Chương 3
QUẢN LÝ TIẾN TRÌNH

Tiến trình

- Khái niệm về tiến trình
- Lập lịch tiến trình
- Các thao tác trên tiến trình
- Hợp tác giữa các tiến trình
- Luồng
- Truyền thông giữa các tiến trình

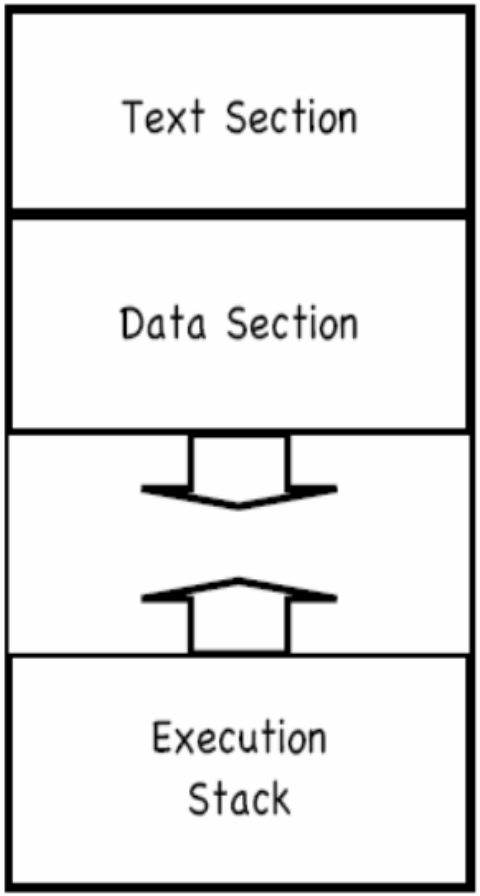
Khái niệm về tiến trình

- Một Hệ điều hành thực hiện nhiều chương trình
  - Hệ thống xử lý theo lô: công việc (job)
  - Hệ thống chia sẻ thời gian: tác vụ (task)
- Ở đây chúng ta dùng tiến trình và công việc với cùng ý nghĩa

Tiến trình (Process)

- Chương trình đang được thực hiện, bao gồm
  - Phần văn bản
  - Ngăn xếp
  - Phần dữ liệu
  - Giá trị bộ đếm chương trình, thanh ghi
- CPU xử lý tiến trình tuần tự
- Là thực thể hoạt động
  - đối lập với chương trình

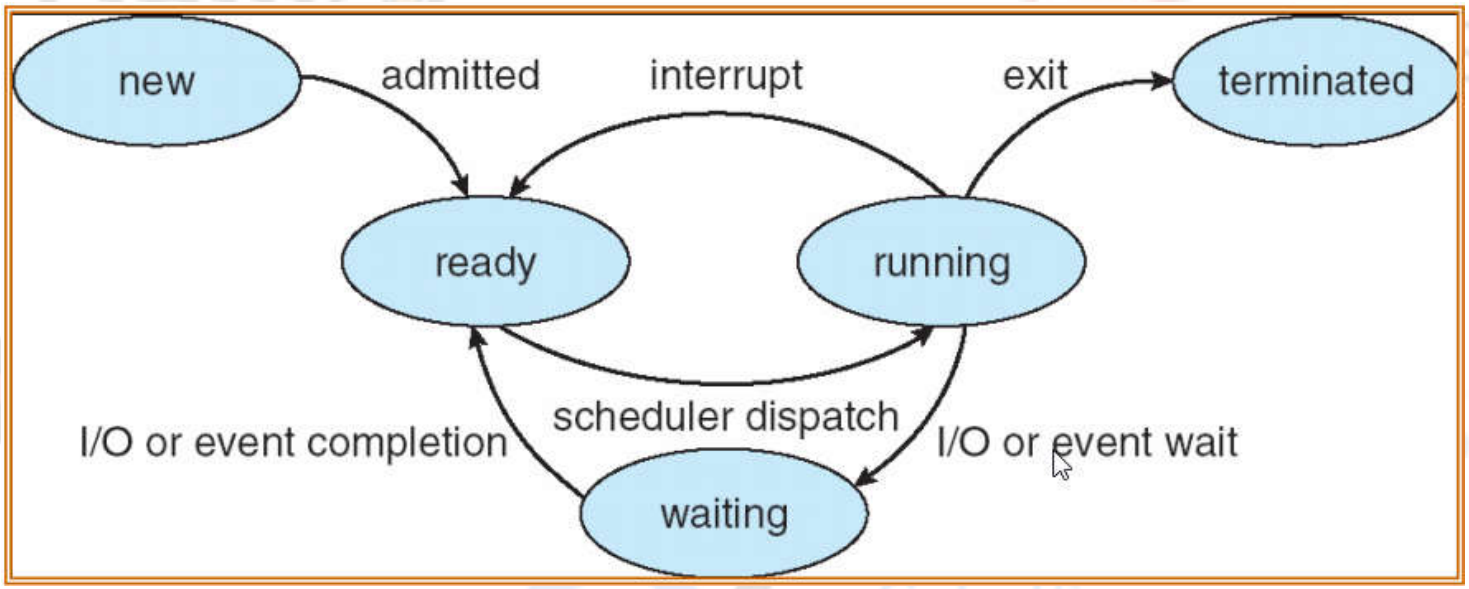
Cấu trúc bộ nhớ tiến trình



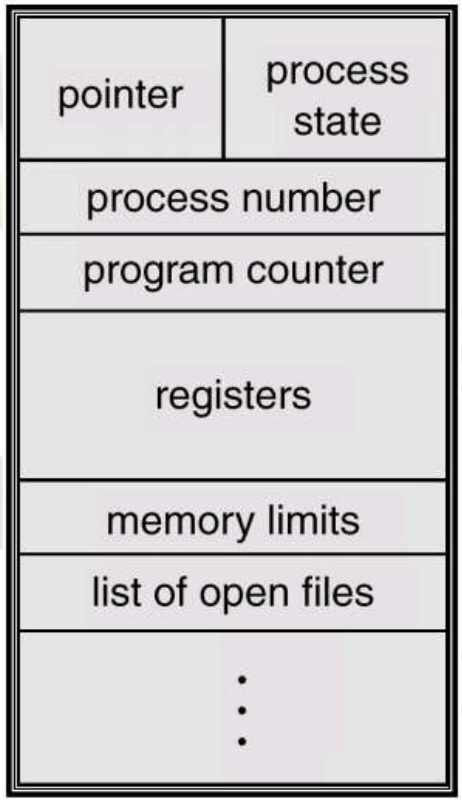
Trạng thái tiến trình

- Tiến trình thay đổi trạng thái trong khi thựchiện
  - New
  - Ready
  - Running
  - Waiting
  - Terminated
- Tại một thời điểm chỉ có một tiến trình ở trạng thái running

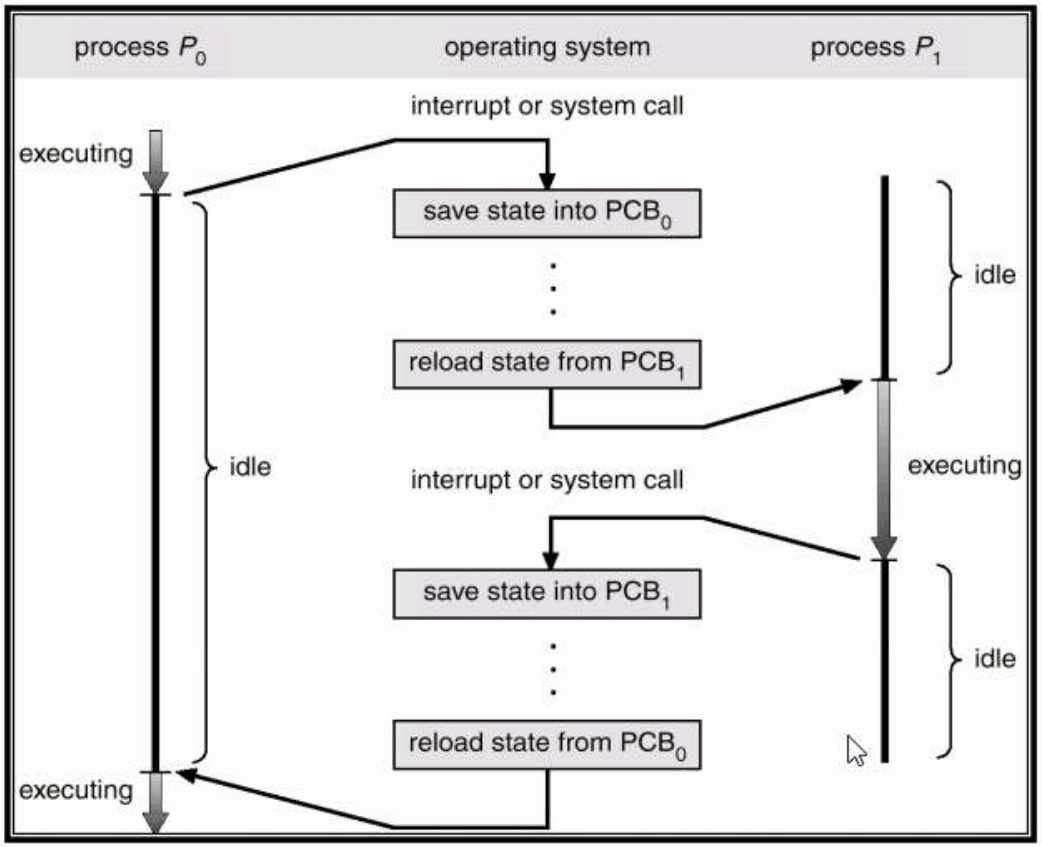
Trạng thái tiến trình



Khối điều khiển tiến trình (PCB)



