



HỆ ĐIỀU HÀNH

CHƯƠNG 3: TIẾN TRÌNH (PHẦN 2)

Trình bày các tác vụ cơ bản trên tiến trình, cách các tiến trình giao tiếp với nhau và khái niệm về mô hình tiểu trình



MỤC TIÊU

4. Biết được các tác vụ cơ bản của một tiến trình
5. Hiểu được cách giao tiếp giữa các tiến trình
6. Trình bày được mô hình tiểu trình



NỘI DUNG

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



CÁC TÁC VỤ ĐỐI VỚI TIẾN TRÌNH

5



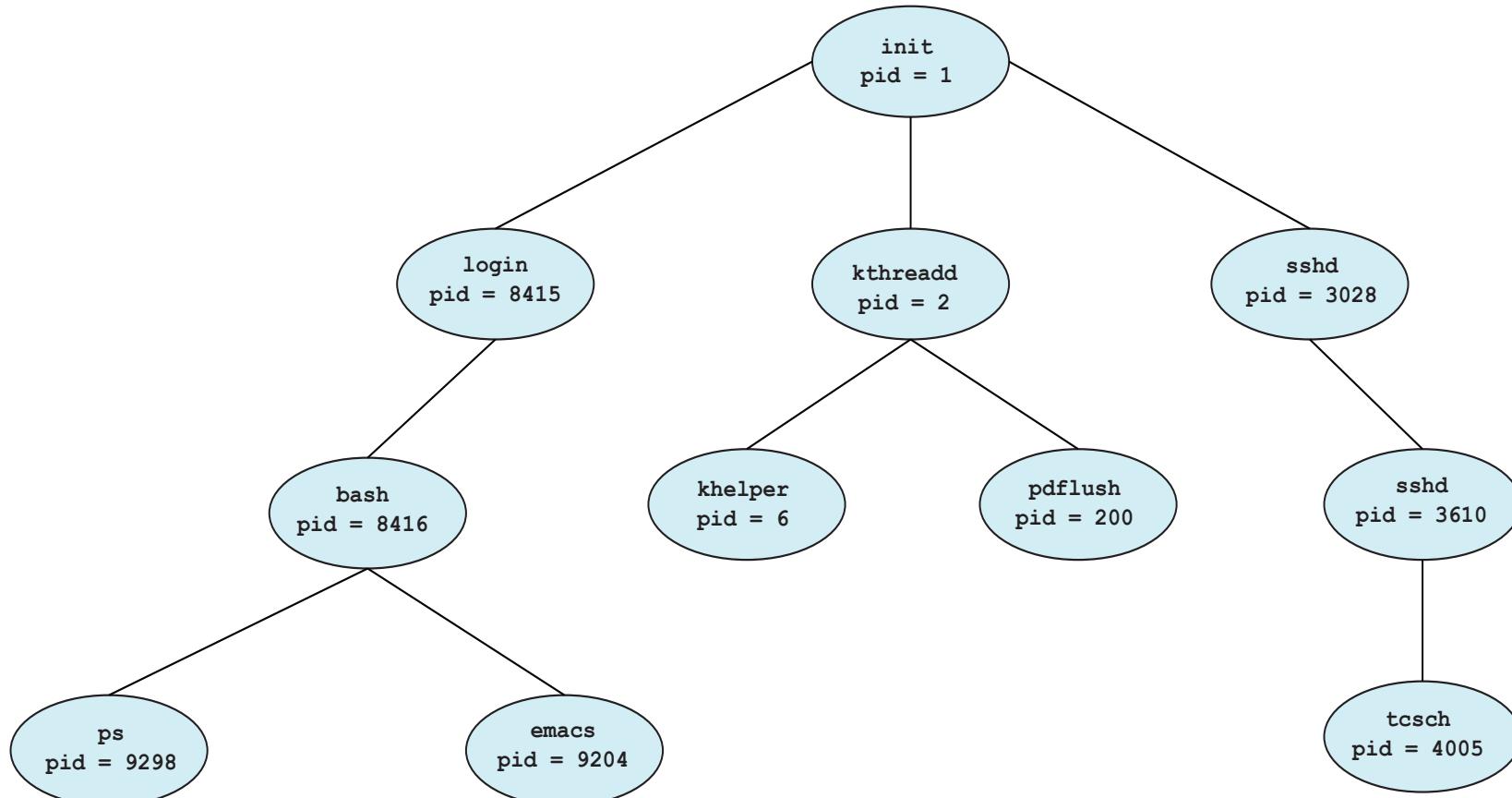
5. Các tác vụ đối với tiến trình

- 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





CÁC TÁC VỤ ĐỐI VỚI TIẾN TRÌNH

5.1. Tạo tiến trình

5



5.1. Tạo tiến trình

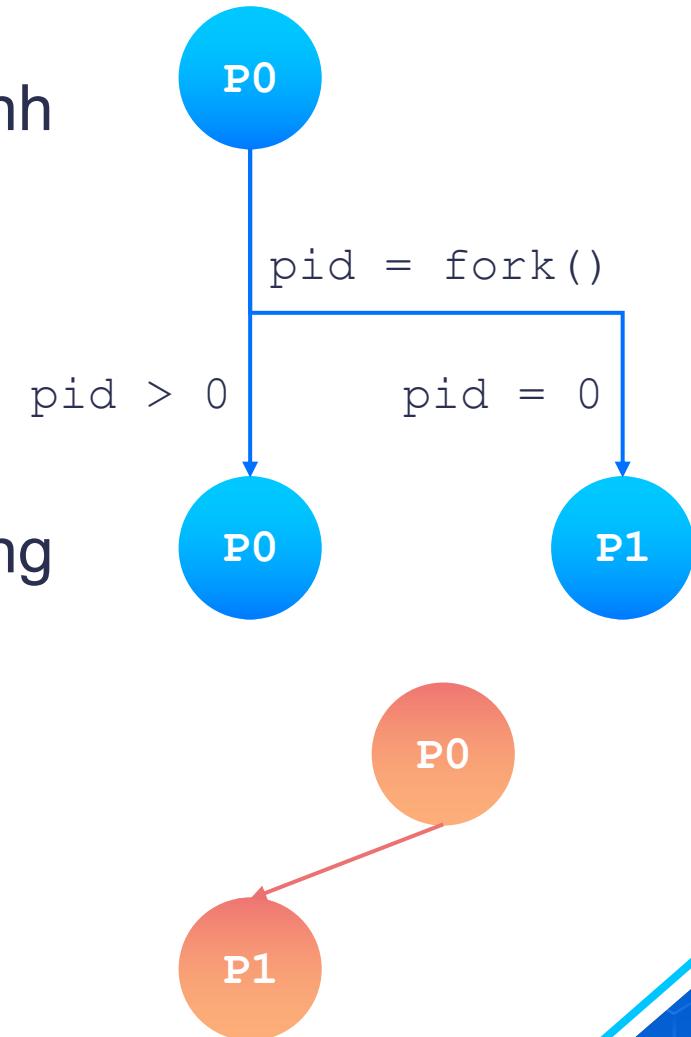
- Tiến trình con nhận tài nguyên: từ OS hoặc từ tiến trình cha.
- Chia sẻ tài nguyên của tiến trình cha:
 - tiến trình cha và con chia sẻ mọi tài nguyên.
 - tiến trình con chia sẻ một phần tài nguyên của cha.
- Trình tự thực thi:
 - tiến trình cha và con thực thi đồng thời (concurrently).
 - tiến trình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



