Nguyên lý hệ điều hành

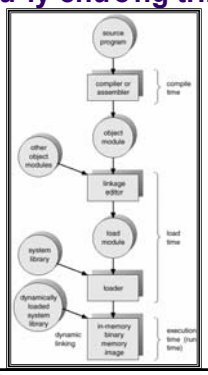
Nguyễn Hải Châu
Khoa Công nghệ thông tin
Trường Đại học Công nghệ

Quản lý bộ nhớ

Giới thiệu

- Chương trình được HĐH đưa vào bộ nhớ, sau đó tạo tiến trình để thực hiện
- *Input queue* – Là hàng chờ các tiến trình trên đĩa đang chờ được đưa vào bộ nhớ để thực hiện
- Các chương trình của NSD phải qua một số bước chuẩn bị trước khi được thực hiện

Các bước xử lý chương trình NSD



Chuyển đổi địa chỉ

Có 3 cách chuyển đổi địa chỉ lệnh và dữ liệu của chương trình vào bộ nhớ:

- **Khi dịch chương trình (compile-time)**: Sinh mã có địa chỉ cố định; phải dịch lại nếu cần thay đổi địa chỉ.
- **Khi nạp chương trình (load-time)**: Phải sinh mã có thể định vị lại nếu như địa chỉ bộ nhớ không được biết ở thời điểm dịch chương trình
- **Khi thực hiện chương trình (execution-time)**: Ánh xạ địa chỉ khi chương trình được thực hiện nếu như tiến trình có thể chuyển giữa các segment bộ nhớ. Cần có hỗ trợ từ phần cứng (ví dụ thanh ghi *base* và *limit*)

Không gian địa chỉ logic (ảo) và địa chỉ vật lý (địa chỉ thật)

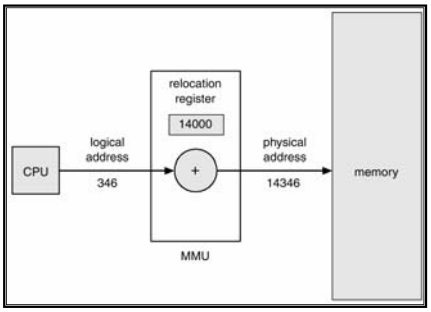
- Để quản lý bộ nhớ một cách hoàn chỉnh, cần có hai cách nhìn địa chỉ khác nhau:
  - *Địa chỉ logic* (*Logical address*) – sinh bởi CPU; còn gọi là địa chỉ ảo (*virtual address*).
  - *Địa chỉ vật lý* (*Physical address*); còn gọi là địa chỉ thật – sinh bởi đơn vị quản lý bộ nhớ
- Địa chỉ thật và ảo giống nhau trong lược đồ ánh xạ địa chỉ "compile-time" và "load-time" và khác nhau trong "execution-time".

Đơn vị quản lý bộ nhớ (MMU)

- Là thiết bị phần cứng dùng để ánh xạ địa chỉ ảo sang địa chỉ vật lý
- Trong MMU, có thanh ghi relocation (định vị lại) dùng để tính toán địa chỉ thực (vật lý) từ địa ảo của một tiến trình của NSD
- Chương trình của NSD làm việc trên địa chỉ ảo và không bao giờ biết địa chỉ vật lý

7

Sử dụng thanh ghi relocation



8

Nạp chương trình động (Dynamic loading)

- Các hàm, thủ tục không được nạp cho đến khi được sử dụng (được gọi đến)
- Cách nạp động này sử dụng bộ nhớ hiệu quả hơn: Các hàm, thủ tục không dùng đến không bao giờ được nạp vào bộ nhớ
- Hữu ích khi có một đoạn mã lớn được sử dụng với tần suất thấp
- Không cần có các đặc điểm đặc biệt từ hệ điều hành về phần cứng/phần mềm

9

Liên kết động (dynamic linking) và thư viện chung (shared library)

