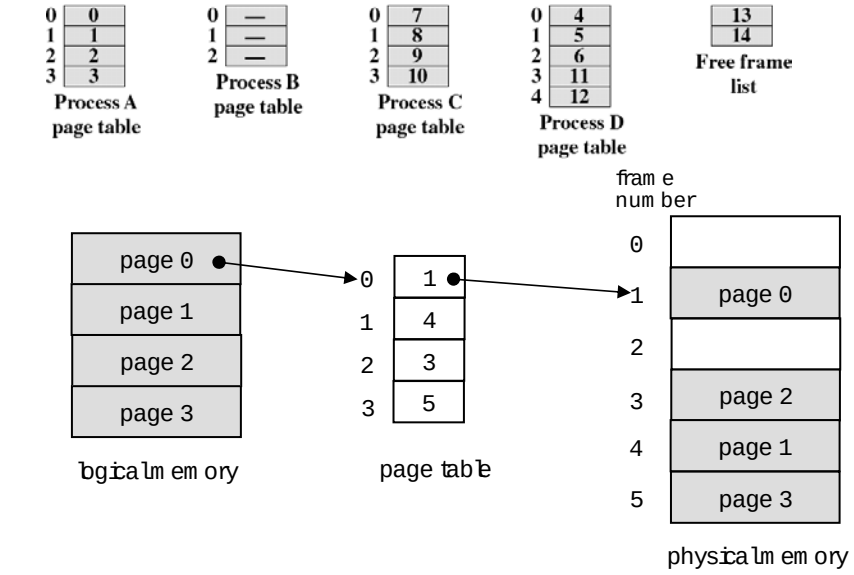
7.Virtual Memory

- Cơ chế phân trang và phân đoạn
- Cơ chế bộ nhớ ảo
- Các chiến lược quản lý
  - Fetch Policy
  - Placement policy
  - Page replacement policy
- Cấp phát frame cho process
- Thrashing

Cơ chế phân trang (paging)

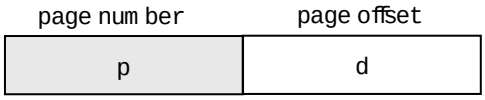
- Cơ chế phân trang cho phép không gian địa chỉ thực (physical address space) của một process có thể *không liên tục nhau*.
- Bộ nhớ thực được chia thành các khối kích thước cố định bằng nhau gọi là ***frame***.
  - Thông thường kích thước của frame là lũy thừa của 2, từ khoảng 512 byte đến 16MB
- Bộ nhớ luận lý (logical memory) cũng được chia thành khối cùng kích thước gọi là trang nhớ (***page***).
- Hệ điều hành phải thiết lập một bảng phân trang (***page table***) để ánh xạ địa chỉ ảo, luận lý thành địa chỉ thực (***address translation scheme***)
  - Mỗi process có một bảng phân trang được quản lý bằng một con trỏ lưu giữ trong PCB. Công việc nạp bảng phân trang vào hệ thống (do CPU dispatcher thực hiện) là một phần của chuyển ngữ cảnh
- Cơ chế phân trang khiến bộ nhớ bị phân mảnh nội, tuy nhiên lại khắc phục được phân mảnh ngoại.

Cơ chế phân trang (t.t)



Mô hình chuyển đổi địa chỉ

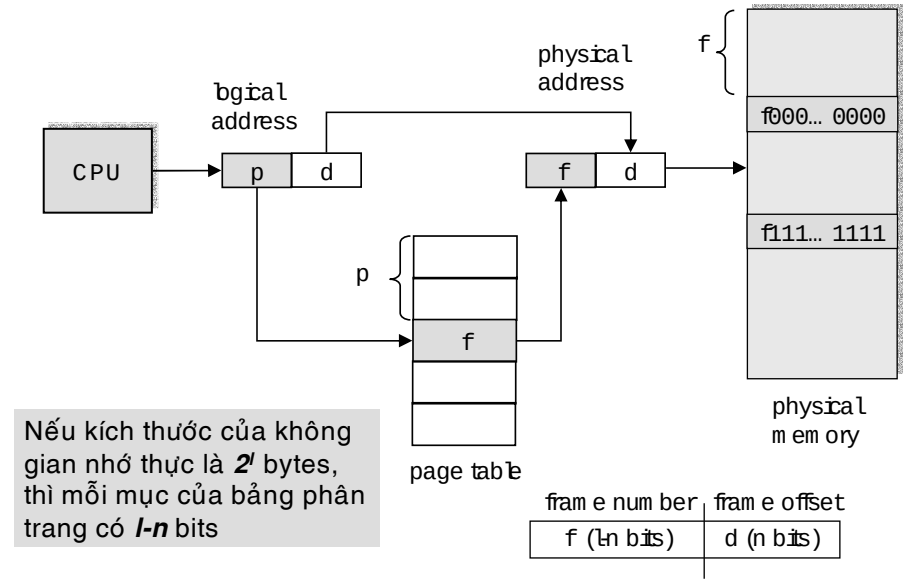
- Địa chỉ nhớ do CPU tạo ra (logical address) gồm có:
  - *Page number (p)* – được dùng làm chỉ mục dò tìm trong bảng phân trang. Mỗi mục trong bảng phân trang chứa địa chỉ cơ sở (hay chỉ số frame) của trang tương ứng trong bộ nhớ thực.
  - *Page offset (d)* – được kết hợp với địa chỉ cơ sở (base address) để định vị một địa chỉ thực.
- Nếu kích thước của không gian địa chỉ ảo là 2^m, kích thước của trang là 2^n



m-n bits (*định vị từ 0 ÷ 2^{m-n} -1*) — *n bits* (*định vị từ 0 ÷ 2^n-1*)

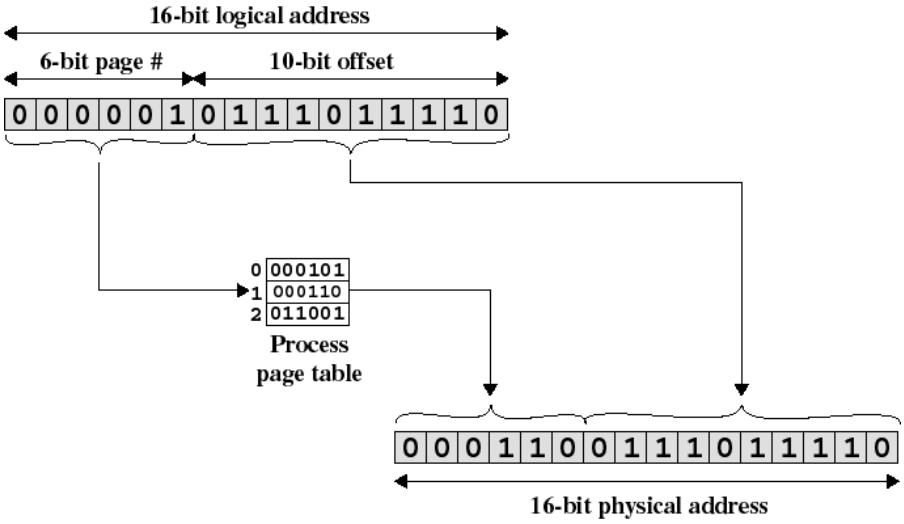
Do đó, bảng phân trang sẽ có tổng cộng $2^m/2^n = 2^{m-n}$ mục

Paging Hardware



Nếu kích thước của không gian nhớ thực là 2^l bytes, thì mỗi mục của bảng phân trang có $l-n$ bits

Chuyển đổi bộ nhớ với paging



Hiện thực bảng phân trang

- Bảng phân trang được lưu giữ trong bộ nhớ chính (kernel memory)
  - Mỗi process có một bảng phân trang
  - Thanh ghi page-table base (PTBR) trỏ đến bảng phân trang
  - Thanh ghi page-table length (PTLR) biểu thi kích thước của bảng phân trang (và dùng để bảo vệ bộ nhớ)
- Mỗi tác vụ truy cập dữ liệu/lệnh cần hai thao tác truy xuất vùng nhớ
  - Một thao tác truy xuất bảng phân trang (page number: p) và một thao tác truy xuất dữ liệu/lệnh (page offset: d – displacement)
  - Thường 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 translation look-aside buffers (TLBs)

Associative Register (hardware)

- Thanh ghi kết hợp (*associative register*): hỗ trợ tìm kiếm truy xuất dữ liệu đồng thời với tốc độ cực nhanh.

