

Lab2 实验报告

231880101 孙俊晖

邮箱: 2033282932@qq.com

一、实验完成情况

1.实验进度

我完成了所有内容。

2.实验结果

我通过了用户程序主函数中的所有样例，实现了正常输入/输出字符，以及退格和换行功能。

```
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
, fffffff, 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
, fffffff, 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 stronger than Alice
=====
Test end!!! Good luck!!!
231880101 sjh
```

```
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
, fffffff, 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
, fffffff, 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 stronger than Alice
=====
Test end!!! Good luck!!!
what can i say
```

```

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
, ffffff, 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
, ffffff, 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 stronger than Alice
=====
Test end!!! Good luck!!!
what can i sa

```

3.实验修改的代码位置

(1) 键盘按键的串口回显

1. 设置门描述符&各部分的初始化

在lab2/kernel/kernel/idt.c中，我首先完成了初始化中断门和陷阱门的函数setIntr和setTrap。

```

1  /* 初始化一个中断门(interrupt gate) */
2  static void setIntr(struct GateDescriptor *ptr, uint32_t selector, uint32_t
offset, uint32_t dpl) {
3      // TODO: 初始化interrupt gate
4      ptr->offset_15_0 = offset & 0xffff;
5      ptr->segment = KSEL(selector);
6      ptr->pad0 = 0;
7      ptr->type = INTERRUPT_GATE_32;
8      ptr->system = 0;
9      ptr->privilege_level = dpl;
10     ptr->present = 1;
11     ptr->offset_31_16 = (offset >> 16) & 0xffff;
12 }
13 /* 初始化一个陷阱门(trap gate) */
14 static void setTrap(struct GateDescriptor *ptr, uint32_t selector, uint32_t
offset, uint32_t dpl) {
15     // TODO: 初始化trap gate
16     ptr->offset_15_0 = offset & 0xffff;
17     ptr->segment = KSEL(selector);
18     ptr->pad0 = 0;
19     ptr->type = TRAP_GATE_32;
20     ptr->system = 0;
21     ptr->privilege_level = dpl;
22     ptr->present = 1;
23     ptr->offset_31_16 = (offset >> 16) & 0xffff;
24 }
```

然后我完成了initIdt函数，初始化IDT表的同时为中断设置中断处理函数。

```

1 void initIdt() {
2     int i;
3     /* 为了防止系统异常终止，所有irq都有处理函数(irqEmpty)。 */
4     for (i = 0; i < NR_IRQ; i++) {
5         setTrap(idt + i, SEG_KCODE, (uint32_t)irqEmpty, DPL_KERN);
6     }
7     setTrap(idt + 0x8, SEG_KCODE, (uint32_t)irqDoubleFault, DPL_KERN);
8     // TODO: 填好剩下的表项
9     setTrap(idt + 0xa, SEG_KCODE, (uint32_t)irqInvalidTSS, DPL_KERN);
10    setTrap(idt + 0xb, SEG_KCODE, (uint32_t)irqSegNotPresent, DPL_KERN);
11    setTrap(idt + 0xc, SEG_KCODE, (uint32_t)irqStackSegFault, DPL_KERN);
12    setTrap(idt + 0xd, SEG_KCODE, (uint32_t)irqGProtectFault, DPL_KERN);
13    setTrap(idt + 0xe, SEG_KCODE, (uint32_t)irqPageFault, DPL_KERN);
14    setTrap(idt + 0x11, SEG_KCODE, (uint32_t)irqAlignCheck, DPL_KERN);
15    setTrap(idt + 0x1e, SEG_KCODE, (uint32_t)irqSecException, DPL_KERN);
16    /* Exceptions with DPL = 3 */
17    // TODO: 填好剩下的表项
18    setIntr(idt + 0x21, SEG_KCODE, (uint32_t)irqKeyboard, DPL_KERN);
19    setIntr(idt + 0x80, SEG_KCODE, (uint32_t)irqSyscall, DPL_USER);
20    /* 写入IDT */
21    saveIdt(idt, sizeof(idt));
22 }

