**[PV操作](https://zhuanlan.zhihu.com/p/61326272" \o "https://zhuanlan.zhihu.com/p/61326272" \t "_blank)**

[**深秋无痕**](https://www.zhihu.com/people/shen-qiu-wu-hen-66)

**一、什么是PV操作?**

PV操作由P操作原语和V操作原语组成（原语是不可中断的过程），对信号量进行操作，具体定义如下：

**P（S）：**

1. 信号量S的值减1，即S=S-1；
2. ②如果S>0，则该进程继续执行；否则该进程置为等待状态，排入等待队列。

**V（S）：**

①将信号量S的值加1，即S=S+1；

**二、PV操作的意义：**

我们用信号量及PV操作来实现进程的同步和互斥。PV操作属于进程的低级通信。

**三、什么是信号量？**

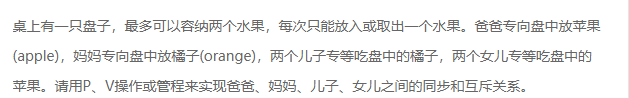
信号量（semaphore）的数据结构为一个值和一个指针，指针指向等待该信号量的下一个进程。 **信号量的值与相应资源的使用情况有关:**①: 当信号量的值大于0时，表示当前可用资源的数量 ②: 当它的值小于0时，其绝对值表示等待使用该资源的进程个数 **注意，信号量的值仅能由PV操作来改变。**

**四、使用PV操作注意事项：**

①: 每个程序中用户实现互斥的P、V操作必须成对出现，先做P操作，进临界区，后做V操作，出临界区。若有多个分支，要认真检查其成对性。 ②: P、V操作应分别紧靠临界区的头尾部，临界区的代码应尽可能短，不能有死循环。 ③: 互斥信号量的初值一般为1。

**五、PV操作实例:**

**5.1 例题1: 吃苹果问题:**



**解题过程:**

semaphore empty**=**2； *//定义empty对应盘子的剩余放水果的位置个数初值为2(空缓冲区个数)*

semaphore apple**=**0； *//定义信号量apple对应盘子里的苹果数量初值为0*

semaphore orange**=**0； *//定义信号量orange对于盘子里的橘子数量初值为0*

semaphore mutex**=**1： *//定义信号量mutex来保护盘子被互斥地访问*

father() { *//爸爸进程*

**while**(1){

P(empty)； *//盘子的剩余放水果的位置减一，如果>=0,说明有位置可以放苹果*

P(mutex)；

在盘子里放一个苹果

V(mutex)；

V(apple)；*//盘中苹果数加一*

}

}

mother() { *//妈妈进程*

**while**(1) {

P(empty)； *//盘子的剩余放水果的位置减一，如果>=0,说明有位置可以放橘子*

P(mutex)； *//互斥变量减一，如果<0，则说明有进程在临界区。则当前进程必须等待。*

在盘子里放一个橘子

V(mutex)； // 进程执行完毕，出了临界区，互斥变量值加一。

V(orange)； *//盘中橘子数加一*

}

}

son() { *//用这段程序产生两个儿子进程*

**while**(1) {

P(orange)； *//盘中橘子个数减一，如果结果>=0时,说明盘中有橘子，可以取*

P(mutex)；

从盘子里拿一个橘子

V(mutex)；

V(empty)； // 取了一个橘子后，盘子的剩余放水果的位置加一

}

}

daughter() { *//用这段程序产生两个女儿进程*

**while**(1 ) {

P(apple)； *//盘中苹果个数减一，如果结果>=0时,说明盘中有苹果，可以取*

P(mutex)；

从盘子里拿出一个苹果

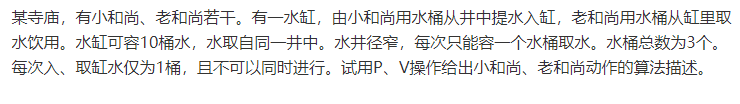
V(mutex)；

V(empty)； // 取了一个橘子后，盘子的剩余放水果的位置加一

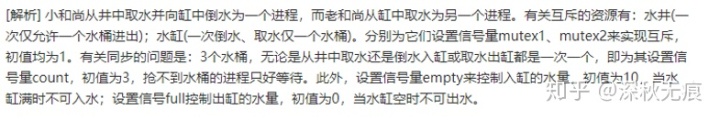
}

}

**5.2 : 和尚打水:**



**分析:**



**解题过程:**

Semaphore mutexA **=** 1; *//保护井被互斥地访问*

Semaphore mutexB **=** 1; *//保护缸被互斥地访问*

Semaphore Count **=** 3； *//桶总数为3*

Semaphore full **=** 0; *//控制出水，刚开始，缸是空的，能取出0桶水*

Semaphore empty **=** 10; *//控制入水，刚开始，缸是空的，能倒入10桶水*

*//小和尚打水：*

Progress（小和尚）{

**while**（1）{

P（empty） *//判断缸是否满了（结果如果>=0,未满），决定是否需要打水*

P（count） *//拿桶（判断是否有空闲的桶，结果如果>=0,有桶）*

P（mutexA）{ *//对井互斥访问*

从井中打水；

V（mutexA） *//互斥*

}

P（mutexB）{ *//对缸互斥访问*

向缸中倒水；

V（mutexB） *//互斥*

}

V（count）*//还桶，空闲桶的数量加一*

V（full） *//缸中水的数量加一，（通知老和尚取水）*

}

}

*//老和尚取水：*

progress（老和尚）{

**while**（1）{

P（full） *//判断缸中是否有水，（结果如果>=0,说明缸中有水，可取水）*

P（count）*//拿桶*

P（mutexB) *//互斥*

从缸中取水；

V（mutexB）*//互斥*

V(count) *//还桶*

V（empty）*//缸中还能容纳的水数量加一，（通知小和尚打水）*

}

}

[发布于 2019-04-03](https://zhuanlan.zhihu.com/p/61326272)

[**PV操作--经典问题分析**](https://blog.csdn.net/y824448940/article/details/53101521)

[sharkshake](https://me.csdn.net/y824448940) 2016-11-09 16:36:05

分类专栏： [操作系统](https://blog.csdn.net/y824448940/category_6443825.html) 文章标签： [操作系统](https://www.csdn.net/gather_22/MtTacg0sNTgzNi1ibG9n.html)

为了提高在实际问题中分析和思考多个线程之间同步互斥问题的能力，接下来将讲解PV操作，这也是[操作系统](http://lib.csdn.net/base/operatingsystem)中的重点和难点。本文将会先简要介绍下PV操作的来源和基本使用方法，然后再通过两道经典的计算机考研真题——放水果和安全岛来示范如何运用PV操作。

先讲讲PV操作的起源和用法。

1962年，荷兰学者Dijksrta在参与X8计算机的开发中设计并实现了具有多道程序运行能力的操作系统——THE Multiprogramming System。为了解决这个操作系统中进程（线程）的同步与互斥问题，他巧妙地利用火车运行控制系统中的“信号灯”（semaphore，或叫“信号量”）概念加以解决。信号量的值大于0时，表示当前可用资源的数量；当它的值小于0时，其绝对值表示等待使用该资源的进程个数。注意，这个信号量的值仅能由PV操作来改变。

PV操作由P操作原语和V操作原语组成（原语也叫原子操作Atomic Operation，是不可中断的过程），对信号量（注意不要和Windows中的[信号量机制](http://blog.csdn.net/morewindows/article/details/7481609)相混淆）进行操作，具体定义如下：

P(S)：

①将信号量S的值减1，即S=S-1；

②如果S>=0，则该进程继续执行；否则该进程置为等待状态。

V(S)：

①将信号量S的值加1，即S=S+1；

②该进程继续执行；如果该信号的等待队列中有等待进程就唤醒一等待进程。

用PV操作实现多线程的同步与互斥是非常简单的，只要考虑逻辑处理上合理严密而不用考虑具体技术细节，因此与写伪代码较为相似。比如有多个进程P1、P2、 ……PN。它们要互斥的访问一个资源。用PV操作来实现就非常方便直观。下面是PV操作代码：

设置信号量为S，初值为1。各进程的操作流程如下：

进程P1              进程P2           ……          进程Pn

P（S）；              P（S）；                           P（S）；

访问资源；         访问资源；                      访问资源；

V（S）；             V（S）；                          V（S）；

**可以看出PV操作会忽略具体的编程细节，让程序员的主要精力放在线程同步互斥的逻辑处理上。因此，通过练习PV操作能快速有效提高程序员对多线程的逻辑思维能力，达到强化“内功”的目的**。

接下来就来几道简单的计算机考研真题。

**第一题 放水果 南京大学计算机考研真题**

桌上有一空盘，允许存放一只水果。爸爸可向盘中放苹果，也可向盘中放桔子，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一只水果供吃者取用，请用P、V原语实现爸爸、儿子、女儿三个并发进程的同步。

这个题目涉及的东西非常之多，光人物就有三个再加水果，盘子等等，确实让人感觉好像无从下手。但**不管题目如何变，只要牢牢的抓住同步和互斥来分析问题就必定能迎刃而解。**

下面先考虑同步情况即所有“等待”情况：

第一．爸爸要等待盘子为空。

第二．儿子要等待盘中水果是桔子。

第三．女儿要等待盘中水果是苹果。

接下来来考虑要互斥处理的资源，看起来盘子好像是要作互斥处理的，但由于题目中的爸爸、儿子、女儿均只有一个，并且他们访问盘子的条件都不一样，所以他们根本不会同时去访问盘子，因此盘子也就不用作互斥处理了。分析至些，这个题目已经没有难度了，下面用PV原语给出答案：

先设置三个信号量，信号量Orange表示盘中有桔子，初值为0。信号量Apple表示盘中有苹果，初值为0。信号量EmptyDish表示盘子为空，初值为1。三个人的操作流程如下所示：

1．爸爸

P(EmptyDish)

if (rand()%2==0)

{

    放桔子

    V(Orange)

}

else

{

    放苹果

    V(Apple)

}

2．儿子

P(Orange)

取桔子

V(EmptyDish)

3．女儿

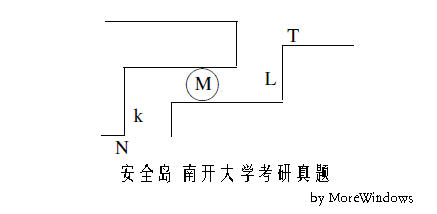
P(Apple)

取苹果

V(EmptyDish)

**第二题 安全岛 南开大学考研真题**

在南开大学至天津大学间有一条弯曲的路，每次只允许一辆自行车通过，但中间有小的安全岛M（同时允许两辆车），可供两辆车在已进入两端小车错车，设计[**算法**](http://lib.csdn.net/base/datastructure)并使用P，V实现。



这个问题应该如何考虑了？同样**只要牢牢的抓住同步和互斥来分析问题就必定能迎刃而解。**

考虑所有“等待”情况：

进入通道之前都需要等待在岛上的位置，有位置才可以进入(同步)

互斥：两个路段都需要互斥

用信号量M来记录安全岛M上空位个数，初值为2。每个进入道路前的人都要先预订安全岛上的空位，订到后再互斥的进入道路。否则就要等待安全岛上有空位。信号量K和L表示道路，初值均为1。然后从N到T的车和从T到N的车的行驶流程如下：

**从N到T的车                     从T到N的车**

P(M)                 P(M)

P(K)                 P(L)

由N到M               由T到M

V(K)                 V(L)

P(L)                 P(K)

V(M)                 V(M)

由M到T               由M到T

V(L)                 V(K)

这种解决方法也是不会造成死锁的。安全岛的解法非常之多，网上还有不少不同的解法，有兴趣的童鞋可以搜索一下。

关于PV操作的总结：

1.在多线程编程中使用PV操作能忽略代码具体的实现细节，这样确保总体思路的正确性。

2.分析PV操作时紧紧抓住同步和互斥；

同步主要分析：谁在等待？等待什么？分析清楚了，同步就出来了；

互斥主要分析：那些资源是共享的，共享资源需进一步思考有没有必要互斥，比如果盘问题中果盘是共享资源，但是果盘操作并不需要互斥；

[**PV操作经典例题——和尚打水**](https://blog.csdn.net/Yun_Ge/article/details/89043419)

[Yun\_Ge](https://me.csdn.net/Yun_Ge) 2019-04-05 14:53:40

分类专栏： [操作系统](https://blog.csdn.net/yun_ge/category_8829160.html) 文章标签： [和尚打水](https://so.csdn.net/so/search/s.do?q=%E5%92%8C%E5%B0%9A%E6%89%93%E6%B0%B4&t=blog&o=vip&s=&l=&f=&viparticle=)

**和尚打水问题**

某寺庙,有小和尚和老和尚若干,有一个水缸,由小和尚提水入缸供老和尚饮用.水缸可以容纳10桶水,水取自同一口井中,由于水井口窄,每次只能容纳一个水桶取水.水桶总数为3个。每次入水、取水仅为一桶,且不可同时进行。试给出有关取水、入水的PV算法描述。  
分析：

* 由题可知，水井窄，只能容纳一人使用，所以需设一个互斥信号量mutex1来代表井
* 每次入水取水不可同时进行，所以需设一个互斥信号量mutex2来代表缸
* 桶为资源信号量，设为count。
* 缸中水的数量和老和尚小和尚都有关，需设置两个不同的信号量，用empty资源信号量来告知小和尚是否可以提水入缸。
* 用full资源信号量来告知老和尚是否可以从缸中取水。

