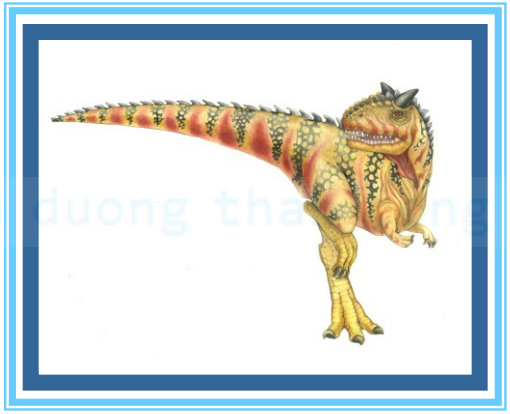
Chương 5: Đồng bộ - 2



UIT
TRƯỜNG ĐẠI HỌC
CÔNG NGHỆ THÔNG TIN

Câu hỏi ôn tập 5 - 1

- Khi nào thì xảy ra tranh chấp race condition?
- Vấn đề Critical Section là gì?
- Yêu cầu của lời giải cho CS problem?
- Có mấy loại giải pháp? Kể tên?

UIT
TRƯỜNG ĐẠI HỌC
CÔNG NGHỆ THÔNG TIN

Mục tiêu

- Hiểu được nhóm giải pháp Busy waiting bao gồm:
  - Các giải pháp phần mềm
  - Các giải pháp phần cứng

Nội dung

- Các giải pháp phần mềm
  - Sử dụng giải thuật kiểm tra luân phiên
  - Sử dụng các biến cờ hiệu
  - Giải pháp của Peterson
  - Giải pháp Bakery
- Các giải pháp phần cứng
  - ~~Cấp ngắt~~
  - ~~Chỉ thị TSL~~
  - Cấm ngắt
  - Các lệnh đặc biệt

UIT
TRƯỜNG ĐẠI HỌC
CÔNG NGHỆ THÔNG TIN

Giải thuật 1

- Biến chia sẻ
  - int turn; /* khởi đầu turn = 0 */
  - nếu turn = i thì Pi được phép vào critical section, với i = 0 hay 1
- Process Pi

```
do {
        while (turn != i);
                critical section
        turn = j;
                remainder section
} while (1);
```


- Thoả mãn mutual exclusion (1)
- Nhưng không thoả mãn yêu cầu về progress (2) và bounded waiting (3)

UIT
TRƯỜNG ĐẠI HỌC
CÔNG NGHỆ THÔNG TIN

Giải thuật 1 (tt)

```
Process P0:
do
  while (turn != 0);
        critical section
  turn := 1;
        remainder section
while (1);
```

```
Process P1:
do
  while (turn != 1);
        critical section
  turn := 0;
        remainder section
while (1);
```

- Ví dụ:

P0 có RS (remainder section) rất lớn còn P1 có RS nhỏ ???

Giải thuật 2

- Biến chia sẻ
  - boolean flag[2]; /* khởi đầu flag[0] = flag[1] = false */
  - Nếu flag[i] = true thì Pi "sẵn sàng" vào critical section.
- Process Pi

```
do {
        flag[ i ] = true;      /* Pi "sẵn sàng" vào CS */
        while ( flag[ j ] );   /* Pi "nhường" Pj       */
                critical section
        flag[ i ] = false;
                remainder section
} while(1);
```

- Bảo đảm được mutual exclusion.
- Không thỏa mãn progress. Vì sao?

Nếu flag[i] = flag[j] = true, cả 2 process đều lặp vòng vô tận tại while, không ai vào được CS

Giải thuật 3 (Peterson)

- Biến chia sẻ
  - Kết hợp cả giải thuật 1 và 2.
- Process Pi, với i = 0 hay 1

```
do {
        flag[ i ] = true;       /*  Process i sẵn sàng  */
        turn = j;               /*  Nhường process j    */
        while (flag[ j ] and turn == j);

                critical section

        flag[ i ] = false;

                remainder section

} while (1);
```

- Thoả mãn được cả 3 yêu cầu.

