实验5 独木桥

实验目的

用信号量来解决实际进程互斥问题，进而加深对信号量、进程互斥的理解，用信号量解决现实问题。

实验内容

整体部分

线程部分

通过分别创立west线程和east线程来模拟从西方向和东方向来的人。

函数部分

通过构建不同线程中执行的不同函数来实现东西方向上桥的方式不同。

第一部分：每次只允许一个人过桥

变量使用

```
sem_t mutex;  //独木桥实现互斥访问，赋予的初始值为1
int i=0;      //一个普通的计数量
int count=0;//用于全局计数，方便观察。
```

伪代码

```
//线程中执行部分
P(mutex);  //加锁
count++; //全局过桥人数++
//过桥

V(mutex); //解锁
```

通过加锁来使独木桥只能一个人访问，在过桥后解锁，后来的再上去，实现只允许一个人过桥。

遇到困难

相对来说还是比较简单的，在算法的部分到没有什么阻碍，带上怎么体现出加锁的过程当然头痛，最后我想到，当每次count加减时打印出对应的当前桥上的人就可以清晰的体现出加锁解锁的过程，之后两个部分的实验也是通过这样的方法。

代码实现

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <time.h>
#include <semaphore.h>
#include <unistd.h>
void* gone();
sem_t mutex;  //独木桥实现互斥访问
 int i=0;     //用于全局计数，方便观察。
int main() {
    sem_init(&mutex, 0, 1);
    int  pc;
    pthread_t p;
    for ( ;i < 15; i++) {
        pc = pthread_create(&p, NULL, gone, NULL);
        if (pc != 0 )
            printf("Thread creation failure!\n");
    }
    sleep(2);
    return EXIT_SUCCESS;

}
int count=0;
void *gone(){
        sem_wait(&mutex);
        count++;//全局过桥人数++
        printf("第%d号进\n",count);
       sleep(0.1);
        printf("第%d号出\n",count);
        sem_post(&mutex);

}
```

实现结果（测试）

```
第1号进
第1号出
第2号进
第2号出
第3号进
 ... ...
第10号进
第10号出
第11号进
第11号出
第12号进
第12号出
第13号进
第13号出
第14号进
```

```
第14号出
第15号进
第15号出
```

####

第二部分： 当独木桥上有行人时，同方向的行人可以同时过桥，相反方向的人必须等待

PS这部分我做了好几个小时，写了好几种方法，但最后还是不太满意。

变量使用

```
sem_t mutex;        //独木桥实现互斥访问
sem_t temp;         //用于访问判断的互斥，防止
int eastCount = 0; //记录当前桥上东方向的人
int westCount = 0; //记录当前桥上西方向的人
int flag=0;         //用于标记桥正在被使用 为1时桥正在被使用
```

mutex作为互斥量来实现桥的互斥访问，但实际效果很不好，因为如果另外一个方向的线程都卡在sem_wait(&mutex);中的话，当一个方向的人都走完后，那么这个方向的线程就会进行一个开一个锁只能进一个的情况，因为后面的线程还是会被锁住。导致实现的效果是进1出1再进1出1这样，没办法实现真正的同时过桥。因此添加了flag作为上锁的标记，用于判断什么时候处于有人在桥上。eastCount 和 westCount 用于两个方向的计数，同时用于打印观察。

伪代码

函数west和east实现上面相似，这里只写其中一部分。

```
P(temp);
//作为临时的加锁，防止多个线程同时访问判断区，导致同时修改判断区。
//如果没有P(&temp)的话，所有的线程可能同时访问自己函数，导致最后结果出现bug。

//进行桥上是否有人的判断

if(flag=0)//桥上没人
{
//进行加锁等操作
P(mutex);
   count++;
    flag=1;
   V(temp);解锁
}
else if(eastCount!=0)//桥上是自己方向的人
{
    count++;
   V(temp);解锁
}
else{ //两者都不是
   V(temp);解锁
    标记
   while(flag!=0&&eastCount==0){}//循环等待另外一边的人
//等待结束
```

```
    P(temp);//同样方式实现互斥方式

    if(flag==0)//可以上桥
        flag=1;
      P(mutex);
     else if(eastCount)//当前表示自己的人不在上面
         V(temp);
         跳转到标记//跳转回循环等待
         ... ...
     count++;
     V(temp);
}

