《操作系统实验》Lab6 实验报告

朱恬骅

09300240004 计算机科学与技术

1 实验目标

1. 熟悉字符设备驱动
2. 学习如何分析和修改字符设备驱动程序

2 实验要求

修改keyboard.S和console.c以实现下述两个功能：

1. 在屏幕的右上角显示系统时间，但是不能覆盖显卡信息
2. 使用诸如Ctrl+Shift+F1/F2上下滚动屏幕

3 实验原理

3.1 控制台驱动程序

在Linux 0.11 内核中，终端控制台驱动程序涉及keyboard.S 和console.c 程序。keyboard.S 用于处理用户键入的字符，把它们放入读缓冲队列read_q 中，并调用copy_to_cooked()函数读取read_q 中的字符，经转换后放入辅助缓冲队列secondary。console.c 程序实现控制台终端的输出处理。例如，当用户在键盘上键入了一个字符时，会引起键盘中断响应（中断请求信号IRQ1,对应中断号INT 33），此时键盘中断处理程序就会从键盘控制器读入对应的键盘扫描码，然后根据使用的键盘扫描码映射表译成相应字符，放入tty 读队列read_q 中。然后调用中断处理程序的C 函数do_tty_interrupt()，它又直接调用行规则函数copy_to_cooked()对该字符

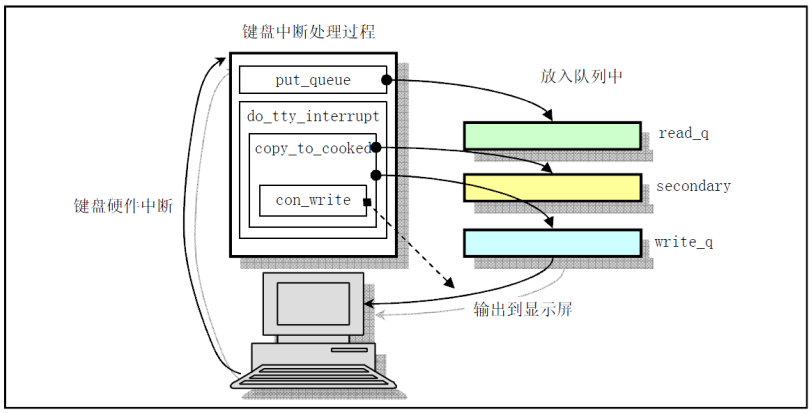

Figure 1: 控制台键盘中断处理过程

进行过滤处理，并放入tty 辅助队列secondary 中，同时把该字符放入tty 写队列write_q 中，并调用写控制台函数con_write()。此时如果该终端的回显（echo）属性是设置的，则该字符会显示到屏幕上。do_tty_interrupt()和copy_to_cooked()函数在tty_io.c 中实现。整个操作过程见图1 所示。

对于进程执行tty 写操作，终端驱动程序是一个字符一个字符进行处理的。在写缓冲队列write_q 没有满时，就从用户缓冲区取一个字符，经过处理放入write_q 中。当把用户数据全部放入write_q 队列或者此时write_q 已满，就调用终端结构tty_struct 中指定的写函数，把write_q 缓冲队列中的数据输出到控制台。对于控制台终端，其写函数是con_write()，在console.c 程序中实现。有关控制台终端操作的驱动程序，主要涉及两个程序。一个是键盘中断处理程序keyboard.S，主要用于把用户键入的字符并放入read_q 缓冲队列中；另一个是屏幕显示处理程序console.c，用于从write_q 队列中取出字符并显示在屏幕上。所有这三个字符缓冲队列与上述函数或文件的关系都可以用图2 清晰地表示出来。

3.2 keyboard.s 程序

该键盘驱动汇编程序主要包括键盘中断处理程序。在英文惯用法中，make 表示键被按下；break 表示键被松开(放开)。该程序首先根据键盘特殊键（例如Alt、Shift、Ctrl、Caps 键）的状态设置程序后面要用到的状态标志变量mode 的值，然后根据引起键盘中断的按键扫描码，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读字符队列（read_q）中。接下来调用C处理函数do_tty_interrupt，该函数仅包含一个对行规程函数copy_to_cooked()的调用。这个行规程函数的主要作用就是