- Liên kết chương trình được thực hiện khi chương trình được thực hiện.
- Một đoạn mã ngắn (stub) được dùng để định vị các hàm tương ứng đã được nạp sẵn trong bộ nhớ
- Stub được thay thế bằng địa chỉ của hàm/thủ tục cần thiết, sau đó thực hiện hàm/thủ tục đó
- HĐH cần kiểm tra các hàm/thủ tục đã được nạp chưa
- Liên kết động rất có lợi khi xây dựng các thư viện chung, khi sửa lỗi (các miếng vá – patch)

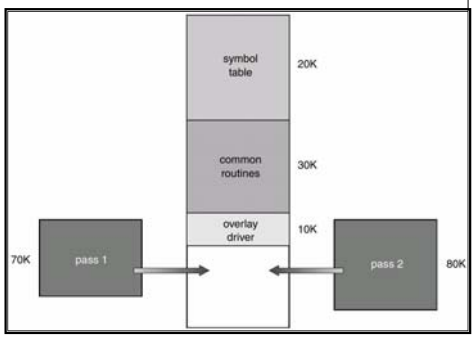
10

Overlays

- Chỉ lưu trong bộ nhớ các phần lệnh và dữ liệu phải sử dụng trong suốt quá trình thực hiện
- Sử dụng khi tiến trình có yêu cầu bộ nhớ lớn hơn dung lượng được cấp phát.
- Cài đặt bởi người sử dụng, lập trình overlays rất phức tạp

11

Ví dụ về overlays



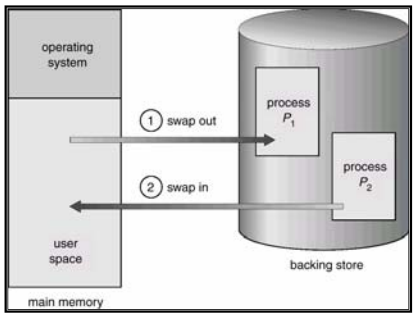
12

Swapping

- *Swapping*: Đưa một tiến trình ra *backing store* để lưu trữ tạm thời, sau đó đưa trở lại bộ nhớ trong để thực hiện.
  - *Backing store* – Vùng đĩa có tốc độ truy cập cao, đủ lớn để chứa được nhiều tiến trình của NSD, có thể truy cập trực tiếp
- *Roll out, roll in* – Phương án swap dành cho lập lịch có ưu tiên: Tiến trình ưu tiến thấp: *roll out*, ưu tiên cao: *roll in* để tiếp tục thực hiện
- Thời gian swap tỷ lệ thuận với dung lượng bộ nhớ được swap vào/ra
- UNIX, Linux, and Windows sử dụng swapping

13

Minh họa swapping



14

Cấp phát liên tục (Contiguous allocation)

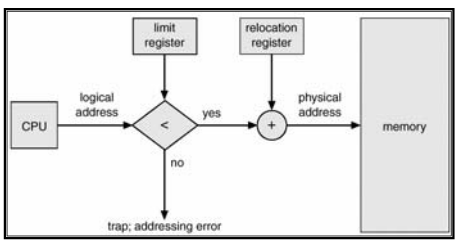
15

Cấp phát bộ nhớ liên tục

- Bộ nhớ trong thường được chia thành 2 phần:
  - Phần dành cho hệ điều hành (resident) thường dùng phần thấp của bộ nhớ với các ngắt
  - NSD dùng phần cao của bộ nhớ. Mỗi tiến trình được cấp phát một *vùng liên tục của bộ nhớ*
- Thanh ghi *relocation* dùng để bảo vệ các tiến trình của NSD và để tránh thay đổi mã và dữ liệu của HĐH
- Thanh ghi *relocation* chứa giá trị nhỏ nhất của địa chỉ vật lý, thanh ghi *limit* chứa độ lớn của miền địa chỉ ảo (*địa chỉ ảo* < *limit*)

