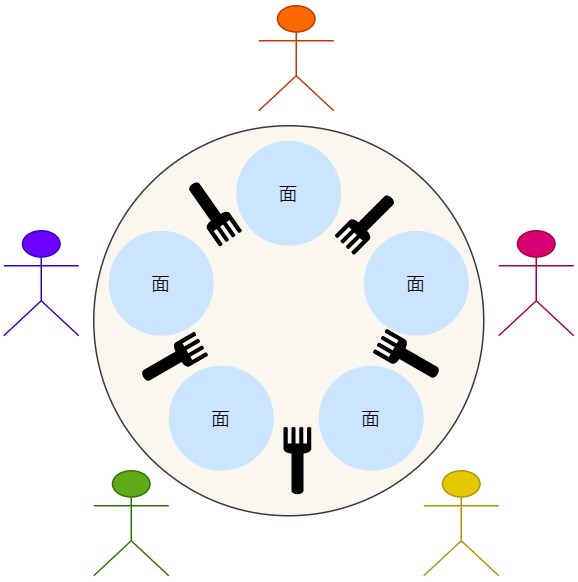
经典同步问题--哲学家就餐问题

哲学家就餐问题



问题描述：

- 五个哲学家围着一张圆桌吃面
- 只有五个叉子，每两个哲学家之间放一个叉子
- 哲学家围在一起思考，中途饿了就会想进餐
- 这些哲学家需要左右两个叉子才会吃面
- 吃完后，放回叉子，继续思考

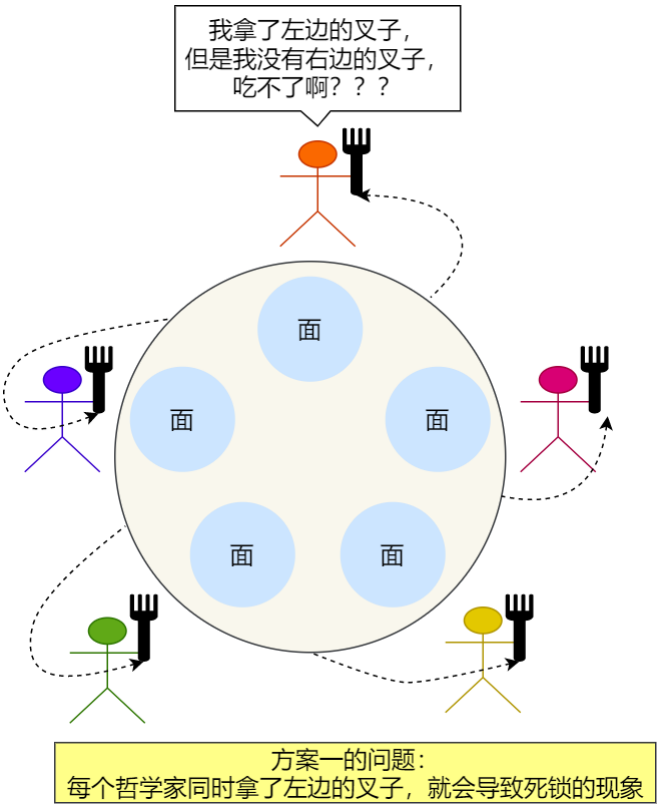
那么如何保证哲学家有序进餐和思考，而不会永远有人拿不到叉子

方案1

用信号量PV 操作来尝试解决它，代码如下

```
#define N 5                         // 哲学家个数
semaphore fork[5];                  // 信号量初值为 1
                                    //也就是叉子的个数

void smart_person(int i)            // i 为哲学家编号 0-4
{

    while(TRUE)
    {
        think();                    // 哲学家思考
        P(fork[i]);                 // 去拿左边的叉子
        P(fork[(i + 1) % N ]); // 去拿右边的叉子
        eat();                      // 进餐
        V(fork[i]);                 // 放下左边的叉子
        V(fork[(i + 1) % N ]); // 放下右边的叉子
    }
}
```

