ĐẠI HỌC ĐÀ NẴNG
TRƯỜNG ĐẠI HỌC BÁCH KHOA

NGUYÊN LÝ HỆ ĐIỀU HÀNH

Khoa Công nghệ thông tin
ThS. Nguyễn Thị Lệ Quyên

Nội dung môn học

- Giới thiệu
- Tiến trình và luồng
- ***Thi giữa kỳ***
- **Quản lý bộ nhớ**
- Vào/ Ra
- Hệ thống file
- Thực hành

QUẢN LÝ BỘ NHỚ

Nội dung

- Bộ nhớ chính (main memory)
- Bộ nhớ ảo (virtual memory)

Bộ nhớ chính

- **Background**
- Cấp phát bộ nhớ liền kề
- Paging
- Structure of the page table
- Swapping
- Example
- Summary

Cần nắm

- Giải thích sự khác nhau giữa logical address và physical address và vai trò của MMU (memory management unit) trong đổi địa chỉ (translating addresses)
- Áp dụng chiến thuật first-fit, best-fit và worst-fit để cấp phát bộ nhớ liên tục
- Giải thích và phân biệt giữa internal và external fragmentation
- Đổi logical sang physical address trong hệ thống phân trang (paging system) bao gồm TLB (translation look-aside buffer)
- Mô tả phân trang phân tầng, phân trang hashed, và inverted page tables
- Mô tả đổi địa chỉ cho kiến trúc máy tính IA-32, x86-64, và ARMv8

Background

- CPU nạp lệnh từ bộ nhớ theo giá trị của PC (program counter), sau đó câu lệnh được decode và execute, kết quả sẽ lưu trữ trở lại trong bộ nhớ
- Basic hardware
- Address Binding
- Logical vs Physical Address Space
- Dynamic Loading
- Dynamic Linking & Shared Libraries

Basic hardware

- Bộ nhớ chính và register là nơi lưu trữ để CPU có thể truy cập trực tiếp
- Nếu dữ liệu không nằm trong bộ nhớ chính hay register thì chúng phải được chuyển vào trước khi CPU thực hiện

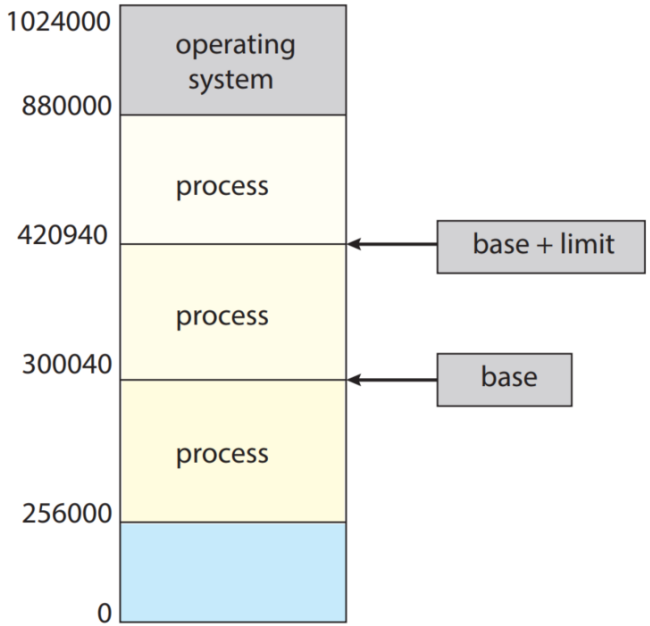


Figure 9.1 A base and a limit register define a logical address space.

Basic hardware

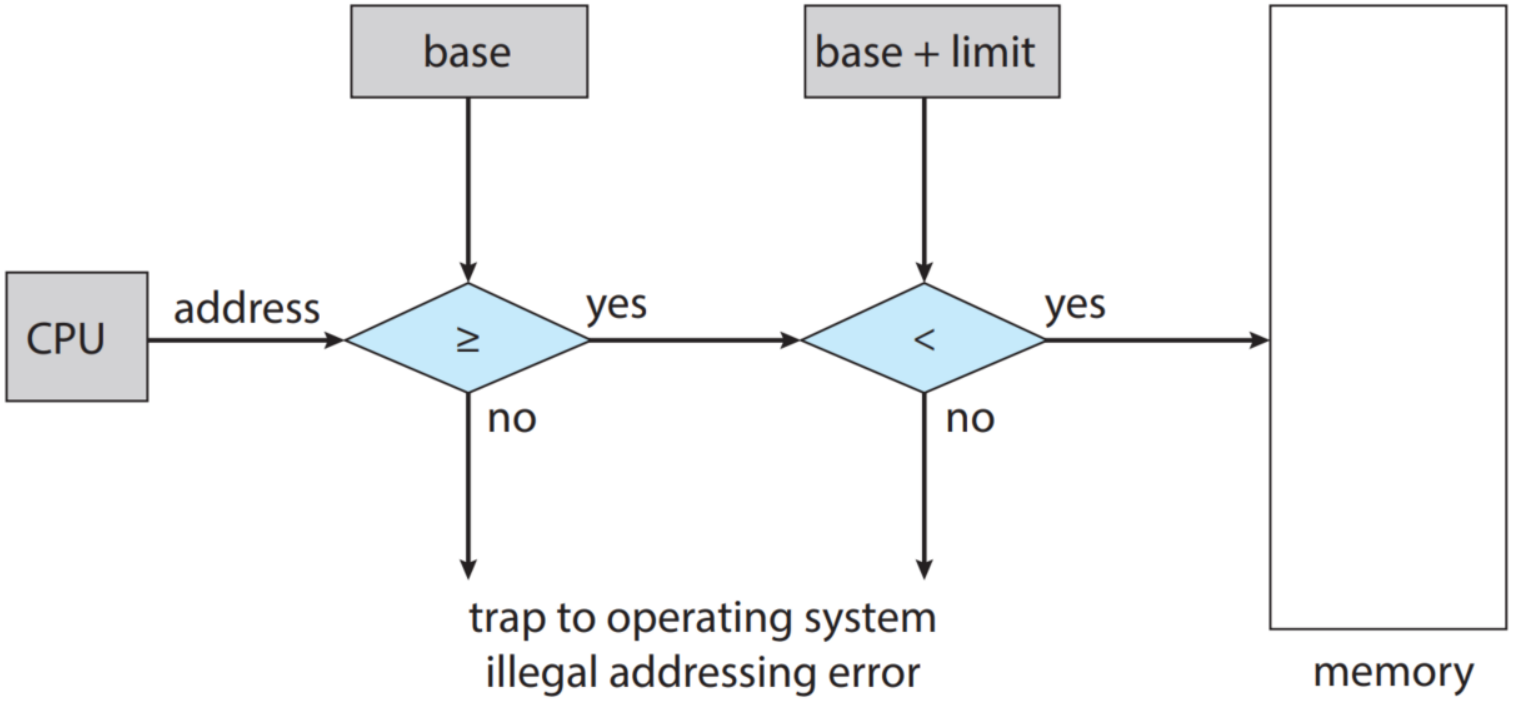


Figure 9.2 Hardware address protection with base and limit registers.

Address Binding

- Việc liên kết các lệnh và dữ liệu với các địa chỉ bộ nhớ có thể được thực hiện ở bất kỳ bước nào trong quá trình thực hiện
  - Compile time - tiến trình nằm trong memory, sinh ra absolute code
  - Load time – nếu không được biết ở compile time thì compiler sẽ sinh ra relocatable code
  - Execution time

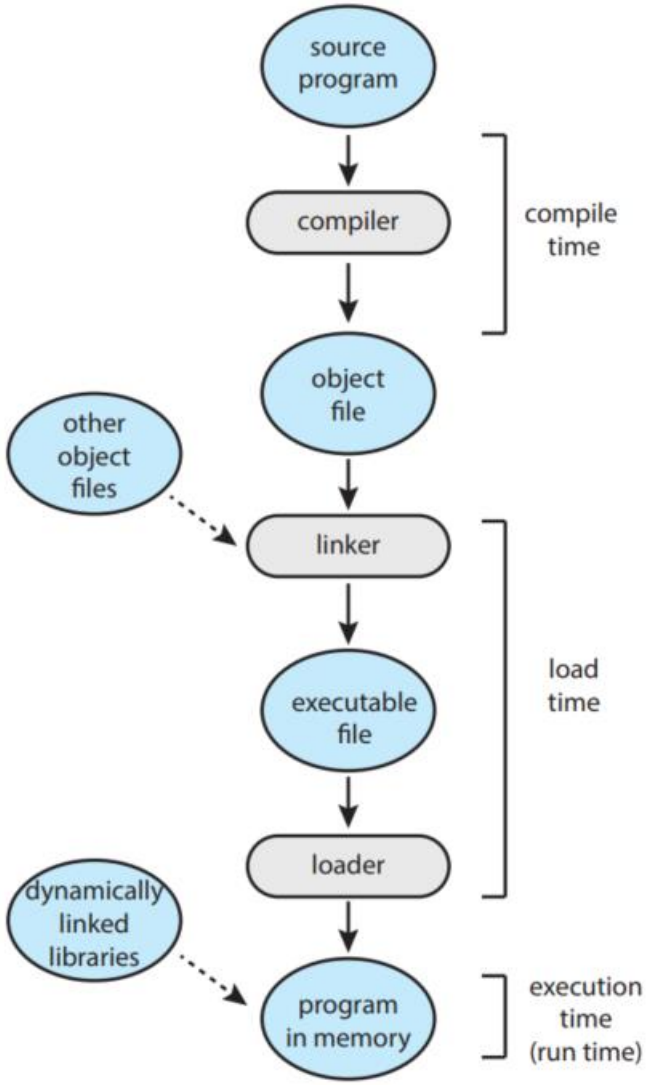


Figure 9.3 Multistep processing of a user program.

Logical vs Physical Address Space

- Địa chỉ sinh ra bởi CPU là logical address
- Địa chỉ được sử dụng bởi memory unit để nạp vào memory-address register là physical address
- Binding address tại thời điểm compile hoặc load sẽ tạo ra các địa chỉ logical và physical giống nhau
- Binding address tại thời điểm thực thi tạo ra các địa chỉ logical và physical khác nhau. Trong trường hợp này, logical address được xem là virtual address
- Tập hợp tất cả các địa chỉ logical được tạo bởi 1 chương trình là logical address space, tương ứng với địa chỉ physical là physical address space ➔ tại thời điểm thực thi logical address space khác physical address space

Logical vs Physical Address Space

- MMU (hardware device) đảm nhận vai trò map từ virtual address sang physical address
- Chương trình người dùng không bao giờ truy cập được vào địa chỉ vật lý thực (real physical address), chỉ làm việc với logical address

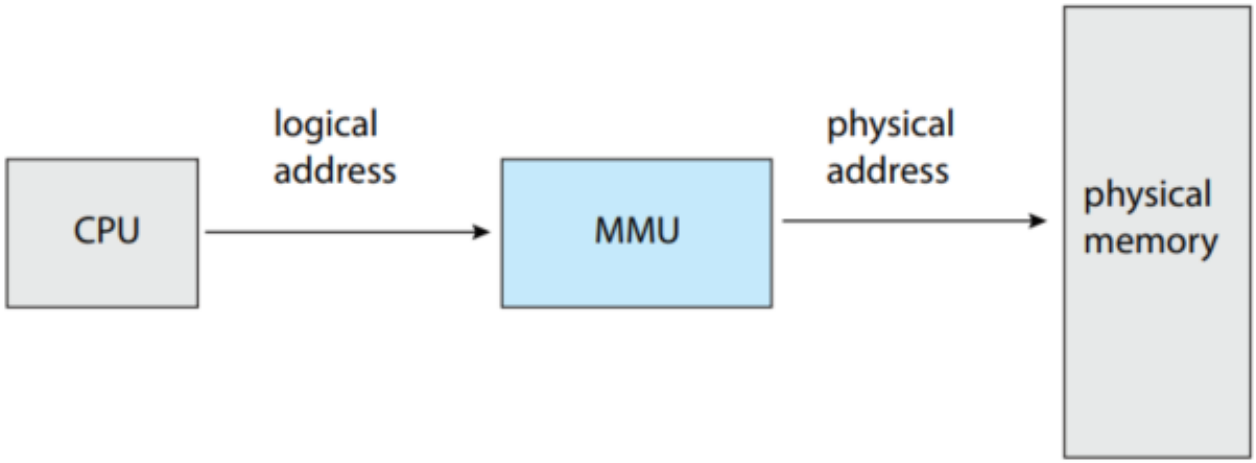


Figure 9.4 Memory management unit (MMU).

Logical vs Physical Address Space

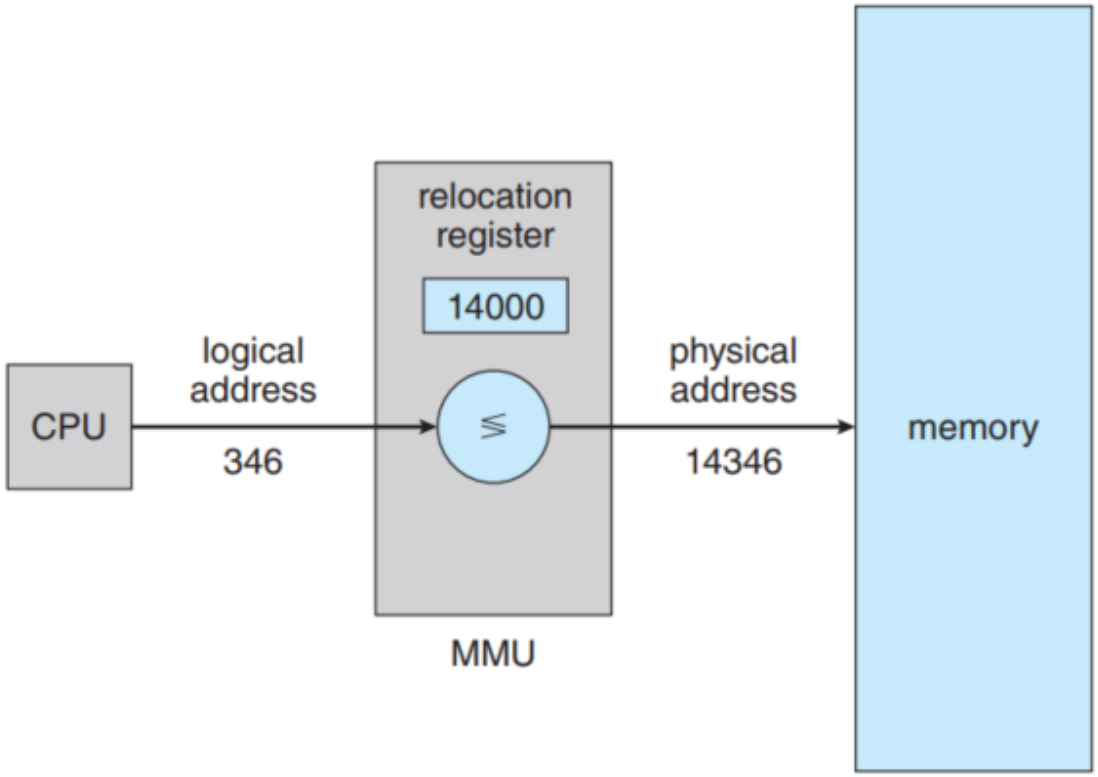


Figure 9.5 Dynamic relocation using a relocation register.

Dynamic Loading

- Toàn bộ chương trình và tất cả dữ liệu của một tiến trình cần nằm trong bộ nhớ vậy lý để tiến trình có thể thực thi → kích thước của 1 tiến trình bị giới hạn bởi kích thước của bộ nhớ vật lý
- Sử dụng dynamic loading (tải động) để tận dụng không gian bộ nhớ tốt hơn
- Phần bộ nhớ sẽ không được tải cho đến khi nó được gọi/ sử dụng
- Đặc biệt hữu ích khi cần số lượng lớn code để xử lý các trường hợp xảy ra không thường xuyên, vd như phần kiểm soát lỗi

Dynamic Linking & Shared Libraries

- Dynamically linked libraries (DLLs) là hệ thống thư viện được liên kết với chương trình người dùng khi chương trình chạy
- Những thư viện này có thể được chia sẻ bởi nhiều tiến trình mà chỉ có 1 DDL trong main memory → shared libraries

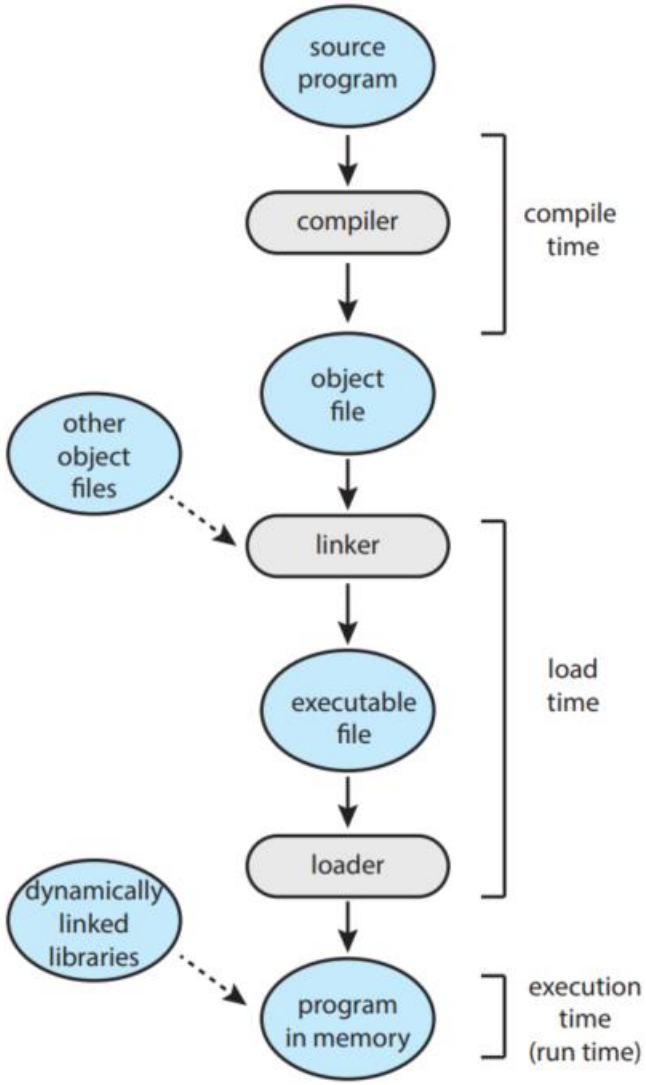


Figure 9.3 Multistep processing of a user program.

Bộ nhớ chính

- Background
- **Contiguous Memory Allocation (Cấp phát bộ nhớ liền kề)**
- Paging
- Structure of the page table
- Swapping
- Example
- Summary

Cấp phát bộ nhớ liền kề

- Bộ nhớ được chia thành 2 phần
  - Hệ điều hành – có thể được đặc ở địa chỉ bộ nhớ thấp hoặc cao (nhiều OS như Linux và Windows đặt OS tại địa chỉ bộ nhớ cao)
  - Tiến trình người dùng
- Không gian địa chỉ của 1 tiến trình được đặt liền kề nhau rồi mới tới tiến trình khác

Bảo vệ bộ nhớ

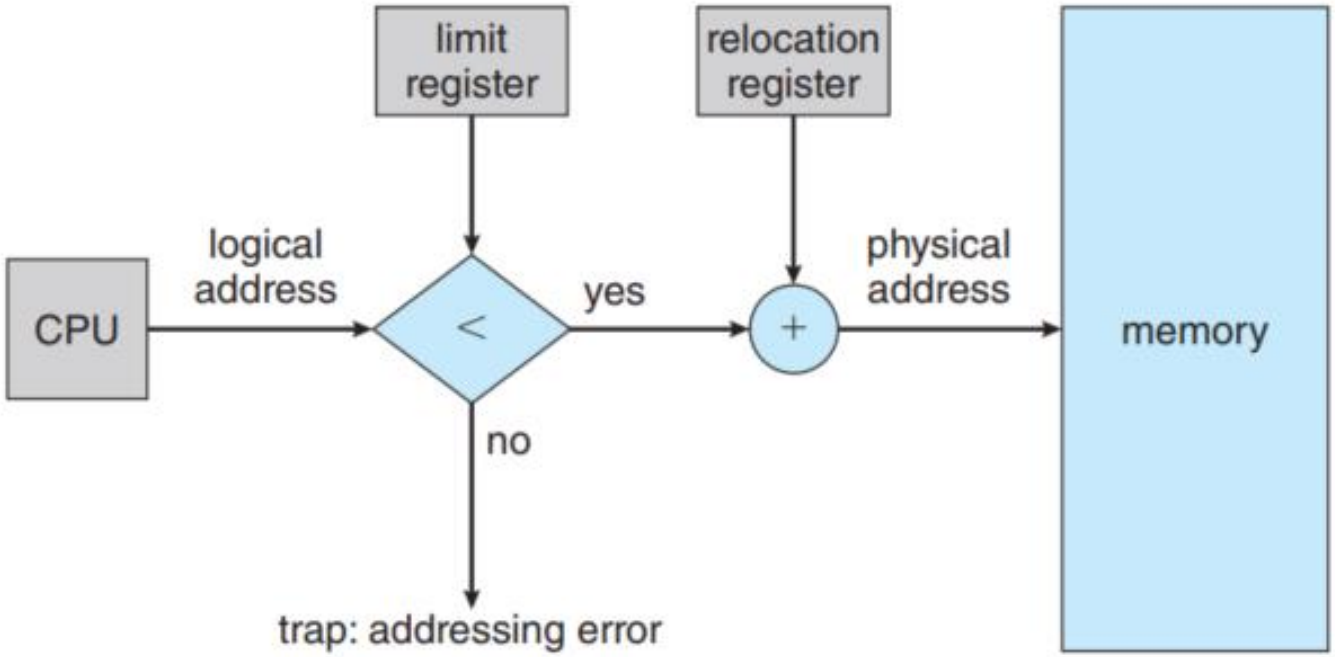


Figure 9.6 Hardware support for relocation and limit registers.

Cấp phát bộ nhớ

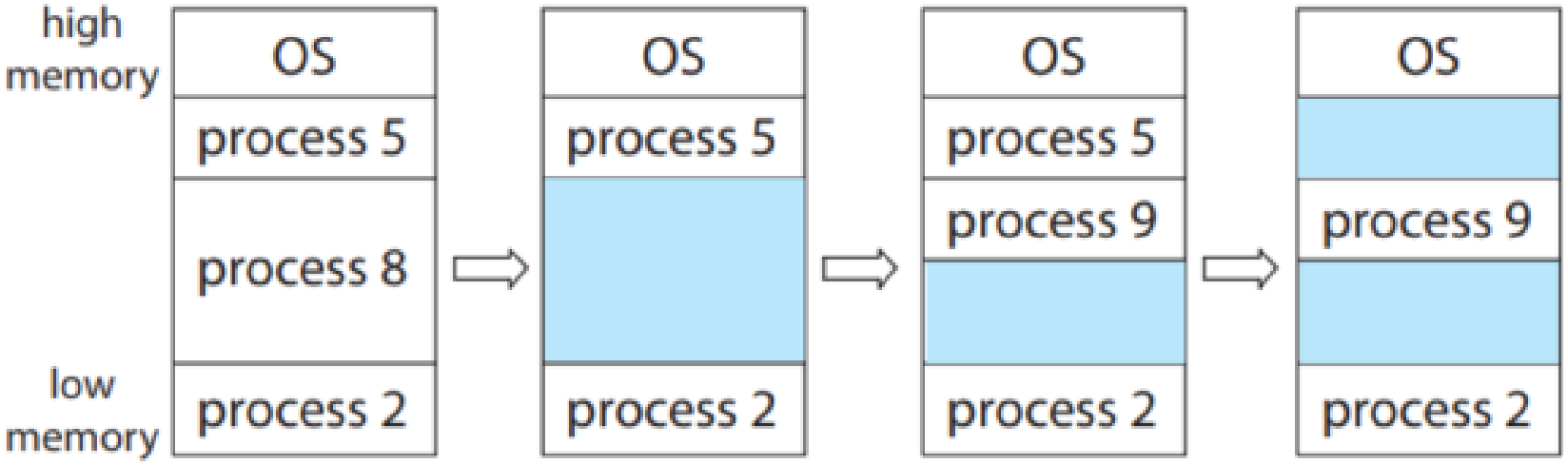


Figure 9.7 Variable partition.

