**05-ch3参考答案**

1. 有两个优先级相同的进程P1和P2，各自执行的操作如下，信号量S1和S2初值均为0。试问P1、P2并发执行后，x、y、z的值各为多少？

P1： P2：

begin begin

y:=1; x:=1;

y:=y+3; x:=x+5;

V(S1); P(S1);

z:=y+1; x:=x+y;

P(S2); V(S2);

y:=z+y z:=z+x;

end. end.

**答**：现对进程语句进行编号，以方便描述。

P1： P2：

begin begin

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

end. end.

①、②、⑤和⑥是不相交语句，可以任何次序交错执行，而结果是唯一的。接着无论系统如何调度进程并发执行，当执行到语句⑦时，可以得到x=10，y=4。按Bernstein条件，语句③的执行结果不受语句⑦的影响，故语句③执行后得到z=5。最后，语句④和⑧并发执行，这时得到了两种结果为：

语句④先执行：x=10，y=9，z=15。

语句⑧先执行：x=10，y=19，z=15。

此外，还有第三种情况，语句③被推迟，直至语句⑧后再执行，于是依次执行以下三个语句：

z:=z+x；

z:=y+1;

y:=z+y;

这时z的值只可能是y+1=5，故y=z+y=5+4=9，而x=10。

第三种情况为：x=10，y=9，z=5。

6． 在一个盒子里，混装了数量相等的黑白围棋子。现在用自动分拣系统把黑子、白子分开，设分拣系统有二个进程P1和P2，其中P1拣白子；P2拣黑子。规定每个进程每次拣一子；当一个进程在拣时，不允许另一个进程去拣；当一个进程拣了一子时，必须让另一个进程去拣。试写出两进程P1和P2能并发正确执行的程序。

**答1**：实质上是两个进程的同步问题，设信号量S1和S2分别表示可拣白子和黑子，不失一般性，若令先拣白子。

var S1,S2:semaphore;

S1:=1;S2:=0;

cobegin

{

process P1

begin

repeat

P(S1);

拣白子

V(S2);

until false;

end

process P2

begin

repeat

P(S2);

拣黑子

V(S1);

until false;

end

}

coend.

**答2：**

TYPE pickup-chess =MONITOR

VAR flag:boolean;

S-black,S-white:codition;

DEFINE pickup-black,pickup-white;

USE wait,signal,check,release;

procedure pickup-black;

begin

check(IM);

if flag then wait(S-black,IM);

flag:=true;

pickup a black;

signal(S-white,IM);

release(IM);

end

procedure pickup-white;

begin

check(IM);

if not flag then wait(S-white,IM);

flag:=false;

pickup a white;

signal(S-black,IM);

release(IM);

end

begin

flag:=true;

end.

main()

{cobegin

process-B( );

process-W( );

coend

}

process-B()

begin

pickup-chess.pickup-black( );

other;

end

process-W()

begin

pickup-chess.pickup-white( );

other;

end

17.Dijkstra临界区软件算法描述如下：

var flag:array[0…n] of (idle,want-in,in\_cs);

turn:integer;tune:0 or 1 or…or,n-1;

process Pi(i=0,1,…,n-1)

var j;integer;

begin

repeat

repeat

flag[i]:=want\_in;

while turn≠i do

if flag[turn]==idle then turn:=i;

flag[i]:=in\_cs;

j:=0;

while (j<n)&(j==i or flag[j]≠in\_cs)

do j:=j+1;

until j≧n;

critical section;

flag[i]:=idle;

…

until false;

end.

