电子科技大学信息与软件工程学院

**实 验 报 告**

学 号 2016220304031

姓 名 邓巧

（实验） 课程名称 计算机操作系统

理论教师 王佳昊

实验教师 杨珊

**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：邓巧 学号：2016220304031 指导教师：王佳昊**

**实验地点：信软楼400 实验时间：2018.4.20**

**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：**

1.哲学家就餐问题

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。

2．生产者消费者问题

1. 掌握进程、线程的概念，熟悉相关的控制语。
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.哲学家就餐问题

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.生产者消费者问题

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.哲学家就餐问题

在linux系统下实现教材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，如吃饭，思考。连续运行30次以上都未出现死锁现象。

2.生产者消费者问题

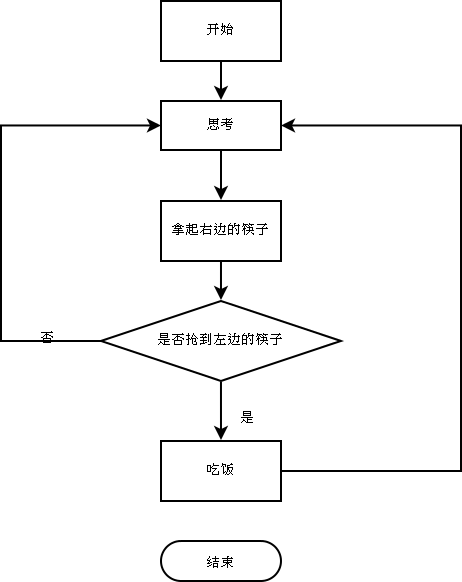
1. 有一群生产者进程在生产产品，并将这些产品提供给消费者进程去消费。为使生产者进程与消费者进程能并发执行，在两者之间设置了一个具有n个缓冲区的缓冲池：生产者进程从文件中读取一个数据，并将它存放到一个缓冲区中；消费者进程从一个缓冲区中取走数据，并输出此数据。生产者和消费者之间必须保持同步原则：不允许消费者进程到一个空缓冲区去取产品；也不允许生产者进程向一个已装满产品且尚未被取走的缓冲区中投放产品。
2. 创建3个进程（或者线程）作为生产者，4个进程（或者线程）作为消费者。创建一个文件作为数据源，文件中事先写入一些内容作为数据。
3. 生产者和消费者进程（或者线程）都具有相同的优先级。