```

然后我顺便完成了lab2/kernel/main.c中的内核main函数，这部分很简单，只要加入各个初始化函数即可。

```

1 void kEntry(void) {
2     initSerial(); // initialize serial port
3     // TODO: 做一系列初始化
4     initIdt(); // initialize idt
5     initIntr(); // initialize 8259a
6     initSeg(); // initialize gdt, tss
7     initVga(); // initialize vga device
8     initKeyTable(); // initialize keyboard device
9     loadUMain(); // load user program, enter user space
10
11    while(1);
12    assert(0);
13 }

```

在lab2/kernel/kernel/kvm.c中，我参照bootloader加载内核的方式完成了加载用户程序的函数，注意与boot.c中不同，加载用户程序时应从磁盘的第201个区域开始读取程序。

```

1 void loadUMain(void) {
2     int i = 0;
3     int offset = 0x1000;
4     unsigned int elf = 0x200000;
5     uint32_t uMainEntry = 0x200000;
6     for (i = 0; i < 200; i++) {
7         readSect((void*)(elf + i*512), 201 + i);
8     }
9     struct ELFHeader *ehdr = (void *)elf;
10    uMainEntry = (uint32_t)(ehdr->entry);
11    for (i = 0; i < 200 * 512; i++) {

```

```

12     *(uint8_t *) (elf + i) = *(uint8_t *) (elf + i + offset);
13 }
14 enterUserSpace(uMainEntry);
15 }
```

已知键盘中断号为0x21，在lab2/kernel/kernel/dolrq.s中，我将irqKeyboard的中断向量号0x21压入栈。

```

1 irqKeyboard:
2     pushl $0
3             # TODO: 将irqKeyboard的中断向量号压入栈
4     pushl $0x21
5     jmp asmDoIrq
```

在lab2/bootloader/start.s中设置esp寄存器的值，这部分和lab1中一模一样，入口地址为0x8000。

```

1 .code32
2 start32:
3     movw $0x10, %ax # setting data segment selector
4     movw %ax, %ds
5     movw %ax, %es
6     movw %ax, %fs
7     movw %ax, %ss
8     movw $0x18, %ax # setting graphics data segment selector
9     movw %ax, %gs
10
11    movl $0x8000, %eax # TODO: setting esp
12    movl %eax, %esp
13    jmp bootMain # jump to bootMain in boot.c
```

2. 完善中断服务例程

根据各中断给出的中断号，在lab2/kernel/kernel/irqHandle.c的irqHandle函数中，根据不同的中断填充其对应调用的处理函数。

```

1 void irqHandle(struct TrapFrame *tf) { // pointer tf = esp
2     asm volatile("movw %%ax, %%ds"::"a"(KSEL(SEG_KDATA)));
3     switch(tf->irq) {
4         // TODO: 填好中断处理程序的调用
5         case -1:
6             break;
7         case 0xd:
8             GProtectFaultHandle(tf);
9             break;
10        case 0x21:
11            KeyboardHandle(tf);
12            break;
13        case 0x80:
14            syscallHandle(tf);
15            break;
16        default:
17            assert(0);
18    }
19 }
```

