# Đồng bộ

TH 106: Hệ điều hành

Khoa CNTT
ĐH KHTN

# Từ khóa

- Critical section (miền găng): đoạn mã nguồn đọc/ghi dữ liệu lên vùng nhớ chia sẻ
- Race condition (tranh đoạt điều khiển): có tiềm năng bị xen kẻ lệnh giữa các tiểu trình khác nhau trong miền găng
  - Kết quả không xác định
- Mutual exclusion: cơ chế đồng bộ đảm bảo chỉ có một tiểu trình duy nhất được thực hiện trong miền găng tại một thời điểm
- Deadlock: tình trạng các tiểu trình bị khóa mãi mãi
- Starvation: đang thực thi nhưng không có tiến triển.

# Tranh đoạt điều khiển

## Tranh đoạt điều khiển là gì? (Race condition)

```
if (taikhoan – tien_rut >= 0)
        taikhoan = taikhoan – tien_rut;
else
        cout << “Không đủ tiền trong tài khoản”;
```

Là tình huống mà kết quả của chương trình phụ thuộc vào sự điều phối của hệ thống

# Miền găng (critical section)

- Đoạn mã nguồn truy cập đến vùng nhớ dùng chung gọi là **miền găng**

```
…
// bắt đầu miền găng
if (taikhoan – tien_rut >= 0)
        taikhoan = taikhoan – tien_rut;
else
        cout << “Không đủ tiền trong tài khoản”;
// kết thúc miền găng
```

# Giải pháp đúng

- Giải pháp cho bài toán tranh đoạt điều khiển phải thỏa:
  1. Chỉ một tiến trình được thực thi trong miền găng tại một thời điểm
  2. Không có giả thiết về tốc độ CPU, số lượng CPU
  3. Không có tiến trình ở ngoài miền găng có thể khóa không cho tiến trình khác vào miền găng
  4. Không có tiến trình nào chờ đợi mãi mãi để vào miền găng

# Giải pháp

- Giải pháp BUSY WAITING
  - **Giải pháp phần mềm**
    - Sử dụng cờ hiệu
    - Kiểm tra luân phiên
    - Giải pháp Peterson
  - **Giải pháp phần cứng**
    - Vô hiệu hóa ngắt
    - Chỉ thị TSL (Test-and-Set)
- Giải pháp "SLEEP and WAKEUP"

# Giải pháp

- Giải pháp "SLEEP and WAKEUP"
  - **Semaphores**
  - **Monitors**

# Giải pháp

- Giải pháp BUSY WAITING
  - Giải pháp phần mềm
    - Sử dụng cờ hiệu
    - Kiểm tra luân phiên
    - Giải pháp Peterson
  - Giải pháp phần cứng
    - Vô hiệu hóa ngắt
    - Chỉ thị TSL (Test-and-Set)

# Sử dụng cờ hiệu

```
while (TRUE) {
    while (lock == 1); // wait
    lock = 1;
    critical-section ();
    lock = 0;
    noncritical-section ();
}
```

- Có thể hai tiến trình ở trong miền găng cùng một lúc

# Giải pháp

- Giải pháp BUSY WAITING
  - Giải pháp phần mềm
    - Sử dụng cờ hiệu
    - Kiểm tra luân phiên
    - Giải pháp Peterson
  - Giải pháp phần cứng
    - Vô hiệu hóa ngắt
    - Chỉ thị TSL (Test-and-Set)

# Kiểm tra luân phiên

```
while (TRUE) {
while (turn != 0); // wait
    critical-section ();
    turn = 1;
    noncritical-section ();
}
```

(a) Cấu trúc tiến trình A

```
while (TRUE) {
while (turn != 1); // wait
    critical-section ();
    turn = 0;
    noncritical-section ();
}
```

(b) Cấu trúc tiến trình B

- Có thể vi phạm điều kiện tiến trình bên ngoài miền găng ngăn không cho tiến trình khác vào miền găng

