử dụng cho trường hợp đa xử lý với nhiều CPU nhưng có bộ nhớ chung
- Nhược điểm:
  - Chờ đợi tích cực
  - Có thể gây đói

5. Cờ hiệu

- Cờ hiệu S là 1biến nguyên được khởi tạo bằng khả năng phục vụ đồng thời của tài nguyên
- Giá trị của S chỉ có thể thay đổi nhờ gọi 2 thao tác là Wait và Signal
  - Wait(S):
    - Giảm S đi 1 đơn vị
    - Nếu giá trị của S<0 thì tiến trình gọi wait(S) sẽ bị phong tỏa
  - Signal(S):
    - Tăng S lên 1 đơn vị
    - Nếu giá trị của S≤0: 1 trong các tiến trình đang bị phong tỏa được giải phóng và có thể thực hiện tiếp

5. Cờ hiệu (tt)

- Wait và Signal là những thao tác nguyên tử
- Để tránh tình trạng chờ đợi tích cực, sử dụng 2 thao tác phong tỏa và đánh thức:
  - Nếu tiến trình thực hiện thao tác wait, giá trị cờ hiệu âm thì thay vì chờ đợi tích cực nó sẽ bị phong tỏa (bởi thao tác block) và thêm vào hàng đợi của cờ hiệu
  - Khi có 1 tiến trình thực hiện thao tác signal thì 1 trong các tiến trình bị khóa sẽ được chuyển sang trạng thái sẵn sàng nhờ thao tác đánh thức (wakeup) chứa trong signal

5. Cờ hiệu (tt)

```
struct semaphore {
        int value;
        process *queue;//danh sách chứa các tiến trình bị phong tỏa
};
void Wait(semaphore& S) {
        S.value--;
        if (S.value < 0) {
        Thêm tiến trình gọi Wait vào S.queue
        block();   //phong tỏa tiến trình
        }
}
void Signal(semaphore& S) {
        S.value++;
        if (S.value <= 0) {
        Lấy một tiến trình P từ S.queue
        wakeup(P);
        }
}
```

5. Cờ hiệu (tt)

- Cờ hiệu được tiến trình sử dụng để gửi tín hiệu trước khi vào và sau khi ra khỏi đoạn nguy hiểm
- Khi tiến trình cần truy cập tài nguyên, thực hiện thao tác Wait của cờ hiệu tương ứng
  - Giá trị cờ hiệu âm sau khi giảm:
    - Tài nguyên được sử dụng hết khả năng
    - Tiến trình thực hiện Wait sẽ bị phong tỏa đến khi tài nguyên được giải phóng
    - Nếu tiến trình khác thực hiện Wait trên cờ hiệu, giá trị cờ hiệu sẽ giảm tiếp
    - Giá trị tuyệt đối của cờ hiệu âm tương ứng với số tiến trình bị phong tỏa
  - Sau khi dùng xong tài nguyên, tiến trình thực hiện thao tác Signal trên cùng cờ hiệu: tăng giá trị cờ hiệu và cho phép một tiến trình đang phong tỏa được thực hiện tiếp

5. Cờ hiệu (tt)

```
const int n; //n là số lượng tiến trình
semaphore S = 1;
void P(int i){    //tiến trình P(i)
    for(;;){    //lặp vô hạn
        Wait(S);
        <Đoạn nguy hiểm>
        Signal(S);
        <Phần còn lại của tiến trình>
    }
}
void main(){
//tắt tiến trình chính, chạy đồng thời n tiến trình
            StartProcess(P(1));
            ...
            StartProcess(P(n));
}
```

