

# 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： 2020211314

学 号： 2020211616

姓 名： 付容天

学 院： 计算机学院（国家示范性软件学院）

2020 年 12 月 4 日

## 一、实验目的

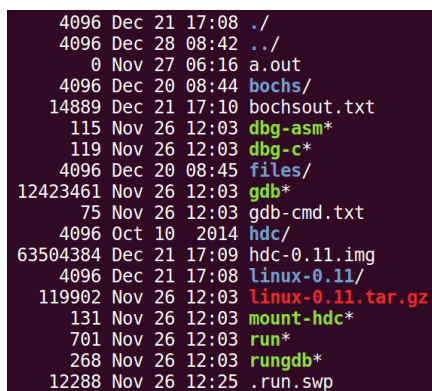
1. 理解 I/O 系统调用函数和 C 标准 I/O 函数的概念和区别；
2. 建立内核空间 I/O 软件层次结构概念，即与设备无关的操作系统软件、设备驱动程序和中断服务程序；
3. 了解 Linux-0.11 字符设备驱动程序及功能，初步理解控制台终端程序的工作原理；
4. 通过阅读源代码，进一步提高 C 语言和汇编程序的编程技巧以及源代码分析能力；
5. 锻炼和提高对复杂工程问题进行分析的能力，并根据需求进行设计和实现的能力。

## 二、实验环境

1. 硬件：学生个人电脑（x86-64）
2. 软件：Windows 10, VMware Workstation 16 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 run gdb*
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“学生本人学号”：激活\*功能，键入学生本人姓名拼音，首尾字母等显示\*  
比如：zhangsan，显示为：\*ha\*gsa\*，  
再次键入“学生本人学号-”：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：

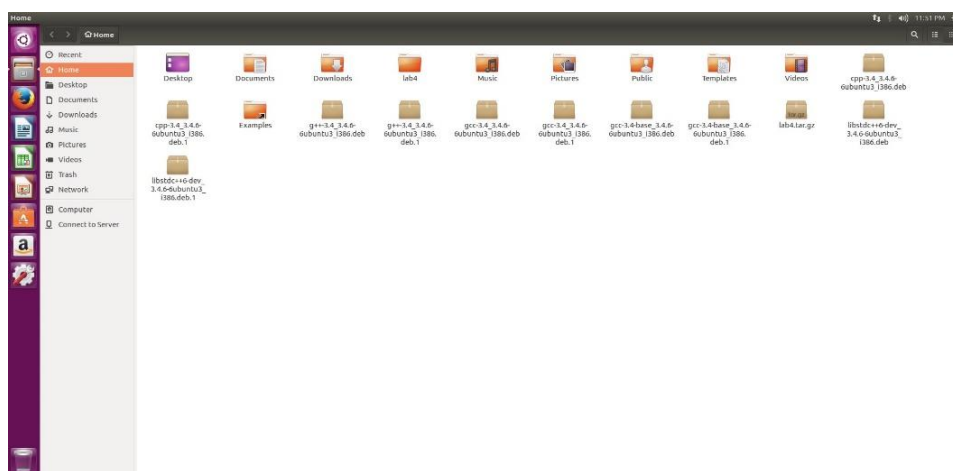
- ✧ 在 C 语言源程序层面进行修改
- ✧ 在汇编语言源程序层面进行修改

实验 4 的其他说明见 lab4.pdf 课件和爱课堂中虚拟机环境搭建相关内容。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

### 第零阶段

第零阶段将完成实验环境的配置，先后下载实验指导上列出的文件，并进行相应的设置，很快就完成了实验环境的配置，截图如下：



(图 1-阶段 0-1)

