



HỆ ĐIỀU HÀNH

CHƯƠNG 3: TIẾN TRÌNH

Trình bày các khái niệm cơ bản về tiến trình, các thông số của tiến trình, các khái niệm cơ bản về định thời tiến trình và giao tiếp giữa các tiến trình, và biết được các tác vụ cơ bản của một tiến trình



MỤC TIÊU

1. Hiểu được khái niệm và các trạng thái của tiến trình
2. Biết được các thông số của tiến trình
3. Biết được các khái niệm về định thời tiến trình
4. Biết được các tác vụ cơ bản của một tiến trình
5. Hiểu được cách giao tiếp giữa các tiến trình



NỘI DUNG

1. Khái niệm cơ bản
2. Trạng thái tiến trình
3. Khối điều khiển tiến trình
4. Định thời tiến trình
5. Các tác vụ đối với tiến trình
6. Sự cộng tác giữa các tiến trình
7. Giao tiếp giữa các tiến trình
8. Tiểu trình



KHÁI NIỆM CƠ BẢN

1



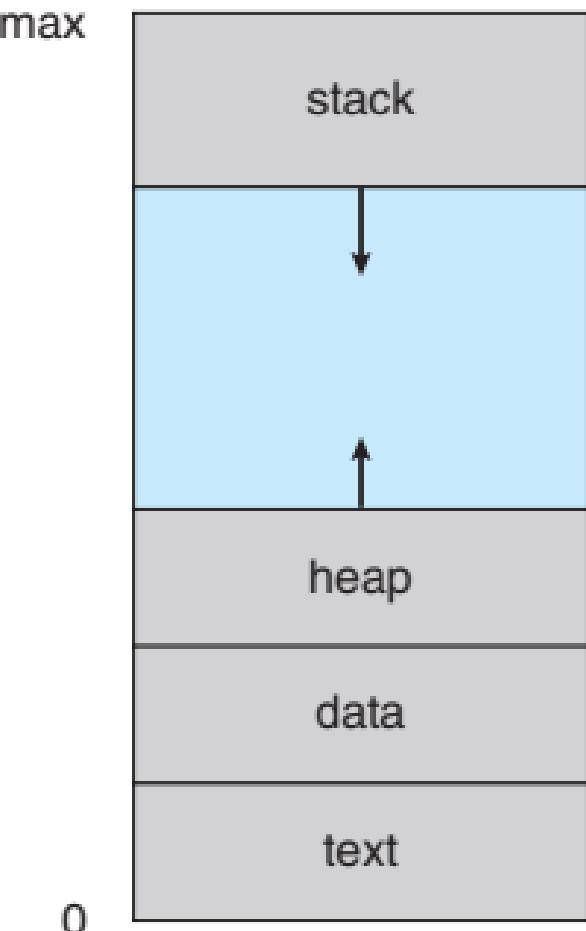
1. Khái niệm cơ bản

- Một hệ điều hành thực thi *chương trình* như là một *tiến trình*
- Tiến trình (process) là gì?
 - **Một chương trình đang thực thi**
- Chương trình là thực thể **bị động** lưu trên đĩa (tập tin thực thi - executable file); tiến trình là thực thể **chủ động**.
- Chương trình trở thành tiến trình khi một tập tin thực thi được nạp vào bộ nhớ.



1. Khái niệm cơ bản

- Một tiến trình bao gồm:
 - Text section (program code)
 - Data section (chứa global variables)
 - Program counter, processor registers
 - Heap section (chứa bộ nhớ cấp phát động)
 - Stack section (chứa dữ liệu tạm thời)
 - Function parameters
 - Return address
 - Local variables

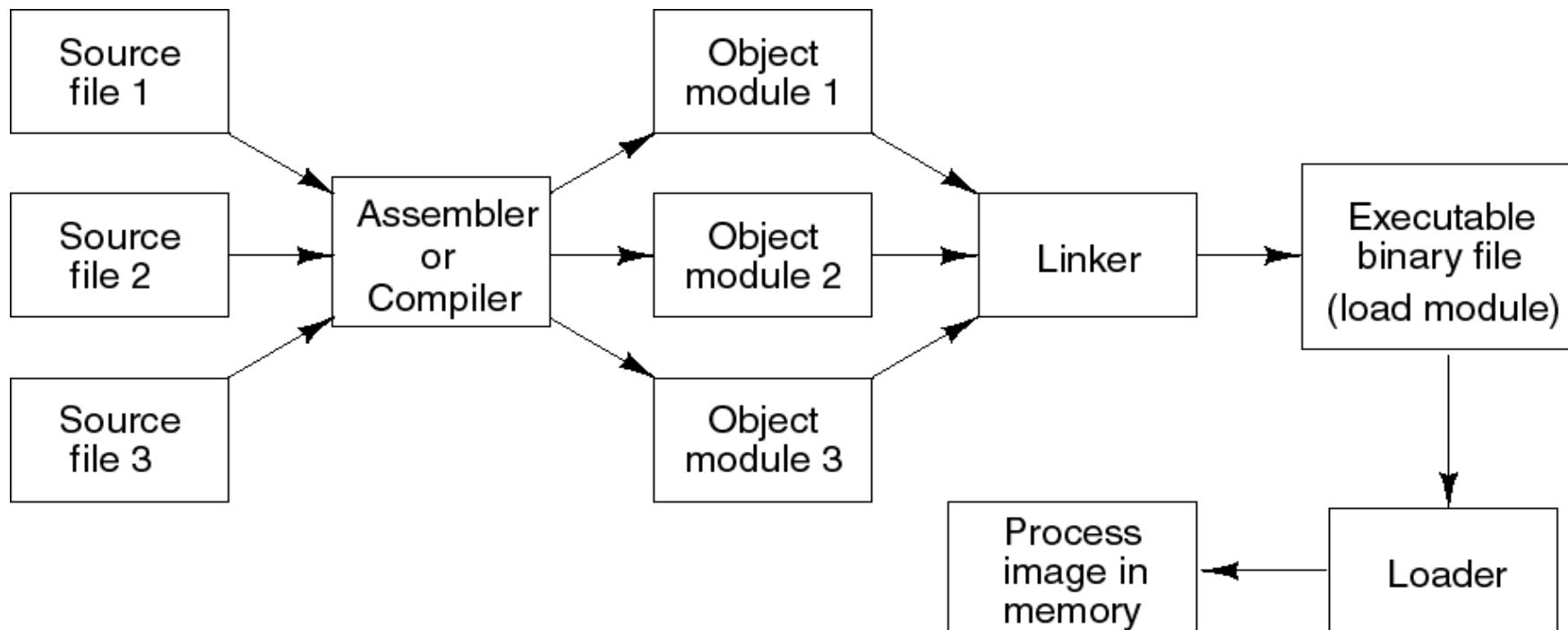


Layout của tiến trình trong bộ nhớ



1. Khái niệm cơ bản

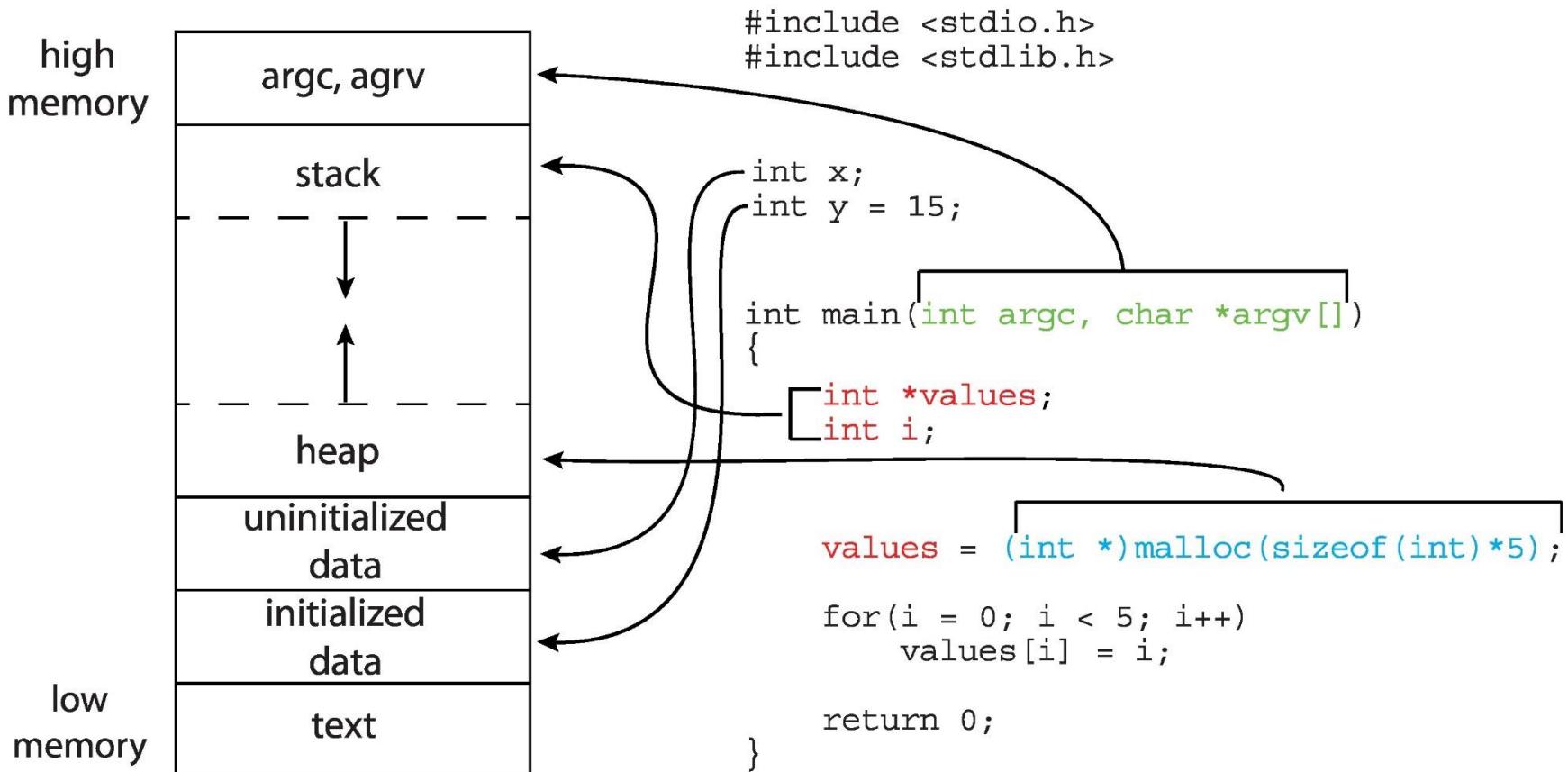
Các bước nạp chương trình vào bộ nhớ:





1. Khái niệm cơ bản

Layout bộ nhớ của một chương trình C





1. Khái niệm cơ bản

- Các bước khởi tạo tiến trình:
 - Cấp phát một định danh duy nhất cho tiến trình
 - Cấp phát không gian nhớ để nạp tiến trình
 - Khởi tạo khối dữ liệu Process Control Block (PCB) cho tiến trình
 - Thiết lập các mối liên hệ cần thiết (ví dụ: sắp PCB vào hàng đợi định thời, ...)