Cấp phát bộ nhớ

- Các cách lựa chọn vùng cấp phát:
  - First-fit – cấp phát vùng trống đầu tiên đủ rộng cho tiến trình
  - Best-fit – cấp phát vùng trống nhỏ nhất đủ rộng cho tiến trình
  - Worst-fit – cấp phát vùng trống rộng nhất đủ rộng cho tiến trình
- Mô phỏng cho thấy first-fit và best-fit tốt hơn worst-fit khi xét tới việc giảm thời gian và mức độ sử dụng bộ nhớ
  - First-fit nhanh hơn best-fit

Phân mảnh (fragmentation)

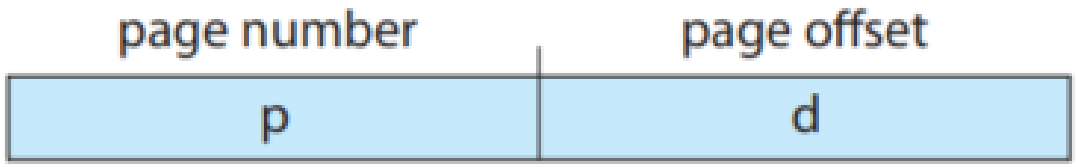
- Cả first-fit và best-fit đều gặp tình trạng phân mảnh ngoài (external fragmentation) – tình trạng tổng bộ nhớ trống đủ cho tiến trình tuy nhiên chúng lại không liên tục, bị phân nhỏ ở nhiều nơi trong bộ nhớ
- Giải pháp 1 là nén (compaction) – dồn những vùng nhớ trống (free) thành một
  - Tốn kém
- Giải pháp 2 là phân trang (paging) – phân nhỏ tiến trình ra và cấp phát nó ở bất cứ nơi đâu có bộ nhớ trống
  - Giải pháp được sử dụng phổ biến nhất

Bộ nhớ chính

- Background
- Cấp phát bộ nhớ liền kề
- **Paging**
- Structure of the page table
- Swapping
- Example
- Summary

Paging

- Phương pháp căn bản nhất là chia bộ nhớ vật lý thành các khối có kích thước bằng nhau gọi là **frames** và chia bộ nhớ logic thành các khối có kích thước tương tự gọi là **pages**
- Khi tiến trình thực thi, mỗi page được nạp vào frame trống
- Mỗi địa chỉ được tạo ra bởi CPU được chia thành 2 phần: page number và page offset



Paging

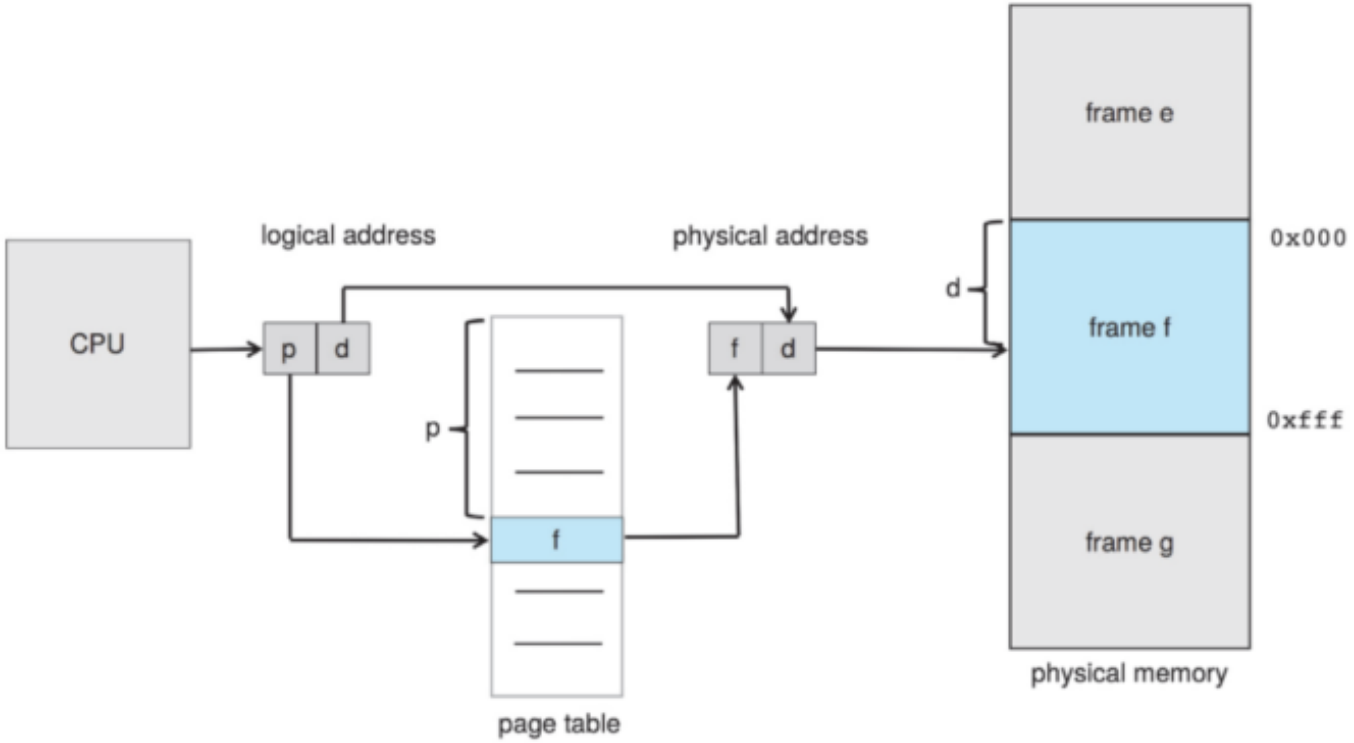


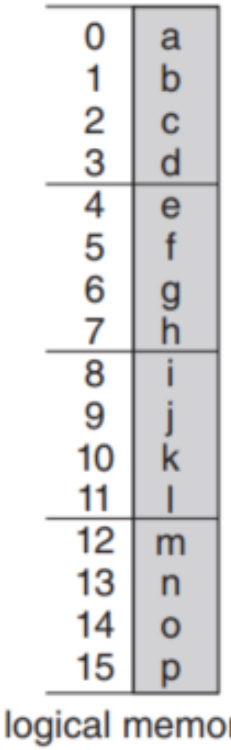
Figure 9.8 Paging hardware.

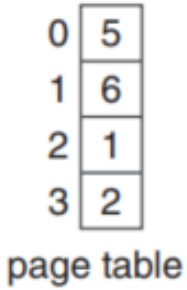
Paging

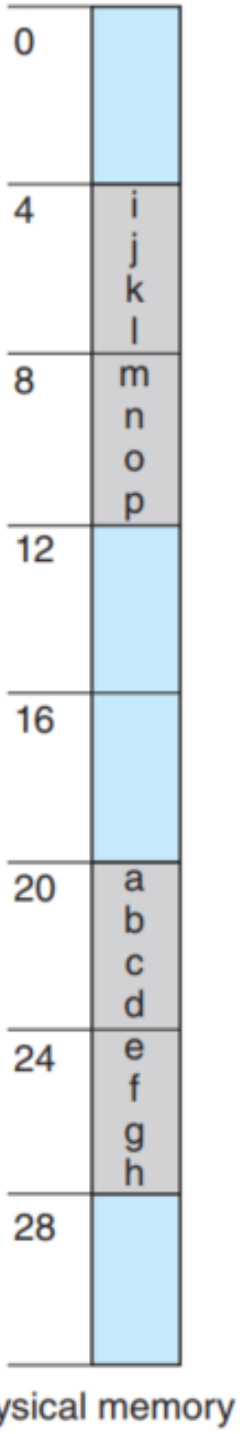
- Page size được quy định bởi phần cứng, thường giữa 4KB và 1GB mỗi page phụ thuộc vào kiến trúc máy tính
- Nếu kích thước logical address space là 2^m và page size là 2^n bytes thì $m - n$ bits đầu tiên của địa chỉ logical chỉ page number và n bits sau chỉ page offset

Paging

- n=2, m=4
- Page size 4bytes, physical memory 32bytes (8pages)







- Page0 ở frame5
  - logical address 0 map tới physical address 20 [=(5x4)+0]
  - Logical address 3 map tới physical address 23[=(5x4)+3]

Paging

- Khi sử dụng paging, sẽ không bị phân mảnh ngoài (external fragmentation) nhưng bị phân mảnh trong (internal fragmentation) – ví dụ page size là 2,048bytes, 1 tiến trình kích thước 72,766bytes sẽ cần 35 pages và 1,086 bytes. Nó sẽ cần 36pages nhưng page cuối sẽ thừa 2,048-1,086=962bytes
- Xem page size trong LINUX
  - $ getconf PAGESIZE

Paging

Figure 9.11 Free frames (a) before allocation and (b) after allocation.

Paging

- Việc chuyển đổi từ địa chỉ logic sang địa chỉ vật lý do hệ điều hành kiểm soát, hoàn toàn ẩn với lập trình viên
  - Lập trình viên nhìn bộ nhớ của chương trình là một không gian duy nhất, không biết về chuyện bộ nhớ bị phân chia
- Hệ điều hành quản lý bộ nhớ vật lý nên cần biết frame nào đã được cấp phát, frame nào đang trống → frame table
- Mỗi tiến trình giữ một page table dùng để chuyển đổi địa chỉ logic sang địa chỉ vật lý, nó cũng được CPU dùng khi tiến trình được cấp phát CPU → paging tăng thời gian context switch

Phần cứng hỗ trợ

- Lưu page table trong main memory và thanh ghi PTBR (page-table base register) trỏ tới page table
  - Nếu có thay đổi trong page table, chỉ cần thay đổi duy nhất 1 thanh ghi → giảm context-switch time
  - Khi truy cập bộ nhớ phải truy cập 2 lần, 1 lần truy cập PTBR, 1 lần truy cập địa chỉ thực tế → làm chậm thời gian truy cập bộ nhớ
- Giải pháp: Sử dụng cache phần cứng đặc biệt, nhỏ gọn, tra cứu nhanh gọi là TLB (translation look-aside buffer)
  - High-speed memory
  - Mỗi entry chưa 2 phần: khóa (hoặc tag) và giá trị
  - Nhỏ gọn (khoảng 32 tới 1024 entry)

Phần cứng hỗ trợ

- TLB chỉ chứa một vài page-table entry. Khi địa chỉ logic được sinh ra bởi CPU, MMU đầu tiên kiểm tra page number có nằm trong TLB hay không.
  - Có – frame number ngay lập tức được sử dụng và truy cập memory
  - Không (TLB miss) – frame number được tìm và sau đó được sử dụng để truy cập bộ nhớ
- Khi TLB đầy, 1 entry sẽ được chọn để thay thế
  - LRU (least recently used)
  - RR
  - Random
  - ...

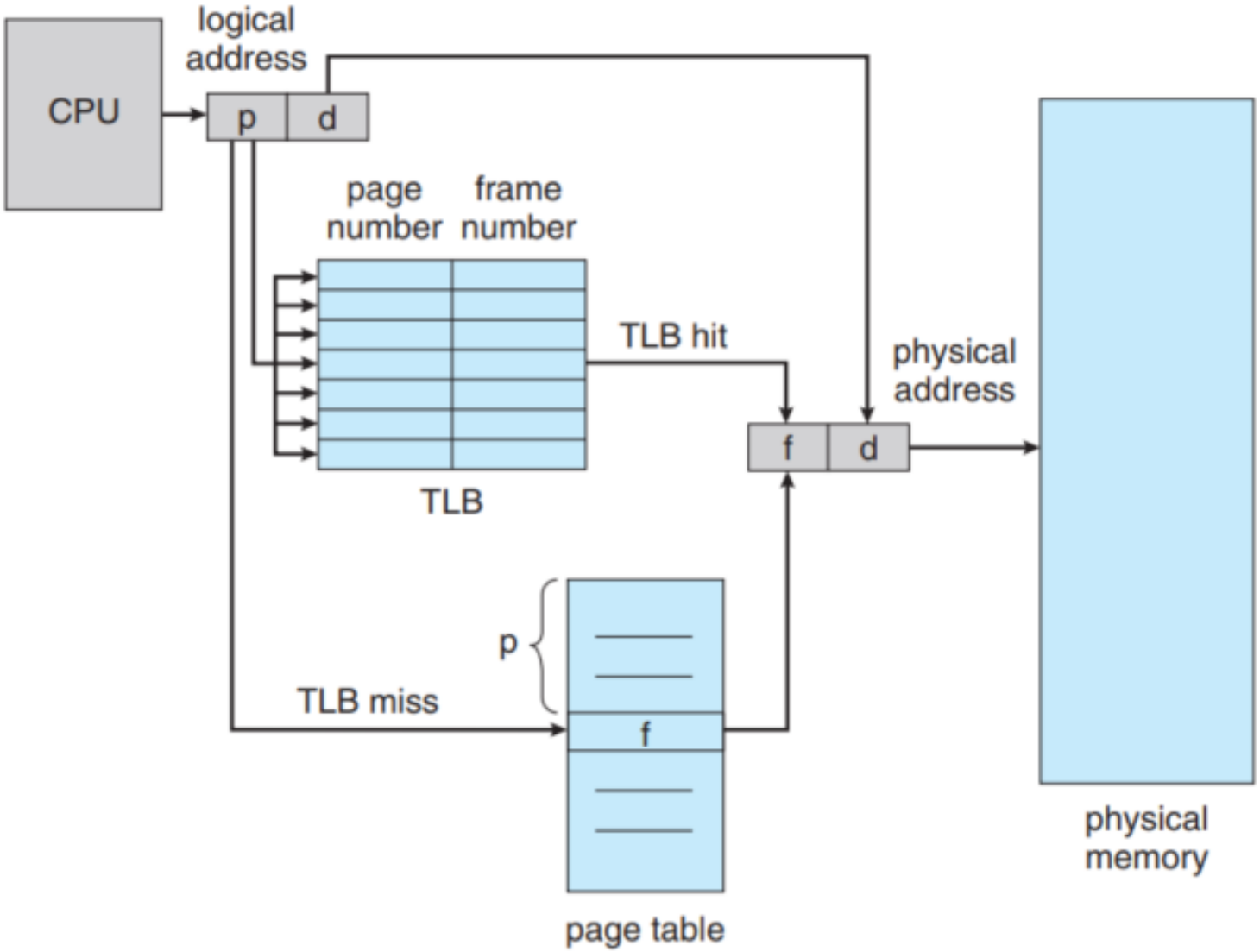


Figure 9.12 Paging hardware with TLB.

Protection

- Việc bảo vệ bộ nhớ trong môi trường phân trang được thực hiện bằng các bit bảo vệ tương ứng với các frame. Các bit này thường được giữ trong page table
- 1 bit có thể xác định một page ở chế độ read-write hay read-only
- Mỗi entry trong page table có thêm 1 bit: valid-invalid bit
  - Valid – page tương ứng thuộc logical address space của tiến trình
  - Invalid – page không thuộc logical address space của tiến trình

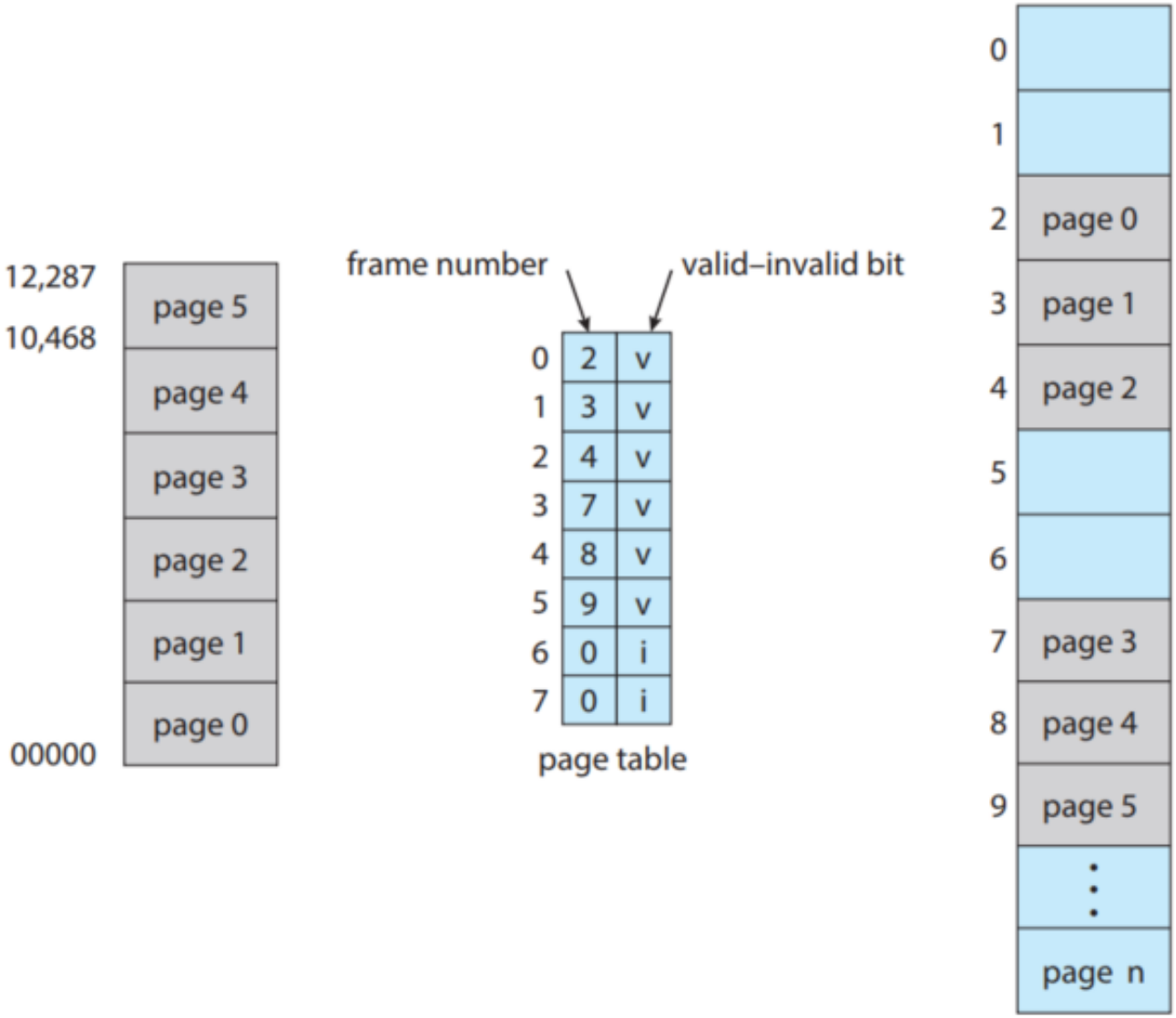


Figure 9.13 Valid (v) or invalid (i) bit in a page table.

Shared pages

- Ưu điểm của paging là có thể chia sẻ code chung – được xem là quan trọng trong môi trường nhiều tiến trình
- Vd trong hệ thống Linux, nhiều tiến trình người dùng sử dụng thư viện C chuẩn libc
  - Có thể tải thư viện vào không gian địa chỉ của mỗi tiến trình. Nếu hệ thống có 40 tiến trình, libc nặng 2MB thì cần 80MB bộ nhớ
  - → có thể chia sẻ để tiết kiệm bộ nhớ

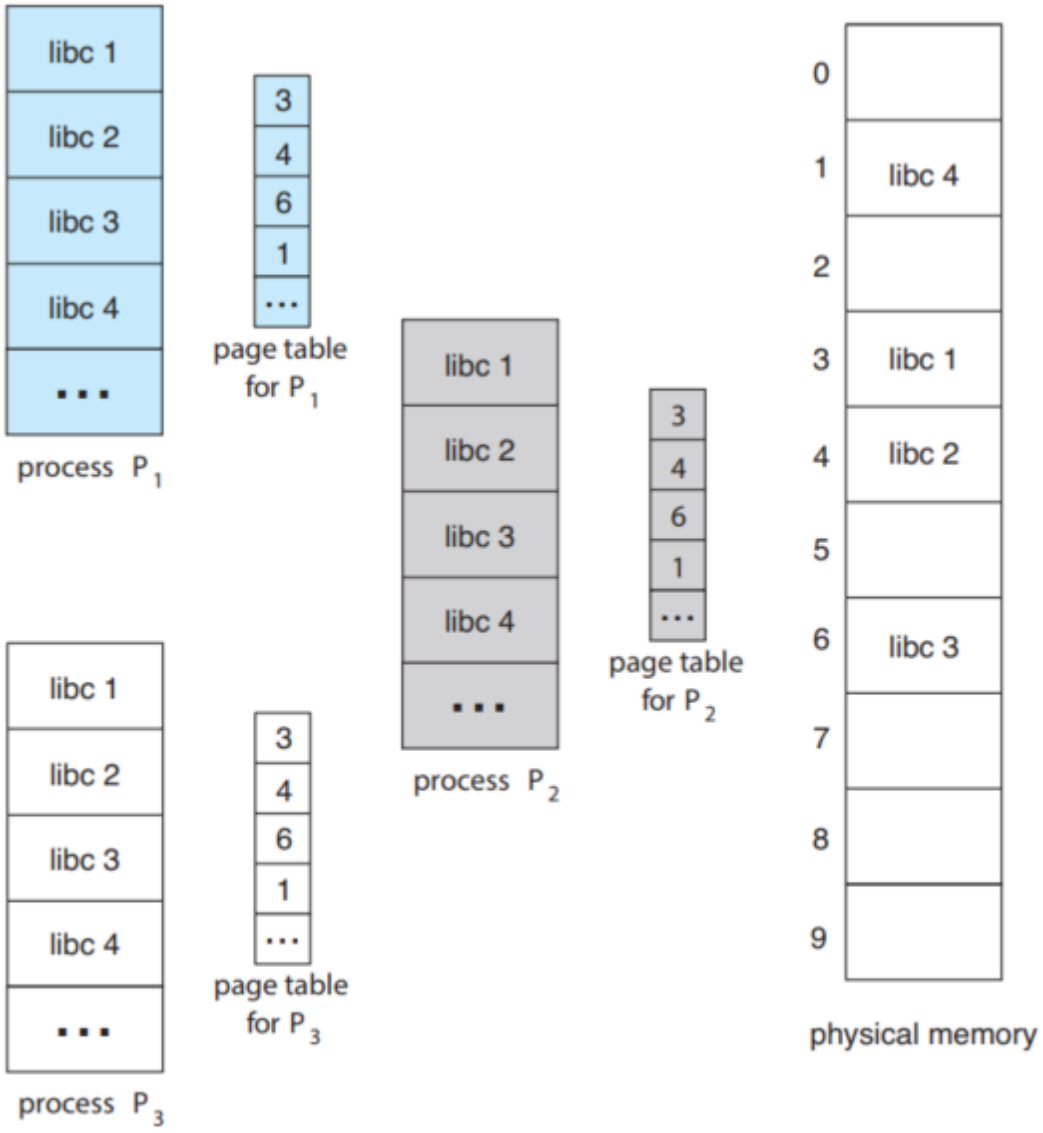


Figure 9.14 Sharing of standard C library in a paging environment.

