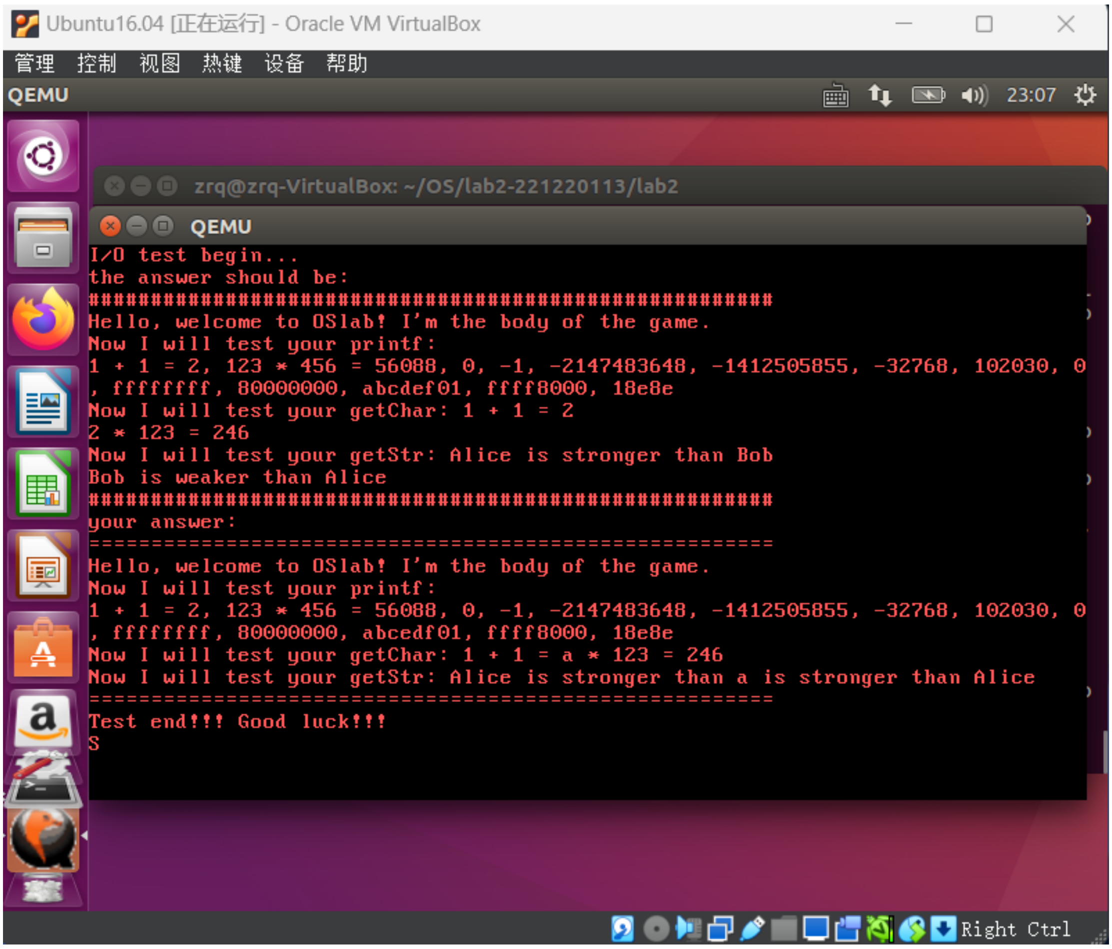
学号：221220113

姓名：仲瑞泉

邮箱：1448853658@qq.com

先给个最终结果



4.1

从为21号idt表项绑定处理函数开始，在do_irq中找到键盘处理函数的名字irqKeyBoard，在idt初始化中绑定到0x21号中断

```c
void initIdt() {
    int i;
    /* 为了防止系统异常终止，所有irq都有处理函数(irqEmpty)。 */
    for (i = 0; i < NR_IRQ; i ++) {
        setTrap(idt + i, SEG_KCODE, (uint32_t)irqEmpty, DPL_KERN);
    }
    /*init your idt here 初始化 IDT 表, 为中断设置中断处理函数*/

    // TODO: 参考上面第48行代码填好剩下的表项
    setTrap(idt + 0x8, SEG_KCODE, (uint32_t)irqDoubleFault, DPL_KERN);
    setTrap(idt + 0xa, SEG_KCODE, (uint32_t)irqInvalidTSS, DPL_KERN);
    setTrap(idt + 0xb, SEG_KCODE, (uint32_t)irqSegNotPresent, DPL_KERN);
    setTrap(idt + 0xc, SEG_KCODE, (uint32_t)irqStackSegFault, DPL_KERN);
    setTrap(idt + 0xd, SEG_KCODE, (uint32_t)irqGProtectFault, DPL_KERN);
    setTrap(idt + 0xe, SEG_KCODE, (uint32_t)irqPageFault, DPL_KERN);
    setTrap(idt + 0x11, SEG_KCODE, (uint32_t)irqAlignCheck, DPL_KERN);
    setTrap(idt + 0x1e, SEG_KCODE, (uint32_t)irqSecException, DPL_KERN);
    setIntr(idt + 0x21, SEG_KCODE, (uint32_t)irqKeyboard, DPL_KERN);
    setIntr(idt + 0x80, SEG_KCODE, (uint32_t)irqSyscall, DPL_USER);

    /* 写入IDT */
    saveIdt(idt, sizeof(idt));//use lidt
}
```

