



COMPUTER ENGINEERING



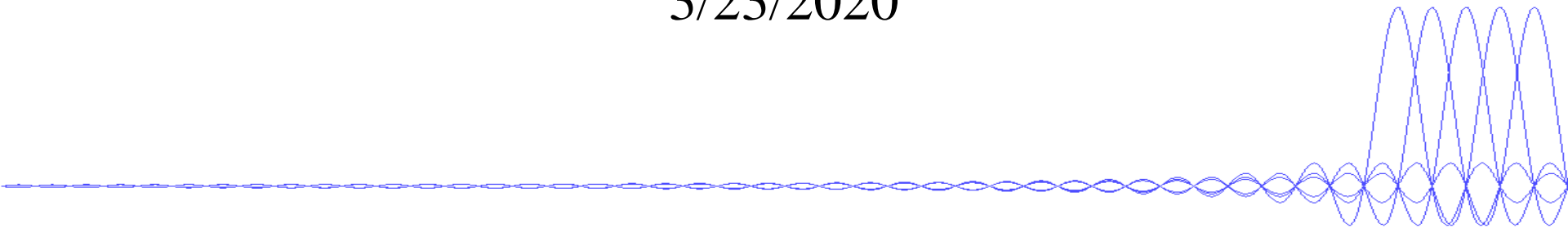
**UIT**  
TRƯỜNG ĐẠI HỌC  
CÔNG NGHỆ THÔNG TIN

# HỆ ĐIỀU HÀNH

## Chương 3

### Tiến trình

3/23/2020





## Câu hỏi ôn tập chương 2

- Nêu các thành phần chính của hệ điều hành?
- Nêu các dịch vụ mà hệ điều hành cung cấp?
- Lời gọi hệ thống là gì? Nêu 1 vài ví dụ?
- Có mấy dạng cấu trúc hệ điều hành? Kể tên?
- Máy ảo dùng để làm gì? Có mấy loại? Cho ví dụ
- Tiến trình là gì? Các nhiệm vụ chính của thành phần quản lý tiến trình?



## Mục tiêu chương 3

- Hiểu được khái niệm và các trạng thái của tiến trình
- Biết được các thông số của tiến trình
- Biết được các khái niệm về định thời tiến trình
- Biết được các tác vụ cơ bản của một tiến trình
- Hiểu được cách giao tiếp giữa các tiến trình



# Nội dung chương 3

- Khái niệm cơ bản
- Trạng thái tiến trình
- Khởi 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



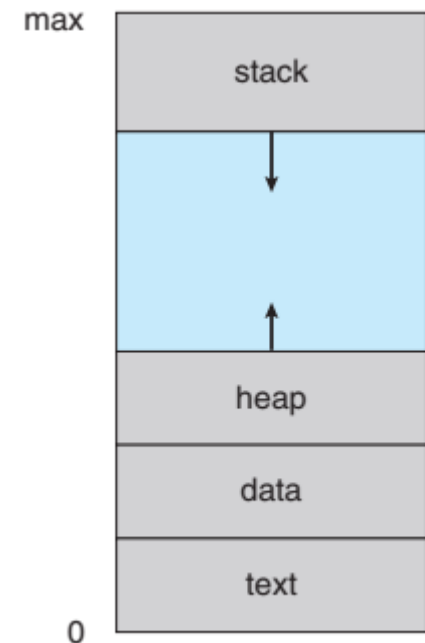
# Khái niệm cơ bản

- Các hoạt động của CPU được gọi là gì?
  - Hệ thống bó (Batch system): jobs
  - Time-shared systems: user program, task
  - Các hoạt động là tương tự → gọi là process
- 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.



