

实验三：开发独立内核的操作系统

18340225 钟婕

实验目的

- 1、加深理解操作系统内核概念
- 2、了解操作系统开发方法
- 3、掌握汇编语言与高级语言混合编程的方法
- 4、掌握独立内核的设计与加载方法
- 5、加强磁盘空间管理工作

实验要求

- 1、知道独立内核设计的需求
- 2、掌握一种x86汇编语言与一种C高级语言混合编程的规定和要求
- 3、设计一个程序，以汇编程序为主入口模块，调用一个C语言编写的函数处理汇编模块定义的数据，然后再由汇编模块完成屏幕输出数据，将程序生成COM格式程序，在DOS或虚拟环境运行。
- 4、汇编语言与高级语言混合编程的方法，重写和扩展实验二的的监控程序，从引导程序分离独立，生成一个COM格式程序的独立内核。
- 5、再设计新的引导程序，实现独立内核的加载引导，确保内核功能不比实验二的监控程序弱，展示原有功能或加强功能可以工作。
- 6、编写实验报告，描述实验工作的过程和必要的细节，如截屏或录屏，以证实实验工作的真实性

实验内容

- 1、寻找或认识一套匹配的汇编与c编译器组合。利用c编译器，将一个样板C程序进行编译，获得符号列表文档，分析全局变量、局部变量、变量初始化、函数调用、参数传递情况，确定一种匹配的汇编语言工具，在实验报告中描述这些工作。
- 2、写一个汇编和c程序混合编程实例，展示你所用的这套组合环境的使用。汇编模块中定义一个字符串，调用C语言的函数，统计其中某个字符出现的次数（函数返回），汇编模块显示统计结果。执行程序可以在DOS中运行。
- 3、重写实验二程序，实验二的的监控程序从引导程序分离独立，生成一个COM格式程序的独立内核，在1.44MB软盘映像中，保存到特定的几个扇区。利用汇编和c程序混合编程监控程序命令保留原有程序功能，如可以按操作选择，执行一个或几个用户程序、加载用户程序和返回监控程序；执行完一个用户程序后，可以执行下一个。
- 4、利用汇编和c程序混合编程的优势，多用c语言扩展监控程序命令处理能力。
- 5、拓展自己的软件项目管理目录，管理实验项目相关文档

实验环境与工具

1、实验环境：

- 1) 实验运行环境：Windows10
- 2) 虚拟机软件：VMware Workstation、DosBox

2、实验工具

- 1) 汇编编译器：NASM、TASM
- 2) C语言编译器：TCC
- 3) 文本编辑器：Visual Studio 2017
- 4) 软盘操作工具：WinHex

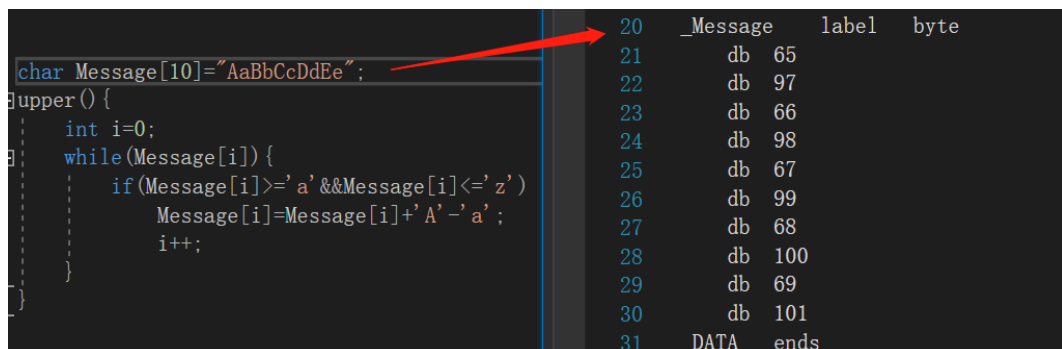
实验方案与过程

• 符号列表文档分析

将老师PPT中upper.c程序用tcc -S upper.c命令得到UPPER.asm文件，分析文件中的分析全局变量、局部变量、变量初始化、函数调用、参数传递情况等。

根据上述操作得到UPPER.asm文件打开后依次分析：

1. C模块定义变量Message：

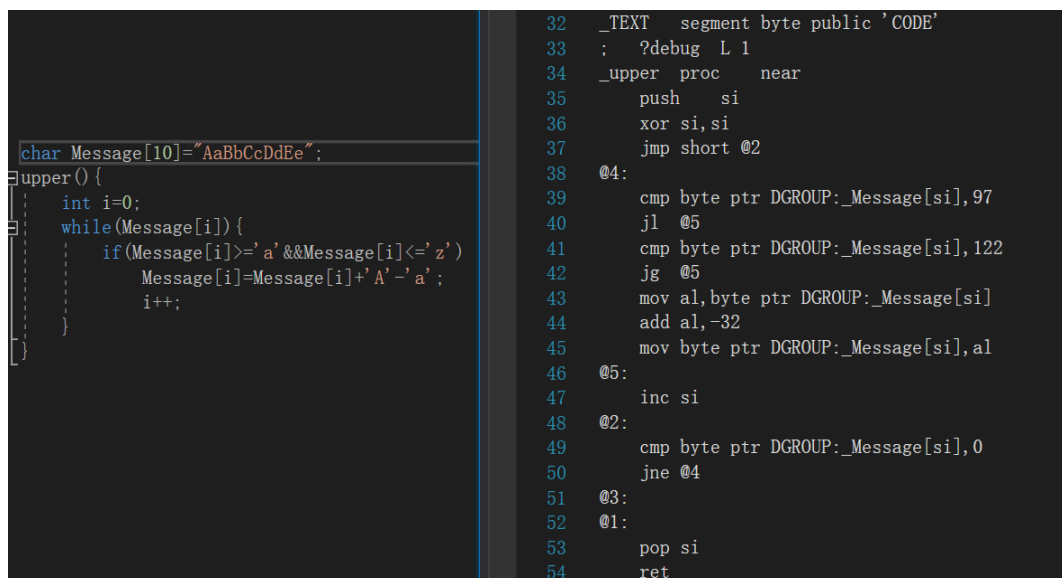


```
char Message[10]="AaBbCcDdEe";
upper() {
    int i=0;
    while(Message[i]) {
        if(Message[i]>='a' && Message[i]<='z')
            Message[i]=Message[i]+'A'-'a';
        i++;
    }
}
```

```
20  _Message  label  byte
21      db  65
22      db  97
23      db  66
24      db  98
25      db  67
26      db  99
27      db  68
28      db  100
29      db  69
30      db  101
31  _DATA  ends
```

汇编模块调用C模块变量会在变量名前加上下划线。字符串Message在汇编中按一个个字节依次定义。

2. 代码段分析：



```
char Message[10]="AaBbCcDdEe";
upper() {
    int i=0;
    while(Message[i]) {
        if(Message[i]>='a' && Message[i]<='z')
            Message[i]=Message[i]+'A'-'a';
        i++;
    }
}
```

```
32  _TEXT  segment byte public 'CODE'
33  ; ?debug  L 1
34  _upper proc  near
35      push    si
36      xor     si,si
37      jmp     short @2
38  @4:
39      cmp     byte ptr DGROUP:_Message[si],97
40      jnl     @5
41      cmp     byte ptr DGROUP:_Message[si],122
42      jg      @5
43      mov     al,byte ptr DGROUP:_Message[si]
44      add     al,-32
45      mov     byte ptr DGROUP:_Message[si],al
46  @5:
47      inc     si
48  @2:
49      cmp     byte ptr DGROUP:_Message[si],0
50      jne     @4
51  @3:
52  @1:
53      pop     si
54      ret
```

```

;si寄存器位变址寄存器，用于字符串数组的下标访问数组。
push si;保护si寄存器
xor si,si;si寄存器置0，数组下标从0开始
;@2段为while循环的判断条件：是否到字符串末尾
;@4段为先判断当前si寄存器访问的数组元素是否为小写，如果不为继续访问下一个数组元素
;如果为小写将当前字符值减去32，变成大写，保存改变以后的元素
;当字符串访问完成后，回复si寄存器的值，返回程序