//过去桥

P(temp);//互斥访问count部分

count部分--;
if(count==0)表示桥上已经没有自己的人
{
    flag=0;
     V(mutex);//解锁
}
```

先给判断区加锁，实现互斥访问，接着判断是否可以上桥，是就执行上桥的指令，否则就进入循环区等待临界区使用完成。

遇到困难

这一部分遇到的困难还是比较多：

1.一开始实现不了互斥的效果，就是双方之间会直接的同时的进行上桥，锁都锁不住，反复运行结果后，我发现两个线程的函数和同时进行，会同时判断桥上是否有人这一步，导致会同时上桥。

解决方法：加入了temp这个临时的锁，让访问判断桥是否空闲的部分可以互斥进行，这样就解决了第一步同时访问判断部分。

2.解决之后，又发现当一个方向的都过去解锁之后，但另一个方向的因为卡在p(mutex)阶段，导致解锁后，过去一个，又马上加锁。没办法一起进去。

解决方法：不把等待上桥的线程卡在p(mutex)阶段，而是在判断无法上桥后放在while循环中暂存。当一个方向全部下桥后，就再判断是否可以上桥。

3.但是之后又出现，可能两个方向都可能有人在等待，然后两个方向都有人同步上线判断，虽然做了互斥判断，但还是会出现一个加锁没开锁的情况。

解决方法：在苦思无果了，我用了goto语句，当判断自己没赶上判断后，就解锁跳转到while循环中重新循环。缺点就是可能产生饥饿。

之后：我暂时没有在出现新bug,但个人电脑中我只能同时开60个线程左右等待，再多可能就会奔溃，而且我可能有点把问题复杂化，可能会有很好的解法，我问了几个同学，他们写的都不太符合我的思路。

代码实现

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <time.h>
#include <semaphore.h>
#include <unistd.h>


void* east();
void* west();
sem_t mutex;        //独木桥实现互斥访问
sem_t temp;         //用于访问判断的互斥
int eastCount = 0; //记录当前桥上东方向的人
int westCount = 0; //记录当前桥上西方向的人
int flag=0;


int main() {
    sem_init(&mutex, 0, 1);
    sem_init(&temp, 0, 1);
    int i, pc1, pc2;
    pthread_t p;
    for (i = 0; i < 7; i++) {
        pc2 = pthread_create(&p, NULL, east, NULL);
        pc1 = pthread_create(&p, NULL, west, NULL);
        if (pc1 != 0 || pc2 != 0)
            printf("Thread creation failure!\n");

    }
    sleep(0.1);
    for (i = 0; i < 7; i++) {
        pc2 = pthread_create(&p, NULL, east, NULL);
        pc1 = pthread_create(&p, NULL, west, NULL);
        if (pc1 != 0 || pc2 != 0)
            printf("Thread creation failure!\n");
    }
    sleep(50);
    return EXIT_SUCCESS;
}
void* east() {
     sem_wait(&temp);  //上锁防止多个线程同时访问桥的状态，导致死锁
     if(flag==0){//桥未使用，可以进
         flag=1;
         sem_wait(&mutex);
         eastCount++;
         printf("东方进\n");
          printf("当前桥上东方人数为%d\n",eastCount);
         sem_post(&temp);//判断完解开临时的锁
     }
     else if(eastCount!=0){//桥上的自己的人，可以进
        eastCount++;
        printf("东方进\n");
```

```
            printf("当前桥上东方人数为%d\n",eastCount);
        sem_post(&temp);
     }
     else{  //桥在被使用，而且没有自己的人，循环等待
        sem_post(&temp); //临时上锁，防止同时访问，导致死锁
        again1:
        while(flag!=0&&eastCount==0){}
        sem_wait(&temp);
         if(flag==0) {
         flag=1;//等待结束后，自己立刻上桥，上锁，防止对面又上人
         sem_wait(&mutex);
        }
        else if(eastCount==0){//不是自己人在上面
            sem_post(&temp); //别人先上了 ，回去继续等待
            goto again1;
         }

         eastCount++;
         printf("东方进\n");
         printf("当前桥上东方人数为%d\n",eastCount);
         sem_post(&temp);
     }

    //过桥时间
    sleep(0.1);

    sem_wait(&temp);
    eastCount--;
    printf("东方出\n");
     printf("当前桥上东方人数为%d\n",eastCount);
    if (eastCount == 0){
        sem_post(&mutex);
        flag=0;
    }
     sem_post(&temp);

}

void* west() {
          sem_wait(&temp);  //上锁防止多个线程同时访问桥的状态，导致死锁
     if(flag==0){//桥未使用，可以进
         flag=1;
         sem_wait(&mutex);
         westCount++;
         printf("西方进\n");
          printf("当前桥上西方人数为%d\n",westCount);
         sem_post(&temp);//判断完解开临时的锁
     }
     else if(westCount!=0){//桥上的自己的人，可以进
        westCount++;
         printf("西方进\n");
          printf("当前桥上西方人数为%d\n",westCount);
        sem_post(&temp);
     }
     else{  //桥在被使用，而且没有自己的人，循环等待
        sem_post(&temp);
        again2:
        while(westCount==0&&flag!=0){}
```

```
        sem_wait(&temp);
         if(flag==0) {
         flag=1;//等待结束后，自己立刻上桥，上锁，防止对面又上人
         sem_wait(&mutex);
        }
        else if(westCount==0){//不是自己人在上面
            sem_post(&temp); //别人先上了 ，回去继续等待
            goto again2;
         }
         westCount++;
         printf("西方进\n");
          printf("当前桥上西方人数为%d\n",westCount);
          sem_post(&temp);
     }



    sleep(0.1);



    sem_wait(&temp);
    westCount--;
     printf("西方出\n");
    printf("当前桥上西方人数为%d\n",westCount);
    if (westCount == 0){
        sem_post(&mutex);
        flag=0;
    }
    sem_post(&temp);

}
```

