电子科技大学信息与软件工程学院

**实 验 报 告**

学 号 2018091602007

姓 名 王乐卿

（实验） 课程名称 操作系统基础

理论教师 刘瑶

实验教师 刘瑶

**电子科技大学教务处制表**

**电 子 科 技 大 学**

**实 验 报 告**

**学生姓名：王乐卿 学号：2018091602007 指导教师：刘瑶**

**实验地点：家 实验时间：2020.06.02**

**一、实验名称：信号量经典问题的实现**

**二、实验学时：4**

**三、实验目的：**

本实验分为两部分，首先实现哲学家就餐问题，要求不能出现死锁。通过本实验熟悉Linux系统的基本环境，了解Linux下进程和线程的实现。然后实现生产者/消费者问题，通过本实验掌握进程间的同步和互斥机制的使用。熟悉基于某操作系统进程和线程的编程。

1. 掌握进程、线程的概念，熟悉相关的控制原语。
2. 掌握进程、线程间的同步原理和方法。
3. 掌握进程、线程间的互斥原理和方法。
4. 掌握使用信号量原语解决进程、线程间互斥和同步方法。

**四、实验原理：**

由Dijkstra提出并解决的哲学家进餐问题(The Dinning Philosophers Problem)是典型的同步问题。该问题是描述有五个哲学家共用一张圆桌，分别坐在周围的五张椅子上，在圆桌上有五个碗和五只筷子，他们的生活方式是交替地进行思考和进餐。平时，一个哲学家进行思考，饥饿时便试图取用其左右最靠近他的筷子，只有在他拿到两只筷子时才能进餐。进餐完毕，放下筷子继续思考。

第二个问题考虑n个缓冲区的缓冲池作为一个共享资源，当生产者进程从数据源—文件中读取数据后将会申请一个缓冲区，并将此数据放入缓冲区中。消费者从一个缓冲区中取走数据，并将其中的内容打印输出。当生产者进程正在访问缓冲区时，消费者进程不能同时访问缓冲区，因此缓冲区是个互斥资源。

**五、实验内容：**

（一）熟悉Ubuntu系统环境和命令

（二） 熟悉Ubuntu系统下的多线程编程。

1. 使用“Ctrl+Alt+T”打开终端；

2. 使用gedit或vim命令打开文本编辑器进行编码：“gedit 文件名.c”

3. 编译程序：

“gcc 文件名.c -o 可执行程序名”（如果只输入 “gcc 文件名.c”，默认可执行程序名为a.out），使用线程库时，gcc编译需要添加-lpthread

4. 执行程序：./可执行程序名

（三）实现哲学家就餐问题

实现教材2.5.2节中所描述的哲学家就餐问题。要求显示出5个哲学家的工作状态，如吃饭，思考。连续运行30次以上都未出现死锁现象。

（四）实现生产者消费者问题

1） 一群生产者进程在生产产品，并将这些产品提供给消费者进程去消费。为使生产者进程与消费者进程能并发执行，在两者之间设置了一个具有n个缓冲区的缓冲池：生产者进程从文件中读取一个数据，并将它存放到一个缓冲区中； 消费者进程从一个缓冲区中取走数据，并输出此数据。生产者和消费者之间必须保持同步原则：不允许消费者进程到一个空缓冲区去取产品；也不允许生产者进程向一个已装满产品且尚未被取走的缓冲区中投放产品。

2） 创建3进程（或者线程）作为生产者，4个进程（或者线程）作为消费者。创建一个文件作为数据源，文件中事先写入一些内容作为数据。

3）生产者和消费者进程（或者线程）都具有相同的优先级。

步骤1：分配具有n个缓冲区的缓冲池，作为共享资源。

步骤2：定义两个资源型信号量empty 和full，empty信号量表示当前空的缓冲区数量，full表示当前满的缓冲区数量。

步骤3：定义互斥信号量mutex，当某个进程访问缓冲区之前先获取此信号量，在对缓冲区的操作完成后再释放此互斥信号量。以此实现多个进程对共享资源的互斥访问。

步骤4：创建3进程（或者线程）作为生产者，4个进程（或者线程）作为消费者。创建一个文件作为数据源，文件中事先写入一些内容作为内容。

步骤5 ：编写代码实现生产者进程的工作内容，即从文件中读取数据，然后申请一个empty信号量和互斥信号量，然后进入临界区操作将读取的数据放入此缓冲区中。并释放empty信号量和互斥信号量。