```

• 热身实验

写一个汇编程序和c程序混合编程实例。汇编模块中定义一个字符串，调用C语言的函数，统计其中某个字符出现的次数（函数返回），汇编模块显示统计结果。

根据实验要求选择TCC+TASM的组合，在汇编中定义字符串str，此模块定义一个函数count，该函数统计'o'字符出现字数，并将次数num返回给汇编模块，由汇编模块输出。count函数的参数为字符指针，在汇编模块调用count函数时，将汇编模块定义的字符串地址压栈，调用完成之后出栈。再利用BIOS显示字符串的功能调用输出个数即可。

代码如下：

汇编模块关键代码：

```

mov ax,offset _str;字符串地址作为参数
push ax;入栈
call near ptr _count ;调用C程序的count函数
pop cx;出栈
datadef:
_str db "HappytoGooooo",0;定义字符串
Info db "The number of o is "

```

C模块部分代码：

```

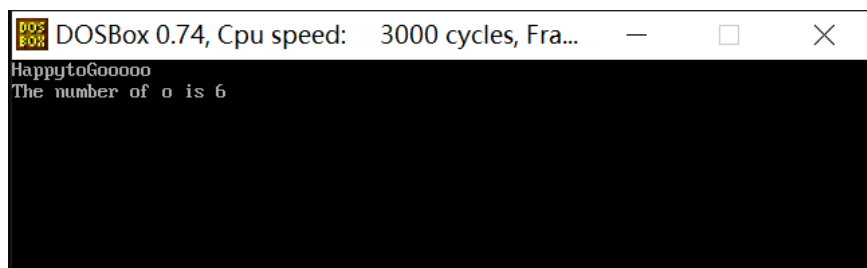
char num='0';//作为返回值，返回给汇编模块

count(char *str)
{
    int index = 0;
    while (str[index] != 0)
    {
        if (str[index] == 'o')//统计'o'字符出现次数
            num++;
        index++;
    }
}

```

实验结果：

在DosBox下运行：



• 实验思路与原理

总体思路

根据实验要求与内容可知，本次实验其实就是在实验二的基础上，将实验二的监控程序从引导扇区程序中分离，利用汇编与C语言的混合编程实现实验二中监控程序的加载用户程序的功能。简而言之，就是利用汇编与C的混合编程实现一个监控程序作为本次实验的操作系统内核。而用户程序则沿用实验二中改进的四个用户程序。当然，由于C语言的加入，我们可以扩展一些操作系统内核的功能，同时使得界面更美观，提升系统的交互性。

在本次实验中，一个重难点就是汇编语言与C语言的混合编程。在本次实验我选择TCC+TASM的编译器组合。下面基于这种组合，简要描述汇编与C混合编程的关系与特点。

基于本人比较粗浅的认识以及对于来源互联网资料的简单理解，汇编语言与C语言的关系可以描述为，汇编语言主要用于实现系统一些底层的功能，比如实现I/O的交互，在实验二中，我们了解了BIOS输入一个字符的功能调用等等。往往用汇编语言实现一些基于硬件的操作，比如更改寄存器的值，加载扇区、输入输出一个字符等等。而对于高级语言，我将之理解为在汇编基础上的功能扩展工具。实际上，真正的操作系统功能纷繁复杂，仅仅依靠有限的汇编指令，难以完成，就算是完成，也需要花费难以估计的工作量。因此，我们需要利用C语言，在基于汇编的一些底层功能上，实现更复杂的功能。

举个例子，汇编语言的BIOS功能调用可以实现输入、输出一个字符的功能，那么我们就可以利用C语言调用汇编的输入、输出单个字符的过程，实现字符串的输入输出。本次实验，便基于这样的思想，利用汇编语言实现一些基本的功能，再通过C语言调用汇编过程实现功能的扩展。

实验原理

由于在前两个实验均是用汇编语言实现的，因此对于汇编的一些语法操作有了大致的了解，对于BIOS的功能调用也有了大致的认识，在此原理部分就不一一赘述。

下面主要叙述汇编与C语言混合编程的语法：

在汇编与C语言的混合编程过程主要涉及到过程调用、参数传递以及变量引用三个问题。

1. 变量引用：

根据将C语言程序编译可知，C编译后的变量名、函数名的前面都加了下划线，比如在C中的变量A，在编译后为_A。因此，在汇编语言中的变量、过程一般都需要加上下划线；而C模块中则无须加上下划线，C模块引用汇编的变量，调用汇编的过程，需要去掉相应的下划线。

2. 汇编模块调用C模块的函数：

汇编模块调用C模块的函数时，需要先用extern声明一个外部函数。当需要参数传递时，遵循右参先入栈的原则，将参数依次以push word ptr[]的形式入栈，然后利用call指令调用相应的函数，函数返回后，在汇编模块需要逆序将各个参数依次出栈。特别注意，参数入栈、出栈都必须以字为单位。

3. C模块调用汇编模块的过程：

C模块需要调用汇编模块的过程时，需要先用extern声明一个外部函数，注意：使用extern时需要注明函数的返回值、函数名、参数。而在汇编模块中，TASM要求也需要在过程名前加上public，同时以过程名+proc+过程内容+过程名+endp的形式组织汇编过程的编写。C模块调用汇编过程时的参数传递，C语言会自动将参数压栈，因此汇编模块从栈中取得参数即可。但是，由于使用call指令时，call会将IP值压栈，因此汇编模块需要从[bp+2]处开始取参数，当然，如果为了保护一些寄存器的值，则需要从bp+2的基础上继续后移取参。汇编模块只需要从栈中取得C传来的参数即可，不需要考虑出栈的问题。

4. 汇编文件的包含问题：

内核程序设计以汇编为入口，但还涉及到一些需要汇编语言编写的汇编过程，如果放在一个汇编ASM文件，文件冗长，不便于查看且易出错，因此，根据老师提供的样板程序了解到可以在一个汇编文件末尾include 另一个汇编文件。

