哲学家就餐问题效率分析及编程模拟实现

Efficiency Analysis and Programming Simulation Implementation of Philosopher's Dining Problem

文俊超　徐艳 *　刘康媛　何鑫

Junchao Wen　Yan Xu*　Kangyuan Liu　Xin He

四川大学锦城学院计算机与软件学院　中国·四川 成都 610000

School of Computer and Software, Jincheng College, Sichuan University, Chengdu, Sichuan, 610000, China

摘　要：论文提出了经典进程同步问题——哲学家就餐问题，并分析解决这类问题所需要实现的算法过程，同时通过代码对此进程同步问题进行了编程模拟实现。使读者更好地理解多道程序执行中的同步机制。

Abstract: This paper puts forward the classical process synchronization problem philosopher dining problem, analyzes the algorithm process needed to solve this kind of problem, and simulates the process synchronization problem through code. Make readers better understand the synchronization mechanism in multiprogramming execution.

关键词：进程同步；信号量；AND 型信号量机制；哲学家就餐问题

Keywords: process synchronization; semaphore; AND-type semaphore mechanism; philosopher's dining problem

DOI: 10.12346/sde.v4i3.6023

1 问题描述及分析

五个哲学家坐在一张圆桌就餐，桌上有五个碗五支筷子，每支筷子放在两位哲学家中间。哲学家交替进行思考就餐，平时在思考，饥饿时便会尝试拿起靠近他左右两边的筷子，只有能拿起两只筷子时才能进餐，进餐完后放下筷子继续思考。

分析：筷子是进程的临界资源，同一时间内只允许一位哲学家使用，若两边筷子正在被他人使用，则此哲学家会进入饥饿状态等待进餐。此问题会涉及先拿左边或先拿右边两种方法，对此进行分析及实现。

2 算法及编程实现

为了实现对筷子的互斥作用，可考虑用一个信号量代表一只筷子，这五个信号量则可以形成一个信号量数组：

```
#define N 5  // 哲学家的个数为 5 个，N=5
sem_t chopstick[5]={1,1,1,1,1};  // 每支筷子信号量初值为 1
void philosopher(int i);  // 五位哲学家的编号 i(0-4)
```

2.1 算法 1

哲学家在思考，饥饿时先拿左边再拿右边，吃完后先放下左边再放下右边。代码如下：

```
void philosopher(int i)
{
    while(1)
{
    printf("philosopher%d thinking\n", i);
    P(chopstick[i]);  // 拿左边筷子
    sleep(3);
    P(chopstick[(i+1)%N]);  // 拿右边筷子
    printf("philosopher%d eating\n", i);
    V(chopstick[i]);  // 放下左边筷子
    V(chopstick[(i+1)%N]);  // 放下右边筷子
    sleep(3);
    }
}
```

测试运行结果截图见图 1。

【作者简介】文俊超（2002-），男，中国四川成都人，本科，从事计算机科学与技术研究。
【通讯作者】徐艳（1979-），女，中国四川宜宾人，硕士，教授，从事算法开发及网络计算和应用研究。

```
philosopher0 thinking
philosopher2 thinking
The philosopher2 gets left 2 chopstick
philosopher4 thinking
The philosopher4 gets left 4 chopstick
philosopher3 thinking
The philosopher3 gets left 3 chopstick
The philosopher0 gets left 0 chopstick
philosopher1 thinking
The philosopher1 gets left 1 chopstick
```

图 **1**

这种算法保证五位哲学家中不会有两位相邻的哲学家同时进餐，可是会面临“死锁”。假设五位哲学家同时进入饥饿状态，同时拿起左边的筷子时，那么 chopstick[5] 的值均减为 0，这样当再去拿右边筷子时，由于无信号量提供，程序只有进入死循环等待，因此形成“死锁”。这个问题又可以采取另外算法来优化解决。

2.2 算法 2

设置一个互斥信号量 mutex，保证哲学家 i 申请到互斥信号量进行就餐时，只有该哲学家进入临界区，不会受到其他哲学家的影响，吃完后退出临界区释放互斥信号量，因此不会形成“死锁”。代码如下：

```
    sem_t mutex=1; // 设置互斥信号量，初值为 1
    void philosopher(int i)
{
            while(1)
{
            printf("philosopher%d thinking\n", i);
            P(mutex); // 申请互斥信号量，进入临界区
            P(chopstick[i]); // 拿左边筷子
            sleep(3);
            P(chopstick[(i+1)%N]); // 拿右边筷子
            printf("philosopher%d eating\n", i);
            V(chopstick[i]); // 放下左边筷子
            V(chopstick[(i+1)%N]); // 放下右边筷子
            V(mutex); // 退出临界区，释放互斥信号量
            sleep(3);
    }
}
```

互斥信号量保证五位哲学家不会因为同时拿起筷子而无信号量提供引起的“死锁”，但是会影响进程效率，五支筷子可以使两位哲学家同时就餐，可此算法对其进行了限制，每次当一位哲学家就餐时另外四位哲学家都不能就餐，降低了进程效率。

