Chương 3. Quản lý lưu trữ

Nội dung chương 3

1. Quản lý bộ nhớ
2. Bộ nhớ ảo
3. Giao diện hệ thống file
4. Cài đặt hệ thống file

1. Quản lý bộ nhớ

1. Cơ sở
2. Swapping
3. Phân phối bộ nhớ liên tục
4. Phân trang (paging)
5. Phân đoạn (segmentation)
6. Phân đoạn kết hợp với phân trang (Segmentation với Paging)

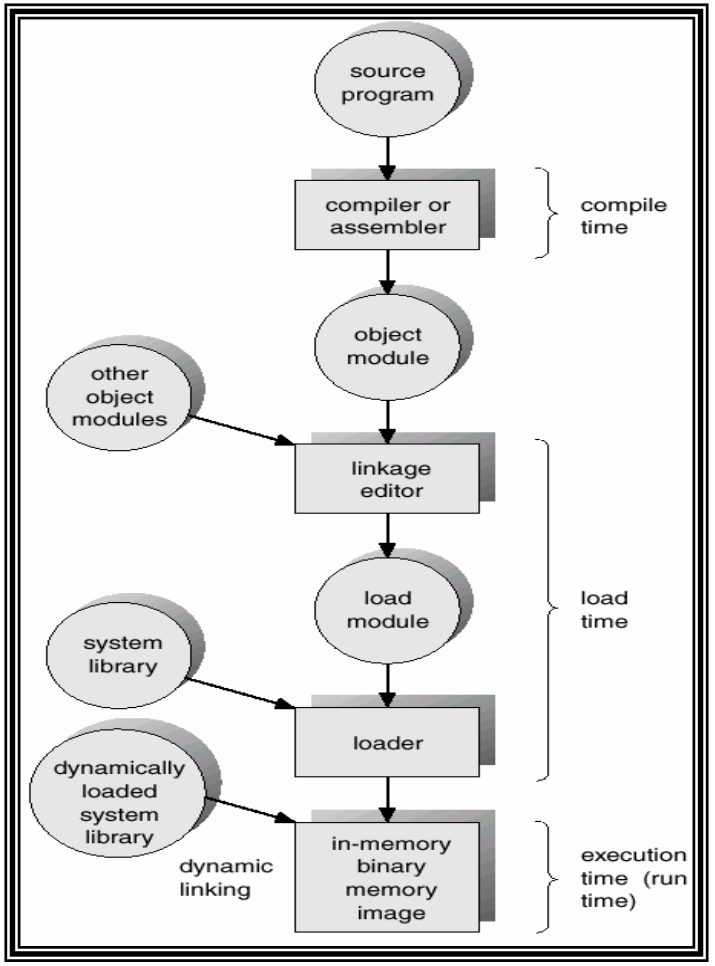
1.1. Cơ sở

- Chương trình muốn thực thi cần phải được tải vào bộ nhớ và đặt trong một tiến trình
- *Hàng đợi vào (Input Queue)*
  - Tập các tiến trình trên đĩa, đang đợi tải vào bộ nhớ để thực hiện
- Các chương trình người dùng muốn được thực thi cần phải qua *một số bước* trong đó có bước gán địa chỉ cho các câu lệnh/dữ liệu.

Gán bộ nhớ cho các câu lệnh và dữ liệu

- Việc gán địa chỉ cho các câu lệnh và dữ liệu được thực thi tại các thời điểm
  - **Biên dịch** – nếu vị trí trong bộ nhớ đã được biết trước – sinh ra mã tuyệt đối (absolute code); cần phải được biên dịch lại nếu vị trí bắt đầu bị thay đổi
  - **Lúc tải (loading time)** – phải sinh ra mã có thể định vị lại (relocatable code) – nếu vị trí trong bộ nhớ không được biết trước
    - Mã có thể định vị lại "14 bytes kể từ đầu module"
  - **Lúc thực thi** – Gán địa chỉ được ***trì hoãn cho đến khi thực thi*** nếu tiến trình có thể thay đổi, từ đoạn bộ nhớ này đến đoạn bộ nhớ khác trong khi thực thi.
    - Yêu cầu phần cứng hỗ trợ cho các ánh xạ địa chỉ (thanh ghi cơ sở, thanh ghi giới hạn)