1. Bài toán triết gia ăn cơm:
   - 5 triết gia ngồi trên ghế quanh 1 bàn tròn
   - Trên bàn có 5 cái đũa: bên phải và bên trái mỗi người có 1 cái
   - Triết gia có thể nhặt 2 chiếc đũa theo thứ tự bất kì: phải nhặt từng chiếc một và đũa không nằm trong tay người khác
   - Khi cầm được cả 2 đũa: triết gia bắt đầu ăn và không đặt đũa trong thời gian ăn
   - Sau khi ăn xong, triết gia đặt 2 đũa xuống bàn
   - => 5 triết gia như 5 tiến trình đồng thời với tài nguyên nguy hiểm là đũa và đoạn nguy hiểm là đoạn dùng đũa để ăn
   - => Mỗi đũa được biểu diễn bằng 1 cờ hiệu
   - Nhặt đũa: wait(); đặt đũa xuống: signal()

1. Bài toán triết gia ăn cơm:

```
semaphore chopstick[5] = {1,1,1,1,1};
void Philosopher(int i){          //tiến trình P(i)
    for(;;){    //lặp vô hạn
        Wait(chopstick[i]);                    //lấy đũa bên trái
        Wait(chopstick[(i+1)%5]);              //lấy đũa bên phải
        <Ăn cơm>
        Signal(chopstick[(i+1)%5]);
        Signal(chopstick[i]);
        <Suy nghĩ>
    }
}
void main(){
// chạy đồng thời 5 tiến trình
            StartProcess(Philosopher(0));
            ...
            StartProcess(Philosopher (4));
}
```

2. Bài toán người sản xuất, người tiêu dùng với bộ đệm hạn chế:
   - Người sản xuất: tạo ra sản phẩm, xếp nó vào 1 chỗ gọi là bộ đệm, mỗi lần 1 sản phẩm
   - Người tiêu dùng: lấy sản phẩm từ bộ đệm, mỗi lần 1 sản phẩm, để sử dụng
   - Dung lượng bộ đệm hạn chế, chứa tối đa N sản phẩm

2. Bài toán người sản xuất, người tiêu dùng với bộ đệm hạn chế:

- 3 yêu cầu đồng bộ:
  1. Người sản xuất và tiêu dùng không được sử dụng bộ đệm cùng lúc
  2. Khi bộ đệm rỗng, người tiêu dùng không nên cố lấy sản phẩm
  3. Khi bộ đệm đầy, người sản xuất không được thêm sản phẩm
- Giải quyết bằng cờ hiệu:
  1. Yêu cầu 1: sử dụng cờ hiệu lock khởi tạo bằng 1
  2. Yêu cầu 2: cờ hiệu empty, khởi tạo bằng 0
  3. Yêu cầu 3: cờ hiệu full, khởi tạo bằng N

2. Bài toán người sản xuất, người tiêu dùng:

Const int N; // kích thước bộ đệm
Semaphore lock = 1;
Semaphore empty = 0;
Semaphore full = N

```
Void producer () {
    for (; ;) {
        <sản xuất>
        wait (full);
        wait (lock);
        <thêm 1 sản phẩm vào bộ đệm>
        signal (lock);
        signal (empty);
    }
}
```

```
Void consumer() {
    for (; ;) {
        wait (empty);
        wait (lock);
        <lấy 1 sản phẩm từ bộ đệm>
        signal (lock);
        signal (full);
        <tiêu dùng>
    }
}
```

```
Void main() {
   startProcess(producer);  startProcess(consumer);
}
```

7. Monitor

- Được định nghĩa dưới dạng một kiểu dữ liệu trừu tượng của ngôn ngữ lập trình bậc cao
- Gồm một dữ liệu riêng, hàm khởi tạo, và một số hàm hoặc phương thức để truy cập dữ liệu:
  - Tiến trình/dòng chỉ có thể truy cập dữ liệu của monitor thông qua các hàm hoặc phương thức của monitor
  - Tại mỗi thời điểm:
    - Chỉ một tiến trình được thực hiện trong monitor
    - Tiến trình khác gọi hàm của monitor sẽ bị phong tỏa, xếp vào hàng đợi của monitor để chờ cho đến khi monitor được giải phóng
  - => Đảm bảo loại trừ tương hỗ đối với đoạn nguy hiểm
  - Đặt tài nguyên nguy hiểm vào trong monitor