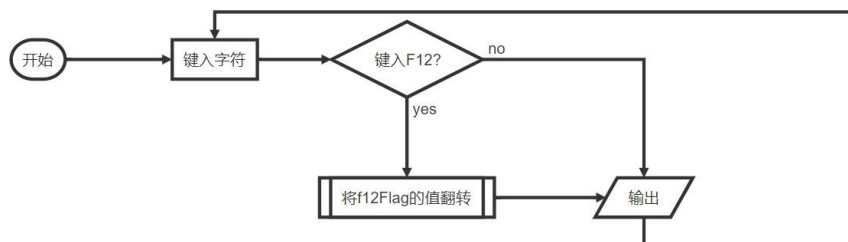
```
rongtianfu@ubuntu:~$ tar -xzf lab4.tar.gz
rongtianfu@ubuntu:~$ ls
Desktop      lab4
Documents   lab4.tar.gz
Downloads   libstdc++6-dev_3.4.6-6ubuntu3_i386.deb
Examples     Music
g++-3.4_3.4.6-6ubuntu3_i386.deb  Pictures
gcc-3.4_3.4.6-6ubuntu3_i386.deb  Public
gcc-3.4-base_3.4.6-6ubuntu3_i386.deb  Templates
libstdc++6-dev_3.4.6-6ubuntu3_i386.deb  Videos

rongtianfu@ubuntu:~$ cd lab4
rongtianfu@ubuntu:~/lab4$ ls
a.out      dbg-asm  gdb      hdc-0.11.img      mount-hdc
bochs      dbg-c   gdb-cmd.txt  linux-0.11      run
bochsout.txt  files  hdc      linux-0.11.tar.gz  run gdb
total 74348
drwxrwxr-x 6 rongtianfu rongtianfu 4096 Dec 4 2019 ./
drwxr-xr-x 17 rongtianfu rongtianfu 4096 Dec 3 17:10 ../
-rw-rw-r-- 1 rongtianfu rongtianfu 0 Nov 26 2018 a.out
drwxr-xr-x 2 rongtianfu rongtianfu 4096 Dec 4 2019 bochs/
-rw-rw-r-- 1 rongtianfu rongtianfu 15376 Dec 4 2019 bochsout.txt
-rwxrwxr-x 1 rongtianfu rongtianfu 115 Nov 25 2018 dbg-asm*
-rwxrwxr-x 1 rongtianfu rongtianfu 119 Nov 25 2018 dbg-c*
drwxrwxr-x 2 rongtianfu rongtianfu 4096 Dec 4 2019 files/
-rwxrwxr-x 1 rongtianfu rongtianfu 12423461 Nov 25 2018 gdb*
-rw-rw-r-- 1 rongtianfu rongtianfu 75 Nov 25 2018 gdb-cmd.txt
drwxr-xr-x 2 rongtianfu rongtianfu 4096 Oct 10 2014 hdc/
-rw-r--r-- 1 rongtianfu rongtianfu 63504384 Dec 4 2019 hdc-0.11.img
drwxrwxr-x 10 rongtianfu rongtianfu 4096 Dec 4 2019 linux-0.11/
-rw-rw-r-- 1 rongtianfu rongtianfu 119902 Nov 25 2018 linux-0.11.tar.gz
-rwxrwxr-x 1 rongtianfu rongtianfu 131 Nov 25 2018 mount-hdc*
-rwxrwxr-x 1 rongtianfu rongtianfu 701 Nov 25 2018 run*
-rwxrwxr-x 1 rongtianfu rongtianfu 268 Nov 25 2018 run gdb*
-rw-r--r-- 1 rongtianfu rongtianfu 12288 Nov 25 2018 .run.swp
rongtianfu@ubuntu:~/lab4$
```

(图 2-阶段 0-2)

## 第一阶段

第一阶段需要实现这样一个功能：键入 F12 键之后，输入的字母中含有姓名首尾字符的字母显示为 \* 号，而且该过程是可逆的，即：再次键入 F12 键的时候将恢复原先状态。状态图如下：



(图 3-阶段 1-1)