16

Minh họa thanh ghi *relocation*, *limit*



17

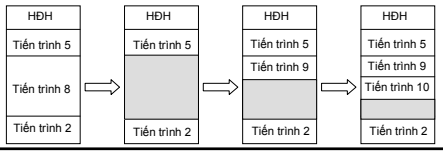
Cấp phát liên tục (tiếp): MFT

- Bộ nhớ được chia thành các khối với cỡ cố định, mỗi tiến trình được cấp phát một khối
- Khi tiến trình kết thúc, khối bộ nhớ đã cấp phát cho tiến trình được giải phóng để cấp phát cho tiến trình khác
- Mức độ đa chương trình bị hạn chế bởi các khối
- Cỡ của tiến trình bị hạn chế bởi cỡ của khối
- Các HĐH/máy tính sử dụng MFT: IBM/360

18

Cấp phát liên tục (tiếp): MVT

- Cấp phát MVT
  - *Hole* – khối bộ nhớ rỗi; các khối rỗi với kích cỡ khác nhau rải rác trong bộ nhớ
  - Một tiến trình sẽ được cấp phát một khối bộ nhớ đủ lớn để thực hiện
  - HĐH có thông tin về các khối đã cấp phát và khối rỗi



19

Các chiến lược cấp phát

- **First-fit**: Cấp phát khối nhớ đầu tiên thỏa mãn điều kiện.
- **Best-fit**: Cấp phát khối nhớ bé nhất thỏa mãn điều kiện: Phải duyệt toàn bộ danh sách khối nhớ
- **Worst-fit**: Cấp phát khối nhớ lớn nhất thỏa mãn điều kiện: Phải duyệt toàn bộ danh sách khối nhớ
- First-fit và best-fit tốt hơn worst-fit theo nghĩa tốc độ và tận dụng bộ nhớ

20

Vấn đề phân mảnh

- **External Fragmentation (*Phân mảnh ngoài*)**: Tổng dung lượng đáp ứng được nhu cầu cấp phát nhưng các khối không liên tục
- **Internal Fragmentation (*Phân mảnh trong*)** – Dung lượng bộ nhớ đã cấp phát cho tiến trình không được sử dụng hết
- Giảm phân mảnh ngoài: Compaction
  - Xáo trộn các khối để các khối nhớ rỗi nằm liên tục
  - Compaction chỉ thực hiện được khi relocation là động, và được thực hiện ở execution-time
- Ví dụ: Tiện ích Defragmentation của Windows

21

Phân trang (Paging)

22

Phân trang (paging)

- Phân trang là chiến lược cấp phát bộ nhớ cho phép không gian địa chỉ logic của một tiến trình có thể không liên tục; tiến trình được cấp phát bộ nhớ vật lý khi có bộ nhớ rỗi
- Bộ nhớ vật lý được chia thành các frame cỡ cố định, nhỏ (là lũy thừa của 2, ví dụ 512, 1024, 8192)
- Chia bộ nhớ ảo thành các khối *cùng cỡ* gọi là trang (page)
- HĐH có danh sách các frame rỗi
- Để thực hiện một chương trình cỡ *n* trang, cần tìm *n* frame rỗi để nạp chương trình
- Có một bảng trang để ánh xạ trang→frame
- Bảng trang: chung trong HĐH, mỗi tiến trình có một copy

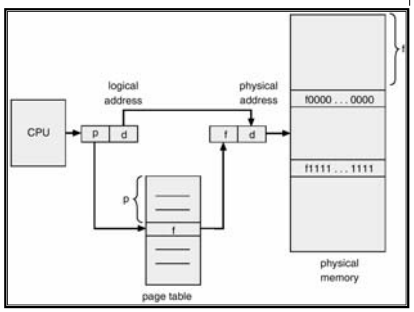
23

Cách đánh địa chỉ theo trang

- Địa chỉ được đánh một cách phân cấp:
  - *Số hiệu trang* (*Page number* - *p*) – Được sử dụng làm chỉ số đến phần tử trong bảng trang chứa địa chỉ cơ sở của các frame trong bộ nhớ vật lý
  - *Offset trang* (*Page offset* - *d*) – Địa chỉ tương đối trong trang
- Địa chỉ ảo có *m* bit, sử dụng *m-n* bit cao làm số hiệu trang và *n* bit thấp làm offset
- Không có *phân mảnh ngoài*, có *phân mảnh trong*:
  - Giảm cỡ trang→Giảm phân mảnh trong→Giảm hiệu năng
  - Tăng cỡ trang→Tăng hiệu suất→Tăng phân mảnh trong