⇒ giải quyết bài toán critical section cho 2 process

Giải thuật Peterson – 2 process

UIT
TRƯỜNG ĐẠI HỌC
CÔNG NGHỆ THÔNG TIN

Process P_0

```
do {
    /*  0 wants in  */
    flag[0] = true;
    /*  0 gives a chance to 1  */
    turn = 1;
    while (flag[1] &&turn == 1);
        critical section
    /*  0 no longer wants in  */
    flag[0] = false;
        remainder section
} while(1);
```

Process P_1

```
do {
    /*  1 wants in  */
    flag[1] = true;
    /*  1 gives a chance to 0  */
    turn = 0;
    while (flag[0] && turn == 0);
        critical section
    /*  1 no longer wants in  */
    flag[1] = false;
        remainder section
} while(1);
```

UIT
TRƯỜNG ĐẠI HỌC
CÔNG NGHỆ THÔNG TIN

Giải thuật 3: Tính đúng đắn

- Giải thuật 3 thỏa *mutual exclusion*, *progress*, và *bounded waiting*
- Mutual exclusion được đảm bảo bởi vì
  - P0 và P1 đều ở trong CS nếu và chỉ nếu flag[0] = flag[1] = true và turn = i cho mỗi Pi (không thể xảy ra)
- Chứng minh thỏa yêu cầu về progress và bounded waiting
  - Pi không thể vào CS nếu và chỉ nếu bị kẹt tại vòng lặp while() với điều kiện flag[j] =true và turn = j
  - Nếu Pj không muốn vào CS thì flag[j] = false và do đó Pi có thể vào CS

UIT
TRƯỜNG ĐẠI HỌC
CÔNG NGHỆ THÔNG TIN

Giải thuật 3: Tính đúng đắn (tt)

- Nếu Pj đã bật flag[j] = true và đang chờ tại while() thì có chỉ hai trường hợp là turn = i hoặc turn = j
- Nếu turn = i và Pi vào CS. Nếu turn = j thì Pj vào CS nhưng sẽ bật flag[j] = false khi thoát ra → cho phép Pi và CS
- Nhưng nếu Pj có đủ thời gian bật flag[j] = true thì Pj cũng phải gán turn = i
- Vì Pi không thay đổi trị của biến turn khi đang kẹt trong vòng lặp while(), Pi sẽ chờ để vào CS nhiều nhất là sau một lần Pj vào CS (bounded waiting)

UIT
TRƯỜNG ĐẠI HỌC
CÔNG NGHỆ THÔNG TIN

Giải thuật bakery: n process

- Trước khi vào CS, process Pi nhận một con số. Process nào giữ con số nhỏ nhất thì được vào CS
- Trường hợp Pi và Pj cùng nhận được một chỉ số:
  - Nếu i < j thì Pi được vào trước
- Khi ra khỏi CS, Pi đặt lại số của mình bằng 0
- Cơ chế cấp số cho các process thường tạo các số theo cơ chế tăng dần, ví dụ 1, 2, 3, 3, 3, 3, 4, 5,…
- Kí hiệu
  - (a,b) < (c,d) nếu a < c hoặc nếu a = c và b < d
  - max(a0,…,ak) là con số b sao cho b ≥ ai với mọi i = 0,…, k

Giải thuật bakery: n process (tt)

```
/*  shared variable  */
boolean     choosing[n];   /*  initially, choosing[i] = false  */
int         num[n];        /*  initially, num[i] = 0           */

do {
        choosing[i] = true;
        num[i]      = max(num[0], num[1],…, num[n-1]) + 1;
        choosing[i] = false;
        for (j = 0; j < n; j++) {
            while (choosing[j]);
            while ((num[j] != 0) && (num[j], j) < (num[i], i));
        }
            critical section
        num[i] = 0;
            remainder section
} while(1);
```

UIT
TRƯỜNG ĐẠI HỌC
CÔNG NGHỆ THÔNG TIN

Từ software đến hardware

- Khuyết điểm của các giải pháp software:
  - Các process khi yêu cầu được vào vùng tranh chấp đều phải liên tục kiểm tra điều kiện (busy waiting), tốn nhiều thời gian xử lý của CPU
  - Nếu thời gian xử lý trong vùng tranh chấp lớn, một giải pháp hiệu quả nên có cơ chế block các process cần đợi.
- Các giải pháp phần cứng:
  - Cấm ngắt (disable interrupts)
  - Dùng các lệnh đặc biệt

