操作系统实验报告

郑重声明：

1、实验手册中的所有实验均有本人独立编码、调试和测试。

2、实验手册中给出的实验数据和结果完全由本人所完成的程序给出。

3、本人了解：不按照前两条要求所完成的实验报告已经构成了抄袭或造假行为，本人将承担相应的不良后果。

姓名：_____陈红宇_____（签名）

学号：_____201430610122_____

班级：_____4 班_____

提交日期：_____4.29_____

成绩：_____

一、 实验目的

熟悉 Linux 操作系统进程通信的系统调用

二、 实验内容

实现生产者和消费者问题。

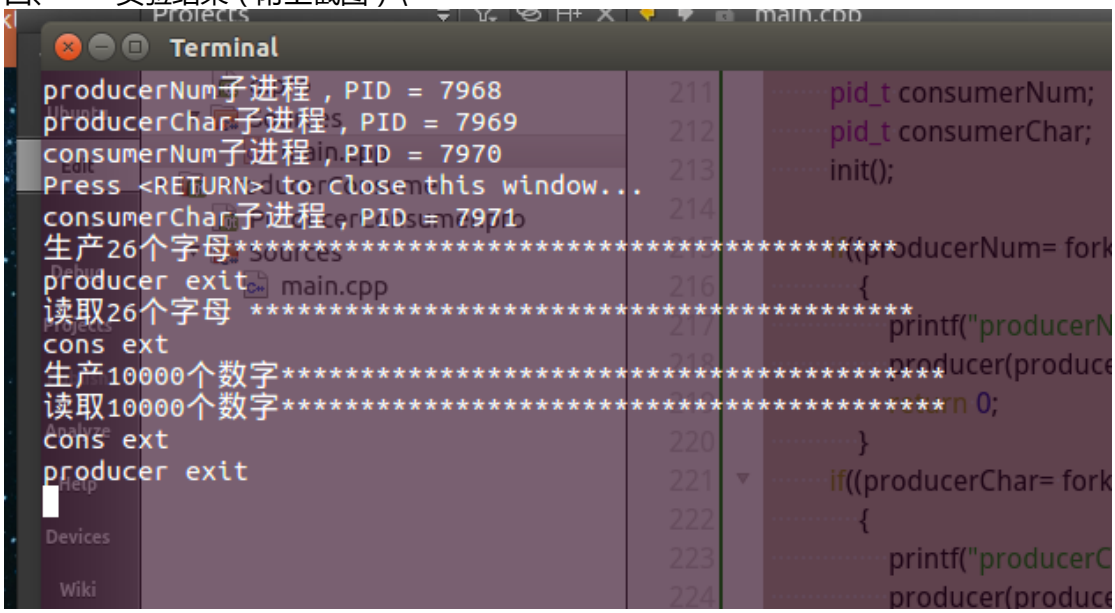
创建两个生产者进程和两个消费者进程，生产者进程 a 需要生成 10000 个整数,每次都将自己的进程号（用 getpid()函数获得）和生成的整数放入共享内存中（共享内存大小为 64Byte）。生产者 b 每次从 26 个英文字母中选一个，并将自己的进程号和选中的字母放入共享内存中，直到 26 个字母全部都选中。消费者进程 c 负责从共享内存中读取数据生产者进程 a 的数据并且将这些数据写入文件 a.out。消费者进程 d 从共享内存读取进程 b 的数据后写入 b.out 中。

