

# 2018 OS lab2 实验报告

姓名:王志邦

学号:161220131

日期:18.5.6

## 一、实验目的

本实验通过实现一个简单的应用程序，并在其中调用一个自定义实现的系统调用，介绍基于中断实现系统调用的全过程

## 二、实验流程

- a) Bootloader 从实模式进入保护模式,加载内核至内存，并跳转执行
- b) 内核初始化 IDT ( Interrupt Descriptor Table，中断描述符表)，初始化 GDT，初始化 TSS ( Task State Segment，任务状态段)
- c) 内核加载用户程序至内存，对内核堆栈进行设置，通过 `iret` 切换至用户空间，执行用户程序
- d) 用户程序调用自定义实现的库函数 `printf` 打印字符串
- e) `printf` 基于中断陷入内核，由内核完成在视频映射的显存地址中写入内容，完成字符串的打印

## 三、实验实现

- a) 从实模式进入保护模式通过 lab1 同样的方法可以实现。在加载内核到内存中时，需要读取 elf 头

```

void bootMain(void) {
    /* 加载内核至内存, 并跳转执行 */
    ELFHeader *elf;
    ProgramHeader *ph, *eph;

    uint8_t *buf = (uint8_t*)0x1000000;
    for(int i = 0; i < 200; i++)
        readSect((void*)(buf+512*i), i+1);

    elf = (ELFHeader*)buf;
    ph = (ProgramHeader*)((uint32_t)elf+elf->phoff);
    eph = ph + elf->phnum;
    for(; ph < eph; ph++){
        if(ph->type == 1){
            uint32_t a = ph->vaddr;
            uint32_t b = (uint32_t)elf + ph->off;
            while(a < ph->vaddr + ph->filesz){
                *(uint8_t*)a = *(uint8_t*)b;
                a++;
                b++;
            }
            while(a < ph->vaddr + ph->memsz){
                *(uint8_t*)a = 0;
                a++;
                b++;
            }
        }
    }
    void (*elf_entry)(void);
    elf_entry = (void*)(elf->entry);
    elf_entry();
}

```

循环调用 readSect 读取 1~200 扇区

b) 初始化 tss 与段寄存器：

```
asm volatile("movl %%esp, %0"::"m"(tss.esp0));
asm volatile("movl %%ss, %0"::"r"(tss.ss0));
asm volatile("ltr %%ax":: "a" (KSEL(SEG_TSS)));

/*设置正确的段寄存器*/
asm volatile("movl %0, %%eax"::"r"(KSEL(SEG_KDATA)));
asm volatile("movw %ax, %ds");
asm volatile("movw %ax, %es");
asm volatile("movw %ax, %ss");
asm volatile("movw %ax, %fs");
asm volatile("movl %0, %%eax"::"r"(KSEL(SEG_VIDEO)));
asm volatile("movw %ax, %gs");

lLdt(0);

void enterUserSpace(uint32_t entry) {
    /*
     * Before enter user space
     * you should set the right segment registers here
     * and use 'iret' to jump to ring3
     */
    asm volatile("pushl %0":: "r"(USEL(SEG_UDATA))); // %ss
    asm volatile("pushl %0":: "r"(128 << 20)); // %esp 128MB
    asm volatile("pushfl"); // %eflags
    asm volatile("pushl %0":: "r"(USEL(SEG_UCODE))); // %cs
    asm volatile("pushl %0":: "r"(entry)); // %eip
    asm volatile("iret");
```

将 ss 与 esp 放入 tss 中进行进程切换

c) 用户进程与内核态的切换与加载内核代码类似

d) Printf 调用：

由用户进程调用 printf，对与 printf 的处理分为，格式化输出与非格式化输出两部分。

```
if(*format != '%')
    syscall(4, (uint32_t)format, 1);
```

1、

直接按照字符顺序进行打印

2、格式输出：s，c，d，x

a) s：将首地址和字符串长度传递给 sys\_write，进行打印

```
case 's':{
    int len;
    s = va_arg(ap, char*);
    for(len = 0; s[len] != '\0'; ++len);
    syscall(4, (uint32_t)s, len);
    break;
}
```

b) c：与非格式化输出相同，将字符单个传入 sys\_write，进行打印

```
case 'c':{
    c = va_arg(ap, int);
    syscall(4, (uint32_t)&c, 1);
    break;
}
```

c) d：十进制数打印，即把 10 进制数转换为字符串，利用：

```
while(d > 0){
    unsigned temp = d % 10;
    index[i++] = temp + '0';
    d = d / 10;
}
```

逐位存储到 index 中，之后从后向前，逐个进行打印

d) x：十六进制数打印与十进制类似，只是转换的方式：

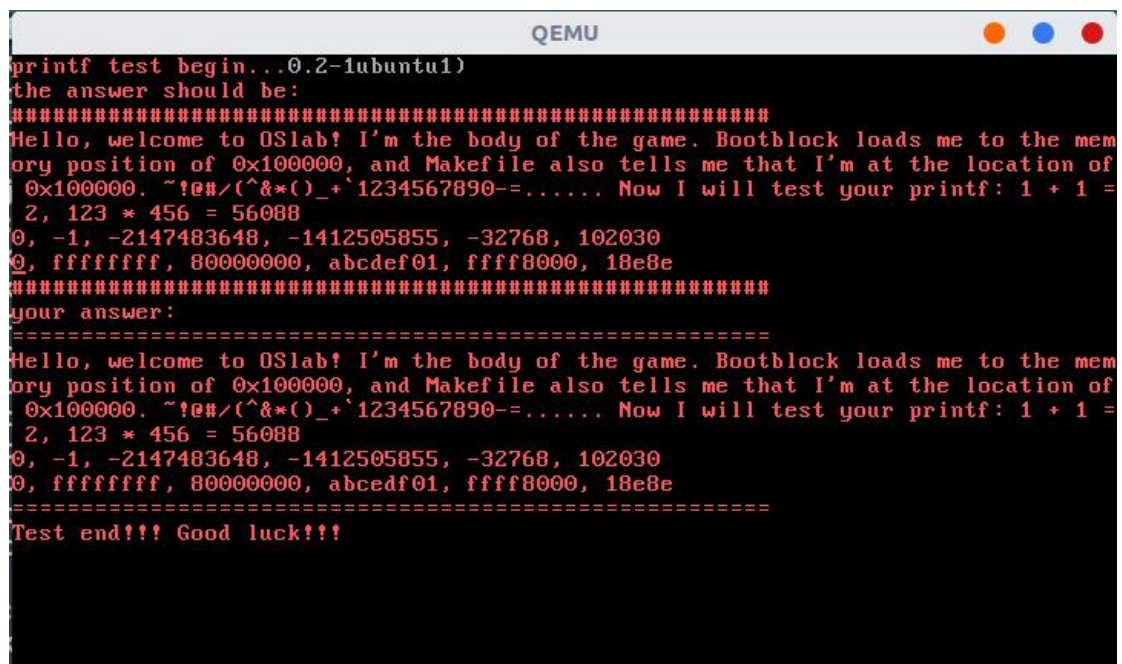
```

while(x > 0){
    unsigned temp = x % 16;
    if(temp < 10)
        index[i++] = temp + '0';
    else
        index[i++] = temp - 10 + 'a';
    x = x / 16;
}

```

在数字打印的时候，要考虑特殊数字，如 0，0x80000000，有符号数的负数

#### 四、实验结果



```

QEMU
printf test begin...0.2-1ubuntu1)
the answer should be:
#####
Hello, welcome to OSlab! I'm the body of the game. Bootblock loads me to the mem
ory 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 mem
ory 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!!!

```