经典同步问题

PV操作题目的解题思路：

1. 关系分析。找出题目中描述的各个进程，分析它们之间的同步、互斥关系。
2. 整理思路。根据各进程的操作流程确定P、V操作的大致顺序。
3. 设置信号量。设置需要的信号量，并根据题目条件确定信号量初值。（互斥信号量初值一般为1，同步信号量的初值要看对应资源的初始值是多少）

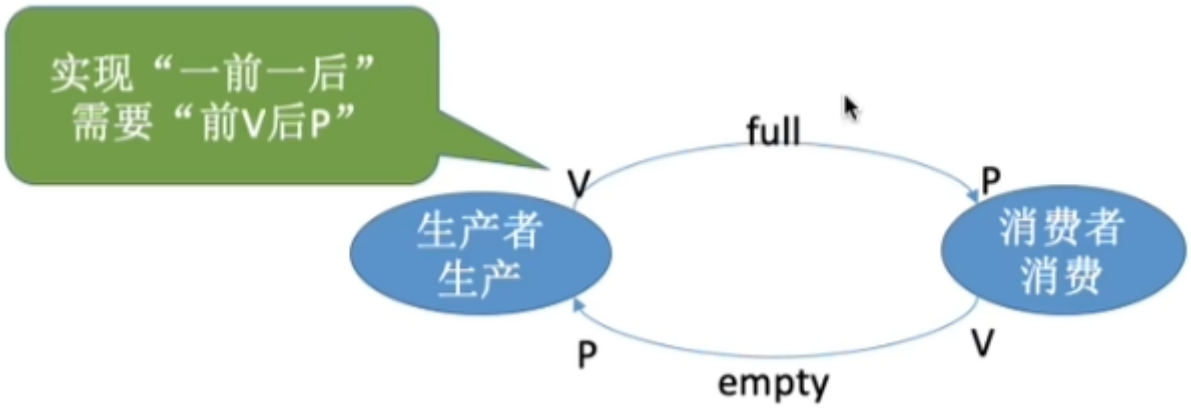
生产者-消费者问题

生产者消费者问题是一个互斥、同步的综合问题。

对于初学者来说最难的是发现题目中隐含的两对同步关系。

有时候是消费者需要等待生产者生产，有时候是生产者要等待消费者消费，这是两个不同的”一前一后问题“，因此也需要设置两个同步信号量。

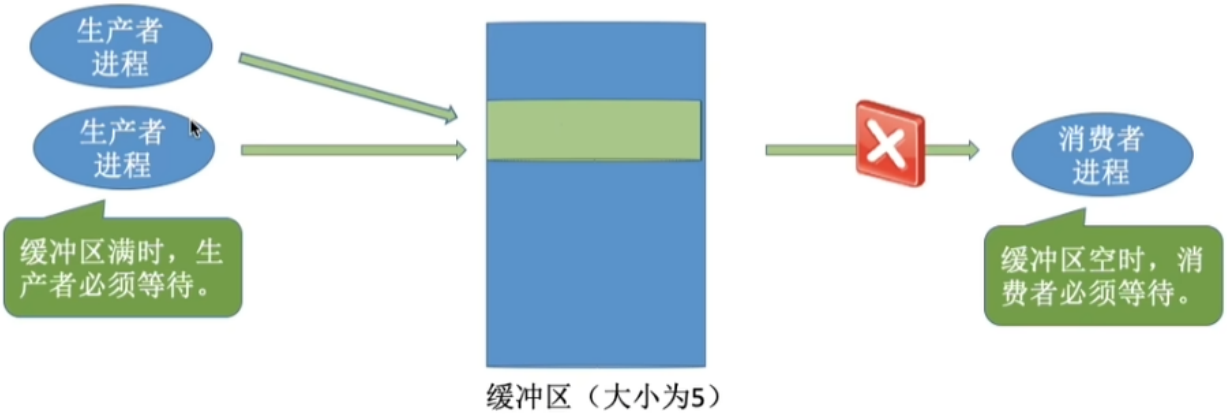
易错点：实现互斥和实现同步的两个P操作的先后顺序（死锁问题）



问题分析

系统中有一组生产者进程和一组消费者进程，生产者进程每次生产一个产品放入缓冲区，消费者进程每次从缓冲区取出一个产品使用。（注：这里的”产品“理解为某种数据）

生产者、消费者共享一个初始为空、大小为n的缓冲区。



只有缓冲区没满时，生产者才能把产品放入缓冲区，否则必须等待。

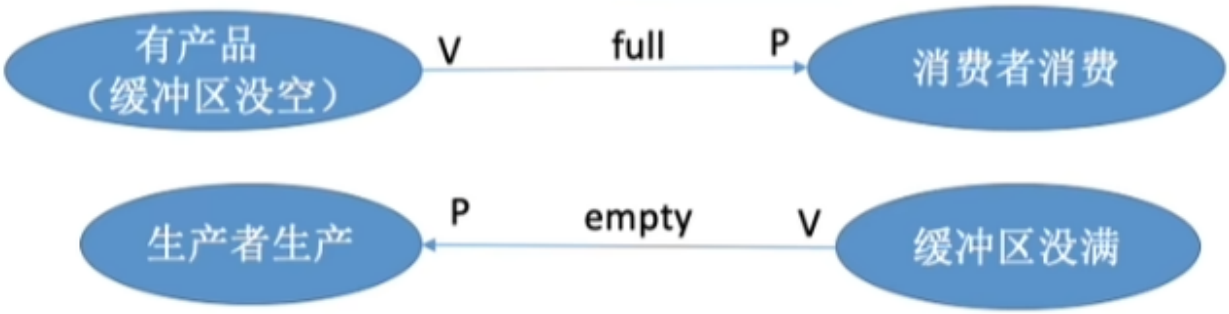
缓冲区没满->生产者生产

只有缓冲区不空时，消费者才能从中取出产品，否则必须等待。

缓冲区没空->消费者消费

缓冲区是临界资源，各进程必须互斥地访问。

互斥关系



1. 关系分析。找出题目中描述的各个进程，分析它们之间的同步、互斥关系。
2. 整理思路。根据各进程的操作流程确定P、V操作的大致顺序。
3. 设置信号量。并根据题目条件确定信号量初值。（互斥信号量初值一般为1，同步信号量的初始值要看对应资源的初始值是多少）

如何实现

- 生产者、消费者共享一个初始为空、大小为n的缓冲区。
- 只有缓冲区没满时，生产者才能把产品放入缓冲区，否则必须等待。
- 只有缓冲区不空时，消费者才能从中取出产品，否则必须等待。
- 缓冲区是临界资源，各进程必须互斥地访问。

```
semaphore mutex = 1;//互斥信号量,实现对缓冲区的互斥访问
semaphore empty = n;//同步信号量,表示空闲缓冲区的数量
semaphore full = 0;//同步信号量,表示产品的数量,也即非空缓冲区的数量
```

```
producer(){
    while(1){
        //生产一个产品;
        P(empty);//消耗一个空闲缓冲区
        P(mutex);
        //把产品放入缓冲区;
        V(mutex);
        V(full);//增加一个产品
    }
}
consumer(){
