Họ và tên: Lê Quang Nam

MSSV: 1712603

**BÀI TẬP LÝ THUYẾT**

Bài tập 14: Bài toán cây cầu cũ

* WaitToGo[2] : Xe chờ để lên cầu với hai điều kiện là Hướng hiện tại hoặc hướng chỉ định
* Int cars : Số lượng xe ở trên cầu : Mặc định là 0
* Curentdirection: Hướng hiện tại của chiếc xe bất kỳ
* Waiter[2]: Số lượng xe đang chờ theo mỗi hướng , có 2 hướng

**Car(int direction)**

{

**RuntoBridge();**

**ArriveBridge(int direction)**

{  
  lock.acquire();

// Khi không thể lên được cầu thì đợi   
   while ((cars == 3) ||  
         (cars > 0 && currentdirection != direction)) // Số lượng xe là 3 hoặc số xe nhiều hơn 0 và hướng hiện tại khác với hướng chỉ định

{   
    waiters[direction]++; // Số lượng xe chờ theo hướng chỉ định tăng lên  
    waitingToGo[direction].wait(); // đợi lên cầu theo hướng chỉ định  
    waiters[direction]--; // Số lượng xe chờ theo hướng chỉ định giảm xuống  
  }  
  
  // Lên cầu  
  cars++;  
  currentdirection = direction;   
  
  lock.release(); // Giải phóng   
}

**PassBridge();  
ExitBridge()**

{  
  lock.acquire();  
  // Xuống cầu  
  cars--;  
  // Nếu xe nào muốn đi theo hướng hiện tại thì wakeup   
  if (waiters[currentdirection] > 0)  
    waitingToGo[currentdirection].signal();  
  //Nếu không có ai thì wakeup xe khác đi theo đường khác

  else if (cars == 0)  
    waitingToGo[1-currentdirection].broadcast();  
  
  lock.release();  
}

**RunFromBridge();**

}

Bài tập 15: Bài toán sản xuất thiết bị xe hơi

MakeChassis(){ // Sản xuất một khung xe

Down(wait);

Produce\_chassis();

Up(chassis);

}

MakeTires(){ // Sản xuất một bánh xe

Produce\_tire();

Up(Tire);

}

Assemble(){ // Gán 4 bánh xe vào khung xe

Down(Tire);

Down(Chassis);

Down(Chassis);

Down(Chassis);

Put\_4\_Tires\_To\_Chassis();

Up(Wait);

}

Bài tập 16: Bài toán vượt sông

* Hàm BoardBoat() : Không chấp nhận 3 nhân viên Micro và 1 Linux Hacker trên 1 chiếc thuyền và ngược lại.
* RowBoat(): Thuyền đầy
* WaitingToBoatHacker() và WaitToBoatEmployee() : Điều kiện đợi bên ngoài để lên thuyền của Nhân viên và hacker
* WaitToRow() : Điều kiện đợi thuyền đi của những người đứng trên thuyền
* Int wHacker : Số lượng Hacker đang chờ để lên thuyền được chỉ định
* Int wEmploy : Số lượng Nhân Viên đang chờ để lên thuyền được chỉ định
* Int aHacker : Số lượng Hacker chưa lên tàu nhưng được chỉ định lên 1 chiếc thuyền
* Int aEmploy : Số lượng Nhân Viên chưa lên tàu nhưng được chỉ định lên 1 chiếc thuyền
* Int inBoat : Số lượng đang trên thuyền

Tất cả các biến trên đều được khởi tạo là 0.

Hacker(){

RuntoRiver();

**HackerArrives()**{

lock.acquire();

wHacker++;

// Trong khi không được phép lên tàu

while (aHacker == 0) {

// Nếu có 1 Hacker và còn chỗ cho 2 người khác thì cho 2 người lên

if (inBoat+aHacker+aEmploy < 4 && wHacker >= 2) {

wHacker-=2; aHacker+=2;

waitingToBoardHacker.signal();

}

// Ngược lại chờ 1 người khác lên làm 1 cặp

else {

waitingToBoardHacker.wait();

}

}

// Trên Thuyền

aHacker--;

BoardBoat();

inBoat++;

// Nếu là người cuối cùng thì đánh thức mọi người khác và rời đi

if (inBoat == 4) {

waitingToRow.broadcast();

RowBoat();

