

操作系统 实验报告

院系： 计算机学院

班级： 计科 2 班

学号： 19335174

姓名： 施天予

指导老师： 凌应标

一、实验题目

独立内核开发

二、实验目的

- 1、熟悉掌握 C 语言和汇编的混合编译
- 2、将实验二的操作系统分离为引导程序和内核，用引导程序加载内核

三、实验要求

- 1、寻找或测试一套匹配的汇编与 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、写一个汇编程和 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。利用汇编程和 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、利用汇编程和 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、重写引导程序，加载 COM 格式程序的独立内核。
- 6、拓展自己的软件项目管理目录，管理实验项目相关文档。

四、实验方案

【实验环境】

- 1、实验运行环境：Windows10

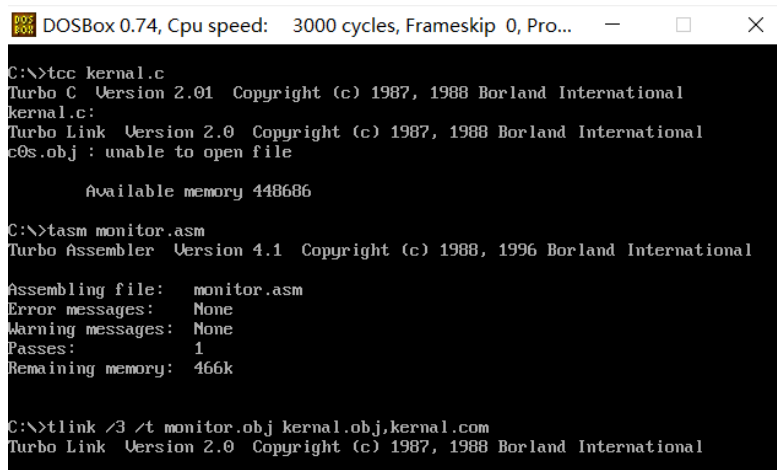
- 2、虚拟机软件：VirtualBox
- 3、NASM 编译器
- 4、DOSBox 下 TCC+TASM+TLINK 混合编译

【实验工具】

- 1、汇编语言：NASM，TASM
- 2、文本编辑器：Notepad++
- 3、相关软件：WinHex

【实验思想】

- 1、汇编与 C 语言的关系
 - 1) 汇编语言的目的
 - a. 设置自身运行模式和环境，设置硬件寄存器
 - b. 设置 I/O 端口和实现 I/O 操作
 - c. 初始化中断向量表和实现中断处理
 - d. 实现控制原语
 - 2) C 语言的目的
 - a. 适合构造复杂的数据结构和相关数据结构的的管理
 - b. 实现复杂的功能或算法
- 2、汇编与 C 语言的混合编译链接
 - 1) TCC 编译 C 语言程序
 - 2) TASM 编译汇编语言程序
 - 3) TLINK 链接 C 和汇编生成的 obj 文件，生成 COM 可执行文件



```
DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Pro...
C:\>tcc kernel.c
Turbo C Version 2.01 Copyright (c) 1987, 1988 Borland International
kernel.c:
Turbo Link Version 2.0 Copyright (c) 1987, 1988 Borland International
c0s.obj : unable to open file

Available memory 448686

C:\>tasm monitor.asm
Turbo Assembler Version 4.1 Copyright (c) 1988, 1996 Borland International
Assembling file: monitor.asm
Error messages: None
Warning messages: None
Passes: 1
Remaining memory: 466k

C:\>tlink /3 /t monitor.obj kernel.obj,kernal.com
Turbo Link Version 2.0 Copyright (c) 1987, 1988 Borland International
```

3) 编译步骤

将老师的 tcc-tasm 文件夹中的六个文件放入之前编译 nasm 的文件夹下，在 DOSBox 中定位到该文件夹，就可以使用 TCC, TASM, TLINK 了（使用 TLINK 时要把汇编生成的 obj 文件写在前面，C 生成的 obj 文件写在后面）。

五、实验过程

1、样板程序的编译与分析（将字符串大写）

【源代码】

```
1 ;程序源代码 (showstr.asm)
2 extrn _upper:near ;声明一个c程序函数upper
3 extrn _Message:near ;声明一个外部变量
4 .8086
5 _TEXT segment byte public 'CODE'
6 DGROUP group _TEXT, _DATA, _BSS
7 | assume cs:_TEXT
8 org 100h
9 start:
10     mov ax, cs
11     mov ds, ax ; DS = CS
12     mov es, ax ; ES = CS
13     mov ss, ax ; SS = CS
14     mov sp, 100h
15     call near ptr _upper
16     mov bp, offset _Message ; BP=当前串的偏移地址
17     mov ax, ds ; ES:BP = 串地址
18     mov es, ax ; 置ES=DS
19     mov cx, 18 ; CX = 串长 (=18)
20     mov ax, 1301h
21     ; AH = 13h (功能号)、AL = 01h (光标置于串尾)
22     mov bx, 0007h
23     ; 页号为0 (BH = 0) 黑底白字 (BL = 07h)
24     mov dh, 20 ; 行号=10
25     mov dl, 10 ; 列号=10
26     int 10h ; BIOS的10h功能: 显示一行字符
27     jmp $
28
29 _TEXT ends
30 ;*****DATA segment*****
31 _DATA segment word public 'DATA'
32 _DATA ends
33 ;*****BSS segment*****
34 _BSS segment word public 'BSS'
35 _BSS ends
36 ;*****end of file*****
37 end start
```

```

1  /*程序源代码 (upper.c) */
2  char Message[18]="OSaaabbbAaBbCcDdEe";
3  upper() {
4      int i=0;
5      while(Message[i]) {
6          if (Message[i]>='a' && Message[i]<='z')
7              Message[i]=Message[i]+'A'-'a';
8              i++;
9      }
10 }

```

【编译运行】

```

C:\>tcc -mt -c -oupper.obj upper.c
Turbo C Version 2.01 Copyright (c) 1987, 1988 Borland International
upper.c:

    Available memory 456414

C:\>tasm showstr.asm showstr.obj
Turbo Assembler Version 4.1 Copyright (c) 1988, 1996 Borland International

Assembling file:  showstr.asm
Error messages:   None
Warning messages: None
Passes:          1
Remaining memory: 468k

C:\>tlink /3 /t showstr.obj upper.obj,showstr.com
Turbo Link Version 2.0 Copyright (c) 1987, 1988 Borland International

```

DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Pro...

```

C:\>showstr.com

OSAAABBBBAABBCCDDEE

```

【变量与传参分析】

全局变量：Message 是 C 中定义的字符串，在汇编程序中用 extrn _Message: near 声明，C 程序中的变量名在汇编要加下划线。

局部变量：i 用于 upper 函数中的一个循环变量。

变量初始化：Message 在 C 中初始化为"0SaaabbbAaBbCcDdEe"。

函数调用：汇编调用 C 的函数要先声明 extrn _upper: near，调用 upper 函数 call near ptr _upper，调用 C 中的函数也要在函数名前加下划线。