    while(1){
        P(full);//消耗一个产品（非空缓冲区）
        P(mutex);
        //从缓冲区取出一个产品;
```

```
        V(mutex);
        V(empty);//增加一个空闲缓冲区
        //使用产品;
    }
}
```

实现互斥是在同一进程中进行一对PV操作

思考：能否改变相邻P、V操作的顺序?

mutex的P操作在前

```
producer(){
    while(1){
        //生产一个产品;
        P(mutex);//1mutex的P操作在前
        P(empty);//2消耗一个空闲缓冲区
        //把产品放入缓冲区;
        V(mutex);
        V(full);//增加一个产品
    }
}
consumer(){
    while(1){
        P(mutex);//3
        P(full);//4消耗一个产品（非空缓冲区）
        //从缓冲区取出一个产品;
        V(mutex);
        V(empty);//增加一个空闲缓冲区
        //使用产品;
    }
}
```

若此时缓冲区内已经放满产品，则empty=0，full=n。

则生产者进程执行1使mutex变为0，再执行2，由于已没有空闲缓冲区，因此生产者被阻塞。

由于生产者阻塞，因此切换回消费者进程。消费者进程执行3，由于mutex为0，即生产者还没释放对临界资源的"锁"，因此消费者也被阻塞。

这就造成了生产者等待消费者释放空闲缓冲区，而消费者又等待生产者释放临界区的情况，生产者和消费者循环等待被对方唤醒，出现"死锁"。

因此，实现互斥的P操作一定要在实现同步的P操作之后。

V操作不会导致进程阻塞，因此两个V操作顺序可以交换。

多生产者-多消费者问题

解决"多生产者-多消费者"的关键在于理清复杂的同步关系。

在分析同步问题（一前一后问题）的时候不能从单个进程行为的角度来分析，要把"一前一后"发生的事看作是两种"事件"的前后关系。

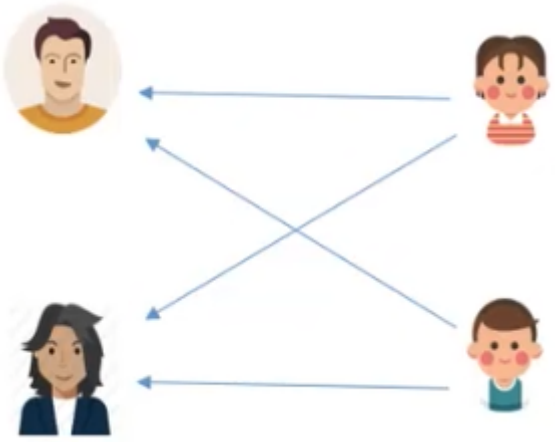
比如，如果从单个进程行为的角度来考虑的话，我们会有以下结论：

如果盘子里装有苹果，那么一定要女儿取走苹果后父亲或母亲才能再放入水果

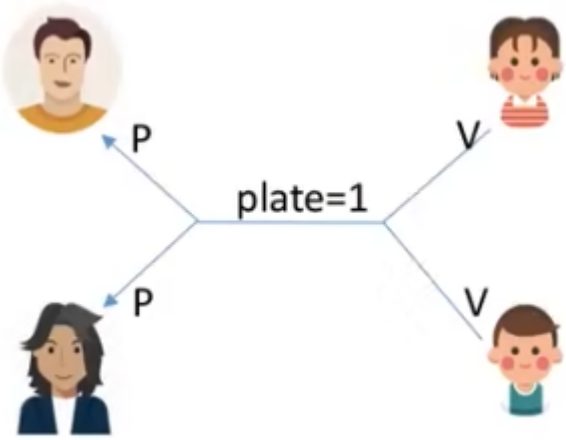
如果盘子里装有橘子，那么一定要儿子取走橘子后父亲或母亲才能再放入水果

这么看是否就意味着要设置四个同步信号量分别实现这四个”一前一后“的关系了？

正确的分析方法应该从”事件“的角度来考虑，我们可以把上述四对”进程行为的前后关系”抽象为一对“事件的前后关系”