设：mutex1=1, mutex2=1, count=3, empty=10, full=0;

void little\_monk( )

{

while(true)

{

wait(empty);

wait(count);

wait(mutex1);

//此期间为取水阶段

signal(mutex1);

wait(mutex2);

//此期间为倒水入缸阶段

signal(mutex2);

signal(count);//还桶

signal(full);//发消息告知老和尚缸里有水

}

}

void old\_monk( )

{

while(true)

{

wait(full);

wait(count);

wait(mutex2);

//此期间为从缸中取水阶段

signal(mutex2);

signal(count);//还桶

signal(empty);//发消息告知小和尚可以取水了

}

}

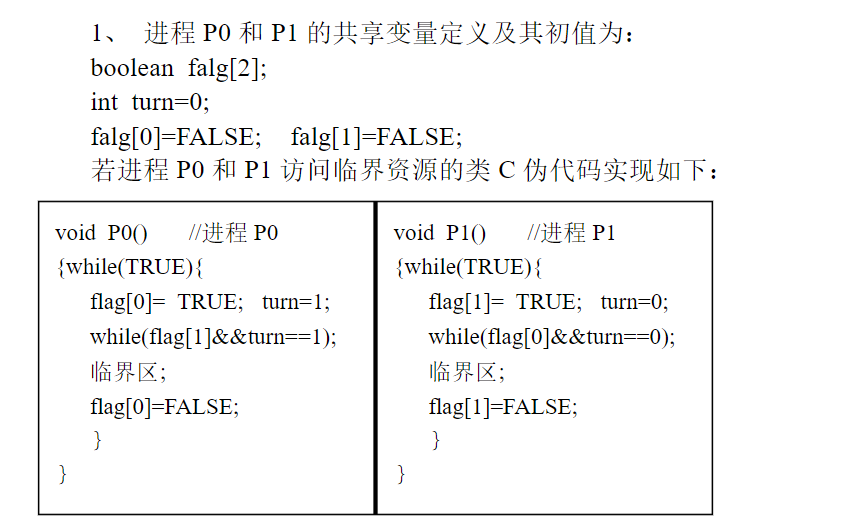
[**PV操作经典习题**](https://www.jianshu.com/p/c54923eaad55)

[萍水间人](https://www.jianshu.com/u/7b4cb9cdc0c8)

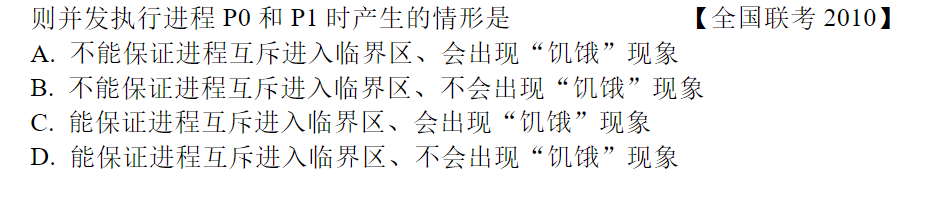
0.25 2019.04.12 07:52:07字数 599

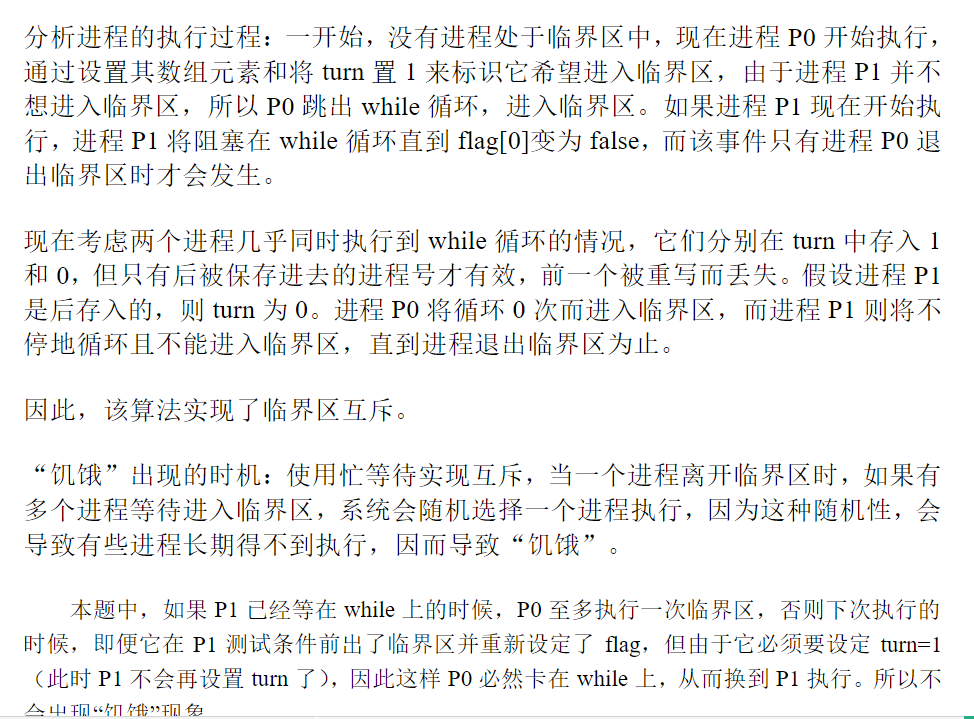
好经典的一些问题

**第一题**



显然这个代码完美的解决了进程间同步与互斥的问题





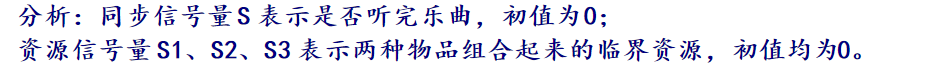
所以答案选D

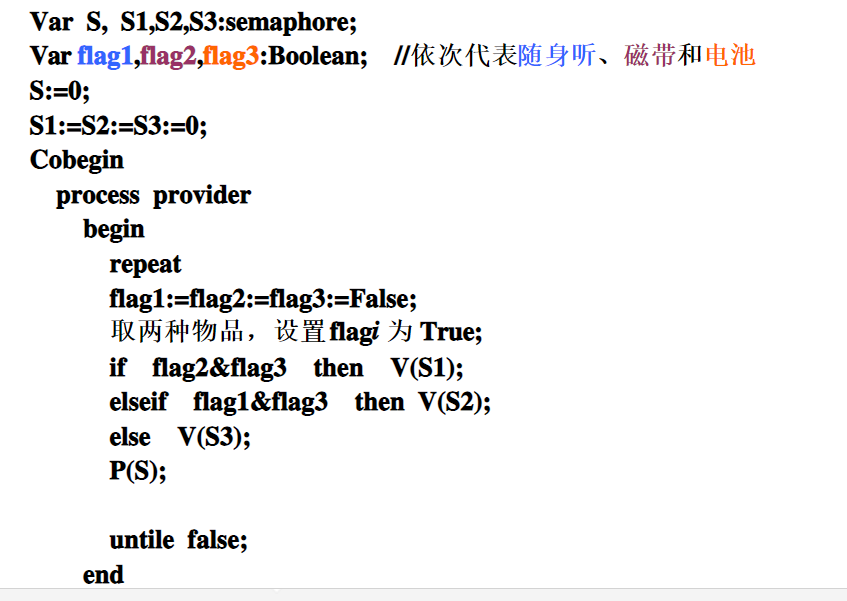
**第二题**

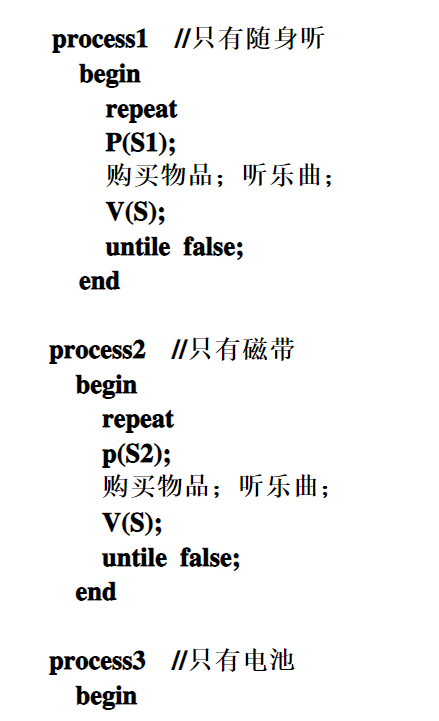
一间酒吧中有3个音乐爱好者， 一个有随身听， 一个有电池， 一个有音乐磁带。 酒吧老板一次出售这三种物品中的任意两种。 当一名音乐爱好者得到三样东西并且听完之后， 酒吧老板才能再一次出售这三种物品。

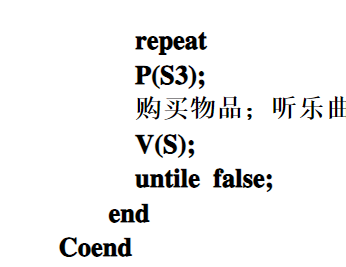
使用PV操作解决。

woc我还在想老板怎么解决卖个谁的问题， 没想到这里直接任取两件物品。。 然后符合条件的就卖出去









**第三题**

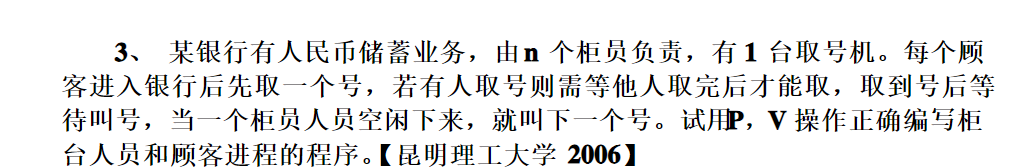
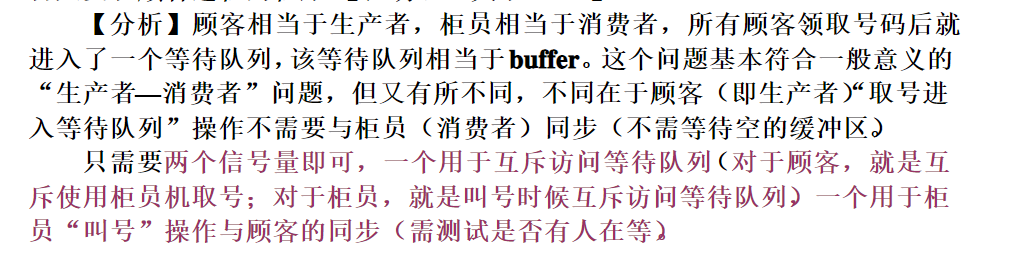
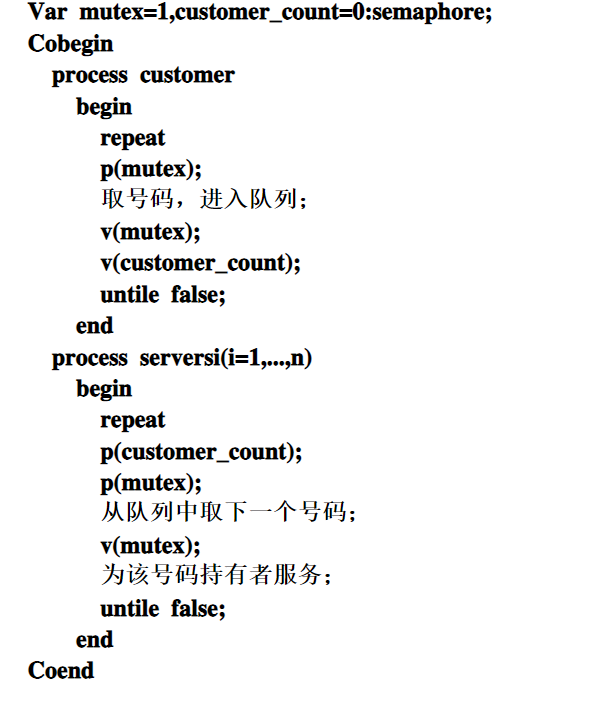
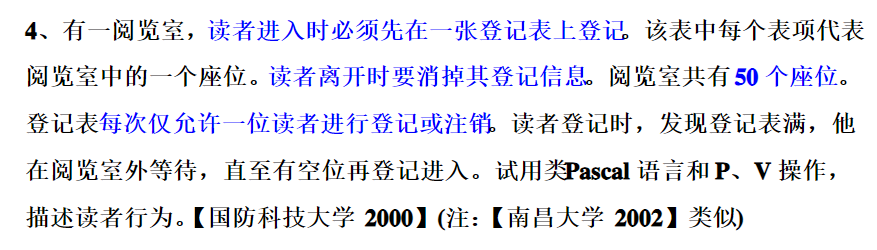


