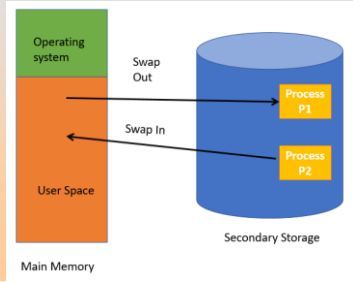
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ ĐÔNG Á
EAST ASIA UNIVERSITY OF TECHNOLOGY

Chương 4

Bộ nhớ chính (Main memory)



Nội dung:

- Kiến thức cơ bản
- Swapping
- Phân phối bộ nhớ liên tiếp - Contiguous Allocation
- Phân trang - Paging
- Phân đoạn - Segmentation
- Kết hợp phân đoạn với phân trang
  - Segmentation with Paging
- Ví dụ: Intel Pentium

Mục tiêu

- Cung cấp mô tả chi tiết các cách tổ chức phần cứng bộ nhớ.
- Thảo luận các kỹ thuật quản lý bộ nhớ khác nhau, bao gồm phân trang và phân đoạn.
- Cung cấp mô tả chi tiết về Intel Pentium, bộ xử lý hỗ trợ cả phân đoạn đơn thuần và phân đoạn kết hợp với phân trang.

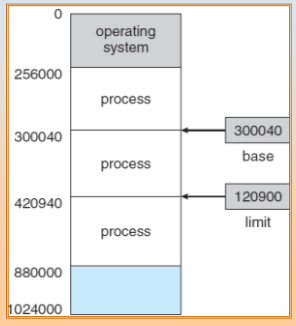
4.1. Kiến thức cơ bản

4.1.1. Về phần cứng

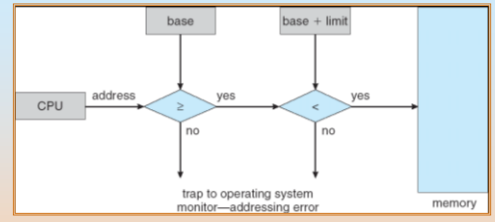
- Chương trình phải được đưa từ đĩa vào bộ nhớ chính và được đặt vào một tiến trình để nó có thể chạy.
- Bộ nhớ chính, cache, các thanh ghi là bộ nhớ mà CPU có thể truy nhập trực tiếp.
- Yêu cầu bộ nhớ phải được bảo vệ để đảm bảo sự hoạt động đúng đắn.

Các thanh ghi base và limit

- Cặp thanh ghi base và limit xác định không gian địa chỉ hợp lệ mà tiến trình của người sử dụng được phép truy nhập.



Bảo vệ bộ nhớ với các thanh ghi base & limit



Sự bảo vệ bộ nhớ được thực hiện bằng cách so sánh mọi địa chỉ trong user mode với các thanh ghi base và limit. Nếu địa chỉ đó nằm ngoài khoảng địa chỉ xác định bởi 2 thanh ghi thì mắc bẫy chuyển sang monitor mode.

Sử dụng 2 thanh ghi base và limit (tiếp)

- Các lệnh nạp các thanh ghi *base* và *limit* là các lệnh đặc quyền.
- Khi thực hiện trong monitor mode, HĐH không hạn chế truy nhập bộ nhớ.

  ⇒ cho phép HĐH nạp chương trình của người sử dụng vào bộ nhớ, đưa các chương trình đó ra ngoài khi có lỗi.

4.1.2. Liên kết các lệnh và dữ liệu tới bộ nhớ

Sự liên kết địa chỉ của các lệnh và dữ liệu (của tiến trình) tới các địa chỉ bộ nhớ có thể xảy ra 3 giai đoạn khác nhau:

- **Compile time**: nếu biết trước địa chỉ bộ nhớ của chương trình thì có thể kết gán địa chỉ tuyệt đối (*absolute* address) lúc biên dịch.

+ Ví dụ: chương trình .COM của MS-DOS, phát biểu assembly: **org xxx**

+ Khuyết điểm: phải biên dịch lại nếu thay đổi địa chỉ nạp chương trình

- **Load time**: tại thời điểm biên dịch, nếu chưa biết quá trình sẽ nằm ở đâu trong bộ nhớ thì compiler phải sinh **địa chỉ có thể tái định vị** (*relocatable* address). Vào thời điểm loading, loader phải chuyển đổi địa chỉ có thể tái định vị thành địa chỉ tuyệt đối dựa trên một địa chỉ nền (base address).

+ Địa chỉ tuyệt đối được tính toán vào thời điểm nạp chương trình → phải tiến hành reload nếu địa chỉ nền thay đổi.

4.1.2. Liên kết các lệnh và dữ liệu tới bộ nhớ

- **Execution time**: nếu trong khi thực thi, process được di chuyển từ vùng nhớ này sang vùng nhớ khác thì việc chuyển đổi địa chỉ được trì hoãn đến thời điểm thực thi

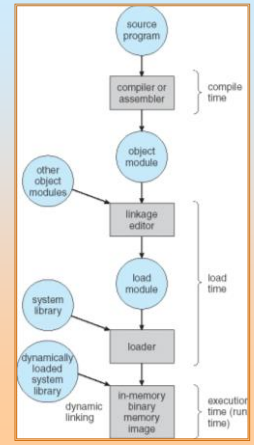
 CPU tạo ra địa chỉ luận lý cho process

 Cần sự hỗ trợ của phần cứng cho việc ánh xạ địa chỉ.

 Ví dụ: dùng thanh ghi base và limit,…

 Sử dụng trong đa số các OS đa dụng (general-purpose) trong đó có các cơ chế swapping, paging, segmentation

Các bước xử lý chương trình người sử dụng



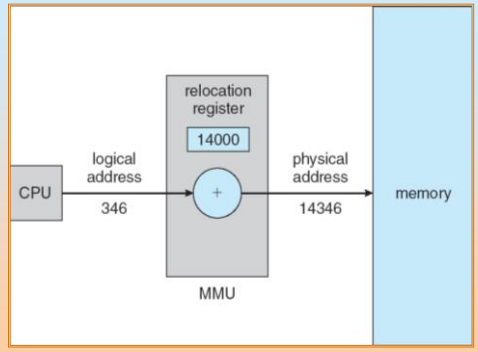
4.1.3. Logical vs. Physical Address Space

- Khái niệm *không gian địa chỉ logic (logical address space)* được tách riêng với *không gian địa chỉ vật lý (physical address space)* để quản lý bộ nhớ thích hợp.
  - *Logical address* – được tạo ra bởi CPU; còn được gọi là địa chỉ ảo (*virtual address)*.
  - *Physical address* – địa chỉ được nhận biết bởi đơn vị quản lý bộ nhớ (Memory Management Unit).
- Các địa chỉ logic (ảo) và vật lý là như nhau trong các giai đoạn liên kết địa chỉ **compile-time** và **load-time**; chúng khác nhau trong **execution-time**.

