Tiến trình

TH 106: Hệ điều hành

Khoa CNTT

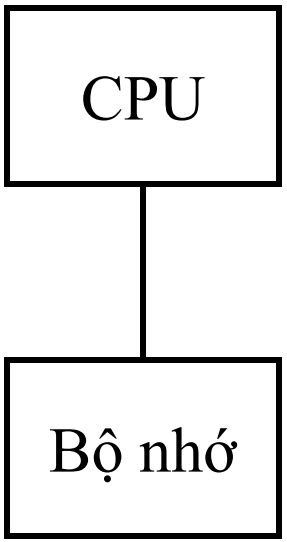
ĐH KHTN

Mô hình Von Neuman

Cả chương trình và dữ liệu đều ở trên bộ nhớ

Vòng lặp thực thi

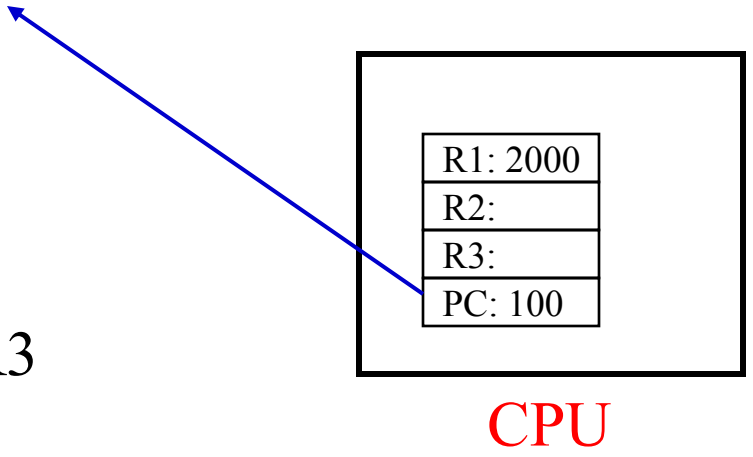
- Lấy lệnh
- Giải mã lệnh
- Thực thi lệnh



Hình ảnh thực thi một chương trình

100	load R1, R2
104	add R1, 4, R1
108	load R1, R3
112	add R2, R3, R3
	…
2000	4
2004	8

Bộ nhớ



Chúng ta lập trình như thế nào?

```
class foo { …};
static int a = 0;
static int b = 0;

void main() {
   foo foo_obj = new foo();
   foo_obj.cheat();
}

void foo::cheat() {
   int c = a;
   a = a + 1;
   if (c < 10) {
       cheat();
   }
}
```

Làm sao tạo chương trình như vậy trong máy von Neuman?

Lưu biến a, b ở đâu?

Biến foo_obj và c lưu ở đâu?

Làm sao thực hiện

foo_obj.cheat()?

Biến toàn cục

Giải quyết biến toàn cục như a và b dễ dàng hơn

Chỉ việc cấp phát bộ nhớ cho chúng

Được làm lúc thời gian biên dịch (*compile time)*

Mỗi lần truy xuất a là việc truy cập vào vùng nhớ đã cấp cho biến a

Giả sử a lưu tại ô 2000

Thì, a = a + 1 có thể được biên dịch như sau

```
loadi    2000, R1
load     R1, R2
add      R2, 1, R2
store    R1, R2
```

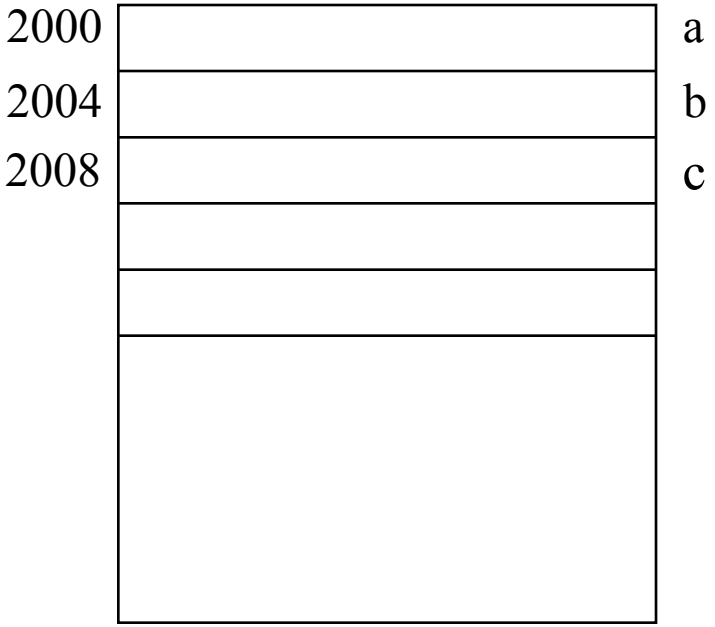
Biến cục bộ

Biến foo_obj trong main() và c trong hàm cheat() thì sao?

Đầu tiên chúng ta sẽ nghĩ là cũng sẽ cấp vùng nhớ cho chúng là xong (như hình bên)

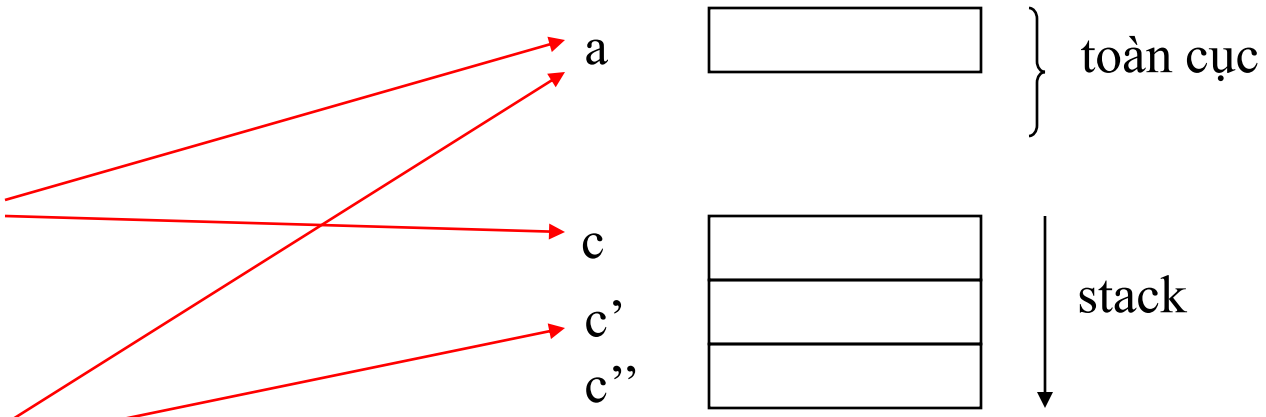
Có vấn đề gì với cách tiếp cận này?

Chúng ta sẽ giải quyết vấn đề này ra sao?



Biến cục bộ

```
cheat();
c = a;
…
cheat();
c = a;
…
```



Cấp phát bộ nhớ mới cho c mỗi khi cheat() được gọi trong lúc chạy CT

Tự động cấp phát bộ nhớ trong stack (thường gọi là stack điều khiển)

Pop ra khỏi stack sau khi thực thi hàm xong: không cần lưu biến này nữa

Đoạn mã cho việc cấp phát/giải phóng bộ nhớ trên stack được phát sinh bởi trình biên dịch trong lúc biên dịch

Đối tượng mới thì sao?

```
foo foo_obj = new foo();
```

foo_obj thật ra là một con trỏ trỏ tới một đối tượng foo

Như đã thảo luận, bộ nhớ trong stack được cấp phát cho foo_obj mỗi khi main() được gọi

Thật sự đối tượng mới được tạo cư trú ở đâu sau lệnh "new foo" ?

Liệu stack có phải là nơi thích hợp để lưu đối tượng?