试说明该算法满足临界区原则。

**答**：为方便描述，把Dijkstra程序的语句进行编号：

repeat

flag[i]:=want\_in; ①

while turn≠i do ②

if flag[turn]==idle then turn:=i; ③

flag[i]:=in\_cs; ④

j:=0;

while (j<n)&(j==i or flag[j]≠in\_cs) ⑤

do j:=j+1; ⑥

until j≧n;

critical section;

flag[i]:=idle; ⑦

…

* 1. 满足互斥条件

当所有的Pj都不在临界区中，满足flag[j]≠in\_cs (对于所有j，j≠i)条件时，Pi才能进入它的临界区，而且进程Pi不会改变除自己外的其他进程所对应的flag[j]的值。另外，进程Pi总是先置自己的flag[i]为in\_cs后，才去判别Pj进程的flag[j]的值是否等于in\_cs，所以，此算法能保证n个进程互斥地进入临界区。

* 1. 不会发生无休止等待进入临界区

由于任何一个进程Pi在执行进入临界区代码时先执行语句①，其相应的flag[i]的值不会是idle。注意到flag[i]=in\_cs并不意味着turn的值一定等于i。我们来看以下情况，不失一般性，令turn的初值为0，且P0不工作，所以，flag[turn]=flag[0]=idle。但是若干个其他进程是可能同时交替执行的，假设让进程Pj(j=1,2,..,n-1)交错执行语句①后(这时flag[j]=want\_in)，再做语句②(第一个while语句)，来查询flag[turn]的状态。显然，都满足turn≠i，所以，都可以执行语句③，让自己的turn为j。但turn仅有一个值，该值为最后一个执行此赋值语句的进程号，设为k、即turn=k(1≤k≦n-1)。接着，进程Pj(j=1,2,..,n-1)交错执行语句④，于是最多同时可能有n-1个进程处于in\_cs状态，但不要忘了仅有一个进程能成功执行语句④，将turn置为自己的值。

假设{P1,P2,…Pm}是一个已将flag[i]置为in\_cs(i=1,2,…,m)(m≤n-1)的进程集合，并且已经假设当前turn=k(1≤k≤m)，则Pk必将在有限时间内首先进入临界区。因为集合中除了Pk之外的所有其他进程终将从它们执行的语句⑤(第二个while循环语句)退出，且这时的j值必小于n，故内嵌until起作用，返回到起始语句①重新执行，再次置flag[i]=want\_in，继续第二轮循环，这时的情况不同了，flag[turn]=flag[k]必定≠idle(而为in\_cs)。而进程Pk发现最终除自身外的所有进程Pj的flag[j]≠in\_cs，并据此可进入其临界区。

1. 如图所示，四个进程Pi（i=0…3）和四个信箱Mj（j=0…3），进程间借助相邻信箱传递消息，即Pi每次从Mi中取一条消息，经加工后送入M(i+1)mod4，其中M0、M1、M2、M3分别可存放3、3、2、2个消息。初始状态下，M0装了三条消息，其余为空。试以P、V操作为工具，写出Pi（i=0…3）的同步工作算法。

P0

P1

P2

P3

M1

M2

M3

M0

**答：**

var mutex1,mutex2,mutex3,mutex0:semaphore;

mutex1:=mutex2:=mutex3:=mutex0:=1;

empty0,empty1,empty2,empty3:semaphore;

empty:=0;empty1:=3;empty:=2:=empty3:=2;

full0,full1,full2,full3:semaphore;

full0:=3;full1:=full2:=full3:=0;

in0,in1,in2,in3,out0,out1,out2,out3;integer;

in0:=in1:=in2:=in3:=out0:=out1:=out2:=out3:=0;

cobegin

{

process P0

begin

repeat

P(full0);

P(mutex0);

从M0[out0]取一条消息;

out0:=(out0+1) mod 3;

V(mutex0);

V(empty0);

加工消息;

P(empty1);

P(mutex1);

消息已M1[in1];

in1:=(in1+1) mod 3;

V(mutex1);

V(full1);

untile false;

end

process P1

begin

repeat

P(full1);

P(mutex1);

从M1[out1]取一条消息;

out1:=(out1+1) mod 3;

V(mutex1);

V(empty1);

加工消息;

P(empty2);

