《操作系统实验》实验报告

实验名称：系统调用

姓名：胡育玮

学号：171860574

邮箱：yeevee@qq.com

班级：17 级计算机科学与技术系 2 班

一、实验目的

1、学习 printf()，scanf()的基本原理

2、掌握系统调用的处理流程

二、实验内容

1. 内核:基于中断建立完整的系统调用机制
2. 库:基于系统调用实现库函数 scanf 和 printf
3. 用戶:实现一个调用 scanf 和 printf 的测试程序

三、实验过程

（1）printf()

打开 syscall.c, 查看 printf()的 3 个“API”函数：

首先是 dec2Str()：

```
int dec2Str(int decimal, char *buffer, int size, int count)
{
    //十进制到字符串

    int i=0;
    int temp;
    int number[16];

    if (decimal < 0) //是一个负数
    {
        buffer[count]='-';
        count++;

        if(count==size)
        {   //如果计数值达到规定的大小（这个规定的大小应为buffer的大小）就开始输出，清空buffer
            syscall(SYS_WRITE, STD_OUT, (uint32_t)buffer, (uint32_t)size, 0, 0); //开始输出
            count = 0;  //计数清零，清空buffer
        }
```

```
        //下面几行：把要输出的数的各位数字存好
        temp = decimal / 10;
        number[i]=temp*10-decimal;
        decimal=temp;
        i++;

        while (decimal != 0)
        {
            temp = decimal / 10;
            number[i]=temp*10-decimal;
            decimal=temp;
            i++;
        }
    }
```

```
    else //正数：直接存好各位数字
    {
        temp=decimal/10;
        number[i]=decimal-temp*10;
        decimal=temp;
        i++;

        while(decimal!=0)
        {
            temp=decimal/10;
            number[i]=decimal-temp*10;
            decimal=temp;
            i++;
        }
    }
```

```
    //把存好的各位数字转移到buffer里
    while (i != 0)
    {
        buffer[count] = number[i - 1] + '0';
        count++;

        if (count == size)
        {   //如果计数值达到规定的大小（这个规定的大小应为buffer的大小）就开始输出，清空buffer
            syscall(SYS_WRITE, STD_OUT, (uint32_t)buffer, (uint32_t)size, 0, 0);
            count = 0;
        }

        i--;
    }


    return count;
}
```

由上述图的注释，这个函数首先判断要进行数-字符串转换的 int 型十进制数是正还是负，若是负则先往 buffer 里写入一个负号。然后下面的代码把要转换的数的各位数字写进转换缓存数组 number[]里，最后把 number[]里保存的各个要输出的字符写入 buffer 里。

需要注意的是，每往 buffer 里写入一个字符，就要判断此时 buffer 是否已满，若满则调用 syscall()把 buffer 里的内容输出并清空 buffer。

最后，该函数返回 buffer 的索引 count.

后面的两个 API 函数：hex2Str()和 str2Str()的功能与 dec2Str()如出一辙，只不过是分别把要输出的十六进制数和字符串转换成可以输出的字符串写入 buffer，期间同样有若 buffer 满则清空的操作。

了解了这三个 API 函数的功能后，再结合 printf()已有的代码，完整的 printf()的实现思路便已成形：扫描 printf()的格式化字符串（也即 printf()的第一个参数），若是普通的可以直接输出的字符则直接放入 buffer 中；若遇到%开头的格式化字符，则判断该格式化参数的类型：是%d, %x, %s 还是%c; 在此基础上提取出 printf()的后面的对应位置的参数，调用上面介绍的 3 个 API 函数，将对应的可输出的字符写入 buffer 中。

完成上述目标还需一个关键的细节，那就是不定数量参数函数的实现。Printf()是一个不定数量参数函数，要实现上面一段中讲述的内容，则需要方便地识别和提取出 printf()的各个参数。这部分的实现依赖如下的几个宏：

```
typedef char* va_list;     //将char*别名为va_list，作为指向不定个数参数的指针类型
#define   _INTSIZEOF(n)   ((sizeof(n) + sizeof(int) - 1) & ~(sizeof(int) - 1))

#define   va_start(ap,v)    (ap = (va_list)&v + _INTSIZEOF(v))
//v是可变数量参数函数的首个参数（固定参数）；该宏让指针ap指向可变数量参数函数的固定参数后的第一个参数

#define   va_arg(ap,t)      (*(t*)((ap += _INTSIZEOF(t)) -   _INTSIZEOF(t)))
//使用该宏时，ap指针指向所要提取的参数；该宏将ap指向的参数作为一个t类型的变量提取出来，并让ap指向下一个参数

#define   va_end(ap)        (ap = (va_list)0)
//该宏将ap指针“清零”
```

