VIETNAM NATIONAL UNIVERSITY HO CHI MINH CITY
UNIVERSITY OF INFORMATION TECHNOLOGY

IT007.P11.CTTN - HDH

CONTIGUOUS MEMORY ALLOCATION

Nhóm Sinh Viên :
Đỗ Quang Lực
Nguyễn Văn Minh
Phan Nhật Tân
Nguyễn Huy Phước

Giảng viên :
Vũ Đức Lung
Thân Thế Tùng

Ngày 14 tháng 12 năm 2024

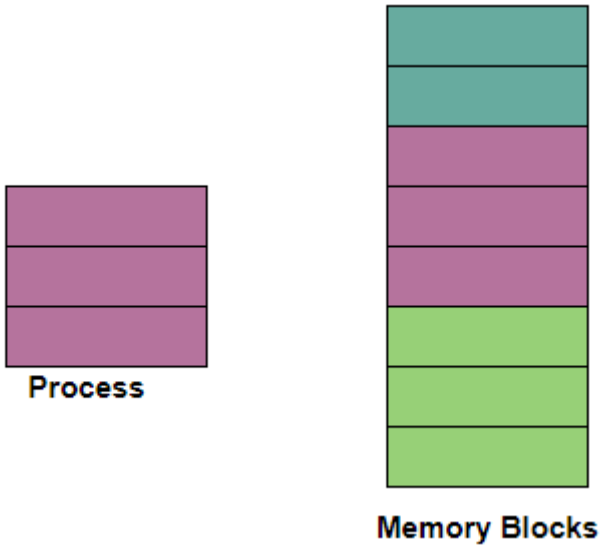
Mục lục

Chương 1

Khái niệm Contiguous Memory Allocation

Contiguous Memory Allocation (Cấp phát bộ nhớ liên tiếp) là một kỹ thuật trong đó hệ điều hành cấp phát một khối bộ nhớ liền kề cho một tiến trình. Bộ nhớ này được phân bổ thành một đoạn duy nhất, liên tục, giúp hệ điều hành dễ dàng quản lý và quá trình truy cập vào bộ nhớ.

Cấp phát bộ nhớ liên tiếp phù hợp với các hệ thống có kích thước bộ nhớ hạn chế và việc truy cập nhanh vào bộ nhớ là rất quan trọng. Cấp phát bộ nhớ liên tiếp đơn giản, dễ quản lý và thường được sử dụng trong các hệ điều hành đơn giản hoặc các hệ thống nhỏ.



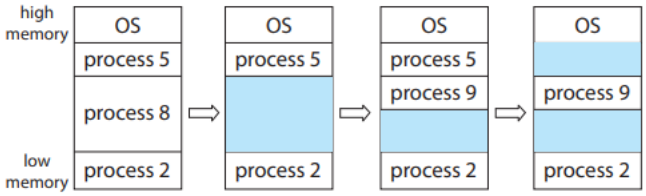
Chương 2

Quản lý phân bổ bộ nhớ:

Phương pháp đơn giản để phân bổ bộ nhớ là gán các tiến trình vào các phân vùng có kích thước thay đổi trong bộ nhớ, mỗi phân vùng chứa chính xác một tiến trình. Trong sơ đồ này, hệ điều hành duy trì một bảng để theo dõi phần nào của bộ nhớ đang trống hoặc đã được sử dụng.

2.1 Quản lý phân vùng

- Ban đầu, toàn bộ bộ nhớ được xem như một khối lớn duy nhất (gọi là "vùng trống").
- Sau khi các tiến trình vào và ra, bộ nhớ sẽ chứa nhiều vùng trống có kích thước khác nhau.
- Ví dụ:



  - Khi tiến trình 8 rời khỏi, sẽ tạo ra một vùng trống liền kề.
  - Sau đó tiến trình được 9 đẩy vào
  - Nếu tiến trình 5 rời đi, sẽ hình thành hai vùng trống không liền kề.

