电子科技大学信息与软件工程学院

**实 验 报 告**

学 号 2017221302006

姓 名 周玉川

（实验） 课程名称 操作系统基础

理论教师 刘瑶

实验教师 文淑华

**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：周玉川 学号：2017221302006 指导教师：刘瑶**

**实验地点：信软楼西305 实验时间：2019.04.22**

**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：**

（1）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。

（2）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：**

1）熟悉Ubuntu系统环境和命令；

2）熟悉Ubuntu系统下的多线程编程；

2）在Ubuntu系统下编程实现哲学家就餐问题。

**第一步：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。要求显示出每个哲学家的工作状态，如就餐，思考。连续运行30次以上都未出现死锁现象。

**第二步：实现生产者/消费者问题**

1）有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。

**六、实验器材（设备、元器件）：**

1. 学生每人一台PC，安装Windows10操作系统。
2. 个人PC安装VMware虚拟机和Ubuntu系统。

**七、实验步骤：**

（一） 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

（二）信号量同步问题实现

步骤1：定义1个资源型信号量数组chopsticks，chopsticks信号量表示当前筷子的状态，是否被取用。

步骤2：创建5个线程作为哲学家，并让他们请求筷子资源，就餐并思考。

步骤3 ：编写代码实现哲学家对筷子资源的请求，并实现状态打印与死锁规避。

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。

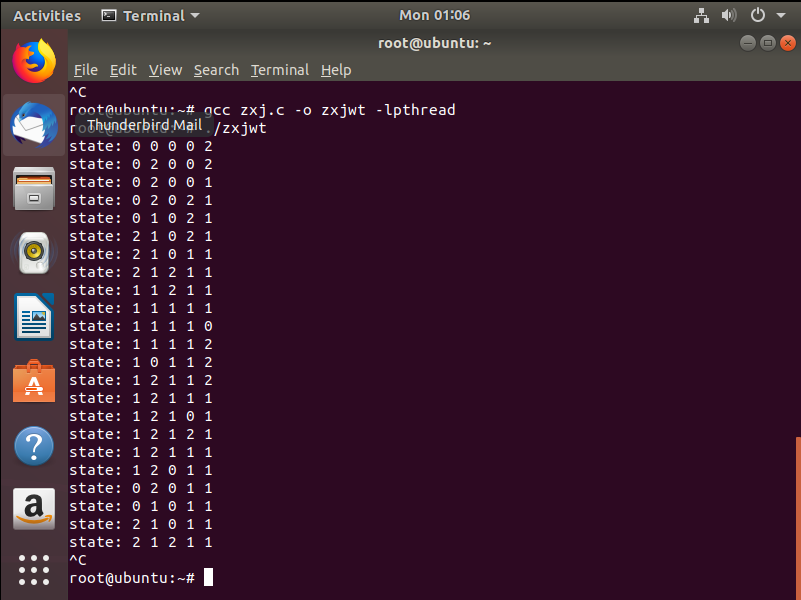
**八、实验结果与分析（含重要数据结果分析或核心代码流程分析）**

（1）哲学家就餐实验：

哲学家就餐问题代码：

|  |
| --- |
| #include<sys/types.h>  #include<unistd.h>  #include<stdlib.h>  #include<stdio.h>  #include<pthread.h>  #include<semaphore.h>  #include<time.h>  #define N 5 //哲学家数量  #define LEFT(i) (i+N-1)%N //左手边哲学家编号  #define RIGHT(i) (i+1)%N //右手边哲家编号  #define HUNGRY 0 //饥饿  #define THINKING 1 //思考  #define EATING 2 //就餐  #define U\_SECOND 1000000 //1秒对应的微秒数  pthread\_mutex\_t mutex; //互斥量  int state[N]; //记录每个哲学家状态  //每个哲学家的思考时间，就餐时间，思考开始时间，就餐开始时间  clock\_t thinking\_time[N], eating\_time[N], start\_eating\_time[N], start\_thinking\_time[N];  //线程函数  void \*thread\_function(void \*arg);  int main()  {  pthread\_mutex\_init(&mutex, NULL);  pthread\_t a,b,c,d,e;  //为每一个哲学家开启一个线程，传递哲学家编号  pthread\_create(&a,NULL,thread\_function,"0");  pthread\_create(&b,NULL,thread\_function,"1");  pthread\_create(&c,NULL,thread\_function,"2");  pthread\_create(&d,NULL,thread\_function,"3");  pthread\_create(&e,NULL,thread\_function,"4");  //初始化随机数种子  srand((unsigned int)(time(NULL)));  while(1)  {  ;  }  }  void \*thread\_function(void \*arg)  {  char \*a = (char \*)arg;  int num = a[0] - '0'; //根据传递参数获取哲学家编号  int rand\_time;  while(1)  {  //关键代码加锁  pthread\_mutex\_lock(&mutex);  //如果该哲学家处于饥饿 并且 左右两位哲学家都没有在就餐 就拿起叉子就餐  if(state[num] == HUNGRY && state[LEFT(num)] != EATING && state[RIGHT(num)] != EATING)  {  state[num] = EATING;  start\_eating\_time[num] = clock(); //记录开始就餐时间  eating\_time[num] = (rand() % 5 + 5) \* U\_SECOND; //随机生成就餐时间  //输出状态  printf("state: %d %d %d %d %d\n",state[0],state[1],state[2],state[3],state[4]);  //printf("%d is eating\n",num);  }  else if(state[num] == EATING)  {  //就餐时间已到 ，开始思考  if(clock() - start\_eating\_time[num] >= eating\_time[num]) //  {  state[num] = THINKING;  //printf("%d is thinking\n",num);  printf("state number is: %d %d %d %d %d\n",state[0],state[1],state[2],state[3],state[4]);  start\_thinking\_time[num] = clock(); //记录开始思考时间  thinking\_time[num] = (rand() % 10 + 10) \* U\_SECOND; //随机生成思考时间  }  }  else if(state[num] == THINKING)  {  //思考一定时间后，哲学家饿了，需要就餐  if(clock() - start\_thinking\_time[num] >= thinking\_time[num])  {  state[num] = HUNGRY;  printf("state number is: %d %d %d %d %d\n",state[0],state[1],state[2],state[3],state[4]);  // printf("%d is hungry\n",num);  }  }  pthread\_mutex\_unlock(&mutex);  }  } |

