

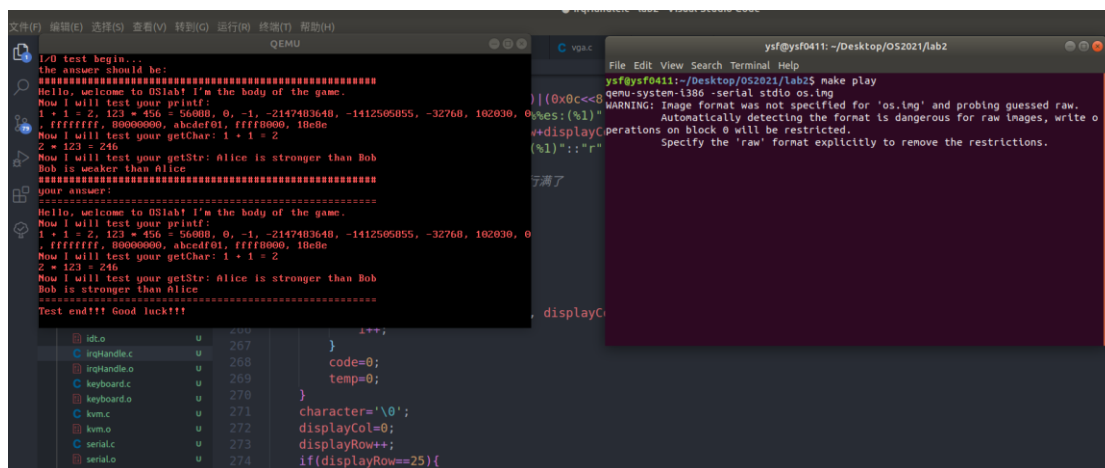
# Lab2 实验报告

191300073 AI 杨斯凡 191300073@smail.nju.edu.cn

## 一、试验进度

我完成了所有实验内容

## 二、实验结果



```
I/O test begin...
the answer should be:
Hello, welcome to OSlab! I'm the body of the game.
Now I will test your printf:
1 * 1 = 2, 123 * 456 = 56800, 0, -1, -2147483648, -1412585855, -32768, 102836, 0
ffffffffff, 80000000, abcdef01, fffff8000, 10be
Now I will test your getchar: 1 * 1 = 2
2 * 123 = 246
Now I will test your getstr: Alice is stronger than Bob
Bob is weaker than Alice
your answer:
Hello, welcome to OSlab! I'm the body of the game.
Now I will test your printf:
1 * 1 = 2, 123 * 456 = 56800, 0, -1, -2147483648, -1412585855, -32768, 102836, 0
ffffffffff, 80000000, abcdef01, fffff8000, 10be
Now I will test your getchar: 1 * 1 = 2
2 * 123 = 246
Now I will test your getstr: Alice is stronger than Bob
Bob is stronger than Alice
Test end!!! Good luck!!!

ls -l
total 16
-rw-r--r-- 1 ysf ysf 270  2021-05-20 14:14 idt.o
-rw-r--r-- 1 ysf ysf 267  2021-05-20 14:14 irqhandle.c
-rw-r--r-- 1 ysf ysf 268  2021-05-20 14:14 irqhandle.o
-rw-r--r-- 1 ysf ysf 269  2021-05-20 14:14 keyboard.c
-rw-r--r-- 1 ysf ysf 270  2021-05-20 14:14 keyboard.o
-rw-r--r-- 1 ysf ysf 271  2021-05-20 14:14 kvm.c
-rw-r--r-- 1 ysf ysf 272  2021-05-20 14:14 kvm.o
-rw-r--r-- 1 ysf ysf 273  2021-05-20 14:14 serial.c
-rw-r--r-- 1 ysf ysf 274  2021-05-20 14:14 serial.o
```

## 三、修改代码位置

首先我填写了 BootLoader 中的 TODO 部分, 因为 0x100000 是一个 ELF 头, 所以使用这个结构体的成员, entry 是内核入口, 然后读出程序头表的偏移量, 然后读出程序的偏移量, 接着加载扇区, 然后跳转到内核执行。在 start.S 中设置 esp 为 0x1fffff。

然后跳转到了 kernel 的 main.c 中执行, 首先会进行串口, IDT, VGA 等初始化, 这部分中, 我首先完成了 IDT 的初始化部分, 其中中断门和陷阱门照着讲义上的指导, 搞清楚每一位的作用之后很快就完成了, 然后在 InitIdt()中, 将保护错误, 段错误, 页错误等使用陷阱门初始化, 然后将系统调用和键盘使用中断门初始化即可。