2.2 Cấp phát và giải phóng bộ nhớ

- Khi một tiến trình yêu cầu bộ nhớ, hệ điều hành xem xét yêu cầu và dung lượng bộ nhớ hiện có để phân bổ.
- Sau khi tiến trình hoàn thành, bộ nhớ được giải phóng và có thể được cấp lại cho tiến trình khác.
- Nếu không đủ bộ nhớ để đáp ứng, hệ điều hành có thể:

- Từ chối yêu cầu và hiển thị thông báo lỗi.
- Đưa tiến trình vào hàng đợi chờ (*wait queue*) để xử lý sau.

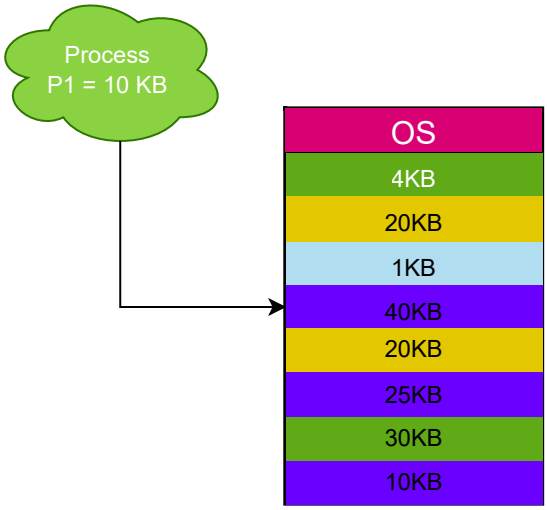
2.3 Quản lý vùng trống

- Bộ nhớ bao gồm nhiều vùng trống có kích thước khác nhau rải rác.
- Khi tiến trình yêu cầu bộ nhớ:
  - Nếu vùng trống quá lớn, nó được chia thành hai phần: một phần cấp phát cho tiến trình, phần còn lại trả về danh sách vùng trống.
  - Nếu các vùng trống liền kề, chúng sẽ được hợp nhất thành một khối lớn hơn.

2.4 Chiến lược phân bổ vùng trống

Các phương pháp phổ biến để chọn vùng trống phù hợp bao gồm:

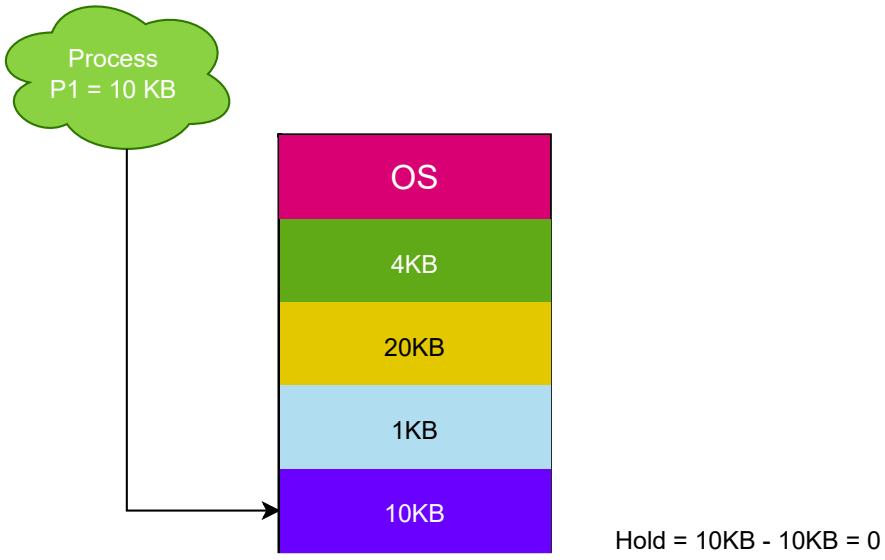
- **First-fit**: Chọn vùng trống đầu tiên đủ lớn.



