**Bài tập về các vấn đề đồng bộ nhiều tiến trình đồng thời**

**Bài 1:** Nêu vai trò của biến turn trong thuật toán Peterson cho bài toán đồng bộ 2 tiến trình? Nếu bỏ lệnh turn đi thì điều gì sẽ xảy ra?

**Bài làm**

* Biến turn trong thuật toán Peterson có vai trò xác định lượt tiến trình nào được vào đoạn nguy hiểm.
* Khi bỏ biến turn trong thuật toán, các tiến trình sẽ rơi vào vòng lặp vô hạn.

**Bài 2:**

Sử dụng thuật toán Test\_and\_Set để thực hiện loại trừ tương hỗ cho bài toán triết gia ăn cơm

Phân tích rõ ràng giải pháp sử dụng lệnh Test\_and\_Set ở trên có thể gây bế tắc hoặc đói không?

**Bài làm**

* Test and Set cho bài toán triết gia ăn cơm

Bool chopstick[5] = {false, false, false, false, false}

Void philosopher(int i){

For(;;){

While(test\_and\_set(chopstick[i]));

While(test\_and\_set(chopstick[(i+1)%5]);

<ăn cơm>

chopstick[i] = false;

chopstick[(i+1)%5] = false;

<suy nghĩ>

}

}

Void main(){

StartProcess(philosopher(1));

StartProcess(philosopher(2));

...

StartProcess(philosopher(5));

}

* Nhược điểm:

+ Sử dụng lệnh Test\_and\_Set ở trên có thể gây bế tắc khi mà cả 5 người đều cầm đũa bên trái, không ai chịu nhường đũa => chờ đợi vòng tròn.

+ sử dụng lệnh Test\_and\_Set ở trên có thể gây đói khi 1 triết gia thực hiện việc lấy đũa chậm hơn các triết gia khác => chưa kịp lấy thì bị lấy mất => đói tài nguyên.

**Bài 3:**

Sử dụng lệnh Test\_and\_Set để thực hiện loại trừ tương hỗ cho bài toán triết gia ăn cơm?

Phân tích rõ ràng giải pháp sử dụng lệnh Test\_and\_Set ở trên có thể gây bế tắc hoặc đói không?

**Bài làm**

* Test and Set cho bài toán triết gia ăn cơm

Bool chopstick[5] = {false, false, false, false, false}

Void philosopher(int i){

For(;;){

While(test\_and\_set(chopstick[i]));

While(test\_and\_set(chopstick[(i+1)%5]);

<ăn cơm>

chopstick[i] = false;

chopstick[(i+1)%5] = false;

<suy nghĩ>

}

}

Void main(){

StartProcess(philosopher(1));

StartProcess(philosopher(2));

...

StartProcess(philosopher(5));

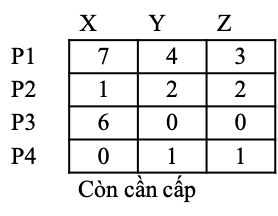
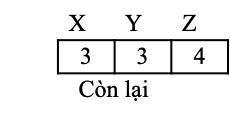
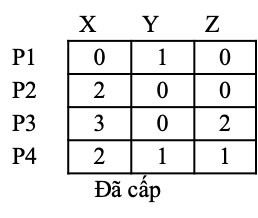
}

* Nhược điểm:

+ Sử dụng lệnh Test\_and\_Set ở trên có thể gây bế tắc khi mà cả 5 người đều cầm đũa bên trái, không ai chịu nhường đũa => chờ đợi vòng tròn.

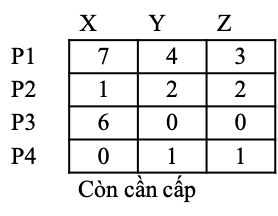
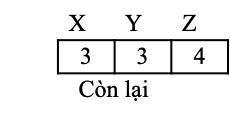
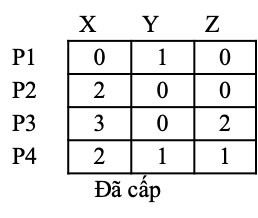
+ sử dụng lệnh Test\_and\_Set ở trên có thể gây đói khi 1 triết gia thực hiện việc lấy đũa chậm hơn các triết gia khác => chưa kịp lấy thì bị lấy mất => đói tài nguyên.

**Bài 4:** Xét trạng thái cấp phát tài nguyên của hệ thống như sau:



P2 yêu cầu cấp phát 1 tài nguyên Y, 2 tài nguyên Z. Sử dụng thuật toán người cho vay để xác định xem yêu cầu của P2 có được đáp ứng hay không?

**Bài làm**



P2 yêu cầu cấp phát 1 tài nguyên Y, 2 tài nguyên Z (0,1,2)

* Tài nguyên còn lại:

|  |  |  |
| --- | --- | --- |
| X | Y | Z |
| 3 | 2 | 2 |

* Yêu cầu trên được đáp ứng vì đây vẫn ở trạng thái an toàn, ta vẫn có cách để cấp phát không dẫn tới bế tắc.

VD: Cấp phát theo thứ tự: P2 🡪 P4 🡪 P3 🡪 P1.

|  |  |  |
| --- | --- | --- |
| X | Y | Z |
| 5 | 2 | 2 |

|  |  |  |
| --- | --- | --- |
| X | Y | Z |
| 7 | 3 | 3 |

|  |  |  |
| --- | --- | --- |
| X | Y | Z |
| 10 | 3 | 7 |

|  |  |  |
| --- | --- | --- |
| X | Y | Z |
| 10 | 5 | 5 |

**Bài 5:** Trình bày cách thực hiện thuật toán Peterson cho bài toán đồng bộ 3 tiến trình?

**Bài làm**

bool flag[3] = {-1};

int turn[2] = {-1};

bool check(int m){

for(int k = 0; k < 3; k++){

if(flag[k] >= m) return true;

}

Return false;

}

Void P0() {  
 for(;;) { //lặp vô hạn  
 for(int m = 0; m< 2;m++){

flag[0] = m;  
 turn[m] = 0;

while(turn[m] == 0 && check(m)) ;//lặp đến khi điều kiện không thỏa  
 <đoạn nguy hiểm>

}

flag[0] = -1;  
 <phần còn lại của tiến trình>  
 }  
}

Void P1() {  
for(;;) { //lặp vô hạn  
 for(int m = 0; m< 2;m++){

flag[1] = m;  
 turn[m] = 1;

while(turn[m] == 1 && check(m)) ;//lặp đến khi điều kiện không thỏa  
 <đoạn nguy hiểm>

}

flag[1] = -1;  
 <phần còn lại của tiến trình>  
 }  
}

Void P2() {  
for(;;) { //lặp vô hạn  
 for(int m = 0; m< 2;m++){

flag[2] = m;  
 turn[m] = 2;

while(turn[m] == 2 && check(m)) ;//lặp đến khi điều kiện không thỏa  
 <đoạn nguy hiểm>

}

flag[2] = -1;  
 <phần còn lại của tiến trình>  
 }  
}

Void main() {  
 flag[0] = flag[1] = false; turn =0;  
 //tắt tiến trình chính, chạy đồng thời 3 tiến trình P0, P1 và P2  
 **startProcess (P0);**

**startProcess(P1);**

**startProcess(P2);**}