实验程序模块组织

将本次实验主要分成三部分：引导扇区程序、内核程序以及用户程序。其中内核程序用TCC和Turbo C混合编程完成。下面列出各个模块的简要功能以及文件组成：

1. 引导扇区程序 boot.bin

BIN格式文件，NASM汇编生成，用以加载操作系统内核，并将控制权交给内核。

2. 内核程序test.com

COM格式文件，包含：

rukou.asm:内核程序的入口，包含内核汇编程序basic.asm。

kernal.c:C语言实现用户交互、操作命令。

basic.asm:汇编语言实现基本输入输出字符、读扇区、清屏基本过程。

3. UP系列文件

包含四个实验二中的用户程序UP1~4，一个记录用户程序相关信息的表格table，还有一个用于刚刚加载操作系统内核时的界面显示Loading。

• 引导扇区程序的设计

本实验中，引导扇区程序的功能很单纯，即为读入操作系统内核程序所在的扇区至内存相应位置，并跳转到相应内存位置执行，将CPU控制权交给操作系统。

引导扇区程序的设计，主要用到BIOS的13H号读扇区功能，由于操作系统内核程序比较庞大，因此预先分配6个扇区（扇区号2~7）给操作系统内核程序存放，下面展示读入操作系统内核程序对应扇区的代码部分。

```
offsetofOS equ 8100H
;调用int 13h读扇区加载操作系统内核至内存相应位置
LoadOS:
    mov ax,cs
    mov es,ax ;置es与cs相等，为段地址
    mov bx,offsetofOS ;数据常量，代表偏移地址，段地址:偏移地址为内存访问地址
    mov ah,02h ;功能号02h读扇区
    mov al,6 ;读入扇区数,操作系统内核的程序数据量大，不止占用一个扇区，因此就分配六个扇区给OS内核
    mov dl,0 ;驱动器号，软盘为0，U盘和硬盘位80h
    mov dh,0 ;磁头号为0
    mov ch,0 ;柱面号为0
    mov cl,2 ;起始扇区号，引导程序存放在首扇区
    int 13h ;中断号
    jmp 800h:100h ;操作系统内核已经加载到内存相应位置，要跳转到内存相应位置，将控制权交给内核

    times 510-($-$$) db 0;将剩余字符填0
    dw 0xAA55;可引导标志
```

特别说明地是，由于内核程序是COM格式程序，类似于实验二的用户程序，规定加载程序到内存的初始偏移量是100H。因此，有了实验二的基础，这里很易于理解：jmp指令要显性说明段前缀的地址，0: 8100H地址与800H: 100H地址是同一地址，这样跳转到COM格式的内核程序执行，程序才能够正确执行。

• 操作系统内核程序的设计

本次实验，我将内核程序的设计分为三个部分：入口程序rukou.asm(汇编描述)、内核功能组织程序kernal.c(C语言描述)、硬件相关基本功能过程basic.asm(汇编描述)。

下面分别说明这三部分：

1. 入口程序

在汇编程序与C语言程序混合编程时，不同于单纯C语言的编程，程序默认从main()入口进入执行。因此要想利用C语言组织整个操作系统的功能操作，首先需要有一个程序跳转至C语言程序的cmain函数，因此，我将入口程序作为这样一个程序，作用就是调用C模块的cmain函数，跳转至cmain处执行。

另外，由于入口程序需要包含底层硬件相关的过程实现的程序basic.asm（有点类似于高级语言中源文件与头文件的关系），因此在入口程序程序段尾需要include basic.asm。

特别地是，因为内核程序要求是COM格式程序，因此开头需要加上org 100h，并且在程序开始处用CS对DS、ES、SS寄存器赋值。另外，由于TASM汇编对程序格式有严格要求，因此需要严格按照老师给的模板，在可以编码区进行汇编编程。

下面显示可编程区的部分代码：

```
org 100h ;COM格式文件的内存起始偏移地址

start:
    mov ax,cs;置其它寄存器与CS相同
    mov ds,ax; DS = CS
    mov es,ax; ES = CS
    mov ss,ax; SS = CS
    mov sp, 0FFF0h ;栈顶指针
    mov ah,2
    mov bh,0
    mov dx,0
    int 10h ;设置光标初始位置
    call near ptr _cmain ;函数调用，进入C语言cmain函数，开始运行OS内核部分
    jmp $ ;无限循环
    include basic.asm ;包含底层硬件相关过程实现程序
```

2. 实现硬件相关功能的程序

由于C语言无法实现一些硬件相关的功能，因此将这部分交由汇编模块实现，之后由C模块调用汇编模块的过程即可。在这部分程序中，我主要实现清屏、输入一个字符、输出一个字符以及读扇区加载用户程序的功能。在实验二的基础上，我们可知，上述这四种功能均可以利用BIOS功能调用实现，包括在这次实验中，我新学习的BIOS输出一个字符的功能调用。因此，对于这四种功能的实现并不难。

我将这四种功能每一个都封装成一个过程供C模块调用，因此需要将过程声明为public，且为了方便C模块的调用，过程名前都要加上下划线。

特别注意的是：每种BIOS功能调用的参数设定；作为汇编过程需要参数时，从栈中取得参数的操作以及要特别注意对寄存器的保护，因为在多过程调用时，保护现场以及恢复现场是至关重要的。

下面分别展示这四种功能：

- 清屏操作

```

public _clear ;为了方便C程序调用加上下划线
_clear proc
; 清屏
    push ax ;保护各个寄存器的值，因为在BIOS调用会更改这些寄存器的值，所以需要保护现场
    push bx
    push cx
    push dx
    ;使用int10h的向上滚屏即清窗口功能
    mov ah,06h ;入口参数，功能号：ah=06h-向上滚屏，ah=07h-向下滚屏
    mov al,0h ;滚动行数（0-清窗口）
    mov ch,0 ;窗口的左上角位置（x坐标）
    mov cl,0 ;窗口的左上角位置（y坐标）
    mov dh,24 ;窗口的右下角位置（x坐标）
    mov dl,79 ;窗口的右下角位置（y坐标）
    mov bh,7 ;空白区域的缺省属性
    mov bl,0
    int 10h ;中断号
    mov ah,2
    mov bh,0
    mov dx,0
    int 10h ;设置光标位置
    pop dx ;恢复寄存器的值，即恢复现场
    pop cx ;这样的操作有利于防止程序出现一些奇怪的错误，尽量减少在调用过程中对寄存器的破坏
    pop bx
    pop ax
    ret ;返回，因为这些过程模块均是被调用执行的，所以需要返回主程序
_clear endp

```

■ 输入一个字符

```

;调用中断号16h的0h功能读入一个字符
public _scanfCh ;过程名加上下划线，且声明为public，供C调用
_scanfCh proc
    mov ah,0 ;功能号
    int 16h
    mov byte ptr [_InCh],al ;用一个字符指针保存输入的字符，InCh为C模块的变量，byte ptr指定类型
    ret
_scanfCh endp

```

■ 输出一个字符

实验二时发现从键盘输入一个字符，用户无法在显示区域观察到字符，这不利于用户对于输入字符正确性的判断，因此，在本次实验中，为了增强界面的可观性，我在BIOS调用大全中找到，BIOS有输出一个字符的功能调用，因此打算将用户输入的字符利用一个C变量保存，再利用C模块将此C变量作为参数调用底层程序的输出一个字符的功能，输出用户输入的字符，便于观看。

输出一个字符的过程，有一个字节类型的参数，由调用者通过栈传递，由于过程调用时会将IP压栈，而且访问栈中参数时，不能直接利用栈指针SP，需要利用BP，因此出于保护BP寄存器的目的，也需要将BP寄存器的原值入栈，因此利用BP寄存器访问栈中参数时，需要从[BP+4]处访问。另外，根据实验原理部分的描述，C模块调用汇编过程，参数自动入栈，执行后汇编模块也不需要参数出栈。

```

;调用中断号10h的0Eh功能，在Teletype模式下显示字符
public _printfch
_printfch proc
    push bp ;保护bp的值，因为接下来要用bp去访问栈（参数在栈中）
    mov bp,sp ;栈顶指针sp赋给bp
    mov ax,[bp+4];IP、bp会入栈，因此参数在bp+4处
    mov ah,0EH ;ah为功能号
    mov bl,0h ;前景色（图形模式）
    int 10h ;10h号功能调用
    mov sp,bp;保险起见，将bp值还给sp
    pop bp ;恢复bp的值
    ret ;返回
_printfch endp

