



# HỆ ĐIỀU HÀNH

# Operating Systems

1

## NỘI DUNG

---

- Chương 1: Tổng quan
- Chương 2: Quản lý tiến trình
- Chương 3: Deadlock
- Chương 4: Quản lý bộ nhớ**
- Chương 5: Hệ thống file
- Chương 6: Quản lý nhập xuất

---

2

2

## Chương 4

# Quản lý bộ nhớ



ĐH KHTN TpHCM

3  
www.cunghodaptrinh.com

3

## NỘI DUNG

---

1. Vấn đề quản lý bộ nhớ
2. Mô hình quản lý bộ nhớ thực
3. Mô hình quản lý bộ nhớ ảo

ĐH KHTN TpHCM

4

4

## 4.1. Vấn đề quản lý bộ nhớ

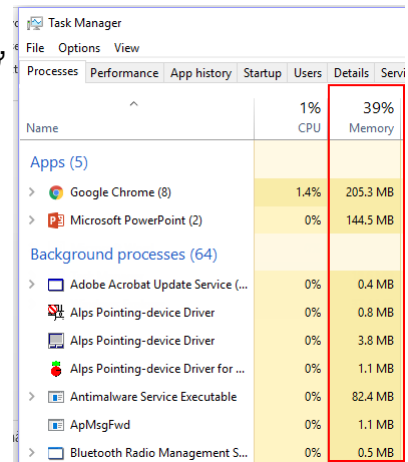
- Bộ nhớ chính là nơi mà CPU có thể truy cập trực tiếp
- Dữ liệu của chương trình phải được ghi vào bộ nhớ chính trước khi được xử lý
- Việc trao đổi thông tin giữa CPU với môi trường ngoài thông qua thao tác đọc/ghi dữ liệu vào một địa chỉ trong bộ nhớ
- Bộ nhớ chính được tổ chức như mảng một chiều các từ nhớ (word), mỗi từ nhớ có một địa chỉ

## Vấn đề quản lý bộ nhớ (tt)

- Quản lý bộ nhớ là công việc của hệ điều hành với sự hỗ trợ của phần cứng nhằm phân phối, sắp xếp các tiến trình trong bộ nhớ sao cho hiệu quả.
- Ba chức năng chính của HĐH đối với quản lý bộ nhớ:
  - Theo dõi những phần nào của bộ nhớ đang được sử dụng.
  - Quyết định những tiến trình nào sẽ được nạp vào bộ nhớ.
  - Định vị và tái định vị không gian bộ nhớ khi cần thiết.
- Mục tiêu quản lý bộ nhớ: nạp càng nhiều tiến trình vào bộ nhớ càng tốt (gia tăng mức độ đa chương)

## Vấn đề quản lý bộ nhớ (tt)

- Trong hầu hết các hệ thống:
  - Kernel sẽ chiếm một phần cố định của bộ nhớ
  - Còn lại phân phối cho các tiến trình



The screenshot shows the Windows Task Manager Performance tab. The 'Memory' section is highlighted with a red box, showing 39% usage. Below it, a table lists the memory usage for various applications and background processes.

Name	CPU	Memory
<b>Apps (5)</b>		
Google Chrome (8)	1.4%	205.3 MB
Microsoft PowerPoint (2)	0%	144.5 MB
<b>Background processes (64)</b>		
Adobe Acrobat Update Service (...)	0%	0.4 MB
Alps Pointing-device Driver	0%	0.8 MB
Alps Pointing-device Driver	0%	3.8 MB
Alps Pointing-device Driver for ...	0%	1.1 MB
Antimalware Service Executable	0%	82.4 MB
ApMsgFwd	0%	1.1 MB
Bluetooth Radio Management S...	0%	0.5 MB

7

7

## Vấn đề quản lý bộ nhớ (tt)

- Các yêu cầu đối với việc quản lý bộ nhớ
  - Cấp phát bộ nhớ cho các tiến trình
  - Tái định vị (relocation): khi swapping,...
  - Bảo vệ: phải kiểm tra truy xuất bộ nhớ có hợp lệ không
  - Chia sẻ: cho phép các tiến trình chia sẻ vùng nhớ chung
  - Kết gán địa chỉ nhớ luận lý của user vào địa chỉ thực
- Hàng đợi nhập hệ thống: các chương trình trên đĩa đang chờ được nạp vào bộ nhớ

8

8

## Vấn đề quản lý bộ nhớ (tt)

- **Địa chỉ vật lý** (physical address - địa chỉ thực) là một vị trí thực trong bộ nhớ chính.
- **Địa chỉ luận lý** (logical address): địa chỉ được CPU tạo ra, là một vị trí nhớ được diễn tả trong một chương trình.
  - Các trình biên dịch (compiler) tạo ra mã lệnh chương trình mà trong đó mọi tham chiếu bộ nhớ đều là địa chỉ luận lý
  - **Địa chỉ tuyệt đối** (absolute address): địa chỉ tương đương với địa chỉ thực.
  - **Địa chỉ tương đối** (relative address) (khả tái định vị - relocatable address) là một kiểu địa chỉ luận lý trong đó các địa chỉ được biểu diễn tương đối so với một vị trí xác định nào đó trong chương trình. (VD: 12 byte so với vị trí bắt đầu chương trình)

9

9

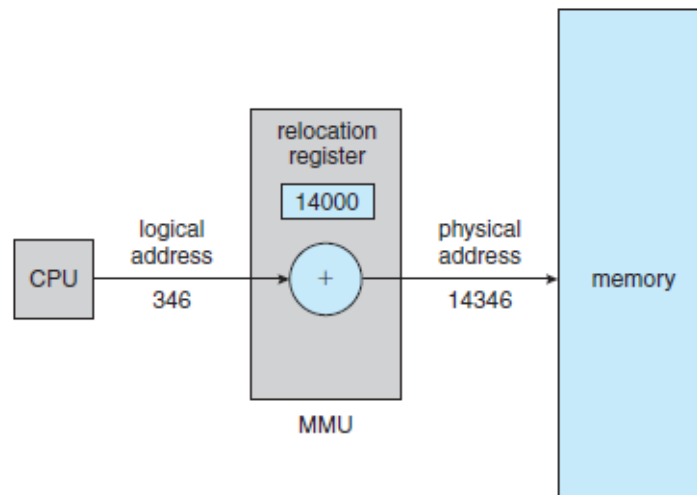
## Vấn đề quản lý bộ nhớ (tt)

- **Chuyển đổi địa chỉ:**
  - Khi một lệnh được thực thi, các tham chiếu đến địa chỉ luận lý phải được chuyển đổi thành địa chỉ thực.
  - Cần có sự hỗ trợ của phần cứng để chuyển đổi.
  - **MMU** (Memory Management Unit): cơ chế phần cứng được sử dụng để chuyển đổi địa chỉ luận lý thành địa chỉ vật lý tại thời điểm xử lý

10

10

## Vấn đề quản lý bộ nhớ (tt)



Dynamic relocation using a relocation register.

11

11

## Vấn đề quản lý bộ nhớ (tt)

- Thời điểm gán kết (chuyển đổi) địa chỉ:
  - Thời điểm biên dịch:
    - Nếu biết trước địa chỉ bộ nhớ của chương trình thì có thể kết gán địa chỉ tuyệt đối lúc biên dịch
    - Thay đổi vị trí nạp chương trình → biên dịch lại
  - Thời điểm nạp:
    - Tại thời điểm biên dịch, nếu chưa biết tiến trình sẽ nằm ở đâu trong bộ nhớ thì TBD phải sinh mã khả tái định vị.
    - Vào thời điểm nạp, phải chuyển đổi địa chỉ khả tái định vị thành địa chỉ thực dựa trên một địa chỉ nền (base address).
    - Địa chỉ thực được tính toán vào thời điểm nạp chương trình → phải nạp lại nếu địa chỉ nền thay đổi

12

12

## Vấn đề quản lý bộ nhớ (tt)

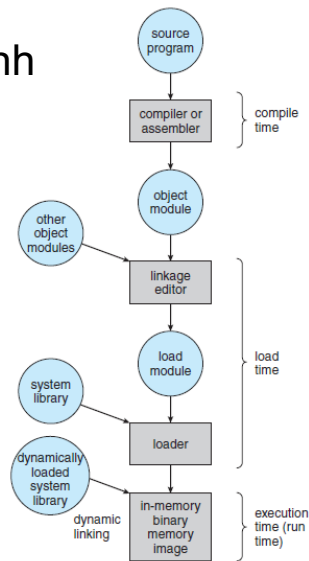
- Thời điểm gán kết (chuyển đổi) địa chỉ:
  - Thời điểm thực thi:
    - Trong quá trình thực thi, tiến trình có thể được di chuyển từ segment này sang segment khác trong bộ nhớ thì quá trình gán kết địa chỉ được thực hiện tại thời điểm thực thi
    - CPU tạo ra địa chỉ luận lý cho process
    - Cần sự hỗ trợ của phần cứng cho việc ánh xạ địa chỉ.
      - Ví dụ: trường hợp địa chỉ luận lý là relocatable thì có thể dùng thanh ghi base và limit,...
    - Sử dụng các cơ chế swapping, paging, segmentation

13

13

## Vấn đề quản lý bộ nhớ (tt)

- Các bước thực thi chương trình



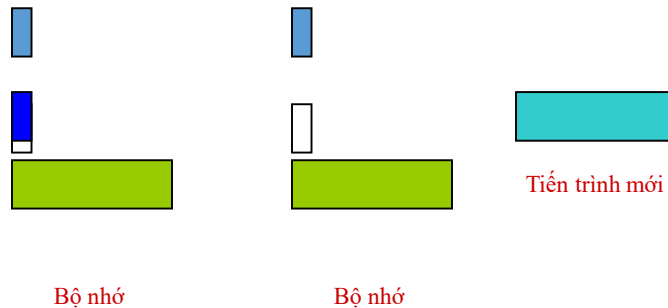
Multistep processing of a user program.