UIT
TRƯỜNG ĐẠI HỌC
CÔNG NGHỆ THÔNG TIN

Cấm ngắt

- Trong hệ thống uniprocessor: mutual exclusion được đảm bảo
- Trong hệ thống multiprocessor: mutual exclusion không được đảm bảo
  - Chỉ cấm ngắt tại CPU thực thi lệnh disable_interrupts, các CPU khác vẫn có thể truy cập bộ nhớ chia sẻ
  - Nếu cấm ngắt tất cả các CPU thì có thể gây tốn thời gian mỗi lần vào vùng tranh chấp

```
Process Pi:

do {
        disable_interrupts();
        critical section
        enable_interrupts();
        remainder section
} while (1);
```


UIT
TRƯỜNG ĐẠI HỌC
CÔNG NGHỆ THÔNG TIN

Lệnh TestAndSet

- Đọc và ghi một biến trong một thao tác atomic (không chia cắt được)

```
boolean TestAndSet( boolean *target){
    boolean rv = *target;
    *target = true;
    return rv;
}
```

- Shared data:
  boolean lock = false;
- Process P_i:

```
do {
    while (TestAndSet(&lock));
        critical section
    lock = false;
        remainder section
} while (1);
```


UIT
TRƯỜNG ĐẠI HỌC
CÔNG NGHỆ THÔNG TIN

Lệnh TestAndSet (tt)

- Mutual exclusion được bảo đảm: nếu Pi vào CS, các process Pj khác đều đang busy waiting
- Khi Pi ra khỏi CS, quá trình chọn lựa process Pj vào CS kế tiếp là tùy ý ⇒ không bảo đảm điều kiện bounded waiting. Do đó có thể xảy ra starvation (bị bỏ đói)
- Các processor (ví dụ Pentium) thông thường cung cấp một lệnh đơn là **Swap(a, b)** có tác dụng hoán chuyển nội dung của a và b.
  - Swap(a, b) cũng có ưu nhược điểm như TestAndSet

Swap và mutual exclusion

- Biến chia sẻ **lock** được khởi tạo giá trị false
- Mỗi process P_i có biến cục bộ **key**
- Process P_i nào thấy giá trị **lock = false** thì được vào CS.
  - Process P_i sẽ loại trừ các process P_j khác khi thiết lập **lock = true**

```
void Swap(boolean *a,
          boolean *b) {
   boolean temp = *a;
   *a = *b;
   *b = temp;
}
```

- Biến chia sẻ (khởi tạo là **false**)

  bool lock;

  ~~**bool key;**~~

- Process P_i

```
do {
    key = true;
    while (key == true)
        Swap(&lock, &key);
    critical section
    lock = false;
    remainder section
} while (1)
```

Không thỏa mãn bounded waiting

UIT
TRƯỜNG ĐẠI HỌC
CÔNG NGHỆ THÔNG TIN

Giải thuật dùng TestAndSet thoả mãn 3 yêu cầu

- Cấu trúc dữ liệu dùng chung (khởi tạo là false)

  bool waiting[n];

  bool lock;

- Mutual exclusion: Pi chỉ có thể vào CS nếu và chỉ nếu hoặc waiting[i] = false, hoặc key = false
  - key = false chỉ khi TestAndSet (hay Swap) được thực thi
    - Process đầu tiên thực thi TestAndSet mới có key == false; các process khác đều phải đợi
  - waiting[i] = false chỉ khi process khác rời khỏi CS
    - Chỉ có một waiting[i] có giá trị false
- Progress: chứng minh tương tự như mutual exclusion
- Bounded waiting: waiting in the cyclic order

(Xem sách tham khảo trang 212)

Giải thuật dùng TestAndSet thoả mãn 3 yêu cầu (tt)

UIT
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

```
do {
        waiting[ i ] = true;
        key = true;
        while (waiting[ i ] && key)
            key = TestAndSet(&lock);
        waiting[ i ] = false;
            critical section
        j = (i + 1) % n;
        while ( (j != i)  &&  !waiting[ j ] )
            j = (j + 1) % n;
        if (j == i)
            lock = false;
        else
            waiting[ j ] = false;
            remainder section
} while (1)
```

Đồng bộ

Nội dung

- Các giải pháp phần mềm
  - Sử dụng giải thuật kiểm tra luân phiên
  - Sử dụng các biến cờ hiệu
  - Giải pháp của Peterson
  - Giải pháp Bakery
- Các giải pháp phần cứng
  - ~~Cấp ngắt~~
  - ~~Chỉ thị TSL~~
  - Cấm ngắt
  - Các lệnh đặc biệt