Page #	Frame #

Số mục của TLB khoảng 8 ÷ 2048

TLB là "cache" của bảng phân trang

Khi có chuyển ngữ cảnh, TLB bị xóa

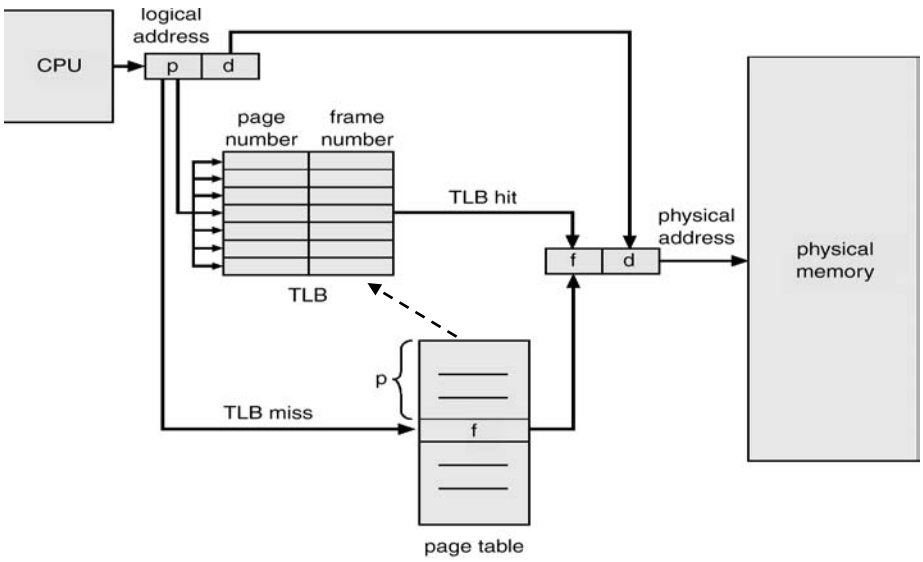
Khi TLB bị đầy, thay thế bằng LRU

Ánh xạ địa chỉ ảo (A', A'')

–Nếu A' nằm trong TLB (HIT) ⇒ lấy ngay được chỉ số frame ⇒ tiết kiệm được ~ 10% thời gian tìm kiếm.

–Ngược lại (MISS), phải tìm chỉ số frame từ bảng phân trang như bình thường.

Paging Hardware với TLB



Effective Access Time (EAT)

- Thời gian tìm kiếm (associative lookup): ε (đơn vị)
- Thời gian một chu kỳ truy xuất bộ nhớ: x (đơn vị)
- *Hit Ratio* – tỉ lệ phần trăm thời gian một chỉ số trang được tìm thấy (HIT) trong TLB; tỉ lệ với số thanh ghi kết hợp của TLB.
  - Kí hiệu hit ratio = α
- Thời gian lấy được trang
  - Khi trang có trong TLB (HIT) $\varepsilon + x$
  - Khi trang không có trong TLB (MISS) $\varepsilon + 2x$
- Thời gian truy xuất hiệu dụng - Effective Access Time (EAT)

$$EAT = (x + \varepsilon)\alpha + (2x + \varepsilon)(1 - \alpha)$$
$$= (2 - \alpha) * x + \varepsilon$$

Effective Access Time (t.t)

- Ví dụ 1
  - Associate lookup = 20
  - Memory access = 100
  - Hit ratio = 0.8
  - EAT = (100 + 20) * 0.8 +
    (200 + 20) * 0.2
    = 1.2 * 100 + 20
    = 140

40% slow in memory access time

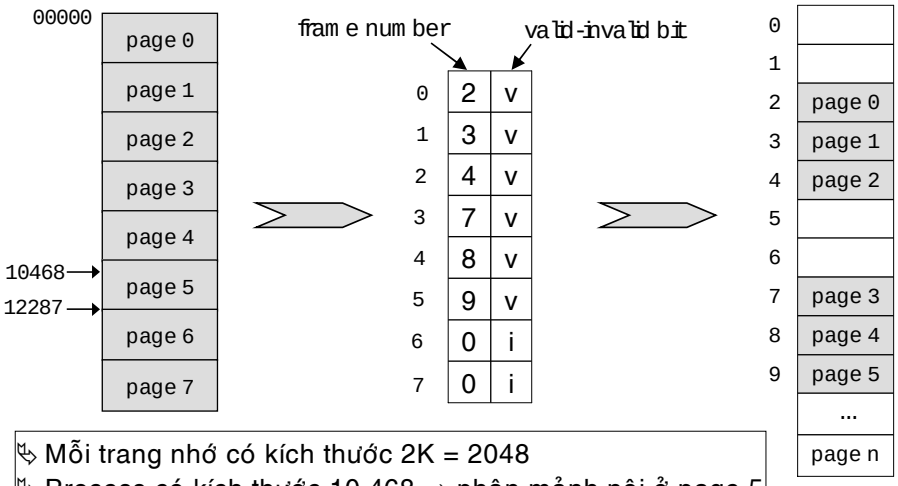
- Ví dụ 2
  - Associate lookup = 20
  - Memory access = 100
  - Hit ratio = 0.98
  - EAT = (100 + 20) * 0.98 +
    (200 + 20) * 0.02
    = 1.02 * 100 + 20
    = 122

22% slow in memory access time

Bảo vệ bộ nhớ

- Việc bảo vệ bộ nhớ được hiện thực bằng cách gắn với frame các bit bảo vệ (protection bits). 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 bộ nhớ tương ứng nằm trong không gian nhớ địa chỉ ảo của process, do đó là một trang hợp lệ.
  - "invalid": cho biết là trang bộ nhớ tương ứng không nằm trong không gian nhớ địa chỉ ảo của process, do đó là một trang bất hợp lệ.

Bảo vệ bằng Valid/Invalid bit

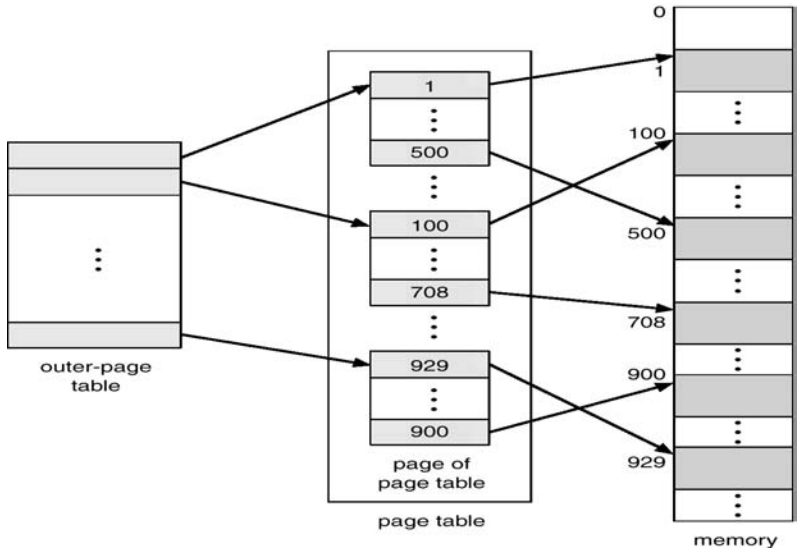


- Mỗi trang nhớ có kích thước 2K = 2048
- Process có kích thước 10,468 ⇒ phân mảnh nội ở page 5
  ⇒ các địa chỉ > 12287 là các địa chỉ invalid.
- Dùng PTLR để kiểm tra kích thước bảng phân trang

Hierarchical Page Table

- Các hệ thống hiện đại đều hỗ trợ không gian địa chỉ ảo rất lớn (2^{32} đến 2^{64}).
  - Kích thước trang nhớ là 4KB (= 2^{12}) ⇒ bảng phân trang sẽ có ~ $2^{32}/2^{12} = 2^{20}$ = 1MB.
  - Giả sử mỗi phần tử là một con trỏ 32 bit thì mỗi process cần 4MB cho bảng phân trang ☹
- Một giải pháp được đặt ra là chia thành nhiều bảng phân trang quản lý các vùng không gian bộ nhớ ảo khác nhau – bảng đa mức (multilevel paging table).
- Cơ chế tạo bảng phân trang 2-mức (two-level page table), hay còn được gọi là forward-mapped page table trong hệ thống Intel Pentium®-II

