**操作系统课程设计**

**题目：**多线程实现“哲学家进餐”问题

**班级：** 计算机1801

**姓名**： 汤礽禾

**学号**： 181604124

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 项目名称 | 考核比重 | 得分 |
| 1 | 方案设计 | 10% |  |
| 2 | 系统设计 | 50% |  |
| 3 | 系统验收 | 15% |  |
| 4 | 课设报告 | 25% |  |
|  | 总成绩 | 100% |  |

2020年6月

目录

[一、课程设计目标 2](#_Toc21478)

[二、课题内容 2](#_Toc31781)

[三、 设计思路 3](#_Toc2835)

[四、源代码 4](#_Toc4467)

[五、运行与测试 6](#_Toc17992)

[六、 心得体会 7](#_Toc27550)

# 一、**课程设计目标**

学习多线程编程，使用线程的同步机制实现“哲学家进餐”问题。具体要求：

1.创建POSIX线程，实现多线程的并发执行，验证多线程共享进程资源的特性。

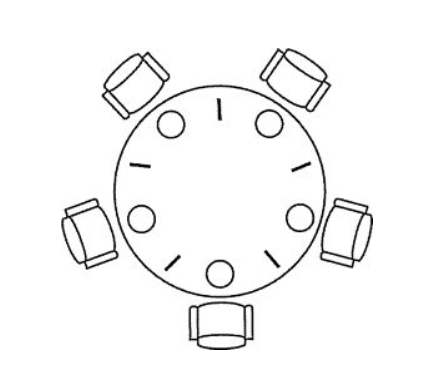
2.使用互斥量和条件变量，或使用信号量实现线程的同步互斥。

3. 验证 “ 哲学家进餐”问题中的死锁情况，并加以解决。

**二、课题内容**

有五个哲学家围坐在一圆桌旁，桌中央有一盘通心粉，每人面前有一只空盘子，每两人之间放一只筷子，即共5只筷子。

每个哲学家的行为是思考和进餐。为了进餐，每个哲学家必须拿到两只筷子，并且每个人只能直接从自己的左边或右边去取筷子。思考时则同时将两支筷子放回原处



规则：

①只有拿到两只筷子时，哲学家才能吃饭；

②如果筷子已经在他人手上，则该哲学家必须等到他人吃完之后才能拿到筷子；

③任何一个哲学家在自己没有拿到两只筷子吃饭之前，决不放下自己手中的筷子。

由此出现的问题：

可能出现死锁问题，因为当五个哲学家都饥饿时，都拿着一支筷子，这样就可能五个哲学家都用不上餐。

1. **设计思路**

**3.1问题描述：**

放在桌子上的筷子是临界资源，在一段时间内只允许一位哲学家使用，为了实现对筷子的互斥使用，可以用一个信号量表示一只筷子，由这五个信号量组成信号量数组。当哲学家饥饿时总是先拿其左边的筷子，成功后，再去拿右边的筷子，又成功后方可就餐。进餐完，又先放下他左边的筷子，再放下右边筷子。这个算法可以保证不会有两个相邻的哲学家同时就餐，但有可能引起死锁。

**3.2死锁产生条件：**

死锁产生的条件：

（1）互斥条件。拿到东西以后不放下，其他人用需要等待

（2）请求与保持条件。获得一些资源，然后还需要另外一些资源时阻塞，原来的不放下。

（3）不剥夺条件。别人拿了之后别人不能抢。

（4）循环与等待条件。即若干线程之间形成一种头尾相接的循环等待关系。

**3.3死锁几种解决方案：**

(1)至多只允许四个哲学家同时进餐，以保证至少有一个哲学家可以进餐，最终总会释放出他所用过的两只筷子，从而可使更多的哲学家进餐；

(2)仅当左右两只筷子均可用时，才允许哲学家拿起筷子就餐

(3)规定奇数号哲学家先拿起左边筷子，然后再去拿右边筷子，而偶数号哲学家则相反。

(4)把筷子顺序编号 fk0, fk1, fk2, fk3, fk4，给每个哲学家分配筷子时，必须依从小号到大号（或者相反顺序）进行。

此实验采用（1）（2）（3）解决死锁。

**3.4模块构建**

拿起筷子：

void pickup\_forks(int id)

