HỆ ĐIỀU HÀNH
(OPERATING SYSTEM)

Trình bày: Nguyễn Hoàng Việt
Khoa Công Nghệ Thông Tin
Đại Học Cần Thơ

Chương 8: Bộ nhớ ảo (Virtual Memory)

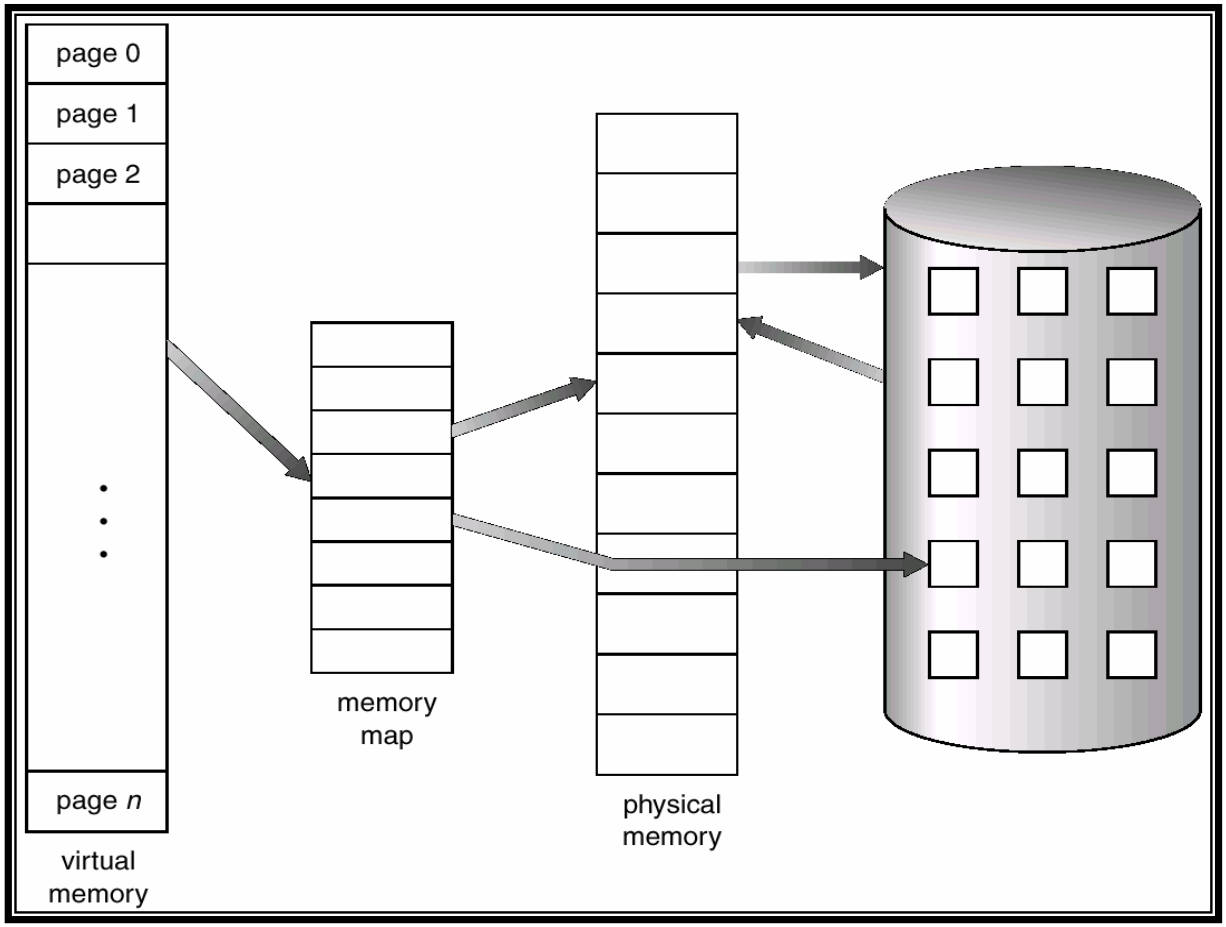
- Tổng quan
- Phân trang theo yêu cầu
- Tạo quá trình
- Thay thế trang
- Cấp các khung trang cho quá trình như thế nào
- Thrashing
- Các ví dụ từ các hệ điều hành cụ thể

Tổng quan (1)

- **Bộ nhớ ảo:** tách biệt khái niệm bộ nhớ luận lý của người dùng ra khỏi khái niệm bộ nhớ vật lý.
  - Chỉ một phần của chương trình cần ở trong bộ nhớ để thực thi ⇨ không gian địa chỉ luận lý có thể lớn hơn nhiều so với không gian địa chỉ vật lý.
  - Giải phóng sự ràng buộc với giới hạn của bộ nhớ thực.
  - Điều này cho phép không gian địa chỉ có thể được chia sẻ giữa nhiều quá trình.
  - Cho phép việc tạo quá trình trở nên hiệu quả hơn.
- Bộ nhớ ảo có thể được cài đặt thông qua:
  - Phân trang theo yêu cầu
  - Phân đoạn theo yêu cầu

Tổng quan (2)

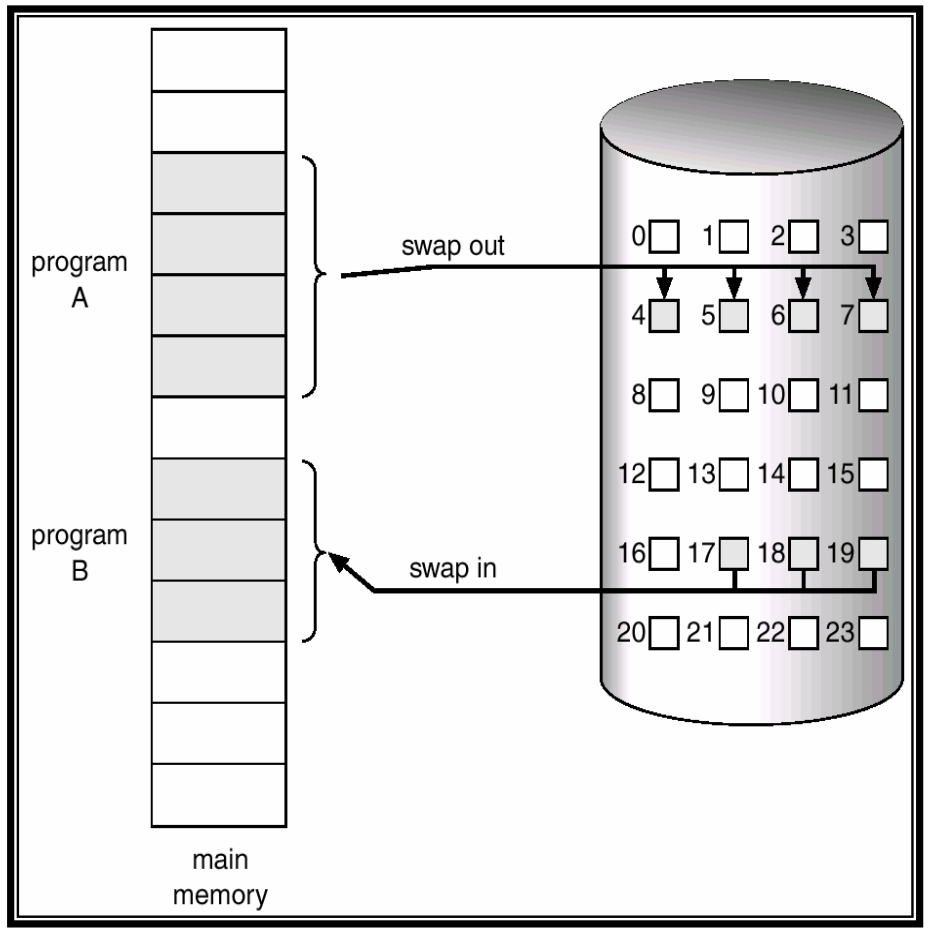
Bộ nhớ ảo thì lớn hơn nhiều so với bộ nhớ vật lý



Phân trang theo yêu cầu (1)
Demand Paging

Chuyển bộ nhớ phân trang vào các khối đĩa liền nhau

- Chỉ mang một trang vào bộ nhớ khi cần thiết.
  - Cần ít thao tác I/O hơn
  - Cần ít bộ nhớ vật lý hơn
  - Đáp ứng yêu cầu về bộ nhớ nhanh hơn
  - Phục vụ nhiều người dùng hơn
- Khi cần một trang ⇒ tham khảo tới nó
  - Tham khảo không hợp lệ ⇒ thoát
  - Trang không có trong bộ nhớ ⇒ đem nó vào bộ nhớ

