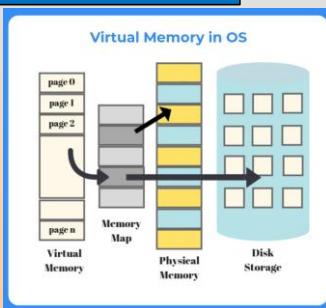




Chương 4

4.4 Bộ nhớ ảo



Mục tiêu

- Mô tả các lợi ích của bộ nhớ ảo.
- Giải thích các khái niệm phân trang theo yêu cầu, các giải thuật thay trang, phân phối các page frame

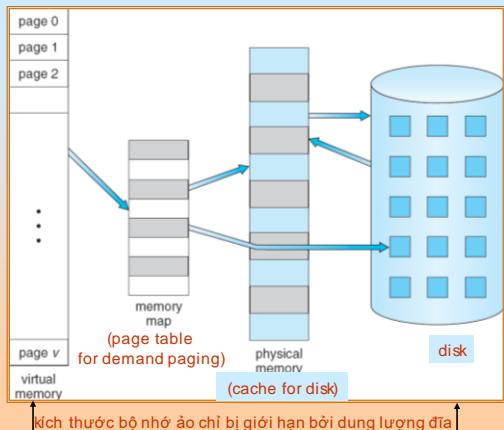
Nội dung:

- Kiến thức cơ bản
- Phân trang theo yêu cầu - Demand Paging
- Thay trang - Page Replacement
- Phân phối các Frames - Allocation of Frames
- Thrashing
- Phân đoạn theo yêu cầu - Demand Segmentation

4.4.1. Kiến thức cơ bản

- **Virtual memory** – sự tách riêng của bộ nhớ logic người sử dụng khỏi bộ nhớ vật lý.
 - Kích thước bộ nhớ vật lý có hạn \Rightarrow giới hạn kích thước ch. trình
 - Thực tế, chỉ một phần của chương trình cần phải đưa vào bộ nhớ (vật lý) để thực hiện \Rightarrow có thể chứa chương trình ở đâu? – **VIRTUAL MEMORY – phần địa cứng được quản lý như RAM**
 - Do đó không gian địa chỉ logic có thể lớn hơn không gian địa chỉ vật lý rất nhiều \Rightarrow cung cấp bộ nhớ rất lớn cho người lập trình
 - Cho phép một số tiến trình chia sẻ không gian địa chỉ.
 - Cho phép tạo tiến trình hiệu quả hơn.
- Bộ nhớ ảo có thể được thực thi thông qua:
 - Demand paging (Windows, Linux...)
 - Demand segmentation (IBMOS/2)

Bộ nhớ ảo lớn hơn bộ nhớ vật lý

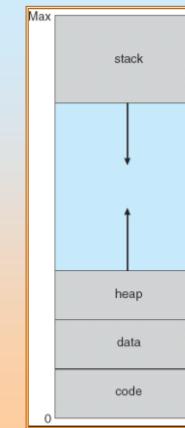


Không gian địa chỉ ảo của tiến trình

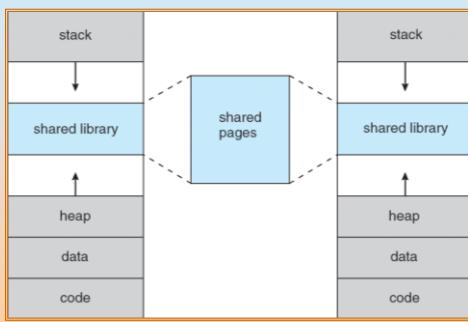
- Là cách nhìn logic (ảo) về cách mà tiến trình được lưu trong bộ nhớ.

- Tiến trình được lưu trong vùng nhớ liên tục, dù thực tế trong bộ nhớ vật lý nó có thể được lưu trong các page frame không liên tục.

- MMU đảm nhận việc ánh xạ các trang logic vào các page frame trong bộ nhớ vật lý.