P(mutex2);

消息已M2[in2];

in2:=(in2+1) mod 2;

V(mutex2);

V(full2);

untile false;

end

process P2

begin

repeat

P(full2);

P(mutex2);

从M2[out2]取一条消息;

out2:=(out2+1) mod 2;

V(mutex2);

V(empty2);

加工消息;

P(empty3);

P(mutex3);

消息已M3[in3];

in3:=(in3+1) mod 2;

V(mutex3);

V(full3);

untile false;

end

process P3

begin

repeat

P(full3);

P(mutex3);

从M3[out3]取一条消息;

out3:=(out3+1) mod 2;

V(mutex3);

V(empty3);

加工消息;

P(empty0);

P(mutex0);

消息已M0[in0];

in0:=(in0+1) mod 3;

V(mutex0);

V(full0);

untile false;

end

}

coend.

1. Jurassic公园有一个恐龙博物馆和一个花园，有m个旅客和n辆车，每辆车仅能乘一个旅客。旅客在博物馆逛了一会，然后，排队乘坐旅行车，当一辆车可用时，它载入一个旅客，再绕花园行驶任意长的时间。若n辆车都已被旅客乘坐游玩，则想坐车的旅客需要等待。如果一辆车已经空闲，但没有游玩的旅客了，那么，车辆要等待。试用信号量和P、V操作同步m个旅客和n辆车子。

**答**：这是一个汇合机制，有两类进程：顾客进程和车辆进程，需要进行汇合、即顾客要坐进车辆后才能游玩，开始时让车辆进程进入等待状态。

var scl(使用的车子总数),sck（空车）,sc,kx,xc,mutex:semaphore;

sck := kx:= sc:= xc:=0;

sc1:=n;mutex:=1;

sharearea:一个登记车辆\被服务乘客信息的共享区;

cobegin

process 顾客i(i=1,2,…)

begin

P(scl); /\*车辆最大数量信号量

P(mutex); /\*封锁共享区，互斥操作

在共享区sharearea登记被服务的顾客的信息：起始和到达地点，行驶时间

V(sck); /\*释放一辆车，即顾客找到一辆空车

P(kx); /\*车辆要配备驾驶员，顾客等待被载，

上车；

V(sc); /\*顾客进程已汇合到车辆进程，即顾客坐进车里

P(xc); /\*待游玩结束后后，顾客等待下车

V(scl); /\*空车辆数加1

end

Process 车辆j(j=1,2,…)

begin

L: P(sck); /\*车辆等待有硕客来使用

在共享区sharearea登记那一辆车被使用，并与顾客进程汇合；

V(mutex); /\*这时可开放共享区，让另一顾客雇车

V(kx); /\*允许顾客用此车辆

P(sc); /\*车辆等待顾客上车

车辆载着顾客开行到目的地；

v(xc); /\*允许顾客下车

goto L;

end

coend

1. 今有k个进程，它们的标号依次为1、2、…、k，如果允许它们同时读文件file，但必须满足条件：参加同时读文件的进程的标号之和需小于K，请使用：1)信号量与P、V操作，2)管程，编写出协调多进程读文件的程序。

**答1)**：使用信号量与P、V操作

var waits,mutex:semaphore;

numbersum:integer:=0;

wait:=0;mutex:=1;

cobegin

{

process readeri(var number:integer;)

begin

P(mutex);

L: if numbersum+number≥K then {V(mutex);P(waits); goto L;}

then numbersum:=numbersum+number;

V(mutex);

Read file;

P(mutex);

numbersub:=numbersum-number;

V(waits);

V(mutex);

