使用 PV 信号量互斥同步解决如下问题：
设某计算进程 CP 和打印进程 IOP 共用一个大小为 N 的 T
CP 进度负责不断地计算数据并送入长度 T 中
IOP 进程负责不断地从 T 中取出数据去打印

```
//不妨设 T 的大小为 N，初始前缀为空
信号量 s_empty = N；
信号量 s_full = 0 ;
//可以保证 s_empty + s_full 恒等于 N
semaphore mutex = 1; //互斥信号量
void   Computing_Process（）{
    而（Computing_Finish（）== false）{
        计算（）；
        P（s_empty）；
        P（mutex）；
        Send_To_Buffer（）；
        V（mutex）；
        V（s_full）；
    }
}
void   Input_Output_Process（）{
    而（Output_Finish（）== false）{
        P（s_full）；
        P（mutex）；
        Get_From_Buffer（）；
        V（mutex）；
        V（s_empty）；
        输出（）；
    }
}
```

吃水果

使用 PV 信号量互斥同步解决如下问题：
桌上有一个空盘，最多允许存放 N 只水果。爸爸可向盘中放一个苹果或者放一个桔子。儿子专等吃盘中的桔子，女儿专等吃盘子中的苹果。
试用 PV 操作实现爸爸，儿子，女儿三个并发进程的同步：

```
//不妨设盘子的大小为 N，初始盘子为空
信号量 s_empty = N；//表示盘子是否为空，初始值 N
信号量 s_orange = 0 ; //表示当前盘子中的桔子数量，初始数量 0
信号量 s_apple = 0 ; //表示当前盘子中的苹果数量，初始数量 0
//可以保证 s_empty + s_orange + s_apple 恒等于 N
信号量互斥= 1 ; //互斥信号量
```

```
   父亲（）{
    而（true）{
        水果 f = Buy_Fruit（）；
        P（s_empty）；
        P（互斥体）；
        Put_On_Plate（f）；
        V（互斥体）；
        如果（f ==橙色）{
            V（s_orange）；
        } 其他 {
            V（s_apple）；
        }
    }
}
     儿子（）{
    而（true）{
        P（s_orange）；
        P（互斥体）；
        Get_Orange_From_Plate（）；
        V（互斥体）；
        V（s_empty）；
        Eat_Orange（）；
    }
}
    女儿（）{
    而（true）{
        P（s_apple）；
        P（互斥体）；
        Get_Apple_From_Plate（）；
        V（互斥体）；
        V（s_empty）；
        Eat_Apple（）；
    }
}
```

哲学家

使用 PV 信号量互斥同步解决如下问题：

有 N 个哲学家，他们的生活方式是交替地进行思考和进餐

哲学家们共用一张圆桌，分别坐在周围的 N 张椅子上，在圆桌上有 N 个碗和五支筷子

平时哲学家进行思考，干旱时便试图取其左，右最靠近他的筷子，只有在他拿到两支筷子时才能进餐

该哲学家进餐完成后，放下左右两只筷子又继续思考。

```
class   Monitor { //根据问题创造管程

    //注意：哲学家 id 依次为[0，N-1]
    Philosopher_State 状态[N]; //哲学家的状态有思考，饮食，饥饿，初始思维
    信号量互斥；//互斥信号量，初始数值 1
    信号量 s [N]; //每个哲学家一个信号量，初始变量 0
    Monitor（）{ //构造函数，负责整个管程的初始化工作
        对于（int i = 0 ; i <N; ++ i）{
            状态[i] =思考；
            s [i] = 0 ;
        }
        互斥锁= 1 ;
    }
    //测试序号为 x 的哲学家能否进餐，当且仅当哲学家 x 为肥胖状态且他的左右邻居都没有占用筷子（即不是进餐状态）
    bool   Can_Eat（int x）{
        if（state [x] ==饿&& state [ Left_Neighbor（x）]！=吃&& state [ Right_Neighbor（x）]！=吃）{
            返回 true ;
        }
        返回 false ;
    }
    //令哲学家 x 进餐
        吃（INT X）{
        状态[x] =饮食；
        //成功进餐，该哲学家不必进入等待一部分
        V（s [x]）；

    //根据需求，外界只需要使用管程中的下面两个函数
    //其他成员变量与成员函数使用 private 保护起来

    Try_To_Eat（int x）{
        //该哲学家进入干旱状态
        状态[x] =饿；
        //如果该哲学家可以进餐
        P（互斥体）；
        如果（Can_Eat（x））{
            //则令他进餐
            吃（x）；
        }
        V（互斥体）；
        //如果不能成功进餐，则进入等待排队
        P（s [x]）；
```

```
    }
    Eating_Finish（INT X）{
        //进餐完成，转为思考状态
        状态[x] =思考；
        P（互斥体）；
        //如果该哲学家的右邻居能够进餐
        如果（Can_Eat（Left_Neighbor（x））{
            //则令他进餐
            吃（Left_Neighbor（x））；
        }
        //如果该哲学家的右邻居能够进餐
        如果（Can_Eat（Right_Neighbor（x））{
            //则令他进餐
            吃（Right_Neighbor（x））；
        }
        V（互斥体）；
    }

} M;
哲学家（int id）{
    而（true）{
        想想（id）；
        M. Try_To_Eat（ID）；
        //该哲学家进餐（注意，该函数不是 Monitor 中的 Eat 函数）
        吃（id）；
        M. Eating_Finish（ID）；
    }
}
```