“操作系统原理与实践”实验报告

信号量的实现和应用

本实验所有代码在这里： http://git.shiyanlou.com/gaoyuanhezhihao/shiyanlou_cs115/src/master/Lab6

首先编写信号量测试用户代码：

```
#include <semaphore.h>
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/shm.h>
#include <fcntl.h>
#define FILECELLSIZE 3
FILE *fpWork;
FILE *fpFakeTerminal;
int main int char
{

    sem_t sem_empty,*pSem_empty=&sem_empty;
    sem_t sem_full,*pSem_full=&sem_full;
    sem_t sem_mutex,*pSem_mutex=&sem_mutex; //only one process can handle the file at
thsame time
    printf ("main start" );
    fflush( stdout );
    int iwait= 0;
    int WriteReadIndex[ 2]={ 0, 0};
    //系统中可能已经存在着三个信号量，把它们清楚掉
    sem_unlink( "empty" );
    sem_unlink( "full" );
    sem_unlink( "mutex" );
    //建立这三个型号量
    pSem_empty=sem_open( "empty" ,O_CREAT | O_EXCL,0600 , 10 );
    pSem_full=sem_open( "full" ,O_CREAT | O_EXCL,0600 , 0);
    pSem_mutex=sem_open( "mutex" ,O_CREAT | O_EXCL,0600 , 1);
    /*打开作为缓存的文件。文件的书写格式是：写进程从文件的4号字节开始顺序写入内容，读进程
    也从4号字节开始顺序读进程内容。为了不同的写进程之间同步写的位置，文件的第0，1字节用来
    保存当前写的位置。同理，为了读进程之间能够同步文件的第2，3用来保存读的位置（这里也可以
    使用一个信号量来同步读写位置）。为了简化，这里并没有完全按照实验指导中的要求，把读过的
    内容删除掉。*/
    fpWork = fopen( "work.dat" , "wb+" );
    fwrite(WriteReadIndex, 2*sizeof ( int ), 1,fpWork);
    fflush(fpWork);
    if (!fork())
    {
        //this is subprocess of writing data
        //写进程
        printf ("writer process start\n" );
        fflush( stdout );
        int i= 0;
        while (i <= 50)
        {
            int FileWriteIndex = 0;
            ++i; //produce a item
            sem_wait(pSem_empty); //P(Empty)
            sem_wait(pSem_mutex); //P(Mutex)

            //#1读取文件中记录的写序号（接下来应该写的位置）
            fseek(fpWork, 0*sizeof ( int ),SEEK_SET); //把文件指针指向文件第一个字节
            fread(&FileWriteIndex, sizeof ( int ), 1,fpWork);
            //#2写入
            fseek(fpWork,(FileWriteIndex+ 2)* sizeof ( int ),SEEK_SET);
            fwrite(&i, sizeof ( int ), 1,fpWork);
            fflush(fpWork);
            printf ("Write Index:%d,%d is writed\n" ,FileWriteIndex,i);
            fflush( stdout );
            //#3更新写序号
            fseek(fpWork, 0*sizeof ( int ),SEEK_SET);
            FileWriteIndex++;
            fwrite(&FileWriteIndex, sizeof ( int ), 1,fpWork);
            fflush(fpWork);

            sem_post(pSem_mutex); //V(Mutex)
            sem_post(pSem_full); //V(Full)
        }
        while ( 1);

    }
    if (!fork())
    {
        //this is the first subprocess of reading data
        //第一个写进程
        int FileReadIndex_1 = 0;
        while ( 1)
        {
            int a= 0;
            sem_wait(pSem_full);
            sem_wait(pSem_mutex);
            //#1
            fseek(fpWork, 1*sizeof ( int ),SEEK_SET);
            fread(&FileReadIndex_1, sizeof ( int ), 1,fpWork);
            //#2
            fseek(fpWork,(FileReadIndex_1+ 2)* sizeof ( int ),SEEK_SET);
            fread(&a, sizeof ( int ), 1,fpWork);
            printf ("ReadIndex1:%d,%d:%d\n" ,FileReadIndex_1,getpid(),a);
            //#3
            fseek(fpWork, 1*sizeof ( int ),SEEK_SET);
            FileReadIndex_1++;
            fwrite(&FileReadIndex_1, sizeof ( int ), 1,fpWork);
            fflush( stdout );
            fflush(fpWork);
            sem_post(pSem_mutex);
            sem_post(pSem_empty);
        }
    }
    if (!fork())
    {
        //this is the second subprocess of reading data
        //第二个写进程
        int FileReadIndex_2 = 0;
        while ( 1)
        {
            int j= 0;
            sem_wait(pSem_full);
            sem_wait(pSem_mutex);
            //#1
            fseek(fpWork, 1*sizeof ( int ),SEEK_SET);
            fread(&FileReadIndex_2, sizeof ( int ), 1,fpWork);
            //#2
            fseek(fpWork,(FileReadIndex_2+ 2)* sizeof ( int ),SEEK_SET);
            fread(&j, sizeof ( int ), 1,fpWork);
            printf ("ReadIndex2:%d,%d:%d\n" ,FileReadIndex_2,getpid(),j);
            //#3
            fseek(fpWork, 1*sizeof ( int ),SEEK_SET);
            FileReadIndex_2++;
            fwrite(&FileReadIndex_2, sizeof ( int ), 1,fpWork);
            fflush(fpWork);
            fflush( stdout );
            sem_post(pSem_mutex);
            sem_post(pSem_empty);
        }
    }
    wait(&iwait);
}
```

