1. 三个进程 P1、P2、P3 互斥使用一个包含 N（N>0）个单元的缓冲区。P1 每次用 produce() 生成一个正整数并用 put()送入缓冲区某一个空单元中；P2 每次用 getodd()从该缓冲区中 取出一个奇数并用 countodd()统计奇数个数；P3 每次用 geteven()从该缓冲区中取出一个 偶数并用 counteven()统计偶数个数。请用信号量机制实现这三个进程的同步与互斥活动， 并说明所定义的信号量的含义。

首先，分析有哪些并发：P1、P2和P3都是并发的。对于临界区资源——缓冲区，有P1写，P2和P3读，需要定义信号量n1进行互斥，n1的初值为1.另外，需要记录缓冲区的单元数，如果小于等于0则不能使用。因此构造第二个信号量empty，记录剩余可用缓冲区的数量，初值为N。

对于同步机制，我们需要在P1读取到一个奇（偶）数时，调用P2（P3）。因此，需要两个信号量来同步二者读取-写入关系，一个even，一个odd。二者都初始化为0.

进程代码如下：

P1:

integer = produce() //获取正整数

P(empty) //要先确定是否有可用的缓冲区！然后将可用缓冲区减一

P(n1) //占用缓冲区

put(integer) //将正整数放到缓冲区

V(n1) //解除缓冲区占用

if integer %2 == 0:

V(even) //如果是偶数，则让偶数信号量++，以开启P3进程

else:

V(odd) //如果是奇数，则让奇数信号量++，以开启P2进程

P2:

P(odd) //消耗掉本次开启的奇数信号量

P(n1) //占用缓冲区

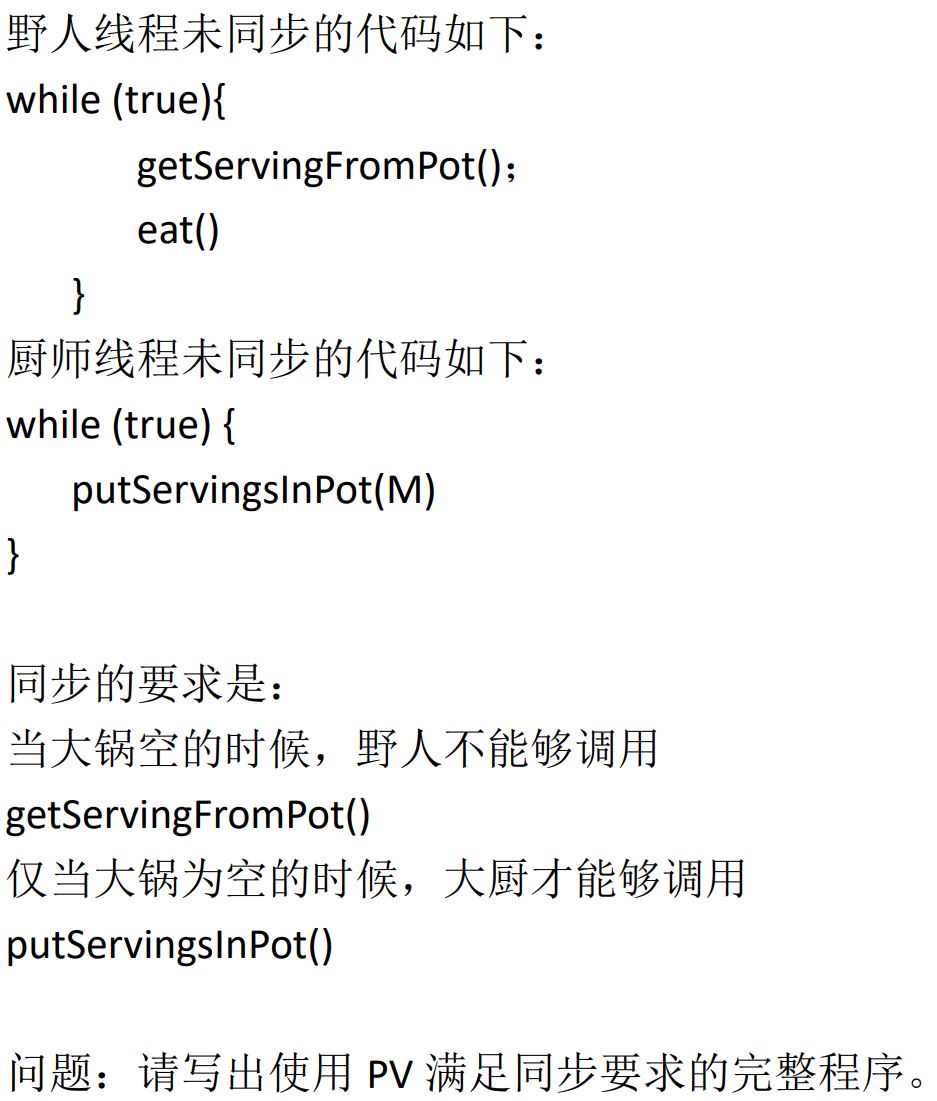
odd = getodd() //从缓冲区获取数字

V(n1) //取消缓冲区占用

V(empty) //都计数完了，可以把缓冲区让出来了

countodd()

P3类似P2.