# Khái niệm cơ bản (tt)

- 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

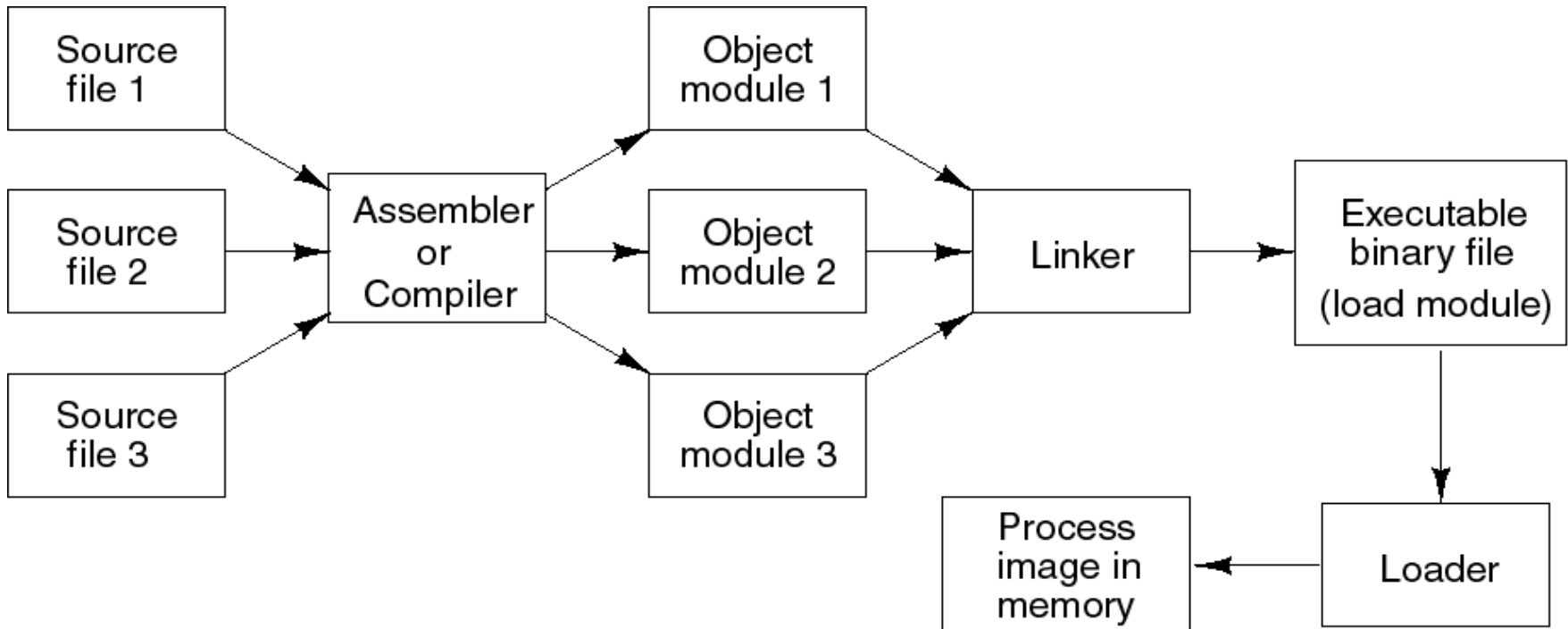


Layout của tiến trình trong bộ nhớ



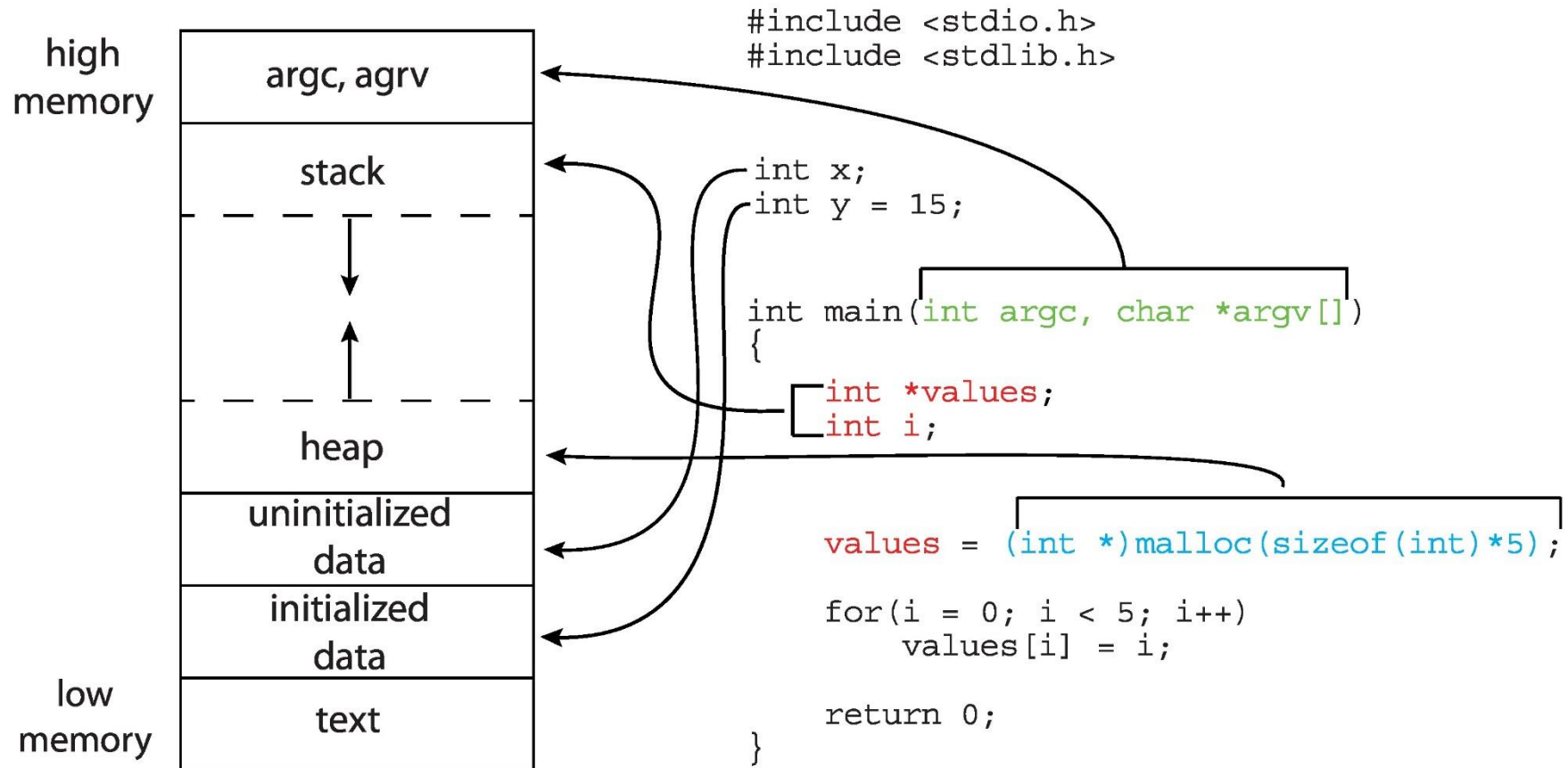
# Khái niệm cơ bản (tt)

