北京郵電大學

实验报告

题目： 键盘驱动程序的分析与修改

班　　级： 2021211301

学　　号： 2021210967

姓　　名： 王心雨

学　　院： 计算机学院

2022 年 11 月 29 日

一、实验目的

1. 理解 I/O 系统调用函数和 C 标准 I/O 函数的概念和区别；
2. 建立内核空间 I/O 软件层次结构概念，即与设备无关的操作系统软件、设备驱动程序和中断服务程序；
3. 了解 Linux-0.11 字符设备驱动程序及功能，初步理解控制台终端程序的工作原理；
4. 通过阅读源代码，进一步提高 C 语言和汇编程序的编程技巧以及源代码分析能力；
5. 锻炼和提高对复杂工程问题进行分析的能力，并根据需求进行设计和实现的能力。

二、实验环境

1. 硬件：学生个人电脑（x86-64）
2. 软件：Windows 10，VMware Workstation 15 Player，32 位 Linux-Ubuntu 16.04.1
3. gcc-3.4 编译环境
4. GDB 调试工具
5. Bochs 模拟器 2.3.7；

三、实验内容

从网盘下载 lab4.tar.gz 文件，解压后进入 lba4 目录得到如下文件和目录：

```
    4096 Dec 21 17:08 ./
    4096 Dec 28 08:42 ../
       0 Nov 27 06:16 a.out
    4096 Dec 20 08:44 bochs/
   14889 Dec 21 17:10 bochsout.txt
     115 Nov 26 12:03 dbg-asm*
     119 Nov 26 12:03 dbg-c*
    4096 Dec 20 08:45 files/
12423461 Nov 26 12:03 gdb*
      75 Nov 26 12:03 gdb-cmd.txt
    4096 Oct 10  2014 hdc/
63504384 Dec 21 17:09 hdc-0.11.img
    4096 Dec 21 17:08 linux-0.11/
  119902 Nov 26 12:03 linux-0.11.tar.gz
     131 Nov 26 12:03 mount-hdc*
     701 Nov 26 12:03 run*
     268 Nov 26 12:03 rungdb*
   12288 Nov 26 12:25 .run.swp
```

实验常用执行命令如下：

- ✧ 执行./run ，可启动 bochs 模拟器，进而加载执行 Linux-0.11 目录下的 Image 文件启动 linux-0.11 操作系统
- ✧ 进入 lab4/linux-0.11 目录，执行 make 编译生成 Image 文件，每次重新编译（make）前需先执行 make clean
- ✧ 如果对 linux-0.11 目录下的某些源文件进行了修改，执行./run init 可把修改文件回复初始状态

本实验包含 2 关，要求如下：

- ✧ Phase 1

  键入 F12，激活*功能，键入学生本人的姓名拼音，首尾字母等显示*

  比如：zhangsan，显示为：*ha*gsa*
- ✧ Phase 2

  键入“学生本人的学号” ：激活*功能，键入学生本人的姓名拼音，首尾字母等显示*

  比如：zhangsan，显示为：*ha*gsa*，

  键入“学生本人的学号-” ：取消显示*功能

提示：完成本实验需要对 lab4/linux-0.11/kernel/chr_drv/目录下的 keyboard.s、console.c 和 tty_io.c 源文件进行分析，理解按下按键到回显到显示频上程序的执行过程，然后对涉及到的数据结构进行分析，完成对前两个源程序的修改。修改方案有两种：

- ✧ 在 C 语言源程序层面进行修改
- ✧ 在汇编语言源程序层面进行修改

实验 4 的其他说明见 lab4.pdf 和 lab4 虚拟机环境安装说明.docx。linux 内核完全注释(高清版).pdf 一书中对源代码有详细的说明和注释。