Memory-Management Unit (MMU)

- Là thiết bị phần cứng ánh xạ địa chỉ ảo tới địa chỉ vật lý.
- Trong lược đồ MMU, giá trị trong thanh ghi định vị (relocation register) được cộng với tất cả địa chỉ được sinh ra bởi tiến trình của người sử dụng tại thời điểm nó được gửi tới bộ nhớ.
- Chương trình của người sử dụng làm việc với các địa chỉ logic; nó không bao giờ nhận ra các địa chỉ vật lý thực.

Định vị động sử dụng thanh ghi định vị



4.1.4. Dynamic Loading

- Tiến trình chỉ được nạp vào bộ nhớ khi nó được gọi.
- Sử dụng không gian bộ nhớ tốt hơn; tiến trình không dùng đến thì không bao giờ được nạp.
- Hữu ích trong trường hợp số lượng lớn mã cần xử lý nhưng hiếm khi xuất hiện.
- Không yêu cầu sự hỗ trợ đặc biệt từ HĐH, được thực hiện thông qua thiết kế chương trình.

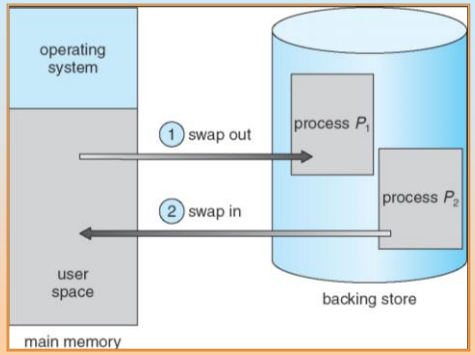
4.1.5. Dynamic Linking

- Việc liên kết hoãn lại đến execution time.
- Đoạn mã nhỏ, *stub*, được sử dụng để định vị thường trình thư viện cư trú trong bộ nhớ (memory-resident library routine) thích hợp, hoặc để nạp thư viện nếu thường trình hiện tại chưa sẵn sàng.
- Khi được thực hiện, stub kiểm tra thường trình cần đến có trong bộ nhớ của tiến trình chưa,
  - **nếu chưa** thì chương trình nạp thường trình vào bộ nhớ;
  - **nếu rồi**: stub tự thay thế chính nó bởi địa chỉ của thường trình rồi thực hiện thường trình đó.
- Liên kết động đặc biệt hữu dụng đối với các thư viện chương trình, nhất là trong việc cập nhật thư viện (vd sửa lỗi)

4.2. Swapping

- Một tiến trình có thể được tạm thời đưa ra khỏi bộ nhớ tới *backing store* (swap out), và sau đó được đưa trở lại bộ nhớ để thực hiện tiếp (swap in).
- Backing store – đĩa tốc độ nhanh, đủ lớn để lưu trữ bản sao của tất cả hình ảnh bộ nhớ cho tất cả người sử dụng; phải cung cấp sự truy nhập trực tiếp tới các hình ảnh bộ nhớ này.
- Hệ thống duy trì 1 hang đợi (ready queue) chứa các tiến trình sẵn sàng chạy có ảnh bộ nhớ được chứa trên vùng lưu trữ (backing store) hoặc trong bộ nhớ.
- Khi trình lập lịch CPU quyết định thực hiện 1 tiến trình, nó gọi trình điều vận.
- Trình điều vận kiểm tra tiến trình tiếp theo trong queue có trong bộ nhớ chưa, nếu chưa có và không còn vùng nhớ rỗi đủ lớn, nó đưa 1 tiến trình trong bộ nhớ ra backing store và đưa tiến trình mong muốn vào.

Giản đồ Swapping



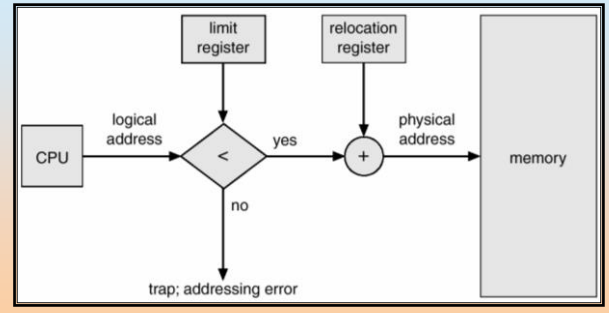
Swapping (tiếp)

- Phần lớn thời gian hoán đổi là thời gian chuyển dữ liệu; tổng thời gian chuyển tỷ lệ thuận với dung lượng bộ nhớ hoán đổi.
  - Vd: giả sử dung lượng tiến trình người sử dụng là 10 MB
    backing store là đĩa cứng có tốc độ truyền dữ liệu là 40 MB/s
    Thời gian truyền 1 chiều từ/đến bộ nhớ = 1/4 s = 250 ms
    giả sử trễ trung bình = 8 ms ⇒ Tổng truyền 1 chiều = 258 ms
    ⇒ Tổng swap in + swap out = 258 x 2 = 516 ms.
- Roll out, roll in – biến thể hoán đổi được sử dụng cho thuật giải lập lịch dựa trên mức ưu tiên (priority-based scheduling); tiến trình có mức ưu tiên thấp hơn bị thay ra để tiến trình có mức ưu tiên cao hơn có thể được nạp và thực hiện.
- Sự hoán đổi khác nhau ở các HĐH UNIX, Linux, Windows.

4.3. Phân phối bộ nhớ liên tiếp

- Bộ nhớ chính được chia thành 2 phần:
  - Nơi HĐH cư trú, thường ở vùng nhớ thấp, chứa bảng vector ngắt.
  - Các tiến trình của người sử dụng được chứa trong vùng nhớ cao.
- Phân phối đơn partition (Single-partition allocation)
  - Các thanh ghi định vị được sử dụng để bảo vệ các tiến trình của người sử dụng không ảnh hưởng lẫn nhau và không thay đổi dữ liệu và mã HĐH.
  - Relocation register chứa giá trị địa chỉ vật lý nhỏ nhất; limit register chứa dải các địa chỉ logic - mỗi địa chỉ logic phải nhỏ hơn limit register.
  - MMU (Memory Management Unit) ánh xạ địa chỉ logic *theo cách động*.