Bộ nhớ chính

- Background
- Cấp phát bộ nhớ liền kề
- Paging
- **Structure of the page table**
- Swapping
- Example
- Summary

Cấu trúc page table

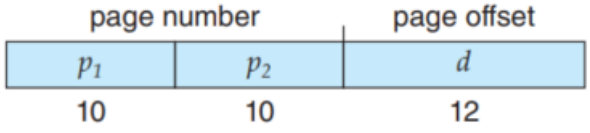
- Hierarchical paging
- Hashed page tables
- Inverted page tables

Hierarchical paging

- Các hệ thống máy tính hiện đại hỗ trợ không gian địa chỉ logic lớn (2^{32} đến 2^{64}) → page table cũng lớn
  - VD hệ thống 32-bit logical address space, page size = 4KB (2^{12}) thì page table có kích thước 1 triệu ($2^{20}=2^{32}/2^{12}$), mỗi entry của table 4 bytes như vậy thì mỗi tiến trình cần 4MB bộ nhớ vt lý để lưu riêng page table
- Không thể cấp phát vùng nhớ liên tục trong bộ nhớ để lưu page table → chia nhỏ page table
  - Two-level paging algorithm
  - n-level paging algorithm

Hierarchical paging

- Xét hệ thống 32-bit logical address space, page size = 4KB
  - Page number 20bits
  - Page offset 12bits



  - Chia page number thành p1 – outer page table, p2 – inner page table

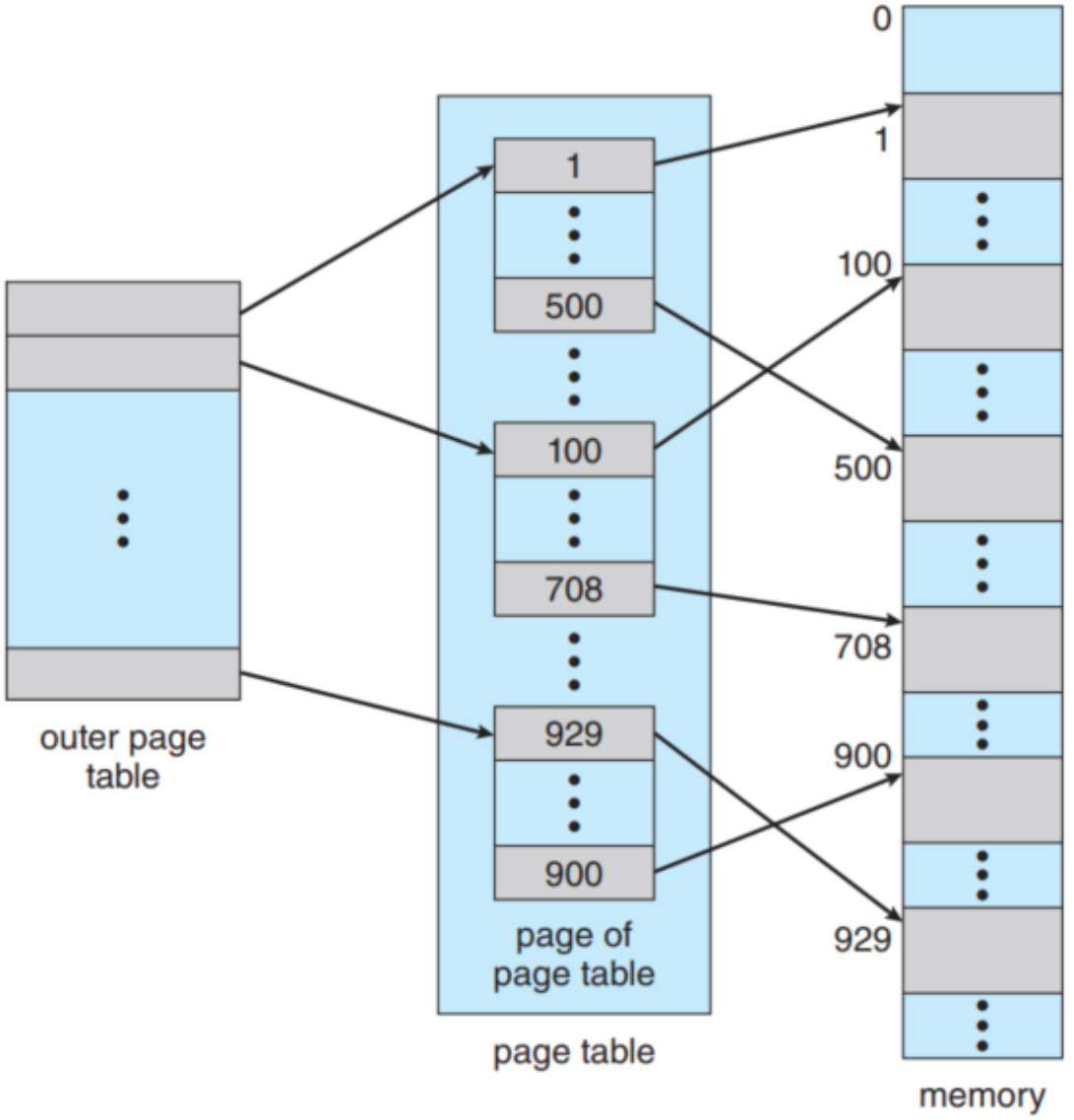


Figure 9.15 A two-level page-table scheme.

Hierarchical paging

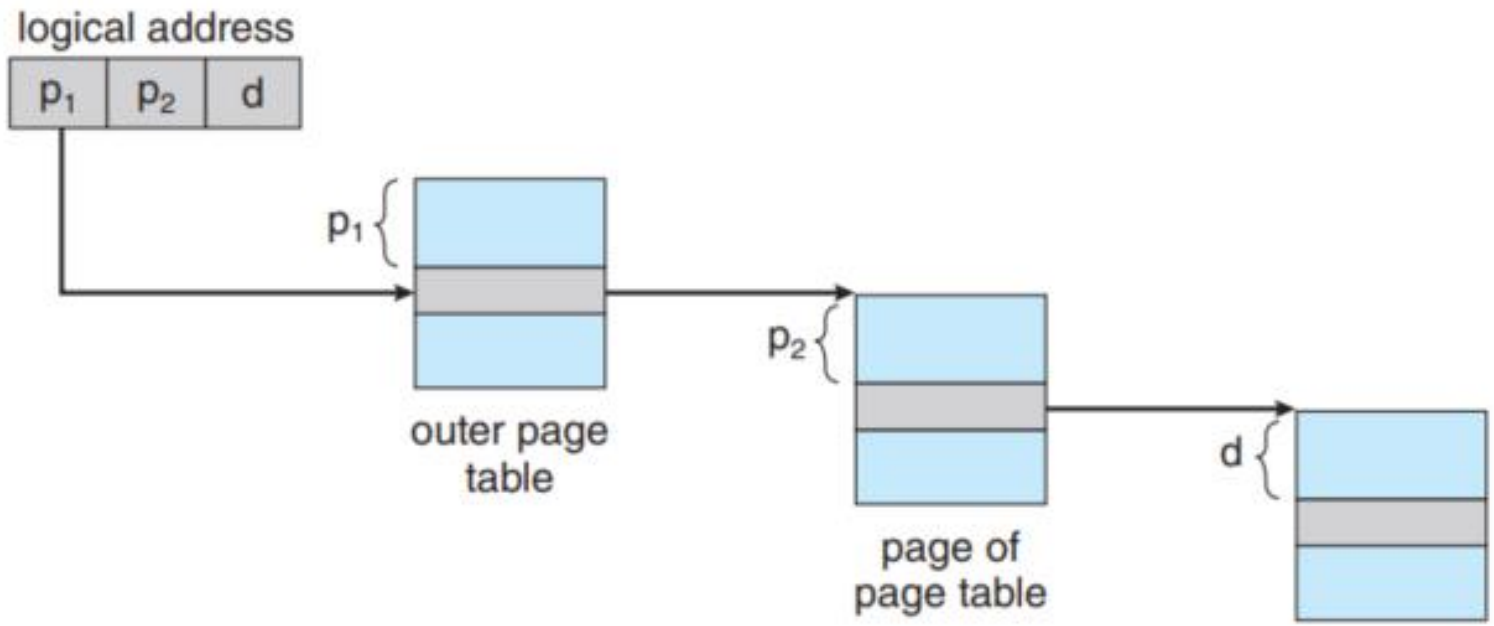
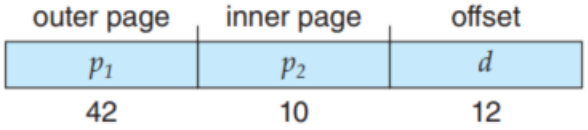


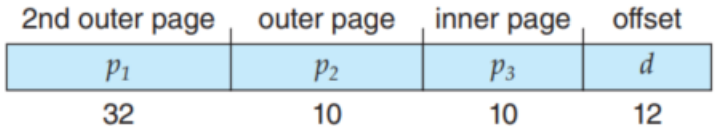
Figure 9.16 Address translation for a two-level 32-bit paging architecture.

Hierarchical paging

- Đối với hệ thống 64-bit logical address space, phương pháp two-level paging không thích hợp
  - Nếu page size = 4KB thì page table có kích thước 2^{52}



  - Outer page table chứa 2^{42} entry hay 2^{44} bytes. Để tránh table lớn, lại chia nhỏ outer page



  - Outer page 2^{34} bytes (16GB) → chia tiếp thành four-level paging
- Đối với hệ thống 64bits, hierarchical paging không thích hợp

Hashed page tables

- Dùng cho hệ thống lớn hơn 32bits address space
- Mỗi entry trong hash table chứa 1 danh sách liên kết các thành phần, mỗi thành phần gồm 3 trường (1) virtual page number, (2) giá trị map tới page frame, (3) con trỏ trỏ tới thành phần tiếp theo
- Thuật toán triển khai như sau:
  - Virtual page number trong virtual adress được hash và lưu vào hash table
  - Virtual page number so sánh với trường (1), nếu khớp, page frame tương ứng sẽ là trường (2), nếu không khớp sẽ tìm ở phần còn lại

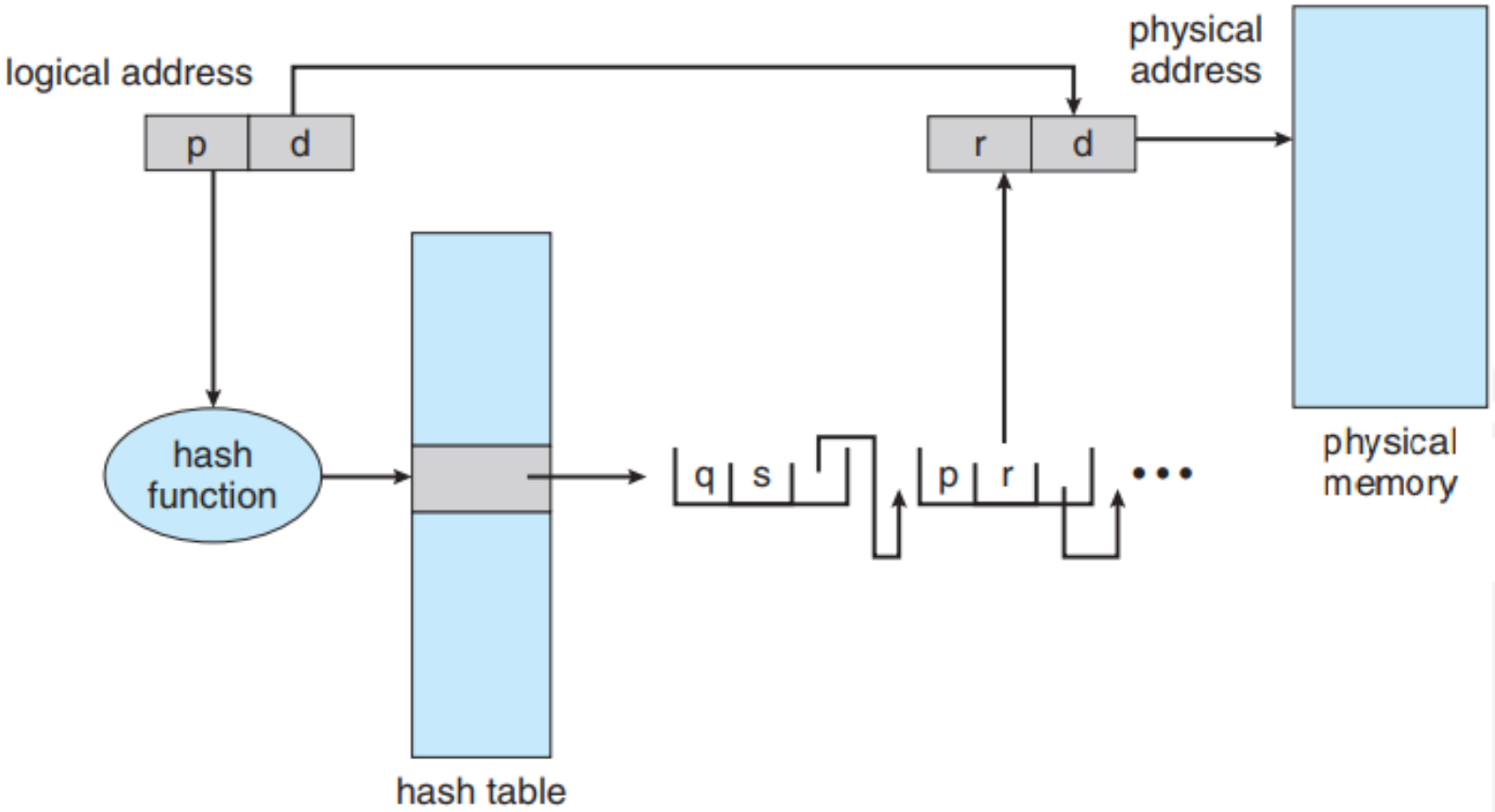


Figure 9.17 Hashed page table.

Inverted Page Tables

- Inverted page table, mỗi entry dùng cho 1 page (hoặc frame) của bộ nhớ, kèm theo thông tin về tiến trình sở hữu page → cả hệ thống chỉ dùng 1 page table
- Mỗi virtual address trong hệ thống chứa: <process-id, page-number, offset>.
- Mỗi entry của inverted page table là 1 cặp <process-id, page-number>

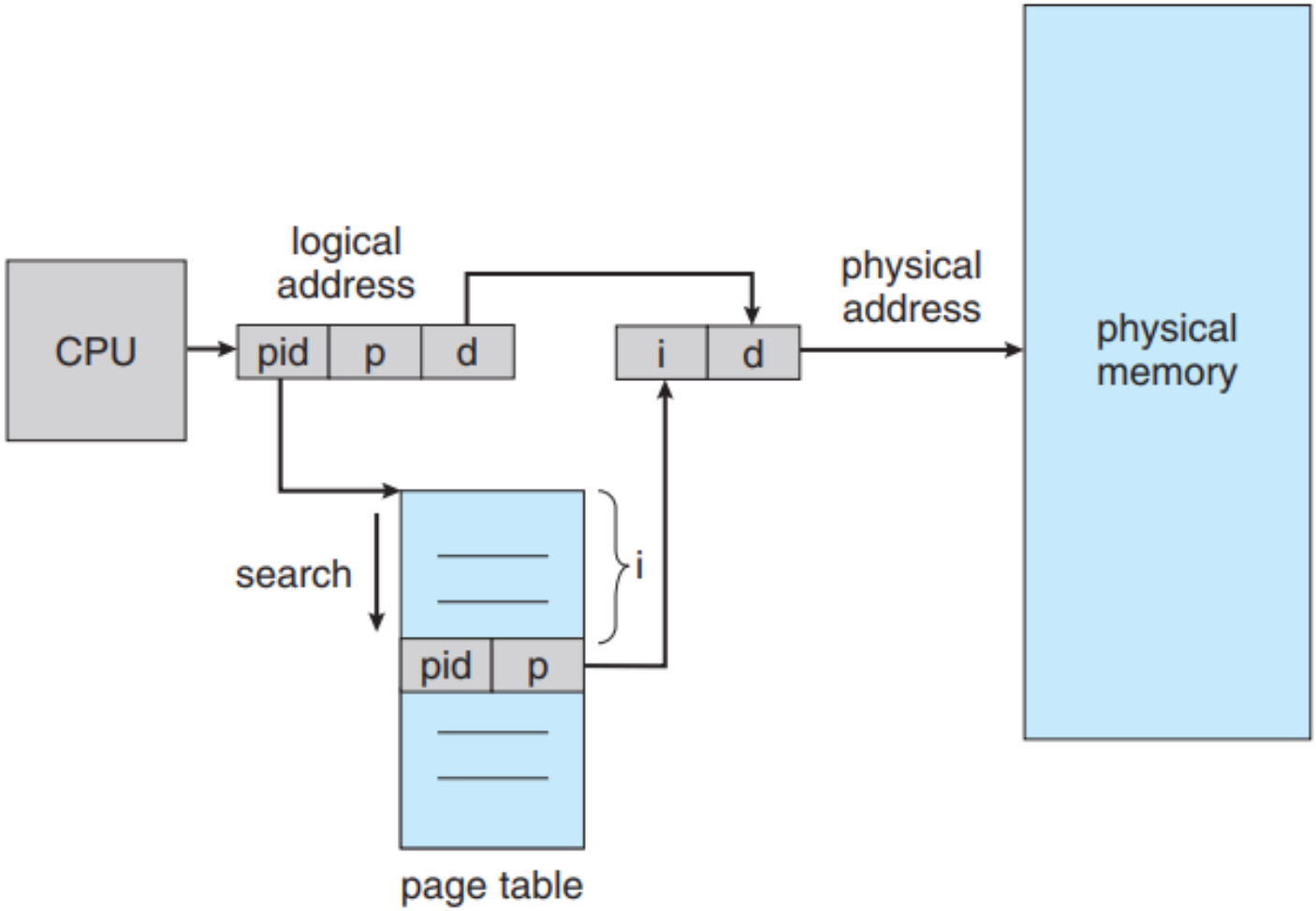


Figure 9.18 Inverted page table.

Inverted Page Tables

- Vấn đề:
  - Tuy phương pháp này giảm kích thước bộ nhớ cần sử dụng cho mỗi page table, nó lại tăng thời gian cần thiết để tìm kiếm 1 page trong table (do page table sắp xếp theo physical address mà tìm kiếm lại tìm trên virtual address) <process-id, page-number, offset>.
    - Để cải thiện, sử dụng hash table → khi tìm kiếm phải cần ít nhất 2 lần đọc bộ nhớ (1 cho hash table, 1 cho page table)
  - Liên quan đến shared memory, khi mỗi tiến trình có page table riêng, chúng có thể cùng map tới 1 physical address nhưng inverted page table thì không thể bởi vì chỉ có 1 virtual page entry cho mỗi physical page

Bộ nhớ chính

- Background
- Cấp phát bộ nhớ liền kề
- Paging
- Structure of the page table
- **Swapping**
- Example
- Summary

Swapping

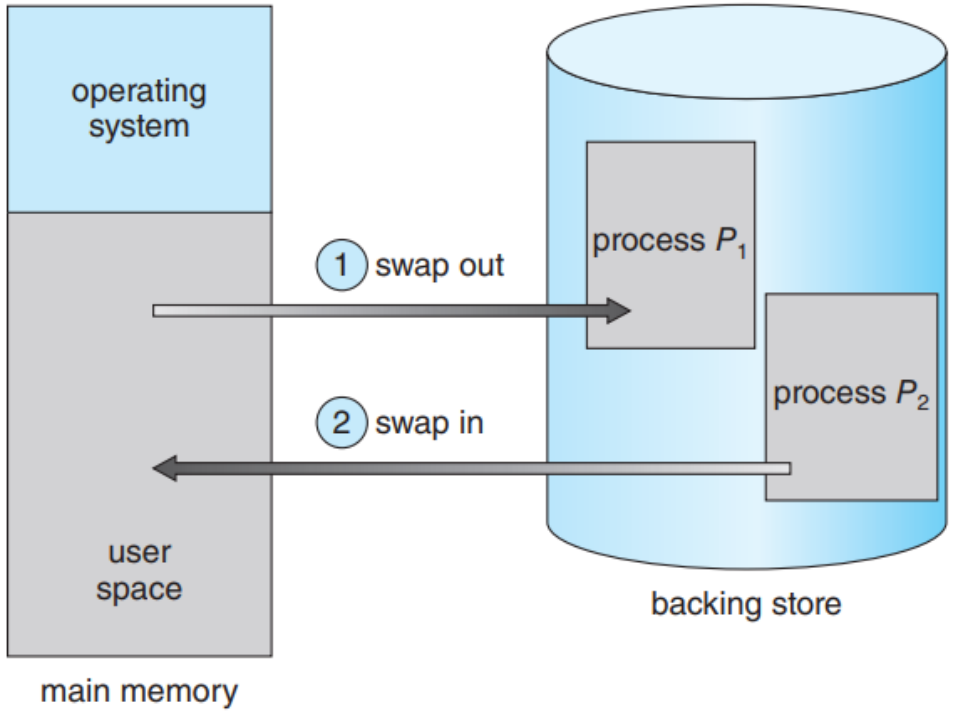


Figure 9.19 Standard swapping of two processes using a disk as a backing store.