在ubuntu下运行正常，输出如第一幅截图所示。

向linux-0.11添加

```
sem_t sem_open( const char *name, unsigned int value);
int sem_wait sem_t ;
int sem_post sem_t ;
int sem_unlink const char ;
```

四个系统调用，为了简化，并没有使用sem_t *作为信号量标记，而是直接用sem_t （#define为int）变量作为信号量唯一的标识。

1.在include/unistd.h中添加四个系统调用功能号

```
131 #define __NR_setregid   71
132 #define __NR_sem_open   72
133 #define __NR_sem_wait   73
134 #define __NR_sem_post   74
135 #define __NR_sem_unlink 75
```

注意，因为系统调用号增加了，而在int 0x80中断处理函数sys_call函数中会判断中断是否在有效的范围内，所以这里还需要修改system_call.s中的nr_system_calls

2.在include/linux/sys.h 的sys_call_table 中添加这四个函数指针

3.实现这四个系统调用。 这一步是实验的关键，也是最难的地方。主要参考的是老师18讲的内容。 首先定义信号量等待任务结构：

```
struct Sem_wait_list{
    tast_struct *TaskNode;
    char name[ 20];
    int value ;
};
```

一个这样的对象表示一个信号量，信号量的名字在name中保存。同时TaskNode成员指向的是等待消费这个信号的任务。value部分表示的是信号值。 为了简化，我们设置系统中最多的信号量个数是80。所有的信号量保存在信号数组中，如下：

```
struct Sem_wait_list SemArray[ 80]={ 0}; // Max number of semaphony is 80
```