Chuyển đổi CPU giữa các tiến trình



Lập lịch tiến trình

- Mục đích của đa chương trình
  - Tăng tính tận dụng CPU
- Mục đích của phân chia thời gian
  - Người dùng có thể tương tác với tiến trình trong lúc nó đang thực thi

➢Xử lý nhiều tiến trình

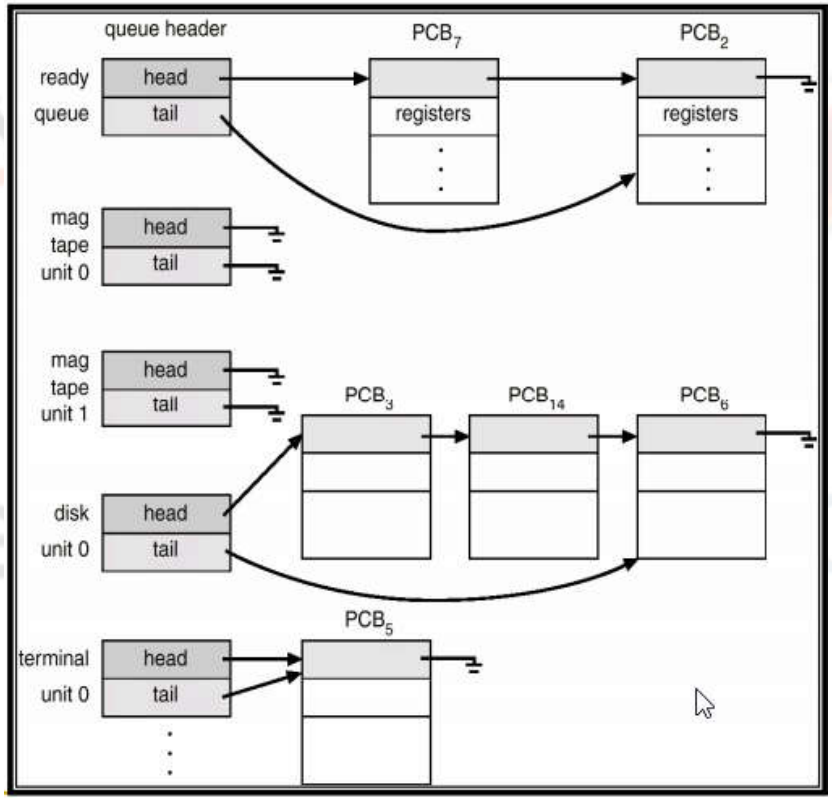
➢Lập lịch tiến trình

Các hàng đợi lập lịch tiến trình

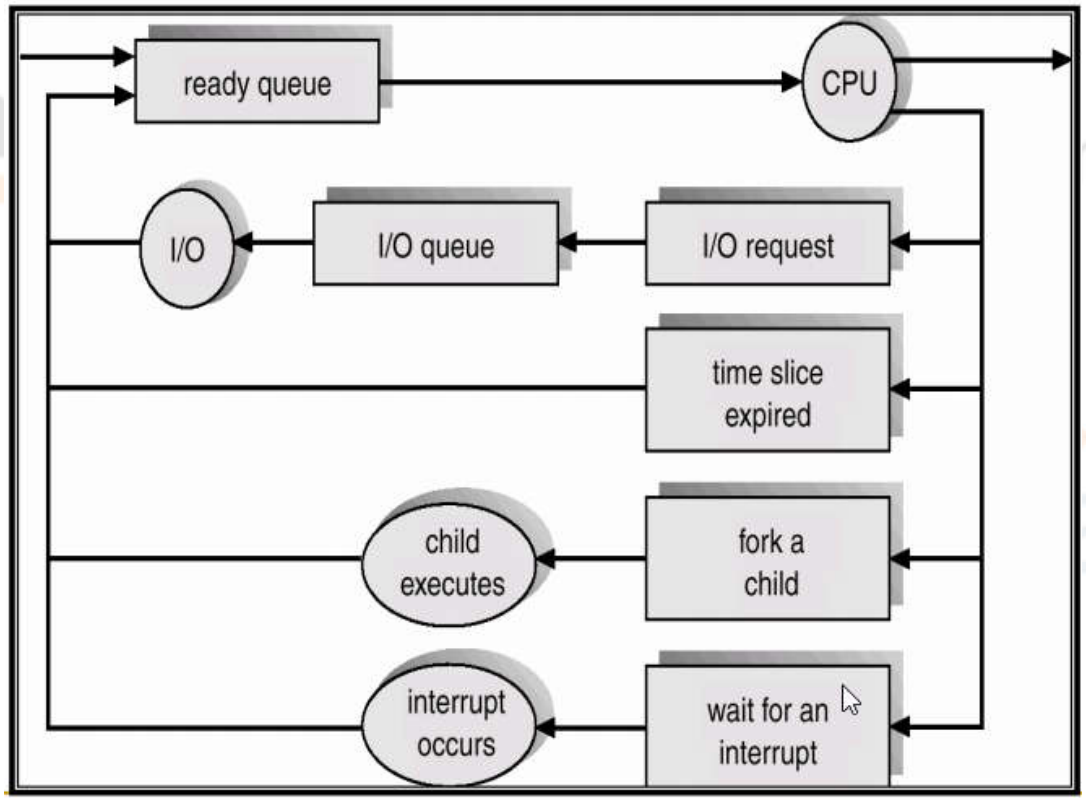
- Hàng đợi công việc
  - Một tập các tiến trình trong hệ thống
- Hàng đợi sẵn sàng
  - Tập các tiến trình trong bộ nhớ trong, sẵn sàng và chỉ chờ thực hiện
- Hàng đợi thiết bị
  - Tập các tiến trình chờ một thiết bị vào /ra

Các tiến trình di trú từ hàng đợi này đến hàng đợi khác

Hàng đợi sẵn sàng và các hàng đợi thiết bị khác nhau



Biểu diễn việc lập lịch tiến trình-Biểu đồ hàng đợi



Vòng đời của một tiến trình

- Khởi tạo: hàng đợi sẵn sàng
- Các sự kiện có thể xảy ra khi tiến trình đã được gán CPU
  - Sinh ra một yêu cầu I/O, đi vào hàng đợi I/O
  - Tạo ra một tiến trình con và đợi cho nó kết thúc
  - Bị tước quyền sử dụng CPU
- Tiếp tục vòng lặp đến khi kết thúc
  - Bị xóa khỏi tất cả các hàng đợi
  - PCB và tất cả các tài nguyên bị thu hồi

Các bộ lập lịch

- Tiến trình lưu trú trong nhiều loại hàng đợi
  - Các bộ lập lịch chọn các tiến trình từ các hàng đợi
- Bộ lập lịch dài hạn
  - Lập lịch công việc – job scheduler
  - Chọn các tiến trình trong tập tiến trình và tải nó vào bộ nhớ để thực hiện.
- Bộ lập lịch ngắn hạn (lập lịch CPU)
  - Chọn trong số các tiến trình trong hàng đợi sẵn sàng để thực hiện