## ■ Các bước nạp chương trình vào bộ nhớ:





# Layout bộ nhớ của một chương trình C







# Khái niệm cơ bản (tt)

## ■ 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, ...)

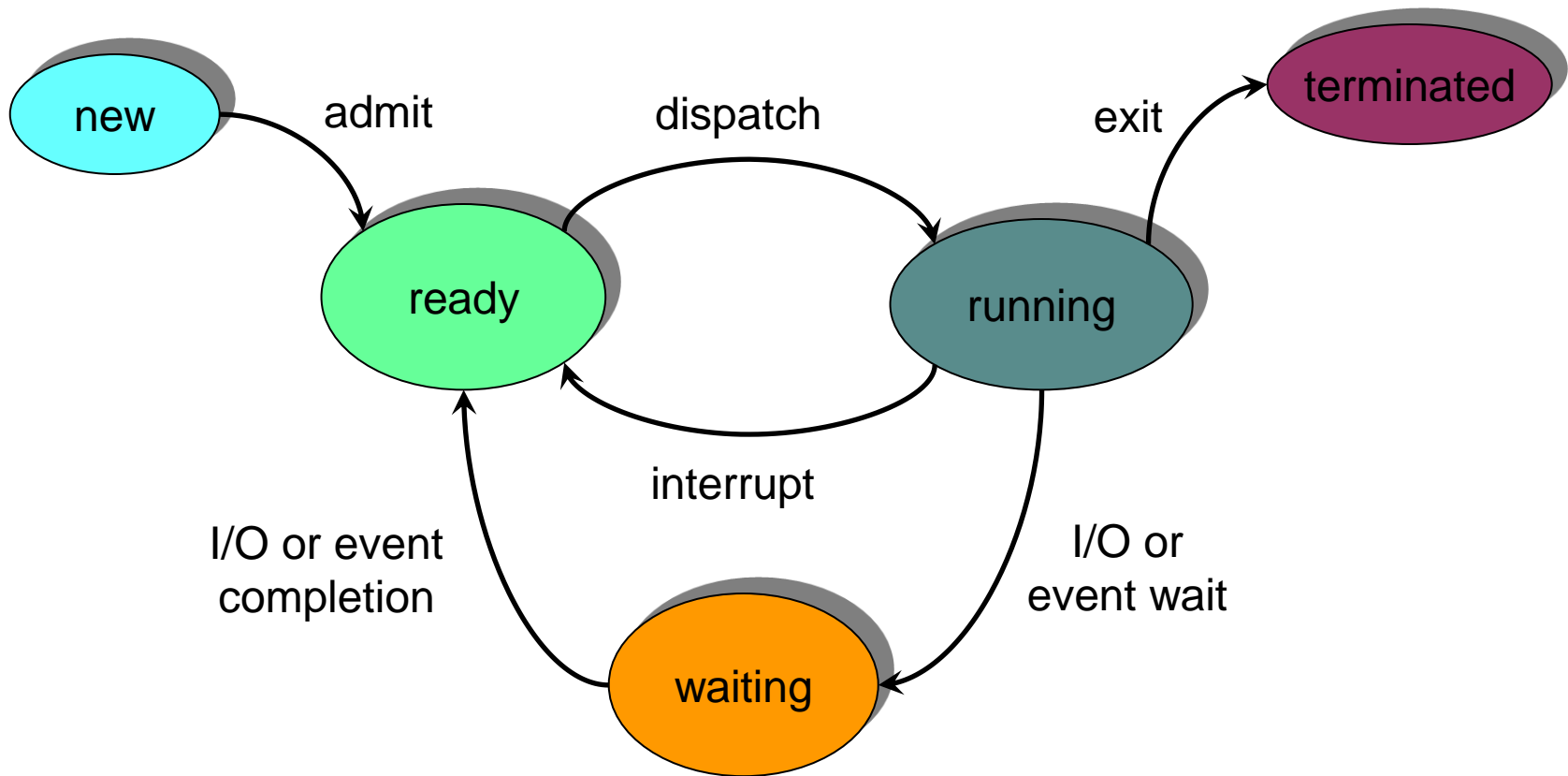


# Trạng thái tiến trình

- 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 là blocked, tiến trình đợi I/O hoàn tất, tín hiệu
- terminated: tiến trình đã kết thúc



