Chương 8

Bộ Nhớ Ảo

Nội dung trình bày

- Tổng quan về bộ nhớ ảo
- Cài đặt bộ nhớ ảo : demand paging
- Cài đặt bộ nhớ ảo : Page Replacement
  - Các giải thuật thay trang (Page Replacement Algorithms)
- Vấn đề cấp phát Frames
- Vấn đề Thrashing
- Cài đặt bộ bộ nhớ ảo : Demand Segmentation

1. Tổng quan bộ nhớ ảo

- 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
  - Cả chương trình thì cũng có đoạn code chưa cần dùng
- *Bộ nhớ ảo* (virtual memory): Bộ nhớ ảo là một kỹ thuật cho phép xử lý một tiến trình không được nạp toàn bộ vào bộ nhớ vật lý

1. Bộ nhớ ảo (tt)

Ưu điểm của bộ nhớ ảo

- Số lượng process trong bộ nhớ nhiều hơn
- Một process có thể thực thi ngay cả khi kích thước của nó lớn hơn bộ nhớ thực
- Giảm nhẹ công việc của lập trình viên

- *Không gian tráo đổi* giữa bộ nhớ chính và bộ nhớ phụ(swap space).

  Ví dụ:
  - swap partition trong Linux
  - file pagefile.sys trong Windows

2. Cài đặt bộ nhớ ảo

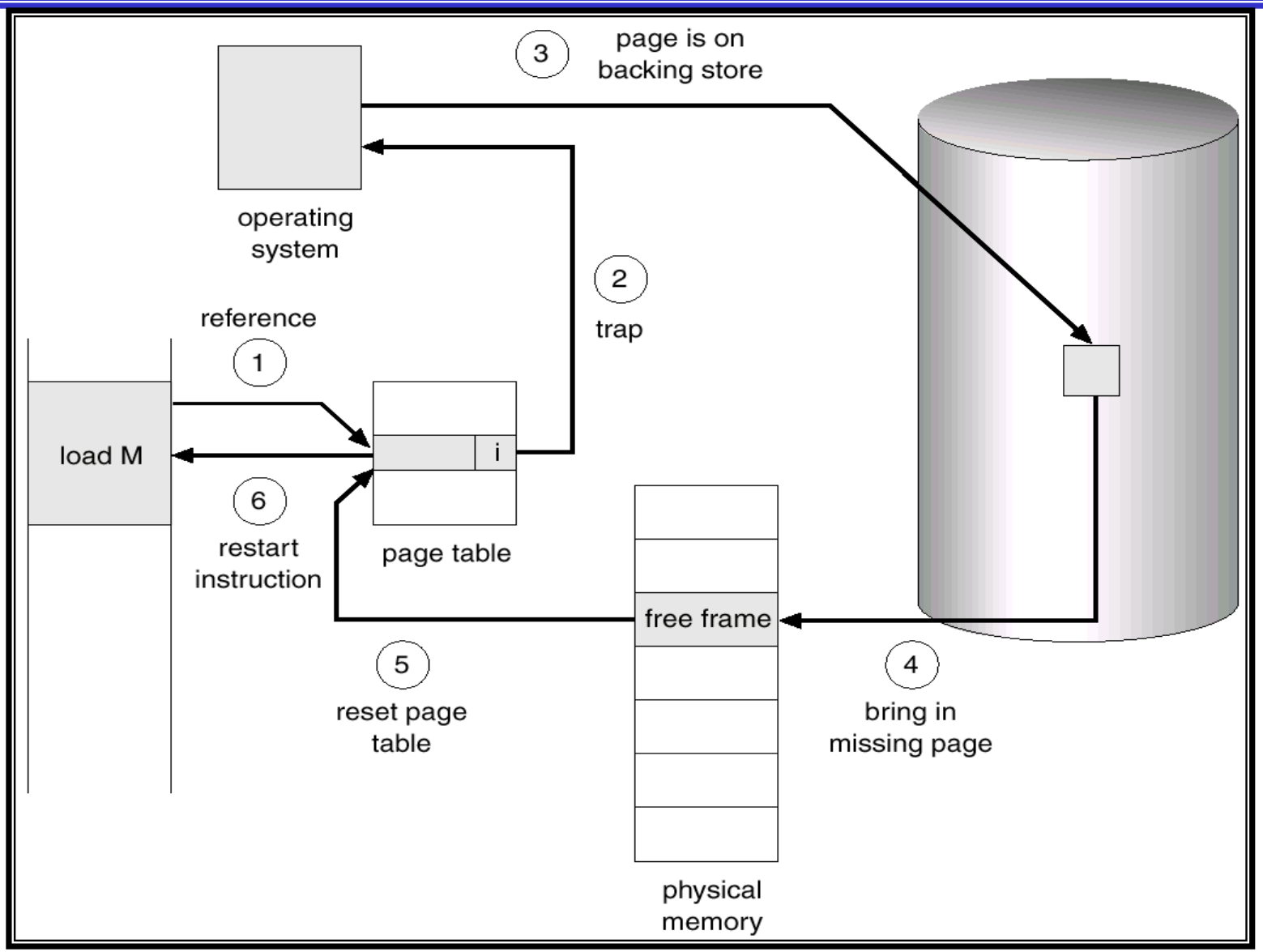
- Có hai kỹ thuật:
  - Phân trang theo yêu cầu (Demand Paging)
  - Phân đoạn theo yêu cầu (Segmentation Paging)
- Phần cứng memory management phải hỗ trợ paging và/hoặc segmentation
- OS phải quản lý sự di chuyển của trang/đoạn giữa bộ nhớ chính và bộ nhớ thứ cấp
- Trong chương này,
  - Chỉ quan tâm đến paging
  - Phần cứng hỗ trợ hiện thực bộ nhớ ảo
  - Các giải thuật của hệ điều hành

2.1.Phân trang theo yêu cầu demand paging

Demand paging: các trang của quá trình chỉ được nạp vào bộ nhớ chính khi được yêu cầu.