Bộ lập lịch ngắn hạn vs. Dài hạn

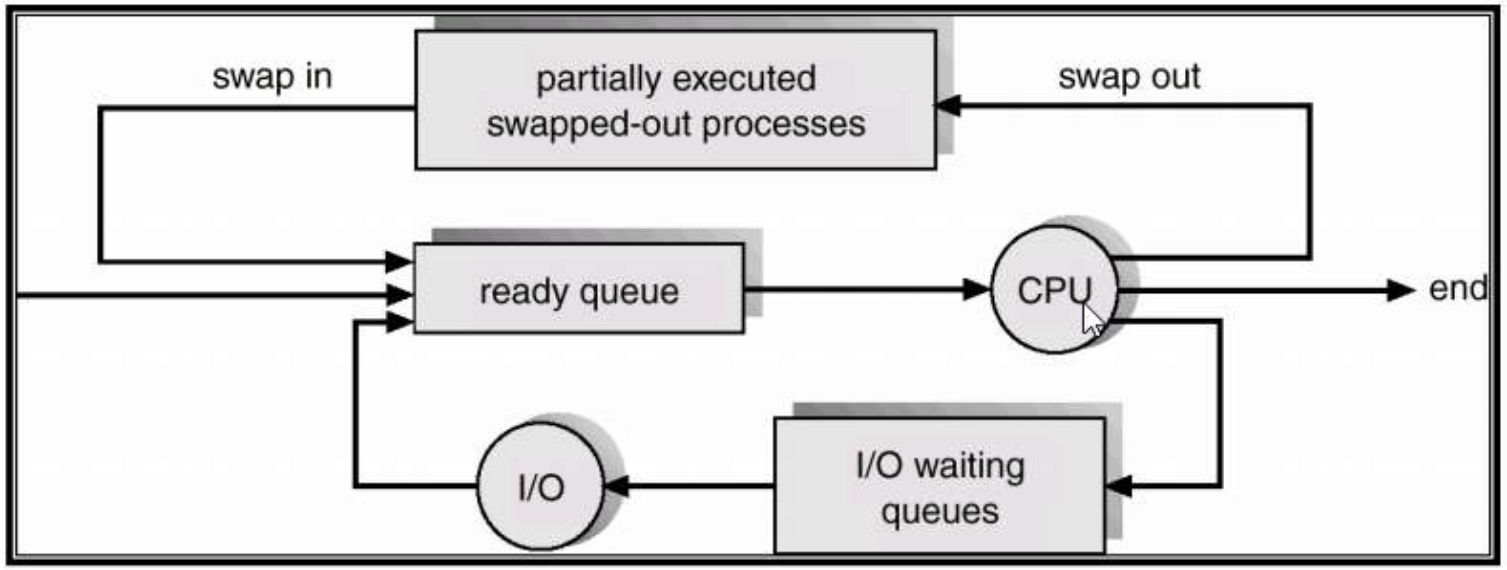
- Tần số thực thi:
  - Ngắn hạn:
    - Thường xuyên
    - Đòi hỏi thực thi nhanh
  - Dài hạn:
    - Không thường xuyên bằng
    - Thể hiện mức độ “đa chương trình”

Bộ lập lịch dài hạn

- Hai loại tiến trình:
  - Giới hạn I/O
  - Giới hạn CPU
- Chọn một kết hợp tốt các tiến trình giới hạn vào /ra và các tiến trình giới hạn CPU.
- Một số hệ thống phân chia thời gian không có bộ lập lịch dài hạn (Unix)

Bộ lập lịch trung hạn

- Sử dụng trong một số HĐH phân chia thời gian



Chuyển đổi ngữ cảnh...

- Chuyển đổi CPU cho một tiến trình khác
- Ngữ cảnh tiến trình
- Hoạt động chuyển đổi ngữ cảnh
- Kernel lưu lại ngữ cảnh của tiến trình cũ trong PCB và tải ngữ cảnh được lưu của tiến trình mới được lập lịch

...Chuyển đổi ngữ cảnh

- Thời gian chuyển đổi ngữ cảnh: lãng phí
  - Phụ thuộc vào máy, thông thường từ 1 đến 1000 micro giây
  - Phụ thuộc vào sự hỗ trợ của phần cứng
  - Các kĩ thuật quản lý bộ nhớ
  - Bottleneck $\rightarrow$ sử dụng các cấu trúc mới như thread để tránh nút cổ chai này

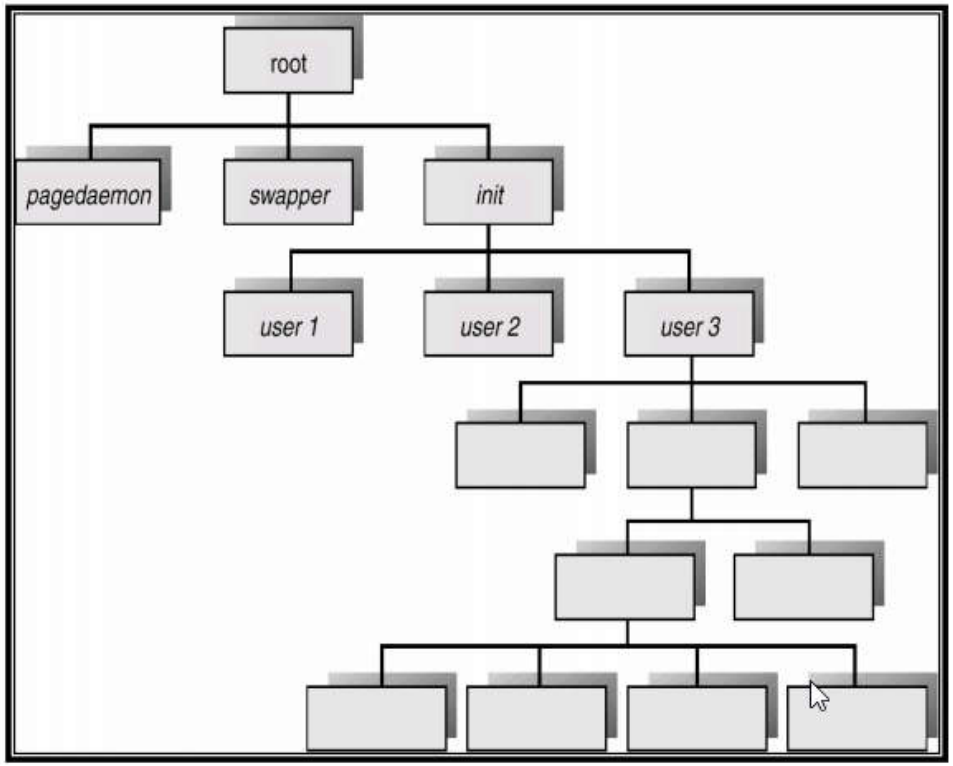
Các thao tác trên tiến trình

- Tạo tiến trình
- Hủy tiến trình

Tạo tiến trình

- Một tiến trình có thể tạo ra nhiều tiến trình con, qua lời gọi hệ thống create_process
  - Tiến trình cha
  - Tiến trình con

Cây tiến trình



Tạo tiến trình

- Chia sẻ tài nguyên
  - Tất cả các tài nguyên
  - Một phần tài nguyên
  - Cũng có thể không chia sẻ tài nguyên
- Thực thi
  - Thực thi đồng thời
  - Thực thi tuần tự

Tạo tiến trình trong Unix

- Mỗi tiến trình có một ID (PID)
- Gọi lời gọi hệ thống fork() để tạo tiến trình mới
- Tiến trình cha có thể đợi hoặc thực hiện đồng thời với tiến trình con
- Không gian địa chỉ của tiến trình con là một bản sao của không gian địa chỉ tiến trình cha
- Mã trả về từ fork()
- Tiến trình con có thể gọi lời gọi hệ thống execlp() để tải một chương trình mới vào thực hiện

Fork() Example

```
#include <stdio.h>
main( int argc, char *argv[] ) {
   int pid;
   /* fork a child process */
   pid = fork();
   /* check fork() return code */
   if ( pid < 0 ) {
        /* some error occurred */
        fprintf( stderr, "Fork failed!\n" );
        exit( -1 );
   } else if ( pid == 0 ) {
        /* this is the child process */
        execlp( "/bin/ls", "ls", NULL );          /* morph into "ls" */
   } else {
        /* this is the parent process. Wait for child to complete */
        wait( NULL );
        printf( "Child completed -- parent now exiting.\n" );
        exit( 0 );
   }
}
```

Hủy tiến trình

- Tiến trình thực thi xong
  - Hệ điều hành thực hiện lệnh exit
  - (có thể) trả về dữ liệu cho tiến trình cha
  - Hủy các tài nguyên đã được phân phối cho tiến trình
- Tiến trình bị hủy bởi một tiến trình khác (tiến trình cha)
  - Tiến trình cha cần biết chỉ số của tiến trình con

Hủy tiến trình

- Tiến trình cha dừng sự hoạt động của tiến trình con vì
  - Tiến trình con dùng quá tài nguyên cho phép
  - Nhiệm vụ của tiến trình con không còn quan trọng
  - Tiến trình cha thoát và Hệ điều hành thực thi cơ chế "hủy theo dây chuyền" (cascading termination)
- Không thực hiện cơ chế hủy theo dây chuyền, tiến trình init trở thành tiến trình cha

Hợp tác giữa các tiến trình

- Các tiến trình cộng tác
  - Tiến trình độc lập
    - Không bị ảnh hưởng bởi tiến trình khác
    - Không chia sẻ dữ liệu
  - Tiến trình hợp tác
    - Bị ảnh hưởng bởi tiến trình khác
    - Dùng chung dữ liệu
- Cần các kĩ thuật giao tiếp/ đồng bộ tiến trình

Vì sao hợp tác tiến trình?