14

14

## Vấn đề quản lý bộ nhớ (tt)

- Bộ nhớ bị phân mảnh trong môi trường đa chương



15

15

## Vấn đề quản lý bộ nhớ (tt)

- Phân mảnh nội (Internal Fragmentation) – mỗi block được cấp phát lớn hơn yêu cầu bộ nhớ một ít
- Phân mảnh ngoại vi (External Fragmentation) – tổng bộ nhớ trống thỏa yêu cầu, nhưng không liên tục
- Giải pháp phân mảnh ngoại vi: kết hợp
  - Chuyển các vùng trống thành một khối bộ nhớ liên tục
  - Chỉ thực hiện được nếu HĐH hỗ trợ biên dịch địa chỉ trong thời gian thực thi

16

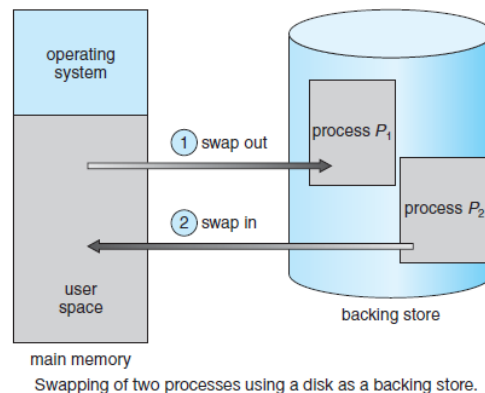
16



## Vấn đề quản lý bộ nhớ (tt)

### •Swaping:

- Tiến trình ở trạng thái chờ được tạm thời chuyển ra bộ nhớ phụ (swap out), sau đó được chuyển vào bộ nhớ chính (swap in) để tiếp tục xử lý



17

17

## Vấn đề quản lý bộ nhớ (tt)

### •Swaping (tt)

- Tăng số lượng các tiến trình đồng hành → hiệu quả trong môi trường đa chương
- Cần xác định vị trí của tiến trình trong bộ nhớ chính khi được chuyển ra/vào, phụ thuộc vào thời điểm kết gán địa chỉ
  - Thời điểm nạp, cần phải cấp phát lại đúng vùng nhớ cho tiến trình
  - Thời điểm xử lý: có thể nạp tiến trình vào một vùng nhớ bất kỳ
- Cần sử dụng bộ nhớ phụ
- Thời gian chuyển đổi: mỗi tiến trình cần được phân chia thời gian sử dụng CPU sao cho không cảm thấy bị chậm trễ do swap

18

18

## 4.2. Mô hình quản lý bộ nhớ thực

---

- Cấp phát liên tục:
  - Phân vùng cố định
  - Phân vùng động
- Cấp phát không liên tục
  - Phân trang
  - Phân đoạn

19

19

### 4.2.1 Phân vùng cố định

---

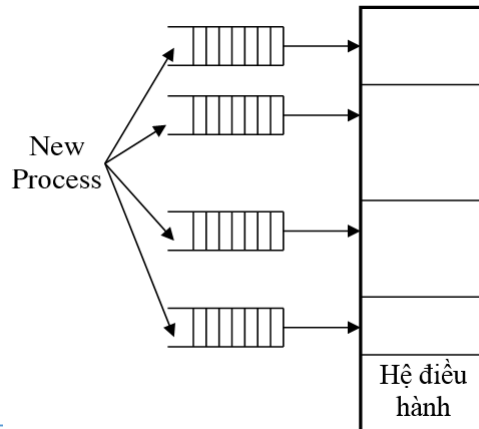
- Bộ nhớ được chia thành  $n$  phân vùng (partition) có kích thước cố định
- Các tiến trình có nhu cầu bộ nhớ sẽ được lưu trữ trong hàng đợi
- Có hai cách tổ chức:
  - Sử dụng nhiều hàng đợi
  - Sử dụng một hàng đợi

20

20

## Phân vùng cố định (tt)

- Sử dụng nhiều hàng đợi:
  - Mỗi phân vùng có một hàng đợi tương ứng
  - Tiến trình được tạo sẽ được đưa vào hàng đợi của phân vùng có kích thước nhỏ nhất đủ chứa nó
  - Hạn chế: có thể có một số hàng đợi trống vì không có tiến trình có kích thước phù hợp

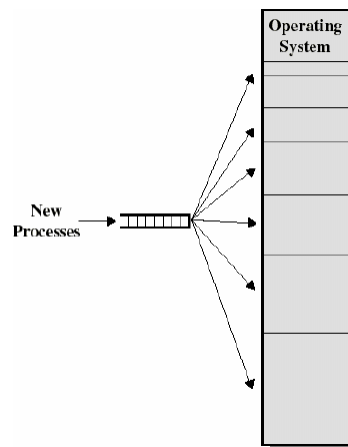


21

21

## Phân vùng cố định (tt)

- Sử dụng một hàng đợi:
  - Chỉ có một hàng đợi chung cho tất cả partition
  - Khi cần nạp một tiến trình vào bộ nhớ chính → chọn partition nhỏ nhất còn trống



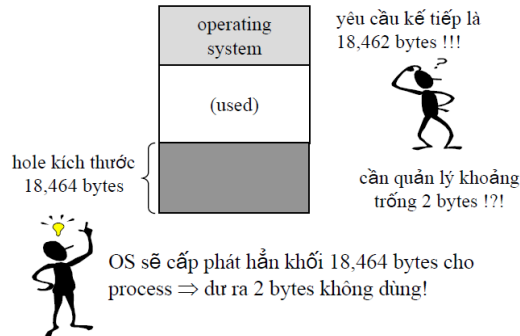
22

22

## Phân vùng cố định (tt)

### • Nhận xét:

- Kích thước tiến trình không vừa đúng bằng kích thước phân vùng chứa nó, phần bộ nhớ còn lại sẽ lãng phí (phân mảnh nội
- Số lượng phân vùng ảnh hưởng đến mức độ đa chương
- → trong môi trường đa chương cần tái định vị (relocation) và bảo vệ các tiến trình.



23

23

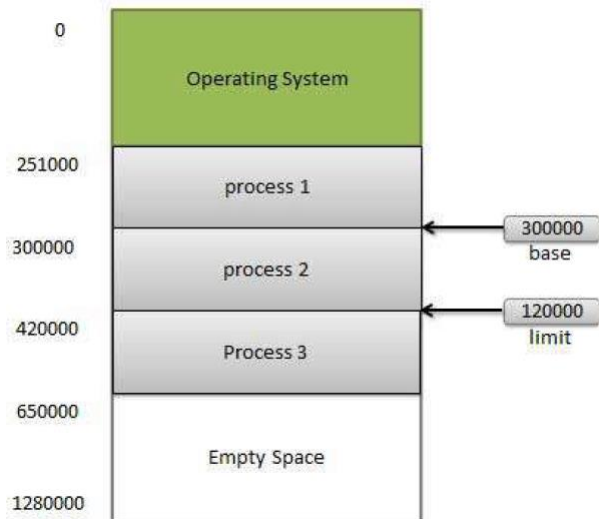
## Phân vùng cố định (tt)

- Tái định vị vào thời điểm nạp chương trình: cập nhật lại các địa chỉ trong chương trình khi chương trình được nạp vào bộ nhớ
- Sử dụng các thanh ghi đặc biệt
  - Thanh ghi nền (base register)
  - Thanh ghi giới hạn (limit register)
  - Khi tiến trình được tạo:
    - Nạp vào thanh ghi nền địa chỉ bắt đầu của phân vùng được cấp cho tiến trình
    - Nạp vào thanh ghi giới hạn kích thước của tiến trình

24

24

## Phân vùng cố định (tt)



25

25

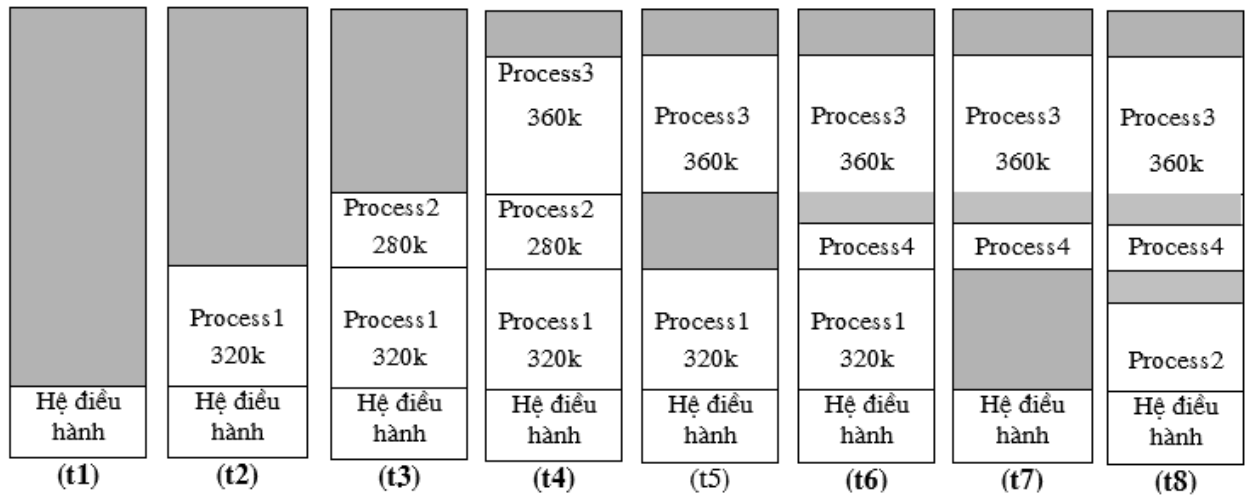
## 4.2.2 Phân vùng động

- Cấp phát vùng nhớ vừa đúng với kích thước của tiến trình được tạo
- Thu hồi vùng nhớ khi tiến trình kết thúc
- ➔ Kích thước vùng nhớ cấp cho tiến trình và vị trí bắt đầu của các phân vùng là động

26

26

## Phân vùng động (tt)

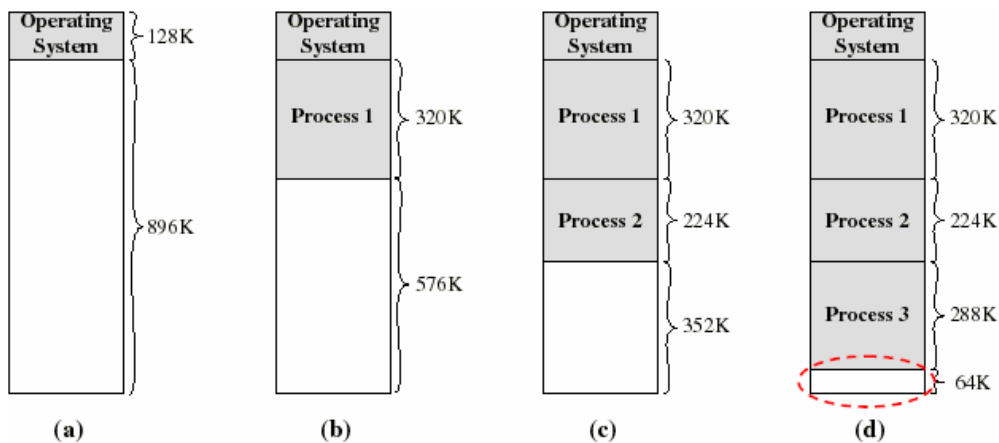


27

27

## Phân vùng động (tt)

- Nhận xét: Không phân mảnh nội, nhưng phân mảnh ngoại vi



28

28

## Phân vùng động (tt)

### • Cơ chế quản lý bộ nhớ

- Sử dụng mảng bit: bộ nhớ được chia thành những đơn vị cấp phát, mỗi bit trên dãy bit quản lý một đơn vị
  - 0: trống
  - 1: đã cấp phát cho tiến trình
- Sử dụng Danh sách liên kết: mỗi phần tử quản lý một vùng trên bộ nhớ, gồm ba trường chính:
  - trường thứ nhất cho biết khối nhớ đã cấp phát (P: process) hay đang còn trống (H: Hole)
  - trường thứ hai cho biết thứ tự của đơn vị cấp phát đầu tiên trong block
  - trường thứ ba cho biết block gồm bao nhiêu đơn vị cấp phát

