

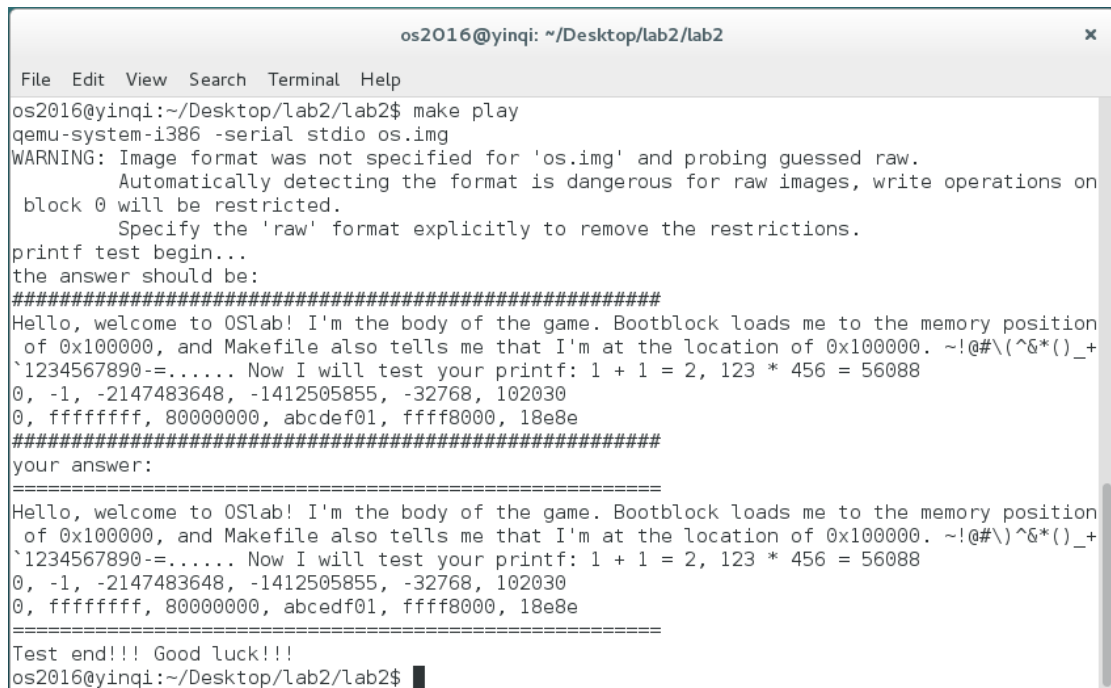
操作系统 Lab2 实验报告

141220132@smail.nju.edu.cn 141220132 银琦

实验进度

实现了所有要求的 printf，未输出到 qemu，未实现清屏和滚屏。

实验结果截图：



```
os2016@yinqi: ~/Desktop/lab2/lab2
File Edit View Search Terminal Help
os2016@yinqi:~/Desktop/lab2/lab2$ make play
qemu-system-i386 -serial stdio os.img
WARNING: Image format was not specified for 'os.img' and probing guessed raw.
Automatically detecting the format is dangerous for raw images, write operations on
block 0 will be restricted.
Specify the 'raw' format explicitly to remove the restrictions.
printf test begin...
the answer should be:
#####
Hello, welcome to OSlab! I'm the body of the game. Bootblock loads me to the memory position
of 0x100000, and Makefile also tells me that I'm at the location of 0x100000. ~!@#%^&*()_+
`1234567890-=..... Now I will test your printf: 1 + 1 = 2, 123 * 456 = 56088
0, -1, -2147483648, -1412505855, -32768, 102030
0, ffffffff, 80000000, abcdef01, ffff8000, 18e8e
#####
your answer:
=====
Hello, welcome to OSlab! I'm the body of the game. Bootblock loads me to the memory position
of 0x100000, and Makefile also tells me that I'm at the location of 0x100000. ~!@#%^&*()_+
`1234567890-=..... Now I will test your printf: 1 + 1 = 2, 123 * 456 = 56088
0, -1, -2147483648, -1412505855, -32768, 102030
0, ffffffff, 80000000, abcdef01, ffff8000, 18e8e
=====
Test end!!! Good luck!!!
os2016@yinqi:~/Desktop/lab2/lab2$
```

