

OS Lab 2 实验报告

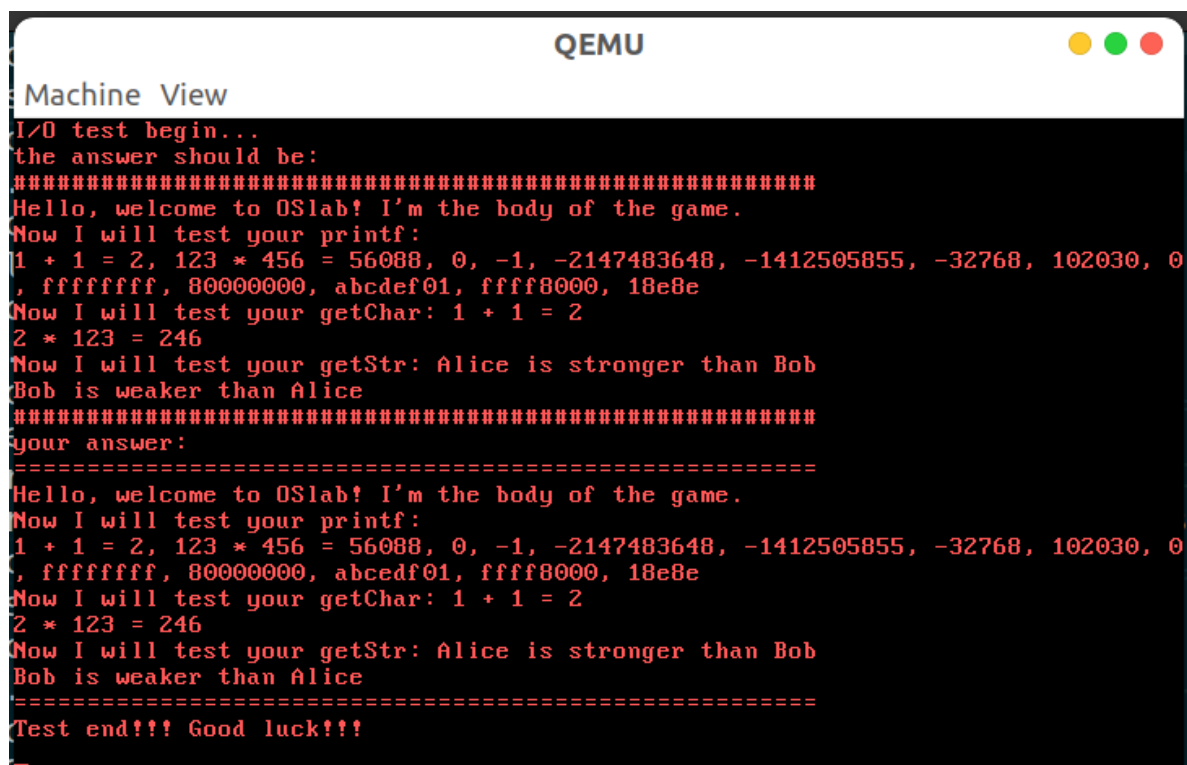
赵思衡 201300024

zhaosh@smail.nju.edu.cn

一、实验进度

我完成了所有内容。

二、实验结果



```
Machine View
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 Bob
Bob is weaker than Alice
=====
Test end!!! Good luck!!!
```

三、代码介绍

- bootloader/Makefile

为了缩减MBR的大小至510字节内，故修改了Makefile的内容

```
#objcopy -O binary bootloader.elf bootloader.bin
objcopy -S -j .text -O binary bootloader.elf
bootloader.bin
```

- bootloader/start.S

```
movl $0x1fffffff,%eax      # DONE:  setting esp
movl %eax,%esp
```

设置了 esp 内容。因为kernel被加载到内存的 0x100000 处，所以为了保证 kernel 与 栈帧不互相覆盖，故设置 esp 为 0x1ffff。

- bootloader/boot.c

在 bootMain 函数中进行了 kernel 的加载。在这个过程中出现了一些问题，或许是发现了框架代码潜在的bug。

- kernel/kernel/dolrq.c

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

根据手册知：键盘的中断向量号为 33，故将 \$0x21 压入栈中

- kernel/kernel/idt.c

在 setIntr 函数中初始化中断门；在 setTrap 中初始化陷阱门；在 initIdt 函数中初始化 IDT 表。

- kernel/kernel/irqHandle.c

在 `irqHandle` 函数中根据中断向量号进行中断处理函数的分配；在 `KeyboardHandle` 函数中进行对键盘中断的细节处理；在 `syscallPrint` 函数中对 `SYS_WRITE` 系统调用的 `STD_OUT` 标准输出进行处理，完成光标的维护和打印到显存；分别在 `syscallGetChar` 函数和 `syscallGetStr` 函数中对 `SYS_READ` 系统调用的 `STD_IN` 标准输出和 `STD_STR` 字符串输出进行处理。