29

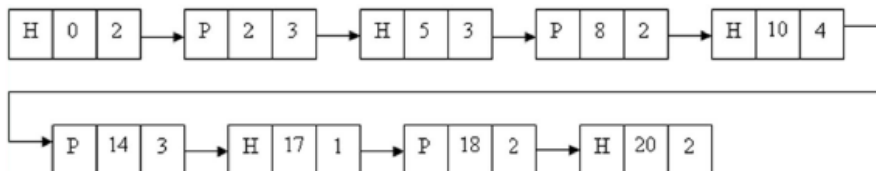
29

## Phân vùng động (tt)

### • Cơ chế quản lý bộ nhớ - ví dụ

		A						B						C				D			
00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21
0	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	1	0	1	1	0	0

quản lý các đơn vị cấp phát bằng mảng các bit



quản lý các đơn vị cấp phát bằng danh sách liên kết

30

30

## Phân vùng động (tt)

### • Chiến lược cấp phát vùng nhớ cho tiến trình:

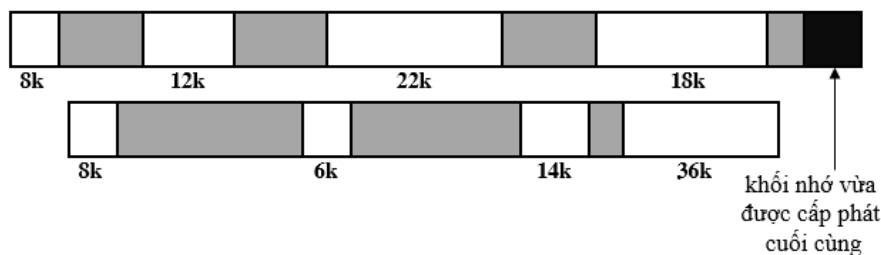
- **Best-fit:** chọn khối nhớ có kích thước nhỏ nhất vừa đúng bằng kích thước của tiến trình cần được nạp vào bộ nhớ.
- **First-fit:** chọn khối nhớ trống đầu tiên có kích thước đủ lớn để nạp tiến trình.
- **Last-fit:** chọn khối nhớ trống cuối cùng có kích thước đủ lớn để nạp tiến trình.
- **Next-fit:** tương tự như First-fit nhưng HĐH bắt đầu quét từ khối nhớ trống kế sau khối nhớ vừa được cấp phát và chọn khối nhớ trống kế tiếp đủ lớn để nạp tiến trình
- **Worst-fit:** chọn khối nhớ có kích thước lớn nhất để nạp tiến trình

31

31

## Phân vùng động (tt)

- Ví dụ: cấp phát khối nhớ cho tiến trình có kích thước 16k theo các chiến lược: First-fit, Best-fit, Next-fit, Last-fit, Worst-fit?



32

32



## Phân vùng động (tt)

### • Bài tập:

- Giả sử bộ nhớ chính được phân thành các phân vùng có thứ tự và kích thước là 580K, 500K, 200K, 300K.
- Các tiến trình có thứ tự và kích thước là 212K, 417K, 112K, 426K sẽ được cấp phát như thế nào theo các chiến lược:
  - First-fit
  - Best-fit
  - Worst-fit
  - Next-fit
  - Last-fit
- Chiến lược cấp phát bộ nhớ nào hiệu quả nhất trong trường hợp này?

33

33

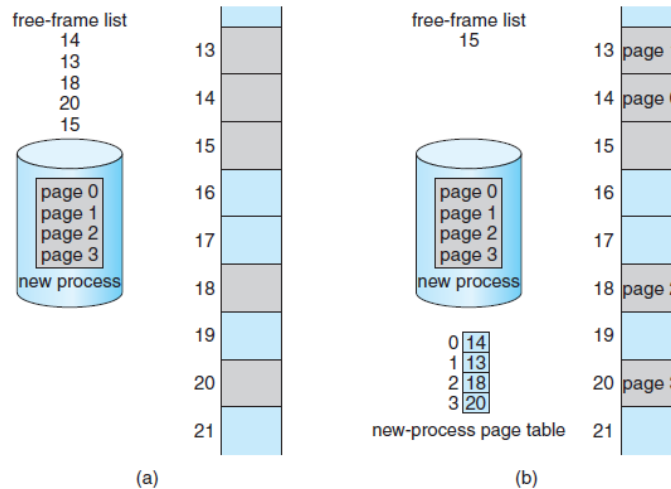
## 4.2.3 Phân trang (paging)

- Cơ chế phân trang cho phép không gian địa chỉ thực (physical address space) của một process có thể không liên tục nhau
- Bộ nhớ vật lý được chia thành các khối (block) có kích thước cố định và bằng nhau, gọi là khung trang (page frame), kích thước của frame là lũy thừa của 2 (512 bytes - 16Mb)
- Bộ nhớ luận lý (không gian địa chỉ luận lý) cũng được chia thành các khối có cùng kích thước với khung trang, gọi là trang (page).
- Khi cần nạp một tiến trình, các trang của tiến trình sẽ được nạp vào các khung trang còn trống

34

34

## Phân trang (tt)



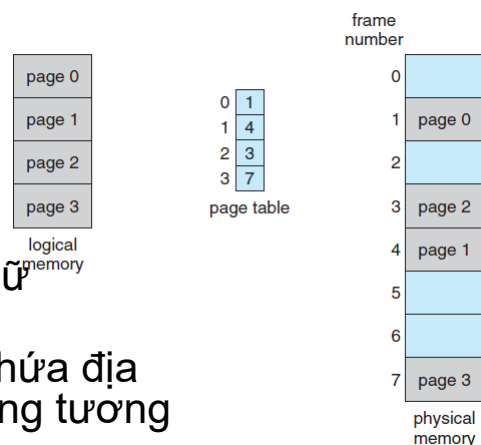
Free frames (a) before allocation and (b) after allocation.

35

35

## Phân trang (tt)

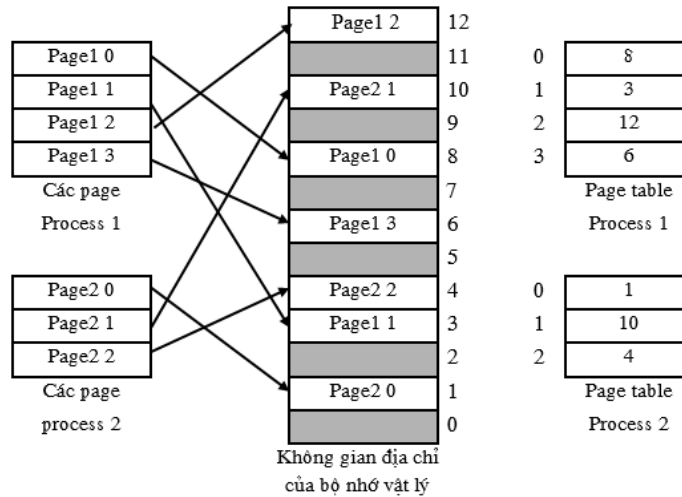
- HĐH phải thiết lập một **bảng phân trang** (page table) để ánh xạ địa chỉ luận lý thành địa chỉ thực
- Mỗi tiến trình có một bảng phân trang, được quản lý bằng một con trỏ lưu giữ trong PCB.
- Mỗi phần tử trong page table chứa địa chỉ bắt đầu của vị trí lưu trữ trang tương ứng trong bộ nhớ vật lý



36

36

## Phân trang (tt)



37

37

## Phân trang (tt)

- Địa chỉ logic phát sinh từ CPU được chia thành hai phần:
  - Page number (p): số hiệu trang, sử dụng như chỉ mục đến phần tử tương ứng trong bảng trang
  - Page offset (d): địa chỉ tương đối trong trang, kết hợp với địa chỉ bắt đầu của trang để tạo ra địa chỉ vật lý mà trình quản lý bộ nhớ sử dụng

38

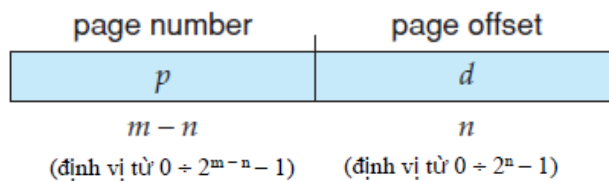
38

## Phân trang (tt)

- Kích thước trang do phần cứng qui định, thường là lũy thừa của 2 (512 bytes - 16M)

VD: Nếu kích thước của không gian địa chỉ là  $2^m$  và kích thước trang là  $2^n$ :

- p: m - n bit cao của địa chỉ ảo biểu diễn số hiệu trang
- d: n bit thấp: địa chỉ tương đối trong trang (độ dời trang)



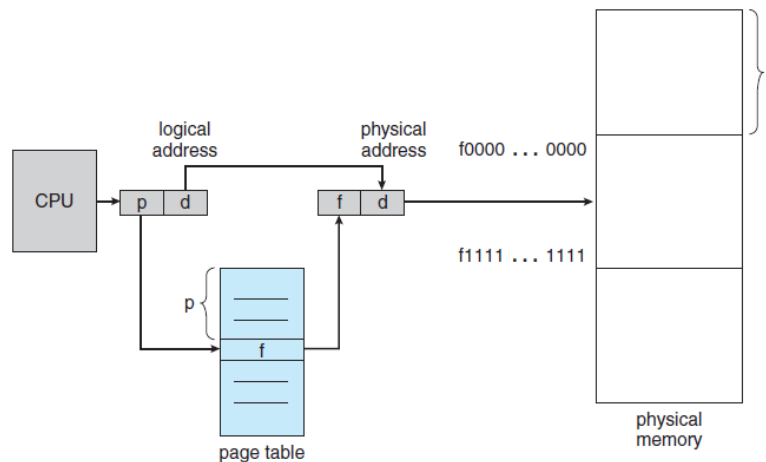
Bảng phân trang sẽ có tổng cộng  $2^m/2^n = 2^{m-n}$  mục (entry)

39

39

## Phân trang (tt)

- Phần cứng chuyển đổi địa chỉ logic sang địa chỉ vật lý:



40

40

## Phân trang (tt)

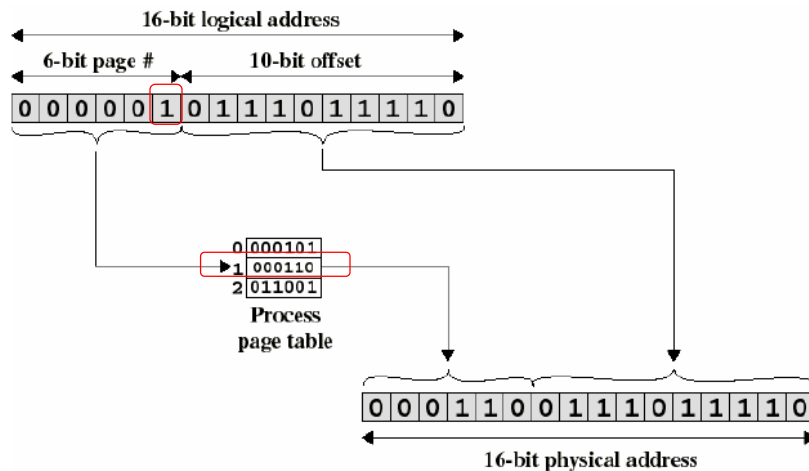
- Quá trình chuyển đổi địa chỉ logic sang địa chỉ vật lý:
  - Lấy m bit cao của địa chỉ => số thứ tự trang
  - Dựa vào bảng trang, tìm được số thứ tự khung vật lý (k)
  - Địa chỉ vật lý bắt đầu của khung là  $k \cdot 2^n$
  - Địa chỉ vật lý của byte được tham chiếu là địa chỉ bắt đầu của khung cộng với độ dời
  - => Chỉ cần thêm số khung vào trước dãy bit biểu diễn độ lệch

41

41

## Phân trang (tt)

- Quá trình chuyển đổi địa chỉ logic sang địa chỉ vật lý :



42

42

## Phân trang (tt)

- Ví dụ: địa chỉ logic với
  - $n=2, m=4$
  - Page size: 4 byte
  - physical memory: 32 byte (8 khung trang)

0	a
1	b
2	c
3	d
4	e
5	f
6	g
7	h
8	i
9	j
10	k
11	l
12	m
13	n
14	o
15	p

logical memory

0	5
1	6
2	1
3	2

page table

0	
4	i j k l
8	m n o p
12	
16	
20	a b c d
24	e f g h
28	

physical memory

Địa chỉ luận lý 0 là trang 0, độ dài 0

Dựa vào bảng trang, trang 0 nằm ở frame 5 →

- Địa chỉ luận lý 0 tương ứng với địa chỉ vật lý 20 ( $5 \times 4 + 0$ )
- Địa chỉ luận lý 3 (trang 0, độ dài 3) tương ứng với địa chỉ vật lý 23 ( $5 \times 4 + 3$ ).

Địa chỉ luận lý 4 là trang 1, độ dài 0

Dựa vào bảng trang, trang 1 nằm ở frame 6 →

- Địa chỉ luận lý 4 tương ứng với địa chỉ vật lý 24 ( $6 \times 4 + 0$ )
- Địa chỉ luận lý 13 tương ứng với địa chỉ vật lý 9.

43

43

## Phân trang (tt)

- Bài tập:
  1. Không gian địa chỉ luận lý có 8 trang, mỗi trang có kích thước 1KB, ánh xạ vào bộ nhớ vật lý có 32 khung trang
    - Địa chỉ logic có bao nhiêu bit?
    - Địa chỉ vật lý có bao nhiêu bit?
    - Bảng trang có bao nhiêu mục? Mỗi mục cần bao nhiêu bit?
  2. Một máy tính có 32 bit địa chỉ logic, 48 bit địa chỉ vật lý, kích thước một trang là 8K. Cho biết :
    - Bộ nhớ có bao nhiêu khung trang?
    - Một bảng trang có bao nhiêu phần tử?
    - Một phần tử cần bao nhiêu bit?

44

44

## Phân trang (tt)

### • Bài tập:

3. Cho bảng phân trang như sau:

Page	Frame
0	3
1	2
2	6
3	4

• Với kích thước mỗi trang là 1Kb, chuyển các địa chỉ logic sau sang địa chỉ vật lý:

- 1240
- 3580
- 5500

45

45

## Phân trang (tt)

### • Cài đặt bảng trang

- Bảng phân trang thường được lưu giữ trong bộ nhớ chính
  - Mỗi tiến trình được hệ điều hành cấp một bảng phân trang
  - Thanh ghi *page-table base* (**PTBR**) trỏ đến bảng phân trang
  - Thanh ghi *page-table length* (**PTLR**) biểu thị kích thước của bảng phân trang (có thể được dùng trong cơ chế bảo vệ bộ nhớ)

### • Nhận xét về cơ chế phân trang:

- Cấp phát không liên tục: các trang của cùng một tiến trình có thể nằm ở các vị trí không liên tục trong bộ nhớ → không phân mảnh ngoại vi
- Có phân mảnh nội: khung cuối cùng có thể không sử dụng hết kích thước

46

46

## Phân trang (tt)

- Thanh ghi kết hợp (associative register - TLB):
  - Mỗi tác vụ truy cập dữ liệu/lệnh cần hai thao tác truy xuất vùng nhớ
    - Dùng page number p làm index để truy cập mục trong bảng phân trang để lấy số frame
    - Dùng page offset d để truy xuất dữ liệu/lệnh trong frame
  - → truy cập bộ nhớ chính hai lần → truy xuất chậm → HĐH dùng một bộ phận cache phần cứng có tốc độ truy xuất và tìm kiếm cao, gọi là thanh ghi kết hợp (**associative register**) hoặc **translationlook-aside buffers** (TLBs)

47

47

## Phân trang (tt)

- TLB (tt)
  - Là thanh ghi hỗ trợ tìm kiếm truy xuất dữ liệu với tốc độ rất nhanh.
  - Số mục của TLB khoảng 8 – 2048
  - TLB là “cache” của bảng phân trang
  - Khi có chuyển ngữ cảnh, TLB bị xóa
  - Khi TLB đầy, thay thế mục dùng LRU
  - Ánh xạ địa chỉ ảo
    - Nếu page number nằm trong TLB (hit, trúng) → lấy ngay được số frame → tiết kiệm được thời gian truy cập bộ nhớ chính để lấy số frame trong bảng phân trang.
    - Ngược lại (miss, sai), phải lấy số frame từ bảng phân trang

Page #	Frame #

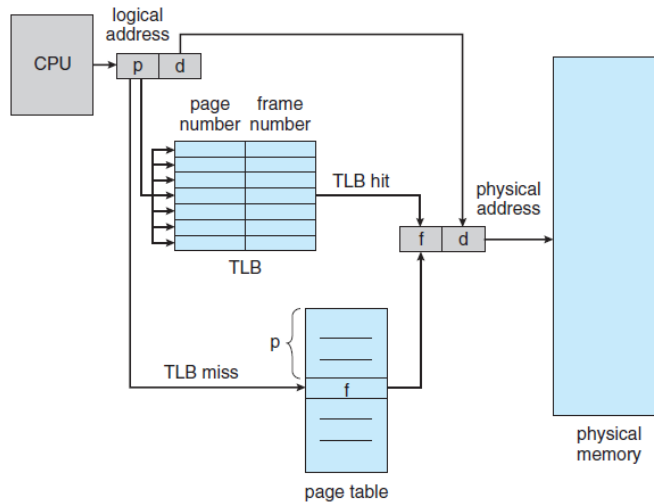
48

48



## Phân trang (tt)

### •Paging hardware với TLB



49

49

## Phân trang (tt)

- Effective access time (EAT): tính thời gian truy xuất hiệu dụng
  - Thời gian tìm kiếm trong TLB (associative lookup):  $\epsilon$
  - Thời gian một chu kỳ truy xuất bộ nhớ:  $x$
  - Hit ratio: tỉ số giữa số lần tìm thấy chỉ số trang (hit) trong TLB và số lần khởi nguồn từ CPU:  $\alpha$
  - Thời gian cần thiết để có được chỉ số frame
    - Trường hợp có trong TLB (hit):  $\epsilon + x$
    - Trường hợp không có trong TLB (miss):  $\epsilon + x + x$
  - Thời gian truy xuất hiệu dụng:
 
$$EAT = (\epsilon + x) \alpha + (\epsilon + 2x)(1 - \alpha) = (2 - \alpha)x + \epsilon$$

50

50

## Phân trang (tt)

- Ví dụ 1:
  - associative lookup = 20 (đơn vị thời gian là nanoseconds)
  - Memory access = 100
  - hit ratio = 0.8
  - $EAT = (100+20) \times 0.8 + (200+20) \times 0.2 = 96 + 44 = 140$
- Ví dụ 2:
  - associative lookup = 20
  - Memory access = 100
  - hit ratio = 0.98
  - $EAT = (100+20) \times 0.98 + (200+20) \times 0.02 = 117.6 + 4.4 = 122$

51

51

## Phân trang (tt)

- Ví dụ 3: Xét hệ thống sử dụng kỹ thuật phân trang (bảng trang lưu trữ trong bộ nhớ chính)
  - Nếu thời gian cho một lần truy xuất bộ nhớ bình thường là 200 nanoseconds thì mất bao nhiêu thời gian cho một thao tác truy xuất bộ nhớ?
  - Nếu sử dụng TLB với hit-ratio là 75%, thời gian để tìm trong TLB là 0, tính thời gian truy xuất bộ nhớ trong hệ thống?