```

■ 加载用户程序

类似于实验二，将用户程序存放在软盘的特定扇区中，内核程序加载某个用户程序时，只需要读取对应用户程序所在的扇区，将其加载在内存自己设定的位置并跳转执行即可。

与实验二不同的是，实验二规定用户程序是COM格式，因此在跳转时需要指定段前缀，才能正确跳转，而在本次实验中老师对于用户程序的格式没有硬性要求，因此考虑到内核已经是COM格式程序，CS寄存器的值不为0且为了保险起见，在本次实验中将用户程序编译成BIN格式程序，因此加载用户程序时，只需要将bx寄存器赋值为偏移量，并利用call指令跳转至相应内存位置执行即可。

实验二中跳转的扇区号是根据用户输入的字符设定，而本次实验，我将扇区号作为此“加载用户程序”的参数，由C模块根据用户的选择将对应用户程序的扇区号以字符形式作为参数传给本过程。因此本过程同样涉及从栈中取参数的过程，需要根据入栈的内容判断从栈中哪个位置取得参数！

下面展示部分代码：

```

;实验要求有一个表存储用户程序相关信息，将之存放在第11个扇区，类似作为第五个用户程序
OffsetofUP equ 0A100H ;用户程序加载至内存的初始偏移量，自己设定

public _UP
_UP proc
    push ds ;涉及在过程中改变值得寄存器最好入栈保护一下
    push es
    push bp ;保护现场，保护ds、es、bp寄存器的值
    mov bp,sp;用bp访问栈中传入的参数，因为IP、ds、es、bp压栈，所以参数在bp+8的位置
    mov ax,cs
    mov es,ax ;置es与cs相等，为段地址，入口时CS为0800H
    mov bx,OffsetofUP ;数据常量，代表偏移地址，段地址:偏移地址为内存访问地址
    mov ah,02h ;功能号02h读扇区
    mov al,1 ;读入扇区数
    mov dl,0 ;驱动器号，软盘为0，U盘和硬盘为80h
    mov dh,0 ;磁头号为0
    mov cx,[bp+8] ;起始扇区号，编号从1开始，传入的参数即为对应用户程序所在扇区
    mov ch,0 ;柱面号为0
    sub cl,48 ;传入的是字符，因此需要转换成整数
    int 13h ;中断号
    ;将对应用户程序加载到内存为0800H:8c00H处
    mov bx,OffsetofUP ;将偏移地址赋值给bx

```



```

call bx ;程序跳转到对应内存位置执行用户程序
pop bp ;恢复现场
pop es
pop ds ;恢复现场，逆序出栈
ret ;函数模块返回
_UP endp

```

3. C模块程序

由于C语言更加高级且功能更加齐全，因此在本次实验选择用C语言作为整个内核程序的主程序，入口程序跳转至C模块的cmain函数，通过在cmain函数里的各种汇编过程，C模块函数调用，通过命令进行操作选择。

下面根据函数功能分别叙述

■ 输入字符串

由于C模块不是单独的C语言编程，因此无法包含C的相关头文件，因此一些库函数等无法使用，包括scanf、printf等，因此在C模块中想要输入字符串需要自己编写一个函数。但好在，汇编底层模块已经实现了基本的功能：输入一个字符，因此可以通过对汇编过程的调用实现。

在程序测试运行的时候发现，由于是字符单个输入，很容易出现输入错误的情况，通过网上查询资料发现BIOS输入backspace退格字符时候，只会将光标后移。因此我添加了退格的操作，通过判断输入字符是否为退格，若为退格，如果不是输入的第一个字符，那么就先输出退格，让光标前移，再调用汇编过程输出一个空格，光标后移，接着再输出一个退格让光标前移，重新调用汇编输入一个字符的过程，再次输入一个字符。记得将存储字符的数组下标前移一个单位，用新输入的字符覆盖原来的字符。而如果是输入的第一个字符就为退格那么重新输入即可。字符串的输入以回车结束，记得最后将字符串末尾加上'\0'。

```

void InStr(char * cmd)//参数用来存储输入的字符串
{
    int index = 0;//字符数组下标
    scanfCh();//汇编过程调用，输入一个字符
    while (InCh != 13)//判断是否为回车
    {
        if (InCh == 8)//退格处理
        {
            if (index != 0)
            {
                printfCh(InCh);//光标前移
                printfCh(32);//输出空格，光标后移
                index--;//数组下标前移
                printfCh(InCh);//光标前移
                scanfCh();//重新输入字符
                continue;
            }
            else
            {
                scanfCh();
                continue;
            }
        }
        printfCh(InCh);//将刚刚输入的字符输出，输入的字符作为参数自动入栈，
        增强界面可观性
        cmd[index] = InCh;//将输入的字符存储在字符数组
    }
}

```

```

        index++;
        scanfCh();//循环输入
    }
    cmd[index] = '\0';//最后加上字符串结尾标志
}

```

■ 输出字符串

为了增强系统的交互性以及可操作性，想要利用汇编的基本过程：输出一个字符实现C模块中输出字符串的操作，用以输出一些提示性的语句。基本思想为：参数传入一个字符数组，通过一个while循环判断当前下标对应的字符是否为'\0'，如果不是，则调用汇编输出一个字符的过程，输出这个字符；反之退出循环。

```

void printf(char *str)
{
    int i = 0;//数组下标
    while (str[i] != 0)//判断是否到字符串结尾
    {
        printfCh(str[i]);//调用汇编过程输出一个字符！
        i++;
    }
}

```

■ 字符串比较

由于通过C模块进行操作选择，因此本次实验的操作命令可以不像实验二那样单一，可以以字符串命令的形式展现。那么当用户输入字符串命令时，为了进行操作匹配，需要将用户输入的命令跟所有菜单命令进行比较，当两者相等时，执行相应的命令。基于上述分析，这就涉及到字符串的比较，基本思想为：设置两个变量分别作为两个字符串的下标，下标从0开始依次比较对应字符串下标对应的字符是否相等，不等时直接跳出循环，返回0代表字符串不相等，否则依次比较，直到到达任一字符串结尾，此时比较两个字符串是否都到达结尾，若是则两个字符串相等，反之不相等。

```

int Cmp(char *a, char *b)
{
    int i = 0;//a数组的下标
    int j = 0;//b数组的下标
    while (a[i] != 0 && b[j] != 0)//当两个字符串均未到达结尾时
    {
        if (a[i] != b[j])//只要发现一个不相等，就返回0
        {
            return 0;
        }
        i++;
        j++;//依次比较
    }
    if (a[i] == 0 && b[j] == 0)//判断两个字符串是否同时到达结尾，即大小是否相等
    {
        return 1;
    }
}

```

■ 运行函数

通过上述的字符串比较函数，进行操作命令的匹配，匹配成功后执行对应的操作，也即执行对应的用户程序，此时调用汇编模块的加载用户程序过程，将对应用户程序所在的扇区号作为参数传递给汇编过程，汇编过程根据参数加载对应用户程序执行并返回。

```
void Run(char* cmd)
{
    if (Cmp(cmd, "UP1"))
    {
        clear();
        UP(56); //执行第一个用户程序，用户程序1存放在第8个扇区
        clear();
    }
    else if (Cmp(cmd, "UP2"))
    {
        clear();
        UP(57); //执行第二个用户程序，用户程序2存放在第9个扇区
        clear();
    }
    else if (Cmp(cmd, "UP3"))
    {
        clear();
        UP(58); //执行第三个用户程序，用户程序3存放在第10个扇区
        clear();
    }
    else if (Cmp(cmd, "UP4"))
    {
        clear();
        UP(59); //执行第四个用户程序，用户程序4存放在第11个扇区
        clear();
    }
    else if (Cmp(cmd, "table"))
    {
        clear();
        UP(60); //执行表格程序，存放在第12个扇区
        clear();
    }
    else if (Cmp(cmd, "DIY"))
    {
        clear();
        printf("Please separate cmds by Spaces.\n\r");
        InStr(command); //执行自定义用户程序执行顺序命令，输入自定义顺序字符串
        ANLcmd(command); //从多命令字符串中依次提取出子命令
    }
    else if (Cmp(cmd, "qb"))
    {
        flag = 0; //终止自定义命令操作，有qb自定义终止，或者多命令执行完自动终止两种方式
    }
    else if (Cmp(cmd, "help"))
    {
        help(); //命令清单，帮助信息
    }
    else if (Cmp(cmd, "cls"))
    {
        clear(); //清屏操作，调用汇编清屏过程
    }
}
```

```

else if (Cmp(cmd, "quit"))
{
    end = 0; //终止系统执行，通过end标志判断
}
}