Tại sao không?

Hình ảnh trong bộ nhớ

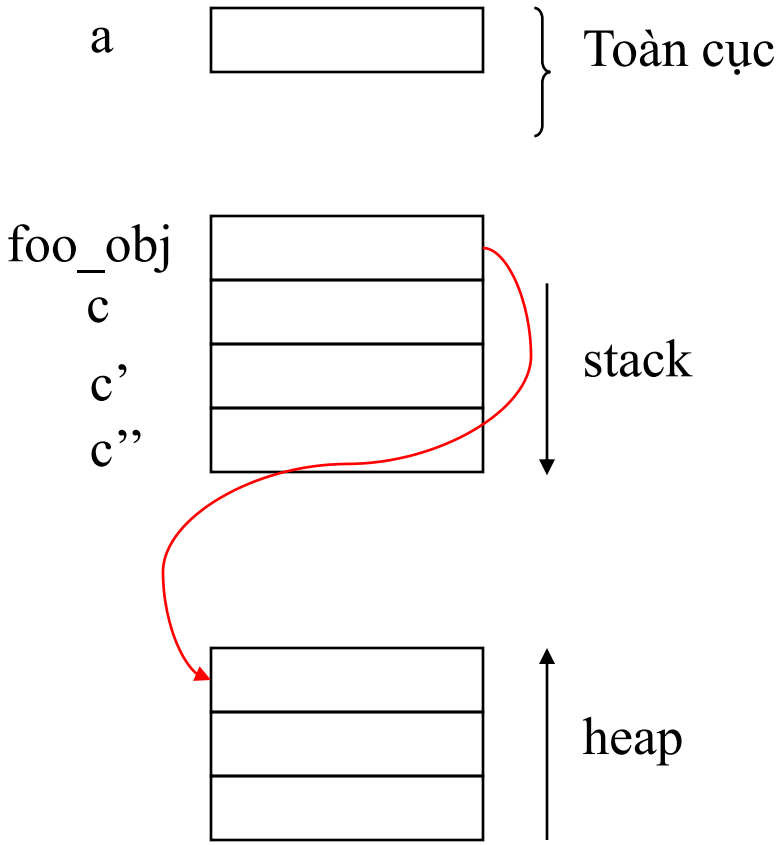
Giả sử chúng ta chạy CT như sau:

1. a = 0
2. b = 0
3. main()
4. foo_obj = new foo()

5. foo.cheat()
6. c = a
7. a = a + 1

8. foo.cheat()
9. c = a
10. a = a + 1

11. foo.cheat()
12. c = a
13. a = a + 1



Truy xuất dữ liệu

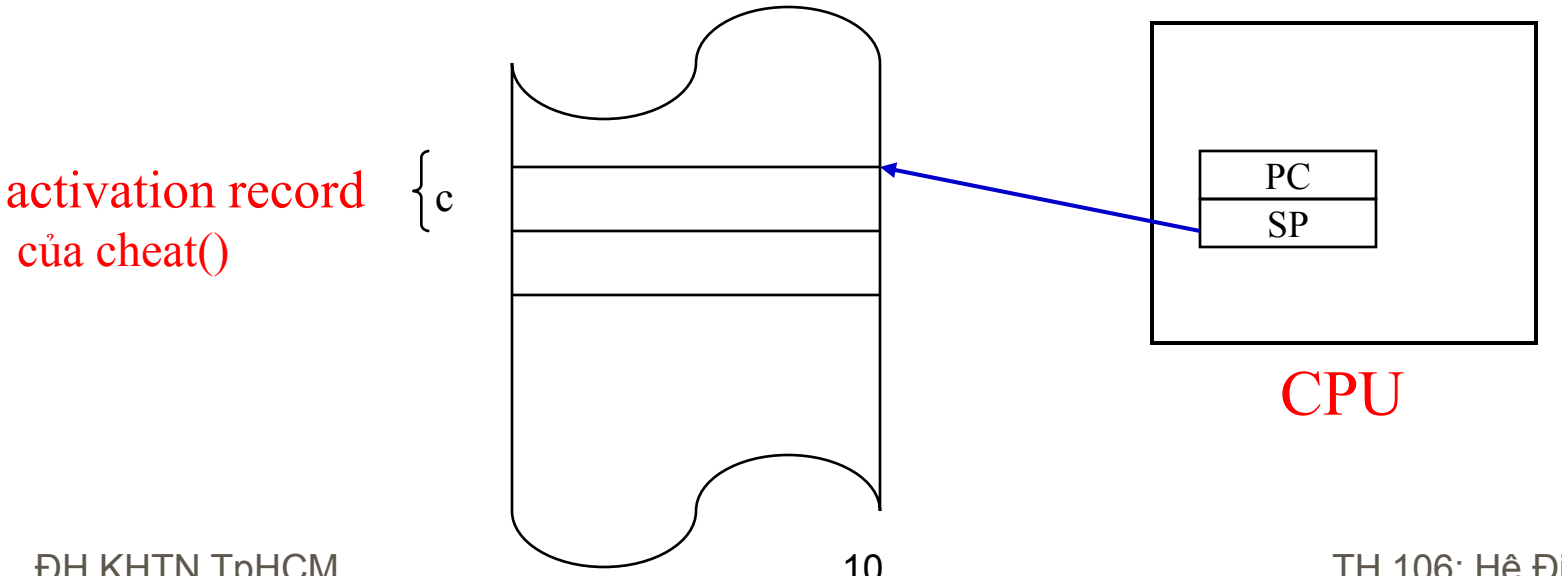
Làm sao tìm dữ liệu được cấp phát động trên stack?

Giải pháp, thiết kế 1 thanh ghi như là stack pointer

Stack pointer luôn trỏ tới activation record hiện hành

- Stack pointer trỏ tới vị trí lệnh đầu tiên của hàm
- Mã thay đổi giá trị stack pointer được phát sinh bởi trình biên dịch

Địa chỉ offsets của biến cục bộ và tham số được lưu trong thanh ghi SP



Truy xuất dữ liệu

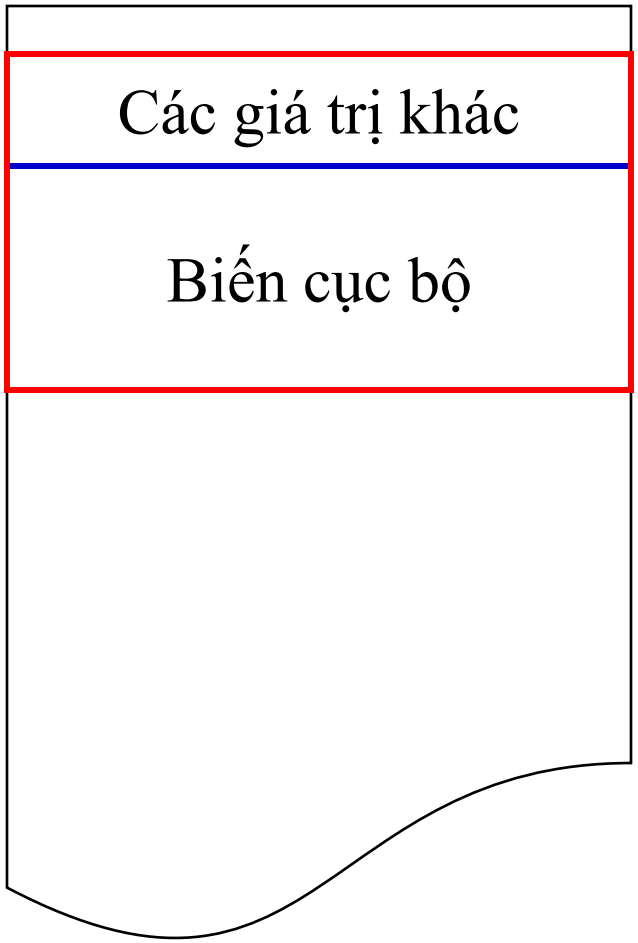
Dòng lệnh

a = a + 1

Được biên dịch thành

```
addi    0, sp, R1     # c là biến duy nhất nên offset = 0
load    R1, R2
addi    1, R2
store   R1, R2
```