- Khi có một tham chiếu đến một trang mà không có trong bộ nhớ chính (valid bit) thì phần cứng sẽ gây ra một ngắt (gọi là *page-fault trap*) kích khởi *page-fault service routine* (PFSR) của hệ điều hành.
- PFSR:
  1. Chuyển process về trạng thái blocked
  2. Phát ra một yêu cầu đọc đĩa để nạp trang được tham chiếu vào một frame trống; trong khi đợi I/O, một process khác được cấp CPU để thực thi
  3. Sau khi I/O hoàn tất, đĩa gây ra một ngắt đến hệ điều hành; PFSR cập nhật page table và chuyển process về trạng thái ready.

2.2. Lỗi trang và các bước xử lý

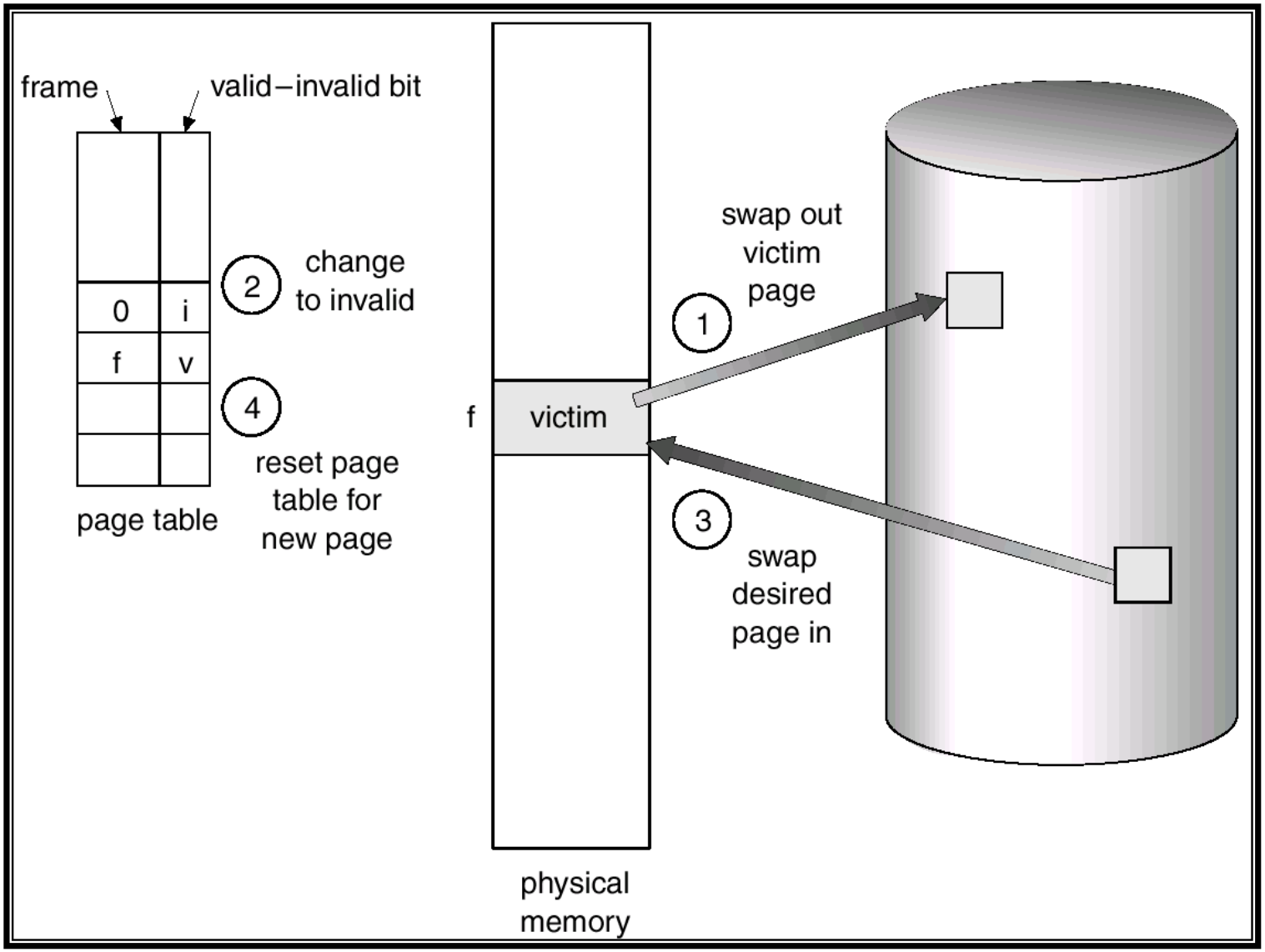


2.3. Thay thế trang nhớ

- Bước 2 của PFSR giả sử phải *thay trang* vì không tìm được frame trống, PFSR được bổ sung như sau

1. Xác định vị trí trên đĩa của trang đang cần
2. Tìm một frame trống:
   a. Nếu có frame trống thì dùng nó
   b. Nếu không có frame trống thì dùng một giải thuật thay trang để chọn một *trang hy sinh* (victim page)
   c. Ghi victim page lên đĩa; cập nhật page table và frame table tương ứng
3. Đọc trang đang cần vào frame trống (đã có được từ bước 2); cập nhật page table và frame table tương ứng.

