6. Memory Management

- Khái niệm cơ sở
- Các kiểu địa chỉ nhớ
- Chuyển đổi địa chỉ nhớ
- Overlay và swapping
- Mô hình quản lý bộ nhớ đơn giản
  - Fixed partitioning
  - Dynamic partitioning

Khái niệm cơ sở

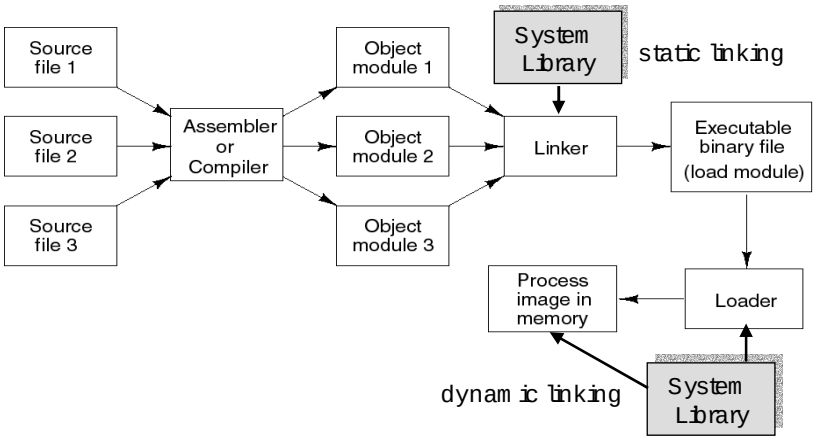
- Quản lý bộ nhớ là công việc của hệ điều hành với sự hỗ trợ của phần cứng nhằm phân phối, sắp xếp các process trong bộ nhớ sao cho hiệu quả.
- Mục tiêu cần đạt được là nạp càng nhiều process vào bộ nhớ càng tốt (*gia tăng mức độ đa chương*)
- Trong hầu hết các hệ thống, kernel sẽ chiếm một phần cố định của bộ nhớ; phần còn lại phân phối cho các process.
- Các yêu cầu đối với việc quản lý bộ nhớ
  - Cấp phát bộ nhớ cho các process
  - Tái định vị (relocation): khi swapping,...
  - Bảo vệ: phải kiểm tra truy xuất bộ nhớ có hợp lệ không?
  - Chia sẻ: cho phép các process chia sẻ vùng nhớ chung
  - Kết gán địa chỉ nhớ luận lý của user vào địa chỉ thực (physical)

Các kiểu địa chỉ nhớ

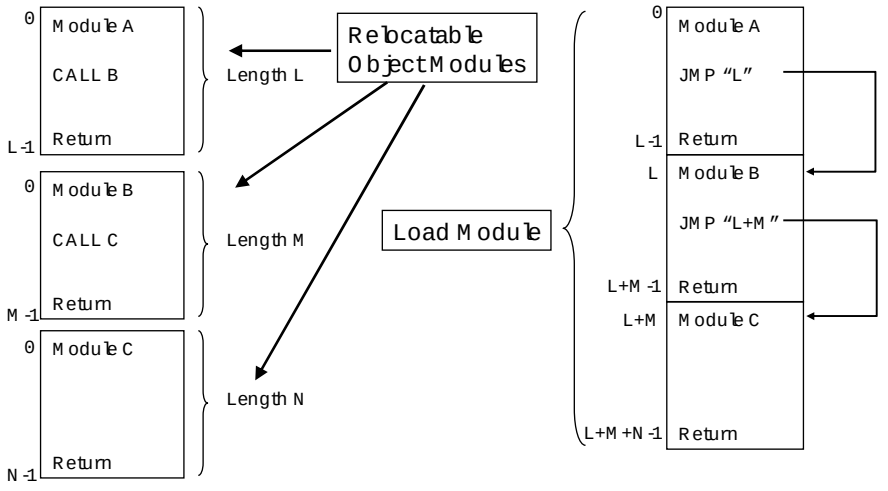
- Địa chỉ vật lý (*physical address*) (địa chỉ thực, địa chỉ tuyệt đối) là một vị trí thực trong bộ nhớ chính.
- Địa chỉ luận lý (*logical address*) là tham chiếu đến một vị trí nhớ độc lập với cấu trúc, tổ chức vật lý của bộ nhớ
  - Các trình biên dịch (compiler) tạo ra mã lệnh chương trình mà trong đó mọi tham chiếu bộ nhớ đều là địa chỉ luận lý
- Địa chỉ tương đối (*relative address*) là một kiểu địa chỉ luận lý trong đó các địa chỉ được biểu diễn tương đối so với một điểm xác định nào đó trong chương trình (ví dụ: 12 byte so với điểm bắt đầu chương trình,...)
- Khi một lệnh được thực thi, các tham chiếu đến địa chỉ luận lý phải được chuyển đổi thành địa chỉ thực. Thao tác chuyển đổi này thường có sự hỗ trợ của phần cứng để đạt hiệu suất cao.