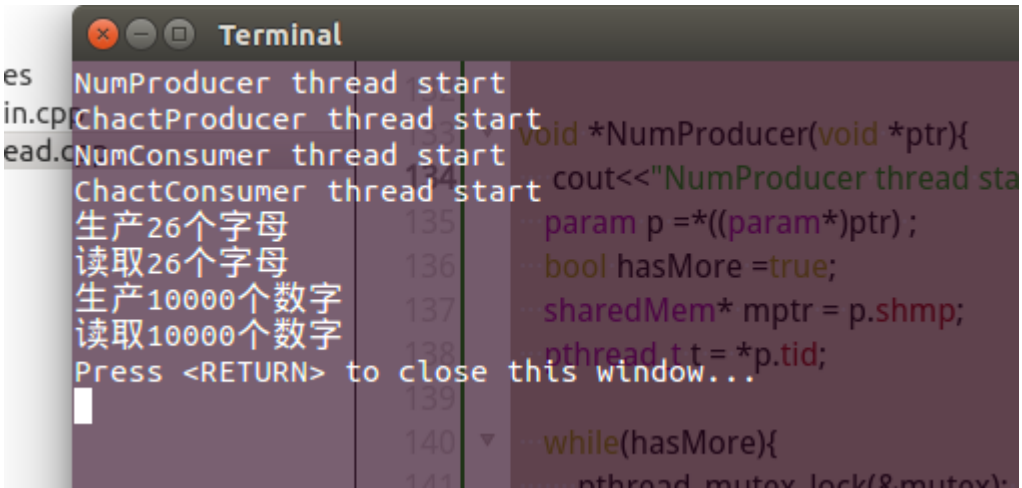
三、 实验原理

进程: 利用 empty 和 full 两个信号量及成对的 down/up 操作实现操作等待队列以同步共享内存,利用一元信号量实现对竞争区的互斥访问.

线程; 通过互斥锁实现对竞争区的竞争互斥访问.在竞争区中,若无所需资源,则等待相应条件变量被唤醒,同时交出锁,完成操作后,唤醒等待在相应条件变量的线程,再解锁

四、 实验结果（附上截图）\





五、　　结论分析

进程间通过对资源 empty/full 信号量的异步修改实现对竞争资源的读写同步,通过互斥实现进程间的公平竞争.