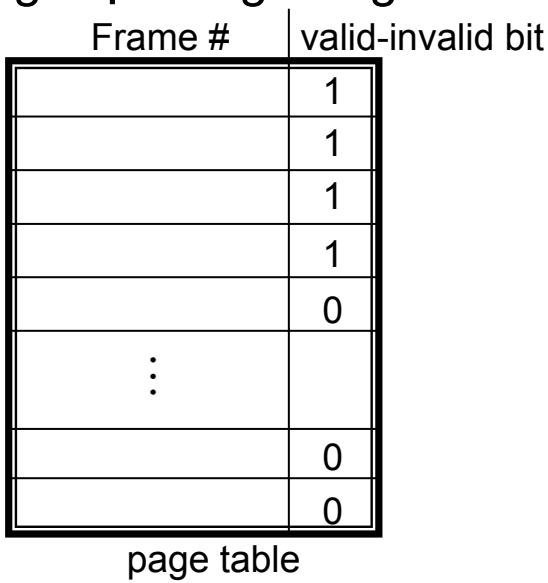


Phân trang theo yêu cầu (2)
Demand Paging (tt)

- Không gian địa chỉ luận lý/bộ nhớ ảo (logical address space) được phân thành các trang (page).
- Không gian địa chỉ thực/bộ nhớ thực (physical address space) được phân thành các khung trang hay khung (page frame/ frame).
- Kích thước của trang và khung trang là bằng nhau.
- Số lượng trang có thể lớn hơn số lượng khung trang rất nhiều ⇨ kích thước của bộ nhớ ảo có thể lớn hơn rất nhiều so với bộ nhớ thực
- Khi một trang được nạp vào một khung trang, trang này được ánh xạ vào khung trang tương ứng. Điều này được quản lý thông qua một bảng trang (page table).

Phân trang theo yêu cầu (3)
Bit hợp lệ - không hợp lệ (Valid/Invalid Bit)

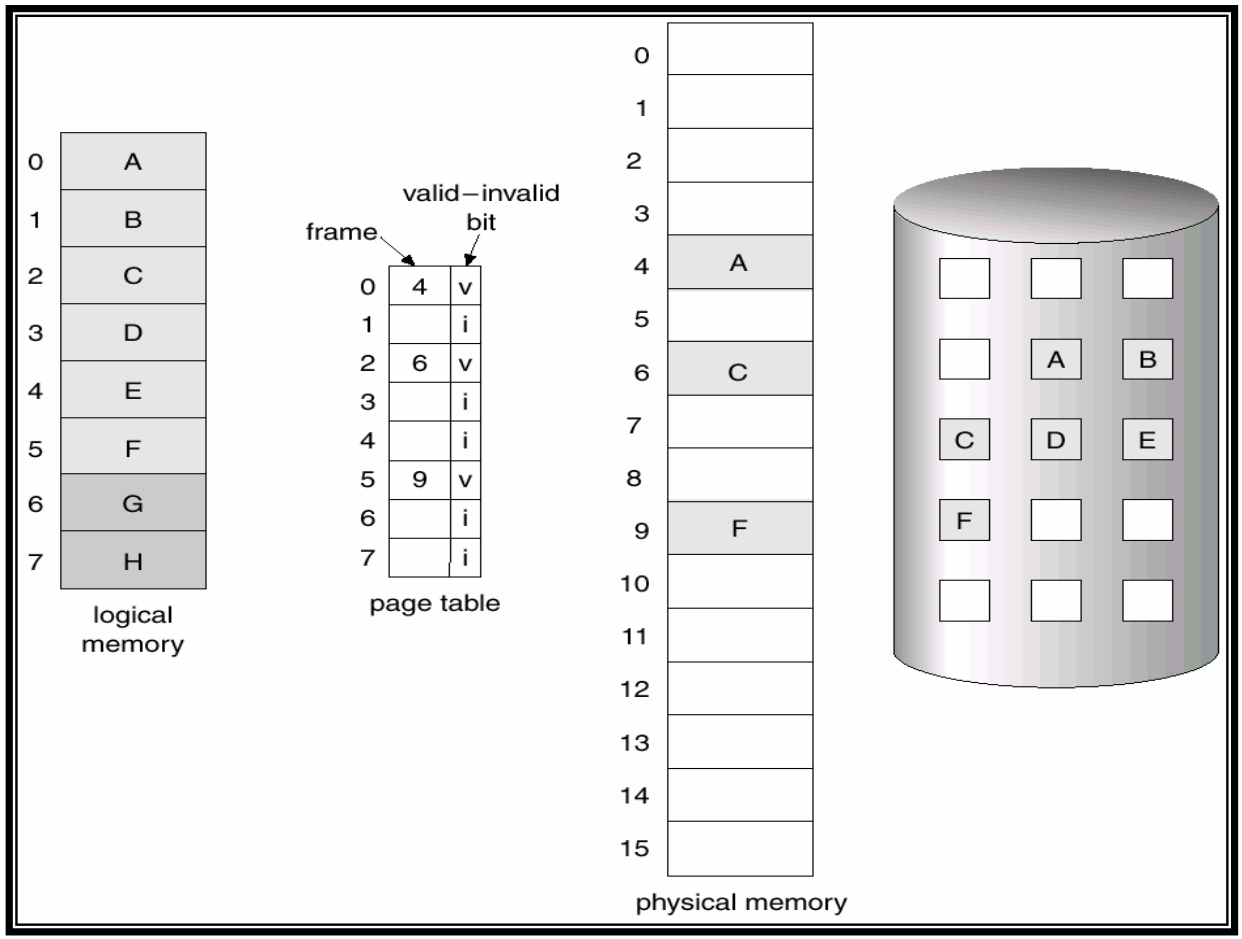
- Kết hợp với mỗi mục từ trong bảng trang một bit valid/invalid (1 ⇒ in-memory, 0 ⇒ not-in-memory hoặc invalid access)
- Khởi đầu bit valid/invalid của mọi mục từ được đặt là 0.
- Ví dụ về thực trạng một bảng trang:



- Trong quá trình dịch địa chỉ, nếu bit valid/invalid trong một mục từ của bảng trang = 0 ⇒ có lỗi về trang (Page Fault).

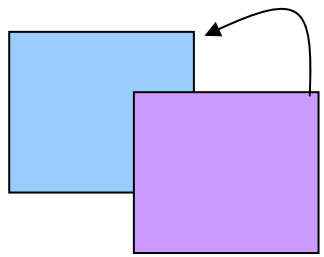
Phân trang theo yêu cầu (4)