Ví dụ các thanh ghi Relocation và Limit

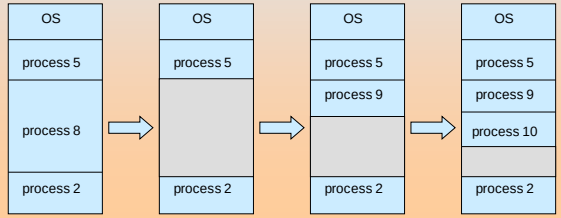


Phân phối liên tiếp (tiếp)

- Phân phối đa partition (Multiple-partition allocation)
  - *Hole* – khối bộ nhớ khả dụng; các hole có kích thước khác nhau nằm rải rác khắp bộ nhớ.
  - Khi một tiến trình đến, nó được phân phối cho một hole đủ lớn.
  - HĐH duy trì thông tin về:

    a) các vùng nhớ đã được phân phối - allocated partitions

    b) các vùng nhớ còn tự do - free partitions (hole)



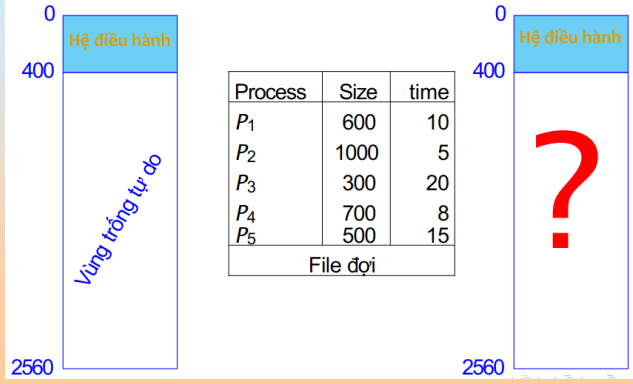
Bài toán phân phối vùng nhớ động

Nguyên tắc: Chỉ có một danh sách quản lý bộ nhớ tự do

- Thời điểm ban đầu toàn bộ bộ nhớ là tự do với các tiến trình ⇒ vùng trống lớn nhất (hole)
- Khi một tiến trình yêu cầu bộ nhớ
  + Tìm trong DS vùng trống một phần tử đủ lớn cho yêu cầu
  + Nếu tìm thấy:
    - Vùng trống được chia thành 2 phần
    - Một phần cung cấp theo yêu cầu
    - Một phần trả lại danh sách vùng trống tự do
  + Nếu không tìm thấy:
    - Phải chờ tới khi có được một vùng trống thỏa mãn
    - Cho phép tiến trình khác trong hàng đợi thực hiện (nếu độ ưu tiên đảm bảo)
- Khi một tiến trình kết thúc
  + Vùng nhớ chiếm được trả về DS quản lý vùng trống tự do
  + Kết hợp với các vùng trống khác liên kề nếu cần thiết

Bài toán phân phối vùng nhớ động

Bộ nhớ chính:



Process	Size	time
P_1	600	10
P_2	1000	5
P_3	300	20
P_4	700	8
P_5	500	15

File đợi

Bài toán phân phối vùng nhớ động

Chiến lược lựa chọn vùng bộ nhớ trống (free memory)

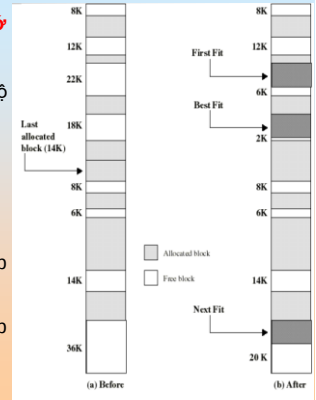
- Dùng để quyết định cấp phát khối bộ nhớ trống nào cho một process
- Mục tiêu: giảm chi phí compaction
- Các chiến lược chọn vị trí (placement):

  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

Bài toán phân phối vùng nhớ động

Chiến lược lựa chọn vùng trống tự do

Bài toán:

- Giả sử các vùng bộ nhớ còn trống có kích thước 100K, 500K, 200K, 300K và 600K (theo thứ tự),
- **First-fit, Best-fit, and Worst-fit**

  Sẽ nạp các tiến trình 212K, 417K, 112K, and 426K như thế nào ?

Bài toán phân phối vùng nhớ động

Buddy Allocation: Cung cấp nhớ

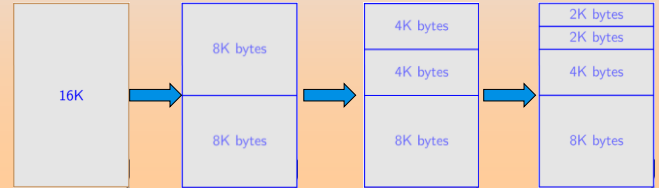
Nguyên tắc: Chia đôi liên tiếp vùng trống tự do cho tới khi thu được vùng trống nhỏ nhất thỏa mãn.

Cung cấp cho yêu cầu *n* bytes:

+ Chia vùng trống tìm được thành 2 khối bằng nhau (gọi là buddies)

+ Tiếp tục chia vùng trống phía trên thành 2 phần cho tới khi đạt vùng trống nhỏ nhất kích thước lớn hơn ***n***

Ví dụ 1: Vùng trống 16KB + Yêu cầu 735 Bytes



Bài toán phân phối vùng nhớ động

Buddy Allocation: Cung cấp nhớ

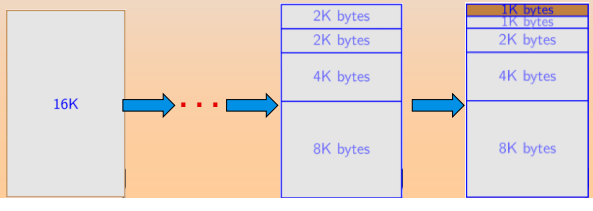
Nguyên tắc: Chia đôi liên tiếp vùng trống tự do cho tới khi thu được vùng trống nhỏ nhất thỏa mãn.

