

### 

**实验报告**

**课程名称:** **实用操作系统教程**

**设计课题: 进程同步与互斥之哲学家进餐问题**

**指导教师: 熊婷**

**专 业：** 计算机科学与技术

**班 级：** 1982066班

**学 号：** 198206606

**姓 名：** 陈文龙

**成 员：** 陈文龙

**二O 二 一 年 12 月 4 日**

**目录**

[1.实验目的 - 2 -](#_Toc90184491)

[2.实验内容和要求 - 2 -](#_Toc90184492)

[3.实验原理 - 2 -](#_Toc90184493)

[4.系统分析 - 3 -](#_Toc90184494)

[5.具体实现 - 3 -](#_Toc90184495)

[**5.1方法一** - 3 -](#_Toc90184496)

[**5.2方法二** - 4 -](#_Toc90184497)

[**5.3方法三** - 5 -](#_Toc90184498)

[6.代码及结果 - 5 -](#_Toc90184499)

[**结果** - 9 -](#_Toc90184500)

[7.总结 - 9 -](#_Toc90184501)

[参考文献： - 9 -](#_Toc90184502)

# 1.实验目的

利用信号量和PV操作来实现进程同步与互斥，让学生掌握进程同步与互斥的原理，以及解决进程同步与互斥的算法，从而进一步巩固进程同步与互斥等相关的内容。

# 2.实验内容和要求

五个哲学家围坐在一圆桌旁，桌中央有一盘通心面，没人面前有一个空盘子，每两人之间放一把叉子。每个哲学家思考、饥饿、然后吃通心面。

要求如下：

1. 理解并掌握进程同步与互斥的原理；
2. 了解进程同步与互斥过程中的基本概念：原语、信号量、临界区以及临界资源等。
3. 了解信号量的分类。

# 3.实验原理

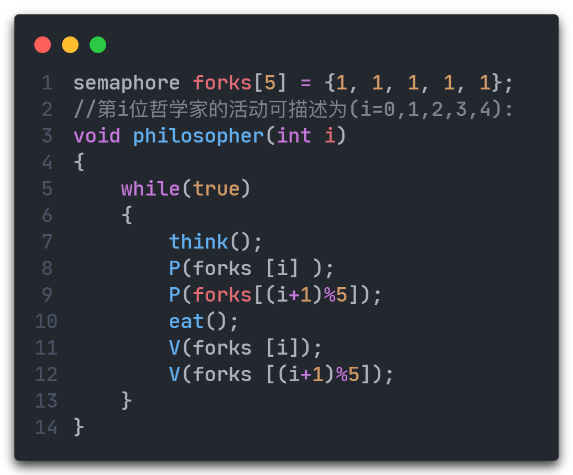
五个哲学家围坐在一圆桌旁，桌中央有一盘通心面，没人面前有一个空盘子，每两人之间放一把叉子。每个哲学家思考、饥饿、然后吃通心面。哲学家之间应遵循以下原则：

1. 每个哲学家必须获得两把叉子才能吃面，且每人只能从自己左边或右边去取叉子；
2. 至多只能允许四个哲学家同时吃面；
3. 每个哲学家必须取得手边的两把叉子才能吃面，否则一把叉子也不取。

# 4.系统分析

5个位哲学家共用一张圆桌，分别坐在周围的五张椅子上，在圆桌上有5个盘子和5把叉子，他们的生活方式是交替的进行思考和进餐。平时哲学家进行思考，饥饿时便试图取用其左右最靠近他的叉子，只有拿到两只叉子时才能进餐。进餐完成之后，放下叉子继续思考。放在圆桌上的叉子是临界资源，必须互斥使用，每位哲学家必须获得两个叉子后才能进餐。

为实现叉子的互斥使用，为5个叉子分别设置一个信号量，初值为1。为简化信号量的定义，可把这5个信号量定义成信号量数组。



# 5.具体实现

## **5.1方法一**

至多4个哲学家同时进餐，保证至少有一个人可以进餐，最终总会释放他所使用的2个叉子使更多的人进餐。初始化叉子的信号量，同时增加信号量room初始值为4，保证最多四个哲学家进餐。申请不到叉子的哲学家进入信号量room的阻塞等待序列。依据先进先出的原则，先申请的哲学家先进餐，吃完后释放叉子，就不会出现某个哲学家饿死的现象。