GProtectFaultHandle函数和syscallHandle函数已经写好了，只需完成KeyboardHandle函数即可。根据输入字符的种类（退格符/回车符/正常字符）分情况进行处理，在这个过程中得注意维护光标的位置。当输入字符为可打印字符时，直接将其打印到显存中。

```
1 void KeyboardHandle(struct TrapFrame *tf){
2     uint32_t code = getKeyCode();
3     if(code == 0xe){ // 退格符
4         // 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
5         if(bufferTail > bufferHead && keyBuffer[bufferTail] != '\n')
6         {
7             int currentHead = bufferTail - 1;
8             while (currentHead > bufferHead && keyBuffer[currentHead - 1] != '\n')
9             {
10                 currentHead--;
11             }
12             if(displayCol > 0)
13             {
14                 displayCol--;
15             }
16             keyBuffer[bufferTail] = '\0';
17             bufferTail--;
18             int sel = USEL(SEG_UDATA);
19             char character = 0;
20             uint16_t data = 0;
21             int pos = 0;
22             asm volatile("movw %0, %%es"::"m"(sel));
23             for (int i = 0; i < bufferTail - currentHead; i++) {
24                 asm volatile("movb %%es:(%1), %0":="r"(character):"r"
25                             (keyBuffer+currentHead+i));
26                 data = character | (0x0c << 8);
27                 pos = (80 * displayRow + displayCol) * 2;
28                 asm volatile("movw %0, (%1)"::"r"(data), "r"(pos + 0xb8000));
29             }
30         }else if(code == 0x1c){ // 回车符
31             // TODO: 处理回车情况
32             displayCol = 0;
33             displayRow++;
34             keyBuffer[bufferTail] = '\n';
35             bufferTail++;
36         }else if(code < 0x81){ // 正常字符
37             // TODO: 注意输入的大小写的实现、不可打印字符的处理
38             char ch = getChar(code);
39             if (ch >= 0x20) {
40                 putChar(ch);
41                 keyBuffer[bufferTail] = ch;
42                 bufferTail++;
43                 int sel = USEL(SEG_UDATA);
44                 char character = ch;
45                 uint16_t data = 0;
46                 int pos = 0;
47                 asm volatile("movw %0, %%es"::"m"(sel));
48                 data = character | (0x0c << 8);
49                 pos = (80 * displayRow + displayCol) * 2;
```

```

50         asm volatile("movw %0, (%1)::\"r"(data), \"r"(pos + 0xb8000));
51         displayCol++;
52         if (displayCol >= 80) {
53             displayCol = 0;
54             displayRow++;
55         }
56         while (displayRow >= 25) {
57             scrollScreen();
58             displayRow--;
59             displayCol = 0;
60         }
61     }
62 }
63 updateCursor(displayRow, displayCol);
64 }
```

(2) 实现printf的处理例程

我在lab2/kernel/kernel/irqHandle.c中完成了与写显存内容密切相关的函数syscallPrint，根据实验手册维护段选择子，并实现了光标的维护和打印到显存，同时要考虑换行和翻页问题。

```

1 void syscallPrint(struct TrapFrame *tf) {
2     int sel = USEL(SEG_UDATA); //TODO: segment selector for user data, need
3     further modification
4     char *str = (char*)tf->edx;
5     int size = tf->ebx;
6     int i = 0;
7     int pos = 0;
8     char character = 0;
9     uint16_t data = 0;
10    asm volatile("movw %0, %%es::\"m"(sel));
11    for (i = 0; i < size; i++) {
12        asm volatile("movb %%es:(%1), %0::=\"r"(character):\"r"(str+i));
13        // TODO: 完成光标的维护和打印到显存
14        if(character == '\n')
15        {
16            displayCol = 0;
17            displayRow++;
18            if (displayRow >= 25) {
19                scrollScreen();
20                displayRow = 24;
21                displayCol = 0;
22            }
23        }
24    }
25    else
26    {
27        data = character | (0x0c << 8);
28        pos = (80 * displayRow + displayCol) * 2;
29        asm volatile("movw %0, (%1)::\"r"(data), \"r"(pos+0xb8000));
30        displayCol++;
31    }
32    if(displayCol >= 80)
33    {
34        displayCol = 0;
35        displayRow++;
```

```

34         if (displayRow >= 25) {
35             scrollScreen();
36             displayRow = 24;
37             displayCol = 0;
38         }
39     }
40 }
41 updateCursor(displayRow, displayCol);
42 }
```