然后初始化中断控制器, GDT 表、配置 TSS 段, VGA 和键盘。最后需要完成对用户程序的加载, 这部分按照提示模仿 bootmain 部分, 从 201 号扇区加载即可。

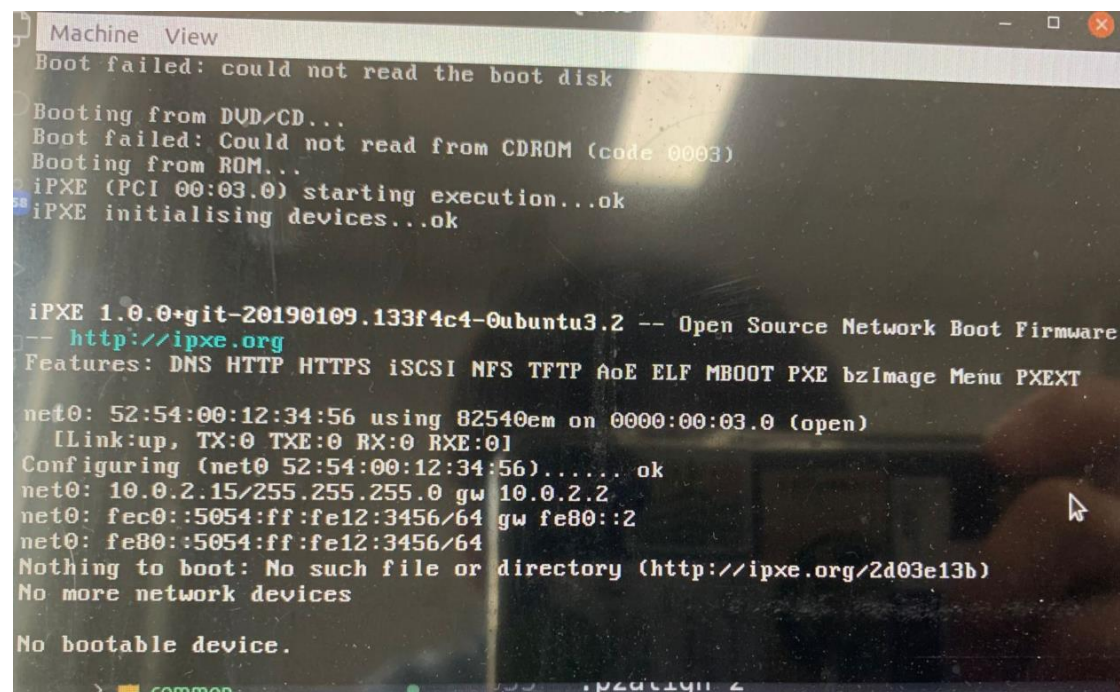
接着是中断处理程序的完善, 首先我照着讲义上的指导, 将 irqKeyboard 的中断向量号 0x21 压入栈中。然后我进行了 printf()的实现, 由于去年 PA 实验中实现过 printf()这个函数, 再加上 lab 中的函数功能很多, 因此这次实现也是比较容易, 但是今年无法使用 stdarg.h 这个库函数对参数进行直接调用, 但是考虑到参数在栈中是连续存储的, 就可以利用这一个特性对参数进行使用, 具体思路是每次遇到%后参数的地址增加来调用下一个参数, 就可以实现 printf(); 然后就是完善字符的打印, 使用了 syscall(SYS\_WRITE, STD\_OUT, (uint32\_t)buffer, (uint32\_t)size, 0, 0)这个函数, 这个函数会进行系统调用, 具体的函数在 irqHandle.c 中进行实现, 首先会根据上下文结构体的 irq 判断中断号在进行相应的处理, 以 printf 为例, 会调用 syscallHandle 函数中, 然后根据 eax 的值判断读或者写, 如果是写, 会调用 syscallprintf 函数, 将字符写到缓存中, 在这里需要注意光标的位置, 因为屏幕是 80\*25 的, 所以一行到 80 个字符后要换行, 并且到 25 行的时候要滚动一次。如果调用了读函数的话, 读取键盘的内容, 以 getstr 为例, 因为 getstr 涉及到字符串的传递, 因此我将字符串的地址当做参数传入, 并且直接在内核态对这个地址进行操作, 在读取输入的时候, 因为有一个最大的字符数, 每次读取到一个字符的时候,