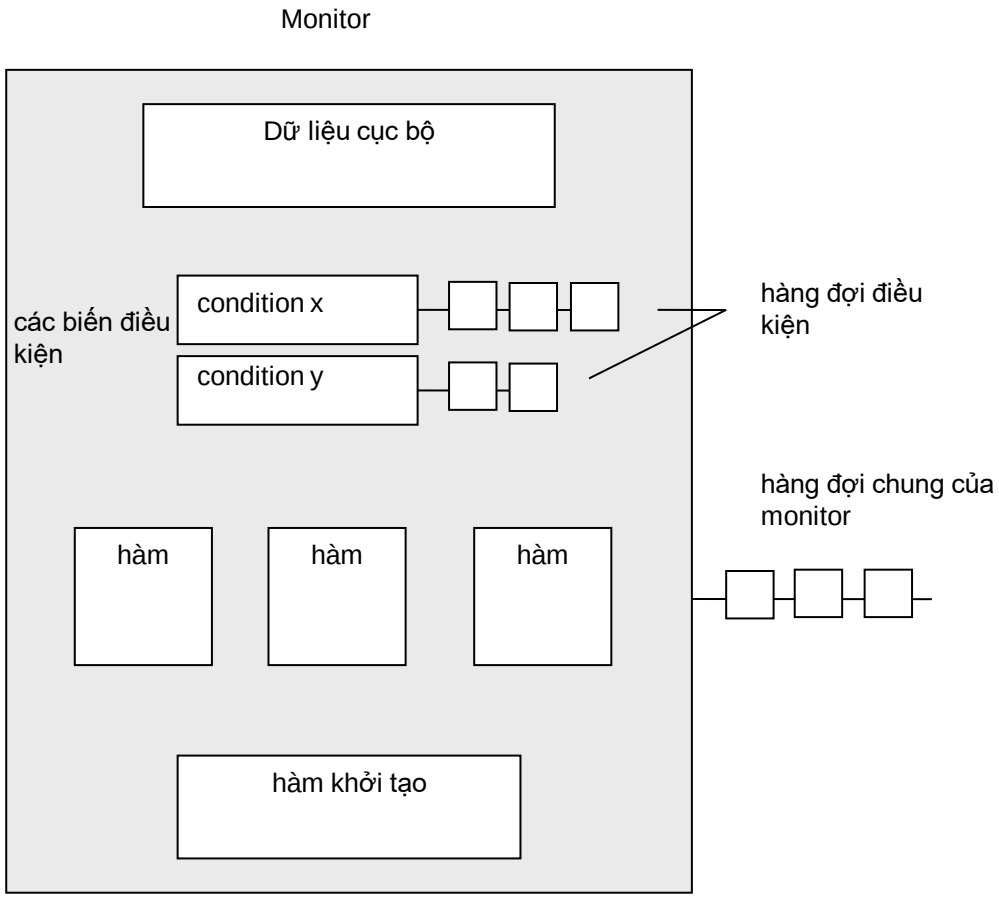
7. Monitor (tt)

- Tiến trình đang thực hiện trong monitor và bị dừng lại để đợi sự kiện => trả lại monitor để tiến trình khác dùng
- Tiến trình chờ đợi sẽ được khôi phục lại từ điểm dừng sau khi điều kiện đang chờ đợi được thỏa mãn
- => Sử dụng các biến điều kiện
- Được khai báo và sử dụng trong monitor với 2 thao tác: cwait() và csignal()

7. Monitor (tt)

- x.cwait():
  - Tiến trình đang ở trong monitor và gọi cwait bị phong tỏa cho tới khi điều kiện x xẩy ra
  - Tiến trình bị xếp vào hàng đợi của biến điều kiện x
  - Monitor được giải phóng và một tiến trình khác sẽ được vào
- x.csignal():
  - Tiến trình gọi csignal để thông báo điều kiện x đã thỏa mãn
  - Nếu có tiến trình đang bị phong tỏa và nằm trong hàng đợi của x do gọi x.cwait() trước đó sẽ được giải phóng
  - Nếu không có tiến trình bị phong tỏa thì thao tác csignal sẽ không có tác dụng gì cả