步骤6：编写代码实现消费者者进程的工作内容，即先申请一个full信号量，和互斥信号量，然后进入临界区操作从缓冲区中读取数据并打印输出。

**六、实验器材（设备、元器件）：**

1. 学生每人一台PC，安装WindowsXP/2000操作系统。
2. 局域网络环境。
3. 个人PC安装VMware虚拟机和Ubuntu系统。

**七、实验步骤：**

（一）实验1哲学家就餐问题得实现

1）算法思想

将筷子编号为0-4，哲学家编号为0-4，每个哲学家左手的筷子编号等于哲学家编号，右手筷子编号等于（左手筷子编号+1）%5，使用互斥量mutex锁定筷子，当拿起筷子时，就锁定筷子，若是无法同时拿到两双筷子，则放下第一双筷子，继续思考，否则进入吃饭状态。

2）流程图

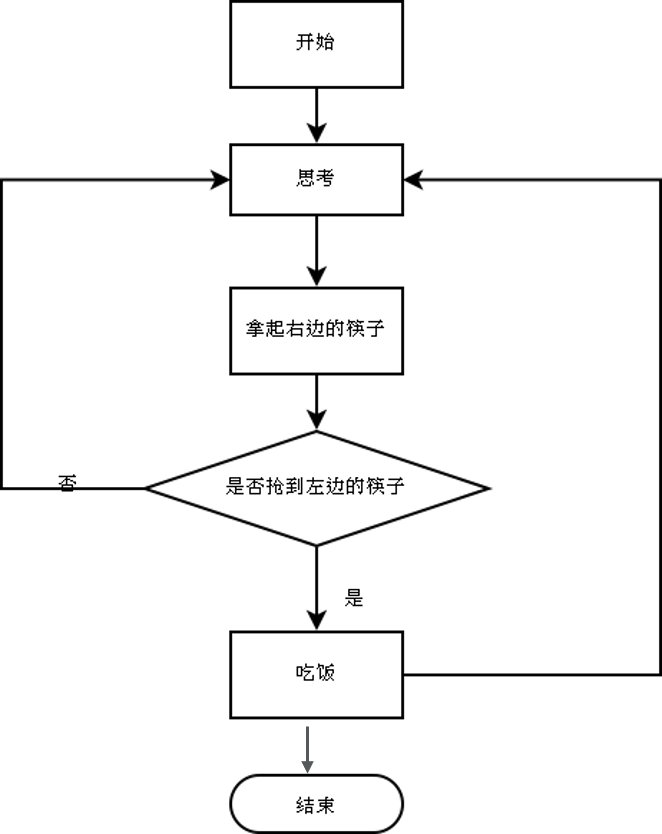


图 1哲学家就餐算法流程图

3）程序代码

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <pthread.h>//Linux系统下的多线程遵循POSIX线程接口，称为pthread  #include <errno.h>  #include <unistd.h> //unistd.h是unix std的意思，是POSIX标准定义的unix类系统定义符号常量的头文件  #include <string.h>  pthread\_mutex\_t chopstick[5] ; //筷子作为mutex  pthread\_t philosophers[5]; //5个哲学家  int philosophersNum[5] = {0, 1, 2, 3, 4};//代表5个哲学家的编号  void \*eat\_think(void \*arg)  {  int philoNum = \*(int \*)arg;  int left=philoNum; //左筷子的编号和哲学家编号相同  int right=(philoNum+1)%5 ;//右筷子的编号等于（哲学家编号+1）%5  while(1){  printf("哲学家%d正在思考问题\n",philoNum);  usleep(3); //思考    printf("哲学家%d准备吃东西\n",philoNum);    pthread\_mutex\_lock(&chopstick[left]); //拿起左手的筷子  printf("哲学家%d拿起筷子%d\n", philoNum, left);  if (pthread\_mutex\_trylock(&chopstick[right]) == EBUSY){ //拿起右手的筷子  pthread\_mutex\_unlock(&chopstick[left]); //如果右边筷子被拿走放下左手的筷子  printf("哲学家%d放下筷子%d\n", philoNum, left);  continue;  }  printf("哲学家%d拿起筷子%d\n", philoNum, right);  printf("哲学家%d正在吃东西\n",philoNum);  usleep(3); //吃饭  pthread\_mutex\_unlock(&chopstick[left]); //放下左手的筷子  printf("哲学家%d放下筷子%d\n", philoNum, left);  pthread\_mutex\_unlock(&chopstick[right]); //放下右手的筷子  printf("哲学家%d放下筷子%d\n", philoNum, right);  }    }  int main(){  //信号量初始化  int i;  for (i = 0; i < 5; i++)  pthread\_mutex\_init(&chopstick[i],NULL);  //创建线程  for(i=0;i<5;i++)  pthread\_create(&philosophers[i],NULL, eat\_think, &philosophersNum[i]);  //挂起线程  for(i=0;i<5;i++)  pthread\_join(philosophers[i],NULL);    //销毁互斥量  for(i=0;i<5;i++)  pthread\_mutex\_destroy(&chopstick[5]);  return 0;  } |