四、源代码的分析及修改

针对一次按键操作对源代码 keyboard.s、console.c 和 tty_io.c 的进行分析，说明分析过程，要配有流程图（不能从书中进行截图）进行说明，给出各阶段的修改思路和代码实现。各阶段需要有较详细的文字说、运行截图、分析过程的内容。

一、知识解读

对源文件进行分析：

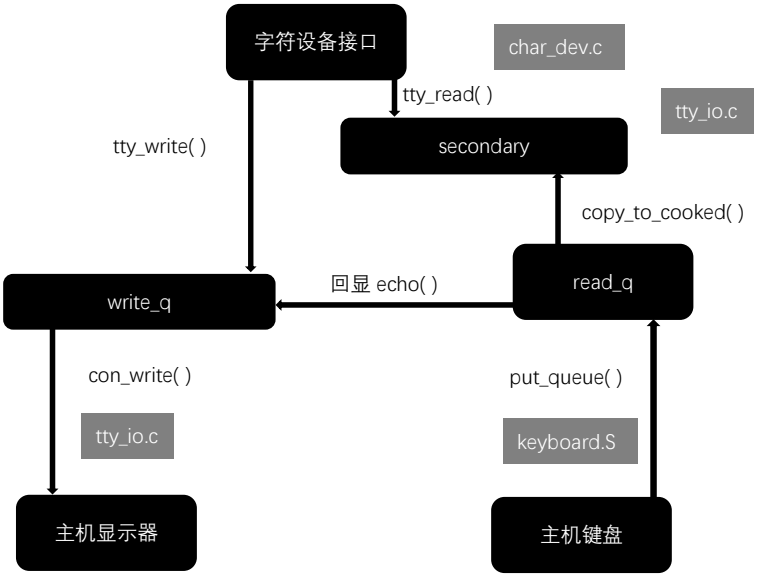
在 Linux 0.11 系统中可以使用两类终端。一类是主机上的控制台终端，另一类是串行硬件终端设备。控制台终端对应有一个 tty_struct 数据结构，主要用来保存终端设备当前的参数、字符 IO 缓冲队列等信息，其中有三个缓冲队列，分别是 read_q：保存从键盘输入的原始字符序列，write_q：保存写到控制台显示屏的数据和 secondary：保存从 read_q 取出的经过行规则程序处理的数据。

keyboard.S 中主要包括键盘中断处理程序，主要包含 con_write()。keyboard.S 和 console.c 接收上层 tty_io.c 程序传递下来的显示字符或控制信息。它首先根据键盘特殊键的状态设置状态标记变量 mode 的值，然后根据引起键盘终端的按键扫描码，调用已经编排成跳转表的相应扫描码处理子程序，把扫描码对应的字符放入 read_q 中，之后调用 tty_io.c 中的 do_tty_interrupt 函数。

console.c 中的控制台初始化程序和控制台写函数 con_write 函数从 write_q 中取出字符序列，根据字符的性质进行在屏幕终端上显示字符、光标移动、字符擦除等操作。

tty_io.c 源文件包含 tty 字符设备读函数 tty_read()函数和写函数 tty_write(),为文件系统提供了上层访问接口；do_tty_interrupt 函数包含对 copy_to_cooked 函数的调用，这个函数对 read_q 中的字符进行适当处理之后放到 secendary 队列中，在此期间，如果设置了回显标志，则字符还会被放入到 write_q 队列中，以在终端屏幕上显示刚键入的字符。