参数传递情况：参数传递时，用 push word ptr _DATA :_b 就是实参 b 压栈，而 push word ptr _DATA :_a 就是实参 a 压栈，然后调用函数 f(a,b)，之后两个 POP 指令是调用后清除栈中的参数。说明调用 C 函数时，参数按后面参数先进栈的顺序压栈，进栈出栈以字为单位。

【列表文件】

在 DOSBox 下运用 tcc 将 upper.c 编译成 upper.asm 文件

```
C:\>tcc -mt -S -oupper.asm upper.c
Turbo C Version 2.01 Copyright (c) 1987, 1988 Borland International
upper.c:

Available memory 458678
```

```
tcc -mt -S -oupper.asm upper.c
```

```
_DATA    segment word public 'DATA'
_Message    label    byte
    db 79
    db 83
    db 97
    db 97
    db 97
    db 98
    db 98
    db 98
    db 65
    db 97
    db 66
    db 98
    db 67
    db 99
    db 68
    db 100
    db 69
    db 101
_DATA    ends
```

可以看到初始化的字符串变量 Message 被放在数据段，如果未初始化会放在 _BSS 段。

2、混合编程实例（统计字符次数）

【程序分析】

C 中的 fun 函数用于统计字符串 Message 中 a 的个数。在汇编程序中先用 public 声明字符串变量 Message 为全局变量，在 datadef 中定义 Message 并初始化。用 extrn 声明 C 中的 fun 函数，在 C 中用 extern char Message[] 声明汇编中定义的 Message 字符串变量。再用 call 调用 fun 函数，利用 BIOS 的 int 10h 输出字符串功能，设置串长 CX=1（统计结果是一个字符），成功实现。

【设计代码】

```
1 ;程序源代码 (count.asm)
2 extrn _fun:near
3 public _Message
4
5 .8086
6 _TEXT segment byte public 'CODE'
7 DGROUP group _TEXT, _DATA, _BSS
8 |       assume cs:_TEXT
9 org 100h
10
11 start:
12     mov ax, cs
13     mov ds, ax           ; DS = CS
14     mov es, ax           ; ES = CS
15     mov ss, ax           ; SS = cs
16     mov sp, 100h
17     call near ptr _fun
18     mov bp, offset _Message ; BP=当前串的偏移地址
19     mov ax, ds           ; ES:BP = 串地址
20     mov es, ax           ; 置ES=DS
21     mov cx, 1           ; CX = 串长 (=1)
22     mov ax, 1301h
23     ; AH = 13h (功能号)、AL = 01h (光标置于串尾)
24     mov bx, 0007h
25     ; 页号为0 (BH = 0) 黑底白字 (BL = 07h)
26     mov dh, 20          ; 行号=10
27     mov dl, 10          ; 列号=10
28     int 10h             ; BIOS的10h功能: 显示一行字符
29     jmp $
30
31 _datadef:
32 _Message db "OSaaabbbAaBbCcDdEe", 0
33
34 _TEXT ends
35 ;*****DATA segment*****
36 _DATA segment word public 'DATA'
37 _DATA ends
38 ;*****BSS segment*****
39 _BSS segment word public 'BSS'
40 _BSS ends
41 ;*****end of file*****
42 end start
```

```

1  /*程序源代码 (fun.c) */
2  extern char Message[];
3
4  fun() {
5      int i=0,k=0;
6      while(Message[i]) {
7          if (Message[i]=='a') k++;
8          i++;
9      }
10     Message[0]=k+'0';
11 }

```

【编译运行】

```

DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Pro...
C:\>tcc fun.c
Turbo C Version 2.01 Copyright (c) 1987, 1988 Borland International
fun.c:
Turbo Link Version 2.0 Copyright (c) 1987, 1988 Borland International
c0s.obj : unable to open file

    Available memory 456554

C:\>tasm count.asm
Turbo Assembler Version 4.1 Copyright (c) 1988, 1996 Borland International

Assembling file:    count.asm
Error messages:     None
Warning messages:   None
Passes:             1
Remaining memory:   468k

C:\>tlink /3 /t count.obj fun.obj,count.com
Turbo Link Version 2.0 Copyright (c) 1987, 1988 Borland International

DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Pro...
C:\>count.com

4

```

“OSaaabbbAaBbCcDdEe”中 a 的数量是 4，结果正确！

注意：_Message 必须定义在数据段_datadef，如果定义在代码段，会显示乱码！

3、生成独立内核

1) 引导程序(bootloader.asm)

上次实验是把引导程序和监控程序合在一起，这次分离后引导程序只要跳转到内核所在的扇区即可。我把引导程序放在首扇区，4个用户程序放在2、3、4、5扇区，内核放在第6扇区起始，内核的代码一共占了4个扇区

```
1 org 7c00h ;BIOS将把引导扇区加载到0:7C00h处，并开始执行
2 OffsetOfMonitor equ 0A100h ;加载操作系统内核
3
4 ReadOs:
5     mov ax,cs ;段地址 ; 存放数据的内存基地址
6     mov es,ax ;设置段地址(不能直接mov es,段地址)
7     mov bx, OffsetOfMonitor ;存放内核的内存偏移地址OffsetOfMonitor
8     mov ah,2 ;功能号
9     mov al,4 ;扇区数，内核占用扇区数 注意：不止加载了一个扇区
10    mov dl,0 ;驱动器号 ; 软盘为0，硬盘和U盘为80H
11    mov dh,0 ;磁头号 ; 起始编号为0
12    mov ch,0 ;柱面号 ; 起始编号为0
13    mov cl,6 ;存放内核的起始扇区号6
14    int 13h ;调用读磁盘BIOS的13h功能
15    jmp 0a00h:100h ;控制权移交给内核
16
17    times 510 - ($ - $$) db 0 ;将前510字节不是0就填0
18    db 0x55, 0xaa
```

2) 内核

a) 监控程序(monitor.asm)

监控程序的代码参考了老师的 afile.asm。其中 include kliba.asm 获取汇编库函数，并声明 extern _cmain: near，使用 call near ptr _cmain 来调用 kernel.c 中的 cmain 函数进入操作系统主界面。

```
1 extern macro %1 ;统一用extern导入外部标识符
2 extrn %1
3 endm
4
5 extern _cmain:near
6
7
8 ;*****
9 .8086 ;*
10 _TEXT segment byte public 'CODE' ;*
11 assume cs:_TEXT ;*
12 DGROUP group _TEXT,_DATA,_BSS ;*
13 org 100h ;*
14 ;*
15 start: ;*
16 ;*****
17 ;
18
19 mov ax,cs
20 mov ds,ax; DS = CS
21 mov es,ax; ES = CS
22 mov ss,ax; SS = CS
23 mov sp,0FFF0h
24 mov ah,2
25 mov bh,0
26 mov dx,0
27 int 10h
28
```

```

29     call near ptr _cmain
30     jmp $
31
32     include kliba.asm
33
34     ;*****
35     ;*
36     _TEXT ends                ;*
37     ;*
38     _DATA segment word public 'DATA' ;*
39     ;*
40     _DATA ends                ;*
41     ;*
42     _BSS      segment word public 'BSS' ;*
43     _BSS ends                ;*
44     ;*
45     end start                ;*
46     ;*****
47