5.1. Tạo tiến trình

Hàm fork()

- Tiến trình con **sao chép không gian địa chỉ** của tiến trình cha → Tiến trình con:
 - Sao chép toàn bộ source code của tiến trình cha
 - Sao chép giá trị của các biến đã được tạo
 - Bắt đầu thực thi từ vị trí mà tiến trình được tạo
- **Giá trị trả về** của hàm **fork** sẽ thuộc 01 trong 03 trường hợp:
 - **Lớn hơn 0**: cho biết đây là tiến trình cha
 - **Bằng 0**: cho biết đây là tiến trình con
 - **Nhỏ hơn 0**: hàm fork() thất bại

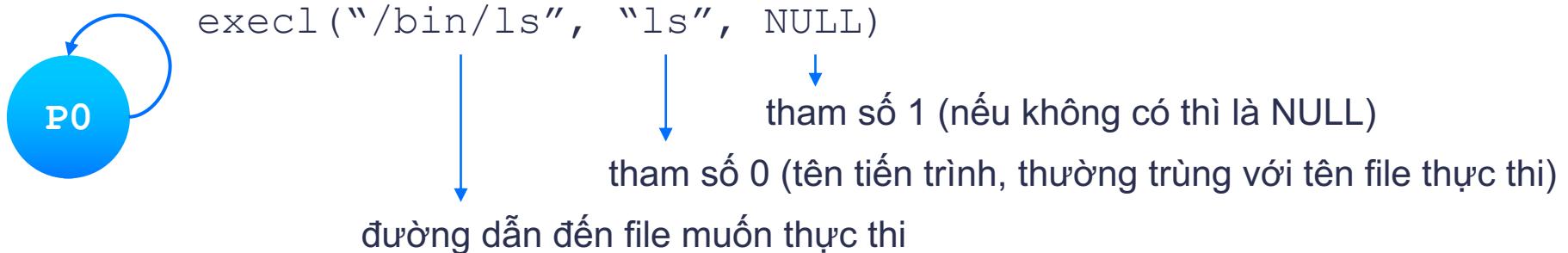




5.1. Tạo tiến trình

Họ hàm exec ()

- Nạp một tác vụ mới vào không gian địa chỉ của tiến trình gọi hàm
 - Tác vụ mới sẽ được ghi đè vào không gian địa chỉ của tiến trình
 - Tiến trình thực thi tác vụ mới thay vì source code ban đầu

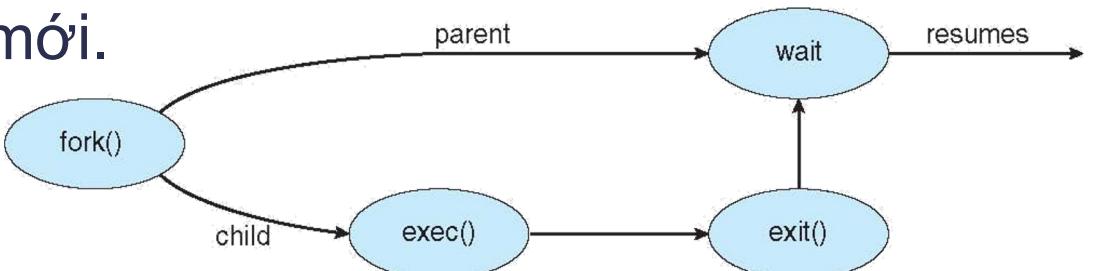




5.1. Tạo tiến trình

Về quan hệ cha/con

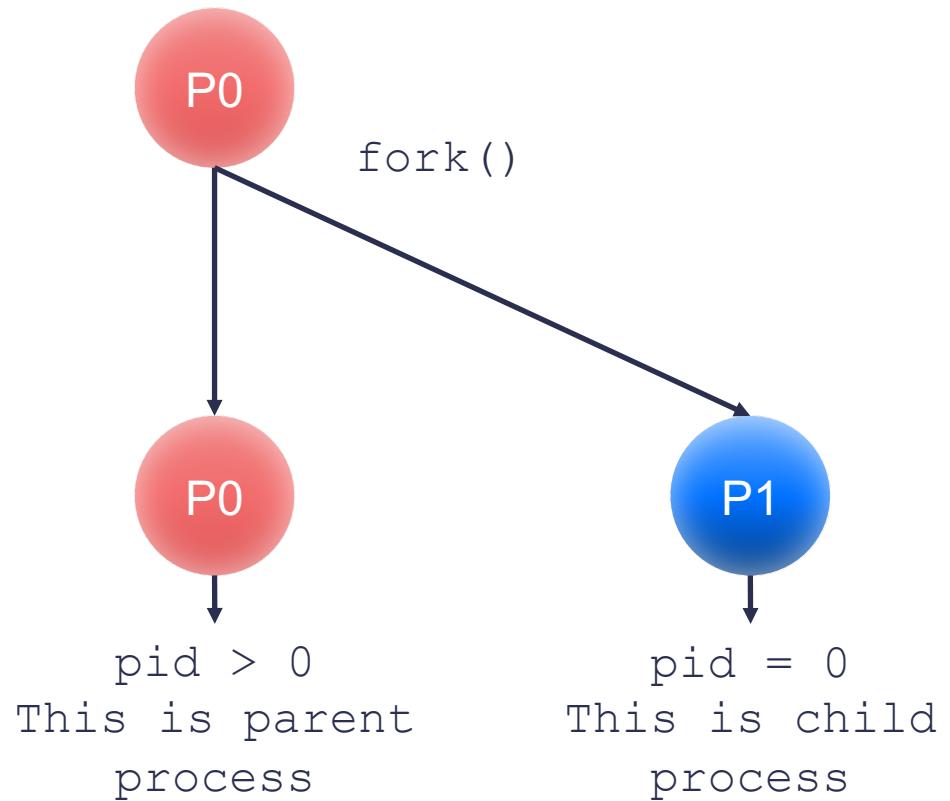
- Không gian địa chỉ:
 - Không gian địa chỉ của tiến trình con được nhân bản từ cha.
 - Không gian địa chỉ của tiến trình con được khởi tạo từ template.
- Ví dụ trong Unix/Linux
 - System call **fork()** tạo một tiến trình mới.
 - System call **exec()** dùng sau **fork()** để nạp một chương trình mới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





5.1. Tạo tiến trình

Ví dụ tạo process với fork()