源码:进程

```
#include "stdio.h"
#include <sys/shm.h>
#include<sys/sem.h>
 #include <semaphore.h>
#include <stdlib.h>
#include <unistd.h>
#include<iostream>
#include <fcntl.h>
#include<fstream>
using namespace std;

#define N 10000 //生产数字数目

struct sharedMem{
    int buff[16];
     int *head;
     int size;
}*shmp;

sem_t  *mutex;
sem_t  *full;
sem_t  *empty;

int shmId = -1;

void produceNum(bool &hasMore){
        static int count = N;
        if(shmp->size>=2){
        shmp->size-=2;
        int data = count;
        *(shmp->head)= data;
        shmp->head++;
        *(shmp->head) = getpid();
        if(shmp->size >0)
           shmp->head++;
        count--;
//      printf("生产%d 数字 \n",count);
        if(count == 0) {
```

```
            printf("生产 10000 个数字*****************************************\n");
            hasMore = false;
        }
        }
}

void produceChact(bool &hasMore ){
    static bool character[26]={false};
    if(shmp->size>=2){
    shmp->size-=2;
    int i;
    for( i =0;i<26;i++){
        if(!character[i]) {
            character[i] = true;
//          printf("生产字母 %c\n",char('a'+i));
           *(shmp->head)= 'a'+i;
            shmp->head++;
           *(shmp->head)= getpid();
           if(shmp->size > 0)
             shmp->head++;
            break;
        }
    }
    if(i == 25){
        printf("生产 26 个字母*****************************************\n");
        hasMore = false;
    }
    }
}
//读取到数据,返回 1,else-1
int consumeNum(bool &hasMore,pid_t pid,ofstream& out)
{
    static int count = 10000;
    //buff 不满时,指针指在空位,满时,指在顶部
    switch(shmp->size){
        case 16: //没有数据
                return -1;
        case  0: //满了，指在顶部
                if( *(shmp->head) == pid)  {
                    out<<*(shmp->head) <<"\t";
                    (shmp->head)--;
                    break;
                }
                else return -1;
        default:
                  if(  *(shmp->head-1) == pid) {
                      out<<*(shmp->head-1) <<"\t";
                      (shmp->head)-=2;
                  }
                  else return -1;
    }
```

```
    shmp->size+=2;
        int data = *(shmp->head);
//      printf("读取数字 %d\n",count);
        out<<data<<"\n";
        count--;
        if(count ==0) {
            printf("读取 10000 个数字*****************************************\n");
            hasMore =false;
        }
        return 1;
}

int consumeChact (bool &hasMore,pid_t pid,ofstream& out)
{
    static int characters= 26;
    switch(shmp->size){
        case 16: //没有数据
                return -1;
                break;
        case  0: //满了，指在顶部
                if( *(shmp->head) == pid)  {
                    out<<pid <<"\t";
                    (shmp->head)--; //指向数据
                    break;
                }
                else return -1;//不是相应生产者产生的数据
        default:
                  if(  *(shmp->head-1) == pid){
                      out<<pid<<"\t";
                      (shmp->head) -=2;
                       break;
                  }
                  else return -1; //不是相应生产者产生的数据
    }
    shmp->size+=2;
        char data = (char)(*(shmp->head));
//      printf("读取字母 %c\n",data);
        out<<data <<"\n";
        characters--;
        if(characters == 0){
            printf("读取 26 个字母 ****************************************\n");
            hasMore=false;
        }
        return 1;
}

typedef void(* Produce)(bool&);
typedef int(*Consume)(bool&,pid_t,ofstream& );
void producer(Produce produce){
    bool hasMore =true;
```

```
    while(hasMore)
    {
            sem_wait(empty);
            sem_wait(mutex);//cout<<" per lock per\n";
            produce( hasMore);
            sem_post(mutex);//cout<<"per unlock per\n";
            sem_post(full);
    }
    cout<<"producer exit\n";
}
void consumer(Consume consume,pid_t t,ofstream&f){
    bool hasMore =true;
    int tag=0;

    while(hasMore)
    {
            sem_wait(full);
            sem_wait(mutex); //cout<<"cer lock cer\n";
            tag= consume(hasMore,t,f);
            sem_post(mutex);//cout<<"cer unlock cer\n";
            if(tag> 0)sem_post(empty); //读取数据成功
            else sem_post(full);//读取失败,恢复 up 信号量以同步
    }
    cout<<"cons ext\n";
}

void init()
{
    if((shmId = shmget(IPC_PRIVATE,sizeof(struct sharedMem),0600)) < 0)
    {
        perror("create shared memory failed");
        exit(1);
    }
    shmp = (sharedMem *)shmat(shmId,0,0);
    shmp->head=shmp->buff;
    shmp->size=16;
    mutex = sem_open("mutex",O_CREAT,0644,1);
    full  = sem_open("full",O_CREAT,0644,0);
    empty = sem_open("empty",O_CREAT,0644,8);

    if(mutex == SEM_FAILED)
    {
      perror("unable to create  mutex semaphore");
      sem_unlink("mutex");
      exit(-1);
    }
    if(full == SEM_FAILED)
    {
      perror("unable to create full semaphore");
      sem_unlink("full");
      exit(-1);
    }
    if(empty == SEM_FAILED)
```

```
    {
      perror("unable to create  empty semaphore");
      sem_unlink("empty");
      exit(-1);
    }

}

    int main()
    {
        pid_t producerNum;
        pid_t producerChar;
        pid_t consumerNum;
        pid_t consumerChar;
        init();

        if((producerNum= fork()) == 0)
            {
                printf("producerNum 子进程，PID = %d\n",getpid());
                producer(produceNum);
                return 0;
            }
        if((producerChar= fork()) == 0)
            {
                printf("producerChar 子进程，PID = %d\n",getpid());
                producer(produceChact);
                return 0;
            }
        if((consumerNum= fork()) == 0)
            {
                printf("consumerNum 子进程，PID = %d\n",getpid());
                ofstream f ("a.out");
                consumer(consumeNum,producerNum,f);
                return 0;
            }
        if((consumerChar= fork()) == 0)
            {
                printf("consumerChar 子进程，PID = %d\n",getpid());
                ofstream f ("b.out");
                consumer(consumeChact,producerChar,f);
                return 0;
            }

        shmctl(shmId, IPC_RMID, 0);
        sem_close(mutex);
        sem_unlink("full");
        sem_unlink("empty");
        return 0;
    }
```