信号量的表示sem_t定义为int变量，这个变量也表示信号量在信号数组中的索引。升级版程序中可以考虑使用链表数据结构来代替这里的数组。

sys_sem_open 函数首先查找当前信号数组，看是否已经存在这一信号量，如果存在，则返回sem_t（在这里也就是数组下标）。如果不存在，则新建一个，再返回sem_t。程序中需要注意的是，在sem_open函数中，通过一个const char *name 参数传递信号量的名字。但是指针参数传递的是应用程序所在地址空间的逻辑地址，在内核中如果直接访问这个地址，访问到的是内核空间中的数据，不会是用户空间的。这一点可以参考前面系统调用的实验指导。

```
sem_t sys_sem_open ( const char int value )
{
    int i= 0;
    int LastEmpty = 0;
    char UserMemoryByte[ 2]={ '\0' , '\0' };
    UserMemoryByte[ 0] = get_fs_byte(name);
    printk( "sys_sem_open:name:%s,value:%d\n" ,UserMemoryByte , value );
    for (i= 1;i<MAX_SEMARRAY_NUM;i++)
    {
        if (SemArray[i].name[ 0] =='\0' )
        {
            LastEmpty = i;
            continue ;
        }
        if (!strcmp(SemArray[i].name,UserMemoryByte))
        {
            printk( "sys_sem_open:find the sem\n" );
            return i;
        }
    }
    //Not found
    if ( 0 == LastEmpty)
    {
        //The array is full
        return - 1;
    }
    //Add new semaphony
    printk( "sys_sem_open:add new sem\n" );
    strcpy(SemArray[LastEmpty].name ,UserMemoryByte);
    SemArray[LastEmpty]. value = value ;
    return LastEmpty;
}
```

为了简化程序的编写难度，在这个函数中，我只使用了传入名称字符串的第一个字符作为 name标识：

```
char UserMemoryByte[ 2]={ '\0' , '\0' };
UserMemoryByte[ 0] = get_fs_byte(name);
```

这个方法只是一时求快之举，实际上这样是很危险的。“empty"，"elementary"将会被识别为同一个信号量。

sys_sem_wait函数中首先查找这个信号量是否存在，如果不存在则返回错误。如果存在则判断这一信号量是否已小于或等于0，如果是则进入睡眠。否则把信号量减一，继续前进。

```
int sys_sem_wait ( )
{
    cli();
    if (sem >= MAX_SEMARRAY_NUM sem<= 0 )
    {
        printk( "sys_sem_wait:sem:%d is out of range\n" ,sem);
        sti();
        return - 1;
    }
    if ( SemArray[sem].name[ 0] == '\0' )
    {
        //This semophony does'n exist
        sti();
        return - 1;
    }
    while ( 0 >= SemArray[sem]. value )
    {
        sleep_on( &(SemArray[sem].TaskNode) );
    }

    //Catch the symaphony
  -- SemArray[sem]. value ;

    sti();
}
```

sys_sem_post函数主要是生产函数，在这个函数中首先也是判断参数所指的信号量是否存在，如果不存在则返回，如果不存在则返回。如果存在则把信号量的value加一。再唤醒等待队列中的任务。

```
int sys_sem_post sem_t
{
    cli();
    if ( SemArray[sem].name[ 0] == '\0' )
    {
        //This semophony does'n exist
        sti();
        return - 1;
    }
    ++SemArray[sem].value;
    wake_up( &(SemArray[sem].TaskNode) );
    sti();
}
```

sys_sem_unlink函数相对比较简单。首先查找这一信号量，如果不存在则返回错误。如果存在则从信号量数组SemArray中删除对应的成员。删除的具体步骤就是把对应成员重置为0。

```
int sys_sem_unlink const char
{

    char UserMemoryByte[ 2]={ '\0' , '\0' };
    UserMemoryByte[ 0] = get_fs_byte(name);
    int i= 0;
    for (i= 1;i<MAX_SEMARRAY_NUM;i++)
    {
        if ( ! strcmp (SemArray[i].name,UserMemoryByte) )
        {
            SemArray[i].name[ 0] = '\0' ;
            SemArray[i].TaskNode = 0;
            SemArray[i].value= 0;
            return 0;
        }
    }
}
```

