mqmelon L55 2015-06-26 19:13:53 信号量的实现和应用

"操作系统原理与实践"实验报告

信号量的实现和应用

信号量的实现和应用#

实验目的##

.加深对进程同步与互斥概念的认识； .掌握信号量的使用，并应用它解决生产者——消费者问题； .掌握信号量的实现原理。 ##实验内容## 本次实验的基本内容是：

- 1.在Ubuntu下编写程序，用信号量解决生产者——消费者问题；
- 2.在0.11中实现信号量，用生产者—消费者程序检验之。
- 3.用信号量解决生产者—消费者问题

在Ubuntu上编写应用程序"pc.c"，解决经典的生产者—消费者问题，完成下面的功能： 建立一个生产者进程，N个消费者进程（N>1）； 用文件建立一个共享缓冲区； 生产者进程依次向缓冲区写入整数0,1,2,...,M，M>=500； 消费者进程从缓冲区读数，每次读一个，并将读出的数字从缓冲区删除，然后将本进程ID和+ 数字输出到标准输出； 缓冲区同时最多只能保存10个数。 一种可能的输出效果是：

```
10: 0
10: 1
10: 2
10: 3
10: 4
11: 5
11: 6
12: 7
……
11: 498
11: 499
```

其中ID的顺序会有较大变化，但冒号后的数字一定是从0开始递增加一的。

实现信号量###

Linux在0.11版还没有实现信号量，Linus把这件富有挑战的工作留给了你。如果能实现一套山寨版的完全符合POSIX规范的信号量，无疑 是很有成就感的。但时间暂时不允许我们这么做，所以先弄一套缩水版的类POSIX信号量，它的函数原型和标准并不完全相同，而且只包含如下系统调用：

```
sem_t *sem_open(const char *name, unsigned int value);
int sem_wait(sem_t *sem);
int sem_post(sem_t *sem);
int sem_unlink(const char *name); //sem_t是信号量类型，根据实现的需要自定义。
```

实验报告##

一、实验步骤###

1、ubuntu下的实验程序

程序中使用了系统定义了的以下几个信号量的函数： sem_open，sem_wait，sem_post，sem_getvalue，sem_unlink。 实验过程中遇到的难点主要是关于文件做缓冲区的问题，由于在程序中使用了同一个文件来供生产者与消费者共用，这样就有一个如何控制缓冲区的问题，即，如何能保证消费者按顺序取出生产者放在缓冲区中的数据？如何能控制生产者只使用10个位置而不导致文件增大？开始时试图使用一个静态变量来解决，证明不可行。又试图使用单独一个信号

实验数据

学习时间	289分钟
操作时间	125分钟
按键次数	0次
实验次数	8次
报告字数	22619字
是否完成	完成

评分

未评分

下一篇

篇

相关报告

操作系统原理与实践: 熟悉实验环境 实验报告

操作系统原理与实践: 熟悉实验环境 实验报告

操作系统原理与实践: 基于内核栈切换的进程切换 实验报告

操作系统原理与实践: 熟悉实验环境 实验报告

操作系统原理与实践: 信号量的实现和应用 实验报告

量来控制，发现也不可行。最终采用的方案是在消费者取出一个数据后即将缓冲区中的后继数据前移，并减小文件大小（即减小缓冲区长度），而生产者每次都将生产的数据放在文件尾处（即缓冲区的尾部）。这样就可以实现消费者从头部取数据，生产者从尾部放数据，从而实现缓冲区不增，按顺序提取。当然这里的缓冲区大小仅有10个，所以，移动缓冲区中的数据，并不会造成太多的性能损失，如果缓冲区很大，那么可能会影响程序的性能。本人也试图去查看系统中关于缓冲区的控制方法，但由于时间关系，没有对相关的代码进行研究，相信如果了解了系统中关于缓冲区的控制方法会有所帮助的。当然，这里减小文件大小使用了系统提供的ftruncate函数。如果系统没有该函数，则必须自己提供。 此外，实验程序如果让生产者提前退出，则会发生消费者进程异常，且还会多产生一个消费者进程，不知是何原因，未能解决，所以此次实验所有子进程均未结束。想大约可以通过加一个信号量来标识是否要退出来解决子进程退出问题，但由于时间关系未能解决，有时间会继续解决该问题。

