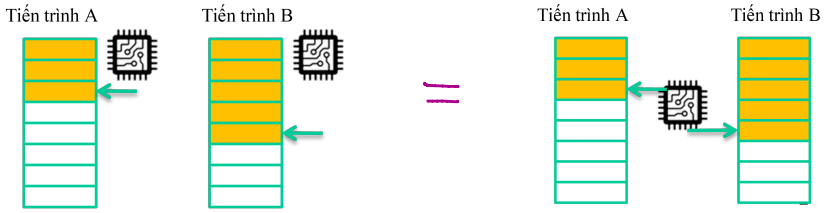
Các bài toán về tương tranh tài nguyên

Saturday, March 14, 2020 8:59 PM

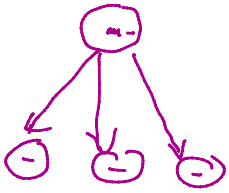
Mô hình tiến trình

Mô hình xử lý tiến trình

Mô hình khái niệm **Mô hình giả lập**



Tiến trình con



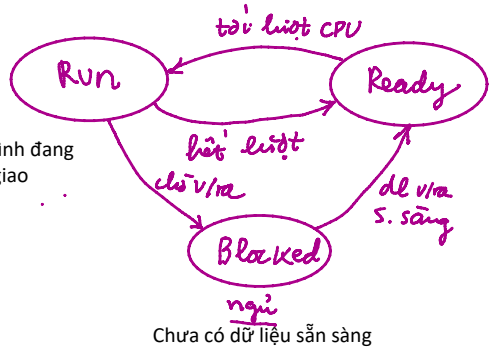
Unix

- 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

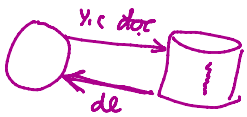
- Tiến trình cha sẽ ngưng cho tới khi tiến trình con kết thúc

Các trạng thái của tiến trình



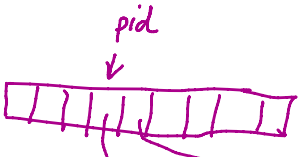
Tiến trình đang được giao CPU

Chưa có dữ liệu sẵn sàng



Tiến trình chờ tới lượt CPU
Dữ liệu đã sẵn sàng

Quản lý các tiến trình

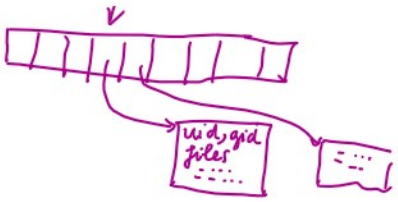


Bảng trạng thái tiến trình

- Thông tin về riêng tiến trình

Bảng trạng thái tiến trình

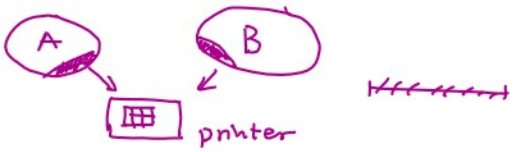
- Thông tin về riêng tiến trình
- Thông tin về bộ nhớ đang sử dụng
- Thông tin về file đang sử dụng



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

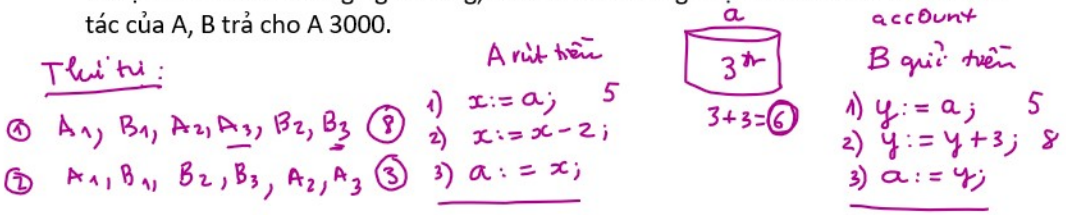
Tình trạng ganh đua



Tình trạng ganh đua xảy ra khi 2 tiến trình cùng xử lý trên một tài nguyên cạnh tranh

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.