Cung cấp cho yêu cầu *n* bytes:

+ Chia vùng trống tìm được thành 2 khối bằng nhau (gọi là buddies)

+ Tiếp tục chia vùng trống phía trên thành 2 phần cho tới khi đạt vùng trống nhỏ nhất kích thước lớn hơn ***n***

Ví dụ 1: Vùng trống 16KB + Yêu cầu 735 Bytes



Bài toán phân phối vùng nhớ động

Buddy Allocation: Cung cấp nhớ

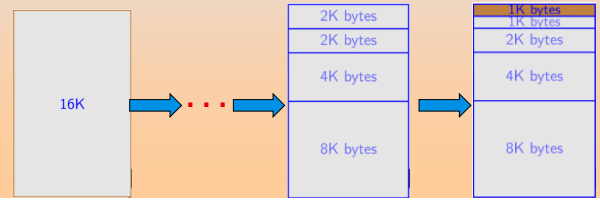
Hệ thống duy trì các danh sách vùng trống kích thước 1, 2, . . . , 2^n bytes

+ Với yêu cầu K, tìm phần tử nhỏ nhất có kích thước lớn hơn K

+ Nếu phần tử nhỏ nhất lớn hơn 2K, chia liên tiếp tới khi được vùng nhỏ nhất có kích thước lớn hơn K

+ Nhận xét:

Với bộ nhớ kích thước n, cần duyệt $log_2 n$ danh sách ⇒ Nhanh

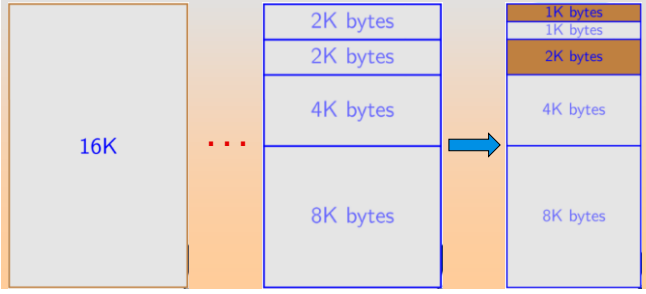


Bài toán phân phối vùng nhớ động

Buddy Allocation: Cung cấp nhớ

Ví dụ 2: Vùng trống 16KB

+ Yêu cầu 735 Bytes
+ Yêu cầu 1205 bytes

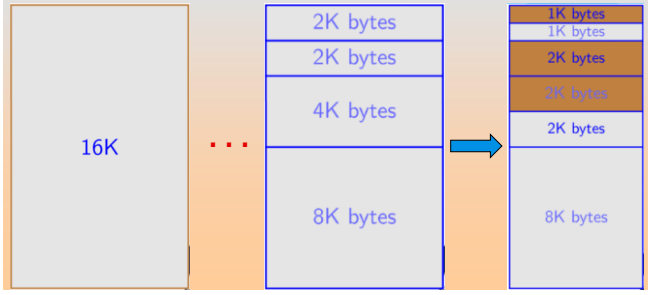


Bài toán phân phối vùng nhớ động

Buddy Allocation: Cung cấp nhớ

Ví dụ 3: Vùng trống 16KB

+ Yêu cầu 735 Bytes
+ Yêu cầu 1205 bytes + Yêu cầu 2010 bytes

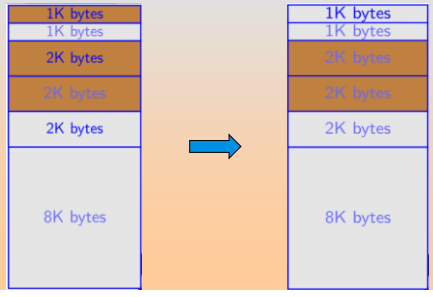


Bài toán phân phối vùng nhớ động

Buddy Allocation: Thu hồi vùng nhớ

+ Có thể kết hợp 2 vùng kề nhau có cùng kích thước
+ Tiếp tục kết hợp liên tiếp cho tới khi tạo ra vùng trống lớn nhất có thể

Ví dụ: Giải phóng vùng nhớ thứ nhất (1K)



Bài toán phân phối vùng nhớ động

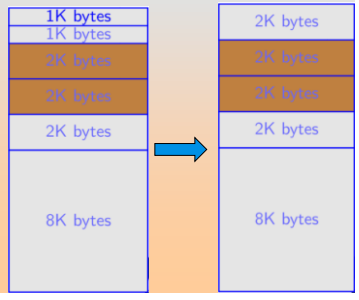
Buddy Allocation: Thu hồi vùng nhớ

+ Có thể kết hợp 2 vùng kề nhau có cùng kích thước
+ Tiếp tục kết hợp liên tiếp cho tới khi tạo ra vùng trống lớn nhất có thể

Ví dụ:

Giải phóng vùng nhớ thứ nhất (1K)

Kết hợp 2 vùng 1K thành vùng 2K



Bài toán phân phối vùng nhớ động

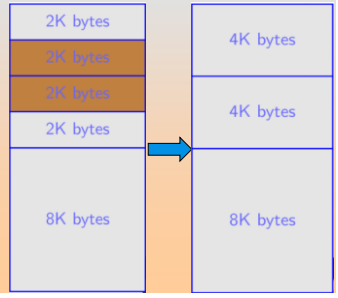
Buddy Allocation: Thu hồi vùng nhớ

+ Có thể kết hợp 2 vùng kề nhau có cùng kích thước

+ Tiếp tục kết hợp liên tiếp cho tới khi tạo ra vùng trống lớn nhất có thể

Ví dụ:

+ Giải phóng vùng nhớ thứ nhất (1K)
  - Kết hợp 2 vùng 1K thành vùng 2K
+ Giải phóng vùng nhớ thứ hai (2K)
  - Kết hợp 2 vùng 2K thành vùng 4K
+ Giải phóng vùng nhớ thứ ba (2K)
  - Kết hợp 2 vùng 2K thành vùng 4K



Bài toán phân phối vùng nhớ động

Vấn đề bố trí lại bộ nhớ

Sau một thời gian hoạt động, các vùng trống nằm rải rác khắp nơi gây ra hiện tượng thiếu bộ nhớ. ⇒ Cần phải bố trí lại bộ nhớ

+ **Dịch chuyển các tiến trình**
  - Vấn đề không đơn giản vì các đối tượng bên trong khi chuyển sang vị trí mới sẽ mang địa chỉ khác đi
  - Sử dụng thanh ghi dịch chuyển (relocation register) chứa giá trị bằng độ dịch chuyển của tiến trình
  - Vấn đề lựa chọn phương pháp để chi phí nhỏ nhất
    - Dịch chuyển tất cả về một phía ⇒ vùng trống lớn nhất
    - Dịch chuyển để tạo ra ngay lập tức một vùng trống vừa vặn
+ **Phương pháp tráo đổi (swapping)**
  - Lựa chọn thời điểm dừng tiến trình đang thực hiện
  - Đưa tiến trình và trạng thái tương ứng ra bên ngoài
    - Giải phóng vùng nhớ để kết hợp với các phần tử liền kề
  - Tái định vị vào vị trí cũ và khôi phục trạng thái cũ
    - Dùng thanh ghi dịch chuyển nếu đưa vào vị trí khác

Sự phân mảnh - Fragmentation

a. External Fragmentation (Phân mảnh ngoại):

+ Kích thước không gian nhớ còn trống tuy đủ lớn để thỏa mãn một yêu cầu cấp phát, nhưng không gian nhớ này lại không liên tục.

