

北京邮电大学

实验报告



题目： 键盘驱动程序的分析与修改

班 级： 2018211318

学 号： 2018210547

姓 名： 胡天翼

学 院： 计算机学院

2019 年 12 月 13 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

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：hagsa*

- 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：hagsa*，

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：

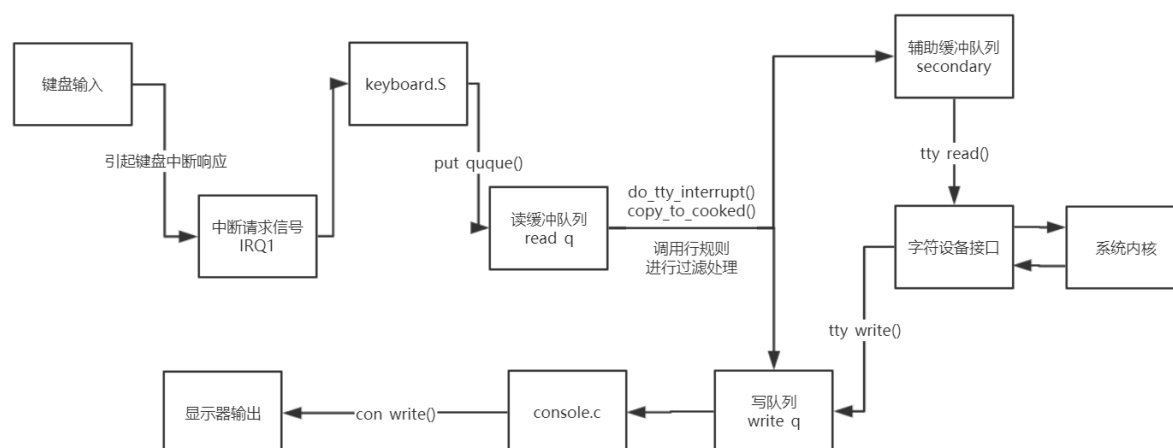
- 在C语言源程序层面进行修改
- 在汇编语言源程序层面进行修改

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

分析阶段

keyboard.s和console.c这两个程序是Linux系统主机中使用显示器和键盘模拟一个硬件终端设备的仿真程序。用户通过它们与终端交互：



准备阶段

1. 增加全局变量f12Flag，每按一次F12，将该变量值翻转

- XT键盘扫描码为

F1	F2	~	1	2	3	4	5	6	7	8	9	0	-	=	\	BS	ESC	NUML	SCRL	SYSR
3B	3C	29	02	03	04	05	06	07	08	09	0A	0B	0C	0D	2B	0E	01	45	46	**
F3	F4	TAB	Q	W	E	R	T	Y	U	I	O	P	[]			Home	↑	PgUp	PrtSc
3D	3E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B			47	48	49	37
F5	F6	CNTL	A	S	D	F	G	H	J	K	L	;	'	ENTER			←	5	→	-
3F	40	1D	1E	1F	20	21	22	23	24	25	26	27	28	1C			4B	4C	4D	4A
F7	F8	LSHFT	Z	X	C	V	B	N	M	,	.	/		RSHT			End	↓	PgDn	+
41	42	2A		2C	2D	2E	2F	30	31	32	33	34	35	36			4F	50	51	4E
F9	F10	ALT			Space										CAPLOCK		Ins		Del	
43	44	38			39										3A		52		53	

F12的扫描码为0x58

2. 在 keyboard.s 中，找到F12对应的分析函数

```
long func, none, none, none      /* 58-5B f12 ? ? ? */
```

3. 在 keyboard.s 中搜索 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      //功能键是F1-F10?
    jbe ok_func      //是,则跳转。
    subb $18,%al     //是功能键F11,F12吗?F11、F12扫描码是0x57、0x58。
    cmpb $10,%al     //是功能键F11?
    jb end_func      //不是,则不处理,返回。
    cmpb $11,%al     //是功能键F12?
    ja end_func
    call change_f12Flag //假若是功能键F12,则跳转到激活“*”状态的函数
ok_func:
    cmpl $4,%ecx      /* check that there is enough room */
    jl end_func
    movl func_table(,%eax,4),%eax
    xorl %ebx,%ebx
    jmp put_queue
end_func:
    ret

```

4. 分析语句，在“是功能键F12”后增添跳转到 `change_f12Flag`。

- 在 `/kernel/chr_drv/console.c` 中增加全局变量 `f12Flag` 与函数 `change_f12Flag`，每按一次F12，调用其进行翻转。