实验程序代码：

```c
#include    <semaphore.h>
#include    <stdio.h>
#include    <stdlib.h>
#include    <unistd.h>
#include    <linux/kernel.h>
#include    <fcntl.h>
#include    <linux/types.h>
#include    <sys/stat.h>
#define BUFFERSIZE    10

sem_t *empty; // = sem_open("empty",O_CREAT | O_EXCL,0644,BUFFERSIZE);
sem_t *full; // = sem_open("full",O_CREAT | O_EXCL,0644,0);
sem_t *mutex; // = sem_open("mutex",O_CREAT | O_EXCL,0644,1);

int main(void)
{
    char err_desc[255];
    char empty_sem[]= "empty";
    char full_sem[]= "full";
    char mutex_sem[]="mutex";
    int in = open("pc.log", O_CREAT|O_RDWR, 0666);
    int of = open("pc.log",O_CREAT|O_RDWR,0666);
    int itemValue = -1;
    int fileLen,tmpValue,i,j;
    //文件大小设置为0;
    ftruncate(in,0);
    empty = sem_open(empty_sem,O_CREAT,0644,BUFFERSIZE);
    if(empty == SEM_FAILED)
    {
        printf("create semaphore empty error!\n");
        exit(1);
    }
    full = sem_open(full_sem,O_CREAT,0644,0);
    if(full == SEM_FAILED)
    {
        printf("create semaphore full error!\n");
        exit(1);
    }

    mutex = sem_open(mutex_sem,O_CREAT,0644,1);
    if(mutex == SEM_FAILED)
    {
        printf("create semaphore mutex error!\n");
        exit(1);
    }

    /* 测试用来查看定义是否真的成功
    sem_getvalue(empty,&tmpValue);
    printf("now empty's value = %d\n",tmpValue);
    sem_getvalue(mutex,&tmpValue);
    printf("now mutex's value = %d\n",tmpValue);
    sem_getvalue(full,&tmpValue);
    printf("now full's value = %d\n",tmpValue);
    */
    if(!fork())
    {
        //printf("producer process %u !now itemValue=%d\n",getpid(),itemValue);
        while(itemValue<500) //根据实验要求，这里是499就不输出500
        {
            itemValue++;
            //printf("now produce %d\n",itemValue);
            if(sem_wait(empty)!=0)
            {
                printf("in producer sem_wait(empty) error!\n");
                perror(err_desc);
                break;
            }
            //producer
            if(sem_wait(mutex)!=0)
            {
                printf("in producer sem_wait(mutex) error!\n");
                perror(err_desc);
                //printf("error msg : %s\n",err_desc);
                break;
            }
            //将数值写到文件尾部
            lseek(in,0,SEEK_END);
            write(in,&itemValue,sizeof(itemValue));
            if(sem_post(mutex)!=0)
            {
                printf("in producer sem_post(mutex) error!\n");
                perror(err_desc);
                break;
```

```
            }
            //发送full信号
            if(sem_post(full)!=0)
            {
                printf("in producer sem_post(full) error!\n");
                perror(err_desc);
                break;
            }

        } //producer process

        //如果此处退出生产者，则会导致消费者也异外退出且会多生成一个消费者原因不明。
        while(1)
            ;
        //close(in);
    }

    for(i=0; i < 5; i++)
    {
        if(!fork())
        {
            //printf("customer process(%u) begin to run!\n",getpid());
            while(1)
            {
                if(sem_wait(full)!=0)
                {
                    printf("in customer %u sem_wait(full) error!\n",getpid());
                    perror(err_desc);
                    break;
                }

                if(sem_wait(mutex)!=0)
                {
                    printf("in customer %u,sem_post(empty) error!",getpid());
                    perror(err_desc);
                    break;
                }
                //读取第一个数值并显示
                lseek(of,0,SEEK_SET);
                read(of,&itemValue,sizeof(itemValue));
                printf("%u:%d\n",getpid(),itemValue);
                //将其余数值前移，截断文件4个字节
                fileLen=lseek(in,0,SEEK_END);
                for(j=1;j<(fileLen/sizeof(itemValue));j++)
                {
                    lseek(in,j*sizeof(itemValue),SEEK_SET);
                    read(in,&tmpValue,sizeof(tmpValue));
                    lseek(in,(j-1)*sizeof(itemValue),SEEK_SET);
                    write(in,&tmpValue,sizeof(tmpValue));
                }
                ftruncate(in,fileLen-sizeof(itemValue));

                if(sem_post(mutex)!=0)
                {
                    printf("in customer %u,sem_post(empty) error!\n",getpid());
                    perror(err_desc);
                    break;
                }
                //发送empty
                if(sem_post(empty)!=0)
                {
                    //此时是否表明已经生产结束?
                    printf("in customer %u,sem_post(empty) error!\n",getpid());
                    perror(err_desc);
                }

            }
            close(of);
        }
    }

    //printf("now, main process exit!\n");
    //return
    return 0;
}
```