TRẠNG THÁI TIỀN TRÌNH

2

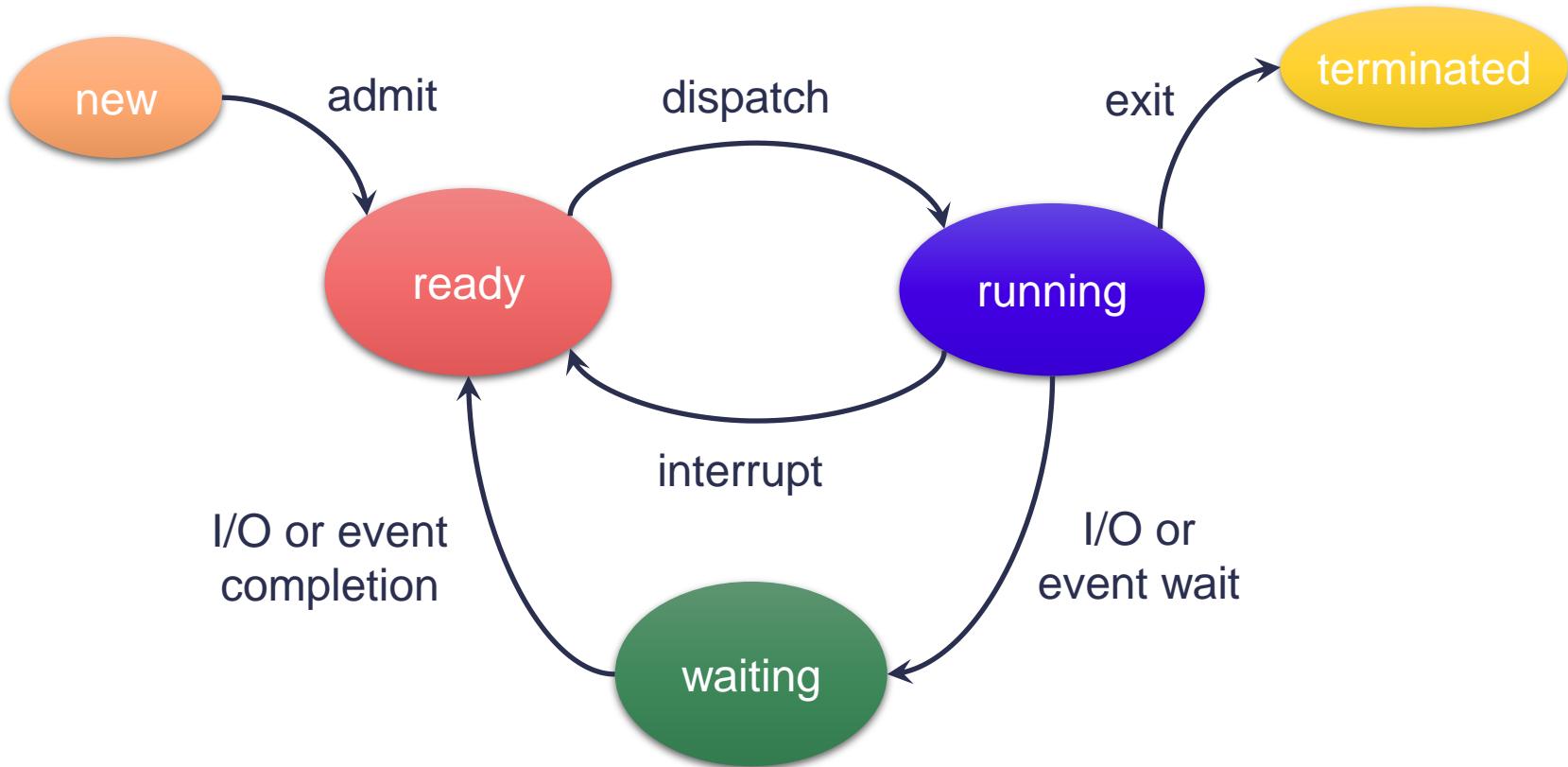


2. Trạng thái tiến trình

- **new**: tiến trình vừa được tạo
- **ready**: tiến trình đã có đủ tài nguyên, chỉ còn cần CPU
- **running**: các lệnh của tiến trình đang được thực thi
- **waiting** (hay **blocked**): tiến trình đợi I/O hoàn tất, hoặc đợi tín hiệu
- **terminated**: tiến trình đã kết thúc



2. Trạng thái tiến trình



Chuyển đổi giữa các trạng thái của tiến trình



2. Trạng thái tiến trình

```
/* test.c */  
int main(int argc, char** argv)  
{  
    printf("Hello world\n");  
    exit(0);  
}
```

- Biên dịch chương trình trong Linux: **gcc test.c -o test**
- Thực thi chương trình test:
./test
- Trong hệ thống sẽ có một tiến trình test được tạo ra, thực thi và kết thúc.

- Chuỗi trạng thái của tiến trình test như sau (trường hợp tốt nhất):
 - new
 - ready
 - running
 - waiting (do chờ I/O khi gọi printf)
 - ready
 - running
 - terminated



2. Trạng thái tiến trình

```
int main (int argc, char** argv)
{
    int i = 2;
    while (i < 5)
    {
        i++;
        if (i % 2 == 0)
        {
            printf ("Hello");
            printf ("Hi");
        }
        else
        {
            printf ("Bye");
        }
    }
    exit (0);
}
```

- Hỏi sau khi kết thúc thì tiến trình khi chạy từ chương trình trên đã nằm trong hàng đợi waiting bao nhiêu lần?

new – ready – running – waiting – ready
– running – waiting – ready – running –
waiting – ready – running – waiting –
ready – running – terminated



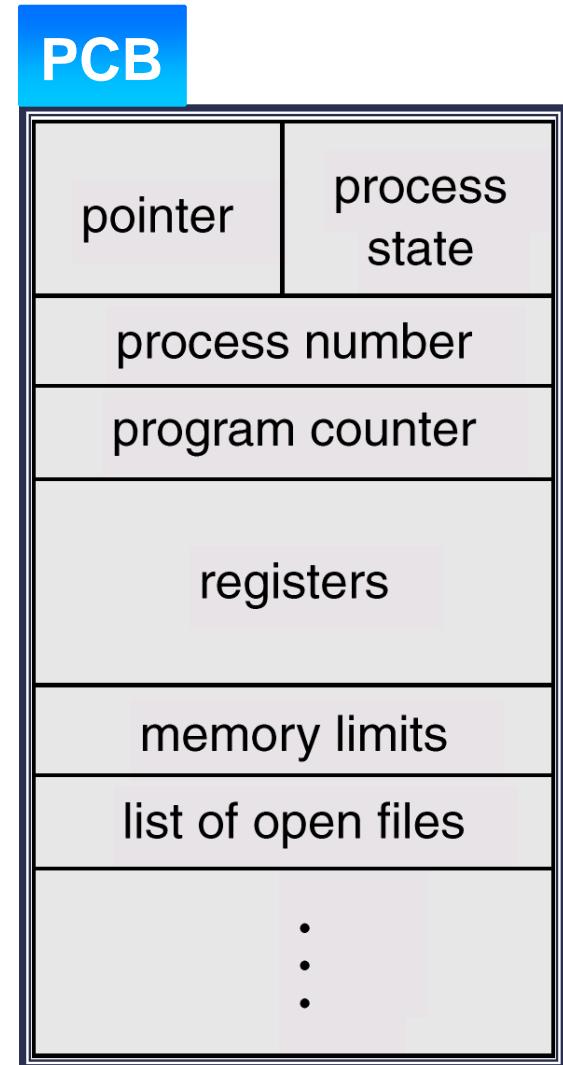
PROCESS CONTROL BLOCK

3



3. Process Control Block

- Mỗi tiến trình trong hệ thống đều được cấp phát một **Process Control Block** (PCB)
 - PCB là một trong các cấu trúc dữ liệu quan trọng nhất của hệ điều hành
- PCB gồm:
 - Trạng thái tiến trình: new, ready, running,...
 - Bộ đếm chương trình
 - Các thanh ghi
 - Thông tin lập thời biểu CPU: độ ưu tiên, ...
 - Thông tin quản lý bộ nhớ
 - Thông tin: lượng CPU, thời gian sử dụng,
 - Thông tin trạng thái I/O