Nạp chương trình vào bộ nhớ(t.t)

- 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

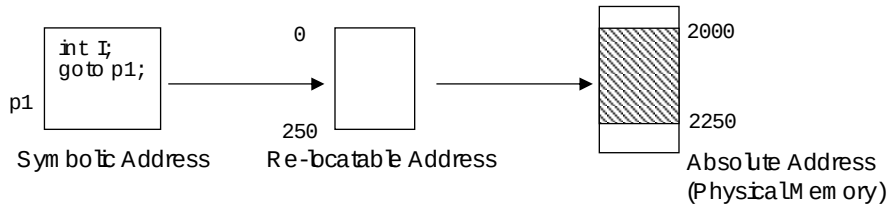


Cơ chế thực hiện linking



Chuyển đổi địa chỉ nhớ

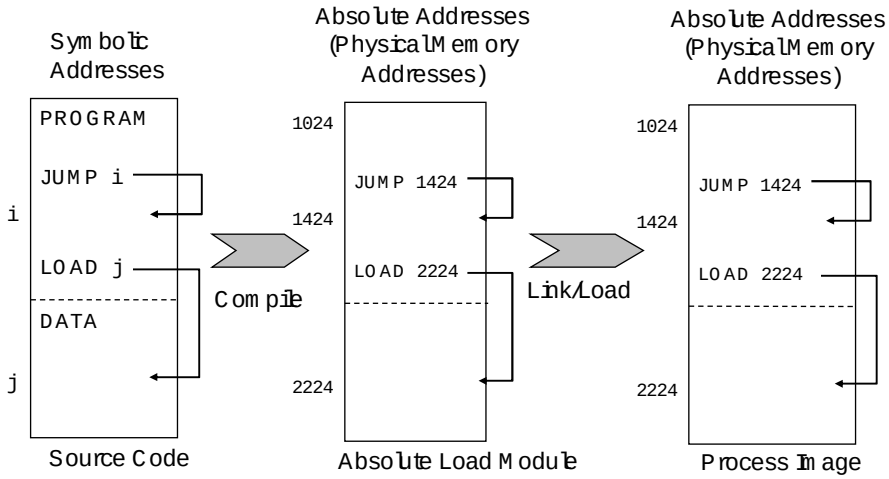
- *Chuyển đổi địa chỉ* – quá trình ánh xạ một địa chỉ từ không gian địa chỉ này sang không gian địa chỉ khác.
- Biểu diễn địa chỉ nhớ
  - Trong *source code*: symbolic (các biến, hằng, pointer,...)
  - Thời điểm *biên dịch* (compile): là địa chỉ khả tái định vị (re-locatable address), hay là địa chỉ tương đối (relative address)
    - Ví dụ: a ở vị trí 14 bytes so với phần header của module.
  - Thời điểm *linking/loading*: là địa chỉ tuyệt đối. Ví dụ: dữ liệu nằm tại địa chỉ bộ nhớ thực: 2030



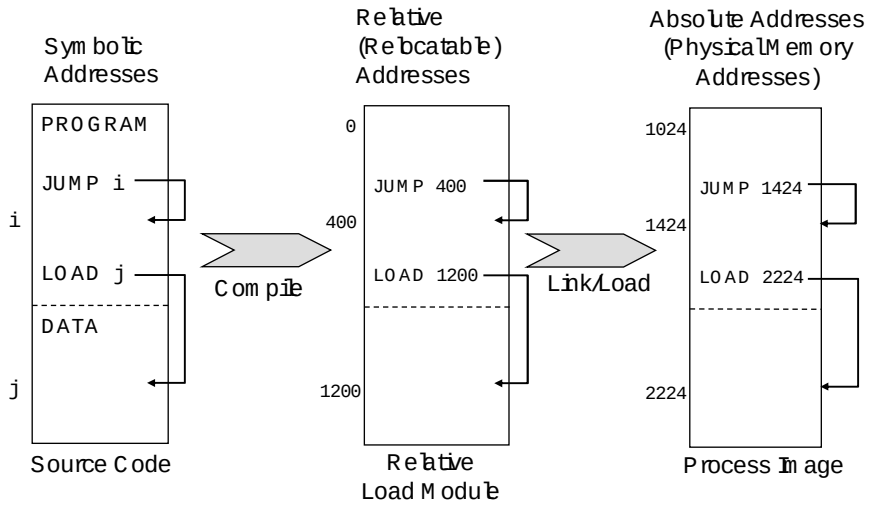
Chuyển đổi địa chỉ (t.t)