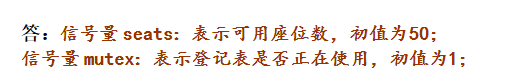
image.png

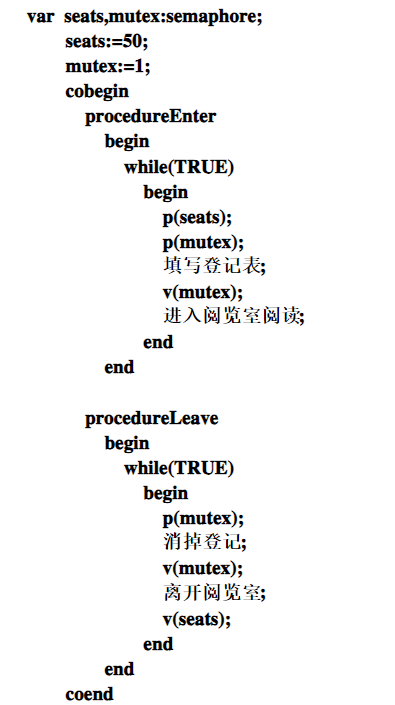




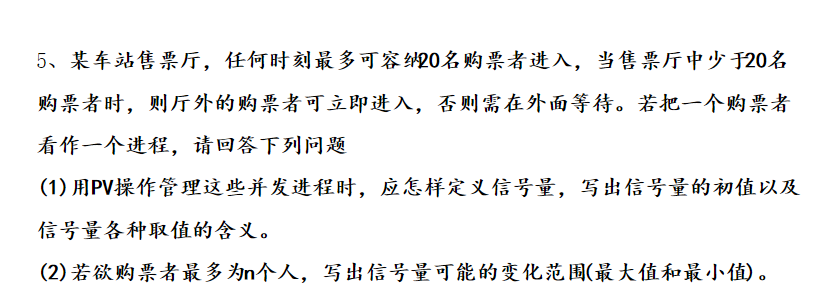
**第四题**

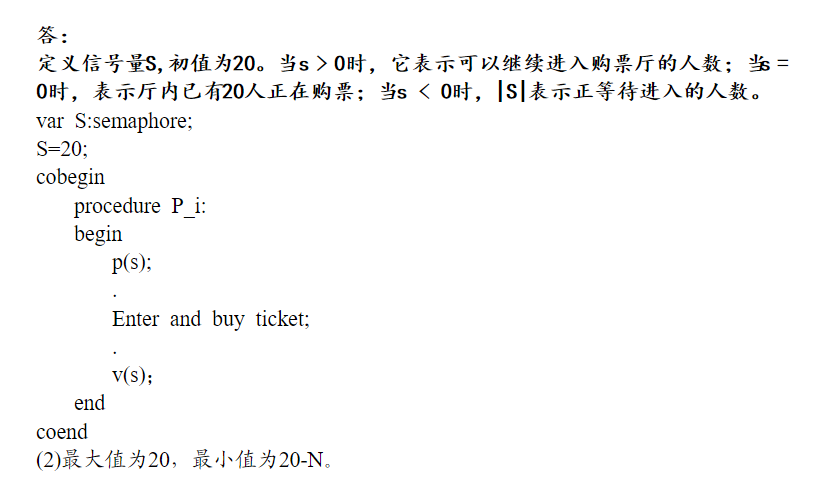






**第五题**





**第六题**

考试的一道题，虽然考前的一张试卷上出现了，但是我却没有看到，有点桑心

三个工人，一个生产车轮，一个生产车架，一个将一个车架与两个车轮组装起来。

原始的题目给的代码如下：

//工人1

生产车架

P(empty)

将车架放入工作台

V(frame)

//工人2

生产车轮

P(empty)

将车轮放入工作台

V(wheel)

//工人三

P(frame)

取一个车架

V(empty)

P(wheel)

P(wheel)

取两个车轮

V(empty)

V(empty)

组装一台车

这段代码十存在死锁问题的，因为如果一个工人连续生产车轮或者车架，直到占满了工作区，这就导致三个工人无法继续工作

所以修改代码如下：

//工人1

生产车架

P(s1)

P(empty)

将车架放入工作台

V(frame)

//工人2

生产车轮

P(s2)

P(empty)

将车轮放入工作台

V(wheel)

//工人三

P(frame)

取一个车架

V(empty)

P(wheel)

P(wheel)

取两个车轮

V(empty)

V(empty)

V(s1)

V(s2)

组装一台车

车架的数量不超过N-1  
车轮的数量不超过N-2

也就是s1 = N-1,s2 = N-2

通过这种方式来限制工人的生产数量。  
而我考试的时候是对生产数量进行计数，而且只想到了对一个工人进行限制，也不知道这种方式行不行

看天命吧

总结了一下考试的经验：  
大学的考试从道理上来讲不算考试，而是叫猜题划重点。  
从大一走来，从不理解为什么那么看重划重点确定哪些不考一步步过来，越来越觉得大学的考试很多时候在于你对信息的掌握，虽然很多东西懂了，但是它不考你很难受，很多东西你不懂但是偏偏要考

对于自己的能够掌握到的复习资料，一定要好好地利用。

暂时想到这么多

[操作系统 – 使用信号量实现N过程屏障](http://www.voidcn.com/article/p-ahytllps-bte.html)

时间 2018-08-14

原文 <http://stackoverflow.com/questions/6331301/implementing-an-n-process-barrier-using-semaphores>

我目前正在进行以前迭代的OS考试培训，我遇到这个问题：

Implement a “N Process Barrier”, that is, making sure that each process out of a group of them waits, at some point in its respective execution, for the other processes to reach their given point.

You have the following ops available:

init(sem,value), wait(sem) and signal(sem)

N是任意数。我可以使它适用于给定数量的进程，但不适用于任何数量。

有任何想法吗？可以用伪代码回复，这不是一个任务，只是个人学习。

这在 [The Little Book of Semaphores](https://greenteapress.com/semaphores/LittleBookOfSemaphores.pdf) 做的很好。

n = the number of threads

count = 0

mutex = Semaphore(1)

barrier = Semaphore(0)

mutex.wait()

count = count + 1

mutex.signal()

if count == n: barrier.signal() # unblock ONE thread

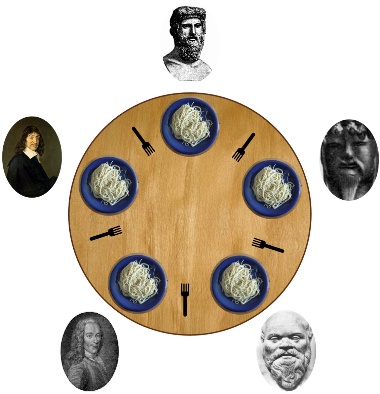
barrier.wait()

barrier.signal() # once we are unblocked, it's our duty to unblock the next thread

[**哲学家进餐问题**](https://mp.weixin.qq.com/s/Cz_jgWor8TKvbtpnYip3Yg)

原创 lhw 操作系统学习 2020-11-11

1965年，Dijkstra提出并解决了一个他称之为哲学家进餐的同步问题。该问题描述如下：五位哲学家围坐在一张圆桌旁，每位哲学面前都有一盘通心粉，相邻两个盘子之间放有一把叉子。每位哲学家的行为是讨论与思考，感到饥饿，然后吃通心粉。由于通心粉很滑，需要两把叉子才能夹住，因此，为了吃通心粉，每位哲学家必须拿到两把叉子，并且每个人只能直接从自己的左边或右边取叉子。



图片来源于网络

该问题最简单的算法描述如下。

/\*信号量数组，初值均为1，表示叉子空闲。\*/

SEMAPHORE fork[5] = {1,1,1,1,1}；

main()

{

创建5哲学家进程Philosopher(i);/\*i表示第i个哲学家\*/

cobegin

/\* 5个哲学家进程并发执行\*/

Philosopher(0)；

Philosopher(1)；

Philosopher(2)；

Philosopher(3)；

Philosopher(4)；

coend

}

Philosopher(int i) { // 第i为哲学家

while (1) {

讨论与思考；

感到饥饿；

P(fork[i])；/\* 取第一把叉子(左边) \*/

P(fork[(i+1) % 5])；/\* 取第二把叉子 (右边)\*/

进食；

V(fork[i])；/\* 释放第一把叉子 \*/

V(fork[(i+1) % 5])；/\* 释放第二把叉子 \*/

}

}

该算法虽然简单，但存在死锁。当五位哲学家都感到饥饿时，都各自取了自己左边的叉子后，谁也无法拿到他右边的叉子，致使谁也无法进餐。为了防止这种现象的出现可采取以下措施：

(1) 最多允许4位哲学家同时取叉子，或者在某位哲学家左边放2把叉子，这样可保证5位哲学家中有一位能取到两把叉子；

(2) 这5位哲学家中的主持人，比如4号哲学家，先取他右边的叉子，然后再取他左边的叉子，其他哲学家取叉子的顺序不变。也可以这样描述：给所有叉子编号，要求哲学家进餐时先取他左、右编号较小的叉子，然后再取另一把叉子，即按死锁的预防一节中介绍的资源有序申请策略申请资源以避免死锁；

(3) 仅当一位哲学家左右两边的叉子都可用时，才允许他取叉子。



2002年的Dijkstra 图片来源于网络

1. 算法1：最多允许4位哲学家同时取叉子

在原算法的基础上增加一个信号量Limit，其初值为4，每位哲学家在用餐前先判断一下是否其他4位哲学家也将要就餐。

/\*信号量数组，初值均为1，表示叉子空闲。\*/

SEMAPHORE fork[5] = {1,1,1,1,1}；

/\*限制最多有4位哲学家可以同时\*/

SEMAPHORE Limit=4；

Philosopher(int i) {

while (1) {

讨论与思考；

感到饥饿；

P(Limit)；

P(fork[i])；/\* 取第一把叉子(左边) \*/

P(fork[(i+1) % 5])；/\* 取第二把叉子 (右边)\*/

进食；

V(fork[i])；/\* 释放第一把叉子 \*/

V(fork[(i+1) % 5])；/\* 释放第二把叉子 \*/

V(Limit)；

}

}

2. 算法2：4号哲学家取叉子的顺序与其他哲学家正好相反。

/\*信号量数组，初值均为1，表示叉子空闲。\*/

SEMAPHORE fork[5] = {1,1,1,1,1}；

Philosopher(int i) {

while (1) {

讨论与思考；

感到饥饿；

if (i==4) {

P(fork[(i+1) % 5])；/\* 取第一把叉子 (右边)\*/

P(fork[i])；/\* 取第二把叉子(左边) \*/

} else {

P(fork[i])；/\* 取第一把叉子(左边) \*/

P(fork[(i+1) % 5])；/\* 取第二把叉子 (右边)\*/

}

进食；

V(fork[i])；/\* 释放第一把叉子 \*/

V(fork[(i+1) % 5])；/\* 释放第二把叉子 \*/

}

}

3. 算法3：仅当一个哲学家左右两边的叉子都可用时，才允许他拿叉子

当哲学家饥饿时查看两边的叉子是否空闲，如空闲立即将两把叉子拿起，然后用餐。算法描述如下。

/\* 指哲学家的状态，初值为“讨论与思考”\*/

Status S[5]={讨论与思考，讨论与思考，讨论与思考，

讨论与思考，讨论与思考}；

/\* 每位哲学家一个信号量，其初值都为0 \*/

SEMAPHORE Ph[5]={0，0，0，0，0}；

/\* 互斥信号量，实现对状态变量 S 的互斥访问 \*/

SEMAPHORE mutex=1；

void test(int i) {

if ((S[i]== 饥饿)&&(S[(i+4)%5]!=进食)

&&(S[(i+1)%5]!=进食)) {

/\* 哲学家i满足进餐条件，其状态改变为进食状态 \*/

S[i]=进食；

/\* 执行V操作，使得哲学家i申请进餐时可进入 \*/

V(Ph[i])；

}

}

void philosopher(int i) {

while(1) {

讨论与思考；

P(mutex)；/\* 对临界资源 S互斥地使用 \*/

S[i]=饥饿；

test(i)；/\* 查哲学家i两边的叉子是否空闲\*/

V(mutex)；

P(ph[i])；/\* 若不满足进餐条件则阻塞，等待其他

哲学家放下叉子并满足进餐条件时唤醒他。\*/

进食；

P(mutex)； /\* 对临界资源S互斥地使用 \*/

S[i]=讨论与思考；

/\* 判断其左边的哲学家是否饥饿,若是且满足

进餐条件则唤醒他。\*/

test((i+4)%5)；

/\* 判断其右边的哲学家是否饥饿,若是且满足

进餐条件则唤醒他。\*/

test((i+1)%5)；

v(mutex)；

}

}

信号量ph[i]表示哲学家i是否具备进餐的条件，初值为0，表示不具备，其值为1表示可以进餐。S[]存储哲学家的状态，是临界资源。函数test(int i)用来判断第i个哲学家是否可以进餐，当该哲学家状态为“饥饿”且其左右哲学家都不在进餐状态时，其状态可改为“进食”，且对p[i]执行V操作，即哲学家i可以进餐了。当哲学家i进餐完毕后，他会查看其左右哲学家是否需要进餐，如果需要且条件成立，立即让其进餐。

4. 2019年一道与哲学家相关的考研题目

43.（8分）有n（n>=3)位哲学家围坐在一张圆桌边，每位哲学家交替地就餐和思考。在圆桌中心有m（m>=1）个碗，每两位哲学家之间有1根筷子。每位哲学家必须取到一个碗和两侧的筷子之后，才能就餐，进餐完毕，将碗和筷子放回原位，并继续思考。为使尽可能多的哲学家同时就餐，且防止出现死锁现象，请使用信号量的P、V操作(wait()、signal()操作）描述上述过程中的互斥与同步，并说明所用信号量及初值的含义。