// Nếu thuyền rỗng thì để 2 hacker và 2 nhân viên sử dụng

inBoat = 0;

waitingToBoardHacker.signal();

waitingToBoardHacker.signal();

waitingToBoardEmployee.signal();

waitingToBoardEmployee.signal();

}

//Ngược lại thì chờ người cuối cùng

else {

waitingToRow.wait();

}

lock.release();

}

CrossRiver();

}

Employee(){

RuntoRiver();

**EmployeeArrives()**{

lock.acquire();

wEmploy++;

// Trong khi không được phép lên tàu

while (aEmploy == 0) {

// Nếu có 1 Nhân viên và còn chỗ cho 2 người khác thì cho 2 người lên

if (inBoat+aEmploy+ahacker < 4 && wEmploy >= 2) {

wEmploy-=2; aEmploy+=2;

waitingToBoardEmployee.signal();

}

// Ngược lại chờ 1 nhân viên khác lên làm 1 cặp

else {

waitingToBoardEmployee.wait();

}

}

// Trên Thuyền

aEmploy--;

BoardBoat();

inBoat++;

// Nếu là người cuối cùng thì đánh thức mọi người khác và rời đi

if (inBoat == 4) {

waitingToRow.broadcast();

RowBoat();

// Nếu thuyền rỗng thì để 2 nhân viên và 2 hacker sử dụng

inBoat = 0;

waitingToBoardEmployee.signal();

waitingToBoardEmployee.signal();

waitingToBoardHacker.signal();

waitingToBoardHacker.signal();

}

//Ngược lại thì chờ người cuối cùng

else {

waitingToRow.wait();

}

lock.release();

}

CrossRiver();

}

Bài 1 Trang 156:

* Tiến trình CPU-bounded:
* Các quy trình đang thực hiện các thuật toán với số lượng tính toán lớn
* Nó có thể được dự kiến sẽ giữ CPU miễn là the scheduler được cho phép
* Các chương trình tương tự hay là mô phỏng có thể bị ràng buộc bởi CPU trong phần lớn vòng đời sử dụng của tiến trình
* Người dùng thường không mong đợi phản hồi ngay lập tức từ máy tính khi chạy chương trình CPU bound
* Nó có độ ưu tiên thấp hơn bởi người lên lịch (the scheduler)
* Tiến trình I/O-bounded:
* Các tiến trình chủ yếu chờ hoàn thành đầu vào hoặc đầu ra (I/O)
* Các tiến trình tương tác với nhau, chẳng hạn như các ứng dụng văn phòng chủ yếu là (I/O) bound toàn bộ vòng đời của tiến trình. Một số tiến trình có thể bị I/O-bound trong một vài khoảng thời gian ngắn
* Thời gian chạy ngắn dự kiến của các tiến trình I/O bound có nghĩa là chúng không duy trì quá trình chạy trong thời gian dài
* Nó có độ ưu tiên cao hơn bởi người lên lịch (the scheduler)

Bài 2 Trang 156:

Việc phân biệt giữa tiến trình CPU-bounded và I/O-bounded là quan trọng đối với các bộ điều phối tiến trình vì:

* Tạo sự công bằng giữa các tiến trình nhập xuất và tính toán tạo nên sự làm việc hiệu quả
* Tăng tốc độ xử lý các tiến trình và quyền chiếm CPU giữa hai tiến trình CPU và I/O là khác nhau
* Tăng tối đa số lượng công việc có thể hoàn thành và giảm thiểu thời gian chiếm CPU
* Tận dụng các khoảng không gian của CPU để sử dụng liên tục
* Ngày nay tốc độ xử lý của CPU ngày càng nhanh so với tốc độ đọc ghi trên thiết bị lưu trữ ngoài. Kéo theo việc điều phối các tiến trình I/O-bound ngày càng quan trọng trong tương lai gần và hiện tại

Bài 3 Trang 156:

Hàng đợi cấp 0:

|  |
| --- |
| A1 |

0 8

Hàng đợi cấp 1:

|  |
| --- |
| B1 |

1. 16

Hàng đợi cấp 2:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| P1 | P2 | P3 | P4 | P5 | P6 |

0 4 11 23 43 68 98

Thời gian chờ trung bình của hàng đợi cấp 0 = 0 ms

Thời gian chờ trung bình của hàng đợi cấp 1 = 0 ms

Thời gian chờ trung bình của hàng đợi cấp 2 = (0+4+11+23+43+68)/6=149/6 ms

Thời gian hoàn thành trung bình của tiến trình 0 = 8 ms

Thời gian hoàn thành trung bình của tiến trình 1 = 16 ms

Thời gian hoàn thành trung bình của tiến trình 2 = (4+11+23+43+68+98)/6=247/6 ms

Bài 4 Trang 157:

1. Theo bộ điều phối FCFS (First Come First Serve)

|  |  |  |  |
| --- | --- | --- | --- |
| P\_1 | P\_2 | P\_3 | P\_4 |

0 10 15 24 30

-Thời gian đợi của P1 = 0

- Thời gian đợi của P2 = 15-5-1=9

-Thời gian đợi của P3 = 24-9-2=13

-Thời gian đợi của P4 = 30-6-3=21

=> Trung bình waiting time = (0+9+13+21)/4=43/4

-Thời gian hoàn thành của P1 = 10-0=10

-Thời gian hoàn thành của P2 = 15-1=14

-Thời gian hoàn thành của P3 = 24-2=22

-Thời gian hoàn thành của P4 = 30-3=27

=> Trung bình turnaround time = (10+14+22+27)/4=73/4

b. Theo bộ điều phối SJF – preemptive (Shorted Job First)

|  |  |  |  |
| --- | --- | --- | --- |
| P\_1 | P\_2 | P\_4 | P\_3 |

0 10 15 21 30

-Thời gian đợi của P1 = 0

- Thời gian đợi của P2 = 15-5-1=9

-Thời gian đợi của P3 = 21-6-3=12

-Thời gian đợi của P4 = 30-9-2=19

=> Trung bình waiting time = (0+9+12+19)/4=10

-Thời gian hoàn thành của P1 = 10-0=10

-Thời gian hoàn thành của P2 = 15-1=14

-Thời gian hoàn thành của P3 = 30-2=28

-Thời gian hoàn thành của P4 = 21-3=18

=> Trung bình turnaround time = (10+14+28+18)/4=35/2

c. Điều phối độ ưu tiên

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| P\_1 | P\_4 | P\_3 | P\_2 | P\_1 |

0 3 9 18 23 30

- Thời gian chờ của P1 = 30-0-10=20

- Thời gian chờ của P2 = 23-5-1=17

-Thời gian chờ của P3 = 18-9-2=7

-Thời gian chờ của P4 = 9-6-3=0

=> Trung bình waiting time = (20+17+7+0)/4=11

-Thời gian hoàn thành của P1 = 30

-Thời gian hoàn thành của P2 = 23-1=22

-Thời gian hoàn thành của P3 = 18-2=16

-Thời gian hoàn thành của P4 = 9-3=6

=> Trung bình turnaround time = (30+22+16+6)/4=37/2

d. Điều phối Round-Robin với time quantum 4

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| P\_1 | P\_2 | P\_3 | P\_4 | P\_1 | P\_2 | P\_3 | P\_4 | P\_1 | P\_3 |

0 4 8 12 16 20 21 25 27 29 30

-Thời gian chờ của P1= 29-0-10=19

-Thời gian chờ của P2= 21-1-5=15

-Thời gian chờ của P3=30-2-9=19

-Thời gian chờ của P4=27-3-6=18

=>Trung bình waiting time = (19+15+19+18)/4=71/4

-Thời gian hoàn thành của P1 = 29-0=29

-Thời gian hoàn thành của P2 = 21-1=20

-Thời gian hoàn thành của P3 = 30-2=28

-Thời gian hoàn thành của P4 = 27-3=24

=> Trung bình turnaround time = (29+21-1+30-2+27-3)/4=101/4

Bài 5 Trang 157:

Hệ điều hành X,Y,Z có thể đã sử dụng thuật toán điều phối:

* RR(Round Robin):Tiến trình 1b có Burst > time quantum nên tiến trình 1b được gọi ra sau cùng

Bài 6 Trang 158:

Ta có sơ đồ Gantt sau với q=4

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| P1 | P2 | P3 | P1 | P2 | P4 | P1 | P4 | P1 | P1 |

0 4 8 11 15 16 20 24 26 30 34

-Thời gian chờ của P1 = 34-0-20= 14

-Thời gian chờ của P2 = 16 -1-5= 10

-Thời gian chờ của P3 = 11-2-3= 6

-Thời gian chờ của P4 = 26-12-6= 8

Vậy thời gian chờ của tiến trình P3 là thấp nhất khi thực hiện điều phối theo chiến lược Round Robin với q=4