- Địa chỉ lệnh (instruction) và dữ liệu (data) được chuyển đổi thành địa chỉ vật lý của bộ nhớ thực có thể xảy ra tại ba thời điểm khác nhau
  - ***Compile time***: nếu biết trước địa chỉ bộ nhớ thì có thể kết gán địa chỉ vật lý (địa chỉ thực) lúc biên dịch.
    - ⇒ Ví dụ: chương trình .COM của MS-DOS, phát biểu assemply ***org xxx***
    - ⇒ Khuyết điểm: phải biên dịch lại nếu thay đổi địa chỉ
  - ***Load time***: tại thời điểm biên dịch, nếu không biết địa chỉ thực thì vào thời điểm loading, phải chuyển đổi địa chỉ khả tái định vị (re-locatable) theo một mốc chuẩn (base address).
    - Địa chỉ thực được tính toán lại vào thời điểm chương trình thực thi ⇒ phải tiến hành reload nếu địa chỉ base thay đổi.

Chuyển đổi vào thời điểm dịch

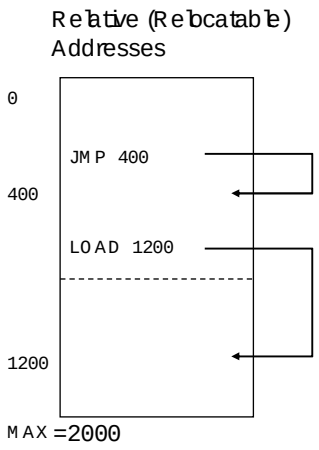


Chuyển đổi vào thời điểm nạp



Chuyển đổi địa chỉ (t.t)

- ***Execution time***: quá trình chuyển đổi được trì hoãn đến thời điểm thực thi (run time)
  - Trong quá trình thực thi, process có thể được di chuyển từ segment này sang segment khác trong bộ nhớ.
  - CPU tạo ra các địa chỉ tương đối cho process
  - Cần sự hỗ trợ của phần cứng cho việc ánh xạ địa chỉ (ví dụ có 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



Dynamic Linking

- Quá trình link một số 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 chỉ chứa các tham chiếu (reference) đến các external module. Các tham chiếu này có thể được chuyển đổi vào hai thời điểm sau:
  - Loading time (load-time dynamic linking)
  - Run time: khi có một lời gọi đến thủ tục được định nghĩa trong external module (run-time dynamic linking)
- OS chịu trách nhiệm tìm các external module và kết nối vào load module (kiểm tra xem external module đã nạp vào bộ nhớ chưa)

Ưu điểm của Dynamic Linking

- Thông thường, *external module* là một thủ tục, 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 process cần dùng *external module* này thì cùng chia sẻ đoạn mã của *external module* $\Rightarrow$ 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 process hay là phần mã của riêng một process (bởi vì chỉ có OS mới có quyền thực hiện việc kiểm tra này).