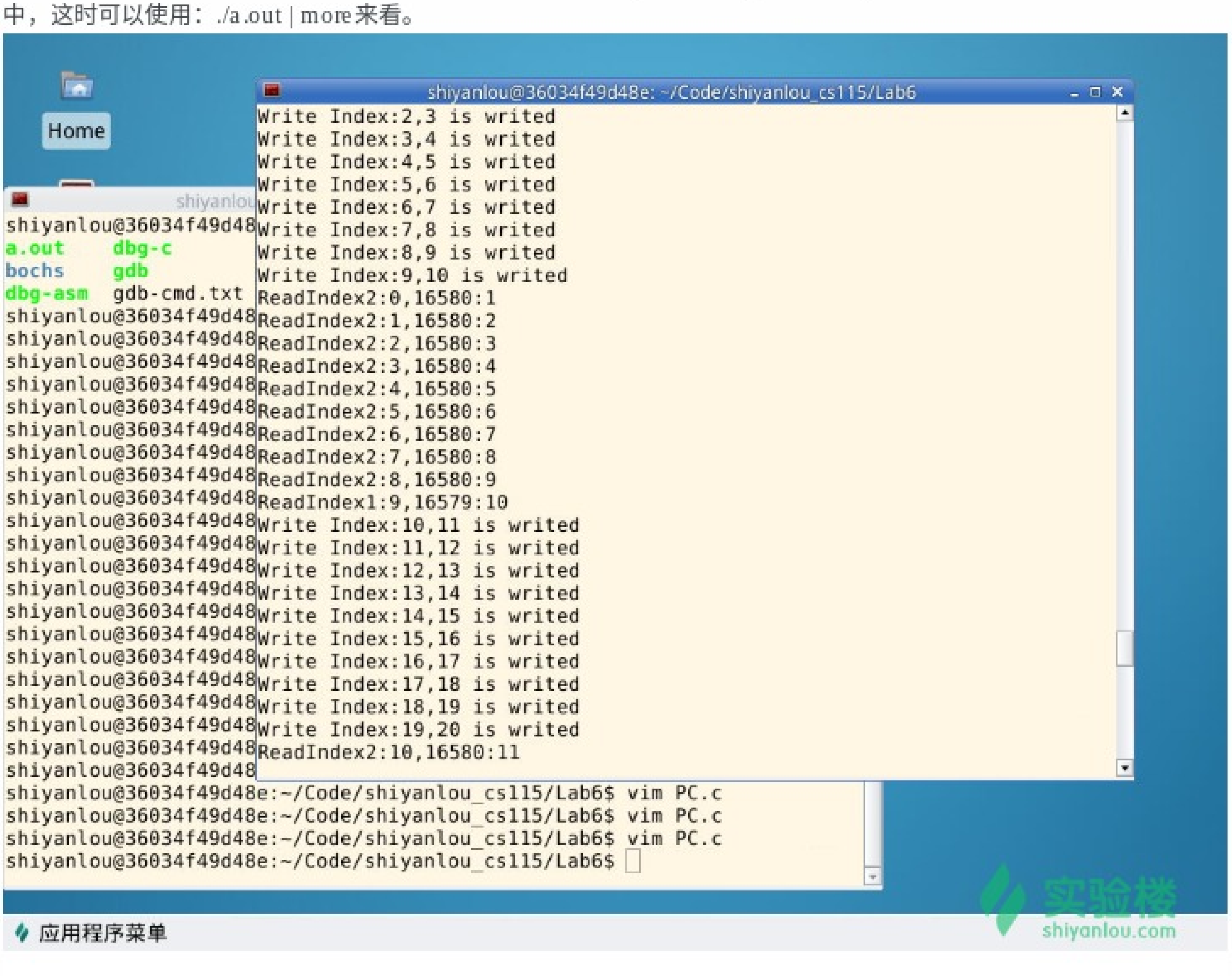
4.编写这四个系统调用对应的API。在include下新建sem.h文件。内容如下：

```
#define __LIBRARY__
#include <unistd.h>
_syscall2( int ,sem_open, const char *,name, unsigned int ,value)
_syscall1( int ,sem_wait, int ,sem)
_syscall1( int ,sem_post, int ,sem)
_syscall1( int ,sem_unlink, const char *,name)
```