4. ĐỊNH THỜI TIỀN TRÌNH

4



4. Định thời tiến trình

Yêu cầu đối với hệ điều hành về quản lý tiến trình

- Hỗ trợ sự thực thi luân phiên giữa nhiều tiến trình
 - Hiệu suất sử dụng CPU
 - Thời gian đáp ứng
- Phân phối tài nguyên hệ thống hợp lý
- Tránh deadlock, trì hoãn vô hạn định
- Cung cấp cơ chế giao tiếp và đồng bộ hoạt động các tiến trình
- Cung cấp cơ chế hỗ trợ user tạo/kết thúc tiến trình



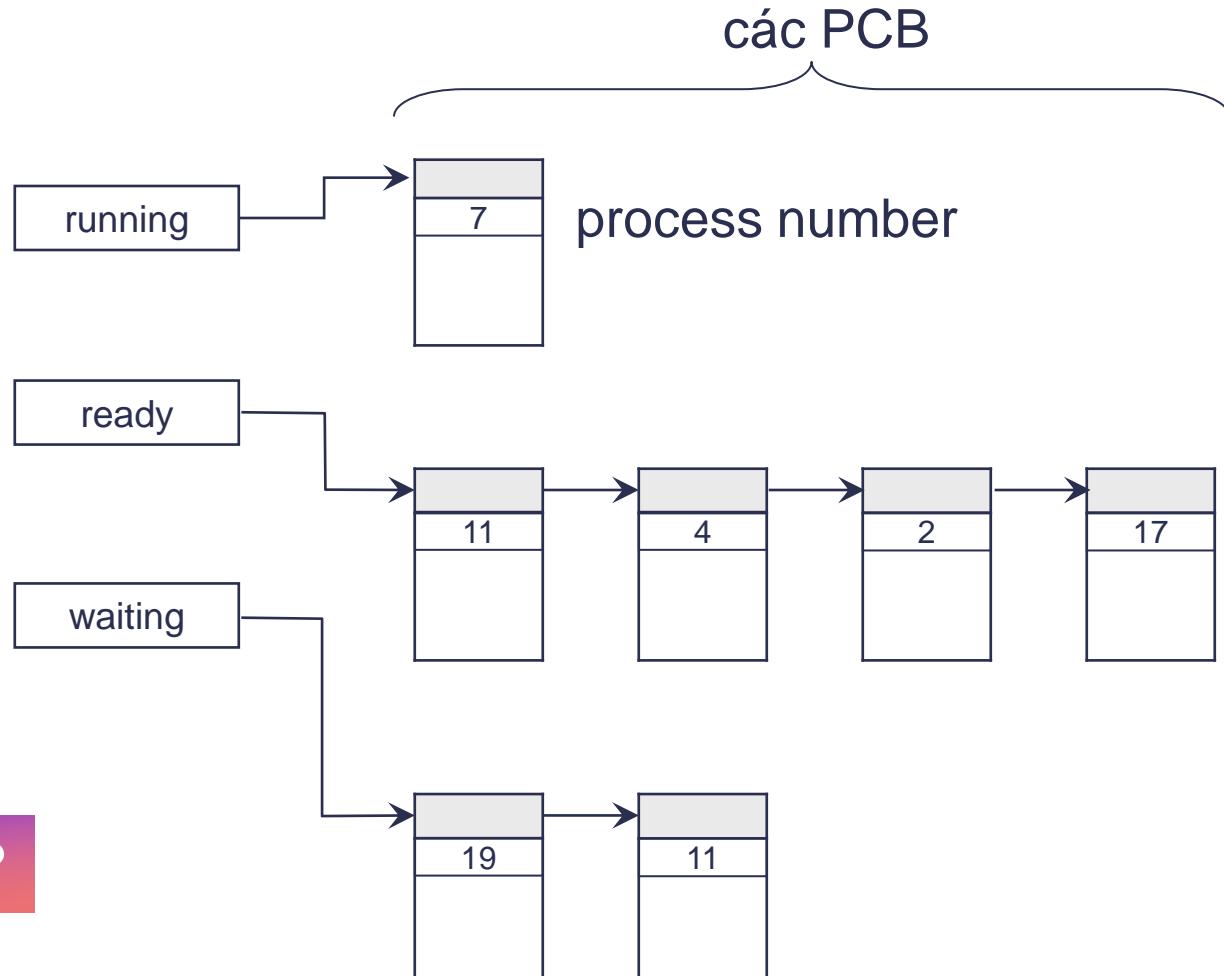
4. Định thời tiến trình

- Tại sao phải định thời?
 - Đa chương
 - Có vài tiến trình chạy tại các thời điểm
 - Mục tiêu: tận dụng tối đa CPU
 - Chia thời
 - User tương tác với mỗi chương trình đang thực thi
 - Mục tiêu: tối thiểu thời gian đáp ứng



4. Định thời tiến trình

Quản lý các tiến trình: các hàng đợi



Có trường hợp sai không?