实验过程：

1. 初始化 GDT 表项和 TSS 段
在 kvm.c 中，初始化 TSS 段，根据要求，只需要设置 tss.esp0 和 tss.ss0，然后使用 ltr 指令加载 TSS 段。设置完之后还要设置段寄存器。
2. 初始化中断描述符表 IDT
仿照 test1 和 test2，设置一个中断函数 sys，本实验中需要 int 0x80，所以设置该中断，在 do_irq.S 中，仿照测试函数，将 int 0x80 中断压入栈，当遇到 80 中断后便会进行处理。
3. 加载用户程序
在 kvm.c 中，填写 load_umain 函数，这个函数与 lab1 中的加载函数相似，在写 lab1 的时候参照了原来的 pa 讲义，当时的理解还不够透彻，但是在加载用户程序的时候，出现了不少 bug，在调试过程中进一步加深了对加载程序的理解，将用户程序加载到了磁盘的第 202 个扇区。
4. 跳转到用户空间
在 kvm.c 中，填写 enter_user_space 函数，通过嵌入汇编，将 SS,ESP,EFLAGS,CS,EIP 压入栈，其中 EIP 的值就是 elf->entry 的值。还要设置段寄存器的值，进入 do_irq.S，在汇

编代码里将 `es` 和 `ds` 入栈，设置正确后，再将它们出栈。

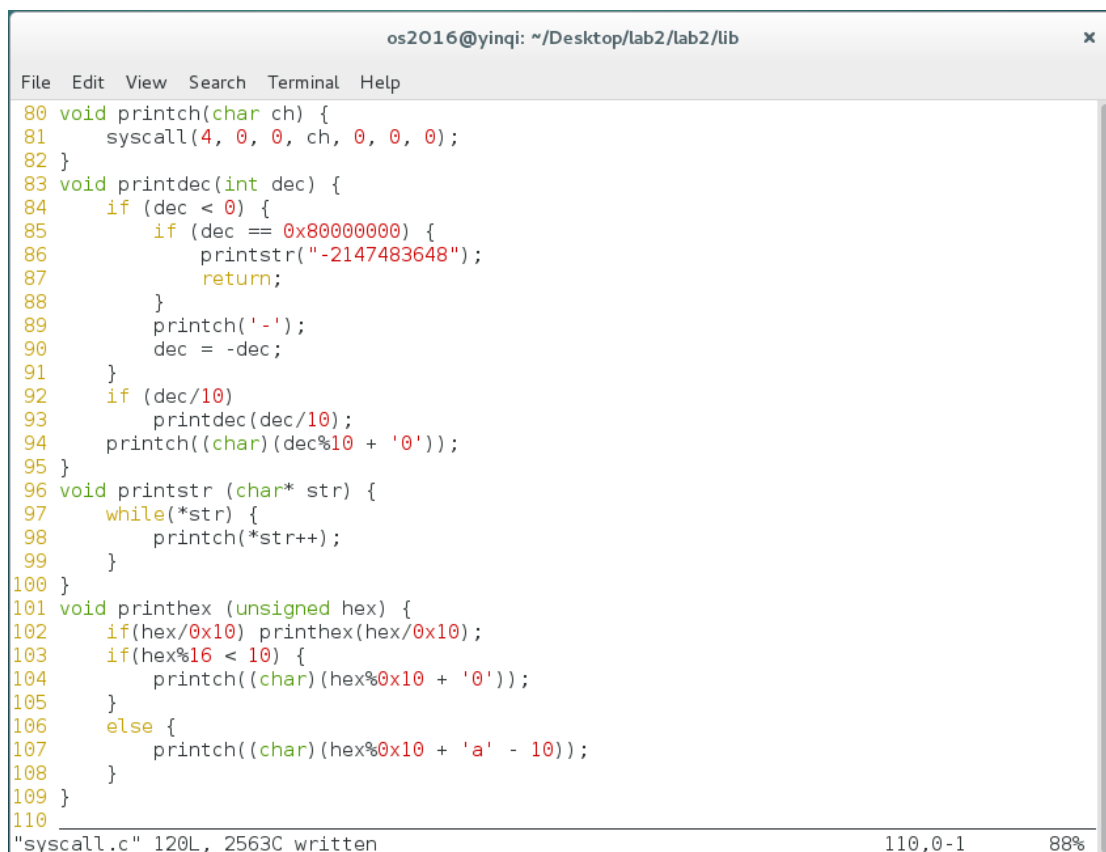
于是，这里出现了一个 bug 让我折腾了很久，因为将 `es`, `ds` 入栈，改变了栈帧的结构，所以要在 `kernel/include/memory.h` 里的 `Trapframe` 结构最前面加上 `uint32_t ds, es`，这样 `eip` 才会跳到正确的值，`irq` 的值也才会正确，否则在处理中断的时候一直都进入 `empty` 函数。

