

HỆ ĐIỀU HÀNH

Phạm Đăng Hải
haipd-fit@mail.hut.edu.vn

Bộ môn Khoa học Máy tính
Viện Công nghệ Thông tin & Truyền Thông

Ngày 31 tháng 3 năm 2011



Chương 3 Quản lý bộ nhớ



Giới thiệu

- Mục đích của hệ thống máy tính: thực hiện chương trình
 - Chương trình và dữ liệu (*toàn bộ hoặc một phần*) phải nằm trong bộ nhớ chính trong khi thực hiện
 - **Byte tích cực:** Những byte nội dung đang được thực hiện tại thời điểm quan sát:
 - Phần chương trình chưa đưa vào bộ nhớ chính được lưu trên bộ nhớ thứ cấp (VD: *đĩa cứng*) \Rightarrow **Bộ nhớ ảo**
 - Cho phép lập trình viên không lo lắng về giới hạn bộ nhớ vật lý



Giới thiệu

- Mục đích của hệ thống máy tính: thực hiện chương trình
 - Chương trình và dữ liệu (*toàn bộ hoặc một phần*) phải nằm trong bộ nhớ chính trong khi thực hiện
 - **Byte tích cực:** Những byte nội dung đang được thực hiện tại thời điểm quan sát:
 - Phần chương trình chưa đưa vào bộ nhớ chính được lưu trên bộ nhớ thứ cấp (VD: *đĩa cứng*) \Rightarrow **Bộ nhớ ảo**
 - Cho phép lập trình viên không lo lắng về giới hạn bộ nhớ vật lý
- Để s/d CPU hiệu quả và tăng tốc độ đáp ứng của hệ thống:
 - Cần luân chuyển CPU thường xuyên giữa các tiến trình
 - Điều phối CPU (*Phần 3- Chương 2*)
 - Cần nhiều tiến trình sẵn sàng trong bộ nhớ
 - **Hệ số song song của hệ thống:** Số tiến trình đồng thời tồn tại trong hệ thống



Giới thiệu

- Mục đích của hệ thống máy tính: thực hiện chương trình
 - Chương trình và dữ liệu (*toàn bộ hoặc một phần*) phải nằm trong bộ nhớ chính trong khi thực hiện
 - **Byte tích cực:** Những byte nội dung đang được thực hiện tại thời điểm quan sát:
 - Phần chương trình chưa đưa vào bộ nhớ chính được lưu trên bộ nhớ thứ cấp (VD: *đĩa cứng*) \Rightarrow **Bộ nhớ ảo**
 - Cho phép lập trình viên không lo lắng về giới hạn bộ nhớ vật lý
- Để s/d CPU hiệu quả và tăng tốc độ đáp ứng của hệ thống:
 - Cần luân chuyển CPU thường xuyên giữa các tiến trình
 - Điều phối CPU (*Phần 3- Chương 2*)
 - Cần nhiều tiến trình sẵn sàng trong bộ nhớ
 - **Hệ số song song của hệ thống:** Số tiến trình đồng thời tồn tại trong hệ thống
- Tồn tại nhiều sơ đồ quản lý bộ nhớ khác nhau
 - Nhiều sơ đồ đòi hỏi trợ giúp từ phần cứng
 - Thiết kế phần cứng có thể được tích hợp chặt chẽ với HDH



Nội dung chính



Nội dung chính

- 1 Tổng quan
- 2 Các chiến lược quản lý bộ nhớ
- 3 Bộ nhớ ảo



1 Tổng quan

- Ví dụ
 - Bộ nhớ và chương trình
 - Liên kết địa chỉ
 - Các cấu trúc chương trình



- ❶ Tạo file thực thi dùng ngôn ngữ máy
- ❷ Tạo file thực thi từ nhiều modul



Ví dụ 1: Tạo chương trình thực thi dùng ngôn ngữ máy

```
#include <stdio.h>
char buf[19]={
    0xEB,0x08,0x48,0x65,0x6C, 0x6C,0x6F,0x21,0x24,0x90,
    0xB4,0x09,0xBA,0x02,0x01,0xCD,0x21,0xCD,0x20};

int main(int argc, char *argv[]){
    int i;
    FILE * f = fopen("Toto.com","w+");
    for(i= 0; i < 19;i++)
        fprintf(f,"%c",buf[i]);
    fclose(f);
    return 0;
}
```



1. Tổng quan

1.1 Ví dụ

Ví dụ 1: Kết quả

File toto.com có kích thước 19 bytes

```
D:\WORKSP~1\MY_COU~1\HDH\Exemples\Filecom>Dir toto.com
Le volume dans le lecteur D s'appelle Data
Le numéro de série du volume est 905D-54DF

Répertoire de D:\WORKSP~1\MY_COU~1\HDH\Exemples\Filecom
21/03/2011  08:28                19 Toto.com
                1 fichier(s)                19 octets
                0 Rép(s)          406 700 032 octets libres

D:\WORKSP~1\MY_COU~1\HDH\Exemples\Filecom>Toto
Hello!
D:\WORKSP~1\MY_COU~1\HDH\Exemples\Filecom>_
```

Nội dung các câu lệnh trong chương trình thực thi *toto.com*?



1. Tổng quan

1.1 Ví dụ

Ví dụ 1: Nội dung file

Dùng *debug* xem nội dung file và dịch ngược ra hợp ngữ

```
D:\WORKSP~1\MY_COU~1\HDH\Exemples\Filecom>debug toto.com

-d 100 11F
1555:0100 EB 08 48 65 6C 6C 6F 21-24 90 B4 09 BA 02 01 CD  ..Hello!$. .....
1555:0110 21 CD 20 00 00 00 00 00-00 00 00 00 00 00 00 00  ?.. .....

-u 100 112
1555:0100 EB08          JMP     010A
1555:0102 48             DEC     AX
1555:0103 65             DB      65
1555:0104 6C             DB      6C
1555:0105 6C             DB      6C
1555:0106 6F             DB      6F
1555:0107 2124          AND     [SI],SP
1555:0109 90             NOP
1555:010A B409          MOU     AH,09
1555:010C BA0201     MOU     DX,0102
1555:010F CD21          INT     21
1555:0111 CD20          INT     20

-g
Hello!
Fin normale du programme

-q

D:\WORKSP~1\MY_COU~1\HDH\Exemples\Filecom>_
```

Data

System call

1. Tổng quan

1.1 Ví dụ

Ví dụ 1: Thực hiện file toto.com

Nội dung file Toto.com (19 bytes)

EB 08	48 65 6C 6C 6F 21 24	90	B4 09	BA 02 01	CD 21	CD 20
-------	----------------------	----	-------	----------	-------	-------

1. Tổng quan

1.1 Ví dụ

Ví dụ 1: Thực hiện file toto.com

Nội dung file Toto.com (19 bytes)

EB 08	48 65 6C 6C 6F 21 24	90	B4 09	BA 02 01	CD 21	CD 20
-------	----------------------	----	-------	----------	-------	-------

Dịch ngược	
JMP	010A
DB	'Hello!\$'
NOP	
MOV	AH, 9
MOV	DX, 0102
INT	21
INT	20

1. Tổng quan

1.1 Ví dụ

Ví dụ 1: Thực hiện file toto.com

Nội dung file Toto.com (19 bytes)

EB 08	48 65 6C 6C 6F 21 24	90	B4 09	BA 02 01	CD 21	CD 20
-------	----------------------	----	-------	----------	-------	-------

Dịch ngược		CS:0000	PSP: Program Segment Prefix	⇐CS:IP
JMP	010A	...	JMP 010A	
DB	'Hello!\$'	CS:0100	'Hello!\$'	
NOP		CS:0102	NOP	
MOV	AH, 9	CS:0109	NOP	
MOV	DX, 0102	CS:010A	MOV AH, 9	
INT	21	CS:010C	MOV DX, 0102	
INT	20	CS:010F	INT 21	
		CS:0111	INT 20	
		CS:0113	...	

- Khi thực hiện, nạp *toto.com* vào bộ nhớ tại địa chỉ *CS:0100*
 - Các thanh ghi đoạn CS, ES, DS, SS cùng trở tới *PSP*
 - Thanh ghi **IP** có giá trị 100 (*CS:IP* trở đến lệnh đầu tiên)
 - **SP** trở tới cuối đoạn; Các thanh ghi thông dụng bị xóa (0)



1. Tổng quan

1.1 Ví dụ

Ví dụ 1: Thực hiện file toto.com

Nội dung file Toto.com (19 bytes)

EB 08	48 65 6C 6C 6F 21 24	90	B4 09	BA 02 01	CD 21	CD 20
-------	----------------------	----	-------	----------	-------	-------

Dịch ngược		CS:0000	PSP: Program Segment Prefix	
JMP 010A		...	JMP 010A	
DB 'Hello!\$'		CS:0100	'Hello!\$'	
NOP		CS:0102	NOP	
MOV AH, 9		CS:0109	MOV AH, 9	⇐CS:IP
MOV DX, 0102		CS:010A	MOV DX, 0102	
INT 21		CS:010C	INT 21	
INT 20		CS:010F	INT 20	
		CS:0111	INT 20	
		CS:0113	...	

- Khi thực hiện, nạp *toto.com* vào bộ nhớ tại địa chỉ *CS:0100*
 - Các thanh ghi đoạn CS, ES, DS, SS cùng trở tới *PSP*
 - Thanh ghi **IP** có giá trị 100 (*CS:IP* trở đến lệnh đầu tiên)
 - **SP** trở tới cuối đoạn; Các thanh ghi thông dụng bị xóa (0)



1. Tổng quan

1.1 Ví dụ

Ví dụ 1: Thực hiện file toto.com

Nội dung file Toto.com (19 bytes)

EB 08	48 65 6C 6C 6F 21 24	90	B4 09	BA 02 01	CD 21	CD 20
-------	----------------------	----	-------	----------	-------	-------

Dịch ngược				
JMP 010A	CS:0000	PSP: Program		
DB 'Hello!\$'	...	Segment Prefix		
NOP	CS:0100	JMP 010A		
MOV AH, 9	CS:0102	'Hello!\$'		
MOV DX, 0102	CS:0109	NOP		
INT 21	CS:010A	MOV AH, 9		
INT 20	CS:010C	MOV DX, 0102	⇐CS:IP	
	CS:010F	INT 21		
	CS:0111	INT 20		
	CS:0113	...		

- Khi thực hiện, nạp *toto.com* vào bộ nhớ tại địa chỉ *CS:0100*
 - Các thanh ghi đoạn CS, ES, DS, SS cùng trở tới *PSP*
 - Thanh ghi **IP** có giá trị 100 (*CS:IP* trở đến lệnh đầu tiên)
 - **SP** trở tới cuối đoạn; Các thanh ghi thông dụng bị xóa (0)



1. Tổng quan

1.1 Ví dụ

Ví dụ 1: Thực hiện file toto.com

Nội dung file Toto.com (19 bytes)

EB 08	48 65 6C 6C 6F 21 24	90	B4 09	BA 02 01	CD 21	CD 20
-------	----------------------	----	-------	----------	-------	-------

Dịch ngược				
JMP 010A	CS:0000		PSP: Program	
DB 'Hello!\$'	...		Segment Prefix	
NOP	CS:0100		JMP 010A	
MOV AH, 9	CS:0102		'Hello!\$'	
MOV DX, 0102	CS:0109		NOP	
INT 21	CS:010A		MOV AH, 9	
INT 20	CS:010C		MOV DX, 0102	
	CS:010F		INT 21	⇐CS:IP
	CS:0111		INT 20	
	CS:0113		...	

- Khi thực hiện, nạp *toto.com* vào bộ nhớ tại địa chỉ *CS:0100*
 - Các thanh ghi đoạn CS, ES, DS,SS cùng trở tới *PSP*
 - Thanh ghi **IP** có giá trị 100 (*CS:IP* trở đến lệnh đầu tiên)
 - **SP** trở tới cuối đoạn; Các thanh ghi thông dụng bị xóa (0)



1. Tổng quan

1.1 Ví dụ

Ví dụ 1: Thực hiện file toto.com

Nội dung file Toto.com (19 bytes)

EB 08	48 65 6C 6C 6F 21 24	90	B4 09	BA 02 01	CD 21	CD 20
-------	----------------------	----	-------	----------	-------	-------

Dịch ngược					
JMP 010A	CS:0000	...	PSP: Program Segment Prefix		Hello!
DB 'Hello!\$'	CS:0100		JMP 010A		
NOP	CS:0102		'Hello!\$'		
MOV AH, 9	CS:0109		NOP		
MOV DX, 0102	CS:010A		MOV AH, 9		
INT 21	CS:010C		MOV DX, 0102		
INT 20	CS:010F		INT 21		
	CS:0111		INT 20	←CS:IP	
	CS:0113		...		

- Khi thực hiện, nạp *toto.com* vào bộ nhớ tại địa chỉ *CS:0100*
 - Các thanh ghi đoạn CS, ES, DS, SS cùng trở tới *PSP*
 - Thanh ghi **IP** có giá trị 100 (*CS:IP* trở đến lệnh đầu tiên)
 - **SP** trở tới cuối đoạn; Các thanh ghi thông dụng bị xóa (0)



1. Tổng quan

1.1 Ví dụ

Ví dụ 1: Thực hiện file toto.com

Nội dung file Toto.com (19 bytes)

EB 08	48 65 6C 6C 6F 21 24	90	B4 09	BA 02 01	CD 21	CD 20
-------	----------------------	----	-------	----------	-------	-------

Dịch ngược				
JMP 010A	CS:0000	PSP: Program	Hello! terminated	
DB 'Hello!\$'	...	Segment Prefix		
NOP	CS:0100	JMP 010A		
MOV AH, 9	CS:0102	'Hello!\$'		
MOV DX, 0102	CS:0109	NOP		
INT 21	CS:010A	MOV AH, 9		
INT 20	CS:010C	MOV DX, 0102		
	CS:010F	INT 21		
	CS:0111	INT 20		
	CS:0113	...		

- Khi thực hiện, nạp *toto.com* vào bộ nhớ tại địa chỉ *CS:0100*
 - Các thanh ghi đoạn CS, ES, DS, SS cùng trở tới *PSP*
 - Thanh ghi **IP** có giá trị 100 (*CS:IP* trở đến lệnh đầu tiên)
 - **SP** trở tới cuối đoạn; Các thanh ghi thông dụng bị xóa (0)



Ví dụ 2: Tạo file thực thi từ nhiều modul

Toto project

file main.c

```
#include <stdio.h>
extern int x, y;
extern void toto();
int main(int argc, char *argv[]){
    toto();
    printf("KQ: %d \n", x * y);
    return 0;
}
```

file M1.c

```
int y = 10;
```

file M2.c

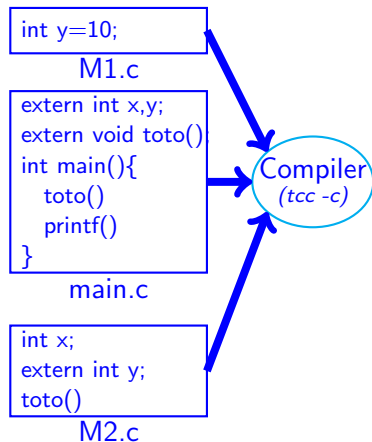
```
int x;
extern int y;
void toto(){
    x = 10 * y;
}
```

Ket qua

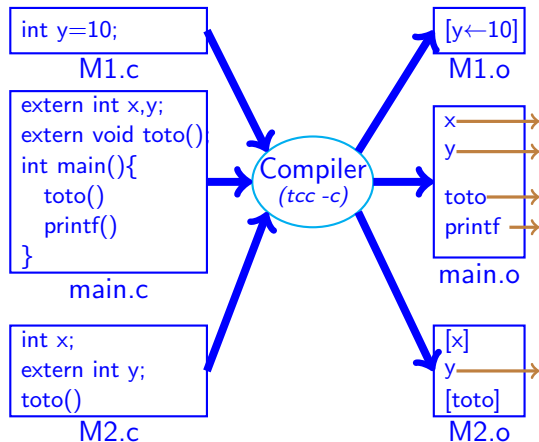
```
KQ: 1000
```



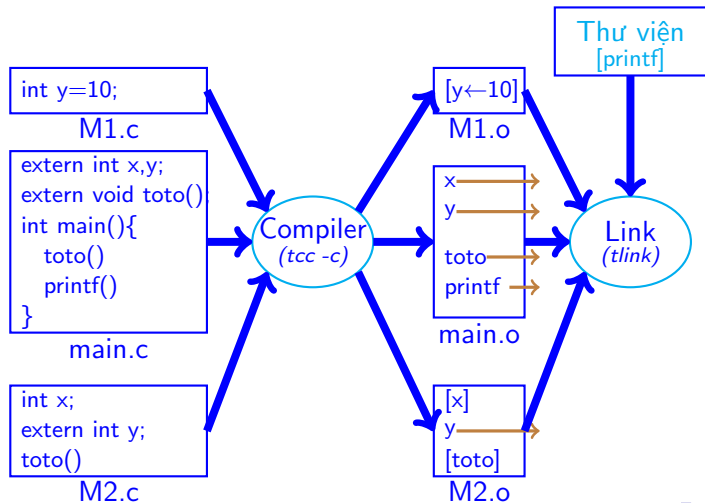
Ví dụ 2: Quá trình xử lý toto project



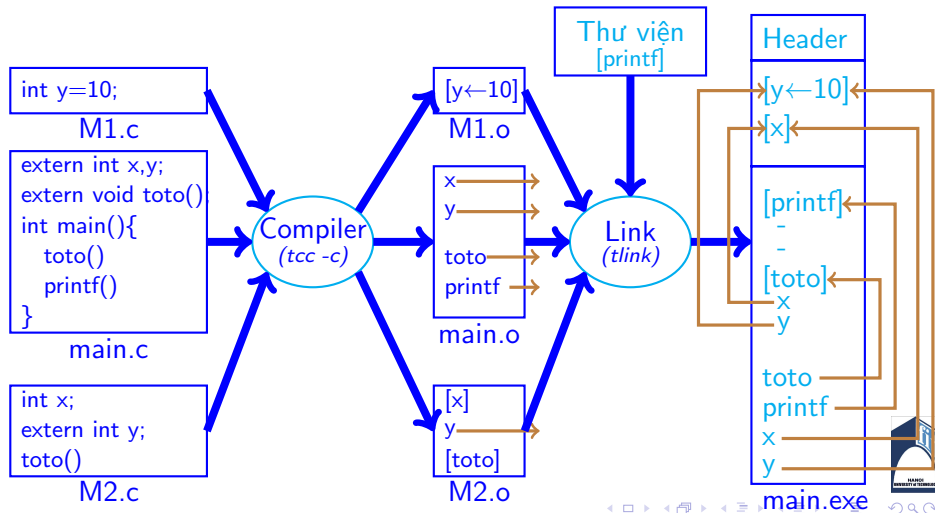
Ví dụ 2: Quá trình xử lý toto project



Ví dụ 2: Quá trình xử lý toto project



Ví dụ 2: Quá trình xử lý toto project



1 Tổng quan

- Ví dụ
- Bộ nhớ và chương trình
- Liên kết địa chỉ
- Các cấu trúc chương trình



Phân cấp bộ nhớ

- Bộ nhớ là tài nguyên quan trọng của hệ thống
 - Chương trình phải nằm trong bộ nhớ trong để thực hiện



Phân cấp bộ nhớ

- Bộ nhớ là tài nguyên quan trọng của hệ thống
 - Chương trình phải nằm trong bộ nhớ trong để thực hiện
- Bộ nhớ được đặc trưng bởi kích thước và tốc độ truy nhập



Phân cấp bộ nhớ

- Bộ nhớ là tài nguyên quan trọng của hệ thống
 - Chương trình phải nằm trong bộ nhớ trong để thực hiện
- Bộ nhớ được đặc trưng bởi kích thước và tốc độ truy nhập
- Bộ nhớ được phân cấp theo tốc độ truy nhập

Loại bộ nhớ	Kích thước	Tốc độ
Thanh ghi (Registers)	bytes	Tốc độ CPU(ηs)
Cache trên VXL	Kilo Bytes	10 nano seconds
Cache mức 2	KiloByte-MegaByte	100 nanoseconds
Bộ nhớ chính	MegaByte-GigaByte	Micro-seconds
Bộ nhớ lưu trữ (Disk)	GigaByte-Terabytes	Mili-Seconds
Băng từ, đĩa quang	Không giới hạn	10 Seconds



Bộ nhớ chính

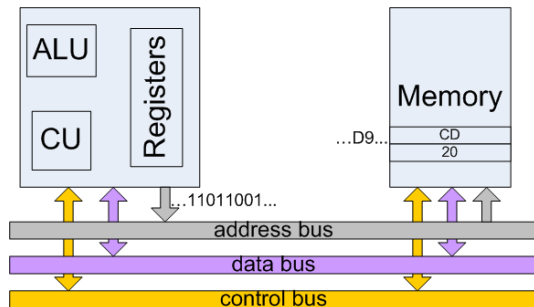
00000

00001

Memory

FFFE

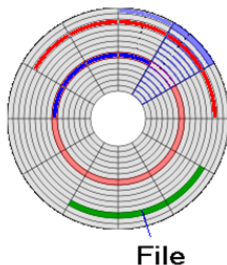
FFFF



- Dùng lưu trữ dữ liệu và chương trình
- Là mảng các ô nhớ kiểu bytes, words
- Mỗi ô nhớ có một địa chỉ riêng
 - **Địa chỉ vật lý:** địa chỉ x/hiện ở chân VXL



Chương trình

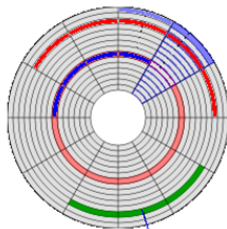


0101011001111001010101010101101

Header	Data	Code
--------	------	------

- Tồn tại trên thiết bị lưu trữ ngoài
- Là các file nhị phân thực thi được
 - Vùng tham số file
 - Lệnh máy (*mã nhị phân*),
 - Vùng dữ liệu (*biến toàn cục*),
 - ...

