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);
        P(mutex);
        把产品放入缓冲区;
        V(mutex);
        V(full);
    }
}
```

```
consumer (){
    while(1){
        P(full);
        P(mutex);
        从缓冲区取出一个产品;
        V(mutex);
        V(empty);
        使用产品;
    }
}
```

实现互斥是在同一进程中进行一对PV操作

消耗一个空闲缓冲区

增加一个产品

消耗一个产品（非空缓冲区）

增加一个空闲缓冲区

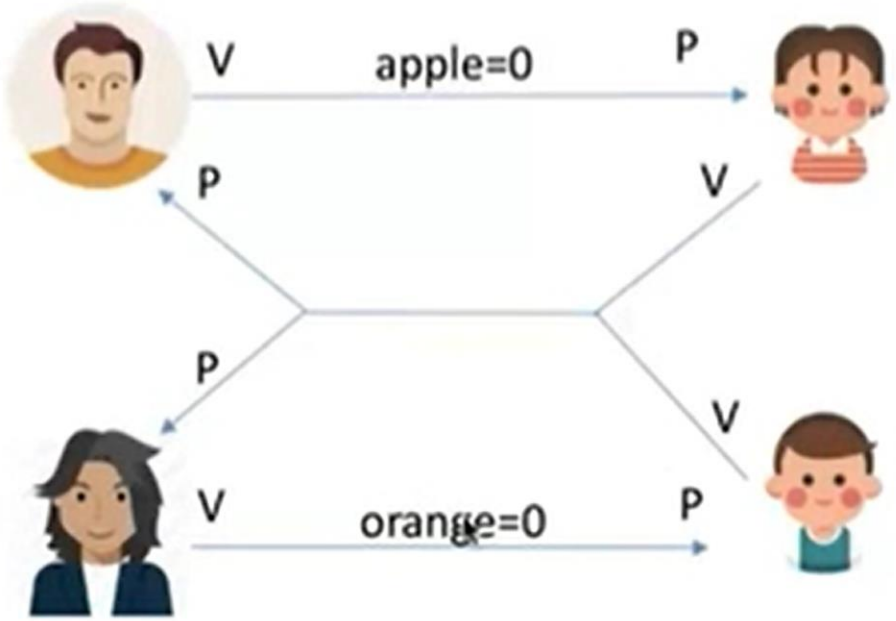
实现两进程的同步关系，是在其中一个进程中执行P，另一进程中执行V

问题分析

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。

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、
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）

互斥：在临界区前后分别PV
同步：前V后P

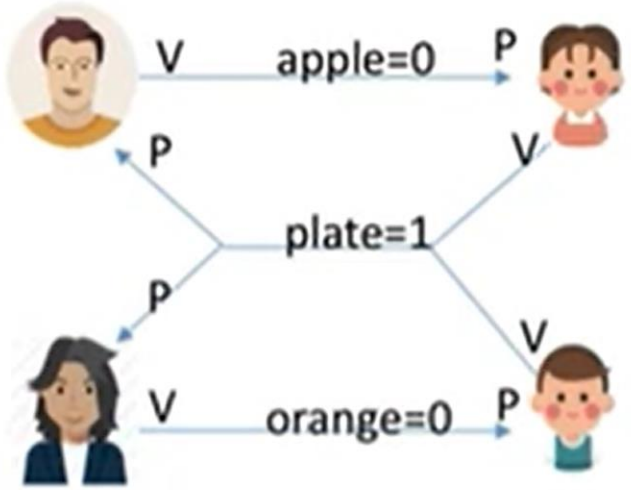


互斥关系：(mutex = 1)
对缓冲区（盘子）的访问要互斥地进行

同步关系（一前一后）：
1. 父亲将苹果放入盘子后，女儿才能取苹果
2. 母亲将橘子放入盘子后，儿子才能取橘子
3. 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

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;
    }
}
```