# Trạng thái tiến trình (tt)



Chuyển đổi giữa các trạng thái của tiến trình



# Trạng thái tiến trình (tt)

```
/* 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



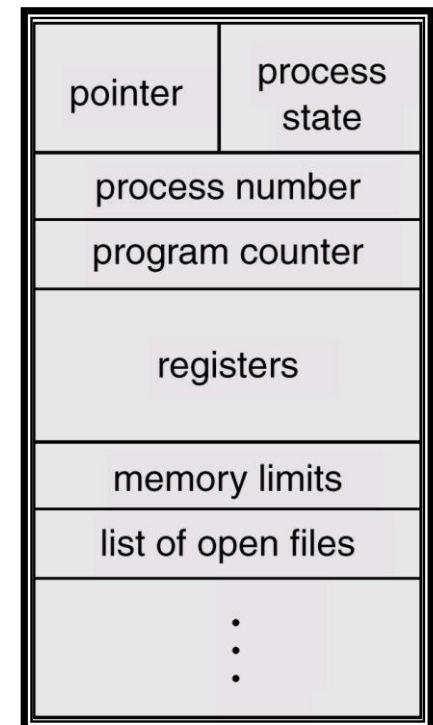
# Process Control Block

- 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





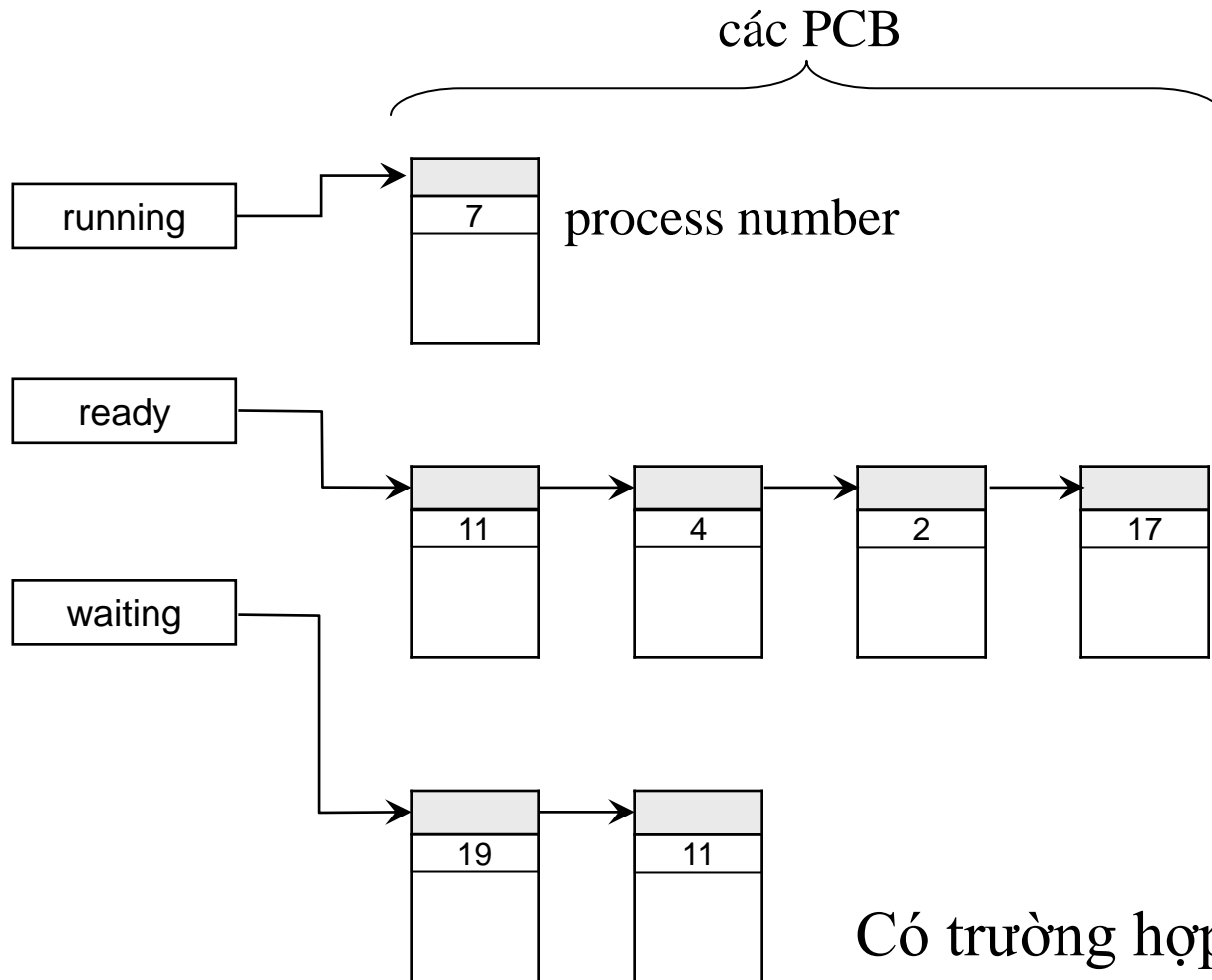
# Yêu cầu đối với hệ điều hành về quản lý tiến trình

- 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



# Quản lý các tiến trình: các hàng đợi

Ví dụ



Có trường hợp sai không?



# 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

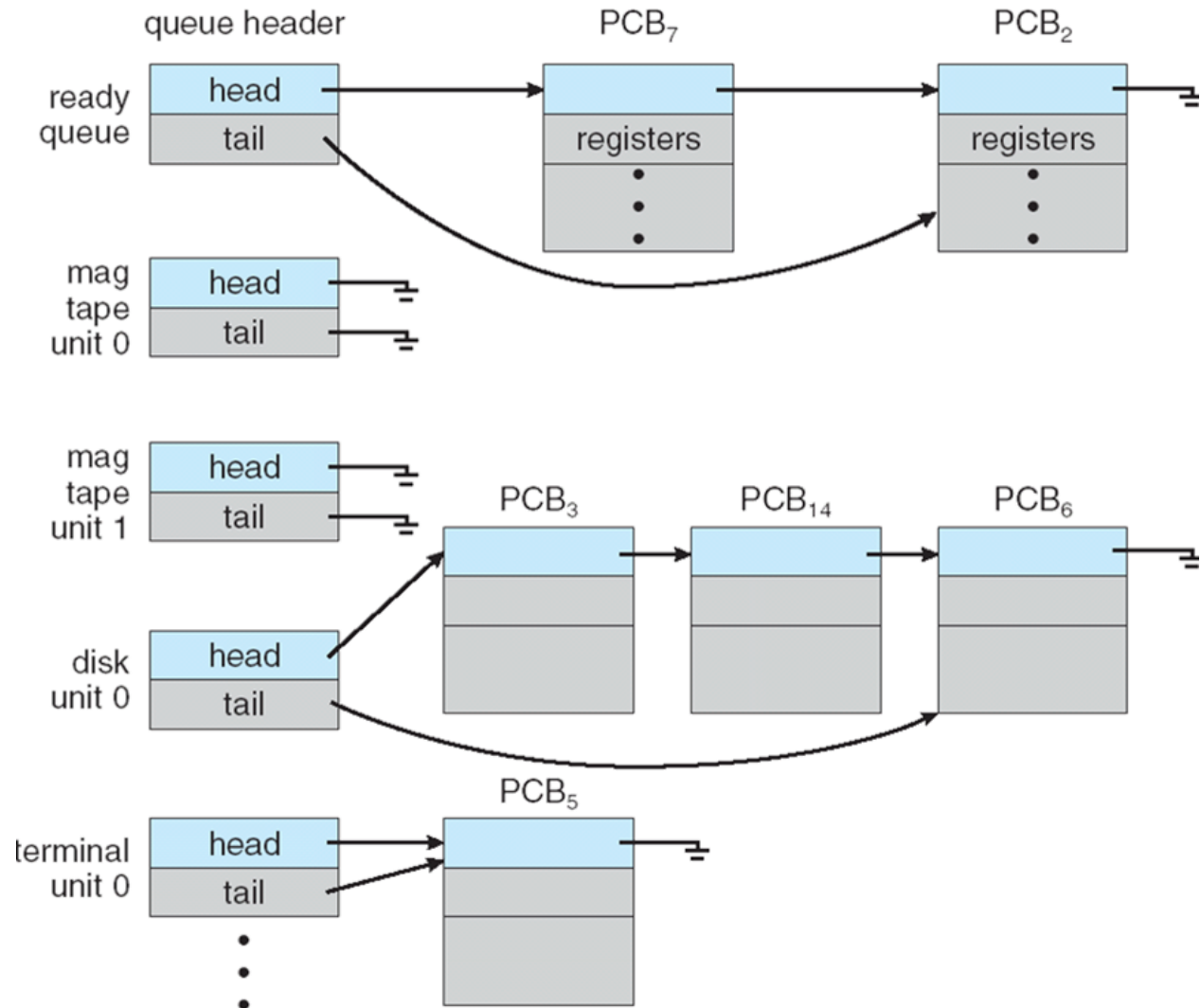