52

52

## Phân trang (tt)

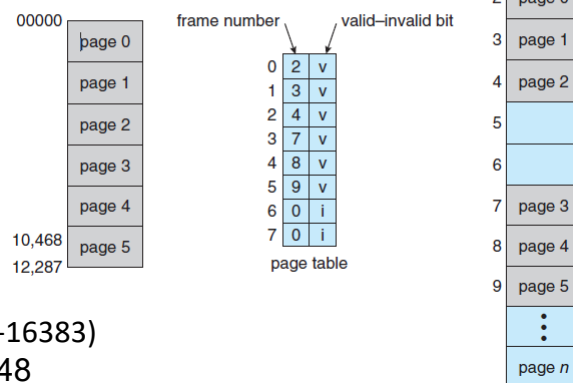
- Bảo vệ bộ nhớ:
  - Việc bảo vệ bộ nhớ được hiện thực bằng cách gắn với frame các *bit bảo vệ* (protection bits) được giữ trong bảng phân trang. Các bit này biểu thị các thuộc tính sau
    - read-only, read-write, execute-only
  - Ngoài ra, còn có một *valid / invalid bit* gắn với mỗi mục trong bảng phân trang
    - “valid”: cho biết là trang của process, do đó là một trang hợp lệ.
    - “invalid”: cho biết là trang không của process, do đó là một trang bất hợp lệ.
    - Dùng PTLR để kiểm tra truy xuất đến bảng phân trang có hợp lệ hay không

53

53

## Phân trang (tt)

- Ví dụ:



Hệ thống với không gian địa chỉ 14 bit (0-16383)

Mỗi trang nhớ có kích thước 2K = 2048

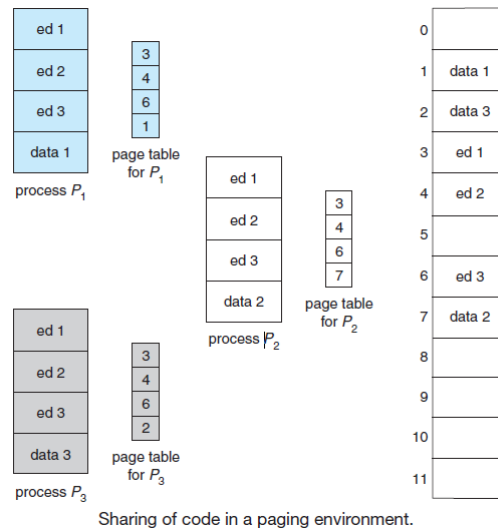
Process chỉ sử dụng các địa chỉ từ 0-10468 → phân mảnh nội ở frame 9 (chứa page 5), các truy cập vào địa chỉ lên đến 12287 là hợp lệ. Các địa chỉ từ 12288-16383 là không hợp lệ (invalid)

54

54

## Phân trang (tt)

- Chia sẻ các trang nhớ: cho phép các tiến trình có thể thực thi một mã lệnh tại một thời điểm



55

55

## Phân trang (tt)

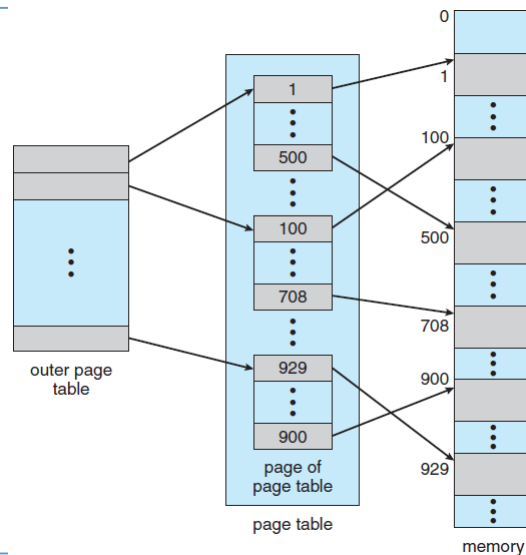
- **Bảng phân trang phân cấp**
  - Các hệ thống hiện đại đều hỗ trợ không gian địa chỉ ảo rất lớn ( $2^{32}$  đến  $2^{64}$ ), ở đây giả sử là  $2^{32}$
  - Giả sử kích thước trang nhớ là 4KB ( $2^{12}$ ) → bảng phân trang sẽ có  $2^{32}/2^{12} = 2^{20} = 1\text{M}$  mục.
  - Giả sử mỗi mục gồm 4 byte thì mỗi process cần 4MB cho bảng phân trang
  - Giải pháp: thay vì dùng một bảng phân trang duy nhất cho mỗi process, “paging” bảng phân trang này, và chỉ giữ những bảng phân trang cần thiết trong bộ nhớ → *bảng phân trang 2 mức* (two-level page table).

56

56

## Phân trang (tt)

### • Bảng phân trang 2 mức



57

57

## Phân trang (tt)

### • Ví dụ:

- 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: 20 bit địa chỉ trang được chia thành:

- 10-bit địa chỉ trang cấp 1
- 10-bit địa chỉ trang cấp 2

- Khi đó địa chỉ logic có dạng:

- $p_1$  là chỉ số đến bảng phân trang mức 1 (outer page)
- $p_2$  là chỉ số đến bảng phân trang mức 2

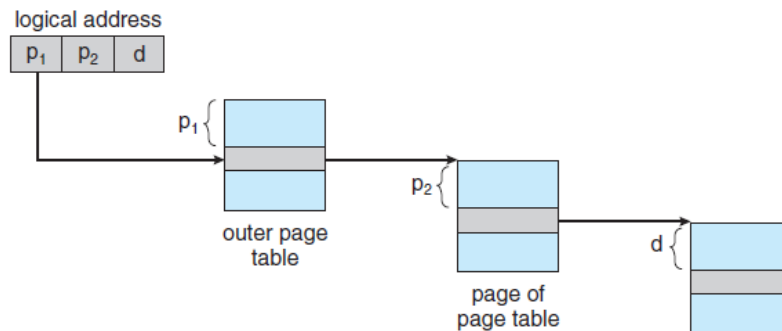
page number		page offset
$p_1$	$p_2$	$d$
10	10	12

58

58

## Phân trang (tt)

- Sơ đồ chuyển đổi địa chỉ (address-translation scheme) cho cơ chế bảng phân trang 2 mức, 32-bit địa chỉ



59

59

## Phân trang (tt)

- Với hệ điều hành 64 bits, bảng trang 2 mức không còn phù hợp:

- Giả sử trang có kích thước 4K ( $2^{12}$ ) → bảng trang có  $2^{52}$  mục
- Với bảng trang 2 mức, địa chỉ logic có dạng:

outer page	inner page	offset
$p_1$	$p_2$	$d$
42	10	12

- Bảng trang mức 1 khá lớn ( $2^{42}$ ) nên có thể được chia tiếp ...

2nd outer page	outer page	inner page	offset
$p_1$	$p_2$	$p_3$	$d$
32	10	10	12

60

60

## Phân trang (tt)

---

- Bảng phân trang băm:
  - Thường sử dụng khi địa chỉ > 32 bit
  - Để giải quyết độ rộng, mỗi mục của bảng băm là một danh sách liên kết. Mỗi phần tử của danh sách gồm các trường:
    - Chỉ số trang ảo
    - Chỉ số frame
    - Con trỏ đến phần tử kế tiếp
  - Chỉ số trang ảo (virtual page number) được biến đổi qua hàm băm thành một hashed value.
  - Phần tử tương ứng sẽ được lưu vào danh sách liên kết tại mục có chỉ số là hashed value

61

61

## Phân trang (tt)

---

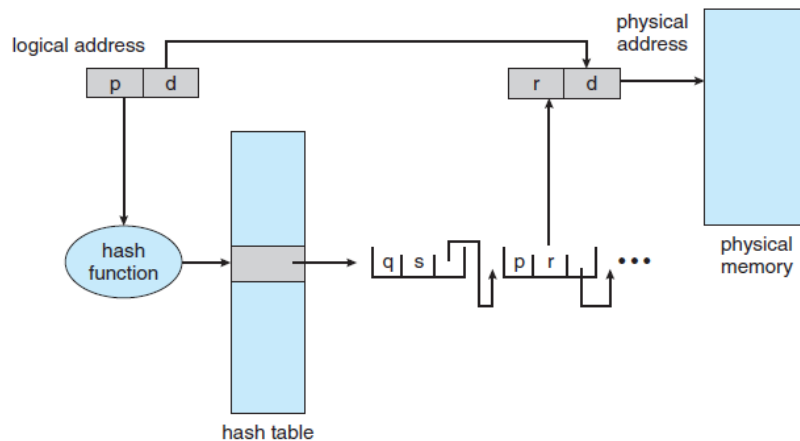
- Bảng phân trang băm:
  - Giải thuật tìm trang:
    - Chỉ số trang ảo được biến đổi thành hashed value (với cùng hàm băm).
    - Hashed value là chỉ số của mục cần truy cập trong bảng băm.
    - → Tìm trong danh sách liên kết phần tử chứa chỉ số trang ảo để lấy chỉ số frame tương ứng

62

62

## Phân trang (tt)

- Bảng phân trang băm:

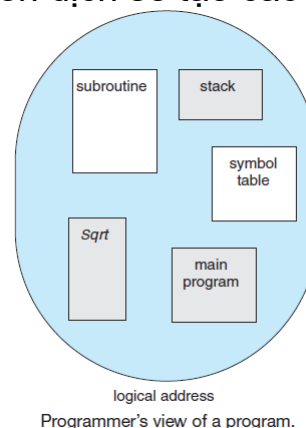


63

63

## 4.2.4 Phân đoạn (Segmentation)

- Với góc nhìn của user:
  - Khi chương trình được biên dịch, trình biên dịch sẽ tạo các segment
  - Trình loader sẽ gán mỗi segment một số định danh riêng
  - main, procedure, function
  - local variables, global variables, common block, stack, symbol table, array, ...



64

64



## Phân đoạn (tt)

- Dùng cơ chế phân đoạn để quản lý bộ nhớ có hỗ trợ user view
- Cơ chế phân trang: chỉ có một không gian địa chỉ
- Phân đoạn: Không gian địa chỉ ảo là một tập các đoạn (segment)
  - Là một dãy các địa chỉ liên tục từ 0
  - Gồm: định danh và kích thước
  - Kích thước các segment có thể khác nhau, có thể thay đổi khi chương trình thực thi (stack,...)
- Địa chỉ luận lý bao gồm: segment number và offset

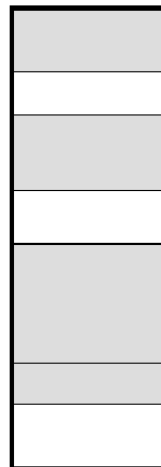
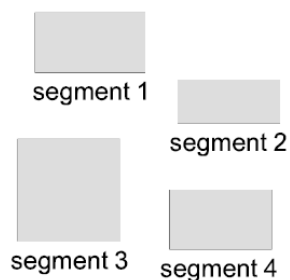
65

65

## Phân đoạn (tt)

logical address space

physical memory space

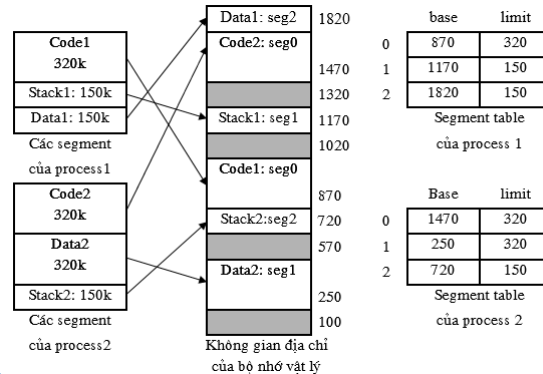


66

66

## Phân đoạn (tt)

- Khi một tiến trình được nạp vào bộ nhớ thì tất cả các đoạn của nó sẽ được nạp vào các phân đoạn còn trống (có thể không liên tiếp nhau) khác nhau trên bộ nhớ.



