

oslab2 实验报告

191220138 杨飞洋

实验进度

已完成所有内容，不过 `getStr()` 和 `getChar()` 函数实现的和实验要求略有不同，在特别注意中有详细提到，麻烦留意一下。

特别注意

我的 `getChar()` 和 `getStr()` 实现的效果和要求的略有不同，这里特意说明一下。

`getChar()`：我的 `getChar()` 只会读取输入的第一个键，并且之后输入的键不会在qemu上显示，回车结束。但是如果上来直接输回车，就会造成糟糕的后果。

`getStr()`：开启 `getStr()` 函数时，需要先在键盘按下一个除了回车的按键，以开启这个输入过程。并且如果输入的很快可能会出现光标移动多格的情况，需要一个键一个键的敲下。并且caps和shift按键的效果可能有点不稳定。

由于自身能力有限，只能实现到这个程度了，麻烦助教留意一下。

实验目的

实现中断机制，加载并初始化内核，加载用户程序

实现系统调用库函数 `printf` 和对应的处理例程

实现由键盘输入到屏幕的显示

实现系统调用库函数 `getChar`、`getStr` 和对应的处理例程

实验过程

加载并初始化内核

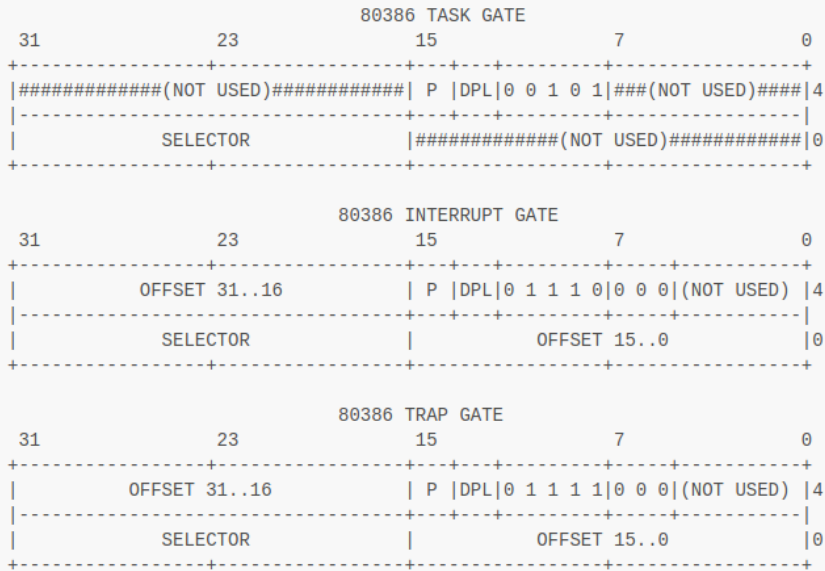
首先 `bootloader` 和 `lab1` 类似，从实模式进入保护模式，加载内核至内存，并跳转执行

内核首先进行一些初始化。

IDT

首先初始化中断门和陷阱门。

中断门和陷阱门的结构如下：



所以可用下面这段代码来实现中断门的初始化:

```
ptr->offset_15_0 = offset & 0xFFFF;
ptr->segment = selector << 3;
ptr->pad0 = 0;
ptr->type = INTERRUPT_GATE_32;
ptr->system = FALSE;
ptr->privilege_level = dpl;
ptr->present = TRUE;
ptr->offset_31_16 = (offset >> 16) & 0xFFFF;
```

陷阱门只需要修改一下 `type` 即可。

再初始化IDT表, 可以根据函数声明后面的注释进行相应的填写。

```
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);
```

再写系统调用, 这里写一下键盘和软中断。

```
setTrap(idt + 0x21, SEG_KCODE, (uint32_t)irqKeyboard, DPL_KERN);
setIntr(idt + 0x80, SEG_KCODE, (uint32_t)irqSyscall, DPL_USER);
```