+ Có thể dùng kết khối (compacting) để gom lại thành vùng nhớ liên tục.

b. Internal Fragmentation (Phân mảnh nội):

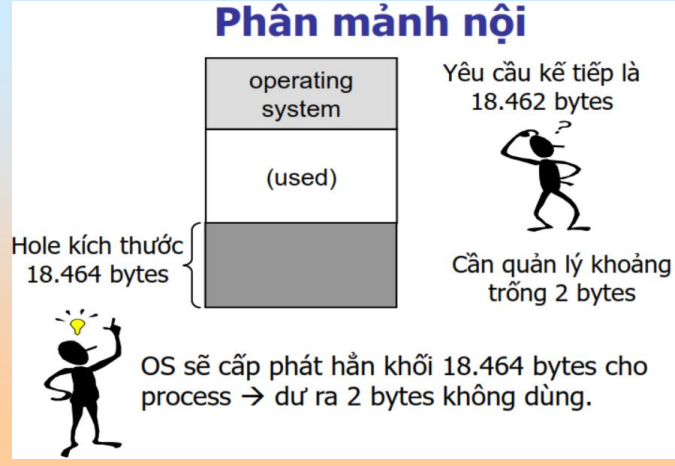
+ Kích thước vùng nhớ được cấp phát lớn hơn vùng nhớ yêu cầu.

- VD: cấp một khoảng trống 18.464 byte cho một process yêu cầu 18.462 byte.

- Thường xảy ra khi bộ nhớ thực được chia thành các khối kích thước cố định (fixed-sized block) và các process được cấp phát theo đơn vị khối. Có thể làm giảm external fragmentation bằng cách nén lại (compaction)
  - Di chuyển các nội dung bộ nhớ để đặt tất cả các vùng nhớ tự do lại với nhau thành một khối lớn.
  - Kết khối chỉ có thể tiến hành nếu sự tái định vị là động, và nó được thực hiện trong thời gian thực thi (execution time).

■ 1 phương pháp khác là phân phối bộ nhớ không liên tục.

Sự phân mảnh - Fragmentation



4.4. Phân trang - Paging

- Không gian địa chỉ logic của một tiến trình có thể không kề nhau; tiến trình được phân phối bộ nhớ vật lý bất kỳ lúc nào sau đó khi bộ nhớ sẵn có.
- Chia bộ nhớ vật lý thành những khối có kích thước cố định là bội số của 2 (512 bytes - 16 MB),được gọi là các **frame.**
- Chia bộ nhớ logic thành các khối cùng kích thước - các **page**.
- Luôn theo dõi tất cả các frame còn trống.
- Để chạy một chương trình có kích thước *n* pages, cần phải tìm *n* frames còn trống và nạp chương trình.
- Thiết lập một bảng phân trang (page table) để biên dịch (translate) các địa chỉ logic thành địa chỉ vật lý.
- Sự phân đoạn diễn ra bên trong (Internal fragmentation).

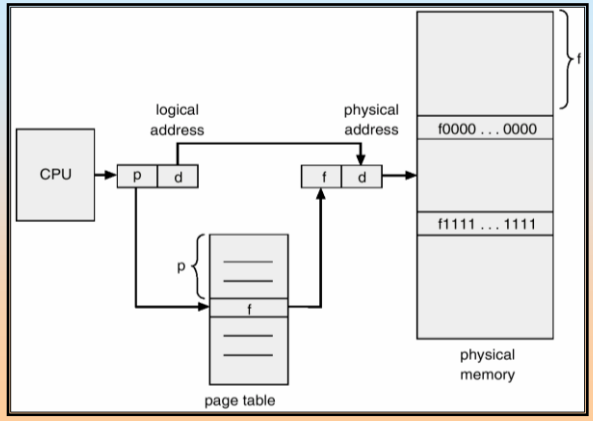
Lược đồ biên dịch địa chỉ

- Địa chỉ được tạo ra bởi CPU được chia thành:
  - *Page number (p)* – được sử dụng làm index tới *page table*, chứa địa chỉ cơ sở (base address) của mỗi trang trong bộ nhớ vật lý.
  - *Page offset (d)* – kết hợp với địa chỉ cơ sở để xác định địa chỉ bộ nhớ vật lý.

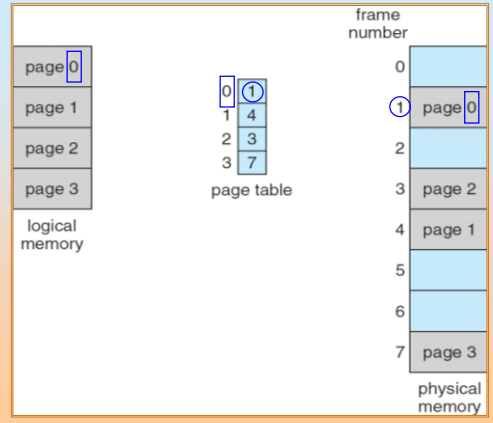
page number	page offset
p	*d*
m - n	*n*

  - 2^n = kích thước của 1 page (byte hoặc word)
  - 2^m = kích thước không gian địa chỉ logic

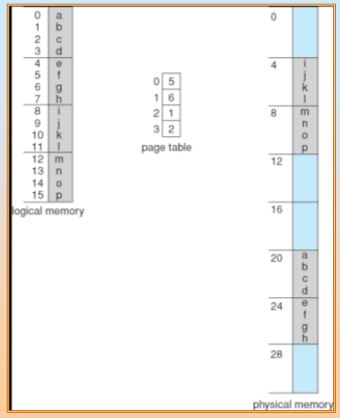
Lược đồ dịch địa chỉ (tiếp)