实现结果（测试）

例:

正确:

```
东方进
当前桥上东方人数为1
东方进
当前桥上东方人数为2
东方出
当前桥上东方人数为1
东方进
当前桥上东方人数为2
东方进
当前桥上东方人数为3
东方出
当前桥上东方人数为2
东方进
当前桥上东方人数为3
东方进
当前桥上东方人数为4
东方出
```

```
当前桥上东方人数为3    //会有出去的时候
东方进
当前桥上东方人数为4
东方进
当前桥上东方人数为5
东方出
当前桥上东方人数为4
东方进
当前桥上东方人数为5
东方进
当前桥上东方人数为6
东方进
当前桥上东方人数为7
东方出
当前桥上东方人数为6
东方进
当前桥上东方人数为7
东方进
当前桥上东方人数为8
东方出
当前桥上东方人数为7
东方出
当前桥上东方人数为6
东方进
当前桥上东方人数为7
东方出
当前桥上东方人数为6
东方出
当前桥上东方人数为5
东方出
当前桥上东方人数为4
东方出
当前桥上东方人数为3
东方出
当前桥上东方人数为2
东方出
当前桥上东方人数为1
东方出
当前桥上东方人数为0

//东方的人都过桥

西方进
当前桥上西方人数为1
西方进
当前桥上西方人数为2
西方进
当前桥上西方人数为3
西方进
当前桥上西方人数为4
西方进
当前桥上西方人数为5
西方进
当前桥上西方人数为6
西方进
当前桥上西方人数为7
西方进
当前桥上西方人数为8
```

```
西方进
当前桥上西方人数为9
西方进
当前桥上西方人数为10
西方进
当前桥上西方人数为11
西方进
当前桥上西方人数为12
西方进
当前桥上西方人数为13
西方进
当前桥上西方人数为14

//等待后一起上

西方出
当前桥上西方人数为13
西方出
当前桥上西方人数为12
西方出
当前桥上西方人数为11
西方出
当前桥上西方人数为10
西方出
当前桥上西方人数为9
西方出
当前桥上西方人数为8
西方出
当前桥上西方人数为7
西方出
当前桥上西方人数为6
西方出
当前桥上西方人数为5
西方出
当前桥上西方人数为4
西方出
当前桥上西方人数为3
西方出
当前桥上西方人数为2
西方出
当前桥上西方人数为1
西方出
当前桥上西方人数为0
```