```

■ 多命令分析函数

类似于实验二的要求需要设计一种命令，可以按照自定义的顺序执行4个用户程序，在实验二中，我是利用栈保存这些顺序的命令。而在C语言中，我们可以通过字符串的截取操作，通过对输入的含多个命令的长字符串进行以空格为分隔符的截取操作，依次传入对应用户程序所在扇区号作为参数，调用汇编：加载用户程序过程并返回再执行下一个子命令，实现多命令的操作。

下面展示代码部分：

```

void ANLcmd(char * cmd)
{
    int index = 0; //多命令字符串下标
    char subcmd[10]; //用以存储子命令的数组
    int j = 0;
    while (cmd[index] != '\0' && end && flag) //当多名字符串到达结尾或执行了
        终止系统命令或执行了终止多命令处理命令时退出执行
    {
        if (cmd[index] == 32) //空格为分隔符
        {
            subcmd[j] = '\0'; //得到一个子命令
            Run(subcmd); //执行子命令
            j = 0; //子命令数组下标重新归零
        }
        else
        {
            subcmd[j] = cmd[index]; //未遇到空格，则读取子命令字符
            j++;
        }
        index++;
    }
    if (cmd[index] == '\0') //最后一个命令
    {
        subcmd[j] = '\0';
        Run(subcmd); //执行最后一个命令
    }
}

```

■ 界面提示信息

为了增强界面的可观性，以及系统的可操作性，设计Menu显示菜单函数以及help命令清单函数，下面以截图形式展示：

```

void Menu()
{
    printf("
    printf("
    printf("
    printf("
    printf("*****\n\r");
    printf("|
    printf("|
    printf("|
    printf("*****\n\r");
    printf("
    printf("
    printf("
    printf("
    printf("*****\n\r");
}

```

```

void help()
{
    printf("\n\rhelp-display the command list.\n\r");
    printf("cls-clear the window.\n\r");
    printf("UP1-run the first program of user.\n\r");
    printf("UP2-run the second program of user.\n\r");
    printf("UP3-run the third program of user.\n\r");
    printf("UP4-run the fourth program of user.\n\r");
    printf("table-display the table of UPInfo.\n\r");
    printf("DIY-run a series of commands.\n\r");
    printf("qb-exit the DIY running.\n\r");
    printf("quit-exit the system.\n\r");
}

```

■ 主函数部分

主函数部分作为入口程序的入口点，需要组织整个系统的执行。在程序刚开始的时，为了模拟古老操作系统启动界面，新建了一个程序Loading用以在系统启动时，利用进度条显示（此部分纳入用户程序中再做详细叙述），因此，cmain函数开始就调用Loading程序，调用汇编模块的加载用户程序过程，读取Loading程序所在扇区并跳转执行。之后，调用Menu函数，显示一些提示性信息，之后便开始循环进行输入字符串命令的过程，用一个全局变量command保存命令，输入命令之后，调用Run函数进行操作命令匹配，根据匹配，调用相应的函数，执行相应的操作。整个循环以quit指令为终止，当用户输入quit指令时，置标志end为0，退出循环，输出退出提示信息即可。

下面展示代码部分：

```

cmain()
{
    UP(61); //启动界面Loading程序在第13个扇区，加载执行
    clear(); //清屏操作
    Menu(); //显示欢迎、提示信息
    while (end) //当输入quit时end=0，退系统
    {
        printf("\n\r>>"); //输出命令行提示符
        InStr(command); //输入字符串命令
        Run(command); //根据命令匹配，执行操作
    }
    clear(); //清屏操作
    printf("\n\rQuit! Bye-bye!"); //退出信息
    scanfCh();
}

```

整个内核程序以C模块为结构中心，C模块提供操作界面，进行操作匹配，程序运行的组织；入口程序起到入口作用，为程序的正确执行提供正确入口；而汇编实现的基本功能的程序则是整个内核程序的底层支持。三者结合，实现整个COM格式的内核程序。

• 用户程序的设计

本次实验的四个用户程序，与实验设计相同，稍有不同之处在于：

1. 本次实验设计将用户程序编译成BIN格式程序，并将用户程序加载到初始偏移量为0A100h处，但由于本次实验是由内核程序作为监控程序实现加载用户程序的功能，而内核程序是COM格式，引导扇区程序将内核程序加载到内存地址为0：8100h处，那么程序跳转至内存之后cs寄存器的值应该为800h，因此在内核中读取用户程序扇区并跳转执行时，用户程序是被加载到800h：A100h处。由于用户程序是BIN格式没有org初始偏移量的限制，因此内核程序跳转至用户程序执行时用：call 0A100h指令即可，用户程序开头用org 0A100h声明程序加载到内存的初始偏移地址并对相应ds、es、ss寄存器赋值即可正确运行。

除此之外，程序返回时不同于实验二的使用jmp 0：7c00h。call指令与ret的配合使用，可以让程序返回至原程序的下一条指令执行，由于这是由C模块调用使用的汇编过程，返回下一条指令执行刚好与C模块的执行思想相吻合，因此使用ret指令返回。

2. 由于实验二的设计中用户可以输入空格结束用户程序的执行返回主程序，而在本次实验中，由于加入了输出字符的操作，用户每输入一个字符，均会被输出，因此在测试时发现，若用户输入空格返回，会使得回到内核程序界面时，程序输出了一个空格，显得不美观。联想到前面在设计输出字符的时候，进行了退格输入的设计，因此，在本实验将用户终止用户程序执行的标志改成：输入退格，这样就不会出现上述情况。
3. 另外，由于本程序设置了自定义顺序执行四个用户程序的命令，因此为了便于自定义顺序时，用户程序可以自动退出，本次实验在四个用户程序中均设置了一个变量作为计数器，但字符运动次数达到计数器上限时，程序自动退出。这大大优化了界面，也优化执行用户程序的返回问题。

由于，在实验二中已经描述了这四个用户程序的相关设计，因此在此不一一赘述。

下面基于上述三点改善，展示相关代码：

org 0A100h;将引导扇区加载到内存中的初始偏移地址为0A100h处

```
;程序的终止由用户输入退格键终止
;调用int 16h输入一个字符判断
Judge:
    mov ah,01h
    int 16h
    mov bl,8;退格键ascii码
    cmp al,bl
    jz Quit
    mov bx,100;字符运动次数上限
    mov ax,word[count];计数变量
    cmp ax,bx
    jz Quit
    jmp Loop

Quit:
    ret;ret指令用栈中的数据，修改IP的内容，从而实现近转移
```