根据《linux 内核完全注释》（本实验报告后简称其为《注释》）一书的第 426 页关于控制台的描述可以看到控制台驱动程序实际上涉及到 keyboard.S 和 console.c 两个程序，每当键入字符的时候，调用 keyboard.S 程序，将键入字符其放入缓冲队列 read\_q 中，经过转换后放入辅助缓冲队列 secondary，之后再由 console.c 程序实现键入字符的显示。因此为了实现我们上述流程图中的功能，可以考虑在 keyboard.S 函数中对键入字符进行检测，如果键入的字符是 F12 键，那么就将表示 F12 的 f12Flag 的值进行翻转。根据《注释》一书第 438 页关于 keyboard.S 程序的描述来看，其中的 func 函数将处理键入字符为 F12 的情况，实验中的 func 函数截图如下：

```
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
```

(图 4-阶段 1-2)

查找资料可知 F12 的扫描码为 0x58，F11 的扫描码为 0x57。键入字符的扫描码存储在寄存器 al 中。上述 func 函数代码段的第 8 行和第 12 行减去定值 59 和定值 18 是为了后面根据功能键的索引号来直观地判断键入的字符。第 13 行判断键入的字符是否是 F11，如果不是则不进行任何处理（没有说明键入的字符是 F11 时进行何种操作）；第 15 行判断键入的字符是否是 F12，如果不是则不进行任何处理（没有说明键入的字符是 F12 时进行何种操作），因此在该汇编代码的后面增添上如上图 4 中所示的语句 call change\_f12Flag，该语句将在进入字符为 F12 的时候被调用，目的是改变 f12Flag 的值。

change\_f12Flag 函数应存储在 console.c 程序中，并且还要在 console.c 程序中改变显示在屏幕上的字符，打开 console.c 程序并在其中新增全局变量 f12Flag 和函数 change\_f12Flag 如下：

```
int f12Flag = 0;

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

(图 5-阶段 1-3)

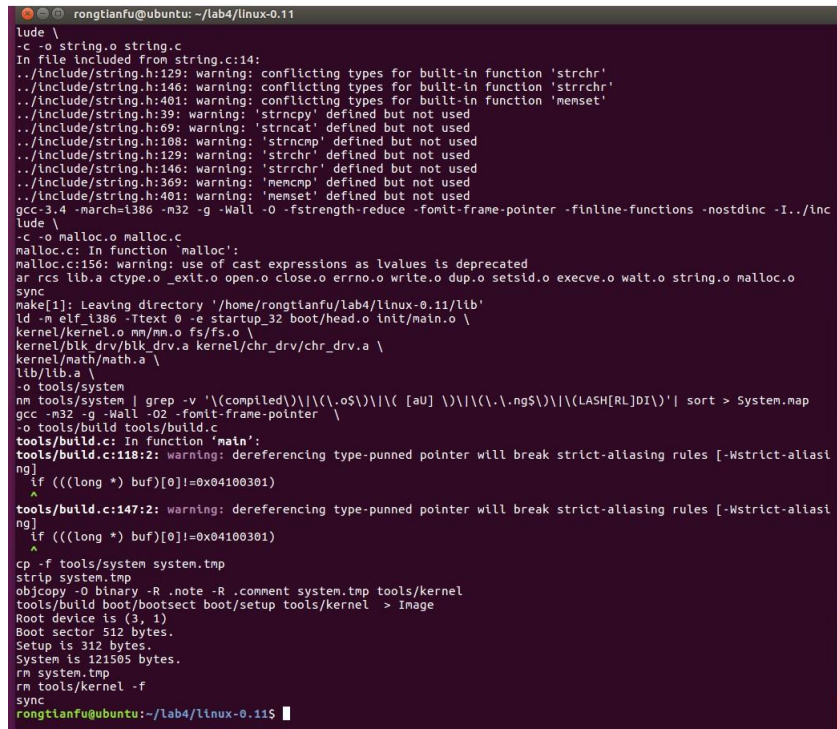
上述的全局变量 f12Flag 标志了当前状态是否需要将姓名首字符显示为 \* 号，函数 change\_f12Flag

实现了将变量 f12Flag 的值进行翻转的功能。现在根据《注释》一书第 462 页关于 con\_write 的描述，我们可以在此函数中实现输出功能的改变，只需要添加如下语句：