```

b) 汇编库函数(kliba.asm)

这段代码在老师的 kliba.asm 的基础上修改。

实现了以下四个函数：

_cls: 清屏

_loadUser: 跳转到指定扇区加载运行用户程序

_getChar: 从键盘读取一个字符

_printChar: 显示一个字符

```

36 ;*****
37 ; void _cls() *
38 ;*****
39 public _cls
40 _cls proc
41 ; 清屏
42     push ax
43     push bx
44     push cx
45     push dx
46     mov ax, 600h ; AH = 6, AL = 0
47     mov bx, 700h ; 黑底白字 (BL = 7)
48     mov cx, 0    ; 左上角: (0, 0)
49     mov dx, 184fh ; 右下角: (24, 79)
50     int 10h      ; 显示中断
51     mov ah, 2
52     mov bh, 0
53     mov dx, 0
54     int 10h
55     pop dx
56     pop cx
57     pop bx
58     pop ax
59     ret
60 _cls endp

```

```

86 ; =====
87 ; void _loadUser();
88 ; =====
89 OffSetOfUserPrg equ 8100h
90 public _loadUser
91 _loadUser proc
92     push ds
93     push es
94     push bp
95     mov bp,sp
96     mov ax,cs                ;段地址 ; 存放数据的内存基地址
97     mov es,ax                ;设置段地址（不能直接mov es,段地址）
98     mov bx, OffSetOfUserPrg ;偏移地址; 存放数据的内存偏移地址
99     mov ah,2                  ;功能号
100    mov al,1                  ;扇区数
101    mov dl,0                  ;驱动器号 ; 软盘为0, 硬盘和U盘为80H
102    mov dh,0                  ;磁头号 ; 起始编号为0
103    mov ch,0                  ;柱面号 ; 起始编号为0
104    mov cl,[bp+8]             ;用户程序在2,3,4,5扇区
105    int 13h ;                  调用读磁盘BIOS的13h功能
106    mov bx,OffsetOfUserPrg
107    call bx
108    pop bp
109    pop es
110    pop ds
111    ret
112 _loadUser endp

```

```

61 ;*****
62 ; void _getChar() *
63 ;*****
64 public _getChar
65 _getChar proc
66     mov ah,0
67     int 16h
68     mov byte ptr [_x],al
69     ret
70 _getChar endp

```

```

71 ; =====
72 ; void _printChar(char ch);
73 ; =====
74 public _printChar
75 _printChar proc
76     push bp
77     mov bp,sp
78     mov al,[bp+4];char\ip\bp
79     mov bl,0
80     mov ah,0eh
81     int 10h
82     mov sp,bp
83     pop bp
84     ret
85 _printChar endp

```

_cls 基于老师给的大致没有改变。_loadUser 利用压栈实现参数传递，参数就是跳转的扇区号。_getChar 函数使用 int 16h 中断指令，_x 是 kernel.c

中定义的字符，用于读入后传到 C 库中。_printChar 函数使用 int 13h 的 0eh 功能输出字符。

c) C 库函数(kernel.c)

在 C 语言程序中，实现了四个基于汇编库函数的拓展函数：

printf: 输出字符串

strcmp: 判断字符串是否相等

InputCom: 从键盘读取命令

RunCom: 运行命令函数

```
void printf(char* str) {
    int i=0;
    while(str[i]) {
        printChar(str[i]);
        i++;
    }
}

int strcmp(char* a,char* b) {
    int i=0,j=0;
    while(a[i]!=0&&b[j]!=0) {
        if(a[i]!=b[j])
            return 0;
        i++;
        j++;
    }
    if(a[i]==0&&b[j]==0)
        return 1;
    else
        return 0;
}
```

在 klib.asm 中实现了一个字符的输入与输出，在 C 中只要循环调用就能变成 printf 输出字符串。

```
void InputCom(char* command) {
    int i=0;
    getChar();
    while(x!=13) {
        if(x==8) {
            printChar(x);
            printChar(32);
            i--;
            printChar(x);
            getChar();
            continue;
        }
        printChar(x);
        command[i]=x;
        i++;
        getChar();
    }
    command[i]='\0';
}
```

```

void RunCom(char *command) {
    if(strcmp(command,"help"))
        help();
    else if(strcmp(command,"cls"))
        cls();
    else if(strcmp(command,"ls"))
        file();
    else if(strcmp(command,"u1")) {
        cls();
        loadUser(2);
        cls();
    }
    else if(strcmp(command,"u2")) {
        cls();
        loadUser(3);
        cls();
    }
    else if(strcmp(command,"u3")) {
        cls();
        loadUser(4);
        cls();
    }
    else if(strcmp(command,"u4")) {
        cls();
        loadUser(5);
        cls();
    }
    else if(strcmp(command,"runall")) {
        cls();
        tasks = 4;
        cls();
    }
    else if(strcmp(command,"quit"))
        symbol=0;
    else
        printf("\n\rWrong command!\n\r");
}

```

这是关于命令的读取与执行的两个函数。在 InputCom 函数中我用 ASCII 是否为 8 判断退格，实现前一个填充空白，防止裸机操作系统的退格让光标后移。在 RunCom 函数中用 strcmp 判断不同的命令，从而执行不同的操作，各种命令的含义在后面会讲到，如果命令是无效的，那么会输出 “Wrong Command!”

d) 操作系统内核的功能(kernel.c)

Screen(): 在进入操作系统时显示的一个简单界面，用 printf 即可完成

```

void Screen() {
    printf("*****\n\r");
    printf("*                               Welcome to my OS!                               *\n\r");
    printf("*                               Enjoy yourself!                               *\n\r");
    printf("*                               STY 19335174                               *\n\r");
    printf("*****\n\r");
    printf("Enter help to get the command list\n\r");
}

```

help(): 帮助菜单，可以告诉用户可以执行哪些命令。Help 用于获取帮助菜单，cls 用于清屏，u1-u4 是分别执行 4 个用户程序，runall 是依次执行一遍用户程序 u1-u4，ls 用于获取文件列表，quit 退出操作系统。

```

void help() {
    printf("\n\rhelp:    get the command list\n\r");
    printf("cls:      clear the screen\n\r");
    printf("u1:        run the user program 1\n\r");
    printf("u2:        run the user program 2\n\r");
    printf("u3:        run the user program 3\n\r");
    printf("u4:        run the user program 4\n\r");
    printf("runall:    run all the user programs\n\r");
    printf("ls:       get the file list\n\r");
    printf("quit:     exit the OS\n\r");
}

