# Các tiến trình

# Mô hình xử lý tiến trình

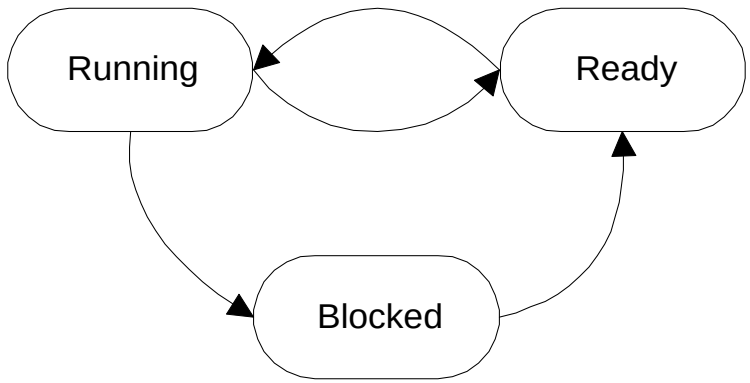
- Mô hình khái niệm
  - giả song song; mỗi tiến trình có một counter IP riêng, chạy như song song
- Mô hình thực tế
  - Chia sẻ thời gian CPU, nhảy qua nhảy lại giữa các tiến trình. Mỗi thời điểm chỉ 1 tiến trình.

# Tiến trình con

- Unix
  - Tiến trình cha sinh tiến trình con
  - Tiến trình con có thể sinh ra tiến trình cháu
  - Tất cả cha, con, cháu đều có thể chạy đồng thời
- DOS
  - Tiến trình cha sẽ ngưng cho tới khi tiến trình con kết thúc

# Các trạng thái của tiến trình

- Đang chạy (running)
- Chờ được chạy (ready)
- Bị ngưng (blocked) cho tới khi có 1 sự kiện bên ngoài xảy ra



# Quản lý các tiến trình

- Bảng trạng thái tiến trình
  - Thông tin về riêng tiến trình
  - Thông tin về bộ nhớ đang sử dụng
  - Thông tin về file đang sử dụng
- Mức ưu tiên
  - Các tiến trình được gán cùng mức ưu tiên sẽ được xếp vào cùng 1 hàng đợi
  - Các hàng đợi được xếp theo mức ưu tiên

# Liên lạc giữa tiến trình

- Tình trạng ganh đua
  - Ví dụ 1: A và B cùng in ra vùng đệm máy in, kết quả: đè lẫn nhau
  - Ví dụ 2: A có 5000 trong ngân hàng, A rút 1000 ra và ghi lại vào là 4000. Giữa thao tác của A B trả cho A 3000.

# Thay đổi luân phiên

```
while (TRUE) {
    while (turn != 0);
    critical_section();
    turn =1;
    noncritical_section();
}
```

```
while (TRUE) {
    while (turn != 1) ; //wait
    critical_section();
    turn =0;
    noncritical_section();
}
```

# Giải pháp Peterson

```
#include “prototype.h”

#define FALSE    0
#define TRUE                1
#define N                   2
// N: number of processses

int turn;  // whose turn ?
int interested[N]; //initially FALSE
```

```
void enter_region(int process)
{                           /* process=0/1 */
    int other;
    other = 1- process;
    interested[process] = TRUE;
    turn = process;
    while (turn==process &&
    interested[other]==TRUE) ;
}
 void leave_region(int process)
{
    interested[process]=FALSE;
}
```

# Giải pháp phần cứng TSL

**enter_region:**

```
tsl register, flag
cmp register, 0
jnz enter_region
ret
```

**leave_region:**

```
mov flag, 0
ret
```

**tsl register, flag**

copy flag to register and set flag=1

# Bài toán cung - cầu

```
#include “prototypes.h”
#define N          100      /* number of slots in buffer */
int count=0;                /* number of items in buffer */
void producer(void)
{   int item;
    while (TRUE) {
        produce_item(&item);
        if (count==N) sleep(); /* buffer is full */
        enter_item(item);
        count++;
        if (count==1) wakeup(consumer);
    }
}
```

# …Bài toán cung - cầu

```
void consumer(void)
{   int item;
    while (TRUE) {
        if (count==0) sleep();       /*buffer is emtpy */
        count--;
        if (count=N-1) wakeup(producer);
        consume_item(item);
    }
}
```

# Giải quyết bằng semaphore

```
#include “prototypes.h”

#define N     100

typedef int semaphore;

semaphore mutex=1;
semaphore empty=N;
semaphore full=0;
```