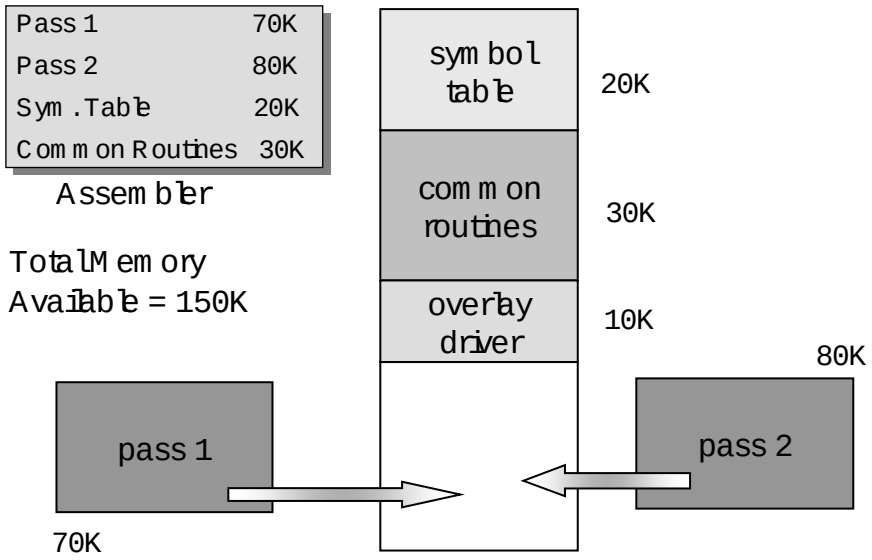
Dynamic Loading

- ***Cơ chế***: chỉ khi nào được gọi đến thì một thủ tục mới được nạp vào bộ nhớ chính ⇒ tăng độ hiệu dụng của bộ nhớ (memory utilization) bởi vì các thủ tục ít được dùng sẽ không bao giờ 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)
- Không cần sự hỗ trợ đặc biệt của 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

Cơ chế Overlay

- Tại mỗi thời điểm, chỉ giữ lại trong bộ nhớ những lệnh hoặc dữ liệu cần thiết, giải phóng các lệnh/dữ liệu chưa hoặc không cần dùng đến.
- Cơ chế này rất hữu dụng khi kích thước một process lớn hơn không gian bộ nhớ cấp cho process đó.
- Cơ chế này được điều khiển bởi người sử dụng (thông qua sự hỗ trợ của các thư viện lập trình) chứ không cần sự hỗ trợ của hệ điều hành

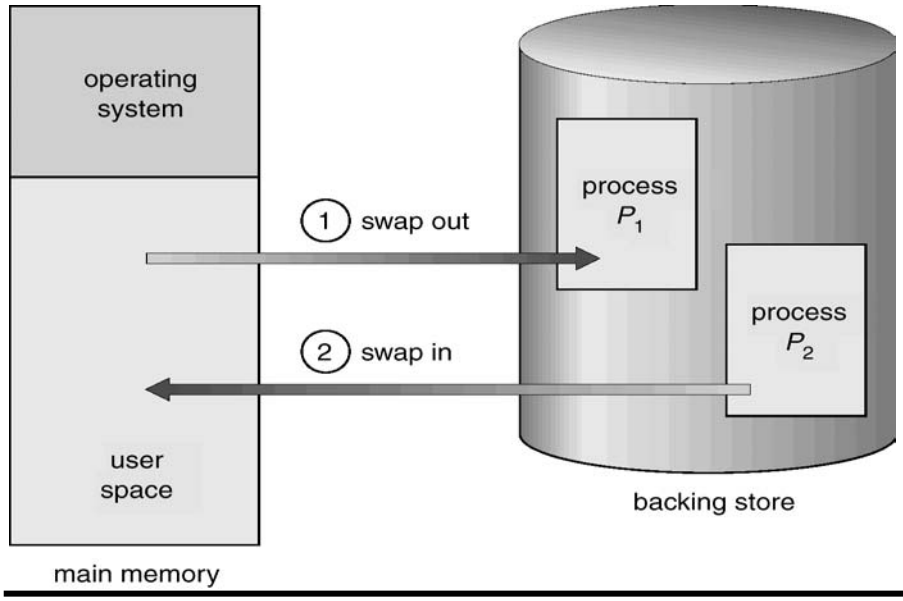
Cơ chế Overlay (t.t)



Cơ chế swapping

- Một process có thể tạm thời bị swap ra khỏi bộ nhớ chính và lưu trên một hệ thống lưu trữ phụ. Sau đó, process có thể được nạp lại vào bộ nhớ để tiếp tục quá trình thực thi
  - Round-robin: swap-out A, swap-in B, thực thi C
  - Roll out, roll in – dùng trong cơ chế định thời theo độ ưu tiên (priority-based scheduling)
    - Process có độ ưu tiên thấp hơn sẽ bị swap-out nhường chỗ cho process có độ ưu tiên cao hơn được nạp vào bộ nhớ để thực thi
  - Medium-term scheduler