代码看起来没问题，拿起叉子用P操作，代表有叉子就直接用，没有叉子时就等待其他哲学家放回叉子。 不过，这种解法存在一个极端的问题： **假设五位哲学家同时拿起左边的叉子，桌面上就没有叉子了，这样就没有人能够拿到他们右边的叉子，也就说每一位哲学家都会在 P(fork[(i + 1) % N]) 这条语句阻塞了，很明显这发生了死锁的现象。**

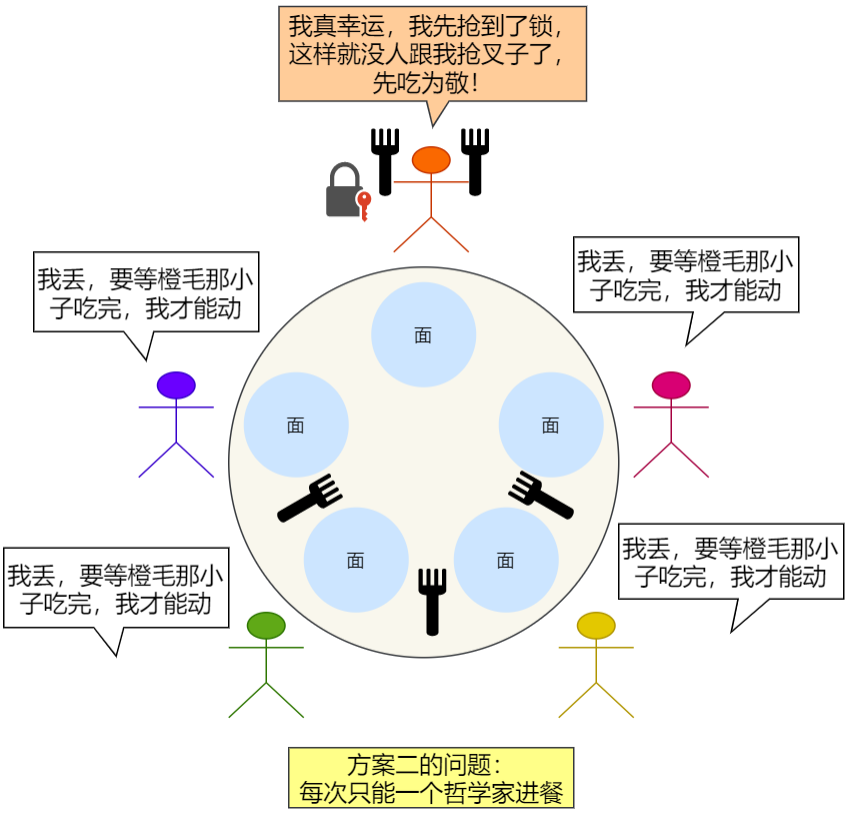


方案2

我们在方案1中，拿叉子前，加个互斥信号量

```
#define N 5                     // 哲学家个数
semaphore fork[5];              // 每个叉子一个信号量，初值为 1
semaphore  mutex;               // 互斥信号量，初值为 1

void smart_person(int i)        // i 为哲学家编号 0-4
{

    while(TRUE)
    {
        think();                // 哲学家思考
        P(mutex);               // 进入临界区
        P(fork[i]);             // 去拿左边的叉子
        P(fork[(i + 1) % N ]); // 去拿右边的叉子
        eat();                  // 进餐
        V(fork[i]);             // 放下左边的叉子
        V(fork[(i + 1) % N ]); // 放下右边的叉子
        V(mutex);               // 退出临界区
    }
}
```

程序中的互斥信号量的作用就在于， 只要有一个哲学家进入了临界区，也就是准备要拿叉子时，其他哲学家都不能动，只有这位哲学家用完叉子了，才能轮到下一个哲学家进餐。
这种方案虽然能让哲学家们按顺序吃饭，但是每次进餐只能有一位哲学家，但是桌面上是有5把叉子，按道理是能可以有两个哲学家同时进餐的，所以从效率角度上，这不是最好的解决方案。