Mô hình bảng 2-mức (two-level)



Phân trang 2-mức

- Một địa chỉ luận lý (trên hệ thống 32-bit với trang nhớ 4K) được chia thành các phần sau:
  - Page number: 20 bit
    - Nếu mỗi mục 4 byte
    - $\Rightarrow 2^{20}$ * 4 byte = 4 MB
  - Page offset: 12 bit

page #	offset
20 bit	12 bit

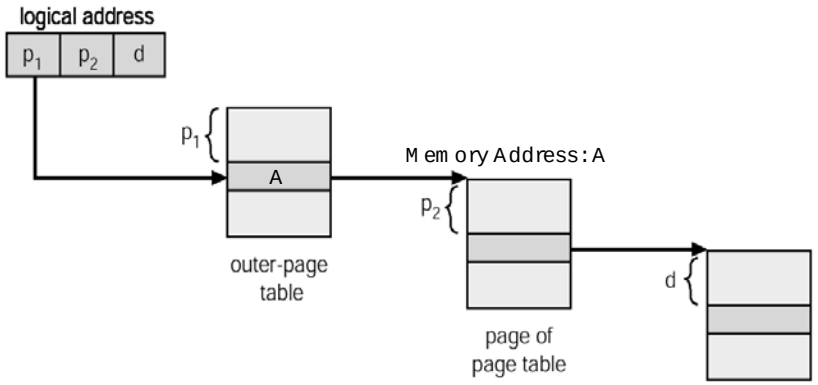
- Bảng phân trang cũng bị chia nhỏ nên page number cũng được chia nhỏ thành 2 phần:
  - 10-bit page number
  - 10-bit page offset

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

- Vì vậy,một địa luận lý sẽ như hình vẽ bên
  - p_1: chỉ mục của bảng ngoài (outer page table)-mức 1
  - p_2 : độ dời (displacement) ở trong trang mức 2 (xác định bởi *p_1)

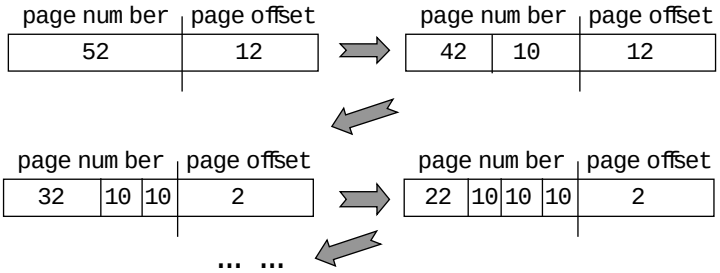
Sơ đồ ánh xạ địa chỉ

- Sơ đồ ánh xạ địa chỉ (address-translation scheme) cho kiến trúc bảng phân trang 2 mức, 32-bit địa chỉ



Phân trang đa mức (multilevel)

- Không gian địa chỉ luận lý 64-bit với trang nhớ 4K
  - Trong sơ đồ phân trang 2-mức, số mục của bảng phân trang = 2^{52} ($2^{64}/2^{12} = 2^{52}$) $\Rightarrow$ quá lớn. Thực hiện tương tự mô hình 2 mức, phân chia thành bảng 3, 4,..., n-mức

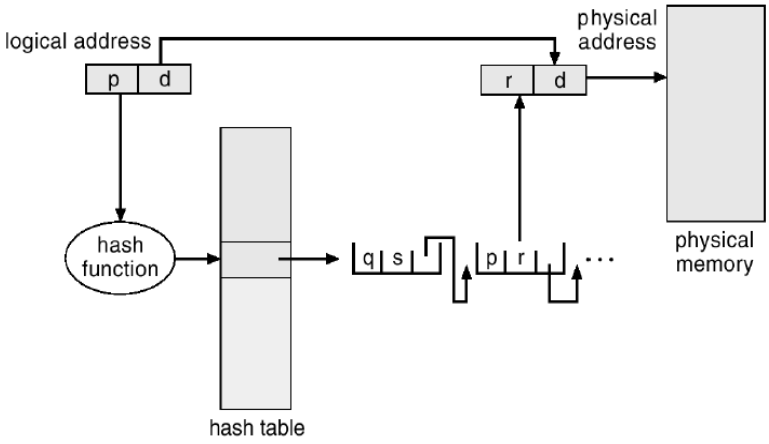


  - Hệ thống SPARC 32-bit hỗ trợ cơ chế 3-mức còn các hệ thống Motorolla 68030 32 bit hỗ trợ cơ chế 4-mức. Hệ thống 64bit UltraSPARC thì dùng bảng phân trang 7-mức
  - Hiệu suất của hệ thống phân trang đa mức ?

Bảng băm (hashed page table)

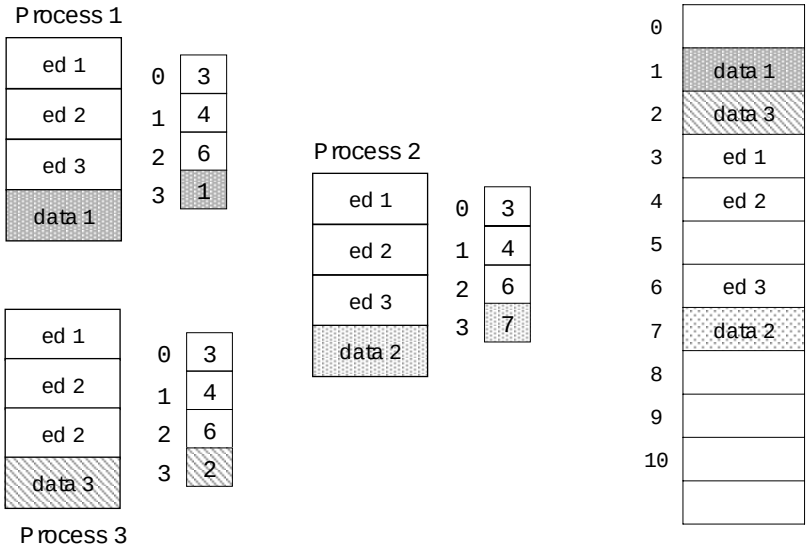
- Dùng ý tưởng của bảng băm để giảm bớt không gian bảng phân trang, tăng tốc độ tìm kiếm trang.
  - Rất phổ biến trong các hệ thống lớn hơn 32 bit địa chỉ.
- Để giải quyết đụng độ, mỗi phần tử của bảng phân trang quản lý một danh sách liên kết. Mỗi phần tử danh sách chứa chỉ số trang ảo và chỉ số frame tương ứng.
  - Chỉ số trang ảo (virtual page number) được biến đổi qua hàm băm thành một *hashed value*. Các thông tin như chỉ số trang ảo và chỉ số frame sẽ được lưu vào danh sách liên kết tại vị trí ứng với *hashed value*.
- Giải thuật dò tìm trang:
  - Chỉ số trang ảo được biến đổi thành *hashed value* (với cùng hàm băm như trên). Hashed value được dùng để tìm ra phần tử tương ứng trong bảng phân trang. Sau đó, dò tìm trong danh sách liên kết với chỉ số trang ảo để trích rút ra được frame tương ứng.

Hashed Page Tables



- Các hệ thống 64-bit địa chỉ thường dùng clustered page table, i.e. mỗi mục trong hash table tham chiếu đến nhiều trang (~ 16 trang) thay vì 1 trang.