Các bước xử lý của tiến trình người dùng



Không gian địa chỉ vật lý và không gian địa chỉ logic

- Khái niệm không gian *địa chỉ logic* gắn với *không gian địa chỉ vật lý* là trung tâm của các kĩ thuật quản lý bộ nhớ
  - *Các địa chỉ logic* – được sinh ra bởi CPU; còn được gọi là *địa chỉ ảo*
  - *Địa chỉ vật lý* – địa chỉ thật trong bộ nhớ, thấy được bởi đơn vị quản lý bộ nhớ
- Như nhau trong lược đồ gán địa chỉ lúc biên dịch, tải
- Khác nhau trong lược đồ gán địa chỉ lúc thực thi

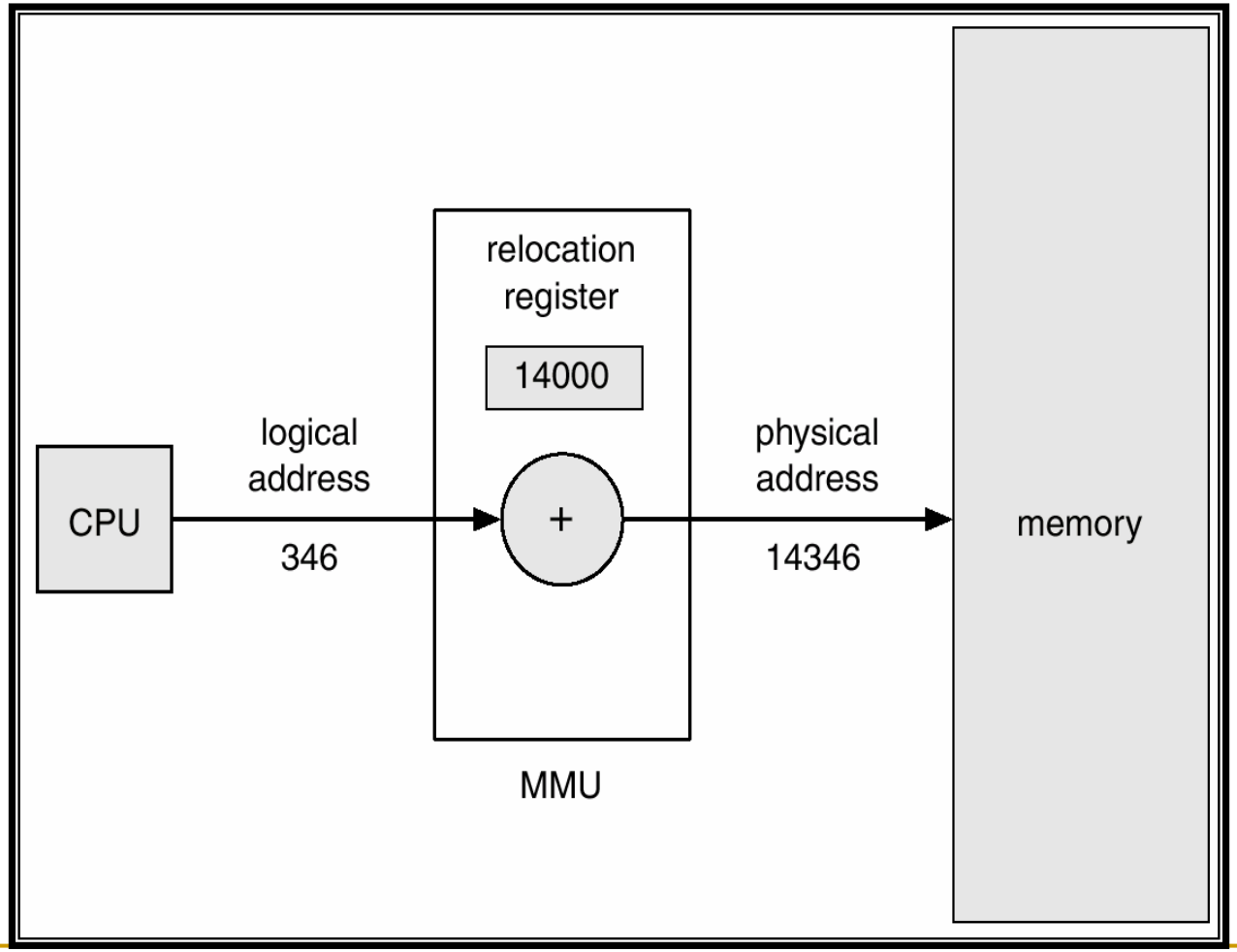
Đơn vị quản lý bộ nhớ (MMU)