- User tương tác với mỗi chương trình đang thực thi
- Mục tiêu: tối thiểu thời gian đáp ứng





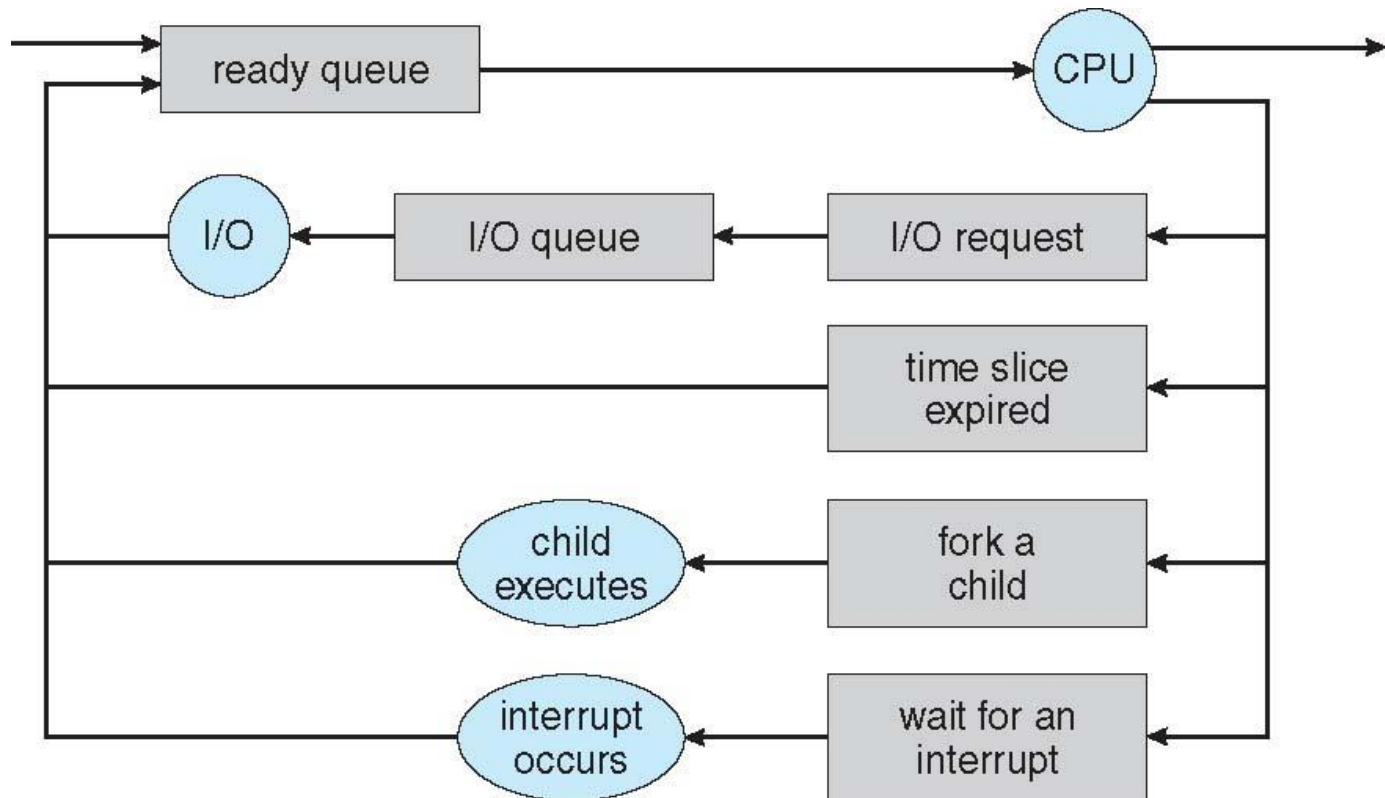
# Các hàng đợi định thời

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





# Các hàng đợi định thời (tt)

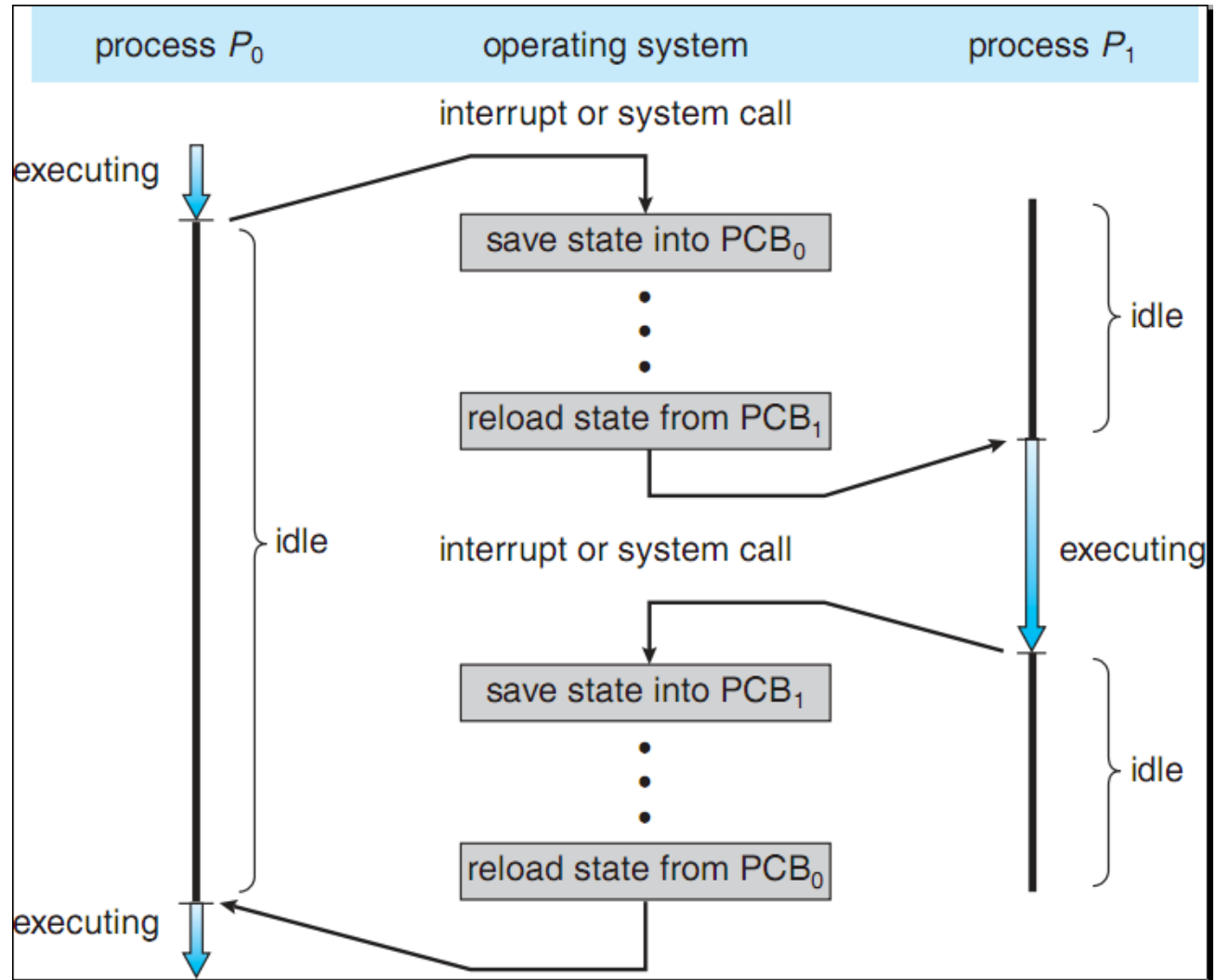


Lưu đồ hàng đợi của định thời tiến trình



# Chuyển ngữ cảnh (context switch)

Chuyển ngữ cảnh: CPU chuyển từ tiến trình này đến tiến trình khác





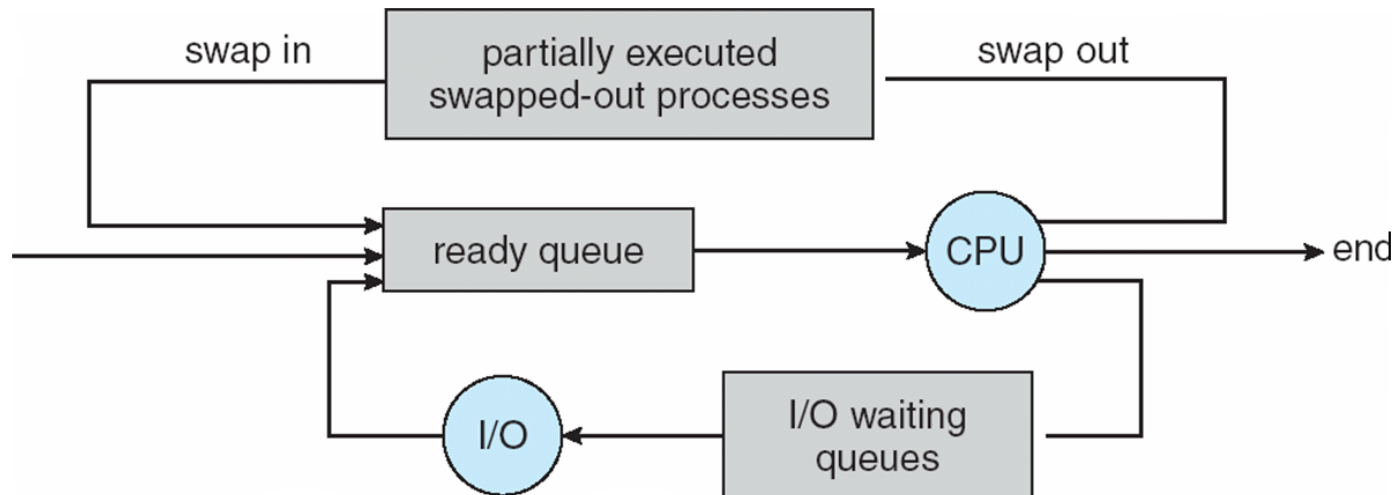
# Bộ định thời

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



