基于 C 的线程同步互斥问题的研究

Research on Thread Synchronization and Mutual Exclusion Based on C

刘康媛　徐艳 * 文俊超　肖慧琴

Kangyuan Liu　Yan Xu*　Junchao Wen　Huiqin Xiao

成都锦城学院计算机与软件学院　中国・四川 成都 610000

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

摘　要: 论文提出了基于C的线程同步与互斥问题,并分析解决文中相关问题,论文中清楚明确地列出了分析思路与算法过程,通过代码和伪代码对线程同步互斥问题进行了编程模拟实现。

Abstract: This paper puts forward the problem of thread synchronization and mutual exclusion based on C, analyzes and solves the relevant problems in this paper, clearly lists the analysis ideas and algorithm process, and simulates the problem of thread synchronization and mutual exclusion through code and pseudo code.

关键词： 线程同步；信号量；PV 操作

Keywords: thread synchronization; signal; PV operation

DOI: 10.12346/sde.v4i3.6041

1 引言

进程是操作系统中资源分配的最小单位，线程是操作系统中被系统独立调度和分派的基本单位，相当于被剥离资源的进程。线程自身不拥有资源，但它与同属一个进程的其他线程共享进程所拥有的全部资源。在同一进程中的多个线程之间可以并发执行。

当一个线程正在修改某一存储区域的内容时，不允许其他线程来读或修改该存储区的内容，这种相互制约的关系称为互斥; 并发线程按各自的速度在向前推进，但在某一时刻，可能需要互相等候与互通消息， 来实现它们之间各操作之间的次序， 这种相互制约的等候与互通消息被称为同步。

2 问题描述

有一座桥，该桥的宽度不能容纳两辆车相向而行，在同一时刻只能有一个方向的车辆行驶，桥两端的人可以随意行走，同方向的车辆与行人互不影响，当车辆遇到对方向的行人时，考虑到安全问题，该车辆必须停止，行人不会停止，直到行人安全后再继续行驶。

该问题是由经典线程同步问题——独木桥问题改编而来，在独木桥问题中，仅要求同一时刻，桥上只能有一辆车行驶。

3 算法及编程实现

3.1 独木桥问题

首先对独木桥问题进行算法实现，问题要求同一时刻只能有一辆车行驶过桥，那么可以将过桥的权限看作一个互斥信号量，由左右两边准备过桥的车共享该信号量，用 bridge 表示该信号量，初值为 1。

```
sem_t bridge;
int conut1,count2;// 定义两个变量，用来表示左右两边需要过车的数量
```

把左右两边车辆过桥的动作用两个线程表示，由于左右两边过桥的代码意义上相同，所以只列出左边车辆的线程，代码如下：

```
void *left_car()
{
    srand((unsigned)time(NULL)*10);// 为了防止随机数每次重复
```

【作者简介】刘康媛（2001-），女，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。

```
        count1=rand()%20;// 车辆数目是 0-20 中随机产生的一个数
        printf(" 有 %d 辆车准备过左桥 \n",count1);
        while(1)
        {
                P(&bridge);// 得到上桥权限
                if(count1>0)
                {
                        printf(" 左边车辆 %d 过桥 ------",count1);
                        V(&bridge);// 车辆下桥
                        printf(" 左边车辆 %d 过桥结束 \n",count1);
                        count1--;
                }
                else
                {
                        V(&bridge);// 释放上桥权限
                        break;
                }
        }
    }
```

测试运行结果如图 1 所示。

```
$ ./dumu.exe
左边有6辆车准备过桥
右边有4辆车准备过桥
左边车辆6过桥------左边车辆6过桥结束
右边车辆4过桥------右边车辆4过桥结束
右边车辆3过桥------右边车辆3过桥结束
左边车辆5过桥------左边车辆5过桥结束
右边车辆2过桥------右边车辆2过桥结束
左边车辆4过桥------左边车辆4过桥结束
右边车辆1过桥------右边车辆1过桥结束
左边车辆3过桥------左边车辆3过桥结束
左边车辆2过桥------左边车辆2过桥结束
左边车辆1过桥------左边车辆1过桥结束
```

图 1 运行结果

满足了一次只能过一辆车的要求，该独木桥问题只用一个互斥信号量就能完成一个简易的过桥操作，只显示结果而没有过程的操作，改编后的题目，因为有了行人的介入，若只是像上述方法一样简单的显示出过桥的行为，则不能直观的感受到车辆在桥上遇到行人后停止的动作，所以需要一个过桥的过程。

3.2 过桥问题

该问题涉及到了桥左右的人与车，所以有分别对应左右人与车的四个线程，两边车的两个线程仍然共享一个互斥信号量 bridge，由于左边（右边）的车辆遇上右边（左边）的行人会停止，行人走后又继续行驶，所以两者之间存在同步关系，因为两边的行人不受约束，所以上桥的时候不用设置互斥信号量去约束他们，所以只需要设置两个同步信号量 SL、SR，初值为 0。

