MÔN: HỆ ĐIỀU HÀNH
Bài thực hành số 6.1: Semaphore

I. Mục tiêu

- Giúp SV củng cố các kiến thức về semaphore: tính chất và công dụng thông qua 1 số thí dụ cụ thể.

II. Nội dung

- Cố gắng trả lời các câu hỏi đề ra về semaphore.

III. Chuẩn đầu ra

- Sinh viên nắm vững và sử dụng thành thạo semaphore để giải quyết loại trừ tương hỗ giữa các process khi chúng cùng truy xuất vào tài nguyên dùng chung đồng thời, cũng như để đồng bộ hóa việc chạy các process.

IV. Qui trình

1. Giới thiệu

Tài nguyên dùng chung là tài nguyên mà nhiều process đồng thời (hay thread) có thể truy xuất được. Nếu để các process truy xuất đồng thời tự do tài nguyên thì dễ làm hư hỏng tài nguyên, do đó cần phải có cơ chế kiểm soát việc truy xuất đồng thời này.

Hiện nay ta dùng cơ chế loại trừ tương hỗ giữa các process đồng thời, nghĩa là tối đa 1 process được phép truy xuất tài nguyên tài từng thời điểm.

Ta gọi "**critical section**" **- CS** là đoạn code của process mà truy xuất tài nguyên dùng chung, do đó nó cần được chạy theo tính nguyên tử (không thể chia cắt, không thể thấy từ ngoài, không thể dừng đột ngột,...)

Tính nguyên tử còn có nghĩa là 2 hay nhiều đoạn CS của nhiều process cùng truy xuất tài nguyên xác định sẽ không bao giờ được chạy đồng thời mà phải là tuần tự.

Loại trừ tương hỗ chỉ là 1 trường hợp đặc biệt của việc đồng bộ hóa các process. Đồng bộ là hoạt động kiểm soát việc chạy các process sao cho:

- thiết lập được 1 thứ tự xảy ra của 1 số hoạt động (gởi/nhận).
- tuân thủ 1 điều kiện (cuộc hẹn, loại trừ tương hỗ,...)

Chờ chủ động là cơ chế dùng CPU chạy lặp lại 1 đoạn code kiểm soát tới lúc mà điều kiện xác định xảy ra, lúc này process sẽ tiếp tục công việc theo thuật giải gốc của mình.

Semaphore là đối tượng cho phép loại trừ tương hỗ trong việc truy xuất 1 tài nguyên xác định mà tránh được việc dừng chờ chủ động. Về góc nhìn người dùng, semaphore có 1 counter chỉ chứa số nguyên dương và 2 tác vụ down() và up() trên biến counter này. Về mặt hiện thực, một trong nhiều cách hiện thực như sau:

- mỗi semaphore có 1 biến counter thuộc kiểu số nguyên, nếu là số dương nó miêu tả số tài nguyên còn rảnh, nếu là số âm nó miêu tả số process đang chờ truy xuất tài nguyên.
- và 1 hàng đợi chứa các process đang chờ thực hiện tác vụ down(), mỗi HĐH có chính sách quản lý hàng đợi riêng.
- lúc tạo semaphore, hàng đợi của nó ở trạng thái trống.

Semaphore được quản lý bởi các tác vụ có cơ chế thực thi nguyên tử, không chia cắt như sau:

- SEM cs = new SEM(cnt): tạo 1 semaphore với counter ban đầu là cnt.
- cs.down(): yêu cầu truy xuất tài nguyên tương ứng. Nếu không còn tài nguyên thì process này sẽ bị giam vào hàng đợi (Blocked).
- cs.up(): giải phóng tài nguyên đang dùng, nếu hàng đợi không trống, chọn 1 process và cho nó chạy tiếp.
- delete(cs): xóa semaphore, nếu hàng đợi của nó không trống thì cần xử lý lỗi.