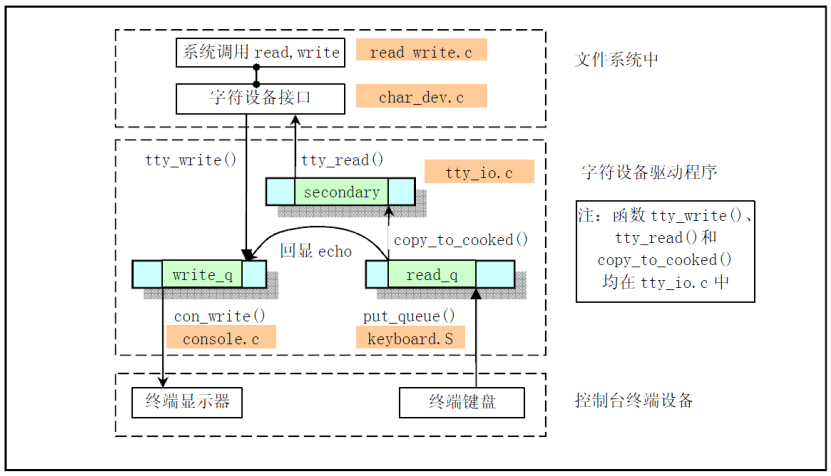

Figure 2: 控制台终端字符缓冲队列以及函数和程序之间的关系

把read_q 读缓冲队列中的字符经过适当处理放入规范模式队列（辅助队列secondary）中，并且在处理过程中，若相应终端设备设置了回显标志，还会把字符放入写队列（write_q）中，从而在终端屏幕上会显示出刚键入的字符。

该程序中需要注意的是模式标识mode的使用。0x4 是模式标志mode 中左ctrl 键对应的比特位(位2)。左shift 键按下，设置mode 中对应的标志位(位0)。另外，key_table 是一张子程序地址跳转表。当取得扫描码后就根据此表调用相应的扫描码处理子程序。大多数调用的子程序是do_self，或者是none，这起决于是按键(make)还是释放键(break)。

3.3 console.c 程序

本文件是内核中最长的程序之一，但功能比较单一。其中的所有子程序都是为了实现终端屏幕写函数con_write()以及进行终端初始化操作。函数con_write()会从终端tty_struct 结构的写缓冲队列write_q 中取出字符或字符序列，然后根据字符的性质（是普通字符还是转义字符序列），把字符显示在终端屏幕上或进行一些光标移动、字符檫除等屏幕控制操作。终端屏幕初始化函数con_init()会根据系统初始化时获得的系统信息，设置有关屏幕的一些基本参数值，用于con_write()函数的操作。

在console.c中，利用引导程序初始化的0x90000开始的一些数据获取显示器的模式、每行的列数等信息，并定义了总的行数。我们需要修改scrup、scrdown以完成滚屏操作。

3.4 回到sched.c

为了获取当前时间，我们需要用到sched.h中定义的CURRENT_TIME。
为了刷新时钟，在sched.c中实现了do_timer()函数。通过修改do_timer()函数，我们能够做到实时更新当前显示的时间。在本次实验中，是用在do_timer中调用console.c中定义的show_timer函数实现的。

4 实验步骤

4.1 添加滚屏和时钟代码

主要思想：在每次上滚/下滚的时候暂时屏蔽时钟，用第一行缓冲区video_topline中的内容替换掉原第一行的内容（即抹掉时钟，恢复被时钟遮蔽的内容），
然后滚屏，再将当前第一行的内容保存到缓冲区，恢复时钟。