另外，由于实验要求需要制作显示一个存储用户程序相关信息的表格，因此类似于实验二的思想我将显示表格作为一个用户程序，通过加载表格程序所在扇区执行显示表格信息操作。

◦ 表格程序

表格程序中，主要利用BIOS显示字符串的功能调用将表格信息以一行行字符串的形式显示。

下面截取部分代码展示：

```

showInfo:
    ;调用int10h显示操作系统启动的相关提示信息
    mov ax,cs
    mov es,ax
    mov bp,Message;偏移地址,采用段地址:偏移地址形式es:bp表示串地址
    mov ax,1301h ;ah=13h为功能号,a1=01h指光标位于串尾
    mov bx,0007h ;bh=00h为页码,b1=07h为显示属性:黑底白字
    mov dx,0000h ;dh为行号,d1为列号
    mov cx,MessageLength
    int 10h

```

```

Message:;显示相关表格信息,含用户程序名、程序在软盘的地址,程序大小
Info:db 'There are four programs of user!',0Dh,0Ah;
UP1name:db 'UP1:Square'
UP1addr:db '||addr:0E00h~1000h:eighth
section||size:407Bytes',0DH,0AH
UP2name:db 'UP2:Single stone'
UP2addr:db '||addr:1000h~1200h:ninth
section||size:438Bytes',0DH,0AH
UP3name:db 'UP3:Double stone'
UP3addr:db '||addr:1200h~1400h:tenth
section||size:508Bytes',0DH,0AH
UP4name:db 'UP4:Sand clock'
UP4addr:db '||addr:1400h~1600h:eleven
section||size:326Bytes',0DH,0AH
MessageLength equ ($-Message)

```

◦ Loading程序

受用户程序的启发,联想到一些古老操作系统启动时出现的蓝屏Loading进度条界面,本次实验也想要设计一个类似的简易版启动时的Loading界面。

设计思想主要为:先将屏幕的中心区域调用清屏功能调用清屏成蓝底,再在屏幕相应位置显示相关信息。进度条的运动展示类似于空格字符向右运动,每隔一定时间单位字符向右运动一个单位,设定一个右边界,当到达右边界时,程序终止,返回内核程序。

将Loading程序类似于表格程序也当成一个用户程序,在引导扇区跳转至内核执行,由入口程序进入cmain函数时,首先将Loading程序所在的扇区号作为参数,调用汇编基本功能实现程序的加载用户程序过程,实现Loading程序的运行并返回。

下面展示部分代码:

```

clear:;清屏成蓝底操作
    mov ah,06h;向上滚屏
    mov al,0h;滚动行数(0~清窗口)
    mov ch,2;左上角X坐标
    mov cl,10;左上角Y坐标,选取全屏的中心部分区域
    mov dh,21;右下角X坐标
    mov dl,70;右下角Y坐标
    mov bh,0x10;空白区域的显示属性:蓝底黑字,无闪烁无加亮
    int 10h;BIOS功能调用

```

```

showInfo:;显示字符串操作，显示多行字符串时，需要谨慎选择行和列，考虑美观问题！
;调用int10h显示操作系统启动的相关提示信息
mov ax,cs
mov es,ax
mov bp,Message;偏移地址，采用段地址：偏移地址形式表示串地址
mov ax,1301h ;ah=13h为功能号，al=01h指光标位于串尾
mov bx,00deh ;bh=00h为页码，bl=07h为显示属性：红底黄字，有闪烁
mov dx,0a19h ;dh为行号，dl为列号,第10行，25列
mov cx,MessageLength
int 10h

```

```

;进度条显示操作，即空格字符向右运动
Loop:
    dec word[num]
    jnz Loop
    mov word[num],delay
    dec word[dnum]
    jnz Loop
    mov word[num],delay
    mov word[dnum],ddelay
    jmp Rt

Rt:
    inc word[y]
    mov bx,word[y]
    mov ax,56
    cmp ax,bx;判断是否到达右边界，到达则程序返回！
    jnz display;为到达则显示字符
    jmp Quit

```

上述展示了一些特殊之处代码，而在显存显示字符在实验一就已实践，在此不再赘述。

• 实验操作过程

1. NASM编译

打开NASM，在框内输入指令，对于引导扇区程序boot.asm、四个用户程序UP系列、表格程序table.asm以及Loading程序，由于不限定二进制文件格式，为了避免出现一些难以发现与解决的错误，我将他们都编译成BIN格式。因此输入nasm+输入文件名+-o+输出文件名即可。生成同名的二进制文件。


```
nasm
Microsoft Windows [版本 10.0.18363.778]
(c) 2019 Microsoft Corporation。保留所有权利。

C:\Users\HP\AppData\Local\bin\NASM>nasm boot.asm -o boot.bin
C:\Users\HP\AppData\Local\bin\NASM>nasm UP1.asm -o UP1.bin
C:\Users\HP\AppData\Local\bin\NASM>nasm UP2.asm -o UP2.bin
C:\Users\HP\AppData\Local\bin\NASM>nasm UP3.asm -o UP3.bin
C:\Users\HP\AppData\Local\bin\NASM>nasm UP4.asm -o UP4.bin
C:\Users\HP\AppData\Local\bin\NASM>nasm table.asm -o table.bin
C:\Users\HP\AppData\Local\bin\NASM>nasm Loading.asm -o Loading.bin
C:\Users\HP\AppData\Local\bin\NASM>_
```

2. TCC+TASM+TLINK链接编译

在DosBox中链接编译内核程序：rukou.asm、kernal.c、baxis.asm

■ TCC编译

用tcc+.c 形式生成同名.obj文件：

```
DOSBox 0.74, Cpu speed: 3000 cycles, Fra...
D:\>cd t
D:\T>tcc kernal.c
Turbo C Version 2.01 Copyright (c) 1987, 1988 Borland International
kernal.c:
Turbo Link Version 2.0 Copyright (c) 1987, 1988 Borland International
c0s.obj : unable to open file

Available memory 445908
D:\T>
```

■ TASM编译

用tasm+.asm形式先生成同名.obj文件，由于rukou.asm包含了basic.asm文件，因此还需要用tasm+rukou.obj形式再编译一次：

```
D:\T>tasm rukou.asm
Turbo Assembler Version 4.1 Copyright (c) 1988, 1996 Borland International

Assembling file:   rukou.asm
Error messages:    None
Warning messages:  None
Passes:            1
Remaining memory:  466k

D:\T>_
```

```
D:\T>tasm rukou.obj
Turbo Assembler Version 4.1 Copyright (c) 1988, 1996 Borland International

Assembling file:   rukou.ASM to obj.OBJ
Error messages:    None
Warning messages:  None
Passes:            1
Remaining memory:  466k

D:\T>_
```

■ TLINK链接

用tlink /t /3 .obj .obj, .com形式生成内核COM格式文件：

```
D:\T>tlink /t /3 rukou.obj kernal.obj, test.com
Turbo Link Version 2.0 Copyright (c) 1987, 1988 Borland International
D:\T>_
```

3. winHex工具编辑软盘

不同于实验一，在Linux环境使用dd命令进行1.44MB虚拟软盘映像的生成，在实验二中，我利用WinHex编辑软盘，两者相比较，我认为WinHex工具更为简便操作。因此，本次实验依然使用WinHex编辑软盘。

首先生成一个1.44MB的空白软盘test2.img.利用winHex工具打开上述生成的各个二进制文件，先将“boot.bin”二进制文件内容拷贝到空白软盘首扇区，然后将内核程序test.com存放在扇区号为2开始的六个扇区空间，起始地址为200h；接着再从第8个扇区开始，依次存放四个用户程序(UP1~4)、表格程序以及Loading程序。第一个用户程序UP1存放在第二个扇区（E00H~1000H），第二个用户程序UP2存放在第三个扇区（1000H~1200H）的规则，以此类推将UP1~UP4的二进制文件以及表格程序和Loading程序拷贝到对应扇区中。

