HỆ ĐIỀU HÀNH

CHƯƠNG 7: QUẢN LÝ BỘ NHỚ

Trình bày các khái niệm cơ bản về bộ nhớ, các kiểu địa chỉ nhớ và cách chuyển đổi giữa các kiểu này, các cơ chế và mô hình quản lý bộ nhớ

Các nội dung đã học

- Chương 1: Tổng quan về hệ điều hành
- Chương 2: Cấu trúc hệ điều hành
- Chương 3: Quản lý tiến trình
- Chương 4: Định thời CPU
- Chương 5: Đồng bộ hoá tiến trình
- Chương 6: Tắc nghẽn
- **Chương 7: Quản lý bộ nhớ**
- Chương 8: Bộ nhớ ảo
- Chương 9: Hệ điều hành Linux và Hệ điều hành Windows

MỤC TIÊU

1. Hiểu được các khái niệm cơ sở về bộ nhớ
2. Hiểu được các kiểu địa chỉ nhớ và cách chuyển đổi giữa các kiểu này
3. Hiểu được các cơ chế và mô hình quản lý bộ nhớ

NỘI DUNG

1. Khái niệm cơ sở
2. Các kiểu địa chỉ nhớ
3. Chuyển đổi địa chỉ nhớ
4. Mô hình quản lý bộ nhớ
5. Cơ chế phân trang
6. Cơ chế swapping

KHÁI NIỆM CƠ SỞ

01.

7.1. Khái niệm cơ sở

- Chương trình phải được mang vào trong bộ nhớ và đặt nó trong một tiến trình để được xử lý.
- Input Queue – Một tập hợp của những tiến trình trên đĩa mà đang chờ để được mang vào trong bộ nhớ để thực thi.
- User programs trải qua nhiều bước trước khi được xử lý.

7.1. Khái niệm cơ sở

7.1. Khái niệm cơ sở

7.1. Khái niệm cơ sở

- **Quản lý bộ nhớ** là công việc của hệ điều hành với sự hỗ trợ của phần cứng nhằm phân phối, sắp xếp các tiến trình trong bộ nhớ sao cho hiệu quả.
- **Mục tiêu cần đạt được** là nạp càng nhiều tiến trình vào bộ nhớ càng tốt (gia tăng mức độ đa chương).
- Trong hầu hết các hệ thống, **kernel sẽ chiếm một phần cố định của bộ nhớ**; phần còn lại phân phối cho các tiến trình.

7.1. Khái niệm cơ sở

Các yêu cầu đối với việc quản lý bộ nhớ

- Cấp phát bộ nhớ cho các tiến trình
- Tái định vị (relocation): khi swapping,…
- Bảo vệ: kiểm tra truy xuất bộ nhớ có hợp lệ không
- Chia sẻ: cho phép các tiến trình chia sẻ vùng nhớ chung
- Kết gán địa chỉ nhớ luận lý của user vào địa chỉ thực

CÁC KIỂU ĐỊA CHỈ NHỚ

02.

7.2. Các kiểu địa chỉ nhớ

- **Địa chỉ vật lý** (**physical address**) (**địa chỉ thực**) là một vị trí thực trong bộ nhớ chính.
- **Địa chỉ luận lý** (**logical address**) là một vị trí nhớ được diễn tả trong một chương trình (còn gọi là địa chỉ ảo - **virtual address**).
  - Các trình biên dịch (compiler) tạo ra mã lệnh chương trình mà trong đó mọi tham chiếu bộ nhớ đều là địa chỉ luận lý.
- **Địa chỉ tuyệt đối** (**absolute address**): địa chỉ tương đương với địa chỉ thực.
- **Địa chỉ tương đối** (**relative address**) (địa chỉ *khả tái định vị*, relocatable address) là một kiểu địa chỉ luận lý trong đó các địa chỉ được biểu diễn tương đối so với một vị trí xác định nào đó trong chương trình.
  - Ví dụ: 12 byte so với vị trí bắt đầu chương trình,…

7.2. Các kiểu địa chỉ nhớ

7.2. Các kiểu địa chỉ nhớ

Nạp chương trình vào bộ nhớ

- **Bộ linker**: kết hợp các object module thành một file nhị phân khả thực thi gọi là load module.
- **Bộ loader**: nạp (load) module vào bộ nhớ chính.