2.3. Thay thế trang nhớ (tt)



2.4. Các thuật toán thay thế trang

Hai vấn đề chủ yếu:

- Frame-allocation algorithm
  - Cấp phát cho process bao nhiêu frame của bộ nhớ thực?
- Page-replacement algorithm
  - Chọn frame của process sẽ được thay thế trang nhớ
  - Mục tiêu: số lượng page-fault nhỏ nhất
  - Được đánh giá bằng cách thực thi giải thuật đối với một *chuỗi tham chiếu bộ nhớ* (memory reference string) và xác định số lần xảy ra page fault

- Ví dụ

  Thứ tự tham chiếu các địa chỉ nhớ, với page size = 100:

  0100, **04**32, **01**01, **06**12, **01**02, **01**03, **01**04, **01**01, **06**11, **01**02, **01**03, **01**04, **01**01, **06**10, **01**02, **01**03, **01**04, **01**01, **06**09, **01**02, **01**05

⇒ các trang nhớ sau được tham chiếu lần lượt = chuỗi tham chiếu bộ nhớ (trang nhớ)

1, **4**, **1**, **6**, **1**,
1, **1**, **1**, **6**, **1**,
1, **1**, **1**, **6**, **1**,
1, **1**, **1**, **6**, **1**,
1

a) Giải thuật thay trang *FIFO*

- Các dữ liệu cần biết ban đầu:
  - Số khung trang
  - Tình trạng ban đầu
  - Chuỗi tham chiếu
  - Trang nhớ cũ nhất sẽ được thay thế

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	4	4	4	0	0	0	0	0	0	0	7	7	7
	0	0	0	0	3	3	3	2	2	2	2	2	1	1	1	1	1	0	0
		1	1	1	1	0	0	0	3	3	3	3	3	2	2	2	2	2	1
*	*	*	*		*	*	*	*	*	*			*	*			*	*	*

Nghịch lý *Belady*

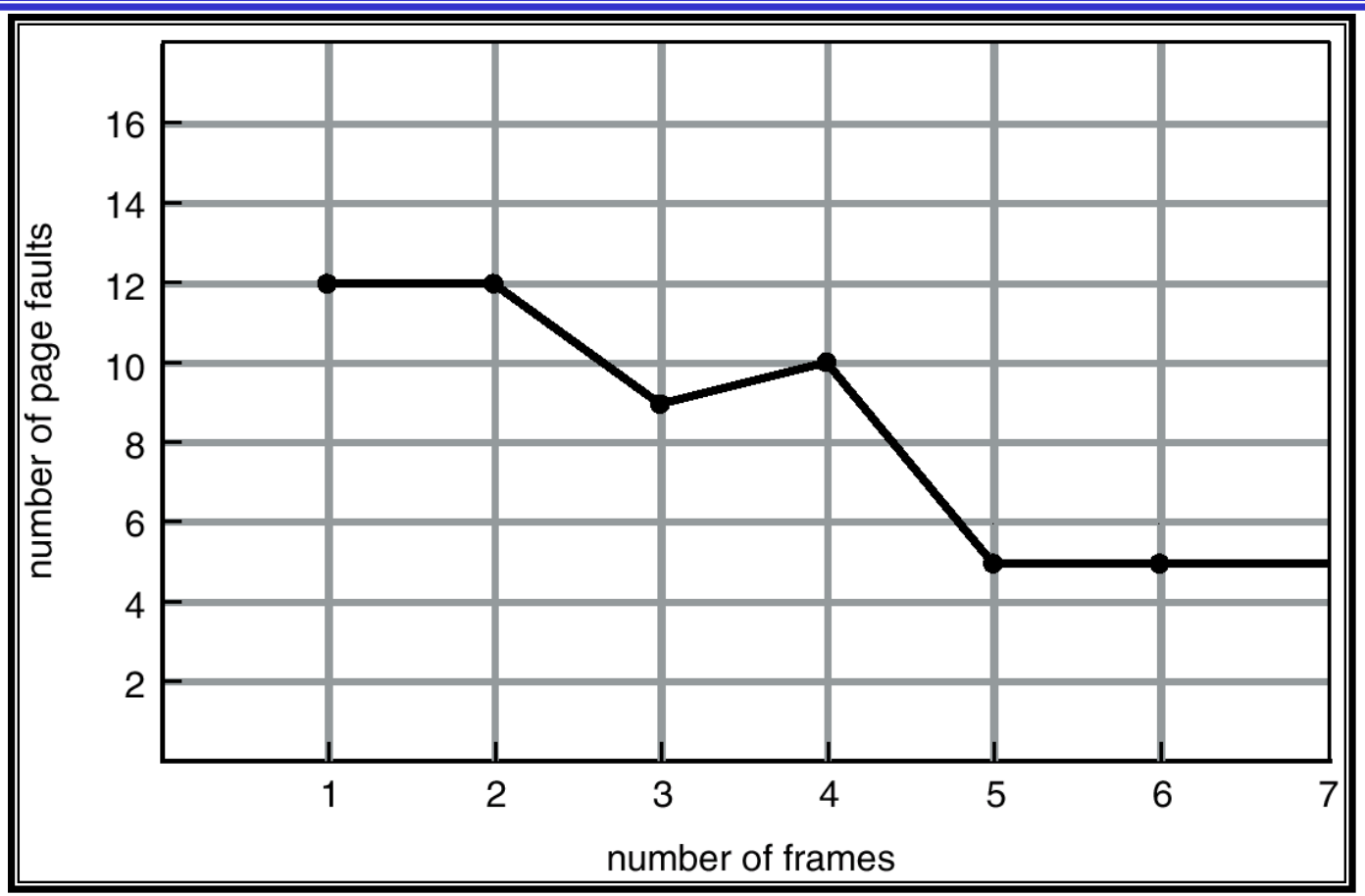
Sử dụng 3 khung trang , sẽ có 9 lỗi trang phát sinh

