

## 一、实验目的

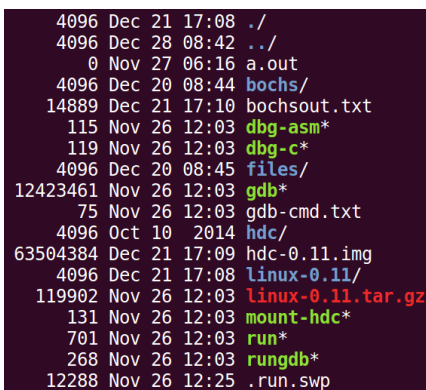
1. 理解 I/O 系统调用函数和 C 标准 I/O 函数的概念和区别；
2. 建立内核空间 I/O 软件层次结构概念，即与设备无关的操作系统软件、设备驱动程序和中断服务程序；
3. 了解 Linux-0.11 字符设备驱动程序及功能，初步理解控制台终端程序的工作原理；
4. 通过阅读源代码，进一步提高 C 语言和汇编程序的编程技巧以及源代码分析能力；
5. 锻炼和提高对复杂工程问题进行分析的能力，并根据需求进行设计和实现的能力。

## 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

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。

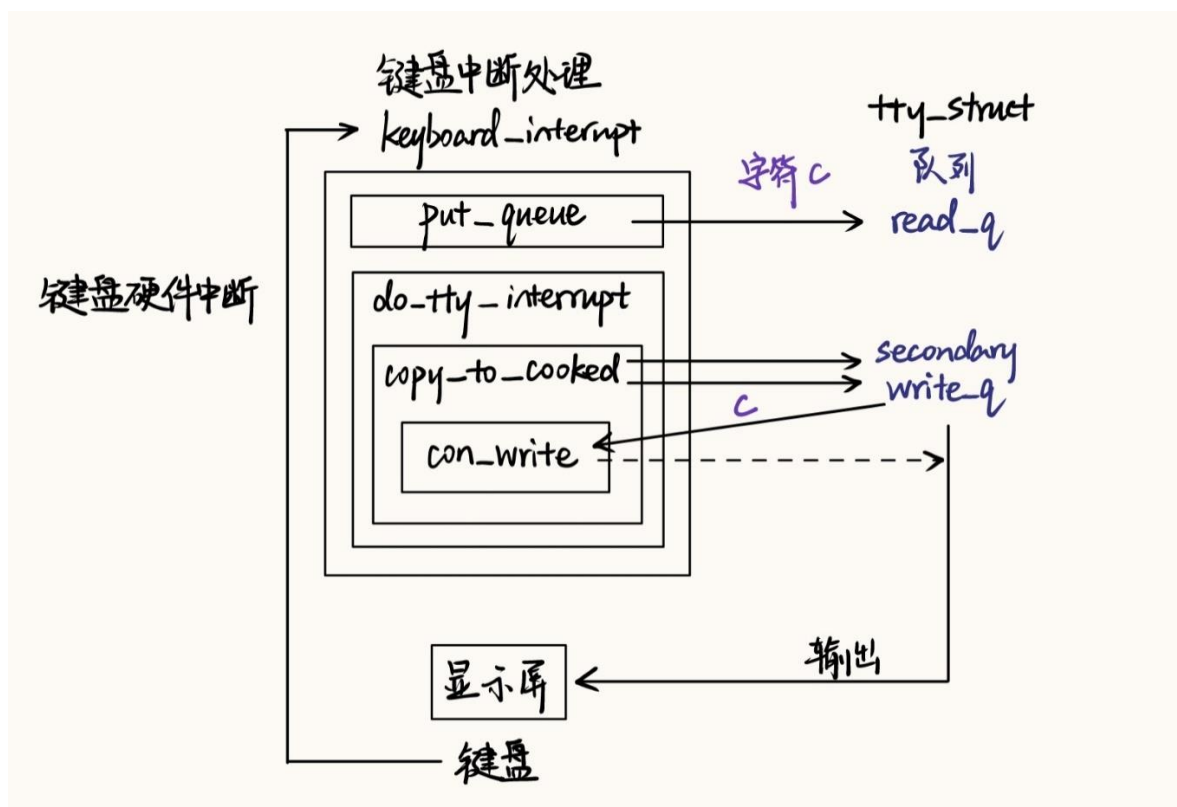
tty\_io.c: 包含 tty 字符设备读函数 tty\_read() 函数和写函数 tty\_write(), 为文件系统提供了上层访问接口。tty\_struct 中定义了读缓冲队列 (read\_q), 写缓冲队列 (write\_q) 和辅助缓冲队列 (secondary);

console.c: 控制台相关功能，主要包含控制台初始化程序和控制台写函数 con\_write() （管理所有控制字符和换码字符序列，这些字符给应用程序提供全部的屏幕管理操作）；

keyboard.S: 包含了键盘输入的底层汇编代码，用于与硬件交互，主要实现了键盘中断处理过程（异步异常）keyboard\_interrupt;