7.2. Các kiểu địa chỉ nhớ

Cơ chế thực hiện linking

7.2. Các kiểu địa chỉ nhớ

Nạp chương trình vào bộ nhớ

CHUYỂN ĐỔI ĐỊA CHỈ

7.3.1. Chuyển đổi địa chỉ

03.

7.3.1. Chuyển đổi địa chỉ

- **Chuyển đổi địa chỉ**: quá trình ánh xạ một địa chỉ từ không gian địa chỉ này sang không gian địa chỉ khác.
- **Biểu diễn địa chỉ nhớ**
  - Trong source code: symbolic (các biến, hằng, pointer,…)
  - Trong thời điểm biên dịch: thường là địa chỉ khả tái định vị.
    - Ví dụ: a ở vị trí 12 byte so với vị trí bắt đầu module.
  - Thời điểm linking/loading: có thể là địa chỉ thực.
    - Ví dụ: dữ liệu nằm tại địa chỉ bộ nhớ thực 2030.

7.3.1. Chuyển đổi địa chỉ

- Địa chỉ lệnh và dữ liệu được chuyển đổi thành địa chỉ thực có thể xảy ra tại **ba thời điểm** khác nhau.
  - **Compile time**: nếu biết trước địa chỉ bộ nhớ của chương trình thì có thể kết gán địa chỉ tuyệt đối lúc biên dịch.
    - Ví dụ: chương trình .COM của MS-DOS.
    - Khuyết điểm: phải biên dịch lại nếu thay đổi địa chỉ nạp chương trình.
  - **Load time**: vào thời điểm loading, loader phải chuyển đổi địa chỉ khả tái định vị thành địa chỉ thực dựa trên một địa chỉ nền.
    - Địa chỉ thực được tính toán vào thời điểm nạp chương trình.

    => Phải tiến hành reload nếu địa chỉ nền thay đổi.

7.3.1. Chuyển đổi địa chỉ

Sinh địa chỉ tuyệt đối vào thời điểm dịch

7.3.1. Chuyển đổi địa chỉ

Sinh địa chỉ tuyệt đối vào thời điểm nạp

7.3.1. Chuyển đổi địa chỉ

- **Excution time**: khi trong quá trình thực thi, tiến trình có thể được di chuyển từ segment này sang segment khác trong bộ nhớ thì quá trình chuyển đổi địa chỉ được trì hoãn đến thời điểm thực thi.
    - Cần sự hỗ trợ của phần cứng cho việc ánh xạ địa chỉ
        - Ví dụ: Trường hợp địa chỉ luận lý là relocatable thì có thể dùng thanh ghi base và limit, …
    - Sử dụng trong đa số các OS đa dụng trong đó có các cơ chế swapping, paging, segmentation, …

CHUYỂN ĐỔI ĐỊA CHỈ

7.3.2. Dynamic linking

03.

7.3.2. Dynamic linking

- Quá trình link đến một module ngoài (external module) được thực hiện sau khi đã tạo xong load module (i.e. file có thể thực thi, executable).
  - Ví dụ trong Windows: module ngoài là các file .DLL còn trong Unix, các module ngoài là các file .so (shared library).
- Load module chứa các stub tham chiếu (refer) đến routine của external module.
  - Lúc thực thi, khi stub được thực thi lần đầu (do process gọi routine lần đầu), stub nạp routine vào bộ nhớ, tự thay thế bằng địa chỉ của routine và routine được thực thi.
  - Các lần gọi routine sau sẽ xảy ra bình thường.
- Stub cần sự hỗ trợ của OS (như kiểm tra xem routine đã được nạp vào bộ nhớ chưa).

