

# 《操作系统》实验报告

## LAB2 系统调用

姓名：陈攀岭

学号：171860516

邮箱：171860516@sma i l . n j u . e d u . c n

## 一、实验要求

通过实现一个简单的应用程序，并在其中调用两个自定义实现的系统调用，介绍基于中断实现系统调用的全过程。

内核：基于中断建立完整的系统调用机制。

库：基于系统调用实现库函数 scanf 和 printf。

用户：实现一个调用 scanf 和 printf 的测试程序。

## 二、实验原理

1. Bootloader 从实模式进入保护模式，加载内核至内存，并跳转执行；
2. 内核初始化 IDT (Interrupt Descriptor Table, 中断描述符表)，初始化 GDT，初始化 TSS (Task State Segment, 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. 用户程序调用自定义实现的库函数 scanf 完成格式化输入和 printf 完成格式化输出；
5. scanf 基于中断陷入内核，内核扫描按键状态获取输入完成格式化输入；
6. printf 基于中断陷入内核，由内核完成在视频映射的显存地址中写入内容，完成字符串的打印。

## 三、实验步骤

### 1. 实验任务

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

### 2. 代码实现

- (1) printf 函数：

参数 format 是字符指针，指向 printf 第一个字符串；而参数... 是不定参数，也是 printf 实现的核心。

依次扫描 format 字符串，如果遇到字符不是 '%', 直接存入 buffer；否则，判断该字符的下一字符。若下一字符为 '%', 向 buffer 中存入一个 '%'; 如果为格式化字符，则根据格式化字符的分类，通过参数指针的偏移得到相应的参数类型和数值，调用格式转化的 API 将参数转化为字符串类型存入 buffer 中。format 遍历结束后，通过系统调用将 buffer 输出到标准输出 STD\_OUT 中。

```
int printf(const char *format,...){
    int i=0; // format index
    char buffer[MAX_BUFFER_SIZE];
    int count=0; // buffer index
    int index=0; // parameter index
    void *paraList=(void*)&format; // address of format in stack
    // int state=0; // 0: legal character; 1: '%'; 2: illegal format
    int decimal=0;
    uint32_t hexadecimal=0;
    char *string=0;
    char character=0;
    while(format[i]!=0){
        // TODO: support more format %s %d %x and so on
        if(format[i] != '%') {
            buffer[count]=format[i];
            count++;
            if(count==MAX_BUFFER_SIZE) {
                syscall(SYS_WRITE, STD_OUT, (uint32_t)buffer, (uint32_t)MAX_BUFFER_SIZE, 0, 0);
                count=0;
            }
            i++;
        }
    }
}
```

```

else {
    index++;
    i++;
    switch(format[i]) {
        case 'd':
        {
            decimal = *(int *)(paraList + index * 4);
            count = dec2Str(decimal, buffer, MAX_BUFFER_SIZE, count);
            break;
        }
        case 'x':
        {
            hexadecimal = *(uint32_t *)(paraList + index * 4);
            count = hex2Str(hexadecimal, buffer, MAX_BUFFER_SIZE, count);
            break;
        }
        case 's':
        {
            string = *(char **)(paraList + index * 4);
            count = str2Str(string, buffer, MAX_BUFFER_SIZE, count);
            break;
        }
        case 'c':
        {
            character = *(char *)(paraList + index * 4);
            buffer[count] = character;
            count++;
            if(count==MAX_BUFFER_SIZE) {
                syscall(SYS_WRITE, STD_OUT, (uint32_t)buffer, (uint32_t)MAX_BUFFER_SIZE, 0, 0);
                count=0;
            }
            break;
        }
        case '%':
        {
            buffer[count]=format[i];
            count++;
            if(count==MAX_BUFFER_SIZE) {
                syscall(SYS_WRITE, STD_OUT, (uint32_t)buffer, (uint32_t)MAX_BUFFER_SIZE, 0, 0);
                count=0;
            }
            break;
        }
        default::
        {
            i++;
        }
    }
}
if(count!=0)
    syscall(SYS_WRITE, STD_OUT, (uint32_t)buffer, (uint32_t)count, 0, 0);
return 0;
}