一次按键操作引起键盘中断，中断处理程序 keyboard\_interrupt 首先读取当前按键的扫描码，然后判断是否为特殊扫描码 0xe0 或 0xe1，都不是则调用跳转表 key\_table 中的子程序，把扫描码对应的字符放入 read\_q 中。之后调用 do\_tty\_interrupt，它直接调用 copy\_to\_cooked，把 read\_q 中的字符经过处理后放到 secondary 中同时把字符放到 write\_q 中，并调用 con\_write。此时如果该终端的回显标志 L\_ECHO 置位，则该字符会显示到屏幕上。

流程图



## Phase 1

### 1. 在 C 语言源程序层面进行修改

查看 console.c

修改:

定义全局变量 f12\_flag

函数 change\_f12\_flag: 每按一次 f12 该值翻转

```
int f12_flag=0;

void change_f12_flag(void)
{
    switch(f12_flag){
        case 1:
            f12_flag=0;
            break;
        case 0:
            f12_flag=1;
            break;
    }
}
```

分析 con\_write:

主要由 switch 语句组成, 每次处理一个字符。在正常方式下, 显示字符使用当前属性直接写到显示内存中。该函数会从终端 tty\_struct 结构的写队列 write\_q 中取出字符或字符序列, 然后根据字符的性质 (普通字符/控制字符/转义序列/控制序列), 实现一些屏幕控制操作。

修改:

为了实现键入姓名时首尾字母显示\*, 在 case 0 下普通显示字符的操作下, 添加激活\*功能且字符为首尾字母时, c 赋值为 '\*' 的语句。

```
void con_write(struct tty_struct * tty)
{
    int nr;
    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 (f12_flag && (c == 108 || c == 103))
                        c = '*';
                    if (x >= video_num_columns) {
                        x -= video_num_columns;
                        pos -= video_size_row;
                        lf();
                    }
                    __asm__ ("movb attr, %%ah\n\t"
                           "movw %%ax, %1\n\t"
                           :: "a" (c), "m" (*(short *)pos)
                           );
                    pos += 2;
                    x++;
                }
            }
        }
    }
}
```

### 2. 在汇编语言源程序层面进行修改

查看 keyboard.S

key\_table 是扫描码到对应按键处理程序的转跳表, 观察可知 func 处理 f1~f12

```
.long alt,do_self,caps,func          /* 38-3B alt sp caps f1 */
.long func,func,func,func           /* 3C-3F f2 f3 f4 f5 */
.long func,func,func,func           /* 40-43 f6 f7 f8 f9 */
.long func,num,scroll,cursor        /* 44-47 f10 num scr home */
.long cursor,cursor,do_self,cursor  /* 48-4B up pgup - left */
.long cursor,cursor,do_self,cursor  /* 4C-4F n5 right + end */
.long cursor,cursor,cursor,cursor   /* 50-53 dn pgdn ins del */
.long none,none,do_self,func        /* 54-57 sysreq ? < f11 */
.long func,none,none,none           /* 58-5B f12 ? ? ? */
```

f12 对应扫描码为 0x58

分析 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
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
```

把功能键扫描码与 f1 (0x3b) 做差，偏移量放进 %al

判断完 f1~f10 后又将偏移量减 18

$0x58 - 0x3b - 18 = 11$

```
cmpb $11,%al
ja end_func
```

这里判断 %al 值是否大于 11 即是否为 f12，不是，则不处理，返回

修改：

是 f12 则调用 change\_f12\_flag 函数

```
cmpb $11,%al
jne ok_func
call change_f12_flag
```

运行测试

```
[/usr/root]# liyutong
liyutong: command not found
```

按 f12，启动\*功能

```
[/usr/root]# *iyuton*
```

Phase 2

查看 console.c，修改代码

修改：

定义全局变量：act 激活\*功能标志，cnt 计数，num 记录学号-

```
int act=0;
int cnt=0;
char num[11]="2022211414-";
```

con\_write

为了实现键入姓名时首尾字母显示\*，在 case 0 下普通显示字符的操作下，逐个字符判断输入是否为学号或学号-，并对两种情况修改 act，添加激活\*功能且字符为首尾字母时，c 赋值为 '\*' 的语句。

```

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 (cnt<11 && c==num[cnt])
                        cnt++;
                    else
                        cnt=0;
                    if (cnt==10)
                        act=1;
                    if (cnt==11)
                        act=0;
                    if (act && (c==108 ||c==103))
                        c='*';
                }
            }
        }
    }
}

```

运行测试

```

[/usr/root]# liyutong
liyutong: command not found
[/usr/root]# 2022211414
2022211414: command not found
[/usr/root]# *iyuton*
*iyuton*: command not found
[/usr/root]# 2022211414-
2022211414-: command not found
[/usr/root]# liyutong
liyutong: command not found
[/usr/root]#

```

## 五、总结体会