```

file(): 一个文件列表。利用 printf 函数显示 4 个用户程序的文件大小。

```

void file() {
    printf("\n\rThe Number Of Files: 4\n\r");
    printf("\n\r1.Number      402B\n\r");
    printf("\n\r2.Name        397B\n\r");
    printf("\n\r3.Rectangle   298B\n\r");
    printf("\n\r4.Stone       393B\n\r");
}

```

cmain 主程序：监控程序会调用 cmain 来实现操作系统的正常运行，cmain 里可以调用各种函数。在 cmain 中先调用 Screen 函数显示个性界面。Symbol 是一个标志变量，当输入命令 quit 时 symbol=0 从而跳出 while 循环，退出操作系统。对于命令 runall，我用一个变量 tasks 使其保留剩余要执行的用户程序个数，在每次 while 循环中如果 tasks 不为 0 就执行用户程序，否则就输入新的命令。因为一共有 4 个用户程序，所以用 loadUser(6-tasks) 可以完美按顺序执行 u1-u4 用户程序（用户程序在 2-5 扇区），用一个 getChar 函数在每次用户程序返回时停顿，否则在执行完第一个用户程序后后面的用户程序只会一闪而过。

```

cmain() {
    cls();
    Screen();
    symbol=1;
    while(symbol) {
        if (tasks!=0) {
            loadUser(6-tasks);
            tasks--;
            getChar();
            continue;
        }
        printf("\n\r>>");
        InputCom(command);
        RunCom(command);
    }
    cls();
    printf("\n\rThank you for using!\n\rSee you next time!\n\r");
    getChar();
}

```

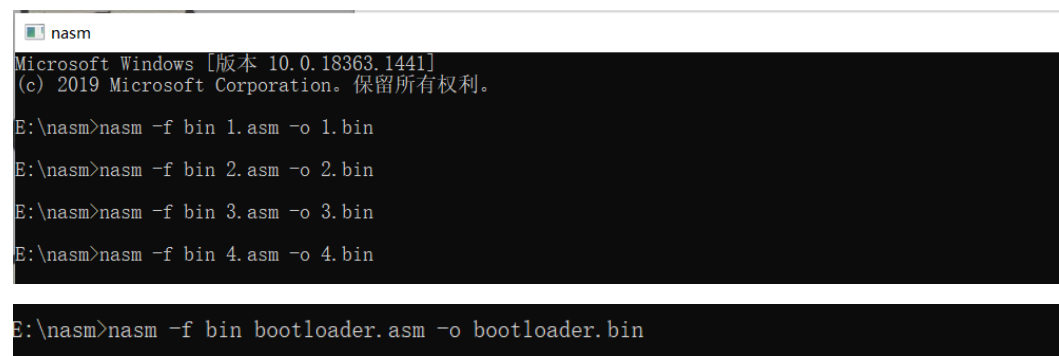
3) 用户程序

用户程序沿用了实验二的 4 个用户程序。唯一的改进是使用了 call 而不是 jmp 跳入用户程序，在用户程序中也用 ret 进行返回。用户程序在 2-5 扇区。在用户程序界面按空格返回操作系统界面。

```
149      ; 输入空格后弹出程序
150      mov ah,1
151      int 16h
152      mov bl,20h
153      cmp al,bl
154      jz Quit
155      jmp loop1
156
157 Quit:
158      ret
159 end:
160      jmp $ ; 停止画框，无限循环
```

4) NASM 编译与 TCC-TASM-TLINK 混合编译链接

用户程序和引导程序用 NASM 编译

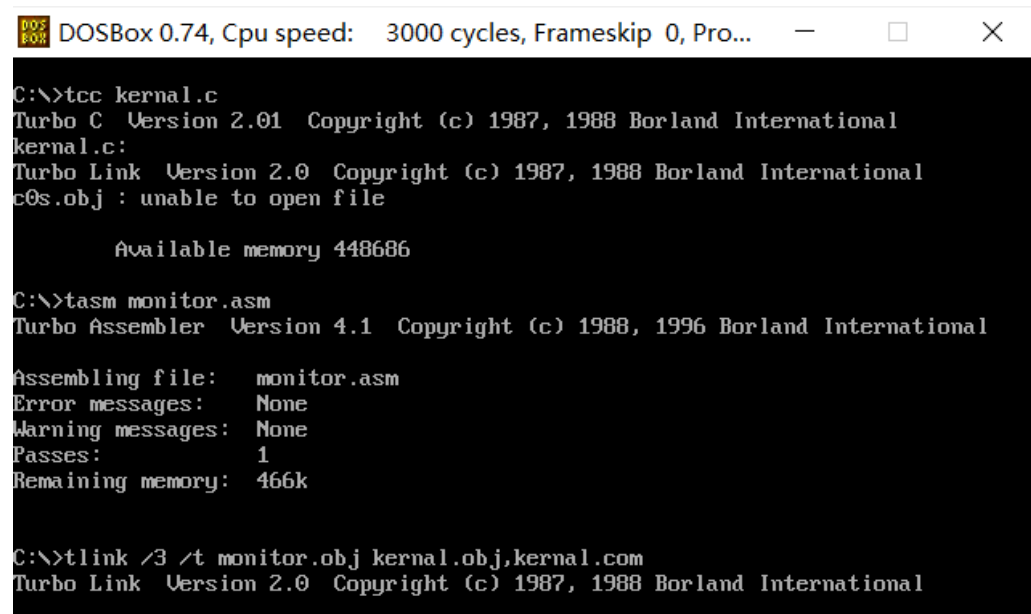


```
nasm
Microsoft Windows [版本 10.0.18363.1441]
(c) 2019 Microsoft Corporation。保留所有权利。

E:\nasm>nasm -f bin 1.asm -o 1.bin
E:\nasm>nasm -f bin 2.asm -o 2.bin
E:\nasm>nasm -f bin 3.asm -o 3.bin
E:\nasm>nasm -f bin 4.asm -o 4.bin

E:\nasm>nasm -f bin bootloader.asm -o bootloader.bin
```

内核的三个程序用 TCC-TASM-TLINK 混合编译链接



```
DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Pro...
C:\>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 448686

C:\>tasm monitor.asm
Turbo Assembler Version 4.1 Copyright (c) 1988, 1996 Borland International

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

C:\>tlink /3 /t monitor.obj kernal.obj,kernal.com
Turbo Link Version 2.0 Copyright (c) 1987, 1988 Borland International
```


5) 使用 WinHex 修改软盘

使用 WinHex 打开所有的 COM 文件和 BIN 文件。将引导程序 bootloder.bin 放入首扇区，将 4 个用户程序分别放入 2-5 扇区，将内核 kernal.com 放入第 6 扇区。