Swapping

- Standard swapping
- Swapping with paging
- Swapping on mobile systems

Standard Swapping

- Di chuyển cả tiến trình ra ngoài bộ nhớ và quay lại bộ nhớ
- Ưu điểm:
  - Cho phép đăng ký vượt mức kích thước bộ nhớ vật lý thực tế. Các tiến trình nhàn rỗi hoặc hầu như không hoạt động được dùng để di chuyển ra ngoài nhường chỗ cho các tiến trình có nhu cầu thực thi
- Nhược điểm
  - Tốn thời gian

Swapping with paging

- Thay vì di chuyển toàn bộ tiến trình, chỉ di chuyển các page
- Ưu điểm
  - Cho phép đăng ký vượt mức kích thước bộ nhớ vật lý thực tế.
  - Ít tốn thời gian hơn so với standard swapping

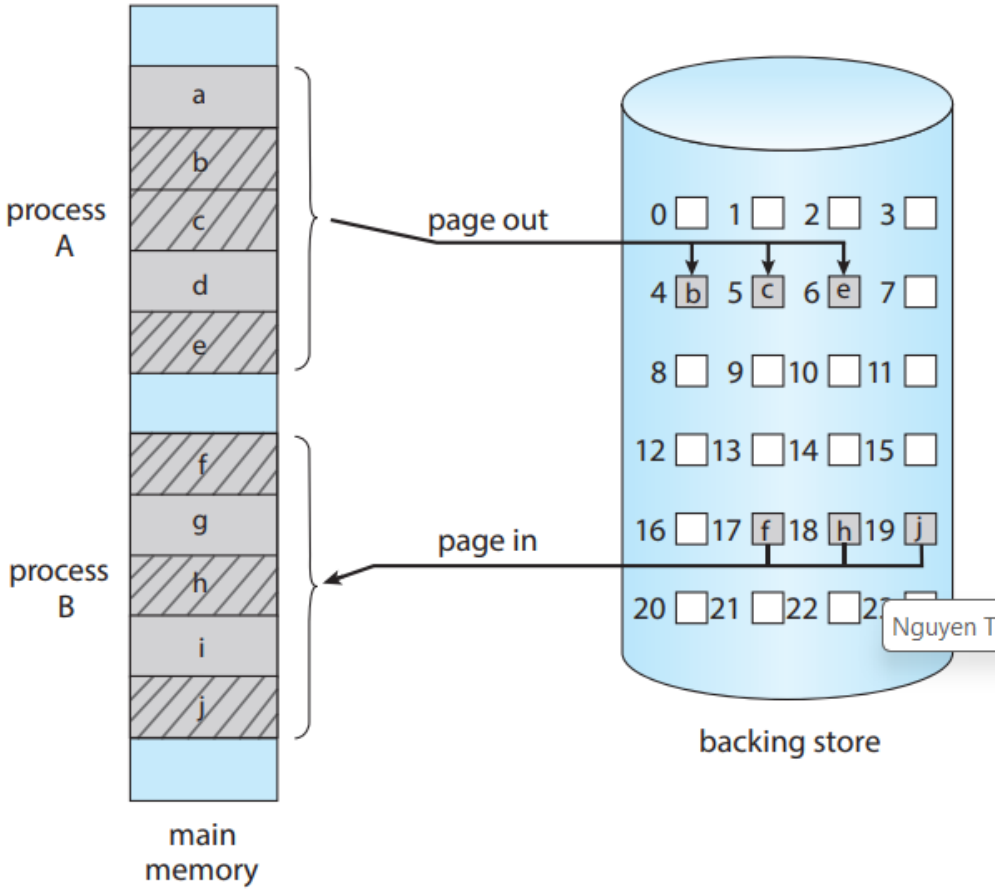


Figure 9.20 Swapping with paging.

Swapping on mobile systems

- Hầu hết OS cho PCs và servers hỗ trợ swapping pages nhưng các hệ thống mobile thì không hỗ trợ
  - Thiết bị mobile sử dụng flash memory thay vì hard disk có kích thước lớn để lưu trữ dữ liệu (nonvolatile storage)
  - Số lượng ghi hạn chế mà flash memory có thể chịu đựng trước khi trở nên không đáng tin cậy và thông lượng (throughput) kém giữa main memory và flash memory

Swapping on mobile systems

- Apple iOS yêu cầu các ứng dụng tự nguyện từ bỏ bộ nhớ được phân bộ khi bộ nhớ trống giảm xuống dưới 1 ngưỡng nhất định.
  - Read-only data (vd như code) sẽ bị xóa khỏi bộ nhớ chính và sau đó được tải lại nếu cần
  - Dữ liệu đã được sửa đổi (vd như stack) sẽ không bao giờ bị xóa
  - Bất kỳ ứng dụng nào không giải phóng đủ bộ nhớ đều có thể bị hệ điều hành chấm dứt hoạt động
- Android
  - Chấm dứt một tiến trình nếu không đủ bộ nhớ trống nhưng trước đó, Android ghi trạng thái ứng dụng vào flash memory để ứng dụng có thể khởi động nhanh chóng

Bộ nhớ chính

- Background
- Cấp phát bộ nhớ liền kề
- Paging
- Structure of the page table
- Swapping
- **Example: Intel 32- and 64-bit Architectures**
- Summary

Example

- IA-32 Architecture
- X86-64

IA-32 Architecture

- Quản lý bộ nhớ trong hệ thống IA-32 được chia thành 2 phần:
  - Segmentation
  - Paging
- Segmentation unit và paging unit tạo thành MMU

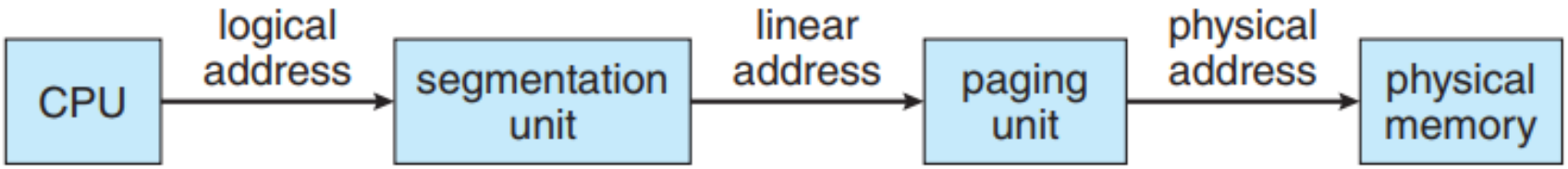


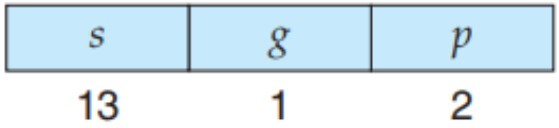
Figure 9.21 Logical to physical address translation in IA-32.

IA-32 Segmentation

- Kiến trúc IA-32 cho phép 1 phân đoạn có thể lớn 4GB, số lượng phân đoạn tối đa cho 1 tiến trình là 16K
- Không gian địa chỉ logic của 1 tiến trình được chia thành 2 phân vùng
  - vùng 1 bao gồm 8K phân đoạn riêng (private) cho tiến trình, thông tin được lưu trữ trong LDT (local descriptor table)
  - vùng 2 bao gồm 8K phân đoạn chia sẻ giữa tất cả các tiến trình, thông tin được lưu trữ trong GDT (global descriptor table)
- Mỗi entry trong LDT và GDT chưa 8-byte segment descriptor với thông tin chi tiết về segment cụ thể, bao gồm base location và limit

IA-32 Segmentation

- Địa chỉ logic là 1 cặp (selector, offset), trong đó offset là 1 số 32 bit – vị trí của byte trong segment và selector là 1 số 16 bit
  - s – segment number
  - g – chỉ định là GDT hay LDT?
  - p - protection



- Máy có 6 thanh ghi segment, cho phép 6 segment được xử lý cùng một lúc bởi 1 tiến trình. Cũng có 6 thanh ghi microprogram để chứa các bộ mô tả tương ứng từ LDT và GDT. → Bộ đệm này giúp Pentium tránh phải đọc bộ mô tả từ bộ nhớ cho mỗi tham chiếu bộ nhớ.

IA-32 Segmentation

- Địa chỉ tuyến tính (linear address) dài 32 bit.
- Thanh ghi segment trỏ tới entry tương ứng trong LDT hoặc DGT. Thông tin base và limit về segment được sử dụng để tạo ra linear adddress
- Đầu tiên limit được sử dụng để kiểm tra xem address có hợp lệ. Nếu không, memory fault, nếu hợp lệ, giá trị của offset được thêm vào giá trị của base, tạo thành 32bit linear address

IA-32 Segmentation

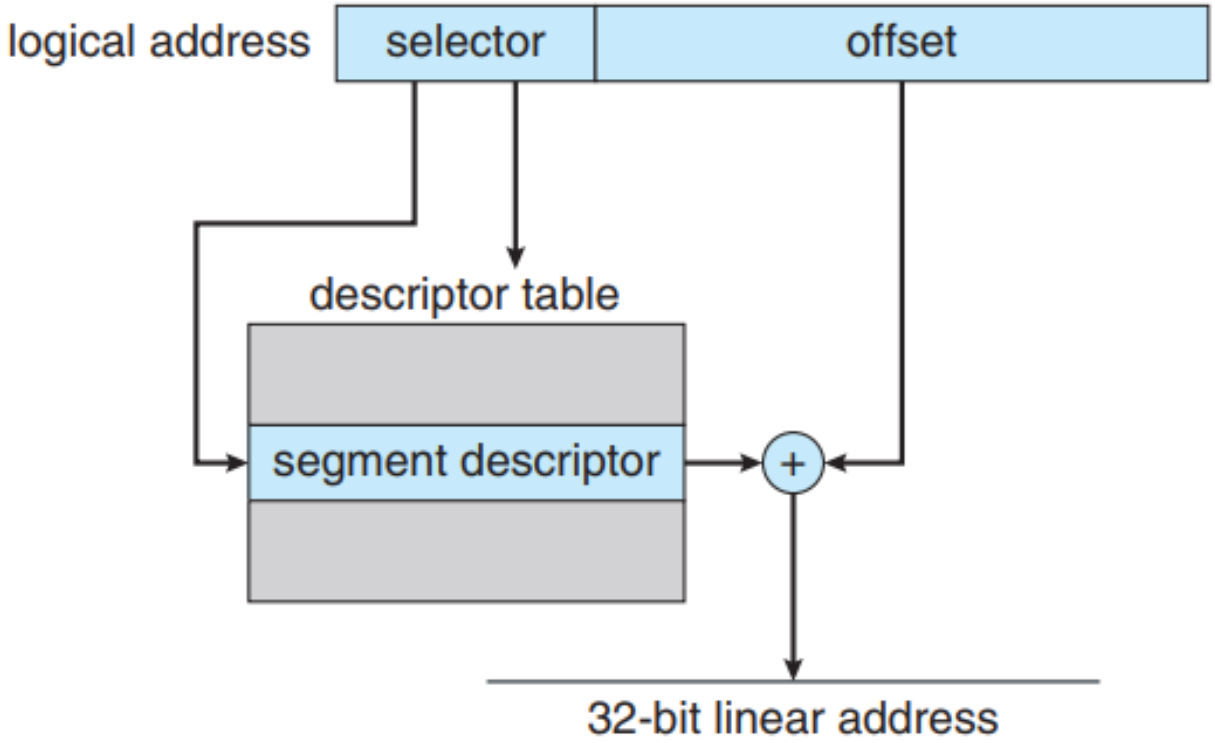
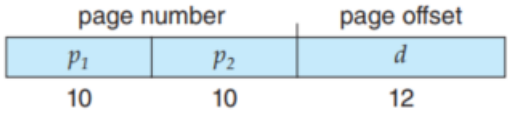


Figure 9.22 IA-32 segmentation.

IA-32 Paging

- Kiến trúc IA-32 cho phép 1 page có kích thước là 4KB hoặc 4MB. Với 4KB page, IA-32 sử dụng phương pháp two-level paging để chia 32bit linear address thành:



- Phương pháp address-translation

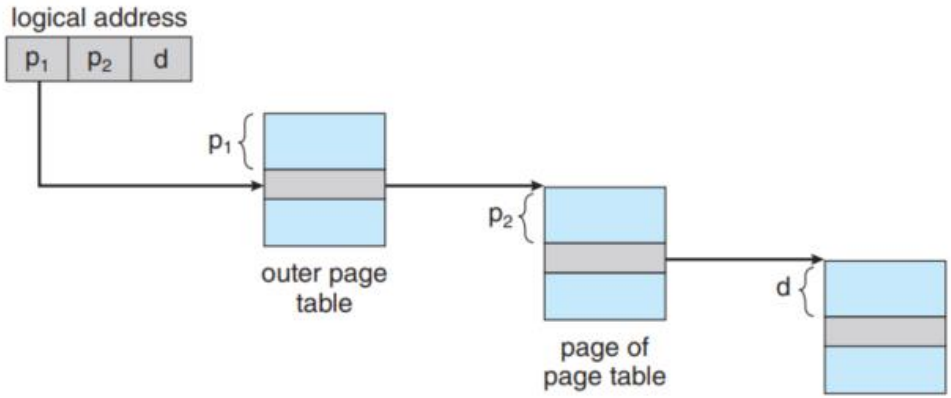


Figure 9.16 Address translation for a two-level 32-bit paging architecture.

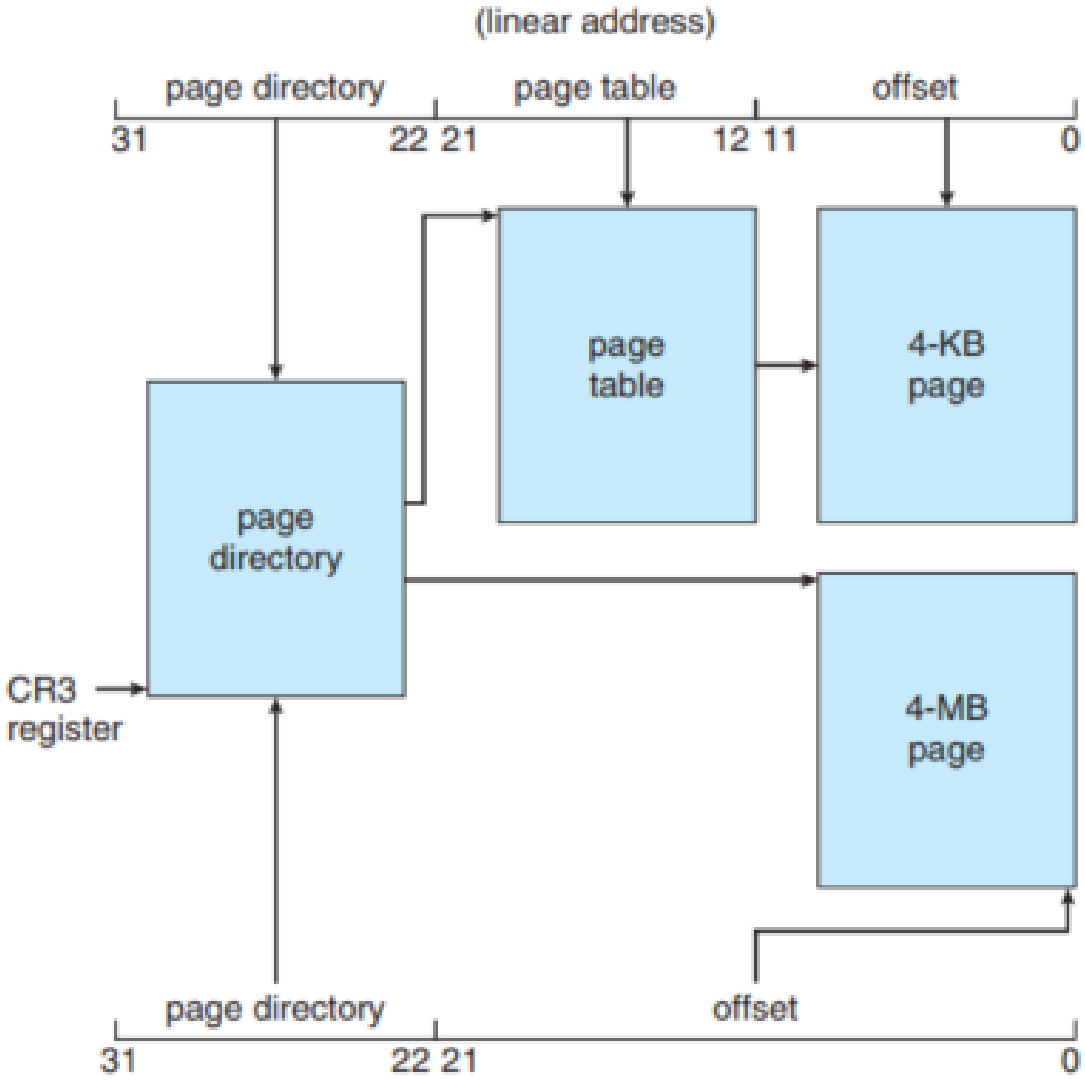


Figure 9.23 Paging in the IA-32 architecture.

IA-32 Paging

- Khi các nhà phát triển phần mềm phát hiện ra những hạn chế của bộ nhớ 4GB của kiến trúc 32bit, Intel phát triện PAE (page address extension) – tiện ích mở rộng địa chỉ trang, cho phép bộ xử lý 32bit truy cập vào không gian địa chỉ vật lý lớn hơn 4GB.
- Sự khác biệt là sử dụng phương pháp three-level thay vì two-level, trong đó 2 bit trên cùng tham chiếu đến bảng con trỏ thư mục trang.

IA-32 Paging

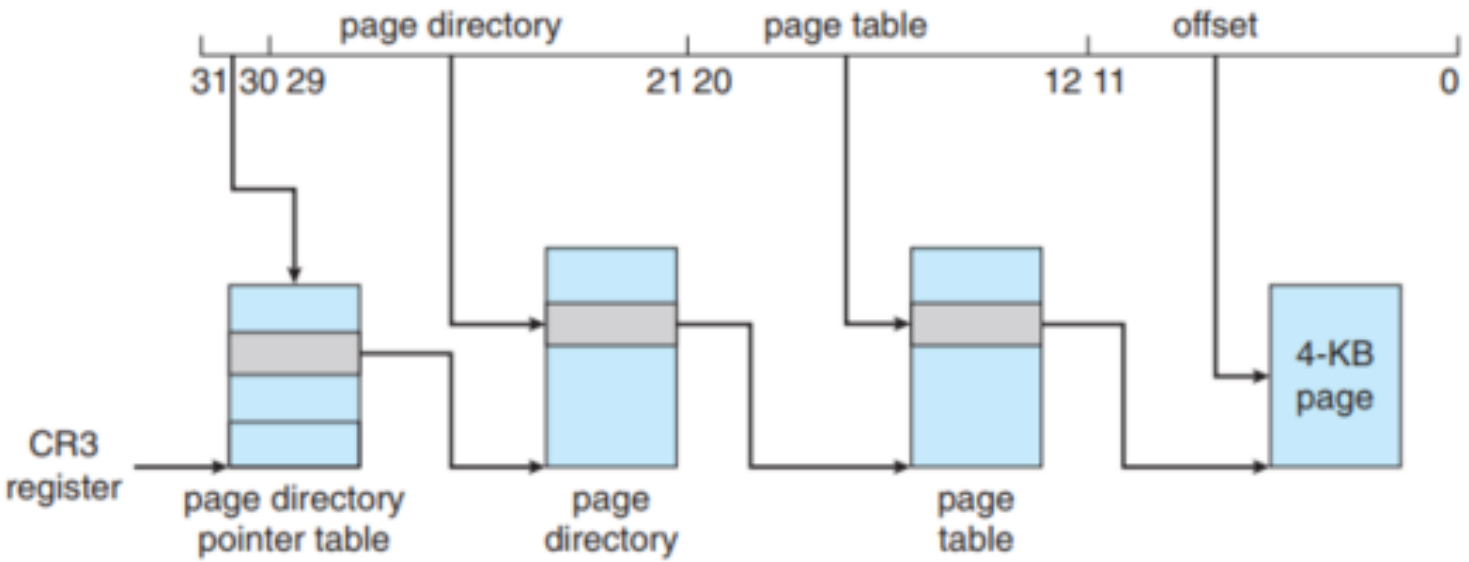


Figure 9.24 Page address extensions.

x86-64

- Hỗ trợ 64bit address space → 2^{64} byte bộ nhớ có thể định địa chỉ (lớn hơn 16 triệu triệu hay 16exabytes)
- Thực tế có ít hơn 64bit được sử dụng để biểu diễn đại chỉ trong các thiết kế hiện tại
- Kiến trúc x86-64 hiện cung cấp địa chỉ ảo 48bit với sự hỗ trợ cho kích thước trang 4KB, 2MB hoặc 1GB bằng cách sử dụng four-level paging hierarchy.

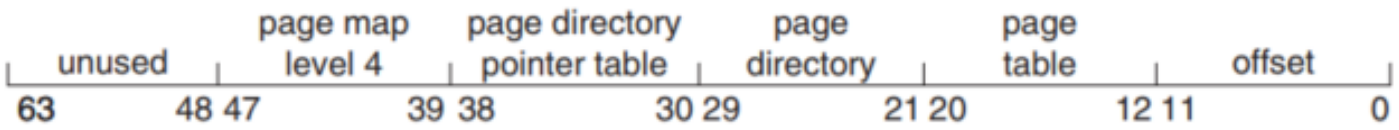


Figure 9.25 x86-64 linear address.

Bộ nhớ ảo

- Background
- Demand Paging
- Copy-on-write
- Page Replacement
- Allocation of frames
- Thrashing
- Memory Compression
- Other Considerations
- Operating-System Examples
- Summary

Cần nắm

- Định nghĩa bộ nhớ ảo và mô tả ưu điểm của nó
- Mô tả cách page được tải vào memory sử dụng cơ chế demand paging
- Triển khai các thuật toán thay thế trang FIFO, optimal, LRU

Background

- Các câu lệnh muốn thực thi được phải nằm trong physical memory
- Trong vài trường hợp, toàn bộ chương trình không cần phải nằm trong memory:
  - Chương trình thường có code xử lý những điều kiện lỗi bất thường
  - Array, list và table thường được cấp phát bộ nhớ nhiều hơn chúng thực sự cần.
  - Một số tùy chọn và tính năng nhất định của chương trình hiếm được sử dụng
- Ngay cả khi cả chương trình cần được nạp vào trong memory thì chúng cũng không phải cùng một thời điểm

Background

- Lợi ích khi nạp một phần chương trình:
  - Chương trình không còn bị hạn chế bởi dung lượng bộ nhớ vật lý hiện có. Người dùng có thể viết chương trình cho không gian địa chỉ lớn, đơn giản hóa tác vụ lập trình
  - Chương trình chiếm ít bộ nhớ nên có thể chạy nhiều chương trình hơn cùng lúc, với mức sử dụng và thông lượng CPU tăng tương ứng nhưng không tăng thời gian phản hồi hoặc thời gian quay vòng
  - Cần ít I/O hơn để tải hoặc hoán đổi các phần chương trình vào bộ nhớ, do đó chương trình sẽ chạy nhanh hơn

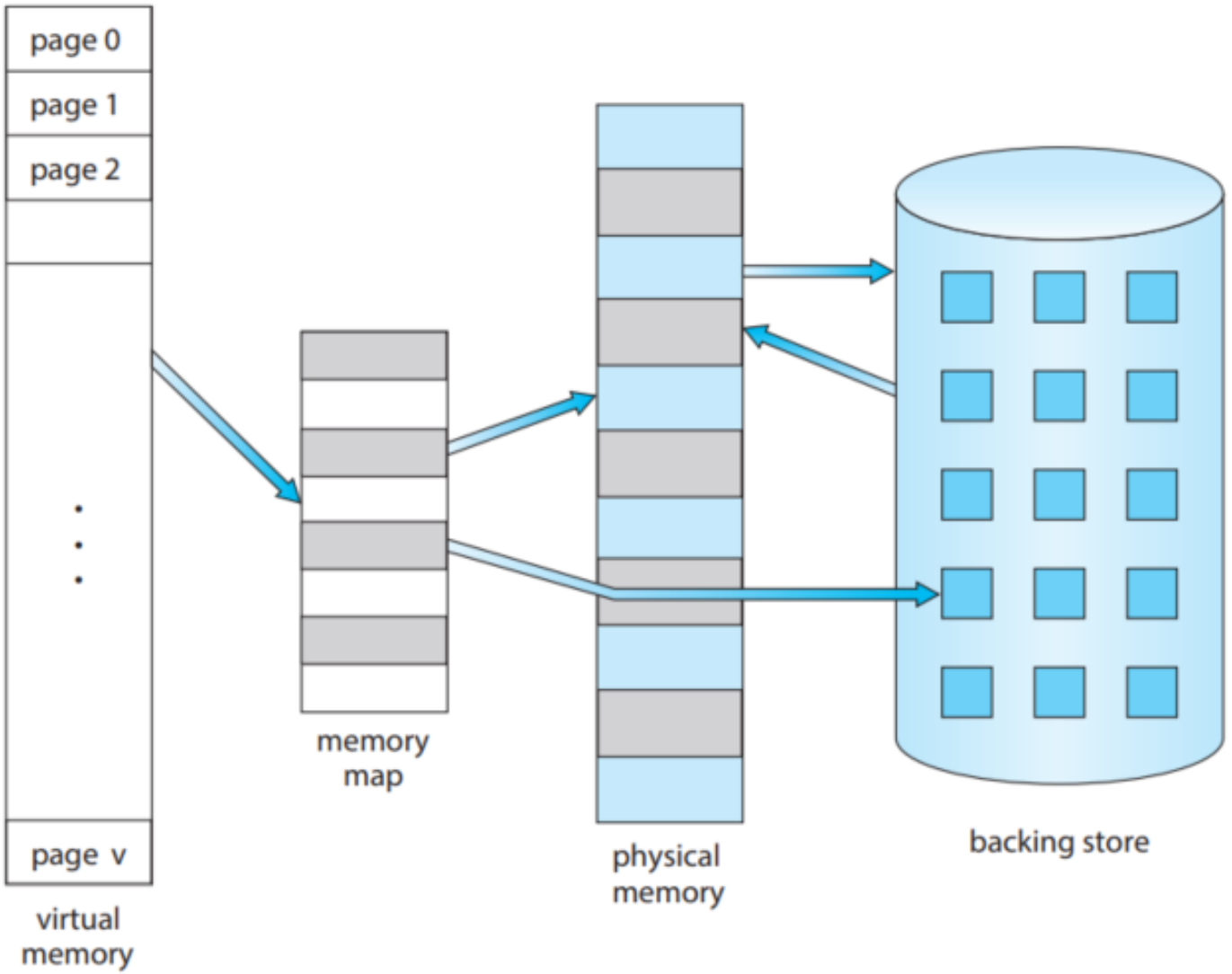


Figure 10.1 Diagram showing virtual memory that is larger than physical memory.

