

设备控制实验

班级 122030704 学号 12208990406 姓名 刘宇轩

目标	目标 3 得分	目标 4 得分
自评分	98	98
批改分		

一、实验目的

- 1、加深对操作系统设备管理基本原理的认识，实践键盘中断、扫描码等概念；
- 2、通过实践掌握 Linux 0.11 对键盘终端和显示器终端的处理过程。
- 3、能够通过自主学习，解决在内核编译运行中的问题。

二、实验结果及分析

```
Ok.
[usr/root]# ls
README      hello      hello.o    linux0.tgz  shoe
gcclib140   hello.c    linux-0.00 mtools.howto shoelace.tar.Z
[usr/root]# **
*****
*****
[****/****]# ls
README      hello      hello.o    linux0.tgz  shoe
gcclib140   hello.c    linux-0.00 mtools.howto shoelace.tar.Z
[usr/root]# **
*****
*****
[****/****]#
```

通过 flag_12 变量记录 F12 的按下的次数，然后通过 con_write 函数中编写对于 flag_12 的判断语句进行对输出 c 的修改，及在输出中是否需要将内容变为*。

三、问题及解决方法

1. 修改全部函数后在 bochs 中按 F12 显示如下：

```

LIS: command not found
[/usr/root]# ls
README      hello      hello.o      linux0.tgz   shoe
gcclib140   hello.c    linux-0.00   mtools.howto shoelace.tar.Z
[/usr/root]# 0: pid=0, state=1, 2672 (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
L0: pid=0, state=1, 2588 (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
L0: pid=0, state=1, 2588 (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
L0: pid=0, state=1, 2588 (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
L
LIS: command not found

```

现在查找前面的代码编写部分，发现在 `/include/linux/tty.h` 文件中，代码应该在在 `endif` 之前

```

d void copy_to_cooked(struct tty_struct * tty);

extern int flag_12;
void press_f12_handle(void);
#endif

```

之后运行还出错，然后查找代码发现在 `kernel/chr_drv/keyboard.S` 文件中修改 `key_table` 代码时将 `.long press_f12_handle,none,none,none` 添加错位，修改后如下：

```

.long cursor,cursor,do_self,cursor /* 4C-4F fn 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 ??? */
.long press_f12_handle,none,none,none
.long none,none,none,none /* 5C-5F ??? */
.long none,none,none,none /* 60-63 ??? */

```

之后编译内核，运行 `bochs`
实验成功

```

Ok.
[/usr/root]# ls
README      hello      hello.o      linux0.tgz   shoe
gcclib140   hello.c    linux-0.00   mtools.howto shoelace.tar.Z
[/usr/root]# **
*****
*****
[/usr/root]# ls
README      hello      hello.o      linux0.tgz   shoe
gcclib140   hello.c    linux-0.00   mtools.howto shoelace.tar.Z
[/usr/root]# **
*****
*****
[/usr/root]#

```

四、实验思考

1. 在原始代码中，按下 `F12`，中断响应后，中断服务程序会调用 `func`？它实现的是什么功能？
会调用 `func`。

显示当前运行的进程状态。输出显示了多个进程的状态，包括进程 ID (pid)、状态 (state)、以及内核栈 (kernel stack) 的使用情况。

2. 在你的实现中，是否把向文件输出的字符也过滤了？如果是，那么怎么能只过滤向终端输出的字符？如果不是，那么怎么能把向文件输出的字符也一并进行过滤？

在我的实现中，过滤功能仅针对终端输出的字符，而不会影响向文件的输出。因为我在 `con_write` 函数中添加了判断逻辑，仅在输出到终端时才进行字符替换。要实现对文件输出的字符过滤，可以在文件系统的写操作函数中添加类似的逻辑判断，及是可以在 `put_queue` 系统调用的相关实现中加入对输出字符的检查和替换逻辑。

五、实验步骤

1. 添加 F12 功能键盘处理

1.1 修改 kernel/chr_drv/tty_io.c 文件

```
24 }
25
26 int flag_12 = 0;
27 void press_f12_handle(void)
28 {
29     if (flag_12 == 0)
30         flag_12 = 1;
31     else if (flag_12 == 1)
32         flag_12 = 0;
33 }
```

1.2 修改 /include/linux/tty.h 文件

```
void copy_to_cooked(struct tty_struct * tty);
extern int flag_12;
void press_f12_handle(void);
#endif
```

1.3 修改 kernel/chr_drv/keyboard.S 文件

```
.long cursor,cursor,do_self,cursor /* 4C-4F f13 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 ??? */
.long press_f12_handle,none,none,none
.long none,none,none,none /* 5C-5F ??? */
.long none,none,none,none /* 60-63 ??? */
```

2. 修改 linux-0.11/kernel/chr_drv/console.c 文件，修改 `con_write` 函数

```

        if();
    }

    if (flag_12 == 1)
    {
        if ((c>='A'&& c<='Z') || (c>='a'&& c<='z') || (c>='0'&& c<='9'))
            c = '*';
    }

    __asm__ ("movb attr, %%ah\n\t"
            "movw %%ax, %1\n\t"
            :: "a" (c), "m" (*(short *)nos)

```

3. 编译内核，运行 bochs

```

Root device is (3, 1)
Boot sector 512 bytes.
Setup is 312 bytes.
System is 121508 bytes.
rm system.tmp
rm tools/kernel -f
sync
shiyanolou@67612988dbf1c4a4d

```

编译成功

```

Ok.
t
[usr/root]# ls
README      hello      hello.o    linux0.tgz  shoe
gcc1ib140   hello.c    linux-0.00 mtools.howto shoelace.tar.Z
[usr/root]# **
*****
*****
[/**/****]# ls
README      hello      hello.o    linux0.tgz  shoe
gcc1ib140   hello.c    linux-0.00 mtools.howto shoelace.tar.Z
[usr/root]# **
*****
*****
[/**/****]#

```

实验完成

六、实验收获

通过本次实验，我学会了如何修改 Linux 0.11 的终端设备处理代码，以实现
对键盘输入和字符显示的非常规控制，这涉及到对 keyboard.S 和 console.c 文件
的关键修改，并且深入理解了 Linux 0.11 操作系统中键盘终端和显示器终端的处
理过程，包括键盘中断的响应机制和字符输出的精细控制。实验过程中，通过自
主学习和查阅资料，有效解决了遇到的问题，显著的提升了我的自学能力和问题
解决技巧。同时，加深了我对操作系统设备管理基本原理的认识。

并且我也发现了很多好玩的系统内核修改方法和玩法，可以对键盘输入或
者输出控制，编写出自己的一套逻辑。

七、参考资料

<https://cidhy.blog.csdn.net/article/details/127737970>

https://blog.csdn.net/weixin_43166958/article/details/104194920

<https://blog.csdn.net/Watson2016/article/details/72188549>

<https://blog.csdn.net/realfancy/article/details/90544792>

<https://cidhy.blog.csdn.net/article/details/127737970>

https://pubs.opengroup.org/onlinepubs/9699919799/basedefs/V1_chap11.ht

[ml#tag_11](#)