盘子变空事件->放入水果事件。“盘子变空事件”既可由儿子引发，也可由女儿引发； “放水果事件”既可能是父亲执行，也可能是母亲执行。这样的话，就可以用一个同步信号量解决问题了

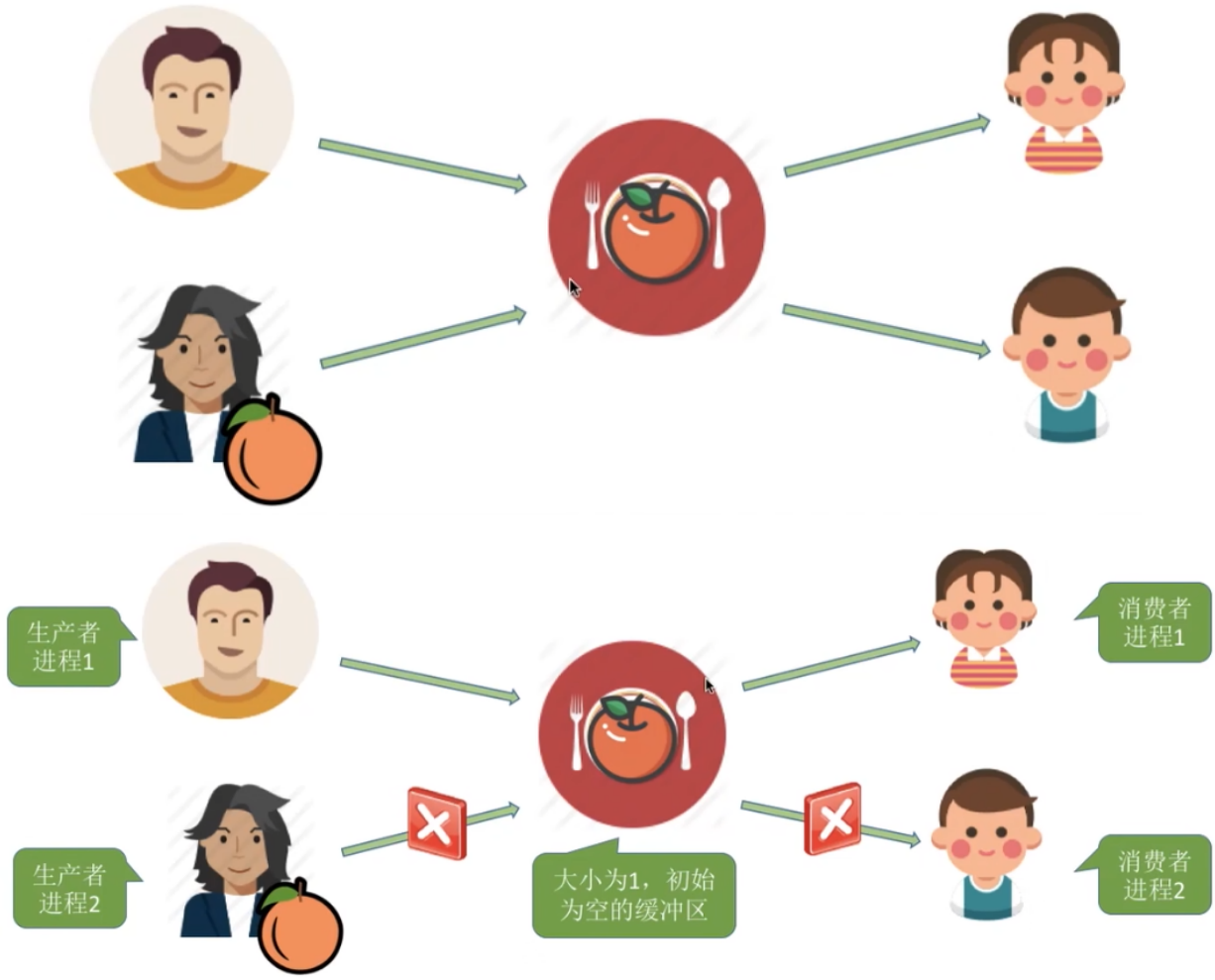


在生产者-消费者问题中，如果缓冲区大小为1，那么有可能不需要设置互斥信号量就可以实现互斥访问缓冲区的功能。当然，这不是绝对的，要具体问题具体分析。

如果来不及仔细分析，可以加上互斥信号量，保证各进程一定会互斥地访问缓冲区。但需要注意的是，实现互斥的P操作一定要在实现同步的P操作之后，否则可能引起”死锁“。

问题描述

桌子上有一只盘子，每次只能向其中放入一个水果。爸爸专向盘子中放苹果，妈妈专向盘子中放橘子，儿子专等着吃盘子中的橘子，女儿专等着吃盘子中的苹果。只有盘子空时，爸爸或妈妈才可向盘子中放一个水果。仅当盘子中有自己需要的水果时，儿子或女儿可以从盘子中取出水果。用PV操作实现上述过程。



