现在读者可能会想，“如果哲学家在拿不到右边叉子时等待一段随机时间，而不是等待相同的时间，这样发生互锁的可能性就很小了，事情就可以继续了。”这种想法是对的，而且在几乎所有的应用程序中，稍后再试的办法并不会演化成为一个问题。例如，在流行的局域网以太网中，如果两台计算机同时发送包，那么每台计算机等待一段随机时间之后再尝试。在实践中，该方案工作良好。但是，在少数的应用中，人们希望有一种能够始终工作的方案，它不能因为一串不可靠的随机数字而导致失败（想象一下核电站中的安全控制系统）。

对图2-45中的算法可做如下改进，它既不会发生死锁又不会产生饥饿：使用一个二元信号量对调用think之后的五个语句进行保护。在开始拿叉子之前，哲学家先对互斥量 mutex执行down操作。在放回叉子后，他再对mutex执行up操作。从理论上讲，这种解法是可行的。但从实际角度来看，这里有性能上的局限：在任何一时刻只能有一位哲学家进餐。而五把叉子实际上可以允许两位哲学家同时进餐。

图2-46中的解法不仅没有死锁，而且对于任意位哲学家的情况都能获得最大的并行度。算法中使用一个数组state跟踪每一个哲学家是在进餐、思考还是饥饿状态（正在试图拿叉子）。一个哲学家只有在两个邻居都没有进餐时才允许进入到进餐状态。第 i 个哲学家的邻居则由宏LEFT和RIGHT定义，换言之，若 i 为2，则LEFT为1，RIGHT为3。

```
#define N            5              /* 哲学家数目 */
#define LEFT         (i+N-1)%N      /* i 的左邻居编号 */
#define RIGHT        (i+1)%N        /* i 的右邻居编号 */
#define THINKING     0              /* 哲学家在思考 */
#define HUNGRY       1              /* 哲学家试图拿起叉子 */
#define EATING       2              /* 哲学家进餐 */
typedef int semaphore;              /* 信号量是一种特殊的整型数据 */
int state[N];                       /* 数组用来跟踪记录每位哲学家的状态 */
semaphore mutex = 1;                /* 临界区的互斥 */
semaphore s[N];                     /* 每个哲学家一个信号量 */

void philosopher(int i)             /* i：哲学家编号，从0到N-1 */
{
    while (TRUE) {                  /* 无限循环 */
    }
}

void take_forks(int i)              /* i：哲学家编号，从0到N-1 */
{
    down(&mutex);                   /* 进入临界区 */
    state[i] = HUNGRY;              /* 记录哲学家i处于饥饿的状态 */
    test(i);                        /* 尝试获取2把叉子 */
    up(&mutex);                     /* 离开临界区 */
    down(&s[i]);                    /* 如果得不到需要的叉子则阻塞 */
}

void put_forks(i)                   /* i：哲学家编号，从0到N-1 */
{
    down(&mutex);                   /* 进入临界区 */
    state[i] = THINKING;            /* 哲学家已经就餐完毕 */
    test(LEFT);                     /* 检查左边的邻居现在可以吃吗 */
    test(RIGHT);                    /* 检查右边的邻居现在可以吃吗 */
    up(&mutex);                     /* 离开临界区 */
}

void test(i)                        /* i：哲学家编号，从0到N-1 */
{
    if (state[i] == HUNGRY && state[LEFT] != EATING && state[RIGHT] != EATING) {
        state[i] = EATING;
        up(&s[i]);
    }
}
```

图2-46 哲学家就餐问题的一个解法

该程序使用了一个信号量数组，每个信号量对应一位哲学家，这样在所需的叉子被占用时，想进餐的哲学家就被阻塞。注意，每个进程将过程philosopher作为主代码运行，而其他过程take_forks、