由于要显示出一个过桥的过程，所以可以设置等待、行进、停止、结束这四种状态，以便能跟清楚的看出当前车辆与行人的过桥状态。

```
#define WAITING 0       // 准备状态
#define WALKING 1       // 行进状态
#define STOPPING 2      // 停止状态
#define OVER 3                  // 结束状态
int state[4]={0};//state[0] 记录左边车辆的状态、state[1] 记录右边车辆的状态、state[2] 记录左边行人的状态、state[3] 记录右边行人的状态
sem_t bridge,SL,SR;// 初始值分别为 1、0、0
```

方法一：

该方法是假设当车辆与人都行至桥上时，因为左边的车辆与右边的行人和右边的车辆与左边的行人的情况相同，所以以左边的车辆和右边的行人为例，左边车辆判断对面是否有人，即判断右边行人是否与左边车辆同是行进状态，若是，则左边车辆进行同步信号量的 P 操作，由于同步信号量初始值为 0，所以该车辆对应的线程被挂起，状态由行走状态变为停止状态，而右边的行人在行至桥上时，也会判断对面是否有车辆，若有，则在行走完剩下的路程后进行对应同步信号量的 V 操作，若没有则不用进行 V 操作，而左边车辆会在右边行人进行 V 操作后继续行驶完成最后的路程。

伪代码如下：

```
// 左边车辆的线程
while(1)
{
    P(&bridge);// 得到上桥权限
    改变状态为 WAITING；
    改变状态为 WALKING；
    过桥；
    if(state[1]==WALKING&&state[3]==WALKING)
            满足条件则将状态改为停止状态
            P(&SL);
    继续过桥；
    过桥结束，状态改为 OVER；
    V(&bridge);// 释放上桥权限
}
// 右边行人的线程
while(1)
    {
    改变状态为 WAITING；
    改变状态为 WALKING；
            过桥；
            if(state[1]==WALKING&&state[3]==WALKING)
```

```
                    继续过桥；
                    过桥结束，状态变为 OVER；
                    V(&SL);
            else
                    继续过桥；
                    过桥结束，状态变为 OVER；
    }
```

该方法的弊端是，车辆与行人一上桥就会将状态改变为行进状态，如果车辆通过状态判断是否对面有行人，则车辆必然会变成停止状态，直到行人过桥结束后才能继续行驶，这样的现象缺乏合理性与公平性，所以要在该方法的基础上进行改进。

方法二：

设计一个既有过桥过程，又能对双方都公平的方法。假设把桥分成十段，过桥的过程就可以用 for 循环来表示。还是以左边的车辆和右边的行人为例， 代码如下：

```
// 左边车辆的线程
void *left_car()
{
    while(1)
    {
            P(&bridge);// 得到上桥的权限
            state[0]=WAITING;// 改变状态为等待状态，准备上桥
            printf("car_left waiting\n");
            sleep(1);
            go_to(0);// 该函数的作用是改变状态为行进状态
            car_left_wait();// 该函数表示为在桥上行驶的过程
            state[0]=OVER;// 过桥结束，改变状态
            printf("car_left over\n");
            sleep(1);
            V(&bridge);// 释放上桥的权限
    }
}
// 过桥过程的函数实现
void car_left_wait()
{
    int j;
    for(j=1;j<11;j++)
    {
            n=j;
            if(n+m==10||n+m==11)// 车辆判断是否该停下
            {
                    state[0]=STOPPING;
                    printf("car_left is stopping\n");
                    P(&SL);// 执行 P 操作，该线程被挂起
            }
            printf("car_left is walking %d\n",n);// 输出车辆已行驶的路程，只有这句话输出后，才能代表车辆正在行驶那段路程
    }
}
```

因为我们将桥分成了十段，所以可以通过 for 循环来循环十次表示过桥的过程，而函数中的 n 是表示左边车辆行驶的路程、m 是表示右边行人行走的路程，在车辆每行驶一段路程之前，都要判断与右边行人的距离是否满足 n+m==10 或 n+m==11。

n+m==10,n 与 m 的值是各自行进方向的行进路程，例如左边车辆已经行驶了 8 段路程（n），而右边行人行走了 2 段路程（m），即 n+m==10，这时他们在桥上就是相遇的情况，此时的车辆就必须停止，若继续行驶则可能会发生事故。

n+m==11，该条件也是一种相遇情况，因为同一时刻只能有一个线程正在执行，执行时会有三个线程并发执行，但其先后顺序都是随机的，没有固定顺序要求。例如，左边车辆行驶到第四段（n=4），右边行人也行走到（m=4），此时左边车辆先行驶，进入第五个循环（n=5），因为 n+m=9，所以车辆不会停止，右边行人也行走到第五段（m=5），若接下来是左边车辆要行驶，则进入第六层循环（n=6），此时并没有行驶到第六段，只是在行走前先做一个预判，判断是否会遇上行人，但此时 n+m=11，若只有 n+m==10 只一个判断条件，则左边车辆不会停止，会继续行驶，那么就会出现安全问题。