Background

- Không gian địa chỉ ảo bao gồm phần có thể mở rộng (hole) được gọi là không gian địa chỉ thưa thớt (sparse address space)

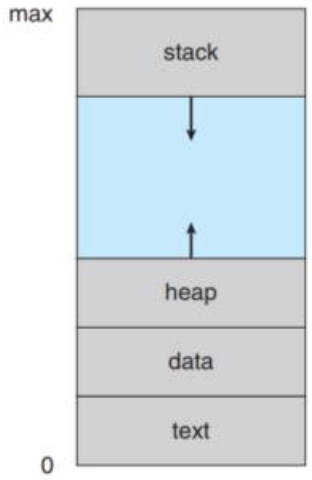


Figure 10.2 Virtual address space of a process in memory.

Bộ nhớ ảo

- Background
- **Demand Paging**
- Copy-on-write
- Page Replacement
- Allocation of frames
- Thrashing
- Memory Compression
- Other Considerations
- Operating-System Examples
- Summary

Demand Paging

- Chỉ load page của tiến trình khi cần – demand paging
- Valid-invalid bit
  - v: page hợp lệ và nằm trong memory
  - i: page không hợp lệ hoặc hợp lệ nhưng không nằm trong memory

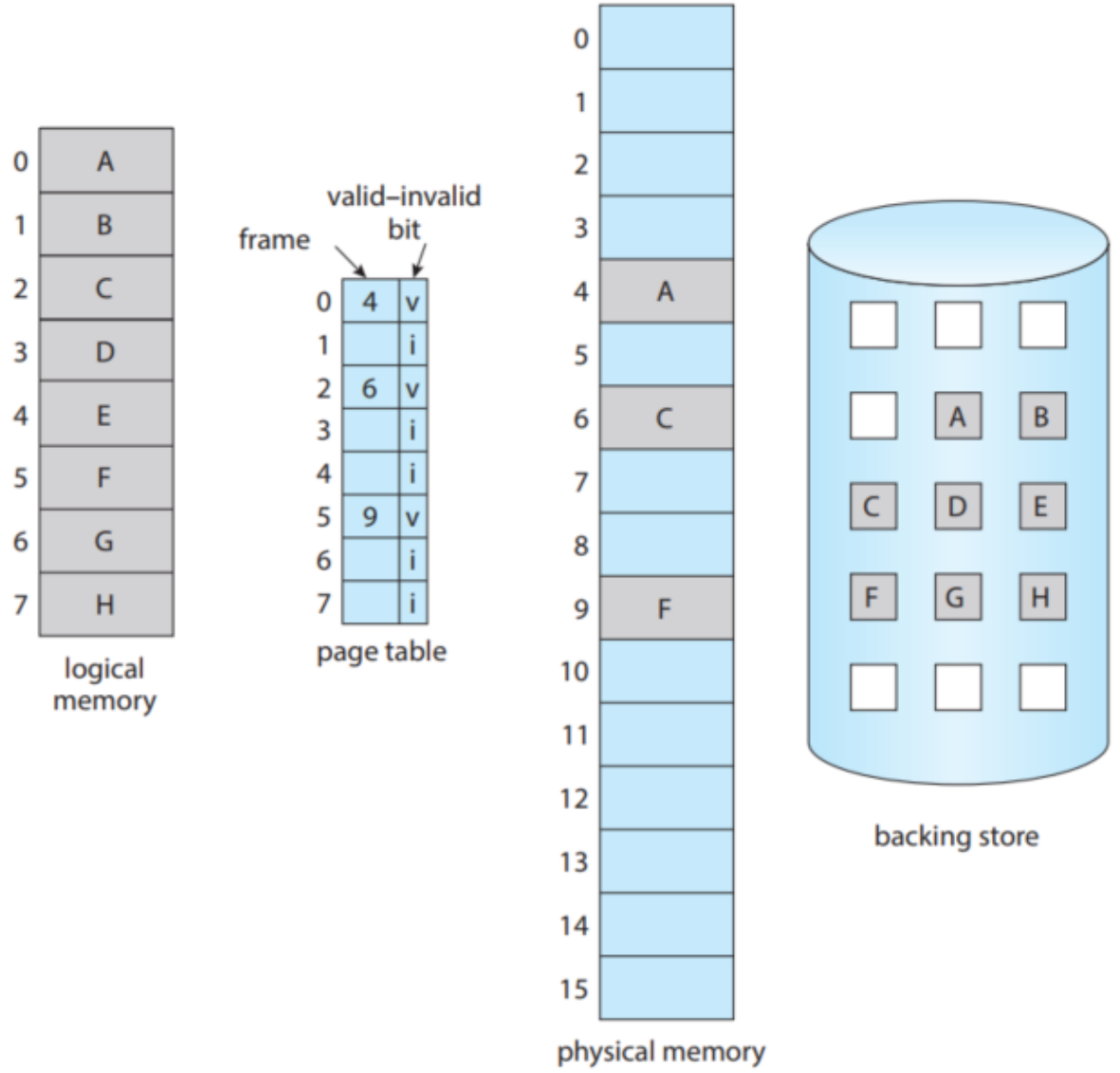


Figure 10.4 Page table when some pages are not in main memory.

Demand Paging

- Tiến trình cố truy cập vào 1 page không nằm trong memory → **page fault**

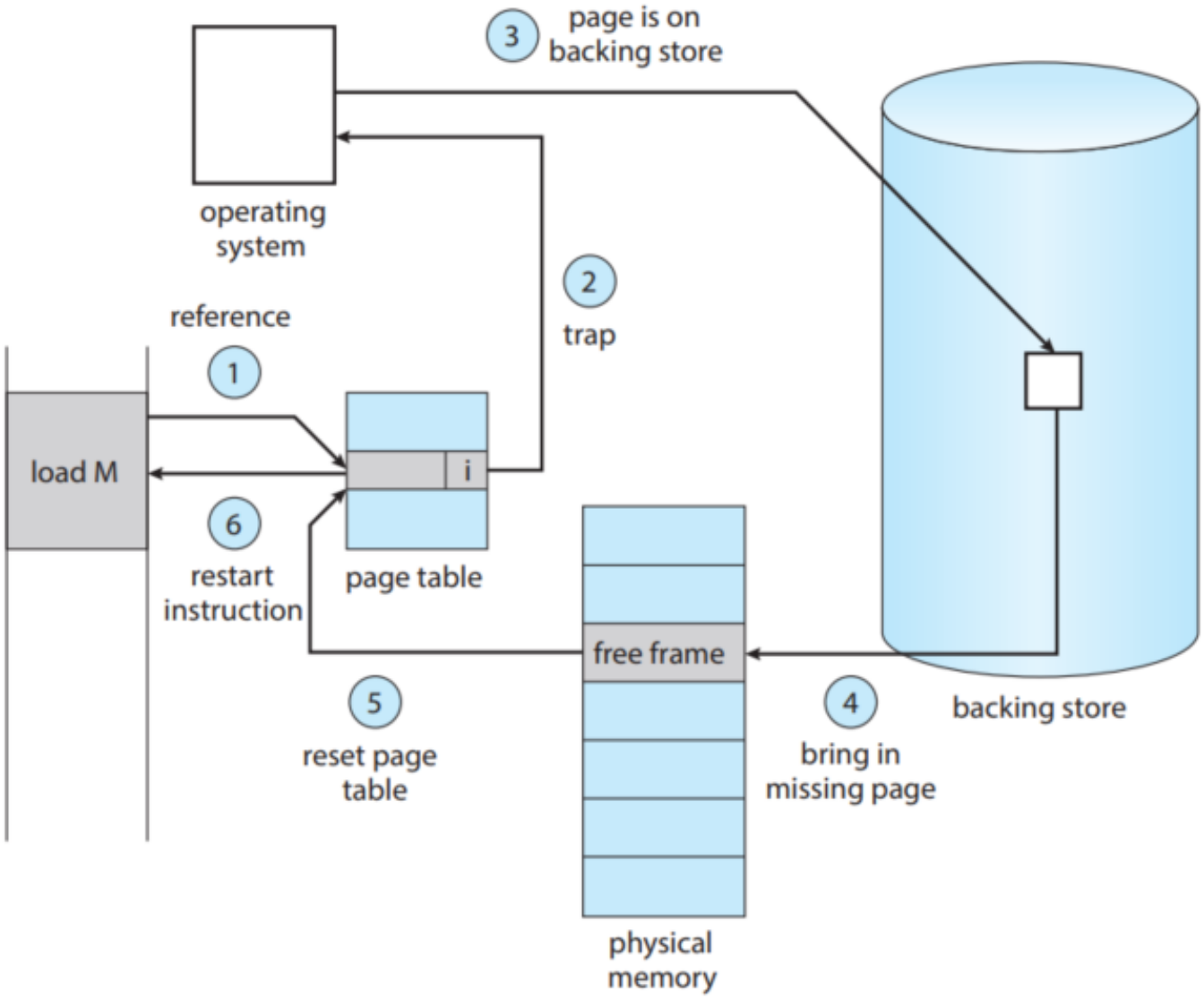


Figure 10.5 Steps in handling a page fault.

Demand paging

- Xử lý page fault
  - Kiểm tra internal table (thường trong PCB) xem thử tham chiếu valid hay invalid
  - Nếu tham chiếu invalid thì kết thúc tiến trình, nếu valid nhưng chưa được đưa vào memory thì đưa vào
  - Tìm free frame
  - Nạp page tương ứng vào frame tìm được
  - Sau khi hoàn thành thì chỉnh sửa internal table và page table để biết là page đã nằm trong memory
  - Bắt đầu thực thi lại câu lệnh bị ngắt bởi trap. Tiến trình bây giờ có thể truy cập vào page

Demand paging

- Phần cứng hỗ trợ demand paging giống phần cứng hỗ trợ cho paging và swapping:
  - Page table: Bảng chứa bit valid-invalid hoặc 1 giá trị bit đặc biệt cho cơ chế bảo vệ (protection)
  - Secondary memory: Phần bộ nhớ lưu trữ các page không hiện diện trong main memory. Secondary memory thường là high-speed disk hoặc thiết bị NVM – được biết đến như là thiết bị swap.

Demand paging

- 1 yêu cầu quan trọng của demand paging là khởi động lại câu lệnh chính xác lúc gặp page fault.
- VD các bước thực thi:
  - Fetch và decode câu lệnh ADD
  - Fetch A
  - Fetch B
  - ADD A và B
  - Lưu tổng vào C
- Nếu bị page fault tại bước lưu trữ vào C (vì C nằm trong page chưa được nạp vào memory), như vậy phải swap page vào memory, sửa page table, khởi động lại câu lệnh từ đầu → lặp lại các bước đã thực hiện

Free-frame list

- Khi page fault diễn ra, OS phải mang page đó từ secondary storage vào main memory, OS có danh sách frame free có thể chưa page

Figure 10.6 List of free frames.

Hiệu suất Demand Paging

- Demand paging ảnh hưởng đến hiệu suất của hệ thống máy tính.
- Giả sử memory-access time ma=10 nanoseconds
- Nếu ko có page fault, thời gian truy cập bằng với memory-access time
- Nếu có page fault, phải đợi page từ secondary memory được đưa vào main memory rồi mới truy cập
- Gọi p là xác suất xảy ra page fault (0<=p<=1), mong muốn p tiềm cận 0 – có nghĩa ra rất ít page fault

$$\text{effective access time} = (1 - p) \times ma + p \times \text{page fault time}.$$

Hiệu suất Demand Paging

- Khi page fault xảy ra:
  1. Trap OS
  2. Lưu các thanh ghi và trạng thái tiến trình
  3. Xác định rằng ngắt là page fault
  4. Kiểm tra xem page tham chiếu có hợp lệ hay không, và xác định vị trí của page trong secondary storage
  5. Đưa ra lệnh đọc từ storage tới free frame
     - Đợi ở hàng đợi cho đến khi yêu cầu đọc được thực hiện
     - Đợi thiết bị seek và / hoặc thời gian chờ
     - Bắt đầu chuyển page tới free frame

Hiệu suất Demand Paging

- Khi page fault xảy ra:
  6. Trong khi đợi, cấp phát CPU core cho các tiến trình khác
  7. Nhận 1 ngắt từ storage I/O subsystem báo hoàn thành I/O
  8. Lưu thanh ghi và trạng thái tiến trình của tiến trình đang thực thi ở bước 6
  9. Xác định rằng ngắt đến từ secondary storage
  10. Sửa lại page table và các bản liên quan khác
  11. Đợi CPU core cấp phát cho tiến trình lần nữa
  12. Khôi phục thanh ghi, trạng thái tiến trình, page table mới và tiếp tục câu lệnh bị gián đoạn/ ngắt

Hiệu suất Demand Paging

- Trong nhiều trường hợp chỉ có 3 bước chính
  1. Phục vụ ngắt page fault
  2. Đọc page
  3. Khởi động lại tiến trình

Bộ nhớ ảo

- Background
- Demand Paging
- **Copy-on-write**
- Page Replacement
- Allocation of frames
- Thrashing
- Memory Compression
- Other Considerations
- Operating-System Examples
- Summary

Copy-on-write

- Khi sử dụng system call fork() tạo 1 tiến trình con thì không gian bộ nhớ của tiến trình cha sẽ được copy sang cho tiến trình con. Tuy nhiên nếu nhiều tiến trình con được thực thi thì việc copy không gian bộ nhớ của tiến trình cha trở nên không cần thiết.
- Sử dụng công nghệ copy-on-write – cho phép tiến trình cha và tiến trình con cùng chia sẻ chung các page. Những page chia sẻ này được đánh dấu là copy-on-write page, khi tiến trình sửa (write) page thì một bản copy của page được tạo ra.

Copy-on-write

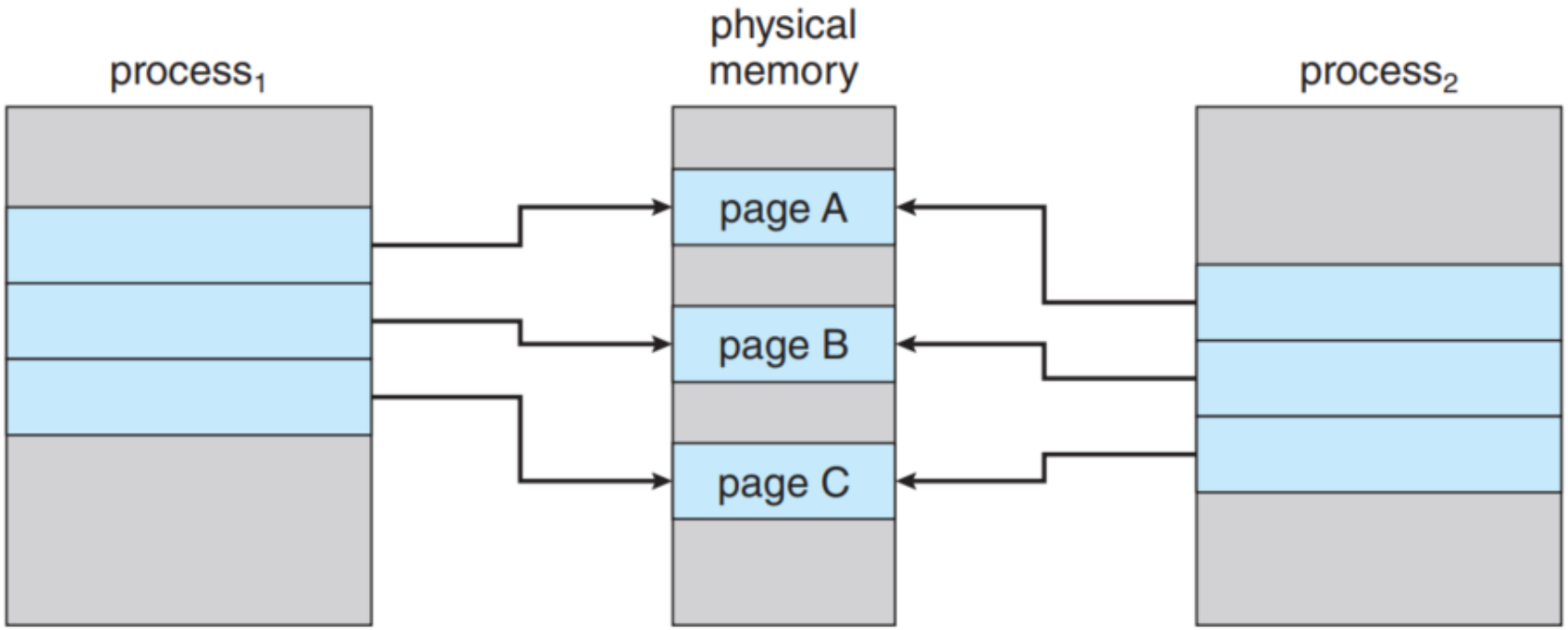


Figure 10.7 Before process 1 modifies page C.

Copy-on-write

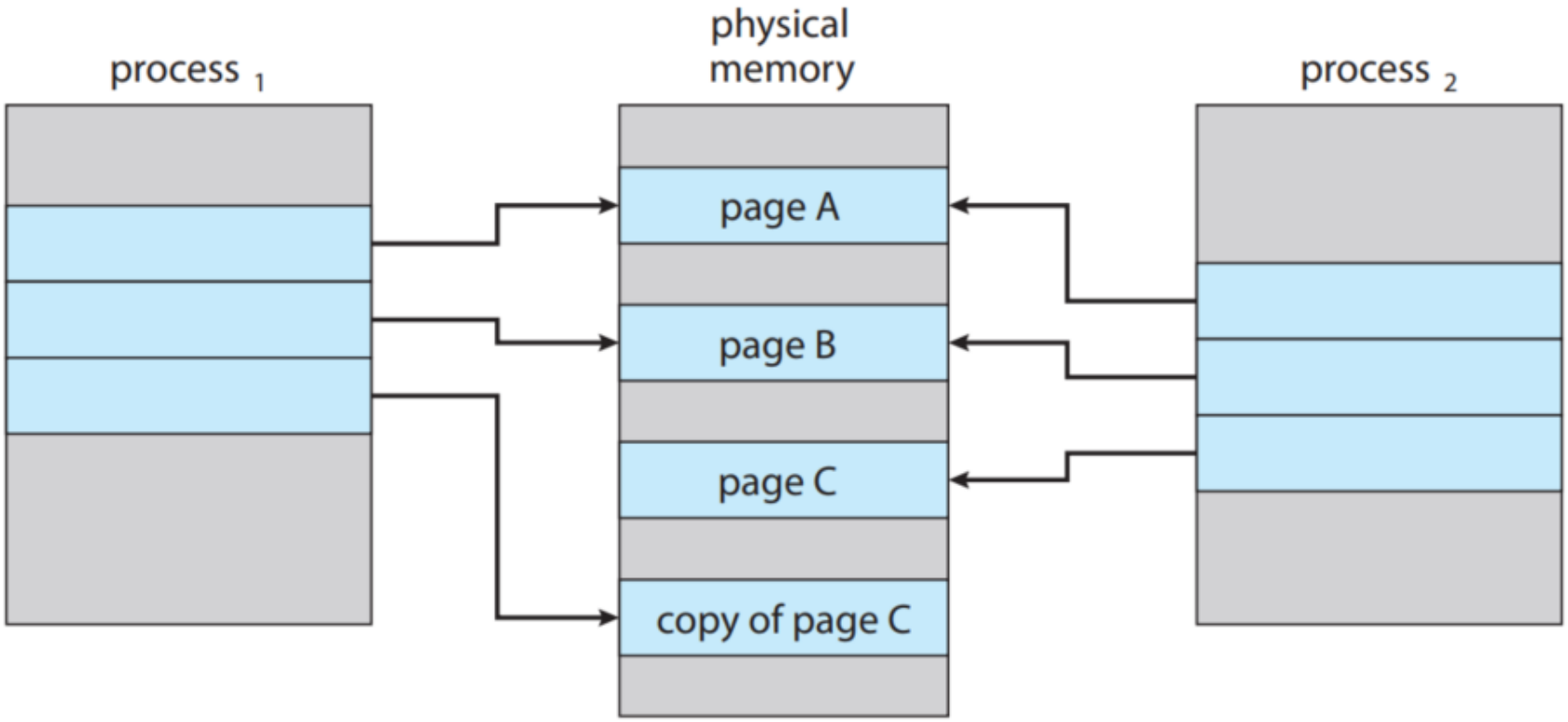


Figure 10.8 After process 1 modifies page C.