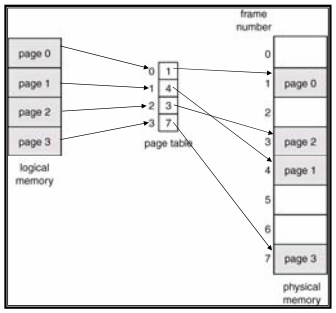
24

Chuyển đổi địa chỉ



25

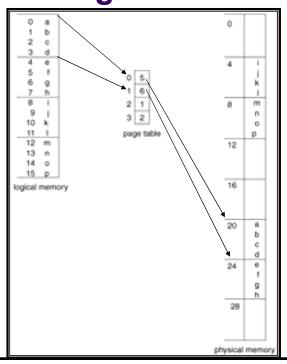
Ví dụ phân trang 1



26

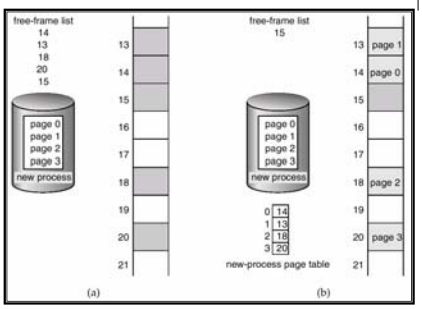
Ví dụ phân trang 2

Cỡ của một trang là 4 bytes



27

Bảng frame rỗi



Trước cấp phát Sau cấp phát

28

Cài đặt bảng trang

- Bảng trang được lưu ở bộ nhớ trong
- *Thanh ghi cơ sở bảng trang* (*page-table base register*) (PTBR) trỏ đến bảng trang
- *Thanh ghi độ dài bảng trang* (*page-table length register*) (PTLR) lưu cỡ bảng trang
- Sử dụng bảng trang, mọi thao tác truy cập dữ liệu/lệnh cần tới 2 lần truy cập bộ nhớ (1 cho bảng trang, 1 cho dữ liệu/lệnh)

29

Cài đặt bảng trang (tiếp)

- Truy cập bộ nhớ hai lần: Giảm tốc độ
- Giải quyết vấn đề 2 lần truy cập bộ nhớ: Sử dụng phần cứng cache có tốc độ truy cập cao gọi là bộ nhớ kết hợp (*associative memory*) hoặc vùng đệm hỗ trợ chuyển đổi (*translation look-aside buffers -TLB*)
- Mỗi phần tử trong TLB có hai phần: khóa và giá trị
- Số lượng các phần tử của TLB thường từ 64 đến 1024

30

Bộ nhớ kết hợp

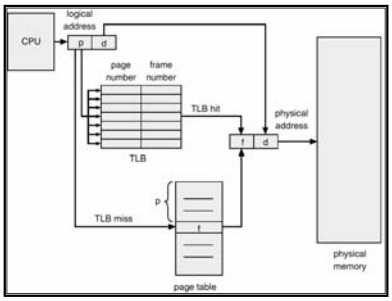
- Bộ nhớ kết hợp

Page #	Frame #

- Chuyển đổi địa chỉ (A´, A´´)
  if A´ nằm trong thanh ghi kết hợp, lấy *frame#*.
  else lấy *frame#* từ bảng trang trong bộ nhớ