1. 关系分析。找出题目中描述的各个进程，分析它们之间的同步、互斥关系。

   互斥关系（mutex=1）：对缓冲区（盘子）的访问要互斥地进行

   同步关系（一前一后）：

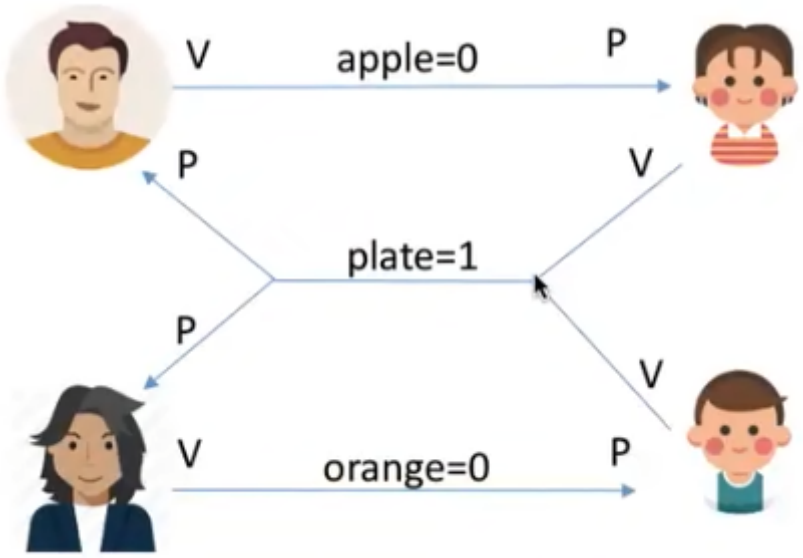
   - 父亲将苹果放入盘子后，女儿才能取苹果
   - 母亲将橘子放入盘子后，儿子才能取橘子
   - 只有盘子为空时，父亲或母亲才能放入水果

     ”盘子为空“这个事件可以由儿子或女儿触发，事件发生后才允许父亲或母亲放水果

2. 整理思路。根据各进程的操作流程确定P、V操作的大致顺序。

   互斥：在临界区前后分别PV同步：前V后P

3. 设置信号量。设置需要的信号量，并根据题目条件确定信号量初值。（互斥信号量初值一般为1，同步信号量的初始值要看对应资源的初始值是多少）



如何实现

```
semaphore mutex = 1;//实现互斥访问盘子（缓冲区）
semaphore apple = 0;//盘子中有几个苹果
semaphore orange = 0;//盘子中有几个橘子
semaphore plate = 1;//盘子中还可以放多少个水果
```

```
dad(){
    while(1){
        //准备一个苹果;
        P(plate);
        P(mutex);
        //把苹果放入盘子;
        V(mutex);
        V(apple);
    }
}
```

```
mom(){
    while(1){
        //准备一个橘子;
        P(plate);
        P(mutex);
        //把橘子放入盘子;
        V(mutex);
        V(orange);
    }
}
```

```
daughter(){
    while(1){
        P(apple);
        P(mutex);
        //从盘中取出苹果;
        V(mutex);
        V(plate);
        //吃掉苹果;
    }
}
```

```
son(){
    while(1){
        P(orange);
        P(mutex);
        //从盘中取出橘子;
        V(mutex);
        V(plate);
        //吃掉橘子;
    }
}
```

问题：可不可以不用互斥信号量?

分析：刚开始，儿子、女儿进程即使上处理机运行也会被阻塞。如果刚开始是父亲进程先上处理机运行，则：

父亲P(plate)，可以访问盘子->

母亲P(plate)，阻塞等待盘子->

父亲放入苹果V(apple)，女儿进程被唤醒，其他进程即使运行也都会阻塞，暂时不可能访问临界资源（盘子）->

女儿P(apple)，访问盘子，V(plate)，等待盘子的母亲进程被唤醒->

母亲进程访问盘子（其他进程暂时都无法进入临界区）->...

原因在于：本题中的缓冲区大小为1，在任何时刻，apple、orange、plate三个同步信号量中最多只有一个是1.因此在任何时刻，最多只有一个进程的P操作不会被阻塞，并顺利地进入临界区

问题：如果盘子（缓冲区）容量为2

父亲P(plate)，可以访问盘子->

母亲P(plate)，可以访问盘子->

父亲在往盘子里放苹果，同时母亲也可以往盘子里放橘子。于是就出现了两个进程同时访问缓冲区的情况，有可能导致两个进程写入缓冲区的数据相互覆盖的情况。因此，如果缓冲区大小大于1，就必须专门设置一个互斥信号量mutex来保证互斥访问缓冲区。

