3-7

这里使用三个信号量：互斥访问临界区的信号量（mutex-初值为1），另两个用于控制缓冲区中的消息数量（full-初值为0）&（empty-初值为缓冲区的个数=1）来完成同步。

现可用PV操作描述如下：

//进程A1、A2….、An发送消息

P(empty) //等待缓冲区有空闲空间

P(mutex) //进入临界区

//将消息写入缓冲区

V(mutex) //离开临界区

V(full) //增加缓冲区中的数量

//进程B接收消息

P(full) //等待缓冲区有消息

P(mutex) //进入临界区

//从缓冲区中接收消息  
 V(mutex) //离开临界区

V(empty) //增加缓冲区中的空闲空间

3-8

已经循环缓冲区的容量为100。则当

当IN的下一个位置等于OUT的位置时，表示缓冲区已满。这意味着没有空间可用于写入新的数据。

情况1：(IN+1)%100 = OUT时，说明着缓冲区已满

当IN和OUT的值相等时，表示缓冲区为空。这意味着没有可用的数据供读取。

情况2：IN=OUT时，说明缓冲区为空

3-9

(1) 为描述读者的动作，这里需要编写两个程序，一个程序用于读者的进入和登记，另一个程序用于读者的离开和取消登记。对于进程，这里需要设置两个进程，一个用于进入和登记，另一个用于离开和取消登记。这两个进程之间需要共享一个登记表，以便记录读者的座位号和姓名。它们之间的关系是协同合作，以确保在读者进入和离开时正确更新登记表。

(2)为了让读者能够正确地登记，各读者应该互斥访问登记表，这里设置一个信号量（mutex-初值为1）。对于座位这里设置一个信号量（seats-初值为100）。

现可用PV操作描述如下：

//读者进入和登记

P(seats) //等待可用座位

P(mutex) //进入临界区

//登记读者的座位号和姓名

V(mutex) //离开临界区

//读者离开和取消登记

P(mutex) //进入临界区

//取消登记读者的座位号和姓名

V(mutex) //离开临界区

V(seats) //释放座位，增加可用座位

3-14

设每个进程最多需要x个资源。当每个进程都申请x-1个资源，这个时候如果系统没有资源了，它们中任何一个进程，如果再提出申请1个资源的要求，就会死锁。如果系统还有1个资源，就满足了提出申请资源的某一个进程，它释放所持有的x-1个资源，死锁局面被打破。

所以如果系统死锁必有n(x-1)+1>m,化简后得nx>m+n-1,所以nx>m+n，这跟nx<m+n形成矛盾，因此在这种情况下不会产生死锁。

3-15

这里设置8个同步信号量：

S12: M2等待M1运行结束,初值为0

S13: M3等待M1运行结束,初值为0

S14: M4等待M1运行结束,初值为0

S26: M6等待M2运行结束,初值为0

S36: M6等待M3运行结束,初值为0

S38: M8等待M3运行结束,初值为0

S47: M7等待M4运行结束,初值为0

S78: M8等待M7运行结束,初值为0

S57: M7等待M5运行结束,初值为0

各进程同步程序如下:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| M1 | M2 | M3 | M4 | M5 | M6 | M7 | M8 |
| M1代码  V(S12)  V(S13)  V(S14) | P(S12)  M2代码  V(S26) | P(S13)  M3代码  V(S38) | P(S14)  M4代码  V(S47) | M5代码  V(S57) | P(S26)  P(S36)  M6代码 | P(S47)  P(S57)  M7代码  V(S78) | P(S38)  P(S78)  M8代码 |

3-16

1. 当每个哲学家都试图同时拿起自己两边的叉子时，可能会出现死锁。具体来说，如果每个哲学家都试图拿起右边的叉子，然后等待左边的叉子，这将导致所有哲学家都无法获得两把叉子而无法吃饭。
2. 1. 给每个叉子设置一个信号量，初始值为1。