Activation Record



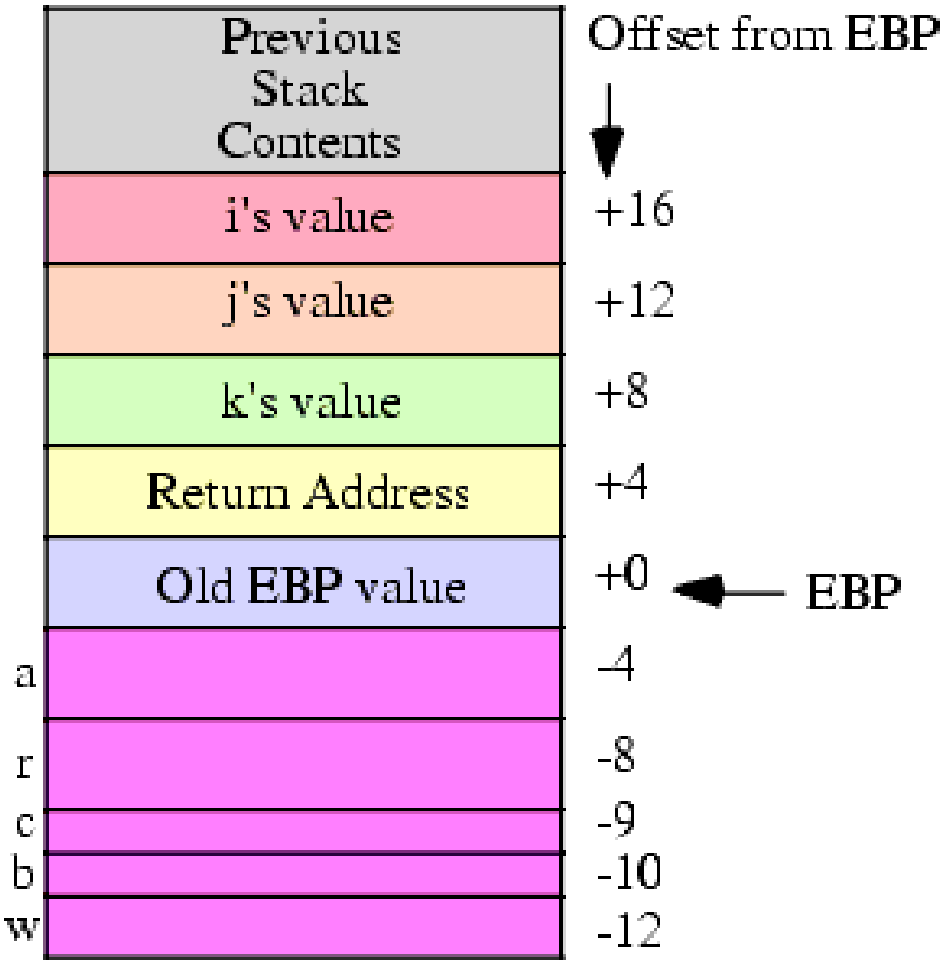
Chúng ta chỉ mới thảo luận về việc cấp phát biến cục bộ trên stack

Activation record được dùng để lưu:

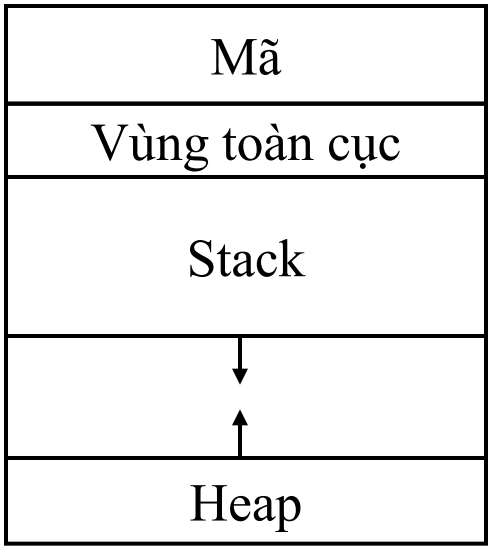
- Tham số của thủ tục
- Địa chỉ bắt đầu của activation record trước
- Địa chỉ trả về
- Các biến cục bộ
- …

Activation Record

```
procedure ARDemo (
k : int; j : int; i : dword );

var

        a : int32;

        r : real32;

        c : char;

        b : boolean;

        w : word;

begin ARDemo;

. . .

end ARDemo;
```



Tổ chức bộ nhớ trong lúc CT đang thực thi



Bộ nhớ

Mỗi biến được phân loại vào các vùng lưu trữ

Biến toàn cục (static)

- Được cấp phát trong vùng toàn cục lúc biên dịch

Biến cục bộ và tham số của hàm

- Cấp phát động trên stack

Các đối tượng được tạo mới khi thực thi (sử dụng phương thức new)

- Cấp phát động từ heap
- Các đối tượng tồn tại vì lời gọi hàm???
- Bộ thu gom rác (garbage) sẽ thu hồi lại vùng nhớ của đối tượng khi nó không còn dùng nữa

Kết luận

Tiến trình = xem như một tập các tài nguyên dùng để chạy một chương trình

= một chương trình đang thực thi
= nội dung bộ nhớ + nội dung các thanh ghi (+ trang thái I/O)

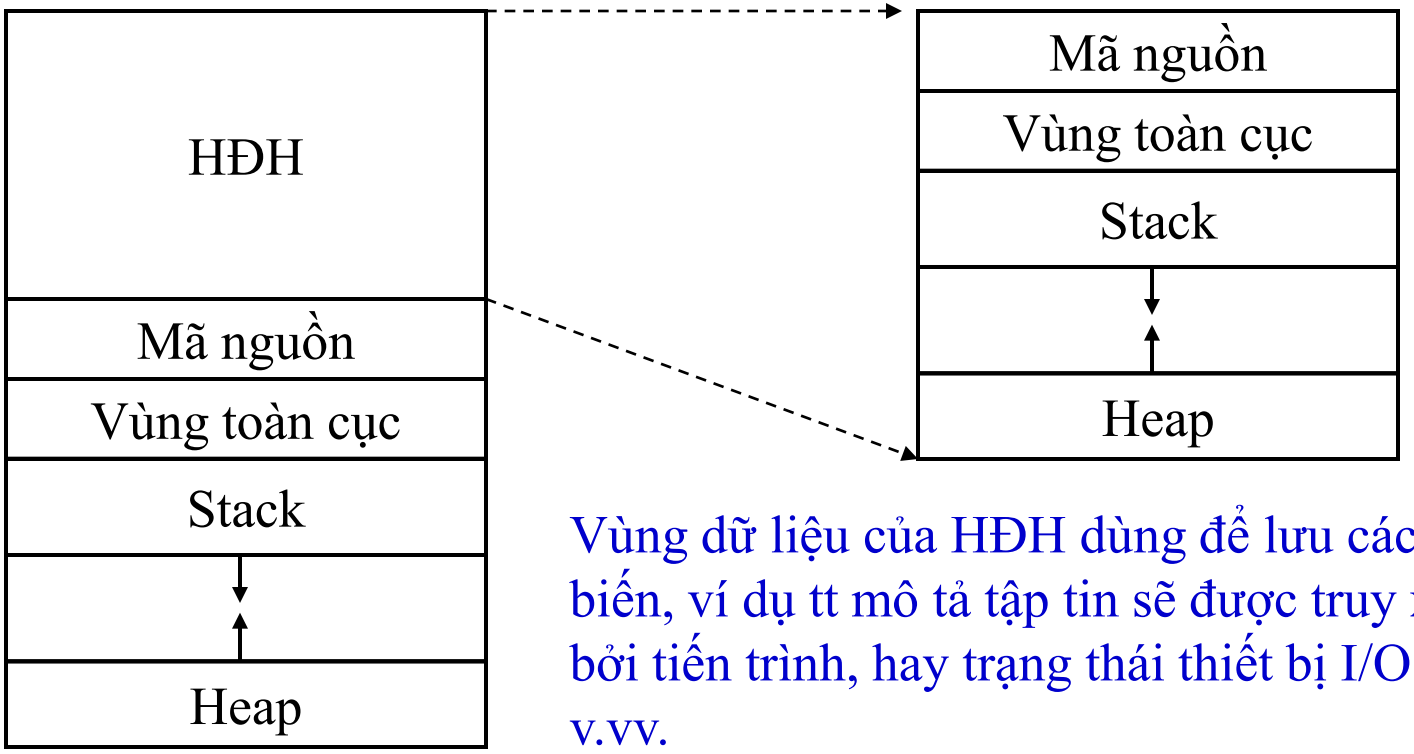
Stack + nội dung thanh ghi biểu diễn *execution context* hoặc *thread of control*