知识回顾与重要考点

总结：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。

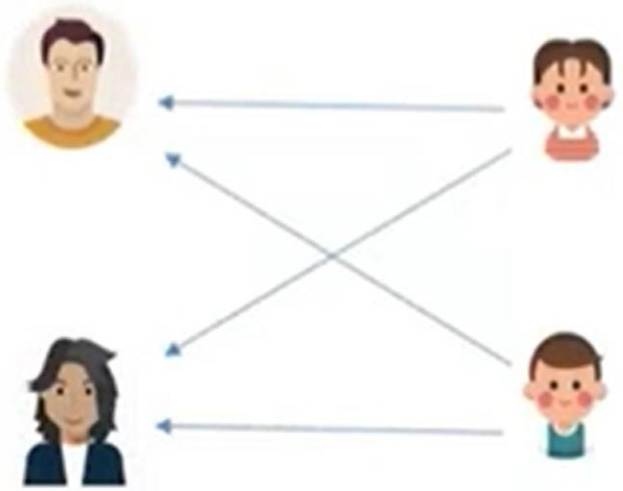
建议：在考试中如果来不及仔细分析，可以加上互斥信号量，保证各进程一定会互斥地访问缓冲区。但需要注意的是，实现互斥的P操作一定要在实现同步的P操作之后，否则可能引起“死锁”。