Mô hình phân trang



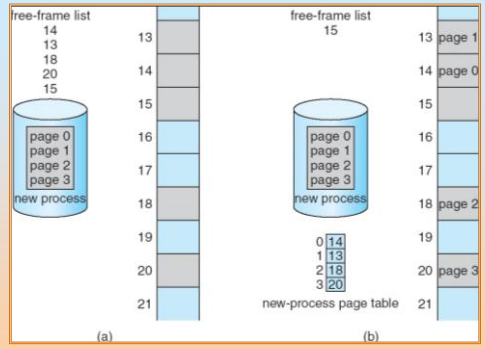
Ví dụ phân trang



32-byte memory and 4-byte pages

Vd: Địa chỉ logic 4 (page 1, offset 0); theo bảng phân trang, trang 1 được ánh xạ tới frame 6 ⇒ địa chỉ logic ánh xạ tới địa chỉ vật lý = 6 x 4 + 0 = 24

Các Frame rỗi



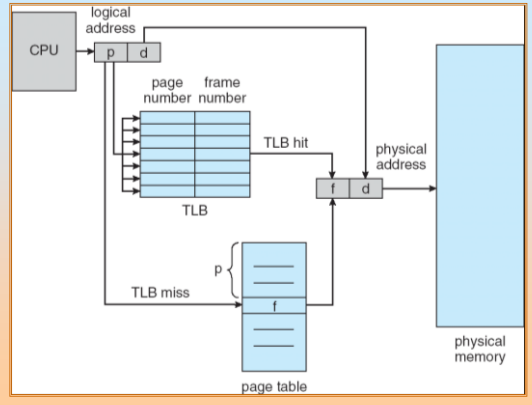
Trước khi phân phối

Sau khi phân phối

Sự thực thi của Page Table

- Page table được lưu trong bộ nhớ chính.
- *Page-table base register (*PTBR) chỉ tới page table.
- *Page-table length register* (PRLR) cho biết kích thước của page table.
- Trong lược đồ phân trang, mọi sự truy nhập lệnh/dữ liệu yêu cầu 2 lần truy nhập bộ nhớ: một cho page table và một cho lệnh/dữ liệu → truy nhập chậm hơn.
- Vấn đề 2 lần truy nhập bộ nhớ có thể được giải quyết bằng cách sử dụng một bộ nhớ cache tra cứu nhanh đặc biệt gọi là *bộ nhớ liên kết - associative memory (TLB).*

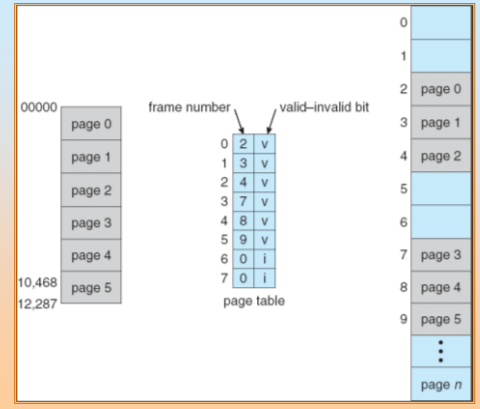
Phân trang với TLB



Bảo vệ bộ nhớ

- Sự bảo vệ bộ nhớ được thực thi bằng cách kết hợp protection bit với mỗi frame
- Mỗi phần tử trong bảng phân trang được gắn thêm một **valid–invalid bit**
  - "valid" (=1) cho biết trang tương ứng ở trong không gian địa chỉ logic của tiến trình, và do đó là một trang hợp lệ.
  - "invalid" (=0) cho biết rằng trang không có trong không gian địa chỉ logic.

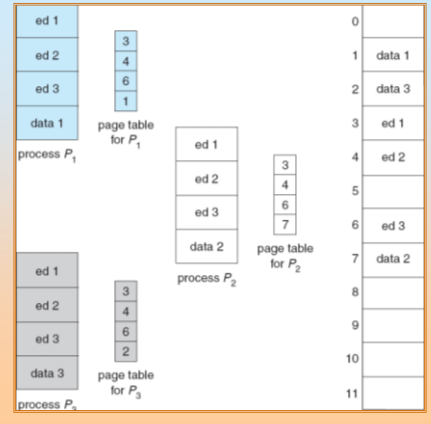
Valid (v) - Invalid (i) Bit trong bảng phân trang



Các trang chia sẻ

- Một lợi điểm của phân trang là khả năng chia sẻ mã chung, đặc biệt quan trọng trong môi trường chia sẻ thời gian.
- **Shared code**
  - Một bản copy của read-only code được chia sẻ giữa các tiến trình (i.e., text editors, compilers, window systems).
  - Mã chia sẻ phải xuất hiện tại cùng vị trí trong không gian địa chỉ logic của tất cả các tiến trình.
- **Private code and data**
  - Mỗi tiến trình giữ một bản copy mã và dữ liệu riêng.
  - Các trang của mã và dữ liệu riêng có thể xuất hiện tại bất kỳ đâu trong không gian địa chỉ logic.

Shared Pages Example



Vd: hệ thống hỗ trợ 40 user, mỗi user thực hiện một text editor gồm 150KB mã và 50KB dữ liệu.

- Nếu không chia sẻ dữ liệu, cần 8000KB.

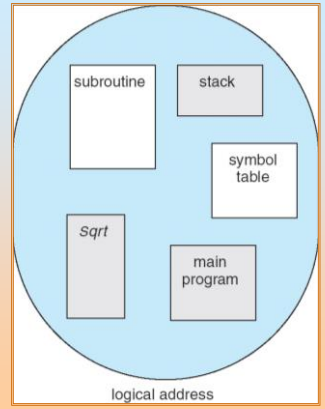
- Nếu chia sẻ 150KB mã, chỉ tốn 150 + 40 x 50 = 2150 KB bộ nhớ.

