**实验5：软中断实现系统服务**

姓名：姜洋帆 学号：17341068

院系：数据科学与计算机学院 专业：17级计算机科学（大数据）

【**实验题目**】软中断实现系统服务

**【实验目的】**

1. 了解系统调用。

2. 封装C库，完成系统调用的实现。

【**实验要求】**

1. 解释系统调用的意义及规划一组系统调用功能。
2. 完成一个系统调用实现，使用同一个中断号，通过不同的功能号实现不同功能。

【**实验方案】**

1. 硬件及虚拟机配置：Lenovo PC ；Oracle Virtual Box
2. 软件工具及作用：

Notepad++ ：写代码

Nasm：编译引导程序的asm文件到bin文件

Tasm：编译16位的内核代码

Tcc：编译16位C语言，实现C语言与x86联合编译

Sublime ：查看、编辑二进制文件，将编译完成的程序机器码写入软盘

C程序：

1. 自己编写的一个C语言程序，用来将编译生成的二进制文件填充为1.44M大小，并将后缀改为.img，生成1.44M的软盘
2. 内核程序，与之前实验内容大体相似，将一些简单的功能（输入、输出、清屏、重启（reboot）等功能）改为系统中断调用，通过给不同的功能号调用不同的功能。

86汇编：

1. Myos5.asm实现引导程序，偏移量位7c00h，将内核加载至内存并跳转进入内核程序
2. Kernel.asm、ckernal.c为内核程序的主要部分，自定义了一些需要的中断，完成内核管理机器的一些基本功能
3. Kliba.asm实现内核程序以及一些常用的函数，一些比较基本的操作改为使用21号系统中断实现。
4. Syscall.asm为这个实验新增的部分，系统调用功能在卸载这个文件内。
5. 程序功能

Myos1作为引导程序，设置好内核的偏移和段地址，加载内核程序，然后跳入内核程序，进行下一步操作。主要功能与实验4类似，通过命令行输入执行各种程序，增加了一个显示系统调用手册的指令，其他功能与之前实验版本的功能一致（实现方式不一样，把部分函数、功能的实现改为系统调用）

本次实验主要实现21号系统中断，具体的功能有：

1. 显示系统中断本身的信息，相当于一个使用手册，标出了每个中断功能号的作用。
2. 实现重启功能（冷启动），在虚拟机上的效果就是引导程序重新加载内核，所有数据都不会保留。
3. 读入字符函数，在之前版本为getChar函数，这里用系统中断来实现。
4. 将之前实验版本的清屏函数(cls)改为系统中断调用实现，功能号为5。
5. printChar函数改为系统中断，功能为打印字符（需要从C代码中传入参数）。
6. backspace改为系统中断，功能为实现命令行输入的退格功能。
7. 程序设计

程序结构与之前实验一致，修改的部分只有汇编库代码和少量C语言代码。

系统中断的主要功能在syscall.asm中完成，中断的实现与之前的类似，前后需要入栈出栈，需要注意的是，不管调用哪个中断功能，都需要加上in al,60h，将键盘缓存读走，不然输入功能会出现问题。

根据不同的功能号实现不同的功能，思路是在调用中断时，把功能号传给al寄存器，在21h中断内想办法获取到这个al的值，再用cmp指令比较，调用不同的功能模块即可。

其中前四个功能由于没有涉及到传参，不需要操作栈指针sp，直接把功能模块代码复制进中断模块，进行简单的封装即可。而后面两个功能由于需要从C模块传参数，涉及到对栈的操作（要找到传进来的参数的位置），而实现中断本身需要大量的push、pop操作，而且还有一些隐式的压栈出栈，需要修改寻找参数的方式，花了一些时间去处理这个问题。

五、主要代码

**引导程序没有修改**

**内核程序（汇编部分）**

添加了一个系统调用中断21h，修改了部分原来的中断号。基本功能与之前实验一致。

**21h号系统中断：**

大体的结构和上个实验实现的中断一致，压栈出栈、一些端口的读写，注意开中断等。将不同的功能封装在不同的模块中。在外部调用21h中断时，需要将功能号写入al寄存器中，系统中断程序根据al的值，选择执行对应的模块。由于在判度al之前，就对ax寄存及有修改了，所以这里先用bx寄存器保存ax的值（在刚进入中断程序时执行），然后在需要的地方把值重新赋给ax，使用cmp指令判度即可。

具体代码如下（不包括各模块的具体实现代码，具体代码可见附件syscall.asm）

system\_call:

;中断调用的参数

sti

push bx ;将参数ax传进来，并弹出到bx寄存器中

mov bx,ax

push dx

push ax

push es

push cx

push bp

push ds

mov ax,cs

mov ds,ax

mov es,ax

mov ax,bx ; 将传进来的ax参数赋值给ax

cmp al,0

jz system\_msg

cmp al,1

jz reboot

cmp al,2

jz sgetChar

cmp al,3

jz run\_system\_cls

cmp al,4

jz sys\_print

cmp al,5

jz sys\_back

system\_msg:

call printMsg

jmp system\_call\_end

reboot:

call my\_reboot

jmp system\_call\_end

sgetChar:

call igetChar

jmp system\_call\_end

run\_system\_cls:

call system\_call\_cls

jmp system\_call\_end

sys\_print:

call print

jmp system\_call\_end

sys\_back:

call back\_space

jmp system\_call\_end

system\_call\_end:

in al,60h

mov al,20h ; AL = EOI

out 20h,al ; 发送EOI到主8529A

out 0A0h,al ; 发送EOI到从8529A

pop ds

pop bp

pop cx

pop es

pop ax

pop dx

pop bx

iret

**系统中断功能：**

**（以打getChar功能为例）**

0-3号功能的实现方式较为简单，思路也是一致的，这里就以1号功能为例。

这部分代码写在syscall文件内，系统中断程序根据al的值，可以直接调用不同的模块。需要在模块之前加开中断，防止不能调用系统原本的中断。不需要额外保护寄存器（除非这个模块本身需要使用），因为ret后没有读取寄存器的操作了。

igetChar:

sti

push ax

mov ah,0

int 16h ;0号功能调用从键盘读入一个字符放入al中

mov byte ptr [\_chBuf],al

pop ax

ret

**4、5号功能实现（涉及到外部传参）：**

4、5号功能实现的是原本的printchar和backspace函数，这两个函数在原来设计的时候都是需要从C语言中传递参数的，在汇编中寻找参数，只需要[sp+4]这个位置（栈是向下增长的，调用函数时，段间调用先入栈段基址Cs，再入栈IP，所以需要加4），一开始把中断调用也当作函数来考虑，根据这个思路，将这部分函数写到21中断，额外增加了一次调用和12次push操作，理论上应该时增长了24字节，可是按照这个数量改写后，发现不能正常输出字符。

这里卡了不少时间，发现字符应该是对应着输出了，只是字符地址读错了。最后在24附近试了一些其他数据，发现正确的数据应该是26。后来上网查找资料后，发现中断调用压栈与函数调用(call)压栈数据不一样，INT指令要先将FLAGS（状态标志位）入栈，再将CS入栈，再将IP入栈，也就是说比call指令多压了一个寄存器…所以应该再增加2。

代码如下，其中bp+26这里就是出问题的地方：

print:

sti

push bp

mov bp,sp

mov al,[bp+26] ;指向栈顶元素，即字符

mov bl,2

mov ah,0eh ;显示字符光标前移

int 10h

mov sp,bp

pop bp

ret

backspace与这个类似，注意栈指针偏移量即可。

**另外实现了一个重启功能，也放在21h中断中：**

由于个人走的是实模式，只要使用一个简单的调用int19h就可以了，这个相当于冷启动，把引导重新装入0x7c00h中，实模式可以直接使用特权指令，所以不容易出问题。

另外还有一种实现方式是jmp ffff:0；相当于热启动，保护模式可能需要使用这种方式进行。

代码如下：

my\_reboot:

sti

int 19h

ret

**内核程序（C语言部分）**

这里基本上没有什么改动，增加了打印系统中断相关信息的函数，额外增加了一些代码补全。

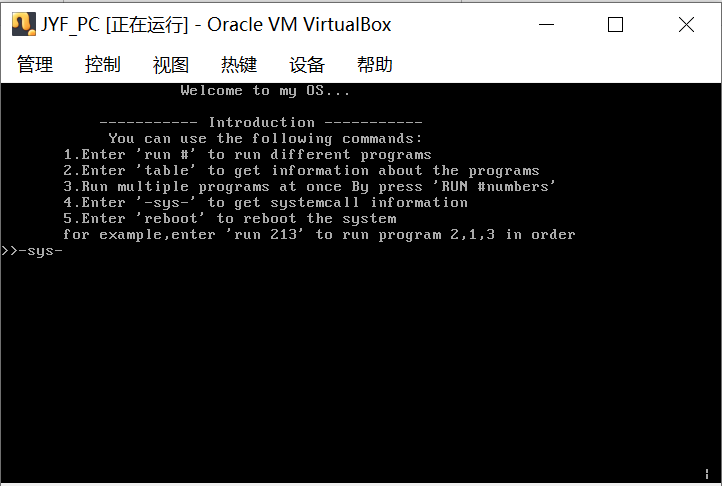
**【实验过程】**

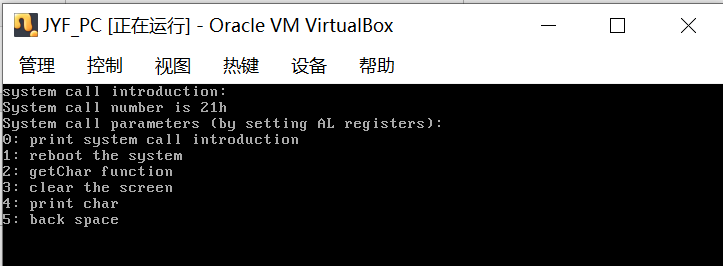
**功能展示**

**保留原本的所有功能（无敌风火轮，程序运行时的键盘响应中断）**

1. **输入-sys-，内核调用int 21的0号功能，打印系统中断的相关信息。**

**printChar相关功能正常实现，getchar功能和backspace功能也正常。**

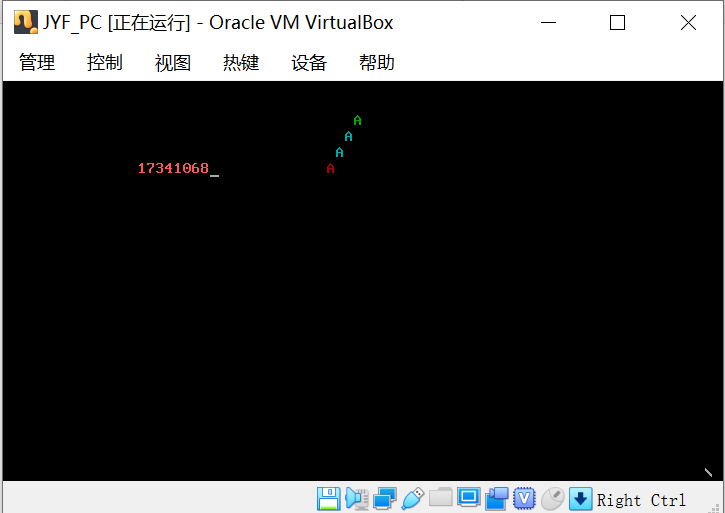




**程序结束后返回主界面，cls功能也正常实现。**

1. **测试原功能 以run 1指令为例：**

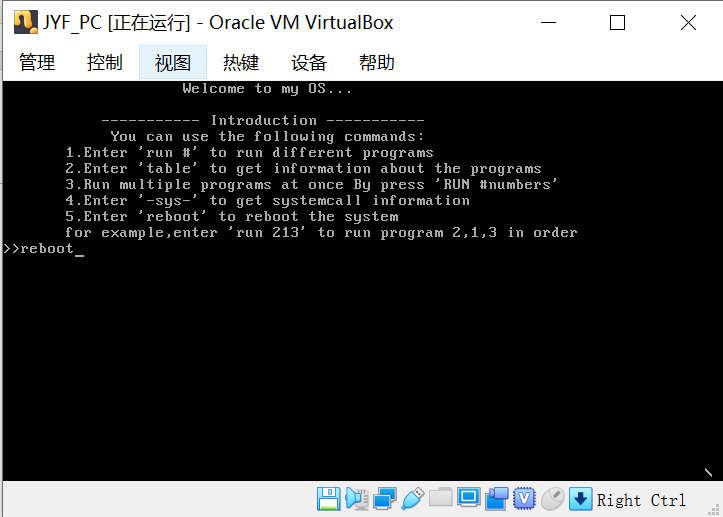
**风火轮、键盘响应中断正常。**





1. **测试reboot指令**

**输入命令并敲回车执行后，效果如下，虚拟机重新将引导程序装载进内存（重新‘自举’），实现冷启动，启动后进入主菜单，各项功能正常。**





程序1、2、3、4与实验4的用户程序功能一致。具体效果可使用虚拟机加载附件中的myos5.img文件（vbox下测试通过），输入不同指令查看相关效果，实现效果与实验4一致。

**【实验总结】**

这次实验相比前几次简单了不少，只需要实现一个系统中断调用，实现一个根据不功能号调用不同功能的需求即可。

实现这个需求的意义在于，中断向量是有限的，而通过这种方式来扩展，可以将大量基础的功能封装成系统调用，方便给上层开发人员调用。

本次实验的核心就是对系统中断功能号的保护，由于设计较多的push操作，寄存器的值容易跟丢，需要一些细心和耐心。另外在将printChar和backspace函数封装进系统中断时遇到了一些问题，主要是因为涉及到了C语言传参数，需要跟踪这个参数在栈中的位置，而中断调用内有不少push操作，同时没有注意到int中断隐式的压栈操作和call指令不同（增加了flag寄存器的压栈），导致这里出现了一些问题。说明自己的汇编基础还是不好，很多基本功能实现的细节不清楚，还是需要学习一个。