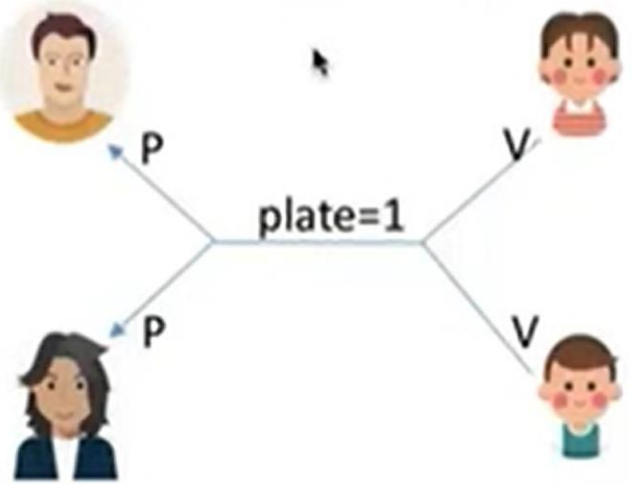
PV 操作题目的解题思路：

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，同步信号量的初始值要看对应资源的初始值是多少）

知识回顾与重要考点

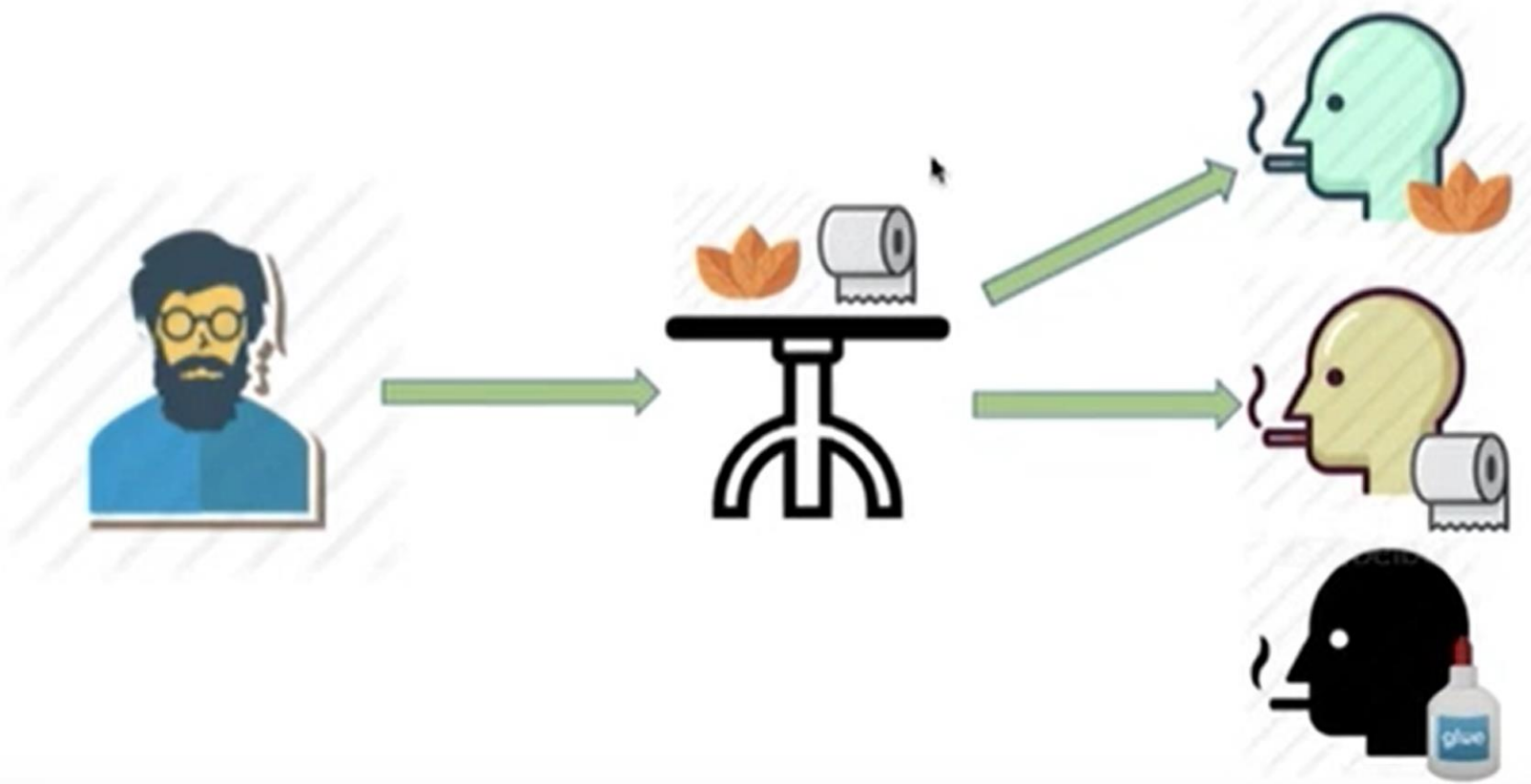
解决“多生产者-多消费者问题”的关键在于理清复杂的同步关系。
在分析同步问题（一前一后问题）的时候不能从单个进程行为的角度来分析，要把“一前一后”发生的事看做是两种“事件”的前后关系。
比如，如果从单个进程行为的角度来考虑的话，我们会有以下结论：
如果盘子里装有苹果，那么一定要女儿取走苹果后父亲或母亲才能再放入水果
如果盘子里装有橘子，那么一定要儿子取走橘子后父亲或母亲才能再放入水果
这么看是否就意味着要设置四个同步信号量分别实现这四个“一前一后”的关系了？
正确的分析方法应该从“事件”的角度来考虑，我们可以把上述四对“进程行为的前后关系”抽象为一对“事件的前后关系”
盘子变空事件→放入水果事件。“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