31

Phân trang phần cứng với TLB



32

Thời gian truy cập hiệu quả

- Thời gian tìm kiếm ở thanh ghi kết hợp = ε (đơn vị thời gian)
- Thời gian truy cập bộ nhớ là *n* đơn vị thời gian
- Hit ratio: Số phần trăm (%) địa chỉ trang được tìm thấy ở các thanh ghi kết hợp/TLB
- Hit ratio = α
- Thời gian truy cập hiệu quả (EAT):
  EAT = $(n + \varepsilon)\,\alpha + (2n + \varepsilon)(1 - \alpha) = 2n + \varepsilon - \alpha n$

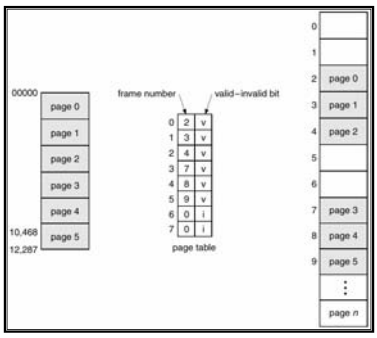
33

Bảo vệ bộ nhớ

- Bộ nhớ được bảo vệ nhờ kết hợp bit bảo vệ trong mỗi phần tử ở bảng trang
- Bit hợp lệ-không hợp lệ (*valid-invalid*) kết nối với mỗi phần tử trong bảng trang:
  - "valid" chỉ ra rằng trang thuộc không gian địa chỉ logic của tiến trình → trang hợp lệ
  - "invalid" chỉ ra rằng trang không thuộc không gian địa chỉ logic của tiến trình

34

Ví dụ bit valid (v)/invalid (i) trong bảng trang



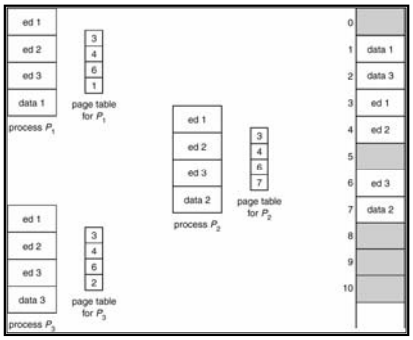
35

Các trang chung

- Mã dùng chung
  - Nhiều tiến trình (soạn thảo, compiler...) có thể dùng chung các đoạn mã reentrant (đoạn mã không tự thay đổi chính nó)
  - Đoạn mã chung phải xuất hiện ở cùng một vị trí địa chỉ trong không gian địa chỉ logic/ảo của tất cả các tiến trình
- Mã lệnh và dữ liệu riêng
  - Mỗi tiến trình có một bản riêng chứa lệnh và dữ liệu
  - Các trang chứa lệnh và dữ liệu riêng có thể ở bất kỳ vị trí nào trong không gian địa chỉ của tiến trình

36

Ví dụ các trang chung



Cấu trúc bảng trang

Bảng trang phân cấp

Bảng trang băm

Bảng trang ngược

Bảng trang phân cấp

- Bộ nhớ máy tính lớn (2^{32}-2^{64} bytes): Nếu dùng bảng trang một cấp thì bảng trang có cỡ rất lớn: Tốn bộ nhớ, tìm kiếm chậm
- Không gian địa chỉ logic được quản lý bởi nhiều bảng trang ở nhiều cấp
- Một kỹ thuật đơn giản nhất là bảng trang hai cấp. Có thể có bảng trang hai, ba, bốn cấp

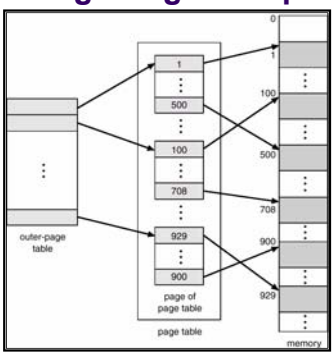
Ví dụ bảng trang hai cấp

- Địa chỉ logic (trên máy 32-bit, trang cỡ 4K=2^{12}) được chia thành:
  - Địa chỉ trang: 20 bits.
  - Địa chỉ offset: 12 bits.
- Bảng trang 2 cấp (địa chỉ 20 bit) được chia thành:
  - 10-bit địa chỉ trang cấp 1
  - 10-bit địa chỉ trang cấp 2
- Khi đó địa chỉ logic có dạng:

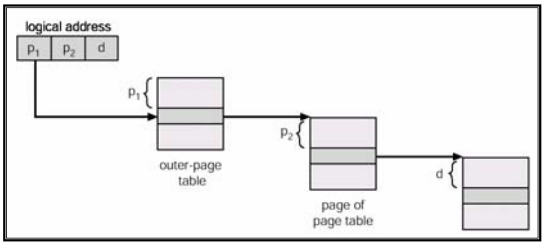
Địa chỉ trang		Offset
p_1	p_2	d
10	10	12

trong đó p_1 là chỉ số đến bảng trang ngoài, p_2 là chỉ số đến trang (thực sự) ở bảng trang ngoài

Sơ đồ bảng trang hai cấp



Tính địa chỉ với bảng trang hai cấp

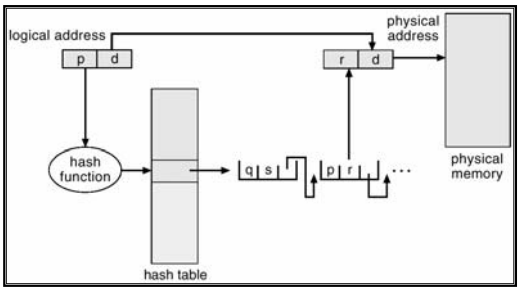


Bảng trang băm

- Thường sử dụng khi địa chỉ > 32 bit
- Số hiệu/địa chỉ trang được băm trong bảng trang. Bảng trang này chứa dãy các phần tử (các trang) được băm ở cùng một vị trí
- Số hiệu trang được so sánh trong dãy các trang được băm ở cùng một vị trí để từ đó tìm ra frame vật lý

43

Bảng trang băm



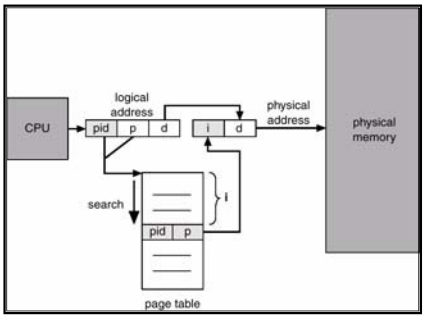
44

Bảng trang ngược

- Giải pháp giảm bộ nhớ lưu các bảng trang
- Mỗi phần tử trong bảng ứng với một frame
- Mỗi phần tử chứa địa chỉ ảo của trang và thông tin về tiến trình đang sử dụng trang đó
- Giảm dung lượng bộ nhớ cần để lưu các bảng trang, nhưng tăng thời gian cần để tìm trong bảng khi cần tham chiếu đến một trang
- Sử dụng bảng băm để hạn chế số lần tìm kiếm trong các phần tử bảng trang

45

Kiến trúc bảng trang ngược



46

Phân đoạn (Segmentation)

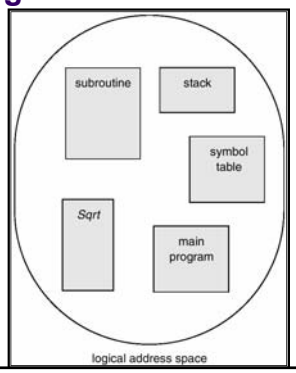
47

Phân đoạn

- Phương thức quản lý bộ nhớ cho phép NSD "nhìn" bộ nhớ một cách dễ dàng dưới góc độ lập trình
- Một chương trình gồm nhiều phân đoạn, mỗi phân đoạn thể hiện dưới góc độ lập trình ở dạng:
  - main program, // Chương trình chính
  - function, // Các hàm
  - method, // Các phương thức
  - object, // Các đối tượng, lớp
  - local/global variables, // Các biến
  - common block, // Các khối chung
  - stack, // Ngăn xếp
  - symbol table, arrays // Bảng ký hiệu, mảng

