

Os lab2 系统调用实验报告

一. 实验目的

1. 学习基于中断的完整的系统调用机制
2. 了解并实现 printf、scanf 实现原理
3. 熟悉底层的 io 实现过程

二. 实验设计

1. 程序的流程是：

- (1) Bootloader 从实模式进入保护模式，加载内核至内存，跳转执行
- (2) 内核初始化 IDT（中断描述符表），初始化 GDT（全局描述符表），初始化 TSS（任务状态段）
- (3) 内核加载用户程序至内存，对内核堆栈进行设置，通过 iret 切换至用户空间，执行用户程序
- (4) 用户程序调用库函数 scanf 和 printf 陷入内核，由内核完成读取键盘输入，在视频映射的显存地址中写入完成字符串的打印等内容
- (5) 完成测试用例

2. 需要完成的任务：

- (1) lib/syscall.c 的 printf 和 scanf
- (2) kernel/kernel/irqHandle.c 中的 syscallScan

3. 代码思路

(1) printf

函数框架将 printf 第一个参数 format 逐个读入；

如果不是特定的 token 则输入到 buffer 中，则每当 buffer 满了则调用 syscall 写接口到 std_out；

实现 token 的思路利用 switch 语句检查后的关键词，然后调用对应接口（框架代码中已给出）处理，其中需要思考的是如何获取函数的更多参数，利用栈的原理，将指针 paraList 从 format 依次增 4 个字节即可访问对应的参数，因为函数的参数是从高到低压栈，参数列表按从右到左顺序。利用指针类型转换可以实现取到确定地址所需类型的变量！index 表示解析的 token 个数

```
switch(format[i]){
    case 'd':{
        paraList+=4;
        decimal=*(int*)paraList;
        count=dec2Str(decimal,buffer,MAX_BUFFER_SIZE,count);
        index++;
        break;
    }
    case '%':{
        buffer[count]='%';
        count++;
        break;
    }
    case 'x':{
        paraList+=4;
        hexadecimal=*(uint32_t*)paraList;
        count=hex2Str(hexadecimal,buffer,MAX_BUFFER_SIZE,count);
        index++;
        break;
    }
    case 'c':{
        paraList+=4;
        character=(char)*(int*)paraList;
        buffer[count]=character;
        count++;
        index++;
        break;
    }
    case 's':{
        paraList+=4;
        string=*(char**)paraList;
        count=str2Str(string,buffer,MAX_BUFFER_SIZE,count);
        index++;
        break;
    }
    default:return -1;break;
}
```

(2) scanf

和 printf 函数实现功能类似，首先逐步读入第一个参数对应数组 format，
每当 buffer 是空，则调用 syscall 的读接口到 std_in, 如果 format 中不是%的 token，则要在 buffer 中
严格匹配普通字符，否则返回-1；如果是 format 有空格则对应跳过 buffer 中的空格，如果解析到%，则
利用 switch 语句查看解析类型，然后调用特定接口（框架中已给出的函数）。循环在需要的地方使用
buffer 中的数据给传入的参数，参数获取也是利用指针在栈帧中不断加 4 字节获取，和 printf 不同的是
scanf 里参数全是地址，形式上要**利用例如*（void**）**的形式取得对应位置的指针值。
在 switch 外，单独实现了%?s的功能，?表示一个正整数，首先计算 format 中?的取值（循环即可）
然后调用对应的接口，这个**正整数 k 表示读入字符串的长度，k+1 表示包括结束符 ‘\0’ 的长度**，作为参
数 avail 传入 string2String2

Switch 实现截图

```
while(format[i]!=0){
// todo: support more format %s %d %x and so on
if(format[i]!='%'){
    i++;
    switch(format[i]){
        case 'd':{
            paraList+=4;
            decimal=*(int**)paraList;
            if(-1==str2Dec(decimal,buffer,MAX_BUFFER_SIZE,&count))
                return -1;
            index++;
            //printf("d:%d,end d\n",*decimal);
            break;
        }
        case '%':{
            if(buffer[count]==0){
                do{
                    r=syscall(SYS_READ, STD_IN, (uint32_t)buffer, (uint32_t)MAX_BUFFER_SIZE, 0, 0);
                }while(r == 0 || r == -1);
                count=0;
            }
            //if normal character match wrongly then return -1;
            if('%'!=buffer[count])
                return -1;
            count++;
            break;
        }
        case 'x':{
            // printf("begin x\n");
            paraList+=4;
            hexadecimal=*(int**)paraList;
            if(-1==str2Hex(hexadecimal,buffer,MAX_BUFFER_SIZE,&count))
                return -1;
            index++;
            //printf("x:%x,end x\n",*hexadecimal);
            break;
        }
        case 'c':{
            //printf("begin c\n");
            paraList+=4;
            break;
        }
    }
    i++;
}
```

%ks 实现截图

```
if(format[i]>='0'&&format[i]<='9'){
    uint32_t k=format[i]-'0';
    i++;
    while(format[i]>='0'&&format[i]<='9'){
        k=10*k+format[i]-'0';
        i++;
    }
    if(format[i]!='s')
        return -1;
    else{
        paraList+=4;
        string=*(char**)paraList;
        //k plus one because taking '\0' into consideration
        if(-1==str2Str2(string,k+1,buffer,MAX_BUFFER_SIZE,&count))
            return -1;
        //printf("begin %ds\n",k);
        /*int j=0;
        while(j<k){
            if(buffer[count]==0){
                do{
                    r=syscall(SYS_READ, STD_IN, (uint32_t)buffer, (uint32_t)MAX_BUFFER_SIZE, 0, 0);
                }while(r == 0 || r == -1);
                count=0;
            }
            string[j]=buffer[count];
            count++;
            j++;
        }
        string[j]='\0';
        count++;
        index++;
        //printf("s:%s,end %ds\n",string,k);
        */
        index++;
    }
    break;
}
```