2、linux0.11下信号量的实现###

这个实验中也要实现系统调用，基本上可以说是第二个实验的复习，所以在这里也再次将系统调用的添加回顾一下。涉及的系统文件有以下几个：

- /inlude/linux/sys.h //添加系统调用表以及系统调用内核函数定义
- /kernel/systemcall.s //系统调用的宏定义
- /include/unistd.h //定义系统调用函数的对外接口以及系统调用号的宏定义，此次将sem_t的结构直接放在此文件中，没有定义semaphore.h文件。
- /kernel/Makefile //编译内核时将新系统调用文件添加进去
- /kernel/semaphore.c //实现信号量的函数文件。
- /kernel/include/fcntl.h //定义#define F_CHSIZE 8 */melon added 2015-6-24/*
- /kernel/fs/fcntl.c //修改fcntl函数，增加修改文件长度的实现

此次使用的信号量的结构定义：

```
#define NR_SEMAPHORE    64      //为简单实现，系统共可以使用的信号量为64个
#define NR_SEMANAME    255      //信号量的名字长度
typedef struct semaphore
{
    char sem_name[NR_SEMANAME];
    int value;
    struct task_struct * semp;
}sem_t;                 //这里为和ubuntu保持一致，使用了sem_t这个别名。
```

由于linux0.11中并没有实现ftruncate函数来修改文件长度，所以必须添加一个关于ftruncate的实现，由于仅在验证程序中使用，所以仅修改了fcntl函数，增加修改文件长度的代码。其使用方法为：

fcntl(fd,F_CHSIZE,n)，其中，fd为打开的文件号，n为文件长度。实现代码如下：

```
int sys_fcntl(unsigned int fd, unsigned int cmd, unsigned long arg)
{
    struct file * filp;

    if (fd >= NR_OPEN || !(filp = current->filp[fd]))
        return -EBADF;
    switch (cmd) {
        case F_DUPFD:
            return dupfd(fd,arg);
        case F_GETFD:
            return (current->close_on_exec>>fd)&1;
        case F_SETFD:
            if (arg&1)
                current->close_on_exec |= (1<<fd);
            else
                current->close_on_exec &= ~(1<<fd);
            return 0;
        case F_GETFL:
            return filp->f_flags;
        case F_SETFL:
            filp->f_flags &= ~(O_APPEND | O_NONBLOCK);
            filp->f_flags |= arg & (O_APPEND | O_NONBLOCK);
            return 0;
        case F_GETLK:    case F_SETLK:    case F_SETLKW:
            return -1;
        case F_CHSIZE:   //新增的修改文件长度代码
            current->filp[fd]->f_inode->i_size=arg;
            return 0;
        default:
            return -1;
    }
}
```

