题目：面包师有很多面包，由n个销售人员推销，每个顾客进店后取一个号，并且等待叫号。当一个销售人员空闲下来时，就叫下一个号。设计一个销售人员与顾客同步的**[算法](http://lib.csdn.net/base/datastructure" \o "算法与数据结构知识库" \t "http://blog.csdn.net/u011240016/article/details/_blank)**。

这一类题型代表性很强，比如银行柜台，各种商店的服务等，都是一样的模式。

分析：这里有n个销售人员可用，像是n个缓冲区一样。每个顾客取号，等待的看似是号码，实际是销售。因此，这是简化的生产者消费者问题，这里没有互斥问题，仅仅是同步。   
以上是一种典型的错误分析的思路。

因为，即使只有一个销售，可以叫的号码也可以有无数个，虽然会让这一个销售累死。。。

因此，这个过程中，号码是非常关键非常关键的东西。我们设计同步全是围绕着数字而来。

而上面那种把数字等同于销售的做法，简直错透了！

分析到这里，我们明白不同的销售之间其实是在抢数字的，虽然他们本身并不是这么勤奋，看到号码就想抢，给顾客推销。但是实际运作的机制是这样：对每一个号码，互斥访问。

同理，在顾客这边，也不能让两个顾客共用一个号，因而对号码也是互斥访问的。   
号码在顾客这方是顺序增长的。

想到在银行排队办理业务，我突然明白以上的分析其实很多余。。。

那么，我们只需要设计两个信号量用于顾客这里产生的叫号的互斥访问和销售端号码的处理的互斥访问。

int i = 0, j = 0; // i是当前取号值，j是当前叫号值

semaphore mutex\_i = 1, mutex\_j = 1;

Consumer()//号码递增

{

P(mutex\_i);//抢到叫号机

取号；

i++;

V(mutex\_i);

等待叫号；

}

Seller() //从小号开始叫

{

P(mutex\_j);

if(j < i) //j这个号对应是真的顾客

{

叫号j; //顾客过来被服务

j++;//号码增长

V(mutex\_j);

销售面包；

}

else

{

V(mutex\_j);

暂时没号，爱干嘛干嘛；

}

}