# 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)





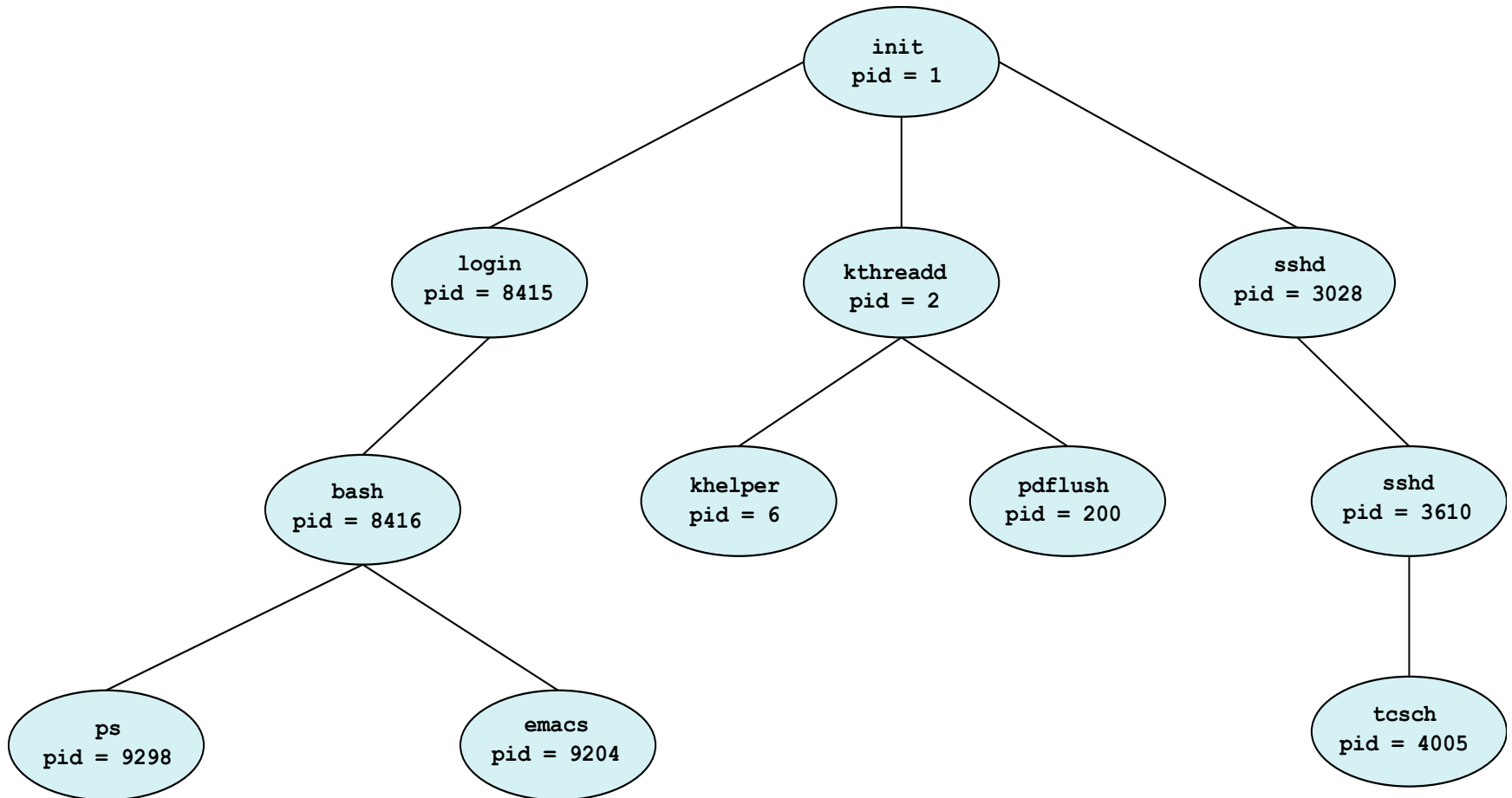
# Các tác vụ đối với tiến trình

## ■ 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 (tiền trình cha)
  - Quan hệ cha-con định nghĩa một cây tiến trình



# Cây tiến trình trong Linux/Unix





# Các tác vụ đối với tiến trình (tt)

## ■ 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





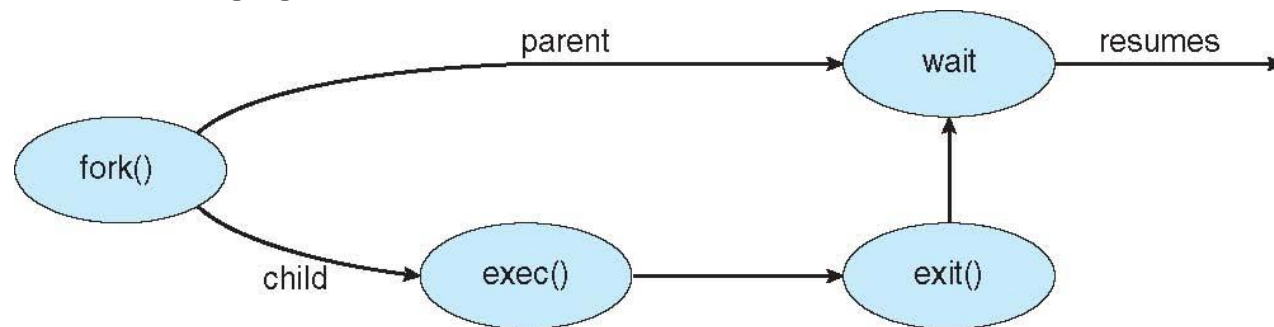
# Về quan hệ cha/con

## ■ 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





# 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);
    }
}
```