再从 do_irq 找到 irqHandle，为 0x21 号绑定键盘处理函数 KeyBoardHandle：

```c
void irqHandle(struct TrapFrame *tf) { // pointer tf = esp
    /*
     * 中断处理程序
     */
    /* Reassign segment register */
    asm volatile("movw %%ax, %%ds"::"a"(KSEL(SEG_KDATA)));

    switch(tf->irq) {
        // TODO: 填好中断处理程序的调用
        case -1:
            break;
        case 0xd:
            GProtectFaultHandle(tf);
            break;
        case 0x21:
            KeyboardHandle(tf);
            break;
        case 0x80:
            syscallHandle(tf);
            break;
        default:assert(0);
    }
}
```

最后到键盘处理函数中处理键盘信号并显示，具体操作可以参考 TODO 上面的代码。

```c
}else if(code < 0x81){
    // TODO: 处理正常的字符
    char character = getChar(code);
    if (character != 0) {
        //处理缓冲区
        keyBuffer[bufferTail++] = character;
        bufferTail %= MAX_KEYBUFFER_SIZE;
        //处理显示
        uint16_t data = character | (0x0c << 8);
        int pos = (80 * displayRow + displayCol) * 2;
        asm volatile("movw %0, (%1)"::"r"(data), "r"(pos + 0xb8000));
        //处理显示位置
        displayCol += 1;
        if (displayCol == 80) {
            displayCol = 0;
            displayRow++;
            if (displayRow == 25) {
                scrollScreen();
                displayRow = 24;
                displayCol = 0;
            }
        }
    }

}
```

4.2

和 4.1 反着来，先写完最底层的输出，具体操作可以参考 TODO 之前的代码

```c
for (i = 0; i < size; i++) {
    asm volatile("movb %%es:(%1), %0":"=r"(character) : "r"(str + i));
    // TODO:完成光标的维护和打印到显存
    if (character != '\n') {
        //处理显示
        data = character | (0x0c << 8);
        pos = (80 * displayRow + displayCol) * 2;
        asm volatile("movw %0, (%1)"::"r"(data), "r"(pos + 0xb8000));
        //处理换行
        displayCol += 1;
        if (displayCol == 80) {
            displayCol = 0;
            displayRow++;
            if (displayRow == 25) {
                scrollScreen();
                displayRow = 24;
                displayCol = 0;
            }
        }
    }
    else {
        displayCol = 0;
        displayRow++;
        if (displayRow == 25) {
            scrollScreen();
            displayRow = 24;
            displayCol = 0;
        }
    }
}
```

4.3

然后去调用链上层的 syscallPrint，TODO 上面定义的变量都可以使用，为了尽可能多的使用上面的变量而不进行删改，state 拿来处理读取状态，实际上不太用得上。

```
while (format[i] != 0) {
    switch (state)
    {
    case 0:
        switch (format[i])
        {
        case '%':
            state = 1;
            break;
        default:
            state = 0;
            buffer[count++] = format[i];
        }
        break;
    case 1:
        switch (format[i])
        {
        case 'c':
            state = 0;
            index += 4;
            character = *(char*)(paraList + index);
            buffer[count++] = character;
            break;
        case 'd':
            state = 0;
            index += 4;
            decimal = *(int*)(paraList + index);
            count = dec2Str(decimal, buffer, MAX_BUFFER_SIZE, count);
            break;
        case 'x':
            state = 0;
            index += 4;
            hexadecimal = *(uint32_t*)(paraList + index);
            count = hex2Str(hexadecimal, buffer, MAX_BUFFER_SIZE, count);
            break;
        case 's':
            state = 0;
            index += 4;
            string = *(char**)(paraList + index);
            count = str2Str(string, buffer, MAX_BUFFER_SIZE, count);
            break;
        default:
            state = 2;
        }
        break;
    }
    i++;
}
if (count != 0)
    syscall(SYS_WRITE, STD_OUT, (uint32_t)buffer, (uint32_t)count, 0, 0);
```