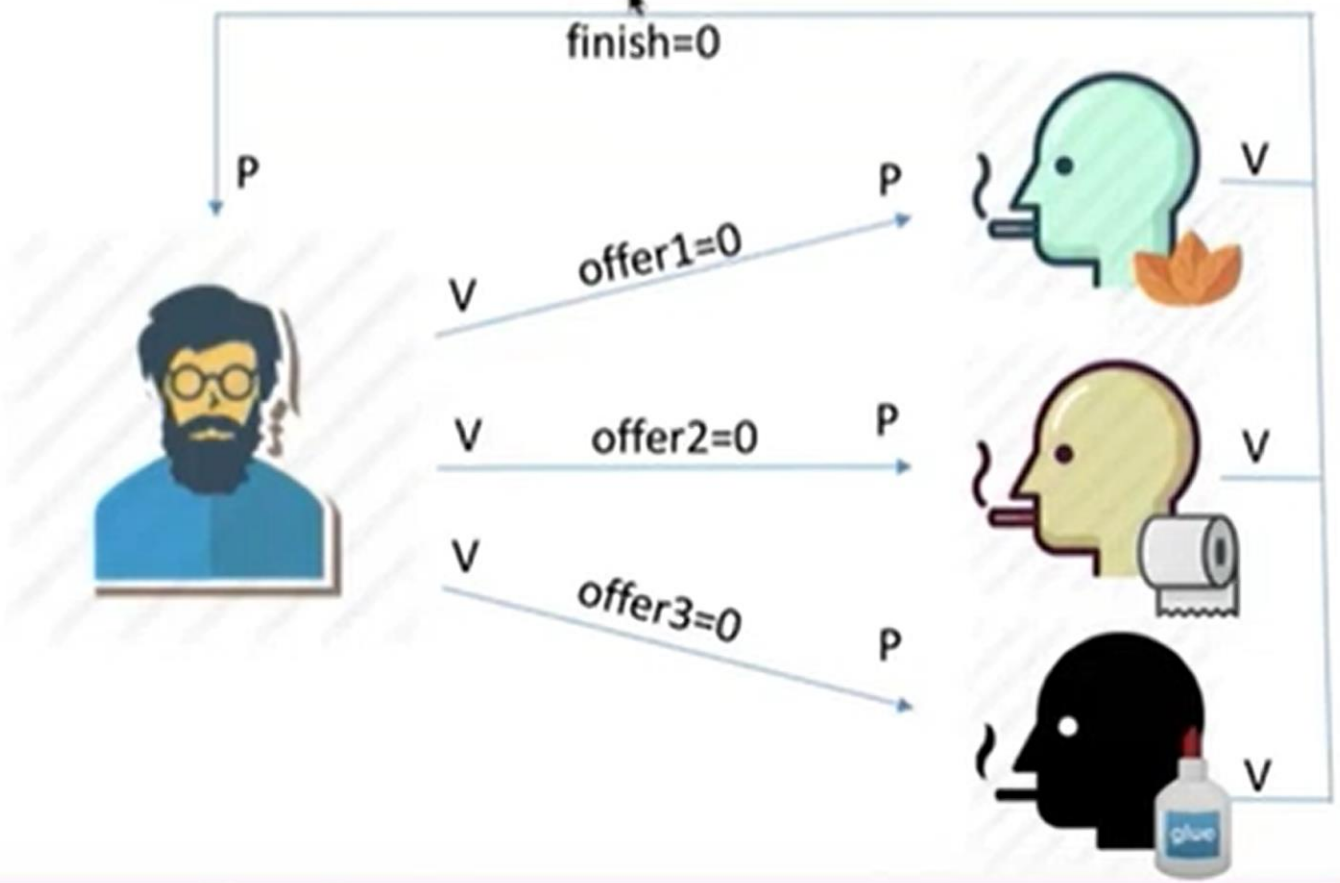
问题描述

假设一个系统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再桌上，这个过程一直重复（让三个抽烟者轮流地抽烟）



问题分析

假设一个系统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再桌上，这个过程一直重复（让三个抽烟者轮流地抽烟）

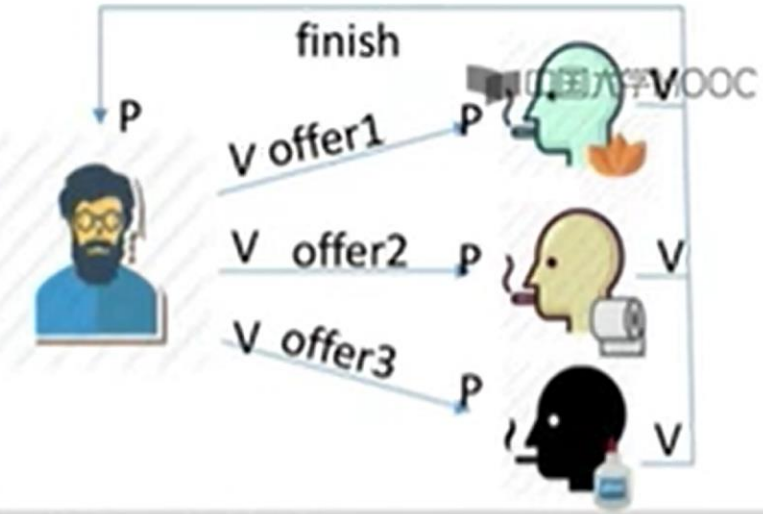


桌上有组合一 → 第一个抽烟者取走东西
桌上有组合二 → 第二个抽烟者取走东西
桌上有组合三 → 第三个抽烟者取走东西
发出完成信号 → 供应者将下一个组合放到桌上