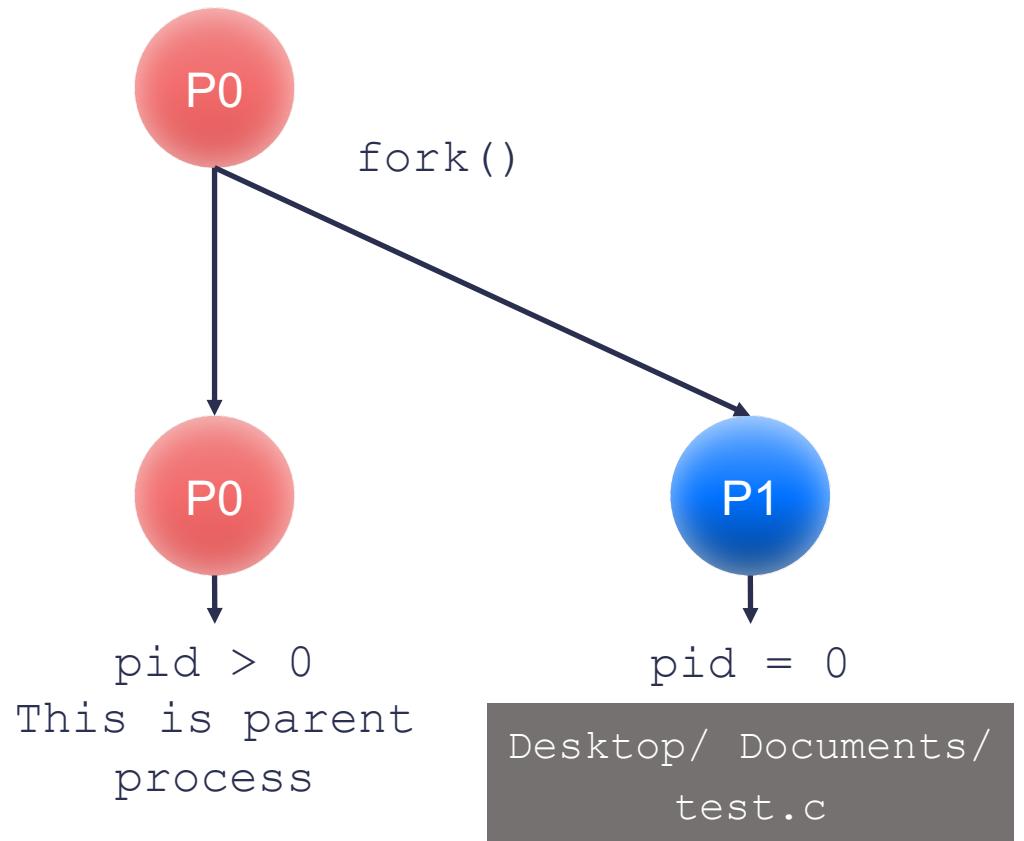


```
#include <stdio.h>
#include <unistd.h>
int main (int argc, char *argv[])
{
    int pid;
    /* create a new process */
    pid = fork();
    if (pid > 0)
    {
        printf("This is parent process");
        wait(NULL);
        exit(0);
    }
    else if (pid == 0)
    {
        printf("This is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



5.1. Tạo 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)
    {
        execlp("/bin/ls", "ls", NULL);
        printf("This is child process");
        exit(0);
    }
    else
    { // pid < 0
        printf("Fork error\n");
        exit(-1);
    }
}
```

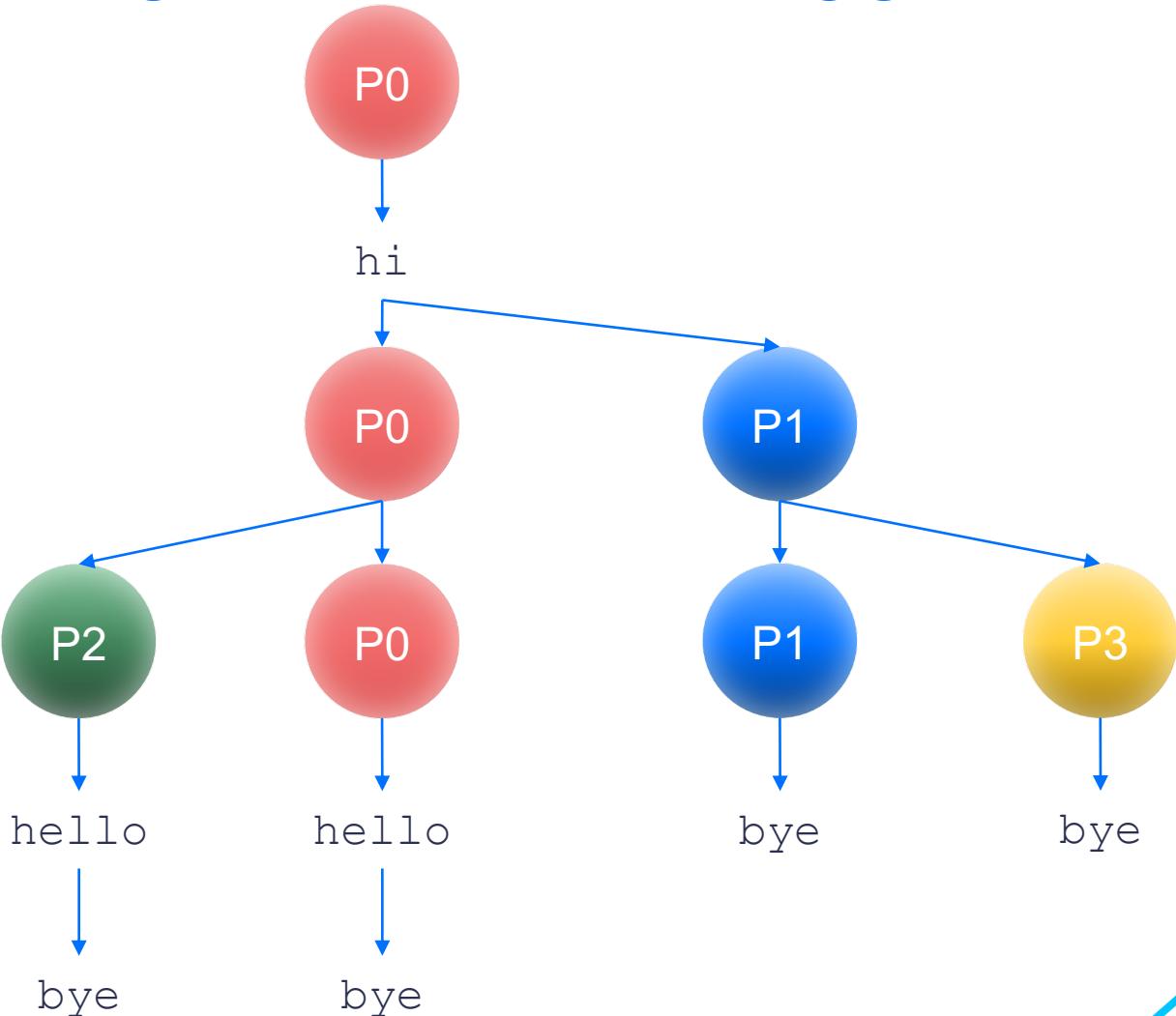


5.1. Tạo tiến trình

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 dưới in ra những gì?