Semaphore.c文件代码：

```
#define __LIBRARY__
#include <unistd.h>
#include <stdarg.h>
#include <errno.h>
#include <asm/segment.h>
#include <asm/system.h>
#include <string.h>
#include <linux/kernel.h>
#include <linux/sched.h>

static sem_t semaphore_list[NR_SEMAPHORE]={{"",0,NULL}};   //全局的信号量数组初始化

/*
以下这两个函数，后来被弃用，发现如果使用，则调度的顺序会与使用系统sleep_on、wake_up函数时大不相同，而且有时会导致某个消费者不能被唤醒。由于实验程序中使用分支来判断是否睡眠，本人分析可能会导致一个唤醒信号会同时唤醒多个消费者，而多个消费者中可能只有一个能正确取到数据，其他消费者在此一轮中均无法取得数据，但程序并未发生异常错误。所以此次实验没有进一步研究关于唤醒问题。
*/
void sys_sem_sleep(struct task_struct **p)
{
    struct task_struct *tmp;

    if (!p)
        return;

    tmp = *p;
    *p = current;

    current->state = TASK_UNINTERRUPTIBLE;

    schedule();
    /*被唤醒后应该到这里，所以把前一个写回去，此处是错误的，因为不知哪一个进程会在此前修改指针，所以此处不应该写回去。 */
    *p=tmp;
}

void sys_sem_wakeup(struct task_struct **p)
{
    if (p && *p) {
        (**p).state=0;
        *p=NULL;
    }
}
/*
以下为信号量的实现函数，为方便调试，比试验指导上多加一个sem_getvalue函数，这样在必要时可以取出信号量的值查看是否正确。
*/
/*新建一个信号量*/
sem_t * sys_sem_open(const char * semname, int value)
{
    int i,cursem=-1;
    char curname[NR_SEMANAME];
    //int name_len=0;
    char* p=(char *)semname;
    char c;
    //取得名字
    for(i=0;i<NR_SEMANAME;i++)
    {
        c=get_fs_byte(p++);
        if(c=='\0')
            break;
        else
            curname[i]=c;//sema[cursem].sem_name[i]=c;
    }
    curname[i]='\0';

    //查看是否已经过定义，如果有，则直接返回已定义的，而且忽略值
    for(i=0; i<NR_SEMAPHORE; i++)
    {
        /*if(semaphore_list[i].sem_name[0] == '\0')
        {
            cursem=i;
            break;
        }
        else*/

        //printk("return semaphore, id is %d,the name is %s,the value is %d\n",i,semaphore_list[i].sem_name,semaphore_list[i].value);
        if(strcmp(curname,semaphore_list[i].sem_name)==0)
        {
            printk("return semaphore, id is %d,the name is %s,the value is %d\n",i,semaphore_list[i].sem_name,semaphore_list[i].value);
```

```
            return &semaphore_list[i];
        }

    }
    //取出未用的空位
    for(i=0; i<NR_SEMAPHORE; i++)
    {
        if(semaphore_list[i].sem_name[0] == '\0')
        {
            cursem=i;
            break;
        }
    }

    if(cursem==-1)
    {
        printk("now,no blank list,cursem=%d\n",cursem);
        return NULL;
    }
    i=0;
    for(;curname[i]!='\0';i++)
    {
        semaphore_list[cursem].sem_name[i]=curname[i];
    }
    semaphore_list[cursem].sem_name[i]='\0';
    semaphore_list[cursem].value=value;
    semaphore_list[cursem].semp=NULL;

    //printk("now semaphore_list[%d] is created! sem's value
=%d\n",cursem,semaphore_list[cursem].value);

    //
    return &semaphore_list[cursem];
}

/**/
int sys_sem_wait(sem_t * sem)
{
    cli();
    sem->value--;
    //printk("semaphore [%s] is wait! value =%d\n",sem->sem_name,sem->value);
    //printk("process [%u] wait !pointor is %u,semp is %u\n",current-
>pid,current,sem->semp);
    if(sem->value<0)
    {
        //printk("process [%u] sleep!pointor is %u,semp is %u\n",current-
>pid,current,sem->semp);
        //sem->semp=current;
        sleep_on(&(sem->semp));
        //sys_sem_sleep(&(sem->semp));
        //schedule();
    }
    sti();
    return 0;
}

/**/
int sys_sem_post(sem_t * sem)
{
    cli();
    sem->value++;
    //printk("semaphore [%s] is post! value =%d\n",sem->sem_name,sem->value);
    //printk("process [%u] post!pointor is %u,semp is %u\n",current-
>pid,current,sem->semp);
    if(sem->value<=0)
    {
        //wake_up
        //printk("process [%u] wakeup!current pointor is %u,semp is
%u\n",current->pid,current,sem->semp);
        wake_up(&(sem->semp));
        //sys_sem_wakeup(&(sem->semp));
    }
    sti();
    return 0;
}

int sys_sem_getvalue(sem_t * sem)
{
    //
    if(sem != NULL)
        return sem->value;
    else
        return -1;
```

```
}

/*删除一个信号量*/
int sys_sem_unlink(const char * name)
{
    int i;//,cursem=-1;
    char curname[NR_SEMANAME];
    //int name_len=0;
    char* p=(char*)name;
    char c;
    //取得名字
    for(i=0;i<NR_SEMANAME;i++)
    {
        c=get_fs_byte(p++);
        if(c=='\0')
            break;
        else
            curname[i]=c;//sema[cursem].sem_name[i]=c;
    }
    curname[i]='\0';
    //判断是否有同名的存在
    for(i=0; i<NR_SEMAPHORE; i++)
    {
        if(strcmp(curname,semaphore_list[i].sem_name)==0)
        {
            semaphore_list[i].sem_name[0]='\0';
            semaphore_list[i].value=0;
            semaphore_list[i].semp=NULL;
            return 0;
        }
    }
    //
    return -1;
}
```

