浙江大学实验报告

课程名称：操作系统分析及实验 实验类型：综合型/设计性

实验项目名称：实验 1 同步互斥和 Linux 内核模块

学生姓名：胡亮泽 专业：计算机科学与技术 学号：3120102116

电子邮件地址（必须）：huliangze@yeah.net 手机：18868103152

实验日期：2014 年 12 月 12 日

一、实验目的

学习使用 Linux 的系统调用和 pthread 线程库编写程序。

充分理解对共享变量的访问需要原子操作。

进一步理解、掌握操作系统进程和线程概念、进程或线程的同步与互斥。

学习编写多线程程序，掌握解决多线程的同步与互斥问题。

学习 Linux 是如何实现模块机制的，掌握如何编写模块程序并进一步掌握内核模块的机理。

通过对 Linux 系统中进程的遍历，进一步理解操作系统进程概念和进程结构。

二、实验内容

1. 程序 P1-1.c 使用 pthread 线程库创建两个线程，两个线程都要访问共享变量 counter，用 gcc 编译程序：

   gcc –o P1-1 P1-1.c –lpthread

   按照源程序的逻辑，两个线程各循环 5 次，每次循环对 counter 变量加 1，最后打印的结果应该是 10。在你的系统中，编译并运行程序，观察运行结果

2. 编写程序实现生产者-消费者经典进程同步问题。使用Linux的Pthread线程库，创建3个生产者线程和5个消费者线程。生产者线程计算当前的时间，把时间、第几次计算的序号（循环次数）和线程ID作为一个消息，把消息放入缓冲区，消费者线程从缓冲区读出一个消息并显示消息。缓冲区大小为8个，每个生产者线程生产5个消息，每个消费者线程消费3个消息，即生产15个消息和消费15个消息。

3. 编写一个Linux的内核模块，其功能是遍历进程，要求输出系统中：每个进程的名字、进程pid、进程的状态、父进程的名字；统计系统中进程个数，统计系统中TASK_RUNNING、TASK_INTERRUPTIBLE、TASK_UNINTERRUPTIBLE、

TASK_ZOMBIE、TASK_STOPPED等状态进程的个数。

三、 主要仪器设备（必填）

VAIO E 系列，Ubuntu 14.04，

3.13.0-40-generic #69-Ubuntu SMP Thu Nov 13 17:56:26 UTC 2014 i686 i686 i686 GNU/Linux

四、操作方法和实验步骤

- **第一题**
  1. 用 vim 编辑相应的代码后用 gcc 编译并运行，可以查看 count 以及其他输出结果。
  2. 改变源程序中的 usleep 参数并重复编译运行的过程，查看输出结果的变化。
  3. 修改源程序，加入互斥锁，重复编译运行过程，观察结果变化
- **第二题**

编写程序，源代码如下：

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>
#include <time.h>
#include <semaphore.h>

#define BUFFER_SIZE 8

typedef struct{
    pthread_t id;
    int loop_time;
    struct tm *time;

}item;

item buffer[BUFFER_SIZE];
sem_t empty, full;
pthread_mutex_t mutex;

int in = 0, out = 0;

void  Producer(void *arg);
void  Consumer(void *arg);

int  main(int argc, char *argv[])
{
    pthread_t producer[3];
    pthread_t consumer[5];


    if(sem_init(&full,0,0) != 0 || sem_init(&empty,0,8) != 0) {
        printf("Semaphore initialization failed!");
        exit(1);
    }
    if(pthread_mutex_init(&mutex,NULL) != 0){
```

```
        printf("Mutex initialization failed!");
        exit(1);
    }

    //创建生产者进程
    pthread_create(&producer[0],NULL,(void *)Producer, NULL);
    pthread_create(&producer[1],NULL,(void *)Producer, NULL);
    pthread_create(&producer[2],NULL,(void *)Producer, NULL);
    //创建消费者进程
    pthread_create(&consumer[0],NULL,(void *)Consumer, NULL);
    pthread_create(&consumer[1],NULL,(void *)Consumer, NULL);
    pthread_create(&consumer[2],NULL,(void *)Consumer, NULL);
    pthread_create(&consumer[3],NULL,(void *)Consumer, NULL);
    pthread_create(&consumer[4],NULL,(void *)Consumer, NULL);



pthread_join(producer[0],NULL); /*等待生产者线程结束*/
pthread_join(producer[1],NULL);
pthread_join(producer[2],NULL);

pthread_join(consumer[0],NULL); /*等待消费者线程结束*/
pthread_join(consumer[1],NULL);
pthread_join(consumer[2],NULL);
pthread_join(consumer[3],NULL);
pthread_join(consumer[4],NULL);


sem_destroy(&full);
sem_destroy(&empty);
pthread_mutex_destroy(&mutex);
    exit(0);
}

void  Producer(void *arg) /*生产者线程执行代码*/
{
    int i;
    time_t ltime;
    item nextProduced;
    for(i = 1 ; i <= 5 ; i++ ){
        usleep(750000);
        sem_wait(&empty);
        pthread_mutex_lock(&mutex);
        nextProduced.loop_time = i;
        time(&ltime);
        nextProduced.time = localtime(&ltime);
        nextProduced.id   = pthread_self();

        buffer[in] = nextProduced;
        in = (in+1) % BUFFER_SIZE;
        pthread_mutex_unlock(&mutex);
        sem_post(&full);
    }