```
if(f12Flag && ((c>64 && c<91)||c>96 && c<123))&&(c == 'f' || c == 'r' || c == 'u' || c == 'n'))c = '*';
```

(图 6-阶段 1-4)

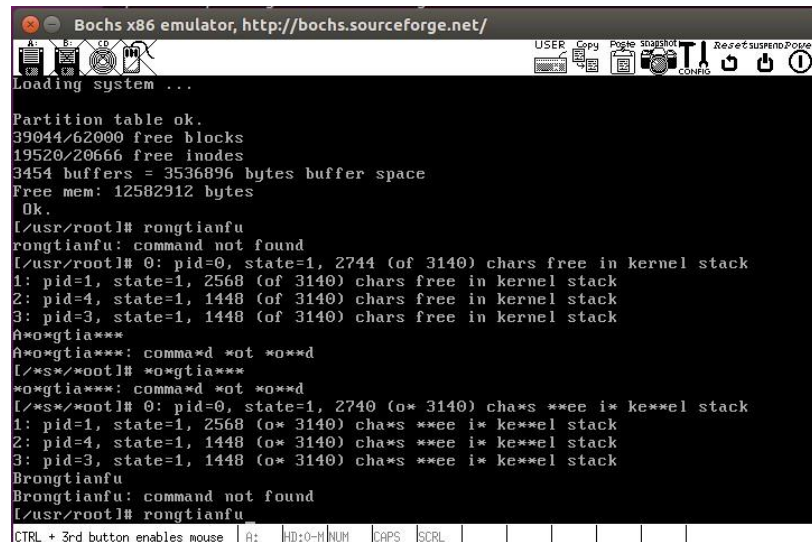
该 if 语句将 f12Flag 的值为 1、键入字符为字母、且字母为姓名首尾字母（付容天，r 与 n 与 f 与 u）的情况下的 c 的值改为 \* 号的编码，因此在后续输出的时候，将会输出 \* 号而不是字母 r 或者 n 或者 f 或者 u。然后在路径 linux-0.11 下使用 make clean 命令和 make 命令可以生成相应的 image 文件，如下所示：



```
lunde \
-c -o string.o string.c
In file included from string.c:14:
../include/string.h:129: warning: conflicting types for built-in function 'strchr'
../include/string.h:146: warning: conflicting types for built-in function 'strchr'
../include/string.h:491: warning: conflicting types for built-in function 'memset'
../include/string.h:39: warning: 'strncpy' defined but not used
../include/string.h:69: warning: 'strncat' defined but not used
../include/string.h:108: warning: 'strncpy' defined but not used
../include/string.h:129: warning: 'strchr' defined but not used
../include/string.h:146: warning: 'strchr' defined but not used
../include/string.h:369: warning: 'memcpy' defined but not used
../include/string.h:491: warning: 'memset' defined but not used
gcc-3.4 -march=i386 -m32 -g -Wall -O -fstrength-reduce -fontt-frame-pointer -finline-functions -nostdinc -I../inc
lunde \
-c -o malloc.o malloc.c
malloc.c: In function 'malloc':
malloc.c:156: warning: use of cast expressions as lvalues is deprecated
ar rcs lib.a ctype.o _exit.o open.o close.o errno.o write.o dup.o setsid.o execve.o wait.o string.o malloc.o
sync
make[1]: Leaving directory '/home/rongtianfu/lab4/linux-0.11/lib'
ld -m elf_i386 -Ttext 0 -e startup_32 boot/head.o init/main.o \
kernel/kernel.o mm/mm.o fs/fs.o \
kernel/blk_drv/blk_drv.a kernel/chr_drv/chr_drv.a \
kernel/math/math.a \
lib/lib.a \
-o tools/system
nm tools/system | grep -v '\(compiled\)\\(\\.o$\)\\( [aU] \\)\\(\\.ng$\)\\(LASH[RL]DI\)\\)' | sort > System.map
gcc -m32 -g -Wall -O2 -fontt-frame-pointer \
-o tools/build tools/build.c
tools/build.c: In function 'main':
tools/build.c:118:2: warning: dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing]
ng)
if (((long *) buf)[0])=0x04100301)
^
tools/build.c:147:2: warning: dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing]
ng)
if (((long *) buf)[0])=0x04100301)
^
cp -f tools/system system.tmp
strip system.tmp
objcopy -O binary -R .note -R .comment system.tmp tools/kernel
tools/build boot/bootsect boot/setup tools/kernel > image
Root device is (3, 1)
Boot sector 512 bytes.
Setup is 312 bytes.
System is 121505 bytes.
rm system.tmp
rm tools/kernel -f
sync
rongtianfu@ubuntu:~/lab4/linux-0.11$
```

(图 7-阶段 1-5)

然后路径 lab4 下使用指令 ./run 可以启动 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，实验截图如下：