5.编写测试程序，在前面PC.c的基础上进行改写。这里特别提出的是，在使用printf函数输出之后，要使用fflush(stdout)把缓存中的内容“冲”出去。不让会导致一些莫名其妙的问题。鄙人因为忘记在输出后加这句，导致调试了3个小时。

```
#include <sem.h>
#include <stdio.h>
#define sem_t int
#define FILECELLSIZE 3
FILE *fpWork;
FILE *fpFakeTerminal;
int main int char
{

    sem_t sem_empty;
    sem_t sem_full;
    sem_t sem_mutex;
    int iwait= 0;
    int WriteReadIndex[ 2]={ 0, 0};
    int i = 0;
    int FileWriteIndex = 0;
    int FileReadIndex_1 = 0;
    int a= 0;
    int FileReadIndex_2 = 0;
    int j= 0;
    printf ("main start" );
    fflush( stdout );
    sem_unlink( "empty" );
    sem_unlink( "full" );
    sem_unlink( "mutex" );
    sem_empty=sem_open( "empty" , 10 );
    sem_full=sem_open( "full" , 0);
    sem_mutex=sem_open( "mutex" , 1);
    fpWork = fopen( "work.dat" , "wb+" );
    fwrite(WriteReadIndex, 2*sizeof ( int ), 1,fpWork);
    fflush(fpWork);
    printf ("Opened file:%d\n" ,fpWork);
    fflush( stdout );
    if (!fork())
    {
        printf ("writer process start\n" );
        fflush( stdout );
        i= 0;
        while (i <= 50)
        {

            ++i;
            sem_wait(sem_empty);
            sem_wait(sem_mutex);
            fseek(fpWork, 0*sizeof ( int ),SEEK_SET);
            fread(&FileWriteIndex, sizeof ( int ), 1,fpWork);
            fseek(fpWork,(FileWriteIndex+ 2)* sizeof ( int ),SEEK_SET);
            fwrite(&i, sizeof ( int ), 1,fpWork);
            fflush(fpWork);
            printf ("Write Index:%d,%d is writed\n" ,FileWriteIndex,i);
            fseek(fpWork, 0*sizeof ( int ),SEEK_SET);
            FileWriteIndex++;
            fwrite(&FileWriteIndex, sizeof ( int ), 1,fpWork);
            fflush(fpWork);
            sem_post(sem_mutex);
            sem_post(sem_full);
        }
        while ( 1);

    }
    if (!fork())
    {
        while ( 1)
        {
            sem_wait(sem_full);
            sem_wait(sem_mutex);

            fseek(fpWork, 1*sizeof ( int ),SEEK_SET);
            fread(&FileReadIndex_1, sizeof ( int ), 1,fpWork);
            fseek(fpWork,(FileReadIndex_1+ 2)* sizeof ( int ),SEEK_SET);
            fread(&a, sizeof ( int ), 1,fpWork);
            printf ("ReadIndex1:%d,%d:%d\n" ,FileReadIndex_1,getpid(),a);
            fseek(fpWork, 1*sizeof ( int ),SEEK_SET);
            FileReadIndex_1++;
            fwrite(&FileReadIndex_1, sizeof ( int ), 1,fpWork);
            fflush( stdout );
            fflush(fpWork);
            sem_post(sem_mutex);
            sem_post(sem_empty);
        }
    }
    if (!fork())
    {
        while ( 1)
        {
            j= 0;
            sem_wait(sem_full);
            sem_wait(sem_mutex);
            fseek(fpWork, 1*sizeof ( int ),SEEK_SET);
            fread(&FileReadIndex_2, sizeof ( int ), 1,fpWork);
            fseek(fpWork,(FileReadIndex_2+ 2)* sizeof ( int ),SEEK_SET);
            fread(&j, sizeof ( int ), 1,fpWork);
            printf ("ReadIndex2:%d,%d:%d\n" ,FileReadIndex_2,getpid(),j);
            fseek(fpWork, 1*sizeof ( int ),SEEK_SET);
            FileReadIndex_2++;
            fwrite(&FileReadIndex_2, sizeof ( int ), 1,fpWork);
            fflush(fpWork);
            fflush( stdout );
            sem_post(sem_mutex);
            sem_post(sem_empty);
        }
    }
    wait(&iwait);
}
```