1	2	3	4	1	2	5	1	2	3	4	5
1	1	1	4	4	4	5	5	5	5	5	5
	2	2	2	1	1	1	1	1	3	3	3
		3	3	3	2	2	2	2	2	4	4
*	*	*	*	*	*	*			*	*	

Sử dụng 4 khung trang , sẽ có 10 lỗi trang phát sinh

1	2	3	4	1	2	5	1	2	3	4	5
1	1	1	1	1	1	5	5	5	5	4	4
	2	2	2	2	2	2	1	1	1	1	5
		3	3	3	3	3	3	2	2	2	2
			4	4	4	4	4	4	3	3	3
*	*	*	*			*	*	*	*	*	*

Nghịch lý *Belady*



Bất thường (anomaly) Belady: số page fault tăng mặc dầu quá trình đã được cấp nhiều frame hơn.

2.4 b)Giải thuật thay trang *OPT(optimal)*

- Giải thuật thay trang OPT
  - Thay thế trang nhớ sẽ được tham chiếu trễ nhất trong tương lai
  - Khó hiện thực?
- Ví dụ: một process có 7 trang, và được cấp 3 frame

sử dụng 3 khung trang, khởi đầu đều trống:

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	2	2	2	2	2	2	2	2	2	7	7	7
	0	0	0	0	0	0	4	4	4	0	0	0	0	0	0	0	0	0	0
		1	1	1	3	3	3	3	3	3	3	3	1	1	1	1	1	1	1
*	*	*	*		*		*			*			*				*		

c) Giải thuật lâu nhất chưa sử dụng *Least Recently Used (LRU)*

- Ví dụ:

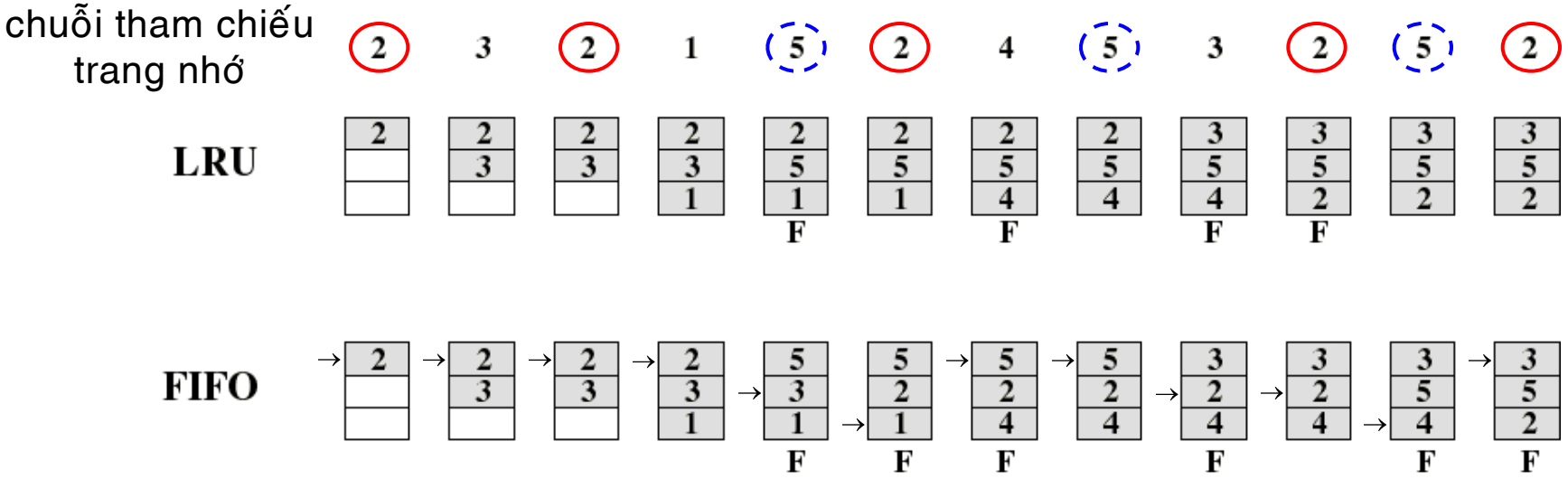
sử dụng 3 khung trang, khởi đầu đều trống:

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	4	4	4	0	0	0	1	1	1	1	1	1	1
	0	0	0	0	0	0	0	0	3	3	3	3	3	3	0	0	0	0	0
		1	1	1	3	3	3	2	2	2	2	2	2	2	2	7	7	7	
*	*	*	*		*		*	*	*	*			*		*		*		

- Mỗi trang được ghi nhận (trong bảng phân trang) thời điểm được tham chiếu ⇒ trang LRU là trang nhớ có thời điểm tham chiếu nhỏ nhất (OS tốn chi phí tìm kiếm trang nhớ LRU này mỗi khi có page fault)
- Do vậy, LRU cần sự hỗ trợ của phần cứng và chi phí cho việc tìm kiếm. Ít CPU cung cấp đủ sự hỗ trợ phần cứng cho giải thuật LRU.