```
Bochs x86 emulator, http://bochs.sourceforge.net/
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]# rongtianfu
rongtianfu: command not found
[/usr/root]# 0: pid=0, state=1, 2744 (of 3140) chars free in kernel stack
1: pid=1, state=1, 2560 (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
A*ongtia***
A*ongtia***: comma*d *ot *o***
[/usr/root]# *ongtia***
*ongtia***: comma*d not *o***
[/usr/root]# 0: pid=0, state=1, 2740 (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
Brongtianfu
Brongtianfu: command not found
[/usr/root]# rongtianfu
```

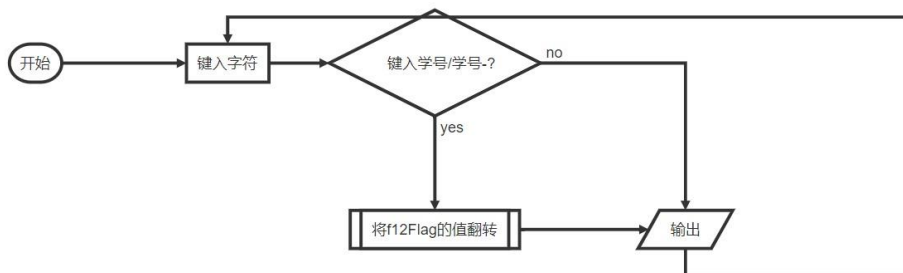
(图 8-阶段 1-6)



从上图 8 可以看到，一开始可以正常输入“rongtianfu”，但是键入 F12 键之后，输入“rongtianfu”就只能显示出“\*ongtia\*\*\*”，并且再次键入 F12 之后，又能正常显示出“rongtianfu”了。第一阶段实验结束。

## 第二阶段

本阶段需要实现这样一个功能：键入本人学号“2020211616”之后，输入的字母中含有姓名首字符的字母显示为 \* 号，而且该过程是可逆的，即：键入“2020211616-”的时候将恢复原先状态。状态图如下：



(图 9-阶段 2-1)

实际上，本阶段的实现思路和第一阶段的实现思路很相近。第一阶段是键入 F12 键之后触发 \* 功能，第二阶段是键入 2020211616 之后触发 \* 功能。在第一阶段中我们设置了变量 f12Flag 来记下 \* 功能是否应该被触发，在第二阶段中，我们只需要对第一阶段的代码进行一些小的改动即可：借用 f12Flag 的值作为是否触发 \* 功能的判断标志，并且引入新的全局变量 count 来辅助 f12Flag 的值是否需要被修改。

下面考虑如何对 f12Flag 的值进行修改。由于输入是一个一个字符进行的，所以设置数组来存储学号并不实际，如何一位一位地判断输入的字符是否是学号的一部分呢？考虑到输入的数字的顺序是确定的，那么我们实际上就可以以此思路为基础写出如下代码：