**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：**

1.哲学家就餐



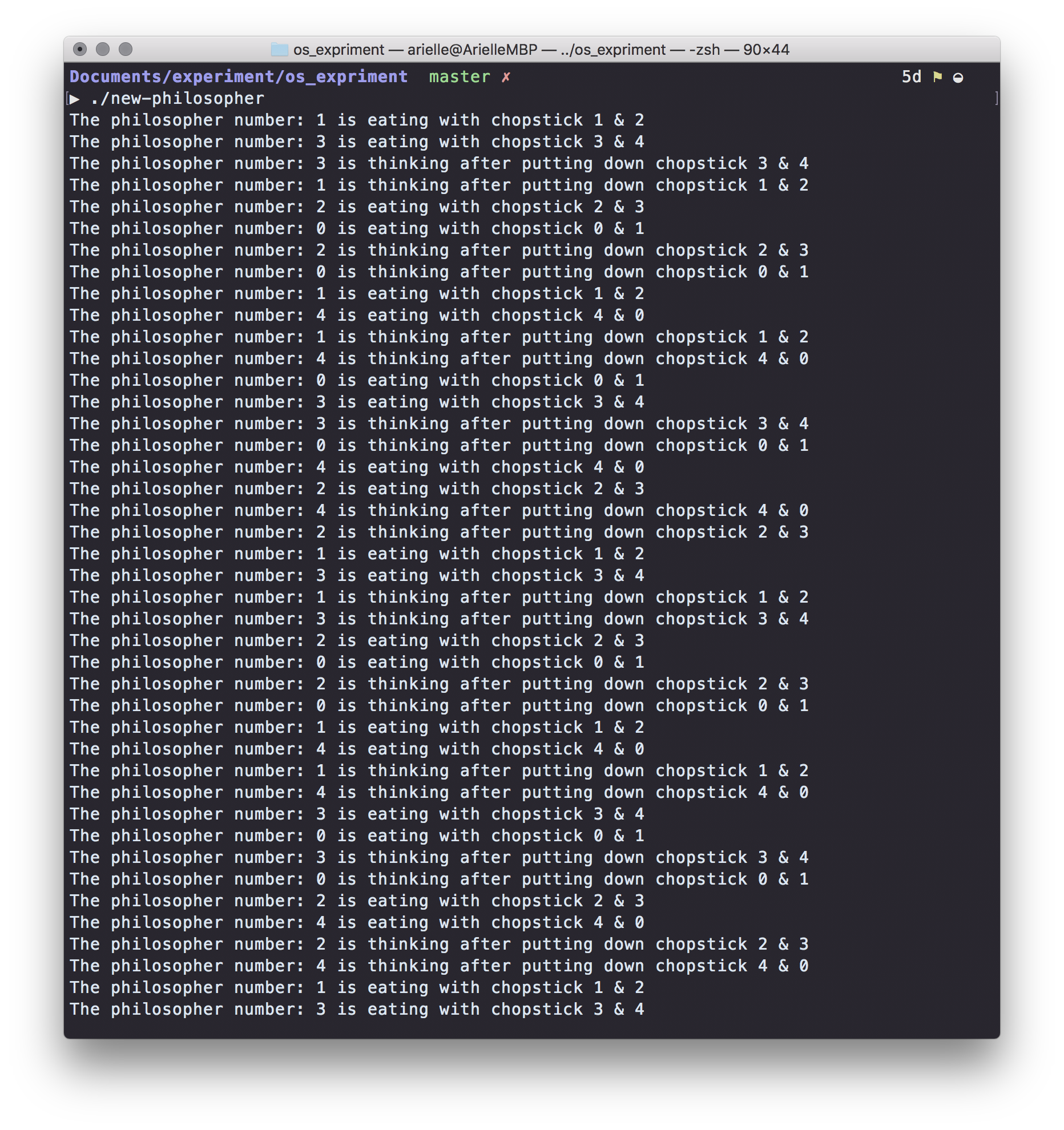
2.生产者消费者

1. 分配具有n个缓冲区的缓冲池，作为共享资源。
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。
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。
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。
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。
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）**