Chia sẻ các trang nhớ

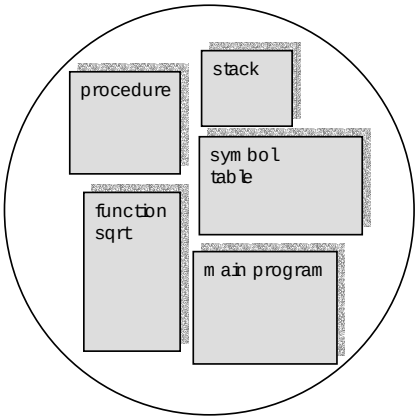


Phân đoạn (segmentation)

- Nhìn lại cơ chế phân trang
  - user-view (không gian địa chỉ ảo) tách biệt với không gian bộ nhớ thực. Cơ chế phân trang thực hiện phép ánh xạ user-view vào bộ nhớ thực.
- Trong thực tế, dưới góc nhìn của user, một chương trình cấu thành từ nhiều phân đoạn (segment). Mỗi phân đoạn là một đơn vị luận lý, ví dụ như:
  - main program, procedure, function, local variables, global variables, common block, stack, symbol table, arrays
- Cơ chế phân đoạn là mô hình quản lý bộ nhớ hỗ trợ user-view
  - Không gian địa chỉ ảo là một tập các phân đoạn (segment), mỗi phân đoạn có tên và kích thước riêng.
  - Một địa chỉ luận lý được định vị bằng tên phân đoạn và độ dời (offset) bên trong phân đoạn đó (so sánh với phân trang ???)

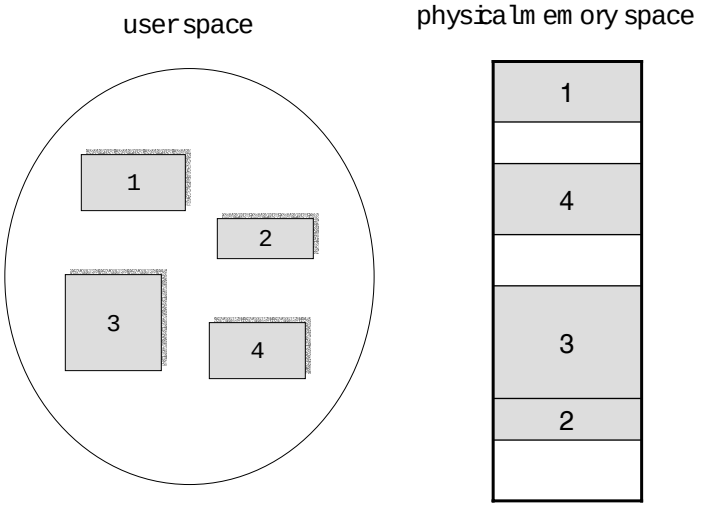
User-view của một chương trình

- Thông thường, một chương trình được biên dịch. Trình biên dịch sẽ tự động xây dựng các segment.
- Ví dụ, trình biên dịch Pascal sẽ tạo ra các segment sau:
  - Global variables
  - Procedure call stack
  - Procedure/function code
  - Local variable
- Trình loader sẽ gán mỗi segment một số định danh riêng.



Logical address space

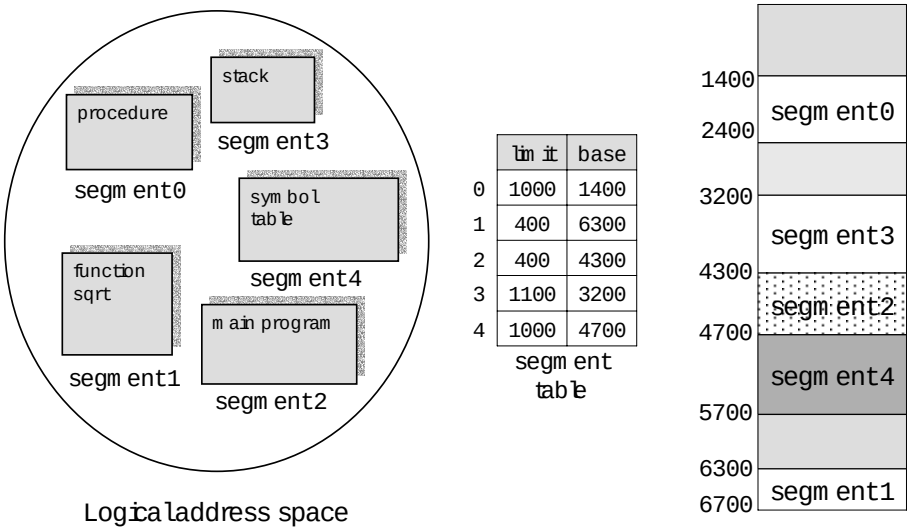
Mô hình cơ chế phân đoạn



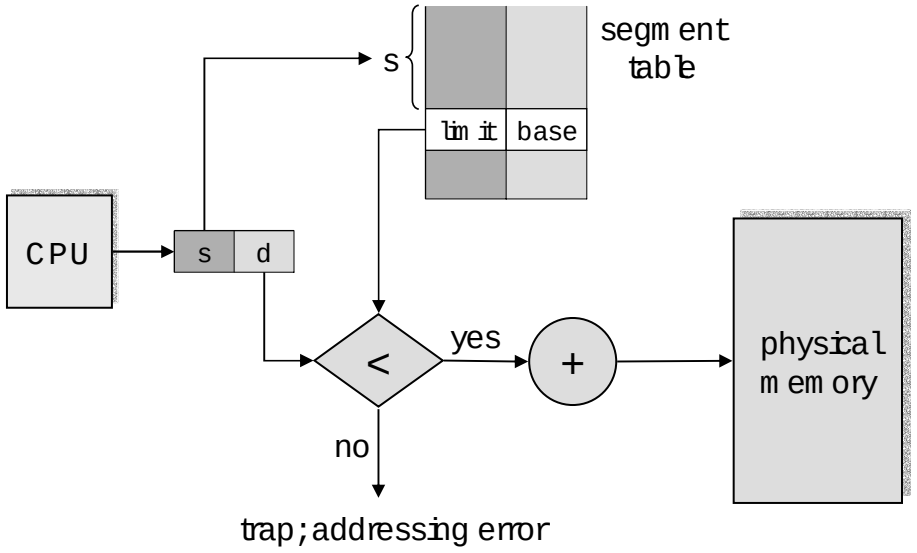
Tổ chức của cơ chế phân đoạn

- Địa chỉ luận lý là một cặp giá trị
  <segment-number, offset>
- Bảng phân đoạn (segment table)
  - base – chứa địa chỉ khởi đầu của phân đoạn trong bộ nhớ
  - limit – xác định kích thước của phân đoạn
- Segment-table base register (STBR): trỏ đến vị trí bảng phân đoạn trong bộ nhớ
- Segment-table length register (STLR): số segment của chương trình
  ⇒ Một chỉ số segment s là hợp lệ nếu s < STLR

