OS\_Chap 2\_PV操作习题课讲义(答案)

例题讲解

1）生产者-消费者问题

一般意义的“生产者—消费者”问题：N个buffer，多个生产者，多个消费者，循环存取buffer。这就是一般意义的“生产者—消费者”问题。利用记录型信号量解决一般意义的“生产者—消费者”问题算法描述，请看教材。

说明：

（1）由于buffer有N个，而且buffer又是临界资源，因此，需要增加一个信号量mutex来实现对buffer的互斥访问，其初始值为1。需要特别强调的是，这种情况下，mutex不能省略。

（2）这种情况下，只要保证为不同的进程分配不同buffer，putdata和getdata操作是可以同时进行的，因此，对buffer的互斥访问不是发生在对buffer的存取操作上，而是发生在对buffer的分配上。

2）“哲学家进餐”问题

教材中解决哲学家进餐问题的算法有可能发生死锁，为避免死锁，可以采用以下三种策略：

策略一原理：至多只允许四个哲学家同时进餐，以保证至少有一个哲学家能够进餐，最终总会释放出他所使用过的两支筷子，从而可使更多的哲学家进餐。定义信号量count，只允许4个哲学家同时进餐，这样就能保证至少有一个哲学家可以就餐。

semaphore chopstick[5]={1，1，1，1，1};

semaphore count=4;

void philosopher(int i)

{while(true)

{

think();

wait(count); //请求进入房间进餐

wait(chopstick[i]); //请求左手边的筷子

wait(chopstick[(i+1)%5]); //请求右手边的筷子

eat();

signal(chopstick[(i+1)%5]); //释放右手边的筷子

signal(chopstick[i]); //释放左手边的筷子

signal(count); //退出房间释放信号量

}

}

策略二原理：仅当哲学家的左右两支筷子都可用时，才允许他拿起筷子进餐。可以利用AND 型信号量机制实现，也可以利用信号量的保护机制实现。利用信号量的保护机制实现的思想是通过记录型信号量mutex对取左侧和右侧筷子的操作进行保护，使之成为一个原子操作，这样可以防止死锁的出现。描述如下：

semaphore mutex = 1 ;

semaphore chopstick[5]={1，1，1，1，1};

void philosopher(int i)

{while(true)

{

think();

wait(mutex);

wait(chopstick[(i+1)%5]);

wait(chopstick[i]);

signal(mutex);

eat();

signal(chopstick[(i+1)%5]);

signal(chopstick[i]);

}

}

策略三原理：规定奇数号的哲学家先拿起他左边的筷子，然后再去拿他右边的筷子；而偶数号的哲学家则相反。按此规定，将是1、2号哲学家竞争1号筷子，3、4号哲学家竞争3号筷子。即五个哲学家都竞争奇数号筷子，获得后，再去竞争偶数号筷子，最后总会有一个哲学家能获得两支筷子而进餐。

semaphore chopstick[5]={1，1，1，1，1};

void philosopher(int i)

{while(true)

{

think();

if(i%2 == 0) //偶数哲学家，先右后左。

{

wait (chopstick[(i + 1)%5]) ;

wait (chopstick[i]) ;

eat();

signal (chopstick[(i + 1)%5]) ;

signal (chopstick[i]) ;

}

else //奇数哲学家，先左后右。

{

wait (chopstick[i]) ;

wait (chopstick[(i + 1)%5]) ;

eat();

signal (chopstick[i]) ;

signal (chopstick[(i + 1)%5]) ;

}

}

3）“读者—写者”问题的演变

从本质上讲，读者—写者问题要解决：读、读共享；写、写互斥；写、读互斥。有两种解决模式：

模式一，读者优先的“读者—写者”问题解决模式（算法描述请参考教材）：

①定义互斥信号量wmutex，实现写、写互斥和写、读互斥。

②定义整型变量Readcount表示正在读的进程数目。由于只要有一个Reader进程在读，便不允许Writer进程写，因此仅当Readcount=0，即无Reader进程在读时，Reader才需要执行Wait(wmutex)操作。若Wait(wmutex)操作成功，Reader进程便可去读，相应地，做Readcount+1操作。同理，仅当Reader进程在执行了Readcount减1操作后其值为0时，才需执行signal(wmutex)操作，以便让Write进程写。

③由于Readcount为多个读进程共享（修改），因此需要以互斥方式访问，为此，需要定义互斥信号量rmutex，保证读进程间互斥访问Readcount。

说明：在读者—写者问题中，实现“读、读共享”是有一定难度的，请掌握该模式（由②和③构成）。

模式二，写者优先的“读者—写者”问题解决模式，算法描述如下：

设3个信号量：

rmutex --- 读互斥信号量，初值为1；

wmutex --- 写互斥信号量，初值为1；

s --- 用于在写进程到达后封锁后续的读者，初值为1；

count --- 共享变量，用于记录当前正在读文件的读者数目，初值为0；

semaphore rmutex=1, wmutex=1, s=1;

int count=0;

main() {

Cobegin

Reader();

Writer();

Coend

}

Reader() {

While(1) {

P(s);

P(rmutex);

If (count==0) P(wmutex); /\*当第1个读者读文件时，阻止写者写\*/

count++

V(rmutex);

V(s);

读文件；

P(rmutex);

count--;

If (count==0) V(wmutex); /\*当最后1个读者读完文件时，允许写者写\*/

V(rmutex);

}

}

Writer() {

While(1) {

P(s);

P(wmutex);

写文件；

V(wmutex);

V(s);