吸烟者问题

吸烟者问题可以为我们解决“可以生产多个产品的单生产者”问题提供一个思路。

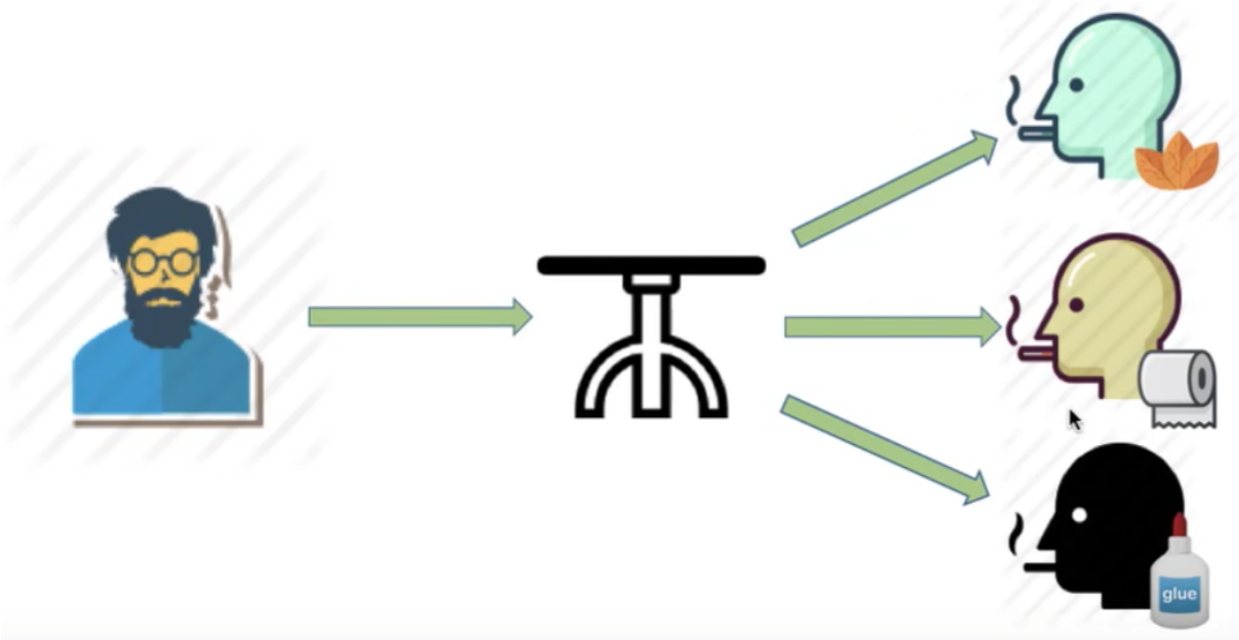
“轮流让各个吸烟者吸烟”必然需要“轮流的在桌上放上组合一、二、三”，注意体会如何用一个整型变量实现这个“轮流”过程的。

对比“每次随机地让一个吸烟者吸烟”

若一个生产者要生产多种产品（或者说会引发多种前驱事件），那么各个V操作应该放在各自对应的“事件”发生之后的位置。

问题描述

假设一个系统中有三个抽烟者进程和一个供应者进程。每个抽烟者不停地卷烟并抽掉它，但是要卷起并抽掉一支烟，抽烟者需要有三种材料：烟草、纸和胶水。三个抽烟者中，第一个拥有烟草、第二个拥有纸、第三个拥有胶水。供应者进程无限地提供三种材料，供应者每次将两种材料放桌子上，拥有剩下那种材料的抽烟者卷一根烟并抽掉它，并给供应者进程一个信号告诉完成了，供应者就会放另外两种材料在桌上，这个过程一直重复（让三个抽烟者轮流地抽烟）