7.3.2. Dynamic linking

Ưu điểm

- Thông thường, external module là một thư viện cung cấp các tiện ích của OS. Các chương trình thực thi có thể dùng các phiên bản khác nhau của external module mà không cần sửa đổi, biên dịch lại.
- Chia sẻ mã (code sharing): một external module chỉ cần nạp vào bộ nhớ một lần. Các tiến trình cần dùng external module này thì cùng chia sẻ đoạn mã của external module ⇒ tiết kiệm không gian nhớ và đĩa.
- Phương pháp dynamic linking cần sự hỗ trợ của OS trong việc kiểm tra xem một thủ tục nào đó có thể được chia sẻ giữa các tiến trình hay là phần mã của riêng một tiến trình (bởi vì chỉ có OS mới có quyền thực hiện việc kiểm tra này).

CHUYỂN ĐỔI ĐỊA CHỈ

7.3.3. Dynamic Loading

03.

7.3.3. Dynamic loading

- Cơ chế: chỉ khi nào cần được gọi đến thì một thủ tục mới được nạp vào bộ nhớ chính ⇒ tăng độ hiệu dụng của bộ nhớ bởi vì các thủ tục không được gọi đến sẽ không chiếm chỗ trong bộ nhớ.
- Rất hiệu quả trong trường hợp tồn tại khối lượng lớn mã chương trình có tần suất sử dụng thấp, không được sử dụng thường xuyên (ví dụ các thủ tục xử lý lỗi).
- Hỗ trợ từ hệ điều hành
  - Thông thường, user chịu trách nhiệm thiết kế và hiện thực các chương trình có dynamic loading.
  - Hệ điều hành chủ yếu cung cấp một số thủ tục thư viện hỗ trợ, tạo điều kiện dễ dàng hơn cho lập trình viên.

MÔ HÌNH QUẢN LÝ BỘ NHỚ

04.

7.4. Mô hình quản lý bộ nhớ

- Trong chương này, mô hình quản lý bộ nhớ là một mô hình đơn giản, không có bộ nhớ ảo.
- Một tiến trình phải được nạp hoàn toàn vào bộ nhớ thì mới được thực thi.
- Các cơ chế quản lý bộ nhớ:
  - Phân chia cố định (fixed partitioning)
  - Phân chia động (dynamic partitioning)
  - Phân trang đơn giản (simple paging)
  - Phân đoạn đơn giản (simple segmentation)

7.4. Mô hình quản lý bộ nhớ

7.4. Mô hình quản lý bộ nhớ

Phân mảnh (fragmentation)

- Phân mảnh ngoại (external fragmentation)
  - Kích thước không gian nhớ còn trống đủ để thỏa mãn một yêu cầu cấp phát, tuy nhiên không gian nhớ này không liên tục ⇒ có thể dùng cơ chế kết khối (compaction) để gom lại thành vùng nhớ liên tục.
- Phân mảnh nội (internal fragmentation)
  - Kích thước vùng nhớ được cấp phát có thể hơi lớn hơn vùng nhớ yêu cầu.
    - Ví dụ: cấp một khoảng trống 18,464 bytes cho một tiến trình yêu cầu 18,462 bytes.
  - Hiện tượng phân mảnh nội thường xảy ra khi bộ nhớ thực được chia thành các khối kích thước cố định (fixed-sized block) và các tiến trình được cấp phát theo đơn vị khối. Ví dụ: cơ chế phân trang (paging).

7.4. Mô hình quản lý bộ nhớ

Phân mảnh nội

OS sẽ cấp phát hẳn khối 18,464 bytes cho process ⇒ dư ra 2 bytes không dùng!

MÔ HÌNH QUẢN LÝ BỘ NHỚ

7.4.1. Fixed partitioning

04.