答案要点：

//信号量bowl用于协调哲学家对碗的使用

// bowl<n，确保不死锁，即至少保证有一个

// 人拿不到碗，那么就能保证至少有一个人

// 能拿到左右的筷子。

Semaphone bowl=min(n-1,m);

//信号量chopsticks用于协调哲学家对筷子的使用

//两个哲学家之间的筷子数量为1

Semaphone chopsticks[n]={1,1,...,1};

CoBegin

Philosopher(0)；

Philosopher(1)；

……

Philosopher(n-1)；

CoEnd

Philosopher(int i) //哲学家i的程序

{

while(true) {

思考；

P(bowl); //取碗

P(chopsticks[i]); // 取左边的筷子

P(chopsticks[(i+1)%n]); // 取右边的筷子

就餐;

V(chopsticks[i]); // 放下左边的筷子

V(chopsticks[(i+1)%n]); // 放下右边的筷子

V(bowl); //放下碗

}

}

[**答疑（进程同步、互斥、死锁）之一**](https://mp.weixin.qq.com/s/RfO_zFxRS4cU0ss9Swua-w)

**原创 lhw 操作系统学习 2021-3-29**

**问题2**：有两个并发进程P1、P2，它们共享同一个变量x，其程序代码如下：

P1( ) {

     x=1;

     y=2;

     if(x>0) z=x+y;

     else z=x\*y;

     print z;

}

P2( ) {

      x=-1;

      a=x+3;

      x=a+x;

      b=a+x;

      c=b\*b;

      print c;

}

问：(1)可能打印出的z值是哪些？（假设每条赋值语句是一个原子操作）

(2)可能打印出的c值有哪些？

答：我们先来分析z的输出值。进程P2的前3行代码会影响进程P1的输出结果。下面具体分析。

(1) 分析可能打印出的z值。  
1) 把P2的前3行代码看作整体，则不论这3行代码在P1的何处执行，输出z的值都为3；2) 如果P2的x=-1在if(x>0) 之前执行，则z=2\*x; 此处的x的可能是-1，也可能是1（a=x+3;x=a+x;执行后），所以输出z的值可能是-2或2；

3) 如果P2的x=-1在if(x>0) 之后z=x+y之前执行，则输出z=1；在z=x+y之后执行，则输出2；

4) 如果P1的x=1在P2的x=-1之后a=x+3之前执行，P2的x=a+x在if(x>0)之前执行，则输出7；

5) 如果P1的x=1在P2的a=x+3之后x=a+x之前执行，P2的x=a+x在if(x>0)之前执行，则输出5。所以可能打印出z的值有-2，1，2，3， 5，7.

(2) 分析可能打印的c值有哪些。c的值主要受a和x值的影响。在执行P2的b=a+x之前，a的取值可能是2或4，当a的值为2时，x的值可能为1或3，当a的值为4时，x的值为5，所以b的可能取值为3，5，9，c的输出值可能为9，25，81.

**问题4**：（考研题目）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，通过叫号选取一位顾客，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

{

从取号机获取一个号码；

等待叫号；

获取服务；

}

process 营业员

{

while(TRUE)

{

叫号；

为客户服务；

}

}

}

coend

请添加必要的信号量和P、V(或 wait()、signal())操作，实现上述过程中的互斥与同步。要求写出完整的过程，说明信号量的含义并赋初值。

解：互斥资源：取号机(一次只允许一位顾客领号)，因此设一个互斥信号量mutex；

同步问题：顾客需要获得空座位等待叫号，当营业员空闲时，将选取一位顾客并为其服务。空座位的有、无影响等待顾客数量，顾客的有、无决定了营业员是否能开始服务，故分别设置信号量empty和full来实现这一同步关系。另外，顾客获得空座位后，需要等待叫号和被服务。这样，顾客与营业员就服务何时开始又构成了一个同步关系，定义信号量service来完成这一同步过程。

semaphore mutex=1； //互斥使用取号机

semaphore empty=10； //空座位的数量

semaphore full=0； //已占座位的数量

semaphore service=0； //等待叫号

cobegin

{

process顾客i

{

P(empty)；

P(mutex)；

从取号机获得一个号；

V(mutex)；

V(full)；

P(service)； //等待叫号

获得服务；

}

process营业员

{

while(TRUE)

{

P(full)；

V(empty)；

V(service)； //叫号

为顾客服务；

}

}

}

coend

**问题5：**（考研题目）面包师有很多面包，由n个销售人员推销。每个顾客进店后取一个号，并且等待叫号，当一个销售人员空闲下来时，就叫下一个号。试设计一个使销售人员和顾客同步的算法。

分析：顾客进店后按序取号，并等待叫号；销售人员空闲之后也是按序叫号，并销售面包。因此同步算法涉及到顾客与销售人员的同步，以及取号和叫号的互斥问题。我们使用两个变量i和j分别记录当前的取号值和叫号值，并各自使用一个互斥信号量用于对i和j的进行访问和修改，使用buyer和seller两个信号量对顾客和销售人员之间的同步。

int i=0; // 取的号码

int j=0; // 叫的号码

semaphore mutex\_i=1; // 共享变量i互斥

semaphore mutex\_j=1; // 共享变量j互斥

semaphore buyer=0; // 顾客人数

semaphore seller=n; // 销售人数

Consumer(){ // 顾客

进入面包店;

p(mutex\_i); // 取号

i++;

V(mutex\_i); // 取号结束

V(buyer);

P(seller);

买面包;

离开;

}

Sell() { //销售

while(1) {

P(buyer);

P(mutex\_j); //互斥访问j

j++;

叫编号为j的顾客

V(mutex\_j); //释放对j的访问

卖面包给顾客j;

顾客离开;

V(seller);

}

}

**问题6**. 一个生产进程和两个消费进程共享一个单缓冲区，缓冲区只能放一条消息，生产进程生产一条消息并向缓冲区写入，两个消费进程都要从缓冲区中取出这条消息消费，此过程循环往复，用信号量机制解决这三个进程间的同步问题。解：该问题是“生产者—消费者”问题的变种，假设生产进程为P，两个消费者进程分别为C1和C2，缓冲区为B．这3个进程间存在以下的同步关系：

(1) 进程P与进程C1和C2存在同步关系，当缓冲区B空闲时，进程P往缓冲区中写一条信息，当缓冲区不空时，C1和C2竞争读缓冲区B中的信息。当B不空时，P等待，当B空时，C1和C2等待。

(2) 进程C1和C2要等待P往缓冲区B里放入一条消息后，竞争读该消息。

(3) 3个进程都要互斥地使用缓冲区B。但由于使用同步信号量可以避免三个进程同时使用缓冲区，因此，无需使用互斥信号量。因此，仅需设置2个同步信号量即可：一个同步信号量为S0，初值为1，表示缓冲区B初始时为空；一个同步信号量为S1，初值为0，表示缓冲区B初始时无消息。

semaphore S0=1；//同步信号量，表示缓冲区空

semaphore S1=0；//同步信号量，表示缓冲区中的消息数

cobegin

{ // 生产者进程

P( ) {

Lp: P(S0)； // 缓冲区空时，写信息

Write a message to Buffer； // 写信息到缓冲区中

V(S1)； // 缓冲区空时，写信息

goto Lp； // 写下一条信息

}

//消费者C1

C1( ) {

Lc1：P(S1) ; // 缓冲区中有消息吗？

从缓冲区中去除消息消费；

V(S0) ; // 释放缓冲区

goto Lc1 ; // 消费下一条消息

}

//消费者C2

C2( ) {

Lc2：P(S1) ; // 缓冲区中有消息吗？

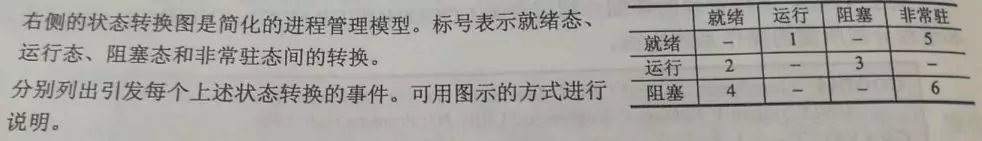
从缓冲区中去除消息消费；

V(S0) ; // 释放缓冲区

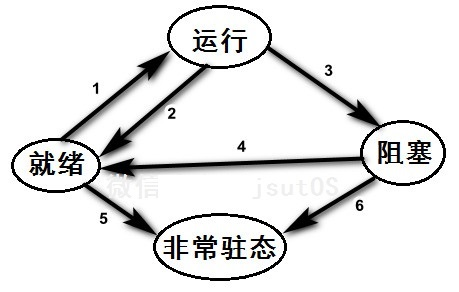
goto Lc2 ; // 消费下一条消息

}

coend

**问题7.**   


这道题目来自《操作系统：精髓与设计原理》，该题出的不怎么样，所以在英文第9版中去掉了这道题，非常驻态类似于挂起状态，当内存不足时，把当前在内存中的一个或多个进程移除内存到辅存中。如果把该题用表表示的状态转换为图来表示的话如下所示。



由图可见，这不是一个完整的状态转换图，缺少从非常驻态到就绪或阻塞的转换，这可能是在第九版中抛弃该题的原因吧。

有了这张图就可以回答本题问题了。

1. 就绪状态到运行状态的转换是由于进程调度程序将CPU分配给该进程。

2. 运行状态到就绪状态转变可能是由于时间片到或被抢先引起的。

3. 运行状态到阻塞状态是由于进程需要进行I/O操作或其他内核请求。

4. 阻塞到就绪是由于进程所等待的事件完成（可能是I/O完成）

5. 就绪状态到非常驻态转换是由于内存不足，系统将该进程所占用的内存中的信息保存在辅存中，把内存空间腾开，让需要的进程使用。

6. 阻塞状态到非常驻态转换与就绪状态到非常驻态转换类似。

**问题8**. 有n个（n为偶数）椅子供人下棋，一盘围棋由两个人来下，下棋者来时需要看一看有没有空椅子。一个服务员，当有两个人到来且都申请到空椅子时，才允许两个人下棋。一个裁判员，当两个下棋者下完棋后，为两个人裁定胜负。两个人下完棋后，交还椅子离去。使用PV操作实现上述过程。

分析： 本题类似理发师问题，但没有说当下棋者看到没有椅子时如何行动，我们假定，下棋者看到没有椅子时等待。

定义变量：

in\_people=0; 表示进入棋盘室还没下棋的棋手人数的奇偶数，0表示偶数，1表示奇数。

out\_people=0; 表示下完棋准备离开棋盘室的棋手人数的奇偶数，0表示偶数，1表示奇数。

定义信号量：

Semaphore Chair=n; //空椅子数，n为偶数

Semaphore Waiter=1；//服务员

Semaphore Referee=1；// 裁判员

Semaphore OpIn=1; // 对in\_people互斥使用

Semaphore OpOut=1; // 对out\_people互斥使用

Semaphore WaitIn=0；//2棋手下棋同步信号量

Semaphore WaitOut=0; //2棋手离开同步信号量

整个过程用PV操作描述如下：

Chess\_player() {

一个棋手走进棋室；

P(Chair); // 申请椅子

P(OpIn); //互斥使用in\_people

in\_people = (in\_people+1) % 2;

if (in\_people == 0) { //偶数, 棋手人数够了，开盘

P(Waiter); // 等待服务员

服务员安排2人下棋;

V(Waiter); //服务员安排完毕

V（WaitIn); // 叫对手过来下棋

V(OpIn); //释放使用in\_people

}

else {

V(OpIn); //释放使用in\_people

P(WaitIn); //等待对手

}

开始下棋；

下棋结束；

P(OpOut); // 互斥使用out\_people

out\_people = (out\_people+1) % 2;

if (out\_people == 0) { // 偶数，将要离开

P(Referee); // 等待裁判员

裁判员裁定胜负;

V(Referee);

V(WaitOut); // 叫对手一块离开

V(OpOut); //释放使用out\_people

}

else {

V(OpOut); //释放使用out\_people

P(WaitOut); //等待对手一块离开

}

V(Chair); // 释放椅子

走出棋盘室;

}

**问题9**：设系统中物理页的数量为150，在T0时刻按表1所示分配给三个进程P1、P2和P3。系统采用银行家算法实施死锁避免策略。对下列内存请求，请分别判断是否安全，如果是安全的，请给出一个可能的进程安全执行序列；如果不是安全的，请说明原因。

(1) 进程P4到达，共需要60个物理页，当前请求分配25页；

(2) 进程P4到达，共需要50个物理页，当前请求分配35页。

|  |  |  |
| --- | --- | --- |
| **表1 T0时刻系统状态** | | |
| 进程 | 最大物理页需求量 | 已分配物理页数量 |
| P1 | 70 | 25 |
| P2 | 60 | 40 |
| P3 | 60 | 45 |

解答：本题不难。按正常思路做即可。

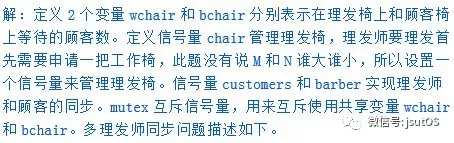
(1) P4到达，需要25个物理页，此时系统中还有40个物理页空闲，如果给P4分配25页，还有15页的剩余，这15页可以分配给P3, P3获得所需要的内存后，会很快运行结束，然后释放60页的物理块，这60页物理块可以分配给P1,P2或P4任一进程，假设分配给P2，P2运行结束释放内存，此时有100个物理块，可满足P1和P4, 系统是安全的，可能的进程执行序列为：P3，P2，P1，P4。注，序列不是唯一的。