Giải pháp luân phiên

Giải pháp

Các tiến trình sẽ luân phiên nhau sử dụng tài nguyên cạnh tranh. Tiến trình nào tới lượt thì sử dụng tài nguyên, sau đó giao lượt cho tiến trình tiếp theo

Mã tiến trình

Biến chung

Int turn; // ==0 lượt của A, ==1 lượt của B

Tiến trình A

```
While (TRUE) {
    While (turn != 0);
    Critical_section(); // cạnh tranh
    Turn = 1; // giao lượt cho B
    Non_critical_section();
}
```

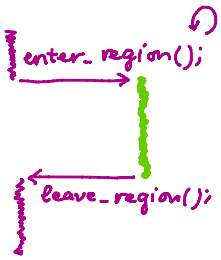
Tiến trình B

```
While (TRUE) {
    While (turn !=1);
    Critical_section();
    Turn = 0; // giao lượt cho A
    Non_critical_section();
}
```

Nhận xét

- Đang giới hạn số tiến trình bằng 2
- Không tách biệt mã tiến trình và đoạn mã xử lý cạnh tranh
- Lượt sử dụng tài nguyên phải là luân phiên. Giao lượt ngay cả khi tiến trình còn lại không có nhu cầu sử dụng
- Lãng phí tài nguyên CPU. Tiến trình chưa có tài nguyên vẫn đòi CPU, để kiểm tra đã tới lượt chưa.

Giải pháp Peterson



Giải pháp

- Các tiến trình muốn vào vùng cạnh tranh cần bày tỏ nguyện vọng sử dụng tài nguyên.
- Tiếp theo tiến trình sẽ gọi hàm **enter_region()** để xin phép vào vùng cạnh tranh
- Nếu vượt qua được hàm enter_region(), tiến trình sẽ chạy đoạn mã cạnh tranh
- Sau khi sử dụng xong tài nguyên, tiến trình cần gọi hàm **leave_region()** để thoát khỏi vùng cạnh tranh, giao lượt cho tiến trình tiếp theo

Mã tiến trình

Khai báo chung
```
#define N 2; // N là số tiến trình
Int turn;  // ==0 tiến trình A, ==1 t. trình B
Int interest[N];  // ==T muốn sd, ==F không
```

Hàm enter_region()
```
Void enter_region(int pid) {
    Int other = 1 - pid;

    Interest[pid] = TRUE;
    Turn = pid;

    While (turn==pid) && (interest[other]==TRUE);
}
```

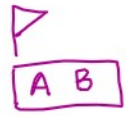
Hàm leave_region()
```
Void leave_region(int pid) {
    Interest[pid] = FALSE;
}
```

Nhận xét

- Đã tách biệt được đoạn mã xử lý cạnh tranh khỏi mã tiến trình
  Mã tiến trình A hoặc B
  ```
  Process(int pid) {
      Enter_region(pid);
      Criticial_section();  // đoạn mã cạnh tranh
      Leave_region(pid);
      Non_critical_section();  // đoạn không cạnh tranh
  }
  ```
- Chi có tiến trình nào có nhu cầu sử dụng tài nguyên thì mới được giao lượt
- Giới hạn số tiến trình bằng 2
- Vẫn lãng phí tài nguyên CPU. Tiến trình chưa có tài nguyên vẫn xếp hàng chờ CPU chỉ để kiểm tra đã tới lượt sử dụng tài nguyên chưa

Giải pháp TSL dùng phần cứng

Một ví dụ về sử dụng biến cờ




```
Int flag;  // ==0 rỗi, ==1 bận

L: if (flag==0) {  // rỗi
       Flag := 1;  // đánh dấu bận
       Critical_section();
       Flag := 0;  // đánh dấu rỗi
   }
   Else goto L;
```