Hệ điều hành tổ chức ntn trên bộ nhớ?

Nhắc lại một trong những chức năng của HĐH là cung cấp giao diện máy ảo giúp cho việc lập trình cho máy dễ dàng hơn

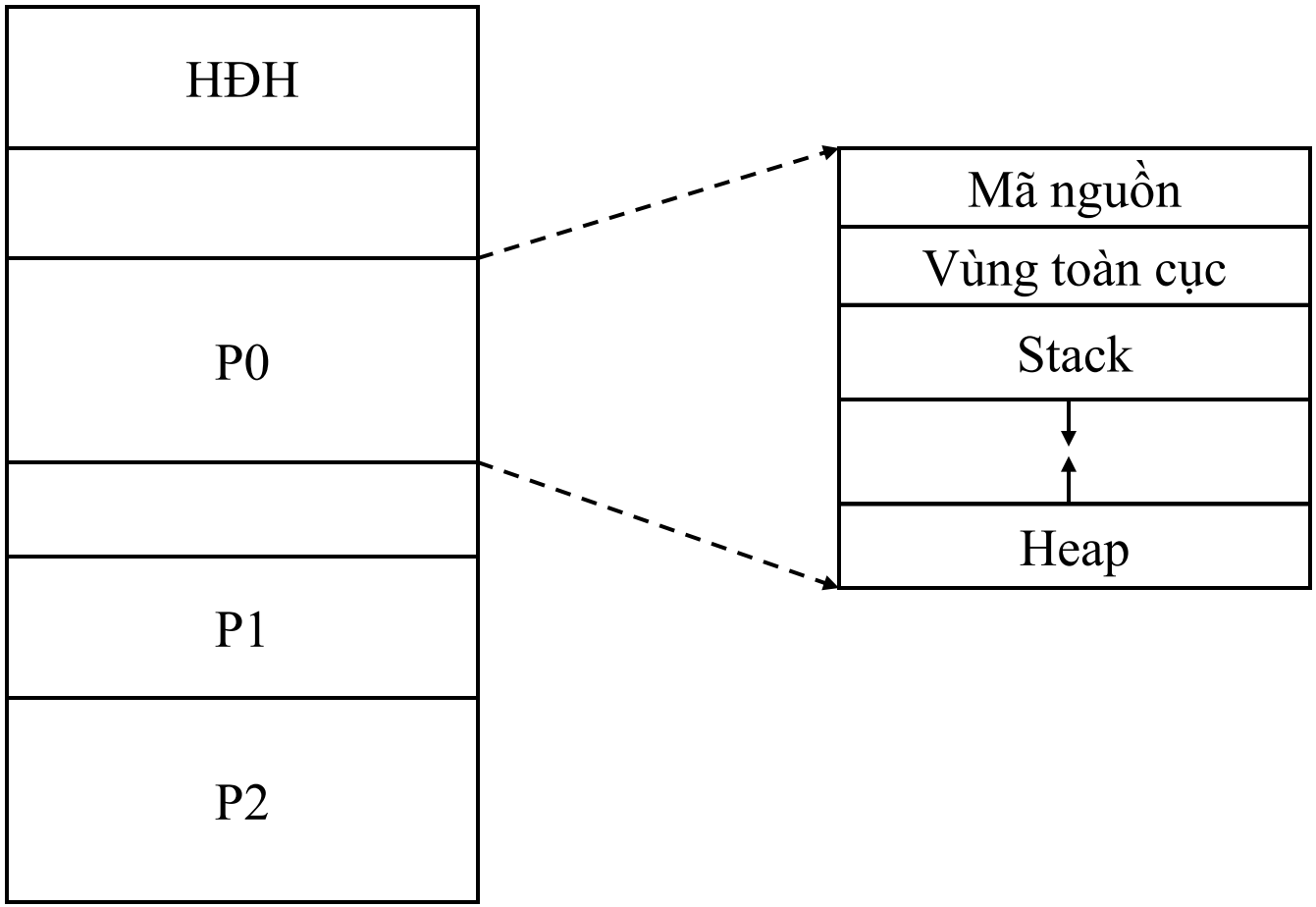
Vì vậy, hình ảnh bộ nhớ của tiến trình phải có chứa HĐH

Bộ nhớ

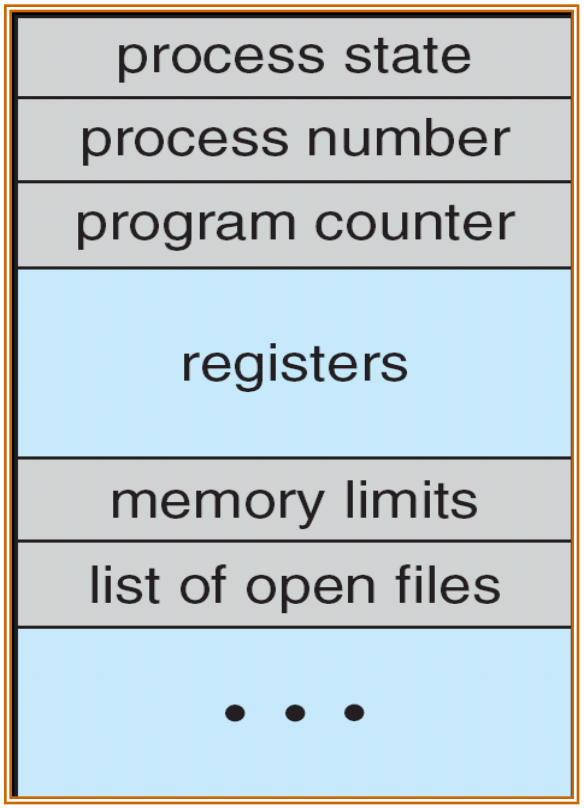


Vùng dữ liệu của HĐH dùng để lưu các biến, ví dụ tt mô tả tập tin sẽ được truy xuất bởi tiến trình, hay trạng thái thiết bị I/O, v.vv.

Điều gì xảy ra nếu nhiều tiến trình thực thi đồng thời?