# Giải pháp

- Giải pháp BUSY WAITING
  - Giải pháp phần mềm
    - Sử dụng cờ hiệu
    - Kiểm tra luân phiên
    - Giải pháp Peterson
  - Giải pháp phần cứng
    - Vô hiệu hóa ngắt
    - Chỉ thị TSL (Test-and-Set)

# Giải pháp Peterson

```
int turn;              /* lưu lượt hiện tại */
int interested[2];  /* khởi động FALSE*/

void enter_region(int process) {//process sẽ là 0 hoặc 1
    int other;                          /* mã số của tiến trình còn lại
    other = 1 − process;          /* tiến trình còn lại */
    interested[process] = TRUE;  // muốn vào miền găng
    turn = other;
    while (turn == other && interested[other] == TRUE) ;
}

void leave_region (int process) {
    interested[process] = FALSE;
}
```

# Giải pháp

- Giải pháp BUSY WAITING
  - **Giải pháp phần mềm**
    - Sử dụng cờ hiệu
    - Kiểm tra luân phiên
    - Giải pháp Peterson
  - **Giải pháp phần cứng**
    - Vô hiệu hóa ngắt
    - Chỉ thị TSL (Test-and-Set)

# Vô hiệu hóa ngắt

- Giải pháp đơn giản là khi một tiến trình vào miền găng thì nó vô hiệu hóa hết tất cả các ngắt
- → Liệu cho phép các tiến trình người dùng có thể vô hiệu hết ngắt là khả thi!

# Giải pháp

- Giải pháp BUSY WAITING
  - Giải pháp phần mềm
    - Sử dụng cờ hiệu
    - Kiểm tra luân phiên
    - Giải pháp Peterson
  - Giải pháp phần cứng
    - Vô hiệu hóa ngắt
    - **Chỉ thị TSL (Test-and-Set)**

# Chỉ thi TSL (Test-and-Set)

```
enter_region:
    TSL RX, LOCK | chép giá trị lock vào RX và gán lock = 1
    CMP RX, #0           | so sánh với 0
    JNE enter_region     | jump nếu khác 0
    RET                  | vào miền găng

leave_region:
    MOVE   LOCK, #0          | gán lock = 0
    RET                      | trả về
```

# Giải pháp “SLEEP and WAKEUP”

- Giải pháp Peterson và TSL đều đúng, tuy nhiên chiếm thời gian CPU vì vòng lặp kiểm tra liên tục
- → giải pháp sleep and wakeup

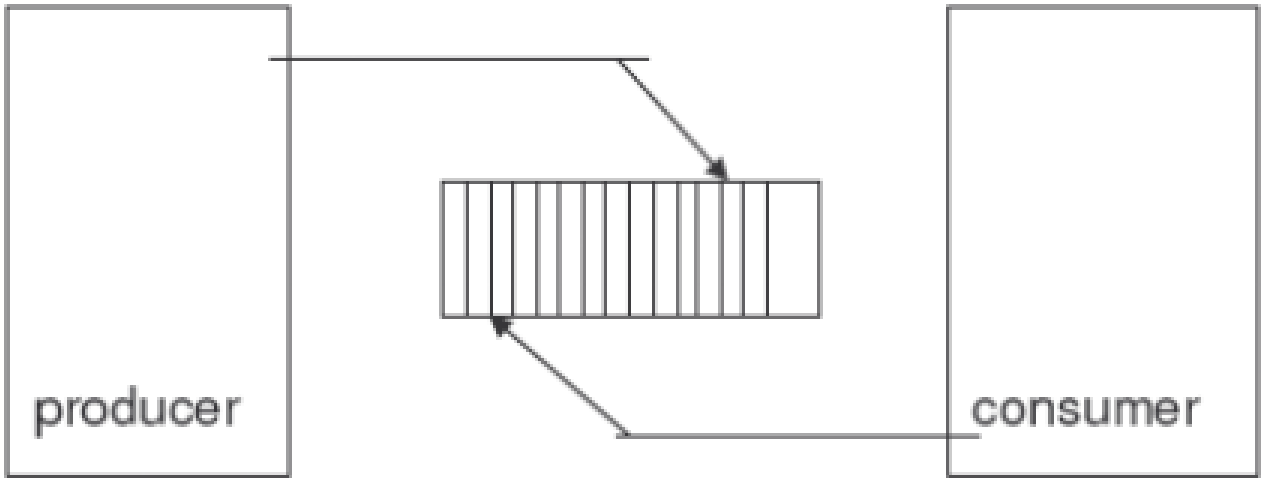
# Ý tưởng chính

```
while (TRUE) {
if (busy){
        blocked = blocked + 1;
        sleep();
    }
else busy = 1;
critical-section ();
busy = 0;
if(blocked){
    wakeup(process);
    blocked = blocked - 1;
    }
noncritical-section ();
}
```