会进行计数，防止超过产生安全问题，并且为了防止按键时间过长导致下次调用的时候这个键还在 press 状态，我在每次弹起之后才停止 getkeycode，这样就保证不会产生按压时间过长导致别的函数接受到了这个 keycode，然后使用 getchar 函数得到这个字符在写到字符串相应位置上。在这次 lab2 中，我的 getchar 会返回最后一个字符，而在 C 语言中则是会返回第一个。

然后是键盘中断，添加了相应的中断号之后只需要完成 KeyboardHandle 函数即可，对几种字符进行分类，回车符进行换行，退格符将前一个位置用黑色填充再将光标刷新到该位置即可，正常字符在当前位置打印即可，只需要判断一行是否满了和是否需要滚动屏幕即可。

#### 四、Bug 以及解决

在 Lab2 中，由于框架代码变多，于是遇到了挺多的 bug。首先是系统版本问题，在刚拿到 lab2 的框架代码后，我整体阅读了框架之后进行了代码的填写，然后在自认为“正确地”完成了所有内容后进行了 make，但是却出现了下面的画面



```
Machine View
Boot failed: could not read the boot disk
Booting from DVD/CD...
Boot failed: Could not read from CDROM (code 0003)
Booting from ROM...
iPXE (PCI 00:03.0) starting execution...ok
iPXE initialising devices...ok

iPXE 1.0.0+git-20190109.133f4c4-Ubuntu3.2 -- Open Source Network Boot Firmware
-- http://ipxe.org
Features: DNS HTTP HTTPS iSCSI NFS TFTP AoE ELF MBOOT PXE bzImage Menu PXEXT

net0: 52:54:00:12:34:56 using 82540em on 0000:00:03.0 (open)
[Link:up, TX:0 TXE:0 RX:0 RXE:0]
Configuring (net0 52:54:00:12:34:56)..... ok
net0: 10.0.2.15/255.255.255.0 gw 10.0.2.2
net0: fec0::5054:ff:fe12:3456/64 gw fe80::2
net0: fe80::5054:ff:fe12:3456/64
Nothing to boot: No such file or directory (http://ipxe.org/2d03e13b)
No more network devices

No bootable device.
```

再向助教询问后，得知版本和 gcc 可能对实验造成影响，我的版本是 Ubuntu20.04 gcc 9.3，我将 gcc 降级到 7.5 之后仍然没办法解决这个问题。然后我在网上查阅了相关资料后得知是系统的问题，于是我装了 Ubuntu18.04 虚拟机之后问题得到了解决，终于不会 failed 了。