严格匹配实现截图：

```
if(buffer[count]==0){
do{
    r=syscall(SYS_READ, STD_IN, (uint32_t)buffer, (uint32_t)MAX_BUFFER_SIZE, 0, 0);
}while(r == 0 || r == -1);
count=0;
}
//if normal character match wrongly then return -1;
//printf("buffer[count]:%c,format[i]:%c\n",buffer[count],format[i]);
if(format[i]!=buffer[count])
    return -1;
count++;
```

(3) 对 syscall.c 框架的 API 函数大致理解

dec2Str: 传入一个 int, 打印

hex2Str: 传入一个 uint32_t, 按 16 进制打印

str2Str: 传入一个字符串, 打印

matchWhiteSpace: 跳过 buffer 中目前所指向的连续的空格, 直到下一个有效值或结束符为止

str2Dec: 传入 buffer 作为接受缓冲数组, 写入一个 int* 指针所指向的区域

str2Hex: 传入 buffer 作为接受缓冲数组, 写入一个 int* 指针所指向的区域 (在 bit 级别上 int 和 uint_32 并无差别)

str2Str2: 传入一个 buffer 作为接受缓冲数组, 传入 avail 为包括结束符在内的有效写入长度, 写入一个 char* (即 string) 指针所指向的区域

(4) syscallScan

按照提示, 需要利用 getKeyCode (实现在 keyboard.c 中) 获取当前的 key code

然后把 keycode 循环写入 keyBuffer 中

这里有几个要注意的地方 (坑)

第一.

getKeyCode 函数的实现当 **没有键盘输入** 的时候 **默认返回 0**, 这是在阅读 keyboard.c 发现的
所以在读入之前要 **先判断键码是否不为 0**, 才能写入 keybuffer

第二

这里键码不需要知道具体的值, 只需要 **getChar** (键码) 后判断对应字符取值即可

第三

Buffertail 的加法是模加法, 这个容易忽略

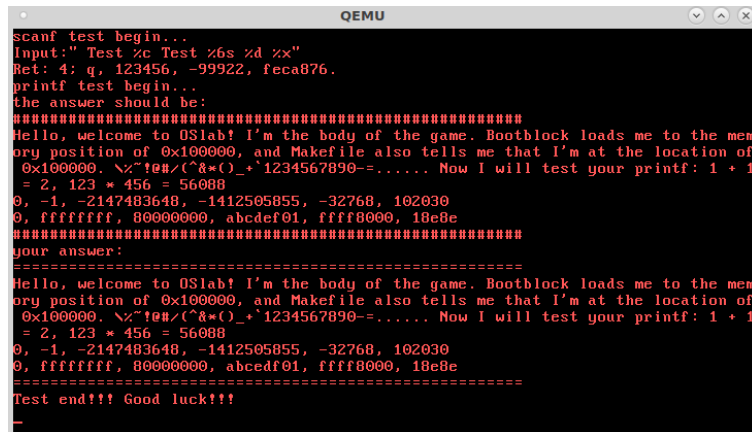
在注意到上述三点后只要 **循环读** keybuffer 即可, 直到接收到一个 **换行符或制表符**, 然后返回即可

下为代码实现截图

```
// TODO: get key code by using getKeyCode and save it into keyBuffer
uint32_t key_code=0;
while(getChar(key_code)!='\n' && getChar(key_code)!='\t'){
    key_code=getKeyCode();
    if(key_code!=0){
        if(getChar(key_code)=='\n' || getChar(key_code)=='\t')
            keyBuffer[bufferTail]=0;
        else
        {
            keyBuffer[bufferTail]=key_code;
        }
        bufferTail++;
        bufferTail%=MAX_KEYBUFFER_SIZE;
    }
}
```

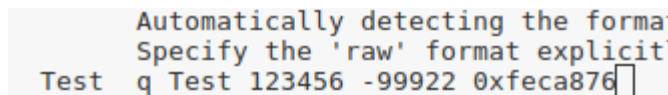
三. 实验结果

Make clean、make、make play 运行代码获取如下结果（输入仅供参考）：



```
QEMU
scanf test begin...
Input:" Test %c Test %6s %d %x"
Ret: 4; q, 123456, -99922, feca876.
printf test begin...
the answer should be:
=====
Hello, welcome to OSlab! I'm the body of the game. Boothblock loads me to the mem
ory position of 0x100000, and Makefile also tells me that I'm at the location of
0x100000. \x"feq/("8*(*)_+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. Boothblock loads me to the mem
ory position of 0x100000, and Makefile also tells me that I'm at the location of
0x100000. \x"feq/("8*(*)_+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!!!
```

对应输入流：



```
Automatically detecting the format
Specify the 'raw' format explicitly
Test q Test 123456 -99922 0xfeca876
```

四. 实验心得

- (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
比如我在实现 syscallScan 的时候，刚开始很茫然，后来仔细阅读框架代码，发现了 keyboard 中有 getChar、getKeyCode 函数的代码，阅读后马上有了思路。
- (2) 按照框架给定函数的思路组织自己的代码会非常方便
本次实验给的接口极大减少了自己的代码量，当然框架 API 也有一些错误，比如 api 的 string2Dec、string2Hexadec，读入 syscall 在刚开始循环前调用一次就可以了，不必在循环中反复检查 buffer 是否空然后调用，因为读取整型在无效字符（分隔符）出现就结束，循环体中调用 syscall 会导致最后一次循环（将要完成读入的时候）检测到 buffer 是空，又要进行输入，与预期要求不符。

五. 实验建议

暂无