Offset	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
00000000	8C	C8	8E	C0	BB	00	A1	B4	02	B0	04	B2	00	B6	00	B5
00000016	00	B1	06	CD	13	EA	00	01	00	0A	00	00	00	00	00	00
00000032	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000048	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000064	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
00000096	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000112	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000128	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000144	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
00000176	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000192	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000208	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000224	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000240	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000256	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000272	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000288	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000304	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000320	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000336	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000352	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000368	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000384	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000400	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000416	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000432	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000448	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000464	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000480	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000496	00	00	00	00	00	00	00	00	00	00	00	00	00	55	AA	
00000512	B4	06	B0	00	B5	00	B1	00	B6	18	B2	4F	B7	07	CD	10
00000528	8C	C8	8E	D8	8E	D0	B8	00	B8	8E	C0	FF	0E	81	82	75
00000544	FA	C7	06	81	82	50	C3	FF	0E	83	82	75	EE	C7	06	81
00000560	82	50	C3	C7	06	83	82	44	02	B0	01	3A	06	85	82	74
00000576	1E	B0	02	3A	06	85	82	74	53	B0	03	3A	06	85	82	0F
00000592	84	85	00	B0	04	3A	06	85	82	0F	84	B5	00	EB	FE	FF
00000608	86	8E	82	FF	06	88	82	8B	1E	86	82	B8	0E	00	29	D8
00000624	74	0E	8B	1E	88	82	B8	28	00	29	D8	74	11	E9	CC	00
00000640	C7	06	86	82	0C	00	C6	06	85	82	02	E9	BE	00	C7	06
00000656	88	82	26	00	C6	06	85	82	04	E9	B0	00	FF	0E	86	82
00000672	FF	06	88	82	8B	1E	88	82	B8	28	00	29	D8	74	0E	8B
00000688	1E	86	82	B8	FF	FF	29	D8	74	11	E9	8F	00	C7	06	88
00000704	82	26	00	C6	06	85	82	03	E9	81	00	C7	06	86	82	01
00000720	00	C6	06	85	82	01	EB	74	FF	0E	86	82	FF	0E	88	82

Offset	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
00001472	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00001488	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00001504	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00001520	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00001536	B4	06	B0	00	B5	00	B1	00	B6	18	B2	4F	B7	07	CD	10
00001552	B8	1A	82	89	C5	B9	0B	00	B8	01	13	BB	07	00	B6	05
00001568	B2	37	CD	10	8C	C8	8E	D8	8E	D0	B8	00	B8	8E	C0	FF
00001584	0E	15	82	75	FA	C7	06	15	82	50	C3	FF	0E	17	82	75
00001600	EE	C7	06	15	82	50	C3	C7	06	17	82	44	02	B0	01	3A
00001616	06	19	82	74	1A	B0	02	3A	06	19	82	74	30	B0	03	3A
00001632	06	19	82	74	46	B0	04	3A	06	19	82	74	5C	EB	FE	FF
00001648	06	25	82	8B	1E	25	82	B8	0C	00	29	D8	74	02	EB	FF
00001664	C7	06	25	82	0B	00	C6	06	19	82	02	EB	5A	FF	0E	27
00001680	82	B8	1E	27	82	B8	50	00	29	D8	74	02	EB	49	C7	06
00001696	27	82	4F	00	C6	06	19	82	03	EB	3C	FF	0E	25	82	8B
00001712	1E	25	82	B8	FF	FF	29	D8	74	02	EB	2B	C7	06	25	82
00001728	00	00	C6	06	19	82	04	EB	1E	FF	0E	27	82	8B	1E	27
00001744	82	B8	27	00	29	D8	74	02	EB	0D	C7	06	27	82	28	00
00001760	C6	06	19	82	01	EB	00	31	C0	A1	25	82	BB	50	00	F7
00001776	E3	03	06	27	82	BB	02	00	F7	E3	89	C5	B4	0F	A0	29
00001792	82	26	89	46	00	B4	01	CD	16	B3	20	38	D8	74	03	E9
00001808	1D	FF	C3	EB	FE	50	C3	44	02	01	31	39	33	33	35	31
00001824	37	34	53	54	59	00	00	28	00	41	00	00	00	00	00	00
00001840	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00001856	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00001872	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00001888	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00001904	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00001920	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00001936	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00001952	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00001968	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00001984	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00002000	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00002016	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00002032	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00002048	B4	06	B0	00	B5	00	B1	00	B6	18	B2	4F	B7	07	CD	10
00002064	8C	C8	8E	D8	8E	D0	B8	00	B8	8E	C0	C6	06	88	82	41
00002080	FF	0E	7F	82	75	FA	C7	06	7F	82	50	C3	FF	0E	81	82
00002096	75	EE	C7	06	7F	82	50	C3	C7	06	81	82	44	02	B0	01
00002112	00	06	83	82	74	1E	B0	02	3A	06	83	82	74	53	B0	03
00002128	3A	06	83	82	0F	84	85	00	B0	04	3A	06	83	82	0F	84
00002144	B5	00	EB	FE	FF	06	84	82	FF	06	86	82	8B	1E	84	82
00002160	B8	19	00	29	D8	74	0E	8B	1E	86	82	B8	50	00	29	D8
00002176	74	11	E9	CC	00	C7	06	84	82	87	00	C6	06	83	82	02
00002192	E9	BE	00	C7	06	86	82	4E	00	C6	06	83	82	04	E9	B0

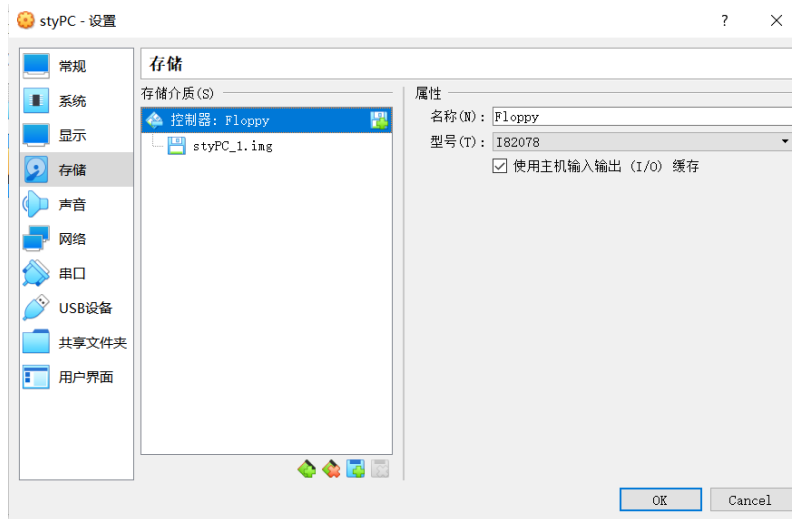
Offset	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
00000736	8B	1E	86	82	B8	FF	FF	29	D8	74	0D	8B	1E	88	82	B8
00000752	FF	FF	29	D8	74	0F	EB	54	C7	06	86	82	01			