{

pthread\_mutex\_lock(&mutex);

if(sticks[id]==0){

pthread\_cond\_wait(&chopstick[id],&mutex);

}

sticks[id]=0;

pthread\_mutex\_unlock(&mutex);

}

放下筷子：

void return\_forks(int id)

{

pthread\_mutex\_lock(&mutex);

sticks[id]=1;

pthread\_cond\_signal(&chopstick[id]);

pthread\_mutex\_unlock(&mutex);

}

吃：

void eat(int id)

{

printf("哲学家 %d 正在吃饭\n",id+1);

sleep(1);

}

思考：

void think(int id)

{

printf("哲学家 %d 正在思考\n",id+1);

sleep(3);

}

哲学家：

while(1){

int id=(int)arg;

think(id);

pthread\_mutex\_lock(&mutex1);

pickup\_forks(id);

printf("哲学家 %d 拿起 %d 号筷子\n",id+1,id);

pickup\_forks((id+1)%5);

printf("哲学家 %d 拿起 %d 号筷子\n",id+1,(id+1)%5);

pthread\_mutex\_unlock(&mutex1);

eat(id);

return\_forks((id+1)%5);

printf("哲学家 %d 放下 %d 号筷子\n",id+1,id);

return\_forks(id);

printf("哲学家 %d 放下 %d 号筷子\n",id+1,(id+1)%5);

}

1. **源代码**

**若不增加任何限制，则发生死锁：**

#include<stdio.h>

#include<stdlib.h>

#include<pthread.h>

#include<unistd.h>

pthread\_mutex\_t mutex=PTHREAD\_MUTEX\_INITIALIZER;

pthread\_cond\_t chopstick[5];//条件变量，不是锁

int sticks[5]={1,1,1,1,1};

void think(int id)

{

printf("哲学家 %d 正在思考\n",id+1);

sleep(1);

}

void eat(int id)

{

printf("哲学家 %d 正在吃饭\n",id+1);

sleep(1);

}

void pickup\_forks(int id)

{

pthread\_mutex\_lock(&mutex);

if(sticks[id]==0){

pthread\_cond\_wait(&chopstick[id],&mutex);

}

sticks[id]=0;

pthread\_mutex\_unlock(&mutex);

}

void return\_forks(int id)

{

pthread\_mutex\_lock(&mutex);

sticks[id]=1;

pthread\_cond\_signal(&chopstick[id]);

pthread\_mutex\_unlock(&mutex);

}

void \*philosopher(void \*arg)

{

while(1){

int id=(int)arg;

think(id);

pickup\_forks(id);

//printf("哲学家 %d 拿起 %d 号筷子\n",id+1,id);

pickup\_forks((id+1)%5);

//printf("哲学家 %d 拿起 %d 号筷子\n",id+1,(id+1)%5);

eat(id);

return\_forks((id+1)%5);

//printf("哲学家 %d 放下 %d 号筷子\n",id+1,id);

return\_forks(id);

//printf("哲学家 %d 放下 %d 号筷子\n",id+1,(id+1)%5);

}

}

int main()

{

int i;

pthread\_t \*ph=(pthread\_t \*)malloc(sizeof(pthread\_t)\*5);

for(i=4;i>=0;i--){

pthread\_cond\_init(&chopstick[i],NULL);

//sleep(1);

}

for(i=4;i>=0;i--){

//pthread\_cond\_init(&chopstick[i],NULL);

pthread\_create(&ph[i],NULL,philosopher,(void \*)i);

sleep(1);

}

for(i=0;i<5;i++){

pthread\_join(ph[i],NULL);

}

pthread\_mutex\_destroy(&mutex);