Chương trình



File

0101011001111001010101010101101

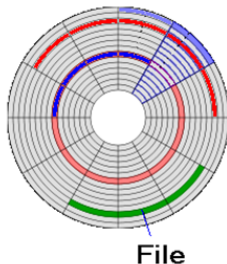
Header

Data

Code

- Tồn tại trên thiết bị lưu trữ ngoài
- Là các file nhị phân thực thi được
 - Vùng tham số file
 - Lệnh máy (*mã nhị phân*),
 - Vùng dữ liệu (*biến toàn cục*),
 - ...
- Phải được đưa vào bộ nhớ trong và được đặt trong một tiến trình để thực hiện (*tiến trình thực hiện chương trình*)

Chương trình



0101011001111001010101010101101

Header	Data	Code
--------	------	------

- Tồn tại trên thiết bị lưu trữ ngoài
- Là các file nhị phân thực thi được
 - Vùng tham số file
 - Lệnh máy (*mã nhị phân*),
 - Vùng dữ liệu (*biến toàn cục*),
 - ...
- Phải được đưa vào bộ nhớ trong và được đặt trong một tiến trình để thực hiện (*tiến trình thực hiện chương trình*)
- **Hàng đợi vào** (*input queue*)
 - Tập các tiến trình ở bộ nhớ ngoài (*thông thường disk*)
 - Đợi để được đưa vào bộ nhớ trong và thực hiện

Thực hiện chương trình

- Nạp chương trình vào bộ nhớ
 - Đọc và phân tích (*dịch*) file thực thi (VD file *.com, file *.exe)
 - Xin vùng nhớ để nạp chương trình từ file trên đĩa
 - Thiết lập các tham số, các thanh ghi tới giá trị thích hợp



Thực hiện chương trình

- Nạp chương trình vào bộ nhớ
 - Đọc và phân tích (*dịch*) file thực thi (VD file *.com, file *.exe)
 - Xin vùng nhớ để nạp chương trình từ file trên đĩa
 - Thiết lập các tham số, các thanh ghi tối giá trị thích hợp
- Thực thi chương trình
 - CPU lấy các lệnh trong bộ nhớ tại vị trí được xác định bởi bộ đếm chương trình (*Program counter*)
 - Cập thanh ghi CS:IP với VXL họ Intel (Ví dụ : 80x86)
 - CPU giải mã lệnh
 - Có thể lấy thêm toán hạng từ bộ nhớ
 - Thực hiện lệnh với toán hạng
 - Nếu cần thiết, lưu kết quả vào bộ nhớ tại một địa chỉ xác định



Thực hiện chương trình

- Nạp chương trình vào bộ nhớ
 - Đọc và phân tích (*dịch*) file thực thi (*VD file *.com, file *.exe*)
 - Xin vùng nhớ để nạp chương trình từ file trên đĩa
 - Thiết lập các tham số, các thanh ghi tối giá trị thích hợp
- Thực thi chương trình
 - CPU lấy các lệnh trong bộ nhớ tại vị trí được xác định bởi bộ đếm chương trình (*Program counter*)
 - Cập thanh ghi CS:IP với VXL họ Intel (*Ví dụ : 80x86*)
 - CPU giải mã lệnh
 - Có thể lấy thêm toán hạng từ bộ nhớ
 - Thực hiện lệnh với toán hạng
 - Nếu cần thiết, lưu kết quả vào bộ nhớ tại một địa chỉ xác định
- Thực hiện xong
 - Giải phóng vùng không gian nhớ dành cho chương trình



Thực hiện chương trình

- Nạp chương trình vào bộ nhớ
 - Đọc và phân tích (*dịch*) file thực thi (VD file *.com, file *.exe)
 - Xin vùng nhớ để nạp chương trình từ file trên đĩa
 - Thiết lập các tham số, các thanh ghi tới giá trị thích hợp
- Thực thi chương trình
 - CPU lấy các lệnh trong bộ nhớ tại vị trí được xác định bởi bộ đếm chương trình (*Program counter*)
 - Cập thanh ghi CS:IP với VXL họ Intel (Ví dụ : 80x86)
 - CPU giải mã lệnh
 - Có thể lấy thêm toán hạng từ bộ nhớ
 - Thực hiện lệnh với toán hạng
 - Nếu cần thiết, lưu kết quả vào bộ nhớ tại một địa chỉ xác định
- Thực hiện xong
 - Giải phóng vùng không gian nhớ dành cho chương trình
- Vấn đề
 - Chương trình có thể được nạp vào vị trí bất kỳ trong bộ nhớ
 - Khi thực hiện chương trình sinh ra chuỗi địa chỉ bộ nhớ

Truy nhập địa chỉ bộ nhớ như thế nào?



1. Tổng quan

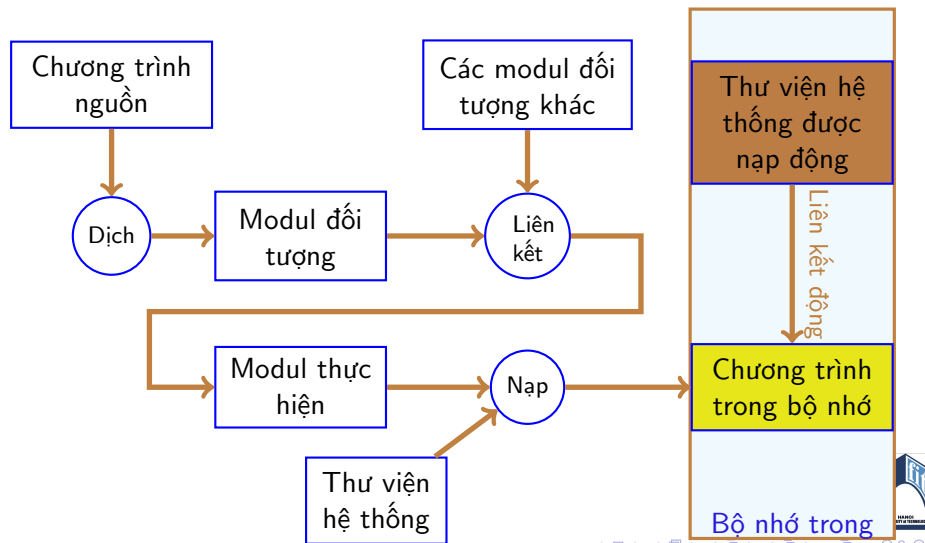
1.3 Liên kết địa chỉ

1 Tổng quan

- Ví dụ
- Bộ nhớ và chương trình
- Liên kết địa chỉ
- Các cấu trúc chương trình



Các bước xử lý chương trình ứng dụng



Các kiểu địa chỉ

- Địa chỉ biểu tượng (*symbolic*)
 - Là tên của đối tượng trong chương trình nguồn
 - Ví dụ: counter, x, y,...
- Địa chỉ tương đối
 - Sinh ra từ địa chỉ biểu tượng trong giai đoạn dịch (*compiler*)
 - Là vị trí tương đối của đối tượng kể từ đầu modul
 - Byte thứ 10 kể từ đầu modul
 - EB08 \Rightarrow JMP +08: Nhảy tới vị trí cách vị trí hiện tại 8 ô
- Địa chỉ tuyệt đối
 - Sinh ra từ địa chỉ tương đối trong giai đoạn nạp chương trình thực thi vào bộ nhớ để thực hiện
 - Là địa chỉ của đối tượng trong bộ nhớ vật lý-địa chỉ vật lý
 - Ví dụ: JMP 010A: Nhảy tới ô nhớ có vị trí 010Ah tại cùng đoạn mã lệnh (CS)
 - Nếu CS=1555h, sẽ đi tới vị trí: $1555h * 10h + 010Ah = 1560Ah$



Xác định địa chỉ

Xác định địa chỉ câu lệnh và dữ liệu trong bộ nhớ có thể thực hiện tại các giai đoạn khác nhau khi xử lý chương trình ứng dụng

- **Giai đoạn dịch:**

- Sử dụng khi biết chương trình sẽ nằm ở đâu trong bộ nhớ
- Khi dịch sẽ sinh ra mã (*địa chỉ*) tuyệt đối
- Phải dịch lại khi vị trí bắt đầu thay đổi

- **Thời điểm nạp:**

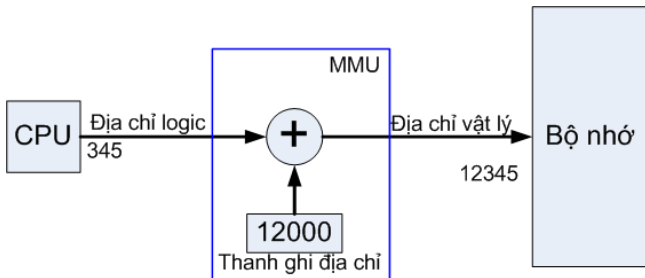
- Sử dụng khi không biết c/trình sẽ nằm ở đâu trong bộ nhớ
- Các đối tượng được dịch ra sẽ mang địa chỉ tương đối
- Xác định địa chỉ được hoãn lại tới khi nạp chương trình vào bộ nhớ

- **Trong khi thực hiện:**

- S/dụng khi các tiến trình có thể thay đổi vị trí trong khi t/hiện
- Xác định địa chỉ được hoãn lại tới khi thực thi chương trình
- Thường đòi hỏi trợ giúp từ phần cứng
- Được sử dụng trong nhiều hệ điều hành



Địa chỉ vật lý-địa chỉ logic



- Địa chỉ logic (*địa chỉ ảo*)
 - Được sinh ra trong tiến trình, (*CPU đưa ra*)
 - Được khối quản lý bộ nhớ (*MMU*) chuyển sang địa chỉ vật lý khi truy nhập tới đối tượng trong chương trình
- Địa chỉ vật lý
 - Địa chỉ của một phần tử (*byte/word*) của bộ nhớ
 - Tương ứng với địa chỉ logic được CPU đưa ra
- Chương trình làm việc với một địa chỉ logic

1. Tổng quan

1.4 Các cấu trúc chương trình

1 Tổng quan

- Ví dụ
- Bộ nhớ và chương trình
- Liên kết địa chỉ
- Các cấu trúc chương trình

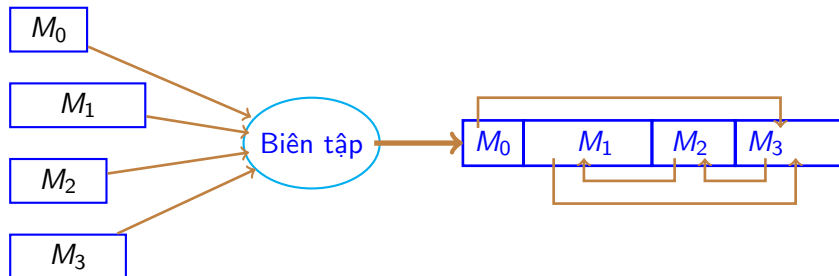


Các cấu trúc chương trình

- ❶ Cấu trúc tuyến tính
- ❷ Cấu trúc nạp động
- ❸ Cấu trúc liên kết động
- ❹ Cấu trúc Overlays



Cấu trúc tuyến tính I



- Sau khi biên tập, các modul được tập hợp thành một chương trình hoàn thiện
 - Chứa đầy đủ các thông tin để có thể thực hiện được
 - Các biến trở ngoài đã thay bằng giá trị cụ thể
 - Để thực hiện, chỉ cần định vị một lần trong bộ nhớ

Cấu trúc tuyến tính II

- Ưu điểm
 - Đơn giản, dễ tổ chức biên tập và định vị chương trình
 - Thời gian thực hiện nhanh
 - Tính lưu động cao
- Nhược điểm
 - Lãng phí nhớ
 - Không phải toàn bộ chương trình đều cần thiết cho thực hiện chương trình
 - Không thực hiện được chương trình có kích thước lớn hơn kích thước bộ nhớ vật lý



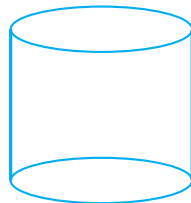
Cấu trúc nạp động

M_0

M_1

M_2

M_3



- Mỗi modul được biên tập riêng

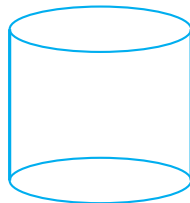
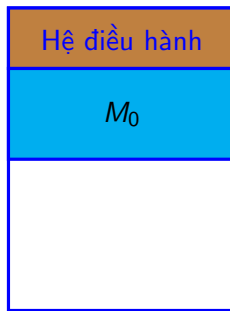
Cấu trúc nạp động

M_0

M_1

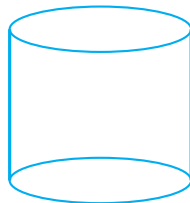
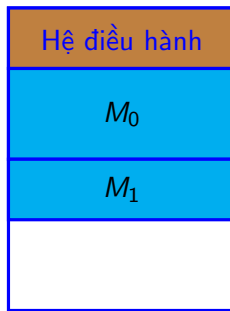
M_2

M_3



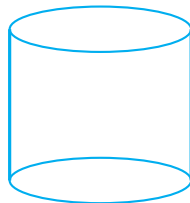
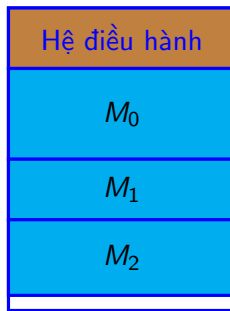
- Mỗi modul được biên tập riêng
- Khi thực hiện, hệ thống sẽ định vị modul gốc

Cấu trúc nạp động

 M_0 M_1 M_2 M_3 

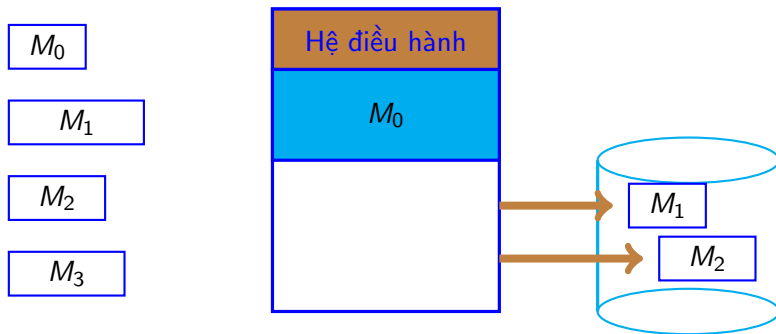
- Mỗi modul được biên tập riêng
- Khi thực hiện, hệ thống sẽ định vị modul gốc
- Cần tới modul nào sẽ xin bộ nhớ và giải nạp modul vào

Cấu trúc nạp động

 M_0 M_1 M_2 M_3 

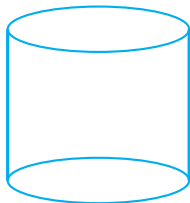
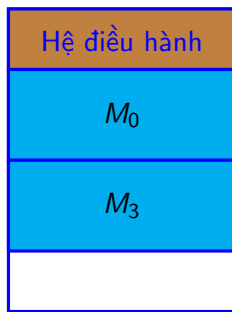
- Mỗi modul được biên tập riêng
- Khi thực hiện, hệ thống sẽ định vị modul gốc
- Cần tới modul nào sẽ xin bộ nhớ và giải nạp modul vào

Cấu trúc nạp động



- Mỗi modul được biên tập riêng
- Khi thực hiện, hệ thống sẽ định vị modul gốc
- Cần tới modul nào sẽ xin bộ nhớ và giải nạp modul vào
- Khi sử dụng xong một modul, hoặc khi thiếu vùng nhớ sẽ đưa nhưng modul không cần thiết ra ngoài

Cấu trúc nạp động

 M_0 M_1 M_2 M_3 

- Mỗi modul được biên tập riêng
- Khi thực hiện, hệ thống sẽ định vị modul gốc
- Cần tới modul nào sẽ xin bộ nhớ và giải nạp modul vào
- Khi sử dụng xong một modul, hoặc khi thiếu vùng nhớ sẽ đưa nhưng modul không cần thiết ra ngoài

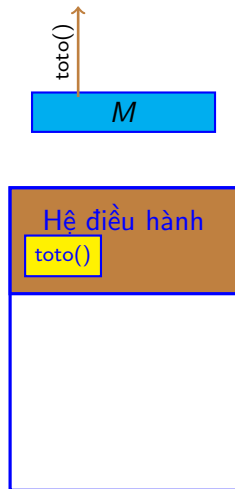
Cấu trúc nạp động (tiếp)

- Có thể sử dụng vùng nhớ nhiều hơn phần dành cho chương trình
- Hiệu quả sử dụng bộ nhớ cao nếu quản lý tốt
 - Sai lầm sẽ dẫn tới lãng phí bộ nhớ và tăng thời gian thực hiện
- Tốc độ thực hiện chậm
- Yêu cầu người sử dụng phải nạp và xóa các modul
 - Người dùng phải nắm rõ hệ thống
 - Giảm tính lưu động



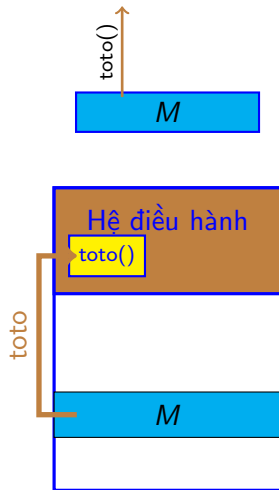
Cấu trúc liên kết động (DLL:Dynamic-link library)

- Các liên kết sẽ hoãn lại cho tới khi thực hiện chương trình
- Một phần của đoạn mã (*stub*) được sử dụng để tìm kiếm thủ tục tương ứng trong thư viện trong bộ nhớ



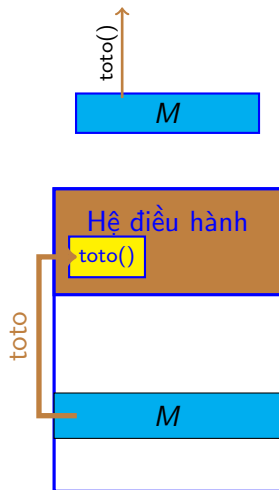
Cấu trúc liên kết động (DLL:Dynamic-link library)

- Các liên kết sẽ hoãn lại cho tới khi thực hiện chương trình
- Một phần của đoạn mã (*stub*) được sử dụng để tìm kiếm thủ tục tương ứng trong thư viện trong bộ nhớ
- Khi tìm thấy, *stub* sẽ được thay thế với địa chỉ của thủ tục và thực hiện thủ tục



Cấu trúc liên kết động (DLL:Dynamic-link library)

- Các liên kết sẽ hoãn lại cho tới khi thực hiện chương trình
- Một phần của đoạn mã (*stub*) được sử dụng để tìm kiếm thủ tục tương ứng trong thư viện trong bộ nhớ
- Khi tìm thấy, *stub* sẽ được thay thế với địa chỉ của thủ tục và thực hiện thủ tục
- Hữu ích cho xây dựng thư viện



Cấu trúc Overlays

- Modul được chia thành các mức
 - Mức 0 chứa modul gốc, nạp và định vị chương trình
 - Mức 1 chứa các Modul được gọi từ những modul ở mức 0 và không đồng thời tồn tại
 - ...