}

void Consumer(void *arg) /*消费者线程执行代码*/
{
    int i;
    item nextConsumed;
```

```
    for(i=1;i<=3;i++){
        usleep(750000);
        sem_wait(&full);
        pthread_mutex_lock(&mutex);
        nextConsumed = buffer[out];
        out = (out + 1) % BUFFER_SIZE;
        printf("ID 为 %lu 的 生 产 者 线 程 在
日，%d:%d:%d\n",nextConsumed.id,nextConsumed.loop_time,1900+nextConsumed.time-
        pthread_mutex_unlock(&mutex);
        sem_post(&empty);
        }

}
```

程序主要的设计思路是：

1. 定义 2 个信号量 full 和 empty 以实现缓冲区 buffer 的变量移除和放入，控制生产者进程和消费者进程访问缓冲区存取数据的量。
2. 定义变量 IN,和 OUT，用以标记当前存取的数据的 index 标志
3. 定义数据类型 item,主要包括系统时间,生产者线程 ID 和生产者循环次数等信息

```
typedef struct{
pthread_t id;
int loop_time;
struct tm *time;

}item;
```

4. 定义互斥锁，将缓冲区数据访问和改变 in,out 等共享内容的代码段定义成临界区，实现互斥访问，比如生产者者进程中

```
sem_wait(&empty);
    pthread_mutex_lock(&mutex);
    nextProduced.loop_time = i;
    time(&ltime);
    nextProduced.time = localtime(&ltime);
    nextProduced.id   = pthread_self();

    buffer[in] = nextProduced;
    in = (in+1) % BUFFER_SIZE;
    pthread_mutex_unlock(&mutex);
    sem_post(&full);
```

   通过信号量 empty 控制数据的存入，当缓冲区未满时可以访问，接下来是通过 sem_post 的函数给信号量 full 加 1，表明又有一个数据写入了。
5. 消费者线程执行函数和生产者线程结构差不多，不同的是消费者线程用信号量 full 控制访问 buffer 并在退出临界区后用 sem_post 方法给 empty 加 1，表明又有一个空间被释放了。
6. 最后在 main 线程中创建 3 个生产者线程和 5 个消费者线程并等待执行。
7. 编译程序并执行，查看结果。

- **第三题**

1. 编写 Makefile 文件和 P1-3.c 文件
2. 通过 make 命令将生成 P1-3.ko 模块
3. 通过 insmod 命令将 P1-3.ko 模块装载进内核并通过 lsmod 命令查看系统内核

模块

```
lk@Wholanz:~$ sudo insmod P1-3.ko
lk@Wholanz:~$ lsmod
Module                  Size  Used by
P1_3                   12399  0
```

可以发现 P1_3 模块已经成功装入了内核

4. 通过 dmesg 命令查看系统日志文件
5. 通过 rmmod 命令将 P1-3.ko 模块从内核中卸载并通过 lsmod 命令查看系统内核模块

```
lk@Wholanz:~$ sudo rmmod P1-3
lk@Wholanz:~$ lsmod
Module                  Size  Used by
vmhgfs                 48609  1
vmw_vmci               60731  1 vmhgfs
```

可以发现模块已经成功移除

Makefile 文件和 P1-3.c 文件已经打包。主要的思路如下：

P1-3.c 主要通过 next_task 宏遍历进程，并通过指向数据结构中的 state 读取进程状态，并给相应的状态技术加 1，以此实现进程计数，比如：