2. 按下按键到回显到显示频上程序的执行过程：



二、准备过程：

安装 linux-0.11 的配置环境，本人姓名 wangxinyu，将 ubuntu 用户名设置为此；接着下载 gcc 与 as86 编译器成功创建了虚拟机，将 lab4.tar 文件拖入虚拟机的 home，进而创建新的 Terminal。首先我们解压 lab4 文件，键入 ll 指令查看压缩包内的具体文件和目录；

```
wangxinyu@ubuntu:~$ ls lab4
a.out  bochsout.txt  dbg-c  gdb          hdc          linux-0.11         mount-hdc  rungdb
bochs  dbg-asm       files  gdb-cmd.txt  hdc-0.11.img  linux-0.11.tar.gz  run
wangxinyu@ubuntu:~$
```

```
wangxinyu@ubuntu:~$ cd lab4
wangxinyu@ubuntu:~/lab4$ ll
total 74348
drwxrwxr-x  6 wangxinyu wangxinyu     4096 Dec  4  2019 ./
drwxr-xr-x 16 wangxinyu wangxinyu     4096 Nov 29 02:23 ../
-rw-rw-r--  1 wangxinyu wangxinyu        0 Nov 26  2018 a.out
drwxr-xr-x  2 wangxinyu wangxinyu     4096 Dec  4  2019 bochs/
-rw-rw-r--  1 wangxinyu wangxinyu    15156 Nov 12 04:02 bochsout.txt
-rwxrwxr-x  1 wangxinyu wangxinyu      115 Nov 25  2018 dbg-asm*
-rwxrwxr-x  1 wangxinyu wangxinyu      119 Nov 25  2018 dbg-c*
drwxrwxr-x  2 wangxinyu wangxinyu     4096 Dec  4  2019 files/
-rwxrwxr-x  1 wangxinyu wangxinyu 12423461 Nov 25  2018 gdb*
-rw-rw-r--  1 wangxinyu wangxinyu       75 Nov 25  2018 gdb-cmd.txt
drwxr-xr-x  2 wangxinyu wangxinyu     4096 Oct 10  2014 hdc/
-rw-r--r--  1 wangxinyu wangxinyu 63504384 Nov 12 04:01 hdc-0.11.img
drwxrwxr-x 10 wangxinyu wangxinyu     4096 Nov 12 04:01 linux-0.11/
-rw-rw-r--  1 wangxinyu wangxinyu   119902 Nov 25  2018 linux-0.11.tar.gz
-rwxrwxr-x  1 wangxinyu wangxinyu      131 Nov 25  2018 mount-hdc*
-rwxrwxr-x  1 wangxinyu wangxinyu      701 Nov 25  2018 run*
-rwxrwxr-x  1 wangxinyu wangxinyu      268 Nov 25  2018 rungdb*
-rw-r--r--  1 wangxinyu wangxinyu    12288 Nov 25  2018 .run.swp
wangxinyu@ubuntu:~/lab4$
```