4. ĐỊNH THỜI TIỀN TRÌNH

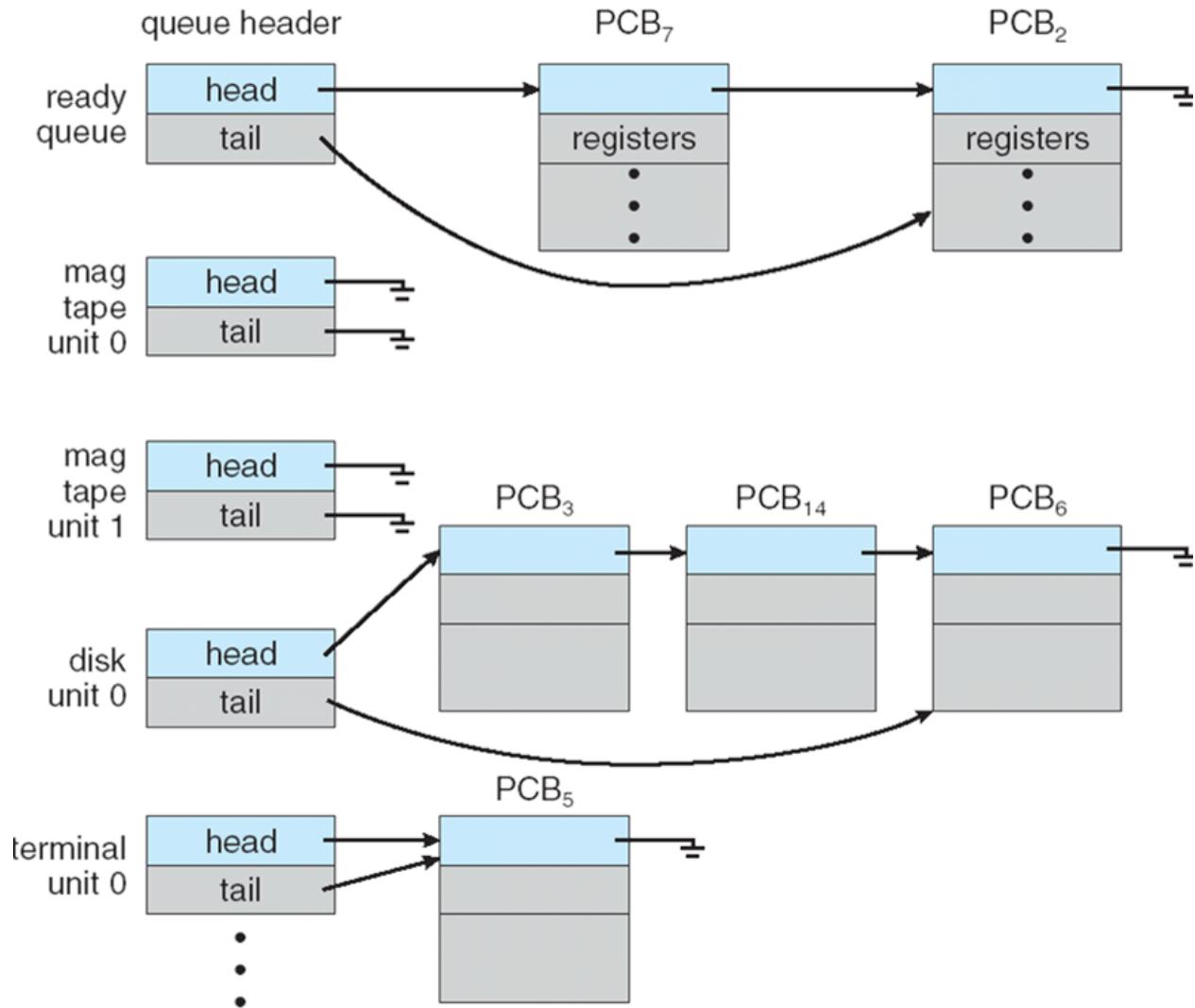
4.1. Các hàng đợi định thời

4



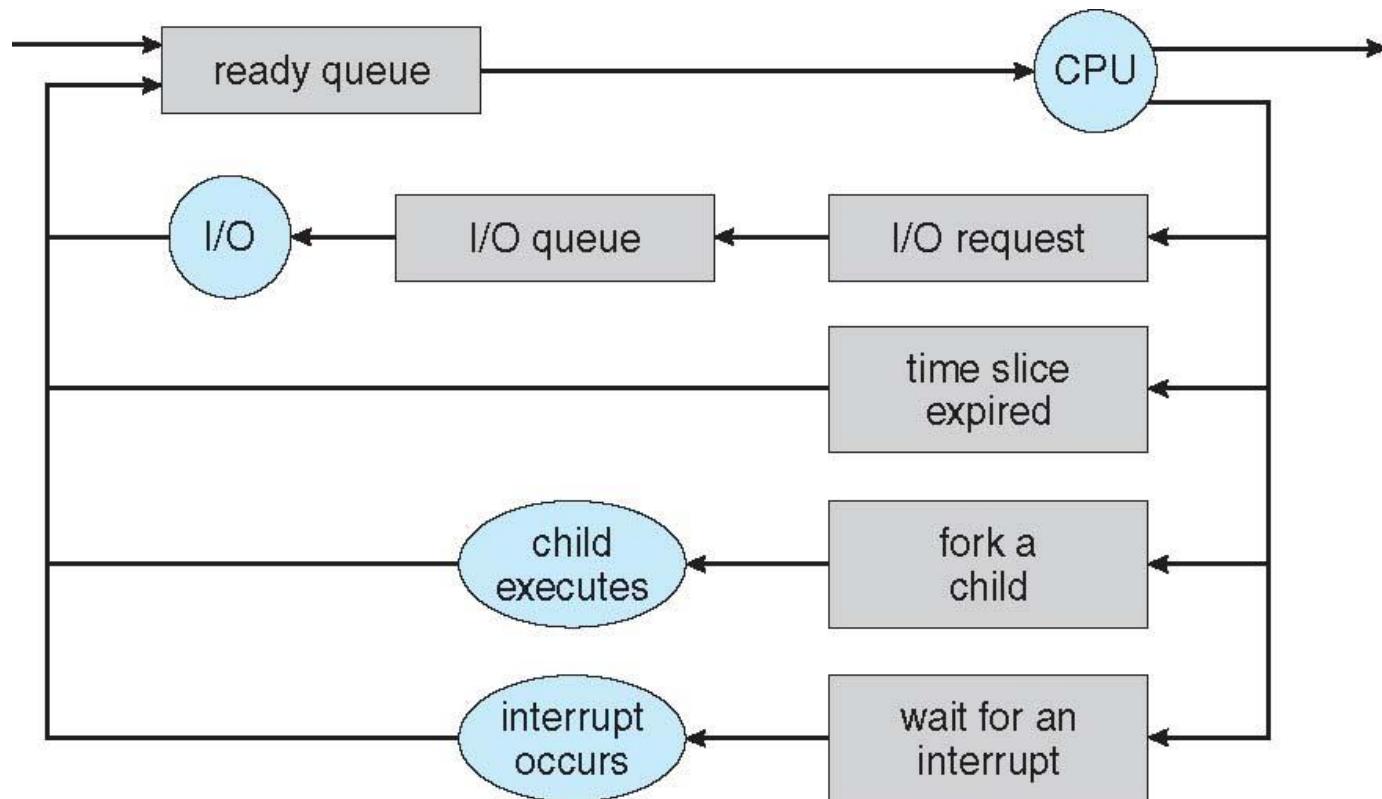
4.1. Các hàng đợi định thời

- Hàng đợi công việc - Job queue
- Hàng đợi sẵn sàng - Ready queue
- Hàng đợi thiết bị - Device queues
- ...





4.1. Các hàng đợi định thời



Lưu đồ hàng đợi của định thời tiến trình



4. ĐỊNH THỜI TIỀN TRÌNH

4.2. Bộ định thời

4



4.2. Bộ định thời

Phân loại bộ định thời

- **Bộ định thời công việc** (Job scheduler) hay **bộ định thời dài** (long-term scheduler)
- **Bộ định thời CPU** hay **bộ định thời ngắn**

Phân loại tiến trình

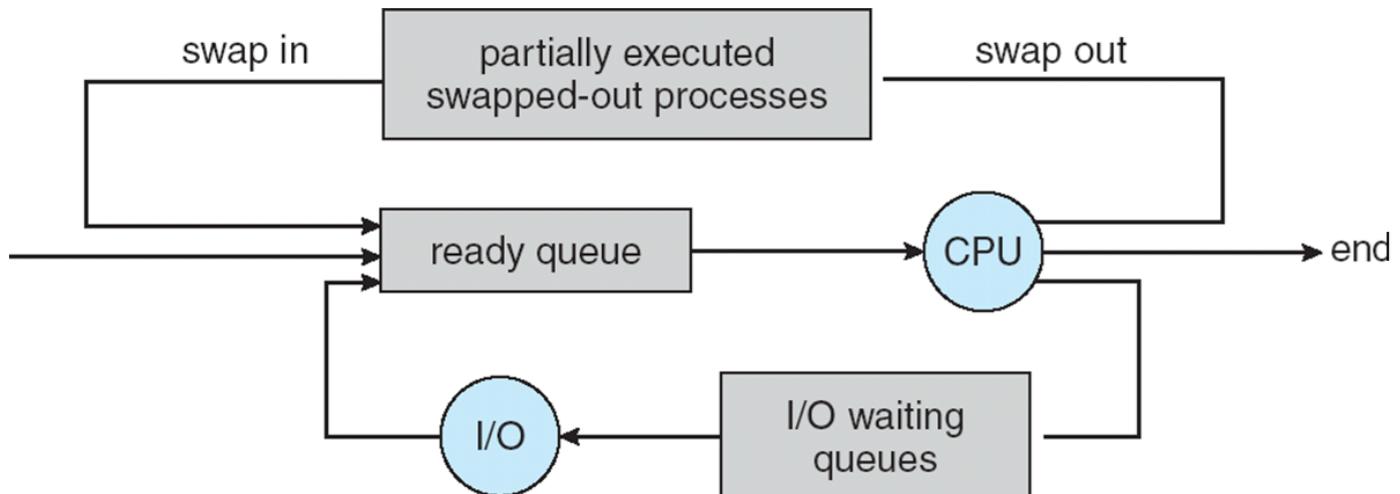
- Các tiến trình có thể mô tả như:
 - **tiến trình hướng I/O**
 - **tiến trình hướng CPU**
- Thời gian thực hiện khác nhau -> kết hợp hài hòa giữa chúng



4.2. Bộ định thời

Bộ định thời trung gian

- Đôi khi hệ điều hành (như time-sharing system) có thêm **medium-term scheduling** để điều chỉnh mức độ đa chương của hệ thống
- Medium-term scheduler:**
 - chuyển tiến trình từ bộ nhớ sang đĩa (swap out)
 - chuyển tiến trình từ đĩa vào bộ nhớ (swap in)



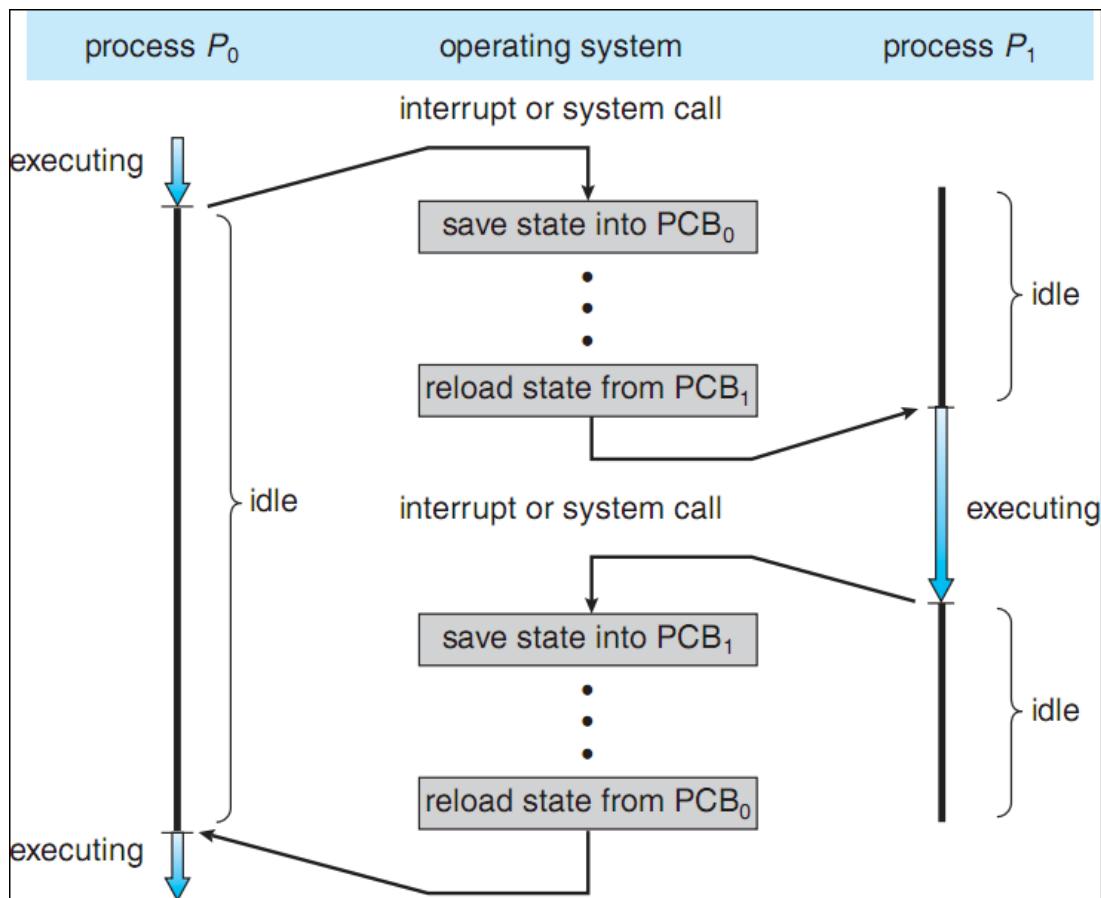


4.2. Bộ định thời

Chuyển ngữ cảnh (context switch)

Chuyển ngữ cảnh:

Quá trình CPU chuyển từ tiến trình này đến tiến trình khác





CÁC TÁC VỤ ĐỐI VỚI TIẾN TRÌNH

5



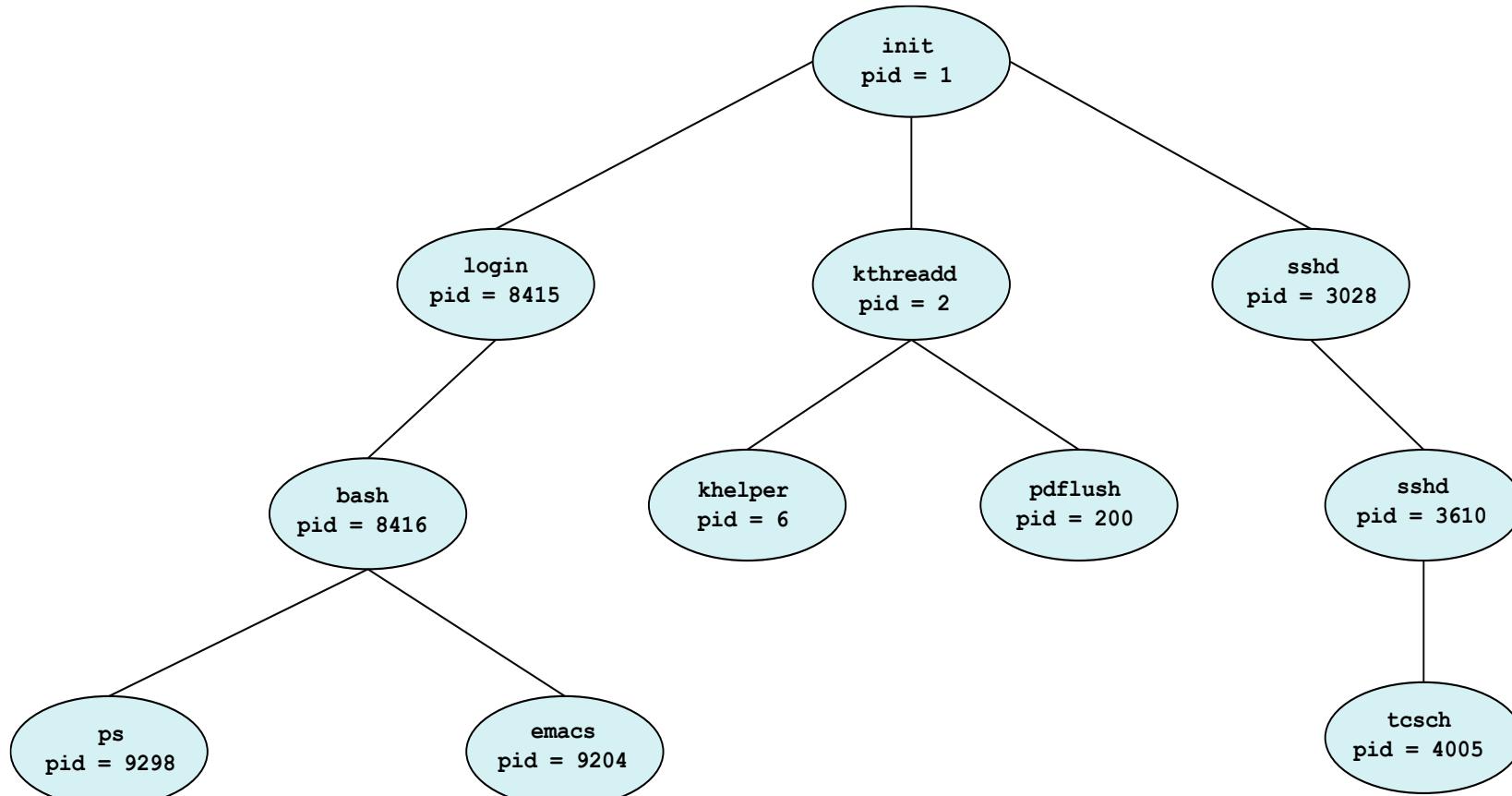
5. Các tác vụ đối với tiến trình

- Tạo tiến trình mới:
 - Một tiến trình có thể tạo nhiều tiến trình mới thông qua một **lời gọi hệ thống** create-process (vd: hàm fork trong Unix)
 - Ví dụ: (Unix) Khi user đăng nhập hệ thống, một command interpreter (shell) sẽ được tạo ra cho user
 - Tiến trình được tạo là tiến trình con của tiến trình tạo (tiến trình cha)
 - Quan hệ cha-con định nghĩa một cây tiến trình



5. Các tác vụ đối với tiến trình

Cây tiến trình trong Linux/Unix





5. Các tác vụ đối với tiến trình

- **Tạo tiến trình mới:**

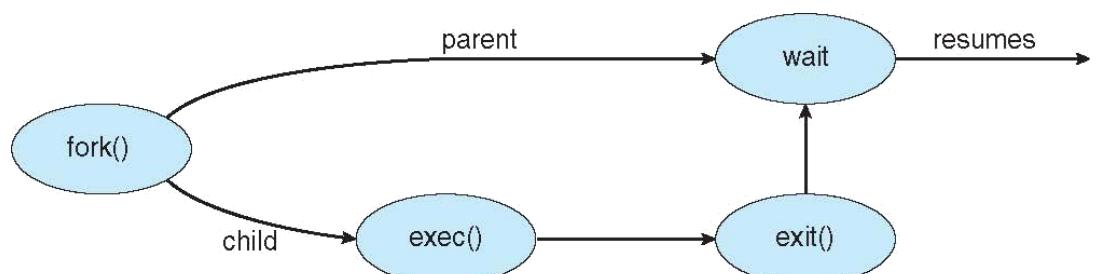
- Tiến trình con nhận tài nguyên: từ HĐH hoặc từ tiến trình cha
- Chia sẻ tài nguyên của tiến trình cha
 - tiến trình cha và con chia sẻ mọi tài nguyên
 - tiến trình con chia sẻ một phần tài nguyên của cha
- Trình tự thực thi
 - tiến trình cha và con thực thi đồng thời (concurrently)
 - tiến trình cha đợi đến khi các tiến trình con kết thúc



5. Các tác vụ đối với tiến trình

Về quan hệ cha/con

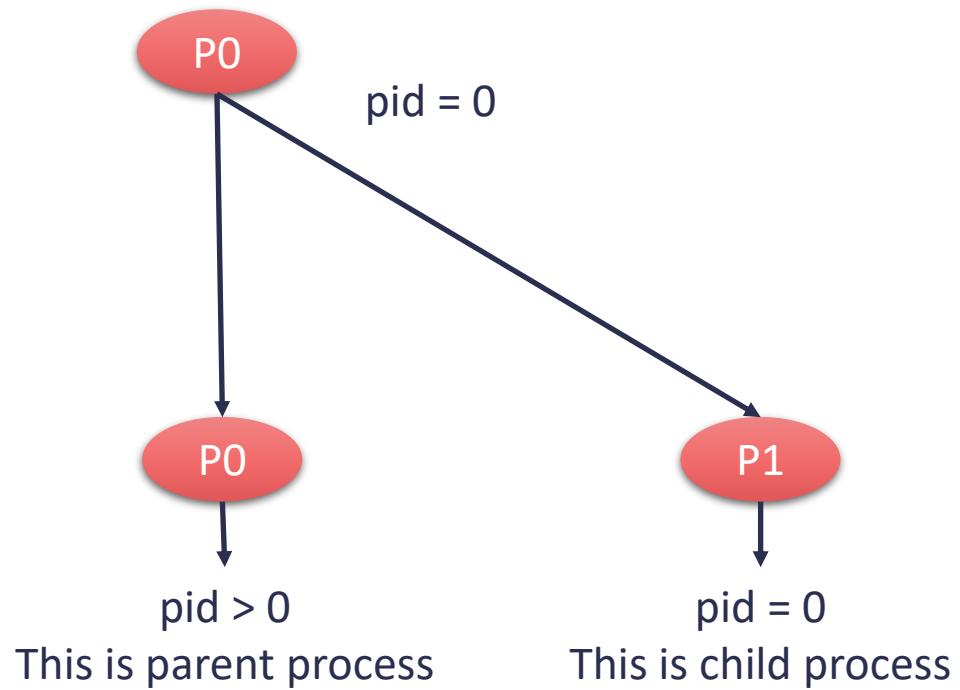
- Không gian địa chỉ:
 - Không gian địa chỉ của tiến trình con được nhân bản từ cha
 - Không gian địa chỉ của tiến trình con được khởi tạo từ template
- Ví dụ trong Unix/Linux
 - System call fork() tạo một tiến trình mới
 - System call exec() dùng sau fork() để nạp một chương trình mới vào không gian nhớ của tiến trình mới