67

67

## Phân đoạn (tt)

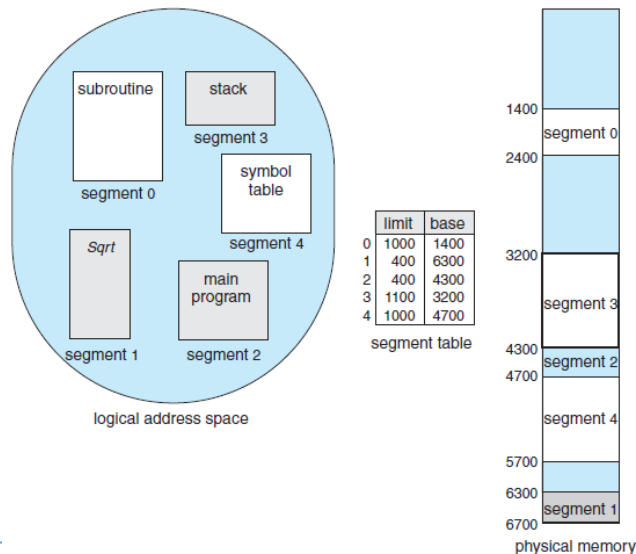
- Mỗi tiến trình có một bảng phân đoạn (SCT: Segment Control Table) riêng
- Mỗi phần tử trong bảng phân đoạn gồm tối thiểu 2 trường:
  - **base**: địa chỉ khởi đầu của phân đoạn trong bộ nhớ vật lý
  - **limit**: kích thước của phân đoạn, còn được dùng để kiểm soát sự truy xuất bất hợp lệ của các tiến trình

68

68

## Phân đoạn (tt)

### • Ví dụ về phân đoạn



69

69

## Phân đoạn (tt)

- Các bảng phân đoạn có kích thước nhỏ có thể được chứa trong các thanh ghi
- Các bảng phân đoạn có kích thước lớn được chứa trong bộ nhớ chính, HĐH dùng một thanh ghi để lưu trữ địa chỉ bắt đầu nơi lưu trữ bảng phân đoạn (**STBR: Segment Table Base Register**)
- Thanh ghi **STLR: Segment Table Length Register** được dùng để ghi kích thước hiện tại của bảng phân đoạn.
- HĐH dùng một danh sách riêng để theo dõi các segment còn trống trên bộ nhớ.

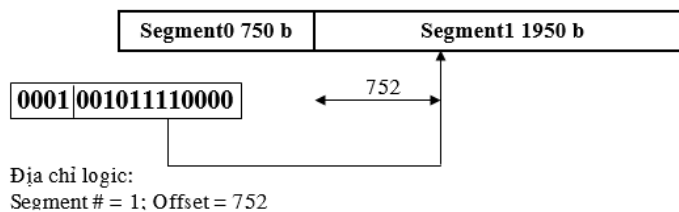
70

70

## Phân đoạn (tt)

• Địa chỉ phát sinh từ CPU được chia thành hai phần:

- **Segment number (s)** – Số hiệu đoạn: sử dụng như chỉ mục đến phần tử trong bảng phân đoạn, cho biết số hiệu đoạn tương ứng cần truy xuất.
- **Segment offset (o)** - Địa chỉ tương đối trong đoạn: giá trị này sẽ được kết hợp với địa chỉ bắt đầu của đoạn để xác định địa chỉ vật lý của ô nhớ cần truy xuất.

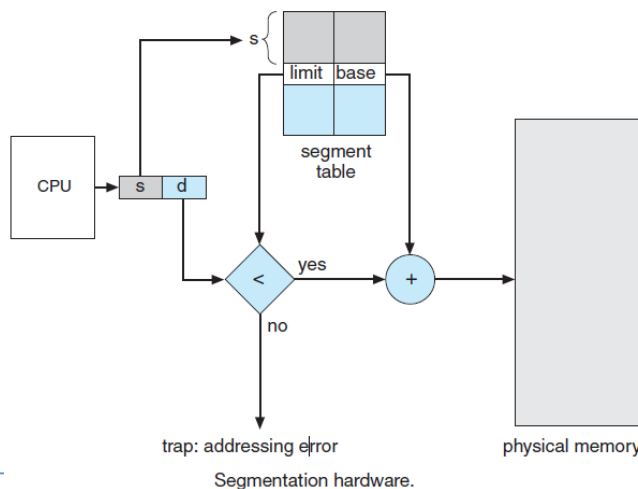


71

71

## Phân đoạn (tt)

• Phần cứng hỗ trợ chuyển đổi địa chỉ



72

72

## Phân đoạn (tt)

### • Các bước chuyển đổi địa chỉ

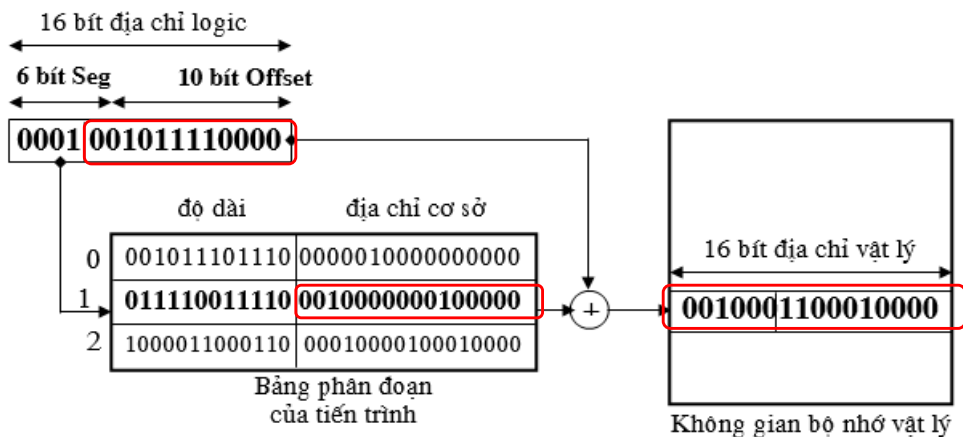
- $n$  bit trái nhất của địa chỉ logic xác định số hiệu (chỉ số) của phân đoạn cần truy xuất.
- Sử dụng số hiệu phân đoạn ở trên để chỉ đến phần tử trong bảng phân đoạn của tiến trình, để tìm địa chỉ vật lý bắt đầu của phân đoạn.
- So sánh thành phần offset của địa chỉ logic ( $m$  bit phải nhất của địa chỉ logic) với thành phần length của phân đoạn. Nếu  $\text{offset} > \text{length}$  thì địa chỉ truy xuất là không hợp lệ.
- Địa chỉ vật lý mong muốn là địa chỉ vật lý bắt đầu của phân đoạn cộng với giá trị offset.
- Giả sử có một địa chỉ logic gồm  $n + m$  bit:  $n = 6$ ,  $m = 10 \rightarrow$  kích thước tối đa của một segment là  $2^{10}$  bytes

73

73

## Phân đoạn (tt)

### • Các bước chuyển đổi địa chỉ (tt)

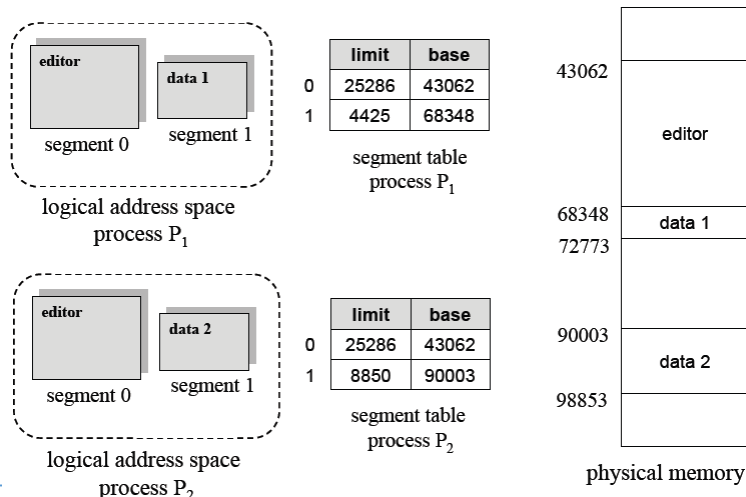


74

74

## Phân đoạn (tt)

### • Chia sẻ các đoạn



75

75

## Phân đoạn (tt)

### • Nhận xét

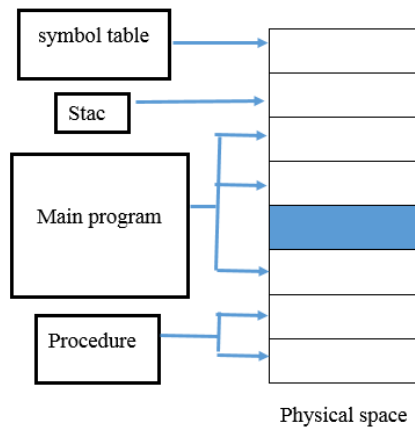
- Các segment có kích thước không bằng nhau → tương tự như sự phân vùng động.
- Một chương trình có thể chiếm giữ hơn một phân vùng có thể không liền kề với nhau.
- Loại trừ được sự phân mảnh nội, vẫn còn phân mảnh ngoại vi (phân vùng động)
- Tường minh đối với người lập trình, cho phép người lập trình tổ chức chương trình và dữ liệu (có thể gán các chương trình và dữ liệu đến các đoạn nhớ khác nhau)
- Cần giải quyết vấn đề cấp phát động: best-fit, first-fit
- đoạn có kích thước quá lớn có thể không nạp được vào bộ nhớ → Kết hợp phân trang với phân đoạn

76

76

## Phân đoạn (tt)

- Kết hợp phân trang và phân đoạn



77

77

## Phân đoạn (tt)

- Bài tập:
- Cho bảng phân đoạn như sau:

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

- Chuyển các địa chỉ logic sau sang địa chỉ vật lý: 0430, 1010

78

78

### 4.3. Mô hình quản lý bộ nhớ ảo

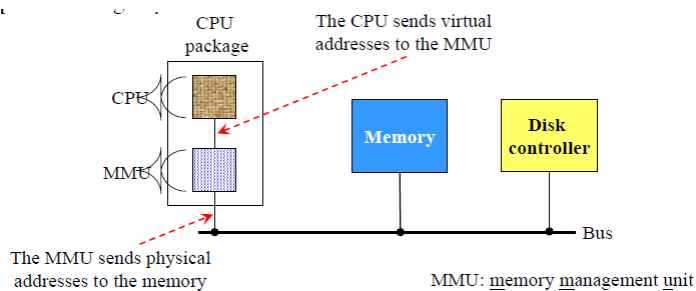
- Đặc điểm
- Kỹ thuật phân trang theo yêu cầu