三、阶段一

1. 翻阅 PPT 以及 linux 内核注释的书籍，之后键入 cd linux-0.11/kernel/chr_drv 进入对应目录，此时先 cd keyboard.S 查看其的源代码；获得了其 key_table 即扫描码到对应按键处理程序的跳转表，分析得知 F1——F12 的扫描码用函数 func 处理，查阅其对应的注释；

```
wangxinyu@ubuntu: ~/lab4/linux-0.11/kernel/chr_drv
key_table:
        .long none,do_self,do_self,do_self
        .long do_self,do_self,do_self,do_self
        .long do_self,do_self,do_self,do_self
        .long do_self,do_self,do_self,do_self
        .long do_self,do_self,do_self,do_self
        .long do_self,do_self,do_self,do_self
        .long do_self,do_self,do_self,do_self
        .long do_self,ctrl,do_self,do_self
        .long do_self,do_self,do_self,do_self
        .long do_self,do_self,do_self,do_self
        .long do_self,do_self,lshift,do_self
        .long do_self,do_self,do_self,do_self
        .long do_self,do_self,do_self,do_self
        .long do_self,minus,rshift,do_self
        .long alt,do_self,caps,func
        .long func,func,func,func
        .long func,func,func,func
        .long func,num,scroll,cursor
        .long cursor,cursor,do_self,cursor
        .long cursor,cursor,do_self,cursor
        .long cursor,cursor,cursor,cursor
        .long none,none,do_self,func
        .long func,none,none,none
        .long none,none,none,none
```

```
508         .long do_self,do_self,do_self,do_self    /* 14-17 t y u i */
509         .long do_self,do_self,do_self,do_self    /* 18-1B o p } ^ */
510         .long do_self,ctrl,do_self,do_self       /* 1C-1F enter ctrl a s */
511         .long do_self,do_self,do_self,do_self    /* 20-23 d f g h */
512         .long do_self,do_self,do_self,do_self    /* 24-27 j k l | */
513         .long do_self,do_self,lshift,do_self     /* 28-2B { para lshift , */
514         .long do_self,do_self,do_self,do_self    /* 2C-2F z x c v */
515         .long do_self,do_self,do_self,do_self    /* 30-33 b n m , */
516         .long do_self,minus,rshift,do_self       /* 34-37 . - rshift * */
517         .long alt,do_self,caps,func              /* 38-3B alt sp caps f1 */
518         .long func,func,func,func                /* 3C-3F f2 f3 f4 f5 */
519         .long func,func,func,func                /* 40-43 f6 f7 f8 f9 */
520         .long func,num,scroll,cursor             /* 44-47 f10 num scr home */
521         .long cursor,cursor,do_self,cursor       /* 48-4B up pgup - left */
522         .long cursor,cursor,do_self,cursor       /* 4C-4F n5 right + end */
523         .long cursor,cursor,cursor,cursor        /* 50-53 dn pgdn ins del */
524         .long none,none,do_self,func             /* 54-57 sysreq ? < f11 */
525         .long func,none,none,none                /* 58-5B f12 ? ? ? */
```

2. 进一步分析 func 函数的汇编代码，根据注释可知，红框中两行是判定是否是 F12 键被按下的指令，因此我们可以在这个 call 一个 F12 的状态函数，起名为 change_F12flag；

```
func:
        pushl %eax
        pushl %ecx
        pushl %edx
        call show_stat
        popl %edx
        popl %ecx
        popl %eax
        subb $0x3B,%al
        jb end_func
        cmpb $9,%al
        jbe ok_func
        subb $18,%al
        cmpb $10,%al
        jb end_func
        cmpb $11,%al
        ja end_func
ok_func:
```

➔

```
func:
        pushl %eax
        pushl %ecx
        pushl %edx
        call show_stat
        popl %edx
        popl %ecx
        popl %eax
        subb $0x3B,%al
        jb end_func
        cmpb $9,%al
        jbe ok_func
        subb $18,%al
        cmpb $10,%al
        jb end_func
        cmpb $11,%al
        ja end_func
        call change_F12flag
ok_func:
```

3. 接下来具体写函数的实现：先定义全局变量 F12flag，初始化为 0；再写 change_F12flag 函数。根据汇编指令，每按一次 F12 键，该函数被调用；

```
int F12flag=0;

void change_F12flag(void)
```