# … semaphore

```
void producer(void)
{   int item;
    while (TRUE) {
        produce_item(&item);
        down(&empty);
        down(&mutex);
        enter_item(item);
        up(&mutex);
        up(&full);
    }
}
```

# … semaphore

```
void consumer(void)
{   int item;
    while (TRUE) {
        down(&full);
        down(&mutex);
        remove_item(&item);
        up(&mutex);
        up(&empty);
        consume_item(item);
    }
}
```

# Sử dụng thông báo

```
#include “prototypes.h”

#define N           100
#define MSIZE       4       /* message size */

typedef int message[MSIZE];
```

# …sử dụng thông báo

```
void producer(void)
{   int item;
    message m;

    while (TRUE) {
        produce_item(&item);
        receive(consumer, &m);
        build_message(&m, item);
        send(consumer, &m);
    }
}
```

# …sử dụng thông báo

```
void consumer(void)
{   int item, i;
    message m;
    for (i=0; i<N; i++) send(producer, &m);
            /* send N empties */
    while (TRUE) {
            receive(producer, &m);
            extract_item(&m, &item);
            send(producer, &m);
            consumer_item(item);
    }
}
```

# Bài toán bữa ăn của các nhà hiền triết

```
#include “prototypes.h”
#define N          5          /* number of philosophers */
void philosopher(int i)
{
    while (TRUE) {
        think();
        take_chopstick(i);
        take_chopstick( (i+1)%N);
        eat();
        put_chopstick(i);
        put_chopstick( (i+1)%N);
    }
}
```

# …bữa ăn: 1 giải pháp

```
#include “prototypes.h”

#define N                   5
#define LEFT                (i-1)%N
#define RIGHT   (i+1)%N
#define THINKING            0
#define HUNGRY              1
#define EATING2

typedef int semaphore;
int state[N];
semaphore mutex=1;
semaphore s[N];      /* 1 semaphore/1philosopher */
```

# …bữa ăn: 1 giải pháp

```
void philosopher(int i)
{
   while (TRUE) {
        think();
        take_chopsticks(i);/* 2 chopsticks*/
        eat();
        put_chopsticks(i);
   }
}
```

# …bữa ăn: 1 giải pháp

```
void take_chopsticks(int i)
{
   down(&mutex);
   state[i]=HUNGRY;
   test(i);          /* try to acquire 2 chopsticks */
   up(&mutex);
   down(&s[i]); /* block if chopsticks were not acquired */
}
```

# …bữa ăn: 1 giải pháp

```
void put_chopsticks(int i)
{
   down(&mutex);
   state[i]=THINKING;
   test(LEFT);         /*see if the left one can now eat */
   test(RIGHT);  /* see if the right one can now eat */
   up(&mutex);
}
```

# …bữa ăn: 1 giải pháp

```
void test(int i)/* i is from 0 to N-1*/
{
   if   (state[i]==HUNGRY &&
         state[LEFT]!=EATING &&
         state[RIGHT]!=EATING) {
         state[i]=EATING;
         up(&s[i]);
   }
}
```

# Bài toán người thợ cạo

```
#include “prototypes.h”

#define CHAIRS     5

typedef int semaphore;

semaphore customers=0;
semaphore barbers=0;
semaphore mutex=1;
int waiting=0;
```

# … người thợ cạo

```
void Barber(void)
{
   while (TRUE) {
        down(customers); /*sleep if #customers=0 */
        down(mutex);/* acquire access to ‘waitin’*/
        waiting--;
        up(barbers);
        up(mutex);      /* release ‘waiting’ */
        cut_hair();
   }
}
```

# … người thợ cạo

```
void Customer(void)
{
    down(mutex);          /* enter critical region */
    if (waiting < CHAIRS) {
        waiting++;
        up(customers);
        up(mutex);
        down(barbers); /*sleep if no free barber */
        get_haircut();
    }
    else up(mutex);       /* shop full, don't wait*/
}
```

# Bài toán đọc và ghi

```
#include “prototypes.h”

typedef int semaphore;

semaphore mutex=1;/* control rc */
semaphore db=1;     /* database control */
int rc=0;                   /* rc */
```

# …Bài toán đọc và ghi

```
void reader(void)
{
    while (TRUE) {
        down(&mutex);
        rc++;
        if (rc==1) down(&db);  /* 1st reader */
        up(&mutex);
        read_data_base();
        down(&mutex);
        rc--;
        if (rc==0) up(&db);          /* last reader */
        up(&mutex);
        use_data_read();             /* non critical section */
    }
}
```

# …Bài toán đọc và ghi

```
void writer(void)
{
   while (TRUE) {
        think_up_data(); /* non critical section */
        down(&db);
        write_data_base();
        up(&db);
   }
}
```