7. Monitor (tt)


Monitor
Dữ liệu cục bộ
các biến điều kiện
condition x
condition y
hàng đợi điều kiện
hàng đợi chung của monitor
hàm
hàm
hàm
hàm khởi tạo

7. Monitor (tt)

```
monitor BoundedBuffer
        product buffer[N];    //bộ đệm chứa N sản phẩm kiểu product
        int count;            //số lượng sản phẩm hiện thời trong bộ đệm
        condition notFull, notEmpty;    //các biến điều kiện
public:
        boundedbuffer( ) {  //khởi tạo
              count = 0;
        }
        void append (product x) {
              if (count == N)
                        notFull.cwait ( );  //dừng và chờ đến khi buffer có chỗ
              <Thêm một sản phẩm vào buffer>
              count++;
              notEmpty.csignal  ();
}
```

7. Monitor (tt)

```
        product take ( ) {
            if (count == 0)
                    notEmptry.cwait ();  //chờ đến khi buffer không rỗng
            <Lấy một sản phẩm x từ buffer>
            count --;
            notFull.csignal ( );
        }
}

void producer ( )  {   //tiến trình người sản xuất
        for (;;){
            <Sản xuất sản phẩm x>
            BoundedBuffer.append (x);
        }
}
```

7. Monitor (tt)

```
void consumer ( )  {   //tiến trình người tiêu dùng
        for (;;){
              product x = BoundedBuffer.take ();
              <Tiêu dùng x>
        }
}
void main() {
        Thực hiện song song producer và consumer.
}
```

V. BẾ TẮC (DEADLOCK)

1. Định nghĩa

- Tình trạng một nhóm tiến trình có cạnh tranh về tài nguyên hay có hợp tác phải dừng vô hạn
- Do tiến trình phải chờ đợi một sự kiện chỉ có thể sinh ra bởi tiến trình khác cũng đang trong trạng thái chờ đợi

- Đồng thời xảy ra 4 điều kiện:
  1. Loại trừ tương hỗ: có tài nguyên nguy hiểm, tại 1 thời điểm duy nhất 1 tiến trình sử dụng
  2. Giữ và chờ: tiến trình giữ tài nguyên trong khi chờ đợi
  3. Không có phân phối lại (no preemption): tài nguyên do tiến trình giữ không thể phân phối lại cho tiến trình khác trừ khi tiến trình đang giữ tự nguyện giải phóng tài nguyên
  4. Chờ đợi vòng tròn: tồn tại nhóm tiến trình P1, P2, …, Pn sao cho P1 chờ đợi tài nguyên do P2 đang giữ, P2 chờ tài nguyên do P3 đang giữ, …, Pn chờ tài nguyên do P1 đang giữ

- Giải quyết vấn đề bế tắc theo cách:
  - Ngăn ngừa (deadlock prevention): đảm bảo để một trong bốn điều kiện xẩy ra bế tắc không bao giờ thỏa mãn
  - Phòng tránh (deadlock avoidance): cho phép một số điều kiện bế tắc được thỏa mãn nhưng đảm bảo để không đạt tới điểm bế tắc
  - Phát hiện và giải quyết (deadlock detection): cho phép bế tắc xẩy ra, phát hiện bế tắc và khôi phục hệ thống về tình trạng không bế tắc

3. Ngăn ngừa bế tắc

- Đảm bảo ít nhất 1 trong 4 điều kiện không xảy ra
- Loại trừ tương hỗ: không thể ngăn ngừa
- Giữ và chờ:
  - Cách 1:
    - Yêu cầu tiến trình phải nhận đủ toàn bộ tài nguyên cần thiết trước khi thực hiện tiếp
    - Nếu không nhận đủ, tiến trình bị phong tỏa để chờ cho đến khi có thể nhận đủ tài nguyên
  - Cách 2:
    - Tiến trình chỉ được yêu cầu tài nguyên nếu không giữ tài nguyên khác
    - Trước khi yêu cầu thêm tài nguyên, tiến trình phải giải phóng tài nguyên đã được cấp và yêu cầu lại (nếu cần) cùng với tài nguyên mới