Bảng trang có một số trang không nằm trong bộ nhớ

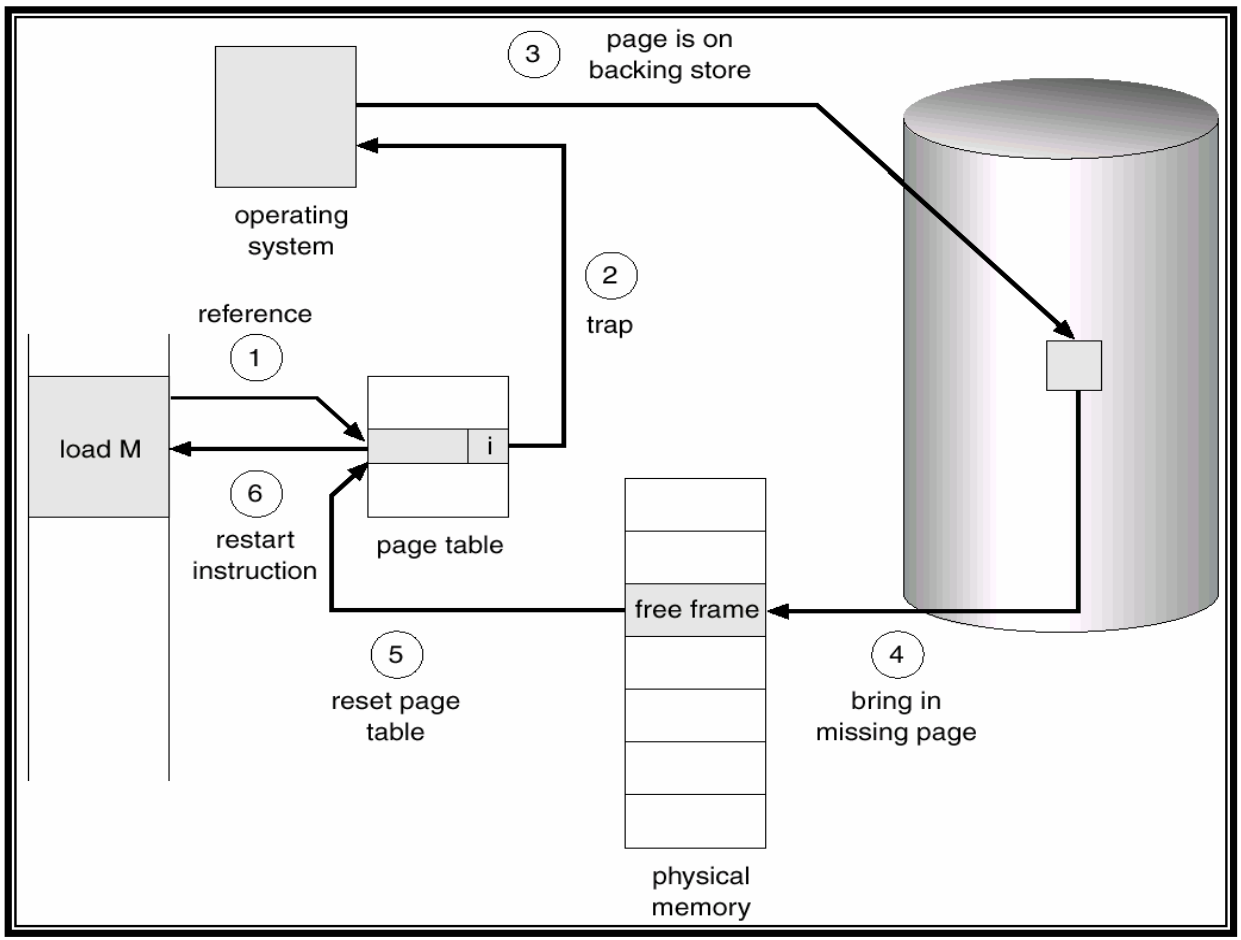


Phân trang theo yêu cầu (5)
Lỗi về trang (Page Fault)

- Tham khảo đầu tiên đến một trang luôn được bắt giao cho hệ điều hành xử lý $\Rightarrow$ lỗi về trang (page fault).
- Hệ điều hành sẽ nhìn vào internal table (chứa trong PCB) để xác định:
  - Tham khảo không hợp lệ $\Rightarrow$ ngừng (dựa vào PCB).
  - Hay là trang không có trong bộ nhớ.
- Lấy một khung trang còn trống.
- Chuyển trang đến khung trang.
- Thiết lập lại bảng trang bằng cách đặt bit kiểm tra = 1.
- Khởi động lại chỉ thị đã bị ngắt bởi trap
- Tiêu chí để thay thế: Least Recently Used
  - Chuyển khối
  - Vị trí tự tăng giảm



Phân trang theo yêu cầu (6)
Các bước xử lý khi có lỗi về trang



Phân trang theo yêu cầu (7)
Điều gì xảy ra nếu không còn khung trang còn trống?

- Thay thế trang: tìm trang nào đó có trong bộ nhớ nhưng không thực sự là đang được sử dụng, chuyển nó ra đĩa!
  - Giải thuật
  - Hiệu suất: mong muốn một giải thuật mà nó sẽ làm cho số lỗi về trang là thấp nhất.
- Có thể một trang sẽ được đem ra/vào bộ nhớ vài lần

Phân trang theo yêu cầu (8)
Hiệu suất của phân trang theo yêu cầu

- Tỉ lệ lỗi trang $0 \leq p \leq 1.0$
  - if $p = 0$ không có lỗi về trang
  - if $p = 1$, mọi tham khảo trang đều bị lỗi
- Thời gian truy xuất hiệu quả (Effective Access Time - EAT)

  EAT = (1 – *p*) x [memory access]
  + *p* x [page fault time]
- [page fault time] = [page fault overhead]
  + [swap page out]
  + [swap page in]
  + [restart overhead]

Phân trang theo yêu cầu (9)
Ví dụ về phân trang theo yêu cầu

- Memory Access Time = 100 ns
- Average page-fault time = 25 ms = 25,000,000 ns

  EAT = (1 – p) x 100 + p x 25,000,000

  = 100 + 24,999,900 x p

  EAT tỉ lệ trực tiếp với page-fault time
- Nếu p = 0.001 (tức là truy xuất 1000 trang sẽ gây ra 1 lỗi trang), EAT sẽ là 25 ms.

Tạo quá trình (1)

- Bộ nhớ ảo tạo ra nhiều tiện lợi trong việc tạo quá trình. Có 2 giải pháp tạo quá trình sử dụng bộ nhớ ảo:
  - Copy-on-Write
  - Memory-Mapped Files

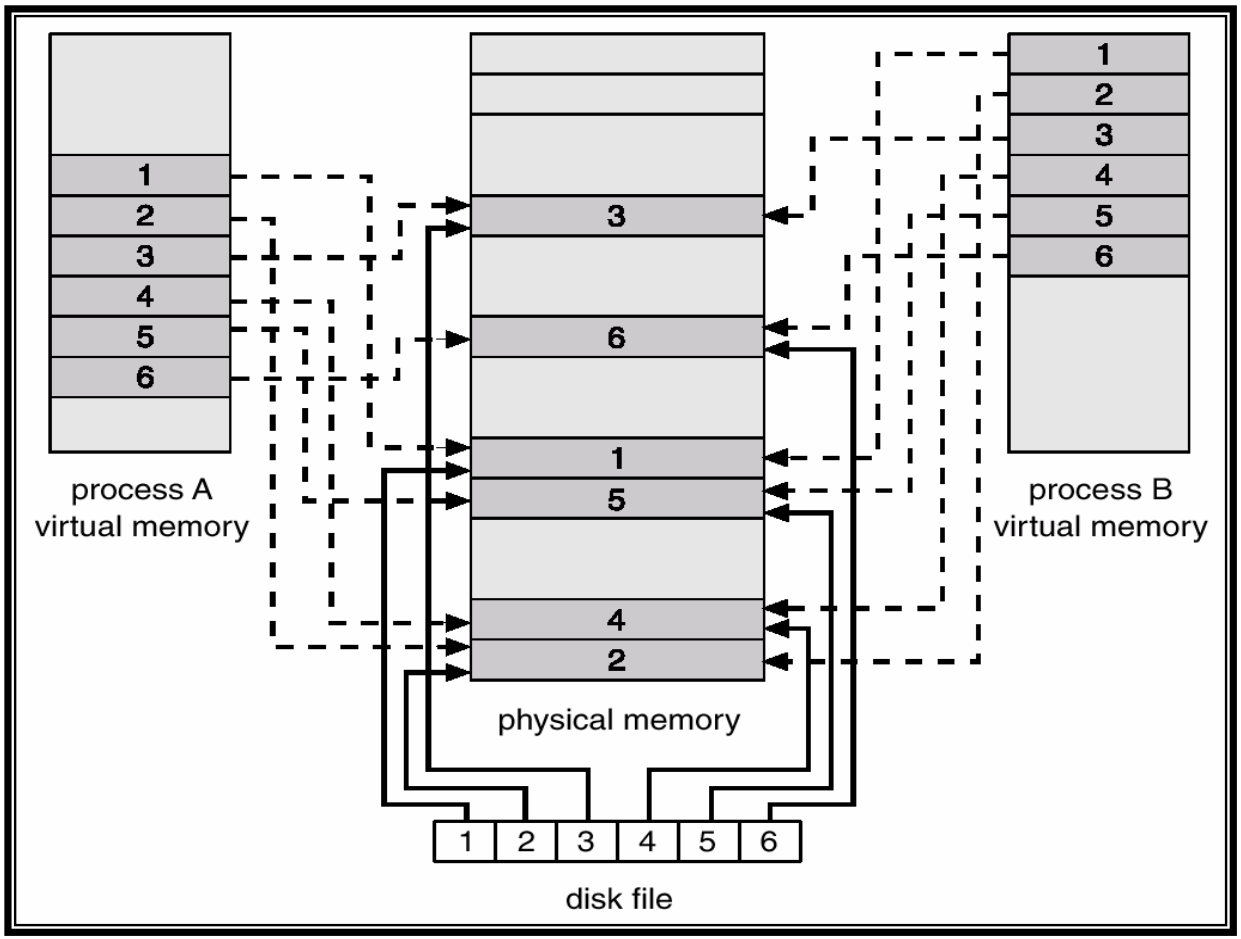
Tạo quá trình (2)
Copy-on-Write

- Copy-on-Write (COW) cho phép cả quá trình cha và con ban đầu chia sẻ cùng các trang trong bộ nhớ.
- Nếu một quá trình cha hoặc con sửa đổi một trang được chia sẻ, thì chỉ trang đó được sao chép thành một bản mới. Trang mới này sẽ được đưa vào không gian địa chỉ của quá trình đã đã sửa đổi nó.
- COW cho phép việc tạo quá trình hiệu quả hơn bởi vì nó chỉ sao chép các trang bị sửa đổi.
- Được dùng bởi các nhiều HĐH, bao gồm Windows XP, Linux, Solaris.