在文件kernel/chr_drv/console.c中：

```
static char                *video_topline=NULL; /* LAB6 */

/* LAB 6 [ */
#define TIME_STR_LEN      8
#define abs(x)            (((x)>0)?(x):-(x))
static char time_str[TIME_STR_LEN + 1] = "??:??:??";

static int doing_scr = 0; /* 这是是否正在滚屏的标识。如果
   为，则不更新时钟。1 */

void show_time(void)
{
        if (doing_scr) return;
        unsigned long t;
        unsigned int h = 0, m = 0, s = 0, i = 0;
        t = CURRENT_TIME % (3600*24);
        h = t/3600%24; m = t/60%60; s = t%60;

        time_str[0] = '0' + h/10%10;
        time_str[1] = '0' + h%10;
        time_str[3] = '0' + m/10%10;
        time_str[4] = '0' + m%10;
```

```
        time_str[6] = '0' + s/10%10;
        time_str[7] = '0' + s%10;

        char *ptr = (char *)origin + video_size_row -
           TIME_STR_LEN * 2;
        for (i=0; i<TIME_STR_LEN; i++) {
                *ptr = time_str[i];
                ptr += 2;
        }
}
/* ] LAB6 */

static void scrup(void)
{
        /* LAB6 [ */
        doing_scr = 1;
        char *ptr = (char *)origin;
        int i;
        for (i=0; i<video_size_row; ++i) {
                *ptr=video_topline[i]; ptr++;
        }
        /* ] LAB6 */

        if (video_type == VIDEO_TYPE_EGAC || video_type
            == VIDEO_TYPE_EGAM)
        {
                if (!top && bottom == video_num_lines)
                   {
                        origin += video_size_row;
                        pos += video_size_row;
                        scr_end += video_size_row;
                        if (scr_end > video_mem_end) {
                                __asm__("cld\n\t"
                                        "rep\n\t"
                                        "movsl\n\t"
                                        "movl␣
                                           video_num_columns
                                           ,%1\n\t"
```

```
                        "rep\n\t"
                        "stosw"
                        ::"a" (
                            video_erase_char
                            ),
                        "c" ((
                            video_num_lines
                            -1)*
                            video_num_columns
                            >>1),
                        "D" (
                            video_mem_start
                            ),
                        "S" (origin)
                        );
                scr_end -= origin-
                    video_mem_start;
                pos -= origin-
                    video_mem_start;
                origin =
                    video_mem_start;
            } else {
                __asm__("cld\n\t"
                        "rep\n\t"
                        "stosw"
                        ::"a" (
                            video_erase_char
                            ),
                        "c" (
                            video_num_columns
                            ),
                        "D" (scr_end-
                            video_size_row
                            )
                        );
            }
            set_origin();
        } else {
```

```
                __asm__("cld\n\t"
                        "rep\n\t"
                        "movsl\n\t"
                        "movl_video_num_columns
                            ,%%ecx\n\t"
                        "rep\n\t"
                        "stosw"
                        ::"a" (video_erase_char
                            ),
                        "c" ((bottom-top-1)*
                            video_num_columns
                            >>1),
                        "D" (origin+
                            video_size_row*top),
                        "S" (origin+
                            video_size_row*(top
                            +1))
                        );
        }
}
else            /* Not EGA/VGA */
{
        __asm__("cld\n\t"
                "rep\n\t"
                "movsl\n\t"
                "movl_video_num_columns,%%ecx\n
                    \t"
                "rep\n\t"
                "stosw"
                ::"a" (video_erase_char),
                "c" ((bottom-top-1)*
                    video_num_columns>>1),
                "D" (origin+video_size_row*top)
                    ,
                "S" (origin+video_size_row*(top