7.4.1. Fixed partitioning

- Khi khởi động hệ thống, bộ nhớ chính được chia thành nhiều phần rời nhau gọi là các partition có kích thước bằng nhau hoặc khác nhau.
- Tiến trình nào có kích thước nhỏ hơn hoặc bằng kích thước partition thì có thể được nạp vào partition đó.
- Nếu chương trình có kích thước lớn hơn partition thì phải dùng cơ chế overlay.
- Nhận xét
  - Không hiệu quả do bị phân mảnh nội: một chương trình dù lớn hay nhỏ đều được cấp phát trọn một partition.

7.4.1. Fixed partitioning - Chiến lược placement

- Partition có kích thước bằng nhau
  - Nếu còn partition trống ⇒ tiến trình mới sẽ được nạp vào partition đó.
  - Nếu không còn partition trống, nhưng trong đó có tiến trình đang bị blocked ⇒ swap tiến trình đó ra bộ nhớ phụ nhường chỗ cho tiến trình mới.
- Partition có kích thước không bằng nhau: giải pháp 1
  - Gán mỗi tiến trình vào partition nhỏ nhất phù hợp với nó.
  - Có hàng đợi cho mỗi partition.
  - Giảm thiểu phân mảnh nội.
  - Vấn đề: có thể có một số hàng đợi trống không (vì không có tiến trình với kích thước tương ứng) và hàng đợi dày đặc.

7.4.1. Fixed partitioning - Chiến lược placement

- Partition có kích thước không bằng nhau: giải pháp 2
  - Chỉ có một hàng đợi chung cho mọi partition.
  - Khi cần nạp một tiến trình vào bộ nhớ chính

  ⇒ chọn partition nhỏ nhất còn trống.

MÔ HÌNH QUẢN LÝ BỘ NHỚ

7.4.2. Dynamic partitioning

04.

7.4.2. Dynamic partitioning

- Số lượng partition không cố định và partition có thể có kích thước khác nhau.
- Mỗi tiến trình được cấp phát chính xác dung lượng bộ nhớ cần thiết.
- Gây ra hiện tượng phân mảnh ngoại.

7.4.2. Dynamic partitioning - Chiến lược placement

- Dùng để quyết định cấp phát khối bộ nhớ trống nào cho một tiến trình.
- Mục tiêu: giảm chi phí compaction.
- Các chiến lược placement
  - Best-fit: chọn khối nhớ trống nhỏ nhất.
  - First-fit: chọn khối nhớ trống phù hợp đầu tiên kể từ đầu bộ nhớ.
  - Next-fit: chọn khối nhớ trống phù hợp đầu tiên kể từ vị trí cấp phát cuối cùng.
  - Worst-fit: chọn khối nhớ trống lớn nhất.

Example Memory Configuration Before and After Allocation of 16 Kbyte Block

CƠ CHẾ PHÂN TRANG

05.

7.5. Cơ chế phân trang

- Cơ chế cấp phát bộ nhớ không liên tục
- Chia bộ nhớ vật lý thành các khối cố định gọi là các khung trang (frames).
  - Kích thước của frame là lũy thừa của 2, từ khoảng 512 byte đến 16MB.
- Chia bộ nhớ luận lý thành các khối nhớ bằng nhau gọi là các trang (pages).
  - Kích thước của page bằng kích thước của frame.
- Chương trình có N trang cần N khung trống (free frames) trong bộ nhớ để nạp vào.
- Thiết lập bảng phân trang (page table) để ánh xạ địa chỉ luận lý thành địa chỉ thực.

7.5. Cơ chế phân trang

CƠ CHẾ PHÂN TRANG

7.5.1. Chuyển đổi địa chỉ trong paging

05.

