

华中科技大学

实 验 报 告

课 程：操作系统原理

实验序号：第 2 次实验

实验名称：线程实现“哲学家就餐”程序

院 系：软件学院

专业班级：

学 号：

姓 名：

2023 年 03 月 28 日

一、实验目的

- (1) 理解进程/线程的概念，会对进程/线程进行基本控制；
- (2) 理解进程/线程死锁的概念好原因，解决死锁的基本原理；
- (3) 理解进程/线程的典型同步机制和应用编程；
- (4) 掌握和推广国产操作系统（推荐银河麒麟或优麒麟）

二、实验内容

在 Linux 下模拟哲学家就餐，同时提供可能会带来死锁的解法和不可能死锁的解法。可能会带来死锁的解法是允许大家同时取筷子，且每次只能取一只筷子；完全不可能产生死锁的解法，例如：尝试拿取两只筷子，两只都能拿则拿，否则都不拿。为增强随机性，各状态间维持 100ms-500ms 内的随机时长。[可选]图形界面显示哲学家取筷，吃饭，放筷，思考等状态。

三、实验环境

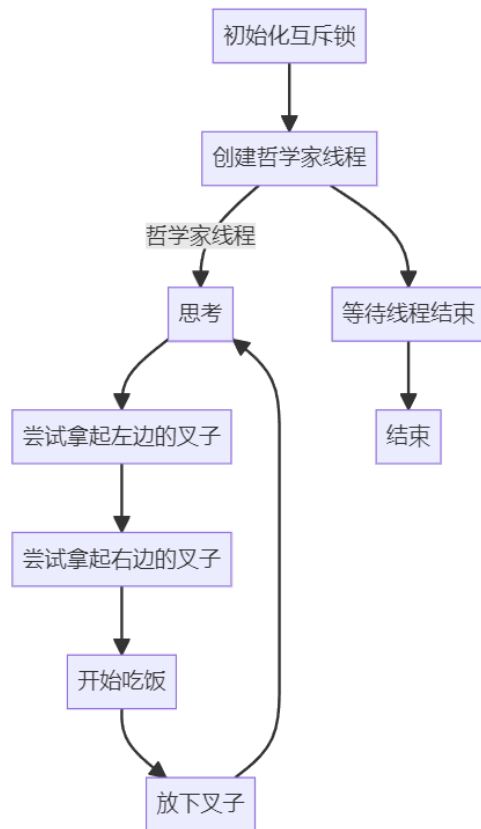
麒麟 Kylin(V10) + gcc/g++ + vi/vim/vscode/notepad++

四、程序的设计思路

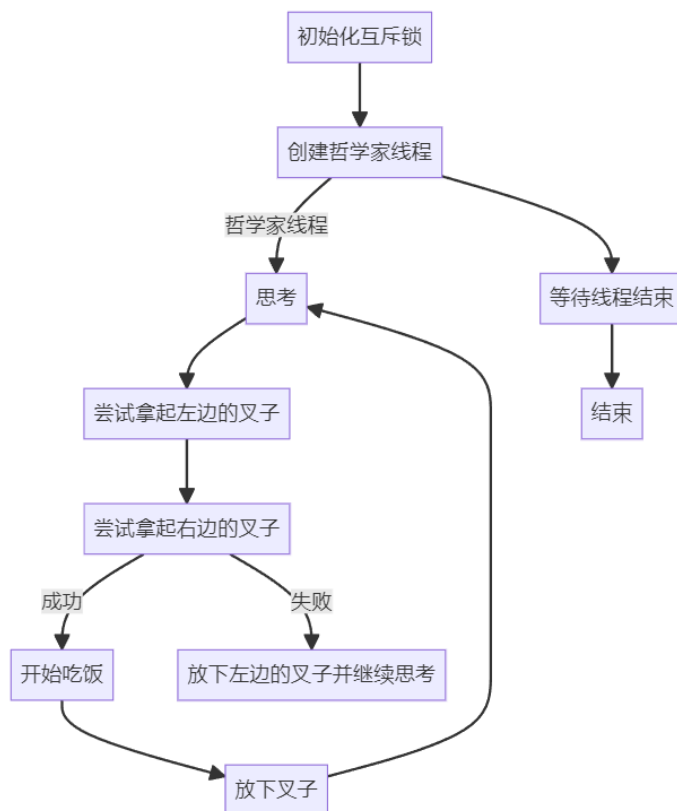
4.1 程序结构（或程序流程、或程序原理，任选一个或多个组合）

4.1.1 死锁解法：

如下列流程图所示：



4.1.2 非死锁解法:



4.2 关键函数或参数或机制

(一)、会死锁解法

1. `int pthread_mutex_lock(pthread_mutex_t *mutex);`

该函数用于对参数指定的互斥锁上锁。这个操作是阻塞调用的，也就是说，如果这个锁此时正在被其它线程占用，那么 `pthread_mutex_lock()` 调用会进入到这个锁的排队队列中，并会进入阻塞状态，直到该锁被解除之后才能继续执行。

上锁成功后返回 0，否则返回一个错误的提示码。函数返回时参数 `mutex` 指定的 `mutex` 对象变成锁住状态，同时该函数的调用线程成为该 `mutex` 对象的拥有者。

例如：

```
pthread_mutex_lock(&chop[i%n]);
printf("philosopher %d left chopstic %d\n",i,i%n);
pthread_mutex_lock(&chop[(i+1)%n]);
printf("philosopher %d right chopstic %d\n",i, (i+1)%n);
```

2. `int pthread_mutex_unlock(&mtx);`

该函数用于对参数指定的互斥锁解锁。在完成了对共享资源的访问后，要对互斥锁进行解锁，否则下一个想要获得这个锁的线程将会死等。

解锁成功后返回 0，否则返回一个错误的提示码。

例如：

```
pthread_mutex_unlock(&chop[i%n]);
pthread_mutex_unlock(&chop[(i+1)%n]);
```

（二）不会死锁解法

1. `int pthread_mutex_trylock(pthread_mutex_t *mutex);`

函数 `pthread_mutex_trylock` 是 `pthread_mutex_lock` 的非阻塞版本。如果 `mutex` 参数所指定的互斥锁已经被锁定的话，调用 `pthread_mutex_trylock` 函数不会阻塞当前线程，而是立即返回一个值来描述互斥锁的状况。

例如：

```
if(pthread_mutex_trylock(&chop[(i+1)%n])==EBUSY){  
    pthread_mutex_unlock(&chop[i%n]);  
    continue;  
}
```

2. `int pthread_mutex_lock(pthread_mutex_t *mutex);`

3. `int pthread_mutex_unlock(&mtx);`

同上。

五、关键代码分析

5.1 程序关键片段一：引入库和定义函数

这部分代码主要是对程序所需的库进行引用，并定义了一个函数 `philosopher`，该函数是哲学家（线程）的行为模式，包括思考、拿起叉子、吃饭和放下叉子。

```
#include<stdio.h>  
  
#include<pthread.h>  
  
#include<stdlib.h>  
  
#include<unistd.h>  
  
#define philosophers_count 5  
  
pthread_mutex_t forks[philosophers_count];
```

```
void *philosopher(void *arg) {  
  
    // ...  
  
}
```

5.2 程序关键片段二：主函数和线程创建

主函数中，首先定义了五个线程变量，然后初始化了五个互斥锁（叉子）。接着创建了五个哲学家线程，并等待这些线程执行完毕。

```
int main() {  
  
    pthread_t philosopher1, philosopher2, philosopher3, philosopher4, philosopher5;  
  
    // 初始化互斥锁  
    for (int i=0;i<philosophers_count;i++) {  
        if(pthread_mutex_init(&forks[i],NULL) != 0) {  
            printf("Failed to initialize mutex %d\n", i);  
            return 1;  
        }  
    }  
  
    int id1=0, id2=1, id3=2, id4=3, id5=4;  
  
    pthread_create(&philosopher1,NULL,philosopher,&id1);  
    pthread_create(&philosopher2,NULL,philosopher,&id2);  
    pthread_create(&philosopher3,NULL,philosopher,&id3);  
    pthread_create(&philosopher4,NULL,philosopher,&id4);  
    pthread_create(&philosopher5,NULL,philosopher,&id5);  
  
    pthread_join(philosopher1,NULL);  
    pthread_join(philosopher2,NULL);  
    pthread_join(philosopher3,NULL);  
    pthread_join(philosopher4,NULL);
```

```
pthread_join(philosopher5,NULL);  
  
return 0;  
  
}
```