- Thiết bị phần cứng thực hiện việc ánh xạ địa chỉ ảo đến địa chỉ vật lý

- Ví dụ về 1 lược đồ MMU đơn giản
  - Giá trị thanh ghi relocation được cộng vào cho mỗi địa chỉ được sinh ra bởi tiến trình người dùng tại thời điểm nó tải vào bộ nhớ.

- Chương trình người dùng làm việc với các *địa chỉ logic;* nó không bao giờ thấy địa chỉ vật lý

Gán địa chỉ động với thanh ghi relocation



Tải động vào bộ nhớ

- Các phương thức không được tải vào bộ nhớ khi nó được gọi
- Tận dụng không gian bộ nhớ tốt hơn
  - Phương thức không được sử dụng sẽ không bao giờ được tải
- Hữu ích khi cần lượng mã lớn để xử lý các trường hợp không thường xuyên
- Không cần phải có sự hỗ trợ đặc biệt của hệ điều hành trong thiết kế chương trình

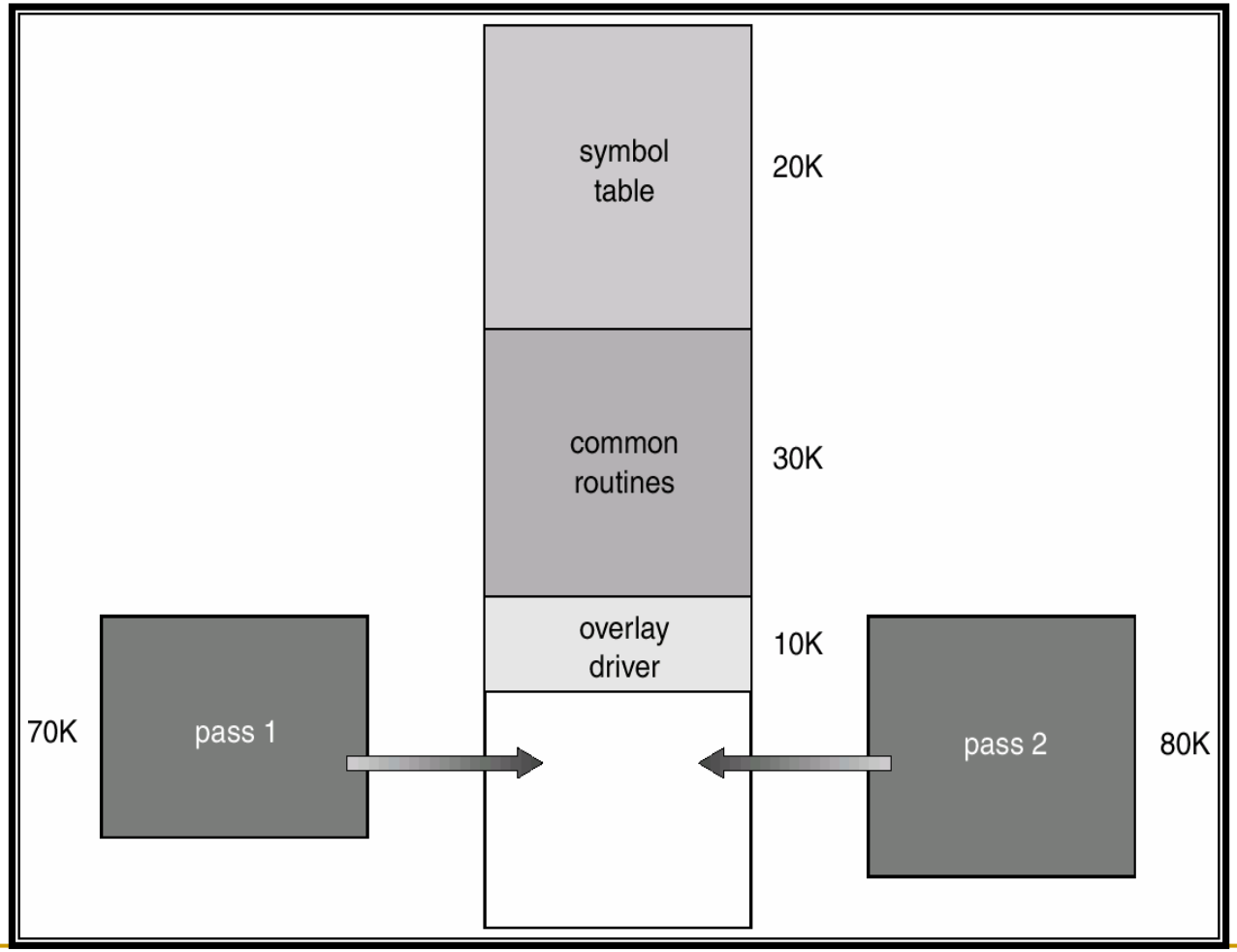
Liên kết động

- Việc liên kết sẽ bị trì hoãn đến thời gian thực thi
- Các đoạn mã nhỏ, gọi là *stub*, được sử dụng để xác định thủ tục thư viện trong vùng bộ nhớ thích hợp.
- Stub được thay thế bởi địa chỉ vật lý của routine và thực thi routine
- Hệ điều hành cần phải kiểm tra xem liệu phương thức có nằm trong địa chỉ bộ nhớ của tiến trình
- Liên kết động rất hữu hiệu cho các thư viện

Overlays

- Chỉ giữ trong bộ nhớ những câu lệnh và dữ liệu cần trong bất cứ thời điểm nào
- Cần khi tiến trình lớn hơn kích cỡ bộ nhớ được gán cho nó
- Được thực thi bởi người dùng, không cần sự hỗ trợ đặc biệt từ hệ điều hành, thiết kế lập trình của cấu trúc overlay tương đối phức tạp