4.4

实际上，对于这两个函数，我遇到了一些问题没能解决。
在进行测试时，getChar 表现为输入后阻塞；getStr 表现为输入后不返回原处继续测试，可以一直输入。

所以为了测试用例看起来像通过了一样，按照下面编写：

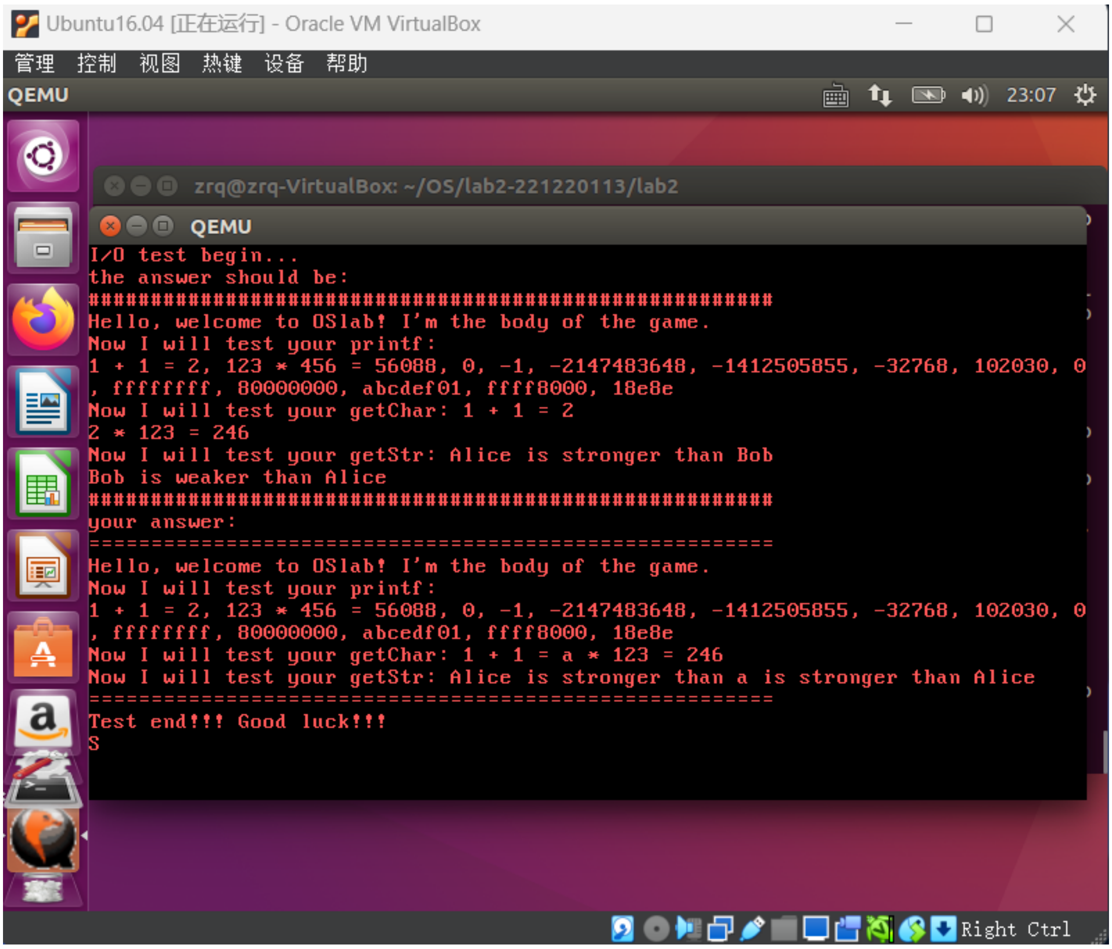
```c
char getChar() { // 对应SYS_READ STD_IN
    // TODO: 实现getChar函数，方式不限
    return '2';
}

void getStr(char* str, int size) { // 对应SYS_READ STD_STR
    // TODO: 实现getStr函数，方式不限
    str[0] = 'B';
    str[1] = 'o';
    str[2] = 'b';
    str[3] = 0;
}
```

4.5

最终进行测试，实际上为了成功测试，要先成功加载 app

```c
void loadUMain(void) {
    // TODO: 参照bootloader加载内核的方式，由kernel加载用户程序
    int i = 0;
    int phoff = 0x34; // program header offset
    int offset = 0x1000; // .text section offset
    uint32_t elf = 0x200000; // physical memory addr to load
    uint32_t uMainEntry = 0x200000;

    for (i = 0; i < 200; i++) {
        readSect((void*)(elf + i * 512), 201 + i);
    }

    struct ELFHeader* elfhdr = (struct ELFHeader*)elf;
    uMainEntry = elfhdr->entry; // entry address of the program
    phoff = elfhdr->phoff;
    struct ProgramHeader* prohdr = (struct ProgramHeader*)(elf + phoff);
    offset = prohdr->off;

    for (i = 0; i < 200 * 512; i++) {
        *(uint8_t*)(elf + i) = *(uint8_t*)(elf + i + offset);
    }

    enterUserSpace(uMainEntry);
}
```

最终测试如下


Ubuntu16.04 [正在运行] - Oracle VM VirtualBox
管理 控制 视图 热键 设备 帮助
QEMU
23:07
zrq@zrq-VirtualBox: ~/OS/lab2-221220113/lab2
QEMU
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 = 56088, 0, -1, -2147483648, -1412505855, -32768, 102030, 0
, ffffffff, 80000000, abcdef01, ffff8000, 18e8e
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 = 56088, 0, -1, -2147483648, -1412505855, -32768, 102030, 0
, ffffffff, 80000000, abcedf01, ffff8000, 18e8e
Now I will test your getChar: 1 + 1 = a * 123 = 246
Now I will test your getStr: Alice is stronger than a is stronger than Alice
==
Test end!!! Good luck!!!
S
Right Ctrl