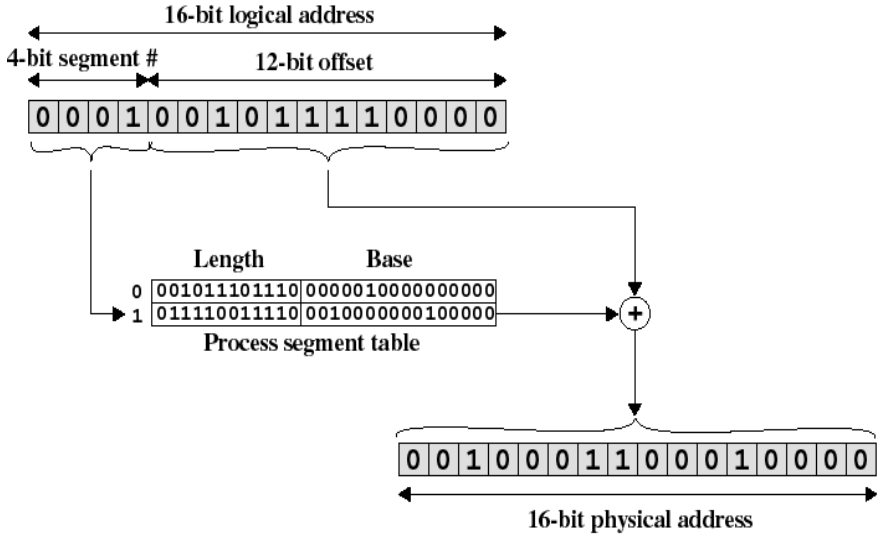
Một ví dụ về cơ chế phân đoạn



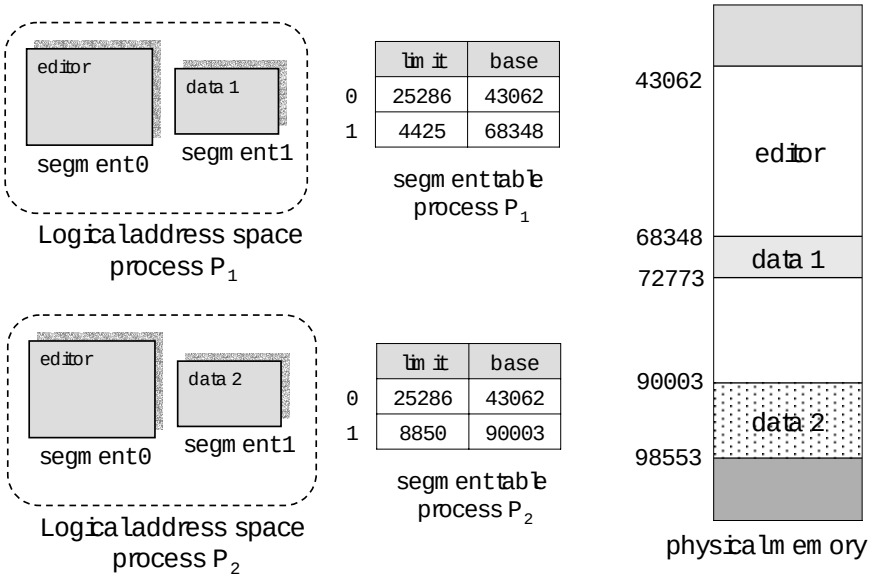
Phần cứng hỗ trợ phân đoạn



Chuyển đổi bộ nhớ phân đoạn



Chia sẻ các phân đoạn



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 có thể được chi thành các phần nhỏ (page hay segment); các phần này được nạp vào các vị trí không liên tục trong bộ nhớ chính
- Nhận xét quan trọng: không phải tất các các phần của một processs 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 cần thiết
  - 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

Quá trình thực thi của process

- OS nạp một số phần của chương trình vào bộ nhớ
- Mỗi bảng phân trang/đoạn có thêm một *present bit* cho biết phần tương ứng có nằm trong bộ nhớ chính hay không.
- Khi có một tham chiếu nằm trong phần không có trong bộ nhớ chính (*present bit = 0*) thì một ngắt được kích hoạt gọi là *memory fault*
- Process chuyển về trạng thái Blocking
- OS phát ra một yêu cầu đọc đĩa để nạp phần được tham chiếu vào bộ nhớ chính và trong khi đó, một process khác được chiếm quyền thực thi
- Sau khi I/O hoàn tất, một ngắt được kích hoạt, báo cho OS chuyển process tương ứng về trạng thái Ready

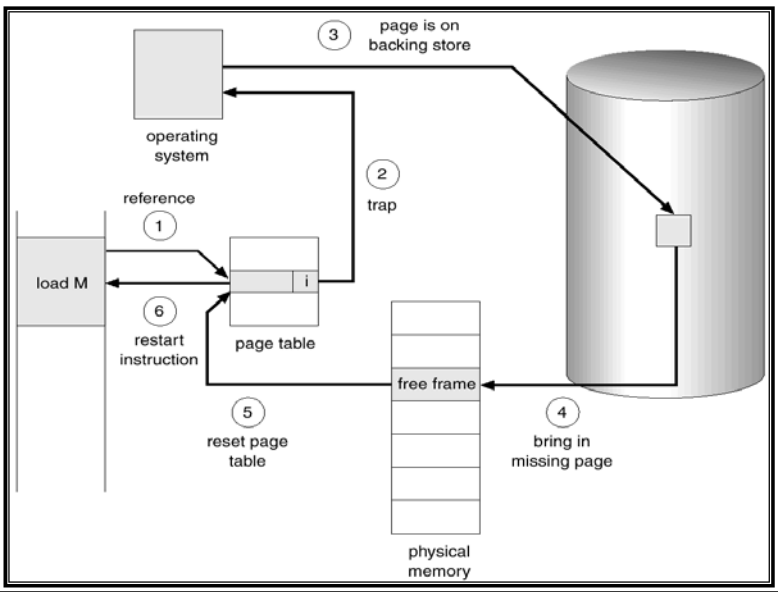
Ưu điểm của bộ nhớ ảo

- 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
- Bộ nhớ tham chiếu bởi một địa chỉ luận lý được gọi là bộ nhớ ảo (virtual memory)
  - Bao gồm bộ nhớ thực + một phần bộ nhớ thứ cấp (đĩa cứng,...)
  - Nhằm đạt hiệu quả cao, các dịch vụ file system thường được bỏ qua; đọc/ghi đĩa trực tiếp với các khối dữ liệu lớn hơn so với khối của hệ thống file.
  - Thông thường phần bộ nhớ ảo được lưu trữ ở một vùng đặc biệt gọi là không gian tráo đổi (swap space). Ví dụ file system *swap* trong Unix/Linux, file *pagefile.sys* trong Windows2K
- Việc chuyển đổi từ địa chỉ luận lý thành địa chỉ thực được thực hiện với sự hỗ trợ của phần cứng (memory management hardware)

Yêu cầu đối với bộ nhớ ảo

- 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 phạm vi chương này, chúng ta thảo luận về sự hỗ trợ cấp phần cứng trước, sau đó là các giải thuật của hệ điều hành
- Sự hỗ trợ của phần cứng đối với phân trang và phân đoạn đã được khảo sát ở chương 9. Chỉ có một điểm khác biệt là mỗi mục (entry) của bảng phân trang/đoạn có thêm các bit trạng thái đặc biệt
  - *Present bit* = 1⇒ hợp lệ và in-memory; = 0 ⇒ not-in-memory hoặc không hợp lệ
  - *Modified bit*: trang/đoạn có thay đổi kể từ khi được nạp vào hay không

Điều khiển page fault



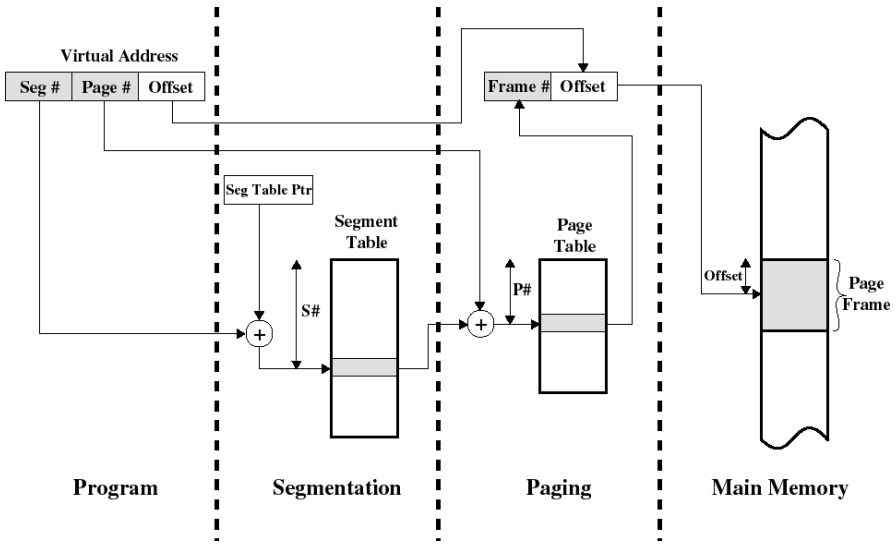
Kết hợp trang và đoạn

- Nhằm kết hợp các ưu điểm đồng thời hạn chế các khuyết điểm của hai mô hình phân trang và phân đoạn
- Có rất nhiều mô hình kết hợp. Sau đây là một mô hình đơn giản

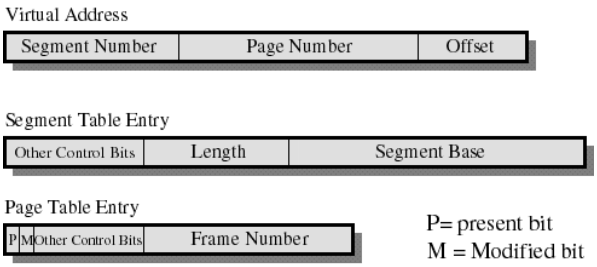
- Mỗi process sẽ có:
  - Một bảng phân đoạn
  - Nhiều bảng phân trang: mỗi phân đoạn có một bảng phân trang

- Một địa chỉ luận lý (địa chỉ ảo) bao gồm:
  - *segment number*: là chỉ mục của một phần tử trong bảng phân đoạn, phần tử này chứa địa chỉ cơ sở (base address) của bảng phân trang trong phân đoạn đó
  - *page number*: là chỉ mục trong bảng phân trang, dùng để tính ra chỉ số frame trong bộ nhớ thực tương ứng
  - *offset*: dùng để định vị một vị trí nhớ trong frame nói trên.