Cấu trúc Overlays

- Modul được chia thành các mức
 - Mức 0 chứa modul gốc, nạp và định vị chương trình
 - Mức 1 chứa các Modul được gọi từ những modul ở mức 0 và không đồng thời tồn tại
 - ...
- Bộ nhớ cũng được chia thành mức ứng với mức chương trình
 - Kích thước bằng kích thước của modul lớn nhất cùng mức



Cấu trúc Overlays

- Modul được chia thành các mức
 - Mức 0 chứa modul gốc, nạp và định vị chương trình
 - Mức 1 chứa các Modul được gọi từ những modul ở mức 0 và không đồng thời tồn tại
 - ...
- Bộ nhớ cũng được chia thành mức ứng với mức chương trình
 - Kích thước bằng kích thước của modul lớn nhất cùng mức
- Để có cấu trúc Overlay, cần cung cấp thêm các thông tin
 - Chương trình bao nhiêu mức, mỗi mức gồm những modul nào
 - Thông tin cung cấp lưu trong file (*sơ đồ overlay*)



Cấu trúc Overlays

- Modul được chia thành các mức
 - Mức 0 chứa modul gốc, nạp và định vị chương trình
 - Mức 1 chứa các Modul được gọi từ những modul ở mức 0 và không đồng thời tồn tại
 - ...
- Bộ nhớ cũng được chia thành mức ứng với mức chương trình
 - Kích thước bằng kích thước của modul lớn nhất cùng mức
- Để có cấu trúc Overlay, cần cung cấp thêm các thông tin
 - Chương trình bao nhiêu mức, mỗi mức gồm những modul nào
 - Thông tin cung cấp lưu trong file (*sơ đồ overlay*)
- Modul mức 0 được biên tập thành file thực thi riêng

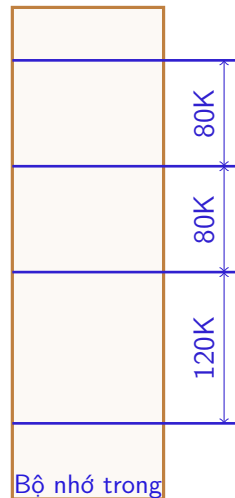
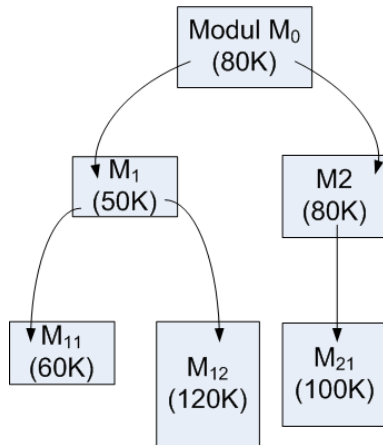


Cấu trúc Overlays

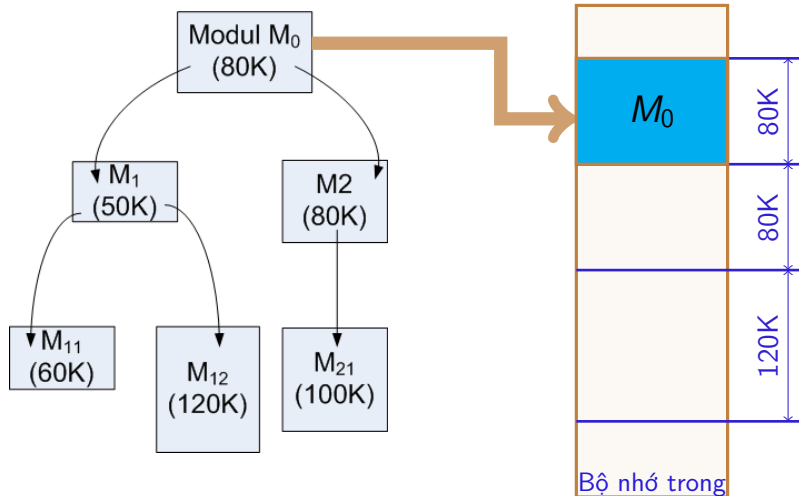
- Modul được chia thành các mức
 - Mức 0 chứa modul gốc, nạp và định vị chương trình
 - Mức 1 chứa các Modul được gọi từ những modul ở mức 0 và không đồng thời tồn tại
 - ...
- Bộ nhớ cũng được chia thành mức ứng với mức chương trình
 - Kích thước bằng kích thước của modul lớn nhất cùng mức
- Để có cấu trúc Overlay, cần cung cấp thêm các thông tin
 - Chương trình bao nhiêu mức, mỗi mức gồm những modul nào
 - Thông tin cung cấp lưu trong file (*sơ đồ overlay*)
- Modul mức 0 được biên tập thành file thực thi riêng
- Khi thực hiện chương trình
 - Nạp modul mức 0 như chương trình tuyến tính
 - Cần tới modul khác, sẽ nạp modul vào mức bộ nhớ tương ứng
 - Nếu có modul đồng mức tồn tại, đưa ra bên ngoài



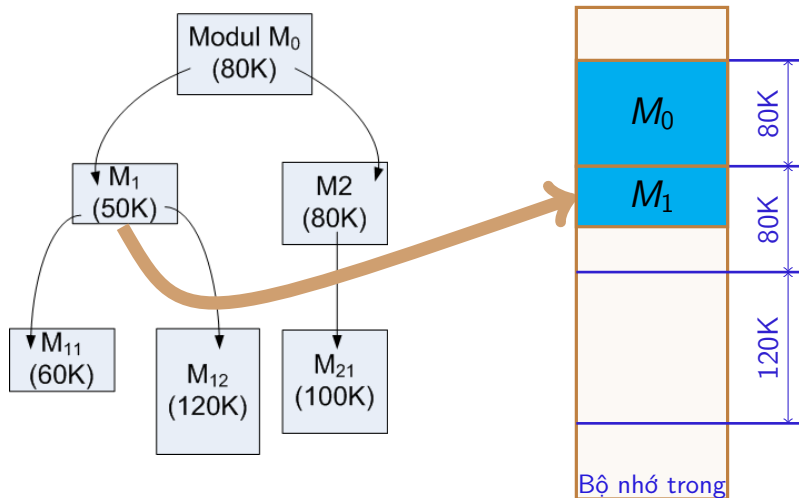
Cấu trúc Overlays: Ví dụ



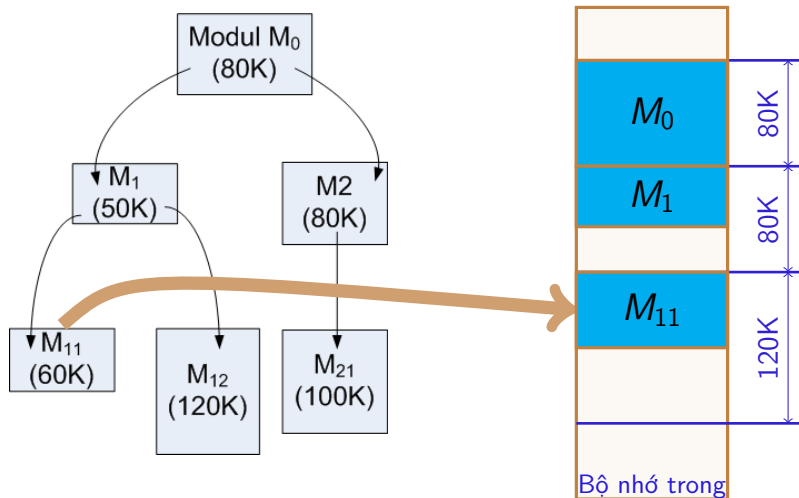
Cấu trúc Overlays: Ví dụ



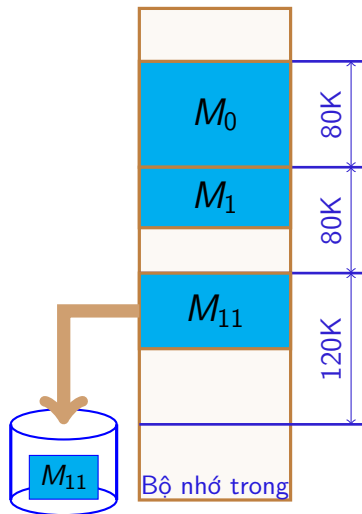
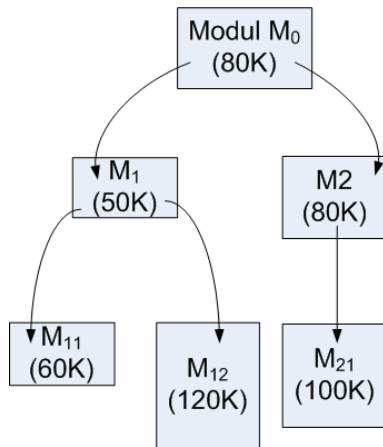
Cấu trúc Overlays: Ví dụ



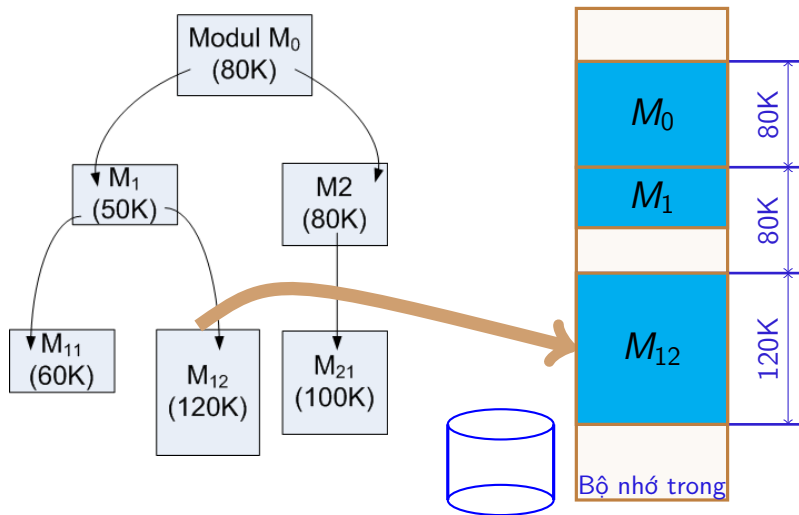
Cấu trúc Overlays: Ví dụ



Cấu trúc Overlays: Ví dụ



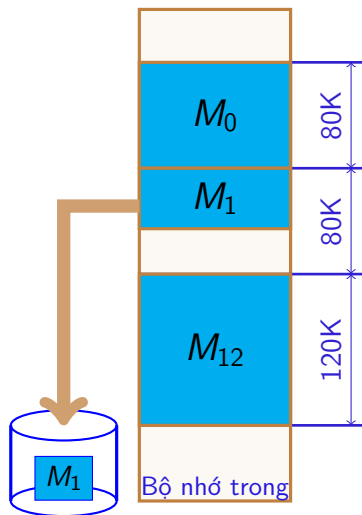
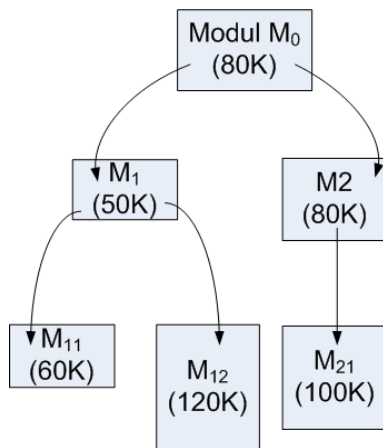
Cấu trúc Overlays: Ví dụ



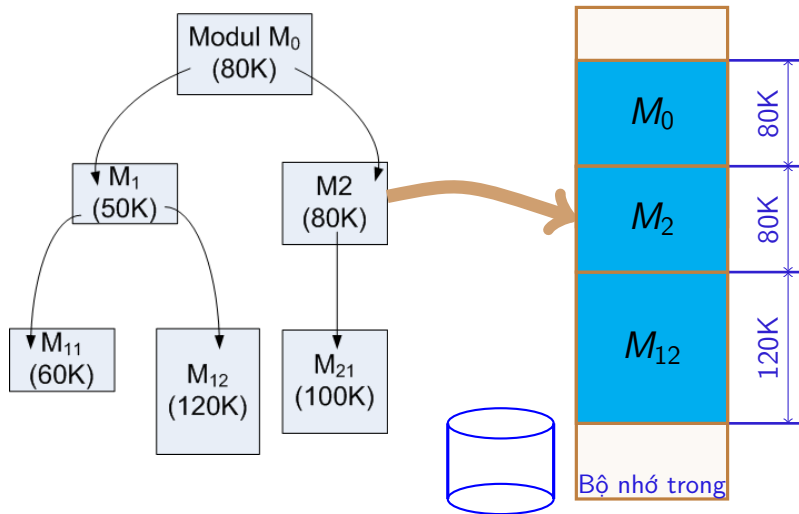
1. Tổng quan

1.4 Các cấu trúc chương trình

Cấu trúc Overlays: Ví dụ



Cấu trúc Overlays: Ví dụ



Cấu trúc Overlays: Nhận xét

- Cho phép dùng chương trình có kích thước lớn hơn kích thước hệ điều hành dành cho



Cấu trúc Overlays: Nhận xét

- Cho phép dùng chương trình có kích thước lớn hơn kích thước hệ điều hành dành cho
- Yêu cầu người sử dụng cung cấp các thông tin phụ
 - Hiệu quả sử dụng phụ thuộc vào các thông tin được cung cấp



Cấu trúc Overlays: Nhận xét

- Cho phép dùng chương trình có kích thước lớn hơn kích thước hệ điều hành dành cho
- Yêu cầu người sử dụng cung cấp các thông tin phụ
 - Hiệu quả sử dụng phụ thuộc vào các thông tin được cung cấp
- Hiệu quả sử dụng bộ nhớ phụ thuộc cách tổ chức các modul trong chương trình
 - Nếu tồn tại một modul có kích thước lớn hơn các modul khác cùng mức rất nhiều \Rightarrow Hiệu quả giảm rõ rệt



Cấu trúc Overlays: Nhận xét

- Cho phép dùng chương trình có kích thước lớn hơn kích thước hệ điều hành dành cho
- Yêu cầu người sử dụng cung cấp các thông tin phụ
 - Hiệu quả sử dụng phụ thuộc vào các thông tin được cung cấp
- Hiệu quả sử dụng bộ nhớ phụ thuộc cách tổ chức các modul trong chương trình
 - Nếu tồn tại một modul có kích thước lớn hơn các modul khác cùng mức rất nhiều \Rightarrow Hiệu quả giảm rõ rệt
- Quá trình nạp các modul là động, nhưng chương trình có tính chất tĩnh \Rightarrow Không thay đổi trong các lần thực hiện



Cấu trúc Overlays: Nhận xét

- Cho phép dùng chương trình có kích thước lớn hơn kích thước hệ điều hành dành cho
- Yêu cầu người sử dụng cung cấp các thông tin phụ
 - Hiệu quả sử dụng phụ thuộc vào các thông tin được cung cấp
- Hiệu quả sử dụng bộ nhớ phụ thuộc cách tổ chức các modul trong chương trình
 - Nếu tồn tại một modul có kích thước lớn hơn các modul khác cùng mức rất nhiều \Rightarrow Hiệu quả giảm rõ rệt
- Quá trình nạp các modul là động, nhưng chương trình có tính chất tĩnh \Rightarrow Không thay đổi trong các lần thực hiện
- Cung cấp thêm bộ nhớ tự do, hiệu quả vẫn không đổi



Kết luận



Nội dung chính

1 Tổng quan

2 Các chiến lược quản lý bộ nhớ

3 Bộ nhớ ảo



2 Các chiến lược quản lý bộ nhớ

- Chiến lược phân chương cố định
- Chiến lược phân chương động
- Chiến lược phân đoạn
- Chiến lược phân trang
- Chiến lược kết hợp phân đoạn-phân trang



Nguyên tắc

- Bộ nhớ được chia thành n phần
 - Mỗi phần gọi là một chương (*partition*)
 - Chương không nhất thiết có kích thước bằng nhau
 - Chương được sử dụng như một vùng nhớ độc lập
 - Tại một thời điểm chỉ cho phép một chương trình tồn tại
 - Các chương trình nằm trong vùng nhớ cho tới khi kết thúc

Nguyên tắc

- Bộ nhớ được chia thành n phần
 - Mỗi phần gọi là một chương (*partition*)
 - Chương không nhất thiết có kích thước bằng nhau
 - Chương được sử dụng như một vùng nhớ độc lập
 - Tại một thời điểm chỉ cho phép một chương trình tồn tại
 - Các chương trình nằm trong vùng nhớ cho tới khi kết thúc
- Ví dụ: Xét hệ thống:



Process	Size	time
P_1	120	20
P_2	80	15
P_3	70	5
P_4	50	5
P_5	140	12
Hàng đợi		

Nhận xét

- Đơn giản, dễ tổ chức bảo vệ
 - Chương trình và vùng nhớ có một khóa bảo vệ
 - So sánh 2 khóa với nhau khi nạp chương trình



Nhận xét

- Đơn giản, dễ tổ chức bảo vệ
 - Chương trình và vùng nhớ có một khóa bảo vệ
 - So sánh 2 khóa với nhau khi nạp chương trình
- Giảm thời gian tìm kiếm



Nhận xét

- Đơn giản, dễ tổ chức bảo vệ
 - Chương trình và vùng nhớ có một khóa bảo vệ
 - So sánh 2 khóa với nhau khi nạp chương trình
- Giảm thời gian tìm kiếm
- Phải sao các modul điều khiển ra làm nhiều bản và lưu ở nhiều nơi



Nhận xét

- Đơn giản, dễ tổ chức bảo vệ
 - Chương trình và vùng nhớ có một khóa bảo vệ
 - So sánh 2 khóa với nhau khi nạp chương trình
- Giảm thời gian tìm kiếm
- Phải sao các modul điều khiển ra làm nhiều bản và lưu ở nhiều nơi
- Hệ số song song không thể vượt quá n



Nhận xét

- Đơn giản, dễ tổ chức bảo vệ
 - Chương trình và vùng nhớ có một khóa bảo vệ
 - So sánh 2 khóa với nhau khi nạp chương trình
 - Giảm thời gian tìm kiếm
 - Phải sao các modul điều khiển ra làm nhiều bản và lưu ở nhiều nơi
 - Hệ số song song không thể vượt quá n
 - Bị phân đoạn bộ nhớ
 - Kích thước chương trình lớn hơn kích thước chương lớn nhất
 - Tổng bộ nhớ tự do còn lớn, nhưng không dùng để nạp các chương trình khác
- ⇒ Sửa lại cấu trúc chương, kết hợp một số chương kề nhau



Nhận xét

- Đơn giản, dễ tổ chức bảo vệ
 - Chương trình và vùng nhớ có một khóa bảo vệ
 - So sánh 2 khóa với nhau khi nạp chương trình
 - Giảm thời gian tìm kiếm
 - Phải sao các modul điều khiển ra làm nhiều bản và lưu ở nhiều nơi
 - Hệ số song song không thể vượt quá n
 - Bị phân đoạn bộ nhớ
 - Kích thước chương trình lớn hơn kích thước chương lớn nhất
 - Tổng bộ nhớ tự do còn lớn, nhưng không dùng để nạp các chương trình khác
- ⇒ Sửa lại cấu trúc chương, kết hợp một số chương kề nhau
- Áp dụng
 - Thường dùng cho quản lý các đĩa dung lượng lớn
 - Hệ điều hành OS/360 của IBM