}

}

例题解析

【例1】 桌上有1空盘，允许存放1个水果。爸爸向盘中放苹果，也可以向盘中放桔子。儿子专等吃盘中的桔子，女儿专等吃盘中的苹果。规定当盘空时一次只能放1个水果供吃者取用。请用Wait()、Signal()原语实现爸爸、儿子、女儿三个并发进程的同步。

【南京大学2000】

【分析】这是复杂情况的“生产者—消费者”问题，既有同步又有互斥。爸爸进程与儿子进程、女儿进程需要同步，儿子进程与女儿进程需要互斥。设置4个信号量S（盘子是否为空，初值为1）、So（盘中是否有桔子，初值为0）、Sa（盘中是否有苹果，初值为0）和mutex（用于对盘子的互斥访问，初值为1）。由于只有一个盘子（相当于只有一个buffer），对盘子的互斥访问发生在对盘子的存取操作上，S、So和Sa就可以保证对盘子的互斥操作了，故mutex也可以省略。

解：设三个信号量：

S --- 盘子是否为空，初值为1；

So --- 盘中是否有桔子，初值为0；

Sa --- 盘中是否有苹果，初值为0；

Semaphore S=1, So=0, Sa=0;

Main() {

Cobegin

Father();

Son();

Daughter();

Coend

}

Father() {

While(1) {

Wait(S); 将水果放入盘中；

If (放入的是桔子) Signal(So);

Else Signal(Sa);

}

}

Son() {

While(1) {

Wait(So); 从盘中取出桔子；Signal(S); 吃桔子；

}

}

Daughter() {

While(1) {

Wait(Sa); 从盘中取出苹果；Signal(S); 吃苹果；

}

}

【例2】 如图1所示，有多个PUT操作同时向Buff1放数据，有一个MOVE操作不断地将Buff1的数据移到Buff2，有多个GET操作不断地从Buff2中将数据取走。Buff1的容量为m，Buff2的容量是n，PUT、MOVE、GET每次操作一个数据，在操作的过程中要保证数据不丢失。试用P、V原语协调PUT、MOVE的操作，并说明每个信号量的含义和初值。



m

n

图1 进程操作图

【分析】这里存在两个一般意义的“生产者—消费者”问题，PUT（生产者）与MOVE（消费者）之间，需要设置三个信号量；MOVE（生产者）与GET（消费者）之间，需要设置三个信号量。PUT进程套用生产者进程即可，MOVE进程只有在Buff1有新数据且Buff2有空闲区的时候才移动数据，GET进程套用消费者进程即可。

答案：设置6个信号量full1、empty1、B-M1、full2、empty2、B-M2，它们的含义和初值如下：

* 1. full1表示Buff1是否有数据，初值为0；
  2. empty1表示Buff1有空间，初值为m；
  3. B-M1表示Buff1是否可操作，初值为1；
  4. Full2表示Buff2是否有数据，初值为0；
  5. Empty2表示Buff2有空间，初值为n；
  6. B-M2表示Buff2是否可操作，初值为1；

<PUT类进程>

{

repeat

P(empty1)； /\*判断Buff1是否有空间，没有则等待 \*/

P(B-M1)； /\*是否可操作Buff1\*/

PUT；

V(B-M1)； /\*设置Buff1可操作标志 \*/

V(full1)； /\*设置Buff1有数据的标志 \*/

until false

}

<MOVE类进程>

{

repeat

P(full1)； /\*判断Buff1是否有数据，没有则等待\*/

P(empty2)； /\*判断Buff2是否有空间，没有则等待\*/

P(B-M1)； /\*是否可操作Buff1 \*/

P(B-M2)； /\*是否可操作Buff2 \*/

MOVE；

V(B-M1)； /\*设置Buff1可操作标志\*/

V(B-M2)； /\*设置Buff2可操作标志\*/

V(empty1)； /\*设置Buff1有空间标志\*/

V(full2)； /\*设置Buff2有数据标志\*/

until false

}

<GET类进程>

{

repeat

P(full2)； /\*判断Buff2是否有数据，没有则等待 \*/

P(B-M2)； /\*是否可操作Buff2\*/

GET；

V(B-M2)； /\*设置Buff2可操作标志 \*/

V(empty2)； /\*设置Buff2有空间的标志 \*/

until false

【例3】（8分）某银行提供1个服务窗口和10个供顾客等待的座位。顾客到达银行时，若有空座位，则到取号机上领取一个号，等待叫号。取号机每次仅允许一位顾客使用，当营业员空闲时，通过叫号选取1位顾客，并为其服务。顾客和营业员的活动过程描述如下：

cobegin

{

process 顾客i

{

从取号机获得一个号码；

等待叫号；

获得服务；

}

process 营业员

{

while （TRUE）

{

叫号；

为顾客服务；

}

}

} coend

请添加必要的信号量和P、V（或wait()、signal()）操作，实现上述过程中的护持与同步。要求写出完整的过程，说明信号量的含义并赋初值。 【全国统考 2011】

解：

semaphore emptySeats:=10 //空闲座位数

semaphore fullSeats:=0 //已占座位数

semaphore mutex:=1 //顾客使用取号机互斥信号量

cobegin

{

process 顾客i

{

wait(emptySeats); //获取一个座位

wait(mutex); //占用取号机取号

从取号机获得一个号码；

signal(mutex) //释放取号机

signal(fullSeats) //座位上增加一个顾客

等待叫号；

signal(emptySeats); //释放一个座位

获得服务；

}

process 营业员

{

while （TRUE）

{

wait(fullSeats); //座位上减少一个顾客

叫号；

为顾客服务；

}

}

} coend

讲评：此题为生产者-消费者问题的翻版。课本上的哲学家进餐问题、生产者–消费者问题、读者–写者问题为基础性的进程同步问题，需要认真真正掌握，以此为基础用于解决其他进程同步问题。