线程:

```
      #include <iostream>
#include<pthread.h>
#include<sys/shm.h>
#include <sys/syscall.h>
#include <stdlib.h>
#include<stdio.h>
#include <unistd.h>
#include <string.h>
#include<fstream>
using namespace std;

pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t cond_empty = PTHREAD_COND_INITIALIZER;
pthread_cond_t cond_full = PTHREAD_COND_INITIALIZER;

struct sharedMem{
    int buff[16];
     int *head ;
     int surplus ;
}*shmp;

struct param{
    sharedMem* shmp;
    pthread_t *tid;
};

void produceNum(sharedMem* shmp,bool &hasMore,pthread_t pid){
        static int count = 0;
        if(shmp->surplus >= 2){
            shmp->surplus-=2;
        int data = random()%1000;
        *(shmp->head)= data;
        shmp->head++;
        *(shmp->head) = pid;
        if(shmp->surplus !=0)
           shmp->head++;
        count++;
   //   printf("生产%d 个数字 \n",count);

        if(count >= 10000) {
            printf("生产 10000 个数字\n");
            hasMore = false;
        }
     }
}
void produceChact(sharedMem* shmp,bool &hasMore,pthread_t pid ){
    static bool character[26]={false};
    if(shmp->surplus>=2){
        shmp->surplus-=2;

    int i;
    for( i =0;i<26;i++){
        if(!character[i]) {
```

```
            character[i] = true;
  //        cout<<"生产字母 "<<char('a'+i);
            *(shmp->head)= 'a'+i;
            shmp->head++;
            *(shmp->head)= pid;
            if(shmp->surplus != 0)
            shmp->head++;
            break;
        }
    }
    if(i == 25){
        printf("生产 26 个字母\n");
        hasMore = false;
    }
    }
}
void consumeNum(sharedMem* shmp,bool &hasMore,pthread_t pid,ofstream &out){
    static int count = 10000;

    switch( (shmp->surplus)  ){
        case 16: //没有数据
                return ;       //应该唤醒生产者
        case  0: //满了，指在顶部
                if( *(shmp->head) == (int)pid) {
                    out<<*(shmp->head) <<"\t";
                    shmp->head--;
                }
                else return ;    //应该唤醒另一个消费者
                break;
        default:
                if(  *(shmp->head-1) == (int)pid) {
                    out<<*(shmp->head-1) <<"\t";
                    shmp->head-=2;
                }
                else return ;
    }
        shmp->surplus+=2;
        int data = *(shmp->head);
      //  printf("读取数字 %d\n",count);
        out<<data<<"\n";
        count--;
        if(count ==0) {
            printf("读取 10000 个数字\n");
            hasMore =false;
        }
}
void consumeChact(sharedMem* shmp,bool &hasMore,pthread_t pid,ofstream &out)
{
    static int characters= 26;
    switch((shmp->surplus ) ){
```

```
        case 16: //没有数据
                return ;
        case  0: //满了，指在顶部
                if( *(shmp->head) == (int)pid) {
                    out<<*(shmp->head) <<"\t";
                    shmp->head--; //指向数据
                }
                else return ;//不是相应生产者产生的数据
                break;
        default:
                if(  *(shmp->head-1) == (int)pid){
                    out<<*(shmp->head-1) <<"\t";
                    shmp->head-=2;
                }
                else return; //不是相应生产者产生的数据
    }
        shmp->surplus+=2;//余量增加
        char data = (char)(*(shmp->head));
   //   printf("读取字母 %d\n",characters);
        out<<data<<"\n";
        characters--;
        if(characters == 0){
            printf("读取 26 个字母 \n");
            hasMore=false;
        }
}

void *NumProducer(void *ptr){