A
1) K.tra
2) Sử dụng

B
1) K.tra
2) Sử dụng

Giải pháp TSL - Test and Set Lock

Cú pháp: TSL register, flag; =0 rỗi =1 bận
Tương đương với 2 lệnh

```
Register := flag;
Flag := 1;
```

Mã tiến trình

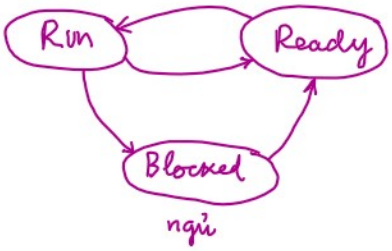
```
Enter_region:
    Tsl register, flag;
    Cmp register, 0;  // compare register với 0
    Jnz enter_region;  // Jump if Not Zero
    Ret

Leave_region:
    Mov flag, 0;   // flag:=0 tài nguyên rỗi
    ret
```

Nhận xét

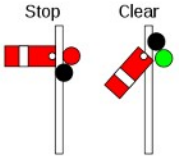
- Đã tách được đoạn mã xử lý cạnh tranh khỏi mã tiến trình
- Chỉ tiến trình nào có nhu cầu sử dụng tài nguyên mới được giao lượt
- Không còn giới hạn số lượng tiến trình
- Vẫn lãng phí tài nguyên CPU. Tiến trình chưa có tài nguyên vẫn xếp hàng chờ CPU chỉ để kiểm tra đã tới lượt dùng tài nguyên chưa

Nhận xét chung về cả 3 phương pháp trên



Khái niệm semaphore

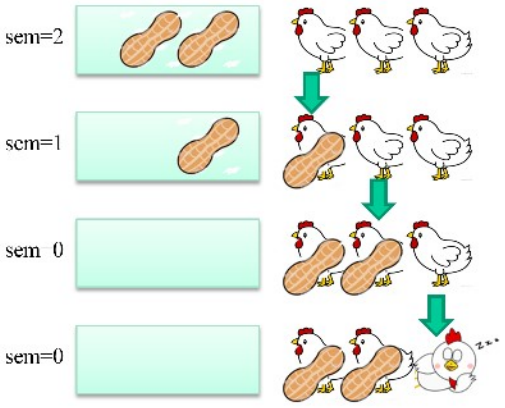
Khái niệm

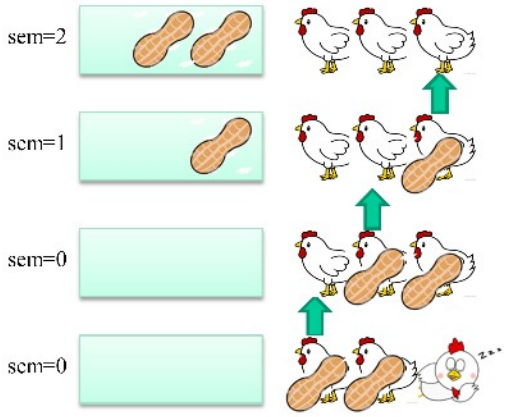


- Đề xuất bởi Edsger Dijkstra năm 1962
- Là biến nguyên không âm
- Chỉ thay đổi giá trị qua 2 thao tác up và down

- **Down(sem)**
  - Nếu sem=0 tiến trình chuyển sang trạng thái blocked, thao tác down được trì hoãn
  - Khác đi, giảm sem 1 đơn vị
- **Up(sem)**
  - Nếu sem=0, đánh thức 1 tiến trình đang ngủ chờ trên sem
  - Tăng sem lên 1 đơn vị

Ví dụ





Bài toán Nhà sản xuất - Người tiêu dùng

Người tiêu dùng
- Đến kho, kiểm tra ngăn hàng
- Nếu có, thì lấy hàng
- Tiêu dùng

Nhà sản xuất
- Sản xuất ra một mặt hàng
- Đến kho, kiểm tra ngăn rỗng
- Nếu có, đưa hàng vào

