4．熟睡的理发师问题

在计算机科学中，“熟睡的理发师问题”是用来描述多个操作系统进程之间的，一个经典的进程之间的通信及同步问题。该问题模拟的描述是有顾客时，让理发师理发；没顾客时，让理发师睡觉。理发师与顾客就代表系统进程。

问题说明

假设有一个理发店只有一个理发师，一张理发时坐的椅子，若干张普通椅子顾客供等候时坐。没有顾客时，理发师就坐在理发的椅子上睡觉。顾客一到，他不是叫醒理发师，就是离开。如果理发师没有睡觉，而在为别人理发，他就会坐下来等候。如果所有的枯木都坐满了人，最后来的顾客就会离开。

在出现竞争的情况下问题就来了，这和其它的排队问题是一样的。实际上，与哲学家就餐问题是一样的。如果没有适当的解决方案，就会导致进程之间的“饿肚子”和“死锁”。如理发师在等一位顾客，顾客在等理发师，进而造成死锁。另外，有的顾客可能也不愿按顺序等候，会让一些在等待的顾客永远都不能理发。

解决方案

最常见的解决方案就是使用三个信号标（Semaphore）：一个给顾客信号标，一个理发师信号标（看他自己是不是闲着），第三个是互斥信号标（Mutual exclusion，缩写成mutex）。一位顾客来了，他想拿到互斥信号标，他就等着直到拿到为止。顾客拿到互斥信号标后，会去查看是否有空着的椅子（可能是等候的椅子，也可能是理发时坐的那张椅子）。

如果没有一张是空着的，他就走了。如果他找到了一张椅子，就会让空椅子的数量减少一张（）标。这位顾客接下来就使用自己的信号标叫醒理发师。这样，互斥信号标就释放出来供其他顾客或理发师使用。如果理发师在忙，这位顾客就会等。理发师就会进入了一个永久的等候循环，等着被在等候的顾客唤醒。一旦他醒过来，他会给所有在等候的顾客发信号，让他们依次理发。

这个问题中只有一个理发师，所以也叫“一个熟睡的理发师问题”。尽管多个理发师的情况会遇到相同的问题，不过解决起来问题要复杂得多。

模拟

下面的伪代码可以保证理发师与顾客之间不会发生死锁，但可能会让某位顾客“饿肚子”。P 和 V 是信号标的函数。

```
+顾客信号标 = 0
+理发师信号标=0
+互斥信号标=1
+int 空椅子数量=N      //所有的椅子数量

理发师（线程/进程）
While(true){           //持续不断地循环
  P（顾客）            //试图为一位顾客服务，如果没有他就睡觉
  P（互斥信号标）      //这时他被叫醒，要修改空椅子的数量
    空椅子数量++       //一张椅子空了出来
  V（理发师）          //理发师准备理发
  V（互斥信号标）      //我们不想再死锁在椅子上
                       //这时理发师在理发
}
```

```
顾客（线程/进程）
while(true) {          //持续不断地循环
  P（互斥信号标）      //想坐到一张椅子上
  if (空椅子数量>0) { //如果还有空着的椅子的话
  空椅子数量--         //顾客坐到一张椅子上了
  V（顾客）            //通知理发师，有一位顾客来了
  V（互斥信号标）      //不会死锁到椅子上
  P（理发师）          //该这位顾客理发了，如果他还在忙，那么他就等着
                       //这时顾客在理发
  }else {              //没有空着的椅子
                       //不好彩
  V（互斥信号标）      //不要忘记释放被锁定的椅子
                       //顾客没有理发就走了
  }
}
```

\

5. 三个烟鬼的问题

这也是计算机领域的并发问题，最早是 1971 年 S.S.Patil 讲述的。

问题描述

假设一支香烟需要：1、烟草；2、卷烟纸；3、一根火柴。

假设一张圆桌上围座着三烟鬼。他们每个人都能提供无穷多的材料：一个有无穷多的烟草；一个有无穷多的卷烟纸；一个有无穷多的火柴。

假设还有一个不吸烟的协调人。他每次都会公正地要求两个人取出一份材料放到桌，然后通知第三个人。第三个人从桌上拿走另外两个人的材料，再加上自己的一份，卷一枝烟就会抽起来。这时，协调人看到桌上空了，就会再次随机叫两人向桌上贡献自己的材料。这个过程会无限地进行下去。

不会有人把桌上的东西藏起来。只有当他抽完一枝烟后，才会再卷另一枝。如果协调人将烟草和卷烟纸放到桌上，而那个有火柴的人仍在吸烟，那么烟草、卷烟纸就会原封不动地放在桌上，走到有火柴的人抽完烟取走桌上的材料。

这个问题是想模拟一个软件程序中的四个角色，只使用了一部分同步前提条件。在 PATIL 的讲解中，只有一个同步前提条件是：信号标（semaphore），四个程序都不允许有条件地“跳转”，只能有一种由信号标操作提供的有条件的“等待”。

观点

PATIL 的观点是 Edsger Dijstra 的信号标方法作用有限。他用“三个烟鬼的问题”来证明这一点，即这种情况下信号标不能解决问题。但是，PATIL 为自己的辩解添加了两个限制条件：

1、 代码不能修改。

2、 解决方案不能使用有条件的语句或使用信号标数组。

如果加上这两个限制条件，三个烟鬼的问题没有办法解决了。Downey 曾其撰写的《信号标的小册子》里表示，第一个限制条件是有意义的。因为如果代码代表的是一个操作系统，每来一个新的应用程序都要修改它不但不合理，也是不可能的。但是，就像 David Parnas 指出的那样，第二个限制条件让重大的问题无法解决。

解决方案

如果我们取消第二个限制条件，使用二位信号标就可以解决“三个烟鬼的问题”。我们可以定义一个二位信号标数组 A，每个烟鬼一个，桌子也有一个对应的二进制信号标 T。将烟鬼的信号标初始化为 0，桌子的初始化为 1。于是，协调人的代码为：

```
While true {
   Wait (T);
   公正地随机地选择烟鬼i和j，第三个为k；
   Signal(A[k]);
}
```

烟鬼的代码为：

```
While true {
   Wait (A[i]);    卷一枝烟
   Signal (T);     抽一枝烟
}
```