(2) P4到达，需要35个物理页，如果给P4分配35页，还有5页的剩余，此时，这5个物理块无法满足任一个进程的需求，导致无进程能正常执行完，系统是不安全的。

[**答疑（进程同步、互斥、死锁）之二**](https://mp.weixin.qq.com/s/seGOMJi1OkyCP_bDIkTeMA)

原创 lhw  操作系统学习  2021-3-29

**题1.** 有m个理发师，n个理发椅和k把等候理发的顾客坐的椅子。如果没有顾客，理发师在休息室；当一个顾客到来时，必须唤醒理发师，进行理发；如果理发师正在理发，又有顾客来到，则如果有空椅子可坐，他就坐下来等，如果没有空椅子，他就离开。为理发师和顾客各编写一段程序描述他们的行为。

  
int wchair=0; /\* 在顾客椅子上等待的顾客 \*/

int bchair=0; /\* 在理发椅上等待的顾客 \*/

semaphore chair=N; /\* 理发椅子数 \*/

semaphore customers=0; /\* 表示等候理发的顾客数，最大值N+K \*/

semaphore barber=M; /\* 理发师人数 \*/

semaphore mutex=1; /\* 互斥信号量，互斥使用wchair和bchair \*/

barber( ) /\* 理发师 \*/

{

while(1) {

P(chair); /\* 理发师要理发，先申请一把工作椅 \*/

P(customers); /\* 若无顾客，理发师睡眠 \*/

cut\_hair； /\* 给顾客理发 \*/

P(mutex)； /\* 理发结束，修改共享变量 \*/

if (wchain>0) {

wchain--；/\* 等候顾客数减一 \*/

}

else {

bchair--; /\* 等候顾客数减一 \*/

}

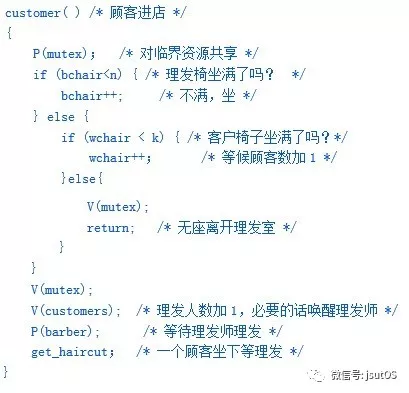
V(mutex)； /\* 释放对临界资源的使用 \*/

V(chair);/\* 释放理发椅 \*/

V(barber)； /\*理发师准备为下一个顾客理发 \*/

}

}



**问题2**. 设有3个进程pa、pb、pc共享一个缓冲区，该缓冲区可存放单个数据。读取一个整数放入缓冲区；进程pb从缓冲区中循环地读出其中的非负数，并求和；进程pc从缓冲区中循环地读出其中的负数，并求和。请使用PV操作，描述实现上述3进程的同步问题。

解答：在本题中，应设置三个信号量S、NonNeg、Neg，信号量S表示缓冲区是否为空，其初值为1；信号量NonNeg表示缓冲区是否为非正数，其初值为0；信号量Neg表示缓冲区是否为负数，其初值为0。同步描述如下：

semaphore S=1;

semaphore NonNeg=0;

semaphore Neg=0;

int Sum\_of\_NonNeg=0,Sum\_of\_neg=0;

pa() {

  while(1){

   从文件中取一个数num;

   if (读文件结束) break;

   P(S);

   将num放入缓冲区中；

   if （num< 0）V(Neg);

   else V(NonNeg);

  }

}

pb( ) {

  while(1){

   P(NonNeg);

   从缓冲区中取出非负数a；

   V(S); // 释放缓冲区

   Sum\_of\_NonNeg = Sum\_of\_NonNeg + a;

  }

}

pc( ) {

  while(1){

   P(Neg);

   从缓冲区中取出负数a；

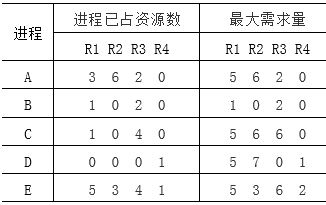
   V(S); // 释放缓冲区

   Sum\_of\_Neg = Sum\_of\_Neg + a;

  }

}

**问题3.**现有5个进程A, B，C，D，E共享R1, R2, R3, R4这四类资源，进程对资源的需求量和目前分配情况如下表。若系统还有剩余资源数分别为R1类2个，R2类6个，R3类2个和R4类1个，请按银行家算法回答下 列问题：(1)目前系统是否处于安全状态？(2)现在如果进程D提出申请(2, 5, 0, 0)个资源，系统是否能为它分 配资源？



解答：(1)系统目前可供分配资源数为(2,6,2,1)，5个进程尚需的资源数分别是A:(2,0,0,0)；B:(0,0,0,0)；C:(4,6,2,0)；D:(5,7,0,0)；E:(0,0,2,1)。剩余资源可满足A，A在有限时间内运行结束，释放资源，可用资源数变为(5,12,4,1)。进程B所需资源已满足，它在有限时间内归还这些资源，可分配资源数为(6,12,6,1)。此时，可用资源可满足C、D、E任一个进程，我们假设依次按C、D、E分配，C获得资源后运行结束，释放资源，可用资源数为(7,12,10,1)。为D分配资源，D运行结束后，释放资源，可用资源数为(7,12,10,2)。为E分配资源，E运行结束后，释放资源，可用资源数为(12,15,14,3)。此刻，系统中的所有进程都能顺利完成，所以，系统是安全的。

(2)若此时给进程D分配(2,5,0,0)个资源，系统剩余的可用资源为(0,1,2,1)。而进程B所需资源已满足，它在有限时间内归还这些资源，可分配资源数为(1,1,4,1)，此刻：A需要（2,0,0,0），C需要（4,6,2,0）,D需要（3,2,0,0）,E需要（0,0,2,1）,剩余资源可满足E。E结束释放资源，可用资源数为（6,4,8,2），此时，剩余资源可满足A和D。A需要（2,0,0,0）,D需要（3,2,0,0）,假设将资源分配给A和D，A和D在有限时间内完成任务释放资源，此时系统中的可用资源数为(9,10,10,3)，此时可用资源数满足C。C获得资源后继续运行，结束后释放资源。所有进程都能正常运行，系统是安全的。所以，系统可以按D的请求为其分配资源。

**问题4.**一对夫妇养了四只宠物，其中有两只狗和两只猫，他们都喜欢吃巧克力。房间里有一个碗，最多能装两块巧克力。宠物正在等着吃巧克力。

(1) 每次只能放入一块巧克力(丈夫或妻子), ——每次只能放进一块。

(2) 每次只能从碗里拿出一块巧克力(被猫或狗吃掉), ——每次只能拿出一块。

(3) 人把巧克力放进碗里和宠物吃巧克力不能同时发生, ——放进和拿出不能同时发生。

(4) 一只宠物一次只能吃一块巧克力, 多只宠物不能同时吃巧克力, ——在同一时间,只能有一种宠物吃一块。

(5) 一次只能一人将巧克力放入碗中(夫妻不能同时将巧克力放入碗中), ——在同一时间, 只能有一个人放进一块。

请用PV原语写出同步算法。

解答：本题属于典型的生产者与消费者问题，按生产者与消费者问题解决即可。

SEMAPHORE full=0； /\*碗中的巧克力数目 \*/

SEMAPHORE empty=2；/\*碗中可装巧克力数目 \*/

SEMAPHORE mutex=1；/\*对碗使用的互斥信号量 \*/

int in=0, out=0; /\* 碗中可装2块巧克力，指示放取巧克力位置 \*/

BUFFER bowl; /\* 定义的碗 \*/

husband() /\* 丈夫 \*/

{

while (1) {

想起往碗里放巧克力了;

P(empty); /\*查看碗里有没有空地方\*/

P(mutex); /\*有空地，准备放巧克力\*/

将一块巧克力放入缓冲区Buffer(in);

in = ( in +1) % 2; /\*计算下次放到位置\*/

V(mutex);

V(full); /\*通知猫狗可以来吃巧克力了\*/

}

}

wife() /\* 妻子 \*/

{

while (1) {

想起往碗里放巧克力了;

P(empty); /\*查看碗里有没有空地方\*/

P(mutex); /\*有空地，准备放巧克力\*/

将一块巧克力放入缓冲区Buffer(in);

in = ( in +1) % 2; /\*计算下次放到位置\*/

V(mutex);

V(full); /\*通知猫狗可以来吃巧克力了\*/

}

}

cat( ) /\* 猫吃巧克力 \*/

{

while (1) {

猫玩累了，想吃巧克力;

P(full);/\*有巧克力吗？\*/

P(mutex);/\*有巧克力，猫要吃\*/

从碗中Buffer(out)取一个巧克力;

out = (out + 1) % n; /\* 计算下一块巧克力位置\*/

吃巧克力;

V(mutex)；

V(empty)；

}

}

dog()

{

while (1) {

狗玩累了，想吃巧克力;

P(full);/\*有巧克力吗？\*/

P(mutex);/\*有巧克力，狗要吃\*/

从碗中Buffer(out)取一个巧克力;

out = (out + 1) % n; /\* 计算下一块巧克力位置\*/

吃巧克力;

V(mutex)；

V(empty)；

}

}

**问题5.**爸爸和妈妈往篮子里分别放苹果和梨，每次只能放一个。苹果数-梨数<m，梨数-苹果数<n，m,n是都正整数。  
 (1)这两个进程之间是否存在互斥关系，若存在，说明互斥访问的临界资源是什么？设定几个信号量可保证互斥关系的实现？

(2)这两个进程之间是否存在同步关系，若存在，设定几个同步信号量实现同步，每个信号量代表的含义是什么？

(3)用P/V原语操作写出这两个进程的代码。

解答：(1)“爸爸和妈妈往篮子里分别放苹果和梨，每次只能放一个。”这句话说明了爸爸每次只能放一个苹果，妈妈每次只能放一个梨，没有说两个人不可同时放，因此篮子不是临界资源。设apple为爸爸往篮子里放的苹果数，初值为0，peer为妈妈往篮子里放的梨数，初值为0，本题没有说篮子里能放多少个苹果和梨，我们假定篮子是无限制空间，想放多少就能放多少。爸爸妈妈都要往篮子里放苹果和梨，但有个条件，apple-peer<m，peer-apple<n，即-n<apple-peer<m,也就是说爸爸妈妈往篮子里放苹果和梨时需要计算，这样的话apple和peer这两个变量成为临界资源，需要设定1个信号量来保证互斥关系的实现。

(2)因为爸爸妈妈放苹果和梨要满足条件：-n<apple-peer<m,所以这两个进程之间存在同步关系，当apple-peer=m-1时，爸爸不能往篮子里放苹果；当apple-peer=1-n时，妈妈不能往篮子里放梨。我们设置两个信号量putApple,putPee实现爸爸妈妈之间的同步，这两个信号量的初值均为0，当apple-peer=m-1时，爸爸往篮子里放苹果时，爸爸对putApple执行P原语操作阻塞；当apple-peer=1-n时，妈妈往篮子里放梨时，妈妈对putPeer执行P原语操作阻塞。另外，我们需要设置两个变量，dadBlock和mamBlock,用来表示爸爸妈妈处于阻塞状态，当对方放一个苹果或梨时唤醒对方。

(3)这两个进程的代码如下：

int apple=0,peer=0;

int dadBlock=0,mamBlock=0;

SEMAPHORE mutex=1;

SEMAPHORE putApple=0;

SEMAPHORE putPeer=0;

father() {

while(1) {

P(mutex);

if (apple-peer == m-1) {

dadBlock=1;

V(mutex);

P(putApple);

}else V(mutex);

爸爸将苹果放到篮子中;

P(mutex);

apple++;

dadApple=0;

V(mutex);

if (mumBlock==1) V(putPeer);

}

}

mother() {

while(1) {

P(mutex);

if (apple-peer == 1-n) {

mumBlock=1;

V(mutex);

P(putPeer);

} else V(mutex);

妈妈将梨放到篮子中;

P(mutex);

peer++;

mumPeer=0;

V(mutex);

if (dadBlock==1) V(putApple);

}

}

注：本题分析是否正确，篮子是否为临界资源，请您留言发表不同意见。

如果你认为篮子也是临界资源，那就把“爸爸将苹果放到篮子中;”和“P(mutex);”；“妈妈将梨放到篮子中; ”和“ P(mutex);” 换个位置即可。

**问题6.**设有两个优先级相同的进程P1和P2，它们共享x,y和z三个变量，信号量S1和S2的初值均为0.试问：P1和P2并发执行后，x,y,z各等于多少?

进程P1和P2的伪代码描述

|  |  |
| --- | --- |
| 进程P1  y=1, z=0;  y=y+2;  V(S1);  z=y+1;  P(S2);  y=z+y; | 进程P2  x=1;  x=x+1;  P(S1);  x=x+y;  V(S2);  z=x+z; |

