



# 操作系统原理实验报告

## 实验五：系统中断

数据科学与计算机学院 17大数据与人工智能

17341015 陈鸿峰

### 一、实验目的

- 明白系统调用的意义并设计一组系统调用功能

### 二、实验要求

- 增加一组系统调用，采用功能号实现
- 设计C程序库以调用这组系统调用

### 三、实验环境

具体环境选择原因已在实验一报告中说明。

- Windows 10系统 + Ubuntu 18.04(LTS)子系统
- gcc 7.3.0 + nasm 2.13.02 + GNU ld (Binutils) 2.3.0
- GNU Make 4.1
- Oracle VM VirtualBox 5.2.8
- Bochs 2.6.9
- Sublime Text 3

虚拟机配置：内存4M，无硬盘，1.44M虚拟软盘引导。

### 四、实验方案

#### 1. 系统调用

因为操作系统要提供的服务非常多，服务子程序数量太多，但中断向量有限，因此，实际做法是专门指定一个中断号对应服务处理程序总入口，然后再将所有服务用功能号区分，并作为一个参数从用户中传递过来，服务程序再进行分支，进入相应的功能实现子程序。规定系统调用服务的中断号是21h。

具体实施中，写了int 21h中断，通过ax传递功能号（类似读磁盘的中断方法int 10h）。目前提供的中断向量功能如下

中断向量号	功能号	功能
int 21h	0	返回shell
	1	调用用户子程序1
	2	调用用户子程序2
	3	调用用户子程序3
	4	调用用户子程序4

整合后的测试系统调用例子如下（用户子程序5）

```
start:
    mov ax, 1
    int 21h
    mov ax, 2
    int 21h
    mov ax, 3
    int 21h
    mov ax, 4
    int 21h
    ret
```

## 2. C输入输出标准库函数

本次实验实现了C的标准输入输出函数，即printf、scanf和sscanf。这几个函数其实只是我提供的输入输出系统调用的封装，在C里面对字符串进行处理，在汇编中进行系统调用，基本实现原有C库函数的功能，具体实施见sysio.h头文件。注意这几个函数都是可以传递可变参数的，举例来说

```
printf("%s %d %c",str,i,c);
printf("This is a test: %s",str);
```

这两者在我的程序中都合法。

同时也增加了大量其他库函数，如

- 判断字符/字符串类型：isdigit、isspace、isnum等
- 字符串拷贝：strncpy、strncpy等
- 字符串分割：split
- 其他字符串函数已在实验三中实现

其他库函数请见\include文件夹内的头文件。

## 3. C/Python解释器

这是本次实验最大的亮点，我编写了一个类C/Python的解释器(interpreter)，并提供了交互界面。在该交互界面中可以进行前面说的输入输出系统调用，并可以进行简单的四则运算，