Process Control Block



Trạng thái của mỗi tiến trình được lưu trong *process control block* (PCB)

Được xem là dữ liệu trong phần dữ liệu của HĐH

Tương tự như là đối tượng của một lớp

PCB

Các thông tin của một tiến trình:

Trạng thái tiến trình

Program counter

CPU registers

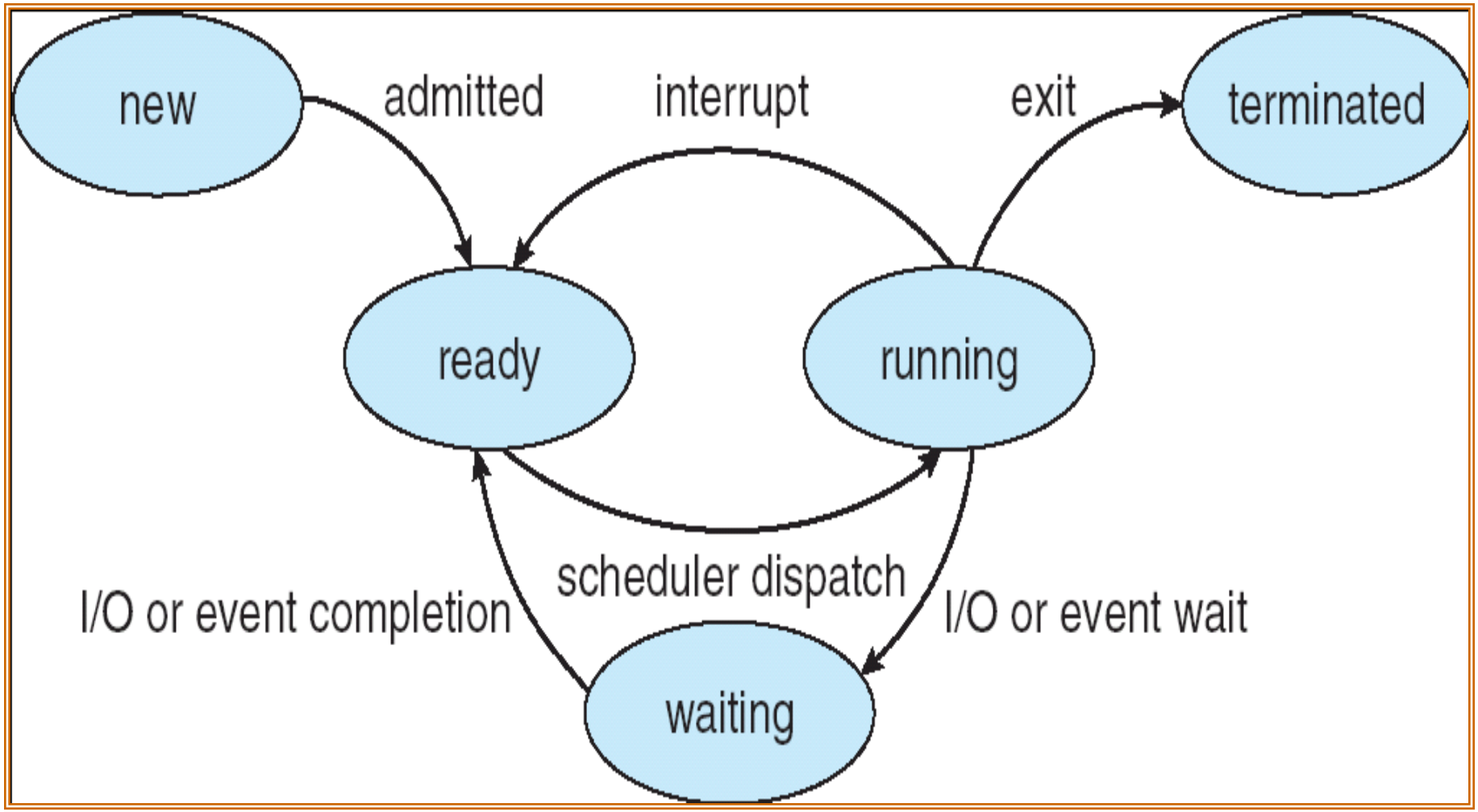
Thông tin lập lịch của CPU

Thông tin liên quan bộ nhớ

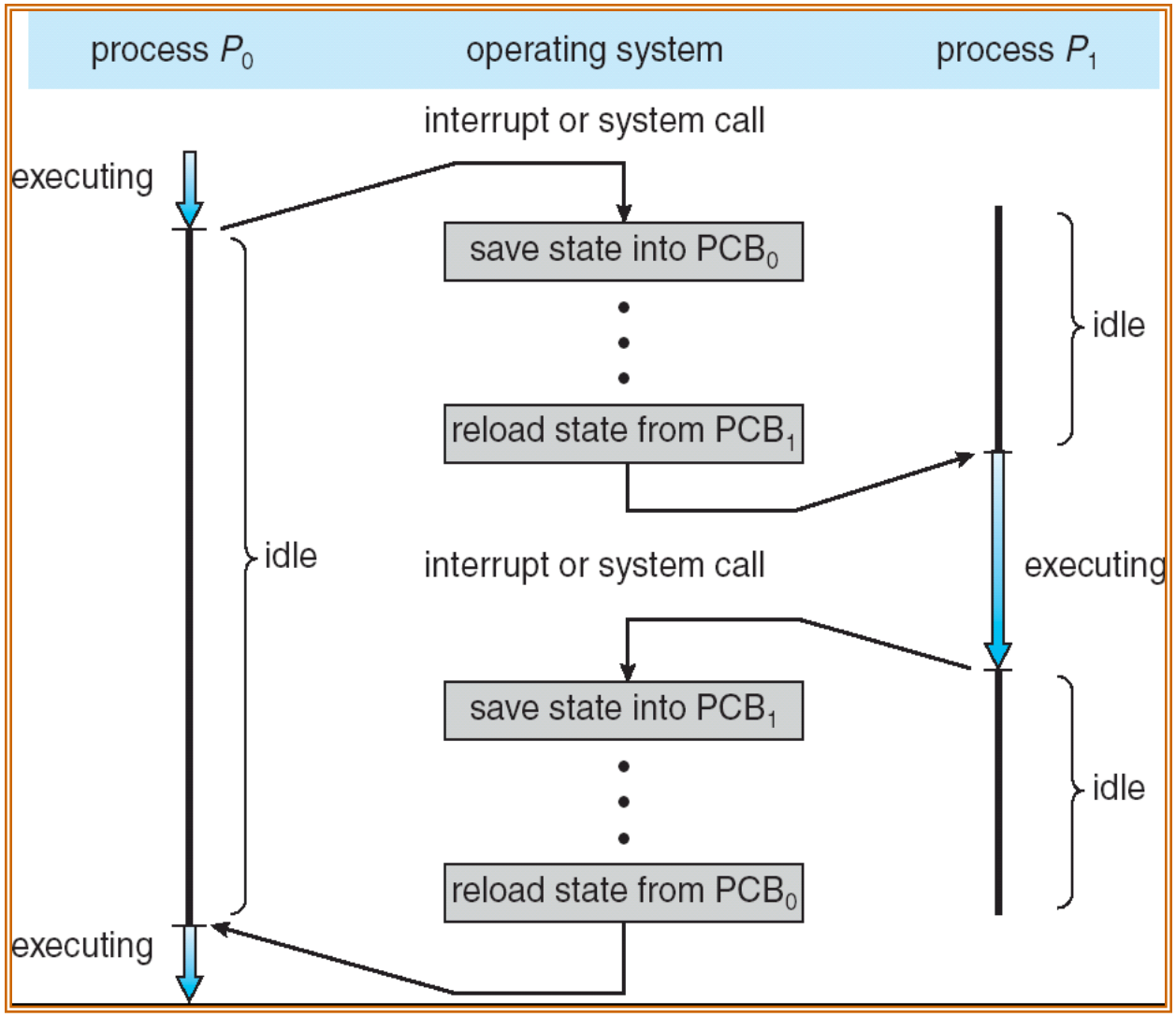
Thông tin sổ sách: *mã số tiến trình, số lượng CPU, thời gian sử dụng CPU,...*