Một giải pháp

```
#define N 10 // Số ngăn trong kho
int count;     // đếm số ngăn hàng

void producer(void) {
    int item;
    while (TRUE) {
        produce_item(&item);
        if (count==N) sleep(); // kho đầy
        enter_item(item);
        count++;
        if (count==1) wakeup(consumer);
    }
}
```

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

Sử dụng semaphore

Khai báo chung

```
#define N 10  // N là số ngăn trong kho
#define int semaphore
semaphore mutex=1;   // Mutual Exclusion, là semaphore nhị phân 0/1
semaphore empty = N;  // số ngăn rỗng
semaphore full;       // số ngăn có hàng
```

Mã tiến trình người tiêu dùng

```
void consumer(void) {
    int item;

    while (TRUE) {
        down(&full);   // nếu kho không có hàng thì ngủ
        down(&mutex);
        get_item(&item);
        up(&mutex);
        up(&empty);  // tăng ngăn rỗng, đánh thức nsx
        consume_item(item);

    }
}
```

Mã tiến trình nhà sản xuất

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

    while (TRUE) {
        produce_item(&item);
        down(&empty); // k.tra ngăn rỗng, nếu không có thì ngủ
        down(&mutex);
        enter_item(item);
        up(&mutex);
        up(&full);  // tăng ngăn có hàng, đánh thức ntd
    }
}
```

Lưu ý: Không được phép thực hiện thao down(&sem) trong vùng down(&mutex) và up(&mutex) nếu các semaphore đó có liên quan tới cùng một tài nguyên

Bài toán Bữa ăn của các nhà hiền triết

5 nhà hiền triết
- Suy ngẫm
- Đói, lấy đũa 2 bên
- Ăn
- Đặt đũa xuống

Yêu cầu: mô phỏng hoạt động của 5 nhà hiền triết

Một giải pháp

Deadlock (bế tắc)

Mỗi NHT lấy được đũa trái và đều chờ đũa phải.

take_chopsticks(i);

```
#define N  5   // 5 nhà hiền triết
void philosopher(int i) {
      while (TRUE) {
            think();
            take_chopstick(i);
            take_chopstick( (i+1)%N);
            eat();
            put_chopstick(i);
            put_chopstick( (i+1)%N);
      }
}
```

Sử dụng semaphore

```
Khai báo chung
#define N 5  // 5 nhà hiền triết
#define LEFT(i)   (i-1)%N
#define RIGHT(i)  (i+1)%N
#define THINKING  0
#define HUNGRY    1
#define EATING    2
#typedef int  semaphore

int state[N];        // biểu diễn trạng thái của mỗi nhà hiền triết
```

```
semaphore s[N];  // s[i] == 1 hai đũa của NHT i đã sẵn sàng để lấy, ==0 chưa sẵn sàng
semaphore mutex=1;   // Mutual Exclusion
```

Tiến trình philosopher

```
void philosopher(int i) {
    while (TRUE) {
        think();
        take_chopsticks(i);   // lấy cả 2 đũa cùng lúc, hoặc chờ
        eat();
        put_chopsticks(i);    // đặt cả 2 đũa xuống
    }
}
```

down(&mutex) ≡ freeze
up(&mutex) ≡ free

mutex : bãn

Hàm take_chopsticks

```
void take_chopsticks(int i) {
    down(&mutex);
    state[i] = HUNGRY;
    test(i);   // Kiểm tra 2 đũa có sẵn sàng không. Nếu có thì dựng semaphore s[i]
    up(&mutex);
    down(&s[i]);  // lấy 2 đũa nếu s[i] bật, ngủ chờ nếu 2 đũa chưa sẵn sàng
}
```

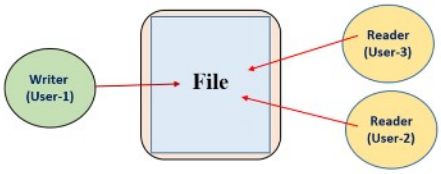
Hàm test