# Semaphore

- Dijkstra đề xuất năm 1965
- Một semaphore là 1 biến có các thuộc tính sau:
  - Một giá trị nguyên dương *e(s)*
  - Một hàng đợi *f(s)* lưu danh sách các tiến trình đang bị khóa
  - Chỉ có hai thao tác được định nghĩa trên semaphore:
    - *Down(s)*: giảm giá trị của semaphore *s* đi 1 đơn vị nếu *e(s)* > 0, và tiếp tục xử lý. Nếu *e(s)* = 0, tiến trình phải chờ đến khi *e(s)* >0.
    - *Up(s)*: tăng giá trị của semaphore *s* lên 1 đơn vị. hệ thống sẽ chọn một trong các tiến trình bị khóa để cho tiếp tục thực thi.

# Bài toán Producer-Consumer

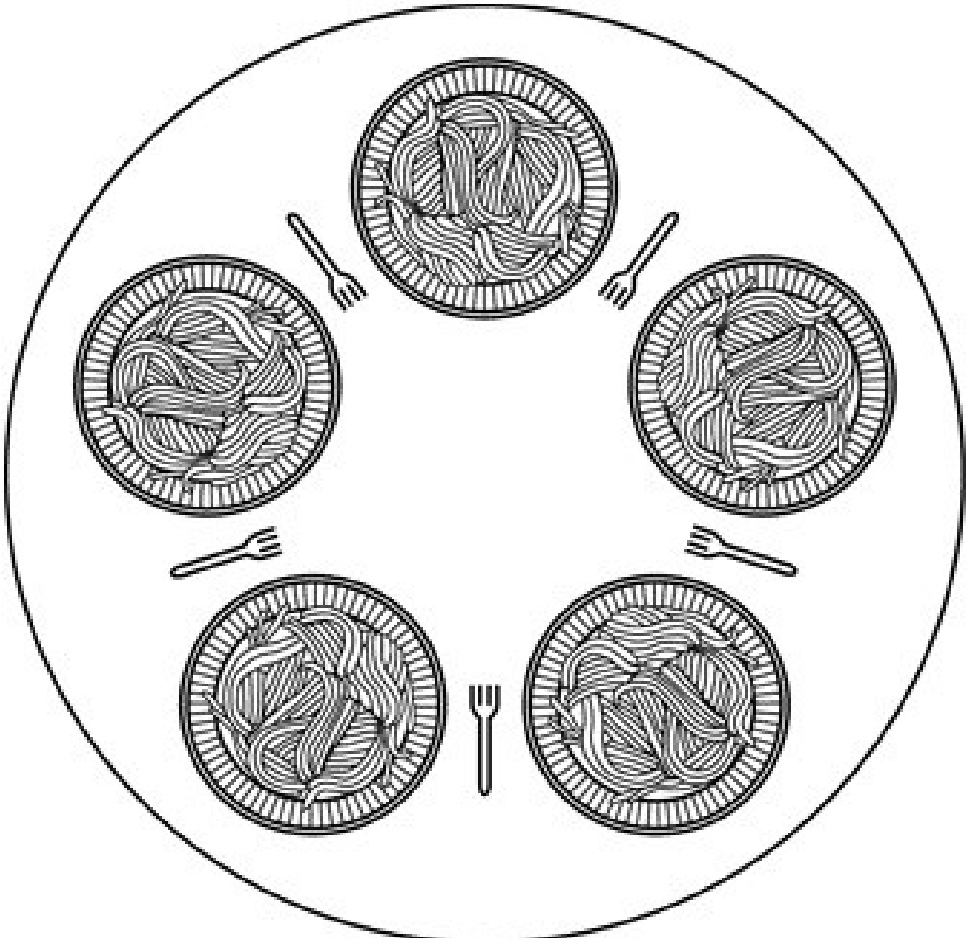


```
#define N 100                       /* kích thước buffer */
semaphore mutex = 1;                /* điều khiển vào miền găng */
semaphore empty = N;                /* đếm số ô rỗng trong buffer*/
semaphore full = 0;                 /* đếm số ô đã chiếm trong buffer*/

void Producer(void) {
    int item;
    while (TRUE) {
        item = produce_item();          /* sản xuất 1 mẫu tin */
        down(&empty);           /* giảm ô rỗng */
        down(&mutex);           /* vào miền găng */
        insert_item(item);      /* đặt mẫu tin vào buffer */
        up(&mutex);             /* rời miền găng */
        up(&full);              /* tăng số ô đã chiếm */
    }
}
```

```
void Consumer(void) {
int item;
while (TRUE) {
     down(&full);                 /* giảm số ô đã chiếm */
     down(&mutex);                /* vào miền găng */
     item a= remove_item();       /* lấy một mẫu tin ra từ buffer*/
     up(&mutex);                  /* rời khỏi miền găng */
     up(&empty);                  /* tăng số ô rỗng */
     consume_item(item);          /* tiêu thụ mẫu tin */
     }
}
```