```
if (c == '2' && (count == 1 || count == 3 || count == 5)) count++;
else if (c == '0' && (count == 2 || count == 4)) count++;
else if (c == '1' && (count == 6 || count == 7 || count == 9)) count++;
else if (c == '6' && count == 8) count++;
else if (c == '6' && count == 10){
    f12Flag = 1;
    count++;
}
else if (c == '-' && count == 11){
    f12Flag = 0;
    count = 1;
}
else count = 1;
```

(图 10-阶段 2-2)

上图 10 中的 count==10 所在的 if 语句用来修改 f12Flag 的值以触发 \* 功能；count==11 所在的 if 语句用来修改 f12Flag 的值以解除 \* 功能；如果没有最后一个 else 语句，那么就会出现这样一种情况：输入的学号 2020211616 不连续（比如输入 20202116176）的时候仍然会触发 \* 功能，设置了该条 else 语句便可以避免这种情况的出现。将 linux 文件重新初始化并按照上面的思路对 console.c 文件进行修改如下：

```
static unsigned long ques=0;
static unsigned char attr=0x07;

static void sysbeep(void);

int f12Flag = 0;
int count = 1;

/*
 * this is what the terminal answers to a ESC-Z or csi0c
 * query (= vt100 response).
```

(图 11-阶段 2-3)

```

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 == 1 || count == 3 || count == 5)) count++;
                    else if (c == '0' && (count == 2 || count == 4)) count++;
                    else if (c == '1' && (count == 6 || count == 7 || count == 9)) count++;
                    else if (c == '6' && count == 8) count++;
                    else if (c == '6' && count == 10)
                        {
                            f12Flag = 1; count++;
                        }
                    else if (count == 11 && c == '.')
                        {
                            f12Flag = 0; count = 1;
                        }
                    else count = 1;

                    if(f12Flag && ((c>64 && c<91)||((c>96 && c<123))&&(c == 'f' || c == 'r' || c == 'u' || c == 'n')) c = '*';

                    asm__ ("movb attr,%0\n\t"
                        "movw %0,%1\n\t"

```

(图 12-阶段 2-4)

接下来退回到 linux-0.11 下使用 make clean 命令和 make 命令可以生成相应的 image 文件，并在路径 lab4 下使用指令 ./run 可以启动 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，实验截图如下：

```

rongtianfu@ubuntu: ~/lab4/linux-0.11
-c -o execve.o execve.c
gcc-3.4 -march=i386 -m32 -g -Wall -O -fstrength-reduce -fomit-frame-pointer -finline-functions -nostdinc -I../include \
-c -o wait.o wait.c
gcc-3.4 -march=i386 -m32 -g -Wall -O -fstrength-reduce -fomit-frame-pointer -finline-functions -nostdinc -I../include \
-c -o string.o string.c
In file included from string.c:14:
../include/string.h:129: warning: conflicting types for built-in function 'strchr'
../include/string.h:146: warning: conflicting types for built-in function 'strchr'
../include/string.h:481: warning: conflicting types for built-in function 'memset'
../include/string.h:39: warning: 'strncpy' defined but not used
../include/string.h:69: warning: 'strncat' defined but not used
../include/string.h:108: warning: 'strncmp' defined but not used
../include/string.h:129: warning: 'strchr' defined but not used
../include/string.h:146: warning: 'strchr' defined but not used
../include/string.h:369: warning: 'memcmp' defined but not used
../include/string.h:481: warning: 'memset' defined but not used
gcc-3.4 -march=i386 -m32 -g -Wall -O -fstrength-reduce -fomit-frame-pointer -finline-functions -nostdinc -I../include \
-c -o malloc.o malloc.c
malloc.c: In function 'malloc':
malloc.c:156: warning: use of cast expressions as lvalues is deprecated
ar rcs lib.a ctype.o _exit.o open.o close.o errno.o write.o dup.o setsid.o execve.o wait.o string.o malloc.o
sync
make[1]: Leaving directory '/home/rongtianfu/lab4/linux-0.11/lib'
ld -m elf_i386 -Ttext 0 -e startup_32 boot/head.o init/main.o \
kernel/kernel.o m/mem.o fs/fs.o \
kernel/bk_drv/bk_drv.o kernel/chr_drv/chr_drv.o \
kernel/math/math.o \
lib/lib.a \
-o tools/system
nm tools/system | grep -v '\(compiled\)\|\(l.o\)\|\( [aU] \)\|\(l.o\)\|ng\)\|\(LASH[RL]DI\)' | sort > System.map
gcc -m32 -g -Wall -O2 -fomit-frame-pointer \
-o tools/build tools/build.c
tools/build.c: In function 'main':
tools/build.c:118:2: warning: dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing]
    if (((long *) buf)[0]) != 0x04100301)
    ^
tools/build.c:147:2: warning: dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing]
    if (((long *) buf)[0]) != 0x04100301)
    ^
cp -f tools/system system.tmp
strip system.tmp
objcopy -O binary -R .note -R .comment system.tmp tools/kernel
tools/build boot/bootsect boot/setup tools/kernel > image
Root device is (3, 1)
Boot sector 512 bytes.
Setup is 312 bytes.
System is 121505 bytes.
rm system.tmp
rm tools/kernel -f
sync
rongtianfu@ubuntu:~/lab4/linux-0.11$

```

(图 13-阶段 2-5)