如何实现

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




```
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);
    }
}
```

```
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"
```

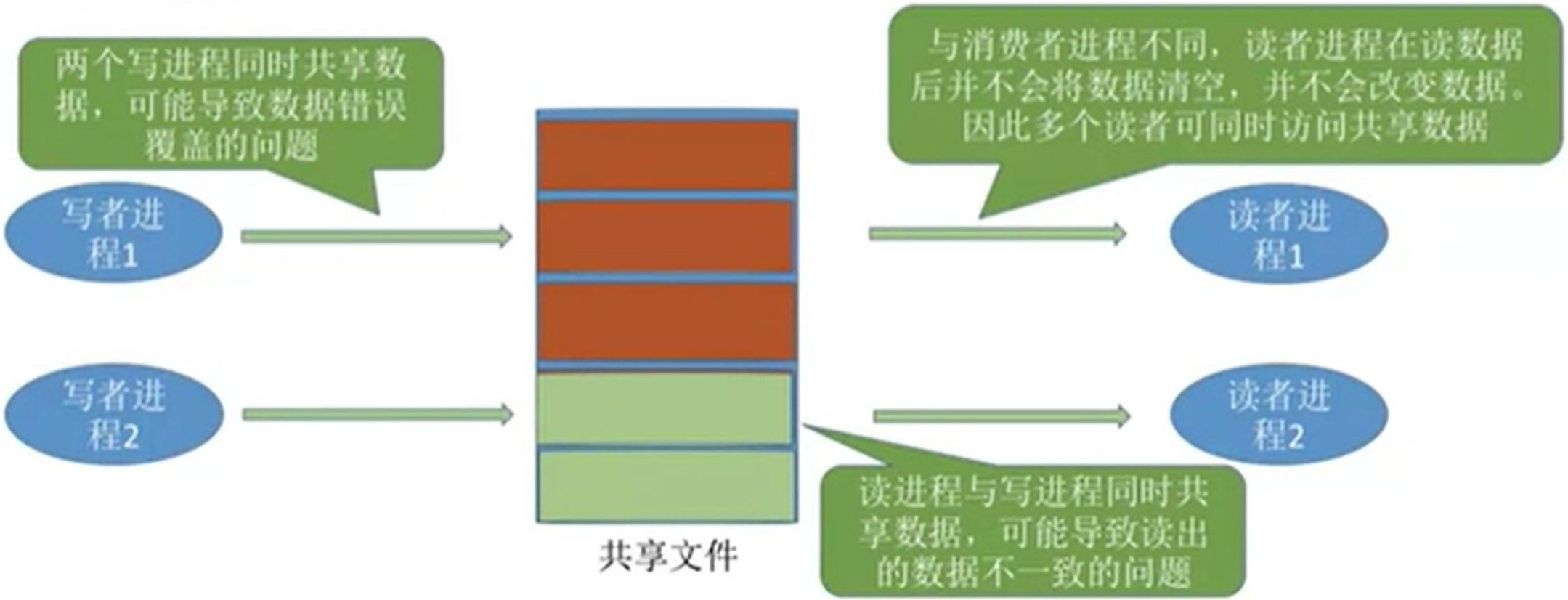
```
smoker1 (){
    while(1){
        P(offer1);
        从桌上拿走组合一；卷烟；抽掉；
        V(finish);
    }
}
```

```
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);
    }
}
```

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。



如何实现


```
semaphore rw=1;        //用于实现对文件的互斥访问。表示当前是否有进程在访问共享文件
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
```

```
writer (){
    while(1){
        P(rw);     //写之前"加锁"
        写文件...
        V(rw);     //写之后"解锁"
    }
}
```

思考：若两个读进程并发执行，则两个读进程有可能先后执行 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。

```
reader (){
    while(1){
        P(mutex);      //各读进程互斥访问count
        if(count==0)
            P(rw);     //第一个读进程负责"加锁"
        count++;       //访问文件的读进程数+1
        V(mutex);
        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)
            V(rw);     //最后一个读进程负责"解锁"
        V(mutex);
    }
}
```

如何实现


```
semaphore rw=1;          //用于实现对文件的互斥访问
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"
```