2 Các chiến lược quản lý bộ nhớ

- Chiến lược phân chương cố định
- **Chiến lược phân chương động**
- Chiến lược phân đoạn
- Chiến lược phân trang
- Chiến lược kết hợp phân đoạn-phân trang



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*)

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*)

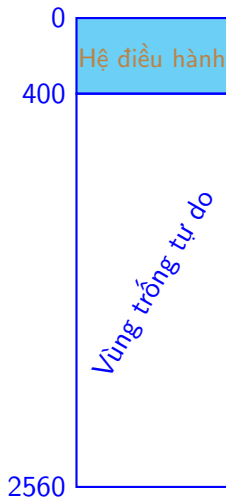
Nguyên tắc

Chỉ có một danh sách quản lý bộ nhớ tự do

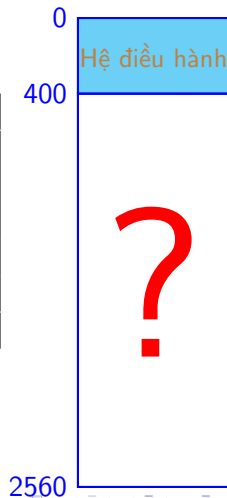
- Thời điểm ban đầu toàn bộ bộ nhớ là tự do với các tiến trình
⇒ vùng trống lớn nhất (*hole*)
- Khi một tiến trình yêu cầu bộ nhớ
 - Tìm trong DS vùng trống một phần tử đủ lớn cho yêu cầu
 - Nếu tìm thấy
 - Vùng trống được chia thành 2 phần
 - Một phần cung cấp theo yêu cầu
 - Một phần trả lại danh sách 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



Ví dụ



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		



Chiến lược lựa chọn vùng trống tự do

Có nhiều chiến lược lựa chọn vùng trống cho yêu cầu

First Fit : Vùng trống đầu tiên thỏa mãn

Best Fit : Vùng trống vừa vặn nhất

Worst Fit : Vùng trống kích thước lớn nhất



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



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

Buddy Allocation: Cung cấp nhớ

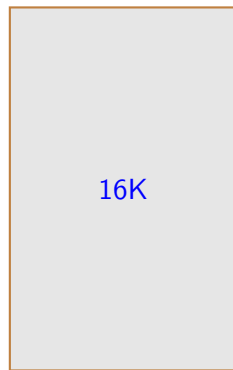
Nguyên tắc: Chia đôi liên tiếp vùng trống tự do cho tới khi thu được vùng trống nhỏ nhất thỏa mãn

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

- Vùng trống 16K Bytes
- Yêu cầu 735 Bytes



Buddy Allocation: Cung cấp nhớ

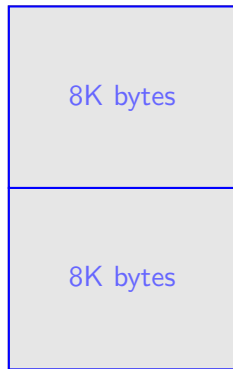
Nguyên tắc: Chia đôi liên tiếp vùng trống tự do cho tới khi thu được vùng trống nhỏ nhất thỏa mãn

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

- Vùng trống 16K Bytes
- Yêu cầu 735 Bytes



Buddy Allocation: Cung cấp nhớ

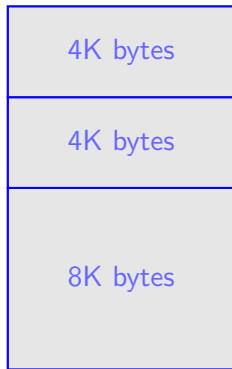
Nguyên tắc: Chia đôi liên tiếp vùng trống tự do cho tới khi thu được vùng trống nhỏ nhất thỏa mãn

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

- Vùng trống 16K Bytes
- Yêu cầu 735 Bytes



Buddy Allocation: Cung cấp nhớ

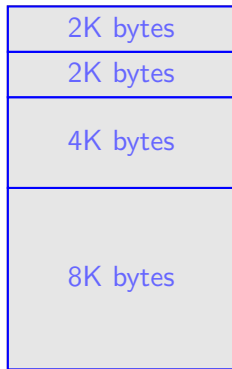
Nguyên tắc: Chia đôi liên tiếp vùng trống tự do cho tới khi thu được vùng trống nhỏ nhất thỏa mãn

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

- Vùng trống 16K Bytes
- Yêu cầu 735 Bytes



Buddy Allocation: Cung cấp nhớ

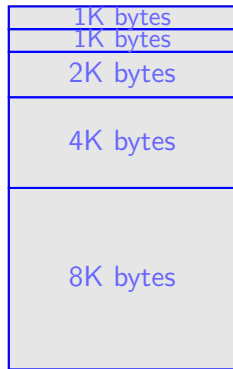
Nguyên tắc: Chia đôi liên tiếp vùng trống tự do cho tới khi thu được vùng trống nhỏ nhất thỏa mãn

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

- Vùng trống 16K Bytes
- Yêu cầu 735 Bytes



Buddy Allocation: Cung cấp nhớ

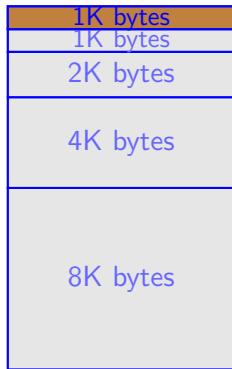
Nguyên tắc: Chia đôi liên tiếp vùng trống tự do cho tới khi thu được vùng trống nhỏ nhất thỏa mãn

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

- Vùng trống 16K Bytes
- Yêu cầu 735 Bytes



Buddy Allocation: Cung cấp nhớ nhanh

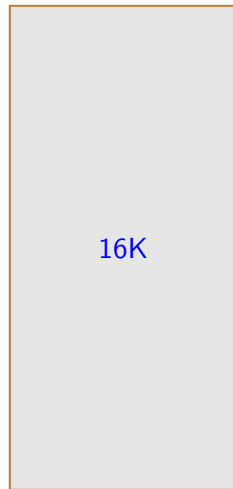
- Hệ thống duy trì các danh sách vùng trống kích thước $1, 2, \dots, 2^n$ bytes
- Với yêu cầu K , tìm phần tử nhỏ nhất kích thước lớn hơn K
- Nếu phần tử nhỏ nhất lớn hơn $2K$, chia liên tiếp tới khi được vùng nhỏ nhất kích thước lớn hơn K
- **Nhận xét:** Với bộ nhớ kích thước n , cần duyệt $\log_2 n$ danh sách \Rightarrow **Nhanh**



Buddy Allocation: Cung cấp nhớ nhanh

- Hệ thống duy trì các danh sách vùng trống kích thước $1, 2, \dots, 2^n$ bytes
- Với yêu cầu K , tìm phần tử nhỏ nhất kích thước lớn hơn K
- Nếu phần tử nhỏ nhất lớn hơn $2K$, chia liên tiếp tới khi được vùng nhỏ nhất kích thước lớn hơn K
- **Nhận xét:** Với bộ nhớ kích thước n , cần duyệt $\log_2 n$ danh sách \Rightarrow **Nhanh**

Ví dụ bộ nhớ 16K bytes

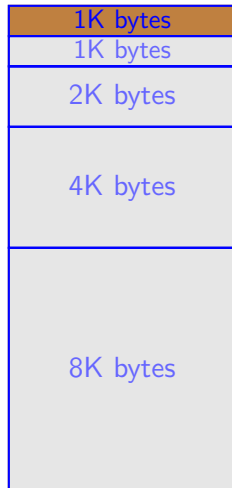


Buddy Allocation: Cung cấp nhớ nhanh

- Hệ thống duy trì các danh sách vùng trống kích thước $1, 2, \dots, 2^n$ bytes
- Với yêu cầu K , tìm phần tử nhỏ nhất kích thước lớn hơn K
- Nếu phần tử nhỏ nhất lớn hơn $2K$, chia liên tiếp tới khi được vùng nhỏ nhất kích thước lớn hơn K
- **Nhận xét:** Với bộ nhớ kích thước n , cần duyệt $\log_2 n$ danh sách \Rightarrow **Nhanh**

Ví dụ bộ nhớ 16K bytes

- Yêu cầu 735 bytes

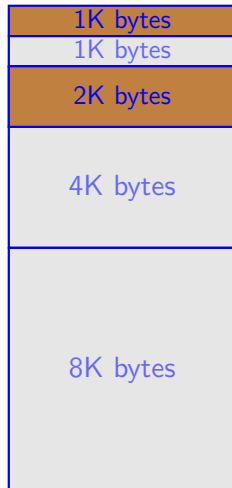


Buddy Allocation: Cung cấp nhớ nhanh

- Hệ thống duy trì các danh sách vùng trống kích thước $1, 2, \dots, 2^n$ bytes
- Với yêu cầu K , tìm phần tử nhỏ nhất kích thước lớn hơn K
- Nếu phần tử nhỏ nhất lớn hơn $2K$, chia liên tiếp tới khi được vùng nhỏ nhất kích thước lớn hơn K
- **Nhận xét:** Với bộ nhớ kích thước n , cần duyệt $\log_2 n$ danh sách \Rightarrow **Nhanh**

Ví dụ bộ nhớ 16K bytes

- Yêu cầu 735 bytes
- Yêu cầu 1205 bytes



Buddy Allocation: Cung cấp nhớ nhanh

- Hệ thống duy trì các danh sách vùng trống kích thước $1, 2, \dots, 2^n$ bytes
- Với yêu cầu K , tìm phần tử nhỏ nhất kích thước lớn hơn K
- Nếu phần tử nhỏ nhất lớn hơn $2K$, chia liên tiếp tới khi được vùng nhỏ nhất kích thước lớn hơn K
- **Nhận xét:** Với bộ nhớ kích thước n , cần duyệt $\log_2 n$ danh sách \Rightarrow **Nhanh**

Ví dụ bộ nhớ 16K bytes

- Yêu cầu 735 bytes
- Yêu cầu 1205 bytes
- Yêu cầu 2010 bytes



Buddy Allocation : Thu hồi vùng nhớ

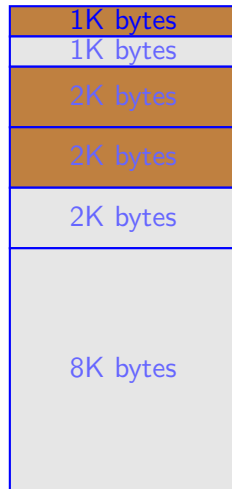
- Có thể kết hợp 2 vùng kề nhau có cùng kích thước
- Tiếp tục kết hợp liên tiếp cho tới khi tạo ra vùng trống lớn nhất có thể



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

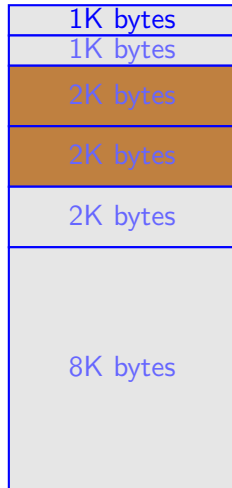


Buddy Allocation : Thu hồi vùng nhớ

- 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)

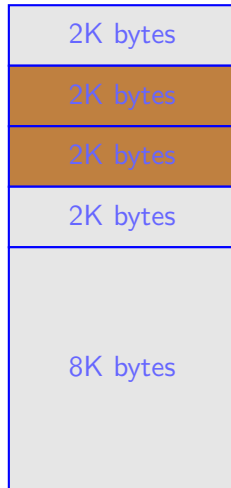


Buddy Allocation : Thu hồi vùng nhớ

- 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

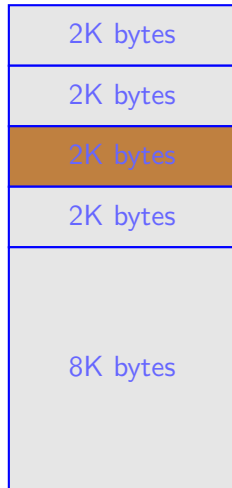


Buddy Allocation : Thu hồi vùng nhớ

- 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)

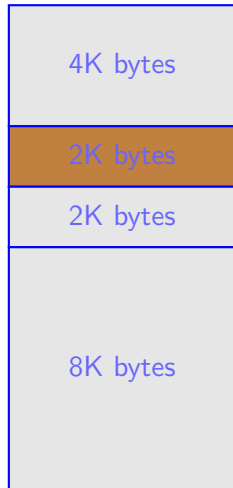


Buddy Allocation : Thu hồi vùng nhớ

- 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

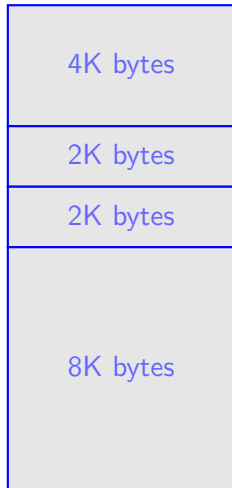


Buddy Allocation : Thu hồi vùng nhớ

- 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)

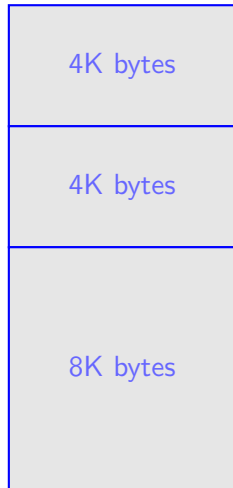


Buddy Allocation : Thu hồi vùng nhớ

- 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

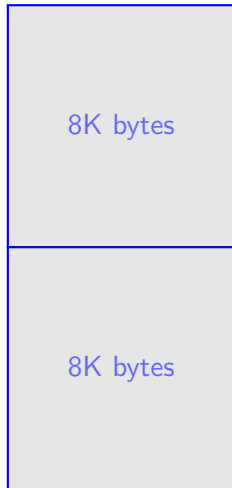


Buddy Allocation : Thu hồi vùng nhớ

- 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
 - Kết hợp 2 vùng 4K thành vùng 8K



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
 - Kết hợp 2 vùng 4K thành vùng 8K
 - Kết hợp 2 vùng 8K thành vùng 16K



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. \Rightarrow Cần phải bố trí lại bộ nhớ



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. \Rightarrow 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 \Rightarrow vùng trống lớn nhất
 - Dịch chuyển để tạo ra ngay lập tức một vùng trống vừa vặn

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. \Rightarrow 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 \Rightarrow vùng trống lớn nhất
 - Dịch chuyển để tạo ra ngay lập tức một vùng trống vừa vặn
- Phương pháp trao 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



Nhận xét

- Không phải sao lưu modul điều khiển ra nhiều nơi



Nhận xét

- Không phải sao lưu modul điều khiển ra nhiều nơi
- Tăng/giảm hệ số song song tùy theo số lượng và kích thước chương trình



Nhận xét

- Không phải sao lưu modul điều khiển ra nhiều nơi
- Tăng/giảm hệ số song song tùy theo số lượng và kích thước chương trình
- Không thực hiện được chương trình có kích thước lớn hơn kích thước bộ nhớ vật lý



Nhận xét

- Không phải sao lưu modul điều khiển ra nhiều nơi
- Tăng/giảm hệ số song song tùy theo số lượng và kích thước chương trình
- Không thực hiện được chương trình có kích thước lớn hơn kích thước bộ nhớ vật lý
- Gây ra hiện tượng rác
 - Bộ nhớ không được sử dụng, nhưng cũng không nằm trong DS quản lý bộ nhớ tự do
 - Do lỗi hệ điều hành
 - Do phần mềm phá hoại



Nhận xét

- Không phải sao lưu modul điều khiển ra nhiều nơi
- Tăng/giảm hệ số song song tùy theo số lượng và kích thước chương trình
- Không thực hiện được chương trình có kích thước lớn hơn kích thước bộ nhớ vật lý
- Gây ra hiện tượng rác
 - Bộ nhớ không được sử dụng, nhưng cũng không nằm trong DS quản lý bộ nhớ tự do
 - Do lỗi hệ điều hành
 - Do phần mềm phá hoại
- Gây ra hiện tượng phân đoạn ngoài
 - Vùng nhớ tự do được quản lý đầy đủ, nhưng nằm rải rác nên không sử dụng được



Nhận xét

- Không phải sao lưu modul điều khiển ra nhiều nơi
- Tăng/giảm hệ số song song tùy theo số lượng và kích thước chương trình
- Không thực hiện được chương trình có kích thước lớn hơn kích thước bộ nhớ vật lý
- Gây ra hiện tượng rác
 - Bộ nhớ không được sử dụng, nhưng cũng không nằm trong DS quản lý bộ nhớ tự do
 - Do lỗi hệ điều hành
 - Do phần mềm phá hoại
- Gây ra hiện tượng phân đoạn ngoài
 - Vùng nhớ tự do được quản lý đầy đủ, nhưng nằm rải rác nên không sử dụng được
- Gây ra hiện tượng phân đoạn trong
 - Vùng nhớ dành cho chương trình nhưng không được chương trình sử dụng tới



2 Các chiến lược quản lý bộ nhớ

- Chiến lược phân chương cố định
- Chiến lược phân chương động
- **Chiến lược phân đoạn**
- Chiến lược phân trang
- Chiến lược kết hợp phân đoạn-phân trang



Chương trình

- Chương trình thường gồm các modul
 - Một chương trình chính (*main program*)
 - Tập các chương trình con
 - Các biến, các cấu trúc dữ liệu,...



Chương trình

- Chương trình thường gồm các modul
 - Một chương trình chính (*main program*)
 - Tập các chương trình con
 - Các biến, các cấu trúc dữ liệu,...
- Các modul, đối tượng trong c/trình được xác định bằng tên
 - Hàm `sqrt()`, thủ tục `printf()` ...
 - `x`, `y`, `counter`, `Buffer`...



Chương trình

- Chương trình thường gồm các modul
 - Một chương trình chính (*main program*)
 - Tập các chương trình con
 - Các biến, các cấu trúc dữ liệu,...
- Các modul, đối tượng trong c/trình được xác định bằng tên
 - Hàm `sqrt()`, thủ tục `printf()` ...
 - `x`, `y`, `counter`, `Buffer`...
- Các p/tử trong modul được x/định theo độ lệch với vị trí đầu
 - Câu lệnh thứ 10 của hàm `sqrt()`...
 - Phần tử thứ 2 của mảng `Buffer`...



Chương trình

- Chương trình thường gồm các modul
 - Một chương trình chính (*main program*)
 - Tập các chương trình con
 - Các biến, các cấu trúc dữ liệu,...
- Các modul, đối tượng trong c/trình được xác định bằng tên
 - Hàm `sqrt()`, thủ tục `printf()` ...
 - `x`, `y`, `counter`, `Buffer`...
- Các p/tử trong modul được x/định theo độ lệch với vị trí đầu
 - Câu lệnh thứ 10 của hàm `sqrt()`...
 - Phần tử thứ 2 của mảng `Buffer`...

Chương trình được tổ chức như thế nào trong bộ nhớ?

- *Stack* nằm trên hay *Data* nằm trên trong bộ nhớ?
- Địa chỉ vật lý các đối tượng ...?

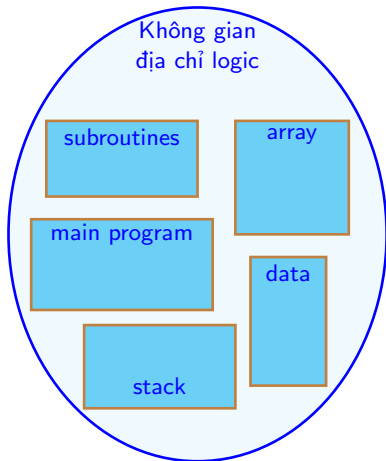
⇒ **Không quan tâm**



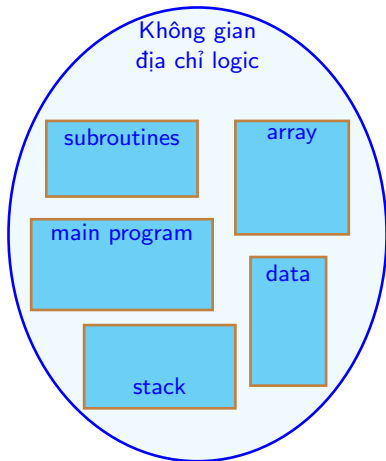
Quan điểm người dùng

Khi đưa c/trình vào bộ nhớ để thực hiện

- C/trình gồm nhiều đoạn khác nhau
 - Mỗi đoạn là một khối logic, ứng với một modul
 - Mã lệnh: `main()`, thủ tục, hàm...
 - Dữ liệu: Đối tượng toàn cục, cục bộ
 - Các đoạn khác: stack, mảng...