同时声明并存储变量。具体实施见interpreter.h，结果见下一节。

测试输入输出函数

## 五、实验结果

执行用户程序5的结果见图1，即连续调用系统调用，注意用户程序都是可再入的——调用完系统中断后依然能回到断点继续执行。

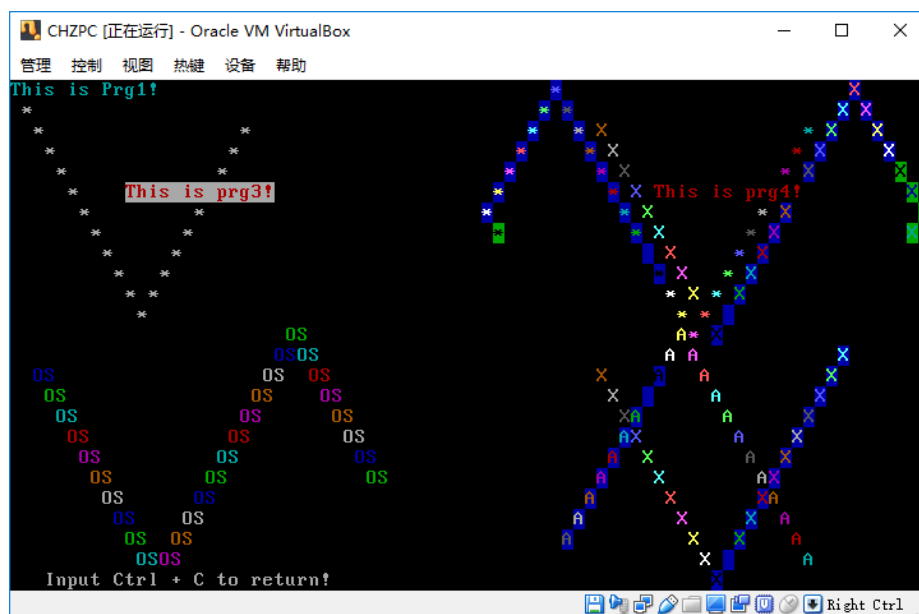
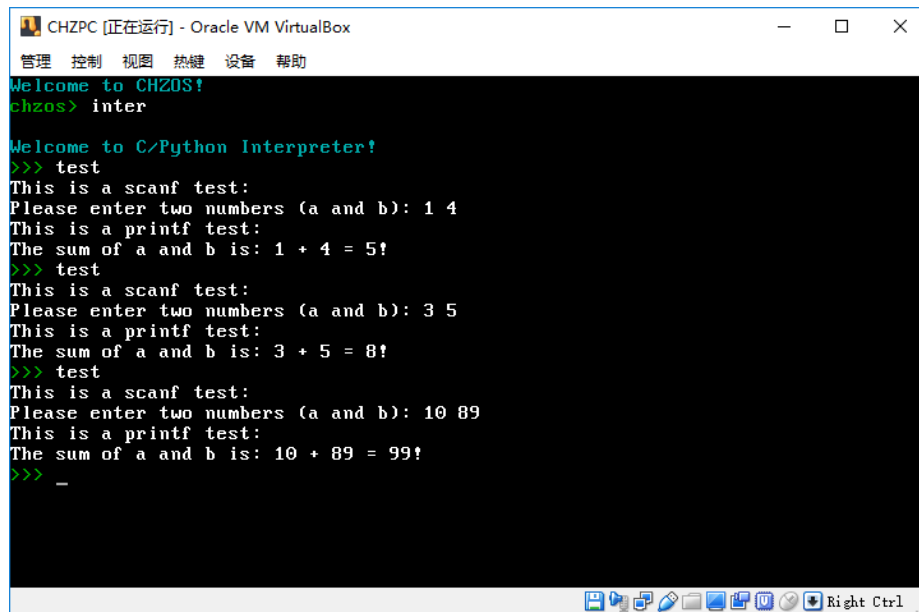


图 1: 系统调用

scanf和printf的测试函数见下面程序，该函数在interpreter.h头文件中。

```
void io_test()
{
    char* str = "This is a scanf test:\n";
    char* format = "%s";
    printf(format, str);
    str = "Please enter two numbers (a and b): ";
    printf(format, str);
    int a, b;
    format = "%d %d";
    scanf(format, &a, &b);
    str = "This is a printf test: ";
    int c = a + b;
    char d = '!';
    format = "%s\nThe sum of a and b is: %d + %d = %d%c\n";
    printf(format, str, a, b, c, d);
}
```

实验结果如图2所示，注意所有的scanf样例都是现场输入的，而不是预先写死在程序中的。



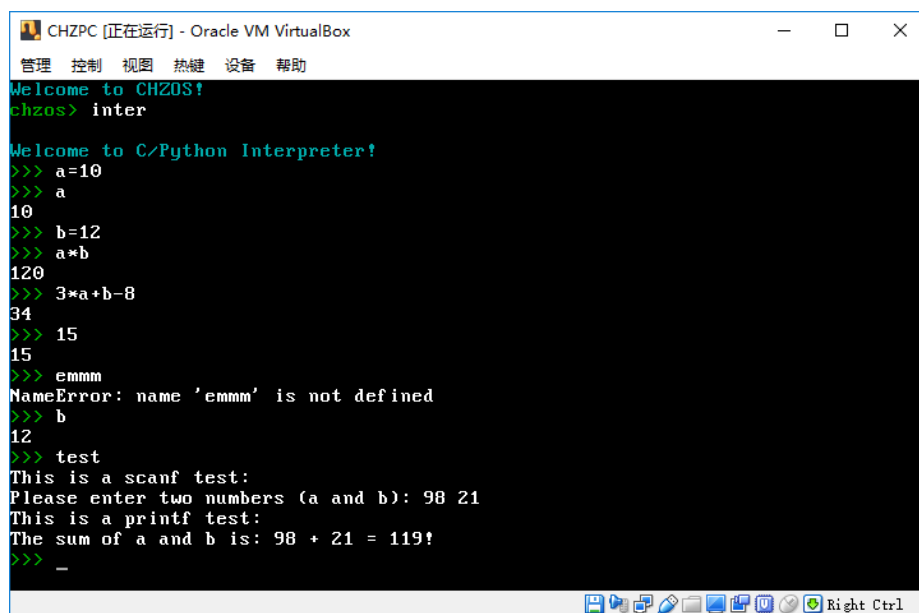
```
CHZPC [正在运行] - Oracle VM VirtualBox
管理 控制 视图 热键 设备 帮助
Welcome to CHZOS!
chzos> inter

Welcome to C/Python Interpreter!
>>> test
This is a scanf test:
Please enter two numbers (a and b): 1 4
This is a printf test:
The sum of a and b is: 1 + 4 = 5!
>>> test
This is a scanf test:
Please enter two numbers (a and b): 3 5
This is a printf test:
The sum of a and b is: 3 + 5 = 8!
>>> test
This is a scanf test:
Please enter two numbers (a and b): 10 89
This is a printf test:
The sum of a and b is: 10 + 89 = 99!
>>> _
chzos>
```