- Chia sẻ thông tin
  - Đồng thời truy cập đến tài nguyên chia sẻ
- Tăng tốc độ tính toán
  - Chia thành các bài toán con, thực thi song song
  - Chỉ có được trong các hệ thống có nhiều thành phần xử lý (đa CPU, đa kênh vào /ra)
- Tính module hóa: Chia nhỏ các chức năng
- Tiện dụng
  - Có thể thực hiện nhiều nhiệm vụ tại một thời điểm

Bài toán "Producer - Consumer"

- Nhà sản xuất (producer)
  - Sinh sản phẩm (thông tin/hàng hoá)
- Người tiêu dùng (consumer)
  - Dùng thông tin/hàng hoá do Nhà sản xuất tạo ra
- Bộ đệm chung
  - Không giới hạn/Giới hạn
  - Hỗ trợ bởi hệ điều hành (thông qua IPC- Inter Process Communication)
  - Do lập trình viên tạo ra bằng cách sử dụng bộ nhớ chia sẻ

...Bài toán Producer – Consumer...

- Các biến dùng chung

```
#define BUFFER_SIZE 10

typedef struct {
    . . .
} item;

item buffer[BUFFER_SIZE];
int in = 0;
int out = 0;
```

Mã lệnh cho Producer

```
while (1) {
    /* produce an item in nextProduced */
    while (((in + 1) % BUFFER_SIZE) == out)
        /* do nothing */
    buffer[in] = nextProduced;
    in = (in + 1) % BUFFER_SIZE;
}
```

Mã lệnh cho Consumer

```
while (1) {
    while (in == out)
        ; // do nothing

    nextConsumed = buffer[out];
    out = (out + 1) % BUFFER_SIZE;
    /* consume the item in nextConsumed */
}
```

Truyền thông giữa các tiến trình

- Các tiến trình có thể giao tiếp thông qua tính năng truyền thông giữa các tiến trình (IPC) của HĐH
- IPC cho phép các tiến trình sử dụng không gian địa chỉ để giao tiếp và đồng bộ
- Hệ thống truyền thông báo

Hệ thống truyền thông báo...

- Giao tiếp giữa các tiến trình người dùng không cần chia sẻ dữ liệu mà thông qua việc truyền thông báo
- Hai thao tác chính
  - Gửi
  - Nhận

...Hệ thống truyền thông báo...

- Thông báo
  - Kích thước cố định
  - Kích thước thay đổi
- Hai tiến trình P & Q truyền thông bằng cách gửi và nhận thông báo
  - Tạo thành một kết nối truyền thông

...Hệ thống truyền thông báo

- Các phương pháp để thiết lập liên kết và các thao tác gửi /nhận
  - Truyền thông trực tiếp/ gián tiếp
  - Truyền thông đối xứng/ bất đối xứng
  - Gửi bản sao hay gửi tham chiếu
  - Các thông điệp kích thước cố định hoặc thay đổi

Định danh

- Các tiến trình muốn giao tiếp với nhau cần có một cách thức để tham chiếu đến nhau!

Truyền thông trực tiếp...

- Phải khai báo tên của người nhận/ gửi trong khi giao tiếp
  - send (P, message)
  - receive (Q, message)
- Tính chất
  - Tự động thiết lập liên kết khi cần giao tiếp
  - Mỗi liên kết có đúng hai tiến trình
  - Có đúng một liên kết giữa bất kì 2 cặp tiến trình

...Truyền thông trực tiếp

- Hệ thống vừa xét: đối xứng địa chỉ
- Hệ thống bất đối xứng địa chỉ
  - Chỉ có người gửi định danh người nhận
  - Người nhận không cần định danh người gửi
    - Send(P, message)
    - Receive(id, message)
  - Thay đổi tên một tiến trình $\rightarrow$ duyệt lại toàn bộ các tiến trình

Truyền thông gián tiếp...

- Gửi và nhận thông qua hòm thư hoặc cổng
- Mỗi hòm thư có một số định danh
- Các thao tác gửi/ nhận
  - Send(A, message) – A là hòm thư
  - Receive(A, message) – nhận một thông báo từ hòm thư A

...Truyền thông gián tiếp

- Tính chất
  - Một liên kết được thiết lập giữa hai tiến trình chỉ khi cả hai là thành viên của một hòm thư
  - Một liên kết có thể có nhiều hơn hai tiến trình
  - Giữa hai tiến trình có thể có nhiều liên kết

Đồng bộ hóa

- Truyền thông báo có thể là phong tỏa hay không phong tỏa (đồng bộ hoặc không đồng bộ)
  - Phong tỏa gửi
  - Không phong tỏa gửi
  - Phong tỏa nhận
  - Không phong tỏa nhận

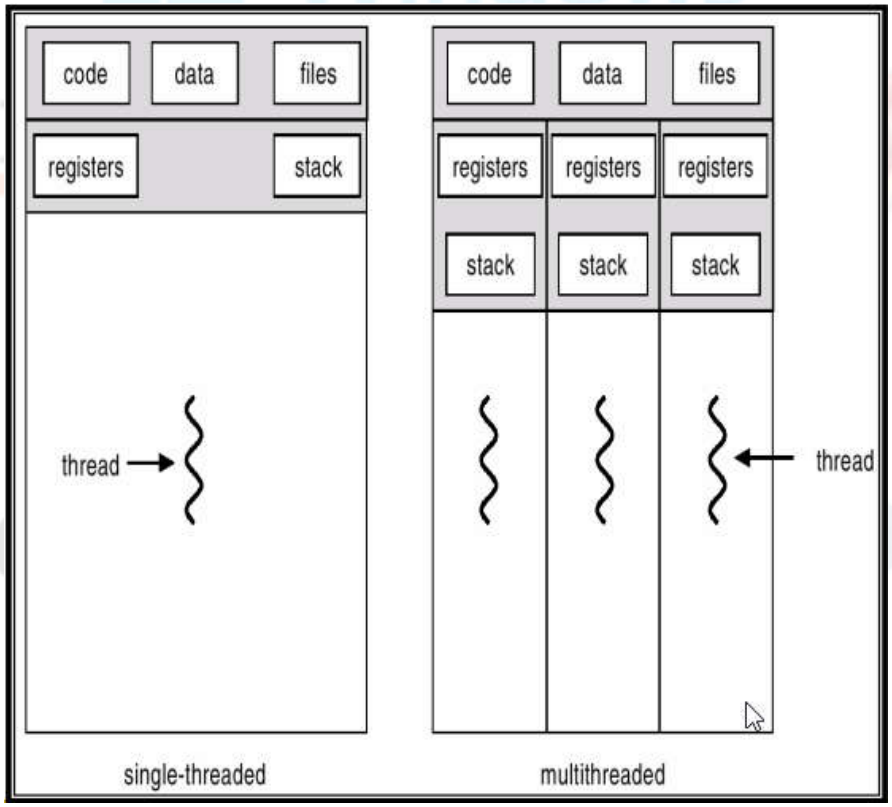
Lưu trữ bộ đệm

- Hàng đợi tạm: các cách thiết kế
  - Zero capacity
  - Bounded capacity
  - Unbounded capacity

Luồng (Thread)

- Tại sao đa luồng?
  - Trình duyệt Web
    - Luồng hiển thị ảnh hoặc văn bản
    - Luồng nhận dữ liệu từ mạng
  - Trình word
    - Luồng hiển thị văn bản
    - Luồng đọc kí tự từ người dùng
    - Luồng thực hiện kiểm tra ngữ pháp và chính tả ngầm

Tiến trình đơn luồng và tiến trình đa luồng



Tại sao đa luồng?

- Một ứng dụng đơn cần thực hiện một số nhiệm vụ tương tự đồng thời
  - Ví dụ: Máy chủ web
- Giải pháp đa tiến trình trước đây
  - Nặng nề hơn
  - Lãng phí do các tiến trình cùng thực hiện một số nhiệm vụ tương tự

Lợi ích

- Tính đáp ứng
- Chia sẻ tài nguyên
- Kinh tế
  - Phân phối tài nguyên và bộ nhớ cho tiến trình tốn kém
- Tận dụng các kiến trúc đa xử lý

Luồng mức người dùng (User - Thread)

- Quản lý luồng được thực hiện bởi thư viện luồng ở mức người dùng
- Examples
  - POSIX *Pthreads*
  - Mach *C-threads*
  - Solaris *threads*