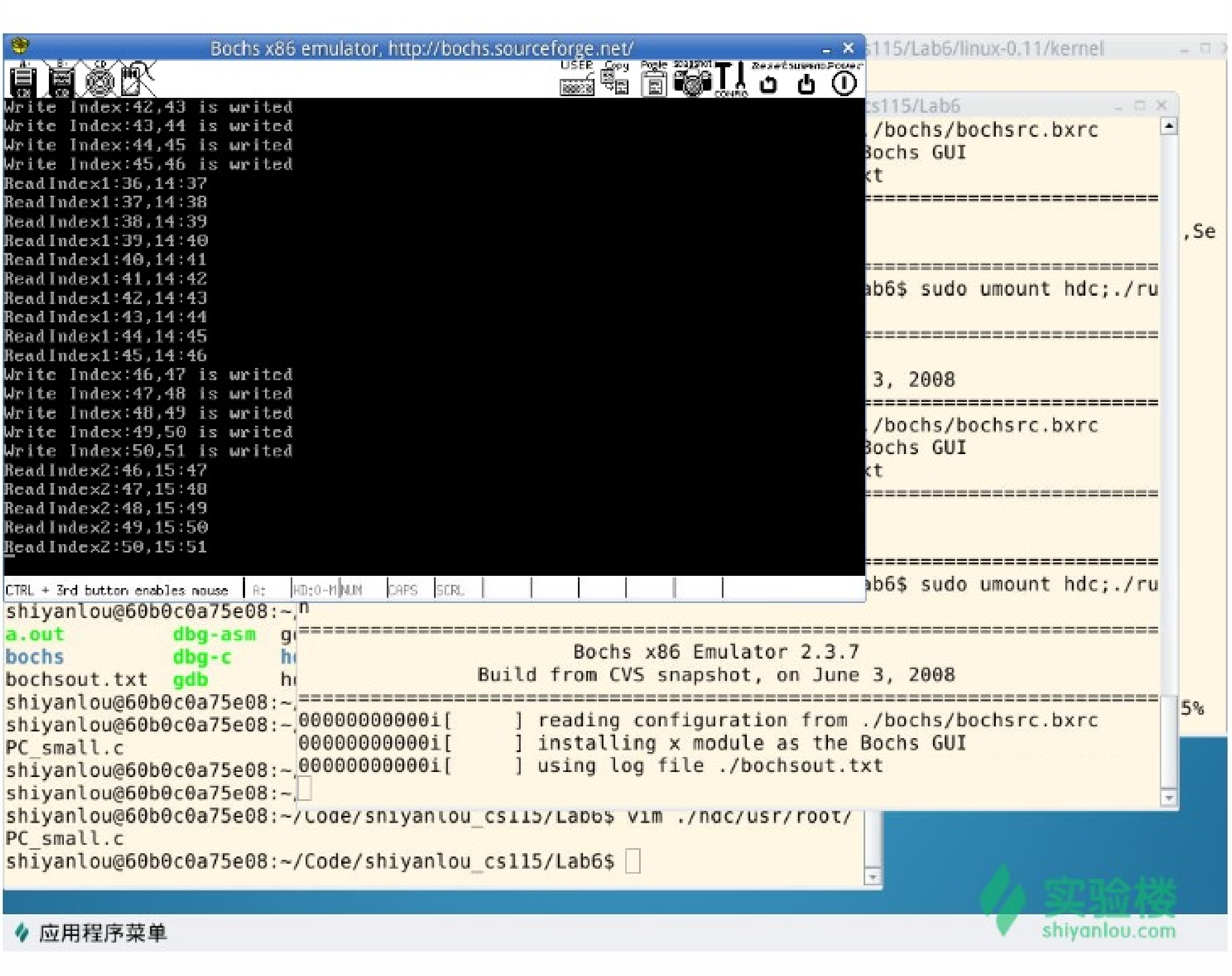
6.调试过程的一些问题。因为在bochs运行linux0.11的时候，大量的输出会导致花屏，可以把内容导出到一个文件中。比如你的程序是a.out。那么使用 ./a.out >> Print.txt函数可以把输出写到Print.txt中。有时候通过ctrl+c停下程序后，屏幕一片空白，敲键盘输入也没有显示。此时只是显示有点问题。直接闭着眼睛敲命令，用vi 打开这个文件即可：vi Print.txt。鄙人在调试的过程中，在系统调用处理函数：比如sys_sem_open 中使用printk输出的信息不会到这个文件中，这时可以使用：./a.out | more来看。