Shared Library Using Virtual Memory



4.4.2. Phân trang theo yêu cầu

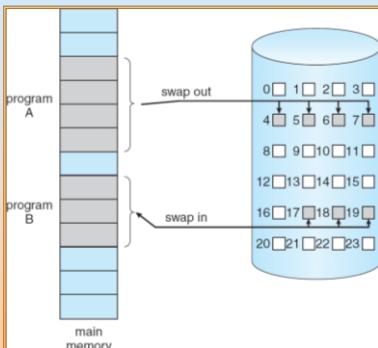
■ Đưa một trang vào bộ nhớ chỉ khi nó được cần đến.

- Cần ít thao tác vào-ra hơn
- Cần ít bộ nhớ hơn
- Đáp ứng nhanh hơn: tiến trình bắt đầu ngay sau khi số trang tối thiểu được nạp vào bộ nhớ
- Nhiều người sử dụng/tiến trình hơn: do mỗi tiến trình dùng ít bộ nhớ hơn

■ Khi trang được cần đến (khi tiến trình tham chiếu đến nó)

- tham chiếu không hợp lệ \Rightarrow hủy bỏ
- không trong bộ nhớ \Rightarrow đưa vào bộ nhớ
- có trong bộ nhớ \Rightarrow truy nhập

Chuyển một vùng nhớ phân trang tới không gian đĩa



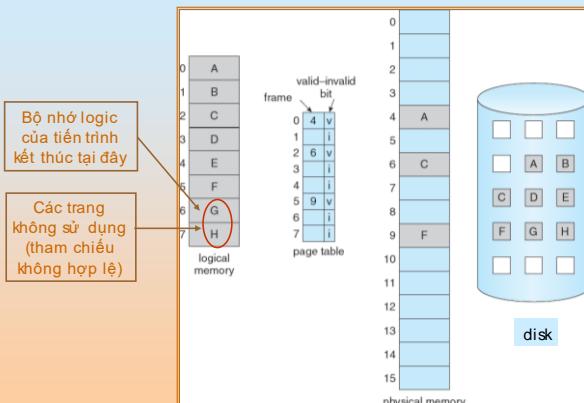
Valid-Invalid Bit

- Mỗi phần tử trong bảng phân trang được gắn thêm một valid-invalid bit ($1 \Rightarrow$ có trong bộ nhớ, $0 \Rightarrow$ không trong bộ nhớ)
- Ban đầu khởi tạo tất cả các v-inv bit bằng 0
- Ví dụ bảng phân trang:

Frame #	v	alid	inv	alid	bit
1					
1					
1					
1					
0					
:					
0					
0					

- Khi thông dịch địa chỉ, nếu v-inv bit bằng 0 \Rightarrow page fault.

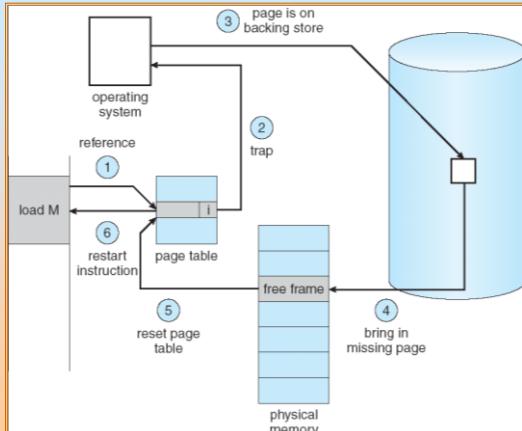
Bảng phân trang khi một số trang không ở trong bộ nhớ chính



Các bước xử lý page fault

- Khi có tham chiếu tới trang, đầu tiên tham chiếu sẽ lập bẫy tới HDH \Rightarrow phát hiện page fault
- HDH tìm trong bảng khác để quyết định:
 - tham chiếu không hợp lệ \Rightarrow hủy bỏ.
 - không có trong bộ nhớ \Rightarrow đưa vào bộ nhớ.
- Nhận frame rỗi
- Copy/Hoán đổi trang vào frame.
- Cập nhật lại bảng phân trang (thiết lập v-inv bit = 1), cập nhật danh sách frame rỗi.
- Khởi động lại lệnh gây page fault

