OS Lab2 实验报告

姓名：孙文博

学号：201830210

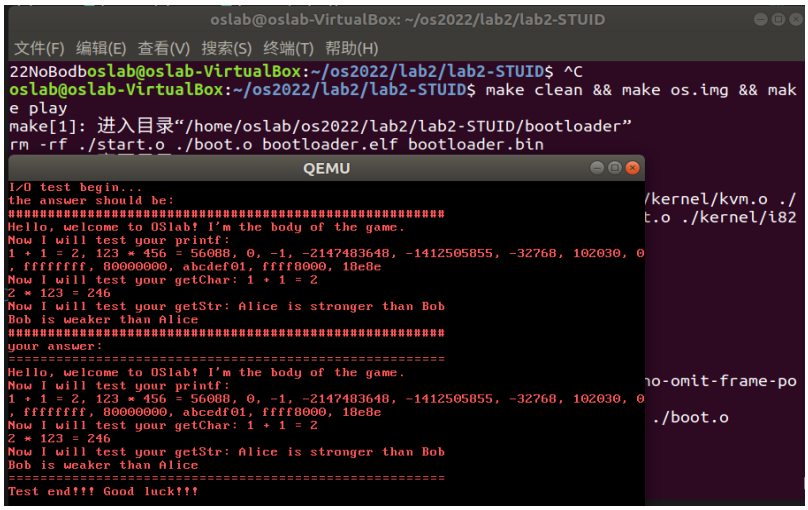
邮箱：201830210@smail.edu.nju.cn

1. 实验进度

完成了实验 `lab2` 的内容及思考题。

2. 实验结果

实现了三个库函数 `printf`，`getChar`，`getStr` 的系统调用，通过了测试用例。



3. 实验修改代码

3.1 装载内核

涉及文件 `bootloader/boot.c`，框架代码中只需要我们填写从 `kernel elf` 文件中解析的程序入口，偏移量等信息。

```
// TODO: 填写kMainEntry、phoff、offset
kMainEntry = (void(*)(void))((struct ELFHeader *)elf)->entry;
phoff =  ((struct ELFHeader *)elf)->phoff;
offset = ((struct ProgramHeader *)(elf + phoff))->off;
```

3.2 在内核中完善中断机制、提供系统服务函数

涉及文件 kernel/kernel/idt.c，需要我们实现中断门和陷阱门的设置，并仿照已有表项填好IDT中断表中的剩余表项。

```
/* 初始化一个中断门(interrupt gate) */
static void setIntr(struct GateDescriptor *ptr, uint32_t selector, uint32_t
offset, uint32_t dpl) {
    // TODO: 初始化interrupt gate
    ptr->pad0 = 0;
    ptr->offset_15_0 = offset & 0xFFFF;
    ptr->segment = selector << 3;
    ptr->type = INTERRUPT_GATE_32;
    ptr->system = FALSE;
    ptr->privilege_level = dpl;
    ptr->present = TRUE;
    ptr->offset_31_16 = (offset >> 16) & 0xFFFF;
}

/* 初始化一个陷阱门(trap gate) */
static void setTrap(struct GateDescriptor *ptr, uint32_t selector, uint32_t
offset, uint32_t dpl) {
    // TODO: 初始化trap gate
    ptr->pad0 = 0;
    ptr->offset_15_0 = offset & 0xFFFF;
    ptr->segment = selector << 3;
    ptr->type = TRAP_GATE_32;
    ptr->system = FALSE;
    ptr->privilege_level = dpl;
    ptr->present = TRUE;
    ptr->offset_31_16 = (offset >> 16) & 0xFFFF;
}
```

```
    // TODO: 填好剩下的表项
    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);

    /* Exceptions with DPL = 3 */
    // TODO: 填好剩下的表项
    setTrap(idt + 0x21, SEG_KCODE, (uint32_t)irqKeyboard, DPL_KERN);
    setIntr(idt + 0x80, SEG_KCODE, (uint32_t)irqSyscall, DPL_USER);
```

3.3 在内核中加载用户程序

涉及到文件 kernel/kernel/idt.c，需要实现 LoaduMain() 函数从内核中加载用户程序。

```
// TODO: 参照bootloader加载内核的方式
    int i = 0;
    int phoff = 0x34;
    int offset = 0x1000;
    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);
    }

    //填写uMainEntry、phoff、offset
    uMainEntry = ((struct ELFHeader *)elf)->entry;
    phoff =  ((struct ELFHeader *)elf)->phoff;
    offset = ((struct ProgramHeader *)(elf + phoff))->off;

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

3.4 实现中断处理程序

涉及到的文件 `kernel/kernel/irqHandle.c` , 需要补充完整 `KeyboardHandle`, `syscallPrint`, `syscallGetChar`, `syscallGetStr` 四个函数(仅展示核心部分），方便用户调用。