LRU và FIFO

- So sánh các giải thuật thay trang LRU và FIFO

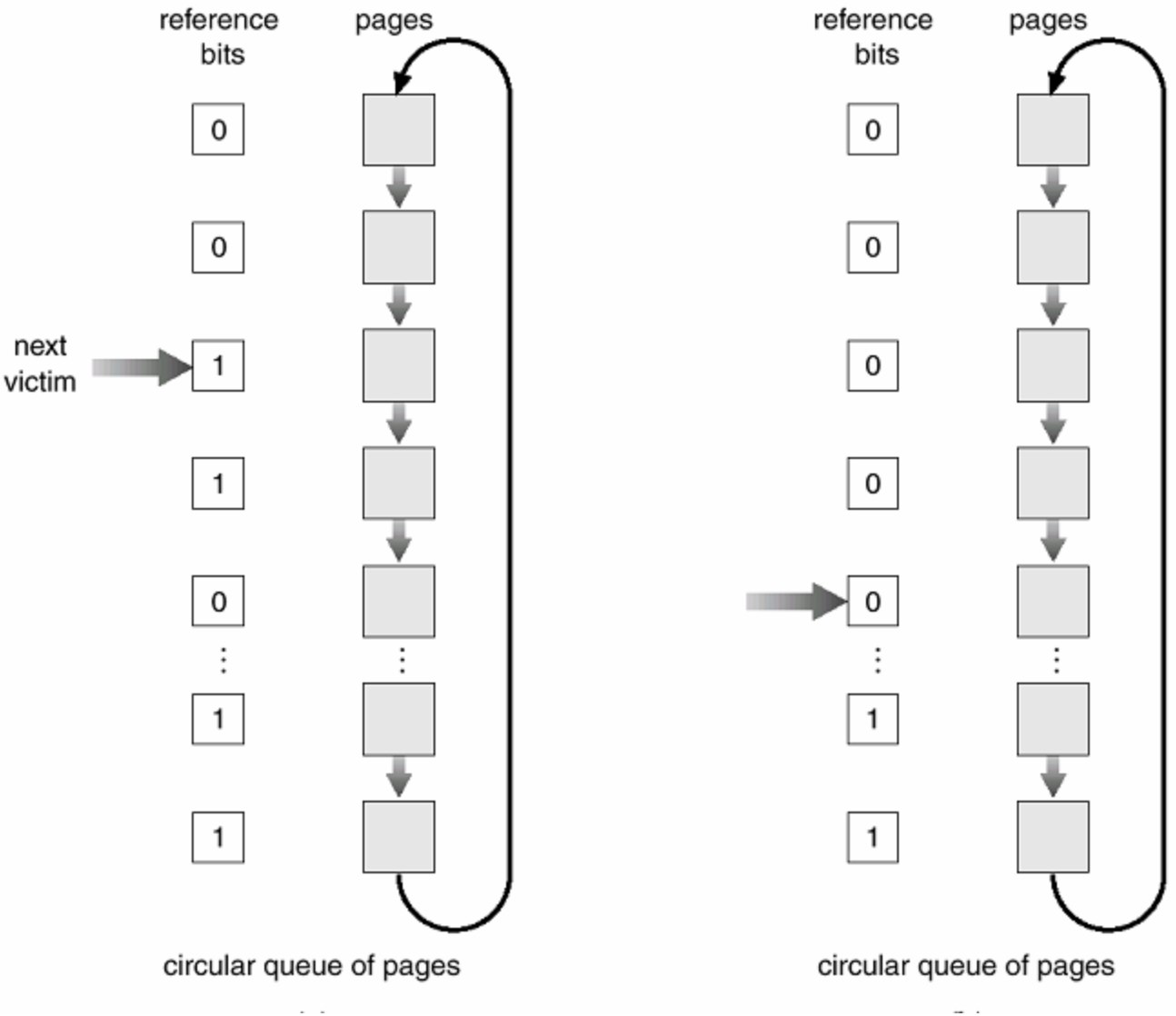


Giải thuật cơ hội thứ hai

- Sử dụng các bit tham khảo tại những khoản thời gian đều đặn
- Dùng một byte cho mỗi trang trong một bảng nằm trong bộ nhớ
- Dùng một thanh ghi dịch chứa lịch sử tham khảo trong 8 lần gần nhất

  VD: 00110101, 00000000, 11111111
- **Là giải thuật thay thế FIFO**, trước khi thay thế một trang xem xét bit tham khảo của nó
- **Đôi khi sử dụng hai bit**: tham khảo và sửa đổi như một cặp (x,x):
  - (0,0) không được dùng mới đây và không được sửa đổi-là trang tốt nhất để thay thế.
  - (0,1) không được dùng mới đây nhưng được sửa đổi-không thật tốt vì trang cần được viết ra trước khi thay thế.
  - (1,0) được dùng mới đây nhưng không được sửa đổi-nó có thể sẽ nhanh chóng được dùng lại.
  - (1,1) được dùng mới đây và được sửa đổi-trang có thể sẽ nhanh chóng được dùng lại và trang sẽ cần được viết ra đĩa trước khi nó có thể được thay thế.