Trạng thái I/O: *danh sách file đang mở, danh sách các thiết bị đã cấp cho tiến trình*

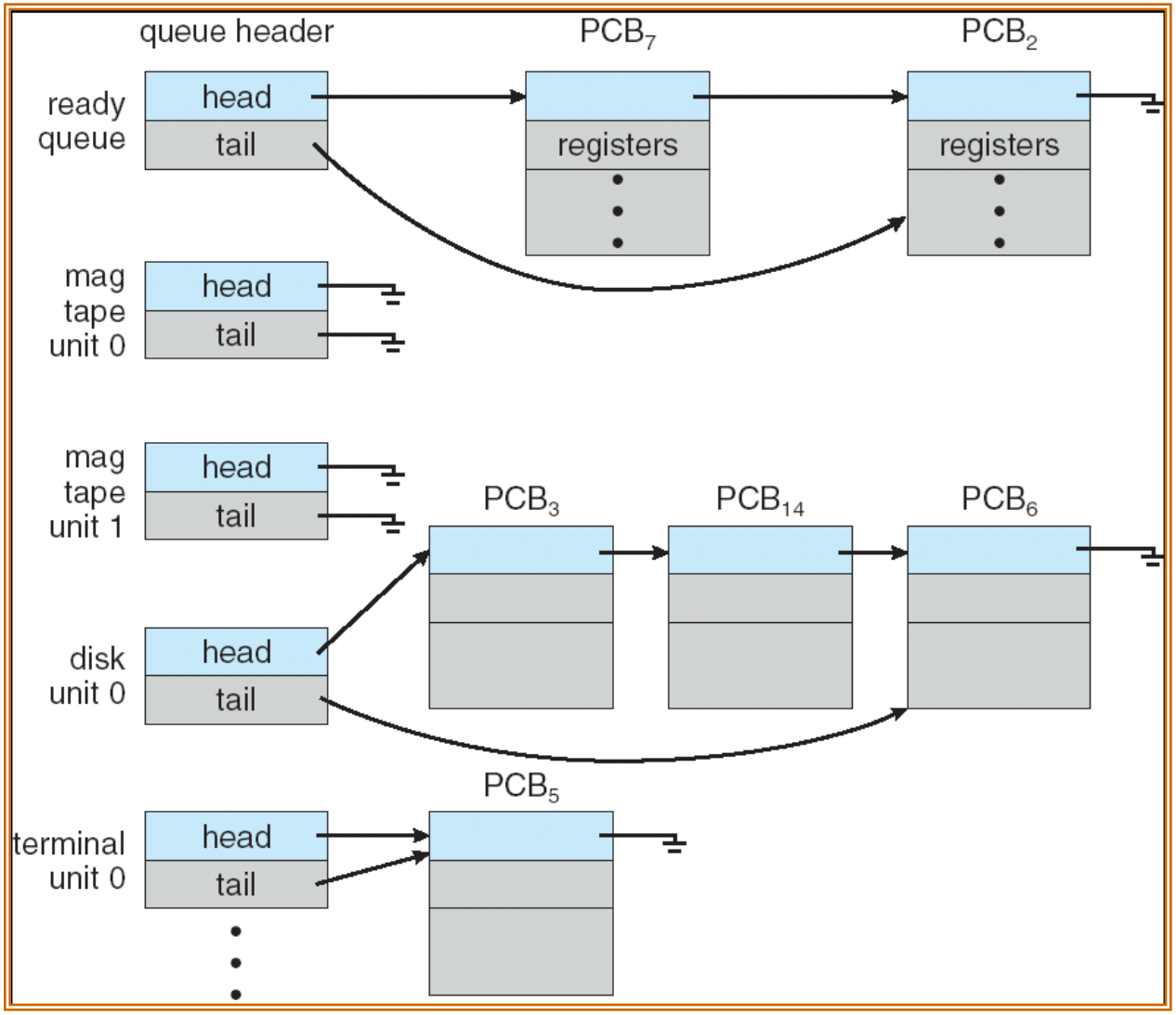
Các trạng thái của tiến trình



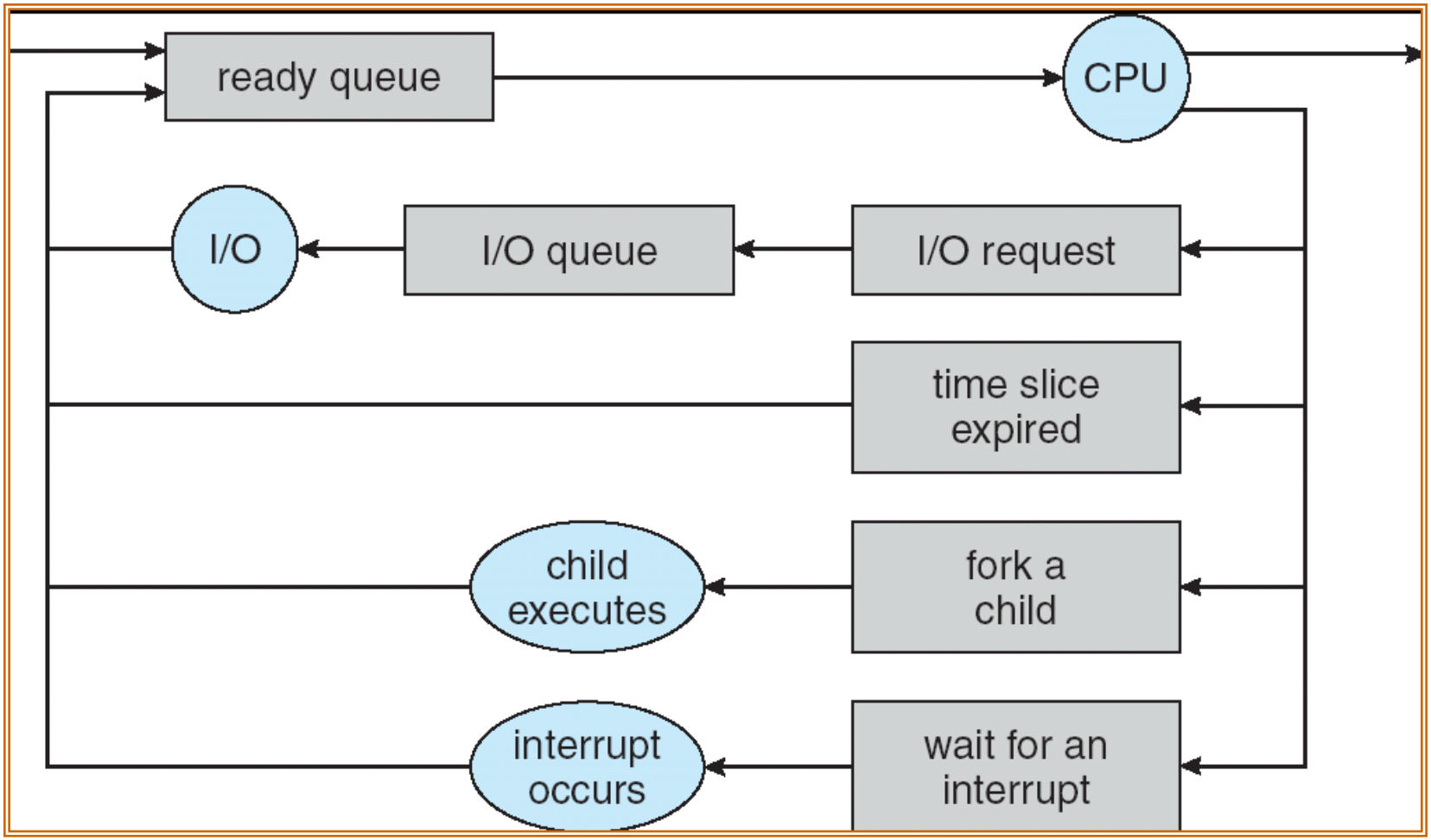
CPU chuyển đổi giữa các tiến trình



Ví dụ hàng đợi Ready



Lập lịch tiến trình



Tạo tiến trình

Làm sao tạo một tiến trình? Sử dụng System call.