```

extern int f12Flag=0;

void change_f12Flag(void){
    switch(f12Flag){
        case 1:
            f12Flag=0;
            break;
        case 0:
            f12Flag=1;
            break;
    }
}

```

5. 修改 `/kernel/chr_drv/console.c` 中的 `con_write()`，`f12Flag`置位时，将英文字母显示为*
分析 `con_write()`

```

void con_write(struct tty_struct * tty)
{
    int nr; //首先取得写缓冲队列中现有字符数nr
    char c;

    nr = CHARS(tty->write_q);
    while (nr--){

```

```

GETCH(tty->write_q,c); //针对队列中的每个字符进行处理
//state=0:初始状态,或者原状态4;或者原状态1,但字符不是 '[';
//1:原状态0,并且字符是转义字符ESC(0x1b=033=27)。处理后恢复状态0。
//2:原状态1,并且字符是 '[';
//3:原状态2,或者原状态3,并且字符是 ';'或数字。
//4:原状态3,并且字符不是 ';'或数字;处理后恢复状态0。
switch(state) {
    case 0:
        if (c>31 && c<127) { //普通字符
            if (x>=video_num_columns) { //换行
                x -= video_num_columns;
                pos -= video_size_row;
                lf();
            }
            __asm__("movb attr,%%ah\n\t" //写字符
                    "movw %%ax,%l\n\t"
                    ::"a" (c),"m" (*(short *)pos)
                    );
            pos += 2;
            x++;
            //后面的代码略去

```

因此，我们在“普通字符”的case下，即22行加入如下语句

```

if(f12Flag && (c>='a'&&c<='z' || c >='A'&&c<='Z'))
    c='*';

```

phase1

键入F12，激活*功能，键入学生姓名拼音，首尾字母等显示*

我的姓名是 hutianyi，因此只要将 h 与 i 置*即可，只需要修改 con_write 中对置*代码的判断语句，其他代码不用改变。

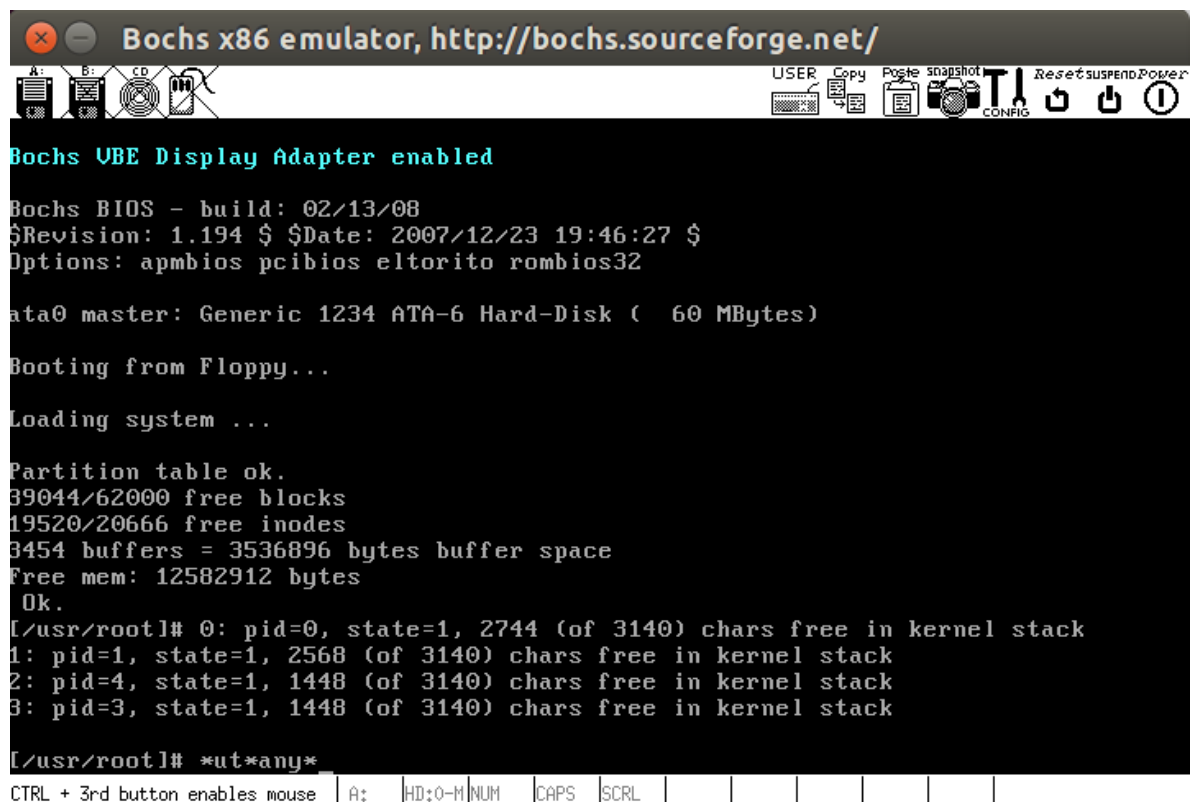
因此，我们在 console.c 的 con_write() 中，在“普通字符”的case下，加入如下语句