```

## (2) scanf 函数:

scanf 可大致看作 printf 函数的逆过程。

依次扫描参数 format 字符串, 当遇到字符 '%' 时, 进入格式化输入状态, 若其后字符为数字, 则设置输入域宽并继续向后一个字符, 根据格式化字符的类型, 通过参数指针得到相应的参数类型及值, 通过格式转化 API 将字符串转化为相应类型, 并且输入个数加一; 当是其余字符时, 若遇到不是 ' ' 或 '\t' 或 '\n' 其中任一个时, 调用 matchWhiteSpace 函数输入字符串存入 buffer 直到遇到 ' ' 或 '\t' 或 '\n', 与 format 相应位置比较, 若字符不同则说明输入有误, 返回-1。

```
scanf(const char *format,...) {
    // TODO: implement scanf function, return the number of input parameters
    int i=0; // format index
    char buffer[MAX_BUFFER_SIZE];
    buffer[0]='\0';
    int count=0; // buffer index
    int index=0; // parameter index
    void *paraList=(void*)&format; // address of format in stack
    int state=0;
    int *decimal=0;
    int *hexadecimal=0;
    char *string=0;
    char *character=0;
    int num = 0;
    int len;
    while(format[i]!='\0'){
        len = 0;
        if(format[i]=='%') {
            i++;
            if(format[i] > '0' && format[i] <= '9') {
                len = (int)(format[i] - '0');
                i++;
            }
            state = 1;
        }
        if(state == 0) {
            if(format[i] != ' ' && format[i] != '\t' && format[i] != '\n') {
                matchWhiteSpace(buffer,MAX_BUFFER_SIZE,&count);
                if(buffer[count]!=format[i])
                {
                    return -1;
                }
            }
            else {
                i++;
                count++;
            }
        }
        else {
            i++;
        }
    }
}

else {
    index++;
    switch(format[i]) {
        case 'd':
        {
            decimal = (int *)(&(uint32_t*)(paraList + index * 4));
            if(str2Dec(decimal, buffer, MAX_BUFFER_SIZE, &count) == 0)
                num++;
            break;
        }
        case 'x':
        {
            hexadecimal = (int *)(&(uint32_t*)(paraList + index * 4));
            if(str2Hex(hexadecimal, buffer, MAX_BUFFER_SIZE, &count) == 0)
                num++;
            break;
        }
        case 'c':
        {
            character = (char *)(&(uint32_t*)(paraList + index * 4));
            if(str2Str2(character, 2, buffer, MAX_BUFFER_SIZE, &count) == 0)
                num++;
            break;
        }
        default:
        {
            string = (char *)(&(uint32_t*)(paraList + index * 4));
            if(str2Str2(string, len + 1, buffer, MAX_BUFFER_SIZE, &count) == 0)
                num++;
        }
    }
    i++;
    state = 0;
}
return num;
```

### (3) syscallScan 函数:

根据框架代码提示, 仅需要完成键盘输入对 keyBuffer 的存入。

显然, 对与键盘输入, 一般是以 ‘ ’ 或 ‘\t’ 或 ‘\n’ 结束一次输入。于是, 依次作为一次输入的结束条件。查看 keyboard.c 可知, ‘ ’, ‘\t’, ‘\n’ 键码分别为 0x39, 0xf, 0x1c, 当输入键码不为以上三个且为有效输入时, 存入 keyBuffer 并将 bufferTail 指针后移一位。输入结束后, 同样操作将结束符存入 keyBuffer 并后移 bufferTail。