8259a

框架代码已经给出了

```

void initIntr(void) {
    outByte(PORT_PIC_MASTER, 0x11); // ICW1, Initialization command
    outByte(PORT_PIC_SLAVE, 0x11); // ICW1, Initialization command
    outByte(PORT_PIC_MASTER + 1, 32); // ICW2, Interrupt Vector Offset 0x20
    outByte(PORT_PIC_SLAVE + 1, 32 + 8); // ICW2, Interrupt Vector Offset 0x28
    outByte(PORT_PIC_MASTER + 1, 1 << 2); // ICW3, Tell Master PIC that there is
a slave
    outByte(PORT_PIC_SLAVE + 1, 2); // ICW3, Tell Slave PIC its cascade identity
    outByte(PORT_PIC_MASTER + 1, 0x3); // ICW4, Auto EOI in 8086/88 mode
    outByte(PORT_PIC_SLAVE + 1, 0x3); // ICW4, Auto EOI in 8086/88 mode
}

```

TSS

框架代码已经给出了

```

void initSeg() { // setup kernel segments
    ...
    ...
}

```

VGA

框架代码已经给出了

```

void initVga() {
    displayRow = 0;
    displayCol = 0;
    displayClear = 0;
    clearScreen();
    updateCursor(0, 0);
}

```

键盘

框架代码已经给出了

```

void initKeyTable() {
    ...
    ...
}

```

随后调用这些init函数进行初始化

```

void kEntry(void) {
    // Interruption is disabled in bootloader
    initSerial(); // initialize serial port
    // TODO: 做一系列初始化
    initIdt(); // initialize idt
    initIntr(); // initialize 8259a
    initSeg(); // initialize gdt, tss
    initVga(); // initialize vga device
    initKeyTable(); // initialize keyboard device
    loadUMain(); // load user program, enter user space
    while(1);
    assert(0);
}

```

执行 `loadUMain()` 函数加载用户程序到内存，实现方式和加载内核几乎一样。

```

int i = 0;
int offset = 0x1000; // .text section offset
uint32_t elf = 0x200000; // physical memory addr to load
uint32_t uMainEntry;
for (i = 0; i < 200; i++) {
    readSect((void*)(elf + i*512), 201+i);
}
uMainEntry = ((struct ELFHeader *)elf)->entry;
for (i = 0; i < 200 * 512; i++) {
    *(unsigned char *) (elf + i) = *(unsigned char *) (elf + i + offset);
}
enterUserSpace(uMainEntry);

```

printf 的实现

首先实现 `syscallPrint()` 函数，进行字符的打印和光标的维护。

如果读到回车，到下一行，如果已经是最后一行了，进行滚屏。

如果不是回车，进行打印，可以用手册上给的代码，光标往后移一格，如果是最后一格了就到下一行，如果到了最后一行，则执行滚屏。

可以利用 `displayRow` 和 `displayCol` 变量和 `scrollScreen()` 函数，结合手册代码完成这个功能。

具体实现代码如下：

```

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();
    }
}
}

```

接下来实现 `syscall.c` 中的 `printf()` 函数

如果读到 '%', 根据读到的下一个字符, 进行相应的转换, 再执行对应的处理函数即可。注意一下 `count` 值的改变。

最后清空缓冲区。

代码如下:

```

if (format[i] == '%')
{
    count--;
    i++;
    paraList += sizeof(format);
    switch (format[i])
    {
        case 'c':
            character = *(char *)paraList;
            buffer[count++] = character;
            break;
        case 's':
            string = *(char **)paraList;
            count = str2Str(string, buffer, (uint32_t)MAX_BUFFER_SIZE, count);
            break;
        case 'x':
            hexadecimal = *(uint32_t *)paraList;
            count = hex2Str(hexadecimal, buffer, (uint32_t)MAX_BUFFER_SIZE,
count);
            break;
        case 'd':
            decimal = *(int *)paraList;
            count = dec2Str(decimal, buffer, (uint32_t)MAX_BUFFER_SIZE, count);
            break;
        case '%':
            paraList -= sizeof(format);
            count++;
            break;
    }
}
if (count == MAX_BUFFER_SIZE) //clean buffer
{

```

```

    syscall(SYS_WRITE, STD_OUT, (uint32_t)buffer, (uint32_t)MAX_BUFFER_SIZE, 0,
0);
    count = 0;
}
i++;