解答：P1和P2共享了变量x、y和z，下面我们来分析这三个变量变化。

变量x。变量x仅在P2中使用，它的值与变量y相关。

变量y。P2仅对y进行了读操作，P1对y进行了读写操作，所以，我们仅分析P2在读y时，P1是否可以对其进行写操作。从P2的伪代码中可以看出，P2读y时，必须先执行P(S1)，而S1初值为0，如果P1对S1不执行V(S1)，P2在此处会阻塞，直到P1执行V(S1)之后才可运行。P1执行V(S1)之后，P1和P2都可以对y进行读操作，但P1还要对y进行写操作，那么我们需要分析P1的写操作y=z+y是否会影响P2的读操作x=x+y。P1在执行y=z+y操作之前需要先执行P(S2)，如果P2没有执行V(S2)，P1的y=z+y不能执行，所以，P1的y=z+y是在P2的x=x+y之后执行，又P2的x=x+y语句是在P1的V(S1)之后P2的V(S2)之前执行，在此期间，y的值为3，所以，x的值为5，而y的值与z的值相关。

变量z。P1和P2均对变量z进行读写操作。P1若不执行V(S1)，则P2无法写z，因此P1先执行z=0语句。当P1执行V(S1)之后，P1和P2都可以对z进行写操作，即P1和P2对z的操作是无序的。所以，y和z的结果与3条语句（z=y+1;y=z+y;z=x+z;）的执行顺序有关。

当这3条语句按z=y+1;y=z+y;z=x+z;顺序执行，x、y和z各等于5、7和9。

当这3条语句按z=x+z;z=y+1;y=z+y;顺序执行，x、y和z各等于5、7和4。

当这3条语句按z=y+1; z=x+z;y=z+y;顺序执行，x、y和z各等于5、12和9。

以上分析正确吗？如果您有其他想法请留言。谢谢！

**问题7 ：**一医院门诊部的候诊室最多允许10名患者进入，若多于10人，则患者必须在门外等待，否则可进入候诊室候诊。门诊部共有3名医生，可同时给患者看病。当候诊室无等待的患者时，医生需等待患者的到来；当医生看病完成时，叫下一个等待的患者。当三个医生都在忙时，患者需等待，患者看病完成后离开候诊室。若把患者和医生看病的过程分别看作进程，试回答下列问题：（1）用信号量管理这些进程时，应怎样定义信号量，写出信号量的初值及信号量的含义；（2）根据所定义的信号量，用P、V原语描述患者进程和医生进程的活动，以保证它们能正确地并发执行。

解：（1）使用4个信号量实现，互斥信号量mutex保证病人互斥地进出候诊室。empty表示候诊室还可以容纳多少病人，full表示候诊室中有多少病人，service表示请求医生服务。

（2）同步算法描述如下：

Semaphore mutex=1；//互斥进出候诊室

Semaphore empty=10；//候诊室可进入人数

Semaphore full=0； //已进入候诊室人数

Semaphore service=0；//等待叫号

cobegin

{

process 病人i

{

P(empty)；

P(mutex)；

进入候诊室；

V(mutex)；

V(full)；

P(service)； //等待叫号

获得服务；

服务结束;

P(mutex)；

走出候诊室；

V(mutex)；

V(empty);

}

process 医生i

{

while(TRUE)

{

P(full)；

V(service)；//叫号

为病人诊病；

服务结束，病人离开；

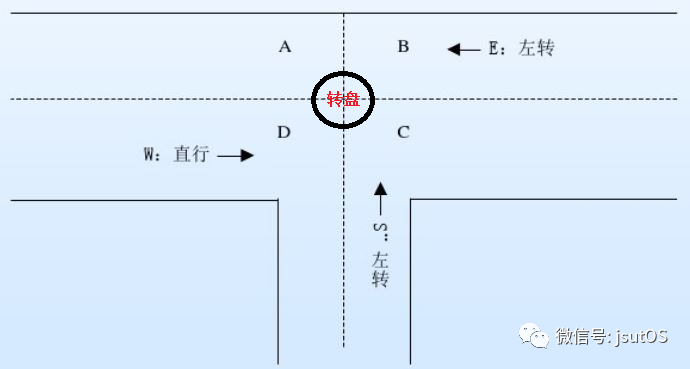
}

}

}

Coend

**问题8：**设有一个T型路口，其中A、B、C、D处各可容纳一辆车，车行方向如图所示（左转要求绕过转盘，直行按箭头所指方向直行），试将三个方向的车辆抽象成进程，写出三个方向的车辆过T型路口的算法。为了提高车辆通过T型路口的效率，你应采取什么措施？



解答：A,B,C,D这4处是临界资源，W：直行,依次经过D,C；E:左转,依次经过B，A,D；S:左转，依次经过C,B,A。W与S共享C，W和E共享D，S与E共享B，A。

这4处最重要的资源是C，一旦C处被S方向的车长期占用，则有可能发生死锁。而C处是由W与S方向的车辆共享，而S首先申请C，W首先申请D，然后申请C，如果W申请到D而C被C占用，此时极有可能阻塞交通。因此，如果W方向的车辆限预先申请C处，得到后再申请D处即不会发生阻塞。3个方向进程的同步算法如下。

SEMAPHORE A=1；

SEMAPHORE B=1；

SEMAPHORE C=1；

SEMAPHORE D=1；

W() { //W：直行

P(C);

P(D);

进入D处;

进入C处;

V(D);

通过C处；

V(C);

}

E() { //E:左转

P(B);

进入B处;

P(A);

进入A处;

V(B);

P(D);

进入D处;

V(A);

通过D;

V(D);

}

S() { //S:左转

P(C);

进入C处;

P(B);

进入B处;

V(C);

P(A);

进入A处;

V(B);

通过A;

V(A);

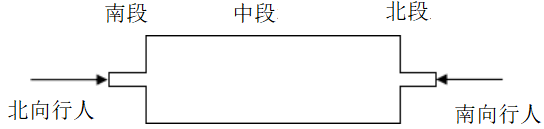
}

“为了提高车辆通过T型路口的效率，你应采取什么措施？”这一问题留给读者。

[**答疑（进程同步、互斥、死锁）之三**](https://mp.weixin.qq.com/s/667BS1mIQoXE-VyJ74bzeQ)

原创 lhw  操作系统学习  2021-3-29

**问题1**：一座小桥横跨南北两岸，分南段、中段和北段三部分，**南段**较窄仅允许单向通行（即当有北向行人依次通过时，南向行人无法通过。同样，当有南向行人依次通过时，北向行人无法通过）；**中段**宽敞，允许多人通过或歇息；**北段**较短，仅允许两人同时通过，整座桥梁最多承重30人，试用信号量及PV操作实现南、北两岸行人过桥的同步，并叙述信号量的含义和初值。



答：设信号量num，用来控制桥上人数，初值为30，表示桥上最多有30人；

南段较窄，仅允许单向通行，设置一个信号量south，表示南段是否可以通行，初值为1。，表示可以通行。南向或北向行人在通行之前对其执行P操作，通过后对其执行V操作。

北段较短，仅允许两人同时通过，设置一个变量n\_num，初值为0，表示要通过北段的人数，设置一个信号量n\_north，初值为1，用来对n\_num互斥访问；设置信号量pass，初值为0，表示不可以通过北段，当为1时表示可以通过北段，用来同步仅允许两人同时通过北段。

同步算法描述如下:

SEMAPHORE num=30;

SEMAPHORE south=1，n\_north=1，pass=0;

int n\_num=0;

SouthBank( )  // 北向行人

{

    南岸的行人要过桥;

    P(num); //判断桥上是否已达到最多人数

    P(south); //若南岸无人过桥那就过，否则等

        通过南侧段;

    V(south); //南岸的行人走过南段了

    进入中间宽敞处，继续行走或歇息;

   P(n\_north); //南岸的行人想通过北段，需要等2人

      n\_num++；

     if (n\_num==2) { //够两个人，就可以过北段

          V(pass); //通知另外一个人通过

        通过北段;

        n\_num=0; //2人通过北段

        V(n\_north); //修改n\_num完毕

      } else {

       V(n\_north); //不够2人，等待

       P(pass); //等另一个人通行  
           通过北段；

     }

    V(num); //北向行人过桥了

}

NorthBank( )  // 南向行人

{

    南向行人要过桥;

    P(num);   //判断桥上是否已达到最多人数

    P(n\_north); //南向行人想通过北段，需要等2人

      n\_num++；

     if (n\_num==2) { //够两个人，就可以过北段  
          V(pass); //通知另外一个人通过

        通过北段;

        n\_num=0; //2人通过北段

        V(n\_north); //修改n\_num完毕

      } else {

       V(n\_north); //不够2人，等待

         P(pass); //等另一个人通行  
           通过北段；

     }

      进入中间宽敞处，继续行走或歇息;

    P(south); //南侧段可以通行吗？

      通过南侧段;

    V(south);

    V(num); //北岸的行人过桥了

}

**问题2：**一座小桥（最多只能承重两个人）横跨南北两岸，任意时刻一方向只允许一人过桥，南侧段和北侧段较窄只能通过一人，桥中央一处宽敞，允许两个人通过或歇息。试用信号灯和PV操作写出南、北岸过桥的同步算法。

解答：控制“任意时刻一方向只允许一人过桥”需要设置两个信号量south和north，分别表示南岸和北岸要通行的人，它们的初值均为1，在通行之前对其执行P操作，通过后对其执行V操作。

“南侧段和北侧段较窄只能通过一人”，说明南侧段和北侧段都属于临界资源，当行人要通过时只能允许一个人通过，不管来自南岸还是北岸。设两个信号量S\_Narrow和N\_Narrow,它们的初值均为1，当行人要通过南侧段和北侧段时对相应信号量执行P操作，通过后执行V操作。

由于任何时刻一方向只允许一人过桥，所以桥上最多有2个人，符合题意：小桥最多只能承重2个人。

SEMAPHORE south=1, north=1;

SEMAPHORE S\_Narrow=1, N\_Narrow=1;

SouthBank( ) { // 南岸

南岸的行人要过桥;

P(south); //若南岸无人过桥那就过，否则等

可以过桥了;

P(S\_Narrow); //南侧段可以通行吗？

通过南侧段;

V(S\_Narrow);

进入中间宽敞处;

P(N\_Narrow); //北侧段可以通行吗？

通过北侧段;

V(N\_Narrow);

V(south); //南岸的行人过桥了

}

NorthBank( ) { // 北岸

北岸的行人要过桥;

P(north); //若北岸无人过桥那就过，否则等

可以过桥了;

P(N\_Narrow); //北侧段可以通行吗？

通过北侧段;

V(N\_Narrow);

进入中间宽敞处;

P(S\_Narrow); //南侧段可以通行吗？

通过南侧段;

V(S\_Narrow);

V(north); //北岸的行人过桥了

}

**问题3：**A single-lane bridge oonnects North Village and South Village. Farmers in the two villages use this bridge to deliver their produce to the neighbouring town. The bridge can become deadlocked if both a northbound and a southbound farmer get on the bridge at the same time (farmers in these villages are stubbom and are unable to back up). 一座独木桥连接通南村和北村。这两个村庄的农民利用这座桥把农产品运到邻近的城镇。如果两个村的农民同时过桥则谁也无法通过（假设这两个村庄的农民都有点问题，无法自行后退）。

a) Using either semaphores or synchronisation, design an algorithm that prevents deadlock. Implement and test your algortthm by designing two threads, one representing a northbound farmer and the other representing a southbound farmer. Once both are on the bridge, each will sleep for a random period of time to simulate traveling across the bridge. Initially, do not be concerned about starvation (the situation in which northbound farmers prevent southbound farmers from using the bridge, or vice-versa).  使用信号量或同步，设计一个防止死锁的算法. 通过设计两个线程来实现和测试您的算法。这两个线程一个表示北村的农民，另一个表示南村的农民。一旦两者都上了桥，每个人都会随机睡眠一段时间，来模拟在桥上的行走。最初，不要担心饥饿（北方农民阻止南方农民使用桥梁的局面，反之亦然）。解：设置两个变量southnum,northnum分别统计南方和北方要过桥的人数，设置两个信号量Smutex和Nmutex，初值均为1，分别表示对这两个变量进行互斥访问，设置互斥信号量mutex对独木桥的互斥使用，其初值为1。一旦一方占用独木桥，只要该方还有人要过桥，那就一直占用下去，直到该方无人过独木桥时释放。

int southnum=0, northnum=0;

SEMAPHORE mutex=1;

SEMAPHORE Smutex=1;

SEMAPHORE Nmutex=1;

SouthFamer() //南方村民

{

P(Smutex);

southnum++;

if (southnum==1)

P(mutex)

V(Smutex);

sleep(rand()); //过桥，睡眠一段时间，模拟过桥过程

P(Smutex);

southnum--;

if (southnum==0)

V(mutex);

V(Smutex);

}

NouthFamer() //北方村民

{

P(Nmutex);

northnum++;

if (northnum==1)

P(mutex)

V(Nmutex);

sleep(rand()); //过桥,睡眠一段时间，模拟过桥过程

P(Nmutex);

southnum--;

if (southnum==0)

V(mutex);

V(Nmutex);

}

b) Modify your implementalion in (a) so that it is starvation-free. 修改(a)中的实现，使其免于饥饿。解：(a)算法存在饥饿现象：如果一方一直占用独木桥，另一方无法通行，这就是饥饿现象。为解决饥饿现象以及公平地使用公共资源，再设置一个信号量fairness，初值为1，按到达独木桥起始端的先后顺序过桥，算法描述如下。

int southnum=0, northnum=0;

SEMAPHORE mutex=1;

SEMAPHORE Smutex=1;

SEMAPHORE Nmutex=1;

SEMAPHORE fairness=1;

SouthFamer() //南方村民

{

P(fairness);