Bộ nhớ ảo

- Background
- Demand Paging
- Copy-on-write
- **Page Replacement**
- Allocation of frames
- Thrashing
- Memory Compression
- Other Considerations
- Operating-System Examples
- Summary

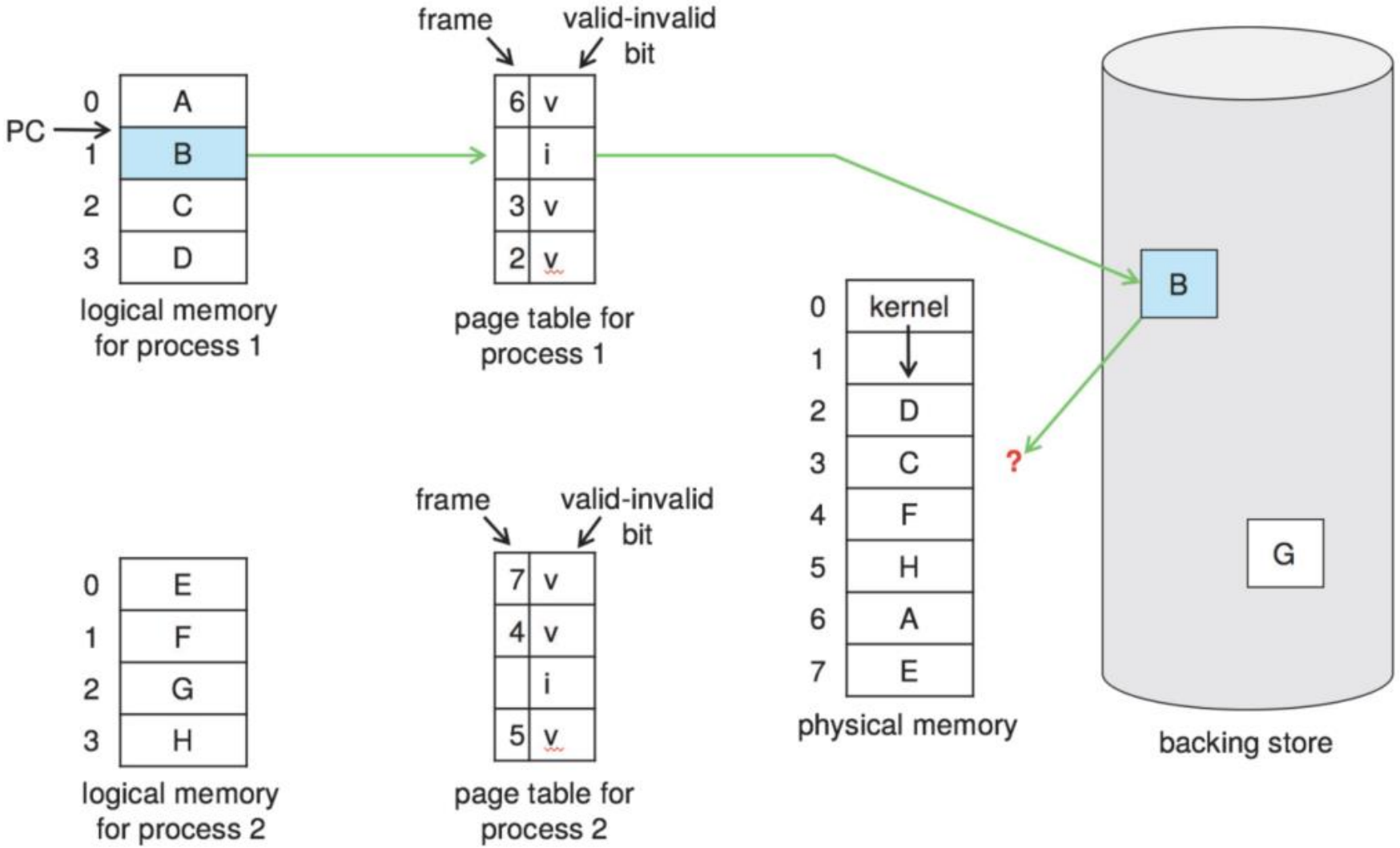


Figure 10.9 Need for page replacement.

Page replacement

- Basic page replacement
- FIFO page replacement
- Optimal Page Replacement
- LRU Page Replacement
- LRU-Approximation Page Replacement
- Couting-Based Page Replacement
- Page-Buffering Algorithms
- Applications and Page Replacement

Thay thế trang cơ bản

- Nếu không có frame nào free thì tìm frame hiện tại không dùng và xóa nội dung nó đi và sửa lại page table.
- Page fault service khi có thay thế trang:
  1. Tìm vị trí của page cần ở secondary storage
  2. Tìm frame free
     - Nếu có thì sử dụng nó
     - Nếu không thì sử dụng thuật toán thay thế trang để chọn nạn nhân
     - Ghi frame nạn nhân vào secondary và thay đổi page table, frame table tương ứng
  3. Đọc nội dung page và frame free mới, đổi page table và frame table
  4. Tiếp tục tiến trình tại thời điểm page fault diễn ra

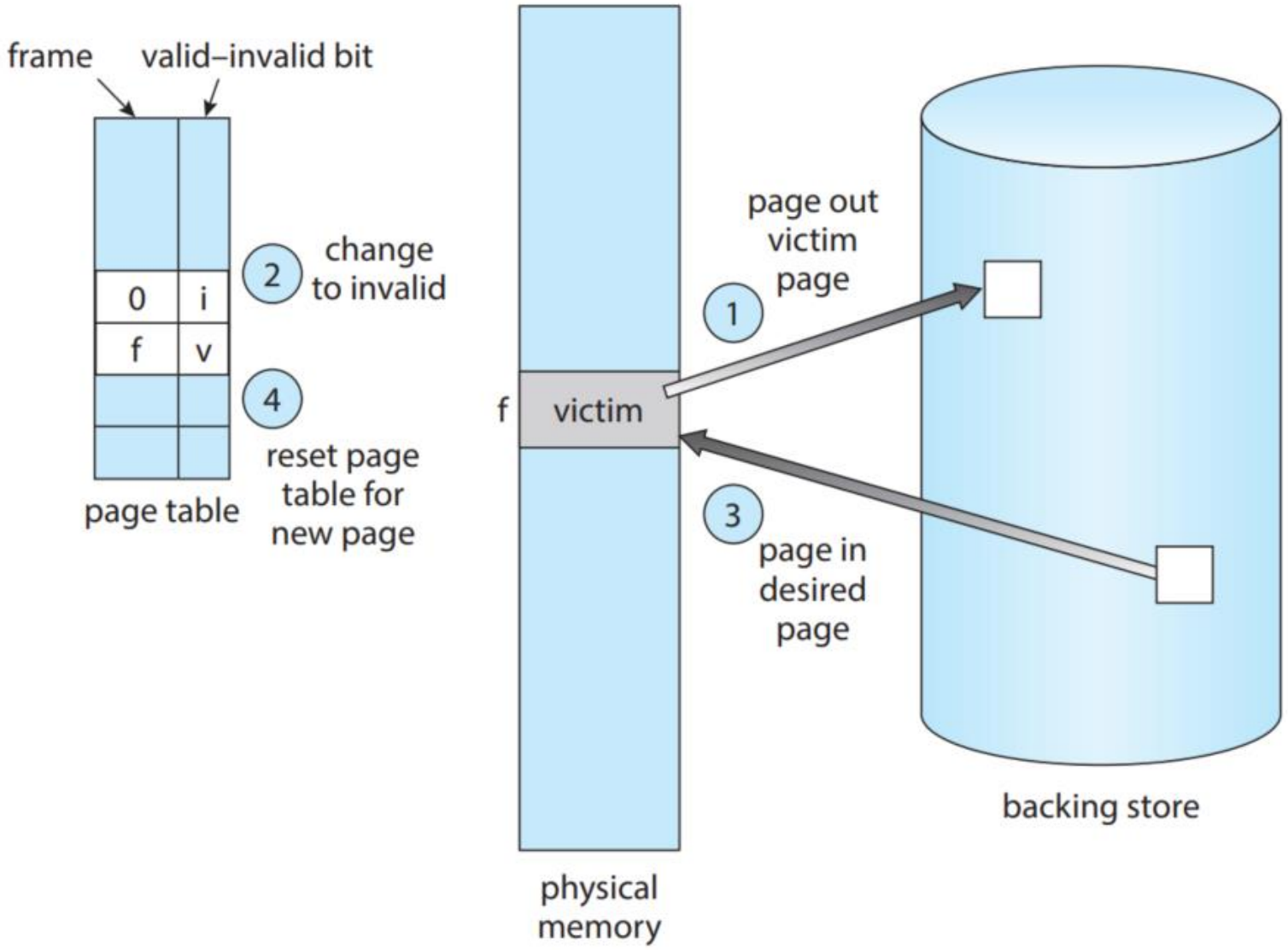


Figure 10.10 Page replacement.

Thay thế trang cơ bản

- Nếu không có frame free, 2 page sẽ được truyền (1 page out, 1 page in) → tăng thời gian truy cập bộ nhớ
- Có thể giảm bằng cách sử dụng modify bit (dirty bit) đánh dấu page bị sửa đổi
- Khi chọn page để thay thế, xét bit modify, nếu có thay đổi sẽ ghi vào secondary storage, nếu không thì không cần.

Thay thế trang cơ bản

- 2 vấn đề chính khi triển khai demand paging
  - Frame-allocation algorithm
    - Có nhiều tiến trình thì phải quyết định có bao nhiêu frame cấp phát cho mỗi tiến trình
  - Page-replacement algorithm
    - Chọn frame nào để thay thế

Thay thế trang cơ bản

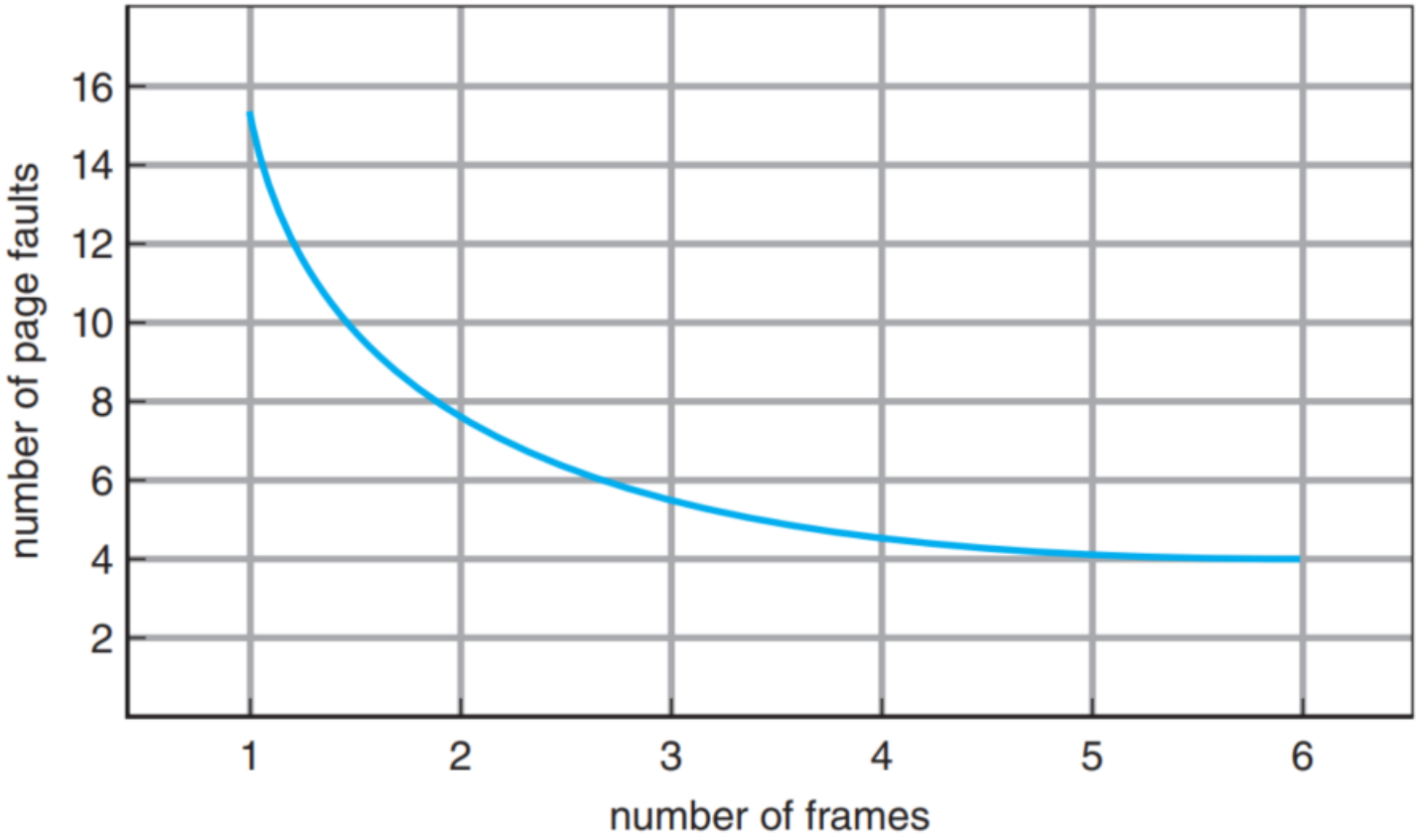


Figure 10.11 Graph of page faults versus number of frames.

FIFO

- Khi cần thay thế trang thì page cũ nhất sẽ được chọn
- Không cần lưu thời gian của từng page được đưa vào physical memory, chỉ cần 1 hàng đợi, page đầu hàng đợi sẽ bị thay thế, khi 1 page mới được thêm vào thì nó sẽ được thêm vào đuôi của hàng đợi
- Ưu điểm:
  - Thuật toán đơn giản, dễ hiểu, dễ lập trình
  - Hiệu suất không phải lúc nào cũng tốt

FIFO

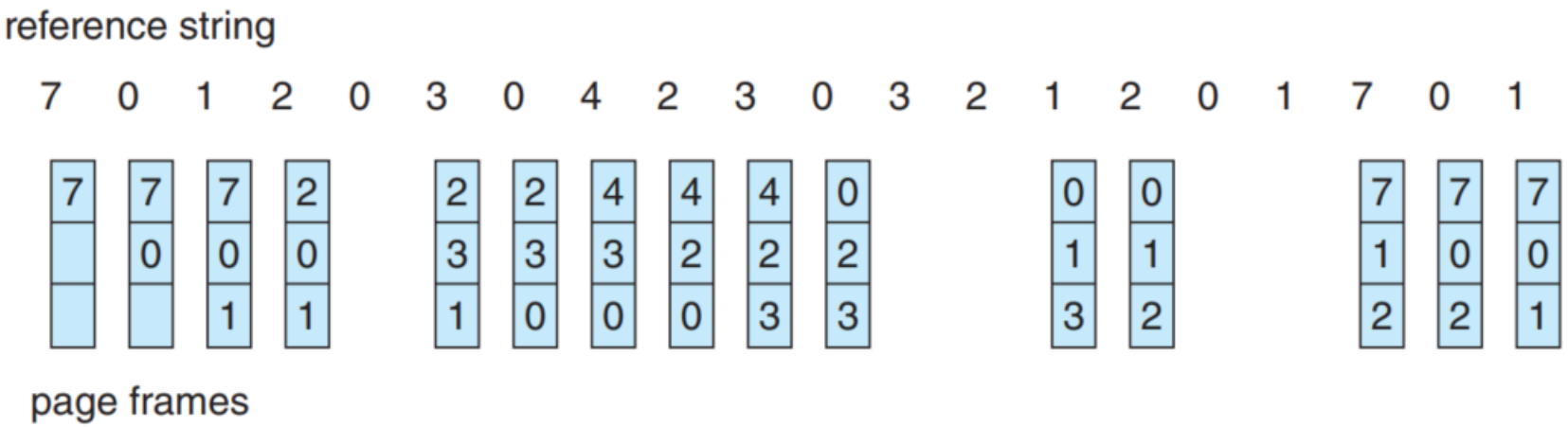


Figure 10.12 FIFO page-replacement algorithm.

FIFO

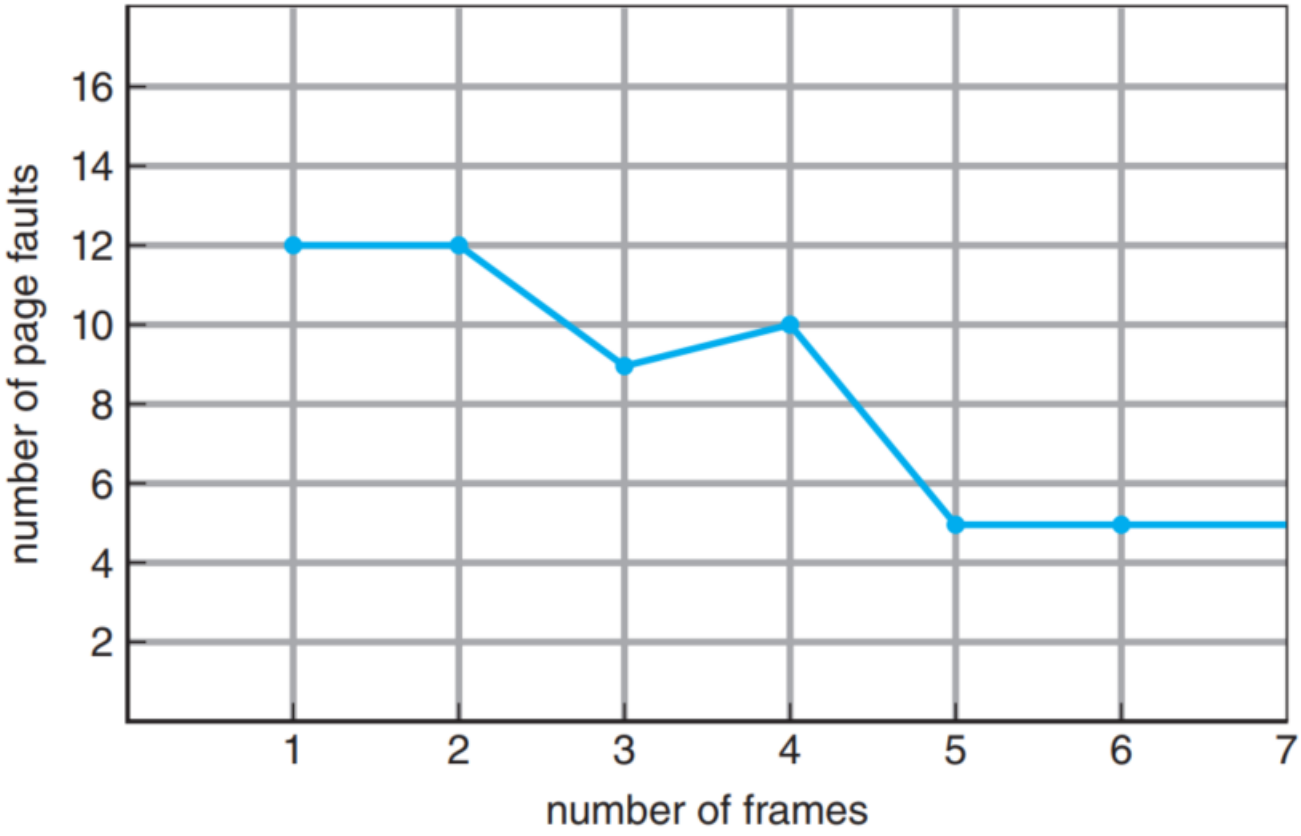


Figure 10.13 Page-fault curve for FIFO replacement on a reference string.

Optimal PR (Tối ưu)

- Thay thế trang sẽ không được sử dụng trong thời gian dài nhất
- Ưu điểm:
  - Thuật toán tối ưu, không bao giờ bị Belady's anomaly
  - Đảm bảo tỉ lệ page-fault thấp nhất cho 1 số lượng frame nhất định

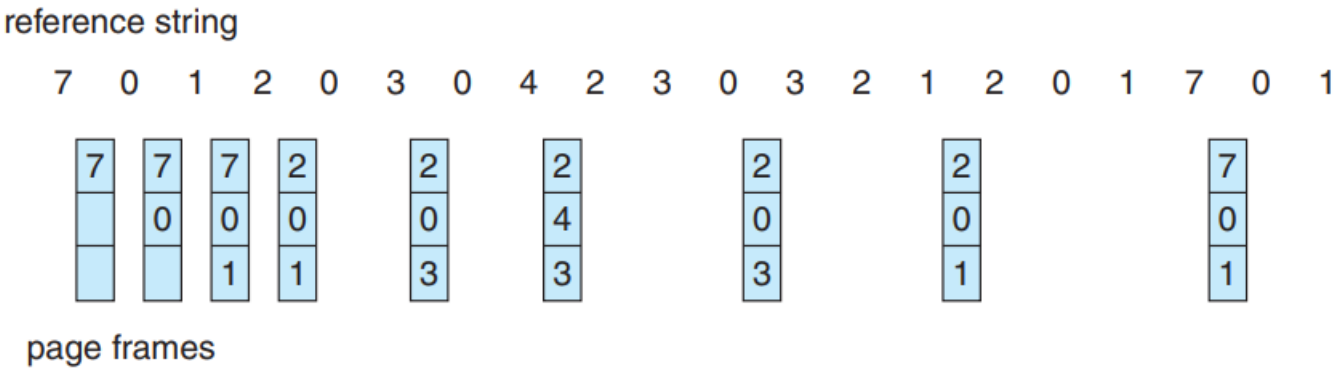


Figure 10.14 Optimal page-replacement algorithm.

Optimal PR (Tối ưu)

- Nhược điểm
  - Khó triển khai vì yêu cầu phải biết dãy page reference được yêu cầu trong tương lai
  - Nên chủ yếu được sử dụng để so sánh trong học thuật

LRU PR

- Thuật toán gần giống Optimal PR
- Thay thế page đã không được sử dụng trong thời gian dài nhất

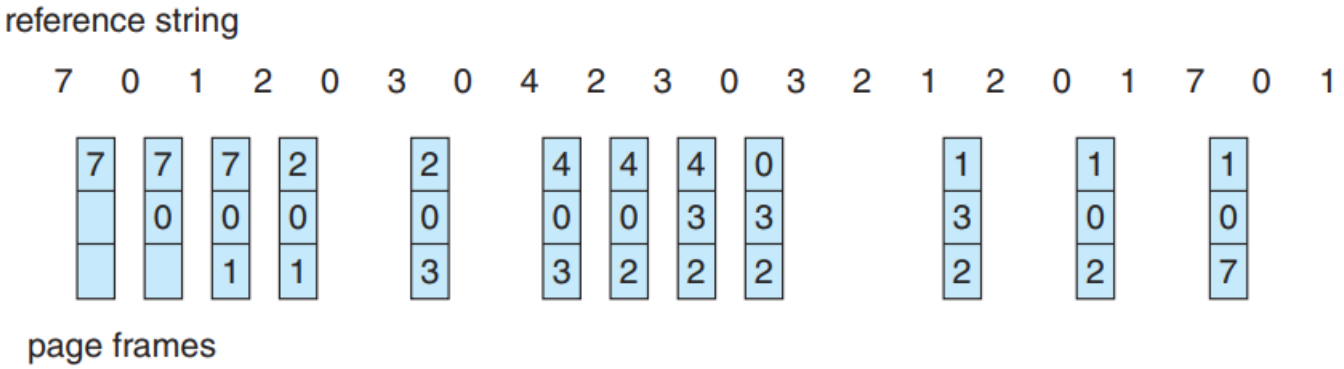


Figure 10.15 LRU page-replacement algorithm.

Triển khai LRU (2 cách)

- Counters
  - Thêm 1 trường time-of-use vào bảng page table, mỗi lần memory được tham chiếu, sẽ tăng lên 1.
  - Sẽ chọn page có giá trị "time" nhỏ nhất để thay thế
  - Nhược điểm: Phải tìm kiếm trên page table
- Stack
  - Tạo 1 stack chứa page number
  - Khi page được tham chiếu, nó sẽ được đưa ra khỏi stack và đặt lên top
  - Sẽ chọn page ở bottom của stack
  - Nhược điểm: Mỗi lần cập nhật stack tốn chi phí

Triển khai LRU (2 cách)

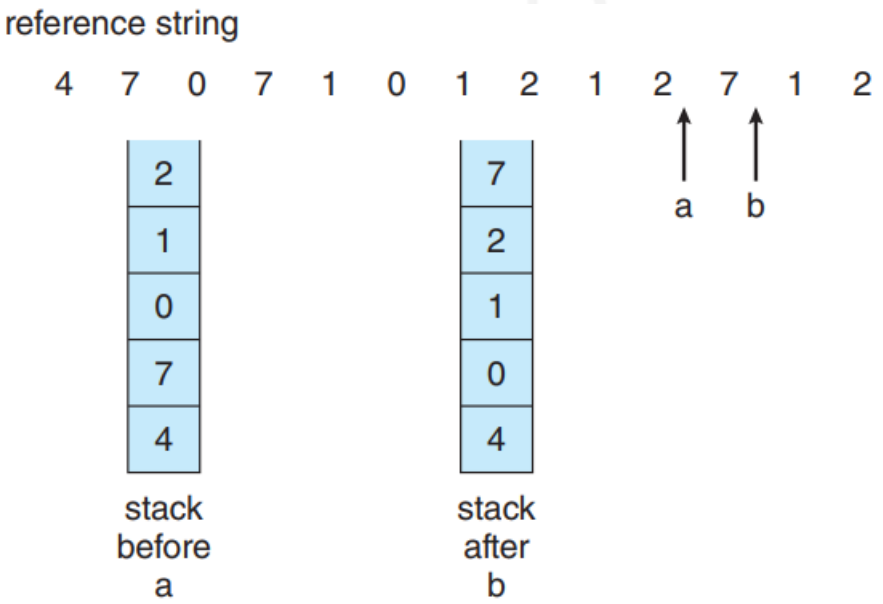


Figure 10.16 Use of a stack to record the most recent page references.