代码 1哲学家就餐代码

（二）实验2生产者消费者问题的实现

1）算法思想

a. 分配具有n个缓冲区的缓冲池，作为共享资源。

b. 定义两个资源型信号量empty 和full，empty信号量表示当前空的缓冲区数量，full表示当前满的缓冲区数量。

c. 定义互斥信号量mutex，当某个进程访问缓冲区之前先获取此信号量，在对缓冲区的操作完成后再释放此互斥信号量。以此实现多个进程对共享资源的互斥访问。

d. 创建3进程（或者线程）作为生产者，4个进程（或者线程）作为消费者。创建一个文件data.txt作为数据源，文件中事先写入一些内容作为内容。

e. 编写代码实现生产者进程的工作内容，即从文件中读取数据，然后申请一个empty信号量，和互斥信号量，然后进入临界区操作将读取的数据放入此缓冲区中。并释放empty信号量和互斥信号量。

f. 编写代码实现消费者者进程的工作内容，即先申请一个full信号量，和互斥信号量，然后进入临界区操作从缓冲区中读取数据并打印输出。

2）流程图

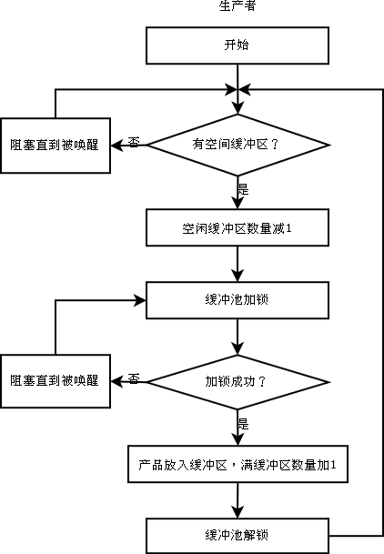


图 2生产者流程图

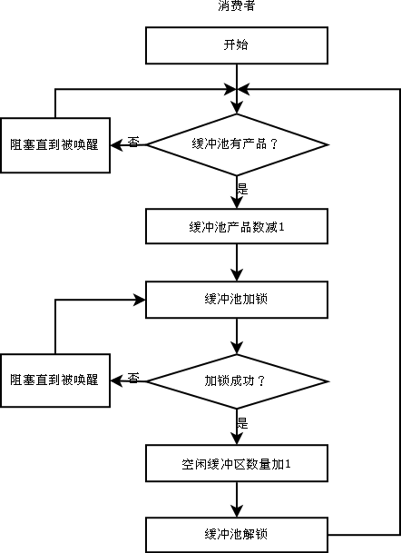


图 3消费者流程图

3）程序代码

|  |
| --- |
| #include <stdio.h>  #include <semaphore.h>  #include <stdlib.h>  #include <pthread.h>  #include <unistd.h>  #define N\_pro 3  #define N\_con 4  #define M 10  //定义缓存区入区位置和出区位置  int in = 0;  int out = 0;  //定义缓存区  int buff[M] = {0};  //定义两个资源型信号量empty 和full，empty信号量表示当前空的缓冲区数量，full表示当前满的缓冲区数量。  sem\_t empty;  sem\_t full;  //定义互斥信号量mutex  pthread\_mutex\_t mutex;  int data;  FILE \*fp;  void \* product(void \*arg)  {  int producer\_id = \*(int \*)arg;  while(1)  {  //模拟生产  sleep(1);    //等待资源量，将empty-1，再与0比较大小，判断缓冲池是否有空  sem\_wait(&empty);    //等待互斥量  pthread\_mutex\_lock(&mutex);    //从文件中取出内容  if(fscanf(fp, "%d", &data)==EOF)  {  fseek(fp, 0, SEEK\_SET);//把文件指针fp指向文件的开头  fscanf(fp, "%d", &data);  }    in = in % M;  buff[in] = data;    printf("生产者%d 在%d号缓冲区 放入产品%d\n", producer\_id, in, buff[in]);  in++; //存入成功，入区位置+1    pthread\_mutex\_unlock(&mutex); //生产者停止使用缓存区  sem\_post(&full); //full资源变量+1，表明缓存区中内容+1  }  }  void \*consume(void \*arg)  {  int consumer\_id = \*(int \*)arg;  while(1)  {  //模拟消费  sleep(1);  sem\_wait(&full);  pthread\_mutex\_lock(&mutex);  out = out % M;  printf("消费者%d 从%d号缓冲区 拿出产品%d\n", consumer\_id, out, buff[out]);  buff[out] = 0; //取出后将原位置为0  out++;  pthread\_mutex\_unlock(&mutex);  sem\_post(&empty);  }  }  int main()  {    pthread\_t producer[N\_pro]; //生产者线程  pthread\_t consumer[N\_con]; //消费者线程  int i;  int ret1[N\_pro];  int ret2[N\_con];  //初始化信号量  int ini1 = sem\_init(&empty, 0, M);  int ini2 = sem\_init(&full, 0, 0);  if(ini1 || ini2 != 0)  {  printf("Sem初始化失败\n");  exit(1);  }  //初始化互斥量  int ini3 = pthread\_mutex\_init(&mutex, NULL);  if(ini3 != 0 )  {  printf("mutex初始化失败\n");  exit(1);  }      //以“只读”方式打开当前目录下的 data.txt 文件，并使文件指针fp 指向该文件  fp = fopen("./data.txt", "r");  if(fp == NULL)  {  printf("文件打开失败\n");  exit(1);  }  //创建生产者线程  for(i = 0; i < N\_pro; i++)  {  int\* id = malloc(sizeof(int)); //每次循坏为id分配一个地址空间，以免主线程切换后子线程还没有运行，造成线程名对应错误  \*id=i;  ret1[i] = pthread\_create(&producer[i], NULL, product, (void\*)(id)); //传入生产者id  if(ret1[i] != 0)  {  printf("生产者线程%d创建失败\n", i);  exit(1);  }  }  //创建消费者线程  for(i = 0; i< N\_con; i++)  {  int\* id = malloc(sizeof(int));  \*id=i;  ret2[i] = pthread\_create(&consumer[i], NULL, consume,(void\*)(id)); //传入消费者id  if(ret2[i] != 0)  {  printf("消费者线程%d创建失败\n", i);  exit(1);  }  }  //挂起线程  for(i = 0; i < N\_pro; i++)  pthread\_join(producer[i], NULL);  for(i = 0; i < N\_con; i++)  pthread\_join(consumer[i], NULL);  //销毁互斥量  pthread\_mutex\_destroy(&mutex);    //销毁信号量  sem\_destroy(&empty);  sem\_destroy(&full);    return 0;  } |