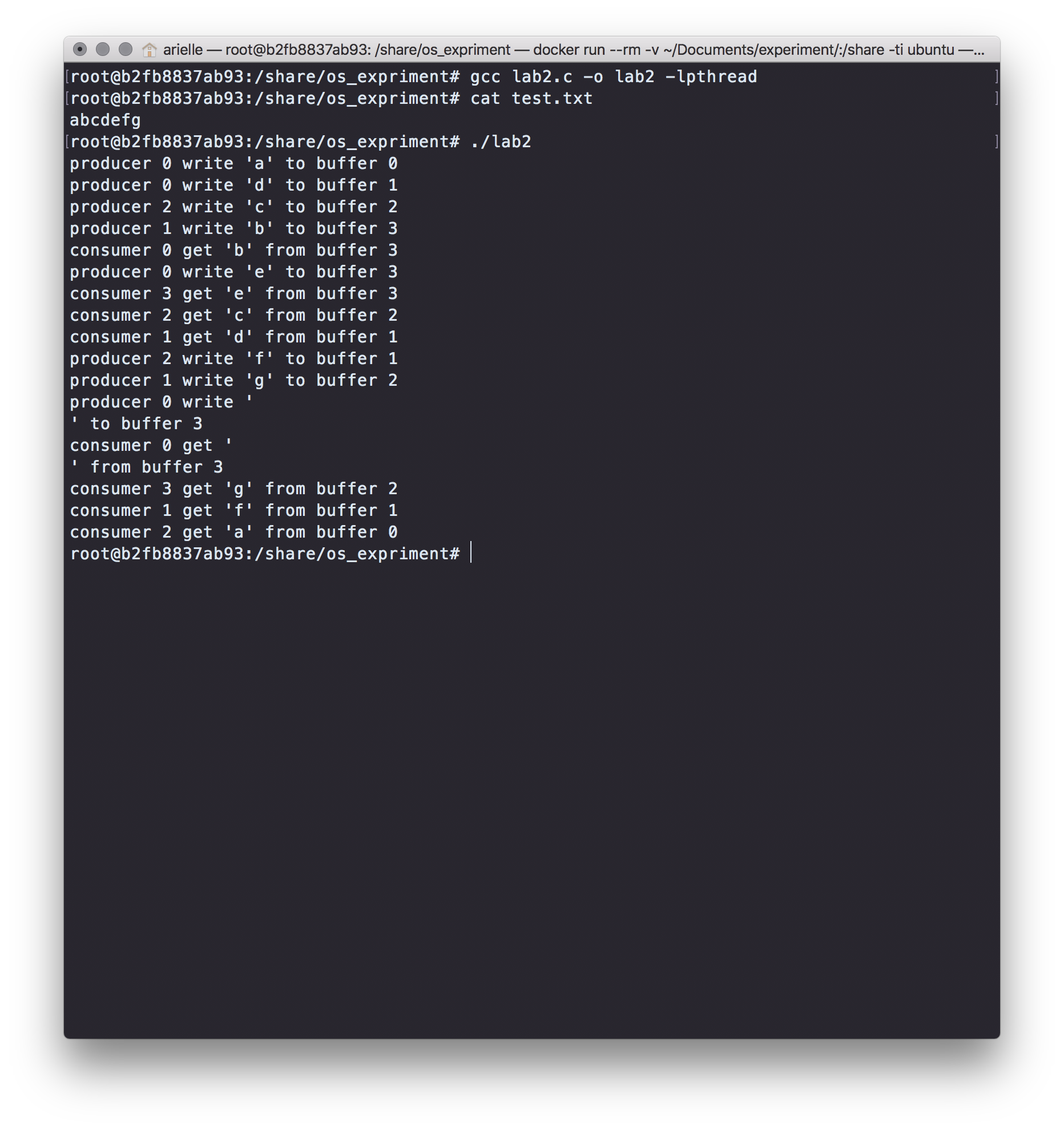
1.哲学家就餐问题

|  |
| --- |
| /\*the philosopher\*/  /\*author:dengqiao 2016220304031\*/  /\*date:2018 4.20\*/  #include <unistd.h>  #include <sys/types.h>  #include <sys/stat.h>  #include <stdio.h>  #include <stdlib.h>  #include <string.h>  #include <time.h>  #include <stdbool.h>  #include <pthread.h>  #define PHILOSOPHER\_N 5  pthread\_mutex\_t chopstick[PHILOSOPHER\_N];  int take\_chopstick(int num);  void release\_chopstick(int num);  void \*philosopher(int \*intarg);  int main(){  void \*ret;  int i,j,k;  pthread\_t tid[PHILOSOPHER\_N];  for(i = 0; i < PHILOSOPHER\_N; i++)  pthread\_mutex\_init(&chopstick[i],NULL);  for(j = 0; j < PHILOSOPHER\_N; j++){  int \*id = (int \*)malloc(sizeof(id));  \*id = j;  pthread\_create(&tid[j],NULL,philosopher,id);  }  for(k = 0; k < PHILOSOPHER\_N; k++)  pthread\_join(tid[k],&ret);  return 0;  }  void \*philosopher(int \*intarg){  int num = \*((int \*)intarg);  int next\_num = (num + 1) % PHILOSOPHER\_N;  while(1){  // sleep(5);  if(!take\_chopstick(num))  continue;  sleep(1);  printf("The philosopher number: %d is eating with chopstick %d & %d\n",num,num,next\_num);  sleep(1);  release\_chopstick(num);  printf("The philosopher number: %d is thinking after putting down chopstick %d & %d\n",num,num,next\_num);  sleep(1);  }  free((int \*)intarg);  return NULL;  }  int take\_chopstick(int num){  int next\_num = (num + 1) % PHILOSOPHER\_N;  pthread\_mutex\_lock(&chopstick[num]);  if(pthread\_mutex\_trylock(&chopstick[next\_num])){  pthread\_mutex\_unlock(&chopstick[num]);  return 0;//can't take chopstick;  }  else{  // printf("takechopstick:%d & %d",num,next\_num);  return 1;  }  }  void release\_chopstick(int num){  int next\_num = (num + 1) % PHILOSOPHER\_N;  // printf("releasechpstick:%d & %d",num,next\_num);  pthread\_mutex\_unlock(&chopstick[num]);  pthread\_mutex\_unlock(&chopstick[next\_num]);  } |



