

Lab 2 实验报告

181250068 李淳

邮箱: 181250068@smail.nju.edu.cn

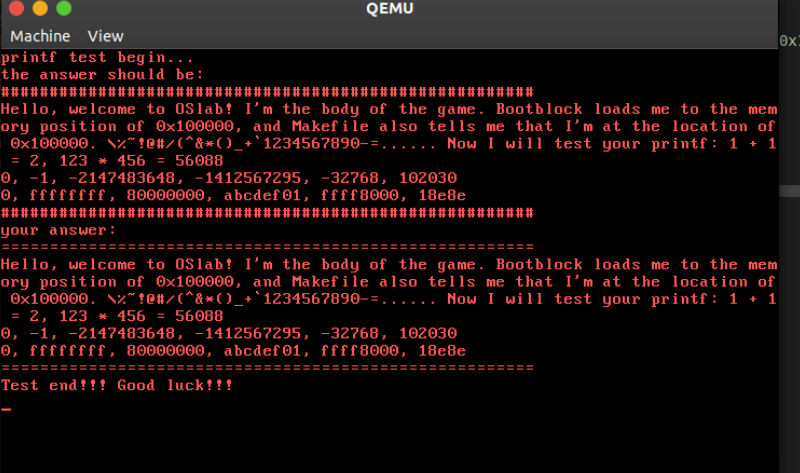
QQ: 936637810

实验进度

我完成了所有内容, `cp` 函数, 键盘按键的串口回显, `printf` 的处理例程和格式化输出

实验结果

```
18 printf("Test begin!\n");
19 printf("the answer should be:\n");
20 printf("#####\n");
21 printf("Hello, welcome to OSlab! I'm the body of the game. ");
22 printf("Bootblock loads me to the memory position of 0x100000, and Makefile also tells me that I'm at the location of
23 printf("\\%~!@#/(^&*()_+'1234567890-=..... ");
24 printf("Now I will test your printf: ");
25 printf("1 + 1 = 2, 123 * 456 = 56088\n0, -1, -2147483648, -1412567295, -32768, 102030\n0, ffffffff, 80000000, abcdef01
26 printf("#####\n");
27 printf("your answer:\n");
28 printf("=====\n");
29 printf("%s %scome %co%", "I
30 printf("%c%c%c%c! ", 'O', '
31 printf("I'm the %s of %s. %s
32 printf("\\%~!@#/(^&*()_+'123
33 printf("Now I will test your
34 printf("%d + %d = %d * %d
35 printf("%d, %d, %d, %d, %d,
36 printf("%x, %x, %x, %x, %
37 printf("=====\n");
38 printf("Test end!!! Good luck\n");
```



```
8192
1024
3960
1020
3943
1024
2
4
6
262
asdasdasdasdasdasdjaksljz

alskdjklaskjflj
lkkasjkl
@!#0&$()!@*#!!0&$()!#*
123214125124124
aaaaa]
```

实验修改的代码

lab2/lib/syscall.c 中的 `printf`, 从第75行-第112行

lab2/utls/genFS/func.c 中的 `cp`, 从第701行-第790行

lab2/kernel/kernel/idt.c 中的 `initIdt`, 第79行

lab2/kernel/kernel/irqHandle.c 中的 `keyboardHandle`, 第63行

lab2/kernel/kernel/irqHandle.c 中的 `syscallPrint`, 第88行-第110行

实验中遇到的问题

1. `CP` 函数不知道如何将文件拷贝到相应的位置

一开始不知道框架代码有没有提供能够将数据从一个地方复制到另一个地方的函数，找到了 `copydata` 后又不清楚被cp的文件的位置，因为它在 `lab2/utls/genFS/main.c` 是 `argv[1]`，最后感觉应该要去看看Makefile，然后猜测那个文件是应该是 `uMain.elf` 和 `fs.bin` 在同一个目录下面，直接打开就可以了