哲学家就餐问题，代码运行成功截图

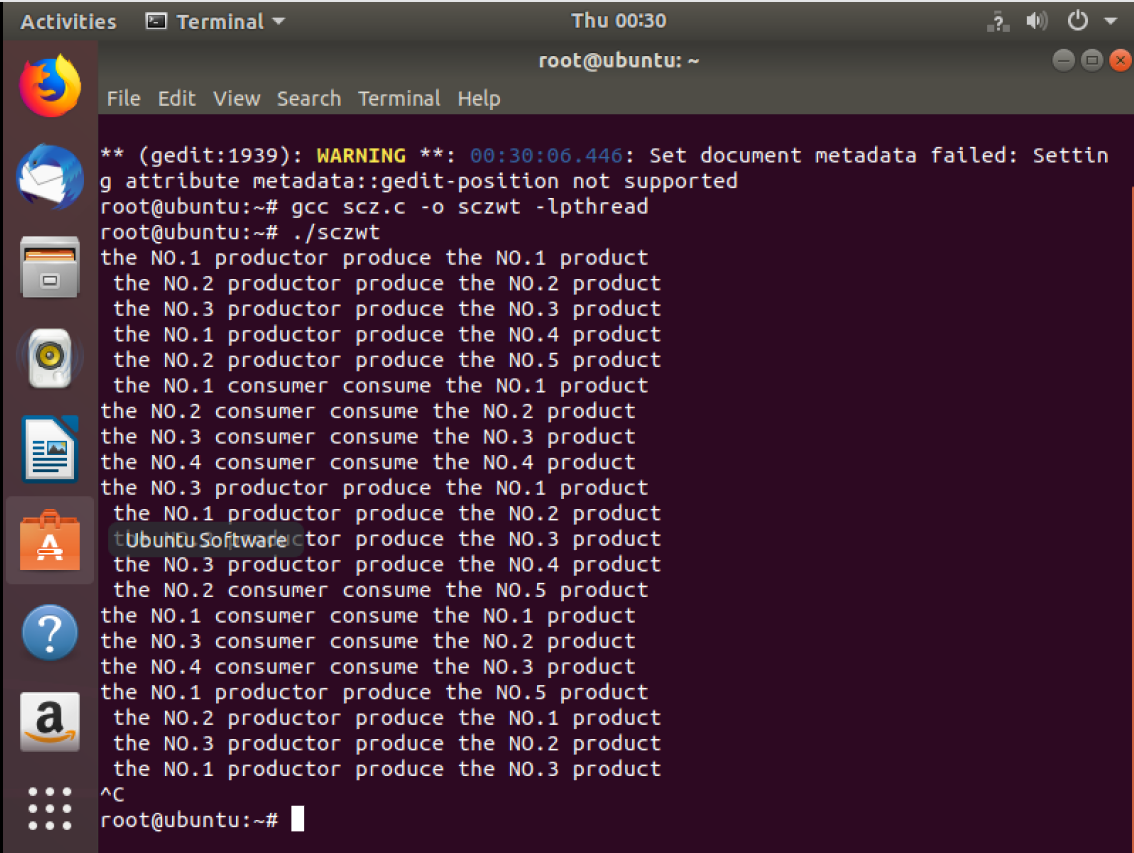


**（2）生产者消费者实验：**

**生产者消费者实验代码：**

|  |
| --- |
| #include <stdio.h>  #include <semaphore.h>  #include <stdlib.h>  #include <pthread.h>  #include <unistd.h>  #define PRODUCTOR 3  #define CONSUMER 4  #define MAX 5  int in = 0;  int out = 0;  int productor\_id = 0;  int consumer\_id = 0;  sem\_t empty\_zoom ;  sem\_t full\_zoom ;  pthread\_mutex\_t product;  void \*productor(){  int id = ++productor\_id;  while(1){  sleep(1);  sem\_wait(&empty\_zoom);  pthread\_mutex\_lock(&product);  in = in % MAX;  printf("the NO.%d productor produce the NO.%d product\n ",id ,in+1);  in += 1;  pthread\_mutex\_unlock(&product);  sem\_post(&full\_zoom);  }  }  void \*consumer(){  int id = ++consumer\_id;  while(1){  sleep(3);  sem\_wait(&full\_zoom);  pthread\_mutex\_lock(&product);  out = out % MAX;  printf("the NO.%d consumer consume the NO.%d product\n",id ,out+1 );  out += 1;  pthread\_mutex\_unlock(&product);  sem\_post(&empty\_zoom);  }  }  int main(){  pthread\_t id1[PRODUCTOR];  pthread\_t id2[CONSUMER];  int i;  int ret1[PRODUCTOR];  int ret2[CONSUMER];  int ini1 = sem\_init(&empty\_zoom, 