Giải thuật cơ hội thứ hai (tt)



2.5.Số lượng frame cấp cho process

- OS phải quyết định cấp cho mỗi process bao nhiêu frame.
  - Cấp ít frame ⇒ nhiều page fault
  - Cấp nhiều frame ⇒ giảm mức độ multiprogramming

- Chiến lược cấp phát tĩnh (fixed-allocation)
  - Số frame cấp cho mỗi process không đổi, được xác định vào thời điểm loading và có thể tùy thuộc vào từng ứng dụng (kích thước của nó,…)
- Chiến lược cấp phát động (variable-allocation)
  - Số frame cấp cho mỗi process có thể thay đổi trong khi nó chạy
    - Nếu tỷ lệ page-fault cao ⇒ cấp thêm frame
    - Nếu tỷ lệ page-fault thấp ⇒ giảm bớt frame
  - OS phải mất chi phí để ước định các process

a) Chiến lược cấp phát tĩnh

- *Cấp phát bằng nhau:* Ví dụ, có 100 frame và 5 process $\rightarrow$ mỗi process được 20 frame
- *Cấp phát theo tỉ lệ:* dựa vào kích thước process

$s_i = \text{size of process } p_i$

$S = \sum s_i$

$m = \text{total number of frames}$

$a_i = \text{allocation for } p_i = \frac{s_i}{S} \times m$

Ví dụ:

$m = 64$

$s_i = 10$

$s_2 = 127$

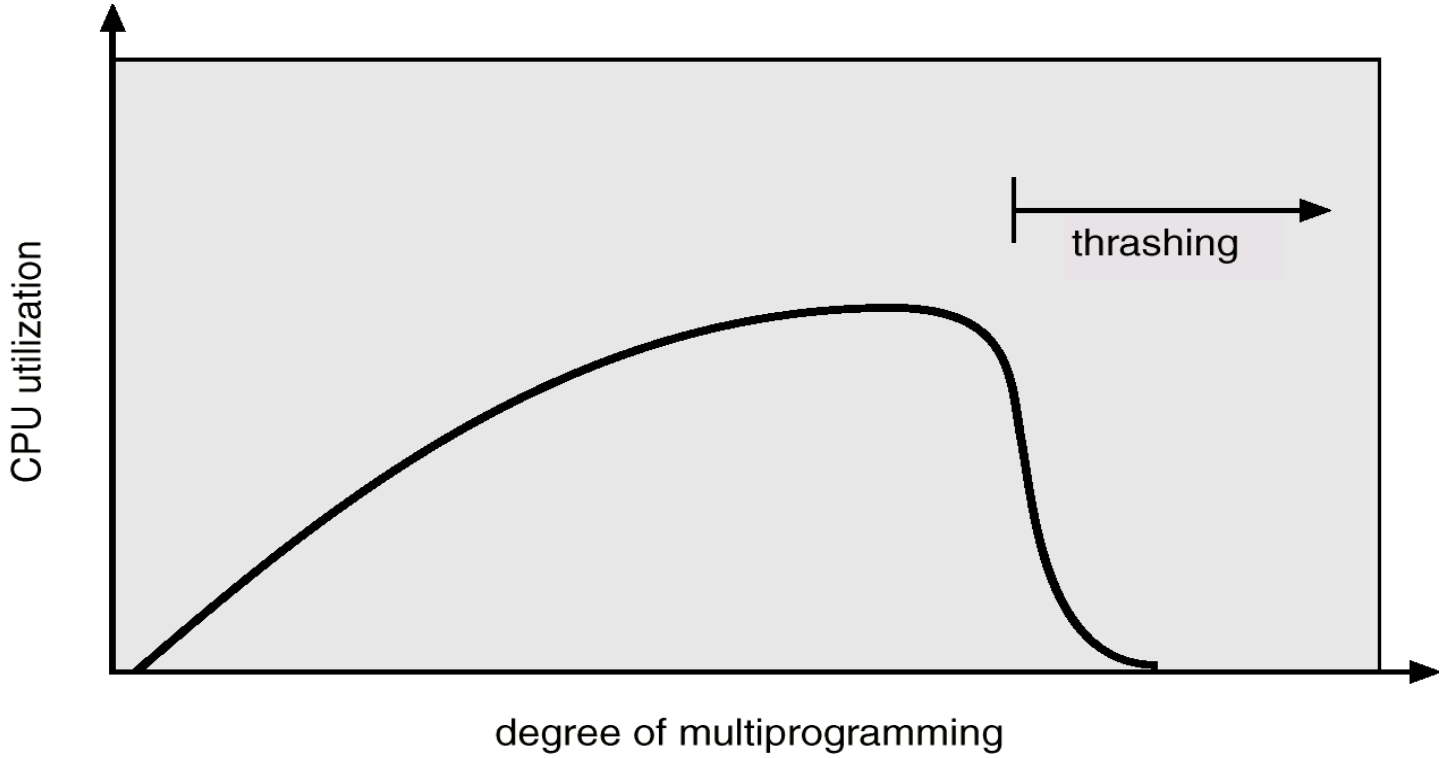
$a_1 = \frac{10}{137} \times 64 \approx 5$

$a_2 = \frac{127}{137} \times 64 \approx 59$

- Cấp phát theo độ ưu tiên

3. Trì trên toàn bộ hệ thống
Thrashing

- Nếu một process không có đủ số frame cần thiết thì tỉ số page faults/sec rất cao.
- *Thrashing:* hiện tượng các trang nhớ của một process bị hoán chuyển vào/ra liên tục.



a)Mô hình cục bộ (Locality)

- Để hạn chế thrashing, hệ điều hành phải cung cấp cho process càng "đủ" frame càng tốt. Bao nhiêu frame thì đủ cho một process thực thi hiệu quả?