```

这样就完成了 `printf()` 函数的实现。

键盘输入处理

如果是退格符，光标往前移（如果光标不在第一个），前一个位置用空白填入。

如果是回车符，光标到下一行的开头，如果最后一行，执行滚屏。

如果是正常字符，注意对caps和shift的处理，其余正常打印，光标正常移动即可。

```

if(code == 0xe){ // 退格符
    // 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
    if(displayCol > 0 && bufferTail > 0){
        displayCol--;
        bufferTail--;
        uint16_t data = 0 | (0x0c << 8);
        int pos = (80 * displayRow + displayCol) * 2;
        asm volatile("movw %0, (%1)" :: "r"(data), "r"(pos + 0xb8000));
    }
    //scrollScreen();
}else if(code == 0x1c){ // 回车符
    // TODO: 处理回车情况
    //displayRow++;
    displayCol = 0;
    if (displayRow == 25)
    {
        displayRow = 24;
        displayCol = 0;
        scrollScreen();
    }
}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
    if(code == 0x2a || code == 0x2a + 0x80 || code == 0x36 || code == 0x36 +
0x80 || code == 0x3a || code == 0x3a + 0x80)
        return;
    char asc = getChar(code);
    uint16_t data = asc | (0x0c << 8);
    int 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();
        }
    }
}

```

```

    }
    keyBuffer[bufferTail] = asc;
    bufferTail++;
}
updateCursor(displayRow, displayCol);

```

getChar()实现

只用了一个字符来存储输入的第一个字符。

syscall.c 中的实现：

```

char getChar(){ // 对应SYS_READ STD_IN
    //TODO: 实现getChar函数，方式不限
    char c = 0;
    return syscall(SYS_READ, STD_IN, (uint32_t)c, 1, 0, 0);
}

```

现在看 syscallGetChar() 函数。

利用 getKeyCode() 函数来进行键盘的读取。

利用循环来让程序停下，执行用户输入。

```

uint32_t code = getKeyCode();
while(!code){
    code = getKeyCode();
}
char asc = getChar(code);
uint16_t data = asc | (0x0c << 8);
int pos = (80 * displayRow + displayCol) * 2;
asm volatile("movw %0, (%1)" :: "r"(data), "r"(pos + 0xb8000));

```

在对光标进行位置调整，这里不赘述了。

再就是判断回车来终止输入，仍然可以用一个循环来判断。

```

while(1){
    code = getKeyCode();
    char character = getChar(code);
    if (character == '\n')
    {
        //处理光标
        break;
    }
}

```

最后把读到的数据赋给 eax 作为 getStr() 函数的返回值

```
tf->eax = asc
```

getStr()实现

因为已经在用户程序中开辟好了空间，所以不用在 `getStr()` 函数中为指针再度分配空间，直接把指针给到 `syscall()` 函数即可。

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

接着实现 `syscallGetStr()` 函数。

首先初始化一个缓冲区 `buffer[]`，再因为之前的输入造成的影响导致 `getKeyCode()` 函数持续返回回车，所以这里需要进行中断。

```
uint32_t code = getKeyCode();
while(code == 0x1c || code == 0x1c + 0x80){
    code = getKeyCode();
}
```

这就是为什么在特别注意中提到的 " 需要先在键盘按下一个除了回车的按键，以开启这个输入过程 "。

加下来开启循环，因为 `getKeyCode()` 函数会持续性地返回上一次的键盘码，可能是按下的也可能是放开的。

我用一个 `code` 变量来存储上一次的返回值，再用一个 `temp` 变量存储当前得到的返回值，如果两者相等，则不管，如果两者相差 `0x80`，表示 `code` 是按下的键盘码，`temp` 是放开的键盘码，不管。如果读入不可显示的变量也不管。否则读入 `temp` 将其转化为 `char` 型存进缓冲区中，注意对 `backspace`、`enter` 的处理。

```
while(1){
    code = getKeyCode();
    char character = getChar(code);
    if (character == '\n')
    {
        //处理光标
        break;
    }
    temp = getKeyCode();
    while(temp == code){
        temp = getKeyCode();
    }
    if(temp == 0xe){ // 退格符
        if(displayCol > 0){
            displayCol--;
            uint16_t data = 0 | (0x0c << 8);
            int pos = (80 * displayRow + displayCol) * 2;
            asm volatile("movw %0, (%1)" :: "r"(data), "r"(pos + 0xb8000));
        }
        if(i > 0)i--;
    }
    else if(temp != 0x0 && temp != 0x2a && temp != 0x2a + 0x80 && temp != 0x36 && temp != 0x36 + 0x80
        && temp != 0x3a && temp != 0x3a + 0x80 && temp != 0x1c){
        char asc = getChar(temp);
```



```

uint16_t data = asc | (0x0c << 8);
int pos = (80 * displayRow + displayCol) * 2;
asm volatile("movw %0, (%1)" :: "r"(data), "r"(pos + 0xb8000));
if(temp - code != 128){
    displayCol += 1;
    buffer[i] = asc;
    i++;
}
//光标处理
}
}