4.5. Phân đoạn - Segmentation

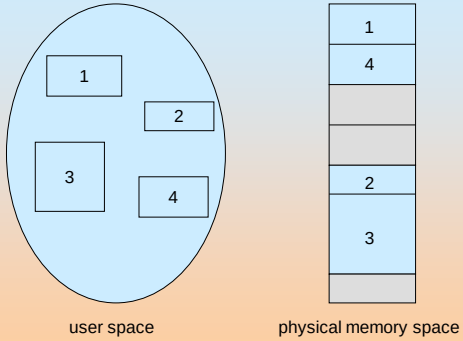
- Là lược đồ quản lý bộ nhớ hỗ trợ cách nhìn của người sử dụng đối với bộ nhớ:
- Mỗi chương trình là một tập hợp các đoạn. Mỗi đoạn là một đơn vị logic như:

  main program,
  procedure,
  function,
  method,
  object,
  local variables, global variables,
  common block,
  stack,
  symbol table, arrays

Chương trình dưới góc nhìn của người sử dụng



Góc nhìn logic của sự phân đoạn



Kiến trúc phân đoạn

- Địa chỉ logic gồm 2 thành phần:

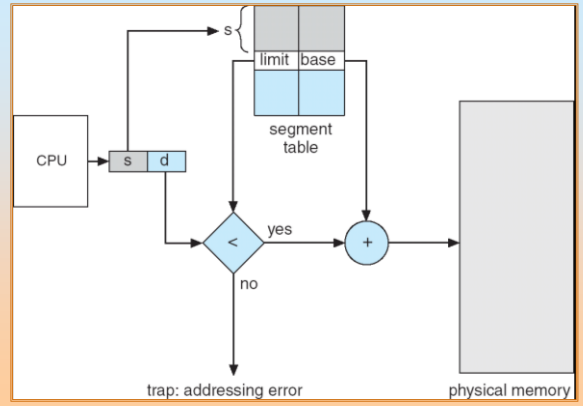
  <segment-number, offset>,
- *Segment table* – tương tự bảng phân trang, mỗi mục của bảng gồm có:
  - **base** – chứa địa chỉ vật lý đầu tiên của đoạn trong bộ nhớ.
  - **limit** – xác định độ dài của đoạn.
- *Segment-table base register (STBR):* trỏ tới vị trí của bảng phân đoạn trong bộ nhớ.
- *Segment-table length register (STLR)*: xác định số đoạn mà một chương trình sử dụng;

  segment number *s* là hợp lệ nếu *s* < STLR.

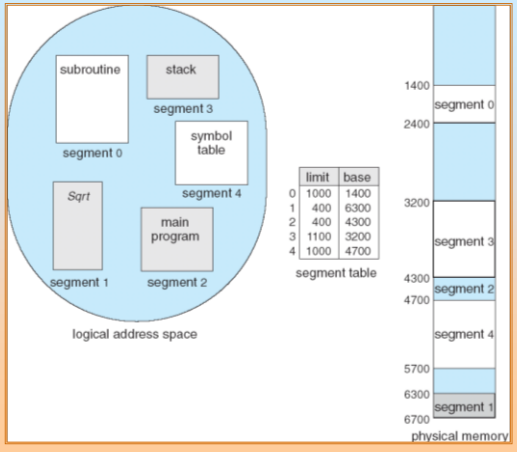
Kiến trúc phân đoạn (tiếp)

- Phân đoạn
  - các đoạn có kích thước khác nhau (khác với phân trang)
- Định vị
  - động
  - được thực hiện bởi bảng phân đoạn
- Phân phối bộ nhớ
  - giải quyết bài toán phân phối bộ nhớ động
  - first fit/best fit
  - có sự phân mảnh ngoài

Lược đồ phân đoạn



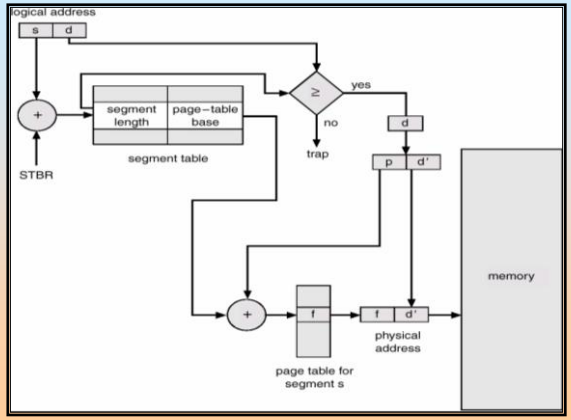
Ví dụ phân đoạn



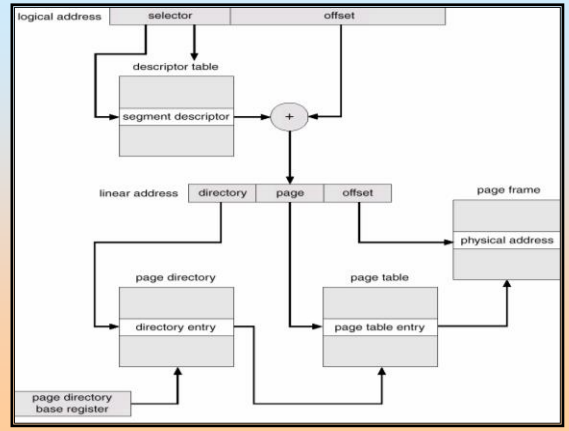
4.6. Kết hợp phân đoạn với phân trang MULTICS

- Bộ nhớ được phân thành các đoạn, sau đó mỗi đoạn lại được phân trang.
- Hệ thống MULTICS giải quyết được vấn đề phân mảnh ngoài và thời gian tìm kiếm dài.
- Giải pháp khác so với phân đoạn ở chỗ mỗi mục của bảng phân đoạn không chứa địa chỉ cơ sở của đoạn mà chứa địa chỉ cơ sở của bảng phân trang của đoạn đó.

Lược đồ MULTICS



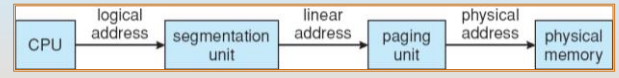
Lược đồ MULTICS của Intel 80386



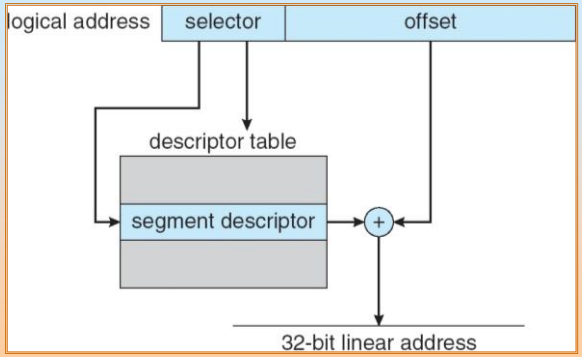
4.7. Example: The Intel Pentium

- Hỗ trợ cả 2 cách quản lý bộ nhớ: phân đoạn, phân đoạn kết hợp với phân trang
- CPU sinh ra các địa chỉ logic
  - địa chỉ này được đưa tới segmentation unit (đơn vị phân đoạn)
    - segmentation unit tạo ra 1 địa chỉ tuyến tính ứng với mỗi địa chỉ logic.
  - Địa chỉ tuyến tính được đưa tới paging unit (đơn vị phân trang)
    - paging unit sinh ra địa chỉ vật lý trong bộ nhớ chính
    - segmentation unit và paging unit tạo thành sự tương ứng của MMU.