2. Hãy hiện thực 2 tác vụ down() và up() bằng cách dùng các tác vụ sau:

- TASK current: miêu tả process hiện hành (Running).
- void insert(TASK t, List<TASK> lstBlocked): thêm process vào hàng đợi
- cs.up(): giải phóng tài nguyên đang dùng, nếu hàng đợi không trống, chọn 1 process và cho nó chạy tiếp.
- TASK remove(List<TASK> lstBlocked): lấy 1 process trong hàng đợi ra (và xóa nó trong hàng đợi).

```
class SEM {
    int cnt;
    List<TASK> lstBlocked = new List<TASK>();
    public SEM(int cnt) { this.cnt = cnt; }
    //tác vụ down
    public void down() {
        cnt--;                          //a
        if (cnt < 0) {                  //b
            current.state = BLOCKED;    //c
            insert(current,cs.lstBlocked);//d
            commut();                   //chuyển ngữ cảnh cho thành phần khác
    chạy
        }
    }
    //tác vụ up
    public void up() {
        TASK t;
        cnt++;                      //x
        if (cnt <= 0) {             //y
            t = Remove(lstBlocked);//z
            t.state = READY;        //t
        }
    }
}
```

3. Hãy giải thích tại sao các tác vụ down và up phải có tính nguyên tử trong khi thực thi và ta hiện thực tính nguyên tử này như thế nào?

Giả sử giá trị ban đầu của cnt là 1. Nếu 2 process thi hành chuỗi lệnh a1 a2 b2 b1, cả 2 process đều bị blocked nhưng biến cnt vẫn là -1, do đó sau này, 1 trong 2 process sẽ không bao giờ được giải phóng. Tệ hơn nửa, nếu process lệnh up cũng là process lệnh down trước đó, thì sẽ có deadlock vì không ai thực thi tác vụ V nữa cả.

Để làm tác vụ down và up có tính nguyên tử, ta có thể cấm ngắt quảng trong lúc thi hành chúng. Điều này chỉ có thể được nếu đoạn code của tác vụ down và up nằm trong HĐH.

4. Tại sao ta không thể dùng cơ chế cấm ngắt trong đoạn code có quyền ưu tiên thấp của ứng dụng?

Vì rất nguy hiểm. Nếu process cấm ngắt, nó sẽ chiếm CPU chạy cho đến khi bỏ cấm ngắt. Thường process có xu hướng độc chiếm CPU, nếu nó không bỏ cấm ngắt, HĐH và các process khác sẽ không bao giờ có cơ hội chiếm CPU nữa.

5. Process đang thực thi đoạn code CS có thể bị ngắt tạm bởi 1 process khác không? Giải thích việc gì sẽ xảy ra.

Được, khi process được chọn để chạy trong khe thời gian kế tiếp không truy xuất tài nguyên dùng chung, có có thể được thực thi bình thường. Còn nếu nó truy xuất tài nguyên dùng chung, nó sẽ bị blocked ngay khi thực hiện lệnh kiểm soát In_Control() và trình lập lịch sẽ chuyển ngữ cảnh cho chọn thành phần khác chạy.

Nếu ta dùng kỹ thuật cấm ngắt để kiểm soát việc đi vào đoạn CS, các process không truy xuất tài nguyên dùng chung sẽ bị hạn chế chạy vì chúng phải đợi đến khi đoạn CS được thực thi xong (bởi process đang chiếm và cấm ngắt CPU) để có thể chạy tiếp được.

Ta xét 3 process chạy đồng thời sau đây (semaphore m đã được thiết lập giá trị đầu là 1 - ta gọi nó là semaphore nhị phân):

Process A	*Process B*	*Process C*
(a) m.down();	(d) m.down();	(g) m.down();
(b) x = x + 1;	(e) x = x * 2;	(h) x = x - 4;
(c) m.up();	(f) m.up();	(i) m.up();

6. Hãy chú ý kịch bản chạy sau: a d b g c e f h i.

Sau mỗi hoạt động trên semaphore m, hãy miêu tả các thông tin sau:

- giá trị của thuộc tính cnt của semaphore.
- nội dung hàng đợi semaphore.
- trạng thái của mỗi process (running, ready, blocked).