错误：

//这部分的错误案例非常非常的错，因为前期之后没有截下来，这里就方一段后期出现的错误。

```
东方进
当前桥上东方人数为1
东方进
当前桥上东方人数为2
东方进
当前桥上东方人数为3
```

```
东方进
当前桥上东方人数为4
东方出
当前桥上东方人数为3
东方进
当前桥上东方人数为4
东方进
当前桥上东方人数为5
......

//看这边出现了东方和西方都上桥的情况，是因为
//在第二次判断时
while(westCount==0&&flag!=0){}
        sem_wait(&temp);//这里没有把判断区加锁
         if(flag==0) {
当前桥上东方人数为1
东方进
当前桥上东方人数为1
西方进
当前桥上西方人数为2
东方出
当前桥上东方人数为0
西方出
当前桥上西方人数为2
西方进
当前桥上西方人数为2
西方进
当前桥上西方人数为1
西方出
当前桥上西方人数为0
西方出
当前桥上西方人数为0
东方进
当前桥上东方人数为2
东方进
当前桥上东方人数为1
东方出
当前桥上东方人数为0
东方出
当前桥上东方人数为0
西方进
当前桥上西方人数为1
东方进
当前桥上东方人数为1
西方出
当前桥上西方人数为0
东方出
当前桥上东方人数为0
```

第三部分：当独木桥上有自东向西的行人时，同方向的行人可以同时过桥，从西向东的方向，只允许一个人单独过桥

变量使用

```
sem_t mutex;  //独木桥实现互斥访问
sem_t temp;
int eastCount = 0; //标记当前桥上从东向西的人
int westCount = 0; //标记当前桥上从西向东的人
int flag=0;
```

和第二部分的变量和含义都相同。

伪代码

```
//当独木桥上有自东向西的行人时，同方向的行人可以同时过桥//
从东到西的伪代码
这一部分和第二部分的伪代码是一样的。

//写从西到东的伪代码
P(temp);//加锁实现互斥访问判断上桥
if(flag==0)//桥当前没人
{
    p(mutex);
    flag=1;//表示临界区被使用
    count++;
    v(temp);
}
else
{
    v(temp);//解开临时锁
    标记
        while(){}//循环等待
    p(temp);//结束等待，重新互斥访问判断区
    if(flag==0)//没人在桥上
    {
        flag=1;
        p(mutex);
    }
    else
    {
        v(temp);//解开临时锁
        跳转到标记
    }
    count++;
    v(temp);//完全判断成功,解开临时锁

}
p(temp);   //互斥访问count
count--;
v(mutex);
flag=0;
v(temp);
```

遇到困难

有了第二部分的铺垫，第三阶段因而没有什么bug。

1.第三阶段是在第二阶段基础上修改，从东方向基本没变，而从西方向只需要把可以同时进，改成进一个就上锁，而且相同的时候也不解锁，只有当西方向人下桥后解锁，但在这时候就有判断问题。

解决方法:把相同时候的可以上桥的代码删除，接着把西方向第二个要上桥的部分加到循环中就可以了。

代码实现

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <time.h>
#include <semaphore.h>
#include <unistd.h>
void* east();
void* west();
sem_t mutex;  //独木桥实现互斥访问
sem_t temp;
int eastCount = 0; //标记当前桥上从东向西的人
int westCount = 0; //标记当前桥上从西向东的人
int flag=0;

int main() {
    sem_init(&mutex, 0, 1);
    sem_init(&temp, 0, 1);
    int i, pc1, pc2;
    pthread_t p;
    for (i = 0; i < 3; i++) {


        pc2 = pthread_create(&p, NULL, west, NULL);
        pc1 = pthread_create(&p, NULL, east, NULL);
        if (pc1 != 0 || pc2 != 0)
            printf("Thread creation failure!\n");

    }

    for (i = 0; i < 3; i++) {

        pc2 = pthread_create(&p, NULL, east, NULL);
        pc1 = pthread_create(&p, NULL, west, NULL);
        if (pc1 != 0 || pc2 != 0)
            printf("Thread creation failure!\n");