Offset	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
00002944	EB	3C	80	3E	0E	08	08	75	20	A0	0E	08	98	5B	E8	CD
00002960	FE	59	B8	20	00	50	E8	C5	FE	59	AE	A0	0E	00	98	50
00002976	E8	BB	FE	59	E8	AE	FE	EB	15	A0	0E	08	98	50	E8	AD
00002992	FE	59	A0	0E	08	8B	5E	04	C6	00	00	5E	5D	C3	B8	2D
00003008	0E	08	0D	75	B8	5E	04	C6	00	00	5E	5D	C3	B8	2D	
00003024	07	50	48	BD	FE	59	B8	48	07	50	E8	B5	FE	59	B8	5F
00003040	07	50	E8	AD	FE	59	B8	76	07	50	E8	A5	FE	59	B8	8D
00003056	07	50	E8	9D	FE	59	C3	55	8B	EC	56	8B	76	04	B8	A4
00003072	0B	C0	74	06	E8	97	FF	E9	B4	00	B8	B0	07	50	56	E8
00003088	DC	00	B8	A9	07	50	56	E8	11	FF	59	59	0B	C0	74	06
00003104	E8	12	FE	E9	C8	00	B8	AD	07	50	56	E8	FD	FE	59	59
00003120	0B	C0	74	06	E8	97	FF	E9	B4	00	B8	B0	07	50	56	E8
00003136	E9	FE	59	59	0B	C0	74	11	E8	EA	FD	B8	02	00	50	E8
00003152	1C	FE	59	E8	DF	FD	E9	95	00	B8	B3	07	50	56	E8	CA
00003168	FE	59	59	0B	C0	74	11	E8	CB	FD	B8	03	00	50	E8	FD
00003184	FD	59	E8	C0	FD	E9	76	00	B8	B6	07	50	56	E8	AB	FE
00003200	59	59	0B	C0	74	10	E8	AC	FD	B8	04	00	50	E8	DE	FD
00003216	59	E8	A1	FD	EB	58	B8	B9	07	50	56	E8	8D	FE	59	59
00003232	0B	C0	74	10	E8	8E	FD	B8	05	00	50	E8	C0	FD	59	E8
00003248	83	FD	EB	3A	B8	BC	07	50	56	E8	6F	FE	59	59	0B	C0
00003264	74	0E	E8	70	FD	C7	06	1B	08	04	00	E8	67	FD	EB	1E
00003280	B8	C3	07	50	56	E8	53	FE	59	59	0B	C0	74	08	C7	06
00003296	1D	08	00	00	EB	08	B8	C8	07	50	E8	A5	FD	59	E8	5D
00003312	C3	E8	41	FD	E8	BA	FD	C7	06	1D	08	01	00	EB	34	83
00003328	3E	1B	08	00	74	15	B8	06	00	2B	06	1B	08	50	E8	5D
00003344	FD	59	FF	0E	1B	08	E8	3C	FD	EB	1B	B8	DB	07	50	E8
00003360	70	FD	59	B8	11	08	50	E8	4D	FE	59	B8	11	08	50	E8
00003376	C5	FE	59	E3	3E	1D	08	00	75	C5	E8	F8	FC	B8	E0	07
00003392	50	E8	4E	FD	59	E8	0D	FD	C3	00	2A	2A	2A	2A	2A	2A
00003408	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A
00003424	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A
00003440	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A
00003456	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A
00003472	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	0A	0D	00	2A	20	20
00003488	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
00003504	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
00003520	63	6F	6D	65	20	74	6F	20	6D	79	20	4F	53	21	20	20
00003536	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
00003552	20	20	20	20	20	20	20	20	20	20	20	20	20	2A	0A	0D
00003568	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
00003584	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
00003600	20	45	6E	6A	6F	79	20	79	6F	75	72	73	65	6C	66	21
00003616	2A	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
00003632	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	2A
00003648	0A	0D	00	2A	20	20	20	20	20	20	20	20	20	20	20	20
00003664	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20

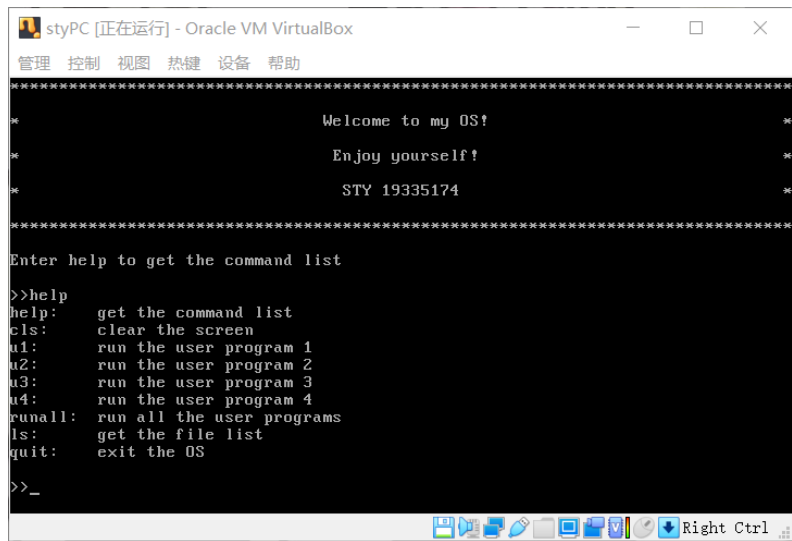
Offset	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
00003680	20	20	20	20	20	53	54	59	20	31	39	33	33	3B	31	37
00003696	34	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
00003712	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
00003728	20	20	2A	0A	0D	00	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A
00003744	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A
00003760	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A
00003776	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A
00003792	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A
00003808	2A	2A	2A	2A	2A	2A	0A	0D	00	45	6E	74	65	72	20	68
00003824	65	6C	70	20	74	6F	20	67	65	74	20	74	68	65	20	63
00003840	6F	6D	6D	61	6E	64	20	6C	69	73	74	0A	0D	00	0A	0D
00003856	68	65	6C	70	3A	20	20	20	20	67	65	74	20	74	68	65
00003872	20	63	6F	6D	6D	61	6E	64	20	6C	69	73	74	0A	0D	00
00003888	63	6C	73	3A	20	20	20	20	20	63	6C	65	61	72	20	74
00003904	68	65	20	73	63	72	65	65	6E	0A	0D	00	75	31	3A	20
00003920	20	20	20	20	20	20	72	75	6E	20	74	68	65	20	75	73
00003936	72	20	70	72	6F	67	72	61	6D	20	31	0A	0D	00	75	32
00003952	3A	20	20	20	20	20	20	72	75	6E	20	74	68	65	20	75
00003968	73	65	72	20	70	72	6F	67	72	61	6D	20	32	0A	0D	00
00003984	75	33	3A	20	20	20	20	20	20	72	75	6E	20	74	68	65
00004000	20	75	73	65	72	20	70	72	6F	67	72	61	6D	20	33	0A
00004016	0D	00	75	34	3A	20	20	20	20	20	20	72	75	6E	20	74
00004032	68	65	20	75	73	65	72	20	70	72	6F	67	72	61	6D	20
00004048	34	0A	0D	00	72	75	6E	61	6C	6C	3A	20	20	72	75	6E
00004064	20	61	6C	6C	20	74	68	65	20	75	73	65	72	20	70	72
00004080	6F	67	72	61	6D	73	0A	0D	00	6C	73	3A	20	20	20	20
00004096	20	20	67	65	74	20	74	68	65	20	66	69	6C	65	20	6C
00004112	69	73	74	0A	0D	00	71	75	69	74	3A	20	20	20	20	65
00004128	78	69	74	20	74	68	65	20	4F	53	0A	0D	00	0A	0D	54
00004144	68	65	20	4E	75	6D	65	65	72	20	4F	66	20	46	69	6C
00004160	65	73	3A	20	34	0A	0D	00	0A	0D	31	2E	4E	75	6D	62
00004176	65	72	20	20	20	20	20	20	34	30	32	42	0A	0D	00	0A
00004192	0D	32	2E	4E	61	6D	65	20	20	20	20	20	20	20	20	33
00004208	39	37	42	0A	0D	00	0A	0D	33	2E	52	65	63	74	61	6E
00004224	67	6C	65	20	20	20	32	39	38	42	0A	0D	00	0A	0D	34
00004240	2E	53	74	6F	6E	65	20	20	20	20	20	20	20	33	39	33
00004256	42	0A	0D	00	68	65	6C	70	00	63	6C	73	00	6C	73	00
00004272	75	31	00	75	32	00	75	33	00	75	34	00	72	75	6E	61
00004288	6C	6C	00	71	75	69	74	00	0A	0D	57	72	6F	6E	67	20
00004304	63	6F	6D	6D	61	6E	64	21	0A	0D	00	0A	0D	3E	3E	00
00004320	0A	0D	54	68	61	6E	6B	20	79	6F	75	20	66	6F	72	20
00004336	75	73	69	6E	67	21	0A	0D	53	65	65	20	79	6F	75	20
00004352	6E	65	78	74	20	74	69	6D	65	21	0A	0D	00	00	00	00
00004368	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00004384	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00004400	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