}

1. 略。
   * 1. 系统有A、B、C、D共4种资源，在某时刻进程P0、P1、P2、P3和P4对资源的占有和需求情况如表，试解答下列问题：

|  |  |  |  |
| --- | --- | --- | --- |
| Process | Allocation | Claim | Available |
| A B C D | A B C D | A B C D |
| P0 | 0 0 3 2 | 0 0 4 4 | 1 6 2 2 |
| P1 | 1 0 0 0 | 2 7 5 0 |  |
| P2 | 1 3 5 4 | 3 6 10 10 |  |
| P3 | 0 3 3 2 | 0 9 8 4 |  |
| P4 | 0 0 1 4 | 0 6 6 10 |  |

* + 1. 系统此时处于安全状态吗？
    2. 若此时P2发出request2(1、2、2、2)，系统能分配资源给它吗？为什么？

**答：**(1)系统处于安全状态，存在安全序列：P0，P3，P4，P1，P2。

(2)不能分配，否则系统会处于不安全状态。

1. 考虑一个共有150个存储单元的系统，如下分配给三个进程，P1最大需求70,己占有25；P2最大需求60,己占有40；P3最大需求60,己占有45。使用银行家算法，以确定下面的任何一个请求是否安全。(1)P4进程到达，P4最大需求60,最初请求25个。(2)P4进程到达，P4最大需求60,最初请求35。如果安全，找出安全序列；如果不安全，给出结果分配情况。

**答：**

* + - 1. 由于系统目前还有150-25-40-45=40个单元，P4进程到达，把25个单元分给它。这时系统还余15个单元，可把15个单元分给P3，它执行完后会释放60个单元。于是可供P1(还要45个单元)，P2(还要20个单元)，P4(还要35个单元)任何一个执行。安全序列为：

P1，P2，P3，P4，P3，P1，P2，P4

P1，P2，P3，P4，P3，P1，P4，P2

P1，P2，P3，P4，P3，P2，P1，P4

P1，P2，P3，P4，P3，P2，P4，P1

P1，P2，P3，P4，P3，P4，P1，P2

P1，P2，P3，P4，P3，P4，P2，P1

* + - 1. P4进程到达，P4最大需求60,最初请求35。如果把35个单元分给P4，系统还余5个单元，不再能满足任何一个进程的需求，系统进入不安全状态。

1. 有一个仓库，可存放X、Y两种产品，仓库的存储空间足够大，但要求：(1) 每次只能存入一种产品X或Y， (2) 满足-N<X产品数量 - Y 产品数量<M。其中，N和M是正整数，试用信号量与P、V操作实现产品X与Y的入库过程。

**答：**本题给出的表达式可分解为制约条件：

－N＜X产品数量－Y产品数量

X产品数量－Y产品数量＜M

也就是说，X产品的数量不能比Y产品的数量少N个以上，X产品的数量不能比Y产品的数量多M个以上。可以设置两个信号量来控制X、Y产品的存放数量：

sx表示当前允许X产品比Y产品多入库的数量，即在当前库存量和Y产品不入库的情况下，还可以允许sx个X产品入库；初始时，若不放Y而仅放X产品，则sx最多为M－1个。

sy表示当前允许Y产品比X产品多入库的数量，即在当前库存量和X产品不入库的情况下，还可以允许sy个Y产品入库。初始时，若不放X而仅放Y产品，则sy最多为N－1个。

当往库中存放入一个X产品时，则允许存入Y产品的数量也增加1，故信号量sy应加1；当往库中存放入一个Y产品时，则允许存入X产品的数量也增加1，故信号量sx应加1。

var mutex：semaphore=1;/\*互斥信号量\*/

sx，sy：semaphore；

sx=M-1; sy=N-1;

cobegin

{

process X

{ repeat

P(sx);

P(mutex);

将X产品入库；

V(mutex);

V(sy);

until false

}

process Y

{ repeat

P(sy);

P(mutex);

将Y产品入库；

V(mutex);

V(px);

until false

}

}

coend.