linux0.11中使用的验证程序：

```c
#define __LIBRARY__
#include    <stdio.h>
#include    <stdlib.h>
#include    <unistd.h>
#include    <linux/kernel.h>
#include    <fcntl.h>
#include    <sys/types.h>
#define BUFFERSIZE    10

static _syscall2(sem_t *,sem_open,const char *,name,int,value);
static _syscall1(int,sem_post,sem_t *,sem);
static _syscall1(int,sem_wait,sem_t *,sem);
static _syscall1(int,sem_getvalue,sem_t *,sem);
static _syscall1(int,sem_unlink,const char*,name);

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

int main(void)
{

    char err_desc[255];
    char empty_sem[]= "empty";
    char full_sem[]= "full";
    char mutex_sem[]="mutex";

    int in = open("pc.log", O_CREAT|O_TRUNC|O_RDWR, 0666);
    int of = open("pc.log", O_CREAT|O_TRUNC|O_RDWR, 0666);
    int log = open("pclog.log", O_CREAT|O_TRUNC|O_RDWR, 0666);
    char buflog[255];   //用来格式化输出到文件中的字符串缓冲区
    int tmp;

    int itemValue = -1;
    int fileLen,tmpValue,i,j;
    //新添加的修改文件长度的函数调用
    if(fcntl(in,F_CHSIZE,0)!=0)
    {
        printf("in main process. ftruncate error!\n");
        close(in);
        close(of);
        return 0;
    }
    empty = sem_open(empty_sem,BUFFERSIZE);

    if(empty == NULL)
    {
        printf("create semaphore empty error!\n");
        exit(1);
    }

    full = sem_open(full_sem,0);
    if(full ==NULL)
    {
        printf("create semaphore full error!\n");
        exit(1);
    }

    mutex = sem_open(mutex_sem,1);
    if(mutex == NULL)
    {
        printf("create semaphore mutex error!\n");
        exit(1);
    }

    tmpValue=sem_getvalue(empty);
    printf("now empty's value = %d\n",tmpValue);

    tmpValue=sem_getvalue(mutex);
    printf("now mutex's value = %d\n",tmpValue);

    tmpValue=sem_getvalue(full);
    printf("now full's value = %d\n",tmpValue);

    if(!fork())
    {
        printf("producer process %u !now itemValue=%d\n",getpid(),itemValue);

        while(itemValue<50)
        {
            itemValue++;

            if(sem_wait(empty)!=0)
```

```
            {
                printf("in producer sem_wait(empty) error!\n");
                perror(err_desc);
                break;
            }


            if(sem_wait(mutex)!=0)
            {
                printf("in producer sem_wait(mutex) error!\n");
                perror(err_desc);

                break;
            }

            fileLen=lseek(in,0,SEEK_END);
            write(in,&itemValue,sizeof(itemValue));

            if(sem_post(mutex)!=0)
            {
                printf("in producer sem_post(mutex) error!\n");
                perror(err_desc);
                break;
            }

            if(sem_post(full)!=0)
            {
                printf("in producer sem_post(full) error!\n");
                perror(err_desc);
                break;
            }

        }
        /*
        tmpValue=sem_getvalue(empty);
        printf("now empty's value = %d\n",tmpValue);

        tmpValue=sem_getvalue(mutex);
        printf("now mutex's value = %d\n",tmpValue);

        tmpValue=sem_getvalue(full);
        printf("now full's value = %d\n",tmpValue);
        */
        while(1)
            ;

        close(in);
    }

    for(i=0; i < 5; i++)
    {
        if(!fork())
        {
            printf("customer process(%u) begin to run!\n",getpid());

            while(1)
            {

                if(sem_wait(full)!=0)
                {
                    printf("in customer %u sem_wait(full) error!\n",getpid());
                    perror(err_desc);
                    break;
                }

                if(sem_wait(mutex)!=0)
                {
                    printf("in customer %u,sem_post(empty) error!",getpid());
                    perror(err_desc);
                    break;
                }

                lseek(of,0,SEEK_SET);
                read(of,&itemValue,sizeof(itemValue));
                /*printf("%u:%d\n",getpid(),itemValue);*/
                /*
                由于虚拟机中的linux0.11在输出时会出现乱码，而修改输出到文件中时会发生无法正确输出实验结果，因此此处修改为实验结果直接输出到一个记录文件中。
                */
                lseek(log,0,SEEK_END);
                sprintf(buflog,"%u:%d\n",getpid(),itemValue);
                write(log,&buflog,sizeof(char)*strlen(buflog));
```

```
                //修改缓冲区长度
                fileLen=lseek(in,0,SEEK_END);
                for(j=1;j<(fileLen/sizeof(itemValue));j++)
                {
                    lseek(in,j*sizeof(itemValue),SEEK_SET);
                    read(in,&tmpValue,sizeof(tmpValue));
                    lseek(in,(j-1)*sizeof(itemValue),SEEK_SET);
                    write(in,&tmpValue,sizeof(tmpValue));
                }

                if(fcntl(in,F_CHSIZE,fileLen-sizeof(tmpValue))!=0)
                {
                    printf("ftruncate error!\n");
                    break;
                }

                if(sem_post(mutex)!=0)
                {
                    printf("in customer %u,sem_post(empty) error!\n",getpid());
                    perror(err_desc);
                    break;
                }

                if(sem_post(empty)!=0)
                {

                    printf("in customer %u,sem_post(empty) error!\n",getpid());
                    perror(err_desc);

                }
            }

            close(of);
        }
    }

    printf("now, main process exit!\n");

    return 0;
}
```