# 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");
}
```



# 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 (tt)

## ■ 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

- 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



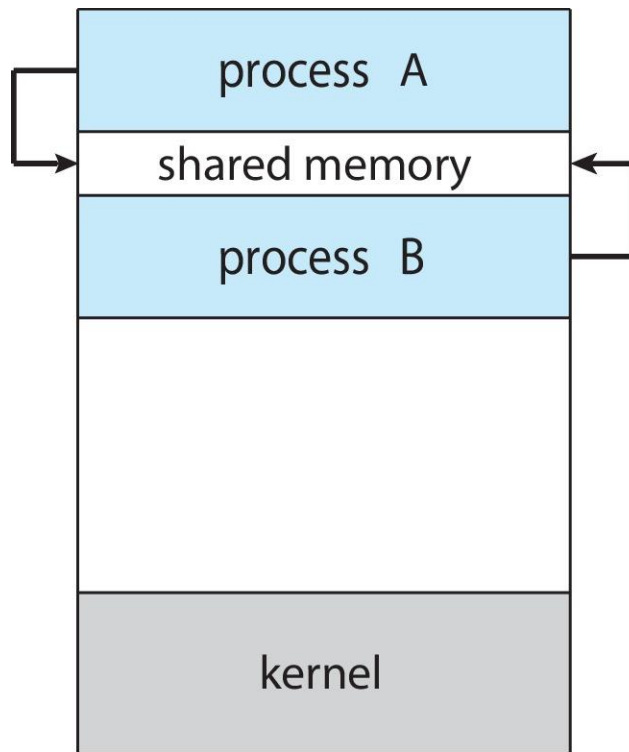
# Giao tiếp liên tiến trình

- IPC 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



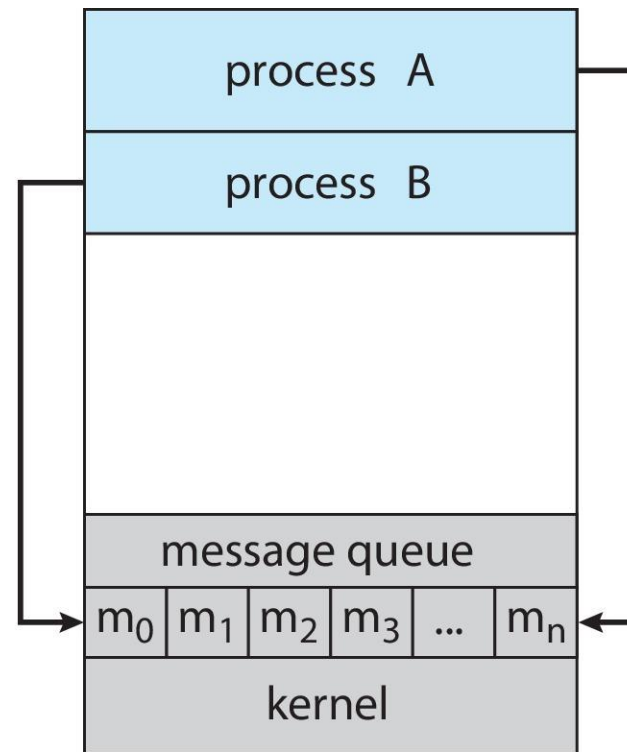
# Giao tiếp liên tiến trình

(a) Shared memory.



(a)

(b) Message passing.



(b)





# 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.



# Hệ thống truyền thông điệp

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



# Hệ thống truyền thông điệp (tt)

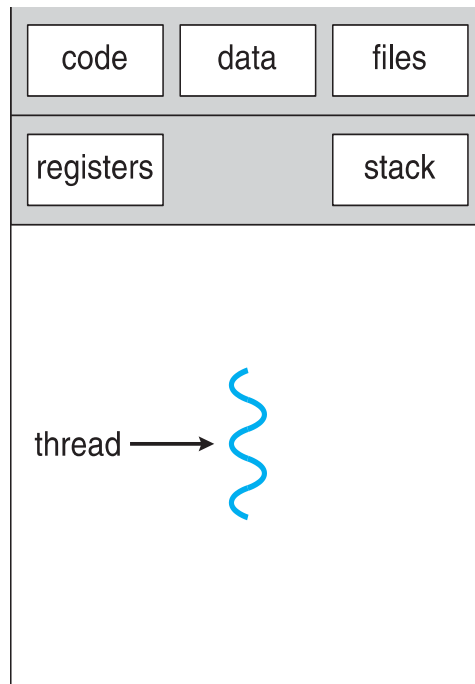
Làm thế nào để các tiến trình giao tiếp nhau?

- 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

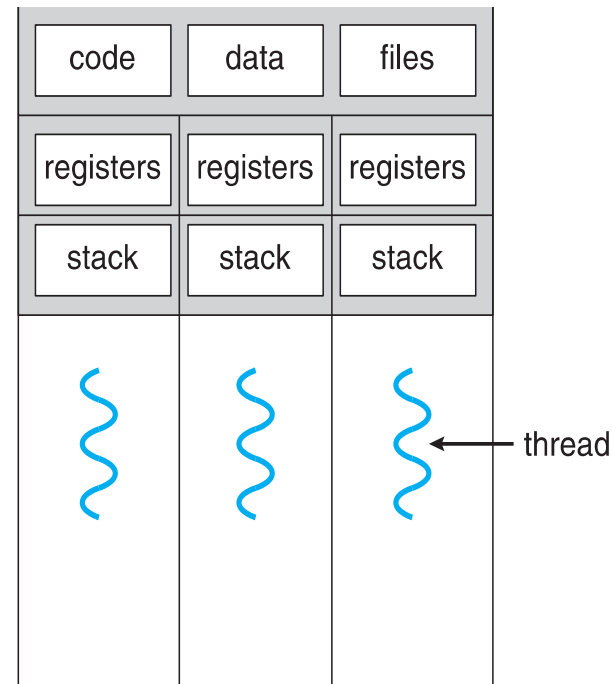


# Tiêu trình

- 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)



single-threaded process



multithreaded process



# PCB và TCB trong mô hình multithreads

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...)

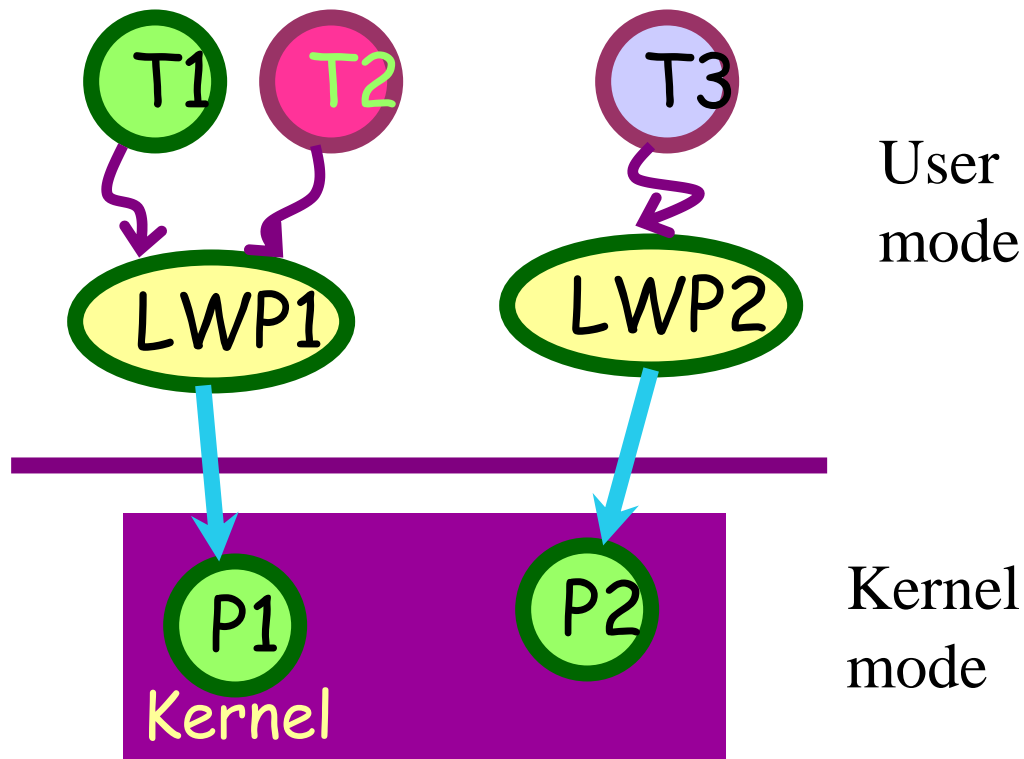


# Lợi ích của tiến trình đa luồng

- 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



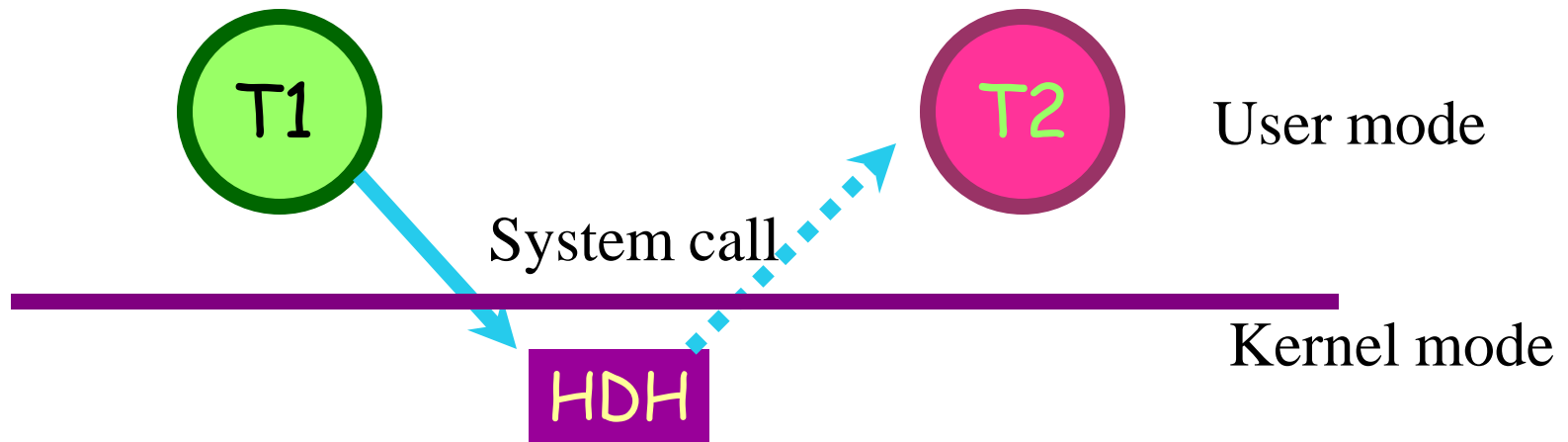
# Tiêu trình người dùng (User thread)



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



# Tiêu trình hạt nhân (Kernel thread)



**Khái niệm tiêu trình được xây dựng bên trong hạt nhân**





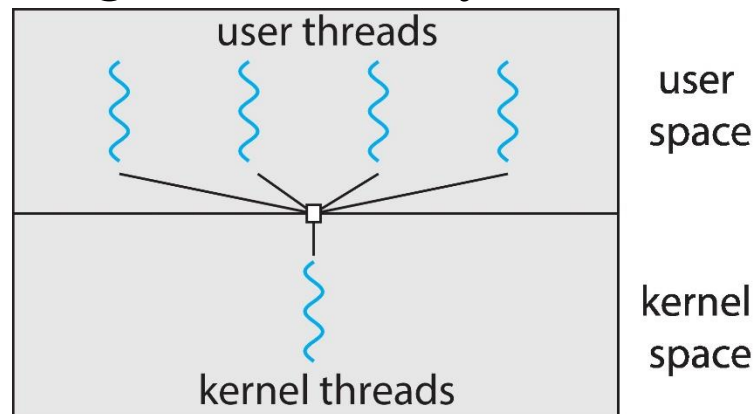