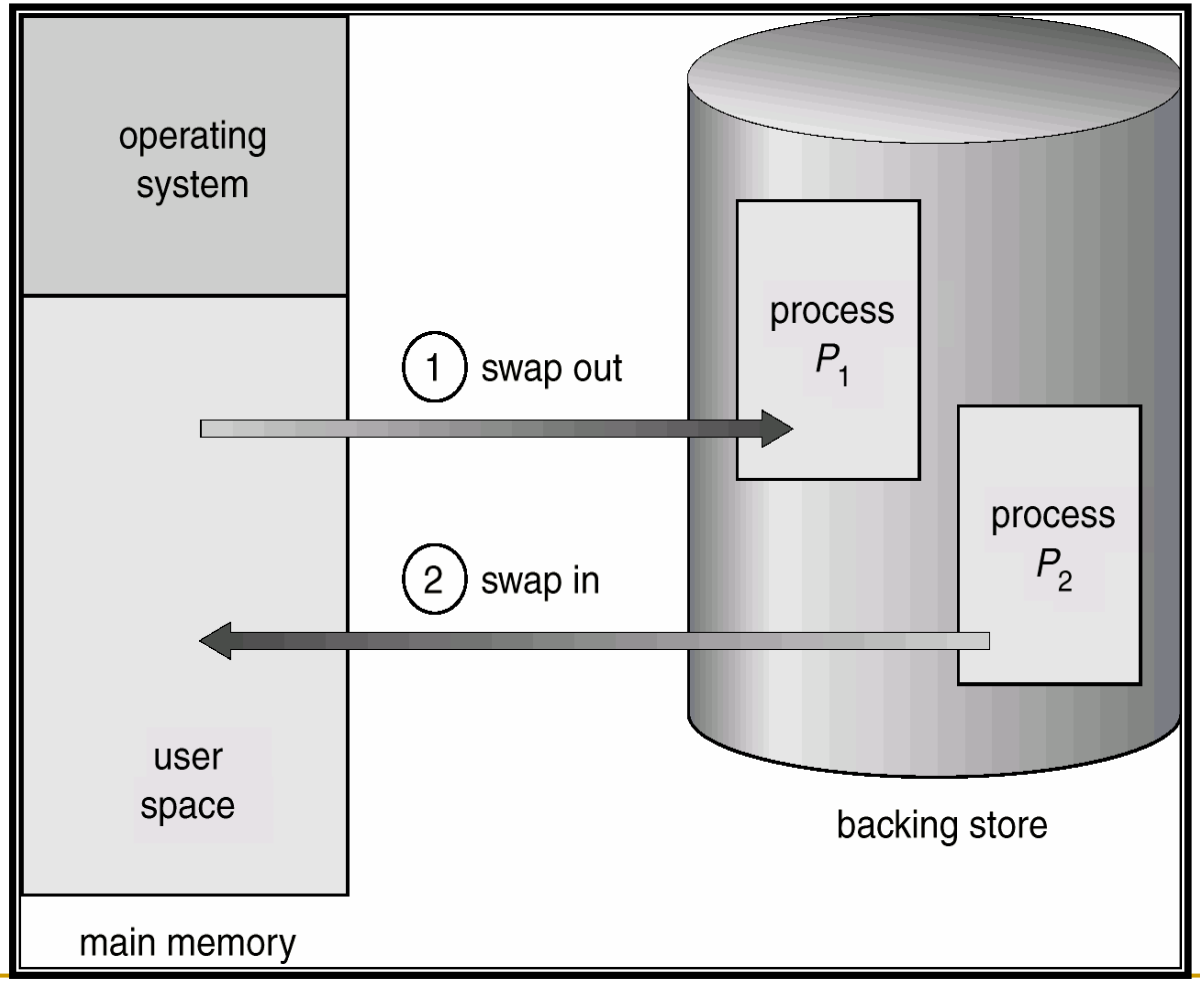
Overlays


symbol table
20K
common routines
30K
overlay driver
10K
70K
pass 1
pass 2
80K

1.2. Swapping

- Một tiến trình có thể bị *swapped* tạm ra *bộ lưu trữ nền* , sau đó được mang trở lại bộ nhớ để thực thi tiếp
- Bộ lưu trữ nền – đĩa tốc độ nhanh, đủ lớn để lưu trữ phiên bản của tất cả ảnh bộ nhớ cho tất cả người dùng; phải cung cấp khả năng truy cập trực tiệp đến các ảnh bộ nhớ này.
- *Roll out, roll in* – biến thể swapping được sử dụng trong thuật toán lấp lịp có ưu tiến; tiến trình có độ ưu tiên thấp nhất bị swap ra cho phép tiến trình có độ ưu tiên cao nhất được tải vào và thực thi.
- Một trong những giai đoạn quan trọng trong thời gian swap là thời gian chuyển đổi ngữ cảnh
  - Tổng thời gian chuyển giao tỉ lệ với tổng *dung lượng* bộ nhớ bị swap.
- Ta có thể thấy nhiều phiên bản biến thể của trên rất nhiều hệ thống, i.e., UNIX, Linux, and Windows.