```
void test(int i) {
    if ( state[i] == HUNGRY &&
        state[LEFT(i)] != EATING &&  // còn đũa trái
        state[RIGHT(i)] != EATING)  {  // đũa phải còn
          state[i] =  EATING;
          up(&s[i]);   // dựng semaphore s[i] nếu 2 đũa sẵn sàng
    }
}
```

Hàm drop_chopsticks

```
void put_chopsticks(int) {
    down(&mutex);
    state[i] = THINKING;
    test(LEFT(i));  // đánh thức NHT trái nếu họ đang ngủ chờ
    test(RIGHT(i)); // đánh thức NHT phải nếu đang ngủ chờ
    up(&mutex);
}
```

Bài toán Đọc và ghi



	R	W
R	✓	✗
W	✗	✗

Tiến trình đọc:

- Nếu là tiến trình đầu tiên, phải lấy quyền Ghi
- Nếu là tiến trình cuối cùng, trả lại quyền ghi
- Nếu là tiến trình đọc tiếp theo, thao tác với tài nguyên

Tiến trình ghi:
- Giành quyền ghi đối với tệp
- Ghi ra tệp
- Trả lại quyền ghi

Khai báo chung

```
semaphore mutex=1;  // giải quyết cạnh tranh
semaphore db = 1;  // giữ quyền ghi đối với tệp
int rc=0;          // reader counter
```

Tiến trình đọc

```
void reader(void) {
    while (TRUE) {
        down(&mutex);
        rc++;     // tăng số đếm t.tr đọc
        if (rc==1) down(&db);   // giành quyền ghi
        up(&mutex);
        read_data();
        down(&mutex);
        rc--;     // giảm số đếm t.tr đọc
        if (rc==0) up(&db);  // trả lại quyền ghi
        up(&mutex);
        use_data();
    }
}
```

Tiến trình ghi

```
void writer(void) {
    while (TRUE) {
        thinkup_data(); // chuẩn bị dl
        down(&db);      // giành quyền ghi
        write_data();
        up(&db);        // trả lại quyền ghi
    }
}
```

Câu hỏi: chúng ta thực hiện down(&db) trong cặp down(&mutex) .... up(&mutex), liệu có dẫn tới sự cố ngủ hàng loạt?
Trả lời: không, vì semaphore mutex gác cho tài nguyên rc, semaphore db gác cho tài nguyên tệp. Hai semaphore này độc lập với nhau.

Bài toán Cửa hàng cắt tóc

sleeping barbers

Cửa hàng có n ghế chờ
Thợ cắt tóc:
- Gọi khách chờ vào cắt tóc, nếu chưa có thì ngủ
- Nếu khách đang ngủ chờ thì đánh thức dậy
- Cắt tóc
- Lặp lại bước đầu

Khách hàng
- Kiểm tra nếu không còn ghế chờ thì bỏ đi
- Vào ngồi chờ, gọi thợ nếu chưa được ngủ chờ
- Được cắt tóc
- Đi về

Sử dụng semaphore

Khai báo chung

```
#define CHAIRS 5  // có 5 ghế chờ

semaphore customers = 0;   // số khách chờ trên ghế
semaphore barbers = 0;     // số thợ cắt tóc sẵn sàng
semaphore mutex = 1;       // mutual exclusion
int waiting = 0;           // số khách hàng chờ trên ghế
```

Tiến trình thợ cắt tóc

```
void barber(void) {
    while (TRUE) {
        down(&customers);  // ngủ nếu chưa có khách
        down(&mutex);      // lấy quyền truy cập wating
        waiting--;
        up(&barbers);
        up(&mutex)
        cut_hair();
    }
}
```

Tiến trình khách hàng

```
void customer(void) {
    down(&mutex);
    if (waiting < CHAIRS) {  // còn ghế
        waiting++;
        up(&customers);  // đánh thức thợ
        up(&mutex);
        down(&barbers);  // ngủ chờ nếu chưa có thợ
        get_hair_cut();
    }
    else up(&mutex);       // bỏ đi
}
```