Sơ đồ chuyển đổi địa chỉ



Mô hình kết hợp đơn giản



- Segment Base: địa chỉ thực của bảng phân trang
- Present bit và modified bits chỉ tồn tại trong bảng phân trang
- Các thông tin bảo vệ và chia sẻ vùng nhớ thường nằm trong bảng phân đoạn
  - Ví dụ: read-only/read-write bit, ...

Các giải thuật của OS

- OS cung cấp các thư viện quản lý bộ nhớ ảo.
- Các chương trình quản lý bộ nhớ ảo của OS phụ thuộc vào sự hỗ trợ của phần cứng trong việc phân trang, phân đoạn hoặc kết hợp cả hai.
- Rất ít hệ thống sử dụng cơ chế phân đoạn thuần túy mà thông thường sẽ sử dụng mô hình kết hợp trong đó các segment được áp dụng cơ chế phân trang. Như vậy, vấn đề cần giải quyết của các giải thuật nạp, thay thế,...chủ yếu hướng đến đối tượng là *trang nhớ*.
  - Fetch policy
  - Placement policy
  - Replacement policy
- Mục tiêu của các giải thuật này là giảm thiểu *page fault*

Fetch & placement policy

- Fetch Policy
  - Xác định thời điểm nên nạp trang nhớ vào bộ nhớ. Có hai chiến lược thông dụng
  - *Demand paging*: chỉ nạp trang vào bộ nhớ chính khi và chỉ khi có tham chiếu đến trang đó (i.e.: chỉ nạp theo yêu cầu mà thôi)
    - Khi process mới thực thi thì có nhiều page fault nhưng sau đó số page fault sẽ giảm dần
  - *Pre-paging* nạp trước các trang nhớ: dựa trên tính locality
- Placement policy
  - Xác định vị trí đặt một trang nhớ của process trong bộ nhớ thực
  - Với hệ thống segmentation thuần túy: first-fit, next fit...
  - Với hệ thống paging hoặc kết hợp pagin/segmentation:
    - Phần cứng quản lý bộ nhớ quyết định vị trí đặt trang, tuy nhiên vì các trang nhớ có kích thước như nhau nên không gặp phải khó khăn gì

Replacement Policy

- Giải quyết vấn đề chọn lựa một trang trong bộ nhớ chính sẽ bị thay thế khi có một trang mới cần nạp vào bộ nhớ chính. Tình huống này xảy ra khi bộ chính đầy (không còn frame trống nào)
- Không phải toàn bộ trang trong bộ nhớ đều có thể được thay, có một số frames bị locked
  - Đa số các kernel lưu giữ các thông tin điều khiển và các cấu trúc dữ liệu quan trọng của hệ thống trong các locked frames
- Các giải thuật thay thế trang
  - Least recently used (LRU)
  - First-in, first-out (FIFO)
  - Clock
- Các giải thuật thay thế trang phụ thuộc vào *resident set* (số frame cấp cho mỗi process)

Hai vấn đề chủ yếu

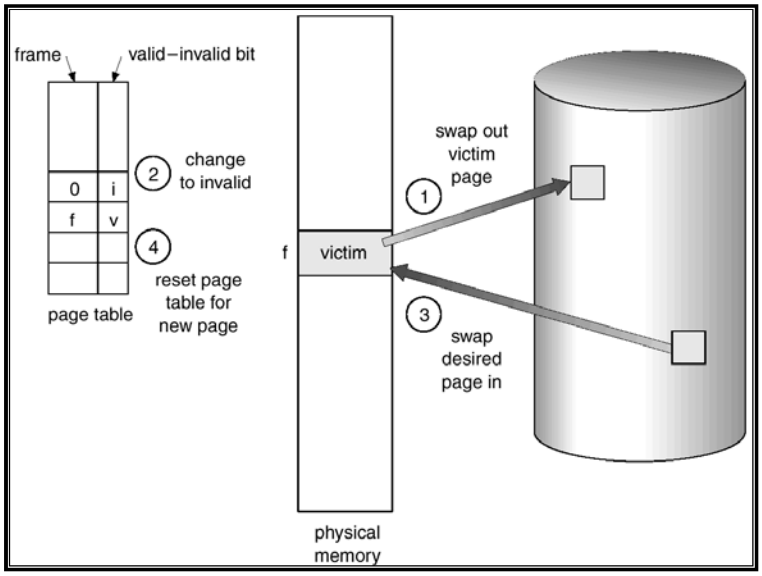
- Frame-allocation algorithm
  - Cấp phát cho mỗi process bao nhiêu frame bộ nhớ thực?
- Page-replacement algorithm
  - Chọn frame sẽ được thay thế trang nhớ
  - Mong muốn có mức độ page-fault nhỏ nhất
  - Lượng giá một giải thuật bằng cách thực thi giải thuật trên một chuỗi tham chiếu bộ nhớ (memory reference) và tính số lần xảy ra page fault

- Thứ tự tham chiếu các địa chỉ nhớ (với page size = 100):
  - **01**00, **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*

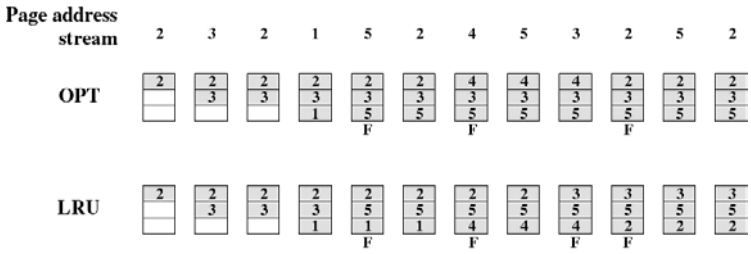
  - **1**, **4**, **1**, **6**, **1**, **1**, **1**, **1**, **6**, **1**, **1**, **1**, **1**, **6**, **1**, **1**, **1**, **1**, **6**, **1**, **1.**

Quá trình thay thế trang nhớ



Least Recently Used (LRU)

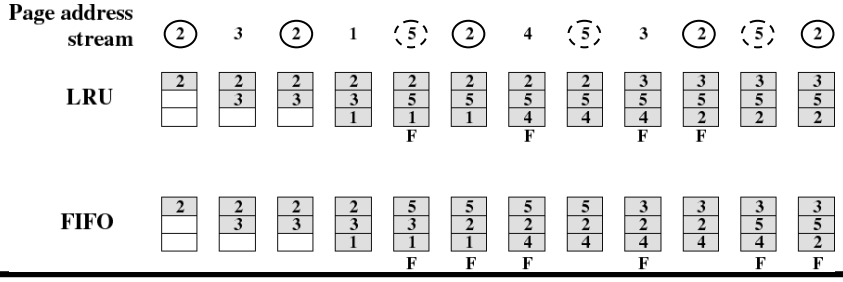
- Thay thế trang nhớ không được tham chiếu lâu nhất
- Ví dụ: một process có 5 trang; resident set = 3