79

79

#### 4.3.1 Đặc điểm mô hình quản lý bộ nhớ ảo

- Nhìn lại paging và segmentation :
  - Các tham chiếu đến bộ nhớ được chuyển đổi động thành địa chỉ thực lúc process đang thực thi
  - Một process gồm các phần nhỏ (page hay segment), các phần này được nạp vào các vùng có thể không liên tục trong bộ nhớ chính



80

80



## Đặc điểm mô hình quản lý bộ nhớ ảo (tt)

### •Nhận xét:

- không phải tất cả các phần của một process cần thiết phải được nạp vào bộ nhớ chính tại cùng một thời điểm. Ví dụ
  - Đoạn mã điều khiển các lỗi hiếm khi xảy ra
  - Các arrays, list, tables được cấp phát bộ nhớ (cấp phát tĩnh) nhiều hơn yêu cầu thực sự
  - Một số tính năng ít khi được dùng của một chương trình
- Ngay cả khi toàn bộ chương trình đều cần dùng thì có thể không cần dùng toàn bộ cùng một lúc

81

81

## Đặc điểm mô hình quản lý bộ nhớ ảo (tt)

### •Bộ nhớ ảo (virtual memory)

- Cơ chế được hiện thực trong hệ điều hành để cho phép thực thi một tiến trình mà chỉ cần giữ trong bộ nhớ chính một phần của không gian địa chỉ luận lý của nó, còn phần còn lại được giữ trên bộ nhớ phụ (đĩa).
- Cần kết hợp kỹ thuật swapping để chuyển các phần của chương trình vào-ra giữa bộ nhớ chính và bộ nhớ phụ khi cần thiết.

82

82

## Đặc điểm mô hình quản lý bộ nhớ ảo (tt)

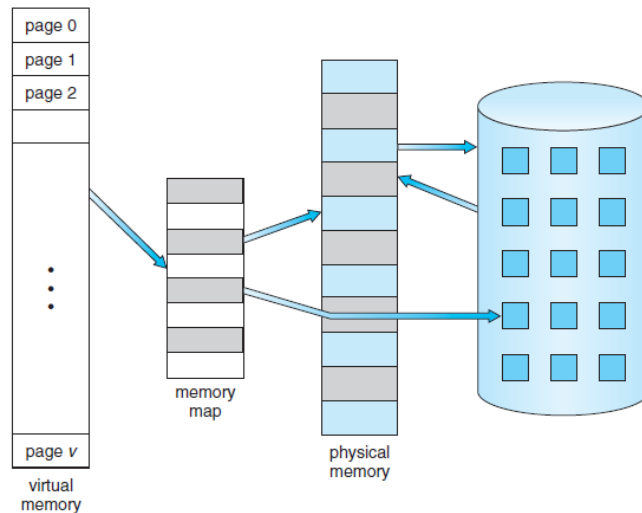


Diagram showing virtual memory that is larger than physical memory.

83

83

## Đặc điểm mô hình quản lý bộ nhớ ảo (tt)

- Ưu điểm của bộ nhớ ảo
  - Tăng số lượng tiến trình đồng thời được nạp vào bộ nhớ
  - Một tiến trình có kích thước lớn hơn bộ nhớ thực vẫn có thể thực thi
  - Bộ nhớ ảo cho phép giảm nhẹ công việc của lập trình viên vì không cần bận tâm đến giới hạn của vùng nhớ vật lý

84

84

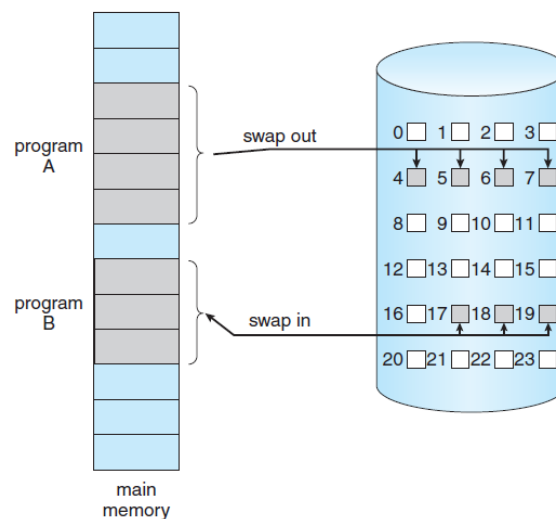
### 4.3.3 Phân trang theo yêu cầu (Demand paging)

- Sử dụng kỹ thuật **phân trang kết hợp với kỹ thuật swapping**.
- Một tiến trình được xem như một tập các trang, thường trú trên bộ nhớ phụ (đĩa).
- Khi cần xử lý, tiến trình sẽ được nạp vào bộ nhớ chính, nhưng chỉ nạp những trang cần thiết trong thời điểm hiện tại → một trang chỉ được nạp vào bộ nhớ chính khi có yêu cầu.

85

85

### Phân trang theo yêu cầu (tt)



Transfer of a paged memory to contiguous disk space.

86

86

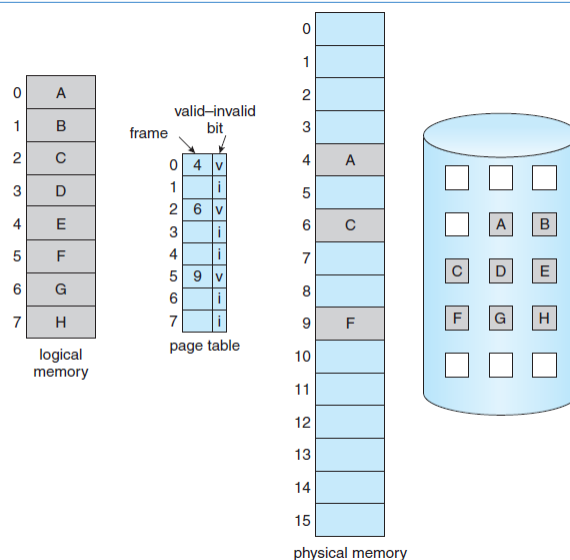
## Phân trang theo yêu cầu (tt)

- Cần cung cấp một cơ chế phần cứng giúp phân biệt các trang đang ở trong bộ nhớ chính và các trang trên đĩa → sử dụng lại bit valid-invalid:
  - valid: trang hợp lệ và đang ở trong bộ nhớ chính.
  - invalid:
    - Trang bất hợp lệ (không thuộc về không gian địa chỉ của tiến trình)
    - hoặc trang hợp lệ nhưng đang được lưu trên bộ nhớ phụ.
- Một phần tử trong bảng trang mô tả cho một trang không nằm trong bộ nhớ chính, sẽ được đánh dấu *invalid* và chứa địa chỉ của trang trên bộ nhớ phụ.

87

87

## Phân trang theo yêu cầu (tt)



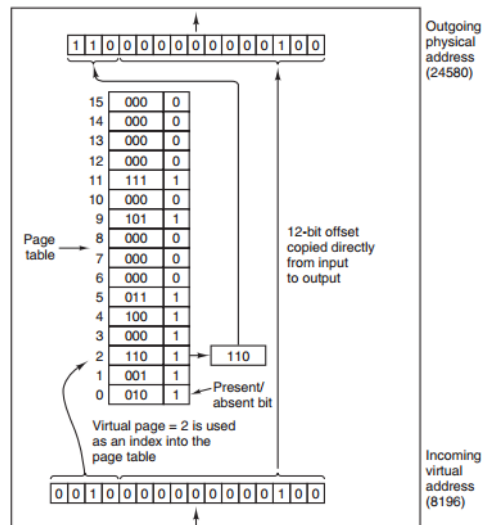
Page table when some pages are not in main memory.

88

88

## Phân trang theo yêu cầu (tt)

- Chuyển đổi địa chỉ bên trong MMU với 16 trang 4 KB



89

89

## Phân trang theo yêu cầu (tt)

- **Cơ chế phân cứng:** kết hợp của cơ chế hỗ trợ kỹ thuật phân trang và kỹ thuật swapping
  - Bảng trang: phản ánh tình trạng của một trang là đang nằm trong bộ nhớ chính hay bộ nhớ phụ.
  - Bộ nhớ phụ:
    - Thường là đĩa, lưu trữ những trang không được nạp vào bộ nhớ chính.
    - Vùng không gian đĩa dùng để lưu trữ tạm các trang trong kỹ thuật swapping được gọi là không gian swapping.

90

90

## Phân trang theo yêu cầu (tt)

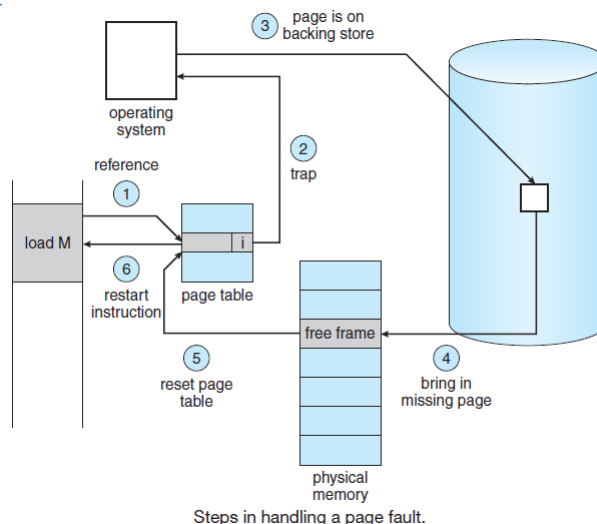
- **Lỗi trang:** Khi có một tham chiếu đến một trang không có trong bộ nhớ chính (present bit = 0) → phần cứng phát sinh một ngắt (page-fault trap) báo cho HĐH xử lý:
  - Kiểm tra truy xuất đến bộ nhớ là hợp lệ hay không hợp lệ
  - Nếu truy xuất không hợp lệ, kết thúc tiến trình.
  - Ngược lại:
    - Tìm vị trí chứa trang muốn truy xuất trên đĩa
    - Tìm một khung trang trống trong bộ nhớ chính → chuyển trang muốn truy xuất từ bộ nhớ phụ vào bộ nhớ chính bằng cách nạp trang cần truy xuất vào khung trang trống đã chọn, cập nhật nội dung bảng trang, bảng khung trang tương ứng
    - Nếu không còn khung trang trống → thay trang

91

91

## Phân trang theo yêu cầu (tt)

### • Lỗi trang (tt)



Steps in handling a page fault.