二、实验总结###

使用信号量来同步进程必须要考虑清楚退出机制问题，也就是信号量的释放时机如何选择以及如何安排，否则会造成信号量的占用无法释放。由于时间问题（本人水平有限，仅实现信号量就花费了一周多的业余时间，遗留问题在以后会继续研究解决）此问题在本次实验中并未解决。

ubuntu系统下的信号量是可以在线程中共享使用的，默认也可以用于进程同步。所以以此看来信号量就是内核的一个全局数据，如何使用取决于程序设计者。但全局数据也是全局资源，必然是有限的，所以使用信号量时应该慎重考虑周全。

另：一个有趣的问题是，由于实验程序未能正确结束，所以得到实验结果后6个子进程均在后台运行，且生产者进程状态为0，其余消费者进程均为2。但经实验，在ubuntu15.04(64位)下，6个子进程均能得到结束，也就是说在15.04版本下得出实验结果后6个子进程均被系统收回了。但在14.04与12.04下均不做收回。不知是不是15.04对进程的收回机制做了变动。

三、实验问题###

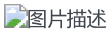
完成实验后，在实验报告中回答如下问题： 在pc.c中去掉所有与信号量有关的代码，再运行程序，执行效果有变化吗？为什么会这样？ 答：去掉进程同步结构，会造成输出混乱。因为无法保证按顺序取数据，写数据时机也不能保证，可能会出现竞争（即同时多进程对文件进行读写）而死锁。但本次实验未发生，大概是实验的次数不足够多。 实验的设计者在第一次编写生产者——消费者程序的时候，是这么做的：

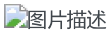
```
Producer()
{
    P(Mutex);  //互斥信号量
    生产一个产品item;
    P(Empty);  //空闲缓存资源
    将item放到空闲缓存中;
    V(Full);  //产品资源
    V(Mutex);
}

Consumer()
{
    P(Mutex);
    P(Full);
    从缓存区取出一个赋值给item;
    V(Empty);
    消费产品item;
    V(Mutex);
}
```

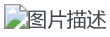
这样可行吗？如果可行，那么它和标准解法在执行效果上会有什么不同？如果不可行，那么它有什么问题使它不可行？