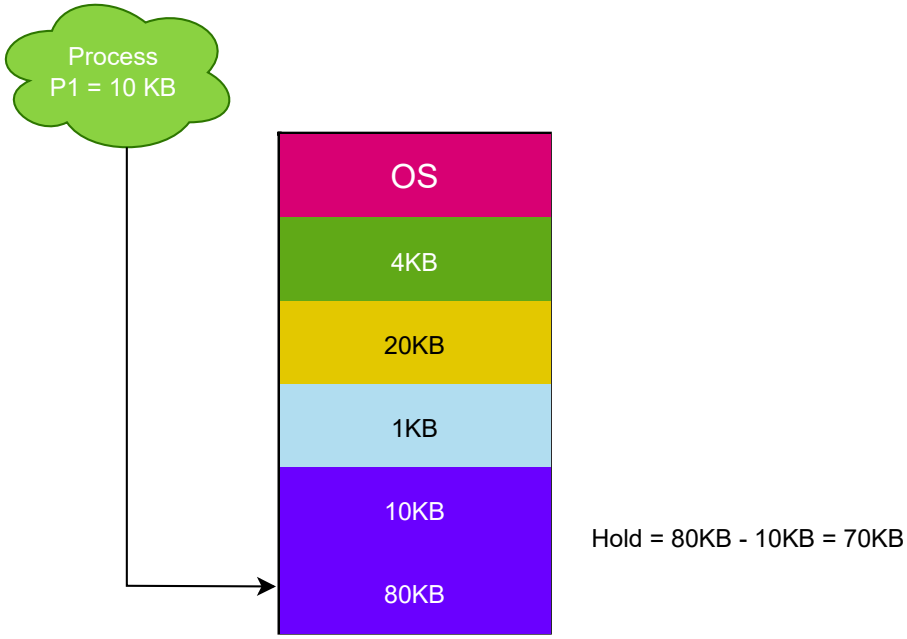
Hình 2.1: First-fit

- **Best-fit**: Chọn vùng trống nhỏ nhất đủ lớn.
- **Worst-fit**: Chọn vùng trống lớn nhất.

Nhìn chung, *first-fit* và *best-fit* thường hiệu quả hơn *worst-fit* về mặt thời gian và sử dụng bộ nhớ.



Hình 2.2: Best-fit



Hình 2.3: Worst-fit

Chương 3

Một số ưu và nhược điểm cấp phát bộ nhớ liên tiếp

3.1 Ưu điểm

- **Đơn giản** Cấp phát bộ nhớ liền kề là một kỹ thuật tương đối đơn giản và dễ hiểu để quản lý bộ nhớ. Kỹ thuật này đòi hỏi ít chi phí hơn và dễ thực hiện.
- **Hiệu quả** Đây là một kỹ thuật hiệu quả để quản lý bộ nhớ. Khi một tiến trình được cấp phát bộ nhớ liền kề, nó có thể truy cập toàn bộ khối bộ nhớ mà không bị gián đoạn.
- **Độ phân mảnh thấp** Vì bộ nhớ được phân bổ thành các khối liền kề, nguy cơ phân mảnh bộ nhớ sẽ thấp hơn. Điều này dẫn đến việc sử dụng bộ nhớ tốt hơn do ít lãng phí tài nguyên.

3.2 Khuyết điểm

- **Tính linh hoạt bị hạn chế** Việc phân bổ bộ nhớ liền kề không linh hoạt vì yêu cầu bộ nhớ phải được phân bổ trong một khối liền kề. Điều này giới hạn số lượng bộ nhớ có thể được cấp phát cho một tiến trình.
- **Lãng phí bộ nhớ** Nếu một tiến trình yêu cầu kích thước bộ nhớ nhỏ hơn khối liền kề được phân bổ cho nó, sẽ dẫn đến tình trạng bộ nhớ không được sử dụng, gây lãng phí.
- **Khó khăn trong việc quản lý kích thước bộ nhớ lớn hơn** Khi kích thước bộ nhớ tăng, việc quản lý phân bổ bộ nhớ liền kề trở nên phức tạp hơn. Tìm kiếm một khối bộ nhớ liền kề đủ lớn để cấp phát cho một tiến trình có thể gặp khó khăn.

3.2.1 Phân mảnh ngoại (*External Fragmentation*)

- Cả chiến lược **first-fit** và **best-fit** đều gặp vấn đề phân mảnh ngoại.
- Khi các tiến trình được nạp và loại bỏ, bộ nhớ trống bị chia thành các vùng nhỏ không liền kề.