Tạo quá trình (3)
Memory-Mapped Files

- Giải pháp Memory-mapped file I/O cho phép việc đọc ghi lên tập tin được xem như thao tác đọc ghi bộ nhớ bằng cách ánh xạ một khối đĩa đến một trang trong bộ nhớ.
- Đầu tiên, một tập tin được đọc bằng cách sử dụng giải pháp phân trang theo yêu cầu. Một phần có kích thước bằng một trang của tập tin được đọc từ hệ thống tập tin vào trong trang vật lý. Sau đó các thao tác đọc ghi lên tập tin được coi như các thao tác đọc ghi bộ nhớ thông thường.
- Đơn giản hóa việc truy xuất tập tin bằng cách xử lý vào ra tập tin thông qua bộ nhớ hơn là bằng cách gọi các lời gọi hệ thống **read(), write()**.
- Cũng cho phép nhiều quá trình ánh xạ cùng một file, do việc chia sẻ các trang trong bộ nhớ.

Việc tạo quá trình (4)
Memory-Mapped Files



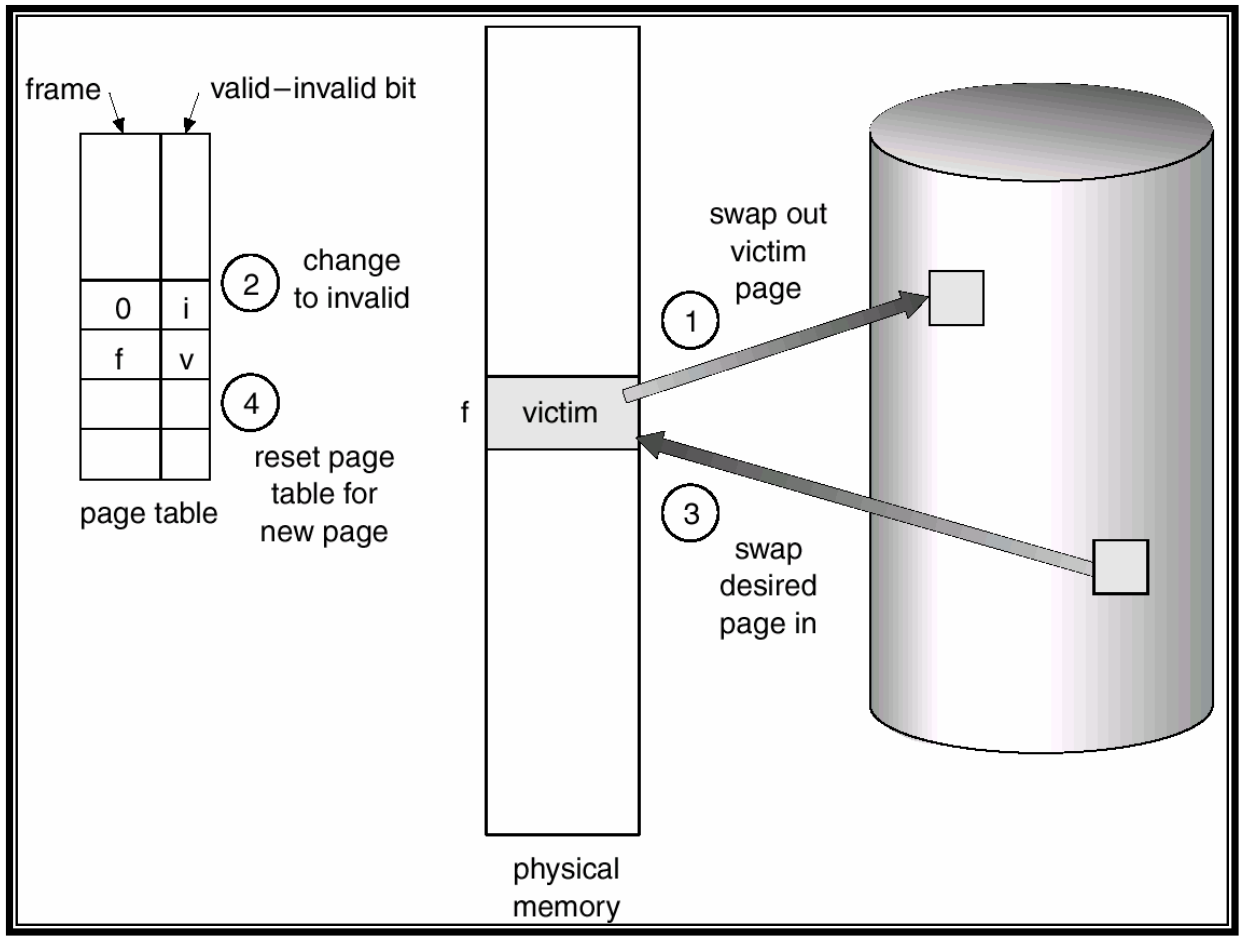
Thay thế trang (1)
Page Replacement

- Cấp phát vượt quá dung lượng bộ nhớ
  - Không còn khung trang trống để nạp một trang từ dĩa vào bộ nhớ.
- Ngăn chặn việc cấp phát bộ nhớ vượt quá mức bằng cách dùng giải pháp thay thế trang.
  - Chọn một trang đã được nạp và không đang được dùng, giải phóng nó bằng cách viết nội dung của nó vào không gian hoán chuyển (swap space).
  - Dùng khung đã được giải phóng này để nạp trang được yêu cầu.
- Sử dụng bit **modify (dirty)** để làm giảm chi phí cho việc chuyển trang ra đĩa.
- Giải pháp thay thế trang hoàn thiện thêm công việc tách biệt hóa bộ nhớ ảo và bộ nhớ vật lý:
  - Bộ nhớ ảo có thể lớn hơn bộ nhớ thực
  - Thực hiện chương trình trong một vài khung, và tìm một khung trống mỗi khi cần thiết.

Thay thế trang (2)
Chiến lược thay thế trang cơ bản

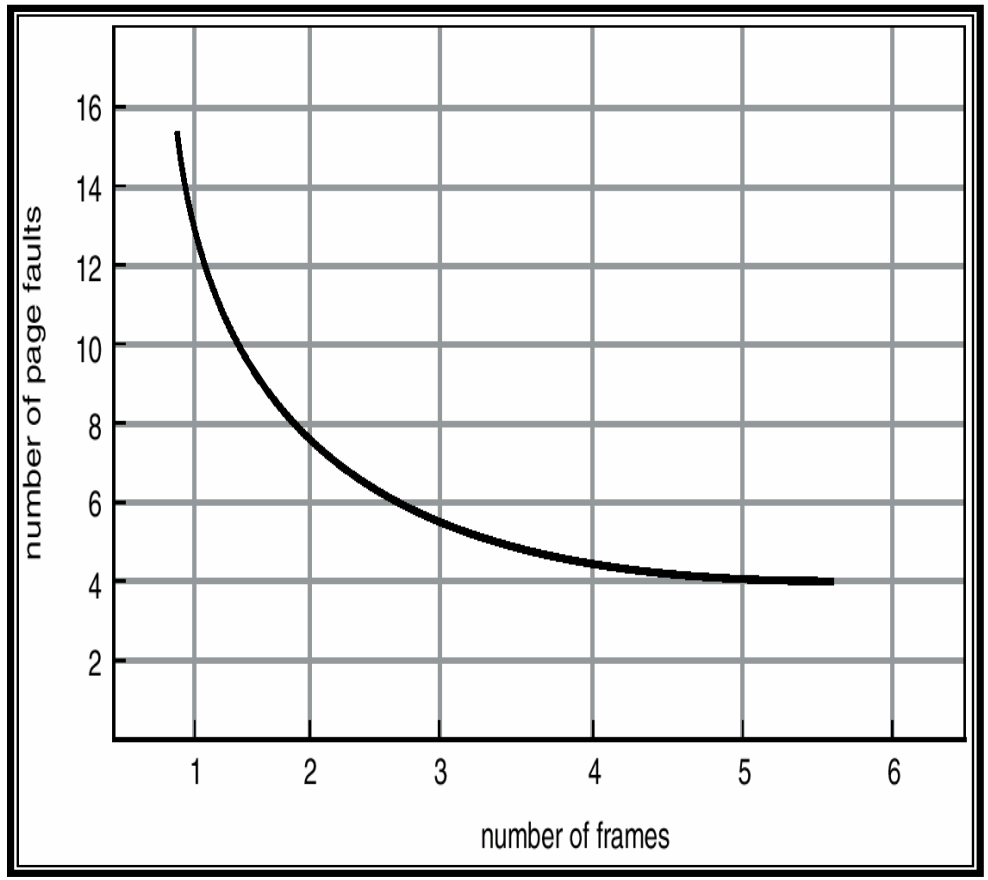
1. Tìm vị trí của trang mong muốn trên đĩa.
2. Tìm một khung trang còn trống:
   - Nếu còn khung trang trống, sử dụng nó.
   - Nếu không còn khung trang trống, sử dụng giải thuật thay thế trang để tìm ra khung trang *nạn nhân*.
3. Đọc trang mong muốn vào khung trang trống mới tìm ra. Cập nhật lại trang và các bảng trang.
4. Tiếp tục quá trình.