- Không có phân phối lại:
  - Cách 1:
    - Khi một tiến trình yêu cầu tài nguyên nhưng không được do đã bị cấp phát, HDH sẽ thu hồi lại toàn bộ tài nguyên nó đang giữ
    - Tiến trình chỉ có thể thực hiện tiếp sau khi lấy được tài nguyên cũ cùng với tài nguyên mới yêu cầu
  - Cách 2:
    - Khi tiến trình yêu cầu tài nguyên, nếu còn trống, sẽ được cấp phát ngay
    - Nếu tài nguyên do tiến trình khác giữ mà tiến trình này đang chờ cấp thêm tài nguyên thì thu hồi lại để cấp cho tiến trình yêu cầu
    - Nếu hai điều kiện trên đều không thỏa thì tiến trình yêu cầu tài nguyên phải chờ

- Chờ đợi vòng tròn:
  - Xác định thứ tự cho các dạng tài nguyên và chỉ cho phép tiến trình yêu cầu tài nguyên sao cho tài nguyên mà tiến trình yêu cầu sau có thứ tự lớn hơn tài nguyên mà nó yêu cầu trước
  - Giả sử trong hệ thống có n dạng tài nguyên ký hiệu R1, R2, …, Rn
  - Giả sử những dạng tài nguyên này được sắp xếp theo thứ tự tăng dần của chỉ số
  - Nếu tiến trình đã yêu cầu một số tài nguyên dạng Ri thì sau đó tiến trình chỉ được phép yêu cầu tài nguyên dạng Rj nếu j > i
  - Nếu tiến trình cần nhiều tài nguyên cùng dạng thì tiến trình phải yêu cầu tất cả tài nguyên dạng đó cùng một lúc

4. Phòng tránh bế tắc

- Ngăn ngừa bế tắc:
  - Sử dụng quy tắc hay ràng buộc khi cấp phát tài nguyên để ngăn ngừa điều kiện xảy ra bế tắc
  - Sử dụng tài nguyên kém hiệu quả, giảm hiệu năng của tiến trình
- Phòng tránh bế tắc:
  - Cho phép 3 điều kiện đầu xảy ra và chỉ đảm bảo sao cho trạng thái bế tắc không bao giờ đạt tới
  - Mỗi yêu cầu cấp tài nguyên của tiến trình sẽ được xem xét và quyết định tùy theo tình hình cụ thể
  - HDH yêu cầu tiến trình cung cấp thông tin về việc sử dụng tài nguyên (số lượng tối đa tài nguyên tiến trình cần sử dụng)

- Khi tiến trình muốn khởi tạo, thông báo dạng tài nguyên và số lượng tài nguyên tối đa cho mỗi dạng sẽ yêu cầu
- Nếu số lượng yêu cầu không vượt quá khả năng hệ thống, tiến trình sẽ được khởi tạo
- *Trạng thái* được xác định bởi tình trạng sử dụng tài nguyên hiện thời trong hệ thống:
  - Số lượng tối đa tài nguyên mà tiến trình yêu cầu:
    - Dưới dạng ma trận M[n][m]: n là số lượng tiến trình, m: số tài nguyên
    - M[i][j]: số lượng tài nguyên tối đa dạng j mà tiến trình i yêu cầu

- Số lượng tài nguyên còn lại:
  - Dưới dạng vectơ A[m]
  - A[j] là số lượng tài nguyên dạng j còn lại và có thể cấp phát
- Lượng tài nguyên đã cấp cho mỗi tiến trình:
  - Dưới dạng ma trận D[n][m]
  - D[i][j] là lượng tài nguyên dạng j đã cấp cho tiến trình i
- Lượng tài nguyên còn cần cấp
  - Dưới dạng ma trận C[n][m]
  - C[i][j]=M[i][j]-D[i][j] là lượng tài nguyên dạng j mà tiến trình i còn cần cấp