# Buổi ăn tối của các triết gia



# Giải pháp 1

```
void philosopher(int i)     // mã số triết gia
{
    while (TRUE) {
        think( );               // triết gia đang suy nghĩ
        take_fork(i);           // lấy nĩa bên trái
        take_fork((i+1) % N);       // lấy nĩa bên phải
        eat();                  // ăn spaghetti
        put_fork(i);            // trả lại nĩa bên trái
        put_fork((i+1) % N);        // trả lại nĩa bên phải
    }
}
```

→ Nếu mỗi triết gia đều nắm giữ một nĩa bên trái

# Giải pháp 1*

```
void philosopher(int i)   {// mã số triết gia
    while (TRUE) {
        think( );                  // triết gia đang suy nghĩ
        down(mutex);
        take_fork(i);              // lấy nĩa bên trái
        take_fork((i+1) % N);      // lấy nĩa bên phải
        eat();                     // ăn spaghetti
        put_fork(i);               // trả lại nĩa bên trái
        put_fork((i+1) % N);       // trả lại nĩa bên phải
        up(mutex);
    }
}
```

→ Chỉ một triết gia được ăn tại 1 lúc

# Giải pháp 2

```
void philosopher (int i)   {
    while (TRUE) {
        think();              // suy nghĩ
        take_forks(i);        // lấy nĩa
        eat();                // ăn
        put_forks(i);         // trả lại nĩa
    }
}
```

# Giải pháp 2 (tt)

```
void take_forks(int i)     {
    down(&mutex);          // vào miền găng
    state[i] = HUNGRY;     // lưu trạng thái HUNGRY
    test(i);               // thử lấy hai nĩa
    up(&mutex);            // ra khỏi miền găng
    down(&s[i]);           // khóa nếu không lấy được nĩa
}
void put_forks(i)  {
     down(&mutex);         // vào miền găng
     state[i] = THINKING;  // ăn xong
     test(LEFT);           // kiểm tra thử triết gia trái có thể ăn
     test(RIGHT);          // kiểm tra thử triết gia phải có thể ăn
     up(&mutex);           // thoát khỏi miền găng
}
```

# Giải pháp 2 (tt)

```
void test(i)  {
    if ( state[i] == HUNGRY &&
        state[LEFT] != EATING &&
        state[RIGHT] != EATING) {

        state[i] = EATING;

        up(&s[i]);

    }
}
```