1. 一个野人部落从一个大锅中一起吃炖肉，这个大锅一次可以存放 M 人份的炖肉。当野人 们想吃的时候，如果锅中不空，他们就自助着从大锅中吃肉。如果大锅空了，他们就叫 醒厨师，等待厨师再做一锅肉。  
     
     
   野人没有食物时，需要阻塞掉；厨师有食物时，需要阻塞掉。因此，设计一个信号量food和另一个信号量empty，food初始化为0，empty初始化为1.野人之间不能同时抢食物，因此还需要一个信号量havefood来控制野人之间的操作，初始化为1。具体的线程代码如下  
     
   厨师线程：  
   while(true):  
    P(empty)  
    putServingslnPot()  
    V(food, M)  
   }  
     
   野人线程：  
   while(true):  
    P(havefood)  
    if food == 0:  
    V(empty)  
    P(food)  
    getServingFromPot()  
    V(havefood)  
   }
2. 系统中有多个生产者进程和消费者进程，共享用一个可以存 1000 个产品的缓冲区（初 始为空），当缓冲区为未满时，生产者进程可以放入一件其生产的产品，否则等待；当缓 冲区为未空时，消费者进程可以取走一件产品，否则等待。要求一个消费者进程从缓冲 区连续取出 10 件产品后，其他消费者进程才可以取产品，请用信号量 P，V 操作实现进 程间的互斥和同步，要求写出完整的过程；并指出所用信号量的含义和初值。

所有进程都要操作缓冲区，缓冲区需要保护，定义信号量mutex来管理缓冲区，初始为1.另外，需要构建一个变量product，来表示共有几件产品，初始化为0；以及两个信号量empty和full，来代表缓冲区空/满，分别初始化为0和1.此外，还需要设置一个信号量lock，用以保护消费者连取10件商品的过程，初始化为1.进程代码如下：

生产者进程：  
while(true):  
 P(full)  
 P(mutex)

product++

produce()

V(mutex)

if product < 1000:   
V(full)

if product == 1:

V(empty)

}

消费者进程：

while(true):

P(lock)

for int i=0; i<10; i++: {

P(empty)

P(mutex)

consume()

product—

V(mutex)

if product > 0:

V(empty)

if product == 999:

V(full)

}

V(lock)

}

1. 读者写者问题的写者优先算法  :  1）共享读;  2）互斥写、读写互斥;  3）写者优先于读者 （一旦有写者，则后续读者必须等待，唤醒时优先考虑写者）

当读进程激活时，仅允许其他的读进程进入，而不允许写进程进入；当写进程进入时，不允许任何进程进入。

为了找出哪个是第一个进入的读进程，哪个是最后一个退出的读进程，需要设置变量read\_count（初始化为0），统计有几个进程正在读。这个变量是所有读进程的临界区，需要设置保护。设置变量write\_count（提高写的优先级，初始化为0）

设置如下几个信号量： write\_read（防止一个写一个读，初值为1），read\_count\_mutex（保护read\_count变量），read\_priority（挂起那些因为优先级而被堵住的读进程），write\_count\_mutex（保护write\_count变量）

读者：

If write\_count > 0:

P(read\_priority) //如果有已经开始等待或已经开始执行的写操作，则要等写操作全部搞完，本读操作才有资格开始

If read\_count == 0: //第一个读操作

P(read\_count\_mutex)

read\_count++

V(read\_count\_mutex)

P(write\_read)

Read\_content()

P(read\_count\_mutex)

read\_count--

V(read\_count\_mutex)

Else:

P(read\_count\_mutex)

read\_count++

V(read\_count\_mutex)

Read\_content()

P(read\_count\_mutex)

read\_count--

V(read\_count\_mutex)

If read\_count == 0:

V(write\_read) //所有读操作都结束了，开始允许写

写者：

P(write\_count\_mutex)

write\_count++ //有写操作了，要提高优先级

V(write\_count\_mutex)

P(write\_read) //等待现有读或写操作结束

Write\_content()

P(write\_count\_mutex)

write\_count-- //写操作完成，释放写操作优先级

V(write\_count\_mutex)

if write\_count > 0:

V(write\_read) //写完之后要先释放写-写占用，唤醒写的

else:

V(read\_priority) //如果没写的了，那写完之后要释放读-写占用

1. 寿司店问题。假设一个寿司店有 5 个座位，如果你到达的时候有一个空座位，你可以立 刻就坐。但是如果你到达的时候 5 个座位都是满的有人已经就坐，这就意味着这些人都 是一起来吃饭的，那么你需要等待所有的人一起离开才能就坐。编写同步原语，实现这 个场景的约束。

定义变量waiting表示正在等待的食客数目，初始化为0；定义变量eating表示正在吃饭的食客数目，初始化为0.

定义变量full，初始化为1，表示5个座位是否都满。定义信号量list，表示正在等待的食客队列。当有用户离开时，需要判断这个用户是不是一波进来满的5个人。如果不是，则不需要修改full信号量，直接让eating变量减一即可。如果是，还需要判断他是不是最后一个人。如果是最后一个人，那他需要负责把full变量置回，以允许新食客进入。另外，需要定义两个互斥信号量，用于确保eating变量和waiting变量这两个临界区不被同时修改。分别定义为waiting\_mutex和eating\_mutex，初始值都是1.代码如下：

食客进程：

If full:

P(waiting\_mutex)

waiting++

V(waiting\_mutex)

P(list) //如果5个人已满，则需要等待5个人的最后一个释放full，唤醒list中的食客。

P(waiting\_mutex)

waiting--

V(waiting\_mutex)

P(eating\_mutex)

eating++ //唤醒之后，waiting--，自己开吃

V(eating\_mutex)

Else:

P(eating\_mutex)

eating++ //如果5个人未满，则直接开吃

If eating == 5:

full = 1

V(eating\_mutex)

eat()

P(eating\_mutex)

eating--

V(eating\_mutex)

If eating == 0: //最后一个食客，需要负责唤醒list里面的食客

If waiting > 5: //一次分不完

For I in range(5):

V(list) //就放出来5个食客

Else:

For I in range(waiting):

V(list) //放出来所有食客，不用修改eating和waiting，进程自己会修改