

221220076

第二次实验报告

个人信息

姓名：落华栋

学号：221220076

邮箱：221220076@smail.nju.edu.cn

实验进度

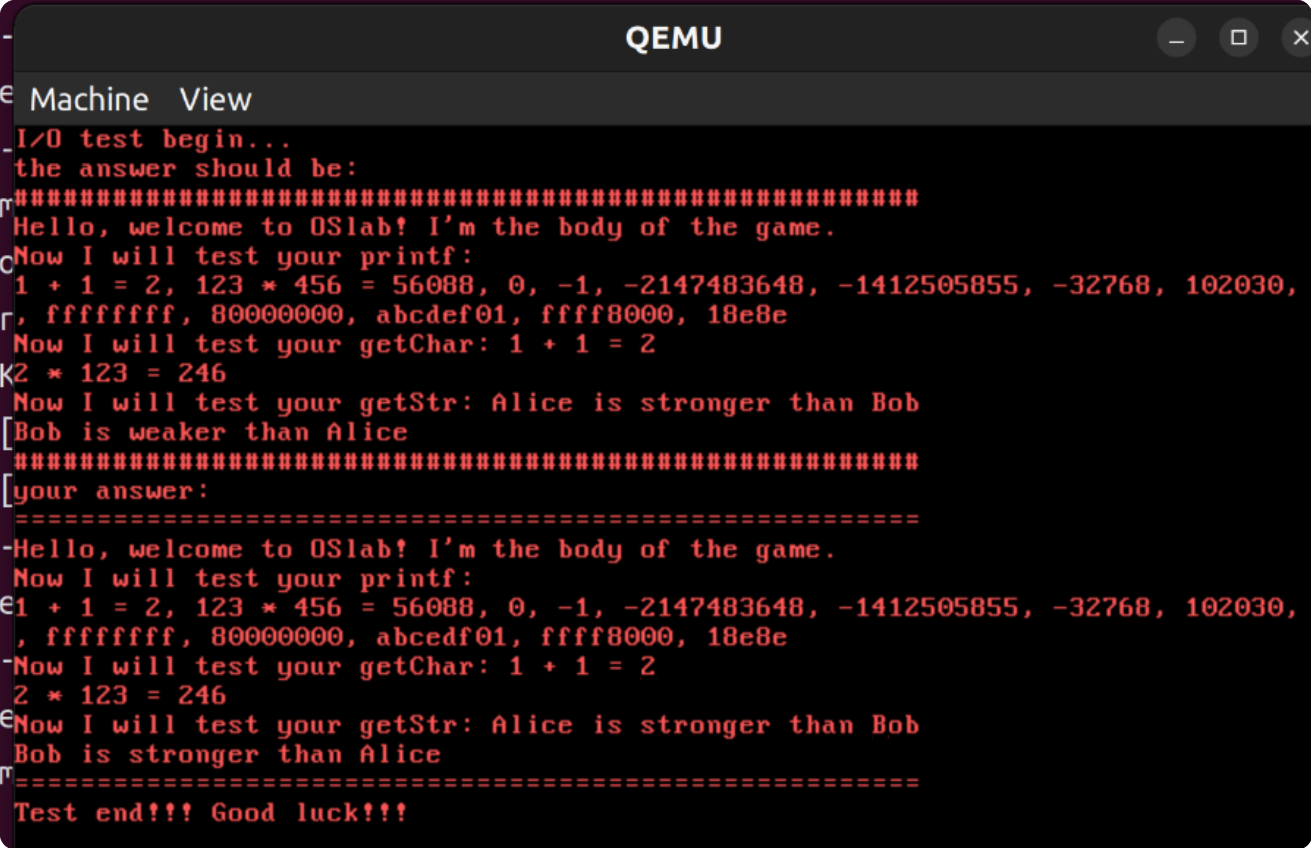
请注意：本人的实验环境为Ubuntu22.04，gcc版本为11.3.0。为适配环境，本人修改了makefile

