

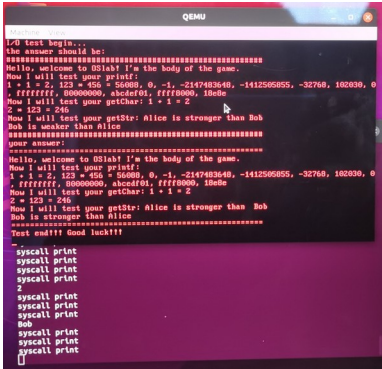
lab 2 实验报告

201300082 吴松栗 1103755779@qq.com

一.实验进度

我已完成所有实验部分

二 实验结果



```
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 * 2 = 2, 123 * 456 = 55680, 0, -1, -2147483648, -1412565855, -32768, 180939, 0
ffffffffff, 88000000, abcdabcd, ffffffff, 10000, 10000
Now I will test your getch: 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 * 2 = 2, 123 * 456 = 55680, 0, -1, -2147483648, -1412565855, -32768, 180939, 0
ffffffffff, 88000000, abcdabcd, ffffffff, 10000, 10000
Now I will test your getch: 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!!!

syscall print
syscall print
syscall print
syscall print
2
syscall print
syscall print
syscall print
Bob
syscall print
syscall print
syscall print
```

三 实验修改代码

- 1.在 doirq.S 中，查询 irqKeyboard 的中断向量号，将其填入。
- 2.在 memory.h 中找到 struct GateDescriptor 的结构，并按照手册对于 idt.c 中 interrupt gate, trap gate 依次初始化。
- 3.在 idt.c 中按照保护模式下 80386 执行指令过程中产生的异常表进行初始化 IDT 表, 为中断设置中断处理函数，以及 set_trap, setIntr。
- 4.然后在 boot.c 中填写 kMainEntry、phoff、offset,再仿照其写入 kvm.c。
- 5.根据不同中断号，在 irqHandle.c 中完成填写。
- 6 光标部分需要注意到光标到一定位置时需要更新位置。getchar 较为简单，借助系统调用实现即可。而 syscallgetchar 则是逻辑较为简单但实现起来稍有困难。
- 7 irqHandle.c 里的 keyboardHandle 函数。
- 8 键盘按键回显：阅读 serial.c，里面提供了显示的接口，阅读 keyboard.c 里面提供了读取键盘数据的接口，思路如下：
 - 1) . 先去 idt.c 里加上对应的中断号和处理函数
 - 2) . 利用 keyboard.c 里的 getKeyCode 获取键码
 - 3) . 利用 keyboard.c 里的 getChar 对上述的键码进行处理
 - 4) . 利用 serial.c 里的 putChar 进行输出
- 9 printf 处理例程：阅读代码可以知道 int80 指令最后会一路落到 syscallPrint 函数且利用手册中的代码块可以实现对应位置的字符打印，则处理换行和滚屏问题。思路如下
 - 1) . 得到数据字符
 - 2) . 判断是否为换行，如果是就换行
 - 3) . 不是换行就对应的打印到位置
 - 4) . 打印完后纵坐标加 1，判断是否满了换行
 - 5) . 最后判断是否需要滚屏
- 10 初始化：先清空 Buf，分别对 idt, 8259a, gdt, tss, vga device, keyboard device 初始化。

四。自由汇报

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？

中断是 I/O 中断处理器正常处理过程的机制。在此过程中，I/O 控制器产生中断，通过中断处理器执行中断操作。当外部设备的 I/O 模块准备好时，它会发送给 CPU 一个中断信号，CPU 则会做出响应，暂停当前程序的处理，去执行 I/O 设备的程序。如果没有中断机制，则 CPU 会不断检查轮询 I/O 操作是否完成，有中断机制，CPU 向 I/O 模块发送读指令，然后调度其它线程，当 I/O 模块 I/O 执行完成后，产生中断信号通知 CPU，CPU 将线程加入线程就绪队列并恢复上下文，当线程处于就绪队列后，可以被操作系统调度，从而继续执行读操作，此时会将数据从操作系统内核缓存读取到用户缓存。因而，中断机制的出现使得 操作系统减少了轮询，提高效率。

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？

因为例如当我们在 ring3，当转移只 ring1 时，堆栈将被自动切换到由 ss1 和 esp1 指定的位置。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

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？

根据寄存器使用的约定，有三个寄存器 eax、edx、ecx 是子程/函数可以随意使用而不用进行保存恢复的，而且 eax 更是作为函数的结果返回。其它的寄存器，如果子程/函数使用了，则必须进行保存恢复的操作，否则可能会产生不可恢复的错误。

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 表示按整型数据的实际长度输出数据；

%x 表示以十六进制数形式输出整数；

%s 用来输出一个字符串；

%c 用来输出一个字符。

5。

块大小	1KB	2KB	4KB
一个文件最大大小	16GB	256GB	2TB
文件系统最大大小	4TB	8TB	16TB

对于 1KB 的块大小

每一个 block 可以记录 $1000/4=256$ 个号码

12 个直接的块含有 12KB

单层 $256*1K=256K$

$256 * 256 = 65536k$

双层 $256 * 256 * 256 * 1K = 16777216k$

单文件最大总量 = $(12 + 256 + 65536 + 16777216) / (1024 * 1024) = 16.06G$

同理计算有 2KB 的

$2000 / 4 = 512$ 则有最大总量 256.50G

同理 4KB 的有 4.00TPS：当 block 单位容量为 4K 时，由于文件系统本身的限制(2T)和原结果有写不同

6 实验感受

(1)

在实验感到无从下手的时候，将框架代码每个文件的关系和功能整理清楚会很有帮助

(2)

按照框架给定函数的思路组织自己的代码会非常方便

(3)

通过对输出函数的代码编写，进一步理解了 `printf` 所引起的系统的一系列工作流。通过中断机制，系统可以实现在输出设备上输出，而通过一步步的抽象封装，输出函数变得易于调用。

其中最让我感到有趣的的就是 `printf` 一步步的封装过程，体现了代码低耦合的设计思想。