lab2实验报告

陈德丹 221220159 邮箱：221220159@smail.nju.edu.cn

实验进度

完成所有实验内容

实验过程

在bootMain中，首先将磁盘的200个扇区搬运到0x100000处，也即为elf头开始的地址，从elf->entry读取kMainEntry，从elf->phoff读取程序头表的偏移量，再从程序头表中读取代码段的偏移量offset，将从代码段开始的内容移动到0x100000处，通过kMainEntry跳转到此处开始执行

```
// TODO: 阅读boot.h查看elf相关信息，填写kMainEntry、phoff、offset
kMainEntry = (void(*)(void))((struct ELFHeader*)elf)->entry;
phoff = ((struct ELFHeader*)elf)->phoff;
offset = ((struct ProgramHeader *)(elf + phoff))->off;
```

在loadUMain中也是类似的操作，只不过从第201个扇区开始读取，用户程序加载到0x200000处

在idt.c中初始化中断门和陷阱门

在initIdt中完成门描述符设置，之后的每次中断便可到相应的中断处理程序执行，注意硬件中断不受DPL影响，8259A的15个中断都为内核级可以禁止用户程序用 int 指令模拟硬件中断

完成这一步后，每次按键，内核会调用 irqKeyboard 进行处理

```
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*/
    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);

    // TODO: 参考上面第48行代码填好剩下的表项
    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
}
```

将irqKeyboard的中断向量号0x21压入栈

```
.global irqKeyboard
irqKeyboard:
    pushl $0
    # TODO: 将irqKeyboard的中断向量号压入栈
    pushl $0x21

    jmp asmDoIrq
```

在irqHandle中，填好中断处理程序的调用

```
switch(tf->irq) {
     case -1:
          break;
     case 0xd:
          GProtectFaultHandle(tf);
          break;
     case 0x21:
          KeyboardHandle(tf);
          break;
     case 0x80:
          syscallHandle(tf);
          break;
     default:assert(0);
}
```

getChar:等按键输入完成的时候，将末尾字符通过eax寄存器传递回来 getStr:等待直到缓冲区接收了回车字符，将段选择子确定为用户数据段，并将缓冲区的字符拷贝到指定的地址。

```
void syscallGetChar(struct TrapFrame *tf){
    // TODO: 自由实现
    enableInterrupt();
    while (bufferHead == bufferTail || keyBuffer[(bufferTail - 1) % MAX_KEYBUFFER_SIZE] != '\n')
        waitForInterrupt();
    disableInterrupt();
    tf->eax = keyBuffer[bufferHead];
    bufferHead = bufferTail;
}

void syscallGetStr(struct TrapFrame *tf){
    // TODO: 自由实现
    int sel = USEL(SEG_UDATA);   // User data/stack的段选择器
    char *str = (char*)tf->edx;
    int size = tf->ebx;
    int i = 0;
    char c = 0;
    asm volatile("movw %0, %%es"::"m"(sel));

    enableInterrupt();
    while(bufferHead == bufferTail || keyBuffer[(bufferTail - 1) % MAX_KEYBUFFER_SIZE] != '\n')
        waitForInterrupt();
    disableInterrupt();

    for (i = 0; i < size; i++) {
        if (bufferHead == bufferTail) break;
        if (keyBuffer[bufferHead] == '\n') break;
        c = keyBuffer[bufferHead];
        if (c != 0)
            asm volatile("movb %0, %%es:(%1)"::"r"(c),"r"(str + i));
```

```
        bufferHead = (bufferHead + 1) % MAX_KEYBUFFER_SIZE;
    }
    asm volatile("movb $0x00, %%es:(%0)"::"r"(str+i));
    bufferHead = bufferTail;
}
```

处理键盘输入（syscallPrint也为类似操作）

```
void KeyboardHandle(struct TrapFrame *tf) {
    uint32_t code = getKeyCode();

    if (code == 0xe) { // 退格符
        //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
        if (displayCol > 0 && displayCol > tail) {
            displayCol--;
            bufferTail = (bufferTail - 1) % MAX_KEYBUFFER_SIZE;
            uint16_t data = 0 | (0x0c << 8); //创建一个16位的数据值，用于在屏幕上表示空白字符
            int pos = (80 * displayRow + displayCol) * 2;
            asm volatile("movw %0, (%1)"::"r"(data),"r"(pos + 0xb8000));
        }
    }
    else if (code == 0x1c) { // 回车符
        //处理回车情况
        keyBuffer[bufferTail] = '\n';
        bufferTail = (bufferTail + 1) % MAX_KEYBUFFER_SIZE;
        if (bufferTail == bufferHead)
            bufferHead = (bufferHead + 1) % MAX_KEYBUFFER_SIZE;
        displayRow++;
        displayCol = 0;
        tail = 0;
        if(displayRow == 25) {
            scrollScreen();
            displayRow = 24;
            displayCol = 0;
        }
    }
    else if (code < 0x81 && code != 0x3a) {
        // TODO: 处理正常的字符
        char c = getChar(code); //defined in keyboard.c
        keyBuffer[bufferTail] = c;
        bufferTail = (bufferTail + 1) % MAX_KEYBUFFER_SIZE;
        if (bufferTail == bufferHead)
            bufferHead = (bufferHead + 1) % MAX_KEYBUFFER_SIZE;

        uint16_t data = c | (0x0c << 8);
        int pos = (80 * displayRow + displayCol) * 2;
        asm volatile("movw %0, (%1)"::"r"(data),"r"(pos + 0xb8000));

        displayCol++;
        if (displayCol == 80) {  // 如果到达行末，转到下一行
            displayRow++;
            displayCol = 0;
            tail = 0;
            if(displayRow == 25) {  //如果屏幕满了，滚动屏幕
                scrollScreen();
                displayRow = 24;
                displayCol = 0;
            }
        }
    }
    updateCursor(displayRow, displayCol);
```

printf核心实现

```
switch (format[i]) {
case 'd':
    index += 4;
    decimal = *(int*)(paraList + index);
    count = dec2Str(decimal, buffer, MAX_BUFFER_SIZE, count);
    break;
case 'x':
    index += 4;
    hexadecimal = *(uint32_t*)(paraList + index);
    count = hex2Str(hexadecimal, buffer, MAX_BUFFER_SIZE, count);
    break;
case 's':
    index += 4;
    string = *(char**)(paraList + index);
    count = str2Str(string, buffer, MAX_BUFFER_SIZE, count);
    break;
case 'c':
    index += 4;
    character = *(char*)(paraList + index);
    buffer[count] = character;
    count++;
    break;
```

实验结果