    cout<<"NumProducer thread start\n";
   param p =*((param*)ptr) ;
   bool hasMore =true;
   sharedMem* mptr = p.shmp;
   pthread_t t = *p.tid;

   while(hasMore){
       pthread_mutex_lock(&mutex); //  cout<<"nper lock \n";
                                   while((mptr->surplus)      <       2){              /*cout<<"nper
wait\n";*/pthread_cond_wait(&cond_empty,&mutex);}
       produceNum(mptr,hasMore,t);
       pthread_cond_signal(&cond_full);
       pthread_mutex_unlock(&mutex);   //cout<<"nper unlock\n";
   }
   pthread_exit(0);
}
void *ChactProducer(void*ptr){
    cout<<"ChactProducer thread start\n";
    param p =*((param*)ptr) ;
    bool hasMore =true;
    sharedMem* mptr = p.shmp;
    pthread_t t =*p.tid;
```

```
    while(hasMore){
        pthread_mutex_lock(&mutex);
        while((mptr->surplus )< 2) pthread_cond_wait(&cond_empty,&mutex);
       produceChact(mptr,hasMore,t);
        pthread_cond_signal(&cond_full);
        pthread_mutex_unlock(&mutex);  //cout<<"cper unlock\n";
    }
    pthread_exit(0);
}
void *NumConsumer(void*ptr){
    cout<<"NumConsumer thread start\n";
    param p =*((param*)ptr) ;
    bool hasMore =true;
    sharedMem* mptr = p.shmp;
    pthread_t t =*p.tid;
    ofstream out("a.out");
    while(hasMore){
        pthread_mutex_lock(&mutex); //cout<<"ncer lock\n";
        while((mptr->surplus )>14) pthread_cond_wait(&cond_full,&mutex);
        consumeNum(mptr,hasMore,t,out);
       pthread_cond_signal(&cond_empty);
        pthread_mutex_unlock(&mutex); //  cout<<"ncer unlock\n";
    }
    pthread_exit(0);
}
void* ChactConsumer(void*ptr){
    cout<<"ChactConsumer thread start\n";
    param p =*((param*)ptr) ;
    bool hasMore =true;
    sharedMem* mptr = p.shmp;
    pthread_t t =*p.tid;
    ofstream out("b,out");
    while(hasMore){
        pthread_mutex_lock(&mutex);;//cout<<"ccer lock \n";
        while((mptr->surplus) > 14) pthread_cond_wait(&cond_full,&mutex);
        consumeChact(mptr,hasMore,t,out);
        pthread_cond_signal(&cond_empty);
        pthread_mutex_unlock(&mutex);//cout<<"ccer unlock\n";
    }
    pthread_exit(0);
}

void getShm(int &shmId){
            if((shmId = shmget(IPC_PRIVATE,sizeof(struct sharedMem),IPC_CREAT|0660))<0){
                        perror("shmget");
                        exit(1);
             }
           shmp = (sharedMem*)shmat(shmId,0,0);
            if(shmp == (void*)-1){
                        perror("shmat");
                        exit(1);
            }

             shmp->head = shmp->buff;
```

```
            shmp->surplus = 16;
}

int main()
{
    int shmId = -1;
    getShm(shmId);
    pthread_t pc,pn,cc,cn;
    param p_pn,p_pc,p_cn,p_cc,*pc_ptr,*pn_ptr,*cn_ptr,*cc_ptr;
    p_pn.shmp = shmp;     p_pn.tid = &pn;   pn_ptr = &p_pn;
    p_pc.shmp=  shmp;     p_pc.tid =  &pc;     pc_ptr = &p_pc;
    p_cn.shmp = shmp;     p_cn.tid =  &pn;     cn_ptr = &p_cn;
    p_cc.shmp= shmp;      p_cc.tid  = &pc;    cc_ptr = &p_cc;

      pthread_create(&pn,0,NumProducer,(void*)pn_ptr);
      pthread_create(&pc,0,ChactProducer,(void*)pc_ptr);
      pthread_create(&cn,0,NumConsumer,(void*)cn_ptr);
      pthread_create(&cc,0,ChactConsumer,(void*)cc_ptr);

      pthread_join(pn,0);
      pthread_join(pc,0);
      pthread_join(cn,0);
      pthread_join(cc,0);

      pthread_cond_destroy(&cond_full);
      pthread_cond_destroy(&cond_empty);
      pthread_mutex_destroy(&mutex);
    return 0;
}
```