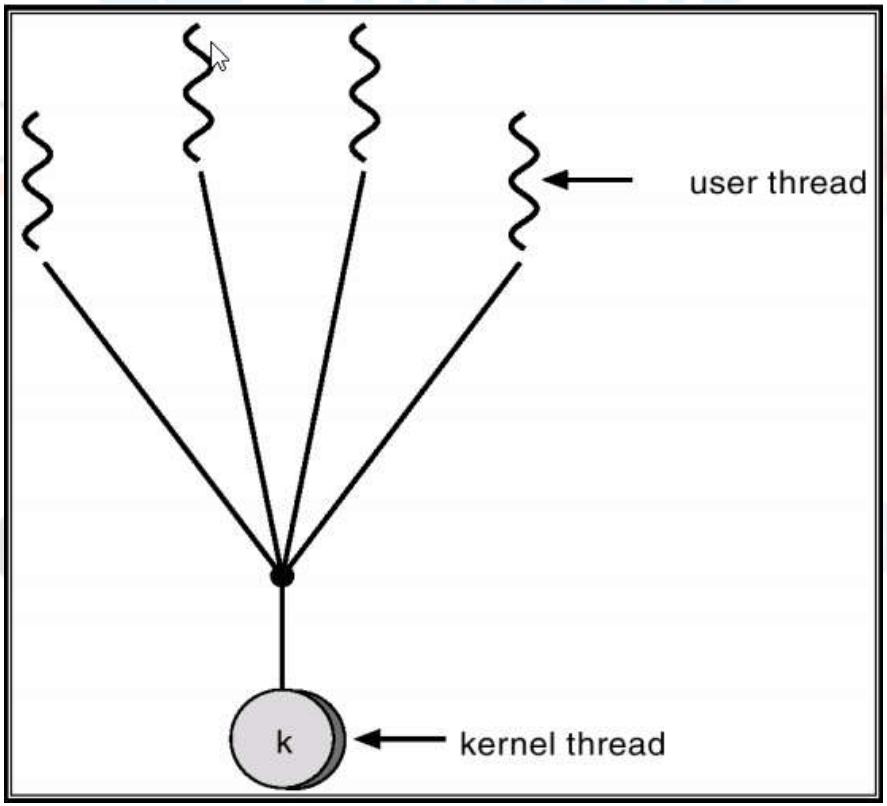
Luồng mức nhân

- Hỗ trợ trực tiếp bởi hệ điều hành
- Ví dụ:
  - Windows 95/98/NT/2000/7/8/10/2003/2012/2019
  - Solaris
  - Tru64 UNIX
  - BeOS
  - Linux

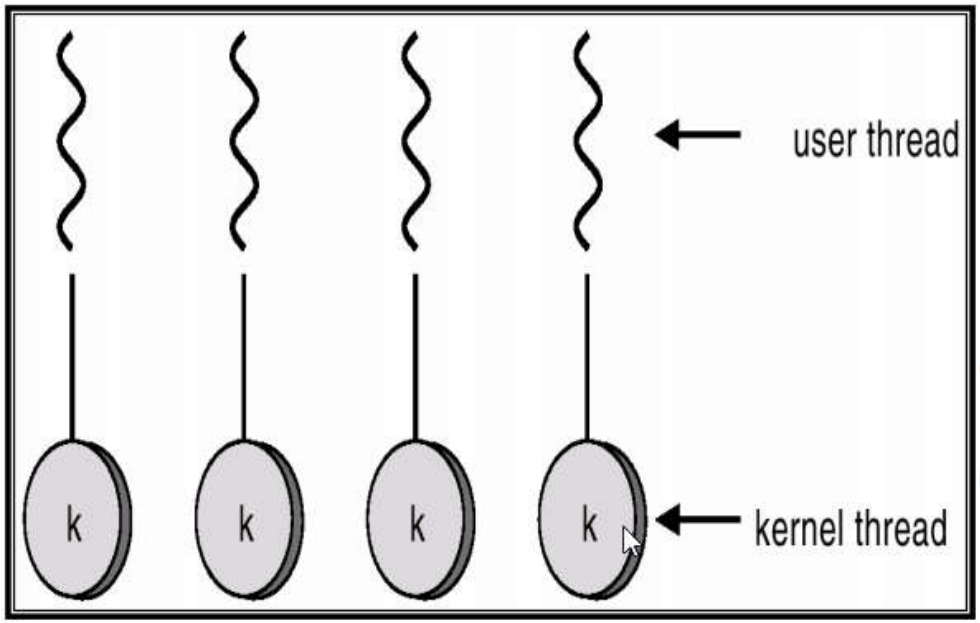
Các mô hình đa luồng

- Mô hình Many-to-One
- Mô hình One-to-One
- Mô hình Many-to-Many

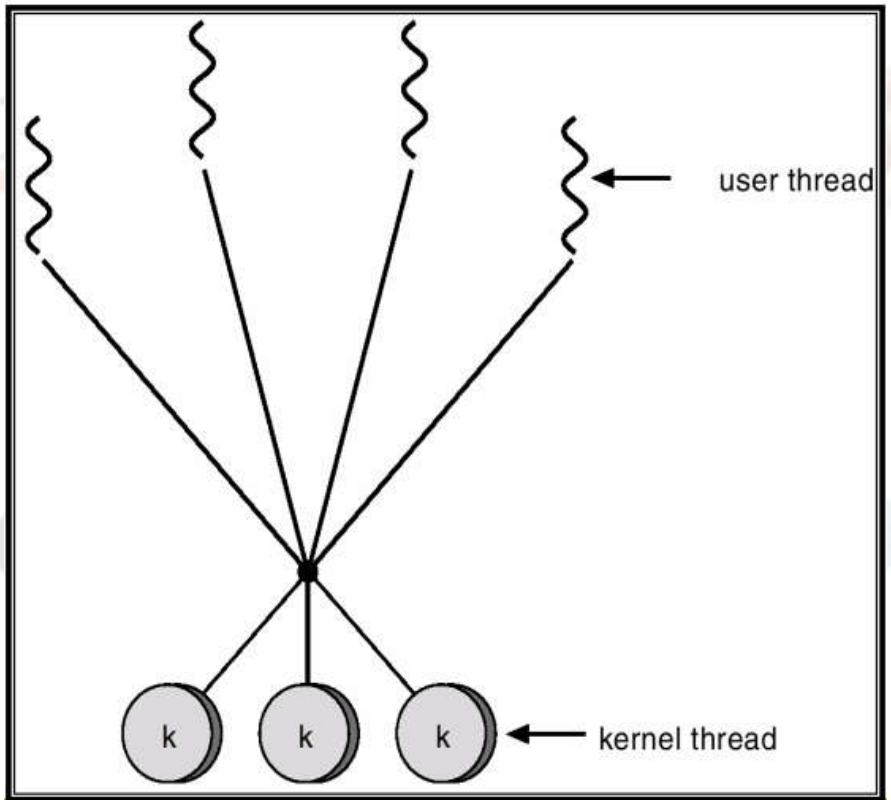
Mô hình Many-to-One



Mô hình One-to-One



Mô hình Many-to-Many



Pthreads

- Chuẩn Posix (IEEE 1003.1c), API cho việc tạo và đồng bộ tiến trình
- API xác định giao diện của thư viện,thực thi tùy thuộc vào cài đặt thư viện.
- Phổ biến trong dòng hệ điều hành Unix.

```c
#include <pthread.h>
#include <stdio.h>

int sum; /* this data is shared by the thread(s) */
void *runner(void *param); /* the thread */

main(int argc, char *argv[])
{
   pthread_t tid; /* the thread identifier */
   pthread_attr_t attr; /* set of thread attributes */
   if (argc != 2) {
      fprintf(stderr,"usage:  a.out <integer value>\n");
      exit();
   }
   if (atoi(argv[1]) < 0) {
      fprintf(stderr,"%d must be <= 0\n",atoi(argv[1]));
      exit();
   }
   /* get the default attributes */
   pthread_attr_init(&attr);
   /* create the thread */
   pthread_create(&tid,&attr,runner,argv[1]);
   /* now wait for the thread to exit */
   pthread_join(tid,NULL);
   printf("sum = %d\n",sum);
}

/* The thread will begin control in this function */
void *runner(void *param)
{
   int upper = atoi(param);
   int i;
   sum = 0;
   if (upper > 0) {
      for (i = 1; i <= upper; i++)
         sum += i;
   }
   pthread_exit(0);
}
```