5.3 程序关键片段三：哲学家线程函数（死锁解法）

在哲学家线程函数中，首先定义了哲学家的 ID，然后进入一个无限循环，模拟哲学家的行为。每个哲学家会先思考一段时间，然后尝试拿起左边和右边的叉子，吃饭，最后放下叉子。

```
void *philosopher(void *arg) {  
    int philosopher_id = *(int *)arg;  
    while(1) {  
        // 哲学家开始思考  
        usleep((rand()%400+100)*1000);  
        // 哲学家尝试拿起左边的叉子  
        pthread_mutex_lock(&forks[philosopher_id%philosophers_count]);  
        printf("Philosopher %d picked up left  
fork %d\n",philosopher_id,philosopher_id%philosophers_count);  
        // 哲学家尝试拿起右边的叉子  
        pthread_mutex_lock(&forks[(philosopher_id+1)%philosophers_count]);  
        printf("Philosopher %d picked up right fork %d\n",philosopher_id,  
(philosopher_id+1)%philosophers_count);  
        // 哲学家开始吃饭  
        usleep((rand()%400+100)*1000);  
        // 哲学家放下叉子  
        pthread_mutex_unlock(&forks[philosopher_id%philosophers_count]);  
        pthread_mutex_unlock(&forks[(philosopher_id+1)%philosophers_count]);  
        printf("Philosopher %d is full\n",philosopher_id);  
    }  
}
```

5.4 程序关键片段四：哲学家线程函数（非死锁解法）

与死锁解法不同的是，只有哲学家在能够同时拿起左右两只筷子的时候才会选择拿起，否则哲学家不会拿起筷子。

```
// 哲学家尝试拿起左边的叉子

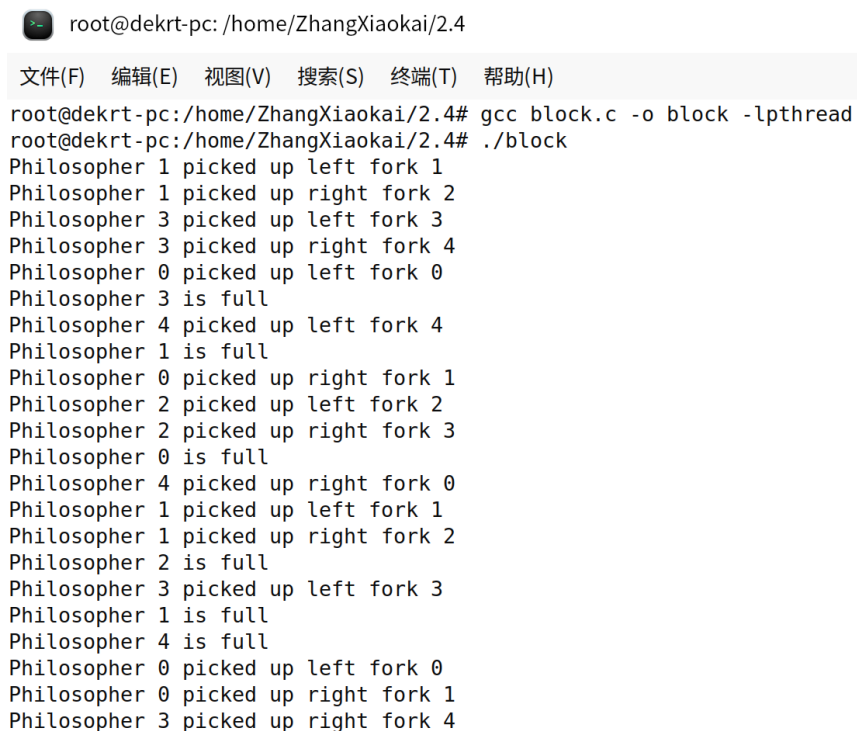
pthread_mutex_lock(&forks[philosopher_id%philosophers_count]);

// 如果右边的叉子被其他哲学家拿起，那么放下左边的叉子并继续思考
if (pthread_mutex_trylock(&forks[(philosopher_id+1)%philosophers_count]) == EBUSY) {
    pthread_mutex_unlock(&forks[philosopher_id%philosophers_count]);
    continue;
}

printf("Philosopher %d picked up left\n", philosopher_id, philosopher_id%philosophers_count);

printf("Philosopher %d picked up right fork %d\n", philosopher_id,
(philosopher_id+1)%philosophers_count);
```