Trong UNIX, một tiến trình có thể tạo một tiến trình khác bằng fork() (*system call*)

```
int pid = fork();           /* ngôn ngữ C */
```

Tiến trình tạo gọi là tiến trình cha và tiến trình mới gọi là tiến trình con

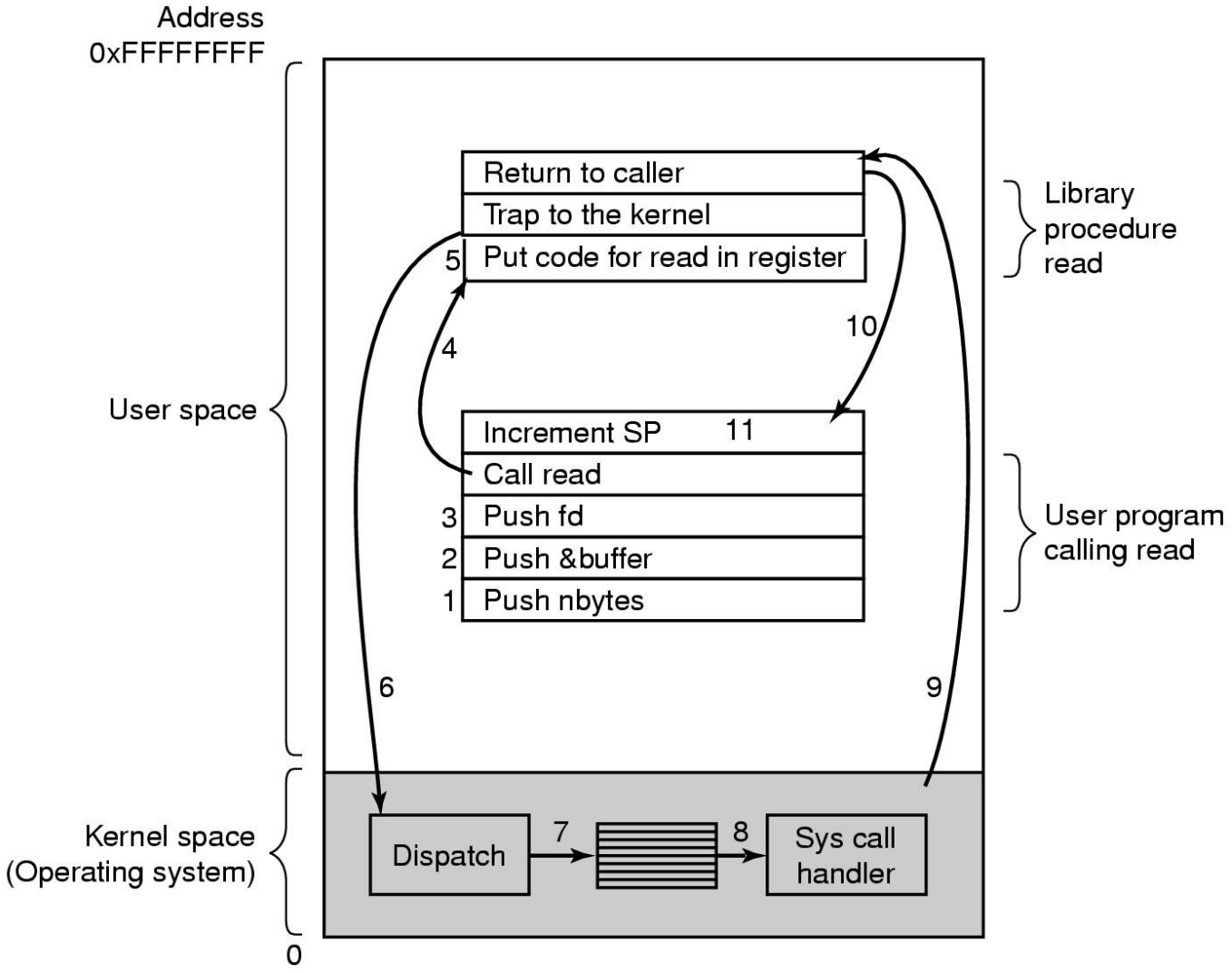
Tiến trình con là một bản sao của tiến trình cha (giống "hình dáng" và cấu trúc điều khiển tiến trình) ngoại trừ identification và trạng thái điều phối

- Tiến trình con và cha chạy trên hai vùng bộ nhớ khác nhau
- Mặc định là không có chia sẻ bộ nhớ
- Chi phí tạo tiến trình là lớn vì quá trình copy

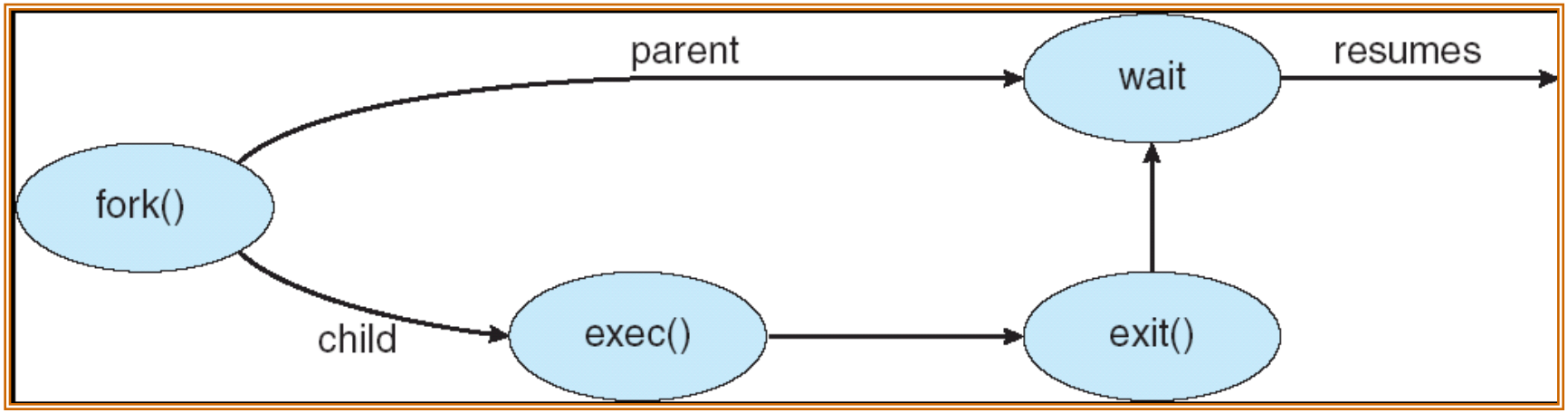
Hàm exec() cũng để tạo một tiến trình, nhưng thường là tạo tiến trình của chương trình mới chứ không phải gọi lại chính nó

System Call

11 bước để thực hiện 1 system call **read (fd, buffer, nbytes)**



Tạo tiến trình



Ví dụ tạo tiến trình sử dụng Fork()

```
int main()
{
Pid_t  pid;
        /* fork another process */
        pid = fork();
        if (pid < 0) { /* error occurred */
                fprintf(stderr, "Fork Failed");
                exit(-1);
        }
        else if (pid == 0) { /* child process */
                execlp("/bin/ls", "ls", NULL);
        }
        else { /* parent process */
                /* parent will wait for the child to complete */
                wait (NULL);
                printf ("Child Complete");
                exit(0);
        }
}
```