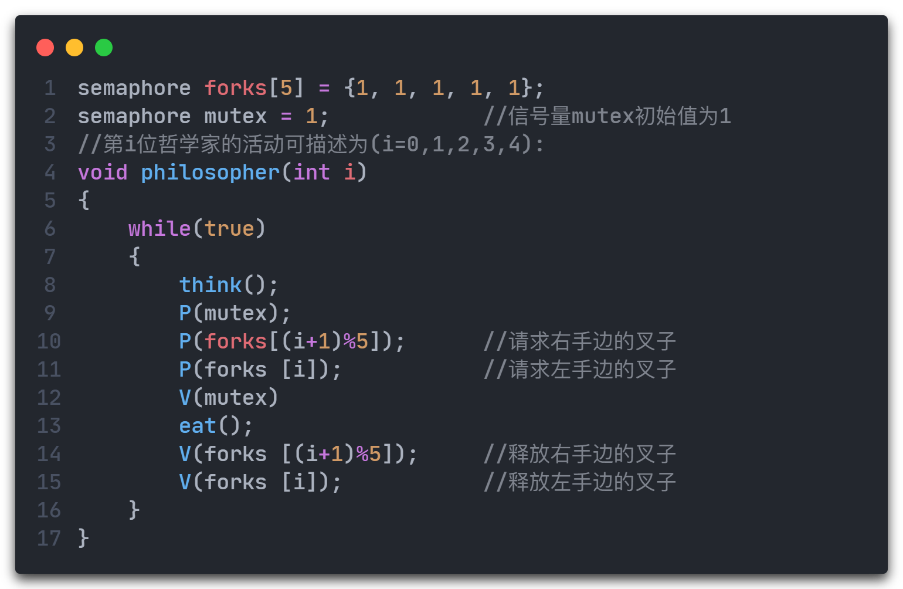
## **5.2方法二**

规定奇数号哲学家先拿起其左边的叉子，再拿右边的，偶数号哲学家则相反。总会有一人拿到两把叉子进餐。申请不到叉子的哲学家进入阻塞等待状态，当其他哲学家进程执行完毕，释放叉子时，会被唤醒，因此不会出现饿死的哲学家进程。



## **5.3方法三**

仅当哲学家的左右两支筷子均可用时，才允许他拿起筷子进餐。只有一个叉子时，哲学家不占有该叉子。利用信号量保护机制实现。通过新添加的信号量mutex,其初始值为1，对eat()之前的取左侧和右侧叉子的操作进行保护，使之成为一个原子操作，这样可以防止死锁的出现。



# 6.代码及结果

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <stdint.h>

#include <stdbool.h>

#include <errno.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <sys/ipc.h>

#include <sys/sem.h>

#include <sys/wait.h>

union semuns{

    int val;

    struct semid\_ds \*buf;

    unsigned short \*array;

    struct seminfo \*\_\_buf;

};

#define ERR\_EXIT(m) \

    do { \

        perror(m); \

        exit(EXIT\_FAILURE); \

    } while(0)

//申请一个资源

int    wait\_1fork(int no,int semid){

    //资源减1

    struct sembuf sb = {no,-1,0};

    int ret;

    ret = semop(semid,&sb,1);

    if(ret < 0) {

        ERR\_EXIT("semop");

    }

    return ret;

}

// 释放一个资源

int free\_1fork(int no,int semid){

    //资源加1

    struct sembuf sb = {no,1,0};

    int ret;

    ret = semop(semid,&sb,1);

    if(ret < 0) {

        ERR\_EXIT("semop");

    }

    return ret;

}

//叉子是一个临界资源

#define DELAY (rand() % 5 + 1)

//相当于P操作

//第一个参数是叉子编号

//第二个参数是信号量编号

void wait\_for\_2fork(int no,int semid){

    //哲学家左边的叉子编号和哲学家编号是一样的

    int left = no;

    //右边的叉子

    int right = (no + 1) % 5;

    //叉子值是两个

    //操作的是两个信号量,即两种资源都满足,才进行操作

    struct sembuf buf[2] = {

        {left,-1,0},

        {right,-1,0}

    };

    //信号集中有5个信号量，只是对其中的资源sembuf进行操作

    semop(semid,buf,2);

}

//相当于V操作  ,释放叉子

void free\_2fork(int no,int semid){

    int left = no;

    int right = (no + 1) % 5;

    struct sembuf buf[2] = {

        {left,1,0},

        {right,1,0}

    };

    semop(semid,buf,2);

}

//哲学家要做的事

void philosophere(int no,int semid){

    srand(getpid());

    for(;;)

    {

    #if 1

        //当两只叉子都可用的时候，哲学家才能进餐

        printf("the philosopher of %d is thinking\n",no);  //思考中

        sleep(DELAY);

        printf("the philosopher of %d is hungry\n",no);    //饥饿

        wait\_for\_2fork(no,semid); //拿到两只叉子才能吃饭

        printf("the philosopher of %d is eating\n",no);    //进餐

        sleep(DELAY);

        free\_2fork(no,semid); //释放两只叉子

    #else

        //可能会造成死锁

        int left = no;

        int right = (no + 1) % 5;

        printf("the philosopher of %d is thinking\n",no);  //思考中

        sleep(DELAY);

        printf("the philosopher of %d is hungry\n",no);    //饥饿

        wait\_1fork(left,semid);         //拿起左叉子（只要有一个资源就申请）

        sleep(DELAY);

        wait\_1fork(right,semid);        //拿起右叉子

        printf("the philosopher of %d is eating\n",no);    //进餐

        sleep(DELAY);

        free\_1fork(left,semid);         //释放左叉子

        free\_1fork(right,semid);          //释放右叉子

    #endif

    }

}

int main(int argc,char \*argv[]){

    int semid;