拷贝操作的过程如下所示：

boot.bin	loading.bin	UP1.bin	UP2.bin	UP3.bin	UP4.bin	table.bin	TEST.COM	test2.img									
Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
00000010	00	B1	02	CD	13	EA	00	01	00	08	00	00	00	00	00	00	± í ë
00000020	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00000030	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00000040	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00000050	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00000060	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00000070	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00000080	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00000090	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000000A0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000000B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000000C0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000000D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000000E0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000000F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00000100	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00000110	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00000120	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00000130	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00000140	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00000150	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00000160	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00000170	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00000180	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00000190	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000001A0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000001B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000001C0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000001D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000001E0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000001F0	00	00	00	00	00	00	00	00	00	00	00	00	00	55	AA		U ²

上图为“boot”引导扇区程序二进制文件内容，利用操作：选中需要复制内容，右键——编辑——复制选块——正常，完成复制操作。

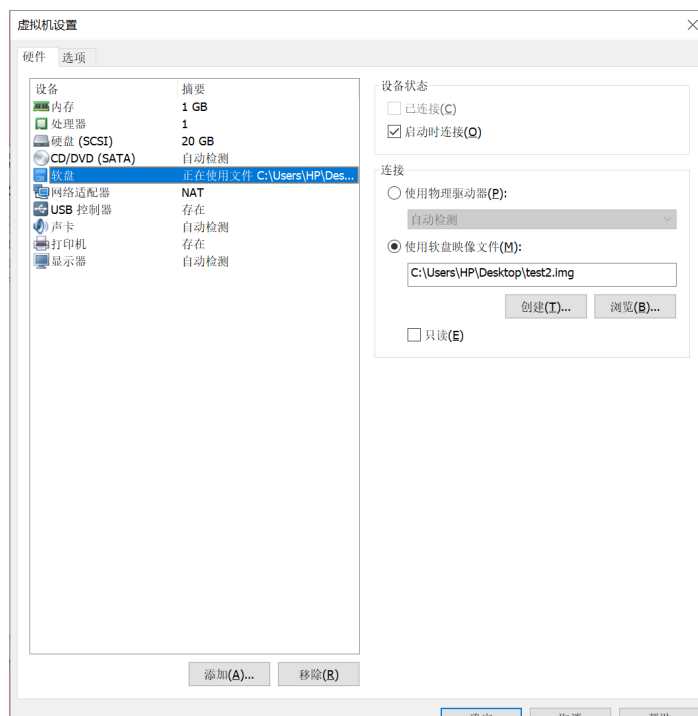
boot.bin	Loading.bin	UP1.bin	UP2.bin	UP3.bin	UP4.bin	table.bin	TEST.COM	test2.img								
Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00000010	00	B1	02	CD	13	EA	00	01	00	08	00	00	00	00	00	00
00000020	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000030	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000040	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000050	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000060	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000070	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000080	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000090	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000000A0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000000B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000000C0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000000D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000000E0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000000F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000100	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000110	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000120	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000130	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000140	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000150	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000160	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000170	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000180	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000190	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000001A0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000001B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000001C0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000001D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000001E0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000001F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	55	AA

上图为1.44MB软盘，将上述“boot”的内容复制到此软盘的首扇区，操作为：右键——剪贴板数据——粘贴。

其余程序的拷贝操作类似。

4. 软盘启动裸机

将上述生成的引导扇区程序二进制文件、内核COM格式文件、用户程序以及表格程序、Loading程序依次按照扇区分配复制到1.44MB软盘的对应位置并保存后，将1.44MB软盘作为引导盘启动虚拟机。

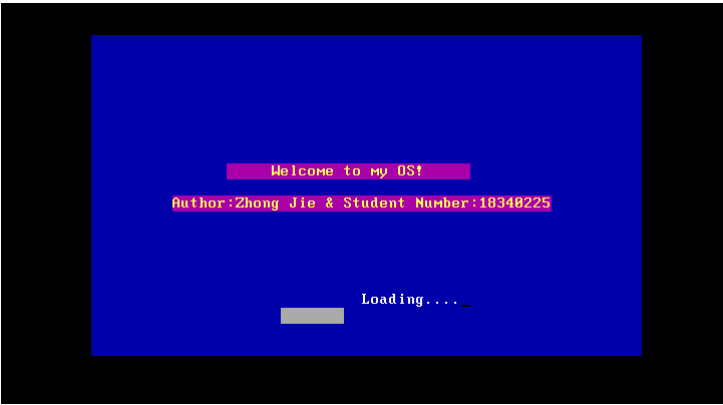


• 实验运行结果

成功启动虚拟机之后，虚拟机就会将程序加载到虚拟机的内存空间中运行，可以看到Loading程序的启动界面，也可以观察到内核程序的界面，接着输入help，依次按照菜单，输入选择运行不同的命令，执行不同的用户程序，并输入退格键返回内核程序，另外测试按照某种指定顺序执行用户程序、显示用户程序相关信息等操作。

程序运行结果如下：

Loading程序加载界面：



内核界面，输入“help”，显示命令清单：

```

      _ _ _ _ _
    _/  _  _  _ \_
   /  _  _  _  \_
  /  _  _  _  \_
 /  _  _  _  \_
/  _  _  _  \_
\  _  _  _  /_
 \  _  _  _ /_
  \  _  _  /_
   \  _  /_
    \  _/
     \_

*****
:      Author:Jie Zhong  Student Number:18340225      :
:      Input help to view the command list.           :
:      Have a good time!                              :
*****

>>help
help-display the command list.
cls-clear the window.
UP1-run the first program of user.
UP2-run the second program of user.
UP3-run the third program of user.
UP4-run the fourth program of user.
table-display the table of UPInfo.
DIY-run a series of commands.
qb-exit the DIY running.
quit-exit the system.

>>_
```

输入“cls”,系统执行清屏操作：

```

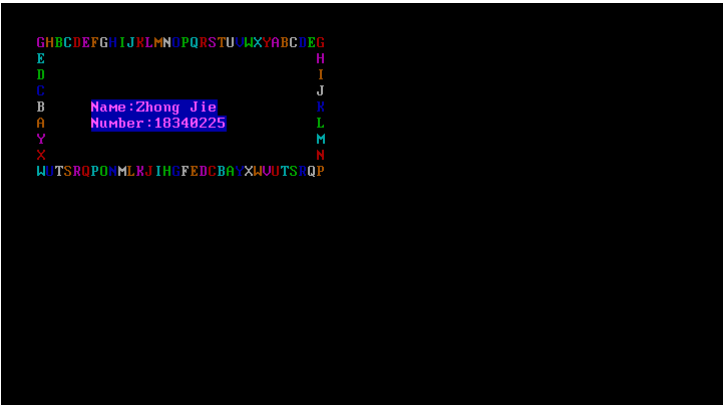
      _ _ _ _ _
    _/  _  _  _ \_
   /  _  _  _  \_
  /  _  _  _  \_
 /  _  _  _  \_
/  _  _  _  \_
\  _  _  _  /_
 \  _  _  _ /_
  \  _  _  /_
   \  _  /_
    \  _/
     \_

*****
:      Author:Jie Zhong  Student Number:18340225      :
:      Input help to view the command list.           :
:      Have a good time!                              :
*****

>>help
help-display the command list.
cls-clear the window.
UP1-run the first program of user.
UP2-run the second program of user.
UP3-run the third program of user.
UP4-run the fourth program of user.
table-display the table of UPInfo.
DIY-run a series of commands.
qb-exit the DIY running.
quit-exit the system.

>>cls
```