LRU-Approximation PR

- Reference bit – được set bởi phần cứng mỗi khi page được tham chiếu (được đọc hoặc ghi bất cứ byte nào của page)
- Tất cả các reference bit được khởi tạo đều là 0 bởi OS. Khi tiến trình thực thi, reference bit của page sẽ set lên 1 nếu page được được sử dụng.

Additional-Reference-Bits Algorithm

- Có thể thu được thông tin thứ tự bổ sung bằng cách ghi lại các reference bit đều đặn.
- Giữ một byte 8 bit cho mỗi trang trong bảng trong bộ nhớ. Trong những khoảng thời gian đều đặn (giả sử cứ sau 100 mili giây), ngắt hẹn giờ sẽ chuyển điều khiển sang hệ điều hành.
- OS chuyển reference bit cho mỗi page thành high-order bit, dịch chuyển các bit khác sang phải 1 bit và loại bỏ low-order bit → Thanh ghi này chứa lịch sử sử dụng page trong 8 khoảng thời gian gần nhất
- Chọn page có giá trị nhỏ nhất để thay thế

Second-Chance Algorithm

- Dựa trên FIFO
- Khi 1 page được chọn, nếu reference bit của nó có giá trị 0, thì thay đổi page; nếu reference bit có giá trị 1, cho page thêm 1 cơ hội mà di chuyển chọn FIFO page tiếp theo
- Khi 1 page đã có cơ hội lần 2, bit reference bit sẽ được set thành 0 và arrival time được reset thành thời điểm hiện tại.
- Như vậy, 1 page được trao cho cơ hội thứ 2 sẽ không bị thay thế cho đến khi tất cả các page khác bị thay thế (hay cho cơ hội lần 2).
- 1 page được sử dụng thường xuyên đủ để giữ reference bit là 1 thì sẽ không bao giờ bị thay thế

Triển khai Second-Chance Algorithm

- Sử dụng circular queue
- 1 con trỏ (đóng vai trò như kim đồng hồ) chỉ định page nào sẽ bị thay thế tiếp theo
- Khi frame cần, con trỏ sẽ đi tới trước cho đến khi gặp page có giá trị reference bit bằng 0, sau đó clear reference bit
- Khi đã tìm thấy page nạn nhân, page đó sẽ bị thay thế, và page mới sẽ được thêm vào circular queue tại vị trí đó
- Trong trường hợp xấu nhất, nếu tất cả reference bit đều được set, con trỏ sẽ đi vòng hết queue, cho mỗi page 1 cơ hội thứ 2. Nó sẽ xóa hết reference bit và second-change trở thành FIFO

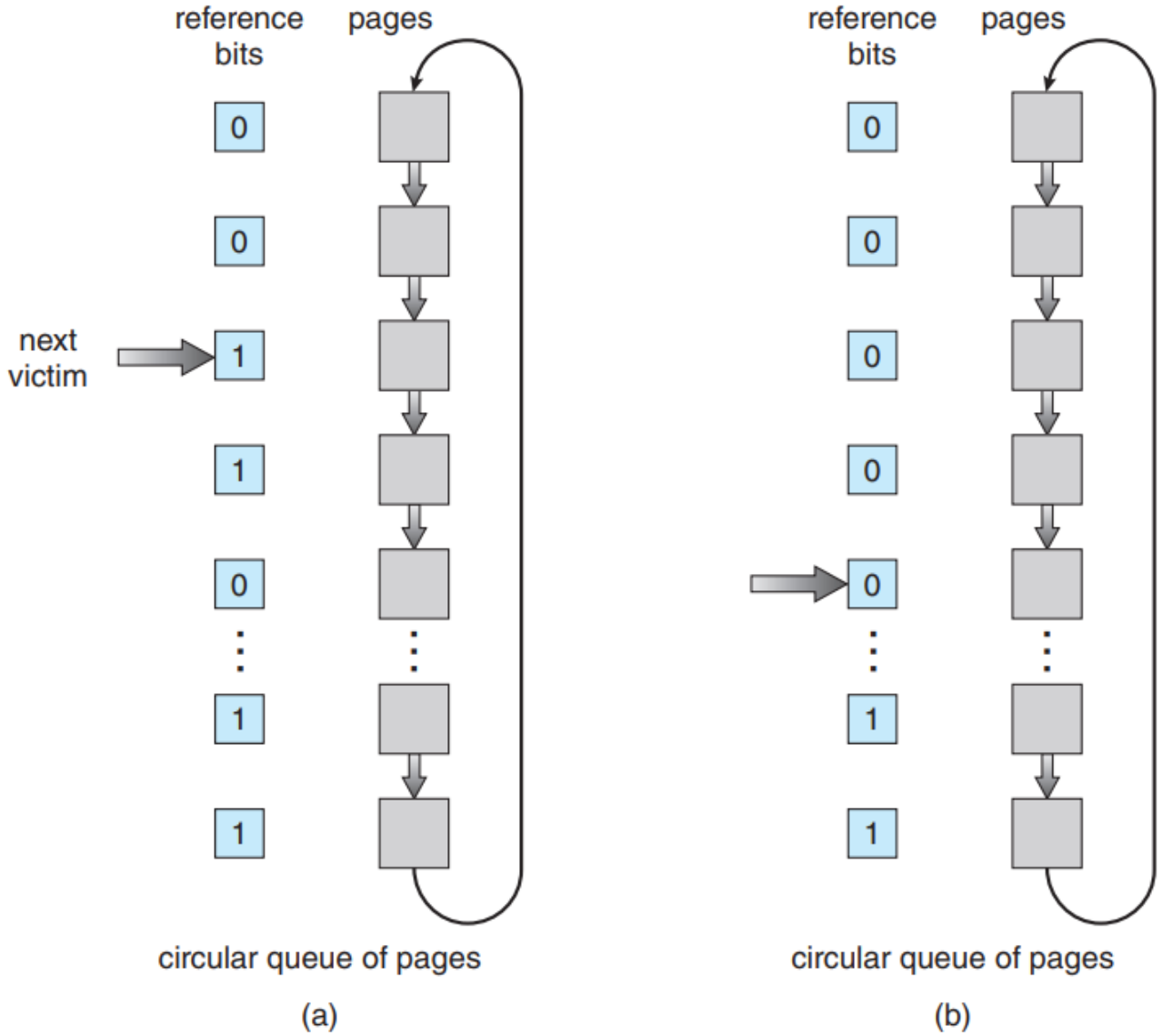


Figure 10.17 Second-chance (clock) page-replacement algorithm.

Mở rộng Second-Chance Algorithm

- Xét cả reference bit và modify bit theo cặp có thứ tự
  - (0,0) không được sử dụng gần đây và cũng không bị sửa đổi – thích hợp nhất để thay thế
  - (0,1) không được sử dụng gần đây nhưng đã bị sửa đổi – không tốt lắm, vì phải lưu lại nó vào backing store trước khi thay thế
  - (1,0) được sử dụng gần đây nhưng không bị sửa đổi – có thể sẽ được sử dụng lại sớm
  - (1,1) được sử dụng gần đây và đã bị sửa đổi – có thể sẽ được sử dụng lại sớm và cần phải lưu lại vào backing store trước khi bị thay thế
- Còn gọi là NRU (Not Recently Used)

Counting-Based Page Replacement

- Đếm số lượng reference cho mỗi page - count
  - LFU (least frequently used) – page nào có count nhỏ nhất thì bị thay thế
    - Problem: VD 1 page ban đầu được sử dụng rất là nhiều nhưng sau đó không sử dụng nữa thì biến count vẫn có giá trị lớn
    - Solution: Mỗi khoảng thời gian, dịch count sang phải 1 bit
    - Còn gọi là NFU (not frequently used)
  - MFU (most frequently used) – vì lập luận rằng những page có count nhỏ nhất có thể vừa được đưa vào memory và chưa kịp được sử dụng
- Cả 2 cách tiếp cận đối với biến count đều tốn kém và thua xa so với Optimal Replacement

Page-Buffering Algorithms

- Thay vì đợi đến khi thay thế page, frame nạn nhân được chọn rồi ghi ra backing store thì tạo 1 danh sách các page bị sửa đổi. Mỗi khi page nhàn rỗi thì page bị sửa đổi sẽ được ghi ra backing store (sau đó bit modify sẽ bị reset về 0)
  - Tăng tỉ lệ page clean thì khi chọn page để thay thế không phải tốn thời gian ghi ra backing store
- Tạo 1 nơi chứa free frame nhưng không xóa nội dung cũ mà giữ lại nó lưu page nào. Sau này nếu 1 page cũ được sử dụng lại thì tìm xem thử còn frame lưu nó không; nếu có thì tái sử dụng, ko cần phải I/O.

Bộ nhớ ảo

- Background
- Demand Paging
- Copy-on-write
- Page Replacement
- **Allocation of frames**
- Thrashing
- Memory Compression
- Other Considerations
- Operating-System Examples
- Summary

Cấp phát frame

- Minimum number of frames
- Allocation algorithms
- Global vs Local Allocation
- Non-Uniform Memory Access

Minimum Number of Frames

- Số lượng frame cấp phát cho mỗi tiến trình giảm thì page-fault tăng → làm chậm quá trình thực thi của tiến tình
- Số lượng fame ít nhất được cấp cho 1 tiến trình được quy định bởi kiến trúc máy tính (computer architecture – cụ thể là ISA Instruction set architecture).
  - Câu lệnh move với nhiều chế độ address có thể cần tới 2 frame
  - Mỗi toán hạng trong 2 toán hạng tham chiếu gián tiếp thì có thể cần tới 6 frames
- Số lượng frame nhiều nhất được cấp cho 1 tiến trình được quy định với dung lượng vật lý của RAM

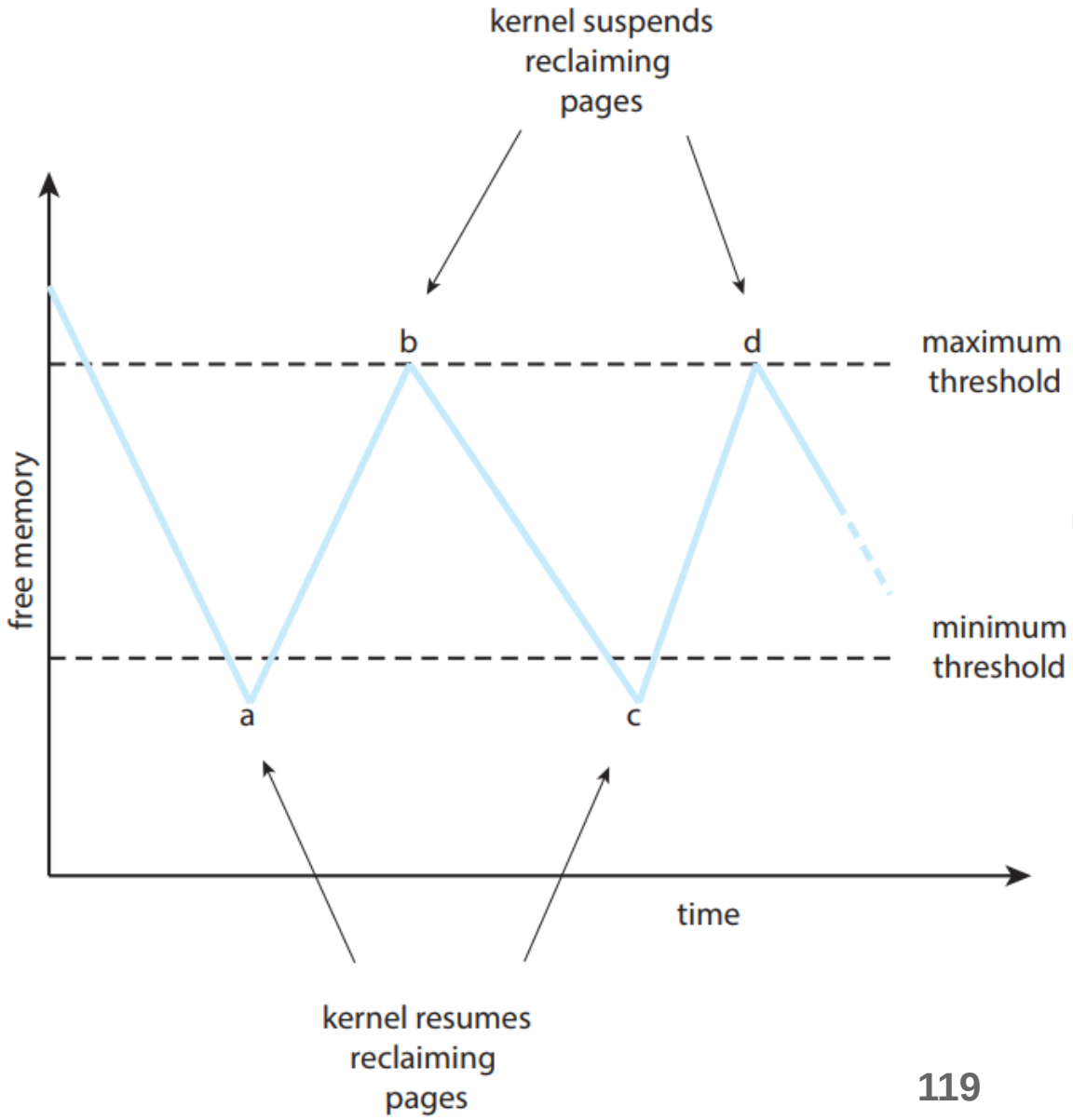
Allocation Algorithms

- Cách 1 dễ nhất để chia m frame cho n tiến trình là cấp phát m/n frame cho mỗi tiến trình. → equal allocation
  - VD 93 frame và 5 tiến trình thì mỗi tiến trình 18 frame và dư 3 frame được sử dụng là free-frame buffer pool.
- Cách 2 là chia theo tỉ lệ kích thước của tiến trình.
  - Giả sử kích thước của tiến trình p_i là s_i và $S = \sum s_i$
  - Nếu số lượng frame sẵn sàng (available) là m thì cấp phát a_i frame cho tiến trình p_i, trong đó a_i xấp xỉ $a_i = \frac{s_i}{S} \times \mathrm{m}$
  - Trong đó a_i là số nguyên phải lớn hơn số lượng frame ít nhất cần cấp phát cho tiến trình và tổng của chúng không vượt quá m

Global vs Local Allocation

- Global replacement
  - Cho phép 1 tiến trình chọn frame thay thế từ toàn bộ frame, ngay cả khi frame đó đang được sử dụng bởi tiến tình khác
- Local replacement
  - Chỉ cho phép tiến trình chọn frame thay thế là frame mà tiến trình đang sử dụng

Thu hồi trang



Thu hồi trang

- Trong nhiều trường hợp không thể thu hồi trang và free memory quá ít, OOM killer (out-of-memory) sẽ chọn tiến trình và chấm dứt nó để giải phóng bộ nhớ.
- Mỗi tiến trình trong Linux sẽ có 1 điểm OOM, tiến trình nào có điểm này cao sẽ bị chấm dứt, điểm này được tính theo phần trăm sử dụng bộ nhớ của tiến trình, phần trăm càng cao, điểm OOM càng cao
  - Có thể xem điểm ở /proc/<pid>/oom_score

Bộ nhớ ảo

- Background
- Demand Paging
- Copy-on-write
- Page Replacement
- Allocation of frames
- **Thrashing**
- Memory Compression
- Other Considerations
- Operating-System Examples
- Summary

Thrashing

- Nguyên nhân
- Working-Set Model
- Page-Fault Frequency

Nguyên nhân

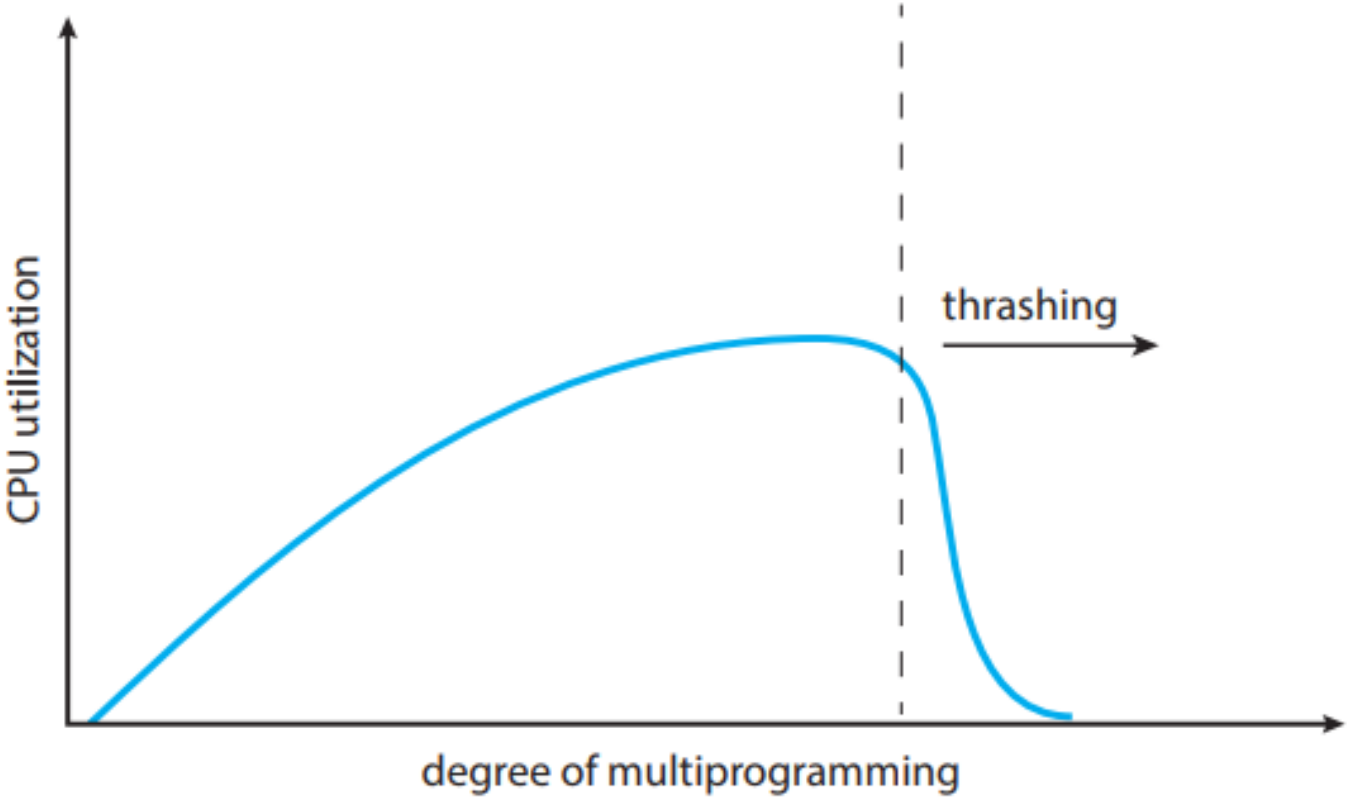


Figure 10.20 Thrashing.

Giải pháp

- Sử dụng local replacement algorithm (priority replacement algorithm)
  - Khi process bị thrashing, nó ko thể lấy frame của tiến trình khác và dẫn tới tiến trình đó cũng bị thrash theo.
  - Tuy nhiên nếu process bị thrash, thì nó sẽ nằm trong hàng đợi để paging hầu hết thời gian → vấn đề ko được giải quyết → làm ảnh hưởng đến thời gian truy cập bộ nhớ cho những tiến trình ko bị thrash
- Locality model

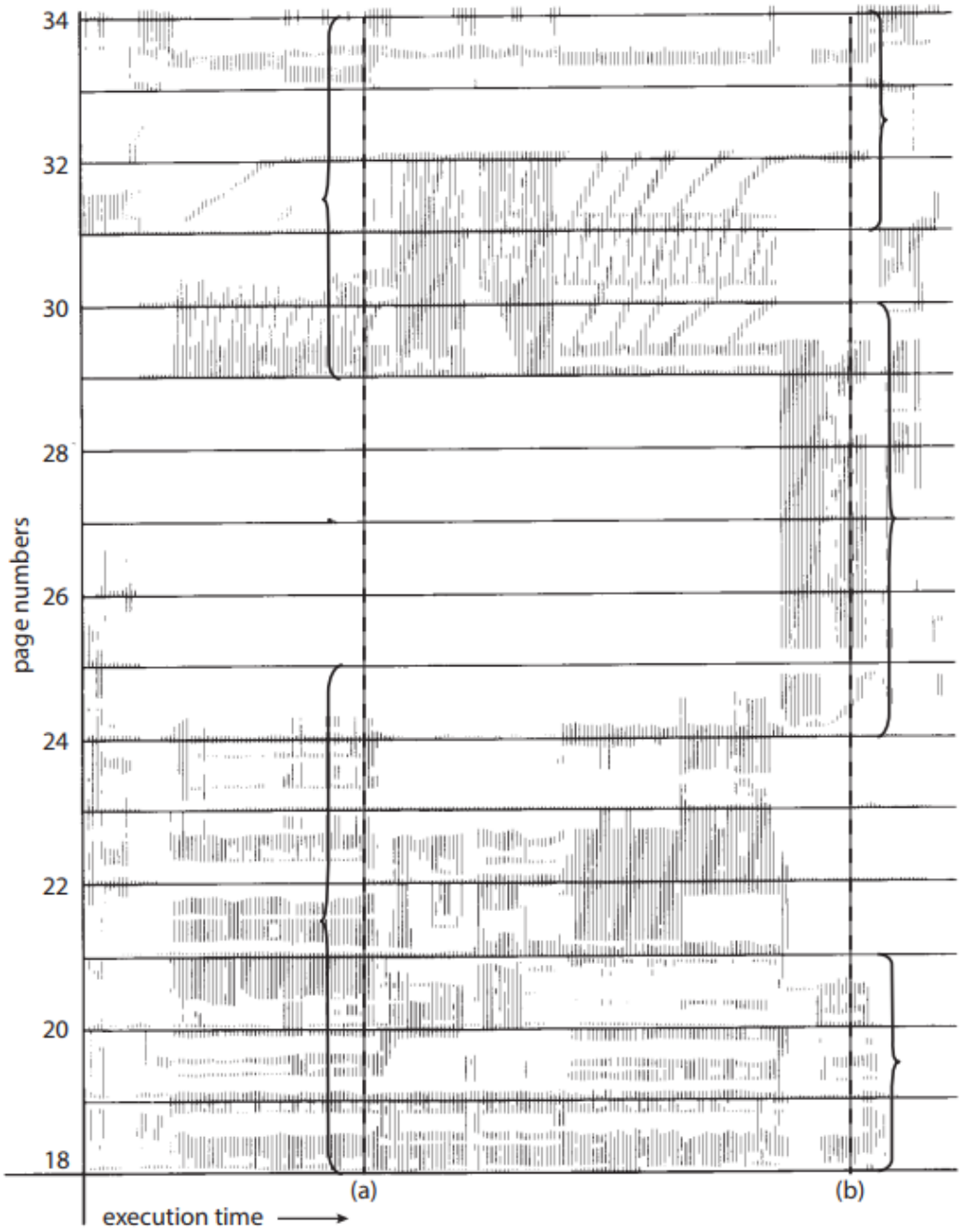


Figure 10.21 Locality in a memory-reference pattern.