    }

    sleep(5);
    return EXIT_SUCCESS;

}
void* east() {
```

```
      sem_wait(&temp);  //上锁防止多个线程同时访问桥的状态，导致死锁
      if(flag==0){//桥未使用，可以进
          flag=1;
          sem_wait(&mutex);
          eastCount++;
          printf("东方进\n");
           printf("当前桥上东方人数为%d\n",eastCount);
          sem_post(&temp);//判断完解开临时的锁
      }
      else if(eastCount!=0){//桥上的自己的人，可以进
         eastCount++;
         printf("东方进\n");
           printf("当前桥上东方人数为%d\n",eastCount);
         sem_post(&temp);
      }
      else{  //桥在被使用，而且没有自己的人，循环等待
         sem_post(&temp);
         again1:
         while(flag!=0&&eastCount==0){}
         sem_wait(&temp);//临时上锁，防止同时访问，导致死锁
          if(flag==0) {
          flag=1;//等待结束后，自己立刻上桥，上锁，防止对面又上人
          sem_wait(&mutex);
         }
         else if(eastCount==0){//不是自己人在上面
             sem_post(&temp); //别人先上了 ，回去继续等待
             goto again1;
          }
          eastCount++;
          printf("东方进\n");
          printf("当前桥上东方人数为%d\n",eastCount);
          sem_post(&temp);
      }


     //过桥时间
     sleep(0.1);
     sem_wait(&temp);
     eastCount--;
     printf("东方出\n");
      printf("当前桥上东方人数为%d\n",eastCount);
     if (eastCount == 0){
         sem_post(&mutex);
         flag=0;
     }
     sem_post(&temp);

}

void* west() {
     sem_wait(&temp);
     if(flag==0){
         sem_wait(&mutex);
         flag=1;
         westCount++;
          printf("西方进\n");
          printf("当前桥上西方人数为%d\n",westCount);
          sem_post(&temp);
```

```
    }
     else{
     sem_post(&temp);
     again2:
     while(flag==1){}
          sem_wait(&temp);
           if(flag==0) {//桥上没人
             flag=1;//等待结束后，自己立刻上桥，上锁，防止对面又上人
             sem_wait(&mutex);
         }
         else{//如果有东方向的人或者又一个西方的人
             sem_post(&temp); //别人先上了 ，回去继续等待
             goto again2;
          }
          westCount++;


          printf("西方进\n");
          printf("当前桥上西方人数为%d\n",westCount);
          sem_post(&temp);

    }

    sleep(0.1);
    sem_wait(&temp);
    westCount--;
     printf("西方出\n");
    printf("当前桥上西方人数为%d\n",westCount);
    sem_post(&mutex);
    flag=0;
    sem_post(&temp);

}
```

实现结果（测试）

```
西方进
当前桥上西方人数为1
西方出
当前桥上西方人数为0
西方进
当前桥上西方人数为1
西方出
当前桥上西方人数为0
东方进
当前桥上东方人数为1
东方进
当前桥上东方人数为2
东方进
当前桥上东方人数为3
东方进
当前桥上东方人数为4
东方进
当前桥上东方人数为5
东方进
```

```
当前桥上东方人数为6
东方出
当前桥上东方人数为5
东方出
当前桥上东方人数为4
东方出
当前桥上东方人数为3
东方出
当前桥上东方人数为2
东方出
当前桥上东方人数为1
东方出
当前桥上东方人数为0
西方进
当前桥上西方人数为1
西方出
当前桥上西方人数为0
西方进
当前桥上西方人数为1
西方出
当前桥上西方人数为0
西方进
当前桥上西方人数为1
西方出
当前桥上西方人数为0
西方进
当前桥上西方人数为1
西方出
当前桥上西方人数为0
```

总结

这个实验说是很简单，但要做到完全的独木桥，要考虑的元素还是很多的，第一版其实出的很快，而且没有实时检查count部分，这样出来的结果看的很像实现和互斥访问，但加了count实时检查后，就出现了很明显的bug,就这样在不断的修bug和产生bug中不断的尝试。写了十几个版本，很累，但这其中，我也不断深入的掌握互斥和信息素部分。

但希望下次自己先深入的思考后在码，不然就会不断的改代码中内耗。