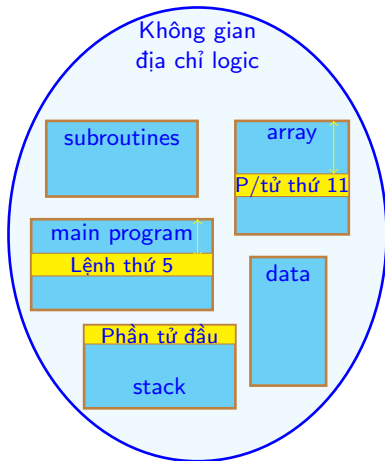
Quan điểm người dùng



Khi đưa c/trình vào bộ nhớ để thực hiện

- C/trình gồm nhiều đoạn khác nhau
 - Mỗi đoạn là một khối logic, ứng với một modul
 - Mã lệnh: `main()`, thủ tục, hàm...
 - Dữ liệu: Đối tượng toàn cục, cục bộ
 - Các đoạn khác: stack, mảng...
- Mỗi đoạn chiếm một vùng liên tục
 - Có vị trí bắt đầu và kích thước
 - Có thể nằm tại bất cứ đâu trong bộ nhớ

Quan điểm người dùng



Khi đưa c/trình vào bộ nhớ để thực hiện

- C/trình gồm nhiều đoạn khác nhau
 - Mỗi đoạn là một khối logic, ứng với một modul
 - Mã lệnh: `main()`, thủ tục, hàm...
 - Dữ liệu: Đối tượng toàn cục, cục bộ
 - Các đoạn khác: stack, mảng...
- Mỗi đoạn chiếm một vùng liên tục
 - Có vị trí bắt đầu và kích thước
 - Có thể nằm tại bất cứ đâu trong bộ nhớ
- Đối tượng trong đoạn được xác định bởi vị trí tương đối so với đầu đoạn
 - Lệnh thứ 5 của chương trình chính
 - Phần tử đầu tiên của stack...

Vị trí các đối tượng trong bộ nhớ?



Không gian địa chỉ logic

subroutine
Đoạn 2

array
Đoạn 3

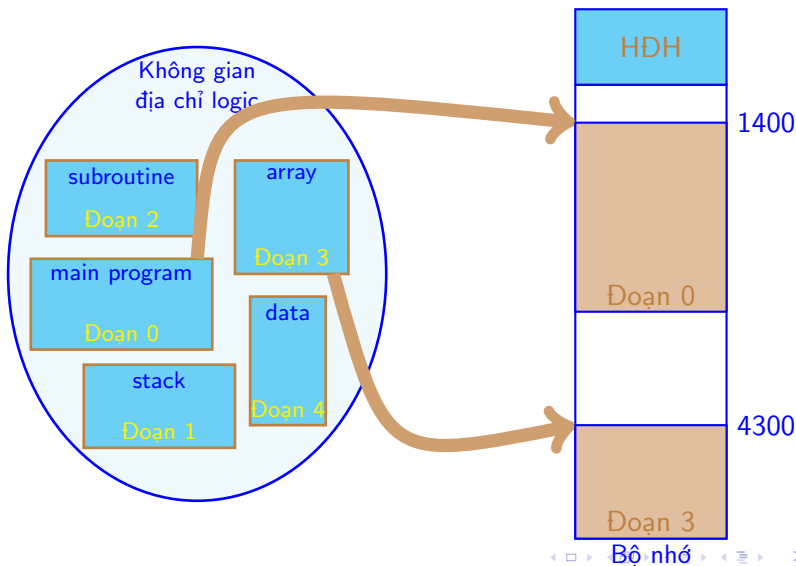
main program
Đoạn 0

data
Đoạn 4

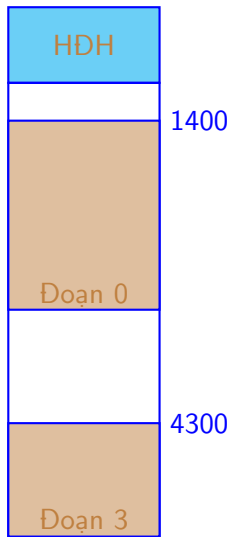
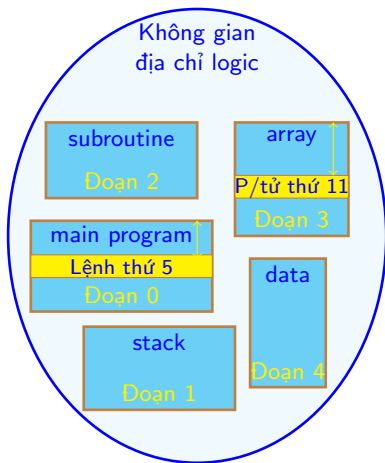
stack
Đoạn 1



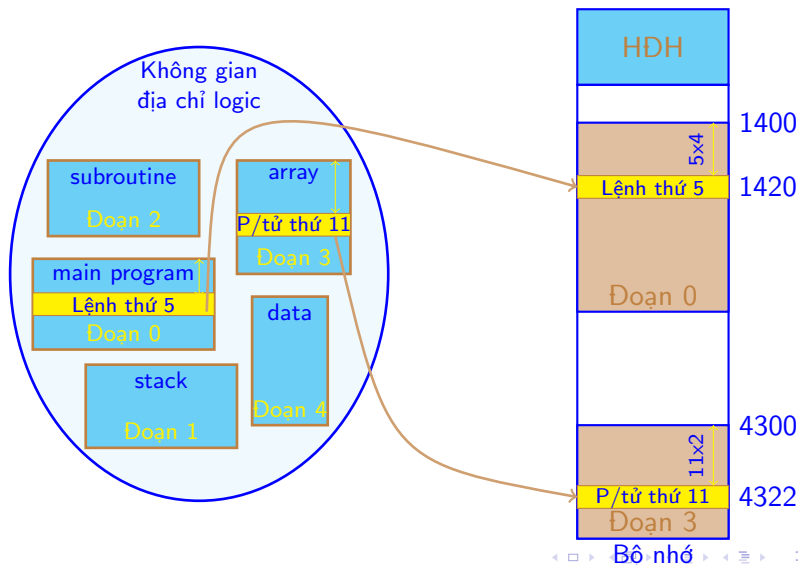
Ví dụ



Ví dụ



Ví dụ



Cấu trúc phân đoạn

- Chương trình là tập hợp các đoạn (*modul, segment*)
 - Tên đoạn (*số hiệu đoạn*), độ dài của đoạn
 - Mỗi đoạn có thể được biên tập riêng.



Cấu trúc phân đoạn

- Chương trình là tập hợp các đoạn (*modul, segment*)
 - Tên đoạn (*số hiệu đoạn*), độ dài của đoạn
 - Mỗi đoạn có thể được biên tập riêng.
- Dịch và biên tập chương trình tạo ra bảng quản lý đoạn (**SCB**: Segment Control Block)
 - Mỗi phần tử của bảng ứng với một đoạn của chương trình

	Mark	Address	Length
0			
⋮
<i>n</i>

- Dấu hiệu (**Mark (0/1)**): Đoạn đã tồn tại trong bộ nhớ
- Địa chỉ (**Address**): Vị trí cơ sở (*base*) của đoạn trong bộ nhớ
- Độ dài (**Length**): Độ dài của đoạn

Cấu trúc phân đoạn

- Chương trình là tập hợp các đoạn (*modul, segment*)
 - Tên đoạn (*số hiệu đoạn*), độ dài của đoạn
 - Mỗi đoạn có thể được biên tập riêng.
- Dịch và biên tập chương trình tạo ra bảng quản lý đoạn (**SCB**: Segment Control Block)
 - Mỗi phần tử của bảng ứng với một đoạn của chương trình

	Mark	Address	Length
0			
⋮
<i>n</i>

- Dấu hiệu (**Mark (0/1)**): Đoạn đã tồn tại trong bộ nhớ
 - Địa chỉ (**Address**): Vị trí cơ sở (*base*) của đoạn trong bộ nhớ
 - Độ dài (**Length**): Độ dài của đoạn
- Địa chỉ truy nhập: tên (*số hiệu*) đoạn và độ lệch trong đoạn

Cấu trúc phân đoạn

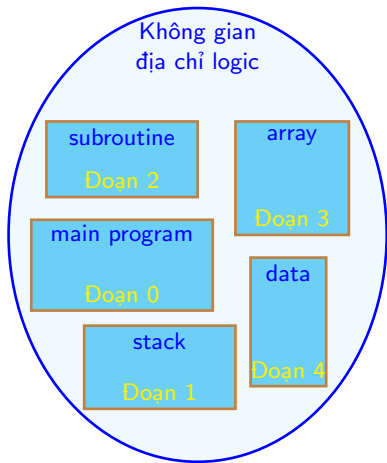
- Chương trình là tập hợp các đoạn (*modul, segment*)
 - Tên đoạn (*số hiệu đoạn*), độ dài của đoạn
 - Mỗi đoạn có thể được biên tập riêng.
- Dịch và biên tập chương trình tạo ra bảng quản lý đoạn (**SCB**: Segment Control Block)
 - Mỗi phần tử của bảng ứng với một đoạn của chương trình

	Mark	Address	Length
0			
⋮
<i>n</i>

- Dấu hiệu (**Mark (0/1)**): Đoạn đã tồn tại trong bộ nhớ
 - Địa chỉ (**Address**): Vị trí cơ sở (*base*) của đoạn trong bộ nhớ
 - Độ dài (**Length**): Độ dài của đoạn
- Địa chỉ truy nhập: tên (*số hiệu*) đoạn và độ lệch trong đoạn

Vấn đề: Chuyển đổi từ địa chỉ 2 chiều \Rightarrow địa chỉ một chiều

Ví dụ

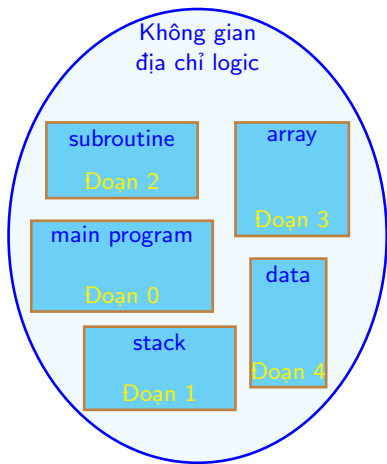


M	A	L
0	-	1000
0	-	400
0	-	400
0	-	1100
0	-	1000
SCB		

HDH

Bộ nhớ

Ví dụ

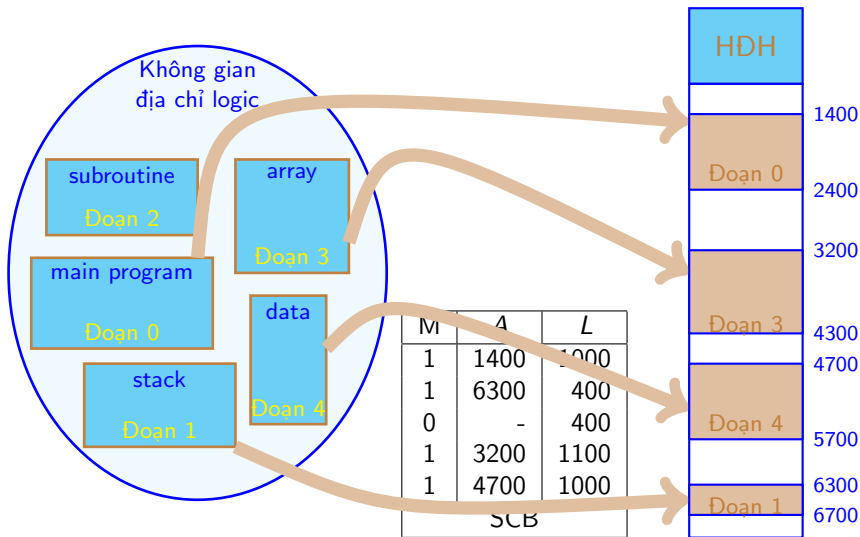


M	A	L
1	1400	1000
1	6300	400
0	-	400
1	3200	1100
1	4700	1000
SCB		

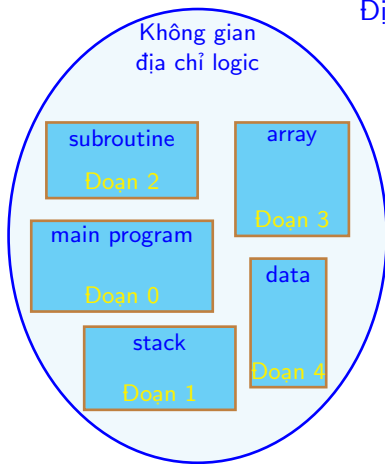
HDH

Bộ nhớ

Ví dụ

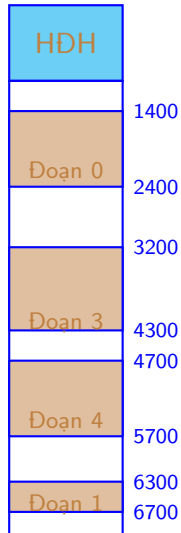


Ví dụ



Địa chỉ $\langle 3,345 \rangle = ?$

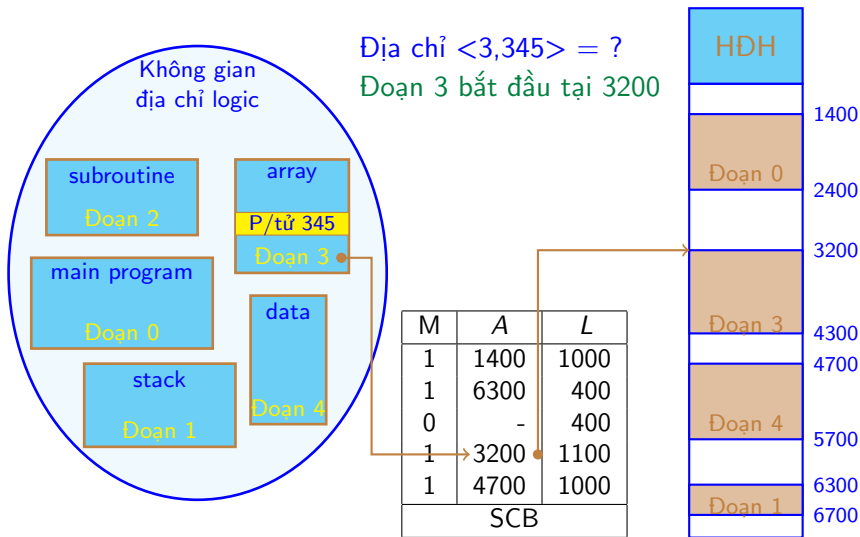
M	A	L
1	1400	1000
1	6300	400
0	-	400
1	3200	1100
1	4700	1000
SCB		



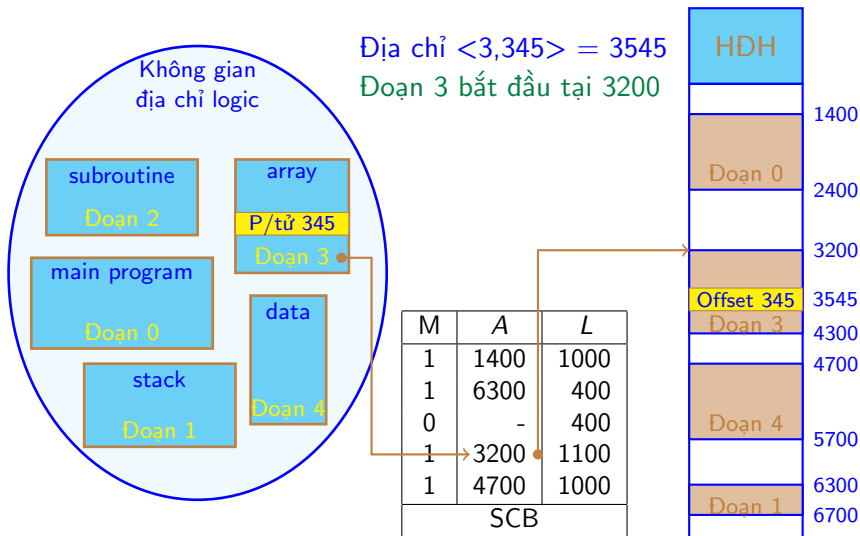
Bộ nhớ



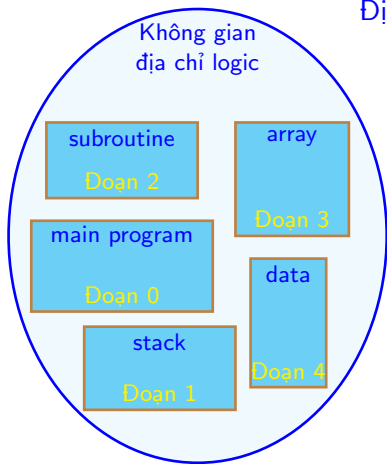
Ví dụ



Ví dụ

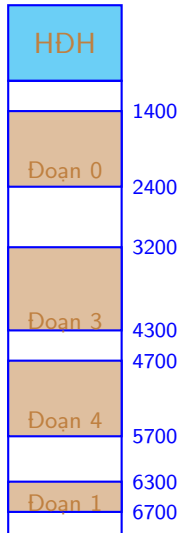


Ví dụ



Địa chỉ $\langle 1,240 \rangle = ?$

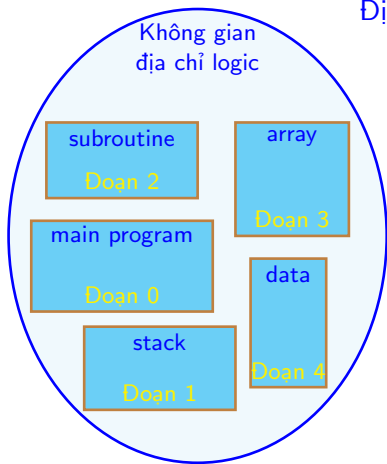
M	A	L
1	1400	1000
1	6300	400
0	-	400
1	3200	1100
1	4700	1000
SCB		



Bộ nhớ

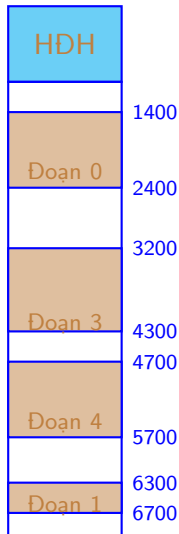


Ví dụ



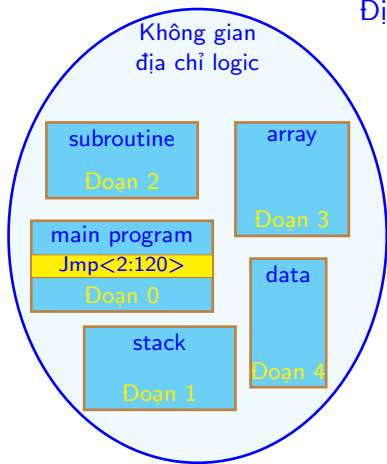
Địa chỉ $\langle 1,240 \rangle = 6540$

M	A	L
1	1400	1000
1	6300	400
0	-	400
1	3200	1100
1	4700	1000
SCB		



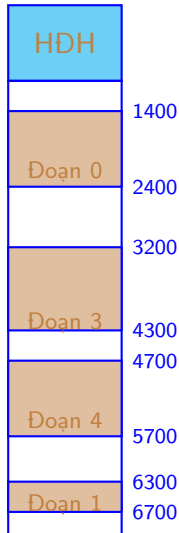
Bộ nhớ

Ví dụ



Địa chỉ <2,120> = ?