```
for (p = &init_task; (p = next_task(p)) != &init_task;)
    {

        switch(p->state)
        {
            case TASK_RUNNING:
                nr_running++;
                pstate = "TASK_RUNNING";
                break;
```

该部分实现了 TASK_RUNNING 状态进程的计数

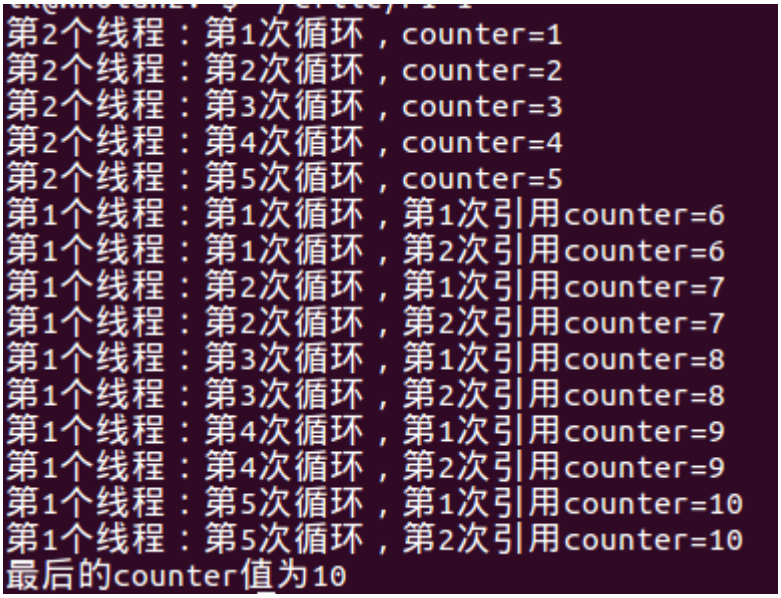
最后利用 printk 函数输出相应的信息。

五、实验结果和分析

- **实验一**
  1. 运行程序，最后的结果为：

发现运行结果为 6，第一个线程并且第一次循环中的两次引用的 counter 值不同，说明在进程一睡眠的过程中进程二改变了 count 值

2. 加入互斥锁后的运行结果为



发现运行结果为 10，已经正确实现了互斥访问临界区

相关问题：

1) 假设程序中 LINE A 行和 LINE B 行之间有很多代码，并多次引用 counter 的值，线程的运行结果会如何？

结果完全不能确定 count 的值，因为运行过程中将会不时地切换到进程 2 运行代码，从而改变 count 的值，因此每一次引用的 count 的值都是不确定的。

2) 请你多次调整两个线程中 usleep 函数的参数（即调整线程的睡眠时间），最后打印的 counter 值会出现什么样的变化？

进程 1 参数	进程 2 参数	Counter 值
30	800	5
30	1000	5
300	10	6
1000	10	6

以上是一些改变参数后的输出结果，经过测试发现最后 count 的值可能为 5 和 6。而 counter 最后的值取决于最后运行的进程。如果为进程 1，则 counter 的值为 6；否则 counter 的值为 5。

结果不是 10，因为最后的 count 值只和 val 变量有关，不管在 printf 函数执行前有无抢占， count 值有无短暂的变化，最后的 count 值都会被重新赋值成 val,因此两个进程中 val 的值变化都是连续的，由于进程 2 后创建，具有更高的优先级，所以进程 2 先运行，val 第一次赋值为 1，进程 1 的 val 第一次赋值为 2，所以最后进程 1 的 val 值一定为 6，而进程 2 的 val 值最后一定为 5，.

因此最后的 count 值取决于最后一次 count 是被哪个进程中的 val 赋值的，如果是进程 1，就是 6，如果是进程 2，就是 5.

3) 通过这个程序例子,你得出什么结论?

互斥锁在进程调度中对访问临界区的作用很大，如果不使用互斥锁，不仅仅会造成结果和逻辑的不同，甚至会造成结果的不确定性。

因此，学会如何正确的在进程中使用互斥锁十分重要。

- **实验二**

运行结果如下：