Thay thế trang (3)
Sơ đồ thay thế trang



Thay thế trang (4)
Các giải thuật thay thế trang

- Mong muốn tỉ lệ lỗi trang thấp nhất.
- Đánh giá giải thuật bằng cách chạy nó trên một chuỗi tham khảo bộ nhớ cụ thể và tính toán số lỗi về trang phát sinh.
- Trong hình, khi số khung tăng lên, số lượng lỗi trang giảm xuống một mức độ tối thiểu.



Thay thế trang (5)
Giải thuật First-In-First-Out (FIFO)

- Chuỗi tham khảo là: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
- 3 khung trang (3 trang có thể nằm trong bộ nhớ cho mỗi quá trình)

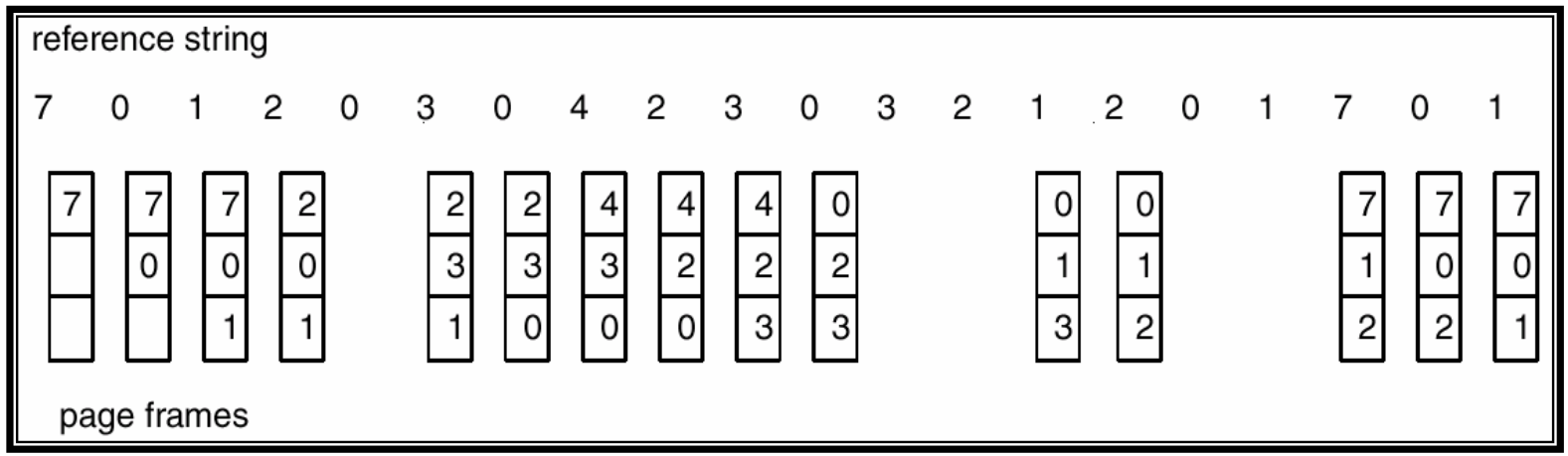
1	1	4	5	
2	2	1	3	9 page faults
3	3	2	4	

- 4 frames

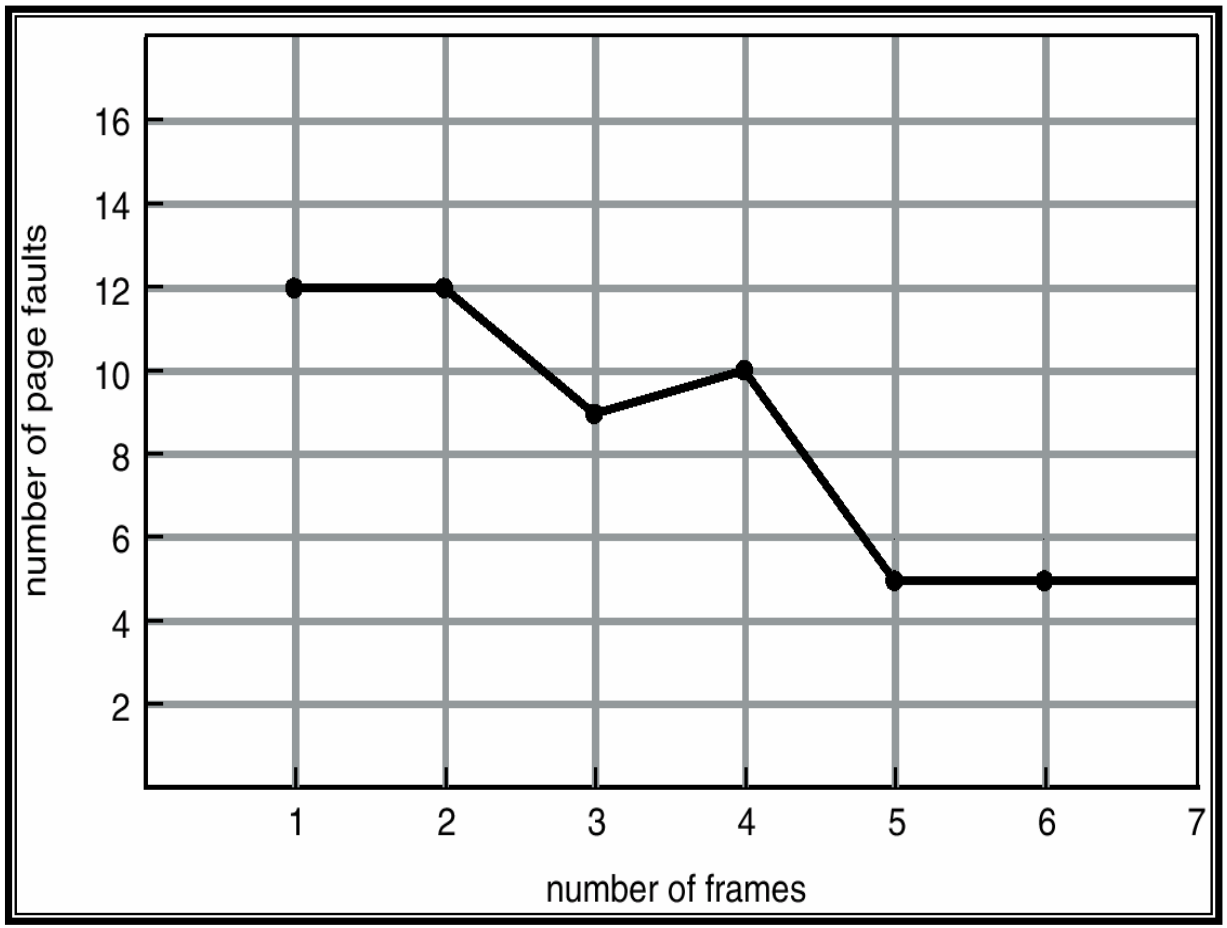
1	1	5	4	
2	2	1	5	10 page faults
3	3	2		
4	4	3		

- Thay thế trang kiểu FIFO Replacement – Belady's Anomaly
  - Càng nhiều khung trang $\Rightarrow$ càng nhiều lỗi về trang

Thay thế trang (6)
Thay thế trang kiểu FIFO



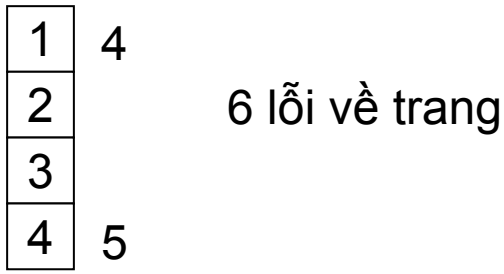
Thay thế trang (7)
Mô phỏng Belady's Anamoly cho FIFO



Thay thế trang (8)
Giải thuật tối ưu (Optimal)