6) 用软盘启动裸机

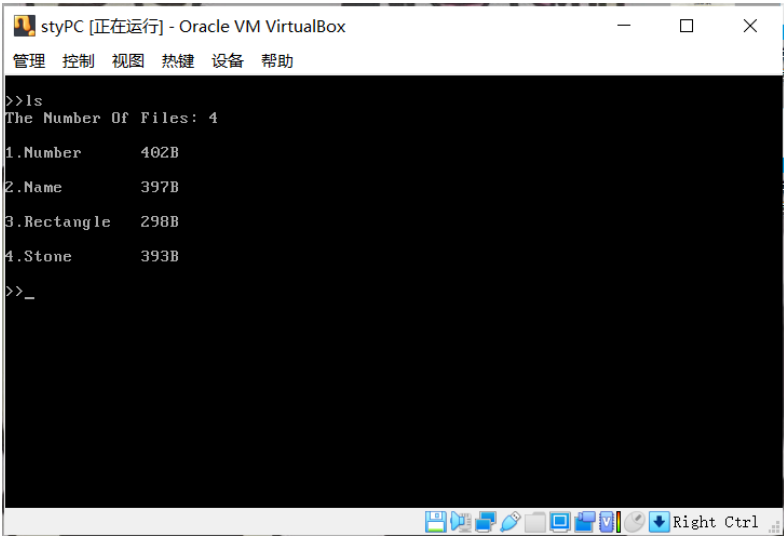
打开虚拟机 VirtualBox



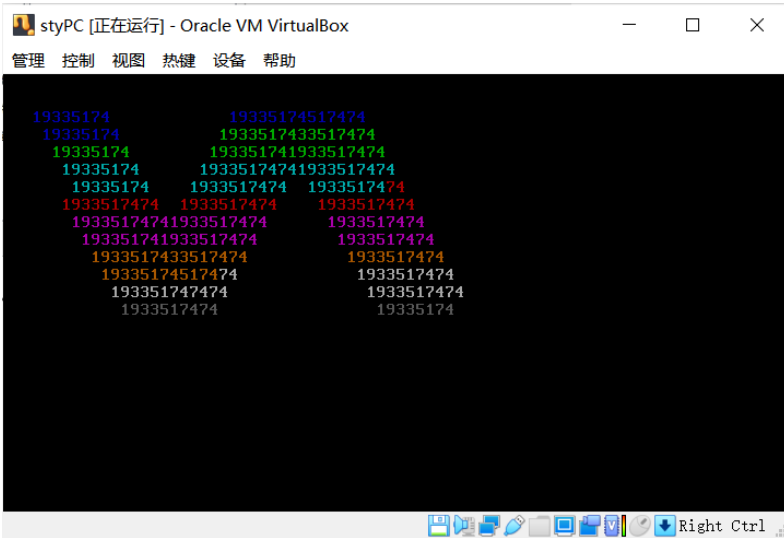
操作系统界面



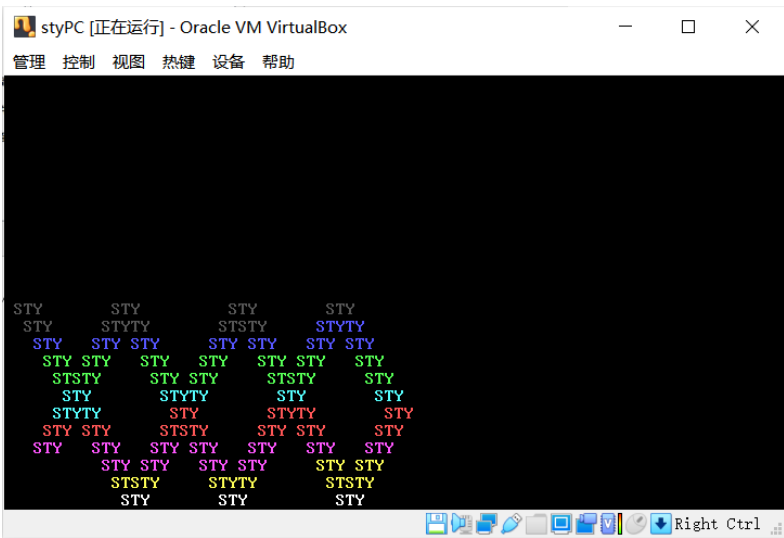
文件列表



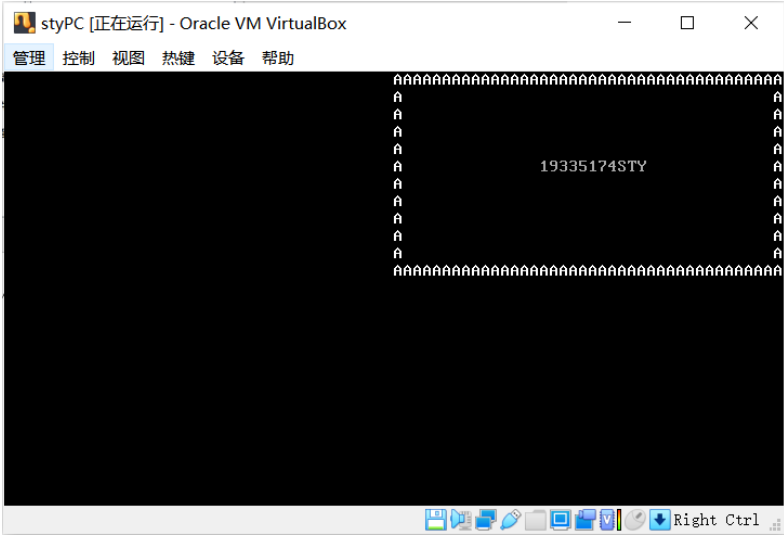
用户程序 1



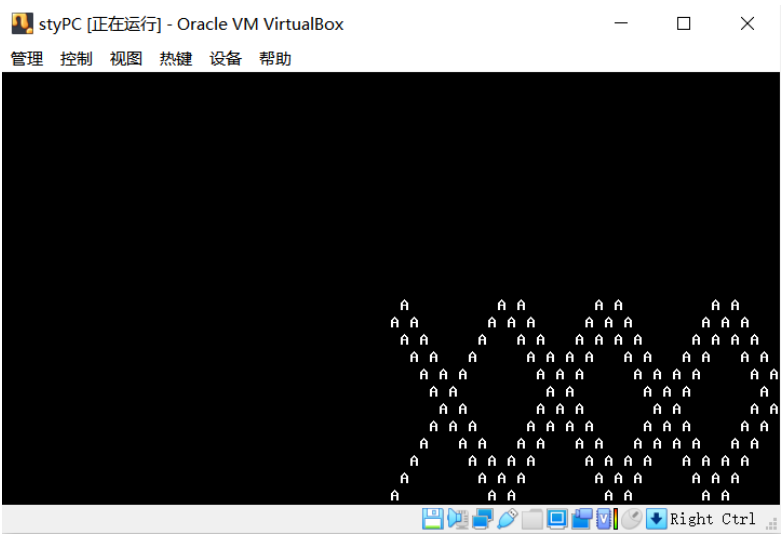
用户程序 2



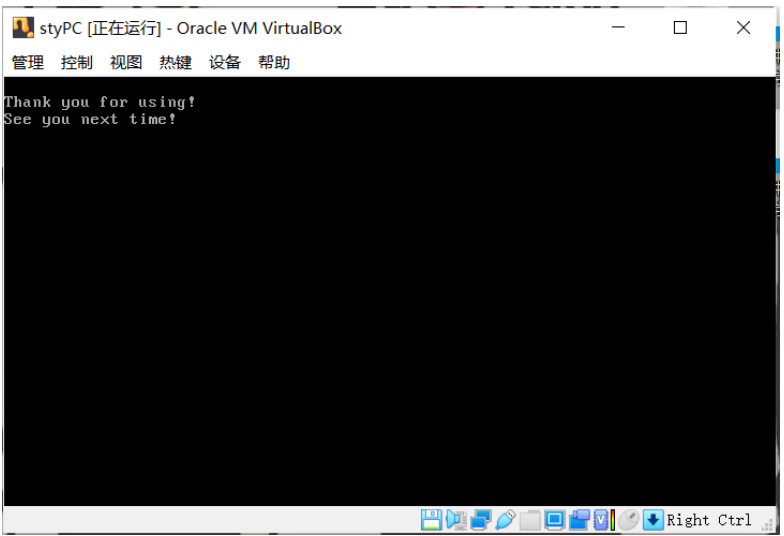
用户程序 3



用户程序 4



退出界面



Runall 命令是依次执行四个用户程序，截图很难显示动画的效果，我在“运行视频”中有演示。

六、实验总结

这次实验中相比前几次实验难度一下提升了很多，在混合编译链接这块我就花了很多时间才明白。我原本是想用 gcc-nasm 的，但是我总是编译链接出错，于是就换用了 tcc-tasm，但是由于 tcc-tasm 过于古老，网上关于这方面的资料也比较少，所以我还是遇到了不少麻烦。好在经过不懈的努力，我终于还是顺利完成了这次实验。

引导程序的设计相对比较简单，只要跳到内核的扇区即可。但是内核的设计十分艰难，特别是在传递参数这一块。压栈的原理比较复杂，让我一开始疑惑了好久，好在老师有代码参考也让我弄明白了。在实验过程中，我发现一些函数的参数如果用全局变量来设计可能相对简单一些，我也使用了一些全局变量来传递参数。在调用用户程序时我也学到了一个新方法，用 call 代替 jmp，这样在用户程序使用 ret 就能返回。