0, MAX);  int ini2 = sem\_init(&full\_zoom, 0, 0);  if(ini1 && ini2 != 0)  {  printf("Sem init failed\n");  exit(1);  }  int ini3 = pthread\_mutex\_init(&product, NULL);  if(ini3 != 0 )  {  printf("mutex init failed\n");  exit(1);  }  for(i = 0; i < PRODUCTOR; i++)  {  ret1[i] = pthread\_create(&id1[i], NULL, productor, (void\*)(&i));  if(ret1[i] != 0)  {  printf("product%d creat failed\n", i);  exit(1);  }  }  for(i = 0; i< CONSUMER; i++)  {  ret2[i] = pthread\_create(&id2[i], NULL, consumer, NULL);  if(ret2[i] != 0)  {  printf("consumer%d creat failed\n", i);  exit(1);  }  }  for(i = 0; i < PRODUCTOR; i++) {pthread\_join(id1[i], NULL);}  for(i = 0; i < CONSUMER; i++) {pthread\_join(id2[i], NULL);}  exit(0);  } |

生产者消费者程序运行成功截图：



**九、总结及心得体会：**

本次实验中我们通过使用记录型信号量实现了哲学家就餐问题，生产者消费者问题，并通过一些修正避免了哲学家就餐的死锁问题。总结一下，在本次实验中我学会了如下知识:

1. linux操作系统信号量库函数的使用。
2. 理解并掌握了进程、线程的概念，熟悉了相关的控制原语。
3. 掌握了使用信号量原语解决进程、线程间互斥和同步方法。
4. 为了解决可能的死锁问题，更加深入地理解了死锁需要满足的互斥、请求与保持、环路等待、不剥夺条件。
5. 学会使用互斥信号量，资源信号量来解决问题。
6. 熟练了linux操作系统编程

**十、对本实验过程及方法、手段的改进建议：**

1. 熟能生巧，一定要多实践，做起实验来才能得心应手。
2. 遇到不会的点或者不熟悉的点，先查书找不到答案再百度。
3. 实验课可以很好的检验平时有没有认真学，而且能纠正对知识理解上的错误。

**报告评分：**

**指导教师签字：**