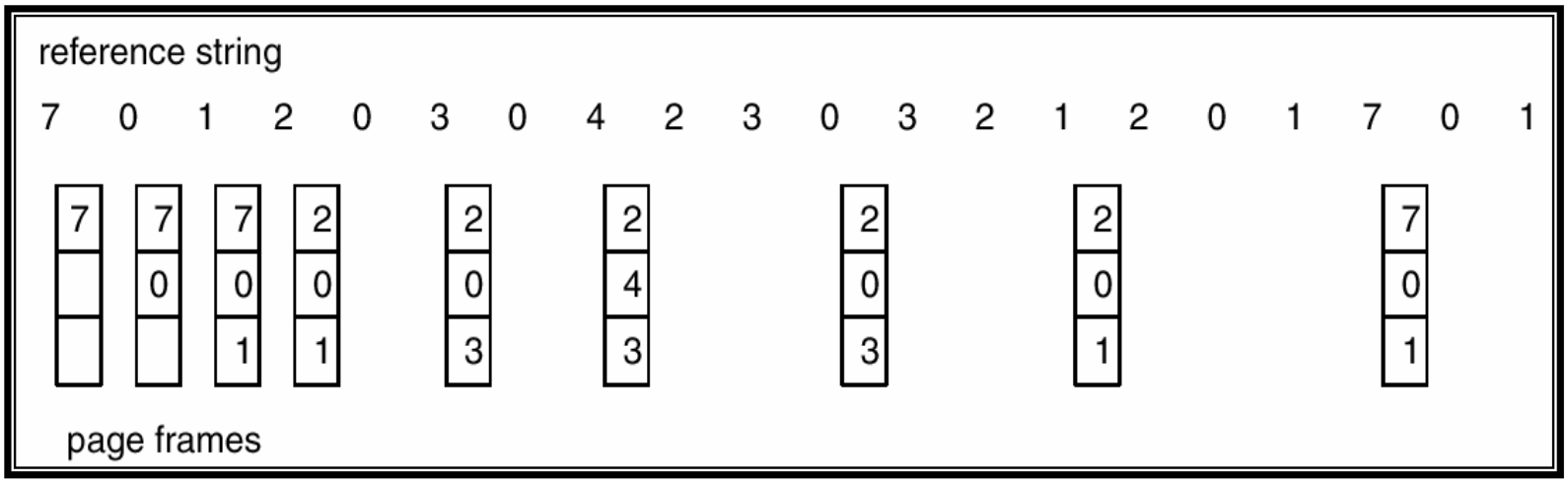
- Thay thế trang sẽ không được sử dụng trong một thời gian dài.
- 4 frames example

  1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5



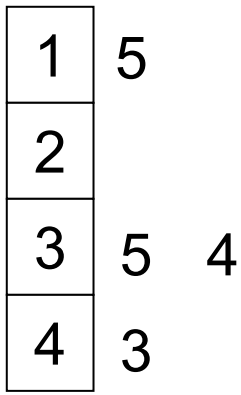
- Bảo đảm tỉ lệ lỗi trang thấp nhất có thể cho một số lượng khung trang cố định
- Khó cài đặt – đòi hỏi phải biết chuỗi tham khảo trang trong tương lai ⇨ Thường được dùng để nghiên cứu so sánh, đo lường các giải thuật.

Thay thế trang (9)
Thay thế trang tối ưu



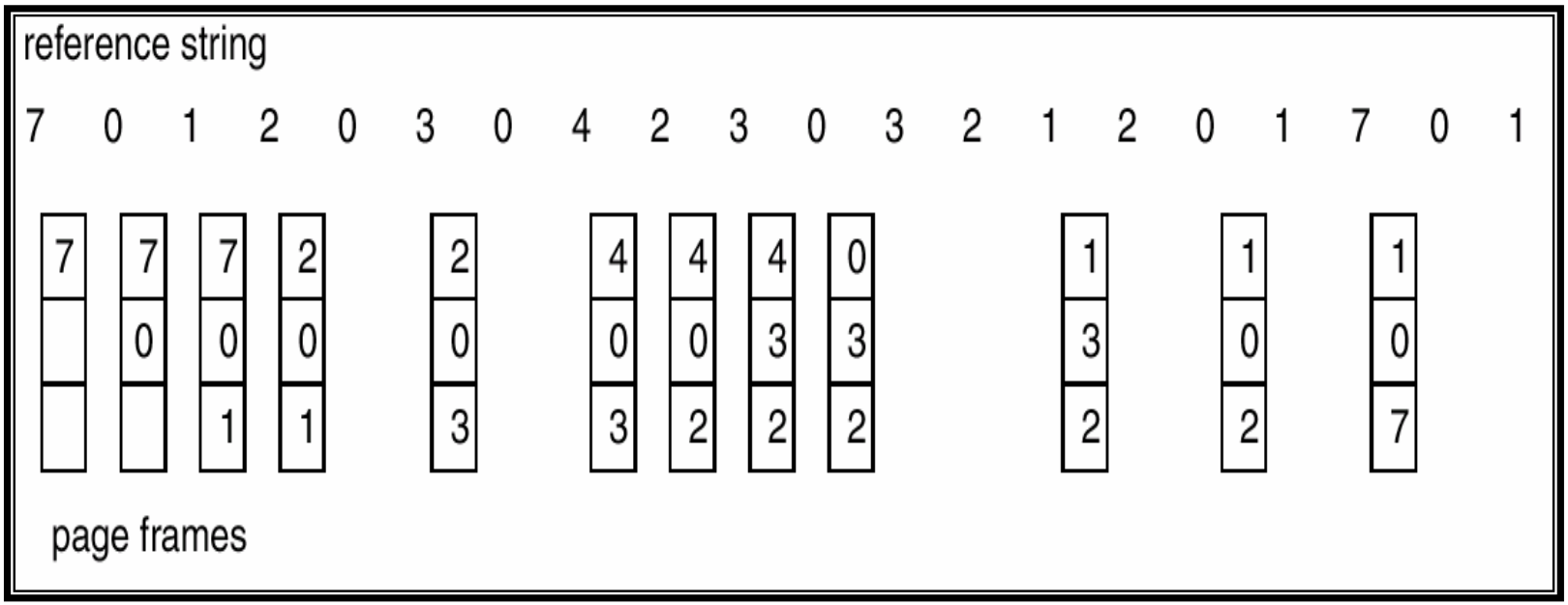
Thay thế trang (10)
Giải thuật Least Recently Used (LRU)

- Thay thế trang không được dùng trong một khoảng thời gian dài nhất
- Chuỗi tham khảo trang: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5



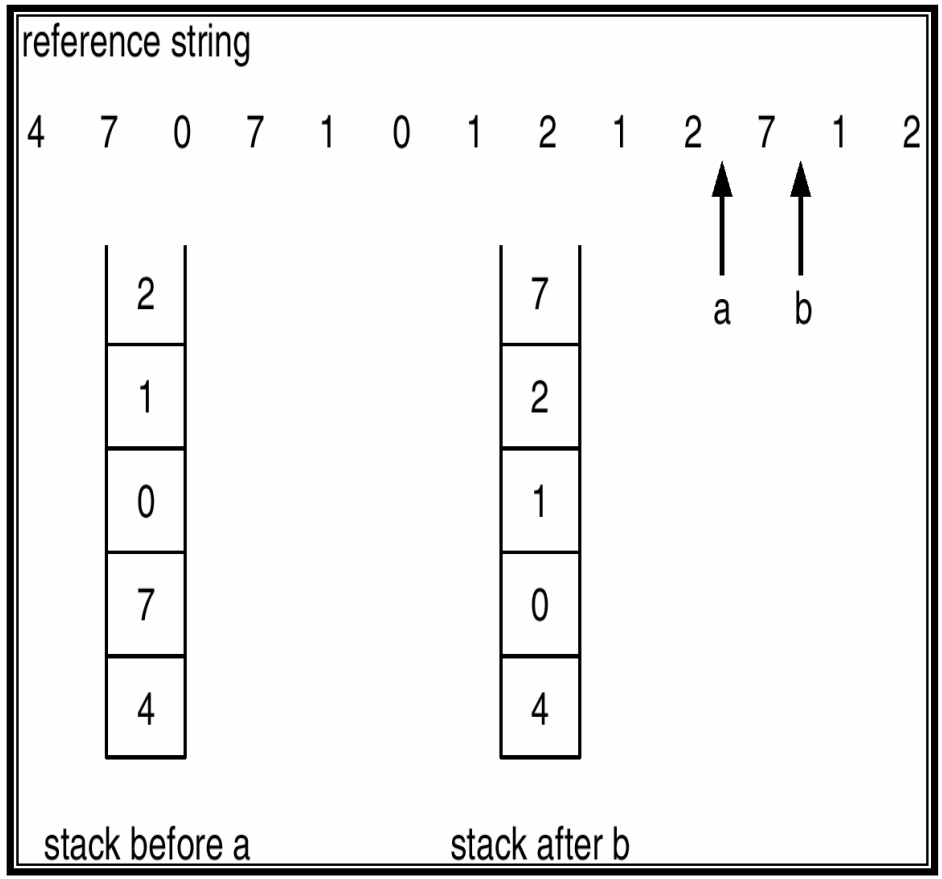
- Cài đặt bộ đếm
  - Mỗi một mục từ trong bảng trang có một bộ đếm (time-of-used field) , mỗi khi trang được tham khảo đến, chép nội dung của đồng hồ hệ thống vào bộ đếm này.
  - Khi một trang cần được thay đổi, nhìn vào các bộ đếm để quyết định trang sẽ được thay đổi.

Thay thế trang (11)
Thay thế trang theo kiểu LRU



Thay thế trang (12)
Giải thuật LRU – Sử dụng stack

- Cài đặt stack – tạo một stack chứa các số thứ tự trang:
  - Khi một trang được tham khảo:
    - ✓ Di chuyển số thứ tự của nó lên đỉnh của stack.
  - Không phải tìm kiếm để thay thế trang.