5.1. Tạo 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("Tien trinh cha \n");
        fork();
    }
```

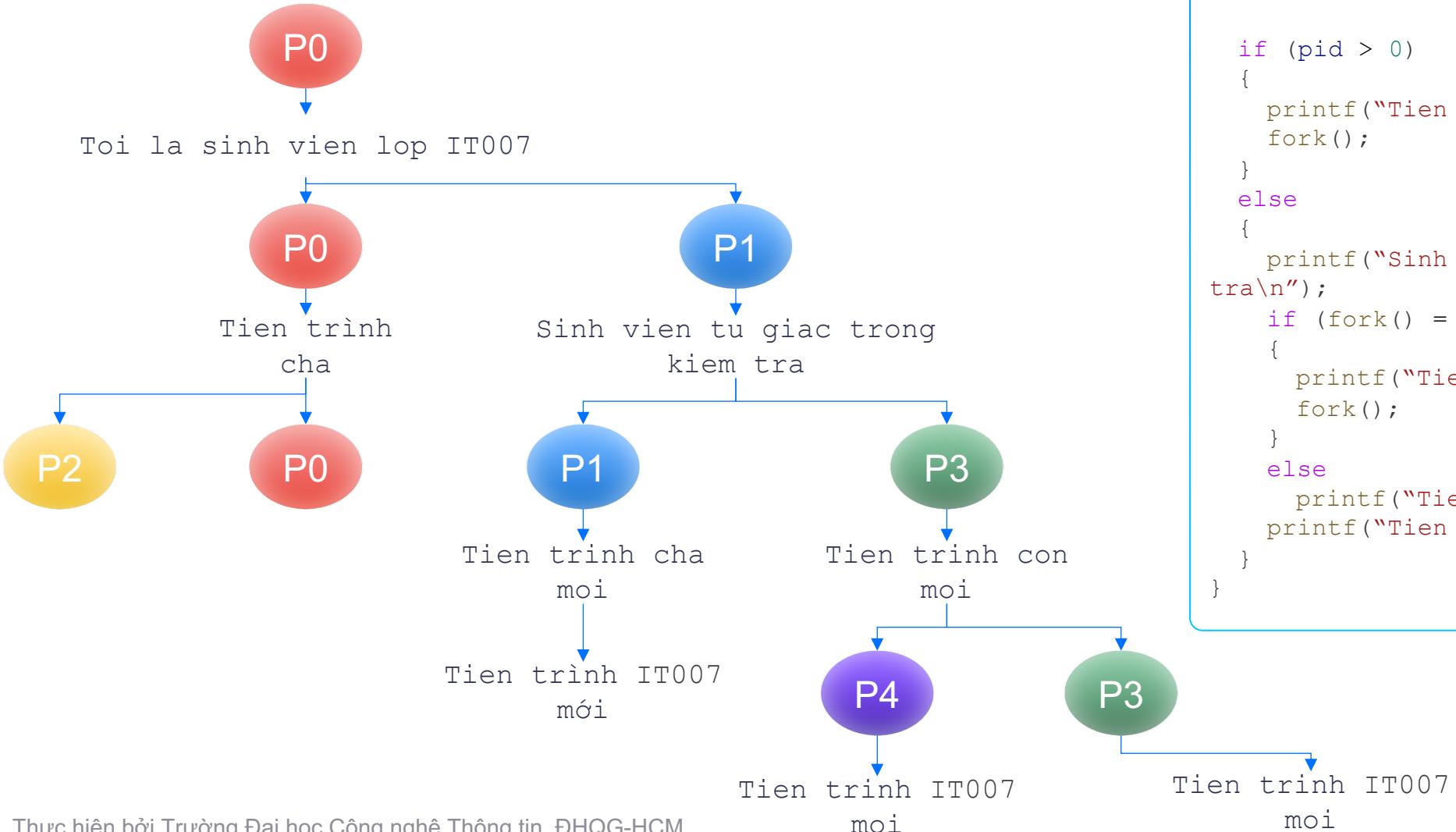
```
else
{
    printf("Sinh vien tu giac trong kiem tra
           \n");

    if (fork() == 0)
    {
        printf("Tien trinh con moi \n");
        fork();
    }
    else
        printf("Tien trinh cha moi \n");
    printf("Tien trinh IT007 moi \n");
}
```



5.1. Tạo tiến trình

Ví dụ tạo process với fork() (tt)



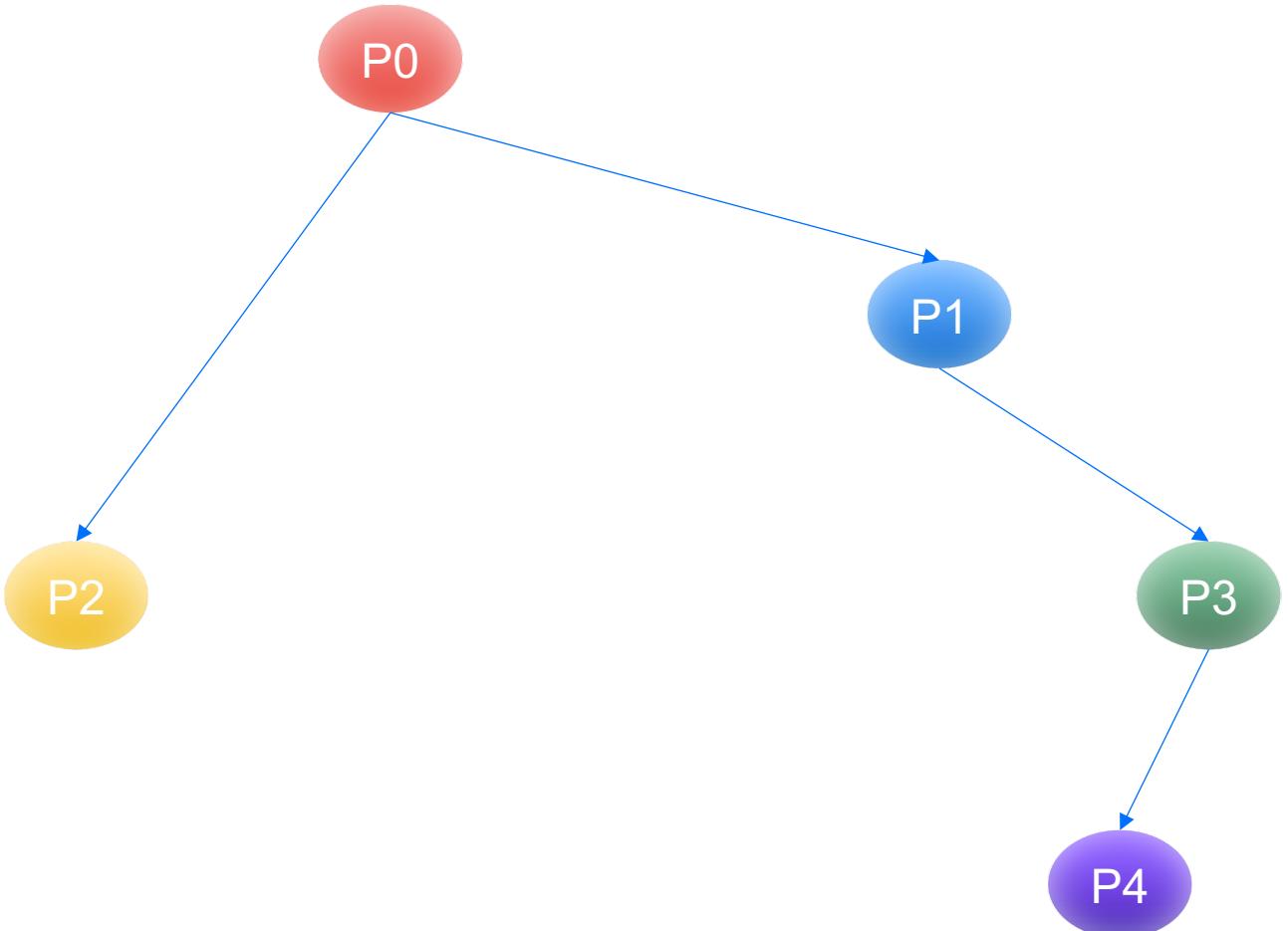
```
int main(int argc, char **argv)
{
    int pid;
    printf("Toi la sinh vien lop IT007\n");
    pid = fork();