```
lk@Wholanz:~$ ~/cfile/pro_cons
ID为3059452736的生产者线程在第1次循环产生这条消息，消息时间为2014年12月16日，10:48:10
ID为3067845440的生产者线程在第1次循环产生这条消息，消息时间为2014年12月16日，10:48:10
ID为3076238144的生产者线程在第1次循环产生这条消息，消息时间为2014年12月16日，10:48:10
ID为3067845440的生产者线程在第2次循环产生这条消息，消息时间为2014年12月16日，10:48:11
ID为3076238144的生产者线程在第2次循环产生这条消息，消息时间为2014年12月16日，10:48:11
ID为3059452736的生产者线程在第2次循环产生这条消息，消息时间为2014年12月16日，10:48:11
ID为3067845440的生产者线程在第3次循环产生这条消息，消息时间为2014年12月16日，10:48:11
ID为3076238144的生产者线程在第3次循环产生这条消息，消息时间为2014年12月16日，10:48:11
ID为3059452736的生产者线程在第3次循环产生这条消息，消息时间为2014年12月16日，10:48:11
ID为3067845440的生产者线程在第4次循环产生这条消息，消息时间为2014年12月16日，10:48:12
ID为3076238144的生产者线程在第4次循环产生这条消息，消息时间为2014年12月16日，10:48:12
ID为3059452736的生产者线程在第4次循环产生这条消息，消息时间为2014年12月16日，10:48:12
ID为3067845440的生产者线程在第5次循环产生这条消息，消息时间为2014年12月16日，10:48:13
ID为3076238144的生产者线程在第5次循环产生这条消息，消息时间为2014年12月16日，10:48:13
ID为3059452736的生产者线程在第5次循环产生这条消息，消息时间为2014年12月16日，10:48:13
```

可以发现结果中读出不同消息的生产者线程 ID，以及在该线程的第几次循环中被产生，和产生的时间。

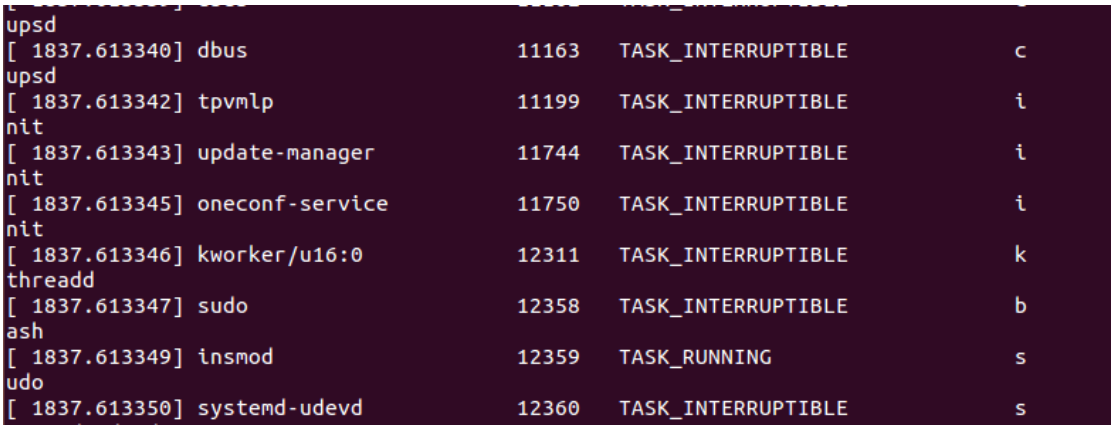
比如第一条消息，就是 ID 为 3059452736 在第 1 次循环，10:48:10 产生的消息

调整参数，usleep 参数，可以使消息产生的时间差变大。

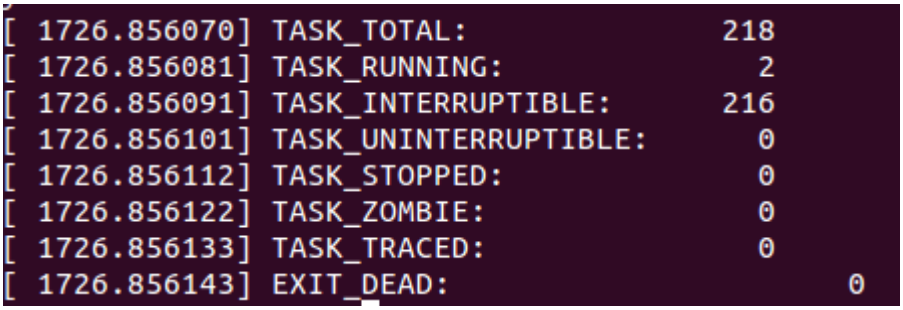
遇到的困难：主要困难是在调整 usleep 参数方面，由于调度过快，最后输出的消息时间差别很小。因此需要将参数变大，使得输出时间出现差距。

- **实验三**

读取系统日志文件后的结果是：



该图是日志中的一部分，展示了某即个进程的运行状态，id，父进程等信息
比如 dbus 进程的 ID 为 11163，运行状态为 TASK_INTERRUPTIBLE,父进程名为 cupsd



该图为最后输出的统计结果，总共有 188 个进程，其中 TASK_RUNNING 状态的进程有 2 个，TASK_INTERRUPTIBLE 状态的进程有 186 个，其余状态的没有。

主要困难：对内核模块的编程不熟悉，以及对 Makefile 文件如何编写和使用不熟悉。通过查阅相关资料后完成了实验。

五、讨论、心得

1. 这次的学习让我更加深入的了解了同步互斥锁的运行机制，通过实验一中是否使用互斥锁前后结果的不同之处，真正了解到了互斥锁在线程运行过程中所起到的作用，并学会了如何正确的使用互斥锁。
2. 在实验二中正确的使用信号量和互斥锁可以正确的保证缓冲区的使用，了解到互斥锁其实是一个特殊的信号量。由此我了解到了信号量存在的意义及其在实际编程中的重大作用。
3. 实验三中通过一个简单的内核模块的编写，使我大致了解了内核模块的意义和作用，以及内核模块化对内核维护所起到的作用，在操作中了解到了 Makefile 的作用以及掌握了 make clean 以及 make 等命令的使用方法以及 Makefile 的书写格式。并对内核模块的编程格式有了一定程度的了解。通过查看输出系统日志了解了内核模块的工作机制。