2.3 算法 3

算法 2 仅设置了一个互斥信号量，导致每次只有一个进程运行，大大降低了效率。对此可以设置一个控制信号量 count（计数器）来控制进程并发数目，我们可以允许至多四位哲学家同时拿左边筷子，这样能保证至少一位哲学家可以进餐，从而不会形成“死锁”，也可以大大提高进程效率。代码如下：

```
sem_t count=4; // 设置计数器信号量 count，初值为 4
    void philosopher(int i)
{
            while(1)
{
            printf("philosopher%d thinking\n", i);
            P(count); // 进入临界区，计数器 count-1
            P(chopstick[i]); // 拿左边筷子
            sleep(3);
            P(chopstick[(i+1)%N]); // 拿右边筷子
            printf("philosopher%d eating\n", i);
            V(chopstick[i]); // 放下左边筷子
            V(chopstick[(i+1)%N]); // 放下右边筷子
            V(count); // 退出临界区，计数器 count+1
            sleep(3);
            }
}
```

上述算法有效地解决了以上的“死锁”问题以及降低效率问题。

2.4 算法 4

此算法会基于哲学家的编号来规定拿起左右筷子的先后性，我们可以规定奇数哲学家先拿左边筷子偶数哲学家先拿右边筷子，即 1、2 号哲学家竞争 1 号筷子，3、4 号玩家竞争 3 号筷子，竞争成功的哲学家再去拿另一边筷子，这样总会有哲学家能成功进餐，不会造成“死锁”。代码如下：

```
void philosopher(int i)
{
    while(1)
    {
    printf("philosopher%d thinking\n", i);
    switch(i%2)
    {
    case 1:
            P(chopstick[i]); // 拿左边筷子
            sleep(3);
            P(chopstick[(i+1)%N]); // 拿右边筷子
                break;
                case 0:
            P(chopstick[(i+1)%N]); // 拿右边筷子
                sleep(3);
            P(chopstick[i]); // 拿左边筷子
            break;
```

```
            }
            printf("philosopher%d eating\n", i);
            V(chopstick[i]); // 放下左边筷子
            V(chopstick[(i+1)%N]); // 放下右边筷子
            sleep(3);
        }
    }
```

上述算法通过编号奇偶性来规定拿左右筷子先后顺序，这样所需资源被占用时，就会进入等待队列，不会造成“死锁”，也能使进程有着最大的并行度。

2.5 算法 5

利用 AND 信号量机制来解决哲学家就餐问题。将哲学家两边的筷子合并起来，如果两边筷子都未被占用，则允许进入临界区进行就餐，如果两边的筷子任意一支被占用的话，则该哲学家不会拿起两边筷子。代码如下：

首先要设两个实现同时拿筷子和同时放筷子的函数：

```
void SP(int i)
{
    P(&chopstick[i]);
    P(&chopstick[(i+1)%N]);
    return;
}
void SV(int i)
{
    V(&chopstick[i]);
    V(&chopstick[(i+1)%N]);
    return;
}
```

代入函数实现：

```
    void philosopher(int i)
{
            while(true)
{
            printf("philosopher%d thinking\n", i);
            SP(i); // 同时拿起两只筷子
            sleep(3);
            printf("philosopher%d eating\n", i);
            SV(i); // 放下左右两支筷子
            sleep(3);
            }
}
```

测试运行结果截图见图 2。

```
philosopher0 thinking
philosopher3 thinking
The philosopher3 gets left 3 and right 4 chopstick
philosopher2 thinking
The philosopher0 gets left 0 and right 1 chopstick
philosopher4 thinking
philosopher1 thinking
philosopher3 eating
The philosopher3 puts down left 3 and right 4 chopstick
The philosopher2 gets left 2 and right 3 chopstick
philosopher0 eating
The philosopher0 puts down left 0 and right 1 chopstick
The philosopher4 gets left 4 and right 5 chopstick
philosopher2 eating
The philosopher2 puts down left 2 and right 3 chopstick
philosopher3 thinking
The philosopher1 gets left 1 and right 2 chopstick
philosopher0 thinking
philosopher4 eating
The philosopher4 puts down left 4 and right 5 chopstick
```

图 2

哲学家就餐这种一个进程需要获得两个共享资源后才能执行的问题，AND 信号量机制是一个非常简洁的算法。其基本思想就是将一个进程在整个运行过程中需要的所有临界资源一次性全部分配给进程使用，用完后一起释放，若有一个所需资源已被占用，则其他所需资源不会分配给该进程。要么全部分配要么都不分配，这样简单解决了“死锁”，也不会影响效率。

3 算法效率对比

算法 2~5 都是基于算法 1 的不完善实现进行优化从而解决“死锁”以及效率低。算法 2 会降低整个系统的效率。其中算法 5 运用 AND 型信号量机制实现了哲学家就餐。解决这类问题就是通过设置信号量以及 P、V 操作来对其进程合理分配资源，实现同步互斥使整个系统有条不紊地运行起来。

参考文献

[1] 汤小丹,梁红兵,哲凤屏.计算机操作系统（第四版）[M].西安:西安电子科技大学出版社,2014.

[2] 王文磊,徐汀荣.经典进程同步问题的研究与实现[J].计算机工程与设计,2006(27):2248-2253.

[3] 朱敬鹏,乔丽,王铁柱.使用P、V操作解决经典进程同步问题[J].商丘师范学院学报,2005(21):95-98.