    if (pid > 0)
    {
        printf("Tien trinh cha\n");
        fork();
    }
    else
    {
        printf("Sinh vien tu giac trong kiem tra\n");
        if (fork() == 0)
        {
            printf("Tien trinh con moi\n");
            fork();
        }
        else
            printf("Tien trinh cha moi\n");
        printf("Tien trinh IT007 moi\n");
    }
}
```



5.1. Tạo tiến trình

Ví dụ tạo process với fork() (tt)



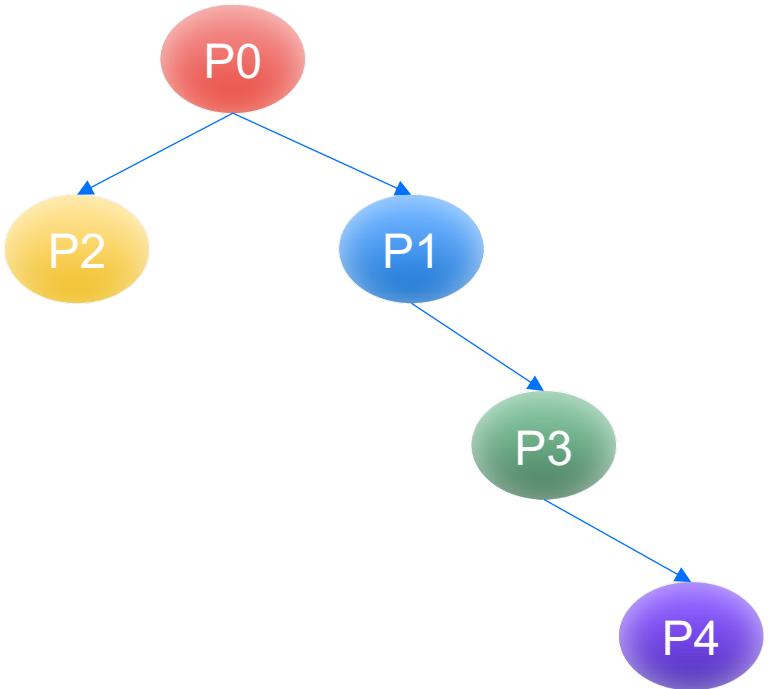
```
int main(int argc, char **argv)
{
    int pid;
    printf("Toi la sinh vien lop IT007\n");
    pid = fork();

    if (pid > 0)
    {
        printf("Tien trinh cha\n");
        fork();
    }
    else
    {
        printf("Sinh vien tu giac trong kiem tra\n");
        if (fork() == 0)
        {
            printf("Tien trinh con moi\n");
            fork();
        }
        else
            printf("Tien trinh cha moi\n");
        printf("Tien trinh IT007 moi\n");
    }
}
```



5.1. Tạo 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();

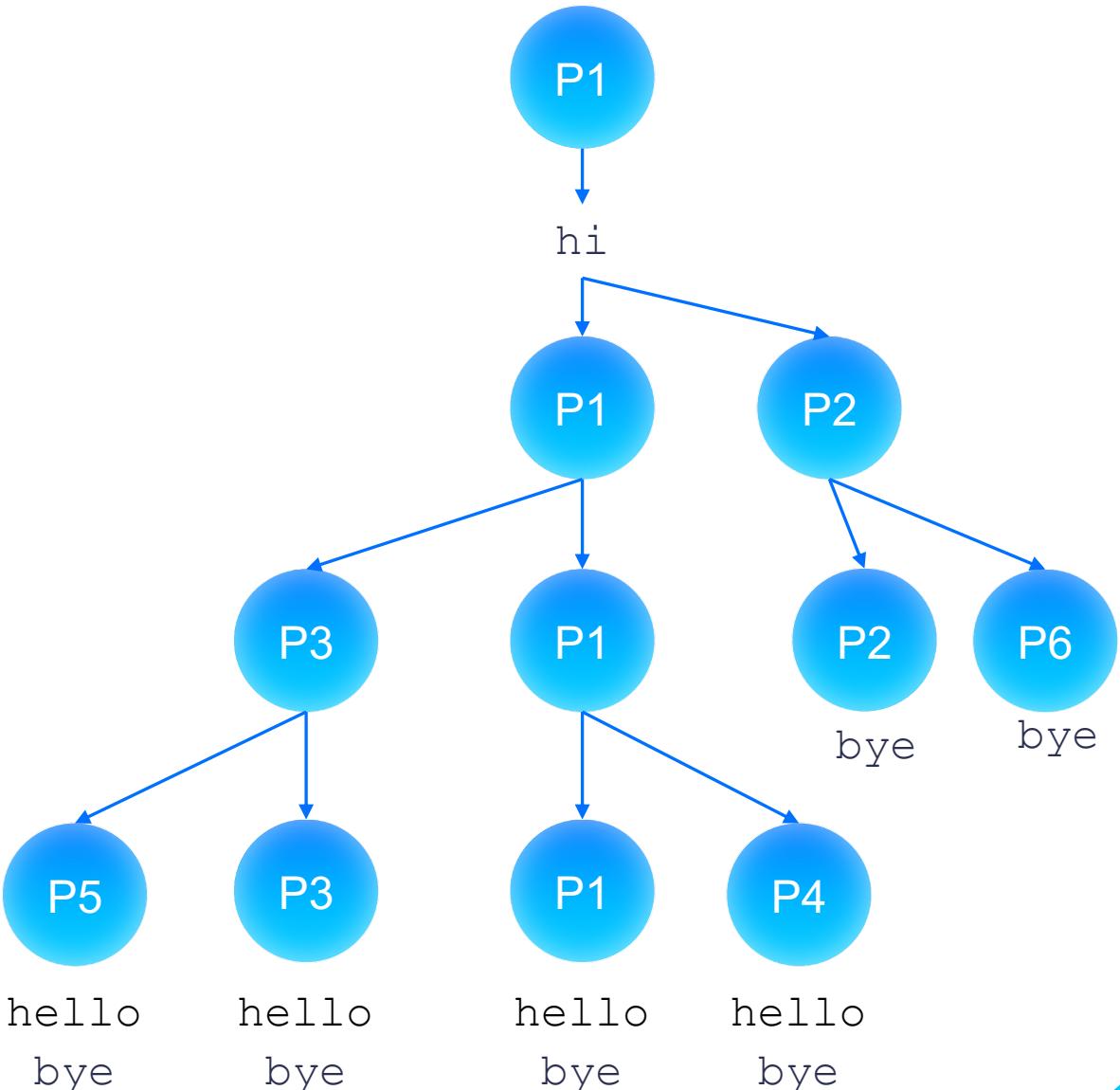
    if (pid > 0)
    {
        printf("Tien trinh cha\n");
        fork();
    }
    else
    {
        printf("Sinh vien tu giac trong kiem tra\n");
        if (fork() == 0)
        {
            printf("Tien trinh con moi\n");
            fork();
        }
        else
            printf("Tien trinh cha moi\n");
        printf("Tien trinh IT007 moi\n");
    }
}
```