5. 中断处理

在 `irq_handle.c` 中，当 `irq` 的值是 `0x80` 时，调用 `do_syscall` 函数，根据 `syscall` 的参数设置，输出相应寄存器的值，我将字符设置在了 `ecx` 寄存器中，所以调用 `putchar` 输出 `ecx` 中的值。

6. 库函数

实现 `printf` 函数，在 `syscall.c` 中，首先在 `syscall` 函数中陷入 `80` 中断，按照讲义的代码写，然后开始实现 `printf`，`printf` 的函数参数个数不确定，在网上查找了实现 `printf` 的方法，大多都使用了 `stdarg.h` 中的函数，于是仿照这个库函数进行了实现，定义了参数 `pArg`，是用来记录参数的，每处理完一个参数，只要使用这个变量加上 `sizeof(format)`，就可以访问下一个参数。然后对参数进行处理，如果是格式化输出，则判断其输出类型，对每个类型编写一个处理函数，对其进行输出，思路是：输出字符的时候使用系统调用，十进制、十六进制、字符串输出都转化为字符输出，这样可以减小工作量；如果不是格式化输出，则一个字符一个字符输出即可。四个类型的输出函数处理如下：



```
os2016@yinqi: ~/Desktop/lab2/lab2/lib
File Edit View Search Terminal Help
80 void printch(char ch) {
81     syscall(4, 0, 0, ch, 0, 0, 0);
82 }
83 void printdec(int dec) {
84     if (dec < 0) {
85         if (dec == 0x80000000) {
86             printstr("-2147483648");
87             return;
88         }
89         printch('-');
90         dec = -dec;
91     }
92     if (dec/10)
93         printdec(dec/10);
94     printch((char)(dec%10 + '0'));
95 }
96 void printstr(char* str) {
97     while(*str) {
98         printch(*str++);
99     }
100 }
101 void printhex(unsigned hex) {
102     if(hex/0x10) printhex(hex/0x10);
103     if(hex%16 < 10) {
104         printch((char)(hex%0x10 + '0'));
105     }
106     else {
107         printch((char)(hex%0x10 + 'a' - 10));
108     }
109 }
110 }
"syscall.c" 120L, 2563C written 110,0-1 88%
```

在实现十六进制输出的时候出现了一些小问题，一开始是仿照十进制写，但是在输出的时候要判断是数字还是字母，因为我是用递归实现的，一开始写的判断条件是 `if (hex < 10)`，结果再测试 `0x80000000` 的时候输出结果是 `8WWWWWWWWW`，分析一下发现，递归函数在返回上一层后，`hex` 的值一定大于 `10`，但是 `hex%16` 的值却不一定，所以对判断条件进行了修改，最终结果正确。

Printf 如下:

```
void printf(const char *format,...){
    char* pArg = NULL;
    pArg = (char*)&format;
    pArg += sizeof(format);
    while (*format) {
        if (*format == '%') {
            switch(*(++format)) {
                case 'c':printf(*((char*)pArg));break;
                case 'd':printf(*((int*)pArg));break;
                case 'x':printf(*((unsigned*)pArg));break;
                case 's':printf(*((char**)pArg));break;
            }
            pArg += sizeof(int);
        }
        else {
            printf(*format);
        }
        ++format;    //here!!!
    }
}
```

收获心得:

1. 了解了 printf 的工作原理。
2. 对内嵌汇编有了一定的了解。
3. 加深了对操作系统工作过程的理解。