P(Smutex);

southnum++;

if (southnum==1)

P(mutex)

V(Smutex);

V(fairness);

sleep(rand()); //过桥，睡眠一段时间，模拟过桥过程

P(Smutex);

southnum--;

if (southnum==0)

V(mutex);

V(Smutex);

}

NouthFamer() //北方村民

{

P(fairness);

P(Nmutex);

northnum++;

if (northnum==1)

P(mutex)

V(Nmutex);

V(fairness);

sleep(rand()); //过桥，睡眠一段时间，模拟过桥过程

P(Nmutex);

southnum--;

if (southnum==0)

V(mutex);

V(Nmutex);

}

请读者分析，下面的算法与上面的算法有什么不同？  
SEMAPHORE mutex=1;

SouthFamer() //南方村民

{

P(mutex)

sleep(rand()); //过桥，睡眠一段时间，模拟过桥过程

V(mutex);

}

NouthFamer() //北方村民

{

P(mutex)

sleep(rand()); //过桥，睡眠一段时间，模拟过桥过程

V(mutex);

}

**问题4：**驾校考场有20台考车和7个考位，每个考生在其中任何一个考位按照规定顺序依次完成5个项目的考试。考车循环使用，每当一个考生考完交车后，下一个考生上车考试。考生可前往考生少的考位考试。在某个考位，只要当前需要进行的考试项目场地空闲，考生即可进入该项目考试。试用信号量和PV操作写出考生考试的过程。

Semaphore cars=20; // 20台考车

Semaphore position=7; // 7个考位

Semaphore mutex=1; //办理借车手续且查找考位

Semaphore Item[7][5]={1,1,1,1,1,

1,1,1,1,1,

1,1,1,1,1,

1,1,1,1,1,

1,1,1,1,1,

1,1,1,1,1,

1,1,1,1,1

}; // 每个考位的每个项目的使用情况

int position[7]={0,0,0,0,0,0,0};//考位人数

examinee() { // 考生

int i,step;

P(cars); //申请一辆考车

P(mutex); //办理借车手续

驾车寻找考位;

step=0;

for(i=1;i<7;i++) {

if (position[step]<positon[i]) step=i;

}

position[step]++; //选中step号考位

V(mutex);

进入step考位;

for(i=0;i<5;i++) {

P(Item[step][i]); //第i个项目

进行第i个项目的考试;

V(Item[step][i]);

}

考试结束；

P(mutex);

position[step]--; //退出step号考位

还车;

V(cars);

V(mutex);

}

**问题5：**驾校画出下面一系列语句的前趋图（假设x,y,z已知），并用信号量实现该前趋图的前趋关系。

S1：a=x+y;

S2：b=z+1;

S3：c=a+b;

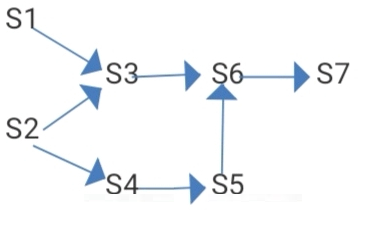
S4：d=b+5;

S5：e=d-x;

S6：f=c+e;

S7：g=f+y;

答：前趋图如下图所示。



用信号量实现该前趋图的前趋关系如下。

semaphore Sem1=0; /\* S1 \*/

semaphore Sem2=0; /\* S2 \*/

semaphore Sem3=0; /\* S3 \*/

semaphore Sem4=0; /\* S4 \*/

semaphore Sem5=0; /\* S5 \*/

semaphore Sem6=0; /\* S6 \*/

S1( ) {

a=x+y;

V(Sem1);

}

S2( ) {

b=z+1;

V(Sem2);

V(Sem2);

}

S3( ) {

P(Sem1);

P(Sem2);

c=a+b;

V(Sem3);

}

S4( ) {

P(Sem2);

d=b+5;

V(Sem4);

}

S5( ) {

P(Sem4);

e=d-x;

V(Sem5);

}

S6( ) {

P(Sem3);

P(Sem5);

f=c+e;

V(Sem6);

}

S7( ) {

P(Sem6);

g=f+y;

}

2. 桌上有一盘子，每次只能放入2个不同水果。爸爸专向盘里放苹果，妈妈专向盘里放橘子，儿子专吃盘中的半橘子和半个苹果，女儿专吃盘中的另外一半苹果和橘子。仅当盘中没有放水果时，爸爸妈妈才可向盘里各放一个水果，而且不允许放2个相同的水果。仅当盘中有自己需要的半个苹果和半个橘子时，儿子和女儿才可从盘里取出，而且爸爸、妈妈放入一次后，儿子和女儿必须各自取一次。把爸爸、妈妈、儿子和女儿看作4个进程，用Wait(),Signal()原语进行管理，使这4个进程能正确地并发执行。算法有问题，请看留言。

解：根据题目描述，资源使用半个苹果或半个橘子为单位分配的，爸爸、妈妈往盘子里放一个苹果或一个橘子，相当于各自放了2个资源。爸爸、妈妈、儿子和女儿四个人的活动可描述如下。

int get\_fruit=0; *// 盘子中可取的水果数，初值为0.*

SEMAPHORE mutex\_plate=1; *// 互斥访问get\_fruit变量*SEMAPHORE Sfather=1; *// father可以放苹果*

SEMAPHORE Smother=1; *// mother可以放橘子*

SEMAPHORE Sapple=0; *// 盘子中有几份半个苹果？初始为0*

SEMAPHORE Sorange=0; *// 盘子中有几份半个橘子？初始为0*

SEMAPHORE mutex=1; *// 互斥访问盘子*

father() {   *// father放苹果*

while(1) {

       P(Sfather); *// 可以放苹果吗？*

       P(mutex);       爸爸往盘子中放一个苹果;

       V(mutex);

       P(mutex\_plate);

       get\_fruit++; *// 放了一个苹果*

       V(mutex\_plate);

       V(Sapple); *// 半个苹果*

       V(Sapple); *// 半个苹果*

}

}

mother() { *// mother放橘子*

   while(1) {

       P(Smother); *// 可以放橘子吗？*

       P(mutex);

       妈妈往盘子中放一个橘子;

       V(mutex);

       P(mutex\_plate);

       get\_fruit++; *// 放了一个橘子*

       V(mutex\_plate);

       V(Sorange); *// 半个橘子*

       V(Sorange); *// 半个橘子*

   }

}

Son() {  *// 儿子吃一半的苹果和一半的橘子*

P(Sapple);   *// 申请半个苹果*

P(Sorange);  *// 申请半个橘子*

P(mutex);    儿子从盘子中取一半的苹果和一半橘子吃;

V(mutex);

    P(mutex\_plate);

    get\_fruit--;

    if (get\_fruil==0) {

       V(Sfather);

       V(Smother);

    }

    V(mutex\_plate);

}

Daughter() {  *// 女儿吃一半的苹果和一半的橘子*

P(Sapple);   *// 申请半个苹果*

P(Sorange);  *// 申请半个橘子*

P(mutex);    女儿从盘子中取一半的苹果和一半橘子吃;

V(mutex);

P(mutex\_plate);

get\_fruit--;

if (get\_fruil==0) {

   V(Sfather);

   V(Smother);

}

V(mutex\_plate);

}

第2题算法有问题，存在儿子或女儿吃掉盘子中所有水果的情况，即儿子或女儿连续吃两次，修改方法如下。

增加2个信号量：SonGet=0，DaughterGet=0分别表示父亲放1个苹果，儿子只能取半个，母亲放1个橘子，女儿只能取半个。

父亲将苹果放入盘子后执行V(SonGet)，儿子在吃之前执行V(SonGet)。

母亲将橘子放入盘子后执行V(DaughterGet)，女子在吃之前执行V(DaughterGet)。

**理发师问题**

原创 OS爱好者 [操作系统学习](javascript:void(0);) 2019-12-26

**1.理发师问题**

睡眠的理发师问题又是一个有趣的进程同步问题。在理发店中，有一位理发师、一把理发椅和N把供等候理发的顾客坐的椅子。如果没有顾客，理发师便在理发椅上睡觉；当一个顾客到来时，他必须唤醒理发师；如果理发师正在理发时又有顾客光临，那么，如果有空椅子可坐，顾客就坐下来等待，否则就离开理发店。只有在理发椅空闲时，顾客才能坐上并等待理发师理发，而理发师只有在理发椅上有顾客时才能进行理发。如果没有顾客等待理发，理发师就睡眠，直到第一位进来的顾客将其唤醒并开始理发。因此，设置两个同步信号量customers和barber，实现理发师与顾客之间的同步。customers表示等候理发的顾客数(不包括正在理发的顾客)，初值为0。barber用于记录正在等待顾客的理发师数，因理发师最初是睡眠的，当第一位顾客到来时才将他唤醒，所以barber的初值设为0。另外，设置一个计数变量waiting，用来记录等候理发的顾客数，初值为0。因waiting是一个共享变量，需设置互斥信号量mutex 对其管理。睡眠的理发师问题描述如下。

int waiting=0；/\*等候理发的顾客数\*/

SEMAPHORE customers=0， barber=0，mutex=1；

barber( ) {

  while(1) {

    P(customers)；/\* 若无顾客，理发师睡眠 \*/

    P(mutex)；/\* 对临界资源waiting共享 \*/

    waiting--；/\* 等候顾客数减一 \*/

    V(barber)；/\* 理发师去为一个顾客理发 \*/

    V(mutex)；/\* 释放对临界资源waiting 的使用 \*/

    cut\_hair( )；/\* 正在理发 \*/

  }

}

customer( int i) {

  P(mutex)；/\* 对临界资源waiting共享 \*/

  if (waiting < N){ /\* 看看有没有空椅子,共有N把椅子 \*/

    waiting++； /\* 等候顾客数加1 \*/

    V(customers)；/\* 必要的话唤醒理发师 \*/

    V(mutex)；/\* 释放对临界资源waiting 的使用 \*/

    P(barber)；/\* 等待理发师理发 \*/

    get\_haircut( )；/\* 一个顾客坐下等理发 \*/

  } else

    V(mutex)；/\* 理发店已满了，离开 \*/

}

**2.多理发师问题**  
有m个理发师，n个理发椅和k把等候理发的顾客坐的椅子。如果没有顾客，理发师在休息室；当一个顾客到来时，必须唤醒理发师，进行理发；如果理发师正在理发，又有顾客来到，则如果有空椅子可坐，他就坐下来等，如果没有空椅子，他就离开。为理发师和顾客各编写一段程序描述他们的行为。

解：

定义2个变量wchair和bchair分别表示在理发椅上和顾客椅上等待的顾客数。

定义信号量chair管理理发椅，理发师要理发首先需要申请一把工作椅，此题没有说M和N谁大谁小，所以设置一个信号量来管理理发椅。

信号量customers和barber实现理发师和顾客的同步。

mutex互斥信号量，用来互斥使用共享变量wchair和bchair。

多理发师同步问题描述如下。

int wchair=0; /\* 在顾客椅子上等待的顾客 \*/

int bchair=0; /\* 在理发椅上等待的顾客 \*/

semaphore chair=N; /\* 理发椅子数 \*/

semaphore customers=0; /\* 表示等候理发的顾客数，最大值N+K \*/

semaphore barber=M;  /\* 理发师人数 \*/

semaphore mutex=1; /\* 互斥信号量，互斥使用wchair和bchair \*/

barber()  /\* 理发师  \*/

{

   while(1) {

P(chair);   /\* 理发师要理发，先申请一把工作椅 \*/

    P(customers);   /\* 若无顾客，理发师睡眠 \*/

    cut\_hair； /\* 给顾客理发 \*/

    P(mutex)；      /\* 理发结束，修改共享变量 \*/

    if (wchain>0) {

      wchain--； /\* 等候顾客数减一 \*/

    } else {

      bchair--;   /\* 等候顾客数减一 \*/

    }

    V(mutex)； /\* 释放对临界资源的使用 \*/

    V(chair);  /\* 释放理发椅 \*/

    V(barber)；/\* 理发师准备为下一个顾客理发 \*/

}

}

customer() /\* 顾客进店 \*/

{

  P(mutex)； /\* 对临界资源共享 \*/

   if (bchair<n) { /\* 理发椅坐满了吗？ \*/

     bchair++;   /\* 不满，坐 \*/

   }

else {

   if (wchair < k) { /\* 客户椅子坐满了吗？\*/

     wchair++；    /\* 等候顾客数加1 \*/

   }

else {

      V(mutex);

      return;  /\* 无座离开理发室 \*/

    }

  }

  V(mutex);

  V(customers); /\* 理发人数加1，必要的话唤醒理发师 \*/

  P(barber);    /\* 等待理发师理发 \*/

  get\_haircut； /\* 一个顾客坐下等理发 \*/

}

**\*\*\*整理到此处\*\*\***

**读者与写者问题**

原创 lhw [操作系统学习](javascript:void(0);) 2019-10-26

读者与写者问题（The readers-writers problem）是指多个进程对一个共享资源进行读写操作的问题。在两组并发执行的进程中，一组进程只要求读数据文件内容，称为读者；另一组进程要求修改数据文件内容，称为写者。对读者和写者的要求是：

(1)允许多个读者同时读，即读者可以同时读数据文件，而不需要互斥。

(2)一个写者不能和其他进程同时访问数据文件，它们之间必须互斥。

当若干读者正在读数据文件时，来了一个写者，写者需要等待所有读者的读操作结束后，才能对数据文件进行写操作。那么，当写者正在等待读者完成读操作期间，又来了新的读者，系统该如何处理呢？