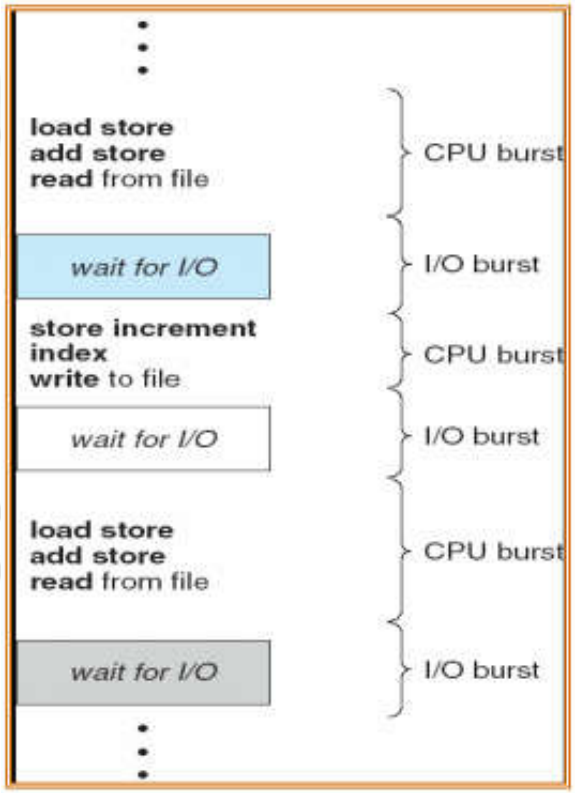
Lập lịch CPU

- Các khái niệm cơ bản
- Điều kiện lập lịch
- Các thuật toán lập lịch

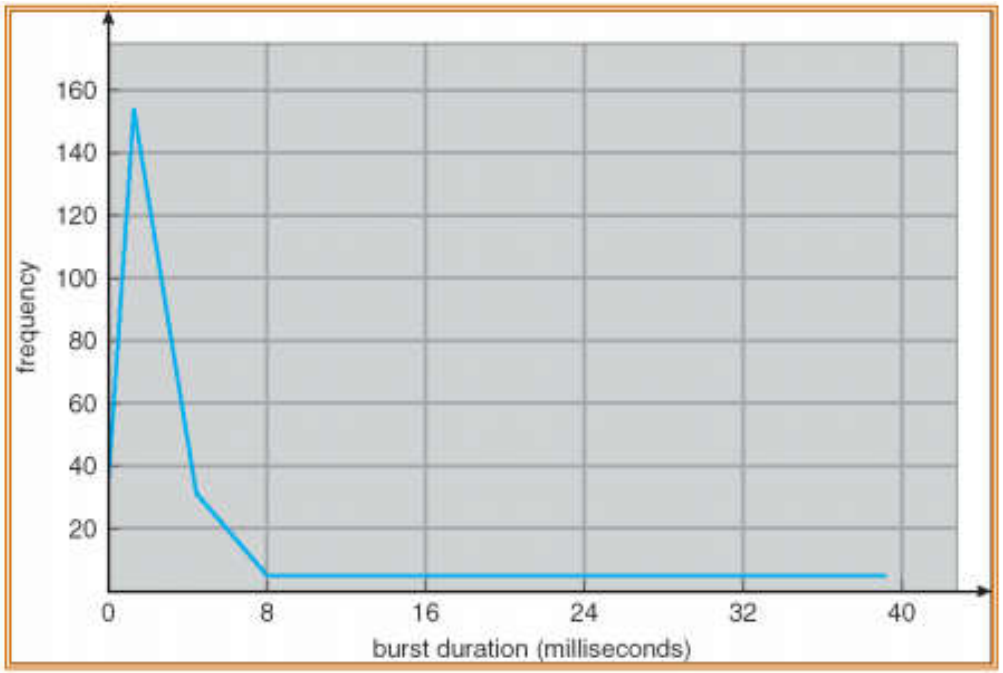
Các khái niệm cơ bản

- Tận dụng tối đa CPU trong đa chương trình
- Chu kỳ của các CPU-I/O burst – Việc thực thi tiến trình là một chu kì của các thực thi bởi CPU và chờ đợi vào ra
- Phân phối CPU burst

Luân phiên giữa các CPU và I/O burst



Biểu đồ tần suất của các CPU burst theo thời gian



Bộ lập lịch CPU

- Chọn trong số các tiến trình trong bộ nhớ trong và ở trạng thái ready để thực hiện (trao quyền sử dụng CPU cho tiến trình đó)
- Việc lập lịch có thể được thực hiện trong một số trường hợp
  1. Tiến trình trạng thái running chuyển sang waiting
  2. Tiến trình trạng thái running chuyển sang trạng thái ready
  3. Có một tiến trình ở trạng thái waiting chuyển sang trạng thái ready
  4. Tiến trình hiện tại kết thúc thực thi
- Lập lịch trong chỉ trong các trường hợp 1 và 4 gọi là lập lịch không chiếm đoạt (hay còn gọi là cộng tác)
- Các trường hợp còn lại gọi là lập lịch chiếm đoạt

Bộ điều vận

- Bộ điều vận có nhiệm vụ chuyển điều khiển CPU cho tiến trình được lựa chọn bởi bộ lập lịch ngắn hạn
- Chức năng của bộ điều vận bao gồm
  - Chuyển đổi ngữ cảnh
  - Chuyển sang user mode
  - Chuyển điều khiển đến một vị trí xác định trong chương trình người dùng để khởi động lại chương trình
- Độ trễ điều vận
  - Thời gian dừng một tiến trình để bắt đầu thực thi tiến trình khác

Điều kiện lập lịch

- Tính tận dụng CPU
  - Thông thường từ 40-90%
- Thông lượng
- Thời gian quay vòng (turnaround time)
- Thời gian chờ
- Thời gian phản ứng

Vấn đề tối ưu các điều kiện

- Cực đại hóa tính tận dụng CPU
- Cực đại hóa thông lượng
- Cực tiểu hóa thời gian quay vòng
- Cực tiểu hóa thời gian đợi
- Cực tiểu hóa thời gian phản ứng

Các thuật toán lập lịch

- Đến trước–Phục vụ trước (FCFS)
- Công việc ngắn nhất trước (SJF)
- Lập lịch với độ ưu tiên
- Lập lịch Round-Robin (RR)
- Lập lịch đa hàng đợi
- Lập lịch với hàng đợi phản hồi

"Đến trước - Phục vụ trước" (FCFS)

- Ba tiến trình đến theo thứ tự là P1, P2, P3
- Thời gian thực hiện của P1, P2 và P3 lần lượt là 24, 3 và 3
- **Biểu đồ Gantt trong lập lịch FCFS**

- **Thời gian chờ=Thời gian kết thúc-Thời gian đến-Thời gian chạy**

  Vậy P1=24-0-24= 0

  P2=27-0-3 =24

  P3=30-0-3 =27

  Thời gian chờ trung bình = 51/3=17

- **Thời gian lưu=Thời gian chờ+Thời gian chạy**

  Vậy P1 = 0+24 = 24

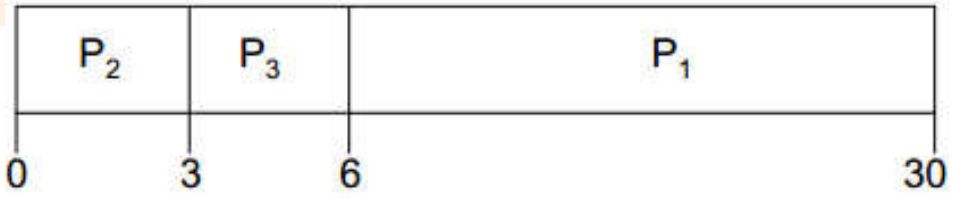
  P2 = 24+3 = 27

  P3 = 27+3 = 30

  Thời gian lưu trung bình=81/3=27

..."Đến trước - Phục vụ trước" (FCFS)

- Nếu các tiến trình đến theo thứ tự P2, P3, P1
- Biểu đồ Gantt cho lập lịch



- Thời gian chờ của P1 = 6; P2 = 0; P3 = 3
- Thời gian chờ trung bình: (6 + 0 + 3)/3 = 3
- Tốt hơn nhiều so với trường hợp trên
- Convoy effect (hiệu quả hộ vệ)

Công việc ngắn nhất trước (SJF-Shortest Job First))

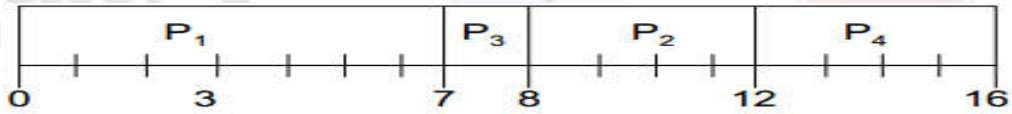
- Liên kết mỗi tiến trình với độ dài của “CPU burst” tiếp theo. Sử dụng độ dài này để lập lịch tiến trình với thời gian ngắn nhất.
- Hai dạng
  - Không chiếm đoạt–Một khi CPU đã được gán cho tiến trình, CPU không thể bị chiếm đoạt bởi một tiến trình nào khác (Non-preemptive).
  - Chiếm đoạt–Nếu một tiến trình mới đến với độ dài CPU burst nhỏ hơn thời gian thực thi còn lại của tiến trình sở hữu CPU, tiến trình mới được chiếm hữu CPU – Dạng “Thời gian còn lại ngắn nhất trước” (SRN) (Preemptive)
- SJF tối ưu thời gian chờ

SJF không chiếm đoạt (SJN-Shortest Job Next):

- **Ví dụ:**

Process	Arrival Time	Burst Time
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

- **Biểu đồ Gantt**



- **Thời gian chờ=Thời gian kết thúc-Thời gian đến-Thời gian chạy**

  Vậy P1 = 7 – 0 - 7 = 0

  P2 = 12 - 2 - 4 = 6

  P3 = 8 – 4 - 1 = 3

  P4 = 16 – 5 – 4 = 7

  Thời gian chờ trung bình = 16/4 = 4

- **Thời gian lưu=Thời gian chờ+Thời gian chạy**

  Vậy P1 = 0 + 7 = 7

  P2 = 6 + 4 = 10

  P3 = 3 + 1 = 4

  P4 = 7 + 4 = 11

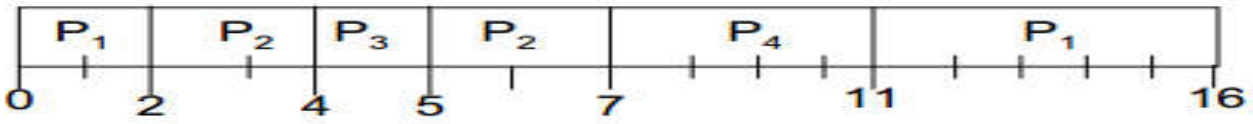
  Thời gian lưu trung bình=32/4=8

SJF chiếm đoạt (SRN-Shortest Remaining Next):