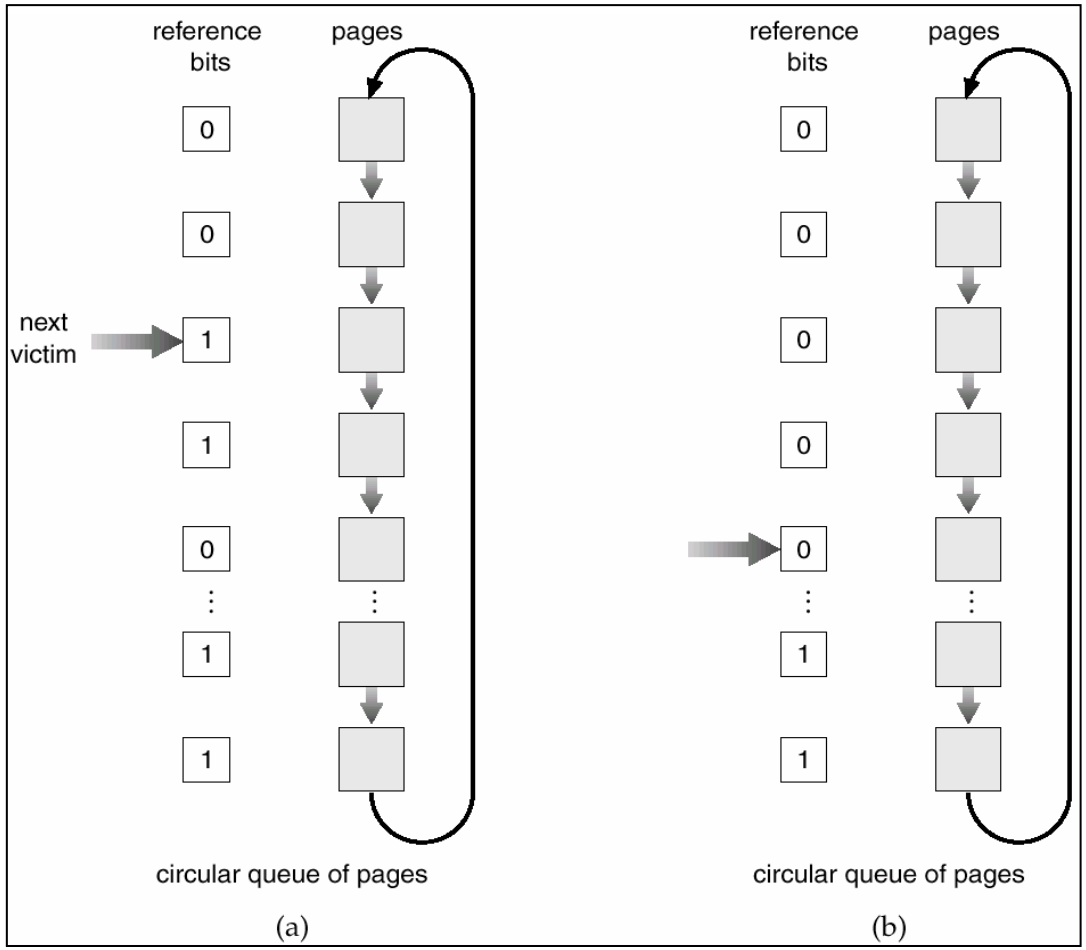


Thay thế trang (13)
Giải thuật LRU dùng Reference Bit

- Bit tham khảo (Reference Bit - RB)
  - Bit tham khảo được đặt bởi phần cứng.
  - Mỗi trang được kết hợp với 1 bit, khởi đầu là 0.
  - Khi trang được tham khảo, bit này được đặt là 1.
  - Thực hiện thế trang có bit này là 0 (nếu có).
  - Tuy nhiên chúng ta không biết thứ tự.
- Giải thuật thêm vào bit tham khảo
  - Ghi lại các bit tham khảo theo những thời gian đều đặn
  - Dùng mục từ 8 bit (8-bits entry) cho mỗi trang.
    - ✓ 00000000 : trang không được dùng trong 8 chu kỳ.
    - ✓ 11111111 : trang đã được dùng tại ít nhất một trong 8 chu kỳ.
    - ✓ Trang với số thấp nhất là trang LRU.

Thay thế trang (14)
Giải thuật Second- Chance

- Kiểm tra bit tham khảo của trang được chọn
- Nếu bit là 0, thay thế trang
- Nếu trang (theo thứ tự giờ) có bit này bằng 1, cho nó 1 số phận thứ 2 (second-chance)
  - Đặt bit là 0 và thời gian đến của nó là thời gian hiện hành.
  - Để lại trang trong bộ nhớ
  - Thay thế trang kế tiếp (theo thứ tự thời gian)
- Cài đặt như một hàng đợi vòng tròn như trong hình.



Thay thế trang (15)
Các giải thuật đếm (Counting Algorithms)

- Tạo bộ đếm đếm số lần một trang được truy cập.
- Giải thuật LFU (Least Frequently Used): thay thế trang với giá trị bộ đếm nhỏ nhất.
- Giải thuật MFU (Most Frequently Used): dựa trên lý luận là một trang có giá trị bộ đếm là nhỏ nhất thì mới được đem vào bộ nhớ và sẽ còn được sử dụng nhiều hơn sau này.

Cấp phát khung trang (1)

- Có hai chiến lược cấp phát khung trang chính:
  - Cấp cố định
  - Cấp theo độ ưu tiên

Cấp phát khung trang (2)
Cấp cố định (Fixed Allocation)

- Cấp công bằng - ví dụ: nếu có 100 khung trang và 5 quá trình, thì cấp cho mỗi quá trình 20 khung trang.
- Cấp tương xứng – cấp khung trang dựa trên kích cỡ của quá trình.

s_i = size of process p_i

$S = \sum s_i$

m = total number of frames

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

$m = 64$

$s_i = 10$

$s_2 = 127$

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

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

Cấp phát khung trang (3)
Cấp theo độ ưu tiên (Priority Allocation)

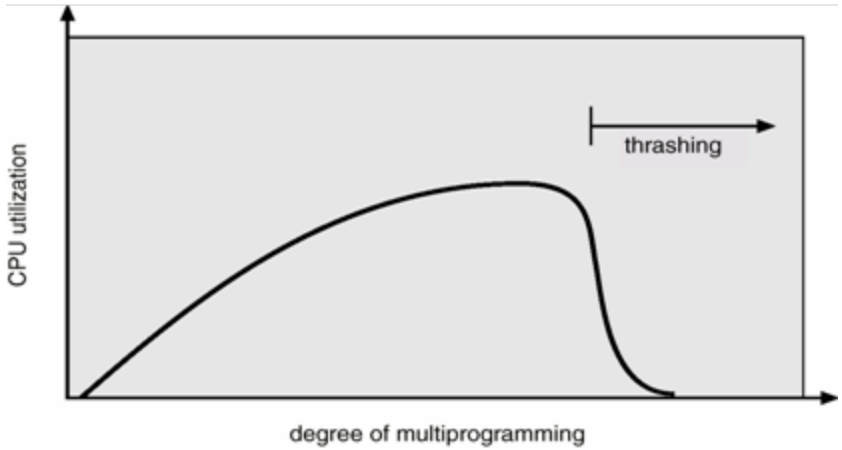
- Sử dụng sơ đồ cấp phát tương xứng, nhưng thay vì sử dụng kích cỡ của quá trình, ta dùng độ ưu tiên.
- Nếu quá trình P_i sinh ra lỗi về trang thì
  - Chọn một khung trang của chính quá trình này làm nạn nhân (thay thế cục bộ - local replacement)
  - Chọn một khung trang của quá trình khác có độ ưu tiên thấp hơn làm nạn nhân (thay thế toàn cục – global replacement).

Thrashing (1)

- Nếu một quá trình không có đủ các trang theo yêu cầu thì số lượng lỗi về trang sẽ rất lớn.
  - Bởi vì các trang trong trạng thái đang dùng, bất kỳ lỗi trang nào cũng dẫn đến thay thế một trang cần lại ngay sau đó.
  - Quá trình tiếp tục lỗi, thay thế các trang mà sẽ lại bị lỗi và phải được mang vào trở lại ngay sau đó.
- **Thrashing** ≡ quá trình luôn bận rộn cho việc chuyển các trang ra và vào.
  - Dùng nhiều thời gian cho lỗi trang hơn cho thực thi
- Điều này sẽ dẫn đến:
  - Hiệu năng sử dụng CPU thấp.
  - Hệ điều hành nghĩ rằng nó cần phải tăng mức độ đa chương lên bởi vì bộ định thời CPU thấy rằng việc sử dụng CPU thấp.
  - Quá trình khác được thêm vào hệ thống ⇨ mỗi quá trình nhận ít khung trang hơn ⇨ lỗi trang càng nhiều hơn.