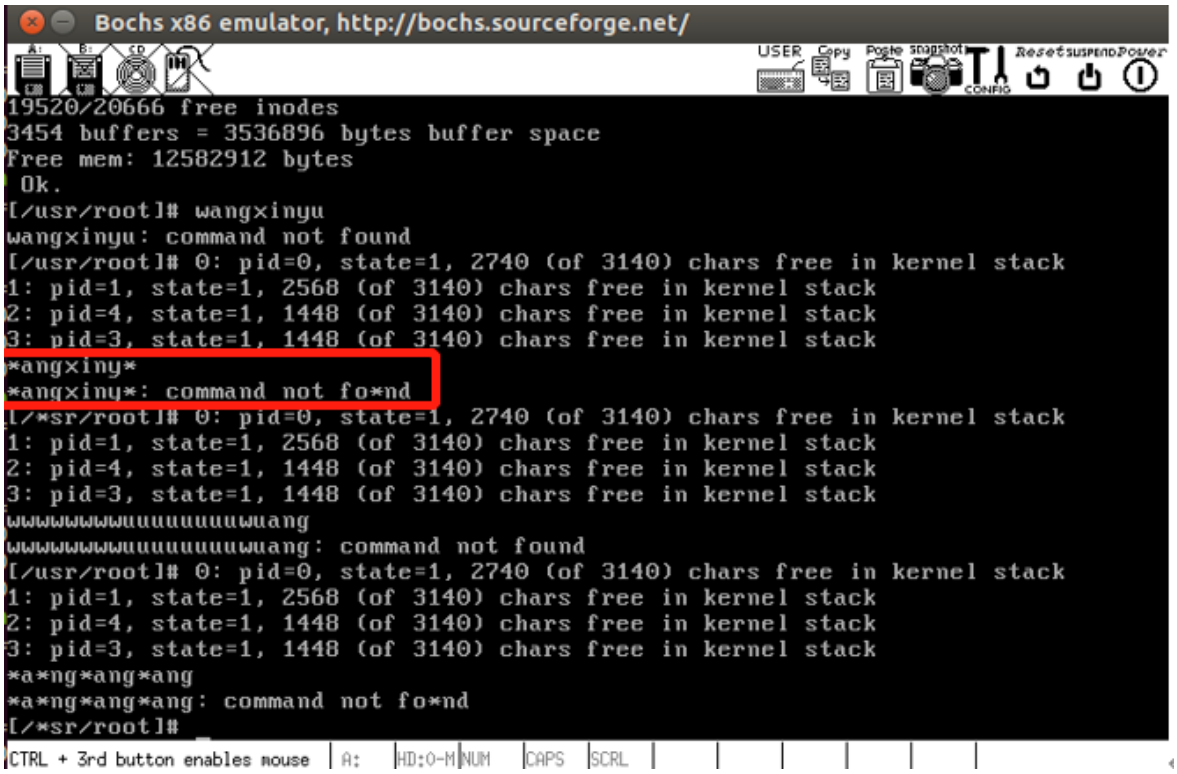
```
void change_F12flag(void)
{
        if(F12flag==1)  F12flag=0;
        else if(F12flag==0)     F12flag=1;
}
void con_write(struct tty_struct * tty)
{
        int nr;
```

4. F12flag 置位时，将英文字母显示为*；此时分析 console.c 源代码中的 con_write 文件；我们需要判断是否当前的输入为首字母‘w’或者‘u’，因此我们增加如下代码，字符 c 在写到内存 pos 处后就显示在屏幕上，更改为*即可实现。

```
void con_write(struct tty_struct * tty)
{
        int nr;
        char c;

        nr = CHARS(tty->write_q);
        while (nr--) {
                GETCH(tty->write_q,c);
                switch(state) {
                        case 0:
                                if (c>31 && c<127) {
                                        if (x>=video_num_columns) {
                                                x -= video_num_columns;
                                                pos -= video_size_row;
                                                lf();
                                        }
                                        if(F12flag==1 &&(c=='w' || c=='u')){
                                                c='*';
                                        }
                                        __asm__("movb attr,%%ah\n\t"
                                        "movw %%ax,%1\n\t"
                                        ::"a" (c),"m" (*(short *)pos)
                                        );
                                        pos += 2;
                                        x++;
```

5. 之后我们通过 cd ..退回 linux-0.11 文件目录下，键入 make clean 和 make 指令继续编译；再键入 cd ..进入 lab4 文件目录下，键入./run 启动 Bochs 模拟器，指令实现如下，符合要求，阶段一完成。