41. 下述流程是解决两进程互斥访问临界区问题的一种方法。试从“互斥”(mutual exclusion)、“空闲让进”(progress)、“有限等待”(bounded waiting)等三方面讨论它的正确性。如果它是正确的，则证明之；如果它不正确，请说明理由。

program attemp;

var c1,c2:integer;

procedure p1; (/\* 对第一个进程p1 \*/)

begin

repeat

Remain Section 1;

repeat

一 c1:=1-c2

until c2<>0;

Critical Section; (/\* 临界区 \*/)

c1:=1

until false

end;

procedure p2; (/\* 对另一个进程p2 \*/)

begin

repeat

Remain Section 2;

repeat

c2:=1-c1

until c1<>0;

Critical Section; (/\* 临界区 \*/)

c2:=1

until false

end;

begin (/\* 主程序 \*/)

c1:=1;

c2:=1;

cobegin

p1;p2 (/\* 两进程p1, p2开始执行 \*/)

coend

end.

**答**：(1)互斥

己知c1和c2的初值为1，若进程P1执行到c1:=1-c2时，进程P2也同时执行c2:=1-c1。这样一来，c1和c2的值都变为0，接着再各自执行repeat ---untile循环语句c1:=1-c2和c2:=1-c1时，c1和c2就又都变回了1。于是，P1和P2会同时进入临界区，不满足互斥条件。

1. 有空让进

设开始无进程在临界区中，进程P1执行了c1:=1-c2，由于c2的初值为1，这使得c1的值变为0但c2仍为1，从而保证了P1进入临界区。当P1退出临界区时，执行了c1:=1，使得P2就可进入临界区。进程P2先执行的情况相似，能保证有空让进的原则。

1. 有限等待

假定进程P1在临界区执行，进程P2申请进入临界区，则因进程P1会在有限时间内执行完并退出临界区，然后，将执行c1:=1，这使得进程P2因c1值为1而立即可进入临界区。因而，能满足有限等待的原则。

49 有P1、P2、P3三个进程共享一个表格F，P1对F只读不写，P2对F只写不读，P3对F先读后写。进程可同时读F，但有进程写时，其他进程不能读和写。用(1)信号量和P、V操作，(2)管程编写三进程能正确工作的程序。

**答**：(1)信号量和P、V操作。

这是读--写者问题的变种。其中，P3既是读者又是写者。读者与写者之间需要互斥，写者与写者之间需要互斥，为提高进程运行的并发性，可让读者尽量优先。

var rmutex,wmutex:semaphore;

rmutex:=wmutex:=1;

count:integer;count:=0;

cobegin

{

process P1

begin

repeat

P(rmutex);

count:=count+1;

if count=1 then P(wmutex);

V(rmutex);

Read F;

P(rmutex);

count:=count-1;

if count=0 then V(wmutex);

V(rmutex);

untile false;

end

process P2

begin

repeat

P(wmutex);

Write F;

V(wmutex);

untile false;

process P3

begin

repeat

P(rmutex);

count:=count+1;

if count=1 then P(wmutex);

V(rmutex);

Read F;

P(rmutex);

count:=count-1;

if count=0 then V(wmutex);

V(rmutex);

P(wmutex);

Write F;

V(wmutex);

untile false;

end

}

coend

(2)管程。

见课本读者写者问题的解。

1. 在一个分页存储管理系统中，用free[index]数组记录每个页框状态，共有n个页框(index=0，…，n-1)。当free[index]=true时，表示第index个页框空闲，free[index]=false时，表示第index个页框。试设计一个管程，它有两个过程acquire和return分别负责分配和回收一个页框。

**答**：

TYPE framemanagement=monitor

VAR free:array[0…n-1] of Boolean;

waitcondition:codition;i:integer;

DEFINE acquire,release;

