文章编号：1672-5913(2009)14-0086-03

计算机操作系统哲学家进餐问题的教学探讨

窦金凤[1]，曹家宝[2]，郭忠文[1]，刘颖健[1]

(1. 中国海洋大学，山东 青岛 266100；2. 青岛阿尔卡特朗讯公司，山东 青岛 266101)

摘　要：本文根据多年的教学经验，利用信号量机制、管程机制等思想对哲学家进餐问题进行研究，提出了解决思路，并在教学实验过程中进行了验证。希望与其他相关领域的学习者共享，方便“操作系统”的教学、学习和应用。

关键词：进程同步；哲学家进餐问题；信号量；死锁；管程

中图分类号：G642　　**文献标识码**：B

1　引言

由荷兰学者 Dijkstra 提出的哲学家进餐问题(The Dinning Philosophers Problem)是经典的同步问题之一。哲学家进餐问题是一大类并发控制问题的典型例子，涉及信号量机制、管程机制以及死锁等操作系统中关键问题的应用，在操作系统文化史上具有非常重要的地位。对该问题的剖析有助于学生深刻地理解计算机系统中的资源共享、进程同步机制、死锁等问题，并能熟练地将该问题的解决思想应用于生活中的控制流程。

2　问题描述

有五个哲学家，共用一张圆桌，分别坐在周围的五把椅子上，圆桌上有五个碗和五只筷子，每人两边各放一只筷子，如图 1 所示。哲学家们是交替思考和进餐，饥饿时便试图取其左右最靠近他的筷子。

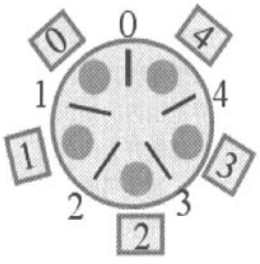

图 1　哲学家进餐描述图

哲学家进餐问题可看作是并发进程并发执行时，处理共享资源的一个有代表性的问题。分析其约束条件：

(1) 只有拿到两只筷子时，哲学家才能吃饭。

(2) 如果筷子已被别人拿走，则必须等别人吃完之后才能拿到筷子。

(3) 任一哲学家在自己未拿到两只筷子吃饭前，不会放下手中拿到的筷子。

3　用信号量机制解决问题

3.1　记录型信号量

筷子是临界资源，一段时间只允许一位哲学家使用。为了表示互斥，用一个信号量表示一只筷子，五个信号量构成信号量数组。由于计算机专业本科生先行课开设 C 语言，所以本文中算法用类 C 语言描述伪码算法。算法描述如下：

```
Semaphore chopstick[5]={1,1,1,1,1};  /*分别表示 5 支筷子*/
 Main()
   {
   cobegin
      philosopher(0);
      philosopher(1);
      philosopher(2);
      philosopher(3);
      philosopher(4);
   coend
   }
```

第 I 位哲学家的活动描述为：

```
philosopher (int I)
   {
    while (true)
    {
```

本研究得到计算机操作系统课程教学改革研究项目(TJY0805)的支持。

作者简介：窦金凤(1976-)，女，山东沂源人，讲师，博士，从事计算机操作系统、传感器网络、海洋学、智能控制方面的研究。

```
        思考;
        wait (chopstick [ I ]);
        wait (chopstick [(I+1)%5]);
        进餐;
        signal (chopstick [I]);
        signal (chopstick [(I+1)%5]);
        }
    }
```

当哲学家饥饿时，总是先去拿他左边的筷子，执行 wait (chopstick [I])，成功后，再去拿他右边的筷子，执行 wait (chopstick [I+1] %5)；成功后便可进餐。进餐毕，先放下他左边的筷子，然后再放下右边的筷子。

当五个哲学家同时去取他左边的筷子，每人拿到一只筷子且不释放，即五个哲学家只得无限等待下去，引起死锁。

3.2 死锁问题的解决

预防死锁即是破坏死锁的必要条件之一。通常用下列方法解决死锁问题。

3.2.1 破坏请求保持条件

利用原子思想完成。即只有拿起两支筷子的哲学家才可以进餐，否则，一支筷子也不拿。

解法一：利用 AND 机制实现第 I 位哲学家的活动描述为：

```
philosopher (int I)
    {
      while (true)
       {
        思考;
swait(chopstick[(I+1)] % 5, chopstick[I]);
        进餐;
     ssignal(chopstick[I], chopstick[(I+1)% 5]);
        }
    }
```

解法二：利用记录型信号量机制实现在初始化中增加一个信号量定义：semaphore mutex = 1;

第 I 位哲学家的活动描述：

```
philosopher (int I)
    {
      while (true)
       {
        思考;
        wait(mutex);
        wait (stick [ I ]);
        wait (stick [(I+1)%5]);
        signal (mutex);
        进餐;
        signal (stick [I]);
        signal (stick [(I+1)%5]);
          }
    }
```

该方法将拿两只筷子的过程作为临界资源，一次只允许一个哲学家进入。

3.2.2 破坏环路等待条件

在上述死锁问题中，哲学家对筷子资源的申请构成了有向环路，如图 2 所示。

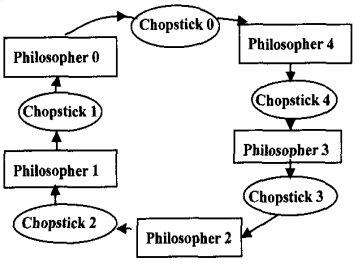

图 2 环路等待

解法一：奇数号哲学家先拿他左边的筷子，偶数号哲学家先拿他右边的筷子。这样破坏了同方向环路，一个哲学家拿到一只筷子后，就阻止了他邻座的一个哲学家吃饭。按此规定，将是 1、2 号哲学家竞争 1 号筷子；3、4 号哲学家竞争 4 号筷子。两种算法描述如下：