- **Ví dụ:**

Process	Arrival Time	Burst Time
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

- **Biểu đồ Gantt**



- **Thời gian chờ=Thời gian kết thúc-Thời gian đến-Thời gian chạy**

  Vậy P1 = 16 – 0 - 7 = 9

  P2 = 7 - 2 - 4 = 1

  P3 = 5 – 4 - 1 = 0

  P4 = 11 – 5 – 4 = 2

  Thời gian chờ trung bình = 12/4 = 3

- **Thời gian lưu=Thời gian chờ+Thời gian chạy**

  Vậy P1 = 9 + 7 = 16

  P2 = 1 + 4 = 5

  P3 = 0 + 1 = 1

  P4 = 2 + 4 = 6

  Thời gian lưu trung bình=28/4=7

Lập lịch với độ ưu tiên

- Mỗi tiến trình liên kết với một độ ưu tiên (số nguyên) xác định
- CPU được phân phối cho tiến trình với độ ưu tiên cao nhất (giá trị độ ưu tiên nhỏ nhất)
  - Chiếm đoạt (preemptive)
  - Không chiếm đoạt (non preemptive)
- SJF là lập lịch với độ ưu tiên trong đó độ ưu tiên chính là khoảng CPU burst tiếp theo
- Vấn đề: "Chết đói" – Những tiến trình với độ ưu tiên thấp có thể sẽ không bao giờ được gán CPU
  - Giải pháp: các tiến trình tăng độ ưu tiên theo thời gian

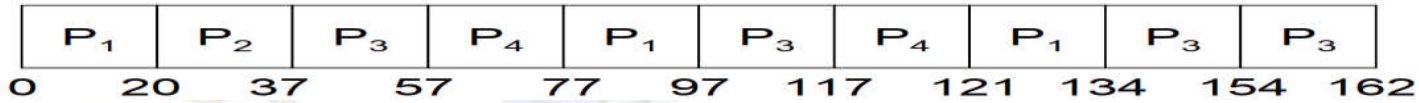
Round Robin (RR)

- Mỗi tiến trình được gán một lượng tử thời gian Q (Quantum) (thường từ 10 – 100 mili giây) để sử dụng CPU
- Hết lượng tử thời gian, tiến trình hiện tại bị tước CPU và đặt vào hàng đợi sẵn sàng nếu chưa kết thúc
- Hiệu năng
  - FIFO
  - Q phải đủ lớn so với thời gian chuyển giao ngữ cảnh

Vídụ: RR với lượng tử thời gian = 20

Process	Burst Time
P_1	53
P_2	17
P_3	68
P_4	24

- **Biểu đồ Gantt**



- **Thời gian chờ=Thời gian kết thúc-Thời gian đến-Thời gian chạy**

Vậy P1 = 134 – 0 - 53 = 81

P2 = 37 - 0 - 17 = 20

P3 = 162 – 0 - 68 = 94

P4 = 121 – 0 – 24 = 97

Thời gian chờ trung bình = 292/4 = 73

- **Thời gian lưu=Thời gian chờ+Thời gian chạy**

Vậy P1 = 81 + 53 = 134

P2 = 20 + 17 = 37

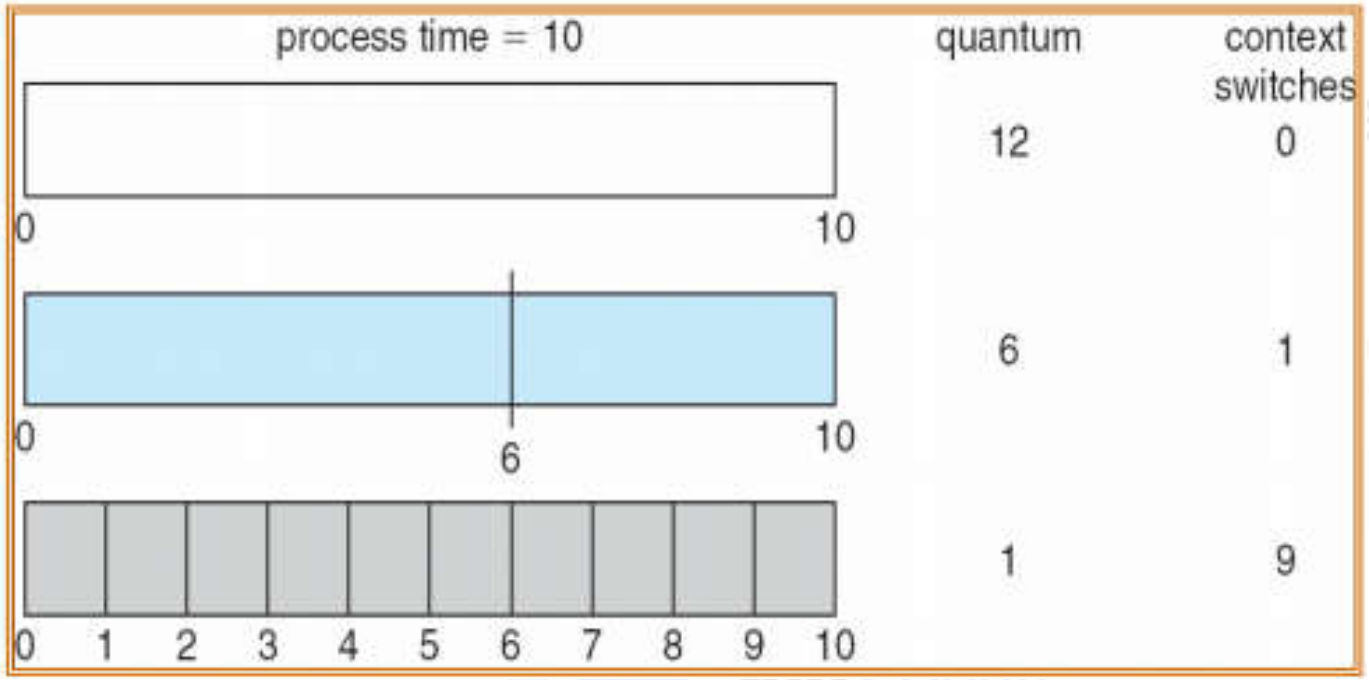
P3 = 97 + 68 = 145

P4 = 97 + 24 = 101

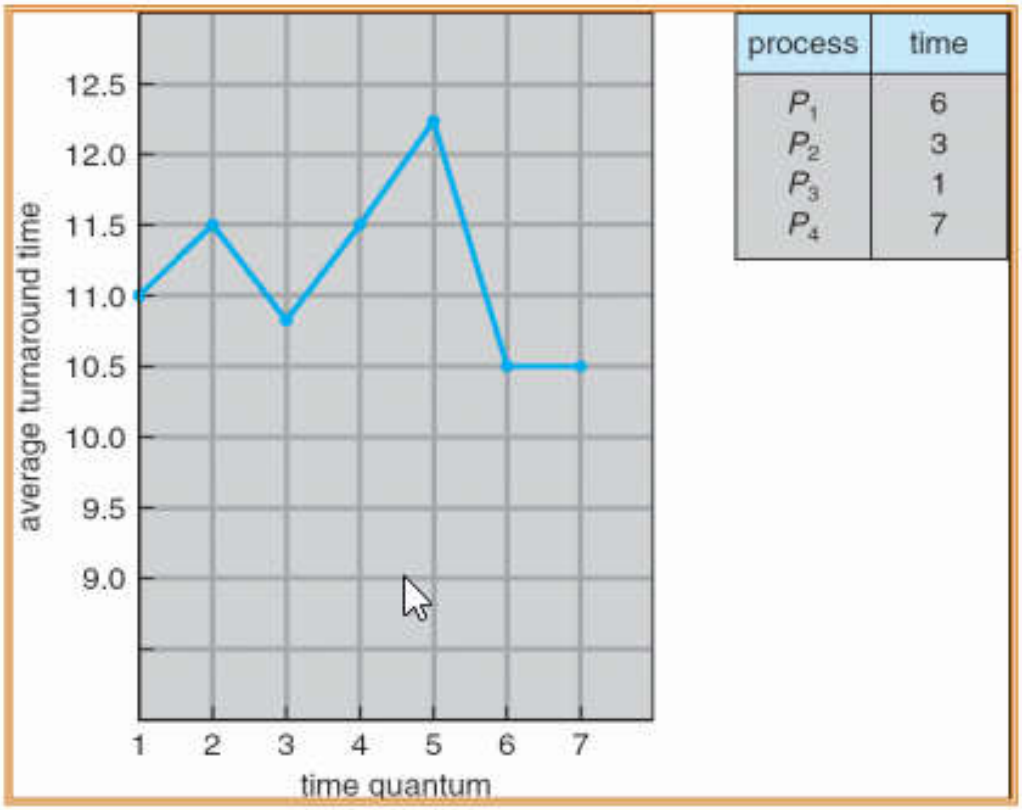
Thời gian lưu trung bình=417/4=104,25

RR tuy có thời gian quay vòng trung bình cao hơn FCFS và SJF nhưng có tính phản ứng tốt hơn