(3) 完善printf的格式化输出

我在lab2/lib/syscall.c中完成了printf函数，支持%d, %x, %s,%c四种格式转换说明符。该部分需要用到文件中已封装好的转换函数，根据不同的格式转换说明符进行相应处理即可。

```

1 void printf(const char *format,...){
2     int i=0; // format index
3     char buffer[MAX_BUFFER_SIZE];
4     int count=0; // buffer index
5     //int index=0; // parameter index
6     void *paraList=(void*)&format+sizeof(uint32_t); // address of format in
stack
7     int state=0; // 0: legal character; 1: '%'; 2: illegal format
8     int decimal=0;
9     uint32_t hexadecimal=0;
10    char *string=0;
11    char character=0;
12    while(format[i]!=0){
13        char c = format[i++];
14        // TODO: in lab2
15        if(state == 0)
16        {
17            if(c == '%')
18            {
19                state = 1;
20            }
21            else
22            {
23                buffer[count] = c;
24                count++;
25            }
26            continue;
27        }
28        else if(state == 1)
29        {
30            if(c == 'd')
31            {
32                decimal = *(int *)paraList;
33                paraList += 4;
34                count = dec2Str(decimal, buffer, MAX_BUFFER_SIZE, count);
35            }
36            else if(c == 'x')
37            {
38                hexadecimal = *(uint32_t *)paraList;
39                paraList += 4;
```

```

40         count = hex2Str(hexadecimal, buffer, MAX_BUFFER_SIZE,
41         count);
42     }
43     else if(c == 'c')
44     {
45         character = *(char *)paraList;
46         paraList += 4;
47         buffer[count++] = character;
48         if (count == MAX_BUFFER_SIZE) {
49             syscall(SYS_WRITE, STD_OUT, (uint32_t)buffer,
50             (uint32_t)count, 0, 0);
51             count = 0;
52         }
53     }
54     else if(c == 's')
55     {
56         string = *(char **)paraList;
57         paraList += 4;
58         count = str2Str(string, buffer, MAX_BUFFER_SIZE, count);
59     }
60     else
61     {
62         state = 2;
63         return;
64     }
65     state = 0;
66 }
67 if(count!=0)
68     syscall(SYS_WRITE, STD_OUT, (uint32_t)buffer, (uint32_t)count, 0,
69     0);
70 }
71 }
```

(4) 实现getChar, getStr的处理例程

参照printf函数的实现过程，我首先在lab2/kernel/kernel/irqHandle.c中实现了getChar和getStr的系统调用函数syscallGetChar和syscallGetStr，同样使用keyBuffer数组来辅助相应功能的实现。

```

1 void syscallGetChar(struct TrapFrame *tf){
2     // TODO: 自由实现
3     int flag = 0;
4     if(keyBuffer[bufferTail - 1] == '\n')
5     {
6         flag = 1;
7     }
8     while (bufferTail > bufferHead && keyBuffer[bufferTail-1] == '\n')
9     {
10        bufferTail--;
11        keyBuffer[bufferTail] = '\0';
12    }
13    if (bufferTail > bufferHead && flag == 1)
```

```
14     {
15         tf->eax = keyBuffer[bufferHead];
16         bufferHead++;
17     }
18     else
19     {
20         tf->eax = 0;
21     }
22 }
23
24 void syscallGetStr(struct TrapFrame *tf){
25     // TODO: 自由实现
26     int flag = 0;
27     int sel = USEL(SEG_UDATA);
28     asm volatile("movw %0, %%es": :"m"(sel));
29     if (keyBuffer[bufferTail - 1] == '\n')
30     {
31         flag = 1;
32     }
33     while (bufferTail > bufferHead && keyBuffer[bufferTail-1] == '\n')
34     {
35         bufferTail--;
36         keyBuffer[bufferTail] = '\0';
37     }
38     if (flag == 0 && bufferTail - bufferHead < tf->ebx)
39     {
40         tf->eax = 0;
41     }
42     else
43     {
44         //参考syscall.c中的dec2str函数
45         char str[256];
46         int count = 0, i = 0;
47         int decimal = bufferTail - bufferHead;
48         int temp;
49         int number[16];
50         // 处理负数
51         if (decimal < 0) {
52             str[count++] = '-';
53             if (count == 256)
54             {
55                 putchar('\n');
56                 count = 0;
57             }
58             temp = decimal / 10;
59             number[i++] = temp * 10 - decimal;
60             decimal = temp;
61             while (decimal != 0)
62             {
63                 temp = decimal / 10;
64                 number[i++] = temp * 10 - decimal;
65                 decimal = temp;
66             }
67         }
68     else
69     {
```

```

70     temp = decimal / 10;
71     number[i++] = decimal - temp * 10;
72     decimal = temp;
73     while (decimal != 0)
74     {
75         temp = decimal / 10;
76         number[i++] = decimal - temp * 10;
77         decimal = temp;
78     }
79 }
80 // 将数字转换为字符串
81 while (i != 0)
82 {
83     str[count++] = number[i - 1] + '0';
84     if (count == 256)
85     {
86         putChar('\n');
87         count = 0;
88     }
89     i--;
90 }
91 str[count] = '\0';
92 // 输出字符串长度
93 while (str[i] != '\0') putChar(str[i++]);
94 putChar('\n');
95     for (int i = 0; i < tf->ebx && i < bufferTail-bufferHead; i++) {
96         asm volatile("movb %1, %%es:(%0)::\"r\"(tf->edx+i), \"r"
97 (keyBuffer[bufferHead+i]));
98     }
99     tf->eax = 1;
100 }
```