7.5.1. Chuyển đổi địa chỉ trong paging

- Địa chỉ luận lý gồm có:
  - **Số hiệu trang (Page number) p**
  - **Địa chỉ tương đối trong trang (Page offset) d**
- Nếu kích thước của không gian địa chỉ ảo là 2^m, và kích thước của trang là 2^n (đơn vị là byte hay word tùy theo kiến trúc máy) thì:

- Bảng trang sẽ có tổng cộng $2^m/2^n = 2^{m-n}$ mục (entry)

7.5.1. Chuyển đổi địa chỉ trong paging

7.5.1. Chuyển đổi địa chỉ trong paging

7.5. Cơ chế phân trang

Before allocation

After allocation

CƠ CHẾ PHÂN TRANG

7.5.2. Cài đặt bảng trang

05.

7.5.2. Cài đặt bảng trang (paging hardware)

- Bảng phân trang thường được lưu giữ trong bộ nhớ chính:
  - Mỗi tiến trình được hệ điều hành cấp một bảng phân trang.
  - **Thanh ghi page-table base** (**PTBR**) trỏ đến bảng phân trang.
  - **Thanh ghi page-table length** (**PTLR**) biểu thị kích thước của bảng phân trang (có thể được dùng trong cơ chế bảo vệ bộ nhớ).
- Theo cơ chế cài đặt này thì một thao tác truy cập lệnh hoặc dữ liệu cần **đến 2 lần truy cập bộ nhớ chính.**
  - Lần 1 cho bảng trang.
  - Lần 2 cho lệnh hoặc dữ liệu.
- 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).

7.5.2. Cài đặt bảng trang

Dùng thanh ghi Page-Table Base Register (PTBR)

7.5.2. Cài đặt bảng trang

Dùng TLB

CƠ CHẾ PHÂN TRANG

7.5.3. Effective Access Time (EAT)

05.

7.5.3. Effective access time (EAT)

- Tính thời gian truy xuất hiệu dụng (effective access time - EAT)
- Thời gian tìm kiếm trong TLB (associative lookup): ε
- Thời gian một chu kỳ truy xuất bộ nhớ: x
- Hit ratio (α): tỉ số giữa số lần chỉ số trang được tìm thấy (hit) trong TLB và số lần truy xuất khởi nguồn từ CPU.
- Thời gian cần thiết để có được chỉ số frame:
  - Khi chỉ số trang có trong TLB (hit) $\varepsilon + x$
  - Khi chỉ số trang không có trong TLB (miss) $\varepsilon + x + x$
- Thời gian truy xuất hiệu dụng:

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

7.5.3. Effective access time (EAT)

Ví dụ 1: đơn vị thời gian nano giây

- Associative lookup = 20
- Memory access = 100
- Hit ratio = 0.8
- EAT = (100 + 20) × 0.8 + (200 + 20) × 0.2

  = 1.2 × 100 + 20

  = 140

Ví dụ 2: đơn vị thời gian nano giây

- Associative lookup = 20
- Memory access = 100
- Hit ratio = 0.98
- EAT = (100 + 20) × 0.98 + (200 + 20) × 0.02

  = 1.02 × 100 + 20

  = 122

CƠ CHẾ PHÂN TRANG

7.5.4. Tổ chức bảng trang

05.

7.5.4. Tổ chức bảng trang

- Các hệ thống hiện đại đều hỗ trợ không gian địa chỉ ảo rất lớn (2^{32} đến 2^{64}), ở đây giả sử là 2^{32}.
  - Giả sử kích thước trang nhớ là 4KB (= 2^{12})

  ⇒ bảng phân trang sẽ có $2^{32}/2^{12} = 2^{20}$ = 1M mục.
  - Giả sử mỗi mục gồm 4 byte thì mỗi tiến trình cần 4MB cho bảng phân trang.
    - Ví dụ: Phân trang 2 cấp

7.5.4. Tổ chức bảng trang

7.5.4. Tổ chức bảng trang

- Bảng trang nghịch đảo (IBM system/38, IBM RISC, IBM RT): sử dụng cho tất cả các tiến trình.

<IDP,p,d>

CƠ CHẾ PHÂN TRANG

7.5.5. Bảo vệ bộ nhớ

05.