                    +1))
                );
}
```

```
        /* LAB6 [ */
        ptr = (char *)origin;
        for (i=0; i<video_size_row; ++i) {
                video_topline[i]=*ptr; ptr++;
        }
        doing_scr = 0;
        /* ] LAB6 */
}

static void scrdown(void)
{
        /* LAB6 [ */
        doing_scr = 1;
        char *ptr = (char *)origin;
        int i;
        for (i=0; i<video_size_row; ++i) {
                *ptr=video_topline[i]; ptr++;
        }
        /* ] LAB6 */

        if (video_type == VIDEO_TYPE_EGAC || video_type
            == VIDEO_TYPE_EGAM)
        {
                __asm__("std\n\t"
                        "rep\n\t"
                        "movsl\n\t"
                        "addl $2,%%edi\n\t"      /* %edi
                            has been decremented by 4
                           */
                        "movl video_num_columns,%%ecx\n
                            \t"
                        "rep\n\t"
                        "stosw"
                        ::"a" (video_erase_char),
                        "c" ((bottom-top-1)*
                            video_num_columns>>1),
                        "D" (origin+video_size_row*
```

```
                        bottom-4),
                    "S" (origin+video_size_row*(
                        bottom-1)-4)
                    );
        }
        else                /* Not EGA/VGA */
        {
                __asm__("std\n\t"
                    "rep\n\t"
                    "movsl\n\t"
                    "addl $2,%%edi\n\t"     /* %edi
                        has been decremented by 4
                        */
                    "movl video_num_columns,%%ecx\n
                        \t"
                    "rep\n\t"
                    "stosw"
                    ::"a" (video_erase_char),
                    "c" ((bottom-top-1)*
                        video_num_columns>>1),
                    "D" (origin+video_size_row*
                        bottom-4),
                    "S" (origin+video_size_row*(
                        bottom-1)-4)
                    );
        }

        /* LAB6 [ */
        ptr = (char *)origin;
        for (i=0; i<video_size_row; ++i) {
                video_topline[i]=*ptr; ptr++;
        }
        doing_scr = 0;
        /* ] LAB6 */
}

/* LAB6 [ */
```

```
void scrup1(void)
{
        doing_scr = 1;
        char *ptr = (char *)origin;
        int i;
        for (i=0; i<video_size_row; ++i) {
                *ptr=video_topline[i]; ptr++;
        }

        origin += video_size_row;
        scr_end += video_size_row;
        if (scr_end > (video_mem_end-4)) {
                __asm__("cld\n\t"
                        "rep\n\t"
                        "movsl\n\t"
                        ::"c" ((video_num_lines-1)*
                           video_num_columns>>1),
                        "D" (video_mem_start),
                        "S" (origin)
                );
                scr_end -= origin-video_mem_start;
                pos -= origin-video_mem_start;
                origin = video_mem_start-video_size_row
                   *(video_num_lines-1);
        }
        set_origin();

        ptr = (char *)origin;
        for (i=0; i<video_size_row; ++i) {
                video_topline[i]=*ptr; ptr++;
        }
        doing_scr = 0;
}

void scrdown1(void)
{
        doing_scr = 1;
        char *ptr = (char *)origin;
```

```
        int i;
        for (i=0; i<video_size_row; ++i) {
                *ptr=video_topline[i]; ptr++;
        }

        origin -= video_size_row;
        scr_end -= video_size_row;
        if (origin < video_mem_start) {
                __asm__("std\n\t"
                        "rep\n\t"
                        "movsl\n\t"
                        ::"c" ((video_num_lines-1)*
                           video_num_columns>>1),
                        "D" (video_mem_end-4),
                        "S" (scr_end-4)
                );
                origin += video_mem_end-scr_end;
                pos += video_mem_end-scr_end;
                scr_end = video_mem_end;
        }
        set_origin();

        ptr = (char *)origin;
        for (i=0; i<video_size_row; ++i) {
                video_topline[i]=*ptr; ptr++;
        }
        doing_scr = 0;
}

/* ] LAB6 */
```