本质上这题也属于”生产者-消费者“问题，更详细的说应该是”可生产多种产品的单生产者-多消费者“。

1. 关系分析。找出题目中描述的各个进程，分析它们之间的同步、互斥关系。
    - 桌子可以抽象为容量为1的缓冲区，要互斥访问

      组合一：纸+胶水

      组合二：烟草+胶水

      组合三：烟草+纸
    - 同步关系（从事件的角度来分析）：

      桌子上有组合一->第一个抽烟者取走东西

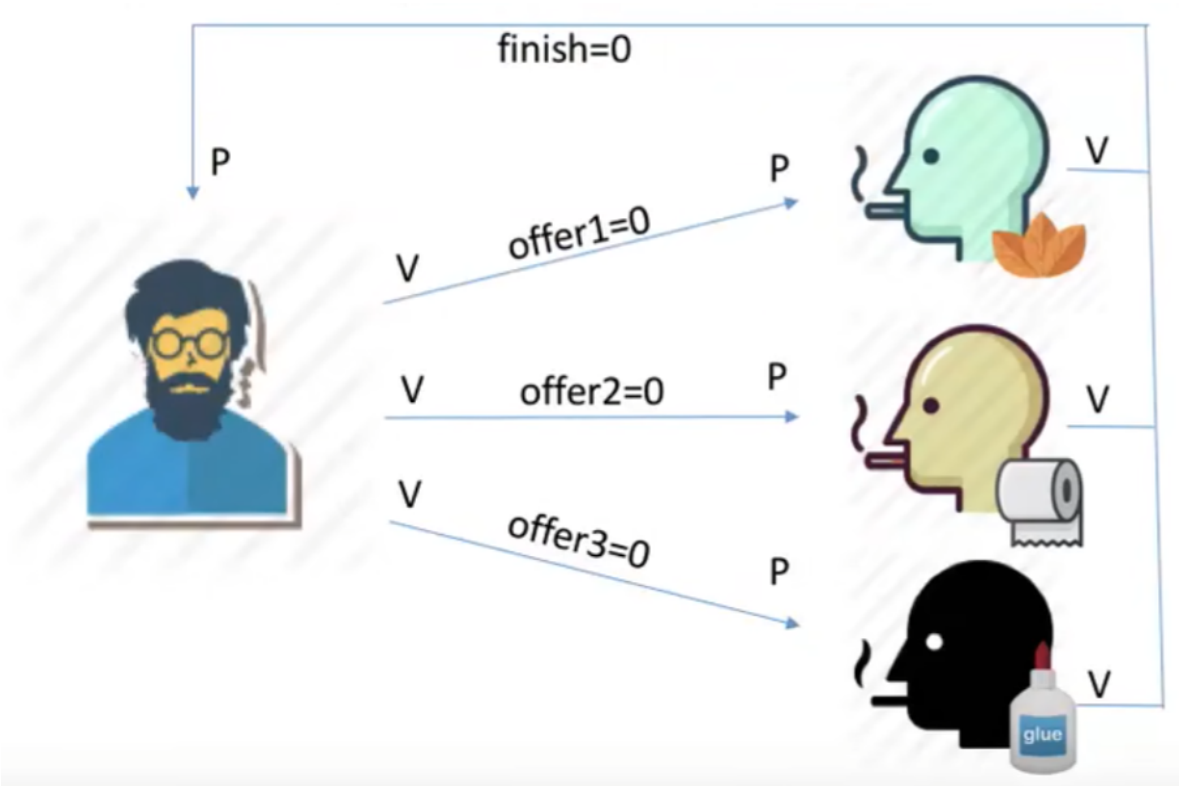
      桌子上有组合二->第二个抽烟者取走东西

      桌子上有组合三->第三个抽烟者取走东西

      发出完成信号->供应者将下一个组合放到桌上

PV操作顺序：“前V后P”

2. 整理思路。根据各进程的操作流程确定P、V操作的大致顺序



- 桌子上有组合一->第一个抽烟者取走东西
- 桌子上有组合二->第二个抽烟者取走东西
- 桌子上有组合三->第三个抽烟者取走东西
- 发出完成信号->供应者将下一个组合放到桌上

如何实现

```
semaphore offer1 = 0;//桌上组合一的数量
semaphore offer2 = 0;//桌上组合二的数量
semaphore offer3 = 0;//桌上组合三的数量
semaphore finish = 0;//抽烟是否完成
int i = 0;//用于实现“三个抽烟者轮流抽烟”
```

```
provider(){
    while(1){
        if(i==0){
            //将组合一放桌上;
            V(offer1);
        } else if(i==1){
            //将组合二放桌上;
            V(offer2);
        } else if(i==2){
            //将组合三放桌上;
            V(offer3);
        }
```

```
        i = (i+1)%3;
        P(finish);
    }
}
```

```
smoker1(){
    while(1){
        P(offer1);
        //从桌上拿走组合一;卷烟;抽掉;
        V(finish)
    }
}
```

```
smoker2(){
    while(1){
        P(offer2);
        //从桌上拿走组合二;卷烟;抽掉;
        V(finish)
    }
}
```

```
smoker3(){
    while(1){
        P(offer3);
        //从桌上拿走组合三;卷烟;抽掉;
        V(finish)
    }
}
```

是否需要设置一个专门的互斥信号量?

缓冲区大小为1，同一时刻，四个同步信号量中至多有一个的值为1

读者写者问题

读者-写者问题为我们解决复杂的互斥问题提供了一个参考思路。

其核心思想在于设置了一个计数器count用来记录当前正在访问共享文件的读进程数。我们可以用count的值来判断当前进入的进程是否是第一个/最后一个读进程，从而做出不同的处理。