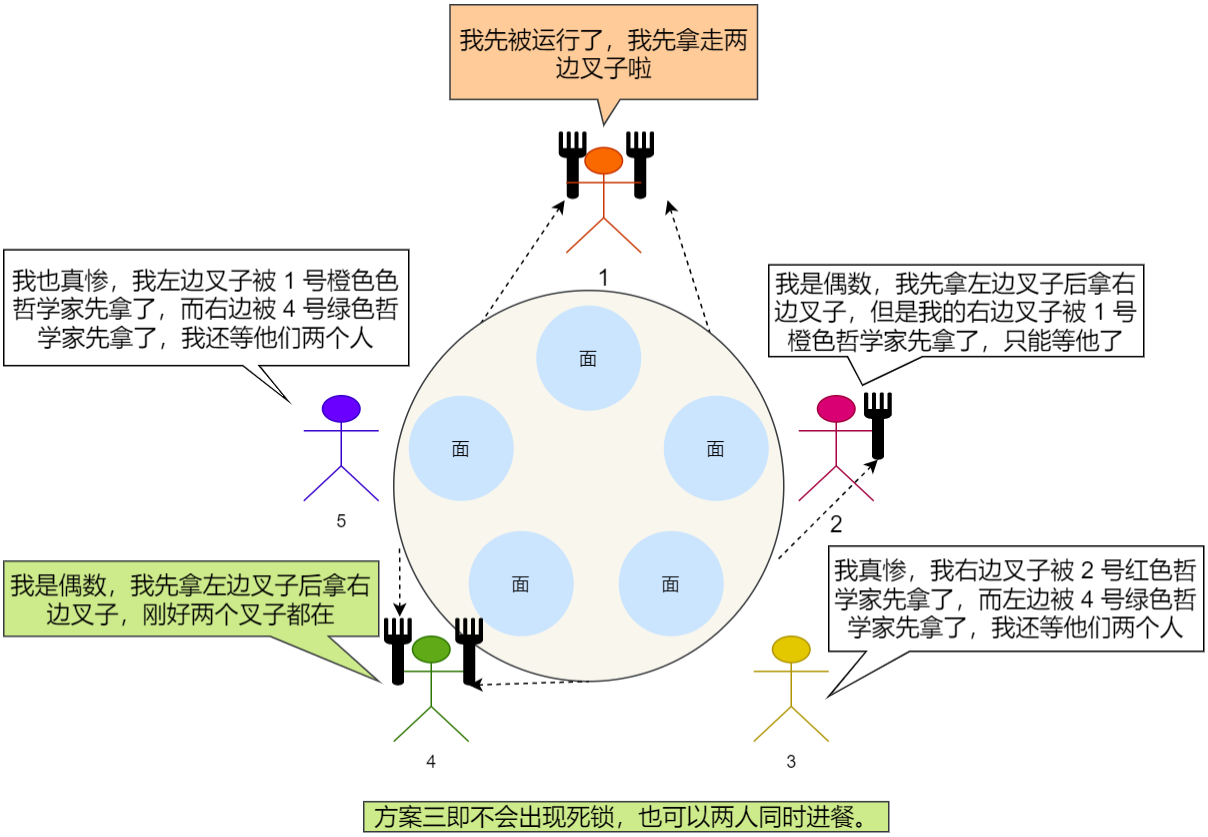
方案3

方案2使用互斥信号量，会导致只能允许一个哲学家就餐，那么我们就不用它。
另外，方案1的问题在于，会出现所有哲学家同时拿左边刀叉的可能性，**那我们就避免哲学家可以同时拿左边的刀叉，采用分支结构，根据哲学家的编号的不同，而采取不同的动作。即让偶数编号的哲学家先拿左边的叉子后拿右边的叉子，奇数编号的哲学家先拿右边的叉子后拿左边的叉子。**

```
#define N 5                     // 哲学家个数
semaphore fork[5];              // 每个叉子一个信号量，初值为 1

void smart_person(int i)        // i 为哲学家编号 0-4
{

    while(TRUE)
    {
        think();                    // 哲学家思考

        if ( i % 2 == 0 )
        {
            P(fork[i]);             // 去拿左边的叉子
            P(fork[(i + 1) % N ]); // 去拿右边的叉子
        }
        else
        {
            P(fork[(i + 1) % N ]); // 去拿右边的叉子
            P(fork[i]);             // 去拿左边的叉子
        }

        eat();                      // 哲学家进餐

        V(fork[i]);                 // 放下左边的叉子
        V(fork[(i + 1) % N ]);      // 放下右边的叉子
    }
}
```