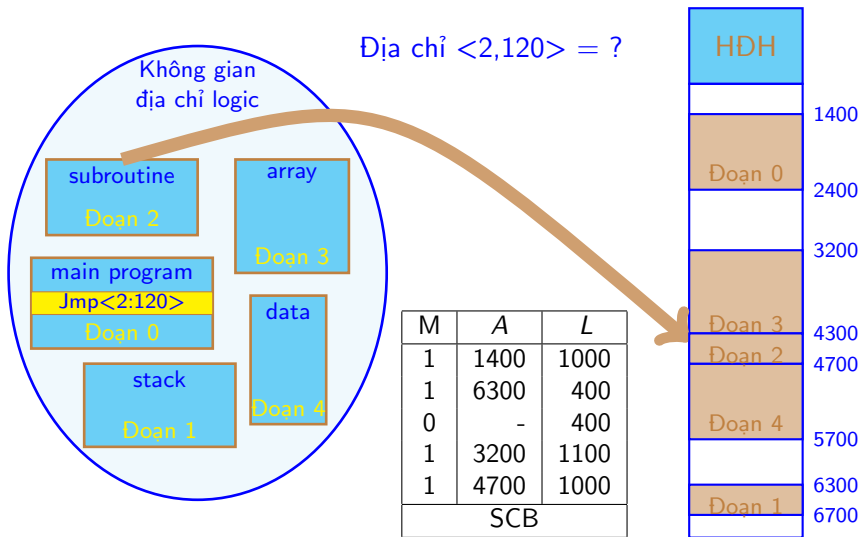
M	A	L
1	1400	1000
1	6300	400
0	-	400
1	3200	1100
1	4700	1000
SCB		



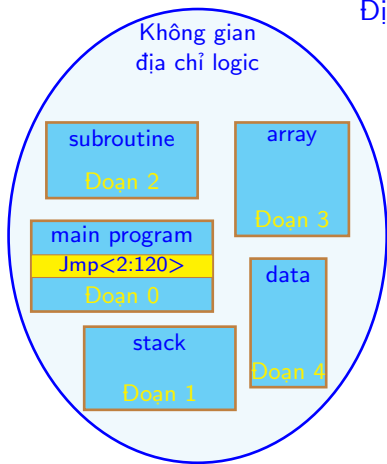
Bộ nhớ



Ví dụ

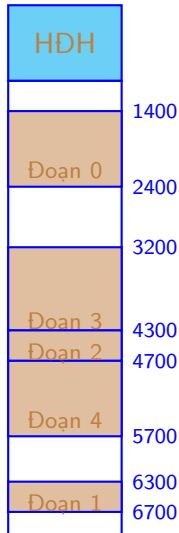


Ví dụ



Địa chỉ $\langle 2, 120 \rangle = ?$

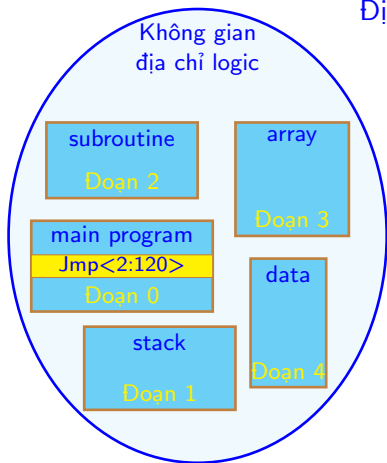
M	A	L
1	1400	1000
1	6300	400
1	4300	400
1	3200	1100
1	4700	1000
SCB		



Bộ nhớ

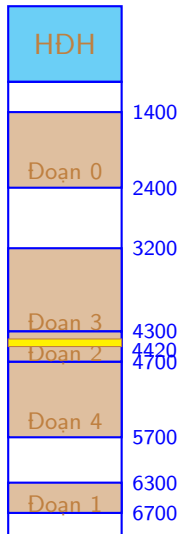


Ví dụ



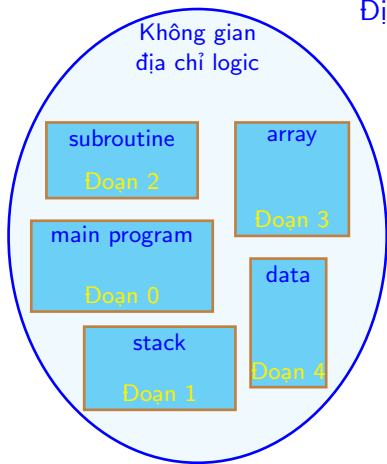
Địa chỉ $\langle 2,120 \rangle = 4420$

M	A	L
1	1400	1000
1	6300	400
1	4300	400
1	3200	1100
1	4700	1000
SCB		



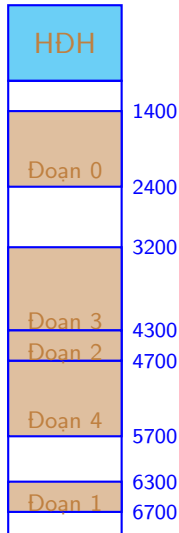
Bộ nhớ

Ví dụ



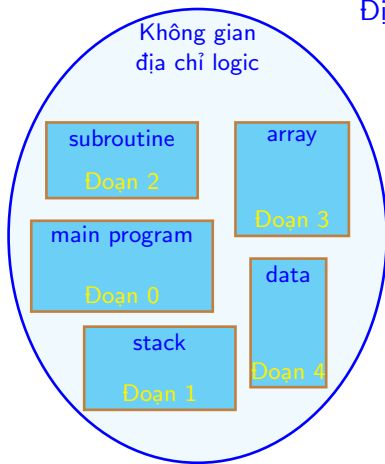
Địa chỉ $\langle 2,450 \rangle = ?$

M	A	L
1	1400	1000
1	6300	400
1	4300	400
1	3200	1100
1	4700	1000
SCB		



Bộ nhớ

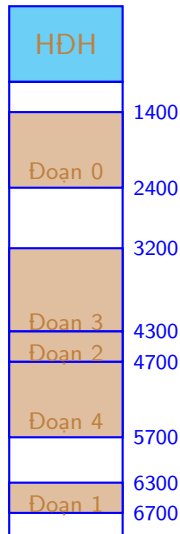
Ví dụ



Địa chỉ $\langle 2,450 \rangle = ?$

Lỗi truy nhập!

M	A	L
1	1400	1000
1	6300	400
1	4300	400
1	3200	1100
1	4700	1000
SCB		



Bộ nhớ

Chuyển đổi địa chỉ

■ Khi thực hiện chương trình

- Bảng quản lý đoạn được nạp vào bộ nhớ
 - STBR (*Segment-table base register*): Vị trí SCB trong bộ nhớ
 - STLR (*Segment-table length register*): Số phần tử của SCB



Chuyển đổi địa chỉ

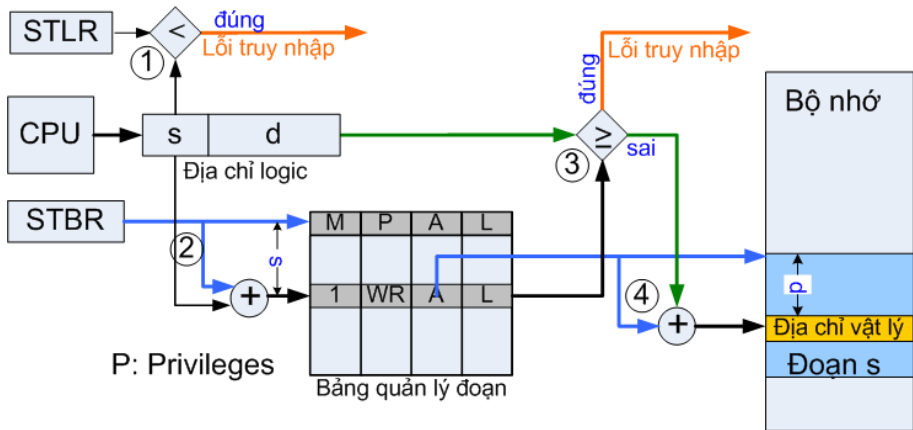
■ Khi thực hiện chương trình

- Bảng quản lý đoạn được nạp vào bộ nhớ
 - STBR (*Segment-table base register*): Vị trí SCB trong bộ nhớ
 - STLRL (*Segment-table length register*): Số phần tử của SCB

■ Truy nhập tới địa chỉ logic $\langle s, d \rangle$

- 1 $s \geq STLRL$: Lỗi
- 2 $STBR + s$: Vị trí phần tử s trong SCB
- 3 Kiểm tra trường dấu hiệu M của phần tử SCB_s
 - $M = 0$: Đoạn s chưa tồn tại trong bộ nhớ \Rightarrow Lỗi truy nhập \Rightarrow Hệ điều hành phải nạp đoạn
 - 1 Xin vùng nhớ có kích thước được ghi trong trường L
 - 2 Tìm modul tương ứng ở bộ nhớ ngoài và nạp và định vị vào vùng nhớ xin được
 - 3 Sửa lại trường địa chỉ A và trường dấu hiệu $M (M = 1)$
 - 4 Truy nhập bộ nhớ như trường hợp không gặp lỗi truy nhập
 - $M = 1$:Đoạn s đã tồn tại trong bộ nhớ
 - 1 $d \geq L_s$: Lỗi truy nhập (vượt quá kích thước đoạn)
 - 2 $d + A_s$: Địa chỉ vật lý cần tìm

Chuyển đổi địa chỉ: Sơ đồ truy nhập



Nhận xét: ưu điểm

- Sơ đồ nạp modul không cần sự tham gia của người sử dụng



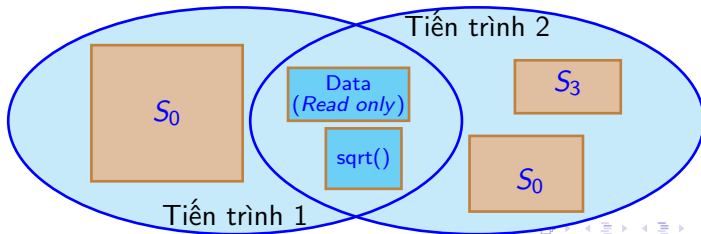
Nhận xét: ưu điểm

- Sơ đồ nạp modul không cần sự tham gia của người sử dụng
- Dễ dàng thực hiện nhiệm vụ bảo vệ đoạn
 - Kiểm tra lỗi truy nhập bộ nhớ
 - Địa chỉ không hợp lệ :vượt quá kích thước đoạn
 - Kiểm tra tính chất truy nhập
 - Đoạn mã: chỉ đọc
 - Viết vào đoạn mã: lỗi truy nhập
 - Kiểm tra quyền truy nhập modul
 - Thêm trường quyền truy nhập(*user/system*) vào SCB

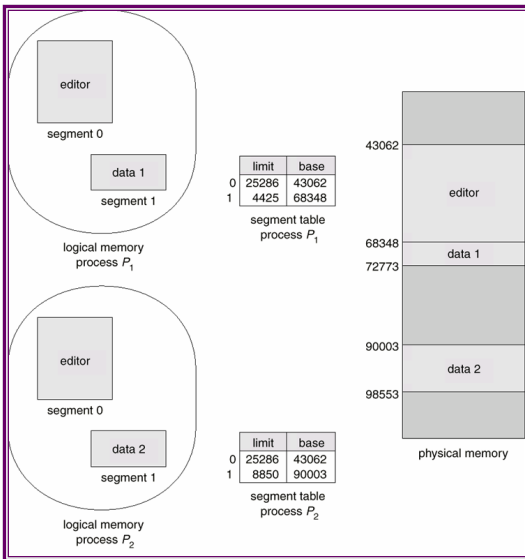


Nhận xét: ưu điểm

- Sơ đồ nạp modul không cần sự tham gia của người sử dụng
- Dễ dàng thực hiện nhiệm vụ bảo vệ đoạn
 - Kiểm tra lỗi truy nhập bộ nhớ
 - Địa chỉ không hợp lệ :vượt quá kích thước đoạn
 - Kiểm tra tính chất truy nhập
 - Đoạn mã: chỉ đọc
 - Viết vào đoạn mã: lỗi truy nhập
 - Kiểm tra quyền truy nhập modul
 - Thêm trường quyền truy nhập(*user/system*) vào SCB
- Cho phép sử dụng chung đoạn (*VD Soạn thảo văn bản*)



Dùng chung đoạn : Vấn đề chính



- Đoạn dùng chung phải cùng số hiệu trong SCB
 - Call (0, 120) ?
 - Read (1, 245) ?
- Giải quyết bằng cách truy nhập gián tiếp
 - JMP + 08
 - Thanh ghi đoạn chứa số hiệu đoạn (ES:BX)

Nhận xét : Nhược điểm

- Hiệu quả sử dụng phụ thuộc vào cấu trúc chương trình



Nhận xét : Nhược điểm

- Hiệu quả sử dụng phụ thuộc vào cấu trúc chương trình
- Bị phân mảnh bộ nhớ
 - Phân phối vùng nhớ theo các chiến lược first fit /best fit...
 - Cần phải bố trí lại bộ nhớ (*dịch chuyển, swapping*)
 - Có thể dựa vào bảng SCB
 - $M \leftarrow 0$: Đoạn chưa được nạp vào
 - Vùng nhớ được xác định bởi A và L được trả về DS tự do
 - Vấn đề lựa chọn modul cần đưa ra
 - Đưa ra modul tồn tại lâu nhất
 - Đưa ra modul có lần sử dụng cuối cách xa nhất
 - Đưa ra modul có tần xuất sử dụng thấp nhất
 - ⇒ Cần phương tiện ghi lại số lần và thời điểm truy nhập đoạn

Giải pháp: phân phối bộ nhớ theo các đoạn bằng nhau (*page*)?



2 Các chiến lược quản lý bộ nhớ

- Chiến lược phân chương cố định
- Chiến lược phân chương động
- Chiến lược phân đoạn
- **Chiến lược phân trang**
- Chiến lược kết hợp phân đoạn-phân trang



Nguyên tắc

- Bộ nhớ vật lý được chia thành từng khối có kích thước bằng nhau: *trang vật lý (frames)*
 - Trang vật lý được đánh số $0, 1, 2, \dots$: địa chỉ vật lý của trang
 - Trang được dùng làm đơn vị phân phối nhớ



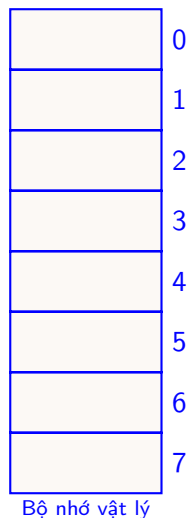
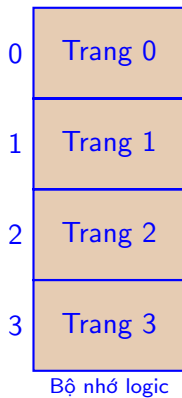
Nguyên tắc

- Bộ nhớ vật lý được chia thành từng khối có kích thước bằng nhau: *trang vật lý (frames)*
 - Trang vật lý được đánh số $0, 1, 2, \dots$: địa chỉ vật lý của trang
 - Trang được dùng làm đơn vị phân phối nhớ
- Bộ nhớ logic (*chương trình*) được chia thành từng trang có kích thước bằng trang vật lý: *trang logic (pages)*

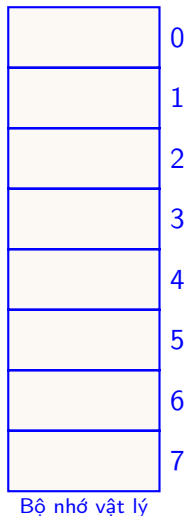
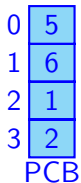
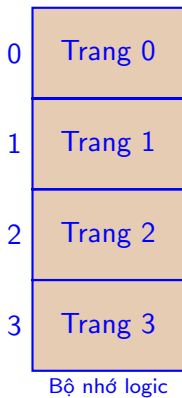
Nguyên tắc

- Bộ nhớ vật lý được chia thành từng khối có kích thước bằng nhau: *trang vật lý (frames)*
 - Trang vật lý được đánh số $0, 1, 2, \dots$: địa chỉ vật lý của trang
 - Trang được dùng làm đơn vị phân phối nhớ
- Bộ nhớ logic (*chương trình*) được chia thành từng trang có kích thước bằng trang vật lý: *trang logic (pages)*
- Khi thực hiện chương trình
 - Nạp trang logic (*từ bộ nhớ ngoài*) vào trang vật lý
 - Xây dựng một bảng quản lý trang (*PCB: Page Control Block*) dùng để xác định mối quan hệ giữa trang vật lý và trang logic
 - Mỗi phần tử của PCB ứng với một trang chương trình
 - Cho biết biết trang vật lý chứa trang logic tương ứng
 - Ví dụ $PCB[8] = 4 \Rightarrow ?$
 - Địa chỉ truy nhập: $\langle \text{Số hiệu trang, độ lệch trong trang} \rangle$
 - Số hiệu trang (**p**) : Chỉ số trong PCB để tìm đ/chỉ cơ sở trang
 - Độ lệch trong trang (**d**): Kết hợp địa chỉ cơ sở của trang để tìm ra đ/chỉ vật lý

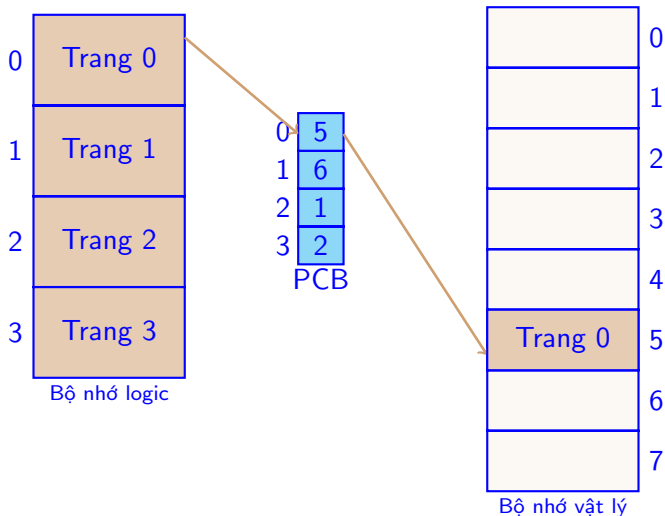
Ví dụ



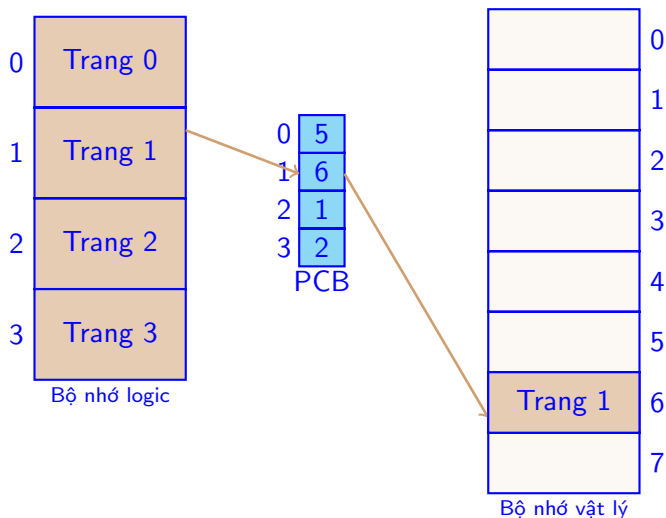
Ví dụ



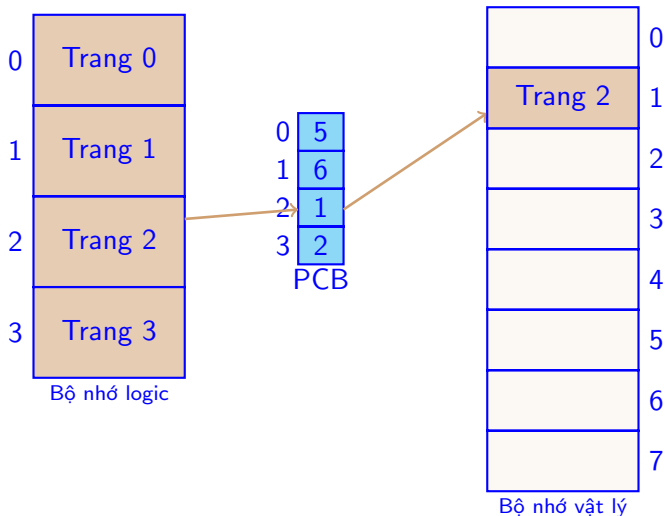
Ví dụ



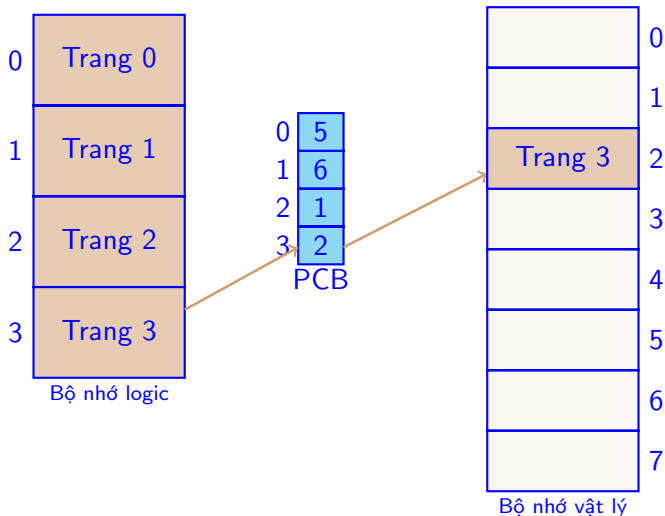
Ví dụ



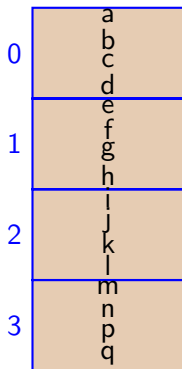
Ví dụ



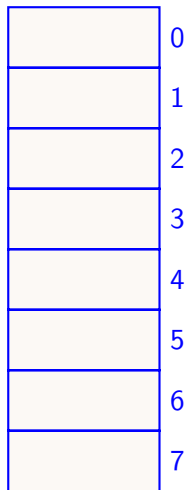
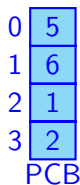
Ví dụ