USE check,wait,signal,return;

procedure acquire(var index:integer;)

begin

check(IM);

for i:=0 to n-1 do

if free[i] then {free[i]:=false; index:=i;}

else wait(waitcondition,IM);

release(IM);

end

procedure return(var index:integer;)

begin

check(IM);

free[index]:=true;

signal(waitcondition,IM);

release(IM);

end

begin

for index:=0 to n-1 do free[index]:=true;

end

进程调用管程申请和归还页框的过程从略。

59.一般型信号量机制(参见汤子瀛等编著的计算机操作系统，西安电子科技大学出版社)

对AND型信号量机制作扩充，便形成了一般型信号量机制，SP(s1,t1,d1;…;sn,tn,dn)和SV(s1,d1;…sn,dn)的定义如下：

procedure SP(s1,t1,d1；…；sn,tn,dn)

var s1,…,sn:semaphore;

t1, …,tn:integer;

d1,…,dn:integer;

begin

if s1>=t1& …&sn>=tn then begin

for i :=1 to n do

si:=si-di;

end

else

{进程进入第一个遇到的满足si<ti条件的si信号量队列等待，

同时将该进程的程序计数器地址回退，置为SP操作处。};

end

end

procedure SV(s1,d1;…sn,dn)

var s1,…sn:semaphore;

d1,…dn:integer;

begin

for i:=1 to n do begin

si:=si+di;

{从所有si信号量等待队列中移出进程并置入就绪队列。};

end

end

其中，ti为这类临界资源的阀值，di为这类临界资源的本次请求数。

试回答一般型信号量机制的主要特点，适用于什么场合？

**答**：在记录型和同时型信号量机制中，P、V或SP、SV仅仅能对信号量施行增1或减1操作，每次只能获得或释放一个临界资源。当一请求n个资源时，便需要n次信号量操作，这样做效率很低。此外，在有些情况下，当资源数量小于一个下限时，便不预分配。为此，可以在分配之前，测试某资源的数量是否大于阀值t。对AND型信号量机制作扩充，便形成了一般型信号量机制。

60.下面是一般信号量的一些特殊情况：

SP(s,d,d)

SP(s,1,1)

SP(s,1,0)

试解释它们的物理含义或所起的作用。

**答：**

SP(s,d,d) 此时在信号量集合中只有一个信号量、即仅处理一种临界资源，但允许每次可以申请d个，当资源数少于d个时，不予分配。

SP(s,1,1) 此时信号量集合已蜕化为记录型信号量(当s>1时)或互斥信号量(s=1时)。

SP(s,1,0) 这是一个特殊且很有用的信号量，当s>=1时，允许多个进程进入指定区域；当s变成0后，将阻止任何进程进入该区域。也就是说，它成了一个可控开关。

61.试利用一般信号量机制解决读者-写者问题。

**答**：对读者-写者问题作一条限制，最多只允许rn个读者同时读。为此，又引入了一个信号量L，赋予其初值为rn，通过执行SP(L,1,1)操作来控制读者的数目，每当一个读者进入时，都要做一次SP(L,1,1)操作，使L的值减1。当有rn个读者进入读后，L便减为0，而第rn+1个读者必然会因执行SP(L,1,1)操作失败而被封锁。

利用一般信号量机制解决读者-写者问题的算法描述如下：

var rn:integer; /\*允许同时读的读进程数

L:semaphore:=rn; /\*控制读进程数信号量,最多rn

W:semaphore:=1;

begin

cobegin

process reader

begin

repeat

SP(L,1,1 ;W,1,0);

Read the file;

SV(L,1);

until false;

end

process writer

begin

Repeat

SP(W,1,1;L,rn,0);

Write the file;

SV(W,1);

until false;

end

coend

end.

上述算法中，SP(W,1,0) 语句起开关作用，只要没有写者进程进入写，由于这时W=1，读者进程就都可以进入读文件。但一旦有写者进程进入写时，其W=0，则任何读者进程及其他写者进程就无法进入读写。SP(W,1,1;L,rn,o) 语句表示仅当既无写者进程在写(这时W=1) 、又无读者进程在读(这时L=rn) 时，写者进程才能进行临界区写文件。