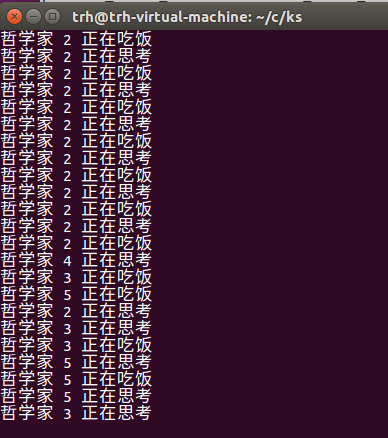
for(i=0;i<5;i++){

pthread\_cond\_destroy(&chopstick[i]);

}

return 0;

}



可发现此时进程已经不动了，说明发生死锁。

**法1：一个哲学家一次拿两个筷子（破坏请求保持）**

#include<stdio.h>

#include<stdlib.h>

#include<pthread.h>

#include<unistd.h>

pthread\_mutex\_t mutex=PTHREAD\_MUTEX\_INITIALIZER;

pthread\_mutex\_t mutex1=PTHREAD\_MUTEX\_INITIALIZER;

pthread\_cond\_t chopstick[5];//条件变量，不是锁

int sticks[5]={1,1,1,1,1};

void think(int id)

{

printf("哲学家 %d 正在思考\n",id+1);

sleep(1);

}

void eat(int id)

{

printf("哲学家 %d 正在吃饭\n",id+1);

sleep(1);

}

void pickup\_forks(int id)

{

pthread\_mutex\_lock(&mutex);

if(sticks[id]==0){

pthread\_cond\_wait(&chopstick[id],&mutex);

}

sticks[id]=0;

pthread\_mutex\_unlock(&mutex);

}

void return\_forks(int id)

{

pthread\_mutex\_lock(&mutex);

sticks[id]=1;

pthread\_cond\_signal(&chopstick[id]);

pthread\_mutex\_unlock(&mutex);

}

void \*philosopher(void \*arg)

{

while(1){

int id=(int)arg;

think(id);

pthread\_mutex\_lock(&mutex1);

pickup\_forks(id);

printf("哲学家 %d 拿起 %d 号筷子\n",id+1,id);

pickup\_forks((id+1)%5);

printf("哲学家 %d 拿起 %d 号筷子\n",id+1,(id+1)%5);

pthread\_mutex\_unlock(&mutex1);

eat(id);

return\_forks((id+1)%5);

printf("哲学家 %d 放下 %d 号筷子\n",id+1,id);

return\_forks(id);

printf("哲学家 %d 放下 %d 号筷子\n",id+1,(id+1)%5);

}

}

int main()

{

int i;

pthread\_t \*ph=(pthread\_t \*)malloc(sizeof(pthread\_t)\*5);

for(i=4;i>=0;i--){

pthread\_cond\_init(&chopstick[i],NULL);

//sleep(1);

}

for(i=4;i>=0;i--){

//pthread\_cond\_init(&chopstick[i],NULL);

pthread\_create(&ph[i],NULL,philosopher,(void \*)i);

sleep(1);

}

for(i=0;i<5;i++){

pthread\_join(ph[i],NULL);

}

pthread\_mutex\_destroy(&mutex);

for(i=0;i<5;i++){

pthread\_cond\_destroy(&chopstick[i]);

}

return 0;

}

**法2：最多四个哲学家同时拿筷子（破坏循环等待）**

#include<stdio.h>

#include<stdlib.h>

#include<pthread.h>

#include <semaphore.h>

#include<unistd.h>

pthread\_mutex\_t mutex=PTHREAD\_MUTEX\_INITIALIZER;

sem\_t S;

pthread\_cond\_t chopstick[5];

int sticks[5]={1,1,1,1,1};

void think(int id)

{

printf("哲学家 %d 正在思考\n",id+1);

sleep(1);

}

void eat(int id)

{

printf("哲学家 %d 正在吃饭\n",id+1);

sleep(1);

}

void pickup\_forks(int id)