  Nguyên lý locality (locality principle)

  - *Locality* là tập các trang được tham chiếu gần nhau
  - Một process gồm nhiều locality, và trong quá trình thực thi, process sẽ chuyển từ locality này sang locality khác

- Vì sao hiện tượng thrashing xuất hiện?

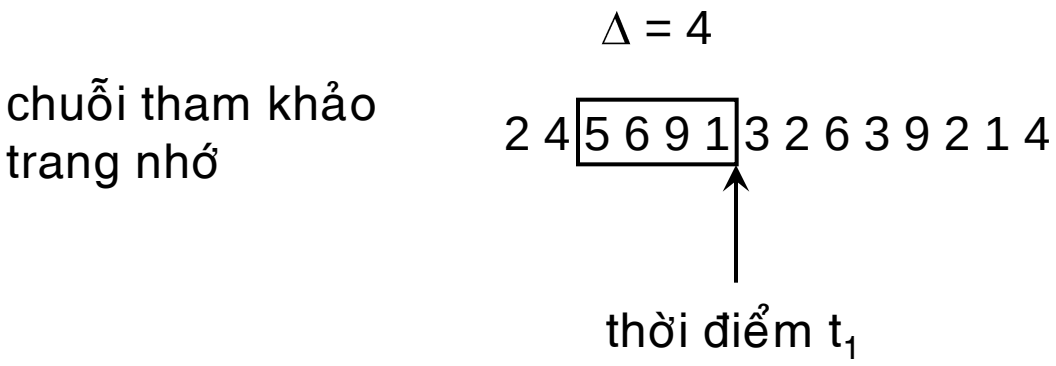
  Khi Σ size of locality > memory size

b) Giải pháp tập làm việc (working set)

Được thiết kế dựa trên nguyên lý locality.

- Xác định xem process thực sự sử dụng bao nhiêu frame.
- Định nghĩa:
  - WS(t) - số lượng các tham chiếu trang nhớ của process gần đây nhất cần được quan sát trong khoảng thời gian Δ.
  - Δ - khoảng thời gian tham chiếu

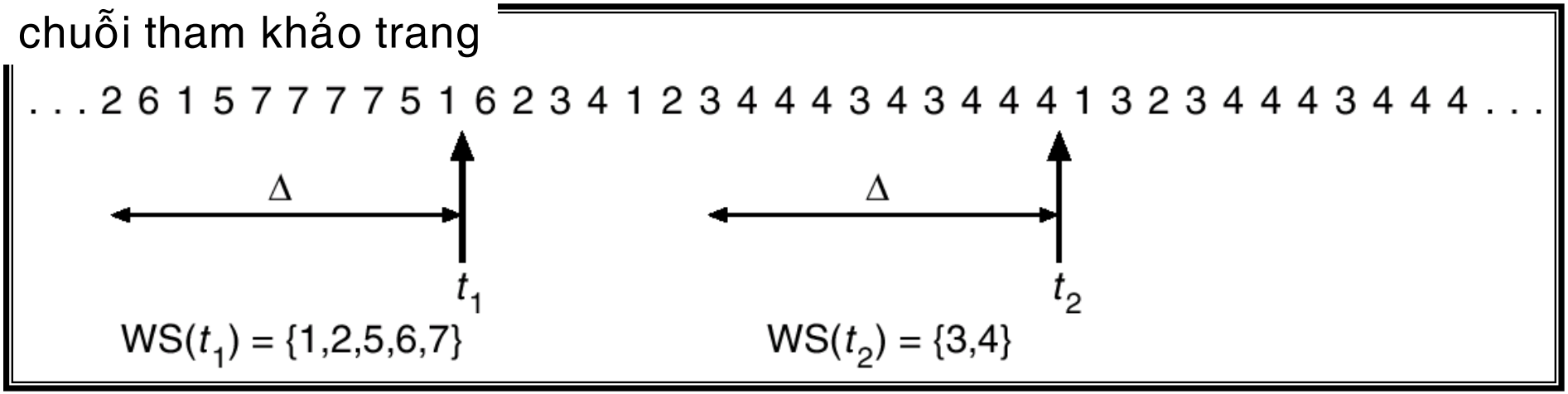
Ví dụ:



b) Giải pháp tập làm việc (working set)

- Định nghĩa: *working set của process P_i* , ký hiệu WS_i , là tập gồm Δ các trang được sử dụng gần đây nhất.

Ví dụ: Δ = 10 và chuỗi tham khảo trang



- Nhận xét:

  Δ quá nhỏ $\Rightarrow$ không đủ bao phủ toàn bộ locality.