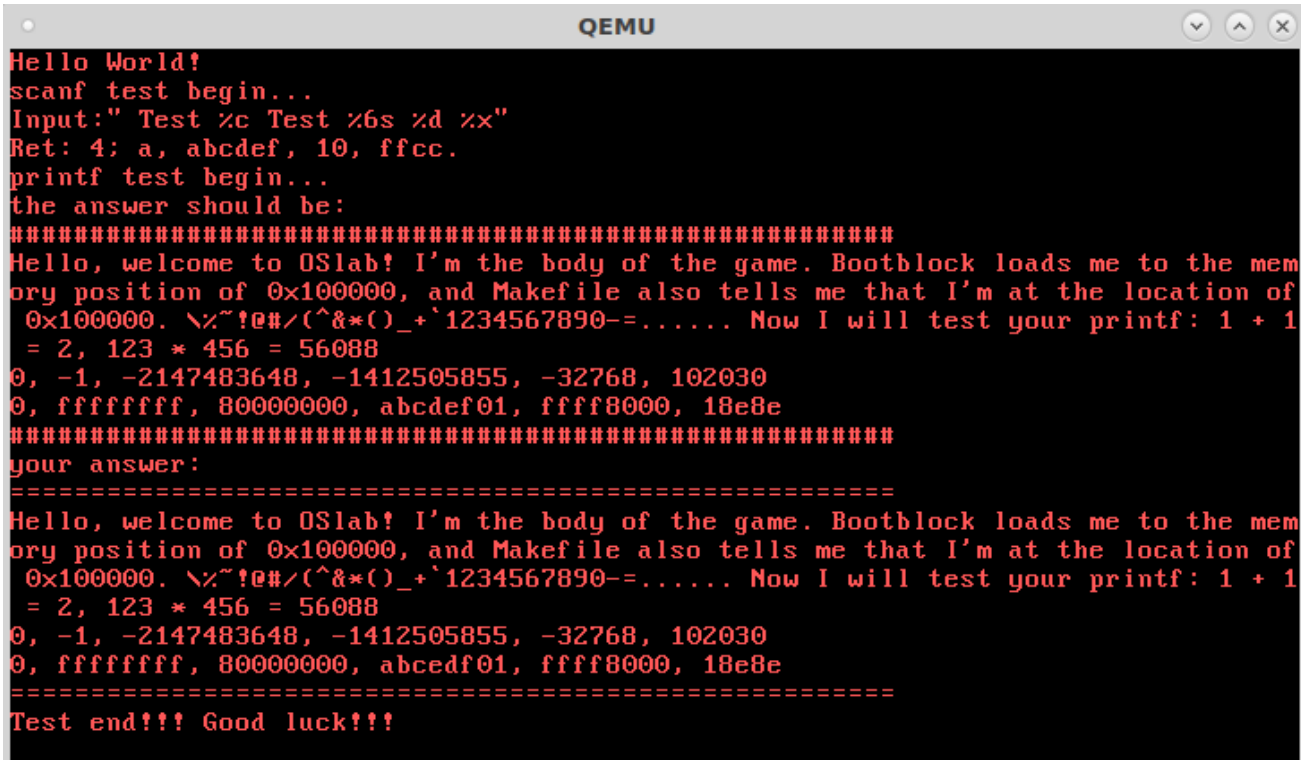
```
void syscallScan(struct TrapFrame *tf){
    // TODO0: get key code by using getKeyCode and save it into keyBuffer
    uint32_t key = getKeyCode();
    while(key != 0x39 && key != 0xf && key != 0x1c) {
        if(key != 0) {
            keyBuffer[bufferTail] = key;
            bufferTail = (bufferTail + 1) % MAX_KEYBUFFER_SIZE;
        }
        key = getKeyCode();
    }
    keyBuffer[bufferTail] = key;
    bufferTail = (bufferTail + 1) % MAX_KEYBUFFER_SIZE;
}
```

## 3. 实验运行

### (1) 给 utils 文件夹下两个 .pl 文件权限

```
cpl@debian:~/OSlab/lab2/lab$ cd utils/
cpl@debian:~/OSlab/lab2/lab/Utils$ chmod 777 genBoot.pl
cpl@debian:~/OSlab/lab2/lab/Utils$ chmod 777 genKernel.pl
```

### (2) make play



The image shows a QEMU window titled "QEMU" with a terminal interface. The terminal output is as follows:

```
Hello World!
scanf test begin...
Input: " Test %c Test %6s %d %x"
Ret: 4: a, abcdef, 10, ffcc.
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. \x~!@#/(^&*()_+'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. \x~!@#/(^&*()_+'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!!!
```