Các bước xử lý page fault (tiếp)



Hiệu năng của phân trang theo yêu cầu

- Tỷ lệ Page Fault - $p : 0 \leq p \leq 1$
 - $p=0$: không có page faults;
 - $p=1$: mọi tham chiếu đều là fault.
- Thời gian truy nhập hiệu quả-Effective Access Time (EAT)

$$EAT = (1 - p) \times ma + p \times [\text{thời gian xử lý page-fault}]$$

Trong đó:

- + ma : memory access - thời gian truy nhập bộ nhớ (10-200 ns)
- + thời gian xử lý page-fault: gồm 3 vấn đề chính:
 - phục vụ ngắt page-fault (1-100 μs, có thể giảm bằng coding)
 - đọc trang vào bộ nhớ (≈ 8 ms)
 - khởi động lại tiến trình (1-100 μs, có thể giảm bằng coding)

Ví dụ

- Thời gian xử lý page-fault: 8 ms
- Thời gian truy nhập bộ nhớ (ma): 200 ns (= 0.2 μs)
- $EAT = (1-p) \times 200 + p \times 8,000,000 \text{ ns}$
 $= 200 + 7,999,800 \times p \text{ ns}$
- Nếu 1000 truy nhập có 1 page fault ($p = 1/1000$),
 $EAT = 8.2 \mu\text{s}$. Máy tính chậm hơn 40 lần vì phân trang có yêu cầu.
- EAT tỷ lệ trực tiếp với tỷ lệ page-fault, nếu p càng lớn thì EAT càng lớn → máy càng chậm.
- Muốn hiệu năng giảm dưới 10%, ta cần có:
 $220 > 200 + 7,999,800 \times p$
 $\rightarrow p < 0.0000025$

Điều gì xảy ra khi không có frame rỗ?

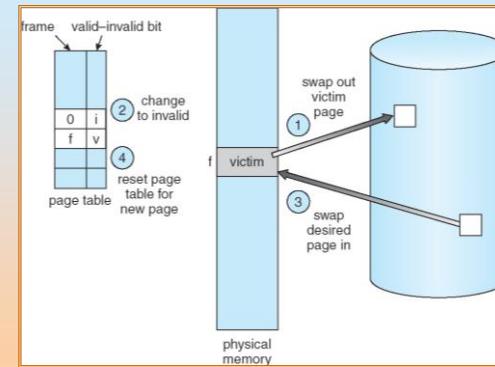
- Thay trang – tìm một số trang trong bộ nhớ nhưng đang không được sử dụng để đưa ra ngoài.
 - giải thuật?
 - hiệu năng? – muốn có một giải thuật tác động đến số lượng tối thiểu page faults.
- Một trang có thể được đưa vào bộ nhớ nhiều lần.

4.4.3. Thay trang

Các bước thay trang:

1. Tìm vị trí của trang được yêu cầu trên đĩa.
2. Tìm một frame rỗng:
 - Nếu có frame rỗng thì sử dụng nó.
 - Nếu không có, sử dụng một giải thuật thay trang để chọn một frame *nạn nhân*.
3. Đưa trang được yêu cầu vào frame rỗng. Cập nhật bảng phân trang và bảng quản lý frame rỗng.
4. Khởi động lại tiến trình.

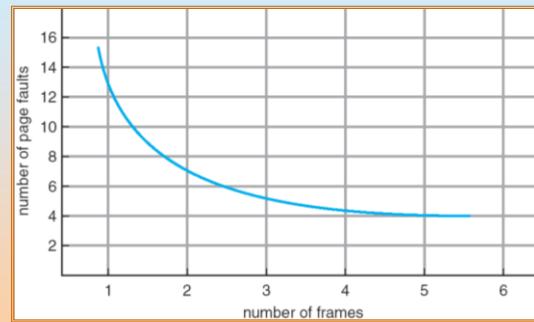
Page Replacement



Các giải thuật thay trang

- Mục đích: tỷ lệ page-fault thấp nhất.
- Đánh giá giải thuật bằng cách chạy nó trên một chuỗi cụ thể các tham chiếu bộ nhớ và tính số page faults trên chuỗi đó.
- Trong tất cả các ví dụ, chuỗi tham chiếu là
1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5.