另外，对count变量的检查和赋值不能一气呵成导致了一些错误，如果需要实现“一气呵成”，自然应该想到用互斥信号量。

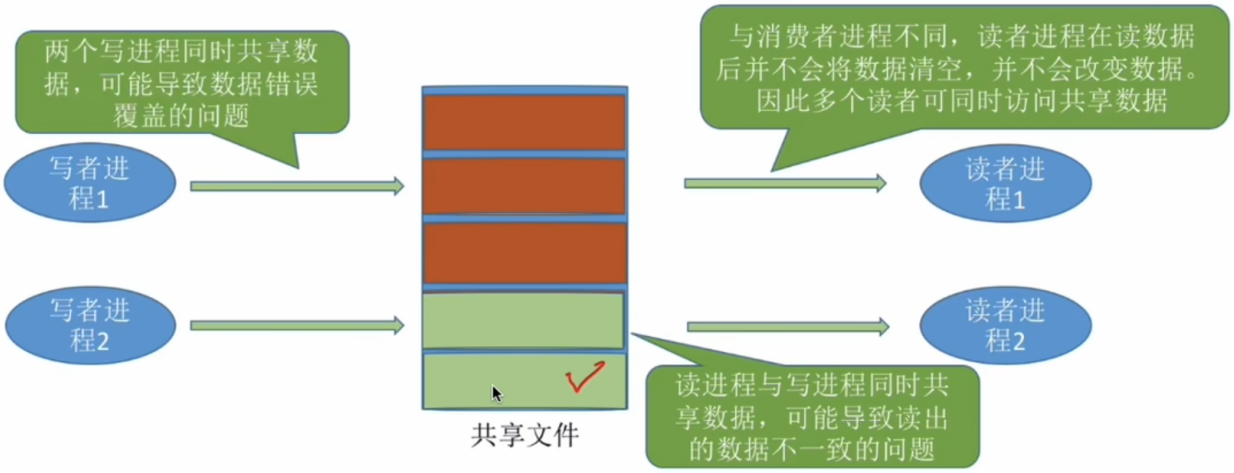
认真体会我们是如何解决“写进程饥饿”问题的。

PV操作题目可以用之前介绍的几种生产者-消费者问题的思想来解决，如果遇到更复杂的问题，可以想想能否用读者写者问题的这几个思想来解决。

问题描述

有读者和写者两组并发进程，共享一个文件，当两个或两个以上的读进程同时访问共享数据时不会产生副作用，但若某个写进程和其他进程（读进程或写进程）同时访问共享数据时则可能导致数据不一致的错误。因此要求：

1. 允许多个读者可以同时对文件执行读操作；
2. 只允许一个写者往文件中写信息；
3. 任一写者在完成写操作之前不允许其他读者或写者工作；
4. 写者执行写操作前，应让已有的读者和写者全部退出。



- 与消费者进程不同，读者进程在读数据后并不会将数据清空，并不会改变数据。因此多个读者可同时访问共享数据
- 读进程与写进程同时共享数据，可能导致读出的数据不一致的问题
- 两个写进程同时共享数据，可能导致数据错误覆盖的问题

1. 关系分析。找出题目中描述的各个进程，分析它们之间的同步、互斥关系。
2. 整理思路。根据各进程的操作流程确定P、V操作的大致顺序
3. 设置信号量。设置需要的信号量，并根据题目条件确定信号量初值。（互斥信号量初值一般为1，同步信号量的初始值要看对应资源的初始值是多少）

两类进程：写进程、读进程

互斥关系：写进程-写进程、写进程-读进程。读进程与读进程不存在互斥问题。

如何实现

```
semaphore rw = 1;//用于实现对共享文件的互斥访问
int count = 0;//记录当前有几个读进程在访问文件
semaphore mutex = 1;//用于保证对count变量的互斥访问
semaphore w = 1;//用于实现“写优先”
```

```
writer(){
    while(1){
        P(w);
        P(rw);//写之前“加锁”
        //写文件...
        V(rw);//写完了“解锁”
        V(w);
    }
}
```

```
reader(){
    while(1){
        P(w);
        P(mutex);//各读进程互斥访问count
        if(count==0)//由第一个读进程负责
            P(rw);//读之前“加锁”
        count++;//访问文件的读进程数+1
        V(mutex);
        V(w);
        //读文件...
        P(mutex);//各读进程互斥访问count
        count--;//访问文件的读进程数-1
        if(count==0)//由最后一个读进程负责
            V(rw);//读完了“解锁”
        V(mutex);
    }
}
```

思考：若两个读进程并发执行，则count=0时两个进程也许都能满足if条件，都会执行P(rw)，从而使第二个读进程阻塞的情况。

如何解决：出现上述问题的原因在于对count变量的检查和赋值无法一气呵成，因此可以设置另一个互斥信号量来保证各读进程对count的访问是互斥的。

潜在的问题：只要有读进程还在读，写进程就要一直阻塞等待，可能“饿死”。因此，这种算法中，读进程是优先的。

分析以下并发执行P(w)的情况：

读者1->读者2

写者1->写者2

写者1->读者1

读者1->写者1->读者2

结论：在这种算法中，连续进入的多个读者可以同时读文件；写者和其他进程不能同时访问文件；写者不会饥饿，但也并不是真正的“写优先”，而是相对公平的先来先服务原则。有的书上把这种算法称为“读写公平法”。