- *Trạng thái an toàn*: trạng thái mà từ đó có ít nhất một phương án cấp phát sao cho bế tắc không xẩy ra
- Cách phòng tránh bế tắc:
  - Khi tiến trình có yêu cầu cấp tài nguyên, hệ thống giả sử tài nguyên được cấp
  - Cập nhật lại trạng thái & xác định xem trạng thái đó có an toàn?
    - Nếu an toàn, tài nguyên sẽ được cấp thật
    - Ngược lại, tiến trình bị phong tỏa &chờ tới khi có thể cấp phát an toàn

4. Phòng tránh bế tắc – thuật toán người cho vay

- Hệ thống có 3 dạng tài nguyên X, Y, Z với số lượng ban đầu X=10, Y=5, Z=7
- 4 tiến trình P1, P2, P3, P4 có yêu cầu tài nguyên tối đa cho trong bảng
- Xét trạng thái sau của hệ thống:
  - Là trạng thái an toàn
  - Có thể tìm ra cách cấp phát không dẫn đến bế tắc, VD: P2, P4, P3, P1

	X	Y	Z
P1	7	5	3
P2	3	2	2
P3	9	0	2
P4	2	2	2

Yêu cầu tối đa

	X	Y	Z
P1	0	1	0
P2	2	0	0
P3	3	0	2
P4	2	1	1

Đã cấp

X	Y	Z
3	3	4

Còn lại

	X	Y	Z
P1	7	4	3
P2	1	2	2
P3	6	0	0
P4	0	1	1

Còn cần cấp

  - P1 yêu cầu cấp phát 3 tài nguyên dạng Y, tức là yêu cầu = (0,3,0). Nếu yêu cầu thỏa mãn, hệ thống chuyển sang trạng thái:
  - Trạng thái không an toàn
  - => yêu cầu (0,3,0) bị từ chối

X	Y	Z
3	0	4

Còn lại

	X	Y	Z
P1	7	1	3
P2	1	2	2
P3	6	0	0
P4	0	1	1

Còn cần cấp

- Thuật toán xác định trạng thái an toàn:

1. Khai báo mảng W kích thước m và mảng F kích thước n. (F[i] chứa tình trạng tiến trình thứ i đã kết thúc hay chưa, W chứa lượng tài nguyên còn lại)

   Khởi tạo W=A và F[i]=false (i=0,…,n-1)
2. Tìm i sao cho:

   F[i] = false và C[i][j] ≤ W[j] với mọi j=0,…,m-1

   Nếu không có i như vậy thì chuyển sang bước 4
3. a) W = W + D[i]