在文件kernel/chr_drv/keyboard.S中：

```
do_f1:
        testb $0x0c,mode
        je 1f
        testb $0x03,mode
        je 1f
        call scrup1
```

```
1:      ret

do_f2:
        testb $0x0c,mode
        je 1f
        testb $0x03,mode
        je 1f
        call scrdown1
1:      ret
```

4.2 更新时钟

在kernel/sched.c中：

```
extern void show_time(void); /* LAB6 */

void do_timer(long cpl)
{
        extern int beepcount;
        extern void sysbeepstop(void);

        if (beepcount)
                if (!--beepcount)
                        sysbeepstop();

        if (cpl)
                current->utime++;
        else
                current->stime++;

        if (next_timer) {
                next_timer->jiffies--;
                while (next_timer && next_timer->
                   jiffies <= 0) {
                        void (*fn)(void);

                        fn = next_timer->fn;
                        next_timer->fn = NULL;
```

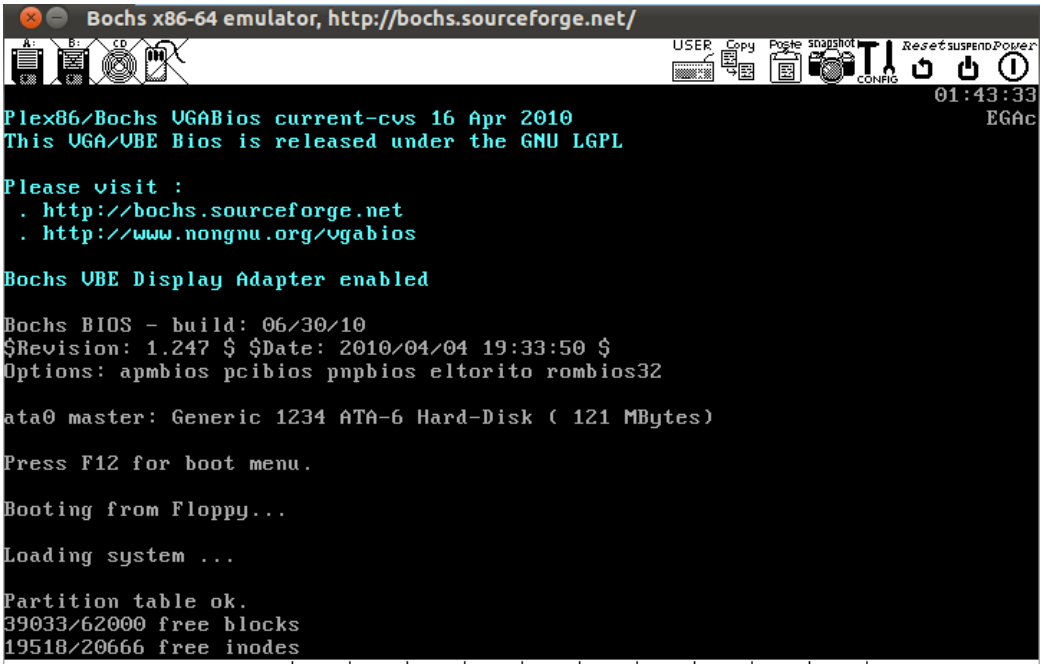

Figure 3: 运行时截图

```
                    next_timer = next_timer->next;
                    (fn)();
                }
        }
        if (current_DOR & 0xf0)
                do_floppy_timer();
        if ((--current->counter)>0) return;
        current->counter=0;
        if (!cpl) return;
        schedule();

        show_time(); /* LAB6 */
}
```

4.3 运行截图

屏幕输出如图3所示。

这就完成了实验。

5 遇到的问题和解决方法

1. 使用time(NULL)函数返回值为负。
   解决：这是一个千年虫问题。将kernel/mktime.c中的year = year - 70
   改成+30。

2. 复制/恢复第一行的时候，会出现乱码。
   解决：有一处多了个*号。另外，一行的大小为两倍的video_num_columns，
   应当用video_size_row。

3. 第一行缓冲区必须完整保存，否则会丢失字符的颜色信息。

4. 对缓冲区的初始化应在显卡信息更新之后进行，否则会丢失第一行的
   内容。

5. 由于bochs的设置问题，bochs中的时钟和外部的时钟并不同步。
   在bxrc中加一句clock: sync=realtime，time0=local即可修正。

6 实验收获

1. 学习了有关字符设备的知识

2. 学习了Linux中关于时间的知识

2011年12月4日