问答题

1. 抢占式调度不可行，假设进程0和1先后到达，且进程1到达时，interested[0]已经被设为TRUE，但while语句尚未执行，此时无论turn是否被设置为0，若进程1优先级高于进程0，则切换至进程1执行，turn被设置为1，while语句陷入循环，进程1永不释放CPU，若无其他设计（如根据运行时间的增加优先级逐渐降低），则进程0永不执行，interested[0]不会被设置为FALSE，进程1也无法退出循环。

非抢占式调度不可行，一旦某一进程进入enter\_region函数，就会直接执行到函数结尾的while循环。假设进程0先到达并完成了enter\_region操作，但在后续执行中因为某些原因（如等待某一消息）暂时让出了CPU，尚未执行leave\_region，则进程1调用enter\_region时会卡在while循环处，此时即使进程0获得了它所需的消息，也无法被唤醒继续执行，interested[0]不会变成FALSE，进程1无法退出循环。

纯粹的抢占式调度和非抢占式调度都不能在所有情况下确保Peterson算法正确，需要配合其他调度，如时间片轮转、动态优先数调度算法，以保证进入while循环等待资源的进程不会因为忙等待阻塞其他进程，导致资源永远无法被使用和释放。

1. 不会出现这样的问题。在优先级调度算法下，一旦H开始运行，L永不运行，也永远无法离开临界区，H无法进入临界区，陷入死循环。在轮询调度算法下，L一定会获得时间片运行、进入、离开临界区，此后H也可以正常完成工作。
2. 该解决方案不能满足进程互斥的所有要求。它满足互斥，即turn在某一时刻只能被设为1和0中的一个值，P0和P1永远不可能同时进入临界区。然而，每次P0或P1在临界区执行完毕后，都将turn修改为另一个值，此时若另一进程并不打算进入临界区，即使原进程想要再次进入，也必须一直等待；如果turn初始化为某一值，但首先到达的时另一个进程，它也不能执行，而必须等待。在实际操作中，不能判断哪一个进程会先到达，并且两个进程不可能是严格一个一次轮流执行的，因此该解决方案不能覆盖现实中的全部情况，不满足一个正确进程互斥设计的全部要求。

应用题

1. P1() { P2() {

y = 1; ① x = 1; ⑤

y = y + 3; ② x = x + 5; ⑥

V(S1); P(S1);

z = y + 1; ③ x = x + y; ⑦

P(S2); V(S2);

y = z + y; ④ z = z + x; ⑧

} }

根据Bernstein条件可判断语句①②③、⑤⑥⑦的结果与执行顺序无关，若这六条语句都执行完毕，④和⑧尚未执行，此时一定有x = 10，y = 4，z = 5。语句④和⑧是并发执行的，若先执行④，结果为x = 10，y = 9，z = 15，否则先执行⑧，结果x = 10，y = 19，z = 15，此外若语句③被推迟至执行完⑧后，即执行③时⑤⑥⑦⑧都已执行完毕，然后再执行④，则有x = 10，y = 9，z = 5，共三种可能的情形。

1. （1）使用信号量和PV操作：

mutex: semaphore;

wait: semaphore;

count: integer;

mutex := 1;

wait := 0;

count := 0;

cobegin

process ReadFile

var Index: integer

begin

P(mutex);

L: {

if count + Index ≥ k then

{ V(mutex); P(wait); goto L; }

}

count := count + Index;

V(mutex);

Read file;

P(mutex);

count := count – Index;

V(wait);

V(mutex);

end;

coend;

（2）使用管程：

TYPE ShareFile = MONITOR

var count: integer

full: condition

define startread, endread, readfile;

use wait, signal, check, release;

procedure startread(Index)

begin

check(IM);

L: {

if count + Index ≥ k then

{ wait(full, IM); goto L; }

}

count := count + Index;

release(IM);

end;

procedure endread(Index)

begin

check(IM);

count := count – Index;

signal(full, IM);

release(IM);

end

procedure readfile(Index)

begin

startread(Index);