```

Bochs x86 emulator, http://bochs.sourceforge.net/
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: 12502912 bytes
OK.
[/usr/root]# rongtianfu
rongtianfu: command not found
[/usr/root]# 2020211616*o*gtia***
2020211616*o*gtia***: command not found
[/usr/root]# *o*gtia***
*o*gtia***: command not found
[/usr/root]# 2020211616-rongtianfu
2020211616-rongtianfu: command not found
[/usr/root]# rongtianfu

```

(图 14-阶段 2-6)

从上图 14 中可以看出：一开始可以正常输入“rongtianfu”，但是一旦键入“2020211616”，那么字母 r 和字母 n 和字母 f 和字母 u 立刻以 \* 号显示，并且再次键入“rongtianfu”的时候，仍以 \* 显示字母 r 和字母 n 和字母 f 和字母 u，直到键入“2020211616-”的时候，字母 r 和 f 立刻回到了正常的显示状态。

## 五、总结体会

这次试验感觉难度中等，花费的时间总计约 5.5 小时。

先来说一说我在实验中遇到的一些问题。一开始配置实验环境的时候，我的 linux 虚拟机不能进行 Paste 操作，这给我带来了很大的困惑。但是重启虚拟机便解决了这个问题。在第一阶段，我一开始没有参照《注释》一书来读代码，因此寻找处理键入 F12 的函数花费了不少的时间，后来参考了《注释》一书的相关内容就很快找到了函数 func。

所有的技术细节都是显然的，唯独需要不断的调试和尝试。一开始我的全局变量设置位置不对，导致无法生成 image 文件，后来尝试了多个位置解决了这个问题。在第二阶段，一开始我以为已经成功完成了实验任务，但是我发现了这样一种情况：输入的学号 2020211616 不连续（比如输入 20202116176）的时候仍然会触发 \* 功能，因此我设计了图 10 中的最后一个 else 语句来处理这种情况。这个问题让我明白，多思考特殊情况、多处理特殊情况，能够增强代码的健壮性，这是非常重要的。

作为计算机系统基础课的最后一个实验，我学习了 linux 的相关知识，并且动手在虚拟机上对 linux 的源代码进行修改，即检测了对汇编代码的掌握程度，又增强了对结合不同文件最终生成 image 文件的认识。在实验中，当我在 linux 的源代码中看到注释中的“(C) 1991 Linus Torvalds”的时候，我突然有一种感觉：就像是和来自 1991 年的大二学生 Linus 在对话，仿佛我们在共同完成同一个任务，虽然科技高速发展，但是这种底层的代码却始终不曾变化，不论是十年之后、百年之后还是千年之后，一旦人们打开 linux 的源代码，就立刻能看到“(C) 1991 Linus Torvalds”，这就像是来自 1991 年的问候。不论过去多少年，我们都能够亲手触摸到真正的历史，大概这就是科技的温度吧！