5. Các tác vụ đối với tiến trình

Ví dụ tạo process với fork()



```
#include <stdio.h>
#include <unistd.h>

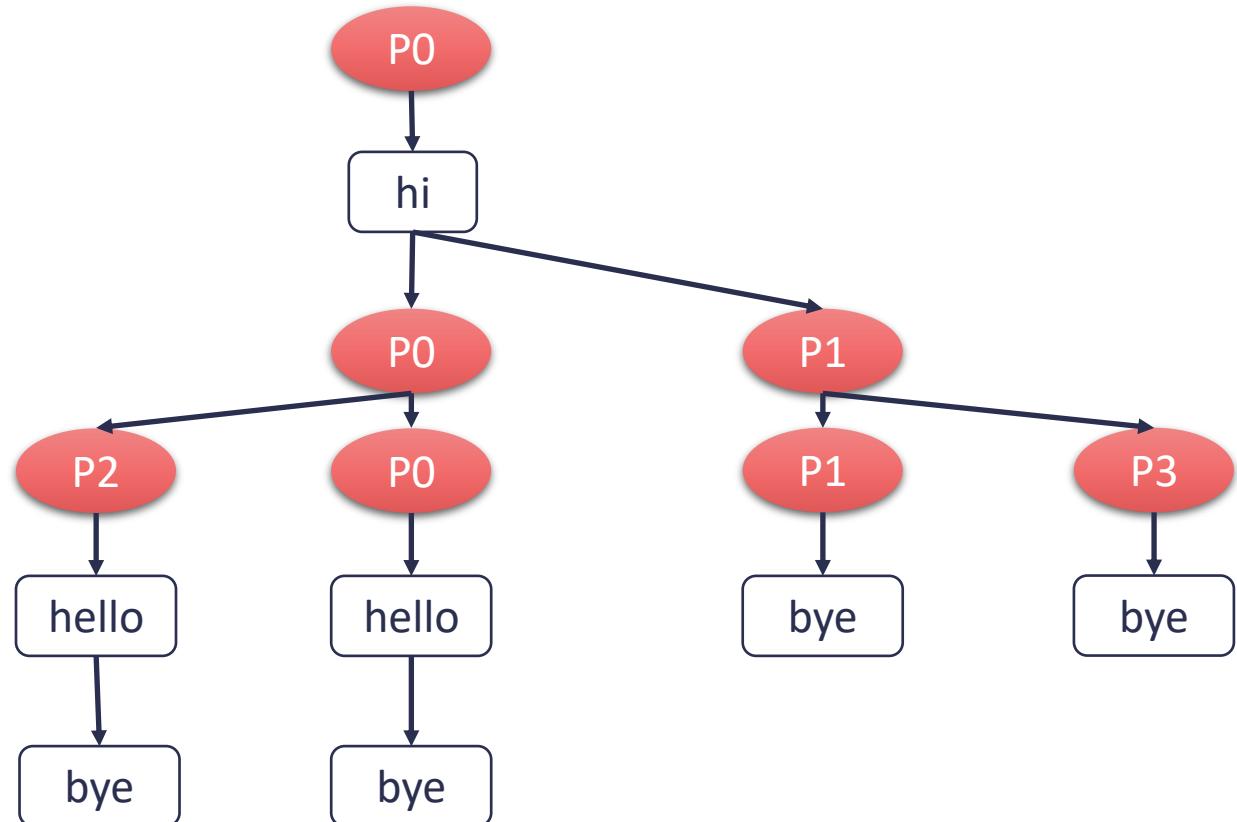
int main (int argc, char *argv[]) {
    int pid;
    /* create a new process */
    pid = fork();
    if (pid > 0) {
        printf("This is parent process");
        wait(NULL);
        exit(0);
    }
    else if (pid == 0) {
        printf("This is child process");
        execlp("/bin/ls", "ls", NULL);
        exit(0);
    }
    else { // pid < 0
        printf("Fork error\n");
        exit(-1);
    }
}
```



5. Các tác vụ đối với tiến trình

Ví dụ tạo process với fork() (tt)

```
#include <stdio.h>
#include <unistd.h>
int main (int argc, char *argv[])
{
    printf("hi");
    int pid = fork();
    if (pid > 0) {
        fork();
        printf("hello");
    }
    else
        fork();
        printf("bye");
}
```



Chương trình trên in ra những gì?



5. Các tác vụ đối với tiến trình

Ví dụ tạo process với fork() (tt)

```
int main (int argc, char **argv)
{
    int pid;
    printf("Toi la sinh vien lop IT007 \n");
    pid = fork();
    if (pid > 0)
    {
        printf("Tiến trình cha \n");
        fork();
    }
    else
    {
        printf("Sinh vien tu giac trong
kiem tra \n");
        if(fork() = 0 ){
            printf("Tiến trình con mới \n");
            fork();
        }
        else
            printf("Tiến trình cha mới \n");
        printf("Tiến trình IT007 mới \n");
    }
}
```



5. Các tác vụ đối với tiến trình

Ví dụ tạo process với fork() (tt)



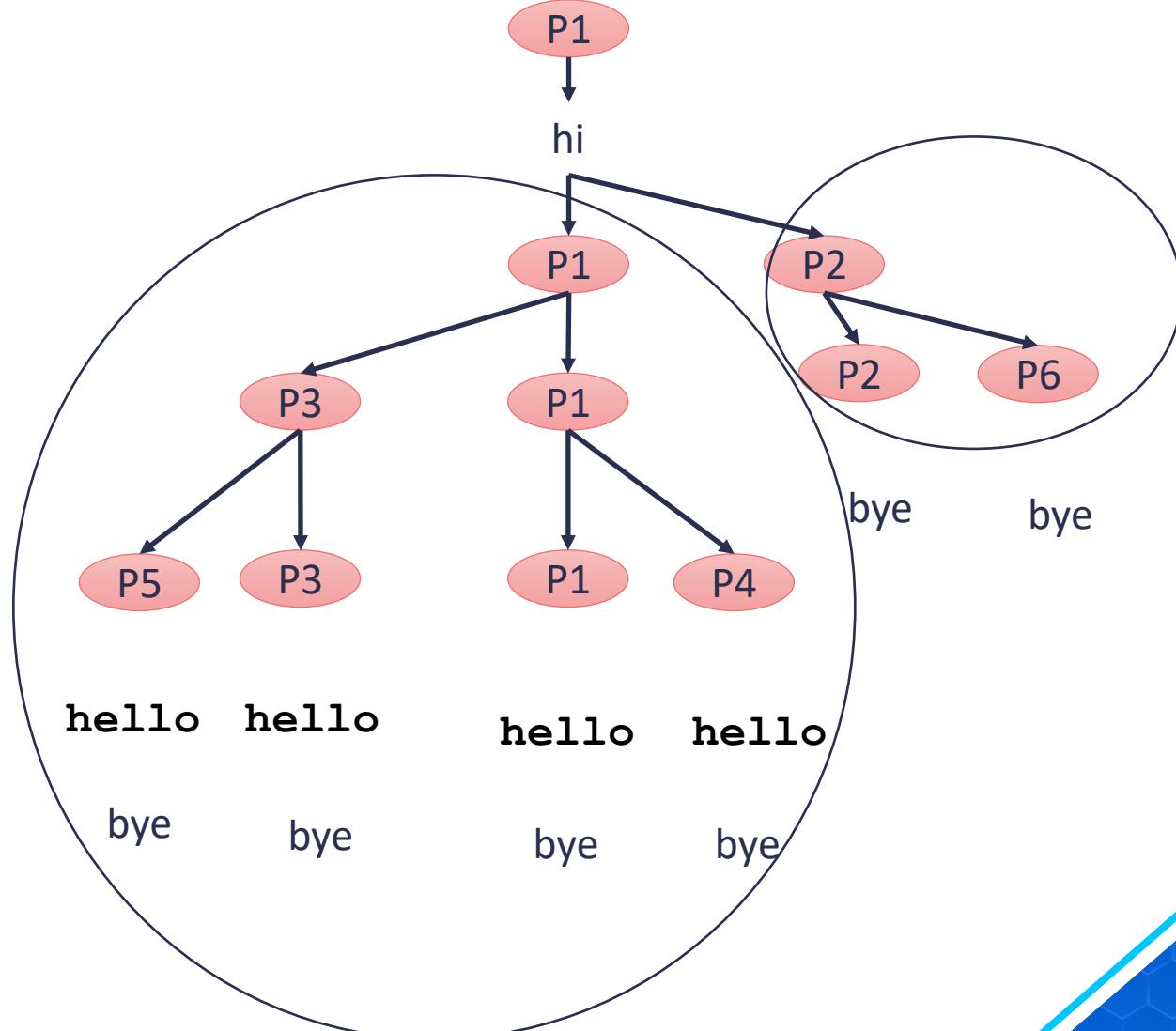
```
int main (int argc, char **argv)
{
    int pid;
    printf("Tôi là sinh viên lớp IT007.K22\n");
    pid = fork();
    if (pid > 0)
    {
        printf("Tiến trình cha \n");
        fork();
    }
    else
    {
        printf("Sinh viên tu giac trong kiem ta \n");
        if(fork() == 0)
            printf("Tiến trình con mới \n");
        fork();
    }
}
```



5. Các tác vụ đối với tiến trình

Ví dụ tạo process với fork() (tt)

```
#include <stdio.h>
#include <unistd.h>
int main (int argc, char *argv[])
{
    int pid;
    printf("hi");
    pid = fork();
    if (pid > 0) {
        fork();
        fork();
        printf("hello");
    }
    else
        fork();
    printf("bye");
}
```





5. Các tác vụ đối với tiến trình

- **Kết thúc tiến trình:**

- Tiến trình tự kết thúc
 - Tiến trình kết thúc khi thực thi lệnh cuối và gọi system routine exit
- Tiến trình kết thúc do tiến trình khác (có đủ quyền, vd: tiến trình cha của nó)
 - Gọi system routine abort với tham số là pid (process identifier) của tiến trình cần được kết thúc
- Hệ điều hành thu hồi tất cả các tài nguyên của tiến trình kết thúc (vùng nhớ, I/O buffer,...)