Read File;

endread(Index);

end

1. （1）由Available = (1, 6, 2, 2)，可以计算出Resource = (3, 12, 14, 14)

如图，存在安全序列P0、P3、P1、P2、P4可以全部执行通过，系统是安全的。

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 资源  进程 | CurrentAvil | | | | Cki – Aki | | | | Allocation | | | | CA + Alloc | | | | Poss  ible |
| A | B | C | D | A | B | C | D | A | B | C | D | A | B | C | D |
| P0 | 1 | 6 | 2 | 2 | 0 | 0 | 1 | 2 | 0 | 0 | 3 | 2 | 1 | 6 | 5 | 4 | T |
| P1 | 1 | 9 | 8 | 6 | 1 | 7 | 5 | 2 | 1 | 0 | 0 | 0 | 2 | 9 | 8 | 6 | T |
| P2 | 2 | 9 | 8 | 6 | 2 | 3 | 5 | 6 | 1 | 3 | 5 | 4 | 3 | 12 | 13 | 10 | T |
| P3 | 1 | 6 | 5 | 4 | 0 | 6 | 5 | 2 | 0 | 3 | 3 | 2 | 1 | 9 | 8 | 6 | T |
| P4 | 3 | 12 | 13 | 10 | 0 | 6 | 5 | 6 | 0 | 0 | 1 | 4 | 3 | 12 | 14 | 14 | T |

（2）若此时进程P1发出请求request1 (1, 2, 2, 2)，假设系统分配了资源给它，则此时有Available = (1, 4, 0, 0)，更新得到的表格如图，可见此时各类资源尚可分配的数量已经无法满足任何一个进程，不存在一个安全序列，故不能这样分配。

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 资源  进程 | CurrentAvil | | | | Cki – Aki | | | | Allocation | | | | CA + Alloc | | | | Poss  ible |
| A | B | C | D | A | B | C | D | A | B | C | D | A | B | C | D |
| P0 |  |  |  |  | 0 | 0 | 1 | 2 | 0 | 0 | 3 | 2 |  |  |  |  | F |
| P1 |  |  |  |  | **0** | **5** | **3** | **0** | **2** | **2** | **2** | **2** |  |  |  |  | F |
| P2 |  |  |  |  | 2 | 3 | 5 | 6 | 1 | 3 | 5 | 4 |  |  |  |  | F |
| P3 |  |  |  |  | 0 | 6 | 5 | 2 | 0 | 3 | 3 | 2 |  |  |  |  | F |
| P4 |  |  |  |  | 0 | 6 | 5 | 6 | 0 | 0 | 1 | 4 |  |  |  |  | F |

1. 由于每个Ai对于buffer中某位的数据只需写入一次，而Bi从buffer中的每一位都要读数据，设置read和new，A写入时遍历1到m寻找一个已被全部n2个B读过的缓冲区，若找到了则写入，更新new提醒B读出新数据。B读出时遍历1到m找到A已写入而自己尚未的缓冲区，每一个都读出数据，更新read统计目前为止读过缓冲区这一位的B。根据上述逻辑设计控制消息发送和接受的程序如图：

mutex: semaphore;

var read: array[1…m] of semaphore;

new: array[1…n2][1…m] of semaphore;

mutex := 1;

read := [m, …, m];

new := [0, …, 0]

[ …… ]

[0, …, 0];

procedure send()

begin

for i from 1 to m do

P(read[i]); //check if read is n2

if P operation succeed then

index = i;

break;

P(mutex);

Put message into buffer[index];

V(mutex);

V(read[index]); //set read to 0

for j from 1 to n2 do

V(new[j][index]); //set new to 1

end

procedure receive(index)

begin

for i from 1 to m do

P(new[index][i]); //check if full is 1

P(mutex);

Get message from buffer[i];

V(mutex);

V(read[i]); //add 1 to read

V(new[index][i]); //set new to 0

end

在Ai中调用send()，Bi中调用对应的receive(i)即可。