5.1. Tạo 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");
}
```





CÁC TÁC VỤ ĐỐI VỚI TIẾN TRÌNH

5.2. Kết thúc tiến trình

5



5.2. 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,...).



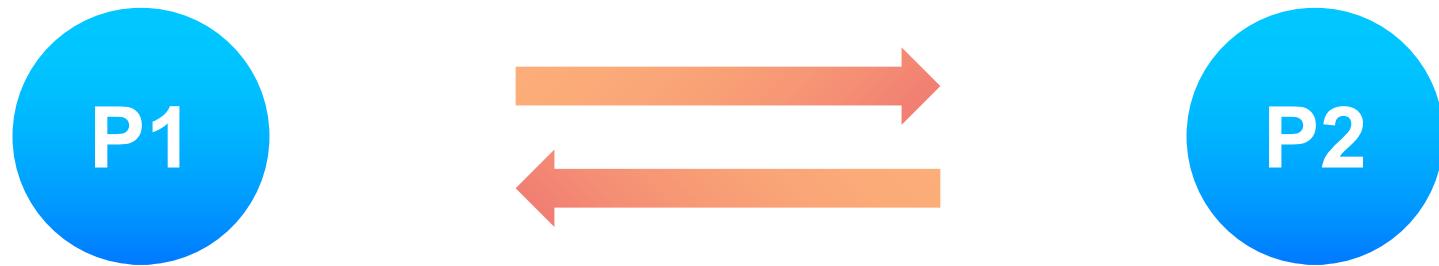
CỘNG TÁC GIỮA CÁC TIẾN TRÌNH

6



6. Cộng tác giữa các tiến trình

- Trong tiến trình thực thi, các tiến trình có thể cộng tác (cooperate) để hoàn thành công việc.
- 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.





6. Cộng tác giữa các tiến trình

- 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



CỘNG TÁC GIỮA CÁC TIẾN TRÌNH

6.1. Giao tiếp liên tiến trình (IPC)

6



6.1. Giao tiếp liên tiến trình (IPC)

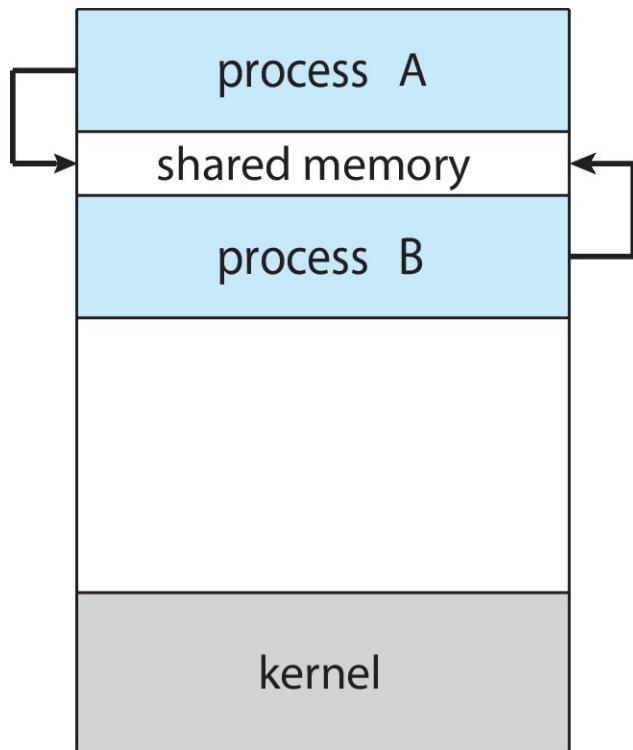
- **IPC (Inter Process Communication)** là cơ chế cung cấp bởi hệ điều hành nhằm giúp các tiến trình:
 - Giao tiếp với nhau
 - Đồng bộ hoạt động
- Hai mô hình IPC:
 - Shared memory
 - Message passing



6.1. Giao tiếp liên tiến trình (IPC)

Shared memory

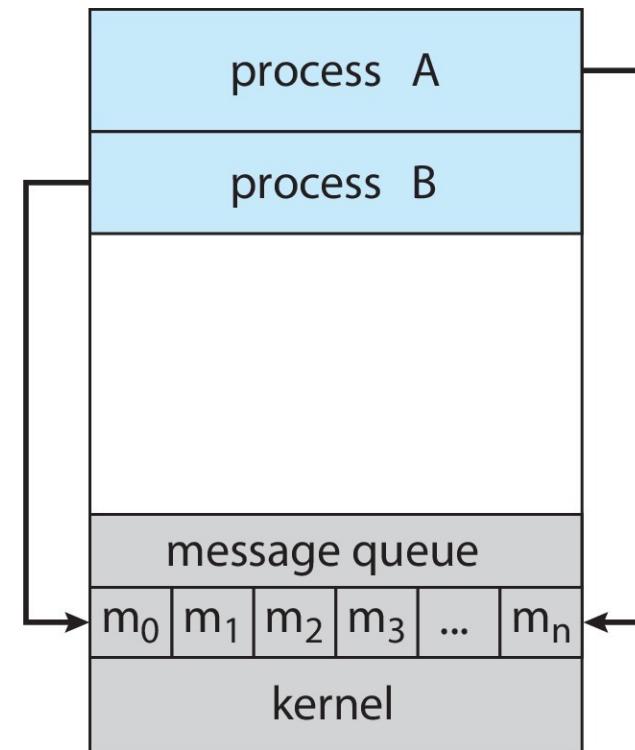
Bộ nhớ được chia sẻ



(a)

Message passing

Truyền thông điệp



(b)



CỘNG TÁC GIỮA CÁC TIẾN TRÌNH

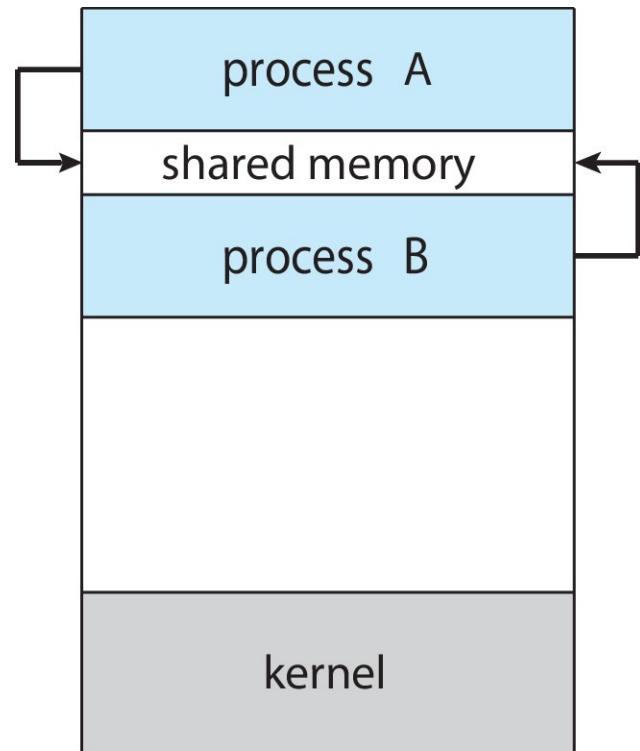
6.2. Bộ nhớ được chia sẻ - Shared memory