Lưu ý rằng ta muốn bỏ qua các hiệu chỉnh trạng thái của process do trình lập lịch để đừng nhầm lẫn với các hiệu chỉnh do hoạt động trên semaphore gây ra.

Tác vụ	**cnt**	**Trạng thái hàng đợi**	**Trạng thái các process**
a	0		A:running; B,C : ready
d	-1	B	B:running->blocked; A,C : ready
g	-2	B, C	C:running->blocked; A : ready; B:blocked
c	-1	C	A : running; B:blocked->ready; C:blocked
f	0		B:running; C:blocked->ready
i	1		C:running

Kịch bản thứ 2 là không thể: khi B thực thi (d), nó bị blocked và chỉ khi A thực thi (c) mới có thể giúp B chạy tiếp. Do đó nó không thể thực thi (e) ngay được.

Ta xét 3 process chạy đồng thời trên 1 máy, biến a là biến dùng chung và ta đã thiết lập nó như sau:

```
int a = 6;
SEM m1 = new SEM(1);
SEM m2 = new SEM(0);
```

Process A	*Process B*	*Process C*
m1.down();	a = a - 5;	m1.up();
a = a + 7;		m2.down();
m2.up();		a = a * 3;

7. Biến a có thể nhận những giá trị nào sau khi chạy 3 process? Ứng với mỗi giá trị có thể của a, hãy miêu tả trình tự thi hành các lệnh tạo ra kết quả tương ứng.

ABC -> 24

ACB -> 34

BAC -> 24

Lưu ý semaphore m1 hoàn toàn không có tác dụng gì trong trường hợp chạy trên.

8.

- Hãy thêm các semaphore cần thiết vào các đoạn lệnh của 3 process trên để sau khi chạy xong 3 process, biến a luôn = 34. Miêu tả phần thiết lập giá trị đầu cho các semaphore được thêm mới.
- cùng câu hỏi trên để giá trị a = 24.

Chỉ 1 kịch bản chạy sau có thể tạo ra a = 34: ta tạo thêm semaphore m3 với giá trị đầu = 0 để kiểm soát B được chạy chỉ sau C hoàn thành:

Process A	**Process B**	**Process C**
m1.down();	m3.down();	m1.up();
a = a + 7;	a = a - 5;	m2.down();
m2.up();		a = a * 3;
		m3.up();

Có 2 kịch bản chạy sau có thể tạo ra a = 24 : ta tạo thêm semaphore m3 với giá trị đầu = 0 để kiểm soát C chạy sau cùng :

Process A	**Process B**	**Process C**
m1.down();	a = a - 5;	m1.up();
a = a + 7;	m3.up();	m2.down();
m2.up();		m3.down();
		a = a * 3;

9. Semaphore nào có thể được loại bỏ mà không cần hiệu chỉnh việc thi hành các process? Giải thích.

Semaphore m1 không bao giờ kẹt. Do đó nó không cần thiết.

10. Ta hiệu chỉnh sự đồng bộ hóa giữa các process như sau :

Process A	**Process B**	**Process C**
m1.down();	m2.down();	m2.down();
m2.down();	a = a - 5;	m1.down();
a = a + 7;	m2.up();	a = a * 3;
m1.up();	m1.up();	m2.up();
m2.up();		

Semaphore m1 và m2 đều được thiết lập giá trị đầu là 1. Kết quả của việc hiệu chỉnh này là gì? Giải thích.

Một số trình tự chạy có thể dẫn đến deadlock. Chẳng hạn:

A:m1.down() C: m2.down() B: m2.down() -> blocked A: m2.down()->blocked C: m1.down() -> blocked.