CỘNG TÁC GIỮA CÁC TIẾN TRÌNH

6



6. Cộng tác giữa các tiến trình

Các tác vụ đối với tiến trình

- Trong tiến trình thực thi, các tiến trình có thể cộng tác (cooperate) để hoàn thành công việc
- Các tiến trình cộng tác để
 - Chia sẻ dữ liệu (information sharing)
 - Tăng tốc tính toán (computational speedup)
 - Nếu hệ thống có nhiều CPU, chia công việc tính toán thành nhiều công việc tính toán nhỏ chạy song song
 - Thực hiện một công việc chung
 - Xây dựng một phần mềm phức tạp bằng cách chia thành các module/process hợp tác nhau
- Sự cộng tác giữa các tiến trình yêu cầu hệ điều hành hỗ trợ cơ chế giao tiếp và cơ chế đồng bộ hoạt động của các tiến trình



CỘNG TÁC GIỮA CÁC TIẾN TRÌNH

6.1. Giao tiếp liên tiến trình (IPC)

6

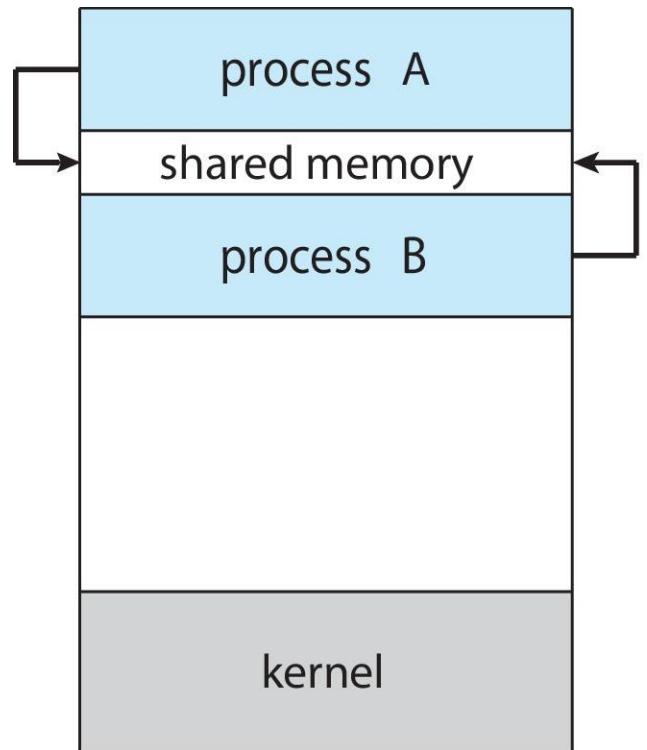


6.1. Giao tiếp liên tiến trình (IPC)

- IPC - Inter Process Communication: là cơ chế cung cấp bởi hệ điều hành nhằm giúp các tiến trình:
 - Giao tiếp với nhau
 - Đồng bộ hoạt động
- Hai mô hình IPC:
 - Shared memory
 - Message passing

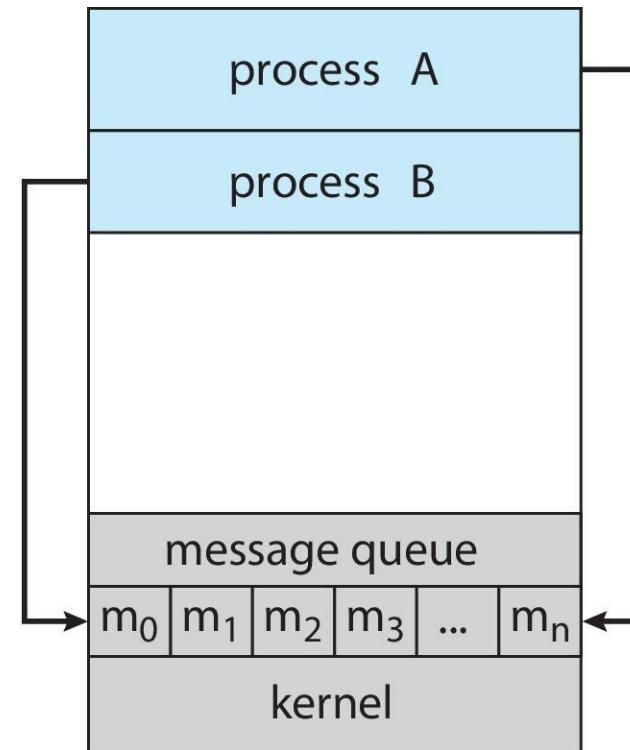
6.1. Giao tiếp liên tiến trình (IPC)

(a) Shared memory.



(a)

(b) Message passing.



(b)



CỘNG TÁC GIỮA CÁC TIẾN TRÌNH

6.2. Bộ nhớ được chia sẻ - Shared memory

6



6.2. Bộ nhớ được chia sẻ - Shared memory

- Một vùng nhớ dùng chung (được chia sẻ chung) giữa các tiến trình cần giao tiếp với nhau.
- Quá trình giao tiếp được thực hiện dưới sự điều khiển của các tiến trình, không phải của hệ điều hành.
- Cần có cơ chế đồng bộ hoạt động của các tiến trình khi chúng cùng truy xuất bộ nhớ dùng chung.



CỘNG TÁC GIỮA CÁC TIẾN TRÌNH

6.3. Hệ thống truyền thông điệp - Message passing

6



6.3. Hệ thống truyền thông điệp - Message passing

Làm thế nào để các tiến trình giao tiếp nhau?

- Đặt tên (Naming)
 - Giao tiếp trực tiếp
 - send(P, msg): gửi thông điệp đến tiến trình P
 - receive(Q, msg): nhận thông điệp đến từ tiến trình Q
 - Giao tiếp gián tiếp: thông qua mailbox hay port
 - send(A, msg): gửi thông điệp đến mailbox A
 - receive(Q, msg): nhận thông điệp từ mailbox B
- Đồng bộ hóa (Synchronization): blocking send, nonblocking send, blocking receive, nonblocking receive



6.3. Hệ thống truyền thông điệp - Message passing

Làm thế nào để các tiến trình giao tiếp nhau?

- Tạo vùng đệm (Buffering): dùng queue để tạm chứa các message
 - Khả năng chứa là 0 (Zero capacity hay no buffering)
 - Bounded capacity: độ dài của queue là giới hạn
 - Unbounded capacity: độ dài của queue là không giới hạn



TIỂU TRÌNH

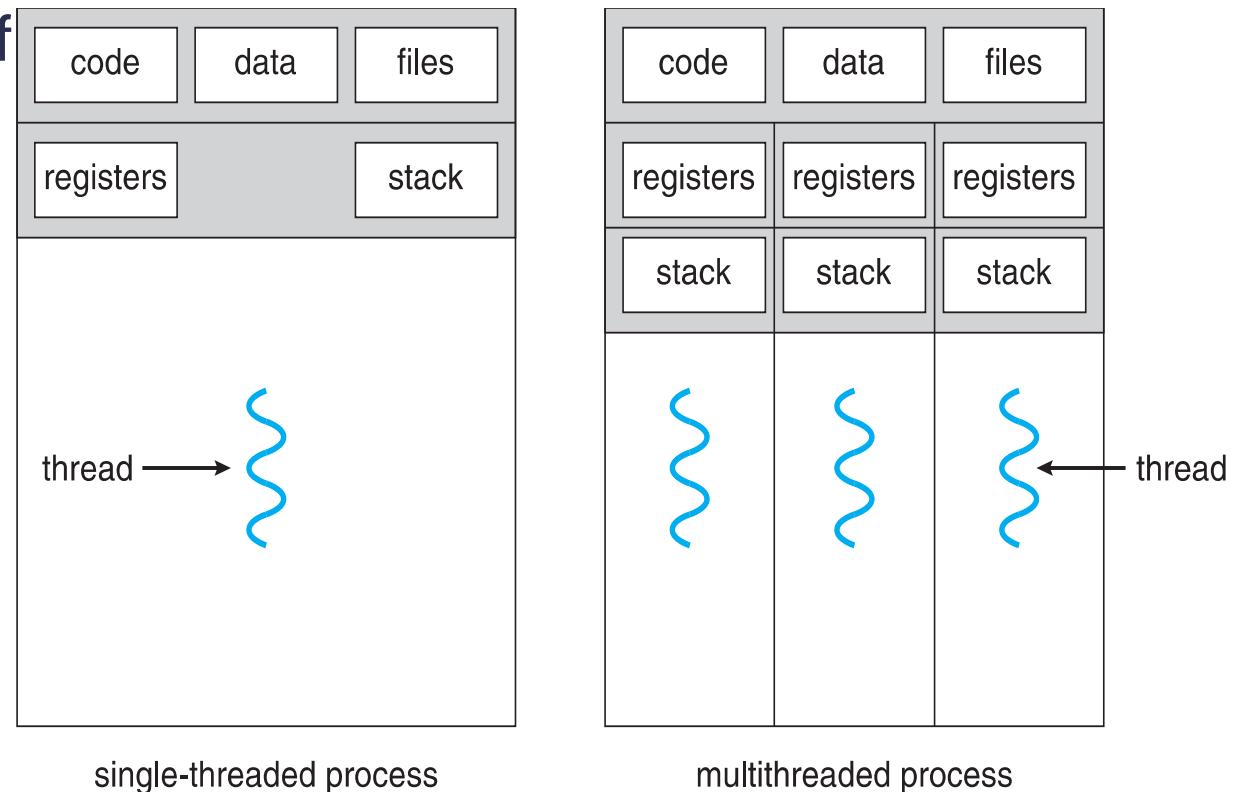
7.1. Tổng quan về tiểu trình

7



7.1. Tổng quan về tiêu trình

- Tiêu trình là một đơn vị cơ bản sử dụng CPU gồm:
 - Thread ID, PC, Registers, Stack và chia sẻ chung code, data, resources (f





7.1. Tổng quan về tiêu trình

PCB và TCB trong mô hình multithreads

PCB	pid	Thread Control Block TCB
	Threads list	
	Context (Mem, global ressources...)	
	Relatives (Dad, children)	
	Scheduling statistic	
	tid	
	State (State, details)	
	Context (IP, local stack...)	