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

- Đặt tên (Naming)

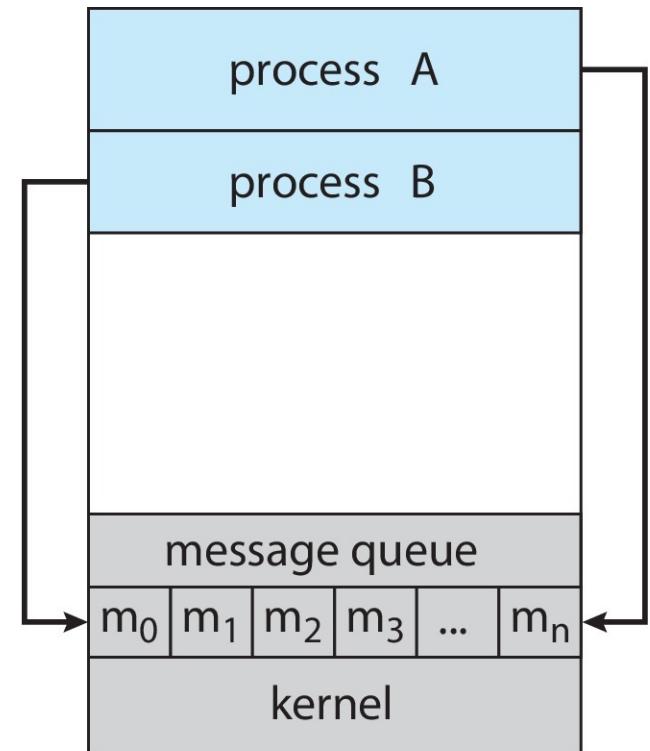
- Giao tiếp trực tiếp

- 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

Các tiến trình giao tiếp nhau thông qua 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

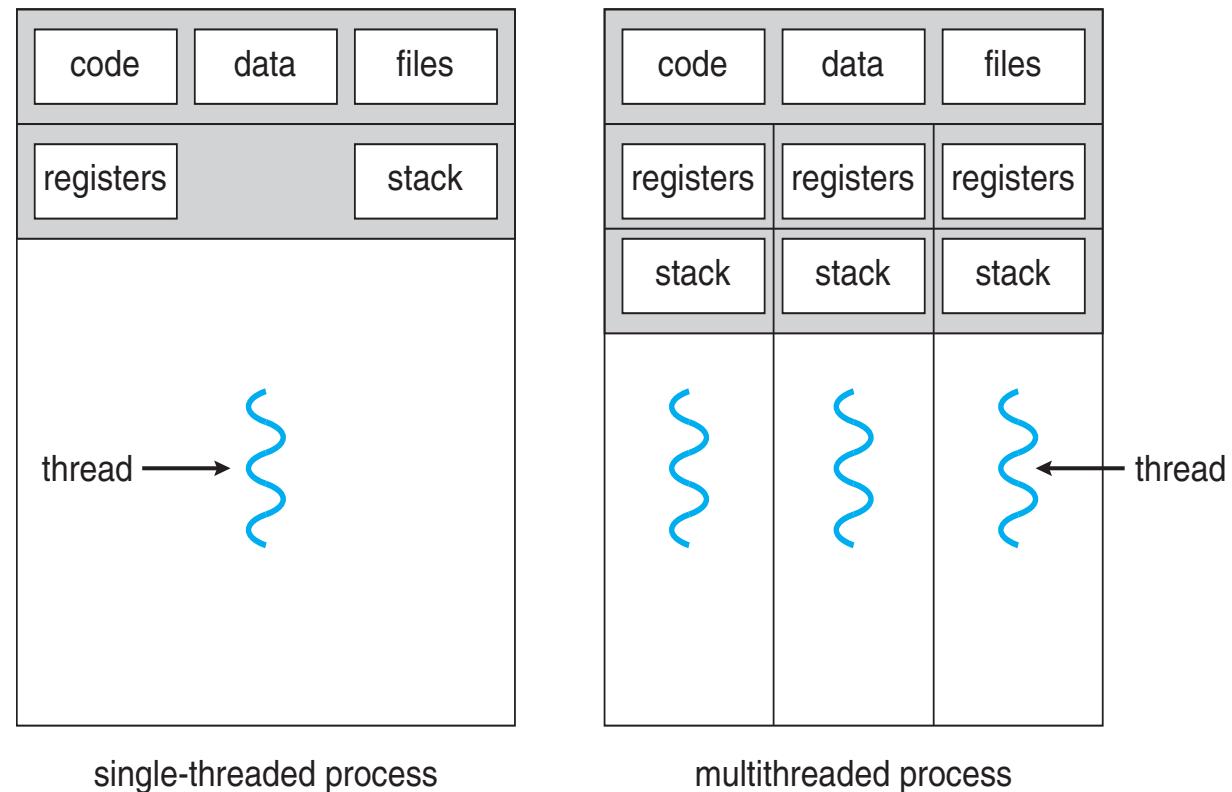
7.1. Tổng quan về tiểu trình

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 (files)





7.1. Tổng quan về tiểu trình

PCB và TCB trong mô hình multithreads

Process Control Block (PCB)

PID
Threads list
Context (Mem, global ressources...)
Relatives (Dad, children)
Scheduling statistic

Thread Control Block (TCB)

TID
State (State, details)
Context (IP, local stack...)



7.1. Tổng quan về tiểu trình

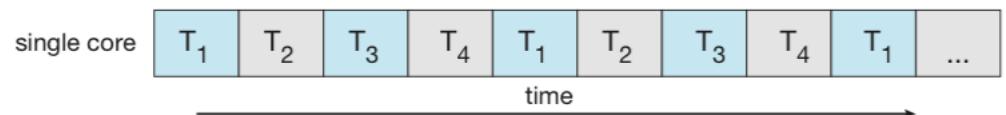
Lợi ích của tiến trình đa luồng

- **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:** dễ dàng và 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.
- **Khả năng mở rộng:** có khả năng thực thi song song trên nhiều lõi xử lý

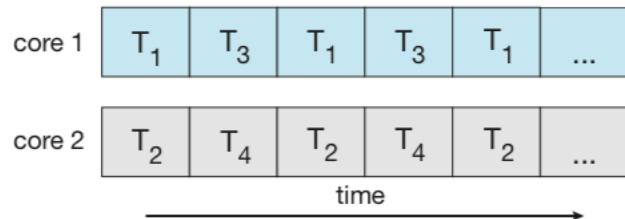


7.1. Tổng quan về tiêu trình

Thực thi *đồng thời*



Thực thi *song song*



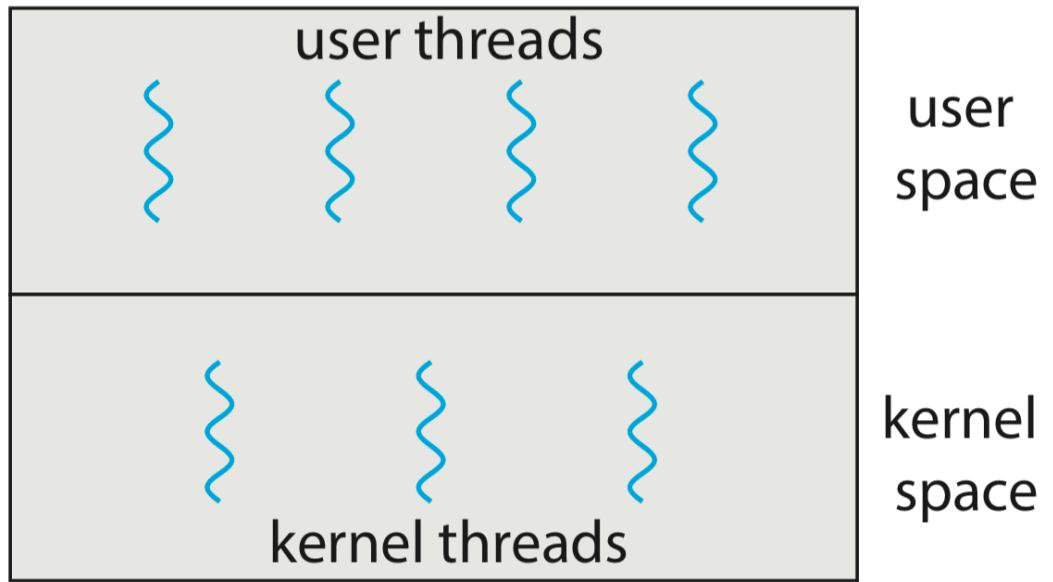
- Nhiều công việc được phép thực thi *đồng thời*.