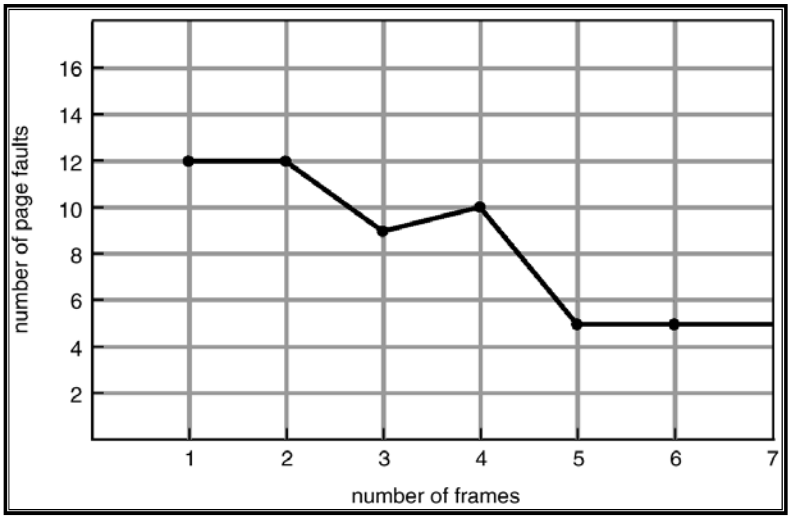
- Mỗi trang được ghi nhận (trong bảng phân trang) thời điểm được tham chiếu ⇒ trang LRU là trang nhớ có giá trị thời gian tham chiếu nhỏ nhất (cần một chi phí tìm kiếm trang nhớ LRU này mỗi khi có page fault)
- Do vậy, LRU cần sự hỗ trợ rất lớn của phần cứng và chi phí khá cao cho việc tìm kiếm. Tuy nhiên, rất ít hệ thống máy tính cung cấp đủ sự hỗ trợ phần cứng cho chiến lược LRU

Giải thuật FIFO

- Xem các frame được cấp phát cho process như là circular buffer
  - Khi bộ đệm đầy, trang nhớ cũ nhất sẽ được thay thế: first-in, first-out
  - Một trang nhớ hay được dùng thông thường sẽ là trang cũ nhất ⇒ hay bị thay thế bởi giải thuật FIFO
  - Hiện thực đơn giản: chỉ cần một con trỏ xoay vòng các frame của process
- So sánh FIFO và LRU



Belady's Anomaly

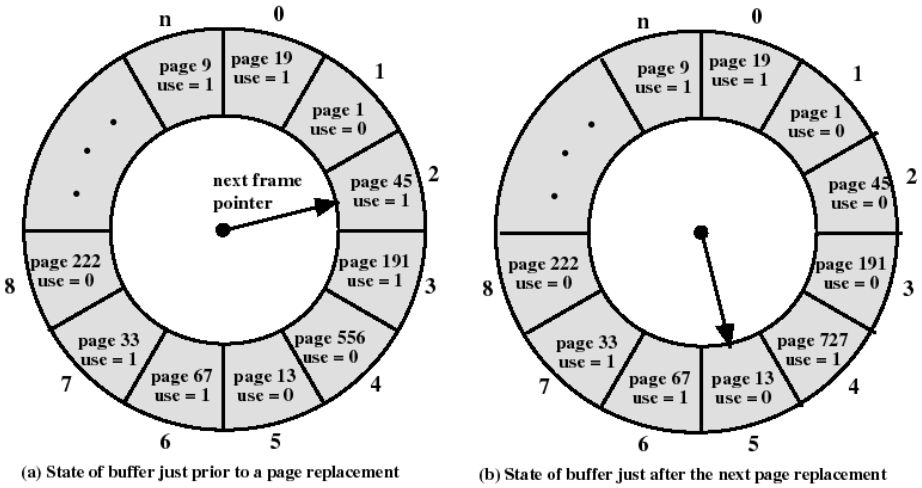


- Bất thường: số page-fault gia tăng khi resident set tăng

Giải thuật Clock (second-chance)

- Resident set cấp cho mỗi process được xem như một bộ đệm xoay vòng (circular buffer)
- Khi một trang được thay, con trỏ sẽ chỉ đến frame kế tiếp trong buffer
- Mỗi frame có một ***use bit***. Bit này được thiết lập trị 1 khi
  - Một trang được nạp lần đầu vào frame
  - Trang chứa trong frame được tham chiếu
- Khi cần thay thế một trang nhớ, trang nhớ nằm trên frame đầu tiên có use bit bằng 0 sẽ được thay thế.
  - Trong suốt quá trình tìm trang nhớ thay thế, tất cả use bit được reset về 0

Ví dụ về giải thuật clock



(a) State of buffer just prior to a page replacement

(b) State of buffer just after the next page replacement

So sánh Clock, FIFO và LRU

Page address stream	2	3	2	1	5	2	4	5	3	2	5	2
LRU	2	2 3	2 3	2 3 1	2 5 1 F	2 5 1	2 5 4 F	2 5 4	3 5 4 F	3 5 2 F	3 5 2	3 5 2
FIFO	2	2 3	2 3	2 3 1	5 3 1 F	5 2 1 F	5 2 4 F	5 2 4	3 2 4 F	3 2 4	3 5 4 F	3 5 2 F
CLOCK	2*	2* 3*	2* 3*	2* 3* 1*	5* 3 1 F	5* 2* 1 F	5* 2* 4* F	5* 2* 4*	3* 2 4 F	3* 2* 4	3* 2 5* F	3* 2* 5*

- Dấu *: use bit tương ứng được thiết lập trị 1
- Giải thuật Clock bảo vệ các trang thường được tham chiếu bằng cách thiết lập use bit bằng 1 với mỗi lần tham chiếu
- Một số kết quả thực nghiệm cho thấy clock có hiệu suất gần với LRU