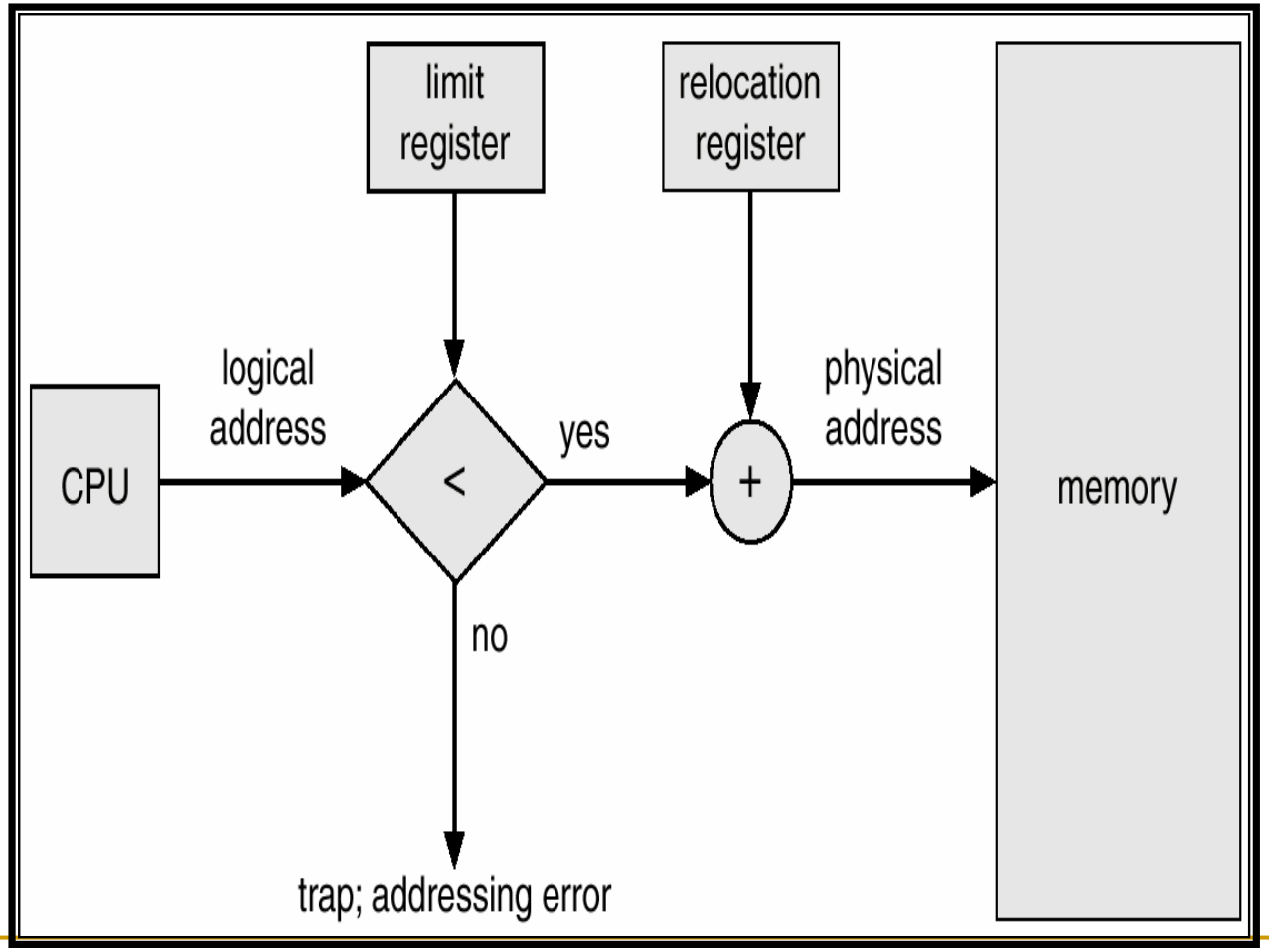
Lược đồ Swapping



1.3. Phân phối bộ nhớ liên tục

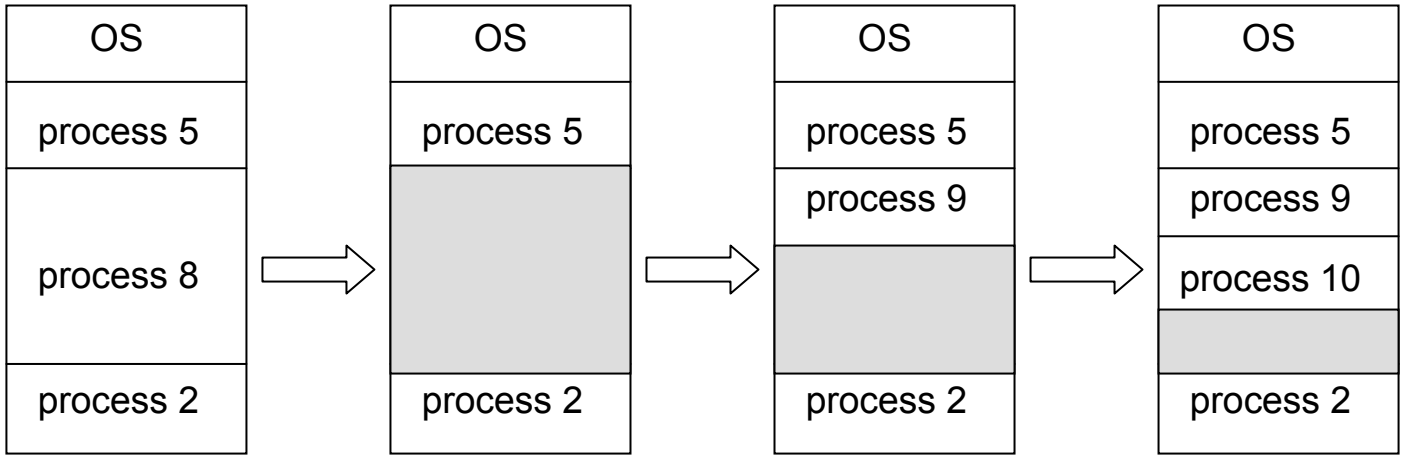
- Bộ nhớ chính thường được chia thành hai phần
  - Phần lưu trú hệ điều hành, thường được tổ chức trong vùng bộ nhớ thấp (địa chỉ thấp) với vector ngắt.
  - Các tiến trình người dùng, thường được tổ chức trong vùng bộ nhớ cao.
- Bảo vệ
  - Lược đồ thanh ghi relocation cho việc bảo vệ các tiến trình người dùng.
  - Thay ghi relocation chứa giá trị của địa chỉa vật lý nhỏ nhất; thanh ghi giới hạn chứa các giá trị từ miền địa chỉ logic – các địa chỉ logic phải có giá trị nhỏ hơn giá trị của thanh ghi giới hạn.

Hỗ trợ phần cứng cho các thanh ghi relocation và thanh ghi limit



Phân phối liên tục (Cont.)

- Phân phối đa phân đoạn
  - *Lỗ hổng*– khối bộ nhớ rỗi; các lỗ hổng với những kích cỡ khác nhau nằm rải rác trong bộ nhớ.
  - Khi một tiến trình cần tải vào bộ nhớ, nó được phân phối vùng bộ nhớ từ lỗ hổng đủ lớn chứa nó.
  - Hệ điều hành quản lý thông tin về:
    a) các phân đoạn đã được phân phối b) Các phân đoạn rỗi (lỗ hổng)