  Δ quá lớn $\Rightarrow$ bao phủ nhiều locality khác nhau.

  $\Delta = \infty$ $\Rightarrow$ bao gồm tất cả các trang được sử dụng.

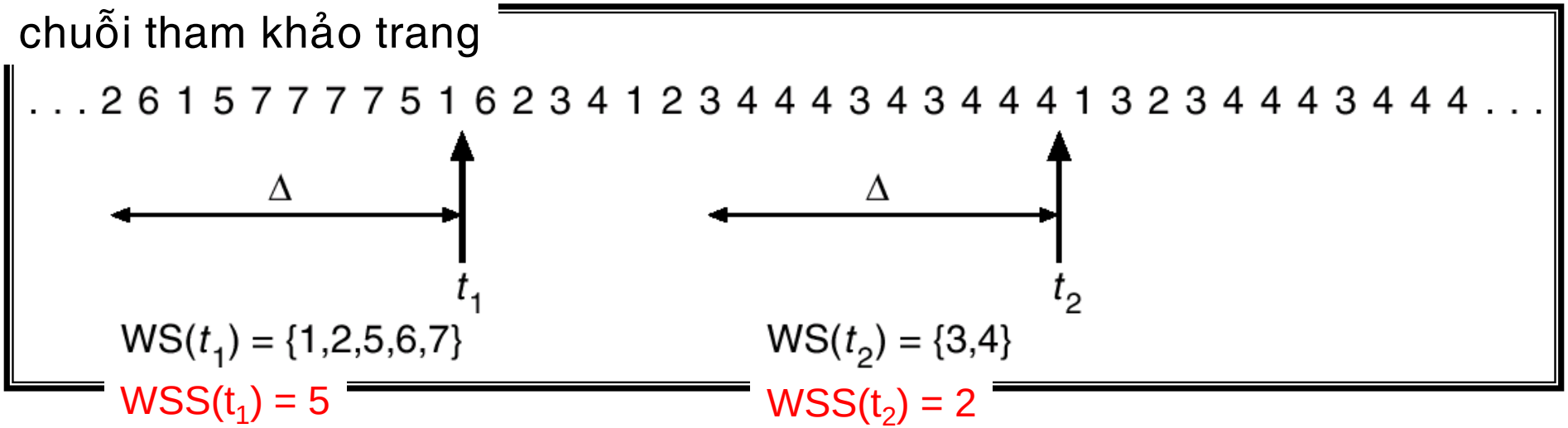
  Dùng working set của một process để xấp xỉ locality của nó.

b) Giải pháp tập làm việc (working set)

Định nghĩa WSS_i là kích thước của working set của P_i :

WSS_i = số lượng các trang trong WS_i

Ví dụ (tiếp): $\Delta = 10$ và chuỗi tham khảo trang



b) Giải pháp tập làm việc (working set)

Đặt D = $\sum WSS_i$ = tổng các working-set size của mọi process trong hệ thống.

- Nhận xét: Nếu D > m (số frame của hệ thống) $\Rightarrow$ sẽ xảy ra thrashing.

- *Giải pháp working set:*
  - Khi khởi tạo một quá trình: cung cấp cho quá trình số lượng frame thỏa mản working-set size của nó.
  - Nếu D > m $\Rightarrow$ tạm dừng một trong các process.
    - Các trang của quá trình được chuyển ra đĩa cứng và các frame của nó được thu hồi.

b) Giải pháp tập làm việc (working set)

- WS loại trừ được tình trạng trì trệ mà vẫn đảm bảo mức độ đa chương
- Theo vết các WS? => WS xấp xỉ (đọc thêm trong sách)

Đọc thêm:

- Hệ thống tập tin
- Hệ thống nhập xuất
- Hệ thống phân tán

Bài tập

- Bài 01: Một máy tính 32-bit địa chỉ, sử dụng một bảng trang nhị cấp. Địa chỉ ảo được phân bổ như sau : 9 bit dành cho bảng trang cấp 1, 11 bit cho bảng trang cấp 2, và cho offset. Cho biết kích thước một trang trong hệ thống, và địa chỉ ảo có bao nhiêu trang ?

- Bài 02: Xét chuỗi truy xuất bộ nhớ sau:

  1, 2 , 3 , 4 , 3 , 5 , 1 , 6 , 2 , 1 , 2 , 3 , 7 , 5 , 3 , 2 , 1 , 2 , 3 , 6

  Có bao nhiêu lỗi trang xảy ra khi sử dụng các thuật toán thay thế sau đây, giả sử có 4 khung trang và ban đầu các khung trang đều trống ?

  a) LRU

  b) FIFO

  c) Optimal

  d) Cơ hội thứ 2

Bài tập

- Cho một process có số frame truy cập như sau:

  1,2,4,3,5,2,3,5,6,7,8,9,1,2,3,4,5,2,3,7,6,4,5,6,7,9,8,1,2,5

- Cho biết có biêu nhiêu tập làm việc (working set)?
- Liệt kê các frame trong từng working set?
- Biết $\Delta = 7$.

Bài tập

- Hệ thống tại một thời điểm có 5 process: process 1 cần 5 frame, process 2 cần 15 frame, process 3 cần 10 frame, process 4 cần 9 frame, process 5 cần 11 frame. Hệ thống hiện có 30 frame. Tính xem mỗi process được cấp bao nhiêu frame?
- Phân bổ đều?
- Theo tỷ lệ?