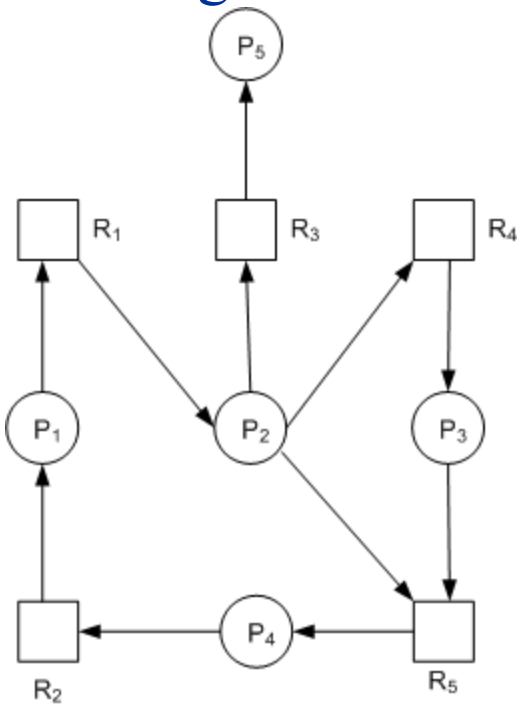
   b) F[i] = true

   c) Quay lại bước 2
4. If F[i] = true với mọi i =0,…,n-1 thì trạng thái an toàn

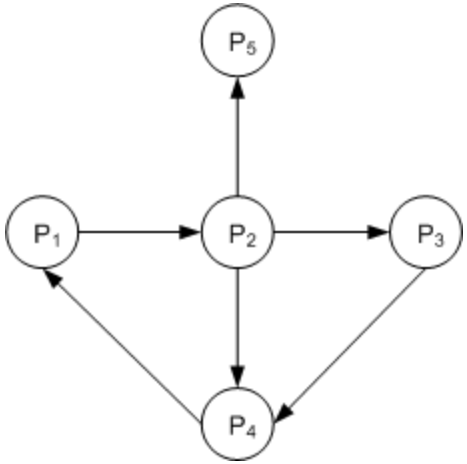
   Else trạng thái không an toàn

- Hệ thống không thực hiện biện pháp ngăn ngừa/phòng tránh => bế tắc có thể xảy ra
- Hệ thống định kỳ kiểm tra để phát hiện có tình trạng bế tắc hay không?
- Nếu có, hệ thống sẽ xử lý để khôi phục lại trạng thái không có bế tắc

- Phát hiện bế tắc:
  - Trường hợp mỗi dạng tài nguyên chỉ có một tài nguyên duy nhất =>sử dụng đồ thị biểu diễn quan hệ chờ đợi lần nhau giữa tiến trình
  - Xây dựng đồ thị cấp phát tài nguyên:
    - Các nút là tiến trình và tài nguyên
    - Tài nguyên được nối với tiến trình bằng cung có hướng nếu tài nguyên được cấp cho tiến `trình đó
    - Tiến trình được nối với tài nguyên bằng cung có hướng nếu tiến trình đang được cấp tài nguyên đó



- Phát hiện bế tắc:
  - Đồ thị chờ đợi:
    - Được xây dựng từ đồ thị cấp phát tài nguyên bằng cách bỏ đi các nút tương ứng với tài nguyên và nhập các cung đi qua nút bị bỏ
    - Cho phép phát hiện tình trạng tiến trình chờ đợi vòng tròn là điều kiện đủ để sinh ra bế tắc
    - Sử dụng thuật toán phát hiện chu trình trên đồ thị có hướng để phát hiện bế tắc trên đồ thị chờ đợi



- Thời điểm phát hiện bế tắc:
  - Bế tắc chỉ có thể xuất hiện sau khi một tiến trình nào đó yêu cầu tài nguyên và không được thỏa mãn
  - => Chạy thuật toán phát hiện bế tắc mỗi khi có yêu cầu cấp phát tài nguyên không được thỏa mãn
  - => Cho phép phát hiện bế tắc ngay khi vừa xẩy ra
  - Chạy thường xuyên làm giảm hiệu năng hệ thống
  - => Giảm tần suất chạy thuật toán phát hiện bế tắc:
    - Sau từng chu kỳ từ vài chục phút tới vài giờ
    - Khi có một số dấu hiệu như hiệu suất sử dụng CPU giảm xuống dưới một ngưỡng nào đó

- Xử lý khi bị bế tắc:
  - Kết thúc tất cả tiến trình đang bị bế tắc
  - Kết thúc lần lượt từng tiến trình đang bị bế tắc đến khi hết bế tắc:
    - HDH phải chạy lại thuật toán phát hiện bế tắc sau khi kết thúc 1 tiến trình
    - HDH có thể chọn thứ tự kết thúc tiến trình dựa trên tiêu chí nào đó
  - Khôi phục tiến trình về thời điểm trước khi bị bế tắc sau đó cho các tiến trình thực hiện lại từ điểm này:
    - Đòi hỏi HDH lưu trữ trạng thái để có thể thực hiện quay lui và khôi phục về các điểm kiểm tra trước đó
    - Khi chạy lại, các tiến trình có thể lại rơi vào bế tắc tiếp
  - Lần lượt thu hồi lại tài nguyên từ các tiến trình bế tắc cho tới khi hết bế tắc