Đồ thị liên hệ giữa Page Faults và số Frame



a) First-In-First-Out (FIFO) Algorithm

- Chuỗi tham chiếu: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
- 3 frames (với mỗi tiến trình: 3 trang có thể nạp vào bộ nhớ tại một thời điểm)

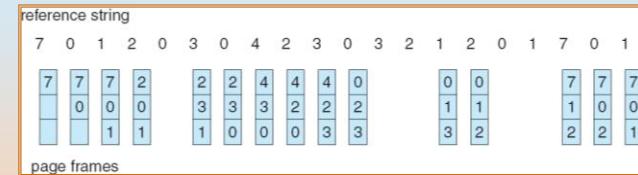
1	1	4	5
2	2	1	3
3	3	2	4

- 4 frames

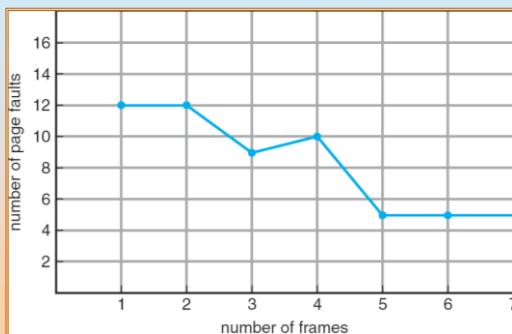
1	1	5	4
2	2	1	5
3	3	2	
4	4	3	

- FIFO Replacement – Sự bất thường của Belady
 - nhiều frames \Rightarrow nhiều page faults

FIFO Page Replacement



Minh họa sự bất thường của Belady



b) Optimal Algorithm

- Thay trang có thời gian sẽ không sử dụng đến lâu nhất.
- ví dụ 4 frames

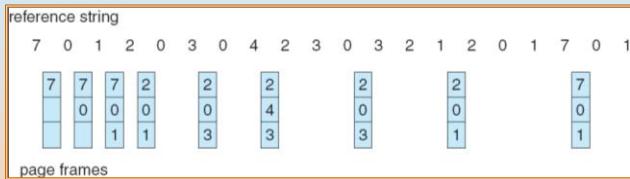
1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

1	4
2	
3	
4	5

6 page faults

- Làm cách nào để biết trang nào sẽ không được sử dụng nữa? \rightarrow Không thể biết được.
- Được sử dụng làm tiêu chuẩn đánh giá các giải thuật khác.

Optimal Page Replacement



c) Least Recently Used (LRU) Algorithm

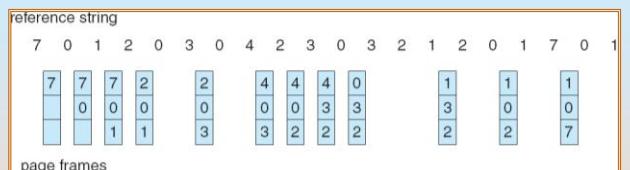
- Thay trang có khoảng thời gian không dùng lâu nhất
- Chuỗi tham chiếu: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5



Sự thực hiện của Bộ đếm (Counter)

- Mọi phần tử bảng có một bộ đếm, mọi thời điểm trang được tham chiếu qua phần tử bảng này, copy clock vào trong bộ đếm.
- Khi trang cần được hoán đổi, tìm trong bộ đếm để xác định trang nào làm nạn nhân.

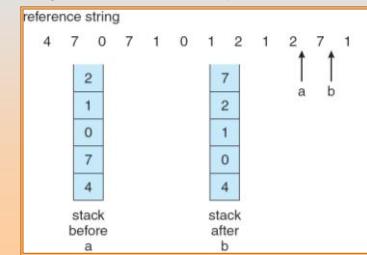
LRU Page Replacement



- Cách xác định trang không sử dụng lâu nhất?
- Sử dụng Bộ đếm (Counter)
 - Mọi phần tử bảng có một bộ đếm, mọi thời điểm trang được tham chiếu qua phần tử bảng này, copy clock vào trong bộ đếm.
 - Khi trang cần được hoán đổi, tìm trong bộ đếm để xác định trang nào làm nạn nhân.