Giải pháp

- Locality model
  - 1 locality là 1 tập hợp các page được sử dụng cùng nhau.
  - Khi 1 tiến trình thực thi, nó sẽ di chuyển từ locality này đến locality khác.
  - 1 chương trình thường bao gồm nhiều locality khác nhau và có thể chồng chéo lên nhau.
  - VD khi 1 hàm được gọi, nó sẽ xác định 1 locality mới nơi các tham chiếu bộ nhớ được thực hiện theo các câu lệnh của hàm (biến cục bộ, biến toàn cục, ...). Khi thoát hàm, tiến trình sẽ rời khỏi locality này

Working-Set Model

- Dựa trên giả thuyết của locality
- Sử dụng tham số Δ để định nghĩa working-set window – chứa tập các page tham chiếu
- Nếu page đang được sử dụng, nó sẽ nằm trong Δ, như vậy working set xấp xỉ locality của chương trình

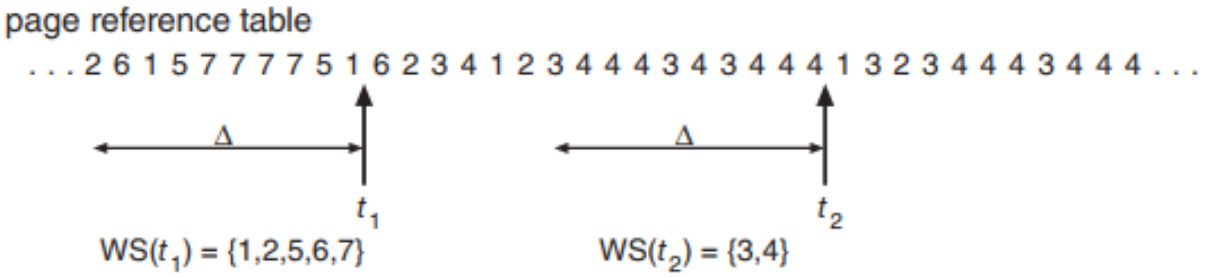


Figure 10.22 Working-set model.

Working-Set Model

- Nếu D là tổng số frame cần thiết theo nhu cầu (demand) và WSS_i là working set size của tiến trình i

$$D = \sum WSS_i$$

- Nếu m là số frame trống trong bộ nhớ, sẽ có 2 khả năng
  - Nếu $D > m$ thrashing sẽ xảy ra
  - Nếu $D \leq m$ thì sẽ không xảy ra thrashing

Page-Fault Frequency

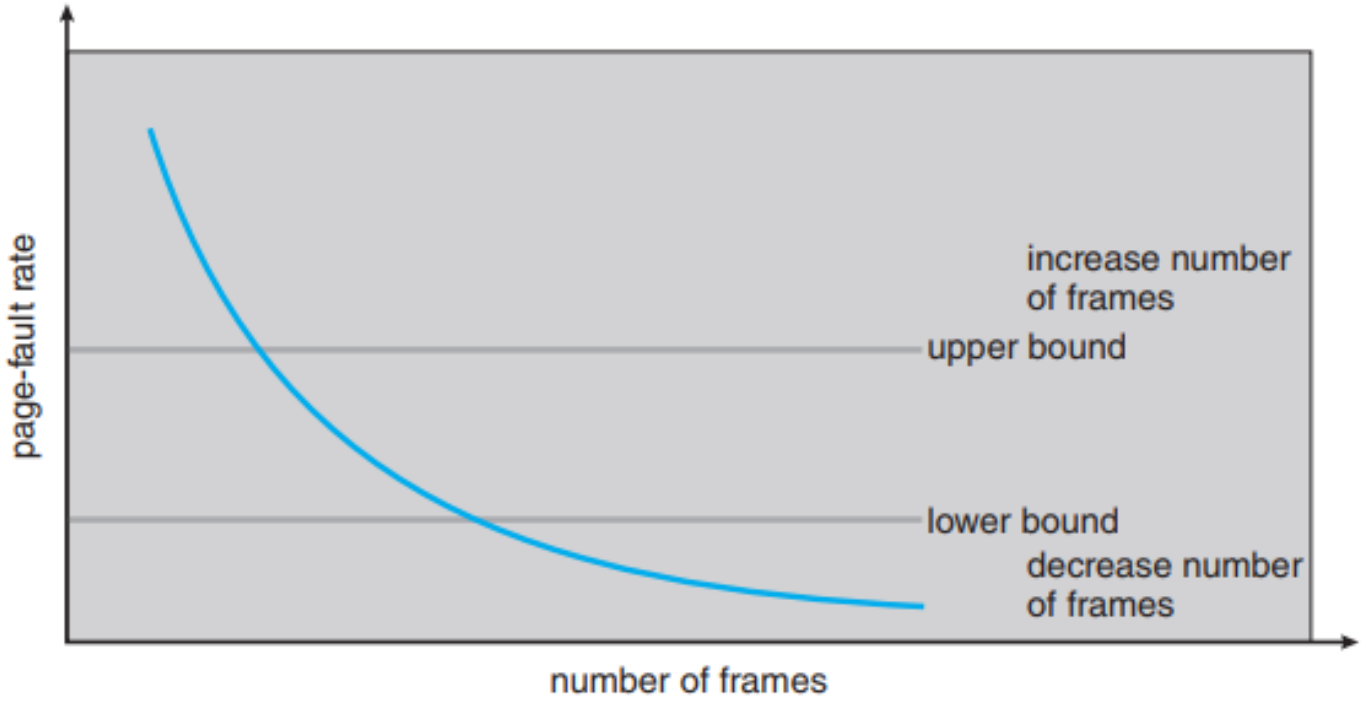


Figure 10.23 Page-fault frequency.

Bộ nhớ ảo

- Background
- Demand Paging
- Copy-on-write
- Page Replacement
- Allocation of frames
- Thrashing
- **Memory Compression**
- Other Considerations
- Operating-System Examples
- Summary

Memory compression

- Một cách khác để phân trang là nén bộ nhớ.
  - Thay vì paging theo frame để swap, nén nhiều frame lại thành 1 frame, cho phép hệ thống giảm mức sử dụng bộ nhớ mà không cần đến swap page.
  - Vì mobile system không hỗ trợ standard swapping hay swapping page nên nén bộ nhớ được sử dụng trong cả Android lẫn iOS
  - Win10 phát triển kiến trúc UWP (Universal Windows Platform)
  - macOS hỗ trợ nén bộ nhớ từ version 10.9. Hiệu suất nhanh hơn cả paging khi sử dụng secondary storage là SSD

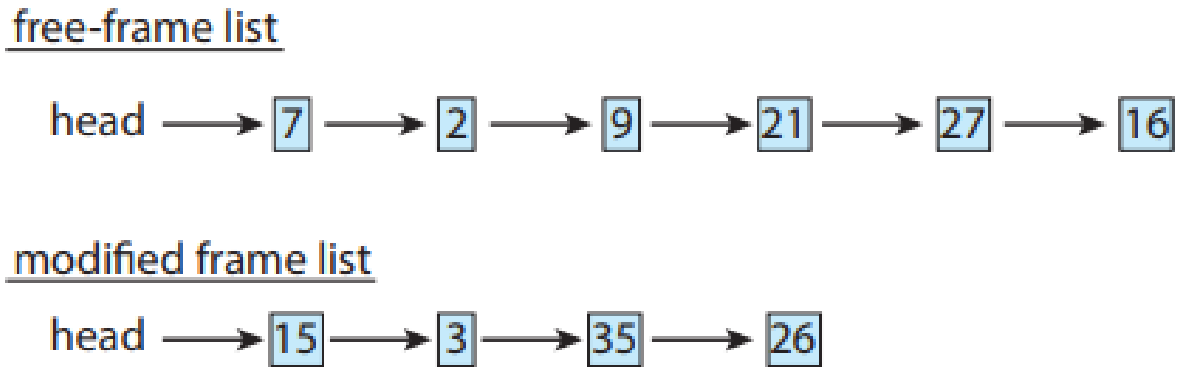


Figure 10.24 Free-frame list before compression.

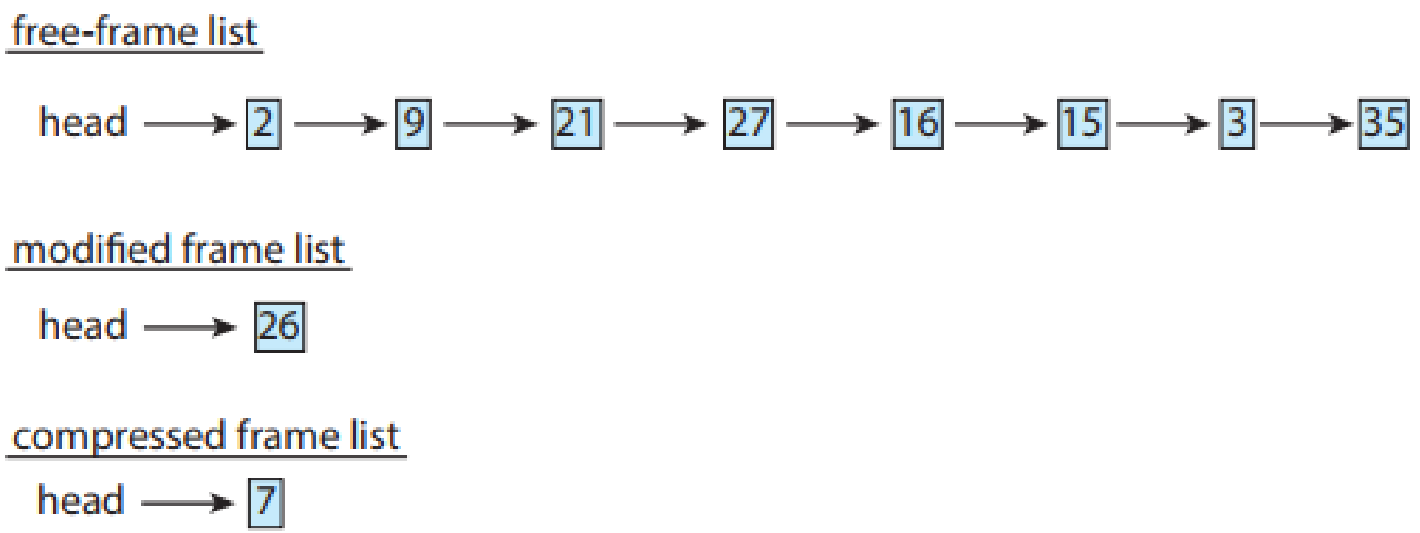


Figure 10.25 Free-frame list after compression

Allocating Kernel Memory

- Bộ nhớ kernel thường được cấp phát từ vùng nhớ khác với các tiến trình chạy ở user mode, có 2 nguyên nhân:
  - Kernel yêu cầu bộ nhớ có cấu trúc dữ liệu với nhiều kích thước khác nhau, trong số chúng có nhỏ hơn kích thước của 1 page → Nếu dùng page sẽ bị internal fragmentation. Điều này quan trọng vì nhiều OS không có kernel code hay data theo hệ thống paging
  - Page được cấp phát cho tiến trình user-mode không cần phải liên tiếp nhau trong RAM. Tuy nhiên, 1 vài thiết bị phần cứng tương tác trực tiếp với RAM → không thể thông qua virtual memory, cần cấp phát vùng nhớ liên tục

Allocating Kernel Memory

- 2 chiến thuật quản lý bộ nhớ đối với tiến trình kernel:
  - Buddy system
  - Slab allocation

Buddy System

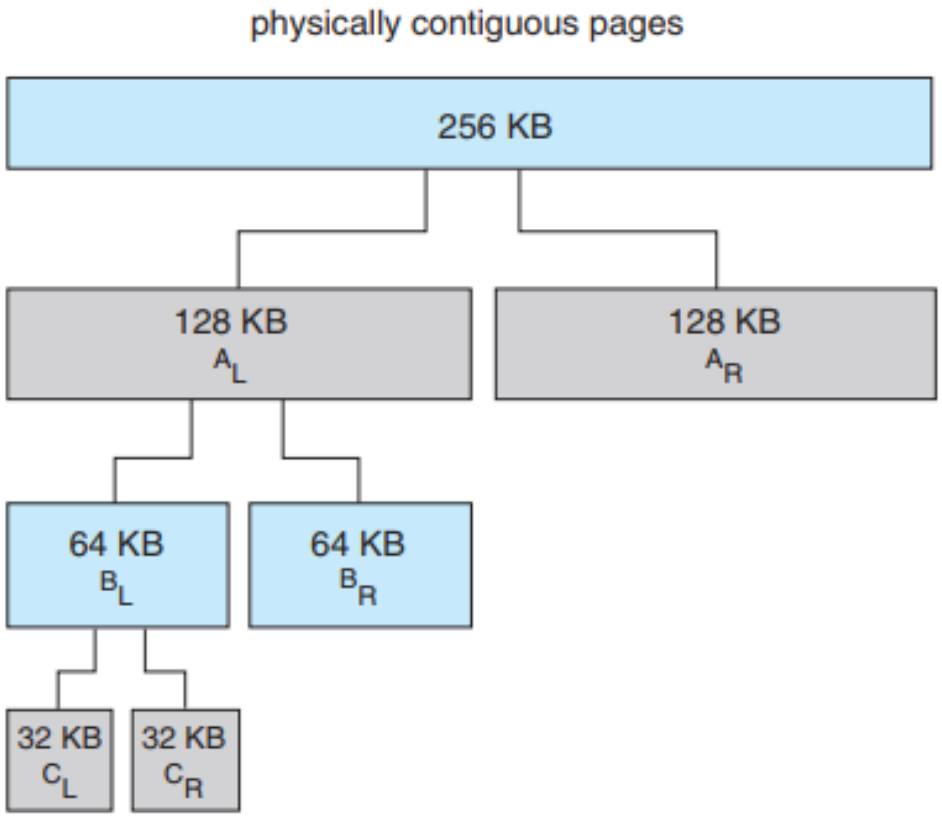


Figure 10.26 Buddy system allocation.

Buddy System

- Cấp phát vùng nhớ theo fixed-size segment, sử dụng power-of-2 allocator (phải thỏa mãn kích thước mũ 2 như 4KB, 8KB, 16KB, ..., nếu không thỏa mãn thì làm tròn lên mũ 2)
- VD kích thước của 1 segment ban đầu là 256KB và kernel cần 21KB bộ nhớ. Segment sẽ bị chia thành 2 buddy, gọi là A_L và A_R, mỗi cái 128KB. Sau đó một trong 2 buddy lại bị chia thành 2 64-KB buddy là B_L và B_R. Tiếp tục vậy ta có 32-KB buddy C_L và C_R. 1 trong 2 sẽ được sử dụng để lưu 21KB yêu cầu.

Slab Allocation

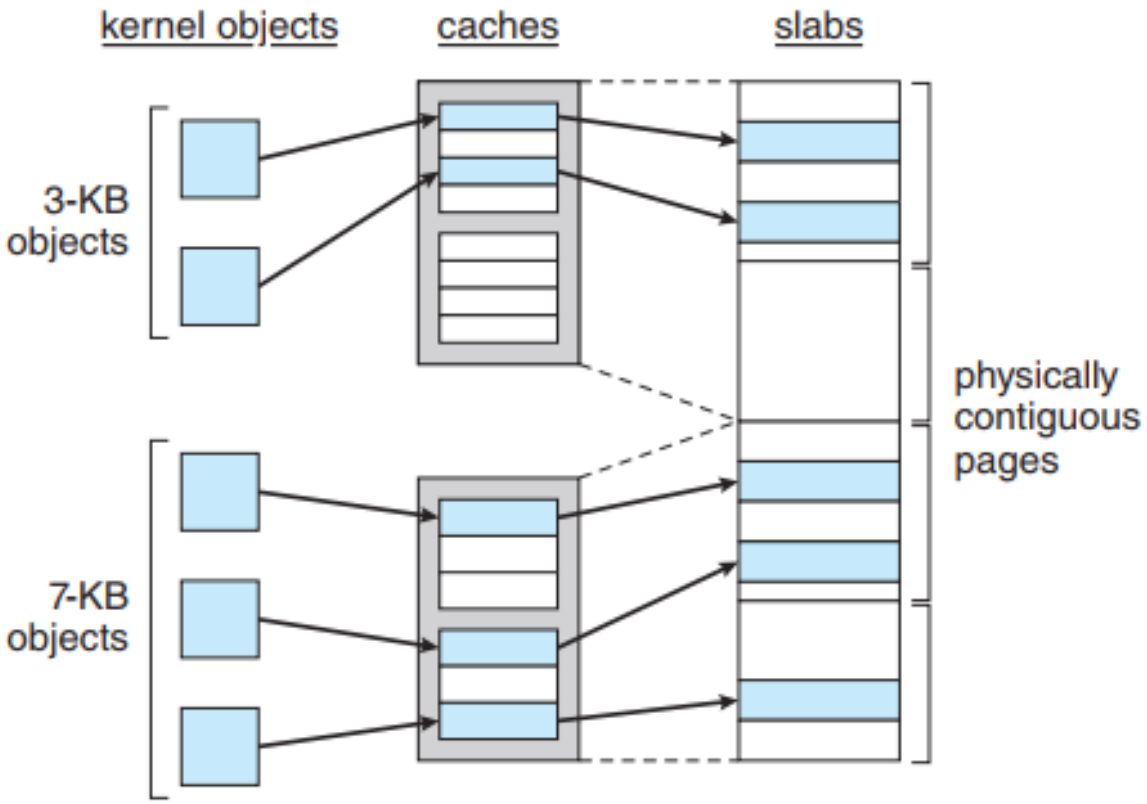


Figure 10.27 Slab allocation.

Slab Allocation

- 1 slab được tạo thành từ 1 hoặc nhiều trang liền kề về mặt vật lý
- 1 cache chứa 1 hoặc nhiều slab
- Chỉ có 1 cache đơn cho mỗi cấu trúc dữ liệu kernel duy nhất
  - 1 cache cho CTDL biểu diễn process descriptors
  - 1 cache cho file objects
  - 1 cache cho semaphores
  - ...

Triển khai Slab Allocation

- Thuật toán cấp phát slab sử dụng cache để lưu trữ kernel object.
- Khi 1 cache tạo ra, 1 vài object (khởi tạo free) được cấp phát cho cache. Số lượng object phụ thuộc vào kích thước của mỗi slab.
  - VD: 1 slab 12-KB sẽ chứa 6 object 2-KB.
- Trong Linux, 1 slab có 3 trang thái:
  - Full – tất cả objject trong slab đều được dùng
  - Empty – tất cả object đều free
  - Partial – dùng 1 phần

Ưu điểm Slab Allocation

- Không lãng phí bộ nhớ vì fragmentation vì mỗi CTDL của kernel đều có cache tương ứng, mỗi cache thì được tạo nên từ 1 hoặc nhiều slab đã được chia thành các kích thước theo object → Chính xác dung lượng bộ nhớ
- Yêu cầu bộ nhớ được đáp ứng nhanh chóng
- Cấp phát slab đặc biệt hiệu quả để quản lý các đối tượng thường xuyên được phân bổ hoặc giải phóng. Việc cấp phát và giải phòng bộ nhớ tốn thời gian. Tuy nhiên các object được tạo trước và do đó có thể được cấp phát nhanh chóng từ cache, nó được đánh dấu free và return lại cache, do đó sẵn sàng ngay lập tức cho yêu cầu tiếp theo của kernel.

Bộ nhớ ảo

- Background
- Demand Paging
- Copy-on-write
- Page Replacement
- Allocation of frames
- Thrashing
- Memory Compression
- **Other Considerations**
- **Operating-System Examples**
- Summary

Bộ nhớ ảo

- Background
- Demand Paging
- Copy-on-write
- Page Replacement
- Allocation of frames
- Thrashing
- Memory Compression
- Other Considerations
- Operating-System Examples
- **Summary**

Summary

- Bộ nhớ ảo trừu tượng hóa bộ nhớ vật lý thành 1 mảng lưu trữ thống nhất cực lớn
- Lợi ích của bộ nhớ ảo:
  - Bộ nhớ chương trình có thể lớn hơn bộ nhớ vật lý
  - Chương trình không cần phải nằm hoàn toàn trong bộ nhớ
  - Các tiến trình có thể chia sẻ bộ nhớ
  - Các tiến trình có thể thực thi hiệu quả hơn
- Phân trang theo yêu cầu (demand page) là một kỹ thuật trong đó các trang chỉ được load khi chúng được yêu cầu trong quá trình thực thi chương trình. Do đó các trang không được yêu cầu sẽ không bao giờ được tải vào bộ nhớ.

Summary

- Page fault xảy ra khi 1 page hiện không có trong bộ nhớ được truy cập. Page phải được đưa từ backing store vào frame trống trong bộ nhớ
- Copy-on-write cho phép 1 tiến trình con chia sẻ cùng 1 không gian địa chỉ với tiến trình cha. Nếu tiến trình con hoặc cha ghi (sửa đổi) 1 page thì 1 bản sao của page đó sẽ được tạo
- Khi bộ nhớ khả dụng sắp hết, thuật toán thay thế trang sẽ chọn một page hiện có trong bộ nhớ để thay thế bằng 1 page mới. Các thuật toán thay thế trang bao gồm FIFO, Optimal và LRU. Các thuật toán LRU thuần túy không thực tế để triển khai và thay vào đó, hầu hết đều sử dụng LRU gần đúng.

Summary

- Các thuật toán thay thế trang toàn cục chọn 1 trang từ bất kỳ tiến trình nào trong hệ thống để thay thế, trong khi các thuật toán thay thế trang cục bộ chọn 1 trang từ tiến trình bị page fault.
- Thrashing xảy ra khi hệ thống dành nhiều thời gian paging hơn là thực thi.
- Một locality đại diện cho 1 tập hợp các trang được sử dụng cùng nhau. Khi một tiến trình thực thi, nó di chuyển từ locality này sang locality khác. 1 tập hợp làm việc dựa trên locality và được định nghĩa là tập hợp các page hiện đang được 1 tiến trình sử dụng.
- Nén bộ nhớ là 1 kỹ thuật quản lý bộ nhớ bằng cách nén một số page thành 1 page duy nhất. Nó là giải pháp thay thế cho paging và được sử dụng trên các hệ thống di động không hỗ trợ phân trang

Summary

- Kernel memory được cấp phát khác với tiến trình ở chế độ người dùng; nó được phân bổ thành các khối liền kế có kích thước khác nhau. 2 kỹ thuật phổ biến để phân bộ bộ nhớ kernel là
  - Buddy system
  - Slab allocation
- Linux, Windows và Solaris quản lý bộ nhớ ảo tương tự nhau, sử dụng demand paging và copy-on-write, cùng các tính năng khác. Mỗi hệ thống cũng sử dụng 1 biến thể của LRU gần đúng được biết đến là clock algorithm.