- Trên hệ thống đơn bộ xử lý, bộ định thời được thiết kế để *tạo ra cảm giác* các công việc được thực *song song* **nhưng thực tế chỉ có 1 công việc được thực thi tại 1 thời điểm**.

- Thực thi nhiều công việc *song song* với nhau bằng cách sử dụng nhiều lõi xử lý.



7.1. Tổng quan về tiêu trình

Phân loại tiêu trình



Tiêu trình người dùng

- Thực thi các đoạn mã trong chương trình người dùng
- Được quản lý mà không cần sự hỗ trợ từ hạt nhân

Tiêu trình hạt nhân

- Thực thi các thao tác hệ thống (qua system call)
- Được hỗ trợ và quản lý trực tiếp bởi hệ điều hành



TIỂU TRÌNH

7.2. Các mô hình đa tiêu trình

7



7.2. Các mô hình đa tiêu trình

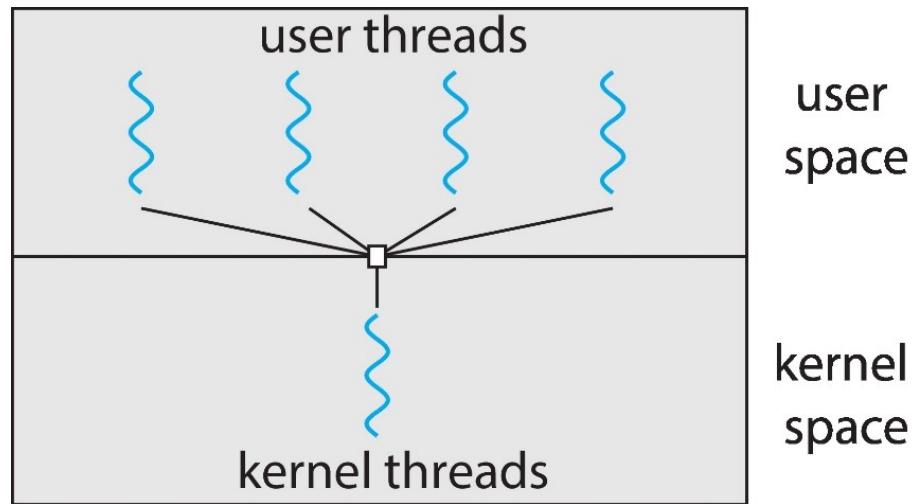
- Nhiều – Một (Many-to-One)
- Một – Một (One-to-One)
- Nhiều – Nhiều (Many-to-Many)



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.

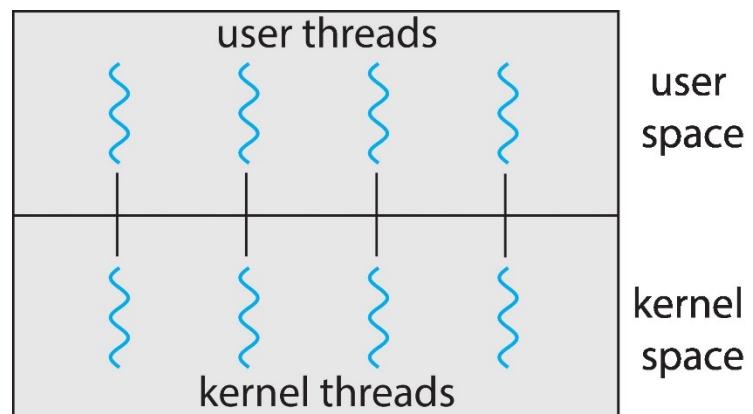




7.2. Các mô hình đa tiêu trình

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

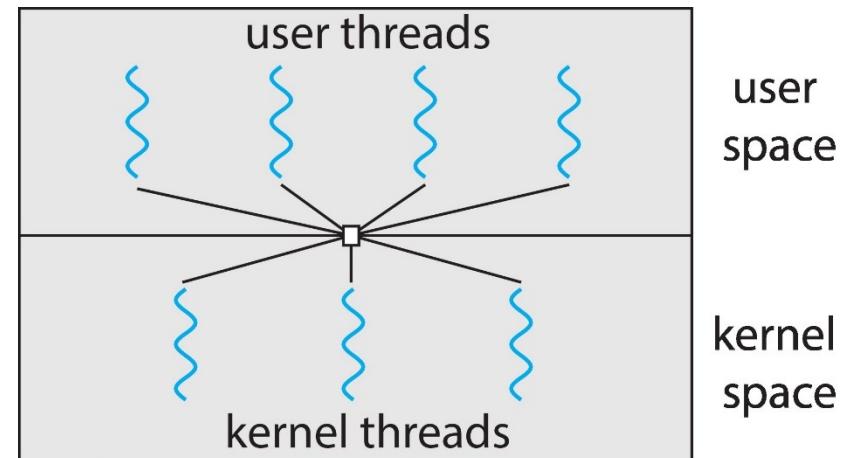




7.2. Các mô hình đa tiêu trình

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 bài học

- 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



Thực hiện bởi Trường Đại học Công nghệ Thông tin, ĐHQG-HCM