Ví dụ

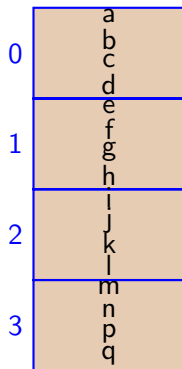


Bộ nhớ logic

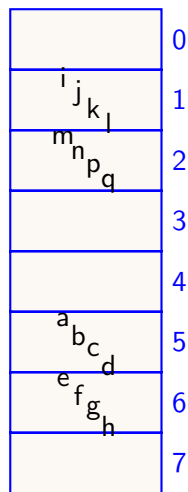
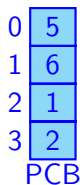


Bộ nhớ vật lý

Ví dụ

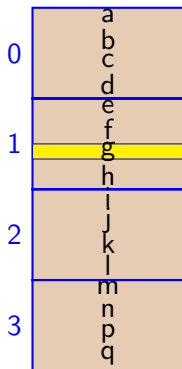


Bộ nhớ logic



Bộ nhớ vật lý

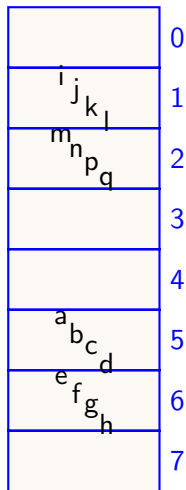
Ví dụ



Bộ nhớ logic

0	5
1	6
2	1
3	2

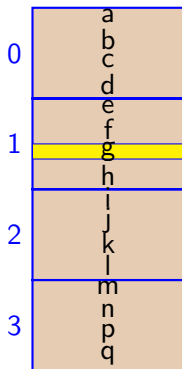
PCB



Bộ nhớ vật lý

Địa chỉ trang 1 độ lệch 2 = ?

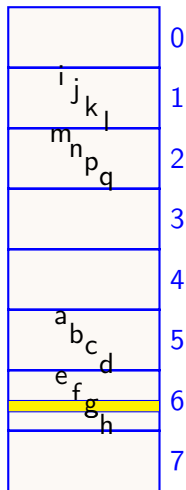
Ví dụ



Bộ nhớ logic

0	5
1	6
2	1
3	2

PCB



Bộ nhớ vật lý

Địa chỉ trang 1 độ lệch 2 = ?

Địa chỉ $\langle 1, 2 \rangle = 6 \cdot 4 + 2 = 26$ (62₄)

Ghi chú

- Dung lượng trang luôn là lũy thừa của 2
 - Cho phép ghép giữa số hiệu trang vật lý và độ lệch trong trang

Số hiệu trang	độ lệch
$n - k$	k



Ghi chú

- Dung lượng trang luôn là lũy thừa của 2
 - Cho phép ghép giữa số hiệu trang vật lý và độ lệch trong trang

Số hiệu trang	độ lệch
$n - k$	k

- Không cần thiết nạp toàn bộ trang logic vào
 - Số trang vật lý phụ thuộc k /thước bộ nhớ, số trang logic tùy ý
 - PCB cần trường dấu hiệu (*Mark*) cho biết trang đã được nạp vào bộ nhớ chưa
 - $M = 0$ Trang chưa tồn tại
 - $M = 1$ Trang đã được đưa vào bộ nhớ vật lý



Ghi chú

- Dung lượng trang luôn là lũy thừa của 2
 - Cho phép ghép giữa số hiệu trang vật lý và độ lệch trong trang

Số hiệu trang	độ lệch
$n - k$	k

- Không cần thiết nạp toàn bộ trang logic vào
 - Số trang vật lý phụ thuộc k /thước bộ nhớ, số trang logic tùy ý
 - PCB cần trường dấu hiệu (*Mark*) cho biết trang đã được nạp vào bộ nhớ chưa
 - $M = 0$ Trang chưa tồn tại
 - $M = 1$ Trang đã được đưa vào bộ nhớ vật lý
- Phân biệt chiến lược phân trang - phân đoạn
 - Chiến lược phân đoạn
 - Các modul phụ thuộc cấu trúc logic của chương trình
 - Chiến lược phân trang
 - Các khối có kích thước độc lập kích thước chương trình
 - Kích thước khối phụ thuộc phần cứng (VD: $2^9 \rightarrow 2^{13}$ bytes)



Thực hiện chương trình

- Nạp chương trình vào bộ nhớ
 - Nếu đủ trang vật lý tự do \Rightarrow nạp toàn bộ
 - Nếu không đủ trang vật lý tự do \Rightarrow nạp toàn bộ



Thực hiện chương trình

- Nạp chương trình vào bộ nhớ
 - Nếu đủ trang vật lý tự do \Rightarrow nạp toàn bộ
 - Nếu không đủ trang vật lý tự do \Rightarrow nạp toàn bộ
- Xây dựng bảng quản lý trang và luôn giữ trong bộ nhớ
 - PTBR (*Page-table base register*) trỏ tới PCB.
 - PRLR (*Page-table length register*) kích thước PCB.

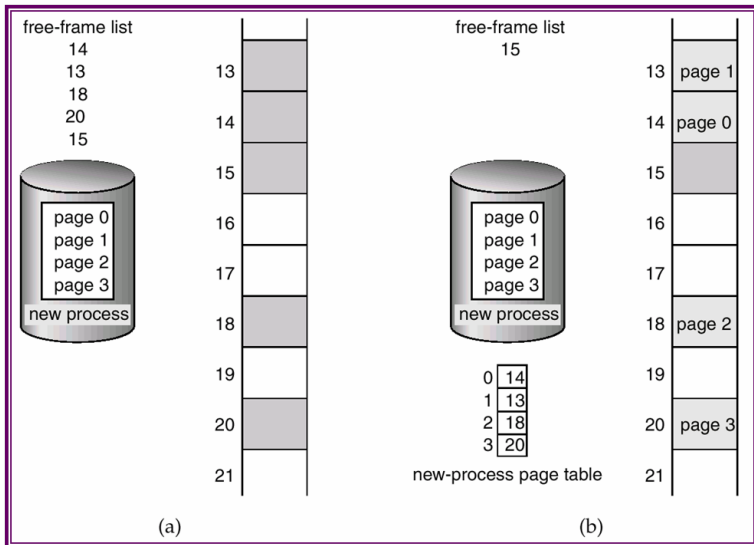


Thực hiện chương trình

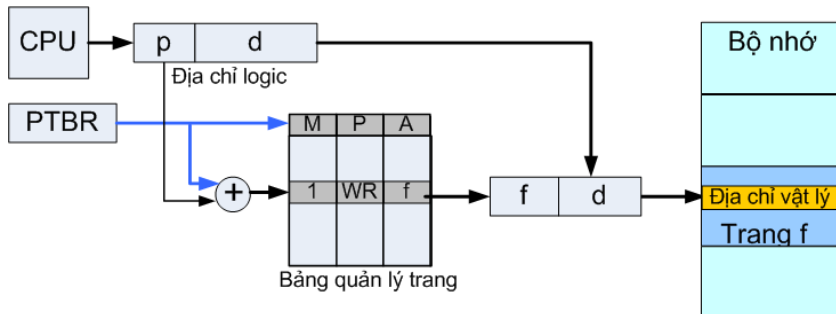
- Nạp chương trình vào bộ nhớ
 - Nếu đủ trang vật lý tự do \Rightarrow nạp toàn bộ
 - Nếu không đủ trang vật lý tự do \Rightarrow nạp toàn bộ
- Xây dựng bảng quản lý trang và luôn giữ trong bộ nhớ
 - PTBR (*Page-table base register*) trỏ tới PCB.
 - PRLR (*Page-table length register*) kích thước PCB.
- Địa chỉ truy nhập dạng $\langle p, d \rangle$



Thực hiện chương trình : minh họa

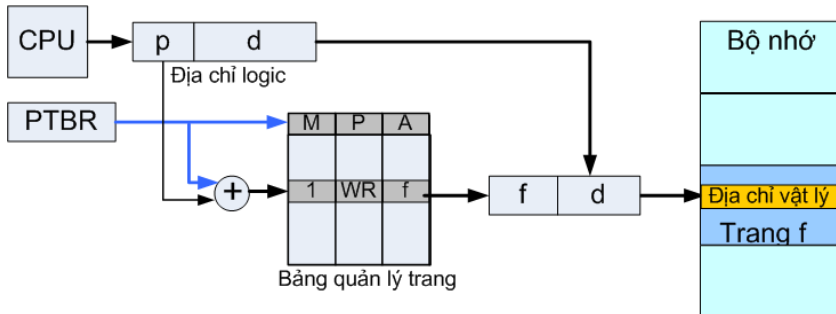


Chuyển đổi địa chỉ: Sơ đồ truy nhập



- $PTRB + p$: Địa chỉ phần tử p của PCB trong bộ nhớ

Chuyển đổi địa chỉ: Sơ đồ truy nhập



- $PTRB + p$: Địa chỉ phần tử p của PCB trong bộ nhớ
- Kiểm tra M_p
 - $M_p = 0$: **Lỗi trang**, sinh một ngắt để tiến hành nạp trang
 - Xin trang vật lý tự do
 - Tìm kiếm trang logic ở bộ nhớ ngoài
 - Sửa lại trường địa chỉ A và dấu hiệu M
 - $M_p = 1$: Trang đã tồn tại,
 - Lấy A_p **ghép** với d ra địa chỉ cần tìm

Nạp trang và thay thế trang

- Nhận xét

- Số trang vật lý dành cho chương trình lớn \Rightarrow Thực hiện nhanh nhưng hệ số song song giảm
- Số trang vật lý dành cho chương trình bé \Rightarrow
- Hệ số song song cao nhưng thực hiện chậm do hay thiếu trang
- Hiệu quả phụ thuộc các chiến lược nạp trang và thay thế trang



Nạp trang và thay thế trang

- Nhận xét

- Số trang vật lý dành cho chương trình lớn \Rightarrow Thực hiện nhanh nhưng hệ số song song giảm
- Số trang vật lý dành cho chương trình bé \Rightarrow
- Hệ số song song cao nhưng thực hiện chậm do hay thiếu trang
- Hiệu quả phụ thuộc các chiến lược nạp trang và thay thế trang

- Các chiến lược nạp trang

Nạp tất cả Nạp toàn bộ chương trình

Nạp trước Dự báo trang cần thiết tiếp theo

Nạp theo yêu cầu Chỉ nạp khi cần thiết



Nạp trang và thay thế trang

- Nhận xét
 - Số trang vật lý dành cho chương trình lớn \Rightarrow Thực hiện nhanh nhưng hệ số song song giảm
 - Số trang vật lý dành cho chương trình bé \Rightarrow
 - Hệ số song song cao nhưng thực hiện chậm do hay thiếu trang
 - Hiệu quả phụ thuộc các chiến lược nạp trang và thay thế trang
- Các chiến lược nạp trang
 - Nạp tất cả** Nạp toàn bộ chương trình
 - Nạp trước** Dự báo trang cần thiết tiếp theo
 - Nạp theo yêu cầu** Chỉ nạp khi cần thiết
- Các chiến lược thay thế trang
 - FIFO** First In First Out
 - LRU** Least Recently Used

...



Ưu điểm

- Tăng tốc độ truy nhập
 - Hai lần truy nhập bộ nhớ (*vào PCB và vào địa chỉ cần tìm*)
 - Thực hiện phép **ghép** thay vì phép **cộng**



Ưu điểm

- Tăng tốc độ truy nhập
 - Hai lần truy nhập bộ nhớ (*vào PCB và vào địa chỉ cần tìm*)
 - Thực hiện phép **ghép** thay vì phép **cộng**
- Không tồn tại hiện tượng phân đoạn ngoài



Ưu điểm

- Tăng tốc độ truy nhập
 - Hai lần truy nhập bộ nhớ (*vào PCB và vào địa chỉ cần tìm*)
 - Thực hiện phép **ghép** thay vì phép **cộng**
- Không tồn tại hiện tượng phân đoạn ngoài
- Hệ số song song cao
 - Chỉ cần một vài trang của chương trình trong bộ nhớ
 - Cho phép viết chương trình lớn tùy ý



Ưu điểm

- Tăng tốc độ truy nhập
 - Hai lần truy nhập bộ nhớ (*vào PCB và vào địa chỉ cần tìm*)
 - Thực hiện phép **ghép** thay vì phép **cộng**
- Không tồn tại hiện tượng phân đoạn ngoài
- Hệ số song song cao
 - Chỉ cần một vài trang của chương trình trong bộ nhớ
 - Cho phép viết chương trình lớn tùy ý
- Dễ dàng thực hiện nhiệm vụ bảo vệ
 - Địa chỉ truy nhập hợp lệ (*vượt quá kích thước*)
 - Tính chất truy nhập (*đọc/ghi*)
 - Quyền truy nhập (*user/system*)

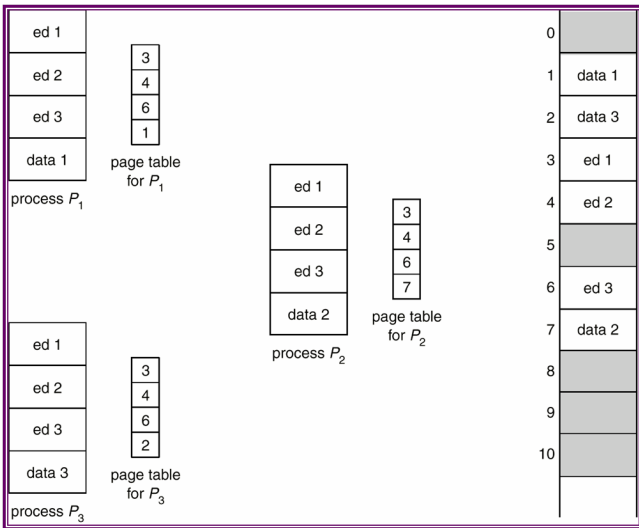


Ưu điểm

- Tăng tốc độ truy nhập
 - Hai lần truy nhập bộ nhớ (*vào PCB và vào địa chỉ cần tìm*)
 - Thực hiện phép **ghép** thay vì phép **cộng**
- Không tồn tại hiện tượng phân đoạn ngoài
- Hệ số song song cao
 - Chỉ cần một vài trang của chương trình trong bộ nhớ
 - Cho phép viết chương trình lớn tùy ý
- Dễ dàng thực hiện nhiệm vụ bảo vệ
 - Địa chỉ truy nhập hợp lệ (*vượt quá kích thước*)
 - Tính chất truy nhập (*đọc/ghi*)
 - Quyền truy nhập (*user/system*)
- Cho phép sử dụng chung trang

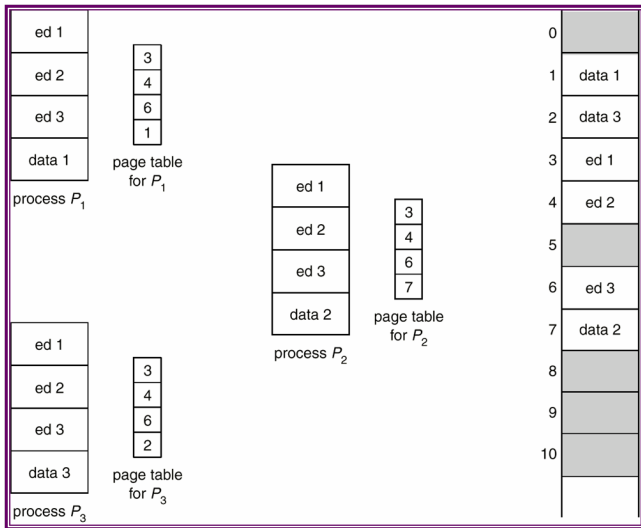


Dùng chung trang : Soạn thảo văn bản



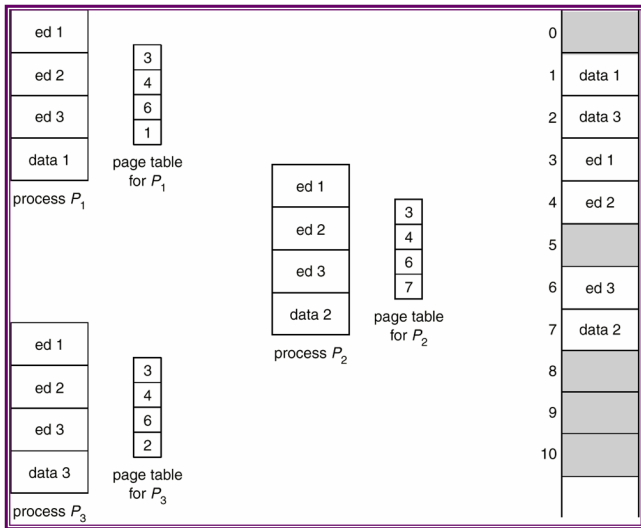
- Mỗi trang 50K
- 3 trang mã
- 1 trang dữ liệu
- 40 người dùng

Dùng chung trang : Soạn thảo văn bản



- Mỗi trang 50K
- 3 trang mã
- 1 trang dữ liệu
- 40 người dùng
- Không dùng chung
 - Cần 8000K

Dùng chung trang : Soạn thảo văn bản



- Mỗi trang 50K
- 3 trang mã
- 1 trang dữ liệu
- 40 người dùng
- Không dùng chung
 - Cần 8000K
- Dùng chung
 - Chỉ cần 2150K

Dùng chung trang: Nguyên tắc

- Quan trọng trong môi trường hoạt động phân chia



Dùng chung trang: Nguyên tắc

- Quan trọng trong môi trường hoạt động phân chia
- Chỉ cần một phiên bản mã chỉ đọc được dùng chung giữa các tiến trình nằm trong bộ nhớ
 - Soạn thảo văn bản, chương trình dịch, hệ cơ sở dữ liệu..