Minh họa cơ chế swapping



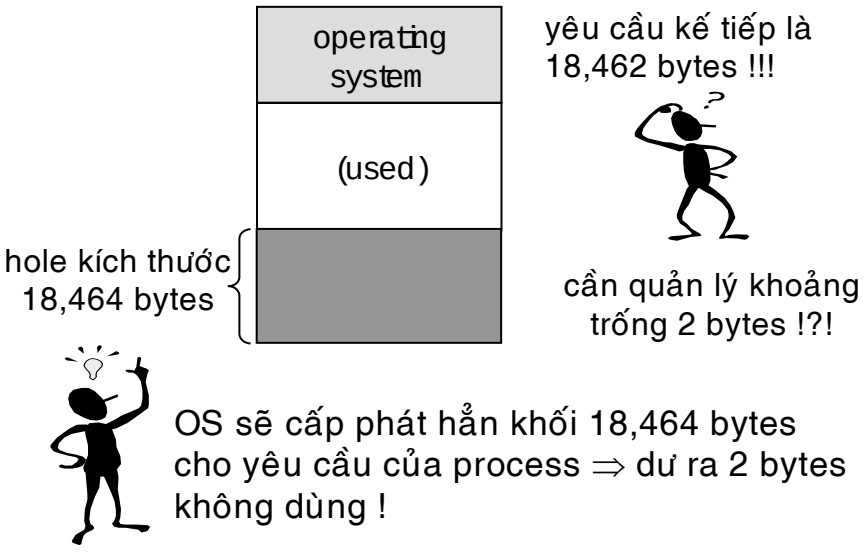
Mô hình quản lý bộ nhớ thực

- Trong chương này, mô hình quản lý bộ nhớ là một mô hình đơn giản, không có bộ nhớ ảo.
- Một process phải được nạp hoàn toàn vào bộ nhớ thì mới được thực thi (ngoại trừ việc sử dụng cơ chế overlay).
- Các cơ chế quản lý bộ nhớ thực sau đây rất ít (hầu như không còn) được dùng trong các hệ thống hiện đại, tuy nhiên đó là các ý tưởng cơ sở cho mô hình quản lý bộ nhớ ảo sau này:
  - Phân chia cố định (fixed partitioning)
  - Phân chia động (dynamic partitioning)
  - Phân trang đơn giản (simple paging)
  - Phân đoạn đơn giản (simple segmentation)

Phân mảnh (fragmentation)

- Phân mảnh ngoại (external fragmentation)
  - Kích thước không gian bộ nhớ còn trống đủ để thỏa mãn một yêu cầu cấp phát, tuy nhiên không gian nhớ này không liên tục ⇒ phải dùng cơ chế kết khối (compaction).
- 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 process yêu cầu 18,462 bytes
  - Hiện tượng phân mảnh nội thường xảy ra khi bộ nhớ thực (physical memory) được chia thành các khối kích thước cố định(fixed-sized block) và các process được cấp phát theo đơn vị khối. Ví dụ: cơ chế phân trang (paging)

Phân mảnh nội

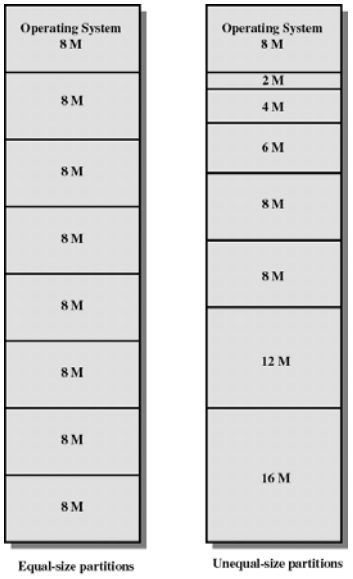


OS sẽ cấp phát hẳn khối 18,464 bytes cho yêu cầu của process ⇒ dư ra 2 bytes không dùng !

Fixed Partitioning

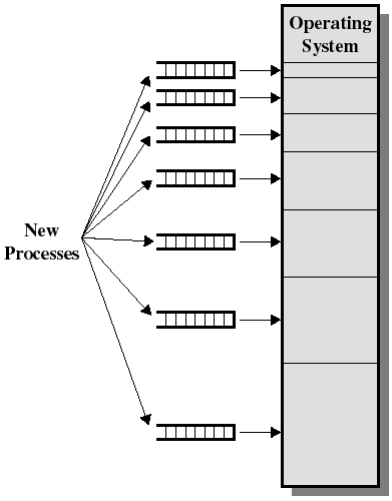
- Chia bộ nhớ chính thành nhiều phần không trùng lấp lên nhau gọi là các *partition* có kích thước bằng nhau hoặc khác nhau
- Process nào có kích thước nhỏ hơn hoặc bằng kích thước partition thì có thể nạp vào partition đó.
- Nếu chương trình có kích thước lớn hơn partition thì phải dùng cơ chế overlay.