（非常抱歉由于环境而给助教造的麻烦，如仍有问题，还请通过邮件或者QQ联系我，十分感谢）

我完成了全部任务，即通过磁盘加载引入内核, 区分内核态和用户态, 完善中断机制, 并基于中断实现用户态I/O函数实现。

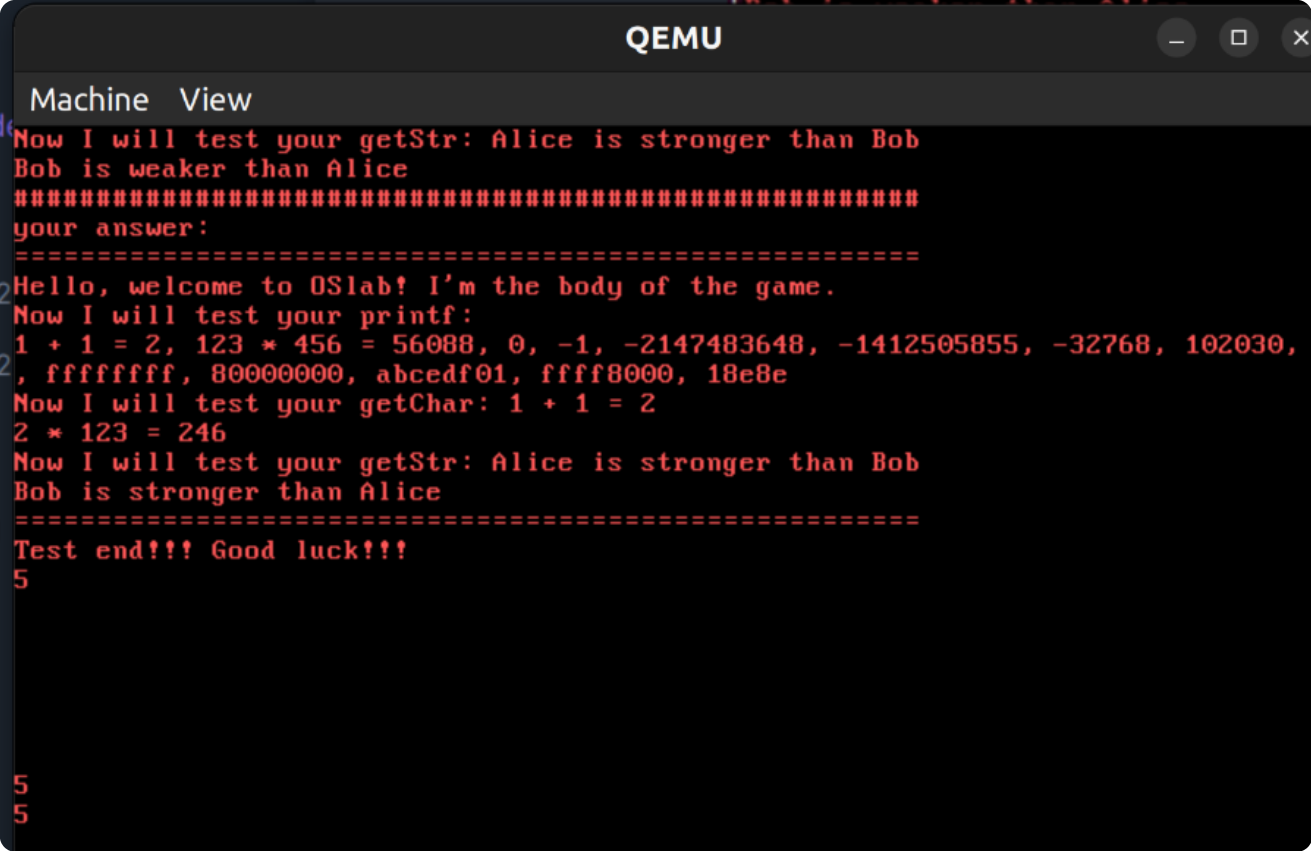
实验结果

下为测试结果



```
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,
, 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,
, 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 stronger than Alice
=====
Test end!!! Good luck!!!
```

下为滚屏测试，是在上述测试结束后的界面进行测试的，可以看出滚屏成功实现



```
QEMU
Machine View
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,
, 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 stronger than Alice
=====
Test end!!! Good luck!!!
5
5
```

其他不适合截屏，故未截图。

修改代码

环境适配导致makefile的修改

因为gcc版本过高导致了 `boot block is too large` 的问题，因此修改 `makefile`，将 `objcopy -O binary bootloader.elf bootloader.bin` 修改为了 `objcopy -S -j .text -O binary bootloader.elf bootloader.bin`。

修改的 `makefile` 路径为：`bootloader/makefile`

从实模式进入保护模式与加载内核

这里首先修改了 `bootloader/boot.c` 填写 `kMainEntry`、`phoff`、`offset`

完善初始化设置

- 首先初始化中断门和陷阱门,修改了 `idt.c` 中的 `setIntr` 和 `setTrap` 函数，之后在 `idt.c` 中为各中断及陷阱补全相应处理函数。
- 之后修改 `main.c` 的初始逻辑，由于各 `init` 函数已实现好, 故只需要加入各 `init` 函数即可, 此处不过多赘述。
- 填充对于中断处理函数的调用: 根据各中断给出的中断号, 在 `irqHandle.c` 文件中, 根据不同的中断填充其对应调用的处理函数。

由kernel加载用户程序

- 首先填写 `kvm.c` 中的 `loadUMain` 函数: 参照bootloader中加载内核的方式进行填写。但是 `bootMain` 从磁盘的前200个512字节中读取程序，所以在 `loadUMain` 中, 应从第201个区域开始读。
- 填写中断处理函数: 在 `irq_handle.c` 填写未完全实现的 `keyboardHandle`，`syscallPrint`，`syscallGetChar`，`syscallGetStr` 等函数. 具体可使用 `keyBuffer` 数组来辅助相应功能的实现。

实现库函数

均在 `lib/syscall.c` 中

- `printf`函数的实现: 使用文件中已封装好的转换函数, 根据不同的case进行处理
- `getChar`和`getStr`函数的实现: 循环调用 `syscall`，当返回值不为0时进行返回

一些思考

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?

在IA-32体系架构中, ring3是用户态, 具有最低的特权级, 只能访问受限的用户空间资源, 不能直接访问内核资源。而其他特权级到ring3的切换是通过使用不同的堆栈, 即加载任务的用户态堆栈指针来实现的, 不需要通过TSS中保存堆栈位置信息来实现切换。

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?

会发生不可恢复的错误。因为这些寄存器被当作通用寄存器, 用于存放临时数据和函数调用时的 参数传递。若没有正确保存和恢复, 这些寄存器中的值在内核执行过程中可能会被内核代码修改, 从而影响用户态的执行。