Dùng chung trang: Nguyên tắc

- Quan trọng trong môi trường hoạt động phân chia
- Chỉ cần một phiên bản mã chỉ đọc được dùng chung giữa các tiến trình nằm trong bộ nhớ
 - Soạn thảo văn bản, chương trình dịch, hệ cơ sở dữ liệu..
- Vấn đề: Mã dùng chung phải không đổi
 - Trang dùng chung phải cùng số hiệu trong bảng quản lý trang của tất cả tiến trình



Dùng chung trang: Nguyên tắc

- Quan trọng trong môi trường hoạt động phân chia
- Chỉ cần một phiên bản mã chỉ đọc được dùng chung giữa các tiến trình nằm trong bộ nhớ
 - Soạn thảo văn bản, chương trình dịch, hệ cơ sở dữ liệu..
- Vấn đề: Mã dùng chung phải không đổi
 - Trang dùng chung phải cùng số hiệu trong bảng quản lý trang của tất cả tiến trình
- Đoạn mã và dữ liệu riêng biệt có thể nằm ở vị trí bất kỳ trong bộ nhớ



Dùng chung trang: Nguyên tắc

- Quan trọng trong môi trường hoạt động phân chia
- Chỉ cần một phiên bản mã chỉ đọc được dùng chung giữa các tiến trình nằm trong bộ nhớ
 - Soạn thảo văn bản, chương trình dịch, hệ cơ sở dữ liệu..
- Vấn đề: Mã dùng chung phải không đổi
 - Trang dùng chung phải cùng số hiệu trong bảng quản lý trang của tất cả tiến trình
- Đoạn mã và dữ liệu riêng biệt có thể nằm ở vị trí bất kỳ trong bộ nhớ
- Các tiến trình cần có một bản copy các thanh ghi và cấu trúc dữ liệu phục vụ cho thực hiện tiến trình



Nhược điểm

- Tồn tại hiện tượng phân đoạn trong
 - Luôn xuất hiện ở trang cuối cùng
 - Giảm hiện tượng phân đoạn trang bởi giảm kích thước trang ?
 - Hay gặp lỗi trang
 - Bảng quản lý trang lớn



Nhược điểm

- Tồn tại hiện tượng phân đoạn trong
 - Luôn xuất hiện ở trang cuối cùng
 - Giảm hiện tượng phân đoạn trang bởi giảm kích thước trang ?
 - Hay gặp lỗi trang
 - Bảng quản lý trang lớn
- Đòi hỏi hỗ trợ của phần cứng
 - Chi phí cho chiến lược phân trang lớn



Nhược điểm

- Tồn tại hiện tượng phân đoạn trong
 - Luôn xuất hiện ở trang cuối cùng
 - Giảm hiện tượng phân đoạn trang bởi giảm kích thước trang ?
 - Hay gặp lỗi trang
 - Bảng quản lý trang lớn
- Đòi hỏi hỗ trợ của phần cứng
 - Chi phí cho chiến lược phân trang lớn
- Khi chương trình lớn, bảng quản lý trang nhiều phần tử
 - Chương trình 2^{30} , trang 2^{12} PCB có 2^{20} phần tử
 - Tồn bộ nhớ lưu trữ PCB
 - Giải quyết: Trang nhiều mức



Trang nhiều mức

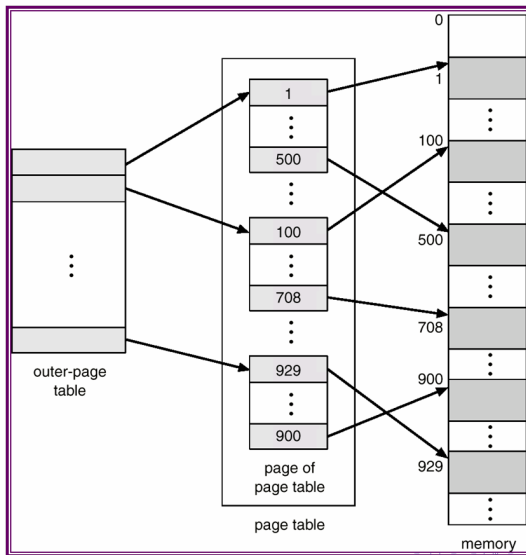
Nguyên tắc: Bảng quản lý trang được phân trang

Ví dụ trang 2 mức

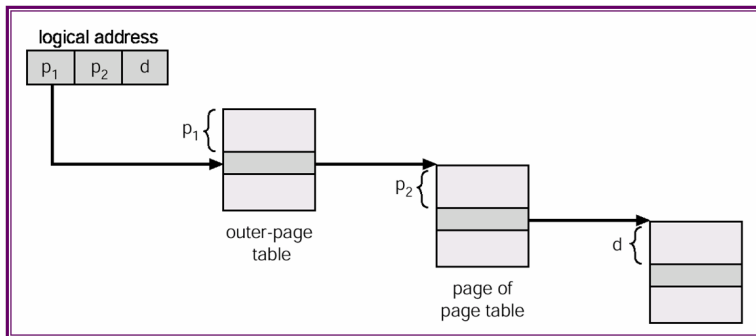
- Máy 32 bít địa chỉ (2^{32}); trang kích thước 4K (2^{12}) được chia
 - Số hiệu trang -20 bit
 - Độ lệch trong trang -12bit
- Bảng trang được phân trang. Số hiệu trang được chia thành
 - Bảng trang ngoài (*thư mục trang*) - 10 bit
 - Độ lệch trong một thư mục trang - 10bit
- Địa chỉ truy nhập có dạng $\langle p_1, p_2, d \rangle$



Trang nhiều mức: Ví dụ trang 2 mức

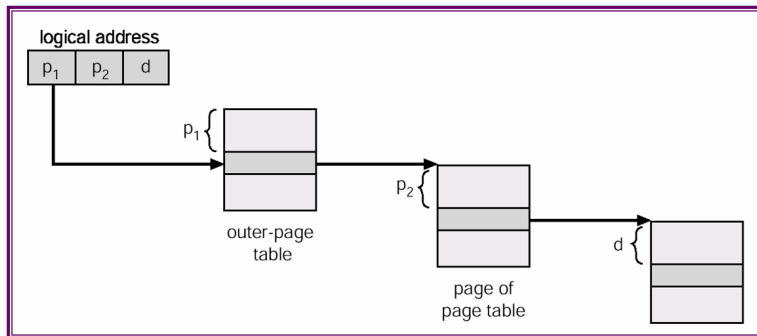


Trang nhiều mức: Truy nhập bộ nhớ



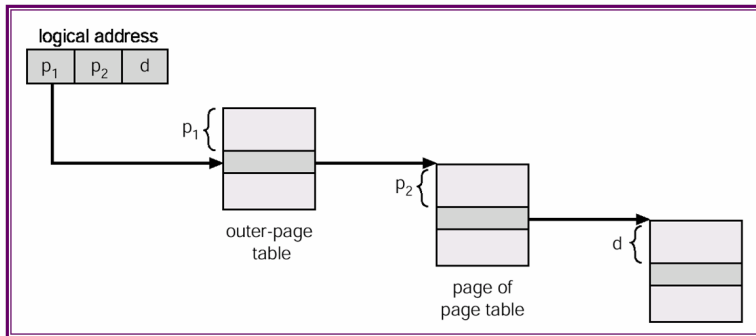
- Khi thực hiện : Hệ thống nạp thư mục trang vào bộ nhớ

Trang nhiều mức: Truy nhập bộ nhớ



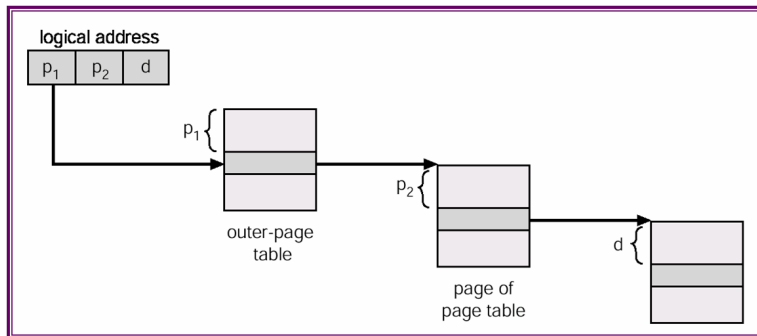
- Khi thực hiện : Hệ thống nạp thư mục trang vào bộ nhớ
- Bảng trang và trang không sử dụng không cần nạp vào bộ nhớ

Trang nhiều mức: Truy nhập bộ nhớ



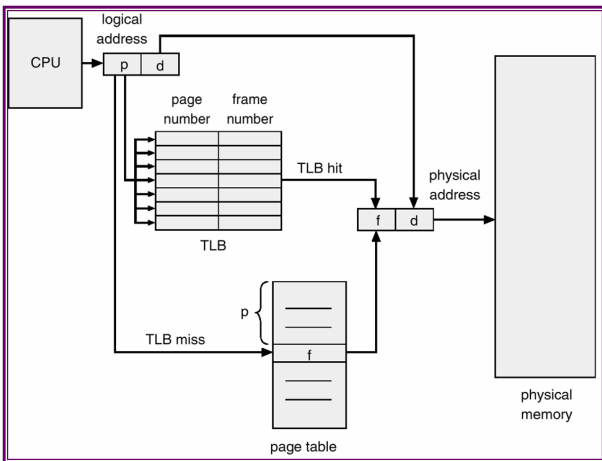
- Khi thực hiện : Hệ thống nạp thư mục trang vào bộ nhớ
- Bảng trang và trang không sử dụng không cần nạp vào bộ nhớ
- Cần 3 lần truy nhập tới bộ nhớ

Trang nhiều mức: Truy nhập bộ nhớ



- Khi thực hiện : Hệ thống nạp thư mục trang vào bộ nhớ
- Bảng trang và trang không sử dụng không cần nạp vào bộ nhớ
- Cần 3 lần truy nhập tới bộ nhớ
- Vấn đề: Với hệ thống 64 bit
 - Trang 3, 4,... mức
 - Cần 4, 5,... lần truy nhập bộ nhớ \Rightarrow chậm
 - Giải quyết: Bộ đệm chuyển hóa địa chỉ

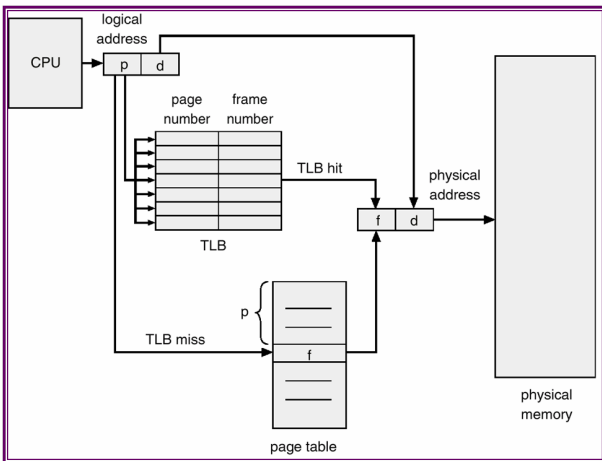
Bộ đệm chuyển hóa địa chỉ (TLB: translation look-aside buffers)



- Tập thanh ghi liên kết (*associative registers*)

- 98% t/nhập qua TLB

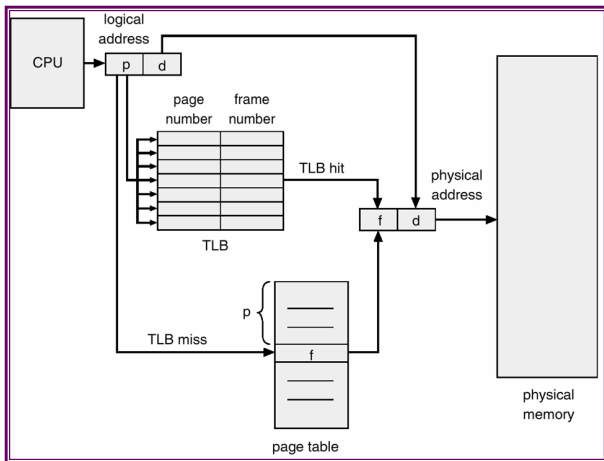
Bộ đệm chuyển hóa địa chỉ (TLB: translation look-aside buffers)



- Tập thanh ghi liên kết (*associative registers*)
- Truy nhập song song

- 98% t/nhập qua TLB

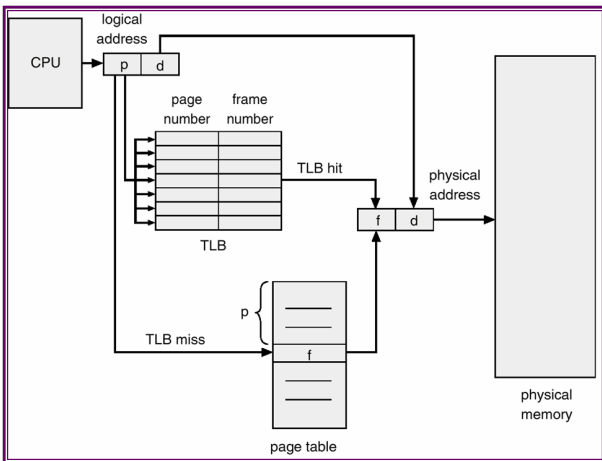
Bộ đệm chuyển hóa địa chỉ (TLB: translation look-aside buffers)



- Tập thanh ghi liên kết (*associative registers*)
- Truy nhập song song
- Mỗi phần tử gồm: khóa và giá trị

- 98% t/nhập qua TLB

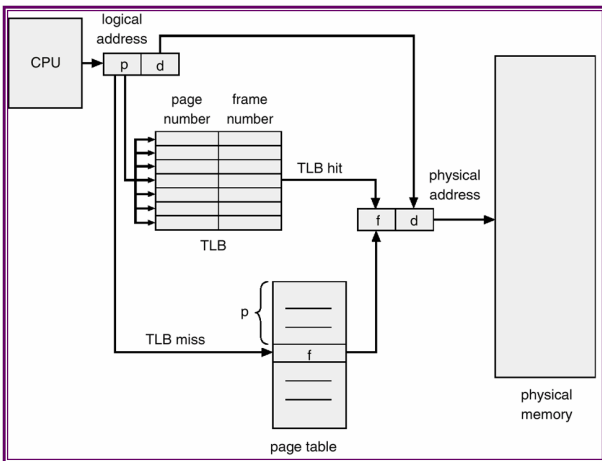
Bộ đệm chuyển hóa địa chỉ (TLB: translation look-aside buffers)



- Tập thanh ghi liên kết (*associative registers*)
- Truy nhập song song
- Mỗi phần tử gồm: khóa và giá trị
- TLB chứa đ/chỉ những trang mới truy nhập

- 98% t/nhập qua TLB

Bộ đệm chuyển hóa địa chỉ (TLB: translation look-aside buffers)



- Tập thanh ghi liên kết (*associative registers*)
- Truy nhập song song
- Mỗi phần tử gồm: khóa và giá trị
- TLB chứa đ/chỉ những trang mới truy nhập
- Khi có yêu cầu
 - Tìm trong TLB
 - Nếu không có, tìm trong PCB
 - Đưa vào TLB
- 98% t/nhập qua TLB

2 Các chiến lược quản lý bộ nhớ

- Chiến lược phân chương cố định
- Chiến lược phân chương động
- Chiến lược phân đoạn
- Chiến lược phân trang
- Chiến lược kết hợp phân đoạn-phân trang

Nguyên tắc

- Chương trình được biên tập theo chế độ phân đoạn
 - Tạo ra bảng quản lý đoạn SCB
 - Mỗi phần tử của bảng quản lý đoạn ứng với một đoạn, gồm 3 trường M, A, L



Nguyên tắc

- Chương trình được biên tập theo chế độ phân đoạn
 - Tạo ra bảng quản lý đoạn SCB
 - Mỗi phần tử của bảng quản lý đoạn ứng với một đoạn, gồm 3 trường M, A, L
- Mỗi đoạn được biên tập riêng theo chế độ phân trang
 - Tạo ra bảng quản lý trang cho từng đoạn



Nguyên tắc

- Chương trình được biên tập theo chế độ phân đoạn
 - Tạo ra bảng quản lý đoạn SCB
 - Mỗi phần tử của bảng quản lý đoạn ứng với một đoạn, gồm 3 trường M, A, L
- Mỗi đoạn được biên tập riêng theo chế độ phân trang
 - Tạo ra bảng quản lý trang cho từng đoạn
- Địa chỉ truy nhập: bộ $3 < s, p, d >$



Nguyên tắc

- Chương trình được biên tập theo chế độ phân đoạn
 - Tạo ra bảng quản lý đoạn SCB
 - Mỗi phần tử của bảng quản lý đoạn ứng với một đoạn, gồm 3 trường M, A, L
- Mỗi đoạn được biên tập riêng theo chế độ phân trang
 - Tạo ra bảng quản lý trang cho từng đoạn
- Địa chỉ truy nhập: bộ $3 < s, p, d >$
- Thực hiện truy nhập địa chỉ
 - $STBR + s \Rightarrow$: địa chỉ phần tử s
 - Kiểm tra trường dấu hiệu M_s , nạp PCB_s nếu cần
 - $A_s + p \Rightarrow$ Địa chỉ phần tử p của PCB_s
 - Kiểm tra trường dấu hiệu M_p , nạp PCB_s nếu cần
 - Ghep A_p với d ra được địa chỉ cần tìm

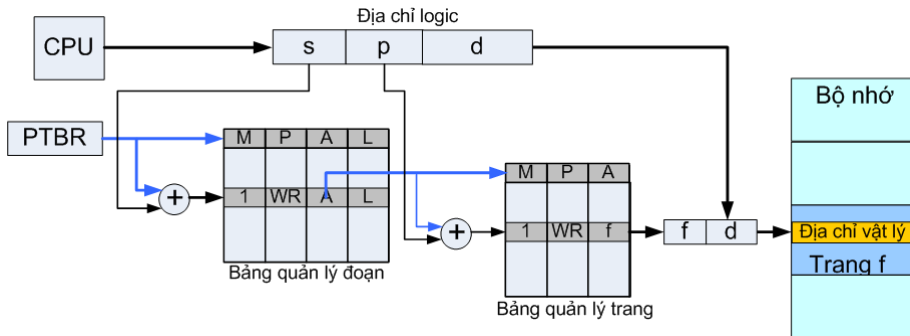


Nguyên tắc

- Chương trình được biên tập theo chế độ phân đoạn
 - Tạo ra bảng quản lý đoạn SCB
 - Mỗi phần tử của bảng quản lý đoạn ứng với một đoạn, gồm 3 trường M, A, L
- Mỗi đoạn được biên tập riêng theo chế độ phân trang
 - Tạo ra bảng quản lý trang cho từng đoạn
- Địa chỉ truy nhập: bộ $3 < s, p, d >$
- Thực hiện truy nhập địa chỉ
 - $STBR + s \Rightarrow$: địa chỉ phần tử s
 - Kiểm tra trường dấu hiệu M_s , nạp PCB_s nếu cần
 - $A_s + p \Rightarrow$ Địa chỉ phần tử p của PCB_s
 - Kiểm tra trường dấu hiệu M_p , nạp PCB_s nếu cần
 - Ghep A_p với d ra được địa chỉ cần tìm
- Được sử dụng trong VXL Intel 80386, MULTICS ...
 - Quản lý bộ nhớ của VXL họ intel?
 - Chế độ thực
 - Chế độ bảo vệ



Sơ đồ truy nhập bộ nhớ



Kết luận



Nội dung chính

- 1 Tổng quan
- 2 Các chiến lược quản lý bộ nhớ
- 3 Bộ nhớ ảo**



3 Bộ nhớ ảo

- 3.1 Giới thiệu



Kết luận

1 Tổng quan

- Ví dụ
- Bộ nhớ và chương trình
- Liên kết địa chỉ
- Các cấu trúc chương trình

2 Các chiến lược quản lý bộ nhớ

- Chiến lược phân chương cố định
- Chiến lược phân chương động
- Chiến lược phân đoạn
- Chiến lược phân trang
- Chiến lược kết hợp phân đoạn-phân trang

3 Bộ nhớ ảo

- 3.1 Giới thiệu