92

92

## Phân trang theo yêu cầu (tt)

### •Copy-on-Write:

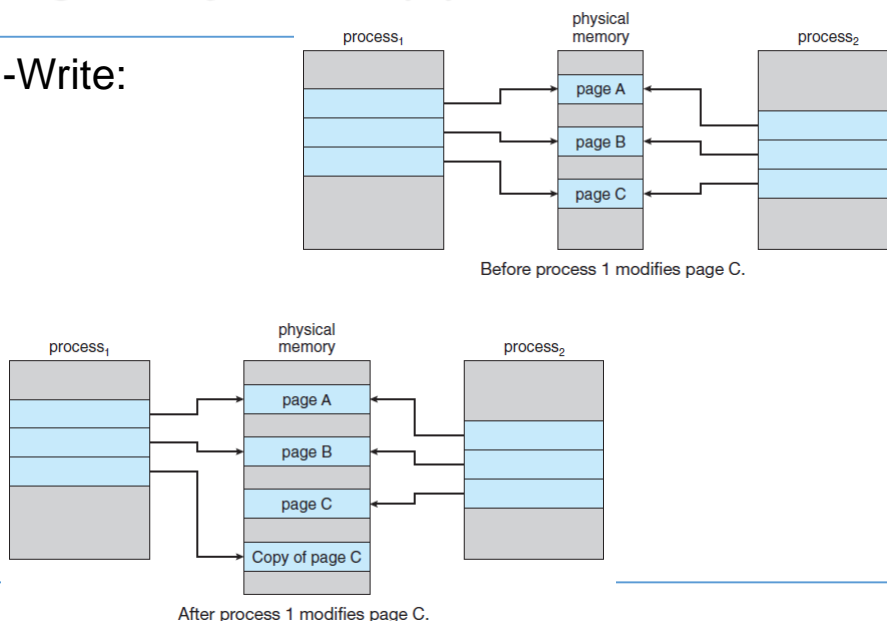
- Là một kỹ thuật phổ biến được sử dụng trong nhiều hệ điều hành (Windows XP, Linux, và Solaris)
- Cho phép tiến trình cha và con dùng chung trang trong bộ nhớ khi mới khởi tạo tiến trình con.
- Nếu một trong hai tiến trình muốn sửa đổi trang dùng chung, thì trang đó mới được copy một bản mới.
- Tạo tiến trình hiệu quả hơn: chỉ các trang bị sửa đổi mới được copy
- Các trang rồi được cấp phát từ một tập hợp các trang được xóa trắng (kỹ thuật **zero-fill-on-demand**)

93

93

## Phân trang theo yêu cầu (tt)

### •Copy-on-Write:



94

94

## Phân trang theo yêu cầu (tt)

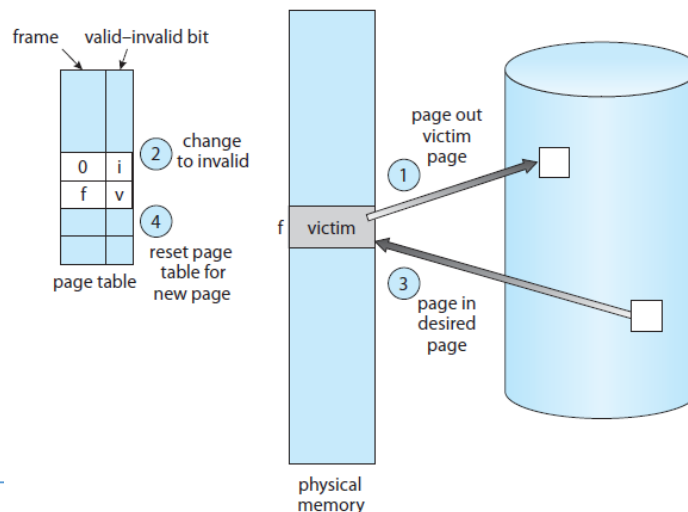
- **Thay thế trang:** (trường hợp không tìm được frame trống)
  - Chuyển một trang ra bộ nhớ phụ và nạp một trang khác vào bộ nhớ chính.
  - Dùng một giải thuật thay trang để chọn một *trang hy sinh*
  - Ghi *trang hy sinh* lên bộ nhớ ngoài (đĩa), đọc trang đang cần vào frame trống (thay vào chỗ *trang hy sinh*)

95

95

## Phân trang theo yêu cầu (tt)

- Thay thế trang:



96

96



## Phân trang theo yêu cầu (tt)

### • Thay thế trang (tt):

- Có thể sử dụng thêm một bit *cập nhật* (dirty bit) để phản ánh tình trạng trang có bị cập nhật hay không
  - dirty bit = 1: nội dung trang có bị sửa đổi
  - Khi cần thay thế một trang, nếu dirty bit = 1 → trang cần được lưu lại trên đĩa, ngược lại (trang không bị thay đổi) → không cần lưu trữ trang trở lại đĩa..

số hiệu trang	bit valid-invalid	dirty bit
---------------	-------------------	-----------

97

97

## Phân trang theo yêu cầu (tt)

- Các thuật toán thay thế trang cần phải đạt mục tiêu là chọn trang “hy sinh” sao cho sau khi thay thế sẽ gây ra ít lỗi trang nhất
- Thuật toán thay thế trang được đánh giá hiệu quả bằng cách xử lý trên một chuỗi các địa chỉ cần truy xuất và tính toán số lượng lỗi trang phát sinh

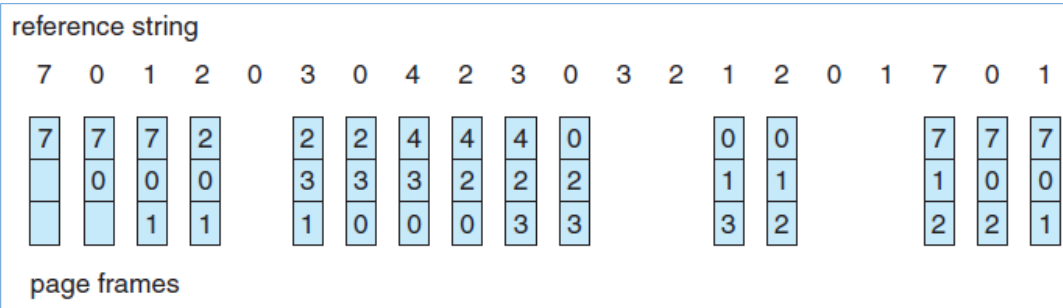
98

98

## Phân trang theo yêu cầu (tt)

### • Giải thuật thay trang FIFO

- Chọn trang ở trong bộ nhớ lâu nhất
- Ví dụ : sử dụng 3 khung trang, ban đầu cả 3 đều trống :



99

99

## Phân trang theo yêu cầu (tt)

### • Giải thuật thay trang FIFO (tt)

- Dễ hiểu, dễ cài đặt
- Có thể cho kết quả không tốt: trang được chọn để thay thế có thể là trang chứa nhiều dữ liệu cần thiết, thường xuyên được sử dụng nên được nạp sớm, do vậy khi bị chuyển ra bộ nhớ phụ sẽ nhanh chóng gây ra lỗi trang.
  - Số lượng lỗi trang xảy ra sẽ tăng lên khi số lượng khung trang nhiều → nghịch lý Belady
  - ví dụ chuỗi tham chiếu 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
  - 3 khung trang: 9 lỗi
  - 4 khung trang: 10 lỗi

100

100

## Phân trang theo yêu cầu (tt)

### • Giải thuật thay trang OPT(optimal)

- Thay thế trang sẽ được tham chiếu trễ nhất trong tương lai
- Ví dụ: sử dụng 3 khung trang, khởi đầu đều trống

reference string																			
7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
7	7	7	2		2		2			2			2				7		
	0	0	0		0		4			0			0				0		
		1	1		3		3			3			1				1		
page frames																			

- Số lượng lỗi trang thấp nhất, không xảy ra nghịch lý Belady,
- Không khả thi, vì không thể biết trước chuỗi truy xuất của tiến trình!

101

101

## Phân trang theo yêu cầu (tt)

### • Giải thuật thay trang *Least Recently Used* (LRU)

- Thay thế trang nhớ không được tham chiếu lâu nhất
- Ví dụ: sử dụng 3 khung trang, khởi đầu đều trống

reference string																			
7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
7	7	7	2		2		4	4	4	0			1		1		1		
	0	0	0		0		0	0	3	3			3		0		0		
		1	1		3		3	2	2	2			2		2		7		
page frames																			

102

102

## Phân trang theo yêu cầu (tt)

### •LRU (tt)

- Cần được cơ chế phần cứng hỗ trợ để xác định thứ tự cho các trang theo thời điểm truy xuất cuối cùng. Có hai cách:
- Sử dụng bộ đếm:
  - Mỗi phần tử trong bảng trang được bổ sung một trường ghi nhận thời điểm truy xuất mới nhất
  - Bổ sung vào cấu trúc của CPU một bộ đếm mỗi lần có sự truy xuất bộ nhớ, giá trị của counter tăng lên 1.
  - Mỗi lần thực hiện truy xuất đến một trang, ghi giá trị của counter vào một trường lưu thời điểm truy xuất mới nhất của phần tử tương ứng với trang trong bảng trang
  - Thay thế trang có giá trị trường lưu thời điểm truy xuất mới nhất là nhỏ nhất

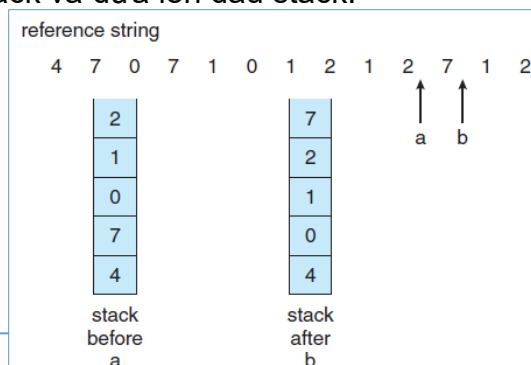
103

103

## Phân trang theo yêu cầu (tt)

### •LRU (tt)

- Sử dụng stack:
  - Tổ chức một stack lưu trữ các số hiệu trang
  - Mỗi khi thực hiện một truy xuất đến một trang, số hiệu của trang sẽ được xóa khỏi vị trí hiện hành trong stack và đưa lên đầu stack.
  - Trang ở đỉnh stack là trang được truy xuất gần nhất, và trang ở đáy stack là trang lâu nhất chưa được sử dụng.



104

104

## Phân trang theo yêu cầu (tt)

---

- Bài tập: cho chuỗi truy xuất bộ nhớ sau: 1, 2, 3, 4, 2, 1, 5, 3, 5, 1, 4, 2, 4, 5, 3, 2, 4  
Giả sử bộ nhớ vật lý có 4 khung trang. Minh họa kết quả trình bày thay thế trang với thuật toán FIFO, OPT, LRU

---

105