7.1. Tổng quan về tiểu trình

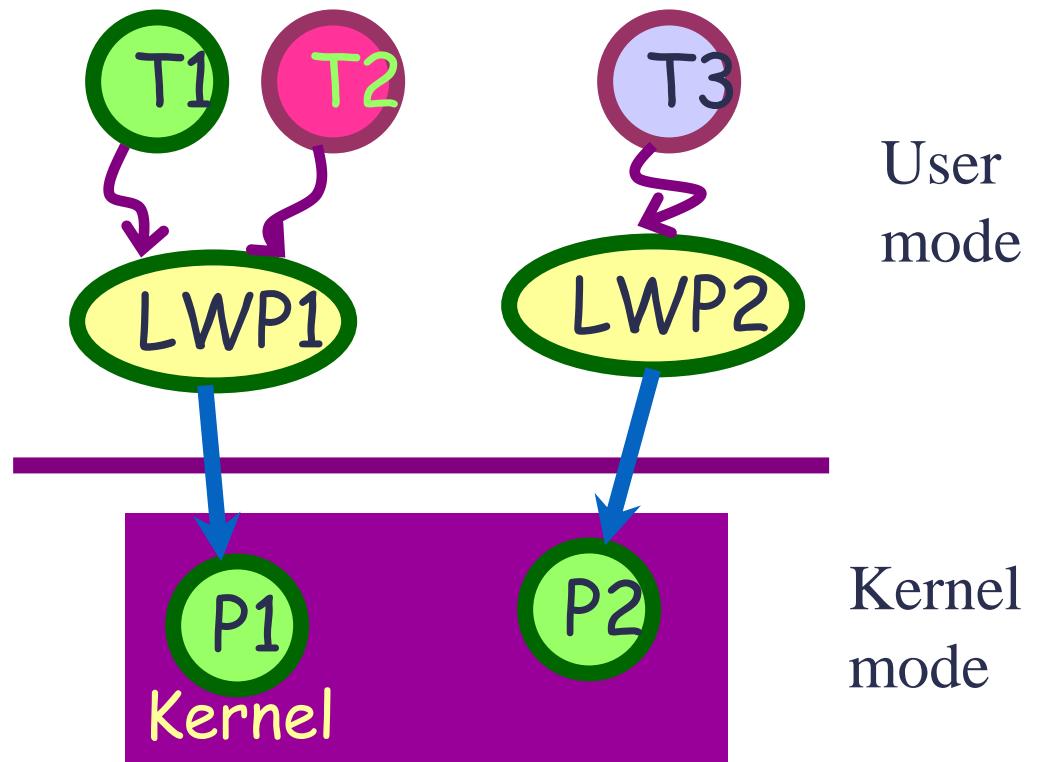
Lợi ích của tiến trình đa luồng

- Đáp ứng nhanh: cho phép chương trình tiếp tục thực thi khi một bộ phận bị khóa hoặc một hoạt động dài
- Chia sẻ tài nguyên: tiết kiệm không gian nhớ
- Kinh tế: tạo và chuyển ngữ cảnh nhanh hơn tiến trình
 - Ví dụ: Trong Solaris 2, tạo process chậm hơn 30 lần, chuyển chậm hơn 5 lần so với thread
 - Trong multiprocessor: có thể thực hiện song song



7.1. Tổng quan về tiểu trình

Tiểu trình người dùng (User thread)



Khái niệm tiểu trình được hỗ trợ bởi một thư viện hoạt động trong user mode



7.1. Tổng quan về tiểu trình

Tiểu trình hạt nhân (Kernel thread)



Khái niệm tiểu trình được xây dựng bên trong hạt nhân



TIỂU TRÌNH

7.2. Các mô hình đa tiêu trình

7



7.2. Các mô hình đa tiêu trình

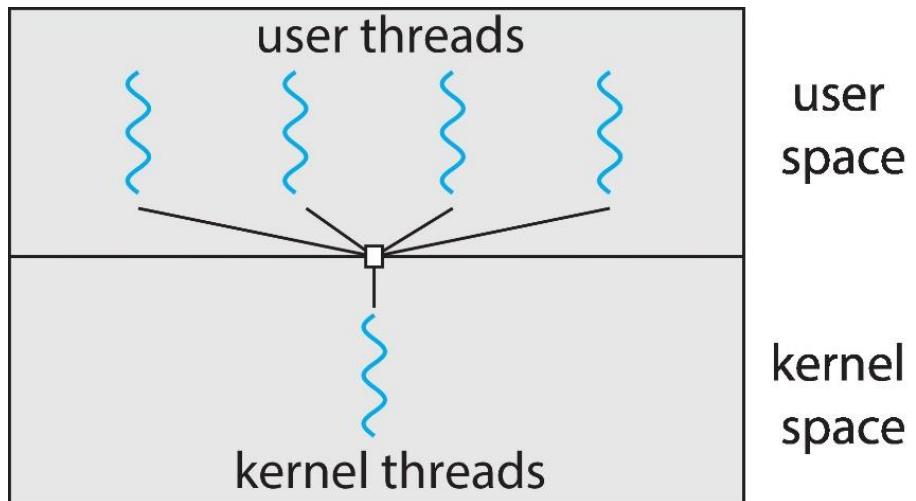
- Nhiều – Một (Many-to-One)
- Một – Một (One-to-One)
- Nhiều – Nhiều (Many-to-Many)



7.2. Các mô hình đa tiêu trình

Mô hình Nhiều – Một (Many-to-One)

- Nhiều tiêu trình người dùng được ánh xạ đến một tiêu trình hạt nhân.
- Một tiêu trình bị block sẽ dẫn đến tất cả tiêu trình bị block.
- Các tiêu trình không thể chạy song song trên các hệ thống đa lõi bởi vì chỉ có một tiêu trình có thể truy xuất nhân tại một thời điểm.
- Rất ít hệ thống sử dụng mô hình này.

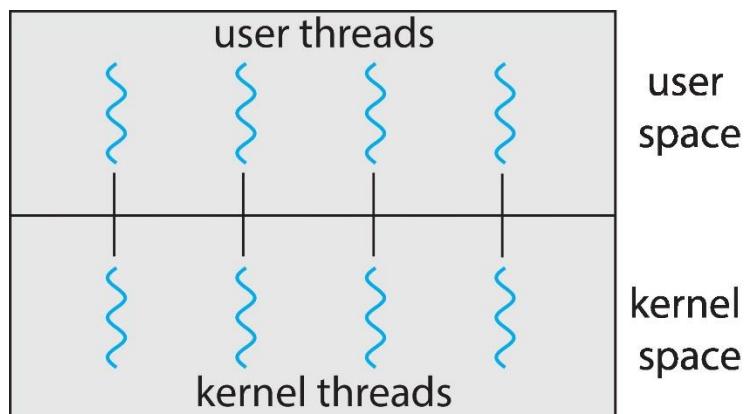




7.2. Các mô hình đa tiêu trình

Mô hình Một – Một (One-to-One)

- Mỗi tiêu trình người dùng ứng với một tiêu trình hạt nhân.
- Tạo một tiêu trình người dùng cũng đồng thời tạo một tiêu trình hạt nhân.
- Tính đồng thời (concurrency) tốt hơn mô hình nhiều – một vì các tiêu trình khác vẫn hoạt động bình thường khi một tiêu trình bị block.
- Nhược điểm: Số lượng tiêu trình của mỗi tiến trình có thể bị hạn chế.
- Nhiều hệ điều hành sử dụng:
 - Windows
 - Linux

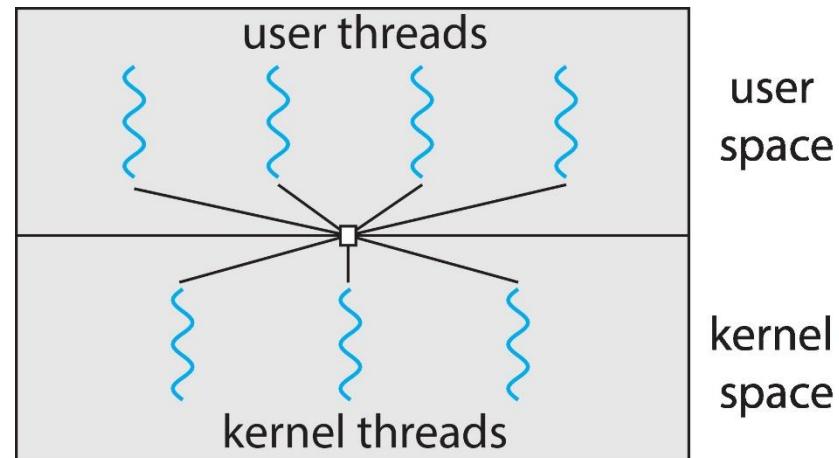




7.2. Các mô hình đa tiêu trình

Mô hình Nhiều – Nhiều (Many-to-Many)

- Các tiêu trình người dùng được ánh xạ với nhiều tiêu trình hạt nhân.
- Cho phép hệ điều hành tạo đủ số lượng tiêu trình hạt nhân => Giải quyết được hạn chế của 2 mô hình trên.
- Khó cài đặt nên ít phổ biến.





Tóm tắt lại nội dung buổi học

- Khái niệm cơ bản
- Trạng thái tiến trình
- Khối điều khiển tiến trình
- Định thời tiến trình
- Các tác vụ đối với tiến trình
- Sự cộng tác giữa các tiến trình
- Giao tiếp giữa các tiến trình
- Tiểu trình



Câu hỏi ôn tập chương 3

- Process control block chứa những thông tin gì?
- Các tác vụ đối với tiến trình?
- Tại sao phải định thời, có mấy loại bộ định thời?



Câu hỏi ôn tập chương 3 (tt)

Nêu cụ thể các trạng thái của tiến trình?

```
/* test.c */  
  
int main(int argc, char** argv)  
{  
    printf("Hello world\n");  
  
    scanf(" Nhập c = %d",&c);  
  
    exit(0);  
}
```



Câu hỏi ôn tập chương 3 (tt)

- Chương trình này in ra những chữ gì?

```
#include <stdio.h>
#include <unistd.h>
int main (int argc, char *argv[])
{
    int pid;
    pid = fork();
    printf(" so 1");
    printf(" so 2");
    fork();
    if (pid < 0) {
        printf("hello");
        fork();
    }else
        fork();
    printf("bye");
}
```



THẢO LUẬN