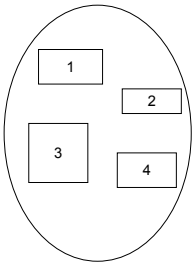
48

Chương trình nhìn từ NSD

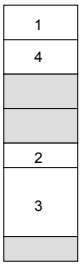


49

Phân đoạn: Cách nhìn logic



Không gian địa chỉ của NSD



Không gian bộ nhớ vật lý

50

Kiến trúc phân đoạn

- Địa chỉ ảo/logic là một bộ đôi:<segment, offset>
- Bảng phân đoạn (*segment table*) – ánh xạ địa chỉ vật lý 2 cấp; mỗi phần tử bảng có:
  - *base*: Địa chỉ vật lý bắt đầu của phân đoạn (segment)
  - *limit*: Độ dài của phân đoạn (segment).

51

Kiến trúc phân đoạn (tiếp)

- Thanh ghi cơ sở bảng phân đoạn (*Segment-table base register STBR*) trỏ đến base
- Thanh ghi độ dài bảng phân đoạn (*Segment-table length register - STLR*) chỉ ra số lượng phân đoạn được sử dụng trong tiến trình;
- Số hiệu phân đoạn *s* là hợp lệ nếu thỏa mãn điều kiện: *s* < STLR.

52

Kiến trúc phân đoạn (tiếp)

- Định vị lại (relocation)
  - Động
  - Sử dụng bảng phân đoạn
- Dùng chung (sharing)
  - Có các phân đoạn dùng chung
  - Sử dụng cùng một số hiệu phân đoạn (segment number)
- Cấp phát (allocation)
  - first fit/best fit
  - Phân mảnh ngoài

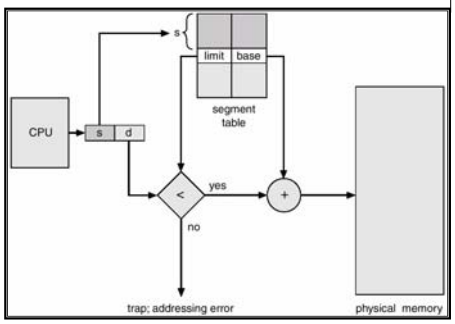
53

Kiến trúc phân đoạn (tiếp)

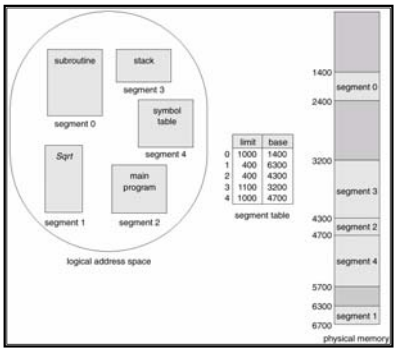
- Bảo vệ bộ nhớ:Mỗi phân đoạn có:
  - Bit kiểm tra = 0 ⇒ phân đoạn không hợp lệ
  - read/write/execute privileges
- Protection bits associated with segments; code sharing occurs at segment level.
- Do phân đoạn có cỡ biến đổi → Gặp vấn đề tương tự trong cấp phát bộ nhớ liên tục
- Kết hợp phân đoạn với phân trang để tăng hiệu quả sử dụng bộ nhớ, dễ cấp phát hơn (ví dụ: MULTICS, Intel 386)

54

Phần cứng phân đoạn



Ví dụ phân đoạn



Tóm tắt

- Địa chỉ logic (ảo)/Địa chỉ vật lý (thật)
- Các phương án ánh xạ địa chỉ của chương trình vào bộ nhớ
- Cấp phát bộ nhớ liên tục, phân mảnh, các chiến lược cấp phát first-fit, best-fit, worst-fit
- Phân trang
  - Trang, frame
  - Bảng trang, bảng trang phân cấp, bảng trang ngược
- Phân đoạn, bảng phân đoạn