- `kernel/kernel/kvm.c`

在 `loadUMain` 函数中通过内核加载用户程序。

- `kernel/main.c`

在 `kEntry` 函数中对内核进行了一系列初始化。

- `lib/syscall.c`

实现了 `getChar`、`getStr` 和 `printf` 三个用户函数。

四、遇到的问题与想法

1. `bootMain` 处框架代码问题

在 `bootloader/boot.c` 文件的 `bootMain` 函数中，框架代码给的提示实现如下：

```

for (i = 0; i < 200; i++) {
    readSect((void*)(elf + i*512), 1+i);
}

// TODO: 填写kMainEntry、phoff、offset

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

```

最初按照这个框架思路来写的时候，一直会出现陷入 boot from hard disk 不继续进行的问题。我在 kMain 中设置了 assert 却未执行，这显然是因为还没有正确加载内核的缘故。所以 bug 只可能出现在 bootMain 中。

接下来我们继续看框架代码。因为其中 elf 为 0x100000，这就默认了 kernel 中的所有需要 load 的程序段都在 0x100000 这个地址后，然而我们通过 Makefile 只能保证 .text 段被链接到 0x100000 处，其他的都只能听 gcc 由命。例如如下 gcc=9.4 版本就会将第一个程序段链接到 0xff000 处。所以这样是不可行的。

Program Headers:							
Type	Offset	VirtAddr	PhysAddr	FileSiz	MemSiz	Flg	Align
LOAD	0x000000	0x000ff000	0x000ff000	0x000d4	0x000d4	R	0x1000
LOAD BR & Bootloader	0x001000	0x00100000	0x00100000	0x00da6	0x00da6	R E	0x1000
LOAD	0x002000	0x00101000	0x00101000	0x004bc	0x004bc	R	0x1000
LOAD	0x003000	0x00103000	0x00103000	0x00120	0x01ee4	RW	0x1000
GNU_STACK	0x000000	0x00000000	0x00000000	0x00000	0x00000	RWE	0x10

所以我们应当对每个段进行分别装载，而不是只根据 offset 进行一个偏移。具体实现细节见工程代码。

2. printf 处 debug 记录

```
QEMU
Machine View
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, *U WUS H *come o *U WUS H *01b ! I'm the *U WUS H * of
*U WUS H *.
Now I will test your printf:
1 + 56088 = 4736, 0 * 0 = 2199552, 0, -32768, 0, 0, 0, 0, 0, ffff8000, 0, 0, 0,
0
Now I will test your getChar: 1 + 1 = * 123 = 246
Now I will test your getStr: Alice is stronger than is stronger than Alice
=====
Test end!!! Good luck!!!
```

在解决了上一个问题之后，程序可以正常的加载到用户程序了，但这时我却又遇见了输出乱码的问题。在阅读了用户程序的代码之后我发现，我将问题锁定到 `printf` 函数中的 `%s` 中。这时我发现我将 format index `i` 和 parameter index `index` 搞混了，这就会导致我的 `paraList` 越界。

五、思考题解答

Ex1:

计算机系统的中断机制在内核处理硬件外设的 I/O 这一过程中发挥了什么作用？

1. 计算机系统的中断机制使得内核可以更高效的处理I/O，使高速的CPU无需等待低速的I/O设备，而是在开始I/O之后，CPU可以做其他需要处理的事情，当I/O处理完之后再通过中断通知内核即可。

2. 计算机系统的中断机制使得内核可以更统一地管理I/O设备，只需提供给其中断号即可，这使得我们的机器可以连接更多的外设而无需事先装配其驱动程序等具有硬件特异性的东西。

Ex2:

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?

由于只是在外层到内层（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 并没有位于最外层 ring3 的堆栈信息；同时在 R3 进入 R0 的时候会把信息压入堆栈（同时自动加载 ring0 的 esp 和 ss），出来的时候只要从堆栈恢复就可以。

Ex3:

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?

如果内核用到了这些寄存器，那么值就被改变，如果这时没有将寄存器的值保存到了栈中，就可能会出现问題。

Ex4:

查阅相关资料，简要说明一下 `%d` , `%x` , `%s` , `%c` 四种格式转换说明符的含义。

- `%d` 代表十进制带符号整数 `int`
- `%x` 代表十六进制无符号整数 `uint`
- `%s` 代表字符串 `char*`
- `%c` 代表单个字符 `char`