LRU Algorithm (tiếp)

- Sử dụng Stack
 - Dùng một stack của các số trang
 - Khi trang được tham chiếu:
 - chuyển nó lên đỉnh \Rightarrow trang không dùng lâu nhất sẽ luôn dưới đáy
 - dùng một danh sách liên kết, cần 6 con trỏ để đổi trang
 - Không cần tìm kiếm để thay thế



d) Các giải thuật tương tự LRU

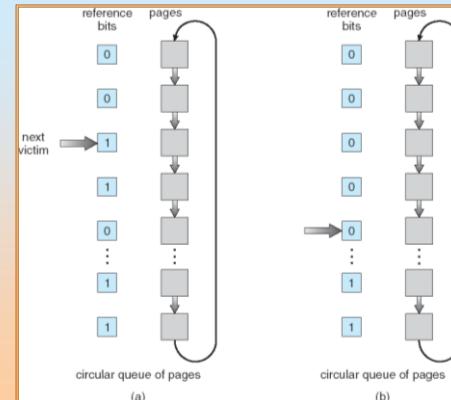
■ Giải thuật dùng thêm bit tham chiếu

- Gắn một bit vào mỗi trang, khởi tạo = 0
- Khi trang được tham chiếu, bit đó được thiết lập = 1.
- Thay trang có bit tham chiếu = 0, nếu có tồn tại

■ Giải thuật cơ hội thứ hai (Second chance)

- Cần có bit tham chiếu. Nếu bit tham chiếu = 0 thì thay trang. Trái lại cho trang đó cơ hội thứ hai và chuyển đến trang tiếp sau
 - thiết lập bit tham chiếu = 0.
 - để trang lại trong bộ nhớ.
 - thay trang kế tiếp (theo FIFO) với luật tương tự.
- Một cách thực hiện giải thuật là sử dụng queue vòng tròn.

Giải thuật thay trang "cơ hội thứ hai"



e) Các giải thuật dựa trên số liệu thống kê

- Dành ra một bộ đếm số tham chiếu đến mỗi trang.
- **Least Frequently Used (LFU) Algorithm:** thay trang đếm được ít nhất (có tần số truy nhập nhỏ nhất).
- **Most Frequently Used (MFU) Algorithm:** thay trang đếm được nhiều nhất (có tần số truy nhập cao nhất), dựa trên lý luận rằng trang đếm được ít nhất là có thể vừa được đưa vào bộ nhớ và chưa kịp được sử dụng.

4.4.4. Phân phối các Frame

- Mỗi tiến trình cần số lượng trang tối thiểu để thực hiện.
- Ví dụ: IBM 370 – cần 6 trang để thực hiện lệnh SS MOVE:
 - lệnh độ dài 6 bytes, có thể chứa trong 2 trang.
 - 2 trang để thực hiện **from**.
 - 2 trang để thực hiện **to**.
- Hai cách phân phối chính:
 - phân phối cố định (fixed allocation)
 - phân phối theo mức ưu tiên (priority allocation)

Phân phối cố định

1. Phân phối công bằng – vd nếu có 100 frames và 5 tiến trình, cho mỗi tiến trình 20 trang.
2. Phân phối theo kích thước của tiến trình

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

$$m = 64$$

$$S = \sum s_i$$

$$s_1 = 10$$

m = total number of frames

$$s_2 = 127$$

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

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

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

Phân phối theo mức ưu tiên

- Nếu tiến trình P_i phát sinh một page fault,
 - chọn thay thế một trong số các frame của nó.
 - frame thay vào đó được chọn từ một tiến trình có mức ưu tiên thấp hơn.