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

```
writer (){
    while(1){
        P(w);
        P(rw);
        写文件...
        V(rw);
        V(w);
    }
}
```

```
reader (){
    while(1){
        P(w);
        P(mutex);
        if(count==0)
            P(rw);
        count++;
        V(mutex);
        V(w);
        读文件...
        P(mutex);
        count--;
        if(count==0)
            V(rw);
        V(mutex);
    }
}
```

问题分析

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、右两根筷子（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。

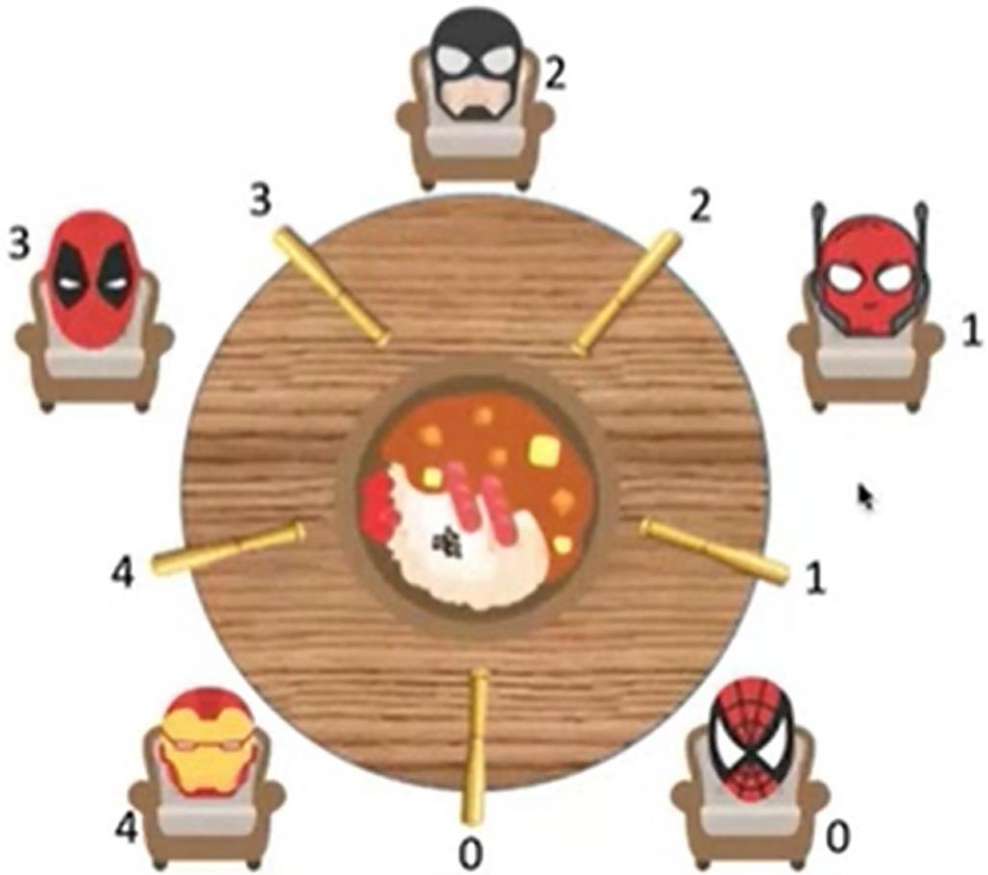

```
semaphore chopstick[5]={1,1,1,1,1};
Pi (){          //i号哲学家的进程
   while(1){
       P(chopstick[i]);              //拿左
       P(chopstick[(i+1)%5]);        //拿右
       吃饭…
       V(chopstick[i]);              //放左
       V(chopstick[(i+1)%5]);        //放右
       思考…
   }
}
```

每位哲学家循环等待右边的人放下筷子（阻塞）。发生“死锁”

如果5个哲学家并发地拿起了自己左手边的筷子…

如何实现

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、右两根筷子（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。



如何防止死锁的发生呢？

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。

如何实现

哲学家，每两个哲学家之间的桌上摆一
用于思考和进餐，哲学家在思考时，
根筷子（一根一根地拿起）。如果筷
根筷子才可以开始进餐，当进餐完毕后

因此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。

更准确的说法应该是：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。


```
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…
   }
}
```