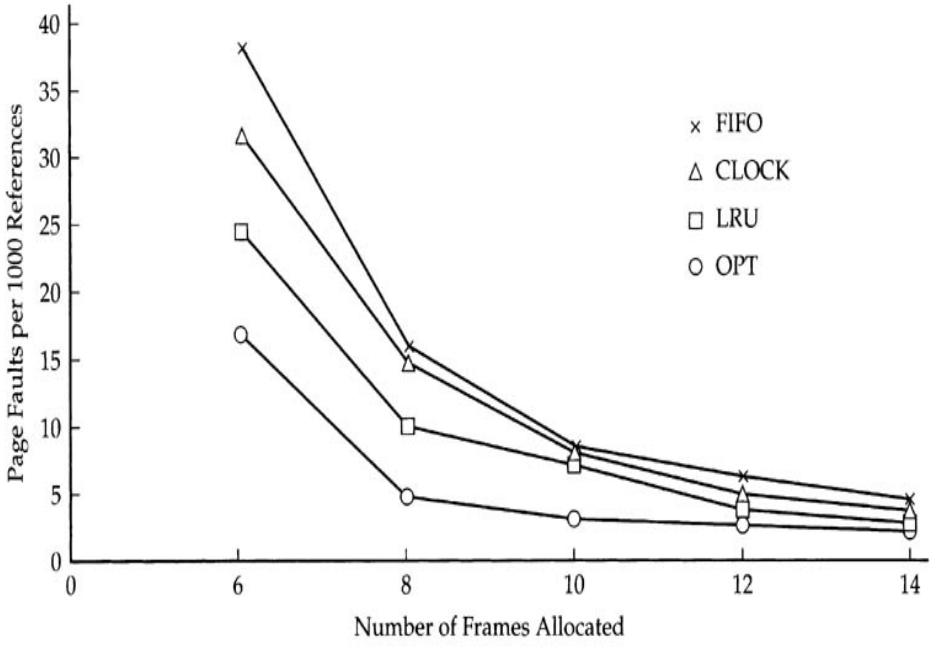


Figure 8.15 Comparison of Fixed-Allocation, Local Page Replacement Algorithms

Kích thước Resident Set

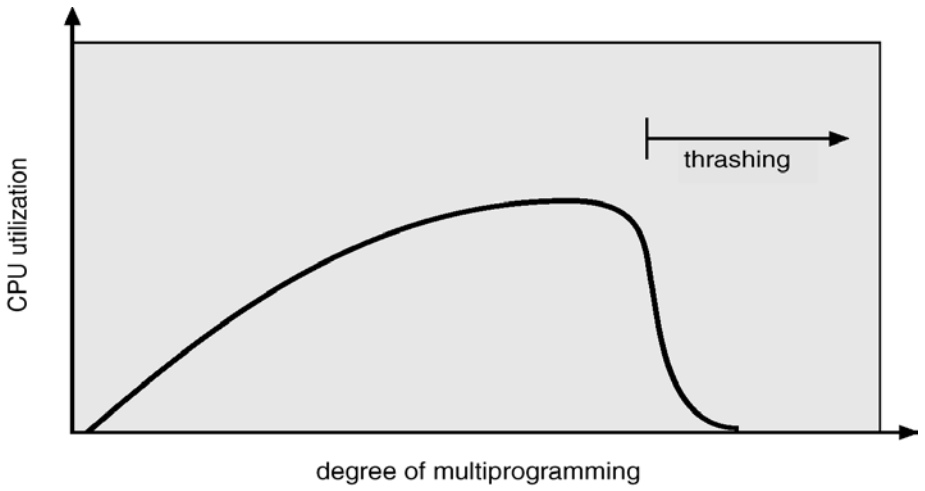
- OS phải quyết định cấp cho mỗi process bao nhiêu frame bộ nhớ.
  - Nếu quá ít frame ⇒ nhiều page fault
  - Cấp nhiều frame ⇒ giảm mức độ multi-programming
- Chiến lược cấp phát tĩnh (fixed-allocation)
  - Số frame cấp cho mỗi process không đổi, được xác định vào thời điểm loading và có thể tùy thuộc vào từng ứng dụng (kích thước,...)
- Chiến lược cấp phát động (variable-allocation)
  - Số frame cấp cho mỗi process có thể thay đổi
    - Nếu tỷ lệ page-fault cao ⇒ cấp thêm frame
    - Nếu tỷ lệ page-fault thấp ⇒ giảm bớt
  - OS phải mất chi phí để ước định các process

Thrashing

- Nếu một process không có đủ số frame cần thiết thì tỉ suất page-fault rất cao. Điều này khiến giảm hiệu suất CPU rất nhiều
- Ví dụ một vòng lặp N lần, mỗi lần tham chiếu đến địa chỉ nằm trong 4 trang nhớ trong khi đó process chỉ được cấp 3 frames.
- Thrashing ≡ hiện tượng một process bị hoán chuyển vào/ra liên tục các trang nhớ.

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

Thrashing Diagram



Mô hình Locality

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

$\Rightarrow$ dựa trên mô hình locality

- Locality là một tập các trang thường hay được tham chiếu cùng nhau (chuỗi con tham chiếu các trang này thường xuất hiện)
  - Trong ví dụ trước, locality sẽ bao gồm 4 trang
- Một chương trình thường gồm nhiều locality và trong quá trình thực thi, process sẽ chuyển từ locality này sang locality khác
  - Ví dụ khi một thủ tục được gọi thì sẽ có một locality mới. Trong locality này, tham chiếu bộ nhớ bao gồm lệnh của thủ tục, biến cục bộ và một phần biến toàn cục. Khi thủ tục kết thúc, process sẽ thoát khỏi locality này (và có thể quay lại sau này).

- Vì sao hiện tượng thrashing xuất hiện ?
  Σ size of locality > memory size

Mô hình Working Set

- Dựa trên giả thiết về locality
- Xác định xem process thực sự sử dụng bao nhiêu frame?
- Định nghĩa $\Delta \equiv$ working-set window $\equiv$ số tham chiếu trang nhớ cố định. Ví dụ: 10,000 lệnh
- WSSi (working set of Process Pi) =
  tổng số trang được tham chiếu trong khoảng Δ gần nhất
  - Δ quá nhỏ $\Rightarrow$ không đủ bao phủ toàn bộ locality.
  - Δ quá lớn $\Rightarrow$ bao phủ nhiều locality.
  - $\Delta = \infty \Rightarrow$ bao gồm toàn bộ chương trình.
- D = Σ WSSi $\equiv$ tổng số frame yêu cầu
  - D > m (số frame của hệ thống) $\Rightarrow$ xảy ra thrashing
  - Nếu D > m $\Rightarrow$ suspend một trong các process.

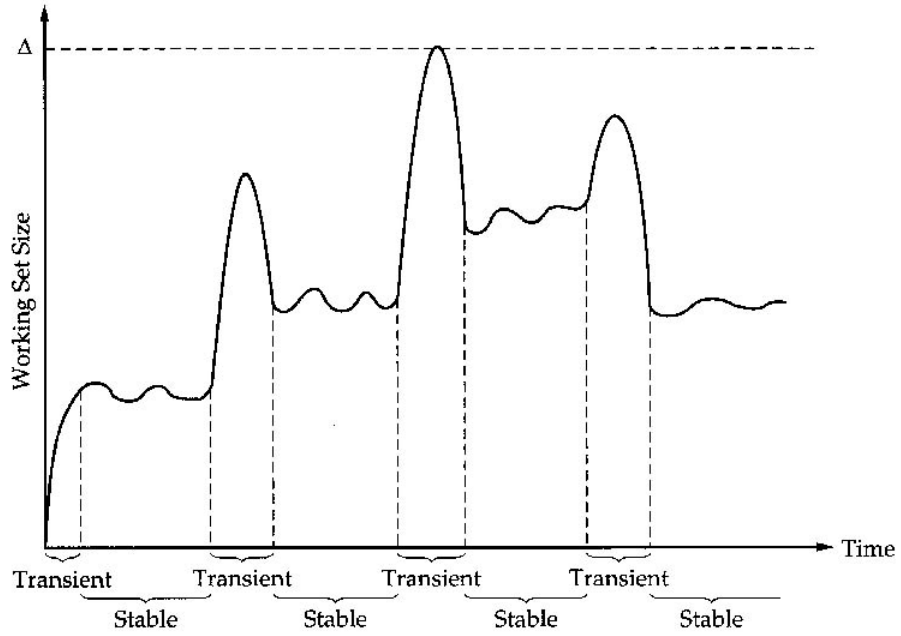


Figure 8.18 Typical Graph of True Working Set Size [MAEK87]

Minh họa mô hình Working-Set

page reference table

. . . 2 6 1 5 7 7 7 7 5 1 6 2 3 4 1 2 3 4 4 4 3 4 3 4 4 4 1 3 2 3 4 4 4 3 4 4 4 . . .

Δ Δ

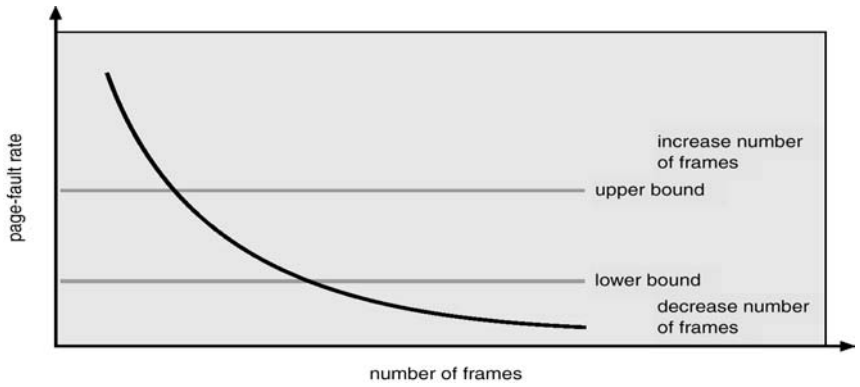
t_1 t_2

$WS(t_1) = \{1,2,5,6,7\}$ $WS(t_2) = \{3,4\}$

Theo dõi các Working Set

- Dùng interval timer kết hợp với reference bit
- Ví dụ: Δ = 10,000
  - Timer interrupt định kỳ 5000 đơn vị thời gian.
  - Giữ trong bộ nhớ 2 bit cho mỗi trang nhớ.
  - Khi timer interrupt xảy ra, copy và reset trị tất cả các reference bits = 0.
  - Nếu 1 trong 2 bit trên = 1 $\Rightarrow$ trang thuộc về working set.
- Cải tiến = 10 bit và interrupt timer định kỳ 1000 đơn vị thời gian.
- Tốn chi phí theo dõi working set và xử lý các timer định kỳ

Mô hình tần suất Page-Fault



- Dùng tần suất page-fault frequency để điều chỉnh mức độ page-fault rate.
  - Tần suất quá thấp $\Leftrightarrow$ process có quá nhiều frame, giảm bớt.
  - Tần suất quá cao $\Leftrightarrow$ process cần thêm frame.