这样做不可行，如果消费者在mutex上睡眠，当被唤醒时后运行后会再次修改mutex值，而无法保证mutex始终在0、1之间变换，这样就不能保证对临界资源的保护。

实验截图##







最后补充 主动结束全部进程###

在以上部分的验证程序中，所有子进程均无法正常结束，现通过增加一个退出信号量来主动结束全部子进程。在ubuntu14.04下验证通过。但未在linux0.11下验证，想来也应该是可以的，所以偷一次懒吧！！^_^

```
#include <semaphore.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include    <linux/kernel.h>
#include    <fcntl.h>
#include    <linux/types.h>
#include    <sys/stat.h>
#define BUFFERSIZE    10

sem_t *empty; // = sem_open("empty",O_CREAT | O_EXCL,0644,BUFFERSIZE);
sem_t *full; // = sem_open("full",O_CREAT | O_EXCL,0644,0);
sem_t *mutex; // = sem_open("mutex",O_CREAT | O_EXCL,0644,1);
sem_t *closetime;

int main(void)
{

    char err_desc[255];
    char empty_sem[]= "empty";
    char full_sem[]= "full";
    char mutex_sem[]="mutex";
    char closetime_sem[]="closetime";

    int in = open("pc.log", O_CREAT|O_RDWR, 0666);
    int of = open("pc.log",O_CREAT|O_RDWR,0666);

    int itemValue = -1;
    int fileLen,tmpValue,i,j,closeValue,fullValue;

    //文件大小设置为0;
    ftruncate(in,0);
    empty = sem_open(empty_sem,O_CREAT,0644,BUFFERSIZE);

    if(empty == SEM_FAILED)
    {
        printf("create semaphore empty error!\n");
        return 1;
    }


    full = sem_open(full_sem,O_CREAT,0644,0);
    if(full == SEM_FAILED)
    {
        printf("create semaphore full error!\n");
        return 1;
    }

    mutex = sem_open(mutex_sem,O_CREAT,0644,1);
    if(mutex == SEM_FAILED)
    {
        printf("create semaphore mutex error!\n");
        return 1;
    }

    closetime = sem_open("closetime",O_CREAT,0644,0);
    if(closetime == SEM_FAILED)
    {
        printf("create semaphore closetime error!\n");
        return 1;
    }

    /*
    sem_getvalue(empty,&tmpValue);
    printf("now empty's value = %d\n",tmpValue);
    sem_getvalue(mutex,&tmpValue);
    printf("now mutex's value = %d\n",tmpValue);
    sem_getvalue(full,&tmpValue);
    printf("now full's value = %d\n",tmpValue);
    sem_getvalue(closetime,&tmpValue);
    printf("now closetime's value = %d\n",tmpValue);
    */

    if(!fork())
    {
        //printf("producer process %u !now itemValue=%d\n",getpid(),itemValue);

        while(itemValue<499)
        {
            itemValue++;
            //printf("now produce %d\n",itemValue);
            if(sem_wait(empty)!=0)
```

```
            {
                printf("in producer sem_wait(empty) error!\n");
                perror(err_desc);
                break;
            }

            //producer
            if(sem_wait(mutex)!=0)
            {
                printf("in producer sem_wait(mutex) error!\n");
                perror(err_desc);
                break;
            }
            //将数值写到文件尾部
            lseek(in,0,SEEK_END);
            write(in,&itemValue,sizeof(itemValue));

            //发送full信号
            if(sem_post(full)!=0)
            {
                printf("in producer sem_post(full) error!\n");
                perror(err_desc);
                break;
            }
            if(sem_post(mutex)!=0)
            {
                printf("in producer sem_post(mutex) error!\n");
                perror(err_desc);
                break;
            }

        } //producer process

        //now the producer finish the work, so post the signal to close all
process.
        sem_getvalue(closetime,&closeValue);
        if(closeValue==0)
        {
            for(i=0;i<5;i++)
            {
                if(sem_post(closetime)!=0)
                {