```

if(f12Flag && (c == 'h' || c == 'i'))
    c='*';

```

• 运行结果



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

Bochs UBE Display Adapter enabled

Bochs BIOS - build: 02/13/08
$Revision: 1.194 $ $Date: 2007/12/23 19:46:27 $
Options: apmbios pcibios eltorito rombios32

ata0 master: Generic 1234 ATA-6 Hard-Disk ( 60 MBytes)

Booting from Floppy...

Loading system ...

Partition table ok.
39044/62000 free blocks
19520/20666 free inodes
3454 buffers = 3536896 bytes buffer space
Free mem: 12582912 bytes
Ok.
[/usr/root]# 0: pid=0, state=1, 2744 (of 3140) chars free in kernel stack
1: pid=1, state=1, 2568 (of 3140) chars free in kernel stack
2: pid=4, state=1, 1448 (of 3140) chars free in kernel stack
3: pid=3, state=1, 1448 (of 3140) chars free in kernel stack

[/usr/root]# *ut*any*

CTRL + 3rd button enables mouse | A: | HD:0-M | NUM | CAPS | SCRL | | | | | | | |
```

phase2

键入“学号”：激活*功能，键入学生姓名，首尾字母等显示*；键入“学号-”：取消该功能。

同样的，考虑在 `con_write()` 进行学号的判断。预先将我的学号 2018210547 保存为一个字符串，然后在读入队列中的每个字符时，将其一一与学号字符串比较，若全部匹配，则再判断下一位字符是否是‘-’。若是，则取消*功能，若是其他任意字符，则激活该功能。