6. Ngăn ngừa bế tắc cho bài toán triết gia ăn cơm

- Đặt hai thao tác lấy đũa của mỗi triết gia vào đoạn nguy hiểm để đảm bảo triết gia lấy được hai đũa cùng một lúc
- Quy ước bất đối xứng về thứ tự lấy đũa: ví dụ người có số thứ tự chẵn lấy đũa trái trước đũa phải, người có số thứ tự lẻ lấy đũa phải trước đũa trái
- Tại mỗi thời điểm chỉ cho tối đa bốn người ngồi vào bàn:
  - Sử dụng thêm một cờ hiệu *table* có giá trị khởi tạo bằng 4
  - Triết gia phải gọi thao tác *wait(table)* trước khi ngồi vào bàn và lấy đũa

6. Ngăn ngừa bế tắc cho bài toán triết gia ăn cơm

```
semaphore chopstick[5] = {1,1,1,1,1,1};
semaphore table = 4;
void Philosopher(int i){        //tiến trình P(i)
    for(;;){    //lặp vô hạn
        wait(table);
        wait(chopstick[i]);                 //lấy đũa bên trái
        wait(chopstick[(i+1)%5]);           //lấy đũa bên phải
        <Ăn cơm>
        signal(chopstick[(i+1)%5]);
        signal(chopstick[i]);
        signal(table);
        <Suy nghĩ>
    }
}
void main(){
            StartProcess(Philosopher(1));
            ...
            StartProcess(Philosopher (5));
}
```

KIỂM TRA

Bài 1: Cho các tiến trình với thời điểm xuất hiện, thời gian (chu kỳ) CPU tiếp theo và số ưu tiên như trong bảng sau (số ưu tiên nhỏ ứng với độ ưu tiên cao). Vẽ biểu đồ thể hiện thứ tự và thời gian cấp phát CPU cho các tiến trình sử dụng thuật toán

a. Điều độ theo mức ưu tiên không có phân phối lại

b. Điều độ ưu tiên tiến trình ngắn nhất

c. Điều độ ưu tiên thời gian còn lại ngắn nhất

Tính thời gian chờ đợi trung bình cho từng trường hợp.

Tiến trình	Thời điểm xuất hiện	Thời gian (độ dài)	Số ưu tiên
P1	0	10	1
P2	2	8	3
P3	4	5	2
P4	5	3	2

KIỂM TRA

Bài 2: Bộ nhớ vật lý có 5 khung. Thứ tự truy cập các trang là 1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6. Vẽ sơ đồ cấp phát bộ nhớ và Có bao nhiêu sự kiện thiếu trang xảy ra nếu sử dụng:

- LRU
- Đồng hồ

Bài 3: Kích thước khung bộ nhớ là 2048 bytes. Hãy chuyển địa chỉ logic 3500, 5060 sang địa chỉ vật lý biết rằng bảng trang như sau:

Số trang	Số khung
0	3
1	5
2	
3	30

KIỂM TRA

Bài 2:

a. Sử dụng lệnh Test_and_Set để thực hiện loại trừ tương hỗ cho bài toán triết gia ăn cơm

b. Phân tích rõ ràng giải pháp sử dụng lệnh Test_and_Set ở trên có thể gây bế tắc hoặc đói không

Bài 3: Xét trạng thái cấp phát tài nguyên của hệ thống như sau:

	X	Y	Z
P1	0	1	0
P2	2	0	0
P3	3	0	2
P4	2	1	1

Đã cấp

X	Y	Z
3	3	4

Còn lại

	X	Y	Z
P1	7	4	3
P2	1	2	2
P3	6	0	0
P4	0	1	1

Còn cần cấp

P2 yêu cầu cấp phát 1 tài nguyên Y, 2 tài nguyên Z. Sử dụng thuật toán người cho vay để xác định xem yêu cầu của P2 có được đáp ứng hay không?