四、阶段 2：

1. 阶段 2 要实现的功能是输入我的学号 2021210967，就能将所有的‘w’和‘u’更改为*号，再次输入学号，能够复原；我们在 Linux 内核注释中了解到，最终实现复原是在学号后加-，这个过程由 do——self 函数完成。即如果输入 2021210967-，即能实现字符复原；

```
521     .long cursor,cursor,do_self,cursor      /* 48-4B up pgup - left */
522     .long cursor,cursor,do_self,cursor      /* 4C-4F n5 right + end */
523     .long cursor,cursor,cursor,cursor       /* 50-53 dn pgdn ins del */
524     .long none,none,do_self,func            /* 54-57 sysreq ? < f11 */
525     .long func,none,none,none               /* 58-5B f12 ? ? ? */
```

2. 首先我们使用./run init 清空阶段 1 文件的修改内容；根据阶段 1 我们已经基本大致清楚了 con_write 函数的机制，因此我们直接在 con)write 函数内部设定有限状态机，设定变量 flag 用于确定是否反转功能，设定变量 count，初始化为 0 用于实现记录输入的有效字符串的个数；

```
int flag=0;

int count=0;


void con_write(struct tty_struct * tty)
{
```

```
void con_write(struct tty_struct * tty)
{
        int nr;
        char c;

        nr = CHARS(tty->write_q);
        while (nr--) {
                GETCH(tty->write_q,c);
                switch(state) {
                        case 0:
                                if (c>31 && c<127) {
                                        if (x>=video_num_columns) {
                                                x -= video_num_columns;
                                                pos -= video_size_row;
                                                lf();
                                        }
                                        if(c=='2' && (count==0 || count==2 || count==4)){
                                                count++;
                                        } else if(c=='0' && (count==1 || count==6)){
                                                count++;
                                        } else if( c=='1' && (count==3 || count==5)){
                                                count++;
                                        } else if( c=='9' && (count==7)){
                                                count++;
                                        } else if( c=='6' && (count==8)){
                                                count++;
                                        } else if( c=='7' && (count==9)){
                                                count++;
                                        } else if( c=='-' && (count==10)){
                                                count++;
                                        } else {
                                                count=0;
                                        }
                                        if(count=10 && flag=0){
                                                flag=1;
                                        }
                                        if(count==11 && flag=1){
                                                flag=0;
                                        }

                                        if(flag && (c=='w' || c=='u')){
                                                c='*';
                                        }
```