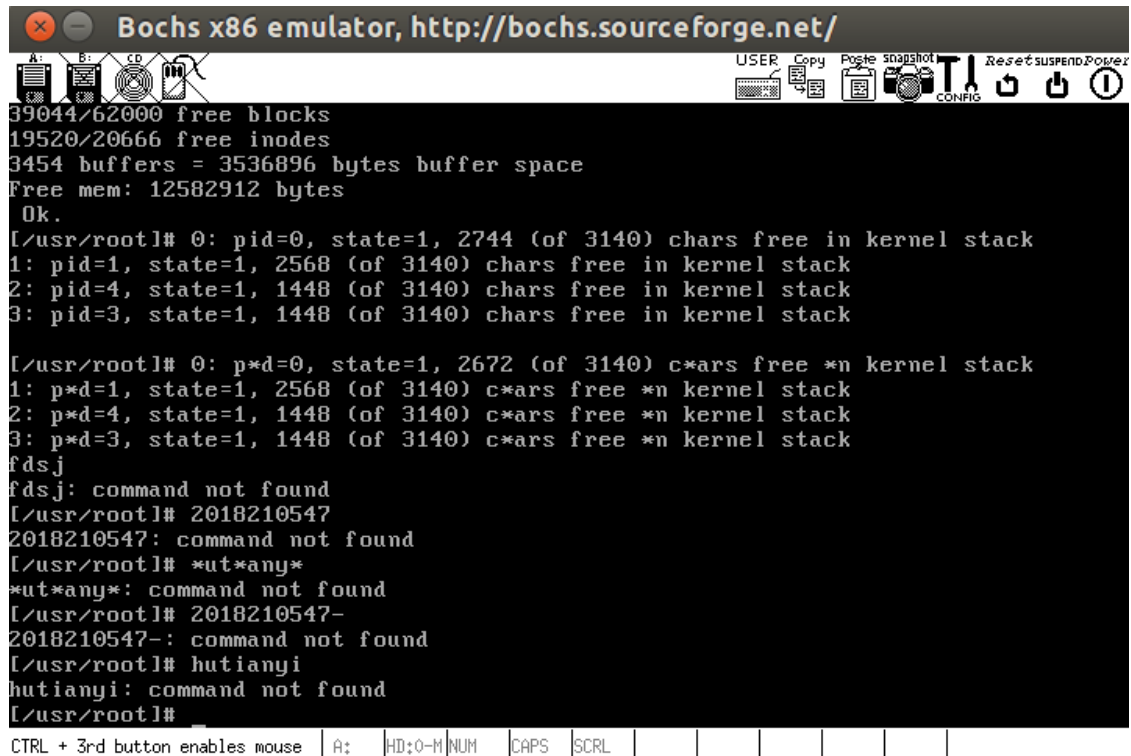
在 `console.c` 的 `con_write()` 中，修改“普通字符”的case下的代码。

```
char match_num[10]="2018210547";
void con_write(struct tty_struct * tty)
{
    int nr, i = 0;
    char c;

    nr = CHARS(tty->write_q);
    while (nr-->0) {
        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 (i == 10){
                        if (c == '-')
                            f12Flag = 0;
                        else f12Flag = 1;
                        i = 0;
                    }
                    if (c == match_num[i]) i++;
                }
            }
        }
    }
}
```

```
        else i = 0;
    if(f12Flag && (c == 'h' || c == 'i'))
        c='*';
    //后面的代码不变，略去
```

- 运行结果



```
Bochs x86 emulator, http://bochs.sourceforge.net/
39044/62000 free blocks
19520/20666 free inodes
3454 buffers = 3536896 bytes buffer space
Free mem: 12582912 bytes
Ok.
[/usr/root]# 0: pid=0, state=1, 2744 (of 3140) chars free in kernel stack
1: pid=1, state=1, 2568 (of 3140) chars free in kernel stack
2: pid=4, state=1, 1448 (of 3140) chars free in kernel stack
3: pid=3, state=1, 1448 (of 3140) chars free in kernel stack

[/usr/root]# 0: p*d=0, state=1, 2672 (of 3140) c*ars free *n kernel stack
1: p*d=1, state=1, 2568 (of 3140) c*ars free *n kernel stack
2: p*d=4, state=1, 1448 (of 3140) c*ars free *n kernel stack
3: p*d=3, state=1, 1448 (of 3140) c*ars free *n kernel stack
fdsj
fdsj: command not found
[/usr/root]# 2018210547
2018210547: command not found
[/usr/root]# *ut*any*
*ut*any*: command not found
[/usr/root]# 2018210547-
2018210547-: command not found
[/usr/root]# hutianyi
hutianyi: command not found
[/usr/root]#
```

五、总结体会

在本次“键盘驱动程序的分析与修改”实验中，我结合了理论课学习的Unix I/O基础知识与实践课学习到的VMware与Linux的操作方法，以及汇编语言与C语言的语法知识，成功地完成了一次对Linux内核的实践探究。实验过程中，我将理论知识应用于实践，得到了进一步的掌握与巩固。

通过这次实验练习，我对通过使用I/O系统调用函数有了初步的体验。在分析汇编程序，查找对应的操作函数的过程中遇到了不少问题。虽然在理论上学习过了Unix I/O的原理，但一开始，面对冗杂的汇编代码，我无从下手。通过对于keyboard.S和console.c执行过程的反复分析及进一步地学习内核地运作方式，我才熟悉了输入字符地处理流程。

这次实验我总共花了将近一周的时间，虽然过程是艰辛的，但是结果确是让人回味的。最后，衷心地感谢所有给予我帮助的老师 and 同学们！