然后第二个 bug 是关于 assert 的，由于上学期在 PA 的时候没有加 assert 导致了很多 bug 的出现并且不容易发现，于是我就会经常使用 assert 来避免 bug，这次看到了可以使用 assert(0)，我就使用了很多的 assert，但是这也导致了很多的 bug，因为去年的 PA 中 printf 匹配规则不完善导致了很多人 bug，于是我在 printf 中用了 assert(0)来避免别的字符匹配方式，但是这个 assert(0)导致编译无法通过，具体的原因到现在我还没有找到。

```
main.c kernel  kvm.c  idt.c  irqHandle.c  keyboard.c  vga.c  syscall.c  main.c app
lib > C syscall.c > ...
124 // buffer[count++]='x';
125 count=hex2Str(hexadecimal,buffer,256,count);
126 break;
127 }
128 case's':
129 string = *(char**)(paraList);
130 count=str2Str(string,buffer,256,count);
131 break;
132 case'c':
133 character = *(char*)(paraList);
134 buffer[count++]=character;
135 if(count==256){
136 syscall(SYS_WRITE, STD_OUT, buffer, count);
137 count=0;
138 }
139 break;
140 default: assert(0);
141 }
142 }
143 }
144 i++;
145 if(state==2) break;
146 }
147 if(count!=0)
148 syscall(SYS_WRITE, STD_OUT, (uint32_t)buffer, (uint32_t)count, 0, 0);
149 }
```

```
ysf@ysf0411: ~/Desktop/OS2021/lab2
File Edit View Search Terminal Help
gointer -Wall -Werror -O2 -I./include -c -o kernel/vga.o kernel/vga.c
gcc -m32 -march=i386 -static -fno-builtin -fno-stack-protector -fno-omit-frame-p
ointer -Wall -Werror -O2 -I./include -c -o kernel/keyboard.o kernel/keyboard.c
gcc -m32 -c -o kernel/doIrq.o kernel/doIrq.S
ld -m elf_i386 -e kEntry -Ttext 0x00100000 -o kMain.elf ./lib/abort.o ./main.o .
./kernel/i8259.o ./kernel/kvm.o ./kernel/idt.o ./kernel/irqHandle.o ./kernel/disk
.o ./kernel/serial.o ./kernel/vga.o ./kernel/keyboard.o ./kernel/doIrq.o
OK: Kernel is 15404 bytes - Extended to 200 sectors
make[1]: Leaving directory '/home/ysf/Desktop/OS2021/lab2/kernel'
make[1]: Entering directory '/home/ysf/Desktop/OS2021/lab2/app'
gcc -m32 -march=i386 -static -fno-builtin -fno-stack-protector -fno-omit-frame-p
ointer -Wall -Werror -O2 -I./lib -c -o main.o main.c
gcc -m32 -march=i386 -static -fno-builtin -fno-stack-protector -fno-omit-frame-p
ointer -Wall -Werror -O2 -I./lib -c -o ../lib/syscall.o ../lib/syscall.c
ld -m elf_i386 -e uEntry -Ttext 0x00000000 -o uMain.elf ./main.o ../lib/syscall.
o
../lib/syscall.o: In function 'printf':
syscall.c:(.text+0x4c1): undefined reference to '__assert_fail'
makefile:15: recipe for target 'uMain.bln' failed
make[1]: *** [uMain.bln] Error 1
make[1]: Leaving directory '/home/ysf/Desktop/OS2021/lab2/app'
makefile:4: recipe for target 'os.img' failed
make: *** [os.img] Error 2
ysf@ysf0411:~/Desktop/OS2021/lab2$
```

这个地方改为 break 之后，运行之后却只有光标闪烁，按键没有反应，这个 bug 困扰了我很久，在检查完很多地方之后我开始怀疑框架代码是否正确，于是我把所有的 assert 都去掉之后就成功的打印了字符，这个 bug 产生的主要原因我认为是可能有其他中断导致了 assert 但是这个 assert 功能只是让 qemu 停下，但是没在终端打印信息也没让 qemu 终止运行，我认为下次可以将 assert 功能完善一下。

```
main.c kernel  kvm.c  idt.c  irqHandle.c  keyboard.c  vga.c  syscall.c  main.c app
kernel > kernel > C irqHandle.c > ...
24 void irqHandle(struct TrapFrame *tf) { // pointer tf = esp
25 /*
26  * 中断处理程序
27  */
28 /* Reassign segment register */
29 asm volatile("movw %%ax, %%ds:::a"(KSEL(SEG_KDATA)));
30 //asm volatile("movw %%ax, %%es:::a"(KSEL(SEG_KDATA)));
31 //asm volatile("movw %%ax, %%fs:::a"(KSEL(SEG_KDATA)));
32 //asm volatile("movw %%ax, %%gs:::a"(KSEL(SEG_KDATA)));
33 switch(tf->irq) {
34 // TODO: 填好中断处理程序的调用
35 case 0xd: GProtectFaultHandle(tf); break;
36 case 0x21: KeyboardHandle(tf); break;
37 case 0x80: syscallHandle(tf); break;
38 default: assert(0);
39 }
40 }
41
42 void GProtectFaultHandle(struct TrapFrame *tf){
43 assert(0);
44 return;
45 }
46
47 void KeyboardHandle(struct TrapFrame *tf){
48 uint32_t code = getKeyCode();
49 if(code == 0xe){ // 退格符
50 // TODO: 要求只能退格用户键盘输入的字符串，且最多退到当行行首
51 if(displayCol==0) return;
52 displayCol--;
```

```
QEMU
ld -m elf_i386 -e uEntry -Ttext 0x00000000 -o
make[1]: Leaving directory '/home/ysf/Desktop/
cat bootloader/bootloader.bin kernel/kMain.elf
qemu-system-i386 -serial stdio os.img
WARNING: Image format was not specified for 'c
Automatically detecting the format is
perations on block 0 will be restricted.
Specify the 'raw' format explicitly t
```