2.生产者消费者

|  |
| --- |
| /\*consumer and producer\*/  /\*author:dengqiao 2016220304031\*/  /\*date:2018 4.23\*/  #include<unistd.h>  #include<sys/types.h>  #include<sys/stat.h>  #include<stdio.h>  #include<stdlib.h>  #include<time.h>  #include<pthread.h>  #include<semaphore.h>  #include <sys/ipc.h>  #include <sys/shm.h>  #include <errno.h>  #include <time.h>  #define CONSUMER\_N 4  #define PRODUCER\_N 3  #define BUFFER\_N CONSUMER\_N  #define SHM\_KEY 0x1234  void \*producer(void \*);  void \*consumer(void \*);  int find\_char(char \*,int);  int find\_empty(char \*,int);  FILE \*fd;  sem\_t sem\_empty;  sem\_t sem\_full;  pthread\_mutex\_t mutex;  void \*shm;  int main(){  int \*id\_p;  int \*id\_c;  int i,j,k;  pthread\_t p\_tid[PRODUCER\_N];  pthread\_t c\_tid[CONSUMER\_N];  pthread\_mutex\_init(&mutex,NULL);  if(sem\_init(&sem\_empty,0,BUFFER\_N) == -1)  {  perror("sem\_init sem\_empty");  exit(errno);  }  if(sem\_init(&sem\_full,0,0) == -1)  {  perror("sem\_init sem\_full");  exit(errno);  }  fd = fopen("test.txt","r");  int shmid;  shm = NULL; // initialize the shm pointer  if((shmid = shmget(SHM\_KEY,sizeof(char) \* BUFFER\_N,IPC\_CREAT | 0666)) == -1)  fprintf(stderr,"shmget failed\n");  if((shm = shmat(shmid,0,0)) == (void \*)-1)  fprintf(stderr,"shmat failed\n");  for(k = 0; k < BUFFER\_N; k++)  \*((char \*)shm + k) = '\0';  for(i = 0; i < PRODUCER\_N; i++){  id\_p = (int \*)malloc(sizeof(int));  \*id\_p = i;  // printf("\*id\_p = %d\n", \*id\_p);  pthread\_create(p\_tid + i,NULL,producer,id\_p);  }  for(j = 0; j < CONSUMER\_N; j++){  id\_c = (int \*)malloc(sizeof(id\_c));  \*id\_c = j;  pthread\_create(c\_tid + j,NULL,consumer,id\_c);  }  void \*ret\_val;  for(i = 0; i < PRODUCER\_N; i++){  pthread\_join(p\_tid[i], &ret\_val);  }  for(i = 0; i < CONSUMER\_N; i++)  {  pthread\_join(c\_tid[i], &ret\_val);  }  // free(id\_p);  // id\_p = NULL;  // free(id\_c);  // id\_c = NULL;  }  void \*consumer(void \*number){  int sp,num;  num = \*(int \*)number;  while(1){  time\_t start\_time = time((time\_t \*)NULL);  while(sem\_trywait(&sem\_full))  {  // printf("!");  time\_t elapse = time((time\_t \*)NULL) - start\_time;  // printf("%d", elapse);  if(elapse > 1)  {  return 0;  }  }  // sem\_wait(&sem\_full);  /\* int temp; \*/  /\* sem\_getvalue(&sem\_full, &temp); \*/  /\* printf("sem\_full = %d\n", temp); \*/  /\* sem\_getvalue(&sem\_empty, &temp); \*/  /\* printf("sem\_empty = %d\n", temp); \*/  pthread\_mutex\_lock(&mutex);  sp = find\_char(shm,BUFFER\_N);  printf("consumer %d get '%c' from buffer %d\n",num,((char \*)shm)[sp],sp);  ((char \*)shm)[sp] = '\0';  pthread\_mutex\_unlock(&mutex);  sem\_post(&sem\_empty);  sleep(1);  }  free((int \*)number);  return (void \*)-1;  }  void \*producer(void \*number){  int c,sp,num;  num = \*(int \*)number;  while((c = getc(fd)) !=EOF){  /\* int temp; \*/  /\* sem\_getvalue(&sem\_full, &temp); \*/  /\* printf("before sem\_full = %d\n", temp); \*/  /\* sem\_getvalue(&sem\_empty, &temp); \*/  /\* printf("before sem\_empty = %d\n", temp); \*/  /\* fflush(stdout); \*/  sem\_wait(&sem\_empty);  pthread\_mutex\_lock(&mutex);  sp = find\_empty(shm,BUFFER\_N);  \*((char \*)shm + sp) = c;  printf("producer %d write '%c' to buffer %d\n",num,c,sp);  //sleep(1);  pthread\_mutex\_unlock(&mutex);  sem\_post(&sem\_full);  }  free((int \*)number);  return (void \*)-1;  }  int find\_empty(char \*buffer,int MAX){  int i;  for(i = 0; \*(buffer + i) != '\0' && i < MAX; i++)  ;  return i;  }  int find\_char(char \*buffer,int MAX){  int i;  for(i = MAX - 1; \*(buffer + i) == '\0' && i >=0; i--)  ;  return i;  } |



**九、总结及心得体会：**

1.应该尽量使用符合POSIX标准的函数

2.遇到问题可以多搜集资料

**十、对本实验过程及方法、手段的改进建议：**

无

**报告评分：**

**指导教师签字：**