shiyanlou@36034f49d48e: ~/Code/shiyanlou_cs115/Lab6

```
Write Index:2,3 is writed
Write Index:3,4 is writed
Write Index:4,5 is writed
Write Index:5,6 is writed
Write Index:6,7 is writed
Write Index:7,8 is writed
Write Index:8,9 is writed
Write Index:9,10 is writed
ReadIndex2:0,16580:1
ReadIndex2:1,16580:2
ReadIndex2:2,16580:3
ReadIndex2:3,16580:4
ReadIndex2:4,16580:5
ReadIndex2:5,16580:6
ReadIndex2:6,16580:7
ReadIndex2:7,16580:8
ReadIndex2:8,16580:9
ReadIndex1:9,16579:10
Write Index:10,11 is writed
Write Index:11,12 is writed
Write Index:12,13 is writed
Write Index:13,14 is writed
Write Index:14,15 is writed
Write Index:15,16 is writed
Write Index:16,17 is writed
Write Index:17,18 is writed
Write Index:18,19 is writed
Write Index:19,20 is writed
ReadIndex2:10,16580:11
```

```
shiyanlou@36034f49d48e:~/Code/shiyanlou_cs115/Lab6$ vim PC.c
shiyanlou@36034f49d48e:~/Code/shiyanlou_cs115/Lab6$ vim PC.c
shiyanlou@36034f49d48e:~/Code/shiyanlou_cs115/Lab6$ vim PC.c
shiyanlou@36034f49d48e:~/Code/shiyanlou_cs115/Lab6$
```



Bochs x86 emulator, http://bochs.sourceforge.net/

```
Write Index:42,43 is writed
Write Index:43,44 is writed
Write Index:44,45 is writed
Write Index:45,46 is writed
ReadIndex1:36,14:37
ReadIndex1:37,14:38
ReadIndex1:38,14:39
ReadIndex1:39,14:40
ReadIndex1:40,14:41
ReadIndex1:41,14:42
ReadIndex1:42,14:43
ReadIndex1:43,14:44
ReadIndex1:44,14:45
ReadIndex1:45,14:46
Write Index:46,47 is writed
Write Index:47,48 is writed
Write Index:48,49 is writed
Write Index:49,50 is writed
Write Index:50,51 is writed
ReadIndex2:46,15:47
ReadIndex2:47,15:48
ReadIndex2:48,15:49
ReadIndex2:49,15:50
ReadIndex2:50,15:51
```

```
========================================================================
                       Bochs x86 Emulator 2.3.7
              Build from CVS snapshot, on June 3, 2008
========================================================================
00000000000i[     ] reading configuration from ./bochs/bochsrc.bxrc
00000000000i[     ] installing x module as the Bochs GUI
00000000000i[     ] using log file ./bochsout.txt
```

```
shiyanlou@60b0c0a75e08:~/Code/shiyanlou_cs115/Lab6$ vim ./hdc/usr/root/
PC_small.c
shiyanlou@60b0c0a75e08:~/Code/shiyanlou_cs115/Lab6$
```

2