- Phân mảnh ngoại xảy ra khi có đủ tổng dung lượng bộ nhớ để đáp ứng yêu cầu, nhưng không có vùng trống liền kề đủ lớn.
- Trong trường hợp xấu nhất, giữa mỗi hai tiến trình có thể có một vùng bộ nhớ trống, dẫn đến lãng phí tài nguyên.
- Theo "50-percent rule": với N khối được cấp phát, trung bình sẽ có $0.5N$ khối bộ nhớ bị mất do phân mảnh, tức là 1/3 bộ nhớ có thể không sử dụng được.

Chương 4

Minh họa sản phẩm

4.1 Hình ảnh minh họa sản phẩm

```
./allocation 100
allocator>RQ P0 10 F
allocator>RQ P1 20 W
allocator>RL P0
allocator>STAT
Addresses [0:9] Unused
Addresses [10:29] Process P1
Addresses [30:99] Unused
```

```
allocator>RQ P2 10 W
allocator>STAT
Addresses [0:9] Unused
Addresses [10:29] Process P1
Addresses [30:39] Process P2
Addresses [40:99] Unused
allocator>C
allocator>STAT
Addresses [0:19] Process P1
Addresses [20:29] Process P2
Addresses [30:99] Unused
allocator>X
Bye
```

Hình 4.1: Minh họa chương trình

4.2 Giải thích kết quả

Chương trình quản lý bộ nhớ được thiết kế để quản lý và cấp phát bộ nhớ cho các tiến trình. Các bước thực hiện chương trình trên terminal:

4.2.1 Khởi tạo chương trình

Chương trình được khởi tạo với kích thước bộ nhớ là 100 đơn vị.

```
./allocation 100
```

4.2.2 Yêu cầu bộ nhớ

Chương trình yêu cầu bộ nhớ cho tiến trình P0 với kích thước 10 đơn vị và loại bộ nhớ là First-fit (F).

```
allocator>RQ P0 10 F
```

Chương trình yêu cầu bộ nhớ cho tiến trình P1 với kích thước 20 đơn vị và loại bộ nhớ là Worst-fit (W).

```
allocator>RQ P1 20 W
```

4.2.3 Giải phóng bộ nhớ

Chương trình giải phóng bộ nhớ của tiến trình P0.

```
allocator>RL P0
```

4.2.4 Trạng thái bộ nhớ

Chương trình hiển thị trạng thái bộ nhớ hiện tại, bao gồm các khối bộ nhớ đã được cấp phát và các khối bộ nhớ còn trống.

```
allocator>STAT
Addresses [0:9] Unused
Addresses [10:29] Process P1
Addresses [30:99] Unused
```

4.2.5 Yêu cầu bộ nhớ

Chương trình yêu cầu bộ nhớ cho tiến trình P2 với kích thước 10 đơn vị và loại bộ nhớ là Worst-fit (W).

```
allocator>RQ P2 10 W
```

4.2.6 Trạng thái bộ nhớ

Chương trình hiển thị trạng thái bộ nhớ hiện tại sau khi yêu cầu bộ nhớ cho tiến trình P2.

```
allocator>STAT
Addresses [0:9] Unused
Addresses [10:29] Process P1
Addresses [30:39] Process P2
Addresses [40:99] Unused
```

4.2.7 Nén bộ nhớ

Chương trình nén bộ nhớ để loại bỏ các khối bộ nhớ trống và hợp nhất các khối bộ nhớ liền kề.

```
allocator>C
```

4.2.8 Trạng thái bộ nhớ

Chương trình hiển thị trạng thái bộ nhớ hiện tại sau khi nén bộ nhớ.

```
allocator>STAT
Addresses [0:19] Process P1
Addresses [20:29] Process P2
Addresses [30:99] Unused
```

4.2.9 Kết thúc chương trình

Chương trình kết thúc và hiển thị thông báo "Bye".

```
allocator>X
Bye
```

Chương 5

Source Code

Source Code có ở đây