- Nhận xét
  - Không hiệu quả do bị phân mảnh nội: một chương trình dù lớn hay nhỏ đều chiếm trọn một partition.



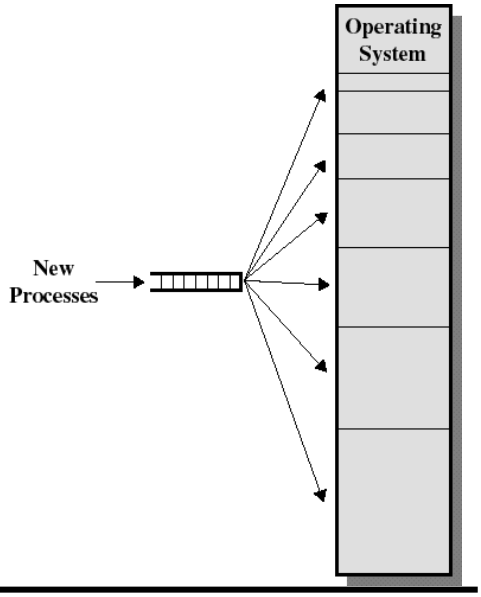
Chiến lược placement

- Partition có kích thước bằng nhau
  - Còn một partition trống ⇒ process mới được nạp vào partition đó
  - Không còn partition trống nhưng trong đó có process đang bị blocked ⇒ swap process đó ra bộ nhớ phụ nhường chỗ cho process mới.
- Partition có kích thước không bằng nhau
  - Gán mỗi process 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 process với kích thước tương ứng) và hàng đợi dày đặc



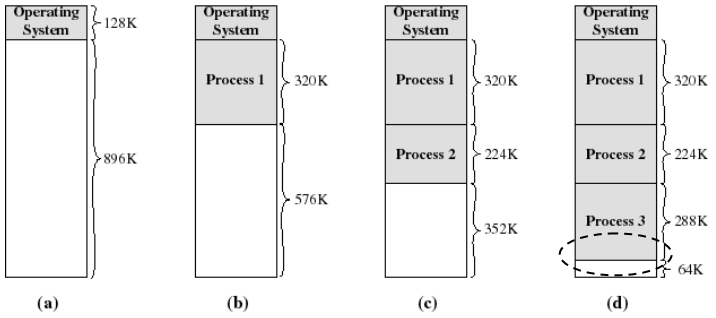
Chiến lược placement (t.t)

- Partition có kích thước không bằng nhau
  - Chỉ có một hàng đợi chung cho các partition
  - Khi cần nạp một process vào bộ nhớ chính ⇒ chọn partition nhỏ nhất còn trống



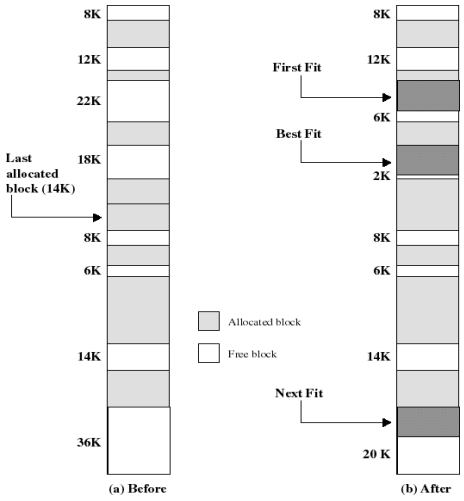
Dynamic Partitioning

- Số lượng partition không cố định và partition có kích thước khác nhau
- Mỗi process được cấp phát chính xác dung lượng bộ nhớ cần thiết
- Gây ra hiện tượng phân mảnh ngoại (*external fragmentation*)



Chiến lược placement

- Dùng để quyết định cấp phát khối bộ nhớ trống nào cho một process
- Mục tiêu: giảm thiểu chi phí compaction (time consuming)
- 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