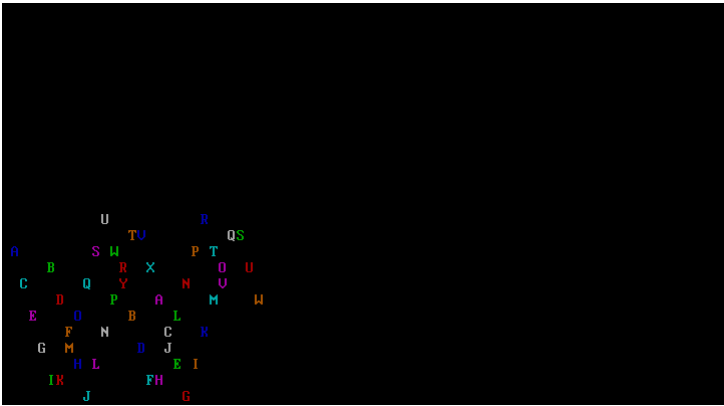
接着输入“UP1”，显示第一个用户程序：

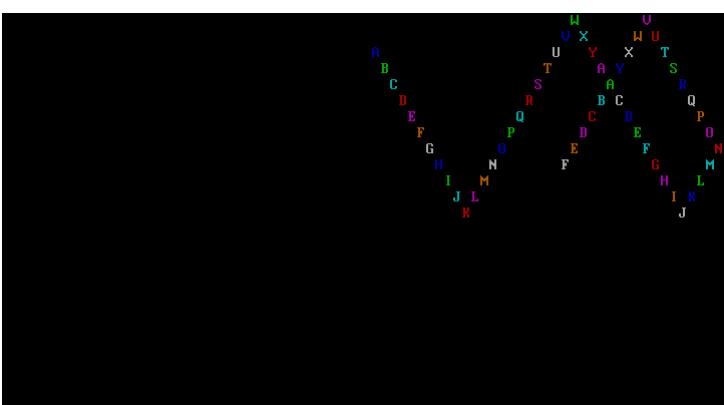


输入退格可以回到内核程序，也可以等待一段时间程序会自动返回，接着输入“UP2”，显示第二个用户程序：


```
Please separate cmds by Spaces.  
UP3 UP2 UP1 UP4
```

按顺序依次执行UP3、UP2、UP1、UP4:





G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
E	D	C	B	A	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
C	B	A	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
B	A	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

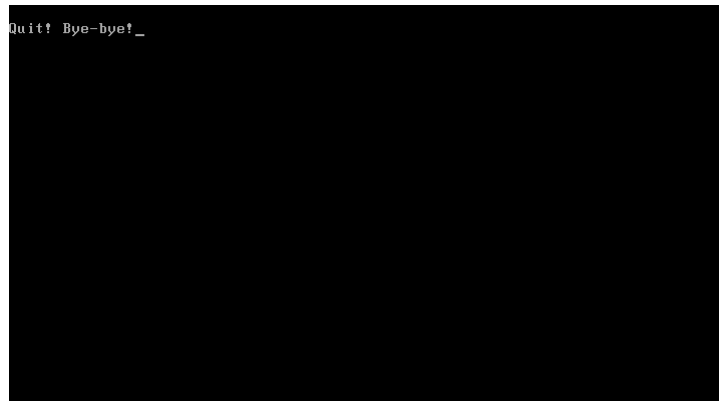
Name:Zhong Jie

Number:18348225

W	V	U	T	S	R	Q	P	O	N	M	L	K	J	I	H	G	F	E	D
C	B	A	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
E	D	C	B	A	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z



输入“quit”退出系统。



至此，完成所有功能测试，实验运行结果与预期相符合。

实验心得

本次实验其实是在上一次实验的基础上，将上一次实验的监控程序的功能集成在内核程序中，作为一个与引导扇区程序相独立的程序。考虑到引导扇区的字节数有限，不便于扩展功能，因此将开发拥有独立内核的操作系统。除此之外，由于汇编语言是更加偏向于硬件的语言，无论是语法结构还是编程组织都不够简洁方便，因此在本次实验引入了汇编语言与C语言混合编程的思想。我认为这也就是本次实验的重难点、关键点所在，其实程序的功能与实验二并无差异，只不过是实现形式上有了更多的不同，C语言的加入可以让我们的系统功能更完备、多样；也可以让我们的编程更加简洁。

由于在大学的大部分时间接触的都是高级语言编程，自然对于高级语言的编程更加的熟悉与亲切，俗话说“扬长避短”，因此本次实验，我就将自己更为熟悉的C语言作为整个内核程序的中心，由C语言模块来组织整个内核的操作命令。但由于在汇编与C语言的混合编程中，我们无法使用C的一些库函数以及scanf、printf等一些输入输出函数，因此，我还需要借助于汇编中的一些BIOS功能调用来实现。在前两次的实验中，我已经了解了一些常用的BIOS调用，包括清屏、输入一个字符、输出一个字符、读扇区以及显示字符串操作，基于此，我就将这些操作由汇编语言实现，组织在汇编模块中，供C模块的调用。当然，由于混合编程不像C的单独编程，程序默认从main函数开始执行，因此如何确定进入内核程序的入口点也是至关重要的，在听取了老师的建议之后，我单独建立了一个汇编程序，用于调用C模块的cmain函数，跳转至内核程序的入口执行点。“良好的开端是成功的一半”，程序顺利从入口进入以后，在C模块的组织运行也是比较容易把握的。

但我在初期编程的时候遇到了不少困难，也让我意识如何选择是一件很重要的事情。由于C模块函数与汇编模块过程可以相互调用，因此这给予了我们编程极大的自由。在编程初期，我便“花样百出”地调用来、调用去，直接造成地后果就是，程序胡乱的调用跳转，甚至我在debug的时候都要循着代码一条条重新看过，才知道接下来会跳转至那个模块执行。这不仅对于整个编程结构带来了不好的影响，也让我本来就不太熟练的混合编程遭到了致命的打击。于是，我推翻自己的代码，重新梳理结构，我认为在两者混合编程的过程中，一定要有主次之分，这样才会有更加分明清晰的结构，自己编程也不容易出错。而综合比较C调用汇编与汇编模块调用C，我发现C调用汇编模块对于我来说更加简易容易上手，因此我

使用汇编描述了上述几个由BIOS调用的基本功能过程，再用C模块去调用这些过程，这样我的程序结构就很明晰了，自己编程写起来也更加清楚与顺手了。

最后一点，在本次实验中使用DosBox编译链接也给我造成了不少的麻烦，tcc、tasm编译命令繁多，不同的命令对同一个程序编译的结果以及报错也会出现差异，最初我是用老师PPT上的命令编译，发现不对，于是，我上网查询，终于找到了大家普遍使用的命令，这解决了我的一个小问题。另外，由于我的入口程序是包含底层基本功能实现程序的，最初我认为，这样三个文件的编译链接与两个文件的没有差别，但当我多次生成COM文件加入扇区，在虚拟机上运行时发现屏幕没有反应。于是，我又上网查询，找到了原来这种include以后的文件，在tasm生成同名obj文件之后，需要再tasm，obj一次才能正确执行。虽然这个过程很艰辛，但很感激我最终找到正确的编译链接命令以及顺序。

总而言之，实验不易，但收获繁多，望自己再接再厉，坚持下去！

参考文献

1. 《BIOS中断大全》：https://blog.csdn.net/weixin_37656939/article/details/79684611
2. 《突破512字节的限制》：<https://blog.csdn.net/cielozhang/article/details/6171783/>
3. 《在汇编程序中调用C函数》：<https://blog.csdn.net/longintchar/article/details/79511747>