哲学家进餐问题

哲学家进餐问题的关键在于解决进程死锁。

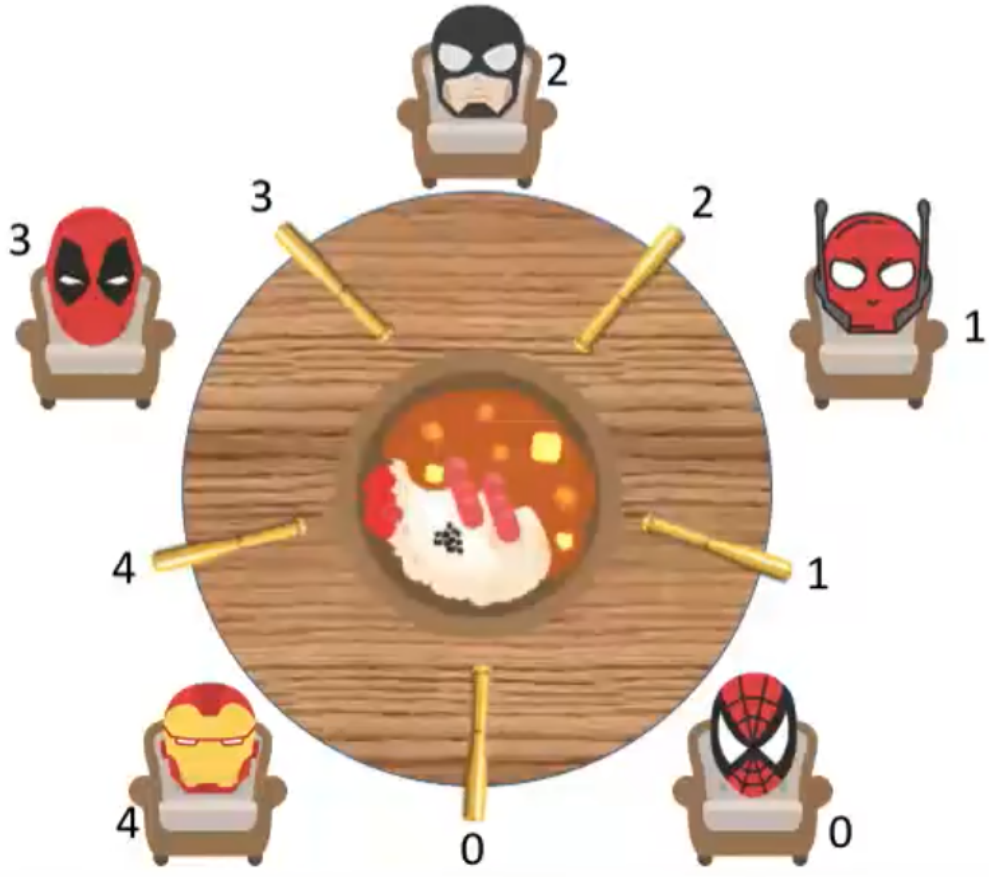
这些进程之间只存在互斥关系，但是与之前接触到的互斥关系不同的是，每个进程都需要同时持有两个临界资源，因此就有“死锁”问题的隐患。

如果遇到了一个进程需要同时持有多个临界资源的情况，应该参考哲学家问题的思想，分析给出的进程之间是否会发生循环等待，是否会发生死锁。

可以参考哲学家就餐问题解决死锁的三种思路。

问题描述

一张圆桌上坐着5名哲学家，每两个哲学家之间的桌上摆一根筷子，桌子的中间是一碗米饭。哲学家们倾注毕生的精力用于思考和进餐，哲学家在思考时，并不影响他人。只有当哲学家饥饿时，才试图拿起左、右筷子（一根一根地拿起）。如果筷子已在他人手上，则需等待。饥饿的哲学家只有同时拿起两根筷子才可以开始进餐，当进餐完毕后，放下筷子继续思考。



1. 关系分析。系统中有5个哲学家进程，5位哲学家与左右邻居对其中间筷子的访问是互斥关系。
2. 整理思路。这个问题中只有互斥关系，但与之前遇到的问题不同的是，每个哲学家进程需要同时持有两个临界资源才能开饭。如何避免临界资源分配不当造成的死锁现象，是哲学家问题的精髓。
3. 信号量设置。定义互斥信号量数组用于实现对5个筷子的互斥访问。并对哲学家按0~4编号，哲学家i左边的筷子编号为i，右边的筷子编号为(i+1)%5。

每个哲学家吃饭前依次拿起左、右两支筷子

```
semaphore chopstick[5] = {1,1,1,1,1};
Pi(){//i号哲学家的进程
    while(1){
        P(chopstick[i]);//拿左
        P(chopstick[(i+1)%5]);//拿右
        //吃饭...
        V(chopstick[i]);//放左
        V(chopstick[(i+1)%5]);//放右
        //思考...
    }
}
```

如果5个哲学家并发地拿起了自己左手边地筷子，每位哲学家循环等待右边的人放下筷子（阻塞）。发生“死锁”

如何实现

如何防止死锁的发生?

1. 可以对哲学家进程施加一些限制条件，比如最多允许四个哲学家同时进餐。这样可以保证至少有一个哲学家是可以拿到左右两只筷子的
2. 要求奇数号哲学家先拿左边的筷子，然后再拿右边的筷子，而偶数号哲学家刚好相反。用这种方法可以保证如果相邻的两个奇偶号哲学家都想吃饭，那么只会有其中一个可以拿起第一只筷子，另一个会直接阻塞。这就避免了占有一支后再等待另一只的情况。
3. 仅当一个哲学家左右两支筷子都可用时才允许他抓起筷子。

```
semaphore chopstick[5] = {1,1,1,1,1};
semaphore mutex = 1;//互斥地取筷子
Pi(){//i号哲学家的进程
    while(1){
        P(mutex);
        P(chopstick[i]);//拿左
        P(chopstick[(i+1)%5]);//拿右
        V(mutex);
        //吃饭...
        V(chopstick[i]);//放左
        V(chopstick[(i+1)%5]);//放右
        //思考...
    }
}
```

这种方法并不能保证只有两边的筷子都可用时，才允许哲学家拿起筷子。

更准确的说法是：各哲学家拿筷子这件事必须互斥的执行。这就保证了即使一个哲学家在拿筷子拿到一半时被阻塞，也不会有别的哲学家会继续尝试拿筷子。这样的话，当前正在吃饭的哲学家放下筷子后，被阻塞的哲学家就可以获得等待的筷子了。