所以在车辆判断是否停止时，需要 n+m==10 或 n+m==11 这两个条件来判断是否已经遇上对面的行人。

```
// 右边行人线程
void *right_peo()
{
    while(1)
    {
            state[2]=WAITING; // 改变状态为等待状态，准备上桥
            printf("peo_right waiting\n");
            sleep(1);
            go_to(2); // 该函数的作用是改变状态为行进状态
            peo_right_wait();// 该函数表示为在桥上行驶的过程
```

```
            sleep(1);
        }
    }
    // 过桥过程的函数实现
    void peo_right_wait()
    {
        int j;
        for(j=1;j<11;j++)
        {
            m=j;
            printf("peo_right is walking %d\n",m);
            if(n+m==13&&state[0]==STOPPING)// 判断是否应该执行 V 操作
                V(&SL);
            sleep(1);
        }
        m=-1;
        state[2]=OVER;
        printf("peo_right over\n");
    }
```

在行人过桥时，会每走一段就要判断自己是否已经走过车辆，即判断 n+m==13 与对面车辆是否处于停止状态，若满足条件，则执行 V 操作，让车辆继续行驶，与方法一相比，方法二能让车辆能在最合适的时间继续行驶。

函数中的 m=-1 是让行人过桥结束后立即改变 m 的值，因为车辆是靠 n+m 的值来判断是否停下，若是当行人刚好过桥结束，还没有重新进入线程进入新的循环改变 m 值时，此时碰巧有车辆准备上桥，在进行第一段的预判时（n=1），就会因为 n+m==11 而进入停止状态，所以为了避免这种情况，在行人过桥结束时就立即改变 m 的值，也不能让 m=0，因为车辆也会因为 n+m==10 而进入停止状态。

测试运行截图如图 2 所示。

图中，0 is walking 的语句，前面的 0、1、2、3 分别代表左边车辆、右边车辆、左边行人与右边行人。过桥问题就是要通过使用信号量来让车辆实现停止与行驶的状态，从而根据各方面的因素写出了完整的人车过桥问题。代码所表示的是，一个线程表示一辆车或一位行人，而要实现一座桥上有多个人多辆车的情况，就需要一次创建多个同一种类的线程，但相应的，判断条件，约束语句也会变得相对麻烦许多。

4 方法总结

方法一是在过桥过程中，通过判断车辆与人是否都处于行进状态，是则车辆停止，直到行人行走结束，车辆才能继续行驶，这样严重浪费了车辆过桥的时间，也存在不公平问题。

方法二是建立在方法一的基础上，但又改进了方法一中出现的问题。方法二中将桥分成十段，车辆与行人都是通过 for 循环来循环十次表示过桥的过程，车辆在每一次循环都会判断是否与对面行人相遇，是则车辆停止，而行人也会在每次循环里判断是否经过对面的车辆，并且对面的车辆是否处于停止状态，若满足条件，则执行 V 操作，使车辆继续行驶。该方法更能清楚正确的表现出整过过桥过程所经历的状态变化。

方法二与方法一不同的一点就是，车辆不再等待行人过桥结束后再继续行驶，而是行人已走过车辆达到安全位置后再继续行驶。

参考文献

[1] 张步忠.Java语言中的线程同步互斥研究[J].安庆师范学院学报(自然科学版),2011,84(4):106-110.

[2] 章五一.线程同步及在C系列语言中的实现方式[J].电脑编程技巧与维护,2015,338(20):8-9+20.

[3] 田春婷.基于Java多线程同步技术的简易模拟售票系统实现[J].电脑编程技巧与维护,2018(12):72-73+113.


```
car_left waiting
peo_left waiting
peo_right waiting
2 is walking
0 is walking
3 is walking
peo_right is walking 1
car_left is walking 1
peo_left is walking 1
car_left is walking 2
car_left is walking 3
car_left is walking 4
car_left is walking 5
car_left is walking 6
car_left is walking 7
car_left is walking 8
car_left is stopping
peo_left is walking 2
peo_right is walking 2
peo_right is walking 3
peo_left is walking 3
peo_right is walking 4
peo_left is walking 4
car_left is walking 9
car_left is walking 10
car_left over
peo_left is walking 5
peo_right is walking 5
car_right waiting
1 is walking
peo_right is walking 6
peo_left is walking 6
car_right is walkng 1
car_right is walkng 2
car_right is walkng 3
car_right is stopping
peo_left is walking 7
peo_right is walking 7
peo_left is walking 8
peo_right is walking 8
peo_left is walking 9
car_right is walkng 4
car_right is walkng 5
car_right is walkng 6
car_right is walkng 7
car_right is walkng 8
car_right is walkng 9
car_right is walkng 10
car_right over
peo_right is walking 9
peo_left is walking 10
car_left waiting
peo_right is walking 10
peo_left over
0 is walking
peo_right over
car_left is walking 1
car_left is walking 2
car_left is walking 3
car_left is walking 4
car_left is walking 5
car_left is walking 6
car_left is walking 7
car_left is walking 8
car_left is walking 9
car_left is walking 10
car_left over
peo_left waiting
car_right waiting
peo_right waiting
```

图 2 测试运行截图