这些宏的功能如图中注释。

至此，printf()的实现所需的基本元素已就位，下面是具体的实现：

```c
int printf(const char *format,...)
{
    int i = 0; // format index
    char buffer[MAX_BUFFER_SIZE];  //buffer在这里定义:  MAX_BUFFER_SIZE 256
    int count = 0; // buffer index


    int dec;
    uint32_t hex; //这里的API不支持负数的16进制数
    char *str;
    int num;


    va_list ap;
    va_start(ap, format);  //将ap指向固定参数后第一个参数的地址
```

①定义好 buffer；定义好 ap 指针；定义好 dec, hex 等缓存变量。需要注意的是，API 函数不支持负的十六进制数的输出。

```c
while (format[i] != 0) //就是 != '\0'
{
    // TODO: support more format %s %d %x and so on
    if (format[i] != '%')
    {
        buffer[count] = format[i];
        count++;
    }
```

②进入 while 循环，扫描格式化字符串 format 中的每个字符。首先处理的情况是：遇到的字符不是’%’，则此时直接将该字符存入 buffer。

```
else if (format[i] == '%' && format[i + 1] != '\0')
{
    i++; //处理百分号后面的字符
    num = 0;

    if (format[i] >= '0' && format[i] <= '9')
    {
        for (; format[i] >= '0' && format[i] <= '9'; i++)
            num = num * 10 + format[i] - '0';
    }
```

③若遇到的是%，且%不是 format 的最后一个字符，则表明后面有格式化字符，于是 i++，开始处理%后面的内容。若遇到的是数字，则说明是%6s 这种格式化字符。此时上面的 if 语句计算%ks 中的 k 值，存入 num 变量中。

```
switch (format[i])
{
    case 'd':
        dec = va_arg(ap, int);
        count = dec2Str(dec, buffer, MAX_BUFFER_SIZE, count);
        break;


    case 'x':
        hex = va_arg(ap, uint32_t);
        count = hex2Str(hex, buffer, MAX_BUFFER_SIZE, count);
        break;


    case 's':
        str = va_arg(ap, char*);
        int len = 0;
        while (str[len] != '\0')
            len++;

        if (num != 0 && len > num)
            str[num] = '\0';

        count = str2Str(str, buffer, MAX_BUFFER_SIZE, count);

        break;
```

④然后判断%后面的是 d, s, c 还是 x. 若是 d, 则将后面的对

应的参数作为 int 型数提取出来，故有 dec = va_arg(ap, int)这个语句；然后调用 dec2Str()，将 dec 保存的数字存入 buffer 中（期间若 buffer 满，则打印 buffer 中的内容并清空）。

对%x 也是同样的操作。

%s 的情况稍微复杂一点：提取出要打印的字符串的首地址并存入 str 后，计算 str 字符串的长度，若超过前面计算的 num（如果有计算的话），则有 str[num] = '\0'，强行去除字符串的后面的部分只保留前 num 个字符，然后再调用 str2Str()来存入 buffer。这符合%ks 这种格式的要求。

```
case 'c':
    //ch =
    buffer[count] = va_arg(ap, char);
    count++;
    break;


case '%':
    buffer[count] = '%';
    count++;
    break;
```

⑤对于%c 的情况，无需调用 API 函数，直接提取出对应的参数并存入 buffer 即可。对于%%这种情况也要处理：此时直接将字符‘%’存入 buffer。

```
        if (count == MAX_BUFFER_SIZE)
        {
            syscall(SYS_WRITE, STD_OUT, (uint32_t)buffer, (uint32_t)MAX_BUFFER_SIZE, 0, 0);
            count = 0;
        }


        i++;
    }


    if (count != 0)
        syscall(SYS_WRITE, STD_OUT, (uint32_t)buffer, (uint32_t)count, 0, 0);

    va_end(ap);
    return 0;
}
```