    //创建信号量集，其中有5个信号量

    semid = semget(IPC\_PRIVATE,5,IPC\_CREAT | 0666);

    if(semid < 0) {

        ERR\_EXIT("semid");

    }

    union semun su;

    su.val = 1;

    int i;

    for(i = 0;i < 5; ++i) {

        //第二个参数也是索引

        semctl(semid,i,SETVAL,su);

    }

    //创建4个子进程

    int num = 0;

    pid\_t pid;

    for(i = 1;i < 5;++i)

    {

       pid = fork();

       if(pid < 0)

        {

           ERR\_EXIT("fork");

        }

        if(0 == pid)  //子进程

        {

            num = i;

            break;

        }

    }

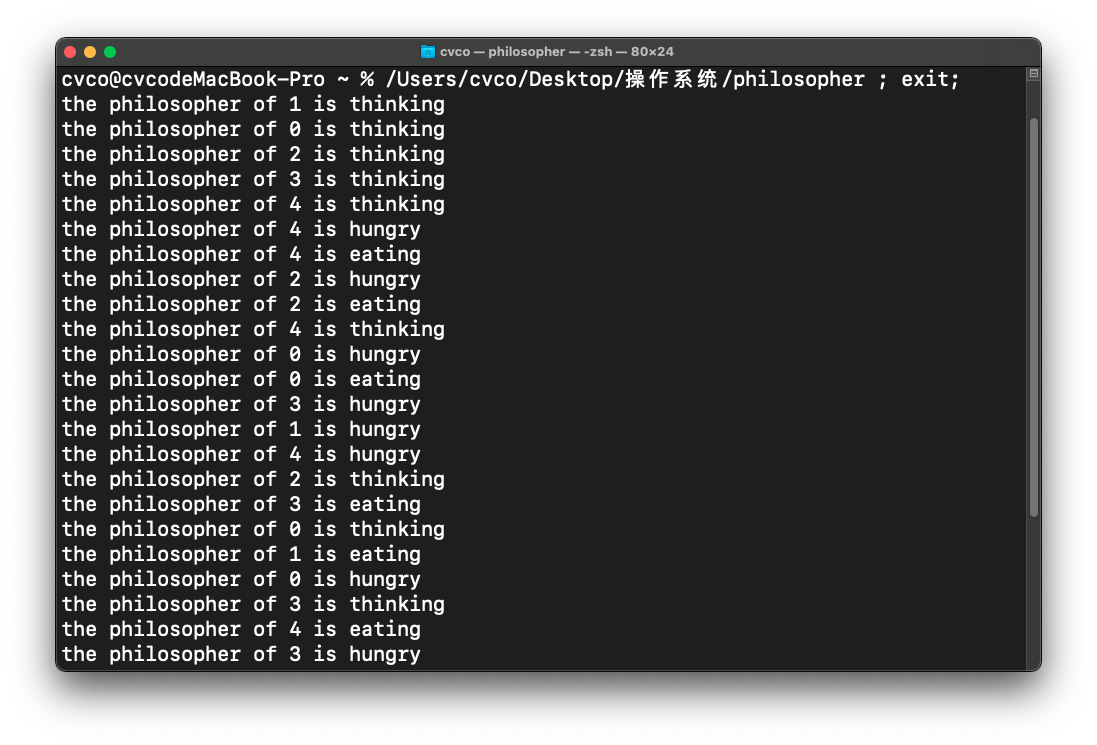
    //哲学家要做的事情

   philosophere(num,semid);

    return 0;

}

## **结果**

****

# 7.总结

本次实验是关于哲学家进餐互斥和同步的问题。问题的实质是如何避免饥饿，也就是死锁，实验一共给出了三种解决方案，分别是设置room，至多四人进餐；设置mutex，当左右筷子都被哲学家拿到方可进餐；规定奇数号和偶数号哲学家的进餐规则。

通过本次实验，我对操作系统的P,V操作和死锁有了进一步的认识，深入地了解P,V操作的实质和避免死锁的重要性，也通过课本的理论知识进一步阐述了现实的实际问题。

参考文献：

［1］李建伟主编. 实用操作系统教程[M]. 北京：清华大学出版社，2016