Sự biên dịch địa chỉ logic thành địa chỉ vật lý trong Pentium



Intel Pentium Segmentation



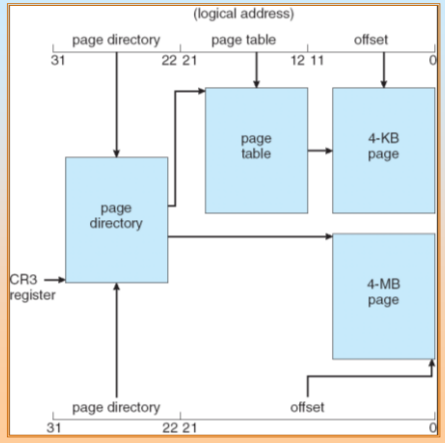
Pentium Paging

- Kích thước trang bằng 4 KB hoặc 4 MB.
- cờ Page Size (trong page directory):
  - = 1: kích thước page frame là 4 MB
    - phân chia địa chỉ tuyến tính 32 bit:

      <page number, page offset> = <10, 22>
  - = 0: kích thước page frame là chuẩn 4 KB
    - Sử dụng lược đồ phân trang 2 mức, phân chia địa chỉ tuyến tính 32-bit như sau:

page number		page offset
p_1	p_2	d
10	10	12

Pentium Paging Architecture

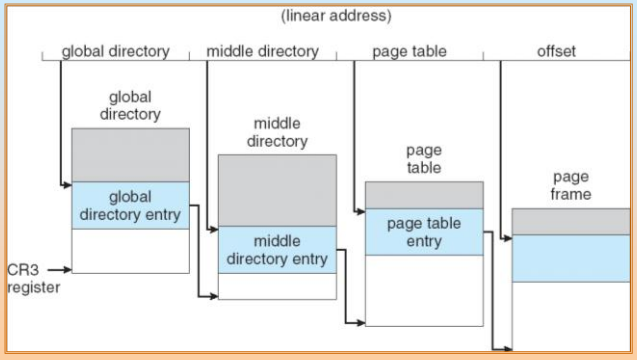


Linear Address in Linux

Được chia thành 4 phần:

global directory	middle directory	page table	offset

Three-level Paging in Linux



BÀI TẬP

1. Các kiểu địa chỉ nhớ ?
2. Nêu cách chuyển đổi địa chỉ vào thời điểm dịch ?
3. Nêu cách chuyển đổi địa chỉ vào thời điểm nạp chương trình ?
4. Nêu cách chuyển đổi địa chỉ vào thời điểm thực thi ?
5. Nêu cơ chế liên kết động ?
6. Nêu cơ chế overlay ?
7. Nêu cơ chế Swapping ?
8. Nêu hiện tượng phân mảnh ngoại và phân mảnh nội ?
9. Nêu cơ chế phân trang ?
10. Nêu cơ chế phân đoạn ?
11. Nêu mô hình kết hợp phân đoạn với phân trang ?

BÀI TẬP

12. Giả sử lúc đầu, bộ nhớ được chia làm 5 vùng nhớ trống có kích thước lần lượt là 100KB, 500KB, 200KB, 300K, 600KB. Nếu các process được kích hoạt chạy lần lượt cần dung lượng bộ nhớ như sau : 212KB, 417KB, 112KB, 426KB, chúng được phân phối như thế nào nếu ta dùng chiến lược **Best-Fit** và **First-Fit** ?

BÀI TẬP

13. Giả sử bộ nhớ chính được phân thành các phân vùng có kích thước là 600K, 500K, 200K, 300K (theo thứ tự), cho biết các tiến trình có kích thước 212K, 417K, 112K và 426K (theo thứ tự) sẽ được cấp phát bộ nhớ như thế nào, nếu sử dụng :

a) Thuật toán First fit

b) Thuật toán Best fit

c) Thuật toán Worst fit

Thuật toán nào cho phép sử dụng bộ nhớ hiệu qủa nhất trong trường hợp trên ?

BÀI TẬP

14. Máy tính sử dụng partition để quản lí bộ nhớ và tại một thời điểm bộ nhớ vật lí của máy đang được sử dụng như sau: 1,5M rỗng, 0.5M P1, 0.5M rỗng, 1.2M P2, 1.6M rỗng, 1.0M P3, 1.2M rỗng, 1.8M P4, 1.7M rỗng, 2.6M P5, 3.5M rỗng.

Các tiến trình mới **xuất hiện** và **kết thúc** theo thứ tự sau: P6: 1.1M, P7: 1.5M, P4 kết thúc, P8: 1.7M, P9: 1.3M, P2 kết thúc, P10: 1.7M, P11: 1,2M, P1 kết thúc, P12: 0.9M, P6 kết thúc.

Hãy giải thích và vẽ sơ đồ bộ nhớ vật lí ứng với các thuật toán cấp phát bộ nhớ chính (**First Fit, Best Fit và Worst Fit**) sau khi tiến trình P6 kết thúc: **First Fit, Best Fit, Worst Fit**

BÀI TẬP

15. Máy tính sử dụng partition để quản lí bộ nhớ, và tại một thời điểm bộ nhớ vật lí của máy đang được sử dụng như sau: 1,1M rỗng, 0.5M P1, 0.8M rỗng, 1.2M P2, 0.6M rỗng, 1.0M P3, 1.2M rỗng, 1.1M P4, 1.1M rỗng, 1.2M P5, 2.5M rỗng.

Các tiến trình mới **xuất hiện** và **kết thúc** theo thứ tự sau: P6: 0.6M, P7: 1.5M, P4 kết thúc, P8: 0.7M, P9: 1.3M, P2 kết thúc, P10: 1.1M, P11: 0,5M, P1 kết thúc, P12: 0.9M, P6 kết thúc.

Hãy giải thích và vẽ sơ đồ bộ nhớ vật lí ứng với các thuật toán cấp phát bộ nhớ chính (**First Fit, Best Fit và Worst Fit**) sau khi tiến trình P6 kết thúc: **First Fit, Best Fit, Worst Fit**