最后在lab2/lib/syscall.c中实现了getChar和getStr函数，这两个函数通过判断返回值是否为0来进行返回。

```

1 char getChar(){ // 对应SYS_READ STD_IN
2     // TODO: 实现getChar函数, 方式不限
3     char c = 0;
4     while (c == 0)
5     {
6         c = (char)syscall(SYS_READ, STD_IN, 0, 0, 0, 0);
7     }
8     return c;
9 }
10
11 void getStr(char *str, int size){ // 对应SYS_READ STD_STR
12     // TODO: 实现getStr函数, 方式不限
13     int c = 0;
14     while (c == 0)
15     {
16         c = syscall(SYS_READ, STD_STR, (uint32_t)str, size, 0, 0);
17     }
18     return;
```

二、问题回答

1.ring3的堆栈在哪里？

ring3的堆栈信息由用户程序自行管理，通常保存在用户程序的堆栈段中，其堆栈指针（ESP）和段寄存器（SS）在用户程序运行时通过代码设置。

2.IA-32提供了4个特权级，但TSS中只有3个堆栈位置信息，分别用于ring0, ring1, ring2的堆栈切换。为什么TSS中没有ring3的堆栈信息？

TSS中仅存储更高特权级的堆栈信息，用于特权级提升时的堆栈切换。在IA-32体系架构中，ring3是用户态，是最低特权级，任何一个向高特权级进行的转移都不可能转移到ring3。因此，不需要在TSS中保存ring3的堆栈信息。

3.我们在使用eax, ecx, edx, ebx, esi, edi前将寄存器的值保存到了栈中，如果去掉保存和恢复的步骤，从内核返回之后会不会产生不可恢复的错误？

会产生不可恢复的错误。因为内核在处理系统调用或中断时，可能会修改这些通用寄存器的值。如果不保存它们的原始值，这些修改将直接影响到用户态程序的寄存器状态。用户态程序依赖于这些寄存器的值来继续执行。如果这些值被内核意外修改，用户态程序的行为将变得不可预测，可能导致程序崩溃、数据错误或其他异常行为。

三、实验心得

本次实验让我深入理解了计算机系统从输入到输出的底层机制，尤其是中断机制和系统调用的实现原理。通过手动实现库函数printf、getChar、getStr，我不仅对这些常用函数有了更直观的认识，还体会到从理论到实践的挑战。在实验过程中，我遇到了各种各样的问题，一开始make时不同文件的代码老是会频繁报错，经常改完还报错，导致我得频繁地make/make clean，甚至第一次弹出窗口时由于IDT表初始化错误，窗口一直在闪。最终，当实验成功运行，屏幕上显示出预期的输出时，我感受到了前所未有的成就感。