Hủy tiến trình

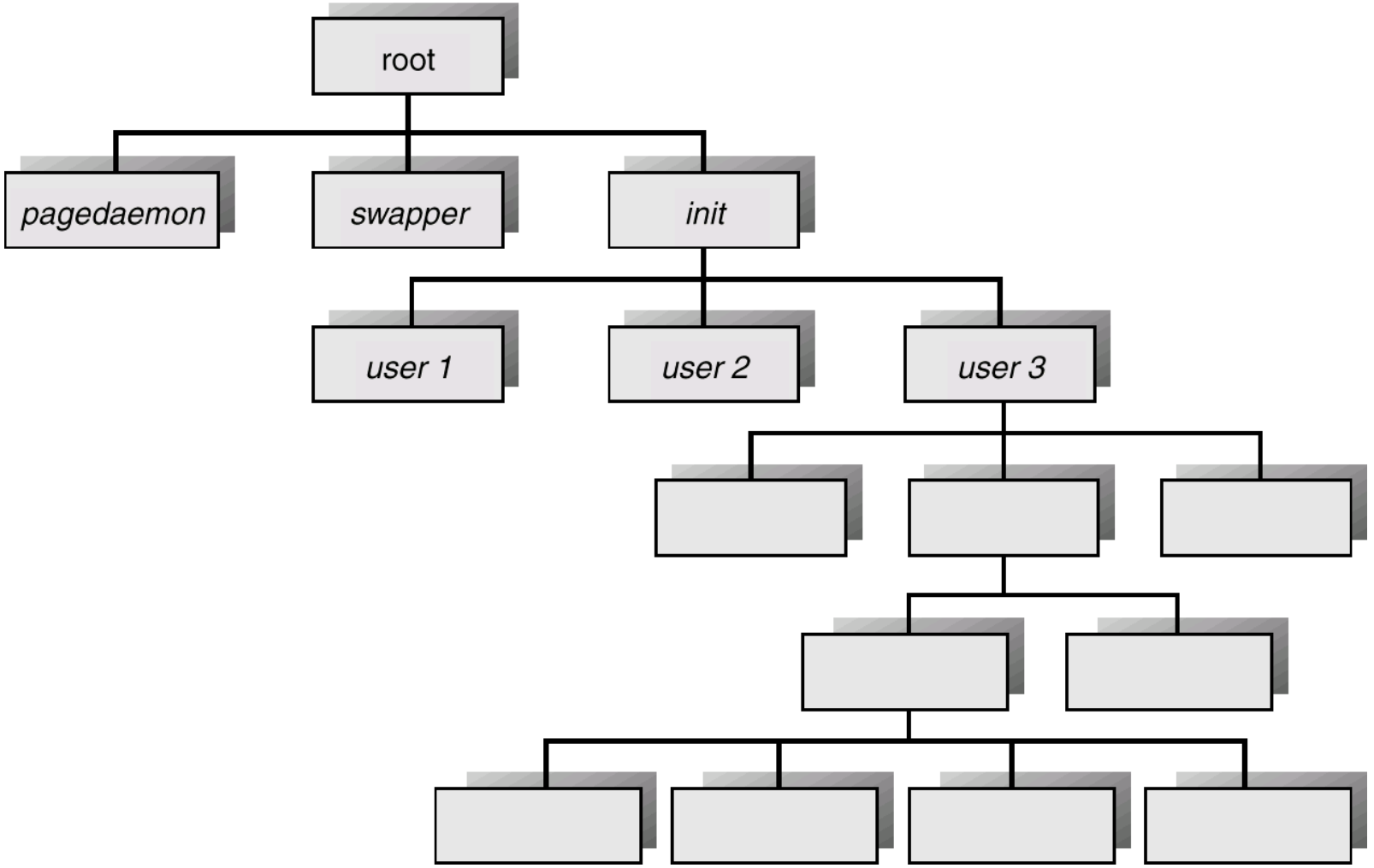
Một tiến trình có thể đợi một tiến trình khác hoàn thành bằng hàm wait() (*system call*)

- Có thể là đợi cho tiến trình con thực thi xong như ví dụ trên
- Cũng có thể đợi một tiến trình bất kì nào đó, phải biết PID của nó

Để "hủy" một tiến trình khác dùng kill() (*system call*)

- Điều gì xảy ra khi kill() được gọi?
- Điều gì xảy ra nếu tiến trình "nạn nhân" vẫn chưa muốn "chết"?

Cây tiến trình trong hệ UNIX thông thường



Tín hiệu (Signals)

Chương trình người dùng gọi các dịch vụ của HĐH bằng system calls

Phải làm sao khi chương trình muốn HĐH báo hiệu cho nó khi có sự kiện xảy ra một cách *bất đồng bộ*?

Phát tín hiệu

Cơ chế của UNIX là báo cho chương trình người dùng biết những sự kiện nó quan tâm

Các sự kiện như là: v.d., lỗi truy cập segment, có thông điệp, kill.

Khi muốn xử lý một sự kiện cụ thể nào đó (signal), tiến trình báo cho HĐH biết và gửi thủ tục xử lý sự kiện này cho HĐH

Tiến trình báo cho HĐH biết sự kiện nó quan tâm như thế nào?

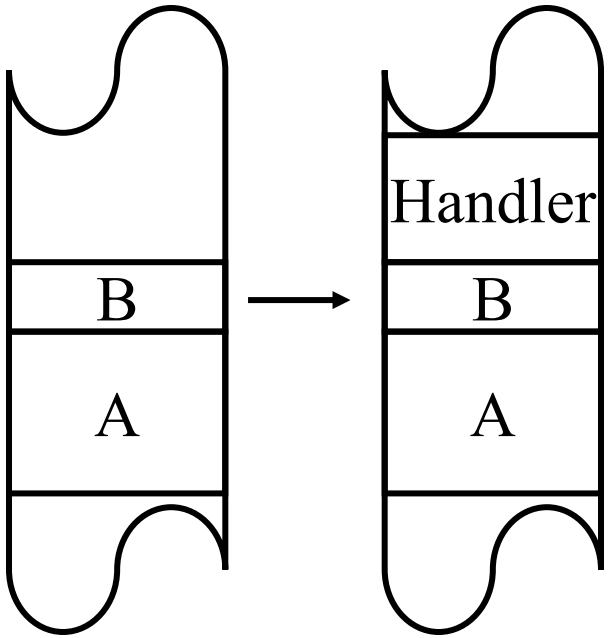
Tín hiệu (Signals tt)

Khi sự kiện quan tâm xảy ra:

Kernel xử lý trước, sau đó tìm xem thử có tiến trình nào muốn xử lý tín hiệu này

Tiến trình được thực thi thì đoạn lệnh xử lý sự kiện được xử lý trước

Khi xong, tiến trình trả về lại vị trí dòng lệnh trước khi sự kiện xảy ra.



Tóm tắt: tiến trình

Một "nhân bản" của chương trình

Mức trừu tượng: một tập các tài nguyên đủ để thực thi một chương trình

- Execution context(s)
- Không gian địa chỉ
- Thao tác tập tin, truyền thông, v.v..

Tất cả các điều trên được "chỉ" vào một khái niệm chung

Gần đây, chia ra thành vài khái niệm

- Tiểu trình, không gian địa chỉ, vùng được bảo vệ, v.v,.

Quản lý tiến trình của HĐH:

- Hỗ trợ người dùng tạo tiến trình và liên lạc giữa các tiến trình (IPC)
- Cấp phát tài nguyên cho tiến trình, tùy thuộc vào chính sách cụ thể
- Tăng khả năng tận dụng hệ thống bằng cách đồng bộ nhiều tiến trình

Tuần tới

Tiểu trình

Đọc Silberschatz chương 5