2. keyboardHandle 函数里面的 TrapFrame 不知道如何使用

现在依然不太明白这个Trap帧该怎么写，不过看手册里面写到TrapFrame的时候是在系统调用，键盘这里应该是硬件发出的中断，会不会是硬件发出的中断直接处理就行了？

3. printf 函数中可变长参数不知道如何寻找

最开始的时候看到 `paraList=(void*)&format` 知道应该是通过format这个指向栈顶的指针来找参数，从理论上讲，既然我们可以从format中取出所有的%开头的，指明参数列表中参数类型的占位符，那么我们完全可以根据不同的类型把所有的参数都找到，而没有必要每次寻找参数都将指针移动同样的距离。

但是在找%c的时候，我发现将指针移动 `sizeof(char)` 没法正确寻址，最后只能去查看了一下C语言的头文件 `stdarg.h`，发现里面用了这样一个宏

```
#define _INTSIZEOF(n) ((sizeof(n)+sizeof(int)-1)&~(sizeof(int) - 1))
```

我把各种类型带进去试了一下后发现这个宏是将不同类型都转换成sizeof(int)的整数倍，这才解决了问题，应该不管是哪个程序执行的过程中发生了压栈，栈指针都是按照sizeof(int)的整数倍变化的。

对内存分段机制，ext文件系统，内存分页机制的思考

联系

内存分段机制，ext文件系统和分页机制都涉及到了分块的想法，ext文件系统中最小的单位就是一个块，内存分段机制依赖一个个段中的基址和偏移量选址，内存分页机制是把虚拟内存分成了一个页。

为什么ext文件系统没有采用类似于分页机制中对物理页的组织方式，通过固定的两级索引来寻找物理页？

不知道

块大小参数的计算

按照多级索引来计算，如果一个块是1KB，由于0-11是数据块的索引，这块占用了12KB，对于二级索引，因为它指向的块中存储的是数据块的索引，因为一个块是4Byte，1KB的块就可以存256个块的索引。

这样可以计算出块大小为1KB时，一个文件的最大大小为 $12 + 256 + 256^2 + 256^3 = 16GB$

对于ext文件系统，采用的是32Bit的块索引空间，所以能最多能有 $1 * 2^{32} = 4TB$

2KB和4KB的时候可以同理计算，但是这些都是理论上的数值，实际使用的时候

幽灵空间

因为一个文件最少会占用一个块，现在的Linux使用的文件系统最小的块大小为4KB。

```
pkun@pkun:~/os2020$ vim t
pkun@pkun:~/os2020$ ls
t
pkun@pkun:~/os2020$ ls -ls
total 4
4 -rw-r--r-- 1 pkun pkun 2 3月 22 14:12 t
```

对目录的硬链接

假如我们可以对目录进行硬链接，现在有目录 `/a/b` 和目录 `/c/d`，然后将 `/d` 硬链接到 `/b` 上，现在我在 `/a/b` 处 `cd`，会回到 `/a`，但是由于 `d` 的 `inode` 记录了 `b` 的 `inode` 的全部信息，所以在 `/c/d` 处 `cd` 的话到底该回到哪里就成了一个问题，会影响对文件系统的使用。

另一方面，对目录的硬链接还有可能在遍历文件系统的时候造成死循环，比如 `/a/b/c/d/e`，其中 `e` 又硬链接到了 `a`，这就非常麻烦了

ring3的堆栈在哪？

ring3的堆栈应该就是程序默认的堆栈。因为只有在发生特权级转换的时候，CPU才会根据代码特权级的不同从TSS中取出相应的堆栈进行转换，根据手册上的信息，旧的SS和ESP的地址会被压到新的堆栈中。

保存寄存器的旧值

会，因为去掉保护和恢复的步骤后，在中断处理程序执行时会改变这些寄存器的值，当中断结束回来的时候，这些寄存器的值已经不是发生中断时那个现场的值了，这样原来的程序执行的时候就会出错