⑥这是最后的部分。上面的那个 if 语句在 while 循环结束的一次循环结束时执行，它判断上面提到的几种仅往 buffer 中写入一个字符的情况（while 循环一次仅处理一个字符或一个格式字符, %d 这种）发生后，buffer 是否满，满则清空。

下面的 if 语句则执行在 while 循环结束后。此时整个格式字符串已扫描完毕。若 buffer 此时非空，也即 count 不为 0，则说明有内容未输出，于是输出。

至此 printf()已实现完毕。

（2）scanf()

Scanf()的 4 个 API 函数的功能分别是：

matchWhiteSpace (char *buffer, int size, int *count)：若 buffer 为空，则调用 syscall 将用户输入的字符读入 size 个到 buffer 中。然后判断*count 索引指向的位置是否为空格，制表符或

换行符\n，若是则(*count)++，让*count 指向下一个字符；不断重复直到*count 指向一个不是空格，\n 或\t 的字符为止。

str2Dec(), str2Hex, str2Str2: 去除开头可能存在的空格后将 buffer 中的特定字符转换为相应的内容（十进制数，十六进制数或字符拷贝），存入相应的地址中。

Scanf()的大致实现思路：

```
int scanf(const char *format,...)
{
    // TODO: implement scanf function, return the numb

    char buffer[MAX_BUFFER_SIZE];
    buffer[0] = '\0';
    int i = 0;      //格式字符串索引
    int count = 0; //用户输入索引
    int num = 0;    //处理%6s这种情况， 统计要读入的字符数
    int res = 0;    //该函数的输出

    va_list ap;
    va_start(ap, format);

    int *dec;
    int *hex;
    char *str;
    char *ch;
```

①各个变量的定义和含义如图。Count 为 buffer 索引。dec, hex 等为缓存指针。

```
while (format[i] != '\0')
{
    num = 0;

    if (format[i] == '%' && format[i + 1] != '\0')  //是百分号
    {
        i++;
        res++; //获得的格式数加一

        if (format[i] >= '0' && format[i] <= '9')
        {
            for (; format[i] >= '0' && format[i] <= '9'; i++)
                num = num * 10 + format[i] - '0';
        }
```

②如上图，基本的思路也是扫描格式字符串中的每个字符。若发现是%, 则表明有一个格式字符出现，res++, i++, 然后若%后面紧跟着数字则表明是%ks 这种形式的格式字符，此时需计算 k 的值，存在 num 变量中。

```
switch (format[i])
{
case 'd':
    //res++;
    dec = va_arg(ap, int*);
    if (str2Dec(dec, buffer, MAX_BUFFER_SIZE, &count) == -1)
    {
        printf("Input Error dec!\n");
        return -1;
    }
    break;

case 'x':
    //res++;
    hex = va_arg(ap, int*);
    if (str2Hex(hex, buffer, MAX_BUFFER_SIZE, &count) == -1)
    {
        printf("Input Error hex!\n");
        return -1;
    }
    break;
```

③然后判断%后面的是 d, s, x, 还是 c. 若是 d, 则用 va_arg 提取出 scanf 函数对应的参数（一个 int 指针），存入 dec 指针

中。然后调用 str2Dec(), 将 buffer 中对应的字符转换为 int 类型的数，存到 dec 指针指向的地址中，期间 buffer 的索引 count 会发生改变。

```
if(buffer[*count]==0)
{
    do
    {
        ret=syscall(SYS_READ, STD_IN, (uint32_t)buffer, (uint32_t)size, 0, 0);
    } while(ret == 0 || ret == -1);

    (*count)=0;
}
```