{

pthread\_mutex\_lock(&mutex);

if(sticks[id]==0){

pthread\_cond\_wait(&chopstick[id],&mutex);

}

sticks[id]=0;

pthread\_mutex\_unlock(&mutex);

}

void return\_forks(int id)

{

pthread\_mutex\_lock(&mutex);

sticks[id]=1;

pthread\_cond\_signal(&chopstick[id]);

pthread\_mutex\_unlock(&mutex);

}

void \*philosopher(void \*arg)

{

while(1){

int id=(int)arg;

think(id);

sem\_wait(&S);

pickup\_forks(id);

pickup\_forks((id+1)%5);

sem\_post(&S);

eat(id);

return\_forks((id+1)%5);

return\_forks(id);

}

}

int main()

{

printf("4 PHILOSOPHERS ONLY\n");

int i;

sem\_init(&S, 0, 4);

pthread\_t \*ph=(pthread\_t \*)malloc(sizeof(pthread\_t)\*5);

for(i=4;i>=0;i--){

pthread\_cond\_init(&chopstick[i],NULL);

//sleep(1);

}

for(i=4;i>=0;i--){

pthread\_create(&ph[i],NULL,philosopher,(void \*)i);

sleep(1);

}

for(i=0;i<5;i++){

pthread\_join(ph[i],NULL);

}

pthread\_mutex\_destroy(&mutex);

for(i=0;i<5;i++){

pthread\_cond\_destroy(&chopstick[i]);

}

return 0;

}

**法3：奇数号哲学家从左到右拿筷子，偶数号哲学家相反（破坏循环等待）**

#include<stdio.h>

#include<stdlib.h>

#include<pthread.h>

#include<unistd.h>

pthread\_mutex\_t mutex=PTHREAD\_MUTEX\_INITIALIZER;

pthread\_mutex\_t mutex1=PTHREAD\_MUTEX\_INITIALIZER;

pthread\_cond\_t chopstick[5];

int sticks[5]={1,1,1,1,1};

void think(int id)

{

printf("哲学家 %d 正在思考\n",id+1);

sleep(1);

}

void eat(int id)

{

printf("哲学家 %d 正在吃饭\n",id+1);

sleep(1);

}

void pickup\_forks(int id)

{

pthread\_mutex\_lock(&mutex);

if(sticks[id]==0){

pthread\_cond\_wait(&chopstick[id],&mutex);

}

sticks[id]=0;

pthread\_mutex\_unlock(&mutex);

}

void return\_forks(int id)

{

pthread\_mutex\_lock(&mutex);

sticks[id]=1;

pthread\_cond\_signal(&chopstick[id]);

pthread\_mutex\_unlock(&mutex);

}

void \*philosopher(void \*arg)

{

while(1){

int id=(int)arg;

think(id);

if(id%2==0)

{//left to right

pickup\_forks(id);

pickup\_forks((id+1)%5);

}

else

{

pickup\_forks((id+1)%5);

pickup\_forks(id);

}

eat(id);

return\_forks((id+1)%5);

return\_forks(id);

}

}

int main()

{

int i;

printf("only take 2 chopsticks\n");

pthread\_t \*ph=(pthread\_t \*)malloc(sizeof(pthread\_t)\*5);//5

for(i=4;i>=0;i--){

pthread\_cond\_init(&chopstick[i],NULL);

//sleep(1);

}

for(i=4;i>=0;i--){

//pthread\_cond\_init(&chopstick[i],NULL);

pthread\_create(&ph[i],NULL,philosopher,(void \*)i);

sleep(1);

}

for(i=0;i<5;i++){

pthread\_join(ph[i],NULL);

}

pthread\_mutex\_destroy(&mutex);

for(i=0;i<5;i++){

pthread\_cond\_destroy(&chopstick[i]);