```
//KeyboardHandle
else if(code < 0x81){ // 正常字符
    // TODO: 注意输入的大小写的实现、不可打印字符的处理
    char character=getChar(code);
    if(character!=0){
        putChar(character);//put char into serial
        uint16_t data=character|(0x0c<<8);
        keyBuffer[bufferTail++]=character;
        bufferTail%=MAX_KEYBUFFER_SIZE;
        int pos=(80*displayRow+displayCol)*2;
        asm volatile("movw %0, (%1)"::"r"(data),"r"(pos+0xb8000));
        displayCol+=1;
        if(displayCol==80){
            displayCol=0;
            displayRow++;
            if(displayRow==25){
                scrollScreen();
                displayRow=24;
                displayCol=0;
            }
        }
    }
}
```

```
//syscallPrint
// TODO: 完成光标的维护和打印到显存
    if (character == '\n'){
        displayRow++;
        displayCol = 0;
        if (displayRow == 25){
            displayRow = 24;
            displayCol = 0;
            scrollScreen();
        }
    }
    else{
        data = character | (0x0c << 8);
        pos = (80 * displayRow + displayCol) * 2;
        asm volatile("movw %0, (%1)" ::"r"(data), "r"(pos + 0xb8000));
        displayCol++;
        if (displayCol == 80){
            displayRow++;
            displayCol = 0;
            if (displayRow == 25){
                displayRow = 24;
                displayCol = 0;
                scrollScreen();
            }
        }
    }
```

```
//syscallGetChar
    char wait=0;
    while(wait==0)
    {
        enableInterrupt();
        wait = keyBuffer[bufferHead+1];
        disableInterrupt();
    }
```

```
//syscallGetStr
    char tpc=0;
    while(tpc!='\n' && i<size)
    {
        while(keyBuffer[i]==0)
        {
            enableInterrupt();
        }
        tpc=keyBuffer[i];
        i++;
```

```
        disableInterrupt();
    }
```

3.5 实现用户层面函数调用

涉及文件 `lib/syscall.c` ,完成测试用例对 `getChar()` 以及 `getStr()` 函数的调用。至此，实验基本完成，执行测试用例。

```
char getChar(){ // 对应SYS_READ STD_IN
    return (char) syscall(SYS_READ,  STD_IN, 0, 0, 0, 0);
}

void getStr(char *str, int size){ // 对应SYS_READ STD_STR
    // TODO: 实现getStr函数
    syscall(SYS_READ, STD_STR, (uint32_t)str,  size, 0, 0);
}
```

---

4. 实验思考题

Exercise1

答：中断机制实现了计算机从内核态到用户态的切换，当外部设备的 `I/O` 模块准备好时，它会发送给 `CPU` 一个中断信号，`CPU` 则会立即做出响应，暂停当前程序的处理去服务该 `I/O` 设备的程序。

Exercise2

答：由于只是在由外层转移到内层（低特权级到高特权级）切换时新堆栈才会从 `TSS` 中取得，所以 `TSS` 中没有位于最外层的 `ring3` 的堆栈信息。

Exercise3

答：会发生不可恢复的错误，例如 `eax` 可能会保存函数返回值，而响应中断会使用 `eax` ，从而会丢失 `eax` 的信息。

Exercise4

答：

- `%d`表示按整型数据的实际长度输出数据。
- `%c`用来输出一个字符。
- `%s`用来输出一个字符串。
- `%x`表示以十六进制数形式输出整数

---

5. 实验心得

本次实验历时三周，整个实验的任务量和难度相比实验一也有很大的提升（😨），不过好在时间充足，有充足的时间可以搜索相关文献以及 `debug()` 😦

最大的困难是一开始填写内核态加载用户程序的时候，`qemu` 一直卡在启动界面不动了，后来发现是 `readSect()` 调用的问题。之后再去实现 `printf()` 等三个函数的中断调用函数时还算比较顺利吧！

最后感谢老师和助教提供的悉心指导的实验手册，感谢群友们的答疑解惑，希望之后的实验也可以一帆风顺！✌