2. 每个哲学家在尝试拿叉子之前，先尝试获取自己左边和右边的叉子。

3. 如果两个叉子都可用，哲学家拿起叉子，进餐。

4. 如果某一把叉子不可用，哲学家将放下已拿到的叉子，并等待直到两个叉子都可用。

5. 当哲学家吃完后，放下叉子，使它们重新可用。

这个算法确保了每个哲学家都能成功获得两把叉子以进餐，并且通过仔细的叉子管理，可以防止死锁的发生。如果某个哲学家不能获得所需的叉子，它会放下已经拿到的叉子，等待叉子可用，以便其他哲学家可以使用它们。这种算法保证了公平性和避免了死锁。

PV实现：

int mutex=1;

int fork[0]=fork[1]=…=fork[4]=1;

第i个哲学家所执行的程序：

do{

//思考

Think()

P(mutex);

//尝试拿起左边的叉子

P(fork[i]);

//尝试拿起右边的叉子

P(fork[(i+1)mod5]);

V(mutex);

吃饭

//放下叉子，使其重新可用

V(fork[i]);

V(fork[(i+1)mod5]);

}while(1)；

3-17

1. 公平竞争（无写者时，读者仍遵循多个读者可以同时读）

rmutex互斥共享readcount; wmutex读写互斥，写互斥，读写进程在z上排队

rmutex=1，wmutex=1，readcount=0

//读者

P(z) //读写进程在z上排队

P(rmutex) //申请对readcount的使用

If(readcount==0) P(wmutex) //第一个读者阻止后面的写者使用

readcount++

V(rmutex) //释放对readcount的使用

V(z) //释放进程

P(rmutex) //申请对readcount的使用

readcount—

if(readcount==0) V(wmutex); //最后一个读者使用完，写者才可使用

V(rmutex) //释放对readcount的使用

//写者

P(z) //读写进程在z上排队

P(wmutex)

Write data

V(wmutex)

V(z)

1. 写者优先

semaphore S=1;

semaphore rmutex=1，wmutex=1, wcMutex=1;

int readcount=0, writecount=0;

//读者

P(S);

P(rmutex)；//申请对readcount的使用

if(readcount==0) P(wmutex);//第一个读者阻止后面的写者使用

readcount++;

V(rmutex);//释放对readcount的使用

V(S);

read;//读者读

P(rmutex);//申请对readcount的使用

readcount--;

if(readcount==0) V(wmutex);//最后一个读者使用完，写者才可使用

V(rmutex);//释放对readcount的使用

//写者

P(wcMutex);//申请对writecount的使用

if(writecount==0) P(S);//第一个写者阻止后面的读者使用

writecount++;

V(wcMutex);//释放对writecount的使用

P(wmutex);

write;//读者读

V(wmutex);

P(wcMutex);//申请对writecount的使用

writecount--;

if(writecount==0) V(S);//最后一个写者使用完，读者才可使用

V(wcMutex);//释放对writecount的使用

3-18

这里设置三个信号量，它们分别是

wakeup=0,wait=0,mutex=1

并使用计数器rc=0来计算顾客的数量

PV操作：

//顾客进程

P(mutex); //访问计数器

rc++;

if(rc==1) V(wakeup); //当只有一个顾客刚来，必须唤醒理发师

else P(wait); //如果不是，顾客在wait上等待

V(mutex); //释放计数器

理发;

//理发师进程

P(wakeup); //理发师睡觉

while(rc!=0)

{

理发();

P(mutex); //访问计数器

rc--;

if(rc!=0) V(wait); //释放等待的顾客，让其来理发

V(mutex) //释放计数器

}

3-19

2和4会发生死锁

3-20

（1）计算第5次分配后进程的状态和占用资源情况

1.P1申请3个，可以满足，系统还剩7个；

2.P2申请2个，可以满足，（因为系统的7个可以使P2运行完）系统还剩5个；

3.P3申请4个，若满足它的请求，可能使以后的任何进程都不能运行完，故P3等待；

4.P1申请2个，满足，（系统还剩5个可以满足P1的最大请求），系统还剩3个；

5.P2申请2个，不能满足，等待。

此时系统的分配情况为：

P1分配5个资源正在运行；P2分配2个资源，等待分配2个资源；P3等待分配4个资源；系统还剩3个资源

1. P1的请求最先完成，进程完成序列：{P1，P2，P3}

3-21

（1）Need矩阵

（2）当前系统处于安全状态

1. 因系统剩余可用向量为（1502），p2的剩余请求向量为（1002），即（1502）>（1002），能满足