7.5.5. Bảo vệ bộ nhớ

- Việc bảo vệ bộ nhớ được hiện thực bằng cách gắn với frame các bit bảo vệ (protection bits) được giữ trong bảng phân trang. 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 của tiến trình, do đó là một trang hợp lệ.
  - “invalid”: cho biết là trang không của tiến trình, do đó là một trang bất hợp lệ.

7.5.5. Bảo vệ bộ nhớ

Bảo vệ bằng valid / invalid bit

- Mỗi trang nhớ có kích thước 2K = 2048.
- Tiến trình có kích thước10468 ⇒ phân mảnh nội ở frame 9 (chứa page 5), các địa chỉ ảo > 12287 là các địa chỉ invalid.
- Dùng PTLR để kiểm tra truy xuất đến bảng phân trang có nằm trong bảng hay không.

7.5.5. Bảo vệ bộ nhớ - Chia sẻ các trang nhớ

Cơ chế hoán vị (swapping)

06.

7.6. Cơ chế hoán vị (swapping)

- Một tiến trình có thể tạm thời bị swap ra khỏi bộ nhớ chính và lưu trên một hệ thống lưu trữ phụ. Sau đó, tiến trình có thể được nạp lại vào bộ nhớ để tiếp tục quá trình thực thi.
- Swapping policy: hai ví dụ
  - Round-robin: swap out P1 (vừa tiêu thụ hết quantum của nó), swap in P2 , thực thi P3 ,…
  - Roll out, roll in: dùng trong cơ chế định thời theo độ ưu tiên (priority-based scheduling)
    - Tiến trình có độ ưu tiên thấp hơn sẽ bị swap out nhường chỗ cho tiến trình có độ ưu tiên cao hơn mới đến được nạp vào bộ nhớ để thực thi.
- Hiện nay, ít hệ thống sử dụng cơ chế swapping trên.

7.6. Cơ chế hoán vị (swapping)

Tóm tắt lại nội dung buổi học

- Khái niệm cơ sở
- Các kiểu địa chỉ nhớ
- Chuyển đổi địa chỉ nhớ
- Mô hình quản lý bộ nhớ
- Cơ chế phân trang
- Cơ chế swapping

Bài tập 1

Giả sử bộ nhớ chính được phân chia thành các phân vùng có kích thước là 600K, 500K, 200K, 300K (theo thứ tự). Các tiến trình có kích thước 212K, 417K, 112K, 426K (theo thứ tự) sẽ được cấp phát bộ nhớ như thế nào, nếu sử dụng: Thuật toán First fit, Best fit, Next fit, Worst fit? Thuật toán nào cho phép sử dụng bộ nhớ hiệu quả nhất trong trường hợp trên?

Bài tập 2

Xét một không gian địa chỉ có 12 trang, mỗi trang có kích thước 2K, ánh xạ vào bộ nhớ vật lý có 32 khung trang.

a. Địa chỉ logic gồm bao nhiêu bit?

b. Địa chỉ physic gồm bao nhiêu bit?

Bài tập 3

Xét một hệ thống sử dụng kỹ thuật phân trang, với bảng trang được lưu trữ trong bộ nhớ chính.

a. Nếu thời gian cho một lần truy xuất bộ nhớ bình thường là 200ns thì mất bao nhiêu thời gian cho một thao tác truy xuất bộ nhớ trong hệ thống này?

b. Nếu sử dụng TLBs với hit-ratio là 75%, thời gian để tìm trong TLBs xem như bằng 0, tính thời gian truy xuất bộ nhớ (EAT) trong hệ thống.

Bài tập 4

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

Bài tập 5

Giả sử địa chỉ ảo 32-bit được phân tách thành 4 trường a, b, c, d. 3 trường đầu tiên được dùng cho bảng trang 3 cấp, trường thứ 4 dành cho offset. Số lượng trang có phụ thuộc vào kích thước của cả 4 trường này không? Nếu không, những trường nào ảnh hưởng đến số lượng trang, những trường nào không ảnh hưởng?

THẢO LUẬN