Thrashing (2)

- Khi mức độ đa chương tăng lên, việc sử dụng CPU cũng tăng lên, mặc dù chậm hơn, đến một ngưỡng cực đại.
- Nếu mức độ đa chương tiếp tục tăng, thrashing sẽ xảy ra.



- Giải thuật thế trang cục bộ có thể giới hạn ảnh hưởng của thrashing
  - Nếu một quá trình bắt đầu thrashing, nó không được lấy các khung từ các quá trình khác và gây ra thrashing cho quá trình sau.
  - Nhưng một quá trình thrashing có thể ảnh hưởng đến các quá trình không thrashing vì nó làm chậm đi hàng đợi thiết bị phân trang.

Thrashing (3)
Ngăn ngừa Thrashing

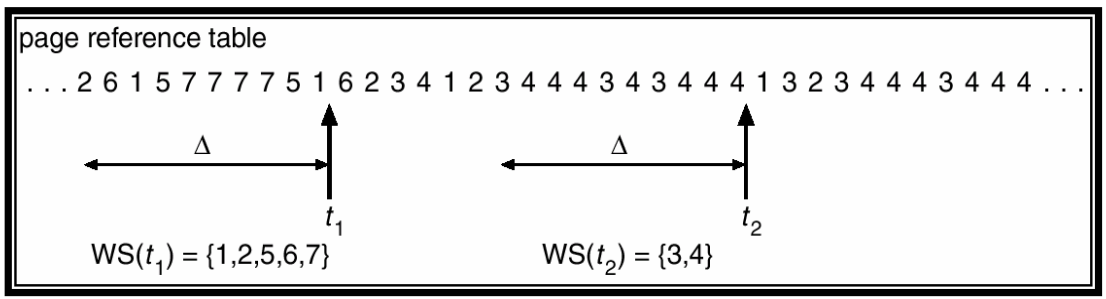
- Để ngăn ngừa thrashing, chúng ta phải cung cấp cho quá trình số khung mà nó cần
  - Vấn đề là làm sao biết được số khung mà quá trình cần
  - Một vài kỹ thuật được sử dụng
- Hai giải pháp ngăn ngừa thrashing
  - Mô hình tập làm việc (Working set model)
  - Sơ đồ tần suất lỗi trang (Page-fault frequency scheme)

Thrashing (4)
Mô hình tập làm việc (Working-Set Model)

- Mô hình tập làm việc bắt đầu bằng cách xem xét có bao nhiêu khung một quá trình đang dùng.
  - Phương pháp này định nghĩa mô hình cục bộ (locality model) của việc thực thi quá trình.
  - Một **cục bộ** là tập hợp các trang được dùng cùng với nhau.
  - Khi một quá trình thực thi, nó sẽ di chuyển từ cục bộ này sang cục bộ khác.
  - Một chương trình nói chung được tạo thành từ một vài cục bộ khác nhau, có thể phủ lấp.
- Giả sử cấp phát khung đủ cho một cục bộ.
  - Quá trình sẽ lỗi (fault) cho các trang của nó đến khi các trang này trong bộ nhớ.
  - Quá trình sẽ không lỗi nữa cho đến khi nó thay đổi cục bộ.
  - Nếu cấp phát số khung ít hơn kích thước cục bộ, thrashing sẽ xảy ra.

Thrashing (5)
Mô hình tập làm việc (Working-Set Model)

- $\Delta \equiv$ cửa sổ tập làm việc (working-set window) $\equiv$ một số lượng cố định các tham khảo trang
  - Xác định Δ tham khảo trang mới nhất
  - Ví dụ: 10,000 tham khảo
- **WS_i** (tập làm việc của quá trình P_i) = tập hợp các trang trong Δ tham khảo trang gần đây nhất
  - Nếu trang đang được dùng, nó sẽ trong tập làm việc. Nếu không còn dùng nữa, nó sẽ ra khỏi tập làm việc sau Δ đơn vị thời gian.
  - Nếu Δ quá nhỏ: không bao quát được toàn bộ nhóm cục bộ.
  - Nếu Δ quá lớn: có thể đã phủ lấp vài nhóm cục bộ.
  - Nếu $\Delta = \infty \Rightarrow$ ta sẽ xem xét toàn bộ chương trình.



Thrashing (6)
Kích thước tập làm việc

- Thuộc tính quan trọng nhất của tập làm việc là kích thước của nó.
  - Gọi WSS_i là kích thước tập làm việc cho quá trình *i*.
  - $\boldsymbol{D = \Sigma\, WSS_i}$ ≡ tổng số các khung cần thiết.
  - *m* là số khung sẵn dùng.
  - Nếu ***D > m*** ⇨ thrashing.
- Chính sách: nếu ***D > m***, tạm dừng một số quá trình.

Thrashing (7)
Theo dõi tập làm việc

- Sử dụng bộ định thời + bit tham khảo
- Ví dụ: Δ = 10,000
  - Bộ định thời phát ra ngắt mỗi 5000 tham khảo.
  - Giữ trong bộ nhớ 2 bit cho một trang.
  - Mỗi khi bộ định thời phát ra ngắt thì chép và xóa nội dung của các bit tham khảo.
  - Khi có một lỗi trang xuất hiện, kiểm tra bit tham khảo hiện tại và 2 bit trong bộ nhớ để xác định là trang có được dùng trong khoảng 10,000 đến 15,000 tham khảo cuối cùng không.
  - Nếu có ít nhất 1 trong các bit bằng 1 $\Rightarrow$ trang nằm trong tập làm việc.
- Giải pháp trên không thật sự chính xác, bởi vì không thể biết được tham khảo trang xuất hiện lúc nào trong khoảng 5,000 tham khảo.
- Cải tiến: dùng 10 bits và phát ngắt mỗi 1000 tham khảo.

Thrashing (8)

Sơ đồ tần suất lỗi trang (PFF: Page-Fault Frequency Scheme)

- Thrashing được xem như là hiện tượng tỷ lệ lỗi trang cao ⇨ kiểm soát tỷ lệ lỗi trang để hạn chế thrashing.
- PFF = số lỗi trang / số chỉ thị đã thực thi
- Nếu PFF quá cao, quá trình cần thêm khung. Ngược lại, nếu PFF quá thấp, quá trình có quá nhiều khung.
- Đặt cận trên và dưới (upper and lower bound) cho PFF.
- Nếu PFF > cận trên, cấp thêm khung cho quá trình. Nếu không có khung sẵn dùng ⇨ chuyển quá trình ra ngoài.
- Nếu PFF < cận dưới ⇨ có thể lấy bớt bộ nhớ của quá trình này.

Ví dụ trên các hệ điều hành (1)
Windows NT, XP

- Sử dụng phân trang theo yêu cầu với kỹ thuật **clustering**. Clustering mang vào bộ nhớ trang bị lỗi và một số trang theo sau.
- Các quá trình được cấp cho **working set minimum** và **working set maximum**.
- Working set minimum là số lượng tối thiểu các trang mà quá trình được bảo đảm cung cấp trong bộ nhớ.
- Một quá trình có thể được cung cấp nhiều trang như có thể cho đến khi số lượng trang đạt đến ngưỡng working set maximum.
- Khi lượng bộ nhớ còn trống trong hệ thống rớt xuống dưới ngưỡng cho phép, chương trình **automatic working set trimming** được thực thi để khôi phục lại lượng bộ nhớ trống.
- Working set trimming xóa các trang của những quá trình có các trang vượt quá ngưỡng working set minimum.

Ví dụ trên các hệ điều hành (2)
Solaris 2

- Duy trì một danh sách các trang trống để cấp cho các quá trình.
- Kết hợp với danh sách trang trống tham số **lotsfree** - ngưỡng để bắt đầu việc phân trang.
  - HĐH sẽ kiểm tra tham số lotsfree.
  - Nếu số trang trống dưới lotsfree, quá trình **pageout** khởi động.
- Pageout dò xét các trang sử dụng giải thuật đồng hồ hai kim (two-handed-clock):
  - Kim đầu của đồng hồ sẽ quét qua tất cả các trang trong bộ nhớ và đặt bít tham khảo là 0.
  - Tại một thời điểm sau đó, kim sau của đồng hồ sẽ kiểm tra và đưa các trang trong bộ nhớ vẫn có bit tham khảo là 0 vào danh sách trống.
- Tham số **scanrate** là tỉ lệ mà các trang được quét/giây. Tỉ lệ này có miền giá trị từ **slowscan** đến **fastscan**. Khi bộ nhớ trống xuống dưới lotsfree, việc quét sẽ bắt đầu tại slowscan và tăng dần đến fastscan tuỳ vào số lượng bộ nhớ trống sẵn dùng.

Ví dụ trên các hệ điều hành (3)
Bộ quét trang Solaris

- Pageout được gọi thường xuyên hay không? ⇨ Còn phụ thuộc vào lượng bộ nhớ còn trống.