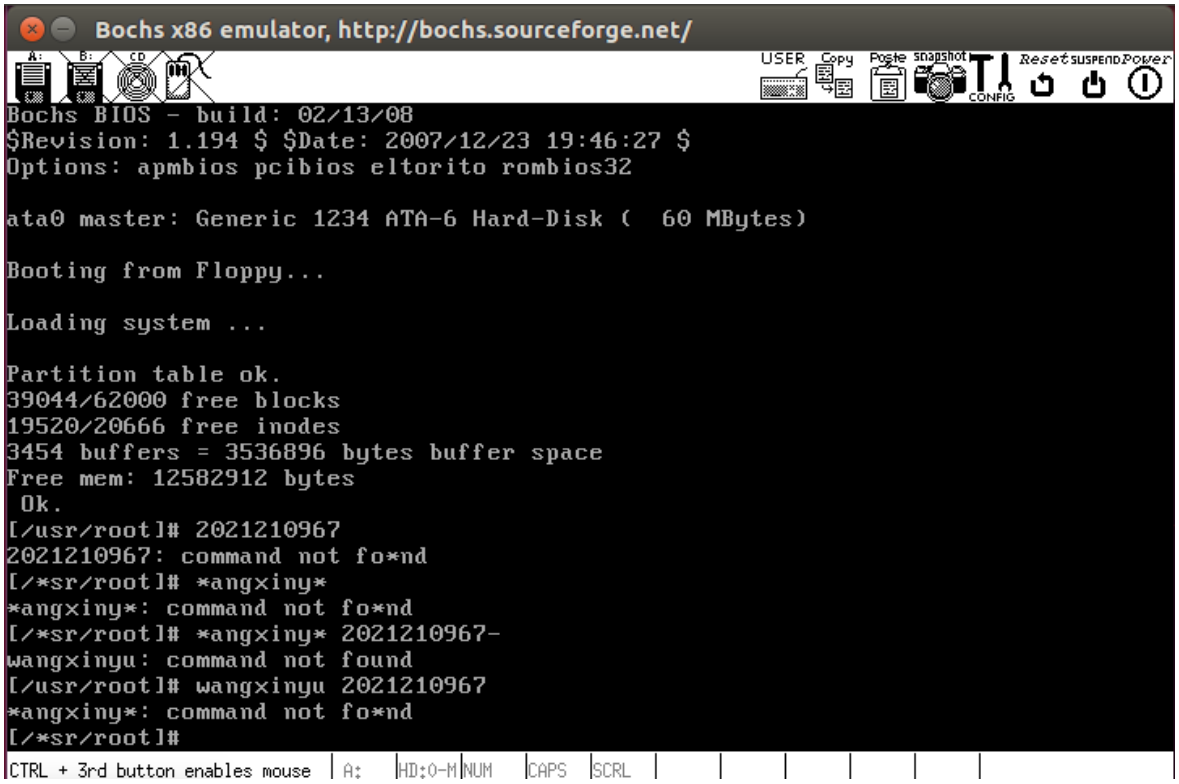
3. 类似于阶段一，在 linux-0.11 目录下键入 make clean 和 make 指令编译，在 lab4 目录下./run 运行 Bochs 模拟器；

```
wangxinyu@ubuntu:~/lab4/linux-0.11$ make clean
rm -f Image System.map tmp_make core boot/bootsect boot/setup
rm -f init/*.o tools/system tools/build boot/*.o
(cd mm;make clean)
make[1]: Entering directory '/home/wangxinyu/lab4/linux-0.11/mm'
rm -f core *.o *.a tmp_make
```

```
wangxinyu@ubuntu:~/lab4/linux-0.11$ make
as86 -0 -a -o boot/bootsect.o boot/bootsect.s
ld86 -0 -s -o boot/bootsect boot/bootsect.o
as86 -0 -a -o boot/setup.o boot/setup.s
ld86 -0 -s -o boot/setup boot/setup.o
gcc-3.4 -m32 -g -I./include -traditional -c boot/head.s
```

```
wangxinyu@ubuntu:~/lab4/linux-0.11$ cd ..
wangxinyu@ubuntu:~/lab4$ ./run
========================================================================
                    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
wangxinyu@ubuntu:~/lab4$
```

4. 模拟结果如下，与要求相符合，阶段二通过。



五、总结体会

总结心得（包括实验过程中遇到的问题、如何解决的、过关或挫败的感受、实验投入的时间和精力、意见和建议等）

实验四对我个人来说可能比前几个实验更为简单一些，当然这个实验的意义并非其表面上的通过而已，在安装配置环境的时候，在一点点对照 gcc 编译器安装指令敲时，在第一次尝试使用虚拟机做实验之后，在能够自己看懂汇编代码以及通过翻阅资料和课堂的 ppt 能够理解实验的实质的时候，我觉得是比较幸福的。当然，我也遇到了很多问题，像 gcc 安装了好些次都没安装成功，以及对于虚拟机而言，我的平板电脑的键盘已经自我锁定了，需要解锁才能够在虚拟机中使用而不是调用 F12 原本的功能，用 Fn+F8 键成功解决了这个问题，虽然这个问题花费了很久的时间，但是也在一遍遍的检查代码的过程中提升了做任何实验的耐心。

第一个实验是修改 F12 的功能实现，通过键盘的程序的插入和函数的调用等等；第二个实验我觉得更简单，只是在 console.c 中加入了一个状态判断的程序，从而实现了扫描码为学号从而实现首尾字母更改的功能！计算机系统的实践课程已经告一段落，希望能够在理论课上获得一个好成绩！也非常感谢帅气睿智的周锋老师的教学以及助教的帮助！感谢！