3.5 临界区问题的软件解法：面包房算法

前面费尽周折，直至 Peterson 算法，其实还只是解决了两个进程之间的互斥问题。那么，当有多个进程时，互斥问题该怎么解决呢?

好在 Lamport 提出了著名的面包房算法（bakery algorithm）。这个算法的名字的由来是因为它源自顾客在面包店中购买面包时的排队原理。顾客在进入面包店前，首先抓一个号，然后按照号码由小到大的次序依次进入面包店购买面包。这里，面包店发放的号码是由小到大的，但是两个或两个以上的顾客却有可能得到相同的号码（要使得所有顾客抓到不同的号码本身就需要解决互斥问题），如果多个顾客抓到相同的号码，则规定按照顾客名字的字典次序进行排序。这里假定顾客是没有重名的。在计算机系统中，顾客就相当于进程，每个进程有一个唯一的标识，我们用 P 加一个下标来表示，如 Pi 和 Pj。

该算法的实现需要引入两个数据结构：

```
int choosing[n];
int number[n];
```

前者表示进程是否正在抓号。正在抓号的进程 choosing[i]为 TRUE，否则为 FALSE；其初值均设为 FALSE。后者用来记录各个进程所抓到的号码。如果 number[i]为 0，则进程 Pi 还没有抓号；如 number[i]不为 0，则其正整数值为进程 Pi 所抓到的号码；其初值均设为 0。

为了方便起见, 我们定义下述表示法：

- $(a,b)<(c,d)$ 如果 $a<c$ or $(a=c$ and $b<d)$.
- $\max(a_0,\ldots,a_{n-1})$ 其值为一正整数 k， 对于所有 i， $0\le i\le n-1$， $k\ge a_i$. 或者说，k 就是 a_0 到 a_{n-1} 所有数的一个上界

我们先给出如图 3.6 所示的面包房算法的典型结构。

```
do {
  choosing[i] = true;
  number[i] = max{number[0],number[1],…,number[n-1]}+1;//选号码
  choosing[i] = false;
  for(j = 0; j<n; j++) {
    while (choosing[j]);
    while ((number[j] != 0) && (number[j],j)<(number[i],i));
  };
  // 临界区
  number[i] = 0;
  // 其余部分
} while(1);
```

图 3.6 进程 Pi 的算法结构

注意 for 语句内嵌套的第 2 个 while 语句。对于 Pi 和 Pj，如果有 i<j，则算法的结果将先为 Pi 服

务，即 Pi 先进入临界区。

可以证明，该算法满足临界区问题的解需要满足的三个条件（留作课后练习）。