

lab2实验讲义

陈晨曦

2023 年 4 月 1 日

1 lab2中你需要完成的任务

src/myOS/start32.S的编写

src/myOS/i386/io.c的编写

src/myOS/dev/uart.c 和 src/myOS/dev/vga.c的编写

src/myOS/printk/myPrintk.c 和 src/myOS/printk/vsprintf.c 的编写

2 src/myOS/start32.S的编写

我们通过src/multibootheader/multibootHeader.S中的call语句进入src/myOS/start32.S中。

```
1 start:
2     call _start
3     hlt
```

在src/myOS/start32.S的_start中，我们可以看到其进入了一个名叫”establish_stack”的栈初始化建立程序中。

```
1 _start:
2     jmp establish_stack
```

需要完成的事情就是对”establish_stack”的填写。（根据myOS.ld文件进行填写?????的具体值）

```
1 establish_stack:
2     movl ?????, %eax # eax = end of bss/start of heap
3     addl $STACK_SIZE, %eax # make room for stack
4     andl $0xfffffc0, %eax # align it on 16 byte boundary
5     movl %eax, %esp # set stack pointer
6     movl %eax, %ebp # set base pointer
```

自行了解”zero_bss”的内容。（无需填写，关键就是对汇编指令repne和stosl的理解）

```
1 zero_bss:
2     cld # make direction flag count up
3     movl $_end, %ecx # find end of .bss
4     movl $_bss_start, %edi # edi = beginning of .bss
5     subl %edi, %ecx # ecx = size of .bss in bytes
6     shrl %ecx # size of .bss in longs
7     shrl %ecx
8     xorl %eax, %eax # value to clear out memory
9     repne # while ecx != 0
10    stosl # clear a long in the bss
```

3 src/myOS/i386/io.c的编写

在src/i386/io.c中，我们通过C语言的asm内嵌式汇编完成IO端口的调用（即inb和outb函数）。inb是从相应的port获取返回值value（8bit数）。outb是将相应的值value（8bit数）送到相应的port里。

```
1 unsigned char inb(unsigned short int port_from){
2     unsigned char value;
3     __asm__ __volatile__ ("inb %w1,%0"::"a"(value):"Nd"(port_from));
4     return value;
5 }
6
7 void outb (unsigned short int port_to, unsigned char value){
8     __asm__ __volatile__ ("outb %b0,%w1"::"a" (value),"Nd" (port_to));
9 }
```

4 src/myOS/dev/uart.c 和 src/myOS/dev/vga.c的编写

在src/myOS/dev/uart.c我们将调用src/myOS/i386/io.c中的inb和outb函数，来实现UART串口的数据传输。UART串口的port的值为0x3F8，所以我们要做的事情就是“向port=0x3F8发送数据”和“从port=0x3F8获取数据”。

```
1 void uart_put_char(unsigned char c){
2     //Send ONE CHARACTER to port 0x3F8
3 }
4
5 unsigned char uart_get_char(void){
6     //Get ONE CHARACTER from port 0x3F8
7 }
8
9 void uart_put_chars(char *str){
10    //Send ONE STRING to port 0x3F8
11 }
```

5 src/myOS/dev/vga.c的编写

在src/myOS/dev/vga.c中我们要实现的任务：

更新当前光标的位置update_cursor

获取当前光标的位置get_cursor_position

清屏函数clear_screen

在VGA屏幕上显示字符串append2screen

光标位置是由一个一维偏移量决定的（并不是（x,y）这样子的二维坐标）。在这种情况下，第0行第0列的偏移量是0，第1行第0列的偏移量是80，第2行第0列的偏移量是160。这个一维偏移量由高8bit和低8bit构成。具体地讲：

当我们往port=0x3D4送0x0F后，我们与port=0x3D5交互的就是这个一维偏移量低8bit的数据。

当我们往port=0x3D4送0x0E后，我们与port=0x3D5交互的就是这个一维偏移量高8bit的数据。

我们VGA屏幕是可以显示25行×80列个字符，但是需要注意的是一个字符占16bit（16bit中的一半对应ASCII码，另一半对应颜色）。

关于“在VGA屏幕上显示字符串”编写的注意事项：

当字符串中含有'\n'抑或是光标已经在行尾时，我们需要进行换行操作。

当前光标位于第25行时，如果需要换行，则需要实现VGA屏幕的滚屏操作。

```

1 //global variable: cur_line and cur_column record the current cursor position
2 short cur_line=0;
3 short cur_column=0;
4
5
6 char * vga_init_p=(char *)0xB8000;
7
8 void update_cursor(void){
9     //use global variable cur_line and cur_column tp update the current cursor position
10    //here, we will use the inb and outb to interact with port = 0x3D4 and port = 0x3D5
11 }
12
13 short get_cursor_position(void){
14     //update global variable cur_line and cur_column using the information from port = 0x3D4 and port = 0x3D5
15     //here, we will use the inb and outb to interact with port = 0x3D4 and port = 0x3D5
16 }
17
18
19 void clear_screen(void) {
20     //clean the whole VGA screen
21 }
22
23 void append2screen(char *str,int color){
24     //Send the str to the current cursor position
25 }

