

实验二 系统调用

161130118 尹浚宇

908664035@qq.com

实验目的

了解 IA-32 中断机制的原理.

学会基于中断机制实现系统调用.

掌握 printf 和 scanf 的原理.

实验内容

实现的库函数 scanf 完成格式化输入和 printf 完成格式化输出.

完成函数 scanf 对应的系统调用 syscallScan.

程序设计思路

观察框架代码 lib\syscall.c, 在文件里已经提供了系统调用对应的函数 syscall, 所以在 printf 和 scanf 函数中, 我们只需要以正确的参数调用 syscall, 就能调用相应的系统功能进行处理. 继续观察框架代码 kernel\irqHandle.c, 我们可以得到系统内核对于中断的处理流程. 首先, 每当一个中断到来, 会进入函数 irqHandle 进行处理, 如果没有发生#GP 异常, 那么就会继续调用函数 syscallHandle 进行进一步处理, 因为我们这里只需要实现 printf 和 scanf 对应的系统调用, 所以函数 syscallHandle 只会处理内核函数 write 和 read. 在对应的函数中, 还会分别调用 syscallPrint 和 syscallScan 进行处理.

函数 syscallPrint 实现在视频映射的显存地址中写入内容, 完成字符串的打印.

函数 syscallScan 实现扫描按键状态获取输入完成格式化输入.

其中 syscallPrint 函数已经实现好了, 这里只需要实现函数 syscallScan. 这里的设计思路是通过一个循环不断读取有效键码存入 keyBuffer 中直到遇到回车中止循环. 由于这里我希望每读取一个键码就将其回显在控制台上, 而不是像框架代码一样读取完成后再一起显示, 所以这里在循环里每读取一个有效键码后, 就调用函数 putchar 进行回显, 具体代码如下:

```
while (1)
{
    uint32_t key_code = getKeyCode();
    char key_char = getChar(key_code);
    if (key_code != 0)
    {
        keyBuffer[bufferTail] = key_code;
        bufferTail = (bufferTail + 1) % MAX_KEYBUFFER_SIZE;
        putchar(key_char);
        if (key_char == '\n')
            break;
    }
}
```

实现好这里后, 我们再考虑 printf 和 scanf 的实现. 由于该函数参数不定的特性, 我们需要找到格式化字符串 format 之后的参数的起始地址. 这里由系统底层栈的特点, 很容易得到参数列表在内存中的起始地址, 具体代码如下:

```
char *vaList = (char*)&format + sizeof(char*);
```

找到了该地址后, 便可根据字符串 format 处理.

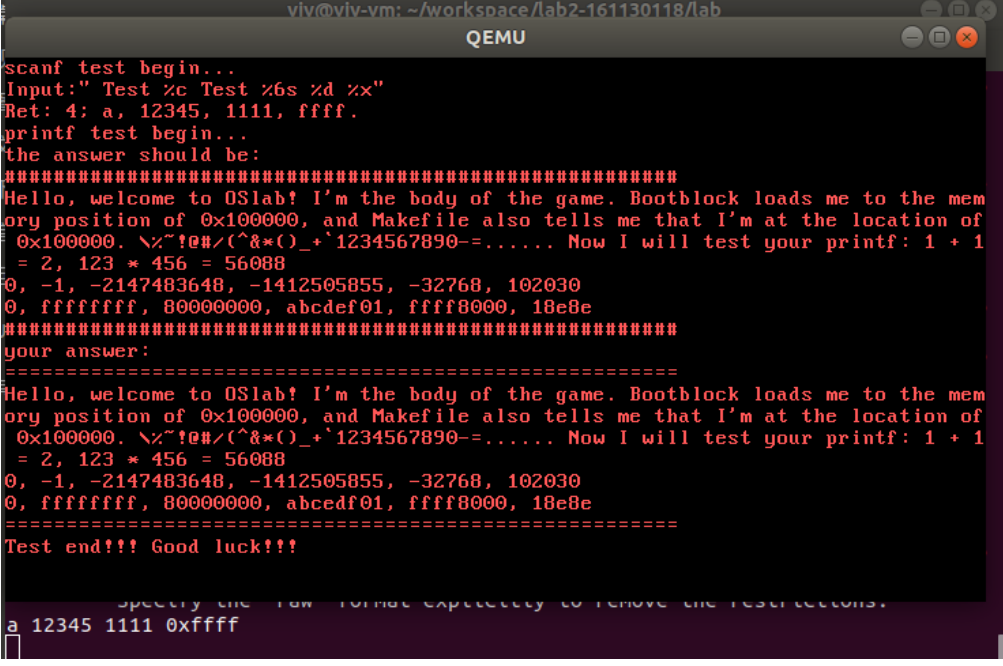
这里需要我们实现的格式化字符有 %c, %d, %s, %x, 框架代码为每个函数提供了数个实现好的

api, 在处理到相应情况时可以直接调用相应 api 进行处理。

这里处理的大致思路是, 循环读取格式化字符串 format 直到结束, 在循环读取的过程中检测格式化字符, 对于不同的格式化字符调用不同的 api 进行处理, 然后使 vaList 指向下一个参数的地址。

这里需要注意的是, i386 规定 1 字节压栈要扩充到 4 字节, 所以 printf 在处理%c 时, vaList 更新地址要加 4 而不是加 1。

程序运行结果如下图:



```
viv@viv-vm: ~/workspace/lab2-161130118/lab
QEMU

scanf test begin...
Input:" Test %c Test %6s %d %x"
Ret: 4: a, 12345, 1111, ffff.
printf test begin...
the answer should be:
#####
Hello, welcome to OSlab! I'm the body of the game. Bootblock loads me to the mem
ory position of 0x100000, and Makefile also tells me that I'm at the location of
0x100000. \%^!@#/(^&*())_+`1234567890-=..... Now I will test your printf: 1 + 1
= 2, 123 * 456 = 56088
0, -1, -2147483648, -1412505855, -32768, 102030
0, ffffffff, 80000000, abcdef01, ffff8000, 18e8e
#####
your answer:
=====
Hello, welcome to OSlab! I'm the body of the game. Bootblock loads me to the mem
ory position of 0x100000, and Makefile also tells me that I'm at the location of
0x100000. \%^!@#/(^&*())_+`1234567890-=..... Now I will test your printf: 1 + 1
= 2, 123 * 456 = 56088
0, -1, -2147483648, -1412505855, -32768, 102030
0, ffffffff, 80000000, abcdef01, ffff8000, 18e8e
=====
Test end!!! Good luck!!!

Specify the raw format explicitly to remove the restrictions.
a 12345 1111 0xffff
```

实验心得

在开始写代码前仔细阅读框架代码能提供很大帮助。