**(1) 读者优先。**若允许新来的读者进行读操作，则称该处理方式为读者优先。读者优先的思想是除非有写者正在写文件，否则没有一个读者需要等待。

**(2) 写着优先。**若不允许新来的读者进行读操作，而需要等待写者写操作完成后，在没有写者时才允许其读，则称该处理方式为写者优先。写者优先的思想是一旦一个写者到来，它应该尽快对文件进行写操作，即如果有一个写者在等待，那么，就不允许新到来的读者进行读操作。

**(3) 公平策略。**读者优先与写者优先这两类读者与写者问题都会导致“饥饿”现象的发生。对于前者，当有新的读者不断地到来，会使写者没有机会进行写操作；对于后者，当有新写者不断地到来，就会使新来的读者挨饿。Hoare提出了一种更公平的策略，由如下规则定义。

**规则1:**在一个读序列中，如果有写者在等待，那么就不允许有新的读者进行读操作。

**规则2:**在一个写操作结束时，所有等待的读者应该比下一个写者有更高的优先权。

**1. 读者优先算法**

在该问题中，数据文件属于临界资源，写者与读者之间、写者与写者之间要互斥地访问该资源。读者与读者不互斥，有读者在读操作期间，写者将被阻塞，即只要第一个读者取得了读文件的权利则其他读者可以跟着读文件，所以，可以将写者与读者之间的互斥看作是写者与第一个读者之间的互斥。当最后一个读者结束读操作时，若有写者在等待，才将写者唤醒。设置互斥信号量mutex，实现读者与写者、写者与写者之间的互斥。当一个写者完成写操作后，若有读者要读或者有其他写者要写，则唤醒其中的一个，具体唤醒哪一个与mutex等待队列的管理有关，一般情况下，唤醒该等待队列之首进程。设置一个变量readcount，其初值为0，它代表正在读操作的读者人数。因为所有读者都要共享该变量，因此它是一个临界资源，采用互斥信号量Rmutex管理该变量。

读者与写者问题（读者优先）的同步算法描述如下：

int readcount=0；    /\* 正在读的读者数 \*/

semaphore mutex=1；   /\* 对数据文件进行保护的互斥信号量 \*/

semaphore Rmutex=1；  /\* 对变量readcount进行保护的互斥信号量 \*/

main() {

   创建M个读者进程read(i)；   /\* i表示第i个读者 \*/

创建N个写者进程write(i)；  /\* i表示第i个写者 \*/

cobegin

     read(1)；read(2)；……；read(M)；   /\* 并发执行的M个读者 \*/

     write(1)；write(2)；……；write(N)；/\* 并发执行的N个写者 \*/

   coend

}

read(int i)   /\* 读者 \*/

{

  while (1) {

P(Rmutex)；

readcount++；

if (readcount==1)

  P(mutex)；/\* 如果该读者是第1位读者，则申请进入读操作，否则直接进入。\*/

V(Rmutex)；

读数据；

P(Rmutex)；/\* 读完后离开时，修改readcount变量。\*/

readcount--；

if (readcount==0)

  V(mutex)；/\* 当读者全部离开时，释放数据文件资源，并负责唤醒写者。\*/

   V（Rmutex）；

 }

}

write(int i)       /\* 写者 \*/

{

  while(1){

   P(mutex)；

   写数据；

   V(mutex)；

  }

}

**2. 写者优先算法**

int readcount=0；  /\* 正在读的读者数 \*/

int writecount=0；  /\* 正在写的读者数 \*/

semaphore mutex=1；/\* 对数据文件进行保护的互斥信号量 \*/

semaphore Rmutex=1；/\* 对变量readcount进行保护的互斥信号量 \*/

semaphore Wmutex=1；/\* 对变量writecount进行保护的互斥信号量 \*/

semaphore write=1；/\* 写者 \*/

main()

{

   创建M个读者进程read(i)；   /\* i表示第i个读者 \*/

创建N个写者进程write(i)；  /\* i表示第i个写者 \*/

cobegin

     read(1)；read(2)；……；read(M)；   /\* 并发执行的M个读者 \*/

     write(1)；write(2)；……；write(N)；/\* 并发执行的N个写者 \*/

   coend

}

read(int i)   /\* 读者 \*/

{

  while (1) {

P(write);  //没有写者则可以读

P(Rmutex)；

readcount++；

if (readcount==1)

  P(mutex)；/\* 如果该读者是第1位读者，则申请进入读操作，否则直接进入。\*/

V(Rmutex)；

V(write);

读数据；

P(Rmutex)；/\* 读完后离开时，修改readcount变量。\*/

readcount--；

if (readcount==0)

  V(mutex)；/\* 当读者全部离开时，释放数据文件资源，并负责唤醒写者。\*/

   V（Rmutex）；

 }

}

write(int i)       /\* 写者 \*/

{

  while(1){

P(Wmutex)；

writecount++；

if (writecount==1)

  P(write); //写者到，则阻止后继读者

V(Wmutex)；

   P(mutex);

   写数据；

   V(mutex);

P(Wmutex)；/\* 写完后离开时，修改writecount变量。\*/

writecount--；

if (writecount==0)

  V(write)；/\* 当写者全部离开时，释放数据文件资源，并负责唤醒读者。\*/

   V（Wmutex）；

  }

}

**3. 读者与写者公平算法**

无写者，则多个读者可同时读，有写者请求，则后来的读者不能进入。在一个写操作结束时，所有等待的读者应该比下一个写者有更高的优先权。

int readcount=0；  /\* 正在读的读者数 \*/

semaphore mutex=1；/\* 对数据文件进行保护的互斥信号量 \*/

semaphore Rmutex=1；/\* 对变量readcount进行保护的互斥信号量 \*/

semaphore first=1; /\* 用于写者来了后，后续读者不能读，

    或一个写者结束后，在下一个写者之前请求读的读者先进入读操作。\*/

main()

{

  创建M个读者进程read(i)；   /\* i表示第i个读者 \*/

创建N个写者进程write(i)；  /\* i表示第i个写者 \*/

cobegin

     read(1)；read(2)；……；read(M)；   /\* 并发执行的M个读者 \*/

     write(1)；write(2)；……；write(N)；/\* 并发执行的N个写者 \*/

   coend

}

read(int i)   /\* 读者 \*/

{

  while (1) {

P(first) /\*若有写者，则阻塞等待\*/

P(Rmutex)；

readcount++；

if (readcount==1)

  P(mutex)；/\* 如果该读者是第1位读者，则申请进入读操作，否则直接进入。\*/

V(Rmutex)；

V(first);

读数据；

P(Rmutex)；/\* 读完后离开时，修改readcount变量。\*/

readcount--；

if (readcount==0)

  V(mutex)；/\* 当读者全部离开时，释放数据文件资源，并负责唤醒写者。\*/

   V（Rmutex）；

 }

}

write(int i)       /\* 写者 \*/

{

  while(1){

   P(first); /\*无读者，则进入，否则等待\*/

   P(mutex)；

   写数据；

   V(mutex)；

   V(first);

  }

}

**4. 类似问题的解答**

一条南北走向的大河上，有一座独木桥，东岸和西岸的汽车可分别通行：为保证安全，东岸或者西岸的汽车需要过桥时，如果桥面上无车，则允许一方的汽车过桥；当一方的汽车全部过完后，另一方的汽车才可以过桥。而且，桥面上最多可以承载12辆汽车同时通行。试用信号量和PV操作，写出东岸、西岸的汽车过独木桥的并发程序。

该题可看作是读者与写者问题（读者优先）的变种。如果理解读者与写者问题，该题就不难解答。

semaphore bridge = 1;  
semaphore e\_mutex = 1;  
semaphore w\_mutex = 1;  
semaphore empty = 12;

int e\_count = 0；

int w\_count = 0;

east() {  
    P(e\_mutex);  
    e\_count++;  
    if(e\_count==1){  
        P(bridge);  
    }  
    V(e\_mutex);  
    P(empty);  
    行驶;  
    V(empty);  
    P(e\_mutex);  
    e\_count--;  
    if(e\_count==0){  
        V(bridge);  
    }  
    V(e\_mutex);  
}

west() {  
    P(w\_mutex);  
    w\_count++;  
    if(w\_count==1){  
        P(bridge);  
    }  
    V(w\_mutex);  
    P(empty);  
    行驶;  
    V(empty);  
    P(w\_mutex);  
    w\_count--;  
    if(w\_count==0){  
        V(bridge);  
    }  
    V(w\_mutex);  
}

**感谢魏翔宇同学提供本题算法描述。**

**一道进程同步题目引发的思考**

原创 lhw [操作系统学习](javascript:void(0);) 2019-08-22

今天看书，发现一个题目很好，就看了看，题目与解答如下图所示。

该题存在的问题是没有对变量z赋初值，这个不应该。另外，作者的分析结果正确吗？下面是我的分析，为了能让分析继续下去，在P1中添加一句对变量z赋初值的语句z=0;进程P1和P2的伪代码描述

|  |  |
| --- | --- |
| 进程P1  y=1, z=0;  y=y+2;  V(S1);  z=y+1;  P(S2);  y=z+y; | 进程P2  x=1;  x=x+1;  P(S1);  x=x+y;  V(S2);  z=x+z; |

分析结果如下。

查看P1和P2伪代码可以知道，这两个进程共享了变量x、y和z，如果两个进程能按照读者与写者的规则使用这三个变量，那么，这两个进程执行后，三个变量的值应该是唯一的，否则存在不确定性。下面我们来分析这三个变量。

变量x。变量x仅在P2中使用，它的值与变量y相关。

变量y。P2仅对y进行了读操作，P1对y进行了读写操作，所以，我们仅分析P2在读y时，P1是否可以对其进行写操作。从P2的伪代码中可以看出，P2要读y时，必须先执行P(S1)，而S1初值为0，如果P1对S1不执行V(S1)，P2在此处会阻塞，直到P1执行V(S1)之后才可运行。P1执行V(S1)之后，P1和P2都可以对y进行读操作，但P1还要对y进行写操作，那么我们需要分析P1的写操作y=x+y与P2的读操作x=x+y是否符合读者与写者规则。P1执行y=x+y操作需要先执行P(S2)，如果P2没有执行V(S2)，P1的y=x+y不能执行，所以，P2在执行x=x+y之后P1的y=x+y才可执行，因此，P1和P2对y的操作符合读者与写着规则。P2的x=x+y语句是在P1的V(S1)之后，P2的V(S2)之前执行，在此期间，y的值为3，所以，P1、P2并发执行后，x的值为5，但y的值与z的值相关。

变量z。P1和P2均对变量z进行读写操作。P1的不执行V(S1)，P2无法写z，因此P1先执行z=0语句。当P1执行V(S1)之后，P1和P2都可以对z进行写操作，即P1和P2对z的操作是无序的。所以，y和z的结果与3条语句（z=y+1;y=z+y;z=x+z;）的执行顺序有关。

当这3条语句按z=y+1;y=z+y;z=x+z;顺序执行，x、y和z各等于5、7和9。

当这3条语句按z=x+z;z=y+1;y=z+y;顺序执行，x、y和z各等于5、7和4。

当这3条语句按z=y+1; z=x+z;y=z+y;顺序执行，x、y和z各等于5、12和9。

以上分析正确吗？如果您有其他想法请留言。谢谢！

**问题1**

有一个活动场地最多可容纳26名同学参与活动。其中一部分同学参与打乒乓球活动（不妨设为活动A），另一部分同学参与网球活动（不妨设为活动B）。规定如下：(1)若活动场地中同学人数已经超过26人，则申请进入活动场地的同学等待；(2)参与A、B两类活动的同学人数之差不能超过7人；即若参与活动A的同学人数比参加活动B的同学人数多7人，则申请参与活动A的同学等待；同样，若参与活动B的同学人数比参加活动A的同学人数超多7人，则申请参与活动B的同学等待；(3)参与A、B两组活动的同学可随时离开；

请用“参与A”、“参与B”、“离开A”、“离开B”及信号量机制描述同学进入活动场地的过程。

【参考答案】

semaphore empty = 26, ping\_pong = 7, tennis = 7; （2分）

进程参与A：（2分）

while(true) {

P(empty)

P(ping\_pong);

加入乒乓球组；

V(tennis);

}

参与B：（2分）

while(true) {

P(empty)

P(tennis);

加入网球组；

V(ping\_pong);

}

离开A：（2分）

while(true) {

离开乒乓球组；

V(empty);

}

离开B：（2分）

while(true) {

离开网球组；

V(empty);

}

**问题2**

一透明箱子，装满了黄、白两种颜色的乒乓球，箱子上部有一小口，大小仅容一手探入。两人同时从箱子中拣出乒乓球，一次拣一个球，一个人只拣黄色的，另一个人只拣白色的。两人已拣出的乒乓球数目之差要小于常数M。请用信号量及PV操作描述该过程。

semaphore quota1= quota2 = M; （2分）

semaphore mutex=1; （2分）

YELLOW：（3分）

do {

Wait(quota1);

Wait(mutex)

Take out a yellow ball;

Signal(mutex)

Signal(quota2);

} while(1)

WHITE：（3分）

do {

Wait(quota2);

Wait(mutex)

Take out a white ball;

Signal(mutex)

Signal(quota1);

} while(1)