Thời gian lượng tử và thời gian chuyển đổi ngữ cảnh



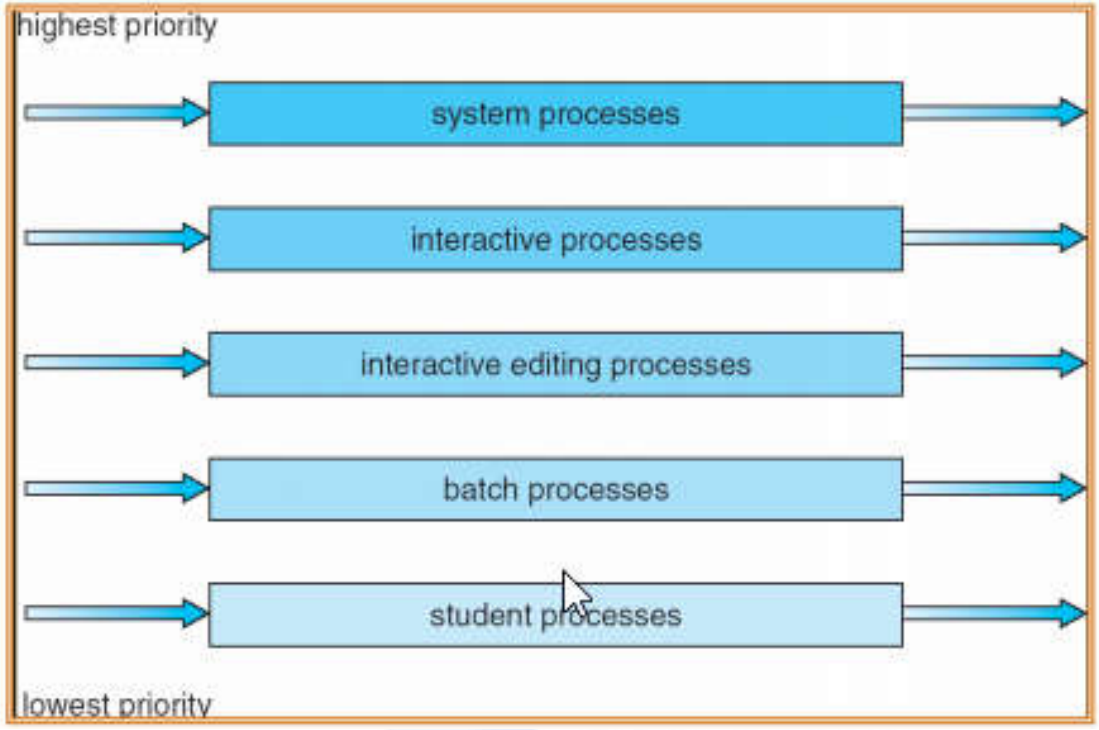
Biến đổi thời gian quay vòng theo thời gian lượng tử



Lập lịch với Hàng đợi nhiều mức

- Sử dụng nhiều hơn một hàng đợi sẵn sàng
  - Ví dụ: hàng đợi nền trước (foreground queue) cho các chương trình tương tác, hàng đợi nền sau (background queue) cho các chương trình duy trì hệ thống, các chương trình theo lô
- Sử dụng các thuật toán lập lịch khác nhau cho các hàng đợi
  - Ví dụ: RR cho hàng đợi nền trước, FCFS cho hàng đợi nền sau
- Chia thời gian cho các hàng đợi
  - Ví dụ: 80% thời gian cho hàng đợi nền trước, 20% cho nền sau

Hàng đợi nhiều mức (Multi Level Queue)



Hàng đợi nhiều mức phản hồi (MFQ)

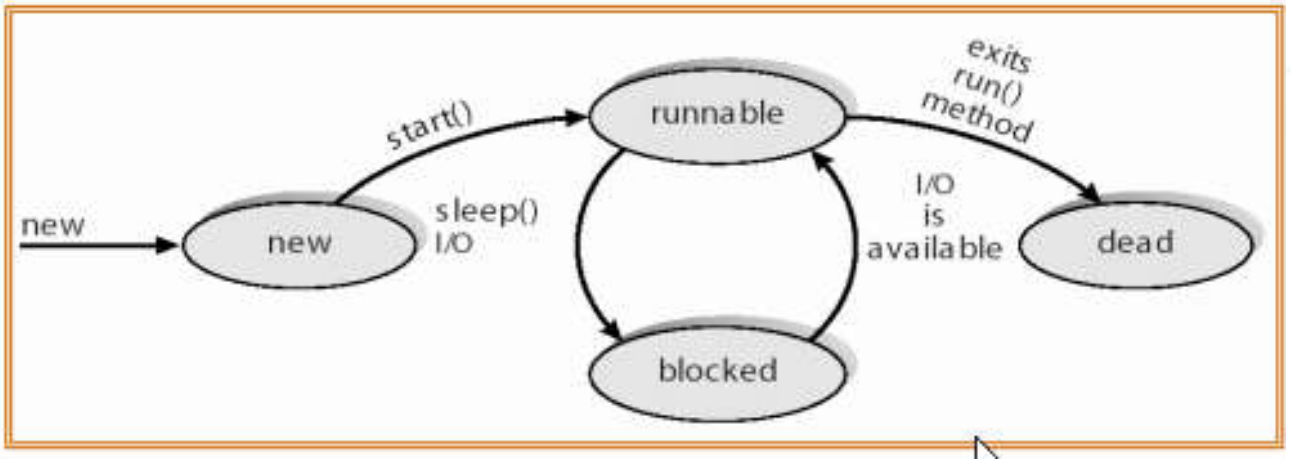
- Không cố định tiến trình trên một hàng đợi
- Ví dụ: các hàng đợi với các độ ưu tiên khác nhau
  - Các tiến trình gắn với Vào/ Ra ở hàng đợi có độ ưu tiên cao
  - Chuyển các tiến trình sử dụng CPU đến các hàng đợi có độ ưu tiên thấp
  - Chuyển các tiến trình phải đợi lâu đều đặn lên phía trên

Lập lịch các luồng trong Java

- JVM sử dụng thuật toán Preemptive, và dựa trên độ ưu tiên trong lập lịch
- Hàng đợi FIFO được sử dụng nếu có nhiều luồng cùng mức ưu tiên

Lập lịch các luồng trong Java

- JVM lập lịch khi:
  1. Tiến trình đang chạy ra khỏi trạng thái runnable
  2. Một tiến trình có độ ưu tiên cao hơn vào trạng thái runnable



Time-Slicing

- Do JVM không hỗ trợ lát cắt thời gian (Time-Slicing), hàm yield() có thể được sử dụng:

```
while (true) {
        // perform CPU-intensive task
        . . .
        Thread.yield();
}
```

Thread.yield() chuyển điều khiển cho luồng khác

Nội dung đã học

- Lý do lập lịch
- Các dạng lập lịch
  - FCFS
  - SJF
  - RR
  - MFQ
- Lập lịch trong Java


Question

Câu hỏi ôn tập...

- 3.1 Palm OS không cung cấp phương tiện để xử lý đồng thời. Thảo luận về ba vấn đề cơ bản mà tiến trình đồng thời thêm vào hệ điều hành
- 3.2 Bộ xử lý Sun UltraSPARC có nhiều bộ thanh ghi. Mô tả hành động của một chuyển đổi ngữ cảnh khi ngữ cảnh mới được tải vào một trong các bộ thanh ghi rồi. Điều gì khác sẽ xảy ra nếu ngữ cảnh ở trong bộ nhớ và mội thanh ghi đều bận
- 3.3 Khi một tiến trình tạo ra một tiến trình mới bằng cách sử dụng hoạt động fork (), trạng thái nào sau đây được chia sẻ giữa tiến trình cha và tiến trình con?
  - a. Stack
  - b. Heap
  - c. Shared memory segments

...Câu hỏi ôn tập

- 3.4. Trong cơ chế RPC, hãy xem xét ngữ nghĩa "đúng một lần". Liệu các thuật toán để thực hiện ngữ nghĩa này thực hiện đúng ngay cả khi tin "ACK" nhắn lại cho khách hàng bị mất do lỗi mạng? Mô tả trình tự của thông điệp và khi nào "đúng một lần» được duy trì
- 3.5 Giả sử một hệ thống phân tán thường hay bị lỗi của máy chủ. Cơ chế nào sẽ được yêu cầu để đảm bảo ngữ nghĩa "đúng một lần" cho việc thực hiện RPCs?