这次实验我也遇到了一些细节问题。我在用 TLINK 混合编译链接时，一开始把 C 生成的 obj 文件写在了前面，这样就会出错，后来我才发现要把会变更生成的 obj 文件写在前面才能正常编译。在使用 tcc 编译时，我一开始用 “//” 来添加中文注释，但是却报错了，研究了一下发现 tcc 比较古老，只能用 “/**/” 的方式注释。我在设计 runall 命令时，一开始直接想用连续跳转到四个不同扇区的方法，却发现这样不可能实现——用户程序 1 还能执行，后面的用户程序只会一闪而过。经过调试我发现，在 cmain 中添加一个 tasks 变量判断是否有剩余的任务没完成，并在每次跳转到用户程序后加一个 getChar() 停顿，就可以实现用户程序的直接跳转了。

总而言之，这次实验的探索之旅让我学到了很多东西。虽然花费的时间和精力很多，但一切都是值得的。希望在后面的实验中，我也能再接再厉，勇往直前！

七、参考文献

《汇编语言 （第 3 版）》

《X86 汇编语言：从实模式到保护模式》

<https://blog.csdn.net/wuxch8/article/details/106921308/>

<https://blog.csdn.net/hual9880705/article/details/8125706>

<https://blog.csdn.net/longintchar/article/details/79511747>

<https://blog.csdn.net/mirage1993/article/details/45116881>

<https://blog.csdn.net/knxw0001/article/details/7248683>