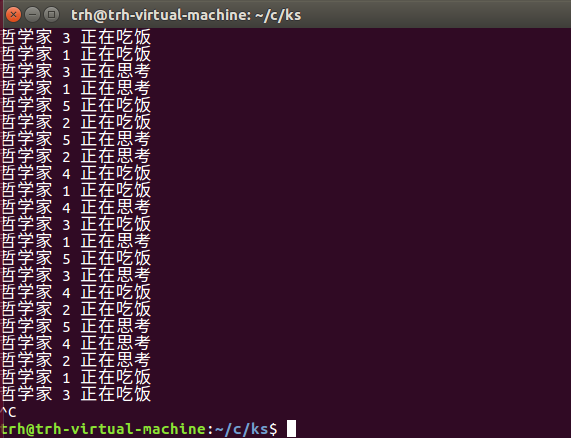
}

return 0;

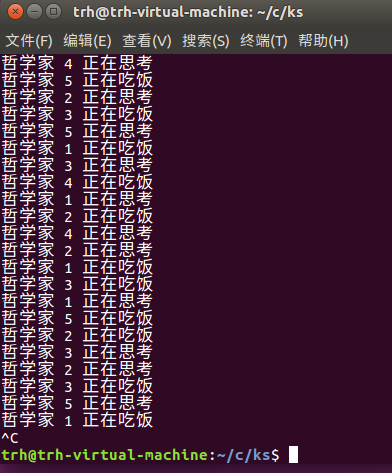
}

1. **运行与测试**

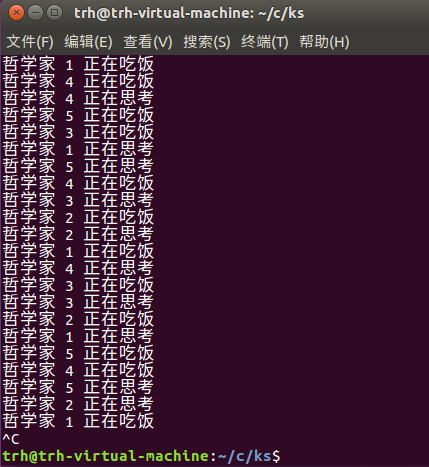
**法1截图：**



**法2截图：**



**法3截图：**



以上三种方法，在去掉sleep（）函数后运行十五分钟，可以发现线程不会自动停止，说明函数不会发生死锁。用ctrl+c软中断使其停止。

1. **心得体会**

循环创建线程容易出错，若将以下语句，

for(i=4;i>=0;i--){

pthread\_cond\_init(&chopstick[i],NULL);

pthread\_create(&ph[i],NULL,philosopher,(void \*)i);

sleep(2);

}

改成

for(i=4;i>=0;i--){

pthread\_cond\_init(&chopstick[i],NULL);

pthread\_create(&ph[i],NULL,philosopher,(void \*)&i);

sleep(2);

}

会发现传到子线程里的i值不正确。原因出在传过去的是i的地址，而父线程中for语句一直运行，将i的地址内的值不断进行+1操作。假设i的地址是0xaa，那么0xaa中的i值一直变化，且由于for语句是一个原语操作，不需要进行用户态的切换，0xaa中的i值变化的飞快，传入子线程内的i值已经不是原本想要传入的i值，因为此时i值进行好几个+1操作了。所以应该把(void \*)&i改为(void \*)i，进行值传递。注意，因为受到参数类型限制，值传递借助强制转换，此处强制转换不会造成数据丢失。我用的虚拟机是64位，int型变量是4字节，指针void\*是8字节，4字节的int\*传入8字节void\*不会发生数据丢失。

还要注意pthread\_cond\_wait(pthread\_cond\_t \*restrict cond, pthread\_mutex\_t \*restrict mutex) ;函数中，先执行的操作为阻塞等待条件变量cond和释放已掌握互斥锁（相当于执行int pthread\_mutex\_unlock(pthread\_mutex\_t \*mutex);所以应在执行wait前不能忘记先锁上mutex。这两个操作具有原子性，为一个原子操作，不可被打断；当cond条件变量被满足时，被唤醒，解除阻塞状态并重新获取mutex。