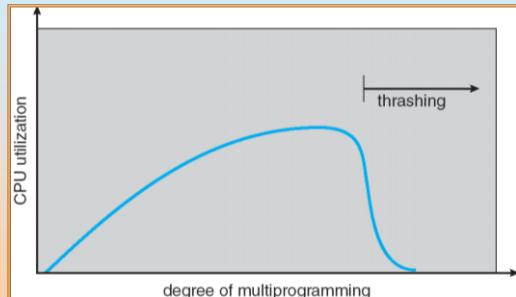
Global vs. Local Allocation

- **Global** replacement – tiến trình chọn một frame thay thế từ tập tất cả các frame; một tiến trình có thể lấy một frame từ một tiến trình khác.
- **Local** replacement – mỗi tiến trình chỉ chọn một frame thay thế từ chính tập các frame đã phân phối cho nó.

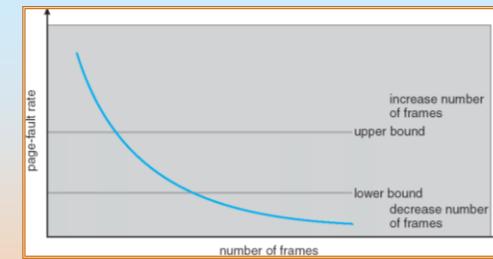
4.4.5. Thrashing

- Nếu một tiến trình không có “đủ” trang, tỷ lệ page fault là rất cao. Điều này dẫn đến:
 - sử dụng CPU thấp.
 - HDH cho rằng nó cần phải tăng mức đa chương trình.
 - một tiến trình khác được thêm vào hệ thống, nó cố gắng giành frame từ các tiến trình đang thực hiện \Rightarrow tỷ lệ page fault càng tăng...
- **Thrashing** = các tiến trình mất nhiều thời gian (bận rộn) với việc hoán đổi các trang vào-ra hơn là thời gian thực hiện.

Thrashing (tiếp)



Lược đồ tần số Page-Fault



- Thiết lập tỷ lệ page-fault “có thể chấp nhận được”
 - Nếu tỷ lệ thực tế quá thấp, tiến trình làm mất frame.
 - Nếu tỷ lệ thực tế quá cao, tiến trình tăng thêm frame.

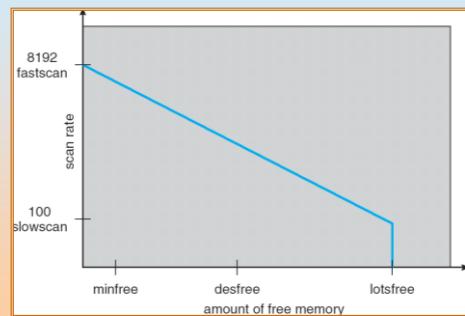
Windows XP

- Sử dụng phân trang theo yêu cầu có **phân cụm (clustering)**: đưa vào bộ nhớ faulting page và một số trang tiếp sau.
- Các tiến trình được ấn định **working set minimum** và **working set maximum**
- Working set minimum là số lượng trang nhỏ nhất mà tiến trình được đảm bảo có trong bộ nhớ.
- Một tiến trình có thể được gán cho càng nhiều trang càng tốt không vượt quá working set maximum của nó.
- Khi dung lượng bộ nhớ rỗi của hệ thống nhỏ hơn một ngưỡng, **automatic working set trimming** được thực hiện để khôi phục lại dung lượng bộ nhớ rỗi.
- Working set trimming loại bỏ số trang của các tiến trình vượt quá working set minimum của chúng.

Solaris

- Duy trì danh sách các trang rỗi để gán các faulting process.
- **Lotsfree** – tham số ngưỡng (dung lượng bộ nhớ rỗi) để bắt đầu phân trang.
- **Desfree** – tham số ngưỡng để tăng phân trang
- **Minfree** – tham số ngưỡng để thực hiện swapping
- Phân trang được thực hiện bởi tiến trình **pageout**
- Pageout quét các trang sử dụng giải thuật clock có sửa đổi
- **Scanrate** là tốc độ các trang được quét. Dải này từ *slowscan* đến *fastscan*
- Pageout được gọi càng thường xuyên phụ thuộc vào dung lượng bộ nhớ rỗi khả dụng.

Solaris 2 Page Scanner



BÀI TẬP