```

最后把缓冲区中的数据赋给 `tf->edx` 中。

```
moveStr(buffer, i, (char*)tf->edx);
```

`moveStr()` 函数：

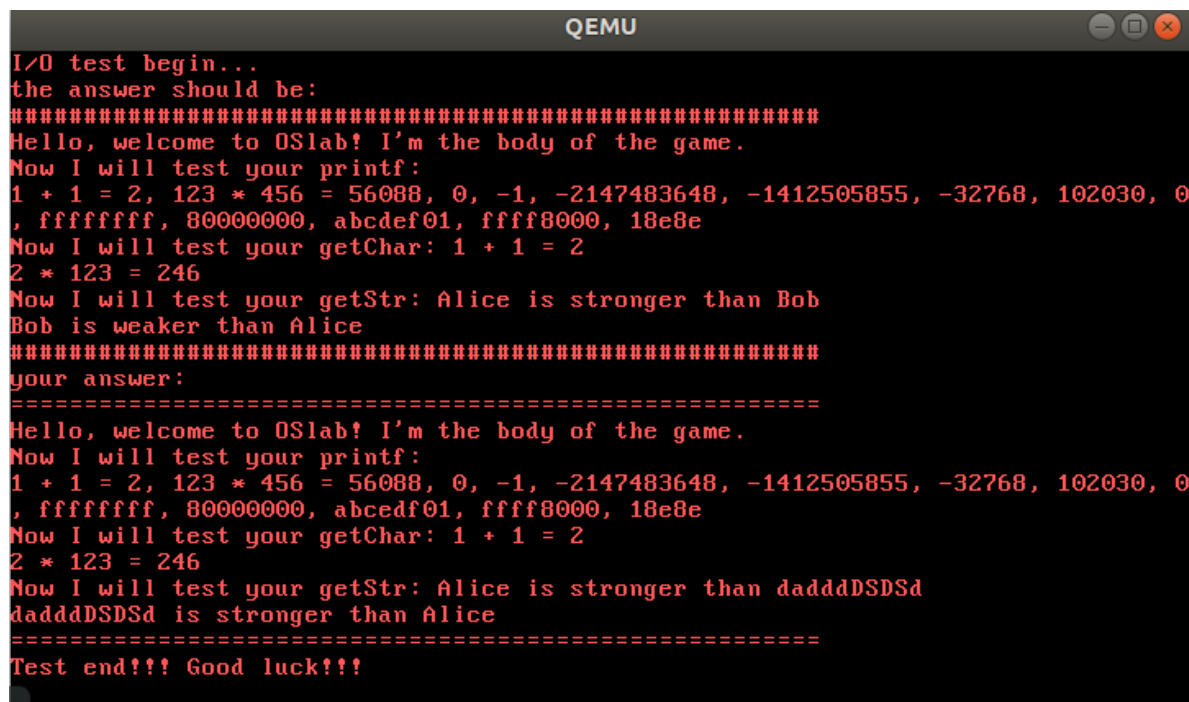
```

void moveStr(char *buf, int len, char *dst){
    int i = 0;
    for(; i < len; i++){
        dst[i] = buf[i];
    }
    dst[i] = 0;
}

```

这样一个稍简陋版的 `getStr()` 函数就实现啦。

实验效果



```

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, 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 dadddDSDSd
dadddDSDSd is stronger than Alice
=====
Test end!!! Good luck!!!

```

实验心得

通过实现对内核的一些初始化，了解了中断机制中的调用过程和一些数据结构。

通过对 `printf()`、`getStr()`、`getChar()` 函数的编写，加深了对系统调用过程的理解。