(1) 第 I 个哲学家的活动：

```
philosopher (int I)
    {
      while (true)
       {
        思考;
    If  I % 2 == 1 then
        wait (stick [ I ]);
        wait (stick [(I+1)%5]);
        进餐;
        signal (stick [I]);
        signal (stick [(I+1)%5]);
else
        wait (stick [(I+1)%5]);
        wait (stick [ I ]);
        进餐;
        signal (stick [(I+1)%5]);
        signal (stick [I]);
          }
        }
```

(2) 第 I 个哲学家的活动：

```
philosopher (int I)
    {
      while (true)
       {
        思考;
      wait (chopstick[I +(I % 2)];
      wait (chopstick[(I+ (I+1) % 2) % 5] )
            进餐;
      signal (chopstick[I +(I % 2)]);
      signal (chopstick[(I+ (I+1) % 2) % 5] );
            }
       }
```

解法二：至多允许四位哲学家进餐，将最后一个哲学家停止申请资源，断开环路。最终能保证有一位哲学家能进餐，用完释放两只筷子，从而使更多的哲学家能够进餐。增加一个信号量定义 semaphore count = 4;算法描述第 I 个哲学家的活动：

```
philosopher (int I)
    {
      while (true)
       {
        思考;
    wait (count);
    wait(chopstick[I ]);
    wait(chopstick[I+1]mod 5);
          进餐;
    signal(chopstick[I]);
    signal(chopstick[I+1]mod 5)
    signal(count)
       }
    }
```

解法三：哲学家申请资源总是按照资源序号先大后小的顺序，这样 0-3 号哲学家先右后左，但是 4 号哲学家先左后右，改变方向，破坏了环路。算法描述第 I 个哲学家的活动：

```
philosopher (int I)
{
    while (true)
      {
      思考;
    if I >(I+1) % 5 then
      wait (chopstick [I ]);
      wait (chopstick [I+1]mod 5);
    else
      wait (chopstick [I+1]mod 5);
      wait (chopstick [I ]);  /*哲学家总是先取最
                                大序号的筷子*/
      进餐;
      signal (chopstick [I ]);
      signal (chopstick [I+1]mod5);
      }
    }
```

3.2.3 破坏非剥夺条件

打破约束条件(3)，设立优先权。比如，根据哲学家等待筷子的时间设定，时间越大优先权越高，优先权高的可以抢夺优先权低的筷子。

4 用管程机制解决哲学家进餐问题

在用信号量机制解决同步问题时，往往比较繁琐，采用面向对象的思想，将资源及资源的共享操作封装起来，用管程来管理，实现哲学家进餐问题，使用起来更加方便。

算法实现描述如下：

4.1 建立管程

```
monitor PC          /*建立管程*/
  {
    semaphore chopstick [5] = {1,1,1,1,1};
    X: condition; /*定义条件变量*/
      void Get (int I)   /*定义取筷子过程*/
        {
      If chopstick[I]=1 and chopstick [ ( i+1 )%
        5] =1 then
        get the chopstick [I] and chopstick
        [ ( i+1 ) % 5];
      else X.wait; /*左右筷子没有，则等待*/
        }
    void Put (int i)   /*定义放下筷子过程*/
      {
      put the chopstick [I ] and chopstick
      ( i+1 ) % 5];
      If X.quene then X.signal;  /*唤醒等待筷子
        的哲学家*/
      }
    }
```

4.2 使用管程

第 I 个哲学家的活动：

```
void philosopher(int I)
  {
    while(true)
      {
        思考;
        get (I);
        进餐;
        put (I);
        }
      }
```

5 结束语

哲学家进餐问题是操作系统中一个常见且复杂的同步问题，它可为多个竞争进程或者线程互斥地访问有限资源这一类问题的解决提供思路。本文根据多年的教学所得，利用不同的机制探讨并提出了这一问题的解决思路。然后提出了解决死锁问题的相关思路，将其应用到教学实验中。提出的解决策略可有效地实现，并且避免饥饿和死锁现象的产生。希望与其他相关领域的学习者共享，方便操作系统的教学、学习和应用。

参考文献：

[1] Andrew S. Tanenbaum, Modern Operating System[M]. 2nd ed. Englewood Cliffs, New Jersey:Prentice Hall, 2001.

[2] Abraham S. 操作系统概念(影印版)[M].6版. 北京: 高等教育出版社,2002.

[3] 汤子瀛,哲凤屏,汤小丹. 计算机操作系统(修订版)[M]. 西安:西安电子科技大学出版社,2002.

[4] 周苏,金海溶,李洁. 操作系统原理实验[M]. 北京:科学出版社,2003.

计算机操作系统哲学家进餐问题的教学探讨

作者： 窦金凤， 曹家宝， 郭忠文， 刘颖健
作者单位： 窦金凤,郭忠文,刘颖健(中国海洋大学,山东青岛,266100)， 曹家宝(青岛阿尔卡特朗讯公司,山东青岛,266101)
刊名： 计算机教育
英文刊名： COMPUTER EDUCATION
年，卷(期)： 2009(14)

参考文献(4条)

1. 周苏;金海溶;李洁 操作系统原理实验 2003

2. 汤子瀛;哲凤屏;汤小丹 计算机操作系统(修订版) 2002

3. Abraham S 操作系统概念(影印版) 2002

4. Andrew S.Tanenbaum Modern Operating System 2001