程序在P操作时，根据哲学家的编号不同，拿起左右两边叉子的顺序不同。另外， V操作是不需要分支

的，因为V操作是不会阻塞的。



方案3即不会出现死锁，也可以两人同时进餐。

方案4

再提出另外一种可行的解决方案，我们用一个数组state来记录每一位哲学家在进程、思考还是饥饿状态（正在试图拿叉子）。

那么，一个哲学家只有在两个邻居都没有进餐时，才可以进入进餐状态。第i个哲学家的左邻右舍，则由宏LEFT和RIGHT定义：

LEFT：(i + 5 – 1) % 5

RIGHT：(i + 1) % 5

比如i为2，则LEFT为1，RIGHT为3。具体代码实现如下

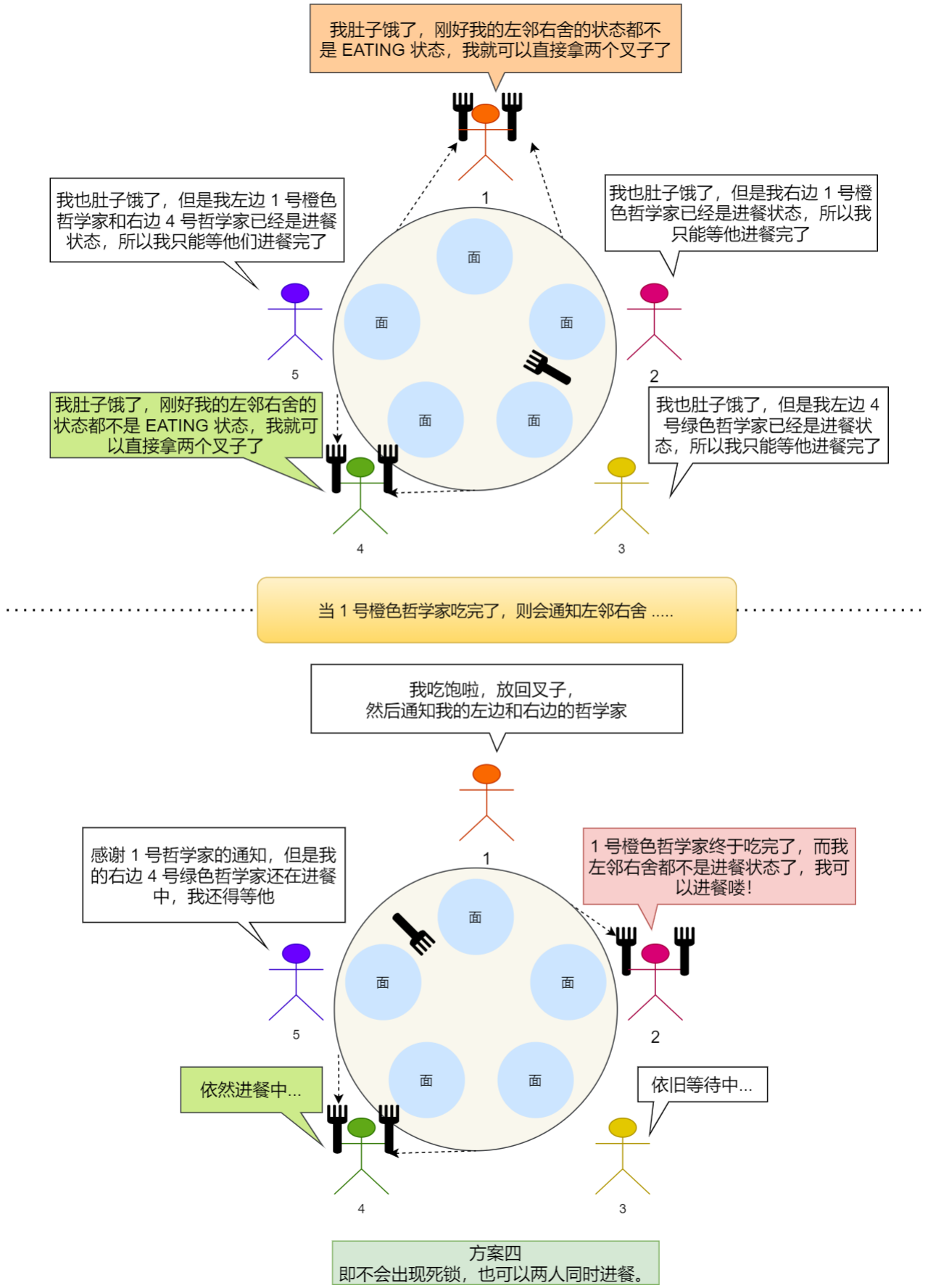
```c
#define N 5                     // 哲学家个数
#define LEFT (i + N  - 1) % N // i 的左边邻居编号
#define RIGHT (i + 1) % N     // i 的右边邻居编号

#define THINKING 0              // 思考状态
#define HUNGRY   1              // 饥饿状态
#define EATING   2              // 进餐状态

int state[N];                   // 数组记录每个哲学家的状态

semaphore s[5];                 // 每个哲学家一个信号量，初值 0
semaphore mutex;                // 互斥信号量，初值为 1


void test(int i)  // i 为哲学家编号 0-4
{
    // 如果 i 号的左边右边哲学家都不是进餐状态，把 i 号哲学家标记为进餐状态
    if ( state[i] == HUNGRY &&
         state[LEFT] != EATING &&
         state[RIGHT] != EATING )
    {
        state[i] = EATING // 两把叉子到手，进餐状态
        V(s[i]);          // 通知第 i 哲学家可以进餐了