如图，在 str2Dec()中，**若 count 指向的位置是终止符'\0'，则需要重新调用 syscall 来读入字符**。

对于%x 的情况，对应的 str2Hex()的功能也是一样的。

```
    case 's':
        //res++;
        str = va_arg(ap, char*);
        int avail;
        if (num == 0)
            avail = 0x7fffffff;
        else
            avail = num + 1;


        if (str2Str2(str, avail, buffer, MAX_BUFFER_SIZE, &count) == -1)
        {
            printf("Input Error str!\n");
            return -1;
        }
        break;


    case 'c':
        //res++;
        ch = va_arg(ap, char*);
        if (str2Str2(ch, 2, buffer, MAX_BUFFER_SIZE, &count) == -1)
        {
            printf("Input Error ch!\n");
            return -1;
        }
        break;
    }
```

④%s 的情况也类似，此时判断 num 是否为 0，若为 0 则传给 str2Str2 函数的 avail 参数的值为 0x7fffffff, 也即 int 类型

下最大的数，否则为 Num + 1. 之所以是 num + 1 而不是 num 是因为 str2Str2 函数最多读取 avail – 1 个字符到其参数 string 指向的地址中（第 avail 个字符为\0），因此应把 num + 1 赋给 avail, 以实现读取 num 个字符的效果。这也是为什么下面的%c 的情况中 avail = 2.

另一点需要注意的是，若格式符为%6s, 而输入时输入超过 6 个字符，且%6s 后面紧接着%d 这个格式符，则会报错退出，原因是：**处理%6s 时，buffer 的索引 count 只在相应字符串上移动了 6 个长度，buffer[*count]在处理%6s 后仍然指向输入的对应%6s 的字符串而不是对应%d 的字符串。到处理%d 时，str2Dec()这个 API 函数监测到 buffer[*count]指向一个字母值，便返回-1，导致错误。**

```
if(buffer[*count]=='-')
{
    state=1;
    sign=1;
    (*count)++;
}

else if(buffer[*count]>='0' &&
    buffer[*count]<='9')
{
    state=2;
    integer=buffer[*count]-'0';
    (*count)++;
}

else if(buffer[*count]==' ' ||
    buffer[*count]=='\t' ||
    buffer[*count]=='\n')
{
    state=0;
    (*count)++;
}

else
    return -1;
```

如图，在 str2Dec()的一开始，若检测到 buffer[*count]不是负

号，数字或空格等，则会返回-1，我写的程序判断 str2Dec()返回-1 后便提示出错

。

```
    else if (format[i] != ' ')  //不是空格，是普通的字符，%除外
    {   //普通的字符：判断用户的输入是否一一与之匹配
        matchWhiteSpace(buffer, MAX_BUFFER_SIZE, &count); //跳过头部的空格
        if (buffer[count] != format[i])
        {
            //printf("%c=%c, count=%d, i=%d\n", buffer[count], format[i], count, i);
            printf("Unformatted Input Error!\n");
            return -1;
        }

        else
            count++;
    }

    i++;
}

va_end(ap);
return res;
```

⑤最后是处理普通字符的情况。此时调用 matchWhiteSpace()来“去除”可能存在的空格，然后判断 format[]中的字符是否与 buffer[]中的相同，不相同则提示错误。

最下面的 i++表示处理完%后面的 d, c 等之后向后移一格，也表示遇到 format[]中的空格应直接跳过而向后移一格。最后 scanf()返回 res. 至此 scanf()的实现已完成。

（3）syscallScan()

```
uint32_t temp;

temp = getKeyCode();
while (temp != 0x1c)
{
    if (temp != 0)
    {
        keyBuffer[bufferTail] = temp;
        bufferTail = (bufferTail + 1) % MAX_KEYBUFFER_SIZE;
    }

    temp = getKeyCode();
}

keyBuffer[bufferTail] = temp; //最后把回车也弄进去
bufferTail = (bufferTail + 1) % MAX_KEYBUFFER_SIZE;

//printf("sys scan half!\n");
```

在这个函数中要完成的工作很简单，只需要循环调用getKeyCode()来读入用户按下的键的键码，然后把键码写入keyBuffer[]中即可。当检测到按下回车时，循环停止，然后把回车的键码也写入。