Bài toán phân phối bộ nhớ động

- Làm thế nào để phân phối tiến trình có kích cỡ *n* vào một danh sách các lỗ hổng còn rỗi
  - First-fit: tìm lỗ hổng đầu tiên đủ lớn
  - Best-fit: tìm lỗ hổng bé nhất, đủ lớn
    - Tìm kiếm trên toàn bộ danh sách các lỗ hổng (trừ phi các lỗ hổng được sắp xếp theo kích cỡ)
    - Sinh ra phần thừa nhỏ nhất
  - Worst-fit: tìm lỗ hổng lớn nhất
    - Cũng phải tìm kiếm
    - Sinh ra phần thừa lớn nhất
- First-fit và best-fit tốt hơn chiến lược worst-fit trên quan điểm tốc độ và sự tận dụng bộ nhớ

Sự phân mảnh

- **Phân mảnh ngoài** – tổng không gian bộ nhớ có thể đáp ứng yêu cầu, nhưng không liên tục
- **Phân mảnh trong** – bộ nhớ được phân phối có thể lớn hơn một chút so với yêu cầu; sự khác biệt về kích cỡ này là nội trong một phân đoạn, và ko được sử dụng
- Làm giảm phân mảnh ngoài bằng kết khối
  - Xáo các nội dung bộ nhớ để đặt tất cả vùng bộ nhớ rỗi cạnh nhau tạo thành một khối lớn
  - Kết khối chỉ thích hợp khi việc phân đoạn lại là động và được thực hiện tại lúc thực thi