    }
}

// 功能：要么拿到两把叉子，要么被阻塞起来
void take_forks(int i)  // i 为哲学家编号 0-4
{
    P(mutex);             // 进入临界区
    state[i] = HUNGRY;  // 标记哲学家处于饥饿状态
    test(i);            // 尝试获取 2 支叉子
    V(mutex);           // 离开临界区
    P(s[i]);            // 没有叉子则阻塞，有叉子则继续正常执行
}

// 功能：把两把叉子放回原处，并在需要的时候，去唤醒左邻右舍
void put_forks(int i)  // i 为哲学家编号 0-4
{
    P(mutex);             // 进入临界区
    state[i] = THINKING; // 吃完饭了，交出叉子，标记思考状态
    test(LEFT);           // 检查左边的左邻右舍是否在进餐，没则唤醒
    test(RIGHT);          // 检查右边的左邻右舍是否在进餐，没则唤醒
    V(mutex);             // 离开临界区
}

// 哲学家主代码
void smart_person(int i) // i 为哲学家编号 0-4
```

```
{

    while(TRUE)
    {
        think();        // 思考
        take_forks(i); // 准备拿去叉子吃饭
        eat();          // 就餐
        put_forks(i);  // 吃完放回叉子
    }
}
```

上面的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/线程将 smart_person 函数作为主代码运行，而其他 take_forks 、 put_forks 和test

只是普通的函数，而非单独的进程/线程。



方案4同样不会出现死锁，也可以两人同时进餐。

文字摘抄：小林《图解系统》