**八、实验结果与分析（含重要数据结果分析或核心代码流程分析）**

（一）实验1运行结果

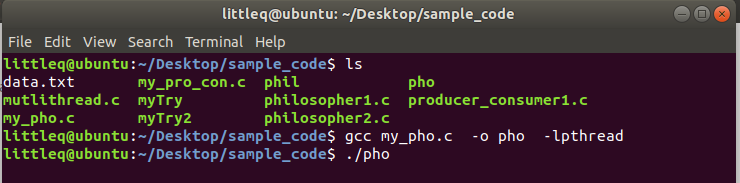


图 4运行命令

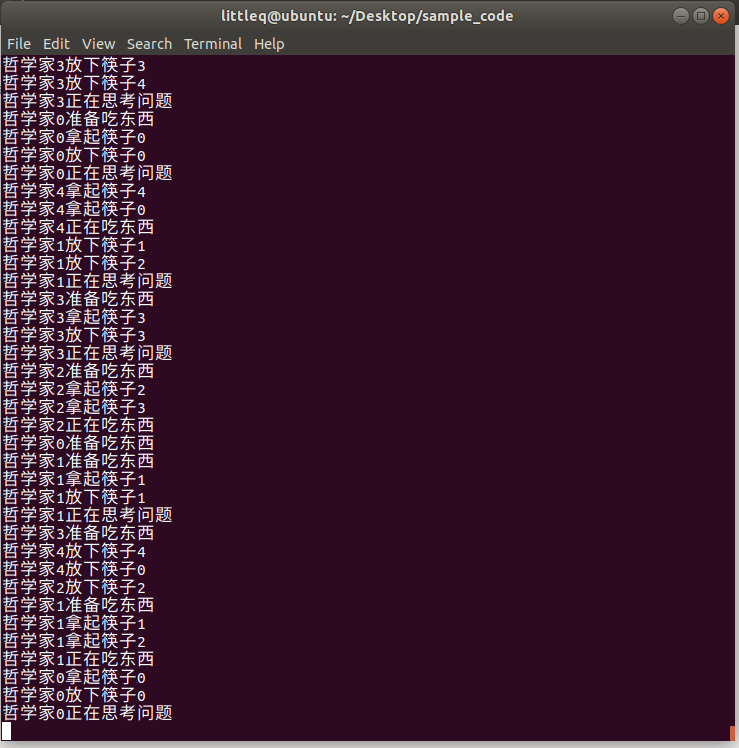


图 5 哲学家就餐运行结果

（二）实验二运行结果

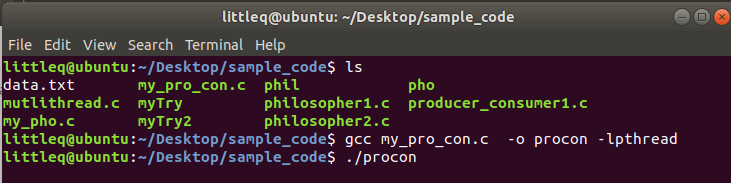


图 6运行命令

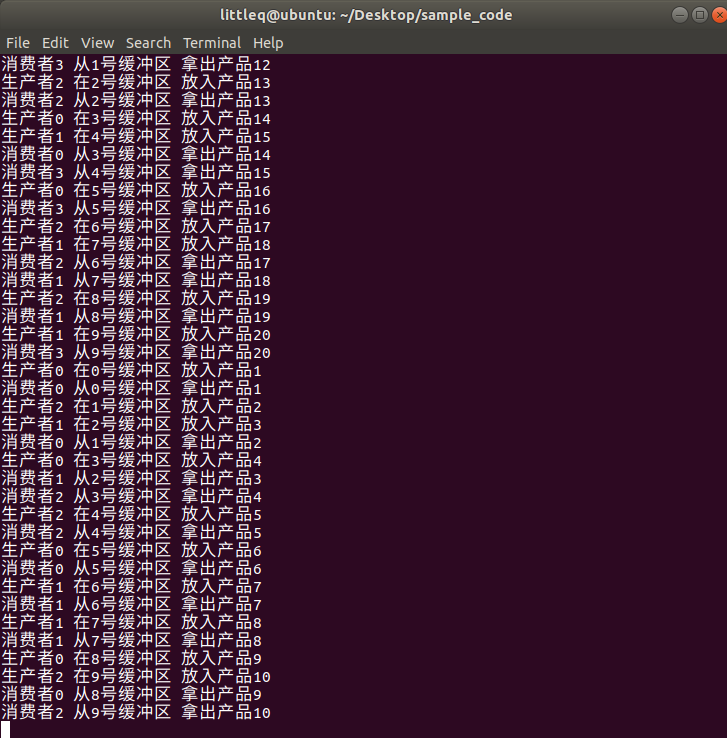


图 7生产者消费者运行结果图

**九、总结及心得体会：**

通过本次实验我更加深入的理解了进程、线程的概念及其相关的控制原语，同时也掌握了进程、线程间的同步原理和方法，进程、线程间的互斥原理和方法和使用信号量原语解决进程、线程间互斥和同步方法。

在实验过程中，我也遇到了很多的困难，比如指针问题，在传入子线程编号i时，主线程已经改变了是下一个i了 子线程还没取的问题，不过在查阅资料和咨询助教老师后都得到了解决。

经过本次实验，我更加深刻的认识到了学以致用的重要性，理论的学习记忆很容易忘却，而真正的实操之后，记忆会更加深刻，同时理解也能更加深入。

**十、对本实验过程及方法、手段的改进建议：**

建议加入一些linux的前导实验，以衔接更为平滑。

**报告评分：**

**指导教师签字：**