# Các mô hình đa tiểu trình

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



# Mô hình Nhiều – Một (Many-to-One)

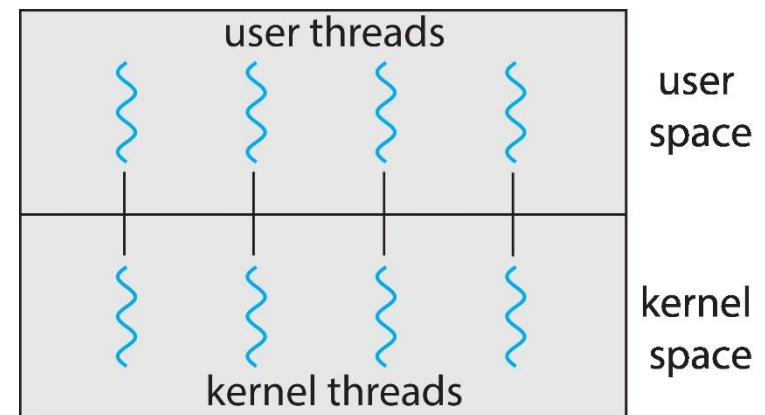
- Nhiều tiểu trình người dùng được ánh xạ đến một tiểu trình hạt nhân.
- Một tiểu trình bị block sẽ dẫn đến tất cả tiểu trình bị block.
- Các tiểu trình không thể chạy song song trên các hệ thống đa lõi bởi vì chỉ có một tiểu trình có thể truy xuất nhân tại một thời điểm.
- Rất ít hệ thống sử dụng mô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





# Mô hình Một – Một (One-to-One)

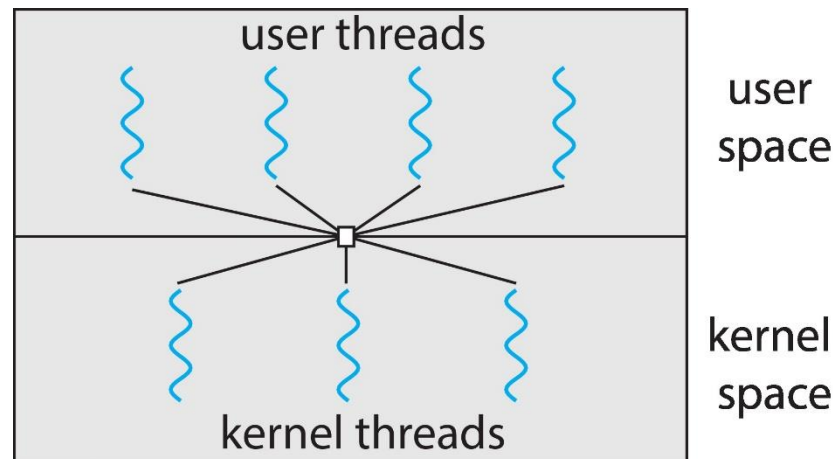
- Mỗi tiểu trình người dùng ứng với một tiểu trình hạt nhân.
- Tạo một tiểu trình người dùng cũng đồng thời tạo một tiểu trình hạt nhân.
- Tính đồng thời (concurrency) tốt hơn mô hình nhiều – một vì các tiểu trình khác vẫn hoạt động bình thường khi một tiểu trình bị block.
- Nhược điểm: Số lượng tiểu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





# Mô hình Nhiều – Nhiều (Many-to-Many)

- Các tiểu trình người dùng được ánh xạ với nhiều tiểu trình hạt nhân.
- Cho phép hệ điều hành tạo đủ số lượng tiểu 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
- Khởi 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

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");
}
```



## Câu hỏi ôn tập chương 3 (tt)

- 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?





COMPUTER ENGINEERING



**UIT**  
TRƯỜNG ĐẠI HỌC  
CÔNG NGHỆ THÔNG TIN

# THẢO LUẬN