```

6 src/myOS/printk/myPrintk.c 和 src/myOS/printk/vsprintf.c 的编写

src/myOS/printk/myPrintk.c文件无需编写，但是在这里，你需要理解C语言的可变参数原理，并掌握va_start，va_end，va_arg的用法。

myPrintk函数和 myPrintf函数的区别在于，它们分别是内核和用户的输出函数，但是在我们的实验中，他们两个的实现方式一致。

在myPrintk函数中，我们利用C语言的可变参数原理，获得va_list argptr变量，并通过vsprintf函数，将格式字符串format和argptr生成正常文本串，存储到kBuf中。再将kBuf中的文本通过append2screen函数显示在VGA屏幕上。举例地将（其中整型变量a和b已经分别提前初始化1和2，）：

format 字符串中的内容为: %d + %d = %d

argptr 中的内容为: a, a, b

那么经过vsprintf后，存储在kBuf中的正常文本为: 1 + 1 = 1

```

1 #include <stdarg.h>
2
3 extern void append2screen(char *str,int color);
4
5 extern int vsprintf(char *buf, const char *fmt, va_list argptr);
6
7 char kBuf[400];
8 int myPrintk(int color,const char *format, ...){
9     int count;
10
11     va_list argptr;
12     va_start(argptr,format);
13
14     count=vsprintf(kBuf,format,argptr);
15
16     append2screen(kBuf,color);
17
18     va_end(argptr);
19 }

```

```

20     return count;
21 }
22
23 char uBuf[400];
24 int myPrintf(int color,const char *format, ...){
25     int count;
26
27     va_list argptr;
28     va_start(argptr,format);
29
30     count=vsprintf(uBuf,format,argptr);
31
32     append2screen(uBuf,color);
33
34     va_end(argptr);
35
36     return count;
37 }

```

vsprintf函数的编写只需要实现%d的格式化字符串的处理，也可以参考网上C语言printf的具体实现，从相应的库函数中移植。

```

1  #include <stdarg.h>
2
3  int vsprintf(char *buf, const char *fmt, va_list argptr){
4      //Suppose that the fmt string only has the "%d"
5  }

```

7 程序运行

我们提供了source2run.sh文件，所以只需在命令行键入”./source2run.sh”即可运行程序。
运行结果如下：

```

chenksy@DESKTOP-KBB198K: /mnt/c/lab2/src
chenksy@DESKTOP-KBB198K: $ cd /mnt/c
chenksy@DESKTOP-KBB198K: /mnt/c $ ks
ks: command not found
chenksy@DESKTOP-KBB198K: /mnt/c $ ls
ls: cannot access 'DumpStack.log.tmp': Permission denied
ls: cannot access 'hiberfil.sys': Permission denied
ls: cannot access 'pagefile.sys': Permission denied
ls: cannot access 'swapfile.sys': Permission denied
System Volume Information
hiberfil.sys
WinREAgent
pagefile.sys
Program Files
Users
Program Files (x86)
Windows
Documents and Settings
appverifui.dll
DumpStack.log.tmp
audio.log
Recovery
chenksy@DESKTOP-KBB198K: /mnt/c $ cd lab2
chenksy@DESKTOP-KBB198K: /mnt/c/lab2 $ ls
chenksy@DESKTOP-KBB198K: /mnt/c/lab2 $ cd src
chenksy@DESKTOP-KBB198K: /mnt/c/lab2/src $ ls
Makefile README.txt source2run.sh
chenksy@DESKTOP-KBB198K: /mnt/c/lab2/src $ ./source2run.sh
./source2run.sh: 2: shell: Permission denied
rm -rf output
ld -r -I myOS/myOS.ld output/multibootheader/multibootHeader.o output/myOS/start3/10
ev/uart.o output/myOS/dev/vga.o output/myOS/i386/io.o output/myOS/printk/myPrintk.o
/userApp/main.o -o output/myOS.elf
make succeed
250 运行结果如下:
251
252 % 正文结束
253 \end{document}
254

```

图 1: 实验结果