然后键盘中断的时候也有一个小 bug，每次打印的字符和按下的不一样，再向助教询问之后，得知可能是我只是将字符的索引当做字符使用，然后我调用了 getchar 函数获取字符之后就可以正常打印了。



首先会一直获取键盘码，然后知道获取到回车键，返回回车键的前一个字符，然后将这个字符赋予 eax 返回，这里我遇到了两个 bug，第一个就是按键时间过长导致 getstr 也接受到了这个字符，于是我将循环的终止条件设置为回车键弹起的时候，这样就避免了这个 bug，同时这也避免了重复字符的问题，但是有大小写和 shift 的问题（后面统一描述）。我完成的时候不知道需要回显，在询问助教之后得知要回显，加上回显即可完成功能。

然后是 getstr，我实现的功能是输入字符达到指定长度或者由于无法返回字符，我就把字符串的地址传入，在内核态对地址操作，这可能会引起安全性问题。

```
void syscallGetStr(struct TrapFrame *tf){
    // TODO: 自由实现
    int sel = USEL(SEG_UDATA);
    char *str = (char*)tf->edx;
    int size = tf->ebx;
    int i = 0;
    uint32_t code=0;
    uint32_t temp=0;
    uint16_t character;
    i=0;
    asm volatile("movw %0, %%es::m"(sel));
    while(i<size-1) {
        while (code!=temp+0x80&&temp<0x81) {
            if(code<0x81&&code!=0){
                temp=code;
            }
            code=getKeyCode();
        }
        if(temp==0x1c) break;
        if(temp<0x81&&temp!=0){
            character=(getChar(temp)|(0x0c<<8));
            asm volatile("movb %0, %%es:(%1)::r"(character),"r"(str+i));
            int pos = (80*displayRow+displayCol)*2;
            asm volatile("movw %0, (%1)::r"(character),"r"(pos+0xb8000));
            displayCol++;
            if(displayCol==80){//一行满了
                displayCol=0;
                displayRow++;
                if(displayRow==25){
                    displayRow--;
                    scrollScreen();
                }
            }
            updateCursor(displayRow, displayCol);
            i++;
        }
        code=0;
        temp=0;
    }
    character='\0';
    displayCol=0;
    displayRow++;
    if(displayRow==25){
        displayRow--;
        scrollScreen();
    }
    asm volatile("movb %0, %%es:(%1)::r"(character),"r"(str+i));
}
```



## 五、心得体会

本次的框架中，键盘部分的大小写切换和 shift 上有一个 bug，就是按压后会输出一个空白符号，并且 shift 有时候会失效，于是我对框架进行了更改，在不需要输出的按键上返回了一个比较大的值来表示不需要输出，并且解决了 shift 失灵的问题，然后删去了 cap 释放的时候的切换，这样也可以正确，另外，我实现了 Tab 空两格的功能，使得键盘功能更完善。

```
uint32_t getKeyCode() {
    uint32_t code = inByte(0x60);
    uint32_t val = inByte(0x61); //laba
    outByte(0x61, val | 0x80);
    outByte(0x61, val);
    if (code <= (KF12_P+0x80)){ // all 1 byte scan code, press or release
        switch(code) {
            case KLSH_P: // press left shift
            case (KLSH_P+0x80): // release left shift it makes shift useless
            case KRSH_P: // press right shift
            case (KRSH_P+0x80): // release right shift
                switchKeyboard();
                return 0xfff;
                break;
            case KCAP_P: // press capslock
                if (keyboardState == 0) {
                    keyboardState = 1;
                    switchKeyboard();
                }
                else if (keyboardState == 1) { //防止一直按着,其实可以传递参数表示大小写
                    keyboardState = 2;
                }
                return 0xFFF;
                break;
            case (KCAP_P+0x80): // release capslock
                if (keyboardState == 1) {
                    keyboardState = 1;
                }
                else if (keyboardState == 2) {
```

```
static inline void switchKeyboard() {
    if (keyTable == keyTableL)
        keyTable = keyTableU;
    else
        keyTable = keyTableL;
}

static inline void my_switchKeyboard(int i) { //让大小写简单了点, 仅仅需要使用i来表示当前是大写还是小写
    if (i==1) //caps on
        keyTable = keyTableU;
    else
        keyTable = keyTableL;
}
```

然后我对更改大小写部分用了我自己的函数, 使用一个全局变量来表示当前是大写还是小写, 这样框架就更清晰。