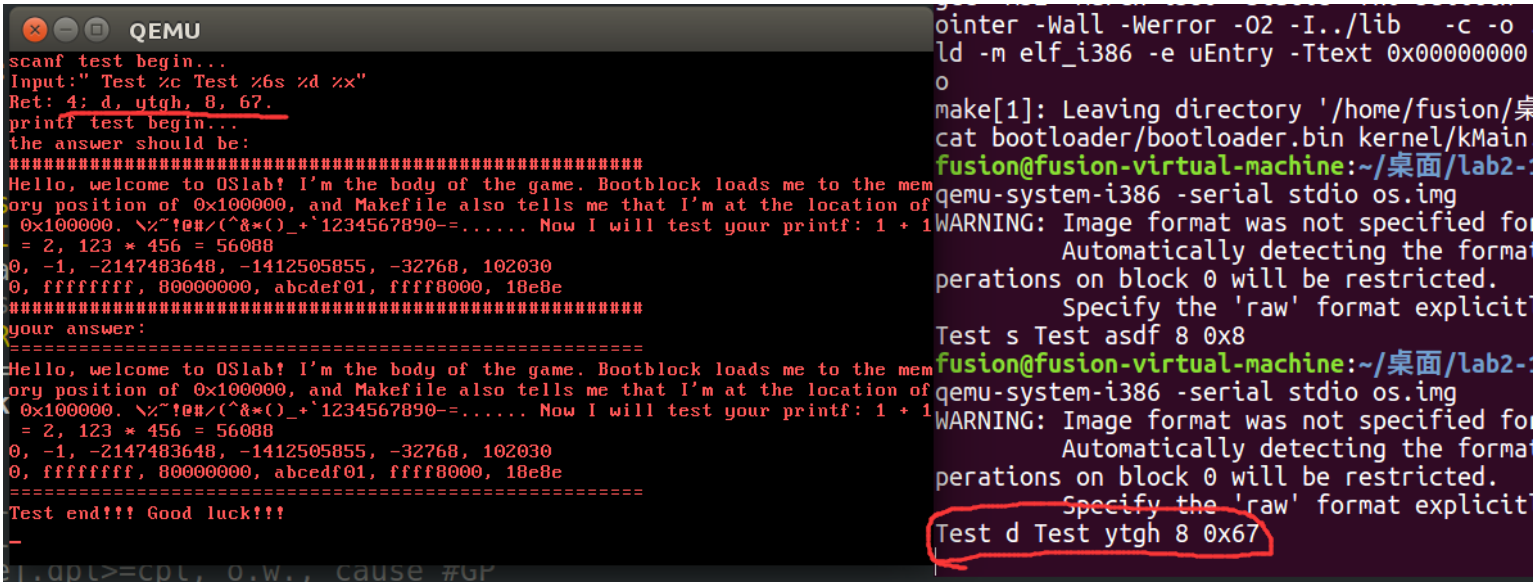
至此实验的全部内容已完成。

（4）效果演示：

Scanf()的格式：

```
while(1)
{
    printf("scanf test begin...\n");
    printf("Input:\" Test %%c Test %%6s %%d %%x\"\n");
    ret = scanf(" Test %c Test %6s %d %x", &cha, str, &dec, &hex);
    //ret = scanf("%c%s%d%x", &cha, str, &dec, &hex);
    printf("Ret: %d; %c, %s, %d, %x.\n", ret, cha, str, dec, hex);
    if (ret == 4)
        break;
}
```

输出效果：注意划红线的部分。


QEMU
scanf test begin...
Input:" Test %c Test %6s %d %x"
Ret: 4; d, ytgh, 8, 67.
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, abcedf01, ffff8000, 18e8e
Test end!!! Good luck!!!
pointer -Wall -Werror -O2 -I../lib -c -o
ld -m elf_i386 -e uEntry -Ttext 0x00000000
o
make[1]: Leaving directory '/home/fusion/桌
cat bootloader/bootloader.bin kernel/kMain
fusion@fusion-virtual-machine:~/桌面/lab2-
qemu-system-i386 -serial stdio os.img
WARNING: Image format was not specified fo
Automatically detecting the forma
perations on block 0 will be restricted.
Specify the 'raw' format explicit
Test s Test asdf 8 0x8
fusion@fusion-virtual-machine:~/桌面/lab2-
qemu-system-i386 -serial stdio os.img
WARNING: Image format was not specified fo
Automatically detecting the forma
perations on block 0 will be restricted.
Specify the 'raw' format explicit
Test d Test ytgh 8 0x67