六、程序运行结果和分析



```
root@dekart-pc: /home/ZhangXiaokai/2.4
文件(F) 编辑(E) 视图(V) 搜索(S) 终端(T) 帮助(H)
root@dekart-pc:/home/ZhangXiaokai/2.4# gcc block.c -o block -lpthread
root@dekart-pc:/home/ZhangXiaokai/2.4# ./block
Philosopher 1 picked up left fork 1
Philosopher 1 picked up right fork 2
Philosopher 3 picked up left fork 3
Philosopher 3 picked up right fork 4
Philosopher 0 picked up left fork 0
Philosopher 3 is full
Philosopher 4 picked up left fork 4
Philosopher 1 is full
Philosopher 0 picked up right fork 1
Philosopher 2 picked up left fork 2
Philosopher 2 picked up right fork 3
Philosopher 0 is full
Philosopher 4 picked up right fork 0
Philosopher 1 picked up left fork 1
Philosopher 1 picked up right fork 2
Philosopher 2 is full
Philosopher 3 picked up left fork 3
Philosopher 1 is full
Philosopher 4 is full
Philosopher 0 picked up left fork 0
Philosopher 0 picked up right fork 1
Philosopher 3 picked up right fork 4
```


当运行死锁解法的程序时，程序运行一段时间之后便会产生死锁，不再产生新的输出。

```
root@dekr-t-pc: /home/ZhangXiaokai/2.4
文件(F) 编辑(E) 视图(V) 搜索(S) 终端(T) 帮助(H)
root@dekr-t-pc:/home/ZhangXiaokai/2.4# gcc non_block.c -o non_block -lpthread
root@dekr-t-pc:/home/ZhangXiaokai/2.4# ./non_block
Philosopher 1 picked up left fork 1
Philosopher 1 picked up right fork 2
Philosopher 3 picked up left fork 3
Philosopher 3 picked up right fork 4
Philosopher 3 is full
Philosopher 4 picked up left fork 4
Philosopher 4 picked up right fork 0
Philosopher 1 is full
Philosopher 2 picked up left fork 2
Philosopher 2 picked up right fork 3
Philosopher 4 is full
Philosopher 0 picked up left fork 0
Philosopher 0 picked up right fork 1
Philosopher 2 is full
Philosopher 3 picked up left fork 3
Philosopher 3 picked up right fork 4
Philosopher 0 is full
Philosopher 1 picked up left fork 1
Philosopher 1 picked up right fork 2
Philosopher 1 is full
Philosopher 0 picked up left fork 0
Philosopher 0 picked up right fork 1
```

当采用非死锁解法时，两个线程正确地运行，并且正确地实现了哲学家就餐的同步关系。

七、实验错误排查和解决方法

7.1 gcc 过程显示 wait 函数出问题

没有添加<sys/wait.h>头文件

7.2 gcc 过程中显示 pthread 不成功

没有加-lpthread;

八、实验参考资料和网址

(1) 教学课件

(2) 参考资料: <https://www.geeksforgeeks.org/dining-philosophers-problem-using-semaphores/>

(3) 网址: <https://github.com/Abdurraheem/Dining-Philosophers-Problem>