图 2: C输入输出函数测试

进入解释器的指令为inter，解释器的实验结果如图3所示。注意

- 在数字输入时会直接进行计算
- 目前支持简单的四则运算指令
- 字母输入默认为指令，若无该条指令则报错
- 可以声明变量存储并运算
- 基本设置与Python命令行一致



```
CHZPC [正在运行] - Oracle VM VirtualBox
管理 控制 视图 热键 设备 帮助
Welcome to CHZOS!
chzos> inter

Welcome to C/Python Interpreter!
>>> a=10
>>> a
10
>>> b=12
>>> a*b
120
>>> 3*a+b-8
34
>>> 15
15
>>> emmm
NameError: name 'emmm' is not defined
>>> b
12
>>> test
This is a scanf test:
Please enter two numbers (a and b): 98 21
This is a printf test:
The sum of a and b is: 98 + 21 = 119!
>>> _
```

图 3: C/Python解释器

## 六、实验总结

本次实验并不难，只需将原来的中断调用全部整合入一个系统调用即可，并通过寄存器传递功能号，实现不同的功能。

以前写高级语言程序调试往往通过在自己期望的断点处进行debug输出，但现在当要自己实现输入输出函数时，便没有一个参照物告诉你哪里错了，所以调试难度非常大。因此，我采用的是核心分离的方法，即用普通的C程序先在正常的电脑上调试输出通过后，再将所有的printf函数换回我提供的库函数putchar。这样现在自己电脑上将字符串输入输出函数调试成功后，再放入操作系统，就不会遇到太多问题。不过要注意由于我现在的内核已经太大了，占了23个扇区，因此加载的用户程序已经不能放在0A100h，需要另找位置，否则原来的内核会被覆盖，导致无法再入。

为了给自己提升难度，于是我自行设计了一个C的解释器，实际上是类似Python的功能与界面。由于C是静态语言，所以传统的C程序一定要通过编译，将高级语言程序转换为机器码才可以在机器上运行。而Python这种动态语言则采用解释器的模式，逐行读取逐行运行。近年来科学计算比较火的Julia这门语言，现在已经出到1.0版，可以对C程序进行解释执行。我想既然我要实现操作系统，也要实现命令行界面，那不如也亲自实施一下，看看难度有多大。实际效果就如实验中所呈现的一样，目前还只能做一些简单的四则运算，之后会添加更多功能。

操作系统这门课真是越来越有趣了，从上层到底层全部打通了，甚至开始自己写编译器/解释器，真不愧是一门系统的课程。

## 七、参考资料

1. 李忠, 王晓波, 余洁, 《x86汇编语言-从实模式到保护模式》, 电子工业出版社, 2013

## 附录 A. 程序清单

由于程序太多, 请直接见压缩文件。

## 附录 B. 附件文件说明

序号	文件	描述
1	bootloader.asm	主引导程序
2	os.asm	内核汇编部分
3	kernel.c	内核C部分
4	Makefile	编译指令文件
5	link.ld	链接文件
6	bochsrc.bxrc	bochs调试文件
7	mydisk.img	核心虚拟软盘
8~13	prgX.asm	用户程序
14~19	/include	内核头文件