                    printf("in producer sem_post(closetime) error!\n");
                    perror(err_desc);
                    break;
                }
            }
        }

        //check the closetime's value
        sem_getvalue(closetime,&closeValue);
        while(closeValue>0)
        {

            sem_getvalue(full,&fullValue);
            if(fullValue<=0) //if full's value lessthan zero, so post the full
            {
                sem_post(full);
            }
            sem_getvalue(closetime,&closeValue);
        }

        printf("now all process exit.\n");

        //close all semaphores
        sem_unlink("empty");
        sem_unlink("full");
        sem_unlink("mutex");
        sem_unlink("closetime");

        close(of);
        close(in);

        exit(0);
    }

    for(i=0; i < 5; i++)
    {
        if(!fork())
        {
            //printf("customer process(%u) begin to run!\n",getpid());

            while(1)
```

```
            {

                //customer
                if(sem_wait(full)!=0)
                {
                    printf("in customer %u sem_wait(full) error!\n",getpid());
                    perror(err_desc);
                    break;
                }

                //check in the closttime's value
                sem_getvalue(closetime,&tmpValue);
                if(tmpValue>0) //it is time for exit!
                {

                    fileLen=lseek(in,0,SEEK_END);
                    //printf("now customer [%u] is ready to exit!
filelen=%d,closetime's value=%d\n",getpid(),fileLen,tmpValue);
                    if(fileLen==0) //now it is time to exit!
                    {
                        if(sem_wait(closetime)!=0)
                        {
                            printf("in customer [%u], sem_wait(closetime)
error!\n");
                            perror(err_desc);
                        }
                        break;
                    }
                }

                if(sem_wait(mutex)!=0)
                {
                    printf("in customer %u,sem_post(empty) error!",getpid());
                    perror(err_desc);
                    break;
                }
                //读取第一个数值并显示
                lseek(of,0,SEEK_SET);
                read(of,&itemValue,sizeof(itemValue));
                printf("%u:%d\n",getpid(),itemValue);
                //将其余数值前移，截断文件4个字节
                fileLen=lseek(in,0,SEEK_END);

                for(j=1;j<(fileLen/sizeof(itemValue));j++)
                {
                    lseek(in,j*sizeof(itemValue),SEEK_SET);
                    read(in,&tmpValue,sizeof(tmpValue));
                    lseek(in,(j-1)*sizeof(itemValue),SEEK_SET);
                    write(in,&tmpValue,sizeof(tmpValue));
                }
                ftruncate(in,fileLen-sizeof(itemValue));

                //发送empty
                if(sem_post(empty)!=0)
                {
                    //此时是否表明已经生产结束?
                    printf("in customer %u,sem_post(empty) error!\n",getpid());
                    perror(err_desc);

                }

                if(sem_post(mutex)!=0)
                {
                    printf("in customer %u,sem_post(empty) error!\n",getpid());
                    perror(err_desc);
                    break;
                }
            }

            //close(of);
            exit(0);
        }
    }

    printf("now, main process exit!\n");
    return 0;
}
```

最后的补充：在我机器上（版本14.04 - 3.16）显示正常，但在实验楼的虚拟机中却会有多余的显示出现经查看版本为3.13，原因不明。